Copyright and publication information

This manual reflects version 2.6 of Ultra C Compiler. Reproduction of this document, in part or whole, by any means, electrical, mechanical, magnetic, optical, chemical, manual, or otherwise is prohibited, without written permission from RadiSys Corporation.

Disclaimer

The information contained herein is believed to be accurate as of the date of publication. However, RadiSys Corporation will not be liable for any damages including indirect or consequential, from use of the OS-9 operating system, Microware-provided software, or reliance on the accuracy of this documentation. The information contained herein is subject to change without notice.

Reproduction notice

The software described in this document is intended to be used on a single computer system. RadiSys Corporation expressly prohibits any reproduction of the software on tape, disk, or any other medium except for backup purposes. Distribution of this software, in part or whole, to any other party or on any other system may constitute copyright infringements and misappropriation of trade secrets and confidential processes which are the property of RadiSys Corporation and/or other parties. Unauthorized distribution of software may cause damages far in excess of the value of the copies involved.
## Contents

Using Functions and Macros ................................................................. 22  
Library Files ..................................................................................... 22  
Program Globals ............................................................................... 23  
  Class 1 Globals ............................................................................... 23  
  Class 2 Globals ............................................................................. 24  
Header Files ..................................................................................... 25  
The C Library ..................................................................................... 43  
  Communicating with Environment Functions .......................... 43  
  Character Classification Functions ........................................... 43  
  Character Conversion Functions .................................................. 44  
  File Positioning Functions .......................................................... 44  
  GetStat/SetStat Functions ......................................................... 45  
  Input/Output Device Functions .................................................... 47  
  Input/Output Functions .............................................................. 47  
  Interprocess Communication Functions ................................. 50  
  Interrupt Manipulation Functions ............................................. 53  
  Mathematical Functions ............................................................ 53  
  Memory Management Functions .............................................. 54  
  Shared Data Access Functions ................................................. 56  
  Miscellaneous Functions ........................................................... 56  
  Module Manipulation Functions ............................................... 58  
  Multibyte Character and String Functions ......................... 59  
  OS System Functions ................................................................. 61  
  POSIX Messaging Functions ...................................................... 61  
  Process Manipulation Functions ............................................. 62  
  Resource Locks ........................................................................... 63  
  Searching and Sorting Functions ............................................ 63  
  String Handling Functions ......................................................... 63  
  Terminal Manipulation Functions (Termcap) ......................... 65  
  Time Functions ........................................................................... 65  
  Processor-Specific Functions .................................................... 68  
  Implementation-Defined Behavior ....................................... 69  
  ANSI “Core” Signals ................................................................. 70  
  OS-9 “Core” Signals ................................................................. 70  
  OS-9 68K Family Signals ............................................................ 70  
  OS-9 80x86 Family Signals ......................................................... 70  
  OS-9 PowerPC™ Family Signals ............................................... 70  
  Processor-Specific Family Signals ........................................ 71  
  68K Family Processors ............................................................... 71  
  80x86 Family Processors ............................................................ 72  
  PowerPC Family Processors ....................................................... 72
### Contents

- ARM Family Processors ................................................................. 72
- SH Family Processors .................................................................. 73
- SPARC Family Processors ............................................................ 73
- Equivalents for the sys_clib.l Functions ....................................... 77
- The Function Description ............................................................ 80
  - Syntax ......................................................................................... 80
  - OS-9 Attributes .......................................................................... 81
  - Library ....................................................................................... 81
  - Errors ....................................................................................... 81
- References to Other Calls ............................................................. 81

### C Shared Library ..................................................................... 81
- abort() ....................................................................................... 88
- abs() ......................................................................................... 89
- access() .................................................................................... 90
-acos() ....................................................................................... 91
-alarm() ....................................................................................... 92
-alloca() ..................................................................................... 93
-alm_atdate() .............................................................................. 94
-alm_atjul() ............................................................................... 96
-alm_cycle() ............................................................................... 98
-alm_delete() ............................................................................ 100
-alm_set() .................................................................................. 101
-asctime() ................................................................................... 102
-asin() ....................................................................................... 104
-assert() ..................................................................................... 105
-atan() ....................................................................................... 106
-atan2() ..................................................................................... 107
-atexit() ..................................................................................... 108
- atof() ....................................................................................... 109
- atoi() ....................................................................................... 110
- atol() ....................................................................................... 111
-_atok() ..................................................................................... 112
-_atoul() ..................................................................................... 113
-attach() ..................................................................................... 114
-bcmp() ...................................................................................... 115
-bcopy() ..................................................................................... 116
-bsearch() .................................................................................. 117
-buildpath() ............................................................................... 119
-bzero() ...................................................................................... 120
calloc() ....................................................................................... 121
-cbrt() ....................................................................................... 122
-ceil() ......................................................................................... 123
-chainc(), chain() ................................................................. 124
-change_const() ...................................................................... 127
-change_static() ..................................................................... 128
-chdir() ...................................................................................... 129
-chmod() ..................................................................................... 130
-chown() ..................................................................................... 131
-chxdir() ..................................................................................... 132
getpwuid() ....................................................................................................................................... 259
getpwnam() ..................................................................................................................................... 258
getpw() ............................................................................................................................................ 255
_get_process_table() ........................................................................................................................ 254
_get_process_desc() ........................................................................................................................... 253
getpid() ............................................................................................................................................... 251
getopt() ............................................................................................................................................... 249
getnextpwnam() .................................................................................................................................. 248
_get_module_dir() ............................................................................................................................... 247
get_gdtr() ........................................................................................................................................... 245
getgid() ............................................................................................................................................... 243
getenv() ............................................................................................................................................... 241
getcwd() ............................................................................................................................................... 240
get_current_tss() .................................................................................................................................. 239
get_const() .......................................................................................................................................... 238
cgetc(), getchar() ............................................................................................................................... 237
fwrite() ............................................................................................................................................... 234
fstat() .................................................................................................................................................. 232
fseek() .................................................................................................................................................. 229
frexp() .................................................................................................................................................. 223
freopen() ............................................................................................................................................... 222
_freemem(), freemem() ....................................................................................................................... 220
_freemin() .......................................................................................................................................... 221
freopen() ............................................................................................................................................... 222
frewp() ................................................................................................................................................. 223
fscanf() ............................................................................................................................................... 224
fseek() .................................................................................................................................................. 229
fsetpos() .............................................................................................................................................. 231
fstat() .................................................................................................................................................. 232
fell() .................................................................................................................................................... 233
fwrite() ............................................................................................................................................... 234
_get_<name>() .................................................................................................................................... 236
getc(), getchar() ................................................................................................................................... 237
get_current_tss() .................................................................................................................................. 238
getcwd() ............................................................................................................................................... 239
getenv() ............................................................................................................................................... 241
getgid() ............................................................................................................................................... 243
get_except_base() .................................................................................................................................. 244
gidtr() ................................................................................................................................................... 245
getime() ............................................................................................................................................... 246
_get_module_dir() ............................................................................................................................... 247
getnextpwnam() .................................................................................................................................. 248
getopt() ............................................................................................................................................... 249
getpid() ............................................................................................................................................... 251
getppid() .......................................................................................................................................... 252
_get_process_desc() ........................................................................................................................... 253
_get_process_table() ........................................................................................................................... 254
getpw() ............................................................................................................................................... 255
getpwent() .......................................................................................................................................... 256
getpwnam() ........................................................................................................................................ 258
getpwuid() ........................................................................................................................................... 259
isupper() ........................................................................................................................................ 319
isxdigit() .................................................................................................................................... 320
_ Julian() .................................................................................................................................... 321
kill() .......................................................................................................................................... 323
labs() ......................................................................................................................................... 324
_lcalloc() ................................................................................................................................. 325
ldeq() ........................................................................................................................................ 326
ldiv() .......................................................................................................................................... 327
_lfree() ...................................................................................................................................... 328
lfree() ......................................................................................................................................... 329
_lmalloc() .................................................................................................................................. 330
localeconv() .............................................................................................................................. 332
localtime() ................................................................................................................................. 335
log() .......................................................................................................................................... 336
log10() ...................................................................................................................................... 337
longjmp() ................................................................................................................................... 338
_lrealloc() .................................................................................................................................. 340
lseek() ....................................................................................................................................... 342
lstat() ........................................................................................................................................ 344
makdir() .................................................................................................................................... 345
make_gdesc() ............................................................................................................................ 347
make_idesc() ........................................................................................................................... 348
make_module() ........................................................................................................................ 349
malloc() .................................................................................................................................... 351
_malloca() .................................................................................................................................. 352
mblen() ..................................................................................................................................... 353
mblenwcs() .................................................................................................................................. 354
mbtowc() .................................................................................................................................... 355
memchr() .................................................................................................................................... 357
memcmp() .................................................................................................................................. 358
memcpy() .................................................................................................................................... 359
memmove() .................................................................................................................................. 360
memset() ..................................................................................................................................... 361
_mkdata_module() .................................................................................................................... 362
mknod() ..................................................................................................................................... 364
mkstemp() .................................................................................................................................. 365
mktemp() ................................................................................................................................... 366
mtime() ....................................................................................................................................... 367
mod_cload() .............................................................................................................................. 368
modf() ........................................................................................................................................ 370
modlink() .................................................................................................................................. 371
modload() .................................................................................................................................. 372
modloadp() .................................................................................................................................. 373
mq_close .................................................................................................................................... 374
mq_getattr ................................................................................................................................. 375
mq_notify ................................................................................................................................. 377
_mq_notify_write ....................................................................................................................... 379
mq_open ..................................................................................................................................... 381
mq_receive ............................................................................................................................... 384
mq_send........................................................................................................................................386
mq_setattr ....................................................................................................................................388
mq.unlink ......................................................................................................................................390
munlink() .....................................................................................................................................391
munload() .....................................................................................................................................392
offsetof() ....................................................................................................................................393
open() ..........................................................................................................................................394
opendir() ......................................................................................................................................395
_os_acqlk() ................................................................................................................................396
_os9_alarm_atdate() ...................................................................................................................398
_os_alarm_atime() ......................................................................................................................399
_os9_alarm_atjul() .......................................................................................................................400
_os_alarm_cycle() .........................................................................................................................401
_os_alarm_delete() ......................................................................................................................402
_os_alarm_reset() ........................................................................................................................403
_os_alarm_set() ...........................................................................................................................404
_os_alias() ...................................................................................................................................405
_os9_allbit() ................................................................................................................................406
_os9_allpd() ..................................................................................................................................407
_os_alltsk() ..................................................................................................................................408
_os_alocproc() ................................................................................................................................409
_os_altmdir() ................................................................................................................................410
_os_aproc() ...................................................................................................................................411
_os9_aproc() ..................................................................................................................................412
_os_attach() ..................................................................................................................................413
_os_cache() ...................................................................................................................................415
_os_caqlk() ...................................................................................................................................417
_os_chain() ...................................................................................................................................418
_os_chainm() ..................................................................................................................................420
_os_chdir() ...................................................................................................................................422
_os_chkmem() .................................................................................................................................423
_os_chmdir() ..................................................................................................................................424
_os_close() ....................................................................................................................................425
_os_clrsigs() ..................................................................................................................................426
_os_cmdperm() ................................................................................................................................427
_os_cmpnam() ................................................................................................................................428
_os_config() ...................................................................................................................................430
_os_cpy_ioproc() ............................................................................................................................431
_os_cpyymem() .............................................................................................................................432
_os_crc() .......................................................................................................................................433
_os_create() ...................................................................................................................................435
_os9_create() ..................................................................................................................................437
_os_crlk() ......................................................................................................................................439
_os_datmod() ...................................................................................................................................440
_os_ddlk() ......................................................................................................................................442
_os9_delbit() ..................................................................................................................................443
_os_delete() ....................................................................................................................................444
_os_dellk() ....................................................................................................................................445
_os_deldir() ....................................................................................................................................446
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_deltsk()</td>
<td>447</td>
</tr>
<tr>
<td>_os_detach()</td>
<td>448</td>
</tr>
<tr>
<td>_os_dexec()</td>
<td>449</td>
</tr>
<tr>
<td>_os_dexit()</td>
<td>452</td>
</tr>
<tr>
<td>_os_dfork()</td>
<td>453</td>
</tr>
<tr>
<td>_os9_dfork()</td>
<td>455</td>
</tr>
<tr>
<td>_os_dforkm()</td>
<td>457</td>
</tr>
<tr>
<td>_os_dup()</td>
<td>459</td>
</tr>
<tr>
<td>_os_ev_allclr()</td>
<td>460</td>
</tr>
<tr>
<td>_os_ev_allset()</td>
<td>461</td>
</tr>
<tr>
<td>_os_ev_anyclr()</td>
<td>462</td>
</tr>
<tr>
<td>_os_ev_anyset()</td>
<td>463</td>
</tr>
<tr>
<td>_os_ev_change()</td>
<td>464</td>
</tr>
<tr>
<td>_os_ev_creat()</td>
<td>466</td>
</tr>
<tr>
<td>_os_ev_delete()</td>
<td>468</td>
</tr>
<tr>
<td>_os_ev_info()</td>
<td>469</td>
</tr>
<tr>
<td>_os9_ev_info()</td>
<td>471</td>
</tr>
<tr>
<td>_os_ev_link()</td>
<td>472</td>
</tr>
<tr>
<td>_os_ev_pulse()</td>
<td>473</td>
</tr>
<tr>
<td>_os_ev_read()</td>
<td>475</td>
</tr>
<tr>
<td>_os_ev_set()</td>
<td>476</td>
</tr>
<tr>
<td>_os_ev_setand()</td>
<td>477</td>
</tr>
<tr>
<td>_os_ev_setor()</td>
<td>478</td>
</tr>
<tr>
<td>_os_ev_setr()</td>
<td>479</td>
</tr>
<tr>
<td>_os_ev_setxor()</td>
<td>481</td>
</tr>
<tr>
<td>_os_ev_signal()</td>
<td>482</td>
</tr>
<tr>
<td>_os_ev_tset()</td>
<td>483</td>
</tr>
<tr>
<td>_os_ev_trywait()</td>
<td>485</td>
</tr>
<tr>
<td>_os_ev_unlink()</td>
<td>486</td>
</tr>
<tr>
<td>_os_ev_wait()</td>
<td>487</td>
</tr>
<tr>
<td>_os9_ev_wait()</td>
<td>489</td>
</tr>
<tr>
<td>_os_ev_waitr()</td>
<td>490</td>
</tr>
<tr>
<td>_os9_ev_waitr()</td>
<td>492</td>
</tr>
<tr>
<td>_os_exec()</td>
<td>494</td>
</tr>
<tr>
<td>os9exec()</td>
<td>498</td>
</tr>
<tr>
<td>_os_exit()</td>
<td>501</td>
</tr>
<tr>
<td>_os_findpd()</td>
<td>502</td>
</tr>
<tr>
<td>_os9_findpd()</td>
<td>503</td>
</tr>
<tr>
<td>_os_firq()</td>
<td>504</td>
</tr>
<tr>
<td>_os_fmod()</td>
<td>506</td>
</tr>
<tr>
<td>_os_fork()</td>
<td>507</td>
</tr>
<tr>
<td>os9fork(), os9forkc()</td>
<td>509</td>
</tr>
<tr>
<td>_os_forkm()</td>
<td>511</td>
</tr>
<tr>
<td>_os_get_blkmap()</td>
<td>513</td>
</tr>
<tr>
<td>_os_getdl()</td>
<td>515</td>
</tr>
<tr>
<td>_os_getime()</td>
<td>516</td>
</tr>
<tr>
<td>_os9_getime()</td>
<td>517</td>
</tr>
<tr>
<td>_os_get_ioproc()</td>
<td>519</td>
</tr>
<tr>
<td>_os_get_mdp()</td>
<td>520</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>_os_get_moddir()</td>
<td>521</td>
</tr>
<tr>
<td>_os_getpd()</td>
<td>523</td>
</tr>
<tr>
<td>_os_get_prdesc()</td>
<td>524</td>
</tr>
<tr>
<td>_os_get_prsrc()</td>
<td>525</td>
</tr>
<tr>
<td>_os_get_prtbl()</td>
<td>526</td>
</tr>
<tr>
<td>_os_getstat()</td>
<td>527</td>
</tr>
<tr>
<td>_os_getsys()</td>
<td>528</td>
</tr>
<tr>
<td>_os_gprdesc()</td>
<td>530</td>
</tr>
<tr>
<td>_os_gregorian()</td>
<td>531</td>
</tr>
<tr>
<td>_os9 gs_cdfd()</td>
<td>532</td>
</tr>
<tr>
<td>_os9 gs_cypd()</td>
<td>533</td>
</tr>
<tr>
<td>_os9 gs_cstats()</td>
<td>534</td>
</tr>
<tr>
<td>_os9 gs_devnm()</td>
<td>535</td>
</tr>
<tr>
<td>_os9 gs_devtyp()</td>
<td>536</td>
</tr>
<tr>
<td>_os9 gs_diskfree()</td>
<td>537</td>
</tr>
<tr>
<td>_os gs_dopt()</td>
<td>538</td>
</tr>
<tr>
<td>_os gs_dsizel()</td>
<td>539</td>
</tr>
<tr>
<td>_os gs_edt()</td>
<td>540</td>
</tr>
<tr>
<td>_os gs_eof()</td>
<td>541</td>
</tr>
<tr>
<td>_os gs_fd()</td>
<td>542</td>
</tr>
<tr>
<td>_os gs_fdaddr()</td>
<td>543</td>
</tr>
<tr>
<td>_os gs_fdinf()</td>
<td>544</td>
</tr>
<tr>
<td>_os9 gs_free()</td>
<td>545</td>
</tr>
<tr>
<td>_os gs_luopt()</td>
<td>546</td>
</tr>
<tr>
<td>_os gs_parity()</td>
<td>547</td>
</tr>
<tr>
<td>_os gs_popt()</td>
<td>548</td>
</tr>
<tr>
<td>_os gs_pos()</td>
<td>549</td>
</tr>
<tr>
<td>_os9 gs_pump()</td>
<td>550</td>
</tr>
<tr>
<td>_os gs_ready()</td>
<td>551</td>
</tr>
<tr>
<td>_os gs_size()</td>
<td>552</td>
</tr>
<tr>
<td>_os id()</td>
<td>553</td>
</tr>
<tr>
<td>_os9 id()</td>
<td>555</td>
</tr>
<tr>
<td>_os_initdata()</td>
<td>556</td>
</tr>
<tr>
<td>_os_intercept()</td>
<td>557</td>
</tr>
<tr>
<td>_os ioconfig()</td>
<td>559</td>
</tr>
<tr>
<td>_os iodel()</td>
<td>560</td>
</tr>
<tr>
<td>_os ioexit()</td>
<td>561</td>
</tr>
<tr>
<td>_os_iofork()</td>
<td>562</td>
</tr>
<tr>
<td>_os9 ioqueue()</td>
<td>563</td>
</tr>
<tr>
<td>_os irq()</td>
<td>564</td>
</tr>
<tr>
<td>_os9 irq()</td>
<td>566</td>
</tr>
<tr>
<td>_os julian()</td>
<td>568</td>
</tr>
<tr>
<td>os9kexec()</td>
<td>569</td>
</tr>
<tr>
<td>_os link()</td>
<td>573</td>
</tr>
<tr>
<td>_os linkm()</td>
<td>575</td>
</tr>
<tr>
<td>_os load()</td>
<td>576</td>
</tr>
<tr>
<td>_os_loadp()</td>
<td>578</td>
</tr>
<tr>
<td>_os makdir()</td>
<td>580</td>
</tr>
<tr>
<td>_os makmdir()</td>
<td>582</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>_os_mem()</td>
<td>583</td>
</tr>
<tr>
<td>_os_mkmodule()</td>
<td>584</td>
</tr>
<tr>
<td>_os_modaddr()</td>
<td>586</td>
</tr>
<tr>
<td>_os_move()</td>
<td>587</td>
</tr>
<tr>
<td>_os_nproc()</td>
<td>588</td>
</tr>
<tr>
<td>_os_open()</td>
<td>589</td>
</tr>
<tr>
<td>_os9_panic()</td>
<td>590</td>
</tr>
<tr>
<td>_os_permit()</td>
<td>591</td>
</tr>
<tr>
<td>_os_perr()</td>
<td>593</td>
</tr>
<tr>
<td>_os_protect()</td>
<td>594</td>
</tr>
<tr>
<td>_os_prsnam()</td>
<td>595</td>
</tr>
<tr>
<td>_os_rdlst()</td>
<td>597</td>
</tr>
<tr>
<td>_os_read()</td>
<td>598</td>
</tr>
<tr>
<td>_os_readln()</td>
<td>600</td>
</tr>
<tr>
<td>_os_rellk()</td>
<td>601</td>
</tr>
<tr>
<td>_os9_retpd()</td>
<td>602</td>
</tr>
<tr>
<td>_os_rte()</td>
<td>603</td>
</tr>
<tr>
<td>_os_rtnprc()</td>
<td>604</td>
</tr>
<tr>
<td>_os9_salarm_atdate()</td>
<td>605</td>
</tr>
<tr>
<td>_os_salarm_atime()</td>
<td>607</td>
</tr>
<tr>
<td>_os9_salarm_atjul()</td>
<td>609</td>
</tr>
<tr>
<td>_os_salarm_cycle()</td>
<td>611</td>
</tr>
<tr>
<td>_os9_salarm_cycle()</td>
<td>613</td>
</tr>
<tr>
<td>_os_salarm_delete()</td>
<td>615</td>
</tr>
<tr>
<td>_os_salarm_delete_sp()</td>
<td>616</td>
</tr>
<tr>
<td>_os_salarm_reset()</td>
<td>617</td>
</tr>
<tr>
<td>_os_salarm_set()</td>
<td>619</td>
</tr>
<tr>
<td>_os9_salarm_set()</td>
<td>621</td>
</tr>
<tr>
<td>_os_scache()</td>
<td>622</td>
</tr>
<tr>
<td>_os9_schbit()</td>
<td>625</td>
</tr>
<tr>
<td>_os_seek()</td>
<td>627</td>
</tr>
<tr>
<td>_os_sema_init()</td>
<td>628</td>
</tr>
<tr>
<td>_os_sema_p()</td>
<td>629</td>
</tr>
<tr>
<td>_os_sema_term()</td>
<td>630</td>
</tr>
<tr>
<td>_os_sema_v()</td>
<td>631</td>
</tr>
<tr>
<td>_os_send()</td>
<td>632</td>
</tr>
<tr>
<td>_os_setcrrc()</td>
<td>633</td>
</tr>
<tr>
<td>_os_setime()</td>
<td>634</td>
</tr>
<tr>
<td>_os9_setime()</td>
<td>635</td>
</tr>
<tr>
<td>_os_setpr()</td>
<td>636</td>
</tr>
<tr>
<td>_os_setstat()</td>
<td>637</td>
</tr>
<tr>
<td>_os_setsvc()</td>
<td>638</td>
</tr>
<tr>
<td>_os9_setsvc()</td>
<td>640</td>
</tr>
<tr>
<td>_os_setsys()</td>
<td>641</td>
</tr>
<tr>
<td>_os_setuid()</td>
<td>642</td>
</tr>
<tr>
<td>_os_sgetstat()</td>
<td>643</td>
</tr>
<tr>
<td>_os_sgs_devnm()</td>
<td>644</td>
</tr>
<tr>
<td>_os_siglnji()</td>
<td>645</td>
</tr>
<tr>
<td>_os_sigmask()</td>
<td>647</td>
</tr>
</tbody>
</table>
_os_sigreset() ................................................................. 649
_os_sigs() .................................................................. 650
_os_sleep() ............................................................... 651
_os9_sleep() ............................................................. 653
_os_slink() ................................................................ 654
_os_slinkm() ............................................................. 656
_os_sqmem() ............................................................... 657
_os9_sqmem() .............................................................. 659
_os_srtmem() .............................................................. 660
_os_ss_attr() ............................................................... 661
_os_ss_break() ............................................................ 662
_os_ss_cache() ............................................................. 663
_os9_ss_cache() ............................................................ 664
_os_ss_close() ............................................................. 665
_os_ss_dcoff() ............................................................. 666
_os_ss_dcon() ............................................................... 667
_os_ss_dopt() ............................................................... 668
_os_ss_dsrtsl() ............................................................. 669
_os_ss_enpts() ............................................................. 669
_os_ss_erase() ............................................................. 670
_os_ss_fd() ................................................................ 671
_os_ss_fillbuff() .......................................................... 672
_os_ss_flushmap() ......................................................... 673
_os_ss_hdlink() ............................................................. 674
_os_ss_link_adj() .......................................................... 675
_os_ss_lock() ............................................................... 676
_os_ss_luopt() ............................................................... 677
_os9_ss_open() ............................................................. 678
_os_ss_pop() ................................................................ 679
_os_ss_relea() .............................................................. 680
_os_ss_rename() ............................................................ 681
_os_ss_resets() ............................................................. 682
_os_ss_reten() .............................................................. 683
_os_ss_rfm() ............................................................... 684
_os_ss_sendsig() .......................................................... 685
_os_ss_size() ............................................................... 686
_os_ss_skip() ............................................................... 687
_os_ss_skipend() .......................................................... 688
_os_ss_ticks() ............................................................... 689
_os_ss_wfm() ............................................................... 690
_os_ss_wtrack() ............................................................. 691
_os_strap() ................................................................ 693
_os9_strap() ................................................................. 696
_os_suspend() .............................................................. 698
_os_sysdbg() ............................................................... 699
_os_sysid() ................................................................. 700
_os_thexit() ................................................................. 702
_os_thfork() ................................................................. 703
_os_thread() ............................................................... 705
_os_tlink() ................................................................. 706
_os_tlinkm() ............................................................................................................................. 708
_os_tranpn() ............................................................................................................................. 709
_os_transadd() .......................................................................................................................... 710
_os9_translate() ....................................................................................................................... 712
_os_uacct() .................................................................................................................................. 713
_os_unlink() .................................................................................................................................. 714
_os_unload() .................................................................................................................................. 715
_os_usema_create() ...................................................................................................................... 716
_os_usema_delete() ...................................................................................................................... 718
_os_usema_link() .......................................................................................................................... 719
_os_usema_p() ............................................................................................................................... 720
_os_usema_reset() ........................................................................................................................ 721
_os_usema_try_p() ........................................................................................................................ 722
_os_usema_unlink() ...................................................................................................................... 723
_os_usema_v() .................................................................................................................................. 724
_os_vmodul() ................................................................................................................................... 725
_os9_vmodul() ................................................................................................................................ 726
_os_wait() ........................................................................................................................................ 727
_os_waitid() .................................................................................................................................... 728
_os_waitlk() .................................................................................................................................... 730
_os_write() ...................................................................................................................................... 731
_os_writef() .................................................................................................................................... 732
_os_yield() ...................................................................................................................................... 733
outc() ............................................................................................................................................. 734
outl() .............................................................................................................................................. 735
outw() ............................................................................................................................................. 736
_parsepath() .................................................................................................................................... 737
pause() .......................................................................................................................................... 739
pclose() ......................................................................................................................................... 740
pererror() ....................................................................................................................................... 741
pffinit() ......................................................................................................................................... 742
pffinit() ......................................................................................................................................... 743
pipe() ............................................................................................................................................ 744
popen() ......................................................................................................................................... 745
pow() ............................................................................................................................................ 746
_parsealloc_rand() .......................................................................................................................... 747
prerr() ........................................................................................................................................... 748
_prgname() .................................................................................................................................... 750
printf() ......................................................................................................................................... 751
_prsnam() ....................................................................................................................................... 752
pthread_attr_destroy() ................................................................................................................. 753
pthread_attr_getdetachstate() ....................................................................................................... 754
(pthread_attr_getinitfunction() ..................................................................................................... 755
(pthread_attr_getpriority() ............................................................................................................ 757
pthread_attr_getstackaddr() .......................................................................................................... 758
pthread_attr_getstacksize() .......................................................................................................... 759
pthread_attr_init() ........................................................................................................................ 760
pthread_attr_setdetachstate() ......................................................................................................... 761
(pthread_attr_setinitfunction() .................................................................................................. 762
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_pthread_setpr()</td>
<td>829</td>
</tr>
<tr>
<td>_pthread_setsignalrange()</td>
<td>830</td>
</tr>
<tr>
<td>pthread_setspecific()</td>
<td>831</td>
</tr>
<tr>
<td>_pthread_setsuspendable()</td>
<td>832</td>
</tr>
<tr>
<td>_pthread_setunsuspendable()</td>
<td>834</td>
</tr>
<tr>
<td>_pthread_suspend()</td>
<td>835</td>
</tr>
<tr>
<td>pthread_testcancel()</td>
<td>837</td>
</tr>
<tr>
<td>putc(), putchar()</td>
<td>838</td>
</tr>
<tr>
<td>putenv()</td>
<td>839</td>
</tr>
<tr>
<td>puts()</td>
<td>840</td>
</tr>
<tr>
<td>putw()</td>
<td>841</td>
</tr>
<tr>
<td>qsort()</td>
<td>842</td>
</tr>
<tr>
<td>raise()</td>
<td>843</td>
</tr>
<tr>
<td>rand()</td>
<td>844</td>
</tr>
<tr>
<td>read()</td>
<td>845</td>
</tr>
<tr>
<td>readdir()</td>
<td>847</td>
</tr>
<tr>
<td>readln()</td>
<td>848</td>
</tr>
<tr>
<td>readdir()</td>
<td>850</td>
</tr>
<tr>
<td>realloc()</td>
<td>850</td>
</tr>
<tr>
<td>regcomp()</td>
<td>851</td>
</tr>
<tr>
<td>regerror()</td>
<td>852</td>
</tr>
<tr>
<td>regex(), regexec()</td>
<td>853</td>
</tr>
<tr>
<td>remove()</td>
<td>854</td>
</tr>
<tr>
<td>rename()</td>
<td>855</td>
</tr>
<tr>
<td>rewind()</td>
<td>856</td>
</tr>
<tr>
<td>rewinding()</td>
<td>857</td>
</tr>
<tr>
<td>rindex()</td>
<td>858</td>
</tr>
<tr>
<td>sbrk()</td>
<td>859</td>
</tr>
<tr>
<td>scanf()</td>
<td>860</td>
</tr>
<tr>
<td>sched_yield()</td>
<td>862</td>
</tr>
<tr>
<td>seekdir()</td>
<td>863</td>
</tr>
<tr>
<td>select()</td>
<td>864</td>
</tr>
<tr>
<td>_set &lt;name&gt;()</td>
<td>865</td>
</tr>
<tr>
<td>setbuf()</td>
<td>866</td>
</tr>
<tr>
<td>_setcsrc()</td>
<td>867</td>
</tr>
<tr>
<td>setenv()</td>
<td>868</td>
</tr>
<tr>
<td>set_except_base()</td>
<td>869</td>
</tr>
<tr>
<td>set_gdtr()</td>
<td>870</td>
</tr>
<tr>
<td>setgid()</td>
<td>871</td>
</tr>
<tr>
<td>setitimer()</td>
<td>872</td>
</tr>
<tr>
<td>setlocale()</td>
<td>873</td>
</tr>
<tr>
<td>setp()</td>
<td>874</td>
</tr>
<tr>
<td>setpnext()</td>
<td>875</td>
</tr>
<tr>
<td>setjmp()</td>
<td>876</td>
</tr>
<tr>
<td>setlocale()</td>
<td>877</td>
</tr>
<tr>
<td>setp()</td>
<td>879</td>
</tr>
<tr>
<td>setpnext()</td>
<td>880</td>
</tr>
<tr>
<td>setstat()</td>
<td>881</td>
</tr>
<tr>
<td>_setsys()</td>
<td>882</td>
</tr>
<tr>
<td>setuid()</td>
<td>883</td>
</tr>
<tr>
<td>setvbuf()</td>
<td>885</td>
</tr>
</tbody>
</table>
sign() .......................................................................................................................................... 890
signal() .................................................................................................................................... 891
sin() .......................................................................................................................................... 894
sinh() ........................................................................................................................................ 895
sl_add(), sl_find(), sl_free(), sl_init() .................................................................................. 896
sleep() ...................................................................................................................................... 898
_sliblink() ................................................................................................................................ 900
snprintf() ............................................................................................................................... 900
sprintf() .................................................................................................................................... 901
sqrt() ......................................................................................................................................... 902
srand() ....................................................................................................................................... 903
srqcmem() .................................................................................................................................. 904
_srqmem() ................................................................................................................................... 906
_srtmem() ................................................................................................................................... 907
_sss_attr() .................................................................................................................................. 908
sscanf() ...................................................................................................................................... 909
_ssd_coff() ............................................................................................................................... 910
_ssd_con() ................................................................................................................................. 911
_ssd_dsrs() ................................................................................................................................ 912
_sss_enrts() ............................................................................................................................. 913
_sss_lock() .................................................................................................................................. 914
_sss_opt() .................................................................................................................................... 915
_sss_pfd() .................................................................................................................................... 917
_sss_rel() .................................................................................................................................... 919
_sss_rest() ................................................................................................................................. 920
_sss_size() .................................................................................................................................. 921
_sss_ssig() ................................................................................................................................... 922
_sstiks() ...................................................................................................................................... 923
_sst_wtrk() ............................................................................................................................... 924
stacksiz(), _stacksiz() ............................................................................................................... 926
stat() .......................................................................................................................................... 927
_strass() ..................................................................................................................................... 928
strcasecmp(), strncasecmp() ....................................................................................................... 929
strcat() ....................................................................................................................................... 930
strchr() ....................................................................................................................................... 931
strcmp() ....................................................................................................................................... 932
strcoll() ....................................................................................................................................... 933
strcpy() ....................................................................................................................................... 934
strcspn() ..................................................................................................................................... 935
strdup() ....................................................................................................................................... 936
strerror() ..................................................................................................................................... 937
strftime() ..................................................................................................................................... 938
strerror() ..................................................................................................................................... 939
strdup() ....................................................................................................................................... 940
strlcpy ......................................................................................................................................... 942
strlcat() ....................................................................................................................................... 943
strncpy() ...................................................................................................................................... 944
strncmp() ..................................................................................................................................... 945
strncpy() ...................................................................................................................................... 946
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>strpbrk()</td>
<td>947</td>
</tr>
<tr>
<td>strftime()</td>
<td>948</td>
</tr>
<tr>
<td>strchr()</td>
<td>949</td>
</tr>
<tr>
<td>strchr()</td>
<td>950</td>
</tr>
<tr>
<td>strsep()</td>
<td>951</td>
</tr>
<tr>
<td>strspn()</td>
<td>952</td>
</tr>
<tr>
<td>strstr()</td>
<td>953</td>
</tr>
<tr>
<td>strtok()</td>
<td>954</td>
</tr>
<tr>
<td>strtoll()</td>
<td>955</td>
</tr>
<tr>
<td>strtof()</td>
<td>956</td>
</tr>
<tr>
<td>strxfrm()</td>
<td>957</td>
</tr>
<tr>
<td>toascii()</td>
<td>958</td>
</tr>
<tr>
<td>toascii()</td>
<td>959</td>
</tr>
<tr>
<td>_tolower()</td>
<td>960</td>
</tr>
<tr>
<td>_tolower()</td>
<td>961</td>
</tr>
<tr>
<td>_toupper()</td>
<td>962</td>
</tr>
<tr>
<td>_toupper()</td>
<td>963</td>
</tr>
<tr>
<td>_tolower()</td>
<td>964</td>
</tr>
<tr>
<td>_toupper()</td>
<td>965</td>
</tr>
<tr>
<td>sys_mktime()</td>
<td>966</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>967</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>968</td>
</tr>
<tr>
<td>_sys_date()</td>
<td>969</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>970</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>971</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>972</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>973</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>974</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>975</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>976</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>977</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>978</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>979</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>980</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>981</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>982</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>983</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>984</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>985</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>986</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>987</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>988</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>989</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>990</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>991</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>992</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>993</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>994</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>995</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>996</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>997</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>998</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>999</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>1000</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>1001</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>1002</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>1003</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>1004</td>
</tr>
<tr>
<td>_sysdate()</td>
<td>1005</td>
</tr>
</tbody>
</table>
Ultra C library calls perform a variety of functions, from classifying characters to managing memory. This section contains information helpful for using the Ultra C library. This information includes:

- Using Functions and Macros
- Library Files
- Program Globals
- Header Files
- The C Library
- C Shared Library
Using Functions and Macros

Ultra C supports both functions and macros. A function consists of compiled C statements. A macro is an identifier defined to represent a value or expression.

Textual replacement in macro expansion can cause arguments of macros to be evaluated repeatedly. Macro implementations of ANSI Standard Library functions are required to evaluate arguments exactly once and to have sufficient parentheses to avoid operator precedence trouble.

You can bypass the macro replacement when you compile. This may be desirable when a large macro is used many times and the space saved by calling the function is outweighed by the extra time needed to call the function. For example, to override the replacement/instantiation of the `putc` macro use:

```c
(putc)(c, fp);
```

or

```c
#undef putc
```

This results in a call to the `putc` function in the library rather than the replacement of the `putc` macro.

All American National Standards Institute (ANSI) macros have a library counterpart of the same name, with the exception of the following:

- `assert()`
- `setjmp()`
- `va_arg()`
- `va_end()`
- `va_start()`

This is not true for non-ANSI macros.

Library Files

The following are important files used during the compilation process. These files are located in the `MROS` structure under `LIB` for the hardware platform. Ultra C libraries are listed in Table 1-1.

<table>
<thead>
<tr>
<th>Library</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clib.l</code></td>
<td>The ANSI standard library</td>
</tr>
<tr>
<td><code>cstart.r</code></td>
<td>Start-up code for compiled programs (with <code>-bc</code> option K &amp; R compatible source mode)</td>
</tr>
<tr>
<td><code>acstart.r</code></td>
<td>By default (same as <code>cstart.r</code> but compiled with ANSI defined)</td>
</tr>
<tr>
<td><code>acstarta.r</code></td>
<td><code>icode</code> linked version of the above</td>
</tr>
<tr>
<td><code>ansi_cstart.r</code></td>
<td>Same as <code>acstart.r</code>, mainly for the 8.3 filenames on DOS</td>
</tr>
</tbody>
</table>
Program Globals

Ultra C defines a number of globals for use by library functions and/or the application. They are divided into two classes:

1. Globals that are unlikely to change from release to release, and
2. Globals that are used by library functions that may change from release to release (see header file cglob.h for current information).

Class 1 Globals

\begin{itemize}
\item \texttt{u_int32 \_totmem} \\
    Total number of bytes that are in the data area for the program (includes stack, global data, and parameters)
\item \texttt{u_int16 \_pathcnt} \\
    Number of open paths the process inherited from the parent process (usually three).
\end{itemize}
void *_sysglob
    Pointer to the kernel's system globals. This variable is only initialized when
the program is a system state process (otherwise it is NULL). OS-9 for 68K
only.

 mh_exec *_modhead
    Pointer to the primary module for the current process

char * _modname
    Pointer to the module name for the current process (unmodifiable).

u_int32 _procid
    Process ID of the current process

u_int32 _maxstack
    Maximum number of bytes consumed from the stack so far. This variable
does not contain a useful value unless stack checking is enabled. Print this
variable prior to exiting to examine how much stack space your application
required.

void *_glob_data
    Pointer to the current processes global static storage. _glob_data can be used
as an argument to _os_intercept().

char **_environ
    Pointer to the environment variable strings for the current process. This
variable is used when forking other processes to pass the current environment
along.

char **environ
    Same as _environ, except that it is available only in K & R source mode for
OS-9 for 68K. environ is provided for compatibility with Microware K & R
C-compiler.

Class 2 Globals

void * _sttop
    Address of the top of the stack.

void * _mtop
    Address of data memory. _mtop is used to check for stack overflow.

void * _stbot
    Deepest stack pointer so far.

void * _csl_a6
    Global static storage pointer for attached csl. OS-9 for 68K only.

void * _csl_glob
    Global static storage pointer for attached csl. OS-9 only.

u_int32 _sbsize
    Size of process memory area. _sbsize is used by sbrk().
FILE (*_fcbs)[ ]
    Pointer to _niob array.

void *ofcbs
    Pointer to compatibility _iob array.

int32 _stklimit
    Amount of verified area left on stack. This value contains the number of bytes
that can be consumed from the stack before further checking needs to be
done.

_mainid
    The process ID of the original process.

    For non-threaded programs, _mainid==procid. For threaded programs, one
thread is _mainid==_procid and all others contain different IDs.

Header Files

Most of the Ultra C library calls are declared in header files. A header file declares a
set of related functions and includes any necessary types and additional macros
needed to use the functions. The contents of these header files are available through
the #include preprocessing directive. Header files may be included in any order.

Ultra C provides the following header files.

Table 1-2. Ultra C Header Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm.h</td>
<td>Used by functions that use OS-9 alarm functions. alarm.h contains prototypes for:</td>
</tr>
<tr>
<td></td>
<td>alm_atdate() alm_atjul()</td>
</tr>
<tr>
<td></td>
<td>alm_cycle() alm_delete()</td>
</tr>
<tr>
<td></td>
<td>alm_set() _os9_alarm_atdate()</td>
</tr>
<tr>
<td></td>
<td>_os_alarm_atime() _os9_alarm_atjul()</td>
</tr>
<tr>
<td></td>
<td>_os_alarm_cycle() _os_alarm_delete()</td>
</tr>
<tr>
<td></td>
<td>_os_alarm_reset() _os_alarm_set()</td>
</tr>
<tr>
<td></td>
<td>_os9_salarm_atdate() _os_salarm_atime()</td>
</tr>
<tr>
<td></td>
<td>_os9_salarm_cycle() _os_salarm_delete()</td>
</tr>
<tr>
<td></td>
<td>_os_salarm_reset() _os_salarm_set()</td>
</tr>
<tr>
<td></td>
<td>_os9_salarm_set()</td>
</tr>
<tr>
<td>assert.h</td>
<td>Defines the assert macro. assert.h contains prototypes for:</td>
</tr>
<tr>
<td></td>
<td>assert()</td>
</tr>
<tr>
<td>aton.h</td>
<td>Used for converting alphabetic characters to numeric values. aton.h contains prototypes for the following function:</td>
</tr>
<tr>
<td></td>
<td>_aton()</td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache.h</td>
<td>Defines the CPU internal cache control information. cache.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>_os_cache()</td>
</tr>
<tr>
<td></td>
<td>_os_scache()</td>
</tr>
<tr>
<td>cdi.h</td>
<td>Contains CD-I specific functions. cdi.h contains prototypes for the following function:</td>
</tr>
<tr>
<td>(OS-9 for 68K only)</td>
<td>_os9_gs_cdfd()</td>
</tr>
<tr>
<td>cglob.h</td>
<td>Declares Ultra C global variables.</td>
</tr>
<tr>
<td>ctype.h</td>
<td>Declares functions that are useful for testing and mapping characters. ctype.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>isalnum(isalpha)</td>
</tr>
<tr>
<td></td>
<td>_isascii(), isascii()iscntrl()</td>
</tr>
<tr>
<td></td>
<td>isdigit()_isec_kana()</td>
</tr>
<tr>
<td></td>
<td>_isuc1(), _isuc2()isgraph()</td>
</tr>
<tr>
<td></td>
<td>_isjis_kana()islower()</td>
</tr>
<tr>
<td></td>
<td>isprint()ispunct()</td>
</tr>
<tr>
<td></td>
<td>_issjis1()_issjis2()</td>
</tr>
<tr>
<td></td>
<td>isspace()isupper()</td>
</tr>
<tr>
<td></td>
<td>isxdigit()toascii(), _toascii()</td>
</tr>
<tr>
<td></td>
<td>tolower()_tolower()</td>
</tr>
<tr>
<td></td>
<td>toupper()_toupper()</td>
</tr>
<tr>
<td>dexec.h</td>
<td>Defines information related to debugger controlled processes. dexec.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>_os_dexec()_os_dexit()</td>
</tr>
<tr>
<td></td>
<td>_os_dfork()_os_dforkm()</td>
</tr>
<tr>
<td></td>
<td>_os9_dfork()</td>
</tr>
<tr>
<td>dir.h</td>
<td>Defines structures and functions for BSD 4.2 directory calls. dir.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>closedir()opendir()</td>
</tr>
<tr>
<td></td>
<td>readdir()rewinddir()</td>
</tr>
<tr>
<td></td>
<td>seekdir()telldir()</td>
</tr>
<tr>
<td>direct.h</td>
<td>Defines the RBF data structures. This is obsolete; it has been replaced with rbf.h. OS-9 for 68k only.</td>
</tr>
<tr>
<td>err.h</td>
<td>Contains definitions for dealing with warnings and errors. This file contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>err(), errx()verr(), verrx()</td>
</tr>
<tr>
<td></td>
<td>vwarn(), vwarnx()warn(), warnx()</td>
</tr>
<tr>
<td>errno.h</td>
<td>Defines macros relating to error reporting. errno.h contains a prototype for the following function:</td>
</tr>
<tr>
<td>†</td>
<td>_os_perr()</td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>events.h</td>
<td>Used for functions using the OS-9 event functions. events.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>_ev_creat()_ev_delete()</td>
</tr>
<tr>
<td></td>
<td>_ev_info()_ev_link()</td>
</tr>
<tr>
<td></td>
<td>_ev_pulse()_ev_read()</td>
</tr>
<tr>
<td></td>
<td>_ev_set()_ev_setr()</td>
</tr>
<tr>
<td></td>
<td>_ev_signal()_ev_unlink()</td>
</tr>
<tr>
<td></td>
<td>_ev_wait()_ev_waitr()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_allclr()_os_ev_allset()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_anycr()_os_ev_anyset()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_change()_os_ev_creat()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_delete()_os_ev_info()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_link()_os_ev_pulse()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_read()_os_ev_set()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_setand()_os_ev_setor()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_setr()_os_ev_setxor()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_signal()_os_ev_tstset()</td>
</tr>
<tr>
<td></td>
<td>_os_ev_unlink()_os_ev_wait()</td>
</tr>
<tr>
<td></td>
<td>_os9_ev_wait()_os_ev_waitr()</td>
</tr>
<tr>
<td></td>
<td>_os9_ev_waitr()_os9_ev_info()</td>
</tr>
<tr>
<td>float.h</td>
<td>† Defines the limits for floating point numbers.</td>
</tr>
<tr>
<td>funcs.h</td>
<td>Symbolic names for the system calls. funcs.h contains a prototype for the following function:</td>
</tr>
<tr>
<td></td>
<td>_os9_setsvc()</td>
</tr>
<tr>
<td>getset.h</td>
<td>Used to read and set values of special purpose registers, device control registers, processor registers, and time-based registers. getset.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td><em>get</em>&lt;name&gt;()</td>
</tr>
<tr>
<td></td>
<td><em>set</em>&lt;name&gt;()</td>
</tr>
<tr>
<td>glob.h</td>
<td>Contains definitions for generating file lists using regular expressions. This file contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>glob(), globfree()</td>
</tr>
<tr>
<td>io.h</td>
<td>Used for functions performing input/output. io.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>_os_alias()_os_attach()</td>
</tr>
<tr>
<td></td>
<td>_os_detach()_os_get_ioproc()</td>
</tr>
<tr>
<td></td>
<td>_os_getdl()_os_getpd()</td>
</tr>
<tr>
<td></td>
<td>_os_rdalst()_os_tranpn()</td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioctl.h</td>
<td>Contains definitions for ioctl() library function.</td>
</tr>
</tbody>
</table>
| ioedt.h  | Defines the current I/O interface editions. ioedt.h contains a prototype for the following function:  
|          | _os_gs_edt()                                                                |
| limits.h | Defines macros that expand to various limits and parameters.                |
| locale.h | Used for localization. locale.h contains prototypes for the following functions:  
|          | localeconv()                                                                |
|          | setlocale()                                                                 |
| lock.h   | Used for the OS-9 resource lock functions. lock.h contains prototypes for the following functions:  
|          | _os_acqlk()_os_caqlk()                                                      |
|          | _os_crlk()_os_dellk()                                                       |
|          | _os_rellk()_os_uwaitlk()                                                    |
| math.h   | Defines the mathematical functions and macros available with Ultra C. math.h contains prototypes for the following functions:  
|          | acos()asin()                                                                |
|          | atan()atan2()                                                                |
|          | ceil()cos()                                                                  |
|          | cosh()exp()                                                                  |
|          | fabs()floor()                                                                |
|          | fmod()frexp()                                                                |
|          | hypot()ldexp()                                                               |
|          | log()log10()                                                                |
|          | modf()pow()                                                                  |
|          | sin()sinh()                                                                  |
|          | sqrt()tan()                                                                  |
|          | tanh()                                                                      |
| memory.h | Functions related to acquiring, examining, and releasing memory. memory.h contains prototypes for:  
|          | _os9_allbit()_os9_delbit()                                                   |
|          | _os_get_blkmap()_os_mem()                                                    |
|          | _os_move()_os9_schbit()                                                      |
|          | _os_srqmem()_os9_sqrmem()                                                    |
|          | _os_srtmem()_os9_translate()                                                 |
|          | _srqmem()_srtmem()                                                           |
|          | sqrcmem()                                                                    |

† Indicates a standard ANSI header.
### Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
</table>
| moddir.h | Module directory information. moddir.h contains prototypes for the following functions:  
|         | _os_altmdir()_os_chmdir()                                                   |
|         | _os_cmdperm()_os_delmdir()                                                  |
|         | _os_fmod()_os_get_mdp()                                                     |
|         | _os_makmdir()                                                               |
|         | † Indicates a standard ANSI header.                                         |
| modes.h | Defines the access modes and permissions for files and directories. modes.h contains prototypes for:  
|         | access() attach()                                                          |
|         | chdir() chown()                                                             |
|         | chmod() chxdir()                                                            |
|         | close() creat()                                                             |
|         | create() detach()                                                           |
|         | dup() lseek()                                                               |
|         | mkdir() mknod()                                                             |
|         | open() _os_chdir()                                                          |
|         | _os_close() _os_create()                                                     |
|         | _os9_create() _os_delete()                                                   |
|         | _os_dup() _os_ioconfig()                                                     |
|         | _os_makdir() _os_open()                                                      |
|         | _os_read() _os_readln()                                                      |
|         | _os_seek() _os9_ss_close()                                                   |
|         | _os9_ss_open() _os_write()                                                   |
|         | _os_writeln() read()                                                        |
|         | readln() unlink()                                                            |
|         | unlinkx() write()                                                            |
|         | writeln()                                                                   |
### Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>module.h</strong></td>
<td>Module-related data structures and functions. <code>module.h</code> contains prototypes for:</td>
</tr>
<tr>
<td></td>
<td><code>crc()</code> <code>get_module_dir()</code> <code>make_module()</code> <code>modcload()</code> <code>moddata_module()</code> <code>modlink()</code> <code>modload()</code> <code>modloadp()</code> <code>munlink()</code> <code>munload()</code> <code>os_crc()</code> <code>os_datmod()</code> <code>os_get_moddir()</code> <code>os_initdata()</code> <code>os_iodel()</code> <code>os_link()</code> <code>os_linkm()</code> <code>os_load()</code> <code>os_loadp()</code> <code>os_mkmodule()</code> <code>os_modaddr()</code> <code>os_setcrc()</code> <code>os_slink()</code> <code>os_slinkm()</code> <code>os_tlinkm()</code> <code>os_tlink()</code> <code>os_unlink()</code> <code>os_unload()</code> <code>os_vmodul()</code> <code>os9_vmodul()</code> <code>prgname()</code> <code>setcrc()</code> <code>sliblink()</code> <code>subcall()</code></td>
</tr>
<tr>
<td><strong>mqueue.h</strong></td>
<td>POSIX messaging data structures and functions. <code>mqueue.h</code> contains prototypes for:</td>
</tr>
<tr>
<td></td>
<td><code>mq_close</code> <code>mq_getattr</code> <code>mq_notify</code> <code>mq_notify_write</code> <code>mq_open</code> <code>mq_receive</code> <code>mq_send</code> <code>mq_setattr</code> <code>mq_unlink</code></td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
### Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>os9def</strong></td>
<td>A series of macros mapping UNIX macros and function calls to their OS-9 versions.</td>
</tr>
<tr>
<td></td>
<td><code>alarm()</code> <code>alloca()</code> <code>bcmp()</code> <code>bcopy()</code> <code>buildpath()</code> <code>bzero()</code> <code>cbrt()</code> <code>crtolf()</code> <code>dup2()</code> <code>exec1()</code> <code>execle()</code> <code>execv()</code> <code>execve()</code> <code>execvp()</code> <code>ffs()</code> <code>getgid()</code> <code>getopt()</code> <code>getppid()</code> <code>getcwd()</code> <code>getwd()</code> <code>ioctl()</code> <code>isatty()</code> <code>lftocr()</code> <code>pclose()</code> <code>pipe()</code> <code>popen()</code> <code>putenv()</code> <code>readv()</code> <code>setgid()</code> <code>tempnam()</code> <code>writev()</code></td>
</tr>
<tr>
<td><strong>os9time.h</strong></td>
<td>Defines interval timer support for OS-9. (UNIX)</td>
</tr>
<tr>
<td></td>
<td><code>gettimeofday()</code> <code>setitimer()</code></td>
</tr>
<tr>
<td><strong>os9types.h</strong></td>
<td>Synchronous I/O multiplexing. <em>os9types.h</em> contains a prototype for the following function:</td>
</tr>
<tr>
<td></td>
<td><code>select()</code></td>
</tr>
<tr>
<td><strong>pmodes.h</strong></td>
<td>POSIX file access and permission modes for OS9000.</td>
</tr>
<tr>
<td></td>
<td>† Indicates a standard ANSI header.</td>
</tr>
</tbody>
</table>
### Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
</table>
| process.h | Process descriptor and functions that manipulate processes. process.h contains prototypes for: chainc(), chain_cpymem()  
getpid()_get_process_desc()  
_get_process_table()getuid()_os9_allpd()  
os_alltsk()_os_alocproc()  
os_aproc()_os9_aproc()  
os_chain()_os_chainm()  
os_chkmem()_os_cpy_ioproc()  
os_cpymem()_os_ddlk()  
os_deltsk()_os_exec()  

os9exec()_os_exit()  
os_findpd()_os9_findpd()  
os_fork()os9fork(), os9forkc()  
os_forkm()_os_get_prtbl()  
os_gprdsc()_os9_gspump()  
os_id()_os9_id()  
os_icexit()_os_iofork()  
os9_ioqueue()_os_nproc()  
os9_panic()_os_permit()  
os_protect()_os9_retpd()  
os rtnprc()_os_setpr()  
os_setuid()_os_suspend()  
os_sysdbg()_os_uacct()  
os_wait()setpr()  
setuid()_sysdbg()  
wait()_os_waitid() |
| procid.h | Process descriptor data structure. procid.h is obsolete; it has been replaced with process.h. |

† Indicates a standard ANSI header.
Chapter 1: Overview

pthreads.h

POSIX thread support routines. The following functions are included in pthread.h:

- pthread_attr_destroy()
- pthread_attr_getdetachstate()
- _pthread_attr_getinitfunction()
- _pthread_attr_getpriority()
- pthread_attr_getstackaddr()
- pthread_attr_setdetachstate()
- _pthread_attr_setinitfunction()
- _pthread_attr_setpriority()
- pthread_attr_setstackaddr()
- pthread_attr_setstacksize()
- pthread_cleanup_push()
- pthread_cond_broadcast()
- pthread_cond_signal()
- pthread_condattr_destroy()
- pthread_condattr_getpshared()
- pthread_condattr_init()
- pthread_condattr_setpshared()
- _pthread_interrupt()
- pthread_mutex_destroy()
- pthread_mutex_getprioceiling()
- pthread_attr_getstacksize()
- pthread_attr_init()
- pthread_cancel()
- pthread_cleanup_pop()
- pthread_cond_timedwait()
- pthread_cond_wait()
- _pthread_interrupt_clear()
- pthread_join()

† Indicates a standard ANSI header.
### Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pthread.h</td>
<td>pthread_key_create()</td>
</tr>
<tr>
<td></td>
<td>pthread_key_delete()</td>
</tr>
<tr>
<td></td>
<td>pthread_create()</td>
</tr>
<tr>
<td></td>
<td>pthread_detach()</td>
</tr>
<tr>
<td></td>
<td>pthread_equal()</td>
</tr>
<tr>
<td></td>
<td>pthread_exit()</td>
</tr>
<tr>
<td></td>
<td>pthread_getspecific()</td>
</tr>
<tr>
<td></td>
<td>_pthread_getstatus()</td>
</tr>
<tr>
<td></td>
<td>pthread_cond_destroy()</td>
</tr>
<tr>
<td></td>
<td>pthread_cond_init()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_init()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_lock()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_setprioceiling()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_trylock()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutex_unlock()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_destroy()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_getprioceiling()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_getprotocol()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_getpshared()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_init()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_setprioceiling()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_setprotocol()</td>
</tr>
<tr>
<td></td>
<td>pthread_mutexattr_setpshared()</td>
</tr>
<tr>
<td></td>
<td>pthread_once()</td>
</tr>
<tr>
<td></td>
<td>_pthread_resume()</td>
</tr>
<tr>
<td></td>
<td>pthread_self()</td>
</tr>
<tr>
<td></td>
<td>pthread_setcancelstate()</td>
</tr>
<tr>
<td></td>
<td>pthread_setcanceltype()</td>
</tr>
<tr>
<td></td>
<td>_pthread_setpr()</td>
</tr>
<tr>
<td></td>
<td>_pthread_setsignalrange()</td>
</tr>
<tr>
<td></td>
<td>pthread_setspecific()</td>
</tr>
<tr>
<td></td>
<td>_pthread_setsuspendable()</td>
</tr>
<tr>
<td></td>
<td>_pthread_setunsuspendable()</td>
</tr>
<tr>
<td></td>
<td>_pthread_suspend()</td>
</tr>
<tr>
<td></td>
<td>pthread_testcancel()</td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pwd.h</td>
<td>Contains the password file definitions.</td>
</tr>
<tr>
<td></td>
<td>endpwent() fgetpwent()</td>
</tr>
<tr>
<td></td>
<td>getnextpwnam() getpw()</td>
</tr>
<tr>
<td></td>
<td>getpwent() getpwnam()</td>
</tr>
<tr>
<td></td>
<td>getpwuid() setpwent()</td>
</tr>
<tr>
<td>rbf.h</td>
<td>Used for functions using the random block file manager (RBF). rbf.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>_gs_gfd() _os_gs_cstats()</td>
</tr>
<tr>
<td></td>
<td>_os_gs_dsize() _os_gs_fd()</td>
</tr>
<tr>
<td></td>
<td>_os_gs_fdaddr() _os_gs_fdinf()</td>
</tr>
<tr>
<td></td>
<td>_os_gs_parity() _os_ss_cache()</td>
</tr>
<tr>
<td></td>
<td>_os_ss_fd() _os_ss_flushmap()</td>
</tr>
<tr>
<td></td>
<td>_os_ss_hdlink() _os_ss_lock()</td>
</tr>
<tr>
<td></td>
<td>_os_ss_ticks() _os_ss_wtrack()</td>
</tr>
<tr>
<td></td>
<td>_ss_lock() _ss_pfd()</td>
</tr>
<tr>
<td></td>
<td>_ss_ticks() _ss_wtrak()</td>
</tr>
<tr>
<td>regexp.h</td>
<td>Contains definitions for regular expression support.</td>
</tr>
<tr>
<td></td>
<td>regcomp() regerror()</td>
</tr>
<tr>
<td></td>
<td>regex(), regexec()</td>
</tr>
<tr>
<td>regs.h</td>
<td>Used to include the proper register header file for the target processor. regs.h contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>change_const() change_static()</td>
</tr>
<tr>
<td></td>
<td>get_const() get_static()</td>
</tr>
<tr>
<td></td>
<td>inc() inc()</td>
</tr>
<tr>
<td></td>
<td>inw() irq_change()</td>
</tr>
<tr>
<td></td>
<td>irq_disable() irq_enable()</td>
</tr>
<tr>
<td></td>
<td>irq_maskget() irq_restore()</td>
</tr>
<tr>
<td></td>
<td>irq_save() make_gdesc()</td>
</tr>
<tr>
<td></td>
<td>_os_cache() (OS-9 only) _os_firq() _os_irq()</td>
</tr>
<tr>
<td></td>
<td>_os9_irq() (OS-9 for 68K only) _os_scache() _os_sysid()</td>
</tr>
<tr>
<td></td>
<td>outc()</td>
</tr>
<tr>
<td></td>
<td>outl() outw()</td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
### Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
</table>
| sbf.h   | Used for functions using the sequential block file manager (SBF). sbf.h contains prototypes for the following functions:  
|         |   _os_ss_erase()_os_ss_rfm()                                                |
|         |   _os_ss_reten()_os_ss_skipend()                                            |
|         |   _os_ss_skip()_os_ss_wfm()                                                 |
| scf.h   | Used for functions using the sequential character file manager (SCF). scf.h contains prototypes for the following functions:  
|         |   _os_ss_dcoff()_os_ss_dcon()                                               |
|         |   _os_ss_dopt()_os_ss_dsrts()                                               |
|         |   _os_ss_enrts()_os_ss_fillbuff()                                           |
|         |   _ss_dcoff()_ss_dcon()                                                     |
|         |   _ss_dsrts()_ss_enrts()                                                    |
| sched.h | Used for the POSIX schedule family of functions. sched.h contains prototypes for the following function:  
|         |   sched_yield()                                                             |
| semaphore.h | Used for functions using binary semaphores. semaphore.h contains prototypes for the following functions:  
|           |   _os_sema_init()_os_sema_p()                                               |
|           |   _os_sema_term()_os_sema_v()                                               |
| setjmp.h † | Used for bypassing the normal function call and return mechanism. setjmp.h contains prototypes for the following functions:  
|           |   longjmp()                                                                 |
|           |   setjmp()                                                                  |
| setsys.h | System global symbolic names. This is obsolete. It has been replaced with sysglob.h. |
| settrap.h | Data structures and functions used to trap user-state exceptions. settrap.h contains prototypes for the following functions:  
|           |   _os_strap()                                                               |
|           |   _os9_strap()                                                              |

† Indicates a standard ANSI header.
## Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sg_codes.h</code></td>
<td>Symbolic names for <code>SetStats</code> and <code>GetStats</code>. <code>sg_codes.h</code> contains prototypes for the following functions:</td>
</tr>
</tbody>
</table>
|               | `getstat()` _gs_devn()`  
|               | _gs_eof()` _gs_opt()`  
|               | _gs_pos()` _gs_rdy()`  
|               | _gs_size()` _os_getstat()`  
|               | _os_gs_cpypd()` _os_gs_devnm()`  
|               | _os_gs_devtyp()` _os_gs_dopt()`  
|               | _os_gs_eof()` _os9_gs_free()`  
|               | _os_gs_luopt()` _os_gs_popt()`  
|               | _os_gs_pos()` _os_gs_ready()`  
|               | _os_gs_size()` _os_setstat()`  
|               | _os_sgetstat()` _os_sgs_devnm()`  
|               | _os_ss_attr()` _os_ss_break()`  
|               | _os_ss_dopt()` _os_ss_link_adj()`  
|               | _os_ss_luopt()` _os_ss_popt()`  
|               | _os_ss_relea()` _os_ss_rename()`  
|               | _os_ss_reset()` _os_ss_sendsig()`  
|               | _os_ss_size()` setstat()`  
|               | _ss_attr()` _ss_opt()`  
|               | _ss_rel()` _ss_rest()`  
|               | _ss_size()` _ss_ssig()`  
| `sgstat.h`    | Path descriptor option section definitions.                                 |
| `signal.h` † | Used for handling various signals. `signal.h` contains prototypes for the following functions:  |
|               | `intercept()` `kill()`  
|               | _os_clrsigs()` _os_intercept()`  
|               | _os_rte()` _os_send()`  
|               | _os_signj()` _os_sigmask()` \  
|               | _os_sigreset()` _os_signs()`  
|               | _os_sleep()` _os9_sleep()`  
|               | `pause()` `raise()`  
|               | `sigmask()` `signal()`  
|               | `sleep()` `tsleep()`  

† Indicates a standard ANSI header.
Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stat.h</td>
<td>Contains definitions for the following functions:</td>
</tr>
<tr>
<td></td>
<td>stat()</td>
</tr>
<tr>
<td></td>
<td>fstat()</td>
</tr>
<tr>
<td></td>
<td>lstat()</td>
</tr>
<tr>
<td>stdarg.h †</td>
<td>Used for advancing through a list of arguments of variable length and type. stdarg.h contains the macros for:</td>
</tr>
<tr>
<td></td>
<td>va_arg()</td>
</tr>
<tr>
<td></td>
<td>va_end()</td>
</tr>
<tr>
<td></td>
<td>va_start()</td>
</tr>
<tr>
<td>stddef.h †</td>
<td>Used for common definitions (ptrdiff_t, size_t, and wchar_t) and the following macro:</td>
</tr>
<tr>
<td></td>
<td>offsetof()</td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
</table>
| stdio.h † | Used for performing input and output. stdio.h contains prototypes for the following functions:  
  _cleareof(), cleareof()_clearerr()_errmsg()  
  fclose()fopen()  
  feof()ferror()  
  fflush()fgetc()  
  fgetpos()fgets()  
  _fileno(), fileno()fopen()  
  fprintf()fputc()  
  fputs()fread()  
  freopen()fscanf()  
  fseek()fsetpos()  
  ftell()fwrite()  
  getc(), getchar()gets()getw()  
  mkstemp()mktemp()  
  perror()prerr()  
  printf()putc(), putchar()  
  puts()putw()  
  remove()rename()  
  rewind()scanf()  
  setbuf()setvbuf()  
  snprintf()sprintf()  
  sscanf()tmpfile()  
  tmpnam()ungetc()  
  vfprintf()vprintf()  
  vsprintf()  
† Indicates a standard ANSI header.
<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdlib.h †</td>
<td>Used for the standard library functions. stdlib.h contains prototypes for the following functions: abort() abs() atexit() atof() atoi() atol() _atou() bsearch() calloc() div() _dtoa() ebrk() exit() _exit() free()_freemem(), freemem() _freemin()ibrk() getenv() labs() _lcalloc()ldiv() _lfree()_lmalloc() _lrealloc()malloc() _mallocmin()mblen() mbstowcs()mbtowc() qsort() _prealloc_rand() rand() realloc() sbrk() srand() stacksiz(), _stacksiz()strtol()stroull() strtoul()system() wcstombs()wctomb()</td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
### Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>string.h †</td>
<td>Used for manipulating arrays of character type. <code>string.h</code> contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>memchr() memcmp() memmove()</td>
</tr>
<tr>
<td></td>
<td>memset() strcasecmp(), strncasecmp()</td>
</tr>
<tr>
<td></td>
<td>strcat() strchr()</td>
</tr>
<tr>
<td></td>
<td>strcmp() strcoll()</td>
</tr>
<tr>
<td></td>
<td>strcpy() strcspn()</td>
</tr>
<tr>
<td></td>
<td>strerror() strlcat()</td>
</tr>
<tr>
<td></td>
<td>strcat() strchrnd()</td>
</tr>
<tr>
<td></td>
<td>strerror() strlcpy()</td>
</tr>
<tr>
<td></td>
<td>strlen() strncat()</td>
</tr>
<tr>
<td></td>
<td>strncmp() strncpy()</td>
</tr>
<tr>
<td></td>
<td>strpbrk() strrcoll()</td>
</tr>
<tr>
<td></td>
<td>strstr() strton()</td>
</tr>
<tr>
<td></td>
<td>strxfrm()</td>
</tr>
<tr>
<td>stringlist.h</td>
<td>Contains definitions for manipulating lists of strings. This file contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>sl_add(), sl_find(), sl_free(), sl_init()</td>
</tr>
<tr>
<td>strings.h</td>
<td>Used for non-ANSI string functions. <code>strings.h</code> contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>_cmpnam() findnstr()</td>
</tr>
<tr>
<td></td>
<td>findstr() index()</td>
</tr>
<tr>
<td></td>
<td>_os_cmpnam() _os_prsnam()</td>
</tr>
<tr>
<td></td>
<td>pause() _prsnam()</td>
</tr>
<tr>
<td></td>
<td>rindex() _strass()</td>
</tr>
<tr>
<td></td>
<td>strhcpy()</td>
</tr>
<tr>
<td>svctbl.h</td>
<td>Contains information for the service table. <code>svctbl.h</code> contains a prototypes for the following function:</td>
</tr>
<tr>
<td></td>
<td>_os_setsvc()</td>
</tr>
<tr>
<td>sysglob.h</td>
<td>Defines the system global variables. <code>sysglob.h</code> contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td>_getsys() _os_config()</td>
</tr>
<tr>
<td></td>
<td>_os_getsys() _os_setsys()</td>
</tr>
<tr>
<td></td>
<td>_setsys()</td>
</tr>
<tr>
<td>sys/time.h</td>
<td>Contains the file time modification function <code>utimes()</code>.</td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
## Table 1-2. Ultra C Header Files (Continued)

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sys/wait.h</code></td>
<td>Contains prototype for the following function:</td>
</tr>
<tr>
<td></td>
<td><code>waitpid()</code></td>
</tr>
<tr>
<td><code>termcap.h</code></td>
<td>Used for the terminal capability functions. <code>termcap.h</code> contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td><code>tgetent()</code> <code>tgetflag()</code> <code>tgetnum()</code> <code>tgetstr()</code> <code>tgoto()</code> <code>tputs()</code></td>
</tr>
<tr>
<td><code>termio.h</code></td>
<td>Contains definition to support <code>ioctl</code>.</td>
</tr>
<tr>
<td><code>threads.h</code></td>
<td>Process thread support routines. <code>threads.h</code> contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td><code>_os_thexit()</code> <code>_os_thfork()</code> <code>_os_thread()</code></td>
</tr>
<tr>
<td><code>time.h</code> †</td>
<td>Used for manipulating time. <code>time.h</code> contains prototypes for the following functions:</td>
</tr>
<tr>
<td></td>
<td><code>asctime()</code> <code>clock()</code> <code>ctime()</code> <code>difftime()</code> <code>gettime()</code> <code>gmtime()</code> <code>_gregorian()</code> <code>_julian()</code> <code>localtime()</code> <code>mktime()</code> <code>_os_getime()</code> <code>_os9_getime()</code> <code>_os_gregorian()</code> <code>_os_julian()</code> <code>os_setime()</code> <code>os9_setime()</code> <code>setime()</code> <code>strftime()</code> <code>strptime()</code> <code>sysdate()</code> <code>sys_mktime()</code> <code>time()</code></td>
</tr>
<tr>
<td><code>times.h</code></td>
<td>Contains definitions to support times.</td>
</tr>
<tr>
<td></td>
<td><code>times()</code></td>
</tr>
<tr>
<td><code>types.h</code></td>
<td>Defines <code>typedefs</code> for consistently sized and signed types.</td>
</tr>
<tr>
<td><code>util.h</code></td>
<td>Contains the definition of <code>fparseln()</code>, which is used to read logical lines from an input stream.</td>
</tr>
<tr>
<td><code>un.h</code></td>
<td>Contains the <code>sockaddr_un</code> structure definition.</td>
</tr>
<tr>
<td><code>virtual.h</code></td>
<td>Contains prototype for the following functions:</td>
</tr>
<tr>
<td></td>
<td><code>_os_transadd()</code></td>
</tr>
</tbody>
</table>

† Indicates a standard ANSI header.
Chapter 1: Overview

The C Library

This section includes a listing of the available C functions, grouped according to functionality. Each listing includes the function’s name, description, and the header file in which it is defined.

Documented features and/or functionality classified as “undefined” by the relevant ANSI/ISO C specification are subject to change in future version of the Ultra C libraries. Be certain that the code you are using does not depend on the behavior of undefined situations, as documented by the ANSI/ISO specification.

Communicating with Environment Functions

The Ultra C libraries contain the functions listed in Table 1-3 for communicating with the environment.

Table 1-3. Environment Communications Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort()</td>
<td>•</td>
<td>Abnormal program termination</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>atexit()</td>
<td>•</td>
<td>Register function to call at normal program termination</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_exit()</td>
<td>•</td>
<td>Task termination</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>exit()</td>
<td>•</td>
<td>Task termination</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>getenv()</td>
<td>•</td>
<td>Get value for environment name</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_os_exit()</td>
<td>•</td>
<td>Terminate calling process</td>
<td>process.h</td>
</tr>
<tr>
<td>system()</td>
<td>•</td>
<td>Shell command execution</td>
<td>stdlib.h</td>
</tr>
</tbody>
</table>

Character Classification Functions

The character classification functions are macros defined in <ctype.h>. Be careful when using these macros to avoid macro expansion side-effects. The macros provide a platform independent method of character classification. Table 1-4 identifies character classification functions.

Table 1-4. Character Classification Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum()</td>
<td>•</td>
<td>Test if parameter is alphanumeric</td>
<td>ctype.h</td>
</tr>
<tr>
<td>isalpha()</td>
<td>•</td>
<td>Test if parameter is alphabetic</td>
<td>ctype.h</td>
</tr>
<tr>
<td>_isascii(), _isascii()</td>
<td>•</td>
<td>Test if parameter is ASCII</td>
<td>ctype.h</td>
</tr>
<tr>
<td>iscntrl()</td>
<td>•</td>
<td>Test if parameter is control code</td>
<td>ctype.h</td>
</tr>
<tr>
<td>isdigit()</td>
<td>•</td>
<td>Test if parameter is digit</td>
<td>ctype.h</td>
</tr>
<tr>
<td>isgraph()</td>
<td>•</td>
<td>Test if parameter is printing character</td>
<td>ctype.h</td>
</tr>
<tr>
<td>_isjis_kana()</td>
<td>•</td>
<td>Test if parameter is Hankaku-Kana</td>
<td>ctype.h</td>
</tr>
<tr>
<td>islower()</td>
<td>•</td>
<td>Test if parameter is lowercase</td>
<td>ctype.h</td>
</tr>
</tbody>
</table>
Chapter 1: Overview

Character Conversion Functions

The character conversion functions allow you to convert a string of characters to their numeric representation and to change the case of characters. Table 1-5 identifies character conversion functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>atof()</td>
<td>•</td>
<td>Alpha to floating conversion</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>atoi()</td>
<td>•</td>
<td>Alpha to integer conversion</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>atol()</td>
<td>•</td>
<td>Alpha to long conversion</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_aton()</td>
<td>•</td>
<td>Alpha to numeric translation</td>
<td>atan.h</td>
</tr>
<tr>
<td>_atou()</td>
<td>•</td>
<td>Alpha to unsigned conversion</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_ dtoa()</td>
<td>•</td>
<td>Double to ASCII conversion</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>toascii(), _toascii()</td>
<td>•</td>
<td>Integer to ASCII translation</td>
<td>ctype.h</td>
</tr>
<tr>
<td>_tolower()</td>
<td>•</td>
<td>Convert character to lowercase</td>
<td>ctype.h</td>
</tr>
<tr>
<td>tolower()</td>
<td>•</td>
<td>Convert character to lowercase</td>
<td>ctype.h</td>
</tr>
<tr>
<td>_toupper()</td>
<td>•</td>
<td>Convert character to uppercase</td>
<td>ctype.h</td>
</tr>
<tr>
<td>toupper()</td>
<td>•</td>
<td>Convert character to uppercase</td>
<td>ctype.h</td>
</tr>
</tbody>
</table>

File Positioning Functions

The functions in Table 1-6 are provided for file positioning.

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>fgetpos()</td>
<td>•</td>
<td>Get current position in file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fseek()</td>
<td>•</td>
<td>Reposition file pointer</td>
<td>stdio.h</td>
</tr>
<tr>
<td>faresetpos()</td>
<td>•</td>
<td>Reposition file pointer</td>
<td>stdio.h</td>
</tr>
</tbody>
</table>
GetStat/SetStat Functions

The standard library provides the GetStat and SetStat functions for determining and setting individual device parameters that are not uniform on all devices or that are highly hardware dependent.

Table 1-7. GetStat/SetStat Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>getstat()</td>
<td>Get file status</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_gs_devn()</td>
<td>Get device name</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_gs_eof()</td>
<td>Check for end of file</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_gs_gfd()</td>
<td>Get file descriptor sector</td>
<td>none</td>
</tr>
<tr>
<td>_gs_opt()</td>
<td>Get path options</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_gs_pos()</td>
<td>Get current file position</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_gs_rdy()</td>
<td>Test for data available</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_gs_size()</td>
<td>Get current file size</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_getstat()</td>
<td>Get file/device status</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_getsys()</td>
<td>Examine system global variable</td>
<td>sysglob.h</td>
</tr>
<tr>
<td>_os9_gs_cdfd()</td>
<td>Return file descriptor</td>
<td>cdi.h</td>
</tr>
<tr>
<td>_os_gs_cpypd()</td>
<td>Copy contents of path descriptor</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_gs_cstats()</td>
<td>Get cache status information</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_gs_devnm()</td>
<td>Return device name</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_gs_devtyp()</td>
<td>Return device type</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_gs_dopt()</td>
<td>Read device path options</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_gs_dsize()</td>
<td>Get size of SCSI devices</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_gs_edt()</td>
<td>Get I/O interface edition number</td>
<td>ioedt.h</td>
</tr>
<tr>
<td>_os_gs_eof()</td>
<td>Test for end of file</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_gs_fd()</td>
<td>Read file descriptor sector</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_gs_fdaddr()</td>
<td>Get file descriptor block address for open file</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_gs_fdinf()</td>
<td>Get specified file descriptor sector</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os9_gs_free()</td>
<td>Return amount of free space on device</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_gs_luopt()</td>
<td>Read logical unit options</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os(gs parity)</td>
<td>Calculate parity of file descriptor</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_gs_popt()</td>
<td>Read path descriptor option section</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_gs_pos()</td>
<td>Get current file position</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_gs_ready()</td>
<td>Test for data ready</td>
<td>sg_codes.h</td>
</tr>
</tbody>
</table>
Table 1-7. GetStat/SetStat Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_ga_size()</td>
<td>Return current file size</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_setstat()</td>
<td>Set file/device status</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_setsvc()</td>
<td>Adds or replaces a system service</td>
<td>funcs.h</td>
</tr>
<tr>
<td>_os_setsys()</td>
<td>Set system global variables</td>
<td>sysglob.h</td>
</tr>
<tr>
<td>_os_sgetstat()</td>
<td>GetStat call using system path number</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_sgs_devnm()</td>
<td>Return device name</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_attr()</td>
<td>Set file attributes</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_break()</td>
<td>Break serial connection</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_cache()</td>
<td>Enable/disable RBF caching</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os9_ss_close()</td>
<td>Notify driver that path has been closed</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_ss_dcoff()</td>
<td>Send signal when data carrier detect line goes false</td>
<td>scf.h</td>
</tr>
<tr>
<td>_os_ss_dcon()</td>
<td>Send signal when data carrier detect line goes true</td>
<td>scf.h</td>
</tr>
<tr>
<td>_os_ss_dopt()</td>
<td>Set device path options</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_drets()</td>
<td>Disable RTS line</td>
<td>scf.h</td>
</tr>
<tr>
<td>_os_ss_enrts()</td>
<td>Enable RTS line</td>
<td>scf.h</td>
</tr>
<tr>
<td>_os_ss_erase()</td>
<td>Erase tape</td>
<td>sbf.h</td>
</tr>
<tr>
<td>_os_ss_fd()</td>
<td>Write file descriptor sector</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_ss_fillbuff()</td>
<td>Fill path buffer with data</td>
<td>scf.h</td>
</tr>
<tr>
<td>_os_ss_flushmap()</td>
<td>Flush cached bit map information for RBF device</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_ss_hdl()</td>
<td>Make hard link to existing file</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_ss_lock()</td>
<td>Lock out record</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os9_ss_luopt()</td>
<td>Write logical unit options</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os9_ss_open()</td>
<td>Notify driver that path has been opened</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_ss_popt()</td>
<td>Write option section of path descriptor</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_relea()</td>
<td>Release device</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_rename()</td>
<td>Rename file</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_reset()</td>
<td>Restore head to track zero</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_reten()</td>
<td>Retension pass on tape drive</td>
<td>sbf.h</td>
</tr>
<tr>
<td>_os_ss_rfm()</td>
<td>Skip tape marks</td>
<td>sbf.h</td>
</tr>
<tr>
<td>_os_ss_sendsig()</td>
<td>Send signal on data ready</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_size()</td>
<td>Set file size</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_os_ss_skip()</td>
<td>Skip blocks</td>
<td>sbf.h</td>
</tr>
<tr>
<td>_os_ss_skipend()</td>
<td>Skip to end of tape</td>
<td>sbf.h</td>
</tr>
<tr>
<td>_os_ss_ticks()</td>
<td>Wait specified number of ticks for record release</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_ss_wfm()</td>
<td>Write tape marks</td>
<td>sbf.h</td>
</tr>
<tr>
<td>_os_ss_wtrack()</td>
<td>Write (format) track</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_os_setstat()</td>
<td>Set file status</td>
<td>sg_codes.h</td>
</tr>
</tbody>
</table>
Chapter 1: Overview

Table 1-7. GetStat/SetStat Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ss_attr()</td>
<td>Set file attributes</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_ss_dcoff()</td>
<td>Send data carrier lost signal to process</td>
<td>scf.h</td>
</tr>
<tr>
<td>_ss_dcon()</td>
<td>Send data carrier present signal to process</td>
<td>scf.h</td>
</tr>
<tr>
<td>_ss_dsrts()</td>
<td>Disable RTS line</td>
<td>scf.h</td>
</tr>
<tr>
<td>_ss_enrts()</td>
<td>Enable RTS line</td>
<td>scf.h</td>
</tr>
<tr>
<td>_ss_lock()</td>
<td>Lock out record</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_ss_opt()</td>
<td>Set path options</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_ss_pfd()</td>
<td>Put file descriptor sector</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_ss_rel()</td>
<td>Release device</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_ss_rest()</td>
<td>Restore device</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_ss_size()</td>
<td>Set current file size</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_ss_ssig()</td>
<td>Send signal on data ready</td>
<td>sg_codes.h</td>
</tr>
<tr>
<td>_ss_tiks()</td>
<td>Wait for record release</td>
<td>rbf.h</td>
</tr>
<tr>
<td>_ss_wtrk()</td>
<td>Write track</td>
<td>rbf.h</td>
</tr>
</tbody>
</table>

Input/Output Device Functions

The following functions are available for performing input/output on devices.

Table 1-8. Input/Output Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>inc()</td>
<td>Get character from I/O address</td>
<td>regs.h</td>
</tr>
<tr>
<td>inl()</td>
<td>Get integer from I/O address</td>
<td>regs.h</td>
</tr>
<tr>
<td>inw()</td>
<td>Get word from I/O address</td>
<td>regs.h</td>
</tr>
<tr>
<td>outc()</td>
<td>Write character to I/O address</td>
<td>regs.h</td>
</tr>
<tr>
<td>outl()</td>
<td>Write integer to I/O address</td>
<td>regs.h</td>
</tr>
<tr>
<td>outw()</td>
<td>Write word to I/O address</td>
<td>regs.h</td>
</tr>
</tbody>
</table>

Input/Output Functions

The following functions make up the I/O functions of the Ultra C libraries. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-9. Ultra C I/O Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>access()</td>
<td></td>
<td>Determine file accessibility</td>
<td>modes.h</td>
</tr>
<tr>
<td>attach()</td>
<td></td>
<td>Attach to device</td>
<td>modes.h</td>
</tr>
<tr>
<td>chdir()</td>
<td></td>
<td>Change current data directory</td>
<td>modes.h</td>
</tr>
<tr>
<td>chmod()</td>
<td></td>
<td>Change file access permissions</td>
<td>modes.h</td>
</tr>
</tbody>
</table>
Table 1-9. Ultra C I/O Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>chown()</td>
<td></td>
<td>Change owner of file</td>
<td>modes.h</td>
</tr>
<tr>
<td>chxdir()</td>
<td></td>
<td>Change current execution directory</td>
<td>modes.h</td>
</tr>
<tr>
<td>_cleareof(), cleareof()</td>
<td></td>
<td>Clear end of file condition</td>
<td>stdio.h</td>
</tr>
<tr>
<td>clearerr()</td>
<td>•</td>
<td>Clear error condition</td>
<td>stdio.h</td>
</tr>
<tr>
<td>close()</td>
<td></td>
<td>Close path</td>
<td>modes.h</td>
</tr>
<tr>
<td>closedir()</td>
<td></td>
<td>Close named directory stream</td>
<td>dir.h</td>
</tr>
<tr>
<td>creat()</td>
<td></td>
<td>Create file</td>
<td>modes.h</td>
</tr>
<tr>
<td>create()</td>
<td></td>
<td>Create file</td>
<td>modes.h</td>
</tr>
<tr>
<td>detach()</td>
<td></td>
<td>Detach device</td>
<td>modes.h</td>
</tr>
<tr>
<td>dup()</td>
<td></td>
<td>Duplicate path</td>
<td>modes.h</td>
</tr>
<tr>
<td>fclose()</td>
<td>•</td>
<td>Close file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fdopen()</td>
<td></td>
<td>Attach path to file pointer</td>
<td>stdio.h</td>
</tr>
<tr>
<td>feof()</td>
<td>•</td>
<td>Check buffered file for end of file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>ferror()</td>
<td>•</td>
<td>Check buffered file for error condition</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fflush()</td>
<td>•</td>
<td>Flush file’s buffer</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fgetc()</td>
<td>•</td>
<td>Get character from file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fgets()</td>
<td>•</td>
<td>Get string from file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>_fileno(), fileno()</td>
<td></td>
<td>Determine path number from file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fopen()</td>
<td>•</td>
<td>Open file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fprintf()</td>
<td>•</td>
<td>Formatted output</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fputc()</td>
<td>•</td>
<td>Output character to file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fputs()</td>
<td>•</td>
<td>Output string to file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fread()</td>
<td>•</td>
<td>Read data from file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>freopen()</td>
<td>•</td>
<td>Re-open file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fscanf()</td>
<td>•</td>
<td>Input string conversion</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fwrite()</td>
<td>•</td>
<td>Write data to file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>getc(), getchar()</td>
<td>•</td>
<td>Get next character from file/standard input</td>
<td>stdio.h</td>
</tr>
<tr>
<td>gets()</td>
<td>•</td>
<td>Get string from file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>getw()</td>
<td></td>
<td>Read word from file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>lseek()</td>
<td></td>
<td>Position file pointer</td>
<td>modes.h</td>
</tr>
<tr>
<td>mkdir()</td>
<td></td>
<td>Create directory</td>
<td>modes.h</td>
</tr>
<tr>
<td>mknod()</td>
<td></td>
<td>Create directory</td>
<td>modes.h</td>
</tr>
<tr>
<td>mktemp()</td>
<td></td>
<td>Create unique file name</td>
<td>stdio.h</td>
</tr>
<tr>
<td>Function</td>
<td>ANSI Standard</td>
<td>Description</td>
<td>Header File</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------</td>
<td>---------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>open()</td>
<td></td>
<td>Open file</td>
<td>modes.h</td>
</tr>
<tr>
<td>opendir()</td>
<td></td>
<td>Open directory</td>
<td>dir.h</td>
</tr>
<tr>
<td>_os_alias()</td>
<td></td>
<td>Create device alias</td>
<td>io.h</td>
</tr>
<tr>
<td>_os_attach()</td>
<td></td>
<td>Attach new device to system</td>
<td>io.h</td>
</tr>
<tr>
<td>_os_chdir()</td>
<td></td>
<td>Change working directory</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_close()</td>
<td></td>
<td>Close path to file/device.</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_cpy_ioproc()</td>
<td></td>
<td>Get pointer to I/O process descriptor</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_create()</td>
<td></td>
<td>Create path to new file</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os9_create()</td>
<td></td>
<td>Create path to new file</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_delete()</td>
<td></td>
<td>Delete file</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_detach()</td>
<td></td>
<td>Remove device from system</td>
<td>io.h</td>
</tr>
<tr>
<td>_os_dup()</td>
<td></td>
<td>Duplicate path</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_get_ioproc()</td>
<td></td>
<td>Get pointer to I/O process descriptor</td>
<td>io.h</td>
</tr>
<tr>
<td>_os_getdl()</td>
<td></td>
<td>Get system I/O device list head pointer</td>
<td>io.h</td>
</tr>
<tr>
<td>_os_ioconfig()</td>
<td></td>
<td>Configure an element of process/system I/O</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_iodel()</td>
<td></td>
<td>Check for use of I/O module</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_ioexit()</td>
<td></td>
<td>Terminate I/O for exiting process</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_iofork()</td>
<td></td>
<td>Set up I/O for new process</td>
<td>process.h</td>
</tr>
<tr>
<td>_os9_ioqueue()</td>
<td></td>
<td>Enter I/O queue</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_mkdir()</td>
<td></td>
<td>Make new directory</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_open()</td>
<td></td>
<td>Open path to file or device</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_rdalst()</td>
<td></td>
<td>Copy system alias list</td>
<td>io.h</td>
</tr>
<tr>
<td>_os_read()</td>
<td></td>
<td>Read data from file or device</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_readln()</td>
<td></td>
<td>Read text line with editing</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_tranpn()</td>
<td></td>
<td>Translate user path to system path</td>
<td>io.h</td>
</tr>
<tr>
<td>_os_write()</td>
<td></td>
<td>Write data to file or device</td>
<td>modes.h</td>
</tr>
<tr>
<td>_os_writeln()</td>
<td></td>
<td>Write line of text with editing</td>
<td>modes.h</td>
</tr>
<tr>
<td>pfinit()</td>
<td></td>
<td>Initialize for float output (obsolete)</td>
<td>none</td>
</tr>
<tr>
<td>pfinit()</td>
<td></td>
<td>Initialize for longs output (obsolete)</td>
<td>none</td>
</tr>
<tr>
<td>printf()</td>
<td></td>
<td>Formatted output</td>
<td>stdio.h</td>
</tr>
<tr>
<td>putchar(), putc()</td>
<td></td>
<td>Put next character to file/standard out</td>
<td>stdio.h</td>
</tr>
</tbody>
</table>
Interprocess Communication Functions

Functions available for interprocess communication are shown in Table 1-10. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-10. Interprocess Communication Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>alm_atdate()</td>
<td></td>
<td>Send signal at Gregorian date/time</td>
<td>alarm.h</td>
</tr>
<tr>
<td>alm_atjul()</td>
<td></td>
<td>Send signal at Julian date/time</td>
<td>alarm.h</td>
</tr>
<tr>
<td>alm_cycle()</td>
<td></td>
<td>Send signal at specified time intervals</td>
<td>alarm.h</td>
</tr>
<tr>
<td>alm_delete()</td>
<td></td>
<td>Remove pending alarm request</td>
<td>alarm.h</td>
</tr>
</tbody>
</table>
Table 1-10. Interprocess Communication Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>alm_set()</td>
<td></td>
<td>Send signal after specified time interval</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_ev_creat()</td>
<td></td>
<td>Create event</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_delete()</td>
<td></td>
<td>Delete event</td>
<td>events.h</td>
</tr>
<tr>
<td>_os9_ev_info()</td>
<td></td>
<td>Returns event information in the caller's buffer</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_info()</td>
<td></td>
<td>Get event information</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_link()</td>
<td></td>
<td>Link to existing event</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_pulse()</td>
<td></td>
<td>Signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_read()</td>
<td></td>
<td>Read event without waiting</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_set()</td>
<td></td>
<td>Set event variable and signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_setr()</td>
<td></td>
<td>Set relative event variable and signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_signal()</td>
<td></td>
<td>Signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_unlink()</td>
<td></td>
<td>Unlink event</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_wait()</td>
<td></td>
<td>Wait for event</td>
<td>events.h</td>
</tr>
<tr>
<td>_ev_waitr()</td>
<td></td>
<td>Wait for relative event</td>
<td>events.h</td>
</tr>
<tr>
<td>interpret()</td>
<td></td>
<td>Set up process signal handler</td>
<td>signal.h</td>
</tr>
<tr>
<td>_os9_alarm_atdate()</td>
<td></td>
<td>Send signal at Gregorian date/time</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os9_alarm_atjul()</td>
<td></td>
<td>Send signal at Julian date/time</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_alarm_atime()</td>
<td></td>
<td>Send signal at specified time</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_alarm_cycle()</td>
<td></td>
<td>Send signal at specified time intervals</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_alarm_delete()</td>
<td></td>
<td>Remove pending alarm request</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_alarm_reset()</td>
<td></td>
<td>Reset existing alarm request</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_alarm_set()</td>
<td></td>
<td>Send signal after specified time interval</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_clrsigs()</td>
<td></td>
<td>Clear process signal queue</td>
<td>signal.h</td>
</tr>
<tr>
<td>_os_ev_allclr()</td>
<td></td>
<td>Wait for all bits defined by mask to become clear</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_allset()</td>
<td></td>
<td>Wait for event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_anyclr()</td>
<td></td>
<td>Wait for event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_anyset()</td>
<td></td>
<td>Wait for event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_change()</td>
<td></td>
<td>Wait for event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_creat()</td>
<td></td>
<td>Create new event</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_delete()</td>
<td></td>
<td>Delete existing event</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_info()</td>
<td></td>
<td>Return event information</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_link()</td>
<td></td>
<td>Link to existing event by name</td>
<td>events.h</td>
</tr>
</tbody>
</table>
### Table 1-10. Interprocess Communication Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_ev_pulse()</td>
<td></td>
<td>Signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_read()</td>
<td></td>
<td>Read event value without waiting</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_set()</td>
<td></td>
<td>Set event variable and signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_setand()</td>
<td></td>
<td>Set event variable and signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_setor()</td>
<td></td>
<td>Set event variable and signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_setr()</td>
<td></td>
<td>Set relative event variable and signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_setxor()</td>
<td></td>
<td>Set event variable and signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_signal()</td>
<td></td>
<td>Signal event occurrence</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_tstset()</td>
<td></td>
<td>Wait for event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_unlink()</td>
<td></td>
<td>Unlink event</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_wait()</td>
<td></td>
<td>Wait for event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os9_ev_wait()</td>
<td></td>
<td>Wait for event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_ev_waitr()</td>
<td></td>
<td>Wait for relative event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os9_ev_waitr()</td>
<td></td>
<td>Wait for relative event to occur</td>
<td>events.h</td>
</tr>
<tr>
<td>_os_intercept()</td>
<td></td>
<td>Set up signal intercept trap</td>
<td>signal.h</td>
</tr>
<tr>
<td>_os9_salarm_atdate()</td>
<td></td>
<td>Execute system-state subroutine at Gregorian date/time</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_salarm_atime()</td>
<td></td>
<td>Execute system-state subroutine at specified time</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os9_salarm_atjul()</td>
<td></td>
<td>Execute system-state subroutine at Julian date/time</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_salarm_cycle()</td>
<td></td>
<td>Set cyclic alarm clock</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os9_salarm_cycle()</td>
<td></td>
<td>Execute system-state subroutine every n ticks/seconds</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_salarm_delete()</td>
<td></td>
<td>Remove pending alarm request</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_salarm_reset()</td>
<td></td>
<td>Reset alarm clock</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_salarm_set()</td>
<td></td>
<td>Set alarm clock</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os9_salarm_set()</td>
<td></td>
<td>Execute system-state subroutine after specified time interval</td>
<td>alarm.h</td>
</tr>
<tr>
<td>_os_sema_init()</td>
<td></td>
<td>Initialize semaphore data structure for use</td>
<td>semaphore.h</td>
</tr>
<tr>
<td>_os_sema_p()</td>
<td></td>
<td>Reserve semaphore (acquire exclusive access)</td>
<td>semaphore.h</td>
</tr>
<tr>
<td>_os_sema_term()</td>
<td></td>
<td>Terminate use of semaphore data structure</td>
<td>semaphore.h</td>
</tr>
</tbody>
</table>
Interrupt Manipulation Functions

The following functions are provided for interrupt manipulation. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-11. Interrupt Manipulation Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>irq_change()</td>
<td>Set IRQ level</td>
<td>regs.h</td>
</tr>
<tr>
<td>irq_disable()</td>
<td>Mask interrupts</td>
<td>regs.h</td>
</tr>
<tr>
<td>irq_enable()</td>
<td>Unmask interrupts</td>
<td>regs.h</td>
</tr>
<tr>
<td>irq_maskget()</td>
<td>Mask interrupts; return previous level</td>
<td>regs.h</td>
</tr>
<tr>
<td>irq_restore()</td>
<td>Restore interrupt level</td>
<td>regs.h</td>
</tr>
<tr>
<td>irq_save()</td>
<td>Return current interrupt level</td>
<td>regs.h</td>
</tr>
<tr>
<td>_os_firq()</td>
<td>Add or remove device from fast IRQ table</td>
<td>regs.h</td>
</tr>
<tr>
<td>_os_irq()</td>
<td>Add or remove device from IRQ table</td>
<td>regs.h</td>
</tr>
<tr>
<td>_os9_irq()</td>
<td>Add or remove device from IRQ table</td>
<td>regs.h</td>
</tr>
<tr>
<td>_os_rte()</td>
<td>Return from interrupt exception</td>
<td>signal.h</td>
</tr>
</tbody>
</table>
Mathematical Functions

The following are the available mathematical functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs()</td>
<td>•</td>
<td>Integer absolute value</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>acos()</td>
<td>•</td>
<td>Arc cosine function</td>
<td>math.h</td>
</tr>
<tr>
<td>asin()</td>
<td>•</td>
<td>Arc sine function</td>
<td>math.h</td>
</tr>
<tr>
<td>atan()</td>
<td>•</td>
<td>Arc tangent function</td>
<td>math.h</td>
</tr>
<tr>
<td>atan2()</td>
<td>•</td>
<td>Arc tangent function</td>
<td>math.h</td>
</tr>
<tr>
<td>ceil()</td>
<td>•</td>
<td>Ceiling function</td>
<td>math.h</td>
</tr>
<tr>
<td>cos()</td>
<td>•</td>
<td>Cosine function</td>
<td>math.h</td>
</tr>
<tr>
<td>cosh()</td>
<td>•</td>
<td>Hyperbolic cosine function</td>
<td>math.h</td>
</tr>
<tr>
<td>div()</td>
<td>•</td>
<td>Compute quotient and remainder</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>exp()</td>
<td>•</td>
<td>Exponential function</td>
<td>math.h</td>
</tr>
<tr>
<td>fabs()</td>
<td>•</td>
<td>Floating absolute value</td>
<td>math.h</td>
</tr>
<tr>
<td>floor()</td>
<td>•</td>
<td>Floor function</td>
<td>math.h</td>
</tr>
<tr>
<td>fmod()</td>
<td>•</td>
<td>Compute floating point remainder</td>
<td>math.h</td>
</tr>
<tr>
<td>frexp()</td>
<td>•</td>
<td>Return portions of floating point number</td>
<td>math.h</td>
</tr>
<tr>
<td>hypot()</td>
<td>•</td>
<td>Euclidean distance function</td>
<td>math.h</td>
</tr>
<tr>
<td>labs()</td>
<td>•</td>
<td>Compute absolute value</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>ldexp()</td>
<td>•</td>
<td>Multiply float by exponent of two</td>
<td>math.h</td>
</tr>
<tr>
<td>ldiv()</td>
<td>•</td>
<td>Compute quotient and remainder</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>log()</td>
<td>•</td>
<td>Natural logarithm</td>
<td>math.h</td>
</tr>
<tr>
<td>log10()</td>
<td>•</td>
<td>Base-ten logarithm</td>
<td>math.h</td>
</tr>
<tr>
<td>modf()</td>
<td>•</td>
<td>Return parts of real number</td>
<td>math.h</td>
</tr>
<tr>
<td>pow()</td>
<td>•</td>
<td>Power function</td>
<td>math.h</td>
</tr>
<tr>
<td>rand()</td>
<td>•</td>
<td>Return random integer</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>sin()</td>
<td>•</td>
<td>Sine function</td>
<td>math.h</td>
</tr>
<tr>
<td>sinh()</td>
<td>•</td>
<td>Hyperbolic sine function</td>
<td>math.h</td>
</tr>
<tr>
<td>sqrt()</td>
<td>•</td>
<td>Square root function</td>
<td>math.h</td>
</tr>
<tr>
<td>srand()</td>
<td>•</td>
<td>Set seed for random number generator</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>tan()</td>
<td>•</td>
<td>Tangent function</td>
<td>math.h</td>
</tr>
<tr>
<td>tanh()</td>
<td>•</td>
<td>Hyperbolic tangent function</td>
<td>math.h</td>
</tr>
</tbody>
</table>

Memory Management Functions

The memory management functions are provided for requesting and freeing memory in a machine and operating system independent manner. ANSI prototypes
for non-ANSI functions are achieved by defining the macro name `OPT_PROTOS` during the compile.

### Table 1-13. Memory Management Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>calloc()</td>
<td>•</td>
<td>Allocate storage for array</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_cpymem()</td>
<td></td>
<td>Copy external memory</td>
<td>process.h</td>
</tr>
<tr>
<td>ebrk()</td>
<td></td>
<td>Get external memory</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>free()</td>
<td>•</td>
<td>Return memory</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_freemem(), freemem()</td>
<td></td>
<td>Determine size of unused stack area</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_freemin()</td>
<td></td>
<td>Set memory reclamation bound</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>ibrk()</td>
<td></td>
<td>Request internal memory</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_lcalloc()</td>
<td></td>
<td>Allocate storage for array (low-overhead)</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_lfree()</td>
<td></td>
<td>Return memory (low-overhead)</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_lmalloc()</td>
<td></td>
<td>Allocate memory from arena (low-overhead)</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_lrealloc()</td>
<td></td>
<td>Resize block of memory (low-overhead)</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>malloc()</td>
<td>•</td>
<td>Allocate memory from arena</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_mallocmin()</td>
<td></td>
<td>Set minimum allocation size</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>memchr()</td>
<td>•</td>
<td>Memory search</td>
<td>string.h</td>
</tr>
<tr>
<td>memcmp()</td>
<td>•</td>
<td>Compare memory</td>
<td>string.h</td>
</tr>
<tr>
<td>memcpy()</td>
<td>•</td>
<td>Copy memory</td>
<td>string.h</td>
</tr>
<tr>
<td>memmove()</td>
<td>•</td>
<td>Move memory</td>
<td>string.h</td>
</tr>
<tr>
<td>memset()</td>
<td>•</td>
<td>Fill memory</td>
<td>string.h</td>
</tr>
<tr>
<td>_os_alltsk()</td>
<td></td>
<td>Allocate task</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_chkmem()</td>
<td></td>
<td>Check memory block’s accessibility</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_c pymem()</td>
<td></td>
<td>Copy external memory</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_deltsk()</td>
<td></td>
<td>Deallocate process descriptor</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_get_blkmap()</td>
<td></td>
<td>Get free memory block map</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os_mem()</td>
<td></td>
<td>Resize data memory area</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os_move()</td>
<td></td>
<td>Move data (low bound first)</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os_permit()</td>
<td></td>
<td>Allow access to memory block</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_protect()</td>
<td></td>
<td>Prevent access to memory block</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_srqmem()</td>
<td></td>
<td>System memory request</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os9_srqmem()</td>
<td></td>
<td>System memory request</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os_srtmem()</td>
<td></td>
<td>Return system memory</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os_transadd()</td>
<td></td>
<td>Translate memory address (OS-9)</td>
<td>virtual.h</td>
</tr>
<tr>
<td>_os9_translate()</td>
<td></td>
<td>Translate memory address (OS-9 for 68K)</td>
<td>memory.h</td>
</tr>
</tbody>
</table>
Chapter 1: Overview

Ultra C Library Reference 56

Shared Data Access Functions

The following two C library functions can be helpful for porting an existing subroutine module. They access two different kinds of data: shared global data and thread-specific data.

The shared global data is automatically shared among all modules that have the same value of \_pthread (i.e. the application and the subroutine module). The thread-specific data is unique to each thread and is visible to all modules that have the same value of \_pthread.

The functions described below must be used to access this data.

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>_pthread_local_slot()</td>
<td></td>
<td>Used when reading or writing thread-specific versions of “core” C run-time variables.</td>
<td></td>
</tr>
<tr>
<td>_pthread_global_slot()</td>
<td></td>
<td>Used when reading or writing global versions of “core” C run-time variables.</td>
<td></td>
</tr>
</tbody>
</table>

Miscellaneous Functions

Miscellaneous functions in Table 1-15 appear in the Ultra C libraries to relieve you from routine tasks. ANSI prototypes for non-ANSI functions are achieved by defining the macro name \_OPT_PROTOS during the compile.

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert()</td>
<td></td>
<td>Place diagnostics in programs</td>
<td>assert.h</td>
</tr>
<tr>
<td>change_static()</td>
<td></td>
<td>Change current static storage pointer</td>
<td>regs.h</td>
</tr>
<tr>
<td>_cmpnam()</td>
<td></td>
<td>Compare two strings</td>
<td>strings.h</td>
</tr>
<tr>
<td>err(), errx()</td>
<td></td>
<td>Display error message</td>
<td>err.h</td>
</tr>
<tr>
<td>_errmsg()</td>
<td></td>
<td>Print error message</td>
<td>stdio.h</td>
</tr>
<tr>
<td>fparseln()</td>
<td></td>
<td>Return next logical line from a stream</td>
<td>util.h</td>
</tr>
<tr>
<td>getcwd()</td>
<td></td>
<td>Copy pathname of specific directory</td>
<td>os9defs.h</td>
</tr>
</tbody>
</table>
**Table 1-15. Miscellaneous Functions (Continued)**

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_static()</td>
<td></td>
<td>Return current static storage pointer</td>
<td>regs.h</td>
</tr>
<tr>
<td>getuid()</td>
<td></td>
<td>Determine user ID number</td>
<td>process.h</td>
</tr>
<tr>
<td>glob(), globfree()</td>
<td></td>
<td>Implement rules for file name pattern matching</td>
<td>glob.h</td>
</tr>
<tr>
<td>localeconv()</td>
<td></td>
<td>Numeric formatting convention inquiry</td>
<td>locale.h</td>
</tr>
<tr>
<td>longjmp()</td>
<td></td>
<td>Non-local goto</td>
<td>setjmp.h</td>
</tr>
<tr>
<td>lstat()</td>
<td></td>
<td>Obtain information about a file.</td>
<td>stat.h</td>
</tr>
<tr>
<td>mkstemp()</td>
<td></td>
<td>Override template to create file</td>
<td>stdio.h</td>
</tr>
<tr>
<td>_os9_allbit()</td>
<td></td>
<td>Set bits in bit map</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os_cache()</td>
<td></td>
<td>Cache control</td>
<td>cache.h</td>
</tr>
<tr>
<td>_os_config()</td>
<td></td>
<td>Configure an element</td>
<td>syslog.h</td>
</tr>
<tr>
<td>_os_cmpnam()</td>
<td></td>
<td>Compare two names</td>
<td>strings.h</td>
</tr>
<tr>
<td>_os9_delbit()</td>
<td></td>
<td>Deallocate in bit map</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os_dexec()</td>
<td></td>
<td>Execute debugged program</td>
<td>dexec.h</td>
</tr>
<tr>
<td>_os_dexit()</td>
<td></td>
<td>Exit debugged program</td>
<td>dexec.h</td>
</tr>
<tr>
<td>_os9_findpdp()</td>
<td></td>
<td>Find fixed block of memory</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_getpdp()</td>
<td></td>
<td>Find path descriptor</td>
<td>io.h</td>
</tr>
<tr>
<td>_os9_panic()</td>
<td></td>
<td>System catastrophic occurrence</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_perr()</td>
<td></td>
<td>Print error message</td>
<td>errno.h</td>
</tr>
<tr>
<td>_os_prsnam()</td>
<td></td>
<td>Parse path name</td>
<td>strings.h</td>
</tr>
<tr>
<td>_os9_retexpd()</td>
<td></td>
<td>Return fixed block of memory</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_scache()</td>
<td></td>
<td>Cache control</td>
<td>regs.h</td>
</tr>
<tr>
<td>_os9_schbit()</td>
<td></td>
<td>Search bit map for free area</td>
<td>memory.h</td>
</tr>
<tr>
<td>_os_setsvc()</td>
<td></td>
<td>Service request table initialization</td>
<td>svctbl.h</td>
</tr>
<tr>
<td>_os9_setsvc()</td>
<td></td>
<td>Service request table initialization</td>
<td>funcs.h</td>
</tr>
<tr>
<td>_os_setuid()</td>
<td></td>
<td>Set user ID number</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_settrap()</td>
<td></td>
<td>Set error trap handler</td>
<td>settrap.h</td>
</tr>
<tr>
<td>_os9_strap()</td>
<td></td>
<td>Set error trap handler</td>
<td>settrap.h</td>
</tr>
<tr>
<td>_os9_strt()</td>
<td></td>
<td>Return system identification</td>
<td>regs.h</td>
</tr>
<tr>
<td>_os_sysid()</td>
<td></td>
<td>User accounting</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_uacct()</td>
<td></td>
<td>Parse disk file (RBF) pathlist</td>
<td>strings.h</td>
</tr>
<tr>
<td>pause()</td>
<td></td>
<td>Map error number</td>
<td>stdio.h</td>
</tr>
<tr>
<td>perror()</td>
<td></td>
<td>Print error message</td>
<td>stdio.h</td>
</tr>
<tr>
<td>prerr()</td>
<td></td>
<td>Get module name</td>
<td>module.h</td>
</tr>
<tr>
<td>_prgname()</td>
<td></td>
<td>Parse path name segment</td>
<td>strings.h</td>
</tr>
<tr>
<td>_prsnam()</td>
<td></td>
<td>Non-local goto</td>
<td>setjmp.h</td>
</tr>
<tr>
<td>setjmp()</td>
<td></td>
<td>Locale control</td>
<td>locale.h</td>
</tr>
<tr>
<td>setlocale()</td>
<td></td>
<td>Set user ID</td>
<td>process.h</td>
</tr>
<tr>
<td>setuid()</td>
<td></td>
<td>Get size of stack used</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>_stacksiz(), _stacksiz()</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 1-15. Miscellaneous Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>_strass()</td>
<td></td>
<td>Structure assignment</td>
<td>strings.h</td>
</tr>
<tr>
<td>tmpnam()</td>
<td></td>
<td>Generate unique valid filename</td>
<td>stdio.h</td>
</tr>
<tr>
<td>va_arg()</td>
<td></td>
<td>Get parameter in variable parameter list</td>
<td>stdarg.h</td>
</tr>
<tr>
<td>va_end()</td>
<td></td>
<td>End references to current variable parameter list</td>
<td>stdarg.h</td>
</tr>
<tr>
<td>va_start()</td>
<td></td>
<td>Initialize variable parameter list</td>
<td>stdarg.h</td>
</tr>
<tr>
<td>verr(), verrx()</td>
<td></td>
<td>Display error message</td>
<td>err.h</td>
</tr>
<tr>
<td>vwarn(), vwarnx()</td>
<td></td>
<td>Display warning message</td>
<td>err.h</td>
</tr>
<tr>
<td>warn(), warnx()</td>
<td></td>
<td>Display warning message</td>
<td>err.h</td>
</tr>
</tbody>
</table>

### Module Manipulation Functions

Table 1-16 shows the module manipulation functions. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-16. Module Manipulation Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>crc</td>
<td></td>
<td>Calculate module CRC</td>
<td>module.h</td>
</tr>
<tr>
<td>_get_module_dir()</td>
<td></td>
<td>Get module directory entry</td>
<td>module.h</td>
</tr>
<tr>
<td>make_module()</td>
<td></td>
<td>Create module</td>
<td>module.h</td>
</tr>
<tr>
<td>_mkdata_module()</td>
<td></td>
<td>Create data memory module</td>
<td>module.h</td>
</tr>
<tr>
<td>modload()</td>
<td></td>
<td>Load module into colored memory</td>
<td>module.h</td>
</tr>
<tr>
<td>modlink()</td>
<td></td>
<td>Link to memory module</td>
<td>module.h</td>
</tr>
<tr>
<td>modload()</td>
<td></td>
<td>Load and link to memory module</td>
<td>module.h</td>
</tr>
<tr>
<td>modloadp()</td>
<td></td>
<td>Load and link to memory module using PATH</td>
<td>module.h</td>
</tr>
<tr>
<td>munlink()</td>
<td></td>
<td>Unlink from module</td>
<td>module.h</td>
</tr>
<tr>
<td>munload()</td>
<td></td>
<td>Unload module</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_altmdir()</td>
<td></td>
<td>Set alternate working module directory</td>
<td>moddir.h</td>
</tr>
<tr>
<td>_os_chainm()</td>
<td></td>
<td>Execute new primary module given pointer to module</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_chmdir()</td>
<td></td>
<td>Change process’ current module directory</td>
<td>moddir.h</td>
</tr>
<tr>
<td>_os_cmdperm()</td>
<td></td>
<td>Change permissions of module directory</td>
<td>moddir.h</td>
</tr>
<tr>
<td>_os_crc()</td>
<td></td>
<td>Generate CRC</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_datmod()</td>
<td></td>
<td>Create data module</td>
<td>module.h</td>
</tr>
</tbody>
</table>
Table 1-16. Module Manipulation Functions  (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_delmdir()</td>
<td></td>
<td>Delete existing module directory</td>
<td>moddir.h</td>
</tr>
<tr>
<td>_os_fmod()</td>
<td></td>
<td>Find module directory entry</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_get_mdp()</td>
<td></td>
<td>Get current and alternate module directory pathlists</td>
<td>moddir.h</td>
</tr>
<tr>
<td>_os_get_moddir()</td>
<td></td>
<td>Get copy of module directory</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_initdata()</td>
<td></td>
<td>Initialize static storage from module</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_link()</td>
<td></td>
<td>Link to memory module</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_linkm()</td>
<td></td>
<td>Link to memory module by module pointer</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_load()</td>
<td></td>
<td>Load module(s) from file</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_loadp()</td>
<td></td>
<td>Load and link to a memory module using PATH</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_makmdir()</td>
<td></td>
<td>Create new module directory</td>
<td>moddir.h</td>
</tr>
<tr>
<td>_os_mkmodule()</td>
<td></td>
<td>Create module of specified color type</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_modaddr()</td>
<td></td>
<td>Find module given pointer</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_setcrc()</td>
<td></td>
<td>Generate valid CRC in module</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_slink()</td>
<td></td>
<td>Install subroutine module</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_slinkm()</td>
<td></td>
<td>Install user subroutine library module by module pointer</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_tlink()</td>
<td></td>
<td>Install system state trap handling module</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_tlinkm()</td>
<td></td>
<td>Install user trap handling module by module pointer</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_unlink()</td>
<td></td>
<td>Unlink module by address</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_unload()</td>
<td></td>
<td>Unlink module by name</td>
<td>module.h</td>
</tr>
<tr>
<td>_os_vmodul()</td>
<td></td>
<td>Verify module</td>
<td>module.h</td>
</tr>
<tr>
<td>_os9_vmodul()</td>
<td></td>
<td>Validate module</td>
<td>module.h</td>
</tr>
<tr>
<td>_subcall()</td>
<td></td>
<td>Re-validate module CRC</td>
<td>module.h</td>
</tr>
<tr>
<td>_setcrc()</td>
<td></td>
<td>Call subroutine module</td>
<td>module.h</td>
</tr>
</tbody>
</table>

Multibyte Character and String Functions

Five functions, identified in Table 1-17, are available for handling multibyte characters and strings.
### Table 1-17. Multi-byte Character/String Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>mblen()</td>
<td>•</td>
<td>Determine number of bytes in multibyte character</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>mbstowcs()</td>
<td>•</td>
<td>Convert sequence of multibyte characters</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>mbtowc()</td>
<td>•</td>
<td>Determine number of bytes in multibyte characters</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>wcstombs()</td>
<td>•</td>
<td>Convert sequence of wide characters to multibyte characters</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>wctomb()</td>
<td>•</td>
<td>Convert wide character to multibyte character</td>
<td>stdlib.h</td>
</tr>
</tbody>
</table>
OS System Functions

The standard library provides a large number of functions directly accessing system calls. Functions available to provide access to selected system calls are identified in Table 1-18. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-18. OS System Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>_gets()</td>
<td>Get system global variables</td>
<td>setsys.h</td>
</tr>
<tr>
<td>_os_sysdbg()</td>
<td>Call system debugger</td>
<td>process.h</td>
</tr>
<tr>
<td>_sets()</td>
<td>Set/examine system global variables</td>
<td>setsys.h</td>
</tr>
<tr>
<td>_sysdbg()</td>
<td>Call system debugger</td>
<td>process.h</td>
</tr>
</tbody>
</table>

POSIX Messaging Functions

The OS-9 implementation of the POSIX specification supports a message queue that is either visible between process or between threads. Currently, only the threaded message queue is supported. You distinguish which type you wish to create by the message queue name passed to mq_open. If the name starts with a '/', then a process wide message queue is assumed. Otherwise, a threaded message queue is created.

There is a thread and non-thread version of the POSIX message library. At this time, OS-9 does not support process-wide message queues. Therefore, the non-threaded version of the library is merely a placeholder until the process-wide message queues are implemented. The functions all return ENOSYS—the POSIX error code that means not implemented.

The supported POSIX messaging functions are listed in Table 1-19.

Table 1-19. POSIX Messaging Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>POSIX Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>mq_close</td>
<td>•</td>
<td>Close a Message Queue</td>
<td>mqueue.h</td>
</tr>
<tr>
<td>mq_getattr</td>
<td>•</td>
<td>Get Message Queue Attributes</td>
<td>mqueue.h</td>
</tr>
<tr>
<td>mq_notify</td>
<td>•</td>
<td>Notify Process That a Message is Available on a Queue</td>
<td>mqueue.h</td>
</tr>
<tr>
<td>_mq_notify_write</td>
<td>•</td>
<td>Notify Process of Space Availability in Message Queue</td>
<td>mqueue.h</td>
</tr>
<tr>
<td>mq_open</td>
<td>•</td>
<td>Open a Message Queue</td>
<td>mqueue.h</td>
</tr>
<tr>
<td>mq_receive</td>
<td>•</td>
<td>Receive a Message From a Message Queue</td>
<td>mqueue.h</td>
</tr>
<tr>
<td>mq_send</td>
<td>•</td>
<td>Send a Message to a Message Queue</td>
<td>mqueue.h</td>
</tr>
<tr>
<td>mq_setattr</td>
<td>•</td>
<td>Set Message Queue Attributes</td>
<td>mqueue.h</td>
</tr>
<tr>
<td>mq_unlink</td>
<td>•</td>
<td>Remove a Message Queue</td>
<td>mqueue.h</td>
</tr>
</tbody>
</table>
Process Manipulation Functions

The standard library provides functions for manipulating processes as identified in Table 1-20. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-20. Process Manipulation Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>chainc()</code>, <code>chain()</code></td>
<td>Load and execute new module</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>getpid()</code></td>
<td>Determine process ID number</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_get_process_desc()</code></td>
<td>Get process descriptor copy</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_get_process_table()</code></td>
<td>Get process table entry</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>kill()</code></td>
<td>Send signal to process</td>
<td><code>signal.h</code></td>
</tr>
<tr>
<td><code>_os9_allpd()</code></td>
<td>Allocate fixed-length block of memory</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_alocproc()</code></td>
<td>Allocate process descriptor</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_aproc()</code></td>
<td>Insert process in active process queue</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os9_aproc()</code></td>
<td>Enter process in active process queue</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_chain()</code></td>
<td>Execute new primary module</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_dfork()</code></td>
<td>Fork process under control of debugger</td>
<td><code>dexec.h</code></td>
</tr>
<tr>
<td><code>_os9_dfork()</code></td>
<td>Fork process under control of debugger</td>
<td><code>dexec.h</code></td>
</tr>
<tr>
<td><code>_os_dforkm()</code></td>
<td>Fork process under control of debugger</td>
<td><code>dexec.h</code></td>
</tr>
<tr>
<td><code>_os_exec()</code></td>
<td>Start child process</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_findpd()</code></td>
<td>Find process descriptor</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_fork()</code></td>
<td>Create new process</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_forkm()</code></td>
<td>Create new process by module pointer</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_get_prtbl()</code></td>
<td>Get copy of process descriptor block table</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_getstat()</code></td>
<td>Get file/device status</td>
<td><code>sg_codes.h</code></td>
</tr>
<tr>
<td><code>_os_getsys()</code></td>
<td>Examine system global variable</td>
<td><code>sysglob.h</code></td>
</tr>
<tr>
<td><code>_os_gprdsc()</code></td>
<td>Get process descriptor copy</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os9_gspump()</code></td>
<td>Return data for specified process’ memory map</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_id()</code></td>
<td>Get process ID/user ID</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os9_id()</code></td>
<td>Get process ID/user ID</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_nproc()</code></td>
<td>Start next process</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_rtnprc()</code></td>
<td>Deallocate process descriptor</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_setpr()</code></td>
<td>Set process priority</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_sleep()</code></td>
<td>Put calling process to sleep</td>
<td><code>signal.h</code></td>
</tr>
<tr>
<td><code>_os9_sleep()</code></td>
<td>Put calling process to sleep</td>
<td><code>signal.h</code></td>
</tr>
<tr>
<td><code>_os_suspend()</code></td>
<td>Suspend process</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>_os_wait()</code></td>
<td>Wait for child process to terminate</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>os9exec()</code></td>
<td>OS-9 for 68K system call processing</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>os9fork()</code>, <code>os9forkc()</code></td>
<td>Create process</td>
<td><code>process.h</code></td>
</tr>
<tr>
<td><code>os9kexec()</code></td>
<td>OS-9 create process</td>
<td><code>types.h</code></td>
</tr>
</tbody>
</table>
Resource Locks

On OS-9 for 68K systems, the functions identified in Table 1-21 are available for use with resource locks. For more information about resource locks, refer to the OS-9 for 68K Technical Manual.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_acqlk()</td>
<td>Acquire ownership of resource lock</td>
<td>lock.h</td>
</tr>
<tr>
<td>_os_caqlk()</td>
<td>Conditionally acquire ownership of resource lock</td>
<td>lock.h</td>
</tr>
<tr>
<td>_os_crlk()</td>
<td>Create new resource lock descriptor</td>
<td>lock.h</td>
</tr>
<tr>
<td>_os_ddlk()</td>
<td>Check for deadlock situation</td>
<td>process.h</td>
</tr>
<tr>
<td>_os_dellk()</td>
<td>Delete existing lock descriptor</td>
<td>lock.h</td>
</tr>
<tr>
<td>_os_rellk()</td>
<td>Release ownership of resource lock</td>
<td>lock.h</td>
</tr>
<tr>
<td>_os_waitlk()</td>
<td>Activate next process waiting to acquire lock</td>
<td>lock.h</td>
</tr>
</tbody>
</table>

Searching and Sorting Functions

Functions included for searching and sorting are identified in Table 1-22.

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>bsearch()</td>
<td></td>
<td>Search array</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>qsort()</td>
<td></td>
<td>Quick sort</td>
<td>stdlib.h</td>
</tr>
</tbody>
</table>

String Handling Functions

The C language does not have a character string data type. Instead, it stores strings as character arrays and the standard library provides functions to manipulate them. The functions are identified in Table 1-23. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>findnstr()</td>
<td></td>
<td>Search string for pattern</td>
<td>strings.h</td>
</tr>
<tr>
<td>findstr()</td>
<td></td>
<td>Search string for pattern</td>
<td>strings.h</td>
</tr>
<tr>
<td>index()</td>
<td></td>
<td>Search for character in string</td>
<td>strings.h</td>
</tr>
</tbody>
</table>
Table 1-23. String Handling Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>rindex()</td>
<td></td>
<td>Search for character in string</td>
<td>strings.h</td>
</tr>
<tr>
<td>sl_add(),</td>
<td></td>
<td>Manipulate string lists</td>
<td>stringlist.h</td>
</tr>
<tr>
<td>sl_find(),</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sl_free(),</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sl_init()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strcasecmp()</td>
<td></td>
<td>Compare null-terminated strings</td>
<td>strings.h</td>
</tr>
<tr>
<td>strncasecmp()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strcat()</td>
<td>•</td>
<td>String catenation</td>
<td>string.h</td>
</tr>
<tr>
<td>strchr()</td>
<td>•</td>
<td>Locate string</td>
<td>string.h</td>
</tr>
<tr>
<td>strcmp()</td>
<td></td>
<td>String comparison</td>
<td>string.h</td>
</tr>
<tr>
<td>strcoll()</td>
<td>•</td>
<td>String comparison</td>
<td>string.h</td>
</tr>
<tr>
<td>strcpy()</td>
<td>•</td>
<td>String copy</td>
<td>string.h</td>
</tr>
<tr>
<td>strcspn()</td>
<td>•</td>
<td>Get string length</td>
<td>string.h</td>
</tr>
<tr>
<td>strerror()</td>
<td>•</td>
<td>Map error message string</td>
<td>string.h</td>
</tr>
<tr>
<td>strftime()</td>
<td>•</td>
<td>Place formatted time in buffer</td>
<td>time.h</td>
</tr>
<tr>
<td>strlcpy()</td>
<td></td>
<td>Copy string src</td>
<td>strings.h</td>
</tr>
<tr>
<td>strlcat()</td>
<td></td>
<td>Concatenate string src</td>
<td>strings.h</td>
</tr>
<tr>
<td>strncat()</td>
<td>•</td>
<td>String catenation</td>
<td>string.h</td>
</tr>
<tr>
<td>strncmp()</td>
<td>•</td>
<td>String comparison</td>
<td>string.h</td>
</tr>
<tr>
<td>strncpy()</td>
<td>•</td>
<td>String copy</td>
<td>string.h</td>
</tr>
<tr>
<td>strpbrk()</td>
<td>•</td>
<td>Locate first occurrence of string</td>
<td>string.h</td>
</tr>
<tr>
<td>strptime()</td>
<td>•</td>
<td>Convert character string in tmf</td>
<td>time.h</td>
</tr>
<tr>
<td>strrchr()</td>
<td>•</td>
<td>Locate last occurrence of string</td>
<td>string.h</td>
</tr>
<tr>
<td>strsep()</td>
<td></td>
<td>Replace character in *stringp</td>
<td>strings.h</td>
</tr>
<tr>
<td>strspn()</td>
<td>•</td>
<td>Compute string length</td>
<td>string.h</td>
</tr>
<tr>
<td>strstr()</td>
<td>•</td>
<td>Locate first occurrence of string</td>
<td>string.h</td>
</tr>
<tr>
<td>strtod()</td>
<td>•</td>
<td>String to double conversion</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>strtok()</td>
<td>•</td>
<td>Break string into tokens</td>
<td>string.h</td>
</tr>
<tr>
<td>strtoll()</td>
<td>•</td>
<td>String to long conversion</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>strtoul()</td>
<td>•</td>
<td>String to unsigned long conversion</td>
<td>stdlib.h</td>
</tr>
<tr>
<td>strxfrm()</td>
<td>•</td>
<td>Transform string</td>
<td>string.h</td>
</tr>
</tbody>
</table>
Terminal Manipulation Functions (Termcap)

The following table lists the terminal manipulation functions.

Table 1-24. Termcap Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>tgetent()</td>
<td>Get termcap entries</td>
<td>termcap.h</td>
</tr>
<tr>
<td>tgetflag()</td>
<td>Check terminal capability presence</td>
<td>termcap.h</td>
</tr>
<tr>
<td>tgetnum()</td>
<td>Get terminal capability ID</td>
<td>termcap.h</td>
</tr>
<tr>
<td>tgetstr()</td>
<td>Get terminal capability</td>
<td>termcap.h</td>
</tr>
<tr>
<td>tgoto()</td>
<td>Get cursor movement capability</td>
<td>termcap.h</td>
</tr>
<tr>
<td>tputs()</td>
<td>Output capability string</td>
<td>termcap.h</td>
</tr>
</tbody>
</table>

Time Functions

The TZ environment variable is used to specify the time zone for the C functions to use. TZ should have the following format:

```
zzz[+/-n][:ddd]
```

zzz is one of the supported time zone names listed in Table 1-25.

Table 1-25. Time Zones and Descriptions

<table>
<thead>
<tr>
<th>Zone</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMT/UTC</td>
<td>Greenwich Mean Time (or Coordinated Universal Time)</td>
</tr>
<tr>
<td>PST/PDT</td>
<td>USA Pacific Standard Time/Daylight Saving Time</td>
</tr>
<tr>
<td>MST/MDT</td>
<td>USA Mountain Standard Time/Daylight Saving Time</td>
</tr>
<tr>
<td>CST/CDT</td>
<td>USA Central Standard Time/Daylight Saving Time</td>
</tr>
<tr>
<td>EST/EDT</td>
<td>USA Eastern Standard Time/Daylight Saving Time</td>
</tr>
<tr>
<td>YST</td>
<td>Yukon Standard Time (Most of Alaska)</td>
</tr>
<tr>
<td>AST</td>
<td>Aleutian/Hawaiian Standard Time</td>
</tr>
<tr>
<td>EET</td>
<td>Eastern European Time</td>
</tr>
<tr>
<td>CET</td>
<td>Central European Time</td>
</tr>
<tr>
<td>WET</td>
<td>Western European Time</td>
</tr>
<tr>
<td>JST</td>
<td>Japan Standard Time</td>
</tr>
<tr>
<td>MIT</td>
<td>Midway Islands Time</td>
</tr>
<tr>
<td>HST</td>
<td>Hawaii Standard Time</td>
</tr>
<tr>
<td>PNT</td>
<td>Phoenix Standard Time</td>
</tr>
<tr>
<td>IRT</td>
<td>Indiana Eastern Standard Time</td>
</tr>
<tr>
<td>PRT</td>
<td>Puerto Rico Standard Time</td>
</tr>
<tr>
<td>CNT</td>
<td>Canada Newfoundland Time</td>
</tr>
<tr>
<td>AGT</td>
<td>Argentina Standard Time</td>
</tr>
<tr>
<td>BEZ</td>
<td>Brazilian Standard Time</td>
</tr>
<tr>
<td>CAT</td>
<td>Central Africa</td>
</tr>
<tr>
<td>ECT</td>
<td>European Central Time</td>
</tr>
<tr>
<td>ART</td>
<td>Arabic Standard Time</td>
</tr>
</tbody>
</table>
Table 1-25. Time Zones and Descriptions (Continued)

<table>
<thead>
<tr>
<th>Zone</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAT</td>
<td>Eastern African Time</td>
</tr>
<tr>
<td>MET</td>
<td>Middle Eastern Time</td>
</tr>
<tr>
<td>NET</td>
<td>Near East Time</td>
</tr>
<tr>
<td>PLT</td>
<td>Pakistan Lahore Time</td>
</tr>
<tr>
<td>IST</td>
<td>India Standard Time</td>
</tr>
<tr>
<td>BST</td>
<td>Bangladesh Standard Time</td>
</tr>
<tr>
<td>VST</td>
<td>Vietnam Standard Time</td>
</tr>
<tr>
<td>CTT</td>
<td>China Taiwan Standard Time</td>
</tr>
<tr>
<td>ACT</td>
<td>Australia Central Time</td>
</tr>
<tr>
<td>AET</td>
<td>Australian Eastern Time</td>
</tr>
<tr>
<td>SST</td>
<td>Solomon Standard Time</td>
</tr>
<tr>
<td>NST</td>
<td>New Zealand Standard Time</td>
</tr>
</tbody>
</table>

n is the optional number of minutes east (+) or west (−) of the time zone.

ddd is the option controlling Daylight Saving Time.

Recognized codes for ddd are shown in Table 1-26.

Table 1-26. Valid ddd Codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>Do not use Daylight Saving Time (DST)</td>
</tr>
<tr>
<td>usa</td>
<td>Conforms to the US Uniform Time Act of 1967 and its various amendments through 1987</td>
</tr>
<tr>
<td></td>
<td>DST begins the first Sunday in April and ends the last Sunday in October.</td>
</tr>
<tr>
<td>eur</td>
<td>Observe European DST, which begins the last Sunday in March and ends the last Sunday in October.</td>
</tr>
<tr>
<td>bra</td>
<td>Observe Brazilian DST, which begins the first Sunday in October and ends the first Sunday after February 11.</td>
</tr>
<tr>
<td>aus</td>
<td>Observe Australian DST, which begins the last Sunday in October and ends the last Sunday in March.</td>
</tr>
<tr>
<td>nz</td>
<td>Observe New Zealand DST, which begins the first Sunday in October and ends the first Sunday after March 15.</td>
</tr>
</tbody>
</table>

For example, the following command specifies that the compiler should use the USA Eastern Daylight Saving Time and should conform to the US Uniform Time Act of 1967:

```bash
setenv TZ EDT:usa
```
Each time zone has its own default Daylight Saving Time setting:

- American time zones default to \texttt{usa}.

Daylight Saving Time, for the U.S. and its territories, is NOT observed in Hawaii (HST), American Samoa, Guam, the Virgin Islands (MIT), Puerto Rico (PRT), the Eastern Time Zone portion of the State of Indiana (IET), and the state of Arizona (not the Navajo Indian Reservation, which does observe) (PNT). Navajo Nation participates in the Daylight Saving Time policy, due to its large size and location in three states.

- European time zones default to \texttt{eur}.
- Brazilian time zone defaults to \texttt{bra}.
- Australian time zones default to \texttt{aus}.
- New Zealand time zone defaults to \texttt{nz}.
- GMT/UTC time zones default to \texttt{no}.

If the \texttt{TZ} environment variable is not set and, if on OS-9, \texttt{m_tmzone} is not set, \texttt{time()} uses USA Central Standard Time (CST).

In addition, the time functions, with the exception of \texttt{clock()}, can be used in both system and user state. However, to use the functions in system state, you will need \texttt{define_environ} must be defined as follows:

```c
void * xenviron;
void **_environ = &xenviron
```

Ultra C provides time functions identified in Table 1-27. ANSI prototypes for non-ANSI functions are achieved by defining the macro name \_OPT_PROTOS during the compile.

\textbf{Table 1-27. Time Functions}

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{asctime()}</td>
<td><code>•</code></td>
<td>Convert broken-down time to string format</td>
<td>time.h</td>
</tr>
<tr>
<td>\texttt{clock()}</td>
<td><code>•</code></td>
<td>Get processor time</td>
<td>time.h</td>
</tr>
<tr>
<td>\texttt{ctime()}</td>
<td><code>•</code></td>
<td>Convert calendar time to string format</td>
<td>time.h</td>
</tr>
<tr>
<td>\texttt{difftime()}</td>
<td><code>•</code></td>
<td>Find temporal difference</td>
<td>time.h</td>
</tr>
<tr>
<td>\texttt{getime()}</td>
<td><code>•</code></td>
<td>Get system time</td>
<td>time.h</td>
</tr>
<tr>
<td>\texttt{gmtime()}</td>
<td><code>•</code></td>
<td>Convert calendar time to Greenwich Mean Time</td>
<td>time.h</td>
</tr>
<tr>
<td>_gregorian()</td>
<td></td>
<td>Convert date/time to Gregorian value</td>
<td>time.h</td>
</tr>
<tr>
<td>_julian()</td>
<td></td>
<td>Convert date/time to Julian value</td>
<td>time.h</td>
</tr>
</tbody>
</table>
Table 1-27. Time Functions (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>ANSI Standard</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>localtime()</td>
<td>•</td>
<td>Convert calendar time to local time</td>
<td>time.h</td>
</tr>
<tr>
<td>mktime()</td>
<td>•</td>
<td>Convert broken-down time to calendar time</td>
<td>time.h</td>
</tr>
<tr>
<td>_os_gettime()</td>
<td></td>
<td>Get system date and time</td>
<td>time.h</td>
</tr>
<tr>
<td>_os9_gettime()</td>
<td></td>
<td>Get system date/time</td>
<td>time.h</td>
</tr>
<tr>
<td>_os_gregorian()</td>
<td></td>
<td>Get Gregorian date</td>
<td>time.h</td>
</tr>
<tr>
<td>_os_julian()</td>
<td></td>
<td>Get Julian date</td>
<td>time.h</td>
</tr>
<tr>
<td>_os_setime()</td>
<td></td>
<td>Set system date/time</td>
<td>time.h</td>
</tr>
<tr>
<td>_os9_setime()</td>
<td></td>
<td>Set system date/time</td>
<td>time.h</td>
</tr>
<tr>
<td>setime()</td>
<td></td>
<td>Set system time</td>
<td>time.h</td>
</tr>
<tr>
<td>setitimer()</td>
<td></td>
<td>Set a timer to specified value</td>
<td>os9time.h</td>
</tr>
<tr>
<td>strftime()</td>
<td>•</td>
<td>Place formatted time in buffer</td>
<td>time.h</td>
</tr>
<tr>
<td>sys_mkmtime()</td>
<td></td>
<td>Convert local time to Greenwich Mean Time</td>
<td>time.h</td>
</tr>
<tr>
<td>_sysdate()</td>
<td></td>
<td>Get current system date/time</td>
<td>time.h</td>
</tr>
<tr>
<td>time()</td>
<td></td>
<td>Get calendar time</td>
<td>time.h</td>
</tr>
<tr>
<td>utimes()</td>
<td>•</td>
<td>Get process times.</td>
<td>sys/time.h</td>
</tr>
</tbody>
</table>

Processor-Specific Functions

Functions available only on the 80x86 family of systems are identified in Table 1-28. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-28. 80x86-Only Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_current_tss()</td>
<td>Get current task segment</td>
<td>regs.h</td>
</tr>
<tr>
<td>get_excpt_base()</td>
<td>Return exception table base address</td>
<td>regs.h</td>
</tr>
<tr>
<td>get_gdtr()</td>
<td>Get global descriptor pointer</td>
<td>regs.h</td>
</tr>
<tr>
<td>make_gdesc()</td>
<td>Make global descriptor table entry</td>
<td>regs.h</td>
</tr>
<tr>
<td>make_idesc()</td>
<td>Make interrupt descriptor table entry</td>
<td>regs.h</td>
</tr>
<tr>
<td>set_excpt_base()</td>
<td>Set exception table base address</td>
<td>regs.h</td>
</tr>
<tr>
<td>set_gdtr()</td>
<td>Set global descriptor pointer</td>
<td>regs.h</td>
</tr>
</tbody>
</table>
These functions are available only on the PowerPC processors as identified in Table 1-29. ANSI prototypes for non-ANSI functions are achieved by defining the macro name _OPT_PROTOS during the compile.

Table 1-29. Power PC-Only Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>get</em>&lt;name&gt;()</td>
<td>Read the value of SPRs, DCRs, PREGs, and TBs</td>
<td>getset.h</td>
</tr>
<tr>
<td><em>set</em>&lt;name&gt;()</td>
<td>Set the value of SPRs, DCRs, PREGs, and TBs</td>
<td>getset.h</td>
</tr>
</tbody>
</table>

Implementation-Defined Behavior

The following are the implementation-defined issues pertaining to the Ultra C library. Each item contains an implementation-defined issue. The number in parentheses included with each item indicates the location in the ANSI specification where you can find more information.

- The null pointer constant to which the macro NULL expands (4.1.5). NULL expands to (void *) 0.

- The diagnostic printed by and the termination behavior of `assert()` (4.2). `assert()` prints a line with the following syntax if the assertion fails (if the NDEBUG macro is not defined):

  Assertion failed:<expression>, file <file>, line <line>

  <expression> expression being tested
  <file> file name of the file containing the assert
  <line> line number of the file containing assert

  `abort()` is called after the assertion failure line is printed. This raises the SIGABRT signal that may be handled by the application.

  Refer to `signal()` for more information about SIGABRT.

- The sets of characters tested by `isalnum()`, `isalpha()`, `iscntrl()`, `islower()`, `isprint()`, and `isupper()` (4.3.1).

  `isalnum()` return non-zero for characters that `isdigit()` or `isalpha()` return non-zero.
  `isalpha()` return non-zero for characters that `islower()` or `isupper()` returns non-zero.
  `iscntrl()` return non-zero for ASCII characters with values from 0 to 0x1f and 0x7f.
  `islower()` return non-zero for ASCII characters with values from 0x61 to 0x7a.
  `isprint()` return non-zero for ASCII characters with values from 0x20 to 0x7e.
  `isupper()` return non-zero for ASCII characters with values from 0x41 to 0x5a.
• Values returned by mathematics functions on domain errors (4.5.1). The mathematics functions return HUGE_VAL on domain errors.

• If the mathematics functions set the integer expression errno to the value of the macro ERANGE on underflow range errors (4.5.1). errno is not set to ERANGE on underflow errors.

• Whether a domain error occurs or zero is returned when the mod() function has a second argument of zero (4.5.6.4). mod() returns zero if it has a second argument of zero.

• The set of signals for the signal() function (4.7.1.1). The set of signals for the signal function varies with the operating system and the processor. A set of signals is also available regardless of operating system.

ANSI “Core” Signals
SIGABRT SIGFPE
SIGINT SIGSEGV
SIGILL

OS-9 “Core” Signals
SIGKILL SIGWAKE SIGQUIT
SIGHUP SIGALRM SIGPIPE
SIGUSR1 SIGUSR2

OS-9 68K Family Signals
SIGADDR SIGCHK
SIGPRIV SIGTRAPV
SIGINT SIGTRACE
SIG1010 SIG1111

OS-9 80x86 Family Signals
SIGPROT SIGSTACK
SIGINT SIGDBLFLT
SIGBRK PT SIGBND CHK

OS-9 PowerPC™ Family Signals
SIGCHKS SIGINST
SIGPRIVSIGALIGN

The semantics for each signal recognized by signal() (4.7.1.1).
The following table lists family signals.

**Table 1-30. Signals**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABRT</td>
<td>Abnormal termination such as initiated by the <code>abort()</code> function.</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>An erroneous arithmetic operation such as zero divide or an operation resulting in overflow. This signal is generated on target-specific conditions.</td>
</tr>
<tr>
<td>SIGILL</td>
<td>Detection of an invalid function image such as illegal instruction. This signal is generated on target-specific conditions.</td>
</tr>
<tr>
<td>SIGINT</td>
<td>Receipt of an interactive attention signal generated by the keyboard interrupt character on OS-9 for 68K and OS-9.</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>An invalid access to storage. This signal is generated on target-specific conditions.</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>A termination request sent to the program.</td>
</tr>
<tr>
<td>SIGKILL</td>
<td>A death request sent to the program. This signal cannot be ignored or handled.</td>
</tr>
<tr>
<td>SIGWAKE</td>
<td>A wakeup request sent to the program. If a process is active when this signal arrives, it is ignored. Otherwise, the process is activated, but the signal handling function is not called.</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>Receipt of an interactive abort signal generated by the keyboard quit character on OS-9.</td>
</tr>
<tr>
<td>SIGHUP</td>
<td>Modem hangup signal. This signal is sent to a process when a modem connection is lost.</td>
</tr>
<tr>
<td>SIGALRM</td>
<td>Not automatically sent. Programs can use this signal for an alarm call.</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>Not automatically sent.</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>Condition is user defined.</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>Condition is user defined.</td>
</tr>
</tbody>
</table>

**Processor-Specific Family Signals**

For information related to the exceptions, see the reference manual for your specific processor.

**68K Family Processors**

The OS-9 68K family signals sent when their associated exception occurs are identified in Table 1-31.

**Table 1-31. OS-9 for 68K Signals**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGADDR</td>
<td>Address Error</td>
</tr>
<tr>
<td>SIGCHK</td>
<td>CHK Instruction</td>
</tr>
<tr>
<td>SIGTRAPV</td>
<td>TRAPV Instruction</td>
</tr>
<tr>
<td>SIGPRIV</td>
<td>Privilege Violation</td>
</tr>
<tr>
<td>SIGTRACE</td>
<td>Trace</td>
</tr>
</tbody>
</table>
80x86 Family Processors

The OS-9 80x86 family signals shown in Table 1-32 are sent when their associated exception occurs.

Table 1-32. OS-9 80x86 Family Exception Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGGPROT</td>
<td>General Protection</td>
</tr>
<tr>
<td>SIGSTACK</td>
<td>Stack Exception</td>
</tr>
<tr>
<td>SIGSEGNP</td>
<td>Segment Not Present</td>
</tr>
<tr>
<td>SIGINVTSS</td>
<td>Invalid TSS</td>
</tr>
<tr>
<td>SIGDBLFLT</td>
<td>Double Fault</td>
</tr>
<tr>
<td>SIGBNDCHK</td>
<td>Bounds Check</td>
</tr>
<tr>
<td>SIGBRKPT</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>SIGNMI</td>
<td>Non-Maskable Interrupt</td>
</tr>
<tr>
<td>SIGDBG</td>
<td>Debug Exceptions</td>
</tr>
</tbody>
</table>

PowerPC Family Processors

The OS-9 PowerPC family signals shown in Table 1-33 are sent when their associated exception occurs.

Table 1-33. OS-9 PowerPC Family Exception Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGCHECK</td>
<td>Machine Check</td>
</tr>
<tr>
<td>SIGINST</td>
<td>Instruction Access</td>
</tr>
<tr>
<td>SIGPRIV</td>
<td>Privilege Violation</td>
</tr>
<tr>
<td>SIGALIGN</td>
<td>Alignment</td>
</tr>
</tbody>
</table>

ARM Family Processors

The OS-9 ARM family signal shown in Table 1-34 is sent when its associated exception occurs.

Table 1-34. OS-9 ARM Family Exception Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGALIGN</td>
<td>Alignment</td>
</tr>
</tbody>
</table>
SH Family Processors

The OS-9 SH3 family signal shown in Table 1-35 is sent when the associated exception occurs.

Table 1-35. OS-9 SH3 Family Exception Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGALIGN</td>
<td>Alignment</td>
</tr>
</tbody>
</table>

SPARC Family Processors

The OS-9 SPARC family signals shown in Table 1-36 are sent when their associated exception occurs.

Table 1-36. OS-9 SPARC Family Exception Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGALIGN</td>
<td>Alignment</td>
</tr>
<tr>
<td>SIGWINDOWOV</td>
<td>Window Overflow</td>
</tr>
<tr>
<td>SIGWINDOWUV</td>
<td>Window Underflow</td>
</tr>
<tr>
<td>SIGATAGOV</td>
<td>Tag Overflow</td>
</tr>
<tr>
<td>SIGCPE</td>
<td>Coprocessor Exception</td>
</tr>
</tbody>
</table>

- The default handling and the handling at program startup for each signal recognized by the signal() function (4.7.1.1).
  
  For each signal recognized by signal(), the default handling is program termination and the program start-up condition is SIG_DFL. The program terminates with an exit status related to the exception or the signal number for non-exception related signals.

- If the equivalent of signal(sig, SIG_DFL); is not executed prior to the call of the signal handler, the blocking of the signal that is performed (4.7.1.1).

  signal(sig, SIG_DFL) is executed before the call to the signal handler. Signals are blocked for the duration of the signal handler, unless the program takes some action that unblocks signal(). Refer to your operating system technical manual for more information.

- Whether the default handling is reset if the SIGILL signal is received by a handler specified to the signal function (4.7.1.1).

  The default handling is reset on all signals.

- Whether the last line of a text stream requires a terminating newline character (4.9.2).

  A terminating newline is not required as the last character of a text stream.

- Whether space characters that are written out to a text stream immediately before a newline character appear when read (4.9.2).

  Space characters written to a text stream immediately before a newline character appears when read.
The number of null characters that may be appended to data written to a binary stream (4.9.2).
Null characters are not appended to data written to a binary stream.

Whether the file position indicator of an append mode stream is initially positioned at the beginning or end of the file (4.9.3).
The file position indicator of an append mode stream is initially positioned at the beginning of the file. Regardless, all writes to a stream in append mode are started at the current end-of-file.

Whether a write on a text stream causes the associated file to be truncated beyond that point (4.9.3).
No truncation of this sort occurs.

The characteristics of file buffering (4.9.3).
The file buffering decision is made when the first request is made to the stream. If the buffering type and size have not been set explicitly by `setvbuf()`, the determination is made based on the device with which the stream is associated. If the device is interactive, line buffering is used. Otherwise, full buffering is used. If the device is RBF (Random Block File Manager), the buffer size is \(2 \times \text{current_sector_size}\).

Whether a zero-length file actually exists (4.9.3).
A zero length file actually does exist.

The rules for composing valid file names (4.9.3).
The rules vary depending on the operating system:
OS-9 for 68KFile names can contain 1 to 28 upper or lower case letters, digits, underscores (_), periods (.), or dollar signs ($).
OS-9File names can contain 1 to 43 upper or lower case letters, digits, underscores (_), periods (.), or dollar signs ($).

Whether the same file can be open multiple times (4.9.3).
A program can open a file more than once, but fully buffered reading and writing to the same file has undefined results.

The effect of the `remove()` function on an open file (4.9.4.1).
`remove()` returns -1 with `errno` set to `EOS_SHARE` if called with the name of an open file.

The effect if a file with the new name exists prior to a call to the `rename()` function (4.9.4.2).
Assuming the calling program has write permission, the file with the new name is deleted and recreated with the new information.

The output for `%p` conversion in the `fprintf()` function (4.9.6.1).
The pointer to `void` is printed as an unsigned hexadecimal integer.
• The input for `%p` conversion in the `fscanf()` function (4.9.6.2).
The input for `%p` conversion is an unsigned hexadecimal integer.

• The interpretation of a ‘-’ character that is neither the first nor the last character in the scanlist for `%[ conversion in the `fscanf()` function (4.9.6.2).
A hyphen (–) character is interpreted simply as the addition of – to the scanlist.

• The value to which the macro `errno` is set by the `fgetpos()` or `ftell()` function on failure (4.9.9.1, 4.9.9.4).

  `errno` is set to the operating system specific error that is returned by either `_os_gs_size()` (in append mode) or `_os_gs_pos()`.

• The messages generated by the `perror()` function (4.9.10.4).
The messages generated by the `perror()` function are located in the system `errmsg` file (/dd/SYS/errmsg, /h0/SYS/errmsg, or /d0/SYS/errmsg). Refer to your operating system technical manual for detailed information. Error messages are generally in the form:

  `<message>`: `#<major>:<minor>` <detail>

  `<message>` is the user-passed message string.
  `<major>` is the family of error.
  `<minor>` is the error number within the family.
  `<detail>` is the information from the system `errmsg` file (if available).

• The behavior of the `calloc()`, `malloc()`, or `realloc()` function if the size requested is zero (4.10.3).
The `calloc()`, `malloc()`, and `realloc()` functions return `NULL` if the requested size is zero.

• The behavior of the `abort()` function with regard to open and temporary files (4.10.4.1).
If the SIGABRT signal handler returns, the program is terminated abnormally flushing open streams, closing open streams, and removing temporary files.

• The status returned by the `exit()` function if the value of the argument is other than zero, `EXIT_SUCCESS`, or `EXIT_FAILURE` (4.10.4.3).
The integer value passed to `exit()` is the exit status of the program.
• The set of environment names and the method for altering the environment list used by the `getenv()` function (4.10.4.3).

The common set of environment names depends on the operating system. Refer to your operating system’s user manual for more information.

The environment list is implemented as a NULL terminated array of pointers to characters. Each string is in the form: `<name>=<value>`

  `<name>` is the name of the environment variable.

  `<value>` is the current value of the environment variable.

To modify the list, you must either find the existing entry or enlarge the array and add an entry. The base of the array is stored in the external variable `_environ`.

• The contents and mode of execution of the string by the `system()` function (4.10.4.5).

The contents of an execution string given to `system()` can be any valid shell command line. The shell used to execute the string is either the current value of the `SHELL` environment variable or `shell` if `SHELL` is unavailable.

• The contents of the error message strings returned by the `strerror()` function (4.11.6.2).

The strings generated by the `strerror()` function are located in the system `errmsg` file (`/dd/SYS/errmsg, /h0/SYS/errmsg, or /d0/SYS/errmsg`). Refer to your operating system technical manual for detailed information. Error messages are generally in the form: `#<major>:<minor> <detail>`

  `<major>` is the family of error.

  `<minor>` is the error number within the family.

  `<detail>` is the information from the system `errmsg` file (if available).

• The local time zone and Daylight Saving Time (4.12.1).

The local time zone and Daylight Saving Time are determined by the `TZ` environment variable on OS-9 for 68K and the `TZ` variable or the `m_tmzone` field in the `init` module on OS-9. Failing to find user defined values, the defaults are Central Standard Time and Central Daylight Time.

• The era for the `clock()` function (4.12.2.1).

The era for `clock()` is the elapsed ticks since the program was forked.
Equivalents for the sys_clib.l Functions

The functions in the sys_clib.l library exist mainly for compatibility with the Microware K & R C compiler, although the calls are accepted in all modes. Whenever possible, you should use the functions in the os_lib.l and clib.l libraries that have the same functionality. The following is a list of the sys_clib.l functions and the corresponding calls in either os_lib.l or clib.l.

Table 1-37. sys_clib.l Function Equivalents

<table>
<thead>
<tr>
<th>sys_clib.l Function</th>
<th>Corresponding Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>access()</td>
<td>_os_open()</td>
</tr>
<tr>
<td>create()</td>
<td>_os_create()</td>
</tr>
<tr>
<td>alm_atdate()</td>
<td>_os_alarm_attime()/_os9_alarm_atdate()</td>
</tr>
<tr>
<td>detach()</td>
<td>_os_detach()</td>
</tr>
<tr>
<td>alm_atjul()</td>
<td>_os_alarm_attime()/_os9_alarm_atjul()</td>
</tr>
<tr>
<td>dup()</td>
<td>_os_dup()</td>
</tr>
<tr>
<td>alm_cycle()</td>
<td>_os_alarm_cycle()</td>
</tr>
<tr>
<td>ebrk()</td>
<td>malloc()</td>
</tr>
<tr>
<td>alm_delete()</td>
<td>_os_alarm_delete()</td>
</tr>
<tr>
<td>_errmsg()</td>
<td>strerror()</td>
</tr>
<tr>
<td>alm_set()</td>
<td>_os_alarm_set()</td>
</tr>
<tr>
<td>_ev_creat()</td>
<td>_os_ev_creat()</td>
</tr>
<tr>
<td>attach()</td>
<td>_os_attach()</td>
</tr>
<tr>
<td>_ev_delete()</td>
<td>_os_ev_delete()</td>
</tr>
<tr>
<td>_ev_info()</td>
<td>_os_ev_info()</td>
</tr>
<tr>
<td>chainc(), chain()</td>
<td>_os_chain()</td>
</tr>
<tr>
<td>_ev_link()</td>
<td>_os_ev_link()</td>
</tr>
<tr>
<td>chdir()</td>
<td>_os_chdir()</td>
</tr>
<tr>
<td>_ev_pulse()</td>
<td>_os_ev_pulse()</td>
</tr>
<tr>
<td>chmod()</td>
<td>_os_ss_attr()</td>
</tr>
<tr>
<td>_ev_read()</td>
<td>_os_ev_read()</td>
</tr>
<tr>
<td>chown()</td>
<td>_os_ss_fd()</td>
</tr>
<tr>
<td>_ev_set()</td>
<td>_os_ev_set()</td>
</tr>
<tr>
<td>chdir()</td>
<td>_os_chdir()</td>
</tr>
<tr>
<td>_ev_setr()</td>
<td>_os_ev_setr()</td>
</tr>
<tr>
<td>_cleareof(), cleareof()</td>
<td>clearerr()</td>
</tr>
<tr>
<td>_ev_signal()</td>
<td>_os_ev_signal()</td>
</tr>
<tr>
<td>_ev_unlink()</td>
<td>_os_ev_unlink()</td>
</tr>
<tr>
<td>close()</td>
<td>_os_close()</td>
</tr>
<tr>
<td>_ev_wait()</td>
<td>_os_ev_wait()/_os9_ev_wait()</td>
</tr>
<tr>
<td>closedir()</td>
<td>_os_close()</td>
</tr>
<tr>
<td>_ev_waitr()</td>
<td>_os_ev_waitr()/_os9_ev_waitr()</td>
</tr>
<tr>
<td>_cmpnam()</td>
<td>_os_cmpnam()</td>
</tr>
<tr>
<td>_exit()</td>
<td>_os_exit()</td>
</tr>
<tr>
<td>_cpymem()</td>
<td>_os_cpymem()</td>
</tr>
<tr>
<td>fdopen()</td>
<td>none</td>
</tr>
</tbody>
</table>
### Table 1-37. sys_clib.l Function Equivalents (Continued)

<table>
<thead>
<tr>
<th>sys_clib.l Function</th>
<th>Corresponding Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>crc()</td>
<td>_os_crc()</td>
</tr>
<tr>
<td>findnstr()</td>
<td>strstr()</td>
</tr>
<tr>
<td>creat()</td>
<td>_os_create()</td>
</tr>
<tr>
<td>findstr()</td>
<td>strstr()</td>
</tr>
<tr>
<td>_get_module_dir()</td>
<td>_os_get_moddir()</td>
</tr>
<tr>
<td>mkdir()</td>
<td>_os_mkdir()</td>
</tr>
<tr>
<td>_get_process_desc()</td>
<td>_os_gprdesc()</td>
</tr>
<tr>
<td>make_module()</td>
<td>_os_mkmodule()</td>
</tr>
<tr>
<td>_get_process_table()</td>
<td>_os_get_prtbl()</td>
</tr>
<tr>
<td>_mkdata_module()</td>
<td>_os_datmod()</td>
</tr>
<tr>
<td>mktime()</td>
<td>_os_gettime()/_os9_gettime()</td>
</tr>
<tr>
<td>mknod()</td>
<td>_os_makdir()</td>
</tr>
<tr>
<td>getpid()</td>
<td>_os_id()</td>
</tr>
<tr>
<td>mktemp()</td>
<td>tmpnam()</td>
</tr>
<tr>
<td>getstat()</td>
<td>_os_getstat()</td>
</tr>
<tr>
<td>modcload()</td>
<td>_os_loadp()</td>
</tr>
<tr>
<td>_getsys()</td>
<td>_os_getsys()</td>
</tr>
<tr>
<td>modlink()</td>
<td>_os_link()</td>
</tr>
<tr>
<td>getuid()</td>
<td>_os_id()</td>
</tr>
<tr>
<td>modload()</td>
<td>_os_load()</td>
</tr>
<tr>
<td>getw()</td>
<td>getc()</td>
</tr>
<tr>
<td>getw()</td>
<td>is replaced by performing two calls to getc()</td>
</tr>
<tr>
<td>modloadp()</td>
<td>_os_loadp()</td>
</tr>
<tr>
<td>_gregorian()</td>
<td>_os_gregorian()</td>
</tr>
<tr>
<td>munlink()</td>
<td>_os_unlink()</td>
</tr>
<tr>
<td>_gs_devn()</td>
<td>_os_gs_devnm()</td>
</tr>
<tr>
<td>munload()</td>
<td>_os_unload()</td>
</tr>
<tr>
<td>_gs_eof()</td>
<td>_os_gs_eof()</td>
</tr>
<tr>
<td>open()</td>
<td>_os_open()</td>
</tr>
<tr>
<td>_gs_gfd()</td>
<td>_os_gs_fd()</td>
</tr>
<tr>
<td>opendir()</td>
<td>_os_open()</td>
</tr>
<tr>
<td>_gs_opt()</td>
<td>_os_gs_popt()</td>
</tr>
<tr>
<td>os9exec()</td>
<td>_os_exec()</td>
</tr>
<tr>
<td>_gs_pos()</td>
<td>_os_gs_pos()</td>
</tr>
<tr>
<td>os9fork(), os9forkc()</td>
<td>_os_fork()</td>
</tr>
<tr>
<td>_gs_rdy()</td>
<td>_os_gs_ready()</td>
</tr>
<tr>
<td>_gs_size()</td>
<td>_os_gs_size()</td>
</tr>
<tr>
<td>os9exec()</td>
<td>_os_exec()</td>
</tr>
<tr>
<td>hypot()</td>
<td>None</td>
</tr>
<tr>
<td>pause()</td>
<td>_os_prsnam()</td>
</tr>
<tr>
<td>ibrk()</td>
<td>malloc()</td>
</tr>
<tr>
<td>pause()</td>
<td>sleep()</td>
</tr>
<tr>
<td>index()</td>
<td>strchr()</td>
</tr>
</tbody>
</table>
### Table 1-37. sys_clib.l Function Equivalents (Continued)

<table>
<thead>
<tr>
<th>sys_clib.l Function</th>
<th>Corresponding Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>pffinit()</td>
<td>none</td>
</tr>
<tr>
<td>intercept()</td>
<td>_os_intercept()</td>
</tr>
<tr>
<td>pflimits()</td>
<td>none</td>
</tr>
<tr>
<td>julian()</td>
<td>_os_julian()</td>
</tr>
<tr>
<td>prerr()</td>
<td>strerror()</td>
</tr>
<tr>
<td>kill()</td>
<td>_os_send()</td>
</tr>
<tr>
<td>prognam()</td>
<td>argv[0]</td>
</tr>
<tr>
<td>ls()</td>
<td>_os_seek()</td>
</tr>
<tr>
<td>prsname()</td>
<td>_os_prsname()</td>
</tr>
<tr>
<td>putw()</td>
<td>putc() putc()</td>
</tr>
<tr>
<td>putw() is replaced by performing two calls to putc()</td>
<td></td>
</tr>
<tr>
<td>ss_enrts()</td>
<td>_os_ss_enrts()</td>
</tr>
<tr>
<td>read()</td>
<td>_os_read()</td>
</tr>
<tr>
<td>ss_lock()</td>
<td>_os_ss_lock()</td>
</tr>
<tr>
<td>readdir()</td>
<td>_os_read()</td>
</tr>
<tr>
<td>ss_opt()</td>
<td>_os_ss_popt()</td>
</tr>
<tr>
<td>readln()</td>
<td>_os_readln()</td>
</tr>
<tr>
<td>ss_pfd()</td>
<td>_os_ss_fd()</td>
</tr>
<tr>
<td>rewinddir()</td>
<td>_os_seek()</td>
</tr>
<tr>
<td>ss_rela()</td>
<td>_os_ss_relea()</td>
</tr>
<tr>
<td>rindex()</td>
<td>strchr()</td>
</tr>
<tr>
<td>ss_reset()</td>
<td>_os_ss_reset()</td>
</tr>
<tr>
<td>sbrk()</td>
<td>malloc()</td>
</tr>
<tr>
<td>ss_size()</td>
<td>_os_ss_size()</td>
</tr>
<tr>
<td>seekdir()</td>
<td>_os_seek()</td>
</tr>
<tr>
<td>ss_ssig()</td>
<td>_os_ss_sendsig()</td>
</tr>
<tr>
<td>setcrc()</td>
<td>_os_setcrc()</td>
</tr>
<tr>
<td>ss_tiks()</td>
<td>_os_ss_ticks()</td>
</tr>
<tr>
<td>settime()</td>
<td>_os_setime() / _os9_setime()</td>
</tr>
<tr>
<td>ss_wtrak()</td>
<td>_os_ss_wtrack()</td>
</tr>
<tr>
<td>setpr()</td>
<td>_os_setpr()</td>
</tr>
<tr>
<td>strass()</td>
<td>memcpy() on structure assignment</td>
</tr>
<tr>
<td>setstat()</td>
<td>_os_setstat()</td>
</tr>
<tr>
<td>strncpy()</td>
<td>none</td>
</tr>
<tr>
<td>ssetsys()</td>
<td>_os_ssetsys()</td>
</tr>
<tr>
<td>sys_mktime()</td>
<td>mktime() for user programs</td>
</tr>
<tr>
<td>setuid()</td>
<td>_os_setuid()</td>
</tr>
<tr>
<td>sysdate()</td>
<td>_os_setime() / _os9_setime()</td>
</tr>
<tr>
<td>sigmask()</td>
<td>_os_sigmask()</td>
</tr>
<tr>
<td>syslog()</td>
<td>_os_sysdbg()</td>
</tr>
<tr>
<td>sleep()</td>
<td>_os_sleep() / _os9_sleep()</td>
</tr>
<tr>
<td>telldir()</td>
<td>_os_gs_pos()</td>
</tr>
<tr>
<td>srqcmem()</td>
<td>_os_srqmem()</td>
</tr>
<tr>
<td>tsleep()</td>
<td>_os_sleep()</td>
</tr>
</tbody>
</table>
The Function Description

Each function description includes a minimum of the following sections:

- Syntax
- OS-9 Attributes
- Description
- Library

In addition, descriptions may also contain sections for the following:

- Examples
- Errors
- References to other calls (See Also)

Syntax

The syntax shows how the function and parameters look if written as a C function definition, even if the actual function is a macro or is written in assembly language.

For example, the syntax for `fopen()` appears as follows:

```c
#include <stdio.h>

FILE *fopen(const char *name, const char *action);
```
This indicates that `fopen()` requires the `<stdio.h>` header file, returns a pointer to a structure of type `FILE`, and requires two parameters, both pointers to constant character strings. The parameter names are suggestions only; you can use any name.

**OS-9 Attributes**

The OS-9 Attributes section lists various attributes of each function in relation to OS-9—including whether the function is compatible with OS-9 and/or OS-9 for 68K; whether the function is in user state and/or system state; whether the function is safe for use in a threaded application; and whether the function is re-entrant. Note that functions which are not re-entrant should not be used in signal handlers unless signals are masked before the functions are used and unmasked after the functions return.

**Library**

The library field indicates the libraries that contain the function.

**Errors**

When an error occurs, C functions typically return an error code in the global variable `errno`. You must include the file `<errno.h>` in C programs to declare `errno`.

**References to Other Calls**

Many functions also have a “See Also” section. Functions listed in the “See Also” section are related functions or are called by the function being described.

**C Shared Library**

The following two CSL (C Shared Library) files are available when configuring OS-9 systems: `cs1` and `mt_cs1`.

Both files contain a module named `cs1`. Use the file `cs1` for systems that execute no threaded `cs1`-using applications. Use the file `mt_cs1` for systems that execute both threaded and non-threaded `cs1`-using applications.

Table 1-38 lists all functions available in `cs1.l`. Table 1-39 lists all functions available in `mt_cs1.l`.

**Table 1-38. CSL Function List**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>access()</code></td>
<td>Determine file accessibility.</td>
</tr>
<tr>
<td><code>asctime()</code></td>
<td>Convert broken-down time to string format.</td>
</tr>
<tr>
<td><code>atexit()</code></td>
<td>Specify function to call at normal program termination.</td>
</tr>
<tr>
<td><code>atof()</code></td>
<td>Alpha to floating conversion.</td>
</tr>
<tr>
<td><code>atoi()</code></td>
<td>Alpha to integer conversion.</td>
</tr>
<tr>
<td><code>atol()</code></td>
<td>Alpha to long conversion.</td>
</tr>
<tr>
<td><code>_atof()</code></td>
<td>Alpha to numeric translation.</td>
</tr>
<tr>
<td><code>_atou()</code></td>
<td>Alpha to unsigned conversion.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>calloc()</td>
<td>Allocate storage for array.</td>
</tr>
<tr>
<td>chmod()</td>
<td>Change file access permissions.</td>
</tr>
<tr>
<td>chown()</td>
<td>Change owner of file.</td>
</tr>
<tr>
<td>clearerr()</td>
<td>Clear error condition.</td>
</tr>
<tr>
<td>close()</td>
<td>Close path.</td>
</tr>
<tr>
<td>create()</td>
<td>Create file.</td>
</tr>
<tr>
<td>_dtoa()</td>
<td>Double to ASCII conversion.</td>
</tr>
<tr>
<td>_errmsg()</td>
<td>Print error message.</td>
</tr>
<tr>
<td>_exit()</td>
<td>Task termination.</td>
</tr>
<tr>
<td>fclose()</td>
<td>Close file.</td>
</tr>
<tr>
<td>fdopen()</td>
<td>Attach path to file pointer.</td>
</tr>
<tr>
<td>feof()</td>
<td>Check buffered file for end of file.</td>
</tr>
<tr>
<td>ferror()</td>
<td>Check buffered file for error condition.</td>
</tr>
<tr>
<td>fflush()</td>
<td>Flush file’s buffer.</td>
</tr>
<tr>
<td>fgetc()</td>
<td>Get character from file.</td>
</tr>
<tr>
<td>fgets()</td>
<td>Get string from file.</td>
</tr>
<tr>
<td>fgetpos()</td>
<td>Get current position in file.</td>
</tr>
<tr>
<td>fputs()</td>
<td>Get string from file.</td>
</tr>
<tr>
<td>fopen()</td>
<td>Open file.</td>
</tr>
<tr>
<td>fprintf()</td>
<td>Formatted output.</td>
</tr>
<tr>
<td>fputc()</td>
<td>Output character to file.</td>
</tr>
<tr>
<td>fputs()</td>
<td>Output string to file.</td>
</tr>
<tr>
<td>fread()</td>
<td>Read data from file.</td>
</tr>
<tr>
<td>free()</td>
<td>Return memory.</td>
</tr>
<tr>
<td>_freemin()</td>
<td>Set memory reclamation bound.</td>
</tr>
<tr>
<td>freopen()</td>
<td>Re-open file.</td>
</tr>
<tr>
<td>fscanf()</td>
<td>Input string conversion.</td>
</tr>
<tr>
<td>fseek()</td>
<td>Reposition file pointer.</td>
</tr>
<tr>
<td>fsetpos()</td>
<td>Set current file position.</td>
</tr>
<tr>
<td>ftell()</td>
<td>Report file pointer position.</td>
</tr>
<tr>
<td>fwrite()</td>
<td>Write data to file.</td>
</tr>
<tr>
<td>getc()</td>
<td>Get next character from file, stdin.</td>
</tr>
<tr>
<td>getenv()</td>
<td>Get value for environment name.</td>
</tr>
<tr>
<td>getpid()</td>
<td>Return group ID.</td>
</tr>
<tr>
<td>gets()</td>
<td>Get string from file.</td>
</tr>
<tr>
<td>_getsyss()</td>
<td>Get system global variables.</td>
</tr>
<tr>
<td>getuid()</td>
<td>Determine user ID number.</td>
</tr>
<tr>
<td>gmtime()</td>
<td>Convert calendar time to Greenwich Mean Time.</td>
</tr>
<tr>
<td>kill()</td>
<td>Send Signal to Process</td>
</tr>
<tr>
<td>_lcalloc()</td>
<td>Allocate storage for array (low-overhead).</td>
</tr>
</tbody>
</table>
### Table 1-38. CSL Function List (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_lfree()</td>
<td>Return memory (low overhead).</td>
</tr>
<tr>
<td>_lmalloc()</td>
<td>Allocate memory from arena (low-overhead).</td>
</tr>
<tr>
<td>localeconv()</td>
<td>Numeric formatting convention inquiry.</td>
</tr>
<tr>
<td>localtime()</td>
<td>Convert Calendar Time to Local Time.</td>
</tr>
<tr>
<td>_lrealloc()</td>
<td>Resize block of memory (low-overhead).</td>
</tr>
<tr>
<td>lseek()</td>
<td>Position file pointer.</td>
</tr>
<tr>
<td>malloc()</td>
<td>Allocate memory from arena.</td>
</tr>
<tr>
<td>_mallocmin()</td>
<td>Set minimum allocation size.</td>
</tr>
<tr>
<td>mbstowcs()</td>
<td>Convert sequence of multibyte characters.</td>
</tr>
<tr>
<td>mbtowc()</td>
<td>Determine number of bytes in multibyte characters.</td>
</tr>
<tr>
<td>memcpy()</td>
<td>Copy memory.</td>
</tr>
<tr>
<td>memmove()</td>
<td>Move memory.</td>
</tr>
<tr>
<td>memset()</td>
<td>File memory.</td>
</tr>
<tr>
<td>mktime()</td>
<td>Convert Broken-Down Time to Calendar Time.</td>
</tr>
<tr>
<td>open()</td>
<td>Open file.</td>
</tr>
<tr>
<td>opendir()</td>
<td>Open directory.</td>
</tr>
<tr>
<td>_os_loadp()</td>
<td>Load and link to memory module using PATH.</td>
</tr>
<tr>
<td>_parsepath()</td>
<td>Parse disk file (RBF) pathlist.</td>
</tr>
<tr>
<td>perror()</td>
<td>Map error number.</td>
</tr>
<tr>
<td>pow()</td>
<td>Power function.</td>
</tr>
<tr>
<td>printf()</td>
<td>Formatted output.</td>
</tr>
<tr>
<td>putc(), putchar()</td>
<td>Put next character to file, standard out.</td>
</tr>
<tr>
<td>puts()</td>
<td>Output string to file.</td>
</tr>
<tr>
<td>read()</td>
<td>Read bytes from path.</td>
</tr>
<tr>
<td>readdir()</td>
<td>Return pointer.</td>
</tr>
<tr>
<td>readln()</td>
<td>Read bytes from path.</td>
</tr>
<tr>
<td>realloc()</td>
<td>Resize block of memory.</td>
</tr>
<tr>
<td>remove()</td>
<td>Remove file.</td>
</tr>
<tr>
<td>rename()</td>
<td>Rename file.</td>
</tr>
<tr>
<td>rewind()</td>
<td>Return file pointer to zero.</td>
</tr>
<tr>
<td>rewinddir()</td>
<td>Reset position of directory stream.</td>
</tr>
<tr>
<td>scanf()</td>
<td>Input strings conversion.</td>
</tr>
<tr>
<td>seekdir()</td>
<td>Set position of next readdir.</td>
</tr>
<tr>
<td>setlocale()</td>
<td>Locale control.</td>
</tr>
<tr>
<td>_setsys()</td>
<td>Set/Examine system global variables.</td>
</tr>
<tr>
<td>setvbuf()</td>
<td>Set up buffer for I/O stream</td>
</tr>
<tr>
<td>sprintf()</td>
<td>Formatted output.</td>
</tr>
<tr>
<td>sscanf()</td>
<td>Input strings conversion.</td>
</tr>
<tr>
<td>strcat()</td>
<td>String catenation.</td>
</tr>
</tbody>
</table>
Table 1-38. CSL Function List (Continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>strchr()</td>
<td>Locale string.</td>
</tr>
<tr>
<td>strcmp()</td>
<td>String comparison.</td>
</tr>
<tr>
<td>strcpy()</td>
<td>String copy.</td>
</tr>
<tr>
<td>strerror()</td>
<td>Map error message string.</td>
</tr>
<tr>
<td>strftime()</td>
<td>Place formatted time in buffer.</td>
</tr>
<tr>
<td>strlen()</td>
<td>Determine string length.</td>
</tr>
<tr>
<td>strncat()</td>
<td>String catenation.</td>
</tr>
<tr>
<td>strncmp()</td>
<td>String comparison.</td>
</tr>
<tr>
<td>strncpy()</td>
<td>String copy.</td>
</tr>
<tr>
<td>strrchr()</td>
<td>Locate last occurrence of string.</td>
</tr>
<tr>
<td>strtol()</td>
<td>String to double conversion.</td>
</tr>
<tr>
<td>strtok()</td>
<td>Break string into tokens.</td>
</tr>
<tr>
<td>strtoll()</td>
<td>String to long conversion.</td>
</tr>
<tr>
<td>strtoul()</td>
<td>String to long conversion.</td>
</tr>
<tr>
<td>system()</td>
<td>Shell command execution.</td>
</tr>
<tr>
<td>telldir()</td>
<td>Return current location.</td>
</tr>
<tr>
<td>time()</td>
<td>Get Calendar Time.</td>
</tr>
<tr>
<td>tmpfile()</td>
<td>Create temporary binary file.</td>
</tr>
<tr>
<td>tmpnam()</td>
<td>Generate unique valid filename.</td>
</tr>
<tr>
<td>ungetc()</td>
<td>Unget character.</td>
</tr>
<tr>
<td>vfprintf()</td>
<td>Print to file.</td>
</tr>
<tr>
<td>vprintf()</td>
<td>Print to standard output.</td>
</tr>
<tr>
<td>vsprintf()</td>
<td>Print to string.</td>
</tr>
<tr>
<td>wcstombs()</td>
<td>Convert sequence of wide chars to multibyte chars</td>
</tr>
<tr>
<td>wctomb()</td>
<td>Convert wide character to multibyte character.</td>
</tr>
<tr>
<td>write()</td>
<td>Write bytes to path.</td>
</tr>
<tr>
<td>writeln()</td>
<td>Write bytes to path.</td>
</tr>
</tbody>
</table>

The following table lists the MT_CSL functions.

Table 1-39. MT_CSL Function List

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pthread_attr_destroy()</td>
<td>Free thread attribute object.</td>
</tr>
<tr>
<td>pthread_attr_getdetachstate()</td>
<td>Get detach state attribute.</td>
</tr>
<tr>
<td>_pthread_attr_getinitfunction()</td>
<td>Get initialization function attribute.</td>
</tr>
<tr>
<td>_pthread_attr_getpriority()</td>
<td>Get priority attribute.</td>
</tr>
<tr>
<td>pthread_attr_getstackaddr()</td>
<td>Get stack address attribute.</td>
</tr>
<tr>
<td>pthread_attr_getstacksize()</td>
<td>Get stack size attribute.</td>
</tr>
<tr>
<td>pthread_attr_init()</td>
<td>Allocate thread creation attribute object.</td>
</tr>
<tr>
<td>pthread_attr_setdetachstate()</td>
<td>Set detached state attribute.</td>
</tr>
<tr>
<td>_pthread_attr_setinitfunction()</td>
<td>Set initialization function attribute.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>_pthread_attr_setpriority()</td>
<td>Set priority attribute.</td>
</tr>
<tr>
<td>pthread_attr_setstackaddr()</td>
<td>Set stack address attribute.</td>
</tr>
<tr>
<td>pthread_attr_setstacksize()</td>
<td>Set stack size attribute.</td>
</tr>
<tr>
<td>pthread_cancel()</td>
<td>Cancel target thread.</td>
</tr>
<tr>
<td>pthread_cleanup_pop()</td>
<td>Pop cleanup routine.</td>
</tr>
<tr>
<td>pthread_cleanup_push()</td>
<td>Push cleanup routine.</td>
</tr>
<tr>
<td>pthread_cond_broadcast()</td>
<td>Release threads waiting for condition variable.</td>
</tr>
<tr>
<td>pthread_cond_init()</td>
<td>Allocate condition variable object.</td>
</tr>
<tr>
<td>pthread_cond_signal()</td>
<td>Release thread waiting for condition variable.</td>
</tr>
<tr>
<td>pthread_cond_timedwait()</td>
<td>Wait on condition variable for specified interval.</td>
</tr>
<tr>
<td>pthread_cond_wait()</td>
<td>Wait on condition variable.</td>
</tr>
<tr>
<td>pthread_create()</td>
<td>Create new thread.</td>
</tr>
<tr>
<td>pthread_detach()</td>
<td>Orphan target thread.</td>
</tr>
<tr>
<td>pthread_exit()</td>
<td>Terminate thread.</td>
</tr>
<tr>
<td>pthread_getspecific()</td>
<td>Get thread-specific data pointer.</td>
</tr>
<tr>
<td>_pthread_getstatus()</td>
<td>Get thread status information.</td>
</tr>
<tr>
<td>_pthread_interrupt()</td>
<td>Interrupt target thread.</td>
</tr>
<tr>
<td>_pthread_interrupt_clear()</td>
<td>Clear interrupt request for target thread.</td>
</tr>
<tr>
<td>pthread_join()</td>
<td>Wait for target thread to terminate.</td>
</tr>
<tr>
<td>pthread_key_create()</td>
<td>Create thread-specific data key.</td>
</tr>
<tr>
<td>pthread_key_delete()</td>
<td>Delete thread-specific data key.</td>
</tr>
<tr>
<td>pthread_mutex_destroy()</td>
<td>Free mutex object.</td>
</tr>
<tr>
<td>pthread_mutex_getprioceiling()</td>
<td>Get mutex priority ceiling.</td>
</tr>
<tr>
<td>pthread_mutex_init()</td>
<td>Allocate mutex object.</td>
</tr>
<tr>
<td>pthread_mutex_lock()</td>
<td>Lock mutex object.</td>
</tr>
<tr>
<td>pthread_mutex_setprioceiling()</td>
<td>Set mutex priority ceiling.</td>
</tr>
<tr>
<td>pthread_mutex_trylock()</td>
<td>Lock mutex object (non-blocking)</td>
</tr>
<tr>
<td>pthread_mutex_unlock()</td>
<td>Unlock mutex object.</td>
</tr>
<tr>
<td>pthread_once()</td>
<td>Execute routine once per process.</td>
</tr>
<tr>
<td>_pthread_resume()</td>
<td>Decrement suspension counter.</td>
</tr>
<tr>
<td>pthread_setcancelstate()</td>
<td>Set cancel state.</td>
</tr>
<tr>
<td>pthread_setcanceltype()</td>
<td>Set cancel type.</td>
</tr>
<tr>
<td>_pthread_setpr()</td>
<td>Set priority for target thread.</td>
</tr>
<tr>
<td>_pthread_setsignalrange()</td>
<td>Set range of signal values.</td>
</tr>
<tr>
<td>pthread_setspecific()</td>
<td>Set thread-specific data pointer.</td>
</tr>
<tr>
<td>_pthread_setsuspendable()</td>
<td>Decrement suspendability counter.</td>
</tr>
<tr>
<td>_pthread_setunsuspendable()</td>
<td>Increment suspendability counter.</td>
</tr>
<tr>
<td>_pthread_suspend()</td>
<td>Increment suspension counter.</td>
</tr>
</tbody>
</table>
This chapter includes library function definitions in alphabetical order according to some special rules.

- Special characters (not letters, numbers, or underscores) are listed first.
- Function calls are listed in alphabetic order next without regard for numbers and underscores.
- If two function calls are identical using these rules, then they are alphabetized according to the following order:
  1. Symbols
  2. Underscores
  3. Alphabetic characters
  4. Numbers
Syntax

```c
#include <stdlib.h>

void abort(void);
```

Description

`abort()` abnormally terminates a program by raising the signal `SIGABRT`. However, if the signal `SIGABRT` is being caught and the signal handler does not return, the program is not terminated. If `SIGABRT` returns, the program is terminated abnormally. This flushes and closes the open streams and removes temporary files. The process returns the status `EXIT_FAILURE`.

⚠️ `abort()` does not return to the caller.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.l
Syntax
#include <stdlib.h>
int abs(int value);

Description
abs() returns the absolute value of its integer parameter.

value
is the integer parameter. (Input)

abs(0x80000000)
returns 0x80000000 as the result.

Applying abs() to the most negative integer yields a result that is the most negative integer.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l
access()

Determine File Accessibility

Syntax

```c
#include <modes.h>

int access(
    const char *name,
    int perm);
```

Description

access() returns zero if the mode(s) specified in the file permissions are correct for the user to access the specified file.

- **name** is a pointer to the name of the file. (Input)
- **perm** may be any legal mode as defined in the `modes.h` header file. (Input)

Use a mode value of zero to verify that a file exists. Only the lower 16 bits of *perm* are used. If the file cannot be accessed, -1 is returned and the appropriate error code is placed in *errno*.

This value may not be compatible with other systems.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`sys_clib.l`
acos()

Arc Cosine Function

Syntax

```c
#include <math.h>
double acos(double x);
```

Description

`acos()` returns the arc cosine of `x` (input), in the range of `[0, π]` radians. A domain error, `EDOM` stored in `errno`, occurs for parameters not in the range `[-1, +1]`. On error, `acos()` returns `HUGE_VAL`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`clib.l`

Possible Error

`EDOM`
alarm()

Sends signal SIGALRM

Syntax
#include <UNIX/os9def.h>
unsigned int alarm(unsigned int seconds);

Description
alarm() requests that a signal be sent to the requesting process after a specified number of seconds. Unless caught or ignored, the signal terminates the process. alarm() sends signal SIGALRM when the alarm expires.

alarm() requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm() request is canceled.

alarm() returns -1 and sets the global value errno if an error occurs.

seconds
is a specified amount of time after which a signal is sent. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
unix.l

See Also
_os9_alarm_atdate()
_os_alarm_set()
alloca()
Allocates Bytes of Space in Stack Frame

Syntax
#include <UNIX/os9def.h>
void *alloca(unsigned int size);

Description
alloca() allocates size number of zero bytes from the malloc() pool. This space is not automatically freed.
alloca() returns a pointer to the allocated bytes. If an error occurs, NULL is returned and the global variable errno is set.

size
is the number of zero bytes from the malloc() pool. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
unix.1
Syntax

```
#include <alarm.h>
int alm_atdate(
    int sigcode,
    int time,
    int date);
```

Description

`alm_atdate()` requests that a signal be sent to the requesting process at a specific Gregorian time and date. The time and date must be in the following format:

**Figure 2-1. alm_atdate() Time and Date Format**

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Hour (0-23)</td>
<td>Minute</td>
<td>Second</td>
</tr>
<tr>
<td>Year (two bytes)</td>
<td>Month</td>
<td>Day</td>
<td></td>
</tr>
</tbody>
</table>

Because the system time and date may be changed, the alarm signal is sent when the time and date become greater than or equal to the alarm time. If errors occur, `-1` is returned and the appropriate error code is placed in the global variable `errno`. If no errors occur, `alm_atdate()` returns the alarm ID.

Refer to the OS-9 Technical Manual for information about alarms.

- `sigcode`: specifies the signal to be sent to the caller. (Input)
- `time`: specifies the time to send the signal. (Input)
- `date`: specifies the Gregorian date for the signal to be sent. (Input)
**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User  
Threads: Safe  
Re-entrant: No

**Library**

sys_clib.l

**See Also**

_os9_alarm_atdate()  
_os_alarm_set()
Set Alarm at Julian Date/Time

Syntax

```c
#include <alarm.h>

int alm_atjul(
    int sigcode,  // specifies the signal to be sent to the caller. (Input)
    int time,    // should contain the time at which to send the signal, expressed as the number of seconds after midnight. (Input)
    int date);   // must contain the Julian date on which to send the signal. (Input)
```

Description

`alm_atjul()` requests that a signal be sent to the requesting process at a specific Julian date and time.

Because the system time and date may be changed, the alarm signal is sent when the system time and date become greater than or equal to the alarm time. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. If errors do not occur, `alm_atjul()` returns the alarm ID.

Refer to your operating system technical manual for information about alarms.

- `sigcode` specifies the signal to be sent to the caller. (Input)
- `time` should contain the time at which to send the signal, expressed as the number of seconds after midnight. (Input)
- `date` must contain the Julian date on which to send the signal. (Input)

The Julian date is the number of days since 1 January 4713 B.C., the beginning of the Julian period. A Julian day begins at midnight on OS-9 systems. Standard Julian days begin twelve hours earlier, at noon. For example, 1:00 a.m. January 2, 4713 B.C. is one hour after the beginning of Julian Day 1 for OS-9, but thirteen hours after the beginning in standard Julian time.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
sys_clib.1

See Also
=os9_alarm_atjul()
=os_alarm_set()
Syntax

```c
#include <alarm.h>

int alm_cycle(
    int sigcode,
    int time_interval);
```

Description

`alm_cycle()` sends a signal to the requesting process after the specified time has elapsed and then resets the alarm to provide a recurring periodic signal. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. If errors do not occur, `alm_cycle()` returns the alarm ID.

- **sigcode**
  - specifies the signal to be sent to the caller. (Input)

- **time_interval**
  - specifies the periodic interval at which the signal is to be sent. (Input)

  For example, if the request is made at time `X` and `time_interval` is `t`, the signal is sent to the requesting process at times `(X + t)`, `(X + 2t)`, `(X + 3t)`, and so forth until the alarm is deleted.

  If the most significant bit of `time_interval` is set, `time_interval` is assumed to be in 256ths of a second. Otherwise, `time_interval` is assumed to be in units of system clock ticks (refer to `CLOCKS_PER_SEC` in the `time.h` header file). The minimum `time_interval` allowed is two system clock ticks.

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
- **Re-entrant:** Yes

Library

`sys_clib.l`
See Also

alm_delete()
alm_set()
alm_cycle()
alm_delete()
_os_alarm_set()
Syntax

#include <alarm.h>

int alm_delete(int alarmid);

Description

alm_delete() cancels the specified alarm request.
If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.
If errors do not occur, alm_delete() returns zero.

Refer to the OS-9 Technical Manual for information about alarms.

alarmid is the alarm ID of alarm to cancel. (Input)
If alarmid is zero, all pending alarm requests are cancelled.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

_os_alarm_delete()
**Syntax**

```c
#include <alarm.h>

int alm_set(
    int sigcode,
    int time);
```

**Description**

`alm_set()` requests that a signal be sent to the requesting process after the specified time has elapsed. For example, if the request is made at time $X$ and `time` is $t$, the signal is sent at time $(X + t)$, unless the alarm request is cancelled.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

If errors do not occur, `alm_set()` returns the alarm ID.

- **sigcode**
  - is the signal code to be sent to the requesting process. (Input)

- **time**
  - specifies the time interval. (Input)
    - If the most significant bit of `time` is set, `time` is assumed to be in 256ths of a second. Otherwise, `time` is assumed to be in units of system clock ticks (refer to `CLOCKS_PER_SEC` in the `time.h` header file). The minimum `time` allowed is two system clock ticks.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

sys_clib.l

**See Also**

`_os_alarm_set()`
asctime()

Convert Broken-Down Time to String Format

Syntax

#include <time.h>
char *asctime(struct tm *tp);

Description

asctime() converts the broken-down time structure into the following 26-byte string format, including the terminating \0:

xxx mmm dd hh:mm:ss yyyy\n\0

asctime() returns a pointer to a static area which may be overwritten. To insure data integrity, use the string or save it immediately.

xxx

is one of the following days of the week:

SunMonTueWedThuFriSat

mmm

is one of the following months of the year:

JanFebMarAprMayJun
JulAugSepOctNovDec

dd

specifies the day of the month.

hh:mm:ss

specifies the time in hours, minutes, and seconds.

yyyy

specifies the year.

tp

is a pointer to the broken-down time structure. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.1
See Also

ctime()
localtime()
gmtime()
sys_mktime()
strftime()
### Syntax

```c
#include <math.h>
double asin(double x);
```

### Description

The `asin()` function returns the arc sine of `x` (input), in the range \([-\pi/2, +\pi/2]\) radians. A domain error, `EDOM` stored in `errno`, occurs for parameters not in the range \([-1, +1]\). On error, `asin()` returns `HUGE_VAL`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

### Library

`clib.l`

### Possible Error

`EDOM`
assert()

Place Diagnostics in Programs

Syntax
#include <assert.h>
void assert(int expression);

Description
assert() is a macro that places diagnostics in programs. If expression is false, assert() writes a message about the failed call to standard error. The following message is printed:
Assertion failed: expression, file filename,
    line line#

After printing the message, assert() calls abort(). This raises the SIGABRT signal that may be handled by the application.

assert() never returns a value.

If NDEBUG is defined as a macro name when the assert.h header file is included, assert() is defined as follows:
#define assert(ignore) ((void)0)

expression
    is the text of the parameter. (Input)

filename
    is the name of the file containing the assert macro.

line#
    is the line number of the file containing the assert macro.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
assert() is a macro defined in assert.h.

See Also
abort() raise()
signal()
**Syntax**

#include <math.h>

double atan(double x);

**Description**

`atan()` returns the arc tangent of `x` (input), in the range `[-\pi/2, +\pi/2]` radians.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

clib.1
atan2()
Arc Tangent Function

Syntax
#include <math.h>
double atan2(
    double y,
    double x);

Description
atan2() returns the arc tangent of y (input) divided by x (input)—that is, atan(y/x),
in the range [-π, π] radians. atan2() uses the signs of both y and x to determine the
quadrant of the returned value. A domain error, EDOM placed in errno, occurs if both
y and x are zero. On error, atan2() returns HUGE_VAL.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

Possible Error
EDOM
atexit()

Specify Function to Call at Normal Program Termination

Syntax

```c
#include <stdlib.h>
int atexit(void (*func)(void));
```

Description

atexit() specifies a function to be called without parameters at normal program termination. Thirty-two functions may be specified.

atexit() returns zero if it succeeds. Otherwise, it returns a non-zero value.

func

is a pointer to the function. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

exit()
**Syntax**

```c
#include <stdlib.h>

double atof(const char *string);
```

**Description**

`atof()` converts a string into its equivalent representation in type `double`. Except that no error indication is given, `atof()` is equivalent to the following:

```c
strtol(string, (char **)NULL)
```

`string` is a pointer to the string. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

`clib.l`

**See Also**

`strtol()`
**Syntax**

```c
#include <stdlib.h>
int atoi(const char *string);
```

**Description**

`atoi()` converts a string into its equivalent representation in type `int`.

Except that no error indication is given, `atoi()` is equivalent to the following:

```c
(int) strtol(nptr,(char**)NULL,10)
```

*string* is a pointer to the string. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

`clib.l`

**See Also**

- `strtol()`
- `strtoll()`
**Syntax**

```c
#include <stdlib.h>
long atol(const char *string);
```

**Description**

`atol()` converts a string into its equivalent representation in type `long`.

Except that no error indication is given, `atol()` is equivalent to the following:

```c
strtol(nptr,(char**)NULL,10)
```

`string` is a pointer to the string. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

`clib.l`

**See Also**

`strtoll()`
### Syntax

```c
#include <aton.h>
_number_type _aton(
    char **buf,
    _number *value);
```

### Description

 `_aton()` converts a string into its associated numeric value, if possible.

#### Table 2-1. `_aton()` Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ERROR</td>
<td>No recognizable number is found, or the value cannot be represented (it would cause overflow or underflow).</td>
</tr>
<tr>
<td>_INTEGER</td>
<td>The recognized number is an integer.</td>
</tr>
<tr>
<td>_REAL</td>
<td>The recognized number is not an integer.</td>
</tr>
</tbody>
</table>

`_aton()` updates `buf` to point to the character just after the recognized string. It updates `value` to contain the recognized number, unless an error occurs.

- **buf** is a pointer to a pointer to a string containing a printable representation of a number expressed in base ten. (Input/Output)
  
The number may be an integer or a real number. You may also express the number in the following exponential format:

  ```
  <number>(E|e)(-|+)<exponent>
  ```

- **value** is a pointer to one of the three possible values that can be returned. (Output)

### Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

### Library

`clib.l`
**Syntax**

```c
#include <stdlib.h>
unsigned _atou(const char *string);
```

**Description**

_atou() converts a string into its appropriate unsigned numeric value, if possible. _atou() treats long and int values identically.

_string_ is a pointer to a string containing a printable representation of a number expressed in the format below. (Input)

`[+][-]<digits>`

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

clib.l
attach()

Attach to Device

Syntax

```c
#include <modes.h>
#include <io.h>
dev_list *attach(
    const char *name,
    int mode);
```

Description

`attach()` causes a new device to become known to the system or verifies that the device is already attached.

If the device descriptor is found and the device is not already attached, `attach()` links to its file manager and device driver, and places their addresses in a new device table entry. If the attach fails, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`name`

is a pointer to the name of the device descriptor. (Input)

`mode`

is the access mode. (Input)

Possible access modes include `FAM_READ`, `FAM_WRITE`, `S_IREAD`, and `S_IWRITE`.

`mode` may be used to verify that subsequent read and/or write operations are permitted. Only the lower 16 bits of `mode` are used.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`sys_clib.l`

See Also

`_os_attach()`

I$Attach OS-9 for 68K Technical Manual
I.AttachOS-9 Technical Manual
bcmp()

Compares Bytes from Two Memory Areas

Syntax

#include <UNIX/os9def.h>

int bcmp(
    void *addr1,
    void *addr2,
    unsigned int len);

Description

bcmp() compares memory at the first address (addr1) against the memory at the second address (addr2). The function returns zero if the memory areas are identical, non-zero otherwise. Both memory areas are assumed to be the given length (len). If len is zero, bcmp() returns zero.

addr1
    is the first memory address. (Input)

addr2
    is the second memory address. (Input)

len
    is the length of both addresses. (Input)

Attributes

Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library

unix.l

See Also

bcopy()
bzero()
ffs()
bcopy()

Copies Bytes from One Memory Area to Another Memory Area

Syntax
#include <UNIX/os9def.h>
void bcopy(
    void *addr1,
    void *addr2,
    unsigned int len);

Description
bcopy() copies bytes, of length len, from one memory area (addr1) to another (addr2). Overlapping memory areas are handled correctly.

addr1
    is the first memory area. (Input)

addr2
    is the second memory area. (Output)

len
    is the length of addr1 and addr2. (Input)

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes
Standards:  ANSI

Library
unix.l

See Also
bcmp()
bzero()
ffs()
**Syntax**

```c
#include <stdlib.h>
void *bsearch(
    const void *key,
    const void *base,
    size_t nmemb,
    size_t size,
    int (*compar) (const void *, const void *)
);
```

**Description**

*bsearch()* searches an array of *nmemb* objects for an element that matches *key*.

The array consists of all elements that compare less than, equal to, and greater than the *key* object, in that order.

*bsearch()* returns a pointer to a matching array element. If no match is found, it returns a null pointer. If two elements compare as equal, which element is matched is unspecified.

- **key** is a pointer to the key object. (Input)

- **base** is a pointer to the initial element of the array. (Input)

- **nmemb** specifies the number of array elements. (Input)

- **size** specifies the size of each array element. (Input)

- **compar** is a pointer to a comparison function. (Input/Output)

  **compar** is called with two parameters that point to the key object and to an array element, in that order. **compar** returns an integer that is:

  - Negative if *key* is considered less than the array element.
  - Zero if *key* is equal to the array element.
  - Positive if *key* is considered greater than the array element.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l
buildpath()

Adds a Filename to the Root Directory Variable

Syntax

```c
#include <UNIX/os9def.h>
char *buildpath(const char *filename);
```

Description

buildpath() adds filename (input) to the value of the environment variable XOS9ROOTDIR. If XOS9ROOTDIR is not set, a default value of /h0 is used.

buildpath() returns the final string in a static buffer that should be saved if additional calls are made to buildpath(). For example:

- **with** XOS9ROOTDIR set to /dd/ETC, buildpath(xfile); returns the following string:
  
  /dd/ETC/xfile

- **without** XOS9ROOTDIR set to /dd/ETC, buildpath(xfile); returns the following string:
  
  /h0/xfile

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

unix.l
**bzero()**

Clear Bytes In Memory

**Syntax**

```c
#include <UNIX/os9def.h>
void bzero(
    void *addr,
    unsigned int len);
```

**Description**

*bzero()* clears bytes in memory starting from an address (*addr*) for a number of bytes (*len*).

*addr*  
  is a memory address. (Input/Output)

*len*  
  is the number of bytes. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

**Library**

`unix.l`

**See Also**

- `bcopy()`
- `bcmp()`
- `ffs()`
calloc()  
Allocate Storage for Array

Syntax
#include <stdlib.h>
void *calloc(
    size_t nmemb,
    size_t size);

Description
calloc() allocates space for an array. The allocated memory is cleared to zeroes.
If the allocation is successful, calloc() returns a pointer to the area. If the allocation
fails or if size is zero, NULL is returned. The space designated by the returned
pointer is suitably aligned to hold an object of any type.

nmemb
    is the number of elements in the array. (Input)
size
    is the size of each array element. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also
ebrk()
free()
malloc()
_os_srqmem()
_os9_srqmem()
realloc()
Syntax
#include <UNIX/os9def.h>
double cbrt(double x);

Description
cbrt() returns the cube root of \( x \) (input).
The declaration double cbrt() must be included in any code that uses cbrt().

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
unix.l
ceil()  
Ceiling Function

Syntax
#include <math.h>
double ceil(double x);

Description
ceil() returns the smallest integer, as a double, that is not less than x (input).

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also
floor()
chainc() and chain() are used to load and execute a new module without creating a new process. They are functionally similar to os9forkc() followed by exit(), but with less system overhead.

**Syntax**

```c
#include <process.h>

int chainc(
    const char *modname,
    int parmsize,
    const char *parmptr,
    int type,
    int lang,
    int datasize,
    int prior,
    int pathent);

int chain(
    const char *modname,
    int parmsize,
    const char *parmptr,
    int type,
    int lang,
    int datasize,
    int prior);
```

**Description**

chainc() executes a new program without the overhead of creating a new process. It is functionally similar to os9forkc() followed by exit(), but with less system overhead.

chainc() effectively resets the calling process' program and data areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

Regardless of whether the chain is successful, chainc() never returns to the caller; you must verify that the program to chain to exists and is executable before chaining. Use modlink() to check the module directory for the program or modload() to check the execution directory.
chain() is the same as chainc() without the pathent parameter. The number of
open paths for the new process to inherit is three.

- _os_exec() is the preferred method by which to chain to C programs.
- Be aware of chaining to a system state program. chain() is an historical
  function and is likely to be removed in a future release. _os_exec() is the
  recommended function.
- Threaded programs may only chain from the main() thread and may not
  chain if multiple threads are currently active within the process.

modname
  is a pointer to a null-terminated module name. (Input)

parmsize
  is generally strlen(parmptr). (Input)

parmptr
  is a pointer to a null-terminated string to be passed to the new module. (Input)

type and lang
  specify the desired type and language of the module; values of zero indicate
  any type or language. Only the lower 16 bits of type and lang are used.
  (Input)

datasize
  allocates extra memory to the new program. If no extra memory is required,
  datasize can be zero. (Input)

prior
  is the new priority at which to run the program. Specify zero if no priority
  change is desired. Only the lower 16 bits of prior are used. (Input)

pathent
  is the number of open paths for the new process to inherit. Only the lower 16
  bits of pathent are used. (Input)

Attributes
Operating System:       OS-9 and OS-9 for 68K
State:                 User and System
Threads:               Safe
Re-entrant:            Yes

Library
sys_clib.l

See Also
modlink()
modload()
_os_chain()

os9fork(), os9forkc()

_os_fork()

_os_exec()

F$Chain, F$Fork

OS-9 for 68K Technical Manual

FCHAIN, F_FORK

OS-9 Technical Manual
change_const()

Change Current Static Storage Pointer

Syntax

```c
#include <regs.h>
void *change_const(void *constptr);
void *_change_const(void *constptr);
```

Description

`change_const()` and `_change_const()` change the current constant data pointer and return the previous constant data pointer.

These functions are not present in the libraries for a given processor if a constant data pointer is not applicable.

`constptr` is the value of the new constant data pointer. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

- `change_const()` - cpu.l, `_change_const()` - os_lib.l
### Syntax

```c
#include <regs.h>
void *change_static(void *dest_stat);
void *change_static(void *dest_stat);
```

### Description

`change_static()` and `_change_static()` change the current static storage pointer and return the previous static storage pointer.

- **dest_stat** is the new static storage pointer. (Input)

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

### Library

- `change_static()` - cpu.l, `_change_static()` - os_lib.l
Syntax
#include <modes.h>

int chdir(const char *dirname);

Description
chdir() changes the current data directory for the calling process.

chdir() returns zero after a successful call. If \texttt{dirname} is not a directory path name or some other error occurs, \texttt{-1} is returned; and the appropriate error code is placed in the global variable \texttt{errno}.

chdir() changes the data directory only for the program containing the function call, not the shell that executes the program. Use the built-in shell command \texttt{chd} to change the shell’s data directory.

\texttt{dirname}

is a pointer to a string containing a path name for the directory. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
\texttt{sys_clib.l}

See Also
\_os\_chdir()
I\$ChgDir \hspace{1cm} OS-9 for 68K Technical Manual
I\_CHDIR \hspace{1cm} OS-9 Technical Manual
chd \hspace{1cm} Using OS-9 for 68K and Using OS-9
## Syntax

```c
#include <modes.h>
int chmod(
    const char *name,
    int perm);
```

## Description

`chmod()` changes the access permission bits associated with a file.

`chmod()` returns zero after a successful call. If the caller is not entitled to change the access permissions or if the file cannot be found, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Only the super user or the owner of the file may change the access permissions.

- **name**
  - must be a pointer to a string containing a file name. (Input)

- **perm**
  - should contain the desired bit pattern for the file permissions. Only the lower 16 bits of `perm` are used. (Input)

## Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

## Library

`sys_clib.l`

## See Also

- `_os_ss_attr()`
- `I$SetStt`  
  - OS-9 for 68K Technical Manual
- `I_SETSTAT, SS_ATTR`  
  - OS-9 Technical Manual
- `attr`  
  - Using OS-9 for 68K and Using OS-9
Syntax

```c
#include <modes.h>
int chown(
    const char *name,
    int newowner);
```

Description

chown() changes the owner number of a file.

Refer to your operating system technical manual for information about how the owner ID is stored.

chown() returns zero after a successful call. If the caller does not have permission to change the owner ID or the file cannot be found, -1 is returned and the appropriate error code is placed in the global variable errno.

Only the super user may change the owner ID.

name
    is a pointer to the name of the file. (Input)

newowner
    is the new owner ID to assign to the file. The owner ID consists of a group ID and a user ID. (Input)

Attributes

Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library

sys_clib.1

See Also

_os_ss_fd()
I$SetStt
I_SETSTAT, SS_FD

OS-9 for 68K Technical Manual
OS-9 Technical Manual
chxdir()

Change Current Execution Directory

Syntax

```c
#include <modes.h>

int chxdir(const char *dirname);
```

Description

`chxdir()` changes the current execution directory for the calling process. `chxdir()` returns 0 after a successful call. If `dirname` is not a directory path name or some other error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`chxdir()` changes the execution directory only for the program containing the function call, not the shell that executes the program. Use the built-in shell command `chx` to change the shell's execution directory.

dirname

is a pointer to a string containing the directory name. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `_os_chdir()`
- `I$ChgDir` OS-9 for 68K Technical Manual
- `I_CHDIR` OS-9 Technical Manual
- `chx` Using OS-9 for 68K and Using OS-9
Syntax

```c
#include <stdio.h>
void _cleareof(FILE *stream);
void cleareof(FILE *stream);
```

Description

`_cleareof()` is a macro that resets the end-of-file condition that causes `feof()` to return a non-zero value.

You can use this to retry an input operation on a terminal after end-of-file is encountered. `getc()` does not read characters until the end-of-file condition is cleared.

If you are executing the compiler in strictly conforming ANSI mode, only `_cleareof()` is available. Otherwise, both calls are available.

These calls are implemented as macros in the `stdio.h` header file.

`stream` is a pointer to the stream. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

`sys_clib.l`

See Also

- `feof()`
- `getc()`, `getchar()`
Syntax
#include <stdio.h>
void clearerr(FILE *stream);

Description
clearerr() clears the end-of-file and error indicators for a stream if compiled in strictly conforming ANSI mode. Otherwise, only the error indicator is cleared.

stream is a pointer to the stream to clear. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also
__cleareof(), cleareof()
feof()
ferror()
fopen()
getc(), getchar()
clock()

Get Processor Time

Syntax

#include <time.h>
clock_t  clock(void);

Description

clock() returns a value (in tick units) approximating the processor time used by the current process.

The returned value may be divided by _CLOCKS_PER_SEC to determine the processor time in seconds. _CLOCKS_PER_SEC is defined in the time.h header file to be the number of ticks per second. The era for clock() is the elapsed ticks for this process since the program was forked.

If clock() cannot determine the processor time, it returns (clock_t) -1 to indicate an error.

In order to use clock() from a system-state module, the global variable _environ must be defined. The following C declarations can be used to properly define and initialize _environ:

void *xenviron;
void **_environ = &xenviron;

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also

F$SetSys,F$Time
F_SETSYS,F_TIME

OS-9 for 68K Technical Manual
OS-9 Technical Manual
**Syntax**

```c
#include <modes.h>
int close(int path);
```

**Description**

close() closes an open path. The path number is usually obtained by a previous call to open(), creat(), create(), or dup(). The standard paths 0, 1, and 2 (standard input, standard output, and standard error, respectively) are not normally closed by user programs. If an error occurs during the close, -1 is returned and the appropriate error code is placed in the global variable errno.

- Use a path number, not a file pointer assigned by fopen().
- The thread-enabled version of this function has a cancel point.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

sys_clib.l

**See Also**

- open()
- creat()
- create()
- dup()
- _os_close()
closedir()
Close Named Directory Stream

Syntax
#include <dir.h>
void closedir(DIR *dirp);

Description
closedir() closes the named directory stream and frees the structure associated
with dirp.

dirp
   is a pointer to the directory stream. (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         No

Library
sys_clib.l

See Also
opendir()
readdir()
rewinddir()
seekdir()
telldir()
_cmpnam()  
Compare Two Strings

Syntax
#include <strings.h>
error code _cmpnam(
    const uchar *pattern,
    const uchar *string,
    uint32  length);

Description
_cmpnam() compares the target string to the pattern string to determine if they match. _cmpnam() returns zero if the strings match. -1 is returned if no match occurs.

target
    is a pointer to the target string. (Input)
    The target name must be terminated by a null byte. Upper and lower case characters are considered to match.

pattern
    is a pointer to the pattern string. (Input)
    Two metacharacters are recognized in the pattern string:
    • A question mark (?) matches any single character.
    • An asterisk (*) matches any string of characters.

patlen
    specifies the length of the pattern string. (Input)

Attributes
Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library
sys_clib.l

See Also
_os_cmpnam()
F$CmpNam          OS-9 for 68K Technical Manual
F_CMPNAM          OS-9 Technical Manual
Cosine Function

Syntax

```c
#include <math.h>

double cos(double x);
```

Description

cos() returns the cosine of x (input) as a double. The value of x is in radians. A domain error, EDOM placed in errno, occurs if x is $\infty$ or $-\infty$. cos() returns HUGE_VAL on error.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.l

Possible Error

EDOM
cosh()

Hyperbolic Cosine Function

Syntax

```c
#include <math.h>

double cosh(double x);
```

Description
cosh() returns the hyperbolic cosine of \( x \) (input). The value of \( x \) is in radians. If the magnitude of \( x \) is too large, a range error, ERANGE placed in errno, occurs and the properly signed HUGE_VAL is returned.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library
clib.l

Possible Error

ERANGE
Syntax
#include <process.h>
int _cpymem(
    int pid,
    int count,
    void *from,
    void *into);

Description
_cpymem() copies memory owned by another process or the system into a buffer. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

pid
is the process ID number of the external process. (Input)

    If pid is zero, the system’s address space is assumed. Only the lower 16 bits of pid are used.

count
is the number of bytes to copy. (Input)

from
is a pointer to the address in the process’ address space from which to copy. (Input)

into
is a pointer to the buffer into which to copy the memory. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib

See Also
_os_cpymem()
Syntax

```c
#include <module.h>
int crc(
    void *ptr,
    unsigned int count,
    int *accum);
```

Description

crc() generates or checks the CRC (cyclic redundancy check) values of sections of memory. Compilers, assemblers, and other module generators use crc() to generate a valid module CRC.

To generate the CRC for a module, complete the following steps:

**Step 1.** Initialize the accumulator to -1 (0xffffffff).

**Step 2.** Perform the CRC over the module.

**Step 3.** Call crc() with a NULL value for ptr.

**Step 4.** Complement the CRC accumulator.

**Step 5.** Write the contents of the accumulator to the module.

- **ptr** is a pointer to the data. (Input)
- **count** specifies the byte count for the data. (Input)
- **accum** points to location at which the CRC accumulator is passed. (Input/Output)
  
crc() also stores the updated CRC accumulator at the location pointed to by accum.

Attributes

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: Yes

Library

sys_clib.l
See Also

_os_crc()

F$CRC

F_CRC

OS-9 for 68K Technical Manual

OS-9 Technical Manual

Example

The following code fragment illustrates a technique for generating a CRC for a module:

```
#include <stdio.h>
#include <module.h>

main()
{
    char *ptr;
    int count, accum = -1;
    /* one or more calls to crc */
    crc(ptr, count, &accum);

    /* The final call to crc(). At */
    crc(ptr, count, &accum);

    /* this point the entire module less the four crc */
    /* bytes has been processed */
    crc(NULL, 0, &accum);
        /* this is a special case to */
        /* run a null byte through the */
        /* crc calculation */
    accum = ~accum; /* complement the accumulator.*/
        /* Previous calls have left the */
        /* most significant byte as 0xff */
        /* Complementing changes this byte*/
        /* to zero */
    fwrite(&accum, 1, sizeof accum, fp);
        /* finally write out the four crc bytes */
}
```
creat()

Create File

Syntax

```c
#include <modes.h>
int creat(
    const char *name,
    int mode);
```

Description

creat() returns a path number to a newly created file. The file is available for writing.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

You do not need to specify write access in mode to write to the file. You cannot create directories with this call; instead, use _os_makdir().

The thread enabled version of this function contains a cancel point. For more information, refer to Using OS-9 Threads.

name

is a pointer to the newly created file. (Input)

If name is the name of an existing file, the file is truncated to zero length and the ownership and permissions remain unchanged.

mode

specifies the access modes for the file. (Input)

The modes.h header file contains the valid mode values. Only the lower 16 bits of mode are used.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l
See Also

open()
_os_create()
_os_mkdir()
_os_ss_size()

I$Create and I$SetStt OS-9 for 68K Technical Manual
I_CREATE and I_SETSTAT OS-9 Technical Manual
SS_SIZE OS-9 Technical Manual
**create()**

Create File

**Syntax**

```c
#include <modes.h>
int create(
    const char *name,
    int mode,
    int perm,
    (int initial_size));
```

**Description**

`create()` returns a path number to the newly created file. If the file already exists or any other error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`create()` is similar to the `creat()` call. However, `create()` allows the caller to specify the exact file attributes desired and does not truncate the file if it is already present. You cannot create directories with this call; instead, use `_os_mkdir()`.

- **name**
  - is a pointer to the newly created file. (Input)

- **mode**
  - specifies the access modes for the file. (Input)
  - The `modes.h` header file contains the valid `mode` values. Only the lower 16 bits of `mode` are used.

- **perm**
  - specifies the file permission attributes. (Input)
  - The `modes.h` header file contains the valid `perm` values. Only the lower 16 bits of `perm` are used.

- **initial_size**
  - may be used to indicate the file’s initial allocation size. (Input)
  - For disk files, the allocation is made even though the file size does not change. For pipe files, the pipe buffer is set to this value. This may be used only if the `FAM_SIZE` (or `S_ISIZE`) bit is set in `mode`. 
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_create()
_os_mkdir()
_os_ss_size()

I$Create and I$SetStat OS-9 for 68K Technical Manual
I_CREATE and I_SETSTAT OS-9 Technical Manual
SS_SIZE OS-9 Technical Manual
crtolf()

Convert Carriage Return Characters to Linefeed Characters

Syntax

```c
#include <UNIX/os9def.h>
void crtolf(
    char *buf,
    unsigned int n);
```

Description
crtolf() converts all carriage return characters to linefeed characters in `buf` for a span of `n` characters.

`buf`

is a pointer to a buffer. (Input/Output).

`n`

is the number of characters spanned. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`unix.l`
ctime()

Convert Calendar Time to String Format

Syntax
#include <time.h>
char *ctime(time_t *t);

Description
ctime() converts the calendar time t into the following 26-byte string format, including the terminating \0:
xxx mmm dd hh:mm:ss yyyy\n\0
ctime() returns a pointer to a static area which may be overwritten. To ensure data integrity, use the string or save it immediately.

xxx
  is one of the following days of the week:
  SunMonTueWedThuFriSat

mmm
  is one of the following months of the year:
  JanFebMarAprMayJun
  JulAugSepOctNovDec

dd
  specifies the day of the month.

hh:mm:ss
  specifies the time in hours, minutes, and seconds.

yyyy
  specifies the year.

ctime(t)
  is the equivalent of asctime(localtime(t)). (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         No
Standards:          ANSI

Library
clib.l

See Also
asctime()  gmtime()  localtime()
**Syntax**

```c
#include <io.h>
#include <modes.h>
int detach(dev_list *ptr);
```

**Description**

`detach()` removes a device from the system device table if no other process is using it.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`ptr` is a pointer returned by either `attach()` or `_os_attach()`. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

`sys_clib.l`

**See Also**

- `_os_attach()`
- `_os_detach()`
- I$_$Detach
- I_DETACH

*OS-9 for 68K Technical Manual*

*OS-9 Technical Manual*
Syntax
#include <time.h>

double difftime(
    time_t timel,
    time_t time0);

Description
difftime() returns the difference in seconds between timel and time0. This value is returned as a double. If you could subtract the time_t values, the return value would be: timel - time0.

time0,timel
    are the compared values for difftime(). (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes
Standards:          ANSI

Library
clib.1

See Also
time()
Syntax
#include <stdlib.h>
div_t div(
    int numer,
    int denom);

Description
div() returns the quotient and remainder of the division of the numerator by the
denominator. If the division is inexact, the resulting quotient is the integer of lesser
magnitude that is the nearest to the algebraic quotient. If the result cannot be
represented, the behavior is undefined. Otherwise,
quot * denom + rem equals numer.
div() returns a structure of type div_t, comprising both the quotient and the
remainder. The div_t structure contains the quot and rem fields.

numer
    is the numerator. (Input)

denom
    is the denominator. (Input)

Attributes
Operating System:          OS-9 and OS-9 for 68K
State:                    User and System
Threads:                  Safe
Re-entrant:               Yes
Standards:                ANSI

Library
clib.l
**Syntax**

```c
#include <stdlib.h>
int _dtoa(
    double fnum,
    char *buff,
    int tot_digs,
    int frac_digs,
    int *sign);
```

**Description**

`_dtoa()` converts a floating point number of type `double` to the ASCII string equivalent. The conversion terminates when either the requested number of digits are converted or the specified number of digits after the decimal point is reached.

If `fnum` represents ±∞ or an IEEE 754 Not a Number and `tot_digs` is greater than or equal to three, `buff` contains, respectively, `Inf` or `NaN`. If `tot_digs` is less than three, `buff` contains, respectively, `I` or `N`. In these cases, `_dtoa()` returns `DBL_MAX_10_EXP`.

`_dtoa()` returns no errors.

- `fnum` is the floating point number to convert. (Input)
- `buff` is a pointer to a buffer for the converted ASCII string. (Output)
- `_dtoa()` returns the signed decimal exponent. The returned string only contains mantissa digits. A null byte is appended to the end of `buff`. `buff` should be one byte larger than the requested number of digits.
- `tot_digs` is the number of digits requested. (Input)
- `frac_digs` is the specified number of digits following the decimal point. (Input)
- `sign` is a pointer to the sign. (Input)
  
  Set the sign in `sign` as follows:
  
  - 1 for a negative sign.
  - 0 for a positive sign.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
clib.1
**Syntax**

```
#include <modes.h>

int dup(int path);
```

**Description**

`dup()` returns a synonymous path number for an existing file or device. The lowest available path number is used.

- **path** specifies the path number. (Input)

`dup()` increments the link count of a path descriptor and returns a different synonymous path number. The path descriptor is not cloned.

⚠️ It is ill-advised for more than one process to perform I/O on the same path concurrently.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`sys_clib.l`

**See Also**

- `_os_dup()`
- I$Dup
- I_DUP

*OS-9 for 68K Technical Manual*

*OS-9 Technical Manual*
dup()
**dup2()**

Specifies the Value of Duplicated Path

**Syntax**

```c
#include <UNIX/os9def.h>
int dup2(int fd1, int fd2);
```

**Description**

dup2() returns a synonymous path number for an existing file or device and specifies the value of the new duplicated path.

- **fd1** specifies the original path number. (Input)
- **fd2** specifies the value of the duplicated path. (Input/Output)

If path `fd2` is already in use, it is first closed.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
- **Re-entrant:** No

**Library**

`unix.l`
ebrk()

Get External Memory

Syntax

```c
#include <stdlib.h>
extern int _memmins;
void *ebrk(unsigned int size);
```

Description

ebrk() returns the amount of memory specified by size. The memory is obtained from the system using the F$SRqMem (for OS-9 for 68K) or _os_srqmem() (for OS-9) system request. It is intended for general purpose memory allocation.

To reduce the overhead involved in requesting small quantities of memory, ebrk() requests memory from the system in a minimum size determined by the global variable _memmins and satisfies the user requests from this memory space. _memmins is initially set to 8192. ebrk() grants memory requests from this memory space provided the requests are no larger than the amount of space.

If the request is larger than the available space, ebrk() wastes the rest of the space and tries to get enough memory from the system to satisfy the request. This method works well for programs that need to get large amounts of not necessarily contiguous memory in little bits and cannot afford the overhead of malloc().

Changing the _memmins variable causes ebrk() to use that value as the F$SRqMem (for OS-9 for 68K) or F_SRQMEM (for OS-9) memory request size.

If the memory request is granted, a pointer (even-byte aligned) to the block is returned. If the request is not granted, -1 is returned and the appropriate error code is placed in the global variable errno.

The memory obtained from ebrk() is not returned until the process terminates.

size

specifies the amount of memory. (Input)

Attributes

<table>
<thead>
<tr>
<th>Operating System:</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td>User and System</td>
</tr>
<tr>
<td>Threads:</td>
<td>Safe</td>
</tr>
<tr>
<td>Re-entrant:</td>
<td>No</td>
</tr>
</tbody>
</table>

Library

sys_clib.l
See Also

sbrk()
ibrk()
malloc()
_os_srqmem()
F$SRqMem
F__SRQMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual
endpwent()
Closes Password File

Syntax
#include <UNIX/pwd.h>
void endpwent();

Description
endpwent() can be used to close a password file after processing is complete.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
unix.l

See Also
fgetpwent()
getpw()  
fgetpwent()
getpwnam()  
getpwuid()  
setpwent()
Syntax
#include <UNIX/err.h>
void err(int eval, const char *fmt, ...);
void errx(int eval, const char *fmt, ...);

Description
The err() family of functions displays a formatted error message on the standard error output. In all cases, the last component of the program name, a colon character, and a space are output. If the fmt argument is not NULL, the formatted error message is output. In the case of the err() function, the error message string affiliated with the current value of the global variable errno is output next, preceded by a colon character and a space if fmt is not NULL. In all cases, the output is followed by a newline character.

These functions do not return, rather they exit with the value of the argument eval.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
unix.l

See Also
verr(), verrx()
Syntax

#include <stdio.h>

int _errmsg(
    int nerr,
    const char *msg,
    [u_int32 arg1,
    u_int32 arg2,
    u_int32 arg3]);

For C++ Programmers:
The parameters for this function are not optional in C++. You must specify each; if you do not, a compilation error will occur.

Description

_errmsg() displays an error message and the program name on the standard error path. For added flexibility in message printing, msg can be a conversion string suitable for fprintf() with up to three additional parameters of any integral type. nerr is returned as the value of the function so _errmsg() can be used as a parameter to a function such as exit() or prerr().

nerr
specifies the error number returned by _errmsg(). (Input)

msg
is a pointer to the error message. (Input)
The message string is displayed in the following format:

prog: <message text>

prog
is the module name of the program, and <message text> is the string pointed to by msg.

[arg1-arg3]
Input
Example
Assume the program calling the function is named `foobar`:

Call: `errmsg(1,"programmed message\n");`
Prints: `foobar: programmed message`
Call: `exit(_errmsg(errno, "unknown option '%c'\n', 'q'));`
Prints: `foobar: unknown option 'q'
Then exits with `errno` as its termination status.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
`sys_clib.l`

See Also
`fprintf()`,
`_prgname()`
**Syntax**

```c
#include <events.h>
int _ev_creat(
    int ev_value,
    int wait_inc,
    int signal_inc,
    const char *ev_name);
```

**Description**

`_ev_creat()` creates an event.

An event ID number is returned if the event is successfully created. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Refer to the OS-9 Technical Manual for more information about events.

- **ev_value**
  is the initial value for the event. (Input)

- **wait_inc**
  is the wait increment. (Input)
  It is added to the value of the event when a successful `_ev_wait()`, `_ev_waitr()`, `_os_ev_wait()`, or `_os_ev_waitr()` is performed.

- **signal_inc**
  is the signal increment. (Input)
  It is added to the value of the event when an `_ev_signal()` or `_os_ev_signal()` is performed.

- **ev_name**
  is a pointer to the name for the event. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
_ev_creat()

Library
sys_clib.1

See Also
_ev_signal()
_ev_wait()
_ev_waitr()
_os_ev_creat()
_os_ev_signal()
_os9_ev_wait()
_os_ev_waitr()
F$Event  OS-9 for 68K Technical Manual
F_EVENT  OS-9 Technical Manual
EV_CREAT  OS-9 Technical Manual
EV_SIGNAL OS-9 Technical Manual
EV_WAIT  OS-9 Technical Manual
EV_WAITR OS-9 Technical Manual
Syntax

```
#include <events.h>
int _ev_delete(const char *ev_name);
```

Description

`_ev_delete()` deletes an event. The use count for the event must be zero before the event can be deleted.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`ev_name` is a pointer to the name of the event. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `_os_ev_delete()`
- F$Event `OS-9 for 68K Technical Manual`
- F_EVENT, EV_DELETE `OS-9 Technical Manual`
 Syntax

```
#include <events.h>

OS-9:
int _ev_info(
    int     ev_index,
    void     *ev_buffer);

OS-9 for 68K:
int _ev_info(
    int     ev_index,
    ev_infostr *ev_buffer);
```

Description

=e_v_info()= returns information about an event. The event table is indexed from zero to one less than the maximum number of events allowed on the system. If ev_index is greater than all active events in the table, -1 is returned and the appropriate error code is placed in the global variable errno.

The format of the information written to ev_buffer is the same regardless of the OS, it matches the OS-9 for 68K ev_infostr structure. For OS-9, use _os_ev_info() instead of this call.

- ev_index corresponds to the starting point in the event table to search for an event. (Input)
- ev_buffer is a pointer to the event structure buffer where _ev_info() will write the event information, if found. (Output)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

sys_clib.l
See Also

_os_ev_info()

F$Event  OS-9 for 68K Technical Manual
F_EVENT, EV_INFO  OS-9 Technical Manual
**Syntax**

```c
#include <events.h>
int _ev_link(const char *ev_name);
```

**Description**

`_ev_link()` links to an existing event. An event ID number is returned if the event is successfully linked.

If an error occurs, –1 is returned and the appropriate error code is placed in the global variable `errno`.

`ev_name` is a pointer to the name of the event. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

sys_clib.l

**See Also**

- `_os_ev_link()`
- `F$Event`  
  OS-9 for 68K Technical Manual
- `F_EVENT, EV_LINK`  
  OS-9 Technical Manual
_ev_pulse()

Signal Event Occurrence

Syntax

```c
#include <events.h>

int _ev_pulse(
    int ev_id,
    int ev_value,
    int allflag);
```

Description

_ev_pulse() indicates that an event has occurred. The current event value is saved. The event variable is set to the value given by `ev_value`, and the normal signal increment is not applied. The saved event value is restored after activating processes, if any.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Refer to your operating system technical manual for more information about events.

`ev_id` is the event ID returned from a call to _ev_creat(), _ev_link(), _os_ev_creat(), or _os_ev_link(). (Input)

`ev_value` sets the value of the event variable. (Input)

`allflag` indicates which process(es) to activate. (Input)

- Only the lower 16 bits of `allflag` are used.
- If `allflag` is zero, the first process waiting for the event is activated.
- If `allflag` is 0x8000, all processes waiting for the event that have a value in range are activated.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l
See Also

_ev_creat()
_ev_link()
_os_ev_creat()
_os_ev_link()
_os_ev_pulse()

F$Event
F_EVENT
V_CREAT
EV_LINK
EV_PULSE
Syntax
#include <events.h>
int _ev_read(int ev_id);

Description
_ev_read() reads the value of an event, without waiting for or affecting the event variable, and returns the current value of the event.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno. Clear the global variable errno prior to calling _ev_read() to distinguish between return of an event value of -1 and an error.

ev_id identifies the event. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.1

Example
errno = 0;
if ((ev_val = _ev_read(ev_id)) == -1 && errno)
    exit(_errmsg(errno, "can't read event value - ");

See Also
_os_ev_read()

F$Event OS-9 for 68k Technical Manual
F_EVENT OS-9 Technical Manual
EV_READ OS-9 Technical Manual
_ev_set()

Set Event Variable and Signal Event Occurrence

Syntax

```c
#include <events.h>
int _ev_set(
    int ev_id,
    int ev_value,
    int allflag);
```

Description

_ev_set() indicates that an event has occurred. The event variable is set to the value given by `ev_value`, and the normal signal increment is not applied. Processes waiting for the event are then activated.

OS-9 for 68K Users: _ev_set() returns the value of the event before changing to `ev_value`. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling _ev_set() to distinguish between return of a prior event value of -1 and an error.

OS-9 Users: _ev_set() returns zero. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

- `ev_id` is the event ID returned from a call to _ev_creat(), _ev_link(), _os_ev_creat(), or _os_ev_link(). (Input)
- `ev_value` sets the value of the event variable. (Input)
- `allflag` specifies the process(es) to activate. (Input)
  - Only the lower 16 bits of `allflag` are used.
    - If `allflag` is zero, the first process waiting for the event is activated.
    - If `allflag` is 0x8000, all processes waiting for the event that have a value in range are activated.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
Library
sys_clib.1

See Also
(ev_creat)
(ev_link)
(os_ev_creat)
(os_ev_link)
(os_ev_set)

F$Event OS-9 for 68K Technical Manual
F_EVENT OS-9 Technical Manual
EV_CREAT OS-9 Technical Manual
EV_LINK OS-9 Technical Manual
EV_SET OS-9 Technical Manual
Set Relative Event Variable and Signal Event Occurrence

Syntax

```c
#include <events.h>
int _ev_setr(
    int ev_id,
    int ev_value,
    int allflag);
```

Description

`_ev_setr()` indicates that an event has occurred. The event variable is incremented by the value given by `ev_value`. Processes waiting for the event are then activated.

OS-9 for 68K Users: `_ev_setr()` returns the value of the event before changing by `ev_value`. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_setr()` to distinguish between return of a prior event value of -1 and an error.

OS-9 Users: `_ev_setr()` returns 0. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`ev_id` is the event ID returned from a call to `_ev_creat()`, `_ev_link()`, `_os_ev_creat()`, or `_os_ev_link()`. (Input)

`ev_value` specifies the amount to increment the event variable. (Input)

`allflag` specifies the process(es) to activate. (Input)

Only the lower 16 bits of `allflag` are used.

- If `allflag` is zero, the first process waiting for the event is activated.
- If `allflag` is 0x8000, all processes waiting for the event that have a value in range are activated.

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

Library

`sys_clib.l`
See Also

_ev_creat()
_ev_link()
_os_ev_creat()
_os_ev_link()
_os_ev_setr()

F$Event OS-9 for 68K Technical Manual
F_EVENT OS-9 Technical Manual
EV_CREAT OS-9 Technical Manual
EV_LINK OS-9 Technical Manual
EV_SETR OS-9 Technical Manual
**Syntax**

```c
#include <events.h>
int _ev_signal(
    int ev_id,
    int allflag);
```

**Description**

`_ev_signal()` indicates that an event has occurred. The current event variable is updated by the signal increment which was given when the event was created.

**OS-9 for 68K Users:** `_ev_signal()` returns the value of the event before incremented by the signal increment. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `_ev_signal()` to distinguish between return of a prior event value of -1 and an error.

**OS-9 Users:** `_ev_signal()` returns zero. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

- `ev_id` is the event ID returned from a call to `_ev_creat()`, `_ev_link()`, `_os_ev_creat()`, or `_os_ev_link()`. (Input)
- `allflag` specifies the process(es) to activate. (Input)
  - Only the lower 16 bits of `allflag` are used.
  - If `allflag` is zero, the first process waiting for the event is activated.
  - If `allflag` is 0x8000, all processes waiting for the event that have a value in range are activated.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

sys_clib.l
See Also

_ev_creat()
_ev_link()
_os_ev_creat()
_os_ev_link()
_os_ev_signal()

F$Event OS-9 for 68K Technical Manual
F_EVENT OS-9 Technical Manual
EV_CREAT OS-9 Technical Manual
EV_LINK OS-9 Technical Manual
EV_SIGNAL OS-9 Technical Manual
**Syntax**

```c
#include <events.h>
int _ev_unlink(int ev_id);
```

**Description**

 `_ev_unlink()` decrements the link count of an event. When the link count becomes zero, you can delete the event using `_ev_delete()` or `_os_ev_delete()`.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`ev_id` specifies the event ID.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

sys_clib.l

**See Also**

- `_ev_delete()`
- `_os_delete()`
- `_os_ev_unlink()`

F$Event OS-9 for 68K Technical Manual
F_EVENT OS-9 Technical Manual
EV_DELETE OS-9 Technical Manual
EV_UNLINK OS-9 Technical Manual
Syntax

```
#include <events.h>
int _ev_wait(
    int ev_id,
    int ev_min,
    int ev_max);
```

Description

_CPP_ 

_wait() waits for an event to occur. The event value is compared to the range values specified by ev_min and ev_max. If the event value is not in the specified range, the process waits until some other process places the value within the range. Once in range, the wait increment is applied to the event value. The actual event value is returned as the value of the function.

_wait() returns the value of the event after completion of wait. If an error occurs, -1 is returned and the appropriate error code is placed in errno. Clear the global variable errno prior to calling _ev_wait() to distinguish between return of a current event value of -1 and an error. Further, if a signal was received by the process, the event value returned is outside of the range ev_min to ev_max.

```c
errno = 0;
if ((ev_val = _ev_wait(ev_id, -5, 0)) == -1 && errno)
    exit(_errmsg(errno, "error waiting on event - "));
if (ev_val < -5 || ev_val > 0)
    _errmsg(1, "received signal while waiting for event
");
```

**ev_id**

specifies the event ID. (Input)

**ev_min**

specifies the minimum range value for the event. (Input)

**ev_max**

specifies the maximum range value for the event. (Input)
Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

_os_ev_wait()
_os9_ev_wait()
F$Event OS-9 for 68K Technical Manual
F_EVENT OS-9 Technical Manual
EV_WAIT OS-9 Technical Manual
**Syntax**

```c
#include <events.h>
int _ev_waitr(
    int ev_id,
    int ev_min,
    int ev_max);
```

**Description**

_`ev_waitr()`_ waits for the specified event to occur. The event value is compared to the relative values specified by `ev_min` and `ev_max`. The current event value is added to the range values before the comparison. If the event value is not in the specified range, the process waits until some other process places the value within the range. Once in range, the wait increment is applied to the event value. The actual event value is returned as the value of the function.

_`ev_waitr()`_ returns the value of the event after completion of the wait. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`. Clear the global variable `errno` prior to calling `ev_waitr()` to distinguish between return of a current event value of -1 and an error. Further, if a signal was received by the process, the event value returned is outside of the relative range `ev_min` to `ev_max`.

- **ev_id**
  - specifies the event ID for the event. (Input)

- **ev_min**
  - specifies the minimum range value for the event. (Input)

- **ev_max**
  - specifies the maximum range value for the event. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

`sys_clib.l`
See Also

_os_ev_waitr()
F$Event OS-9 for 68K Technical Manual
F_EVENT, EV_WAITR OS-9 Technical Manual
execl()

Calls a File With Variable Arguments

Syntax
#include <UNIX/os9def.h>
int execl(
    const char *path,
    const char *arg0,
    ...);

Description
execl() is the UNIX-like exec interface to the _os_chain() system call.

arg0 (input), and any additional variable arguments, are the arguments to pass to
the chained process. arg0 is conventionally the name of the chained program. The
list of variable arguments is terminated by a NULL entry.

The global variable _environ is used for the environment for the call to
_os_chain().

execl() returns -1 and sets the global variable errno to indicate an error. execl()
only returns an error if setting up for the _os_chain() call fails. Once an attempt to
chain occurs, the process exits if any error occurs.

- To use execl() link with unix.l.
- A threaded process may only chain from the main() thread and may not
  chain if multiple threads are active in the process.
- _os_chain() does not do the normal PATH searching that the shell does.

path
    is the pathlist to the program on which to chain. (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         No

Library
unix.l

See Also
execl()
execv()
exevp()
execle()  
Calls a File With Variable Arguments

**Syntax**

```c
#include <UNIX/os9def.h>
int execle(
    const char *path,
    const char *arg0,
    ..., 
    const char *envp[]);
```

**Description**

`execle()` is the UNIX-like exec interface to the `_os_chain()` system call.

The list of variable arguments is terminated by a `NULL` entry. The `NULL` entry is followed by an environment pointer to use for the call to `_os_chain()`. `envp` is a pointer to an array of strings that constitute the environment of the process.

`execle()` returns -1 and sets the global variable `errno` to indicate an error. `execle()` only returns an error if setting up for the `_os_chain()` call fails. Once an attempt to chain occurs, the process exits if any error occurs.

- **To use exec1(), link with unix.1.**
- A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.
  - `_os_chain()` does not do the normal PATH searching that the shell does.

`path`

is the pathlist to the program on which to chain. (Input)

`arg0`

is conventionally the name of the chained program. (Input)

`arg0`, and any additional variable arguments, are the arguments to pass to the chained process.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No

**Library**

`unix.1`

**See Also**

`exec1()`  `execv()`  `execvp()`
execv()
Calls a File With an Argument Vector

Syntax
#include <UNIX/os9def.h>
int execv(
    const char *path,
    char *const argv[]);

Description
execv() is a UNIX-like interface to the _os_chain() system call. The argument to
execv() is an argument vector to pass to the chained process and the path to the
program on which to chain. The last argument should be a NULL entry.

- To use execv(), link with unix.l.
- _os_chain() does not do the normal PATH searching that the shell does.
- A threaded process may only chain from the main() thread and may not
  chain if multiple threads are active in the process.

argv[0]
is the name of the program being chained to. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
unix.l

See Also
execi()  execle()  execvp()
execve()

Calls a File With an Argument Vector

Syntax

#include <UNIX/os9def.h>

int execv(
    const char *path,
    char *const argv[],
    char *const envp[]);

Description

execve() is a UNIX-like interface to the _os_chain() system call. The argument to
execve() is an argument vector to pass to the chained process and the path to the
program on which to chain. The last argument should be a NULL entry.

• To use execve(), link with unix.l.
• _os_chain() does not do the normal PATH searching that the shell does.
• A threaded process may only chain from the main() thread and may not
chain if multiple threads are active in the process.

path
    is the pathlist to the program on which to chain. (Input)

argv[0]
    is the name of the program being chained to. (Input)

envp is a pointer to a null-terminated array of character pointers to null-
terminated strings. (Input)

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library

unix.l

See Also

execl()
execle()
execvp()
execvp() Calls a File With an Argument Vector

Syntax

```
#include <UNIX/os9def.h>
int execvp(
    const char *file,
    char *const argv[]);
```

Description

execvp() is a UNIX-like interface to the _os_chain() system call. The argument to execvp() is an argument vector to pass to the chained process and the path to the program to chain to. The last argument should be a NULL entry.

- To use execve(), link with unix.l.
- _os_chain() does not do the normal PATH searching that the shell does.
- A threaded process may only chain from the main() thread and may not chain if multiple threads are active in the process.

file

argv[0]

is the name of the program being chained to. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

unix.l

See Also

exec1() execle() execv()
Task Termination

Syntax

#include <stdlib.h>
void _exit(int status);

Description

_exit() immediately terminates a program.

An exit status of zero is considered normal termination. Most programs (especially the shell) interpret a non-zero value as an error code.

_exit() cannot return to its caller, and it does not flush and close standard I/O buffers.

status is the exit status. (Input)

The parent process can use status to determine the program’s success or failure.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

_os_exit()  OS-9 for 68K Technical Manual
F$Exit  OS-9 for 68K Technical Manual
F_EXIT  OS-9 Technical Manual
exit()
Task Termination

Syntax

#include <stdlib.h>
void exit(int status);

Description

exit() is the normal means of terminating a task. All functions specified by
atexit() are called in the reverse order of the order in which they were specified.

All open streams with unwritten buffered data are flushed, all open streams are
closed, and all files created by tmpfile() are removed. Control is returned to the
host environment.

exit() cannot return to its caller.

status is the exit status. (Input)

- If status is zero or EXIT_SUCCESS, zero is communicated to the parent
  process executing the _os_wait().
- If status is EXIT_FAILURE, one is communicated.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also

atexit()
_os_exit()
_os_wait()
tmpfile()
F$Exit OS-9 for 68K Technical Manual
F_EXIT OS-9 Technical Manual
**Syntax**

```c
#include <math.h>
double exp(double x);
```

**Description**

`exp()` returns the exponential function of `x` (input), $e^{x}$, raised to the power. A range error, `ERANGE` stored in `errno`, occurs if the magnitude of `x` is too large. In this case, `exp()` returns `HUGE_VAL`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

`clib.l`

**Possible Error**

`ERANGE`
**fabs()**

Floating Absolute Value

**Syntax**

```c
#include <math.h>
double fabs(double x);
```

**Description**

`fabs()` returns the absolute value of `x` (input).

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

**Library**

`clib.1`
fclose()
Close File

Syntax
#include <stdio.h>
int fclose(FILE *stream);

Description
fclose() flushes the file pointed to by stream and closes the associated file. Any
unwritten buffered data for the stream are written to the file; any unread buffered
data are discarded. The stream is disassociated from the file. If the associated buffer
was automatically allocated, it is deallocated. If stream was returned from
tmpfile(), it is deleted.

fclose() returns zero if the stream was successfully closed. Otherwise, EOF is
returned.

stream is a pointer to the C I/O FILE structure. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also
fflush()
fopen()
getc(), getchar()
putc(), putchar()
tmpfile()
fdopen()  
Attach Path to File Pointer

Syntax
#include <stdio.h>
FILE *fdopen(
    int path,
    const char *action);

Description
fdopen() returns a file pointer to the file specified by a currently open path.
Use fdopen() when you require some special processing not provided by fopen() for opening files.

path
is the path to attach. (Input)

action
is a pointer to the file action string. (Input)
The action must be compatible with the access mode of the given path. The valid actions are listed in Table 2-3.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Open for reading</td>
</tr>
<tr>
<td>w</td>
<td>Open for writing</td>
</tr>
<tr>
<td>a</td>
<td>Append (write) at the end of the file or create file for writing if path does not exist</td>
</tr>
<tr>
<td>r+</td>
<td>Open for update</td>
</tr>
<tr>
<td>w+</td>
<td>Create for update</td>
</tr>
<tr>
<td>a+</td>
<td>Create or open for update at end of file</td>
</tr>
<tr>
<td>d</td>
<td>Directory read</td>
</tr>
</tbody>
</table>

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
sys_clib.l

See Also
fopen() freopen()
feof()  
Check Buffered File for End of File

Syntax
#include <stdio.h>
int feof(FILE *stream);

Description
feof() tests the end-of-file indicator for the specified stream. feof() returns a non-zero value only if the end-of-file indicator is set for stream.

stream is a pointer to the C I/O FILE structure. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also
clearerr()
fopen()
ferror()

Check Buffered File for Error Condition

Syntax

```c
#include <stdio.h>
int ferror(FILE *stream);
```

Description

`ferror()` tests the error indicator for the specified stream. `ferror()` returns a non-zero value only if the error indicator is set for `stream`.

`stream` is a pointer to the C I/O `FILE` structure. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.l

See Also

clearerr()
fopen()
**Syntax**

```c
#include <stdio.h>
int fflush(FILE *stream);
```

**Description**

`fflush()` writes any unwritten data for the specified stream to the file. Unwritten data is written if the stream was opened in either `write` or `update` mode.

`fflush()` returns `EOF` if a write error occurs or if the stream is read-only. Otherwise, it returns zero.

`stream` is a pointer to the C I/O `FILE` structure. (Input)

If `stream` is a null pointer, `fflush()` performs this flushing action on all streams opened in either `write` or `update` mode and containing unwritten data.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

**Library**

clib.l

**See Also**

- `fopen()`
- `getc()`, `getchar()`
- `putc()`, `putchar()`
ffs()  
Find First Set Bit

**Syntax**

```c
#include <UNIX/os9def.h>

int ffs(unsigned int i);
```

**Description**

`ffs()` finds the first bit set in the argument passed to it and returns the index of that bit. Bits are numbered starting at one (1) from the least significant. A return value of zero indicates that the value passed to the function was zero.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

unix.l
Syntax

#include <stdio.h>

int fgetc(FILE *stream);

Description

fgetc() returns a character, as an unsigned converted to an int, from the file and advances the associated file position indicator for the stream, if defined.

stream is a pointer to the C I/O FILE structure. (Input)

If the stream is at end-of-file, the end-of-file indicator for the stream is set and fgetc() returns EOF. If a read error occurs, the error indicator for the stream is set and fgetc() returns EOF.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

getc(), getchar()
fgetpos()

Get Current Position in File

Syntax

```c
#include <stdio.h>
int fgetpos(
    FILE *stream,
    fpos_t *pos);
```

Description

fgetpos() stores the current value of the file position indicator. The stored value contains information usable by fsetpos() for repositioning the stream to its position at the time of the fgetpos() call. If successful, fgetpos() returns zero. Otherwise, it returns a non-zero value and the appropriate error code is placed in the global variable errno.

**stream**

is a pointer to the C I/O FILE structure. (Input)

**pos**

is a pointer to the file position indicator. (Output)

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No
- **Standards:** ANSI

Library

clib.l

See Also

fsetpos()
fgetpwent()  
Get Password File Entry

**Syntax**

```c
#include <UNIX/pwd.h>
struct passwd *fgetpwent(FILE *f);
```

**Description**

`fgetpwent()` returns a pointer to the next password structure in the stream (f), which matches the format of the password file `/dd/SYS/password`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

**Library**

`unix.l`

**See Also**

`endpwent()`  
`getpw()`  
`getpwent()`  
`getpwnam()`  
`getpwuid()`  
`setpwent()`
fgets()
Get String from File

Syntax
#include <stdio.h>
char *fgets(
    char *ptr,
    int cnt,
    FILE *stream);

Description
fgets() reads at most one less than the specified number of characters from the stream into an array. No additional characters are read after a newline character (which is returned) or after end-of-file. A null character is added to the end of the string.

fgets() returns ptr if successful. If the end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

ptr
    is a pointer to the array into which to read the characters. (Output)

cnt
    is the number of characters to read. (Input)

stream
    is a pointer to the C I/O FILE structure. (Input)

Attributes
Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library
clib.l

See Also
fgetc()
Syntax
#include <stdio.h>
int _fileno(FILE *stream);
int fileno(FILE *stream);

Description
_fileno() returns the path number associated with the specified file pointer. The path number is not valid unless (stream->_flag & _INIT) is non-zero. This call is only available as a macro.

If you are executing the compiler in strictly conforming ANSI mode, only _fileno() is available. Otherwise, both calls are available.

stream is a pointer to the C I/O FILE structure. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
These macros are defined in the stdio.h header file.

See Also
fopen()
freopen()
findnstr()

Search String for Pattern

Syntax

```c
#include <strings.h>
int findnstr(
    int pos,
    const char *string,
    const char *pattern,
    int len);
```

Description

`findnstr()` searches the string for the first occurrence of the pattern. It starts at position `pos` (where the first position is one, not zero). The returned value is the position of the first matched character of the pattern in the string, or zero if a match is not found. `findnstr()` stops searching at `len`; it may continue past null bytes.

⚠️ The current implementation does not use the most efficient pattern matching algorithm. `strstr()` is preferred for very long strings.

- **pos** specifies the starting position. (Input)
- **string** is a pointer to the string to search. (Input)
- **pattern** is a pointer to the pattern to use for the search. (Input)
- **len** specifies the final position to be searched. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `findstr()`
- `strstr()`
findstr()

Search String for Pattern

Syntax

#include <strings.h>
int findstr(
    int pos,
    const char *string,
    const char *pattern);

Description

findstr() searches the string for the first instance of the pattern. It starts at position pos (where the first position is one, not zero). The returned value is the position of the first matched character of the pattern in the string, or zero if a match is not found. findstr() stops searching when a null byte is found in string.

The current implementation does not use the most efficient pattern matching algorithm. strstr() is preferred for very long strings.

pos
    specifies the starting position. (Input)

string
    is a pointer to the string to search. (Input)

pattern
    is a pointer to the pattern to use for the search. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

findnstr()
strstr()
Syntax

#include <math.h>

double floor(double x);

Description

floor() returns the largest integer, as a double, that is not greater than x (input).

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library

clib.l

See Also

ceil()
Syntax

```c
#include <math.h>

double fmod(
    double x,
    double y);
```

Description

`fmod()` computes the floating-point remainder of `x` (input) divided by `y` (input). If `y` is non-zero, `fmod()` returns the value with the same sign as `x` and a magnitude less than the magnitude of `y`. If `y` is zero, zero is returned.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`clib.l`

Possible Error

EDOM
fopen()
Open File

Syntax

```c
#include <stdio.h>
FILE *fopen(
    const char *name,
    const char *mode);
```

Description

`fopen()` opens a file and associates a stream with it. If an error occurs, `fopen()` returns NULL.

When opened, a stream is fully buffered only if it does not refer to an interactive device. The error and end-of-file indicators for the stream are cleared.

Three file pointers are available and are considered open the moment the program runs:

- `stdin` standard input I/O is to path 0
- `stdout` standard output I/O is to path 1
- `stderr` standard error I/O is to path 2

These macros are defined in the `stdio.h` header file.

You cannot open a directory for writing with `fopen()`.

| name | is a pointer to the name of the file. (Input) |
| mode | is a pointer to the access mode. (Input) |
| mode may be one of the following: |

Table 2-2. Mode Options

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>r</code></td>
<td>Open text file for reading</td>
</tr>
<tr>
<td><code>w</code></td>
<td>Truncate to zero length or create text file for writing</td>
</tr>
<tr>
<td><code>a</code></td>
<td>Append; open or create text file for writing at end-of-file</td>
</tr>
<tr>
<td><code>rb</code></td>
<td>Open binary file for reading</td>
</tr>
<tr>
<td><code>wb</code></td>
<td>Truncate to zero length or create binary file for writing</td>
</tr>
<tr>
<td><code>ab</code></td>
<td>Append; open or create binary file for writing at end-of-file</td>
</tr>
<tr>
<td><code>r+</code></td>
<td>Open text file for update (reading and writing)</td>
</tr>
<tr>
<td><code>w+</code></td>
<td>Truncate to zero length or create text file for update</td>
</tr>
</tbody>
</table>
Table 2-2. Mode Options  (Continued)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a+</td>
<td>Append; open or create text file for update, writing at end-of-file</td>
</tr>
<tr>
<td>r+b, rb+</td>
<td>Open binary file for update (reading and writing)</td>
</tr>
<tr>
<td>w+b, wb+</td>
<td>Truncate to zero length or create binary file for update (reading and writing).</td>
</tr>
<tr>
<td>a+b, ab+</td>
<td>Append; open or create binary file for update, writing at end-of-file</td>
</tr>
<tr>
<td>d</td>
<td>Directory read (OS-9 extension); this is only available in K &amp; R source mode.</td>
</tr>
</tbody>
</table>

Appending an x to any action(s) indicates that names are relative to the current execution directory. The x also implies that the file should have the execute permission bit set. This option is only available for OS-9 in non-ANSI mode.

In modes other than compatibility mode, appending a d to any action(s) indicates that the file is a directory.

mode must point to a string and not a character; `fopen("fun","r")` is correct, `fopen("fun","r'`) is not correct. If the file does not exist or cannot be read, opening a file with read mode fails.

On OS-9 for 68K systems, when you open a file with append mode, the file position indicator is initially positioned at the beginning of the file. However, all subsequent writes to the file are forced to the then current end-of-file, regardless of intervening calls to `fseek()`. On OS-9 systems, if the append mode bit is set and the target file already exists, the file is opened and the associated file pointer points to the end of the file. If an error occurs when opening a file with append mode, refer to `fopen Append Bit subsection in the appropriate processor-specific chapter of the Ultra C/C++ Processor Guide.`

When a file is opened with update mode, both input and output may be performed on the associated stream. However, output may not be directly followed by input without an intervening call to `fflush()` or to a file positioning function (`fseek()`, `fsetpos()`, or `rewind()`). Likewise, input may not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.

Example

Examples using `fopen()` are found in Table 2-3.

Table 2-3. fopen() Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fp = fopen(fred,wx)</code>;</td>
<td>Creates a file for writing in the execution directory</td>
</tr>
<tr>
<td><code>fp = fopen(moe,r)</code>;</td>
<td>Opens an existing file for reading</td>
</tr>
<tr>
<td><code>fp = fopen(stooge/curly,w+);</code></td>
<td>Creates a file for reading and writing</td>
</tr>
</tbody>
</table>

Ultra C Library Reference 209
Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

creat()
fflush()
fseek()
getc(), getchar()
open()
_os_create()
_os9_create()
_os_open()
putc(), putchar()
Syntax

```c
#include <UNIX/util.h>

char *fparseln(FILE *stream, size_t *len, size_t *lineno, const char delim[3],
                   int flags);
```

Description

The `fparseln()` function returns a pointer to the next logical line from the stream referenced by `stream`. This string is NUL terminated and it is dynamically allocated on each invocation. It is the responsibility of the caller to free the pointer.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

util.l
Syntax

```c
#include <stdio.h>
int fprintf(
    FILE *stream,
    const char *format, ...);
```

Description

`printf()`, `fprintf()`, and `sprintf()` are standard C library functions that perform formatted output. Each of these functions converts, formats, and prints any parameters as indicated by the control string.

`fprintf()` writes output to a stream.

`fprintf()` returns when the end of the format string is encountered.

A format specification consists of a percent (%) character followed by (in this order):

1. Zero or more flags that modify the meaning of the conversion specification. You may use the following flags in any order:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>The result of the conversion is left-justified within the field. If this flag is not specified, the result is right-justified.</td>
</tr>
<tr>
<td>+</td>
<td>The result of a signed conversion always begins with a plus or minus sign. If this flag is not specified, the result begins with a sign only when a negative value is converted.</td>
</tr>
<tr>
<td>space</td>
<td>If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a space is prefixed to the result. If the space and + flags both appear, the space flag is ignored.</td>
</tr>
<tr>
<td>#</td>
<td>The result is to be converted to an alternate form. For o conversion, it increases the precision to force the first digit of the result to zero. For x (or X) conversion, a non-zero result has a 0x (or 0X) prefix. For e, E, f, g, and G conversions, the result always contains a decimal point character, even if no digits follow. Normally, a decimal point character appears in the result of these conversions only if a digit follows it. For g and G conversions, trailing zeros are not removed from the result. For other conversions, the behavior is undefined.</td>
</tr>
<tr>
<td>0</td>
<td>For d, i, o, u, x, X, e, E, f, g, and G conversions, leading zeros, following any indication of sign or base, pad the field width; no space padding is performed. If the 0 and - flags both appear, the 0 flag is ignored. For d, i, o, u, x, and X conversions, the 0 flag is ignored if a precision is specified. For other conversions, the behavior is undefined.</td>
</tr>
</tbody>
</table>
2. An optional minimum field width. If the converted value has fewer characters than the field width, it is padded with spaces (by default) on the left (or right, if the left adjustment flag has been given) to the field width. The field width takes the form of an asterisk (*) or a decimal integer.

3. An optional precision that gives:
   - The minimum number of digits to appear for the \(d\), \(i\), \(o\), \(u\), \(x\), and \(X\) conversions.
   - The number of digits to appear after the decimal point character for \(e\), \(E\), and \(f\) conversions.
   - The maximum number of significant digits for the \(g\) and \(G\) conversions.
   - The maximum number of characters to be written from a string in \(s\) conversion.

   The precision takes the form of a period (.) followed either by an asterisk (*) or by an optional decimal integer. If only the period is specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.

4. An optional \(h\), \(l\), or \(L\) that specifies:

   **Table 2-5. Optional Flags for fprintf()**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(h)</td>
<td>The following (d), (i), (o), (u), (x), or (X) conversion specifier applies to a short int or an unsigned short int parameter, or the following (n) conversion specifier applies to a pointer to a short int parameter.</td>
</tr>
<tr>
<td>(l)</td>
<td>The following (d), (i), (o), (u), (x), or (X) conversion specifier applies to a long int or an unsigned long int parameter, or the following (n) conversion specifier applies to a pointer to a long int parameter.</td>
</tr>
<tr>
<td>(L)</td>
<td>The following (e), (E), (f), (g), or (G) conversion specifier applies to a long double parameter. The following (d), (i), (o), (u), (x), or (X) conversion specifier applies to a long long int parameter.</td>
</tr>
</tbody>
</table>

   If an \(h\), \(l\), or \(L\) appears with any other conversion specifier, the behavior is undefined.

5. A character that specifies the type of conversion to apply.

   An asterisk (*) may indicate the field width, precision, or both. In this case, an int parameter supplies the field width or precision. The parameters specifying field width, precision, or both appear in that order before the parameter, if any, to convert.

   A negative field width parameter is taken as a - flag followed by a positive field width.

   A negative precision parameter is taken as if the precision was omitted.
The conversion specifiers and their meanings are:

### Table 2-6. fprintf() Conversion Specifiers

<table>
<thead>
<tr>
<th>Flag</th>
<th>Conversion Specifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>The int parameter is converted to an unsigned char, and the resulting character is written.</td>
</tr>
<tr>
<td>d, i</td>
<td>The int parameter is converted to a signed decimal in the style [-]dddd. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is 1. The result of converting a zero value with a precision of zero is no characters.</td>
</tr>
<tr>
<td>e, E</td>
<td>The double parameter is converted in the style [-]d.ddde±dd, where there is one digit before the decimal point character (which is non-zero if the parameter is non-zero) and the number of digits after the decimal point character is equal to the precision. The default precision is 6. If the precision is zero and the # flag is not specified, no decimal point character appears. The value is rounded to the appropriate number of digits. The E conversion specifier produces a number with E instead of e introducing the exponent. The exponent always contains at least two digits. If the value is zero, the exponent is zero.</td>
</tr>
<tr>
<td>f</td>
<td>The double parameter is converted to decimal notation in the style [-]ddd.ddd. The number of digits after the decimal point character is equal to the precision specification. The default precision is 6. If the precision is zero and the # flag is not specified, no decimal point character appears. If a decimal point character appears, at least one digit appears before it. The value is rounded to the appropriate number of digits.</td>
</tr>
<tr>
<td>g, G</td>
<td>The double parameter is converted in the style f or e (or in style E in the case of a G conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted; style e (or E) is used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result. A decimal point character appears only if it is followed by a digit.</td>
</tr>
<tr>
<td>n</td>
<td>The parameter is a pointer to an integer into which is written the number of characters written to the output stream so far by this call to fprintf(). No parameter is converted.</td>
</tr>
<tr>
<td>o, u, x, X</td>
<td>The unsigned int parameter is converted to unsigned octal (o), unsigned decimal (u), or unsigned hexadecimal notation (x or X) in the style dddd. The letters abcdef are used for x conversion and letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits, it is expanded with leading zeros. The default precision is one. The result of converting a zero value with a precision of zero is no characters.</td>
</tr>
<tr>
<td>P</td>
<td>The parameter is a pointer to a void. The pointer value is converted to a sequence of printable characters as an unsigned hexadecimal integer.</td>
</tr>
</tbody>
</table>
### Table 2-6. fprintf() Conversion Specifiers (Continued)

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>The parameter is a pointer to an array of character type. Characters from the array are written up to, but not including, a terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array contains a null character.</td>
</tr>
</tbody>
</table>
| %    | A % is written. No parameter is converted. The complete conversion specification is %%.

If a conversion specification is invalid, the behavior is undefined.

If any parameter is, or points to, a union or an aggregate (except for an array of character type using %s conversion or a pointer using %p conversion), the behavior is undefined.

A non-existent or small field width never truncates a field. If the result of a conversion is wider than the field width, the field expands to contain the conversion result.

fprintf() returns the number of characters transmitted, or a negative value if an output error occurred.

The sclib.l and sclib.il libraries contain a smaller version of the fprintf() function. Refer to the Using Ultra C/C++ manual for more information about the small library functions.

<table>
<thead>
<tr>
<th>stream</th>
<th>is a pointer to the C I/O FILE structure. (Output)</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>is a pointer to a control string. (Input)</td>
</tr>
</tbody>
</table>

format determines the format, type, and number of parameters that the function expects. If the control string does not correctly match the parameters, the results are not predictable.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No
- **Standards:** ANSI

### Library

clib.l

### See Also

printf()  sprintf()
fputc()

Output Character to File

Syntax

```c
#include <stdio.h>
int fputc(
    int chr,
    FILE *stream);
```

Description

`fputc()` converts a character to an unsigned char. It writes the character to the output stream at the position indicated by the associated file position indicator for the stream, if defined, and advances the indicator appropriately. If the file cannot support positioning requests or the stream is opened in append mode, the character is appended to the output stream. `fputc()` returns the character written. If a write error occurs, the error indicator for the stream is set and `fputc()` returns EOF.

- `chr` is the character to convert. (Input)
- `stream` is a pointer to the C I/O `FILE` structure. (Output)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

clib.l

See Also

`fgetc()`
Syntax

```c
#include <stdio.h>
int fputs(
    const char *str,
    FILE *stream);
```

Description

`fputs()` writes a string to a stream. The terminating null character is not written. `fputs()` returns `EOF` if a write error occurs. Otherwise, it returns a non-negative value.

- `str` is a pointer to the string. (Input)
- `stream` is a pointer to the stream to write. (Output)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

`clib.l`

See Also

`fgets()`
Syntax

```c
#include <stdio.h>
int fread(
    void *ptr,
    size_t size,
    size_t nmemb,
    FILE *stream);
```

Description

`fread()` reads up to `nmemb` elements into an array. The file position indicator for the stream, if defined, is advanced by the number of characters successfully read. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate. If a partial element is read, its value is indeterminate.

`fread()` returns the number of elements successfully read, which may be less than `nmemb` if a read error or end-of-file is encountered. If `size` or `nmemb` is zero, `fread()` returns zero and the array contents and stream state remain unchanged.

- `ptr` is a pointer into the array. (Output)
- `size` specifies the size of each array element. (Input)
- `nmemb` specifies the number of array elements to read. (Input)
- `stream` is a pointer to the C I/O `FILE` structure. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

`clib.l`

See Also

`fwrite()`
Syntax

```c
#include <stdlib.h>

void free(void *ptr);
```

Description

`free()` deallocates the space pointed to by `ptr`. If `ptr` (input) is a null pointer, no action occurs. Otherwise, if the parameter does not match a pointer previously returned by `calloc()`, `malloc()`, or `realloc()`, or if the space has been deallocated by a call to `free()` or `realloc()`, the behavior is undefined.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: no
- Standards: ANSI

Library

`clib.l`

See Also

- `calloc()`
- `malloc()`
- `realloc()`
freemem(), freemem()
Determine Size of Unused Stack Area

Syntax
#include <stdlib.h>
int freemem(void);
int _freemem(void);

Description
freemem() returns the number of bytes allocated for the stack that have not been used. If compiler stack checking is enabled, the stack is checked for possible overflow before entering a function. The lowest address the stack pointer has reached is retained so freemem() can report the number of bytes between the stack limit and the lowest stack value as the unused stack memory.

The program must be compiled with stack checking code in effect for freemem() to return a correct result. freemem() is historical; avoid using it in new code as it is likely to be removed in a future release. These calls are not available when compiled in strictly conforming ANSI mode.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
cstart.r

See Also
stacksiz(), _stacksiz()
Set Memory Reclamation Bound

Syntax

#include <stdlib.h>

void _freemin(int size);

Description

_freemin() allows you to set the minimum size for a free block of memory or concatenation of free blocks for free() to return to the operating system. If a program is known to use a lot of memory, you can instruct free() never to return memory to the operating system.

You do not need to use _freemin() to return memory to the system. The default minimum size to return is 4K. _freemin() ignores any specified non-negative size smaller than 4K.

size is the minimum size for a free block of memory or the concatenation of free blocks. (Input)

- If size is positive, it is used as the minimum number of bytes for a free block of memory to be a candidate for return to the operating system.
- If size is negative, free() never tries to return memory allocated by the ANSI memory allocation functions (calloc(), malloc(), or realloc()) to the system.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

calloc()
free()
malloc()
realloc()
freopen()
Re-Open File

Syntax
#include <stdio.h>
FILE *freopen(
    const char *name,
    const char *action,
    FILE *stream);

Description
freopen() substitutes a file name for the file currently open, if any. The original file is closed, even if the opening of the new file does not succeed.

freopen() is usually used to associate the stdin, stdout, or stderr file pointers with a file instead of a terminal device. It returns the stream passed or a null pointer if the open fails.

name is a pointer to the file name to use for a substitute. (Input)

action specifies the action to take. (Input)
    Refer to fopen() for details on the action values.

stream is a pointer to the C I/O FILE structure. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.1

See Also
fopen()
frexp()  
Return Portions of Floating Point Number

**Syntax**

```c
#include <math.h>

double frexp(
    double value,
    int *exp);
```

**Description**

frexp() breaks a floating point value into a normalized fraction and an integral exponent of two. frexp() returns the fraction x, such that \( \frac{1}{2} \leq x < 1 \) and value = \( x \times 2^{\text{exp}} \).

- `value` is the floating point value. (Input)
- `exp` is a pointer to the integral exponent of two. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

**Library**

clib.1
Syntax

```c
#include <stdio.h>
int fscanf(
    FILE *stream,
    const char *format, ...);
```

Description

The `fscanf()` function reads input from a stream.

The format is a multibyte character sequence, beginning and ending in its initial shift state. The format is composed of zero or more directives:

- One or more white space characters.
- An ordinary multibyte character (neither % nor a white space character).
- A conversion specification.

A percent (%) character introduces each conversion specification. After the %, the following appear in sequence:

- An optional assignment-suppressing asterisk (*) character.
- An optional non-zero decimal integer that specifies the maximum field width.
- An optional h, l, or L indicating the size of the receiving object.

The conversion specifiers d, i, and n are preceded by h if the corresponding parameter is a pointer to a short int rather than pointing to int, by l if it is a pointer to a long int or by L if it is a pointer to long long int.

The conversion specifiers o, u, and x are preceded by h if the corresponding parameter is a pointer to an unsigned short int rather than pointing to an unsigned int, by l if it is a pointer to an unsigned long int or by L if it is a pointer to an unsigned long long int.

The conversion specifiers e, f, and g are preceded by l if the corresponding parameter is a pointer to a double rather than pointing to a float, or by L if it is a pointer to a long double.

If an h, l, or L appears with any other conversion specifier, the behavior is undefined.
• A character that specifies the type of conversion to apply. The following conversion specifiers are valid:

Table 2-7. Conversion Specifiers for fscanf()

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Matches a character sequence of the number specified by the field width. By default, the field width is one. The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence. No null character is added.</td>
</tr>
<tr>
<td>d</td>
<td>Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <code>strtol()</code>. The base parameter is 10. The corresponding parameter is a pointer to an integer.</td>
</tr>
<tr>
<td>e, f, g</td>
<td>Matches an optionally signed floating point number, whose format is the same as expected for the subject string of <code>strtod()</code>. The corresponding parameter is a pointer to a float.</td>
</tr>
<tr>
<td>i</td>
<td>Matches an optionally signed integer, whose format is the same as expected for the subject sequence of <code>strtol()</code>. The base parameter is 0. The corresponding parameter is a pointer to an integer.</td>
</tr>
<tr>
<td>n</td>
<td>No input is consumed. The corresponding parameter is a pointer to an integer into which to write the number of characters read from the input stream so far by this call to <code>fscanf()</code>. Executing a <code>%n</code> directive does not increment the assignment count returned when <code>fscanf()</code> finishes execution.</td>
</tr>
<tr>
<td>o</td>
<td>Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code>. The base parameter is 8. The corresponding parameter is a pointer to an unsigned integer.</td>
</tr>
<tr>
<td>p</td>
<td>Matches the set of sequences that may be produced by the <code>${p}</code> conversion of <code>fprintf()</code>. The corresponding parameter is a pointer to a void. The input item is interpreted the same as <code>${x}</code>. If the input item is a value converted earlier during the same program execution, the resulting pointer compares equal to that value. Otherwise, the behavior of the <code>${p}</code> conversion is undefined.</td>
</tr>
<tr>
<td>s</td>
<td>Matches a sequence of non-white space characters. The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence and a terminating null character. The terminating null character is added automatically.</td>
</tr>
<tr>
<td>u</td>
<td>Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code>. The base parameter is 10. The corresponding parameter is a pointer to an unsigned integer.</td>
</tr>
</tbody>
</table>
fscanf() executes each directive of the format in turn. If a directive fails, `fscanf()` returns. Failures are described as “input failures or matching failures.”

- An input failure is caused by the unavailability of input characters.
- A matching failure is caused by inappropriate input.

A directive composed of white space character(s) is executed by reading input up to the first non-white space character or until no more characters can be read. The first non-white space character remains unread.

A directive containing an ordinary multibyte character is executed by reading the next characters of the stream. If one character differs from one in the directive, the directive fails, and the differing and subsequent characters remain unread.

**Table 2-7. Conversion Specifiers for fscanf() (Continued)**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of <code>strtoul()</code>. The base parameter is 16. The corresponding parameter is a pointer to an unsigned integer.</td>
</tr>
<tr>
<td>[</td>
<td>Matches a non-empty character sequence from a set of expected characters called the <em>scanset</em>. The corresponding parameter is a pointer to the initial character of an array large enough to accept the sequence and a terminating null character. The terminating null character is added automatically. The conversion specifier includes all subsequent characters in the format string, up to and including the matching right bracket (}). The characters between the brackets are the <em>scanlist</em>. The <em>scanlist</em> is the <em>scanset</em>, unless the character after the left bracket is a caret (^). In this case, the <em>scanset</em> contains all characters not appearing in the <em>scanlist</em> between the ^ and the }. If the conversion specifier begins with [], the } is in the <em>scanset</em> and the next right bracket character ends the specification. Otherwise, the first right bracket character ends the specification. A hyphen (-) has no special meaning; it is matched like any other character.</td>
</tr>
<tr>
<td>%</td>
<td>Matches a single %. No conversion or assignment occurs. The complete conversion specification is %%.</td>
</tr>
</tbody>
</table>

The conversion specifiers E, G, and X are also valid and behave the same as, respectively, e, g, and x.

If a conversion specification is invalid, the behavior is undefined.
A directive containing a conversion specification defines a set of matching input sequences for each specifier. A conversion specification is executed in the following steps:

- Input white space characters (as specified by `isspace()`) are skipped, unless the specification includes a [space, c, or n specifier.

- An input item is read from the stream, unless the specification includes an n specifier. An input item is defined as the longest matching sequence of input characters, unless it exceeds a specified field width. If it exceeds the field width, it is the initial subsequence of that length in the sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails. This condition is a matching failure, unless an error prevented input from the stream. If an error prevented input from the stream, it is an input failure.

- Except in the case of a % specifier, the input item (or, in the case of a %n directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails. This condition is a matching failure. Unless assignment suppression was indicated by an asterisk (*), the conversion result is placed in the object pointed to by the first parameter following the format parameter that has not already received a conversion result. If this object does not have an appropriate type or if the conversion result cannot be represented in the space provided, the behavior is undefined.

If the end-of-file is reached during input, conversion terminates. If end-of-file occurs before any characters matching the current directive have been read, other than leading white space where permitted, execution of the current directive terminates with an input failure. Otherwise, unless execution of the current directive is terminated with a matching failure, execution of the following directive, if any, is terminated with an input failure.

If conversion terminates on a conflicting input character, the offending input character is left unread in the input stream. Trailing white space, including newline characters, is left unread unless matched by a directive. You must use the %n directive to determine the success of literal matches and suppressed assignments.

`fscanf()` returns EOF if an input failure occurs before any conversion. Otherwise, `fscanf()` returns the number of input items assigned which can be fewer than provided for, or zero in the event of an early matching failure.

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `fscanf()` function. Refer to the Using Ultra C/C++ manual for more information about the small library functions.
stream is a pointer to the C I/O FILE structure. (Input)

format is a pointer to a character sequence. (Input)

format specifies the allowable input sequences and specifies how to convert these sequences for assignments. If the format is exhausted while parameters remain, the excess parameters are evaluated, but are otherwise ignored.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also
scanf()
sscanf()
strtol()
strtoll()
strtoul()
fseek()
Reposition File Pointer

Syntax
#include <stdio.h>
int fseek(
    FILE *stream,
    long int offset,
    int place);

Description
fseek() sets the file position indicator for a stream.
fseek() returns non-zero only for a request that cannot be satisfied.

The sclib.l and sclib.il libraries contain a smaller version of the fseek() function. Refer to the Using Ultra C/C++ manual for more information about the small library functions.

stream
    is a pointer to the C I/O FILE structure. (Input)

offset
    specifies an amount to offset the file position indicator. (Input)

place
    specifies where to place the file position indicator. (Input)

place
    has three possible values:

Table 2-8. fseek() place Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEEK_SET</td>
<td>The new file position is offset.</td>
</tr>
<tr>
<td>SEEK_CUR</td>
<td>The new file position is the current file position + offset. offset can be negative.</td>
</tr>
<tr>
<td>SEEK_END</td>
<td>The new file position is the current file size + offset (usually offset is negative or zero in this case). A successful call to fseek() clears the end-of-file indicator for the stream and undoes the effects of the ungetc() function on the same stream. After an fseek() call, the next operation may be either input or output.</td>
</tr>
</tbody>
</table>
Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  No
Standards:  ANSI

Library
clib.1

See Also
ftell()
getc(), getchar()
putc(), putchar()
ungetc()
**Syntax**

```c
#include <stdio.h>
int fsetpos(
    FILE *stream,
    const fpos_t *pos);
```

**Description**

`fsetpos()` sets the file position indicator for a stream.

A successful call to `fsetpos()` clears the end-of-file indicator for the stream and undoes any effects of `ungetc()` on the same stream. After an `fsetpos()` call, the next operation on an update stream may be either input or output.

If successful, `fsetpos()` returns zero. Otherwise, it returns a non-zero value and places the appropriate error code in the global variable `errno`.

- **stream** is a pointer to the C I/O `FILE` structure. (Input)
- **pos** is a pointer to the value of the file position indicator. (Input)
- **pos** is returned from `fgetpos()`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No
- **Standards:** ANSI

**Library**

`clib.1`

**See Also**

`fgetpos()`
fstat()  
Get File Status

Syntax
#include <UNIX/stat.h>
int fstat(
    unsigned int fd,
    struct stat *buf);

Description
fstat() obtains the same information about an open file referenced by the argument descriptor, as would be obtained by a stat call.

fstat() returns zero on success. It returns -1 on failure and sets errno to indicate the error.

buf is a pointer to a stat structure into which information about the file is placed. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
unix.l

See Also
stat()
ftell()

Report File Pointer Position

Syntax

#include <stdio.h>

long ftell(FILE *stream);

Description

`ftell()` returns the current value of the file position indicator as the number of bytes from the beginning of the file for a stream. This is not necessarily a meaningful measure of the number of characters written or read.

If successful, `ftell()` returns the current value of the file position indicator for the stream. On failure, `ftell()` returns -1L and the appropriate error code is placed in the global variable `errno`.

`stream` is a pointer to the C I/O `FILE` structure. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

`fseek()`
`getc()`, `getchar()`
`putc()`, `putchar()`
fwrite()

Write Data to File

Syntax

```c
#include <stdio.h>
size_t fwrite(
    const void *ptr,
    size_t size,
    size_t nmemb,
    FILE *stream);
```

Description

fwrite() writes up to nmemb elements from an array to the stream. The file position indicator for the stream, if defined, is advanced by the number of characters successfully written. If an error occurs, the resulting value of the file position indicator for the stream is indeterminate.

fwrite() returns the number of elements successfully written. This is less than nmemb only if a write error is encountered.

A write to a text stream does not cause the associated file to truncate beyond that point.

The sclib.l and sclib.il libraries contain a smaller version of the fwrite() function. Refer to the Using Ultra C/C++ manual for more information about the small library functions.

ptr

is a pointer to the array. (Input)

size

specifies the size of each array element. (Input)

nmemb

specifies the number of array elements to write. (Input)

stream

is a pointer to the C I/O FILE structure. (Output)

Attributes

<table>
<thead>
<tr>
<th>Operating System</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>User and System</td>
</tr>
<tr>
<td>Threads</td>
<td>Safe</td>
</tr>
<tr>
<td>Re-entrant</td>
<td>No</td>
</tr>
<tr>
<td>Standards</td>
<td>ANSI</td>
</tr>
</tbody>
</table>
Library
clib.l

See Also
fwrite()
Syntax
#include <getset.h>
#include <types.h>
u_int32 _get_<name>(void);

Description
The _get_<name>() class of functions can be used to read the value of Special
Purpose Registers (SPRs), Device Control Registers (DCRs), Processor Registers
(PREGs), and Time-Based Registers (TBs).
All functions are provided. It is the programmer's responsibility to avoid using
system-state only registers from user-state and accessing registers that do not exist
on the target processor. All registers are accessible from system-state. Only those
noted are available from user-state. Attempting to use system state registers from
user state causes unpredictable results.

For information on which _get<name>() functions are available for your
processor, refer to the getset.h header file in the following location:
MWOS\OS9000\<PROCESSOR>\DEFS\getset.h

name
is any of the names listed in the following table. There is one exception to the
following table, the dec SPR on the 601 can be read from user-state with the
function _get_user_dec().

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
_set_<name>()
**getc(), getchar()**  
Get Next Character from File, Stdin

**Syntax**
```
#include <stdio.h>
int getc(FILE *stream);
int getchar(void);
```

**Description**
`getc()` returns a character, as an unsigned converted to an `int`, from the file and advances the associated file position indicator for the stream, if defined. If `getc()` is implemented as a macro, it may evaluate `stream` more than once. Therefore, the parameter should never be an expression with side effects. `getchar()` is a macro that is equivalent to `getc(stdin)`.

`getc()` returns the next character from the input stream pointed to by `stream`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `getc()` returns `EOF`. If a read error occurs, the error indicator for the stream is set and `getc()` returns `EOF`.

`getchar()` returns the next character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator for the stream is set and `getchar()` returns `EOF`. If a read error occurs, the error indicator for the stream is set and `getchar()` returns `EOF`.

`stream` is a pointer to the C I/O `FILE` structure. (Input)

**Attributes**
- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

**Library**
```
clib.l
```

**See Also**
- `fgetc()`
- `fopen()`
- `putc()`, `putchar()`
get_const()

Get Current Constant Data

Syntax

```c
#include <regs.h>
void *get_const(void);
void *get_const(void);
```

Description

get_const() and _get_const() return the current constant data pointer. These functions are not present in the libraries for a given processor if a constant data pointer is not applicable.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

get_const() - cpu.l, _get_const() - os_lib.l
**get_current_tss()**

Get Current Task Segment

**Syntax**

```c
#include <types.h>
#include <regs.h>

u_int32 *get_current_tss(void);
```

**Description**

`get_current_tss()` returns the current task segment.

```
get_current_tss() is only supported on the 80x86 family of processors version of the compiler.
```

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`cpu.l`
getcwd()  
Copy Pathname of Current Directory

Syntax
#include <unistd.h>
char *
getcwd(char *buf,
       size_t size);

Description
The getcwd() function copies the absolute pathname of the current working 
directory into the memory referenced by buf and returns a pointer to buf. The size 
argument is the size, in bytes, of the array referenced by buf. If buf is NULL, space 
is allocated as necessary to store the pathname.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
unix.l

See Also
chdir()
malloc()
getenv()  
Get Value for Environment Name

Syntax

```c
#include <stdlib.h>
char *getenv(const char *name);
```

Description

getenv() searches the environment list for a string that matches the specified name.
getenv() returns a pointer to a string associated with the matched list member. The string pointed to cannot be modified by the program. If the specified name cannot be found, a null pointer is returned.

The common set of environment names depends on the operating system. Refer to your operating system’s user manual for more information.

The environment list is implemented as a NULL terminated array of pointers to characters. Each string is in the form:

\[ <name>=<value> \]

- `name` is a pointer to the name. (Input)
- `<name>` is the name of the environment variable.
- `<value>` is the current value of the environment variable.

To modify the list, you must either find the existing entry or enlarge the array and add an entry. The base of the array is stored in the external variable `_environ`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

clib.l
See Also

_os_chain()
_os_chainm()
_os_dfork()
_os9_dfork()
_os_dforkm()
_os_exec()
_os_fork()
os9fork(), os9forkc()
_os_forkm()
Syntax
#include <UNIX/os9def.h>
unsigned int getgid();

Description
getgid() returns the group identification of the current process.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
unix \l
get_except_base()

Return Exception Table Base Address

Syntax
#include <types.h>
#include <regs.h>

u_int32 *get_except_base(void);

Description
get_except_base() returns the exception table base address.

get_except_base() is only supported on the 80x86 family of processors version of the compiler.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l
get_gdtr()
Get Global Descriptor Pointer

Syntax
#include <types.h>
#include <regs.h>
void *get_gdtr(void);

Description
get_gdtr() returns a pointer to the global descriptor.

⚠️ get_gdtr() is only supported on the 80x86 family of processors version of the compiler.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l
Syntax

```c
#include <time.h>
int getime(struct sgtbuf *timebuf);
```

Description

getime() returns the system time in a time buffer. The time units are defined in the time.h header file.

timebuf is a pointer to the time buffer. (Output)

If successful, getime() returns the buffer pointer.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

sys_clib.l

See Also

-os_getime()  
F$Time  
F_TIME  

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax

```c
#include <module.h>
int _get_module_dir(
    void *buffer,
    int count);
```

Description

 `_get_module_dir()` copies the system’s module directory into the buffer for inspection. A maximum of `count` bytes are copied. `_get_module_dir()` returns the number of bytes actually copied.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

  - `buffer` is a pointer to the buffer. (Output)
  - `count` specifies the maximum number of bytes to copy. (Input)

Attributes

- Operating System: OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.1`

See Also

- `_os_get_moddir()`
- F$GModDr

OS-9 for 68K Technical Manual
getnextpwnam()
Get Next Password File Entry for Name

Syntax
#include <UNIX/pwd.h>
struct passwd *getnextpwnam(const char *name);

Description
getnextpwnam() returns NULL if no more password file entries are found for the
name.
This function is useful for OS-9 systems where users have more than one password
file entry.

name-getnextpwnam() returns the next password file entry associated with name. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
unix.l

See Also
endpwent()
fgetpwent()
getpwent()
getpwuid()
getpwnam()
setpwent()
getopt()  
Return Next Option Letter in String

Syntax
#include <UNIX/os9def.h>
extern char *optarg;
extern int optind, opterr;
int getopt(
    int argc,
    char *const *argv,
    const char *optstring);

Description
getopt() returns the next option letter in argv that matches a letter in optstring. optstring must contain the option letters the command using getopt() recognizes. If a letter is followed by a colon, the option is expected to have an argument, or a group of arguments, which must be separated from it by white space.

optarg is set to point to the start of the option argument on return from getopt().

getopt() places in optind the argv index of the next argument to be processed. optind is external and is initialized to 1 before the first call to getopt().

When all options have been processed (up to the first non-option argument), getopt() returns -1. The special option -- may be used to delimit the end of the options. When it is encountered, -1 is returned and -- is skipped.

getopt() prints an error message on the standard error and returns a question mark (?) when it encounters an option letter not included in optstring or no option-argument after an option that expects one. This error message may be disabled by setting opterr to 0.

For example, this code fragment shows how one might process the arguments for a command that can take the mutually exclusive options a and b, and the option o, which requires an option argument:

main(argc, argv);
int argc;
char **argv;
{
    int c;
    extern char *optarg;
    extern int optind;
    .
    .
    .
while ((c = getopt(argc, argv, "abo:")) != -1)
    switch (c) {
        case 'a':
            if (bflg)
                errflg++;
            else
                aflag++;
            break;
        case 'b':
            if (aflg)
                errflg++;
            else
                bproc();
            break;
        case 'o':
            ofile = optarg;
            break;
        case '?':
            errflg++;
    }
if (errflg) {
    (void)fprintf(stderr, "usage: . . .");
    exit (2);
}
for (; optind < argc; optind++) {
    if (access(argv[optind], 4)) {
.
.
.
    }
}

Changing the value of the variable optind, or calling getopt() with different values of argv, may lead to unexpected results.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Unsafe
Re-entrant: No

Library

unix.l
Syntax
#include <process.h>
int getpid(void);

Description
getpid() returns the system process ID number for the calling process. You can use this number for tasks such as creating unique file names.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_id()
_os9_id()
_F$ID OS-9 for 68K Technical Manual
_F_ID OS-9 Technical Manual
getppid()

Return Parent Process ID Number

Syntax

```c
#include <UNIX/os9def.h>
unsigned int getppid(void);
```

Description

`getppid()` returns the system process ID number for the parent of the calling process. You can use this number for tasks such as creating unique file names.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

unix.l

See Also

`getpid()`
**Syntax**

```c
#include <process.h>
int _get_process_desc(
    int pid,
    int count,
    pr_desc *buffer);
```

**Description**

_get_process_desc() copies a process descriptor into the buffer for inspection. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

The following is an example call:

```c
procid pbuf;
_get_process_desc(pid, sizeof(pbuf), &pbuf);
```

*pid* is the process ID number of the desired process. (Input)

Only the lower 16 bits of *pid* are used.

*count* specifies the maximum number of bytes to copy. (Input)

*buffer* is a pointer to the buffer. (Output)

**Attributes**

- Operating System: OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

sys_clib.l

**See Also**

_os_gprdsc()  
F$GPrDsc  
OS-9 for 68K Technical Manual
Syntax
#include <process.h>
int _get_process_table(
    void *buffer,
    int count);

Description
_get_process_table() copies the system's process descriptor table into the buffer for inspection. A maximum of count bytes are copied. The number of bytes actually copied is returned.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

buffer
    is a pointer to the buffer. (Output)

count
    specifies the maximum number of bytes to copy. (Input)

Attributes
Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library
sys_clib.l

See Also
_os_get_prtbl()
F$GPrDBT                OS-9 for 68K Technical Manual
getpw()
Get Name from Password File

Syntax
#include <UNIX/pwd.h>
int getpw(
    int uid,
    char *buf);

Description
getpw() searches the password file for the (numerical) user identification number and fills in the buffer with the corresponding name. It returns non-zero if the \textit{uid} could not be found. The name is null-terminated.

\textbf{uid}
   is the user identification number. (Input)

\textbf{buf}
   is the buffer. (Output)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         No

Library
unix.l

See Also
eendpwent()
fgetpwent()
ggetpwent()
getpwuid()
getpwnam()
setpwent()
getpwent()

Get Password File Entry

Syntax

```c
#include <UNIX/pwd.h>
struct passwd *getpwent(void);
```

Description

`getpwent()`, `getpwuid()` and `getpwnam()` each return a pointer to an object with the following structure containing the fields of a line in the password file. Each line in the file contains a `passwd` structure, declared in the `<UNIX/pwd.h>` header file:

```c
struct passwd {
    char *pw_name;     * login name
    char *pw_passwd;   * password
    int pw_uid;        * user ID
    int pw_gid;        * group ID
    int pw_prio;       * priority
    char *pw_age;      * unsupported -zero length string
    char *pw_comment;  * unsupported -zero length string
    char *pw_gecos;    * unsupported -zero length string
    char *pw_dir;      * initial data directory
    char *pw_xdir;     * initial execution directory
    char *pw_shell;    * initial command executed
    char *pw_junk;     * undefined
};
```

The fields `pw_age`, `pw_comment`, `pw_gecos`, and `pw_junk` are unused. When first called, `getpwent()` returns a pointer to the first `passwd` structure in the file; thereafter, it returns a pointer to the next `passwd` structure in the file; so successive calls can be used to search the entire file. `getpwuid()` searches from the beginning of the file until a numerical user ID matching `uid` is found and returns a pointer to the particular structure in which it was found. `getpwnam()` searches from the beginning of the file until a `login` name matching `name` is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to `setpwent()` has the effect of rewinding the password file to allow repeated searches. `endpwent()` may be called to close the password file when processing is complete.

`fgetpwent()` returns a pointer to the next `passwd` structure in the stream `f`, which matches the format of the password file `/dd/SYS/password`. 
getpwent(), getpwuid(), and getpwnam() return a pointer to struct passwd on success. On EOF or error, or if the requested entry is not found, they return NULL.

- The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected.
- All information is contained in a static area that is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

**Library**

unix.l

**See Also**

- endpwent()
- getpwent()
- getpw()
- getpwuid()
- getpwnam()
- setpwent()
getpwnam()

Get Password File Name

Syntax

```c
#include <UNIX/pwd.h>
struct passwd *getpwnam(const char *nam);
```

Description

`getpwnam()` returns a pointer to an object with the structure shown in `getpwent()` containing the fields of a line in the password file that matches the name `nam`. Each line in the file contains a `passwd` structure, declared in the `<UNIX/pwd.h>` header file.

The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected. All information is contained in a static area which is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

`nam` is the name matching the fields in the password file. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

`unix.l`

See Also

`endpwent()`
`fgetpwent()`
`getpw()`
`getpwent()`
`getpwuid()`
`setpwent()`
getpwuid()

Get Password File User ID

Syntax

```c
#include <UNIX/pwd.h>
struct passwd *getpwuid(unsigned int uid);
```

Description

getpwuid() returns a pointer to an object with the structure shown in getpwnam() containing the fields of a line in the password file that matches the specified user identification. Each line in the file contains a passwd structure, declared in the <UNIX/pwd.h> header file.

⚠️ The above routines use the standard I/O library, which increases the size of programs not otherwise using standard I/O more than might be expected. All information is contained in a static area that is overwritten by subsequent calls to these functions, so it must be copied if it is to be saved.

uid

is the specified user identification. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

unix.l

See Also

- endpwent()
- fgetpwent()
- getpw()
- getpwnam()
- setpwent()
gets()
Get String from File

Syntax
#include <stdio.h>
char *gets(char *str);

Description
gets() reads characters from the standard input file into an array until end-of-file is encountered or a newline character is read. Any newline character is discarded, and a null character is written immediately after the last character read into the array.

gets() returns str if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

str is a pointer to the array. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also
fgets()
getstat() Get File Status

Syntax

```c
#include <sgstat.h>
#include <sg_codes.h>

int getstat(
    int code,
    int path,
    struct sgbuf *buffer); * code = 0

int getstat(
    int code,
    int path); * code = 1 or 6

int getstat(
    int code,
    int path,
    long *size); * code = 2

int getstat(
    int code,
    int path,
    long *pos);  * code = 5
```

Description

getstat() is historical from the 6809 C Compiler. It accesses a few I$GetStt system functions.

When getstat() returns with the value -1, the appropriate error code is placed in the global variable errno.

getstat() is supported for the convenience of programs ported from the 6809.
getstat() is defined as shown in Table 2-9. (Input)

Table 2-9. getstat() Code Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>buffer is the address of the 128 byte buffer into which the path option bytes are copied. The <code>sgstat.h</code> header file contains a struct defined for use by the program.</td>
</tr>
<tr>
<td>1</td>
<td>This applies only to SCF devices and is used to test for data ready. The return value is the number of bytes available or -1 if an error occurred.</td>
</tr>
<tr>
<td>2</td>
<td>size is the address of the long integer into which the current file size is placed. The function returns -1 on error and zero on success.</td>
</tr>
<tr>
<td>5</td>
<td>pos is the address of the long integer into which the current file position is placed. The function returns -1 on error and zero on success.</td>
</tr>
<tr>
<td>6</td>
<td>The function returns -1 on either EOF or error, and 0 if not at EOF.</td>
</tr>
</tbody>
</table>

Only the lower 16 bits of code are used.

path must be the path number of an open file. (Input)

buffer is the address of the 128 byte buffer into which the path option bytes are copied. (Output)

size is the address of the long integer into which the current file size is placed. (Output)

pos is the address of the long integer into which the current file position is placed. (Output)

Attributes
- Operating System: OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library
- sys_clib.l
See Also

_os_gs_devnm()
=os_g_eof()
_os_g_fd()
_os_g_fdinf()
_os_g_popt()
_os_g_pos()
_os_g_ready()
_os_g_size()
I$GetStt
get_static()

Return Current Static Storage Pointer

Syntax

```c
#include <regs.h>
void *get_static(void);
void *_get_static(void);
```

Description

`get_static()` and `_get_static()` return a pointer to the current static storage.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

- `get_static()` - cpu.l, `_get_static` - os_lib.l

See Also

`change_static()`
get_sysglobs()

Return a Pointer to the System Global Variables

Syntax

```c
#include <types.h>
#include <regs.h>
#include <const.h>
#include <stdef.h>
#include <sysglob.h>

void *get_sysglobs(void);
```

Description

get_sysglobs() returns a pointer to the system global variables defined in sysglob.h.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: System
- Threads: Safe
- Re-entrant: Yes

Library

cpu.l

See Also

- sysglob.h
- _os_getsys()
- _os_setsys()
Syntax

```c
#include <sysglob.h>
int _getsys(
    int glob,
    int size);
```

OS-9 for 68K users need to include `setsys.h` also.

Description

`_getsys()` examines a system global variable. These variables are defined in `setsys.h` for C programs. Each of these variables begin with a `D_` prefix.

`_getsys()` returns the value of the variable if the examine request succeeds. If the request fails, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`glob`

is the system global variable to examine. (Input)

- Only the lower 16 bits of `glob` are used.

`size`

is the size of the variable. (Input)

Attributes

Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`sys_clib.l`

See Also

`_os_getsys()`
`_os_setsys()`
F$GetSys
F_GetSys

OS-9 for 68K Technical Manual
OS-9 Technical Manual
gettimeofday()
Get Current Time

Syntax
#include <UNIX/os9time.h>
int gettimeofday(
    struct timeval *tp,
    struct timezone *tzp);

Description
The system's notion of the current time is obtained with the gettimeofday() call. The current time is expressed in elapsed seconds and microseconds since January 1, 1970 (zero hour). The resolution of the system clock is hardware dependent. The time may be updated continuously, or in "ticks." gettimeofday() returns zero (0) on success and -1 on failure and sets errno to indicate the error.

tp
points to a timeval structure, which includes the following members:
long tv_sec;    * seconds since Jan. 1, 1970
long tv_usec;   * and microseconds
If tp is a NULL pointer, the current time information is not returned.

tzp
is not supported by unix.l. (Input)

Attributes
Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library
unix.l
getuid()  
Determine User ID Number

Syntax
#include <process.h>
int getuid(void);

Description
getuid() returns the group/user ID of the current process. The upper word (two bytes) of the value is the group number; the lower word is the user number.
If unsuccessful, -1 is returned and the appropriate error code is placed in the global variable errno.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_id()
_os9_id()
F$ID  OS-9 for 68K Technical Manual
F_ID  OS-9 Technical Manual
Syntax
#include <stdio.h>
int getw(FILE *stream);

Description
getw() returns a word from a file converted to an integer. It returns -1 on error. Therefore, use the macros feof() and ferror() to check the success of getw().

getw() is a machine-dependent function because the size of a word varies from machine to machine. getw() assumes no particular alignment in the file. getw() reads two bytes from the file and sign-extends them to four bytes.

stream is a pointer to the C I/O FILE structure. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
sys_clib.1

See Also
feof()
ferror()
getwd()

Get Working Directory Path Name

Syntax

```
#include <UNIX/os9def.h>
char *getwd(char pathname[MAXPATHLEN]);
```

Description

`getwd()` copies the absolute pathname of the current working directory to `pathname` (output) and returns a pointer to the result.

`getwd()` returns NULL and global variable `errno` if an error occurs.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

`unix.l`
glob(), globfree()  
Implement Rules for File Name Pattern Matching

Syntax

```c
#include <UNIX/glob.h>

int glob(
    const char *pattern,
    int flags,
    const int (*errfunc)(const char *, int),
    glob_t *pglob);

void globfree(
    glob_t *pglob);
```

Description

The `glob()` function is a pathname generator that implements the rules for file name pattern matching used by the shell. The include file `glob.h` defines the structure type `glob_t`, which contains at least the following fields:

```c
typedef struct {
    int gl_pathc; /* count of total paths so far */
    int gl_matchc; /* count of paths matching pattern */
    int gl_offs; /* reserved at beginning of gl_pathv */
    int gl_flags; /* returned flags */
    char **gl_pathv; /* list of paths matching pattern */
} glob_t;
```

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

`unix.l`
gmtime()
Convert Calendar Time to Greenwich Mean Time

Syntax
#include <time.h>
struct tm *gmtime(time_t *tp);

Description
gmtime() converts the calendar time contained in a time structure to Coordinated Universal Time (UTC). gmtime() returns a pointer to that object or a null pointer if UTC is not available.
gmtime() returns a pointer to a static area which may be overwritten. To ensure data integrity, use the structure or save it immediately.

tp is a pointer to the time structure. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also
mktime() (for more information on the time structure tP)
time()
Syntax

```c
#include <time.h>
int _gregorian(int *time, int *date);
```

Description

`_gregorian()` converts the time and date from Julian format to the Gregorian equivalents.

`_gregorian()` modifies the values to the Gregorian format:

**Figure 2-2. _gregorian() Output Date and Time Format**

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time:</td>
<td>0</td>
<td>Hour (0-23)</td>
<td>Minute</td>
</tr>
<tr>
<td>Date:</td>
<td>Year (two bytes)</td>
<td>Month</td>
<td>Day</td>
</tr>
</tbody>
</table>

If successful, `_gregorian()` returns zero. Otherwise, -1 is returned.

Be careful when passing pointers for the `date` and `time` values.

`time` and `date`

are pointers to the Julian time and date values. (Input/Output)

The values must be in the following format:

**Figure 2-3. _gregorian() Input Date and Time Format**

- **Time:** Seconds Since Midnight (0-86399)
- **Date:** Julian Day Number

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
_gregorian()

Library
sys_clib.l

See Also
_os_julian() (OS-9 for 68K)
_os_gregorian() (OS-9 for 68K)
**Syntax**

```c
#include <sg_codes.h>
int _gs_devn(int path, char *buffer);
```

**Description**

`_gs_devn()` allows you to determine the name of the device open on a path. If the path number is invalid, -1 is returned and the appropriate error code is placed in the global variable `errno`.

You must reserve a buffer of at least 32 bytes to receive the device name. Some networked devices using NFM should have a buffer size of 128 or 256 bytes for the `_gs_devn()`.

- **path**
  - is the path number of an open path. (Input)

- **buffer**
  - is a pointer to a character array into which the null-terminated device name is placed. (Output)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`sys_clib.l`

**See Also**

- `_os_gs_devnm()`
- I$GetStt
- I_GETSTAT, SS_DEVNAME

`OS-9 for 68K Technical Manual`

`OS-9 Technical Manual`
_gs_eof()

Check for End of File

Syntax
#include <sg_codes.h>
int _gs_eof(int path);

Description
(gs_eof) determines if the file open on path is at the end of file. If it is at the end of file, 1 is returned. Otherwise, 0 is returned.

(gs_eof) cannot determine end-of-file on SCF devices. SCF devices return an EOS_EOF error when the EOF character is read. Do not use this call if you are using the buffered I/O facility on the path. Instead, use feof().

path specifies the path number of the open file. (Input)
If path is invalid, -1 is returned and the appropriate error code is placed in the global variable errno.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
feof()
_os_gs_eof()
I$GetStt
I_GETSTAT
SS_EOF
OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
Get File Descriptor Sector

Syntax
#include <rbf.h>
int _gs_gfd(
    int path,
    struct fd *buffer,
    int count);

Description
 gs_gfd() places a copy of the RBF file descriptor sector of the file open on path into a buffer. The structure fd, declared in the rbf.h header file, provides a convenient means to access the file descriptor information.

Be sure the buffer is large enough to hold count bytes. A file descriptor is one sector in length and a sector may be between 256 bytes and 32K in size. If an entire descriptor is desired, you must first find the sector size on the device and allocate a buffer. The sector size can be found in the path descriptor’s option section. See _os_gs_popt() and your operating system technical manual for more information. Generally, the struct fd in rbf.h is only large enough to hold a 256-byte file descriptor.

This call is effective only on RBF devices.

path
    specifies the path number of the open file. (Input)

buffer
    is a pointer to the buffer in which to place the file descriptor information. (Output)

count
    specifies the maximum number of bytes to copy. (Input)

    If an error occurs, -1 is returned and the appropriate error value is placed in the global variable errno.

Attributes
Operating System:       OS-9 and OS-9 for 68K
State:                 User and System
Threads:               Safe
Re-entrant:            Yes

Library
sys_clib.l

Ultra C Library Reference 277
See Also

_os_gs_fd()
_os_gs_popt()
_os_ss_fd()

I$GetStt  OS-9 for 68K Technical Manual
I_GETSTAT, SS_FD  OS-9 Technical Manual
**Get Path Options**

**Syntax**
```
#include <sg_codes.h>
#include <sgstat.h>
int _gs_opt(
    int path,
    struct sgbuf *buffer);
```

**Description**

_gs_opt() copies the options section of the path descriptor open on path into a buffer. sgbuf provides a convenient means to access the individual option values. sgbuf is declared in the sgstat.h header file.

If the path is invalid, -1 is returned and the appropriate error code is placed in the global variable errno.

Be sure the buffer is large enough to hold the options. Declaring the buffer as type struct sgbuf is safe because this structure is predefined to be large enough for the options.

If you use _gs_opt() on OS-9, you must use the OS-9 for 68K header file. _gs_opt() is provided on OS-9 solely for compatibility with OS-9 for 68K. If the OS-9 for 68K header file is unavailable, change the source code to use _os_gs_popt() or _os_gs_luopt().

path

is the path number of the open path descriptor. (Input)

buffer

is a pointer to the buffer. (Output)

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

**Library**

sys_clib.l
**See Also**

_[os_gs_popt()](/libref/os_gs_popt)_,
_[os_ss_popt()](/libref/os_ss_popt)_,
_I$GetStt_ [OS-9 for 68K Technical Manual]
_I$GETSTAT_ [OS-9 Technical Manual]
_SS_PATHOPT_ [OS-9 Technical Manual]
_tmode_ [Using OS-9 for 68K or Using OS-9]
`_gs_pos()`

Get Current File Position

**Syntax**

```c
#include <sg_codes.h>

int _gs_pos(int path);
```

**Description**

`_gs_pos()` returns the value of the file pointer for the file open on `path`. This call is effective only on RBF devices. Do not use this call if buffered I/O is being performed on the path; instead use `ftell()`.

- `path` is the path number of the open file. (Input)
- If the path is invalid or the device is not an RBF device, -1 is returned and the appropriate error code is placed in the global variable `errno`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`sys_clib.l`

**See Also**

- `ftell()`
- `_os_gs_pos()`
- `I$GetStt` (*OS-9 for 68K Technical Manual* )
- `I_GETSTAT, SS_POS` (*OS-9 Technical Manual*)
_gs_rdy()

Test for Data Available

Syntax

```c
#include <sg_codes.h>
int _gs_rdy(int path);
```

Description

_gs_rdy() checks the device open on path to see if data is waiting to be read. Read requests to the operating system wait until enough bytes have been read to satisfy the specified byte count, thereby suspending the program until the read is satisfied.

_gs_rdy() can determine the number of bytes, if any, waiting to be read, and then issue a read request for only the number of bytes actually available.

This call is effective only on RBF, SCF, or pipe devices. _gs_rdy() is not intended for use with the buffered I/O calls such as getc(); unpredictable results may occur. Use low-level functions when using _gs_rdy().

path is the path number of the open file. (Input)

- If the path is invalid or no data is available to be read, -1 is returned, and the appropriate error code is placed in the global variable errno. Otherwise, the number of bytes available to be read is returned.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

sys_clib.l

See Also

_os_gs_ready()
_os_read()
_os_readln()
I$GetStt
I_GETSTAT
SS_READY

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
**_gs_size()**

Get Current File Size

**Syntax**

```c
#include <sg_codes.h>
int _gs_size(int path);
```

**Description**

 `_gs_size()` determines the current size of the file open on `path`. This call is effective only on RBF and PIPEMAN devices.

`path` is the path number of the open file. (Input)

If the path is invalid or the device is not an RBF device, -1 is returned and the appropriate error code is placed in the global variable `errno`. Otherwise, the file size is returned.

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

**Library**

sys_clib.l

**See Also**

- `_os_gs_size()`
- I$GetStt "OS-9 for 68K Technical Manual"
- I_GETSTAT "OS-9 Technical Manual"
- SS_SIZE "OS-9 Technical Manual"
Syntax

#include <math.h>

double hypot(double x, double y);

Description

hypot() returns the Euclidean distance function:

\[ \sqrt{x^2 + y^2} \]

hypot() ignores overflows.

x and y are inputs.

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:             User and System
Threads:           Safe
Re-entrant:        Yes

Library

sys_clib.l
### Syntax

```c
#include <stdlib.h>
char *ibrk(unsigned size);
```

### Description

`ibrk()` returns a pointer to a memory block. The returned pointer is aligned to a word boundary. The memory from which `ibrk()` grants requests is the area between the end of the data allocation and the stack:

![Figure 2-4. ibrk() Memory Stack Diagram](image)

If the requested size would cause the `ibrk` area to cross the stack pointer, the request fails. You can use `freemem()` to determine the amount of stack remaining which is also the remaining `ibrk` area.

`ibrk()` is useful to request memory from a fixed-size area of memory, unlike `ebrk()` whose available memory is that of the entire system. The C I/O library functions request the first 2K of I/O buffers from this area, the remainder from `ebrk()`.

If successful, `ibrk()` returns zero. Otherwise, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

Be very careful not to crowd out the stack with `ibrk()` calls. When stack checking is in effect, the program aborts with a ***Stack Overflow*** message if insufficient stack area exists to call a function.

`size`

specifies the size, in bytes, of the memory block. (Input)
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
ebrk()
_freemem(), freemem()
sbrk()
stacksiz(), _stacksiz()
inc()
Get Character From I/O Address

Syntax
#include <regs.h>
#include <types.h>
uchar inc(void *address);

Description
inc() returns the character indicated by the specified I/O address on the 80x86
processors or the memory address on all other processors.

address
is the specified I/O address. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
inl()
inw()
outc()
outl()
outw()
Search for Character in String

Syntax

```c
#include <strings.h>
char *index(
    const char *ptr,
    int ch);
```

Description

`index()` returns a pointer to the first occurrence of a character in a string. If the character is not found, `index()` returns NULL.

- **ptr** is a pointer to the string. (Input)
- **ch** is the character for which to search. (Input)
  
  Only the lower 8 bits of **ch** are used.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

sys_clib.l

Example

This example looks for a period (.) in string and sets the pointer **ptr**. Note that **ch** is a character, not a pointer to a character:

```c
func()
{
    char *ptr,*string;
    if((ptr = index(string,'.'))
        process(ptr);
    else printf("No \'.\' found!\n");
}
```

See Also

- rindex()
- strchr()
- strrchr()
Get Integer from I/O Address

**Syntax**

```c
#include <regs.h>
#include <types.h>

u_int32 inl(void *address);
```

**Description**

`inl()` returns the integer indicated by the specified I/O address on the 80x86 processors or the memory address on all other processors. 

`address` is the specified I/O address. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

cpu.l

**See Also**

- `inc()`
- `inw()`
- `outc()`
- `outl()`
- `outw()`
**Syntax**

```c
#include <signal.h>

int intercept(void (*icpthand)(int));
```

**Description**

`intercept()` instructs the operating system to pass control to a signal handler function when the process receives a signal.

A program does not receive an abort or quit signal from the keyboard (usually ^C and ^E) unless the program performs output to the terminal. This is because the operating system sends the abort/quit signals to the last process to perform I/O to the terminal. If you run the program from the shell and type ^E before the program performs I/O to the terminal, the shell receives the signal and kills the running program. If a program requires immediate control of the terminal, perform some I/O to one of the standard paths such as printing a program banner or getting the terminal options with `_gs_opt()`.

Signal handlers dealing with asynchronous signals should be careful about the following:

- Any I/O using the C library (for example, `printf()`) cannot be performed inside both the intercept handler function and the main program. Consequently, avoid I/O within intercept functions.
- Calling any function that may be working with static storage structures.
- Calling any function that is not re-entrant.
- Modifying any static storage structure other than a variable of type `volatile sig_atomic_t`.

Generally, `intercept()` should not be used with `signal()`. Only signals that are under `SIG_DFL` handling reach a handler installed with this function.

⚠️ You cannot use floating point math within the intercept handler function.

`icpthand` is a pointer to the function.

If the signal handler function declares an `int` parameter, the function has access to the value of the received signal. On return from the signal handler function, the process resumes at the point in the program where it was interrupted by the signal.

If `icpthand` is zero, the signal handler is removed.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_intercept()
_os_send()
_os_sigmask()
signal()
F$ICPT OS-9 for 68K Technical Manual
F$Send OS-9 for 68K Technical Manual
F$SigMask OS-9 for 68K Technical Manual
F_ICPT OS-9 Technical Manual
F_SEND OS-9 Technical Manual
F_SIGMASK OS-9 Technical Manual
Example

As an example, this program is designed to clean up work files and exit when it receives a keyboard quit signal (signal 2 which, normally, is caused by typing ^E on the terminal):

```c
#include <stdio.h>

char *tempname = "tempfile";
FILE *tempfp;
int quittime = 0;

/* the signal handler */
gotsignl(signum)
int signum;
{
    switch(signum) {
        case 2:                * the quit signal
            quittime = 1;
            break;
        default:               * ignore all others
            break;
    }
}

main()
{
    if((tempfp = fopen(tempname,"w") == NULL)
        exit(_errmsg(errno,
            "can't open file - %s
",tempname));
    intercept(gotsignl);
    do {
        do_work();
    } while(!quittime);
    fclose(tempfp);
    unlink(tempname);
    exit(_errmsg(1,"quittin' time!!
"));
}
```
ioctl()

Control Device

Syntax

```c
#include <UNIX/ioctl.h>
#include <UNIX/os9def.h>

int ioctl(
    unsigned int fd,
    unsigned int request,
    caddr_t arg);
```

Description

`ioctl()` performs some of the UNIX-like I/O control functions.

The two functions that are supported are:

- **FIONREAD** — Return the number of bytes available for reading from the path. The number of bytes is returned at the `unsigned long` pointed to by `arg`. `FIONREAD` works with RBF, SCF, Pipe, and Sockman.

- **FIONBIO** — Set the I/O blocking status on the path. `arg` is a pointer to 0 (blocking I/O) or 1 (non-blocking I/O). `FIONBIO` works with Sockman (only `path_opts` block type it gets and checks for).

`ioctl()` returns zero on success for most requests. Some specialized requests may return non-zero values on success. See the specific description for the request. On failure, `ioctl()` returns -1 and sets `errno` to indicate an error.

- **fd**
  - is the path to the device that is affected by the function. (Input)

- **request**
  - is the function to perform. (Input)

- **arg** has various meanings depending on the function to be performed. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`unix.l`
inw()  
Get Word from I/O Address

Syntax
#include <regs.h>
#include <types.h>
uint16 inw(void *address);

Description
inw() returns the word indicated by the specified I/O address on the 80x86 processors or the memory address on all other processors.

address is the specified address. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
inc()
inl()
outc()
outl()
outw()
Syntax
#include <regs.h>
#include <types.h>
status_code irq_change(status_code newval);

Description
irq_change() sets the interrupt level to newval (input). The next time irq_change() is called, it returns the previous interrupt level.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
irq_disable()
irq_enable()
irq_maskget()
irq_restore()
irq_save()
Syntax
#include <regs.h>
void irq_disable(void);

Description
irq_disable() masks interrupts to the highest interrupt level available on the CPU. It is a system-state-only system call.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
irq_change()
irq_enable()
irq_maskget()
irq_restore()
irq_save()
Syntax
#include <regs.h>
void irq_enable(void);

Description
irq_enable() unmasks all interrupts to the lowest interrupt level available on the CPU.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
irq_change()
irq_disable()
irq_maskget()
irq_restore()
irq_save()
Syntax
#include <regs.h>
status_code irq_maskget (void);

Description
irq_maskget() masks all interrupts to the highest level available on the CPU. 
irq_maskget() returns the interrupt level before masking.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
irq_change()
irq_disable()
irq_enable()
irq_restore()
irq_save()
Syntax
#include <regs.h>
void irq_restore(status_code stat_reg);

Description
_irq_restore() sets the interrupt level.

stat_code specifies the value to put in the processor status register. (Input)

Attributes
Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
_irq_change()
_irq_disable()
_irq_enable()
_irq_maskget()
_irq_save()
Syntax

#include <regs.h>
status_code irq_save(void);

Description

irq_save() returns the current value of the status register.

Attributes

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library

cpu.l

See Also

irq_change()
irq_disable()
irq_enable()
irq_maskget()
irq_restore()
isalnum()
See If Parameter Is Alphanumeric

Syntax
#include <ctype.h>
int isalnum(int c);

Description
isalnum() returns a non-zero value for any character for which isalpha() or
digit() is true.

This routine, available as a function or a macro in <ctype.h>, does not operate well
with values outside the range of an unsigned char.

The function checks for invalid input (values that are not representable as an
unsigned char and not equal to the value of the macro EOF) and returns 0 for such
input. The macro does not check for invalid input, which makes it faster than the
function. The macro is the default. To use the function you must explicitly use the
#define preprocessor directive or ensure that the macro expansion does not occur by
placing the function name within parentheses.

For example: b = (isalnum)(c);

c is the character to test. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also
isalpha() iscntrl()
digit() isgraph()
islower() isprint()
ispunct() isspace()
isupper() isxdigit()
isalpha()  
See If Parameter Is Alphabetic

Syntax
#include <ctype.h>
int isalpha(int c);

Description
isalpha() returns a non-zero value for any character for which isupper() or islower() is non-zero. isalpha() returns non-zero only for the characters for which isupper() or islower() is non-zero.

This routine, available as a function or a macro in ctype.h, does not operate well with values outside the range of an unsigned char.

The function checks for invalid input (values that are not representable as an unsigned char and not equal to the value of the macro EOF) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the undef preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: b = (isalpha)(c);

    c

    is the character to test. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also
isalnum() iscntrl()
isdigit() isgraph()
islower() isprint()
ispunct() isspace()
isupper() isxdigit()
See If Parameter Is ASCII

Syntax

```c
#include <ctype.h>
int _isascii(int c);
int isascii(int c);
```

Description

_isascii() returns a non-zero value if \( c \) is an ASCII character (the value is in the range 0x00 through 0x7f). Otherwise, it returns zero. Only _isascii() is available in strictly conforming ANSI mode. Otherwise, both are available. _isascii() and isascii() are macro definitions.

This routine, available as a function or a macro in ctype.h, does not operate well with values outside the range of an unsigned char.

The function checks for invalid input (values that are not representable as an unsigned char and not equal to the value of the macro EOF) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the undef preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```c
b = (isascii)(c);
```

c is the character to test. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

cctype.h

See Also

| isalnum() | isalpha() |
| iscntrl() | isdigit() |
| isgraph() | islower() |
| isprint() | ispunct() |
| isspace() | isupper() |
| isxdigit() |
isatty()  
Determine if a Path is Sequential Character

Syntax
#include <UNIX/os9def.h>
int isatty(int fd);

Description
isatty() returns 1 if the path fd is associated with a SCF device. Otherwise, it returns zero.

fd
is the path to test. (Input)

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes

Library
unix.l
iscntrl()  
See If Parameter Is Control Code

**Syntax**

```c
#include <ctype.h>
int iscntrl(int c);
```

**Description**

iscntrl() is a C macro that returns a non-zero value for ASCII characters with values from 0 to 0x1f and 0x7f.

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an unsigned char.

The function checks for invalid input (values that are not representable as an unsigned char and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```c
b = (iscntrl)(c);
```

`c` is the character to test. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

`ctype.h`

**See Also**

- `isalnum()`
- `isalpha()`
- `isdigit()`
- `isgraph()`
- `islower()`
- `isprint()`
- `ispunct()`
- `isspace()`
- `isupper()`
- `isxdigit()`
is\textsf{digit}()  

See If Parameter Is Digit

Syntax

\begin{verbatim}
#include <ctype.h>
int isdigit(int c);
\end{verbatim}

Description

\textsf{isdigit()} is a C macro that returns a non-zero value for any decimal-digit character.

This routine, available as a function or a macro in \texttt{ctype.h}, does not operate well with values outside the range of an \texttt{unsigned char}.

The function checks for invalid input (values that are not representable as an \texttt{unsigned char} and not equal to the value of the macro \texttt{EOF}) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the \texttt{undef} preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: \begin{verbatim}
b = (isdigit)(c);
\end{verbatim}

\texttt{c}

is the character to test. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User and System  
Threads: Safe  
Re-entrant: Yes  
Standards: ANSI

Library

\texttt{ctype.h}

See Also

\begin{verbatim}
isalnum()  isalpha()  
iscntrl()  isgraph()  
islower()  isprint()  
isupper()  isspace()  
isxdigit()
\end{verbatim}
_iseuc_kana()
Check Parameter for Kaka-Kana Character

Syntax
#include <ctype.h>
int _iseuc_kana(int c);

Description
This function checks to see if the parameter is the first byte of a half- width kaka- kana character. _iseuc_kana() is a C macro that returns a non-zero value if c is the first byte of a half-width Kata-Kana character. Otherwise, it returns zero. The first byte of a half-width Kata-Kana character is 0X8E.

This routine, available as a function or a macro in ctype.h, does not operate well with values outside the range of an unsigned char.

The function checks for invalid input (values that are not representable as an unsigned char and not equal to the value of the macro EOF) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the undef preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: b = (iseuc_kana)(c);

    c
      is the character to test. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
ctype.h

See Also
isalnum() isalpha()
iscntrl()isdigit()
isgraph() islower()
isprint() ispunct()
isspace() isupper()
isxdigit()
Syntax
#include <ctype.h>
int _iseuc1();

Description
_iseuc1() is a C macro that returns a non-zero value if c is the first byte of an EUC packed format JIS x 0208-1990 character. Otherwise, it returns zero. The valid range is between 0xa1 and 0xfe.
_iseuc2() is the same thing for the second byte of the character and has the same valid range.

This routine, available as a function or a macro in ctype.h, does not operate well with values outside the range of an unsigned char.

The function checks for invalid input (values that are not representable as an unsigned char and not equal to the value of the macro EOF) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the undef preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: b = (_iseuc1)(c);

c is the character to test. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
ctype.h

See Also
isalnum() isalpha()
iscntrl() isdigit()
isgraph() islower()
isprint() ispunct()
ispace() isupper()
isxdigit()
isgraph()

See If Parameter Is Printing Character

Syntax

```c
#include <ctype.h>
int isgraph(int c);
```

Description

`isgraph()` is a C macro that returns a non-zero value if `c` is any printing character except a space. Otherwise, it returns zero.

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```c
b = (isgraph)(c);
```

`c` is the character to test. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`ctype.h`

See Also

- `isalnum()`, `isalpha()`
- `iscntrl()`, `isdigit()`
- `islower()`, `isprint()`
- `ispunct()`, `isspace()`
- `isupper()`, `isxdigit()`
See If Parameter Is Hankaku-kana

Syntax

```c
#include <ctype.h>

int _isjis_kana(int c);
```

Description

_isjis_kana() is a C macro that returns a non-zero value if `c` is a Hankaku-Kana. Otherwise, it returns zero. The valid range is between `0xa1` and `0xdf`.

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: `b = (isjis_kana)(c);`

`c` is the character to test. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`ctype.h`

See Also

`isalnum()` `isalpha()`
`iscntrl()` `isdigit()`
`isgraph()` `islower()`
`isprint()` `ispunct()`
`isspace()` `isupper()`
`isxdigit()`
islower()

See If Parameter Is Lowercase

Syntax

```c
#include <ctype.h>
int islower(int c);
```

Description

`islower()` is a C macro that tests for lowercase letters. `islower()` returns a non-zero value for ASCII characters from `0x61` (a) to `0x7a` (z).

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```
b = (islower)(c);
```

`c` is the character to test. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`ctype.h`

See Also

- `isalnum()`
- `isalpha()`
- `iscntrl()`
- `isdigit()`
- `isgraph()`
- `isprint()`
- `ispunct()`
- `isspace()`
- `isupper()`
- `isxdigit()`
isprint()
See If Parameter Is Printable Character

Syntax
#include <ctype.h>
int isprint(int c);

Description
isprint() is a C macro that tests for printing characters. It returns a non-zero value for ASCII characters with values from 0x20 to 0x7e.

This routine, available as a function or a macro in ctype.h, does not operate well with values outside the range of an unsigned char.

The function checks for invalid input (values that are not representable as an unsigned char and not equal to the value of the macro EOF) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the undef preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example: b = (isprint)(c);

c is the character to test. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
ctype.h

See Also
isalnum() isalpha()
iscntrl() isdigit()
isgraph() islower()
ispunct() isspace()
isupper() isxdigit()
ispunct()  
See If Parameter Is Punctuation Character

Syntax
#include <ctype.h>
int ispunct(int c);

Description
ispunct() is a C macro that returns a non-zero value for any printing character that is neither a space nor a character for which isalnum() is non-zero.

This routine, available as a function or a macro in ctype.h, does not operate well with values outside the range of an unsigned char. The function checks for invalid input (values that are not representable as an unsigned char and not equal to the value of the macro EOF) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the undef preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:  \( b = (\text{ispunct})(c); \)

\( c \)

   is the character to test. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
ctype.h

See Also
isalnum() isalpha()
iscntrl() isdigit()
isgraph() islower()
isprint() isspace()
isupper() isxdigit()
See If Parameter Is First Byte of SJIS Kanji

Syntax

```
#include <ctype.h>
int __issjis1(int c);
```

Description

`__issjis1()` is a C macro that returns a non-zero value if `c` is the first byte of a Shift-JIS (Japanese Industrial Standard) Kanji character. Otherwise, it returns zero. The valid ranges for Shift-JIS first bytes are between `0x81` and `0x9f` or between `0xe0` and `0xfc`.

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```
b = (__issjis1)(c);
```

c is the character to test. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`ctype.h`

See Also

`isalnum()  isalpha()`
`iscntrl()  isdigit()`
`isgraph()  islower()`
`isprint()  ispunct()`
`__issjis2()  isspace()`
`isupper()  isxdigit()`
See If Parameter Is Second Byte of SJIS Kanji

Syntax

```c
#include <ctype.h>
int _issjis2(int c);
```

Description

 `_issjis2()` is a C macro that returns a non-zero value if `c` is the second byte of a Shift-JIS (Japanese Industrial Standard) Kanji character. Otherwise, it returns zero. The valid range for Shift-JIS second bytes are between 0x40 and 0xfc.

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`. The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```c
b = (issjis2)(c);
```

`c` is the character to test. (Input)

`c` cannot equal 0x7f.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`ctype.h`

See Also

`isalnum()`  `isalpha()`
`iscntrl()`  `isdigit()`
`isgraph()`  `islower()`
`isprint()`  `ispunct()`
`_issjis1()`  `isspace()`
`isupper()`  `isxdigit()`
### isspace()

#### See If Parameter Is White Space

**Syntax**

```c
#include <ctype.h>

int isspace(int c);
```

**Description**

`isspace()` is a C macro that returns a non-zero value for standard white space characters.

The standard white space characters are:

<table>
<thead>
<tr>
<th>Description</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>\f</td>
</tr>
<tr>
<td>Form feed</td>
<td>\n</td>
</tr>
<tr>
<td>Newline</td>
<td>\r</td>
</tr>
<tr>
<td>Carriage return</td>
<td>\t</td>
</tr>
<tr>
<td>Horizontal tab</td>
<td>\v</td>
</tr>
<tr>
<td>Vertical tab</td>
<td></td>
</tr>
</tbody>
</table>

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```c
b = (isspace)(c);
```

`c` is the character to test. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI
Library

ctype.h

See Also

isalnum()  isalpha()
iscntrl()  isdigit()
isgraph()  islower()
isprint()  ispunct()
isupper()  isxdigit()
isupper() is a C macro that returns a non-zero value for uppercase letters. isupper() returns a non-zero value for ASCII characters with values in the range from 0x41 (A) to 0x5a (Z).

This routine, available as a function or a macro in <ctype.h>, does not operate well with values outside the range of an unsigned char. The function checks for invalid input (values that are not representable as an unsigned char and not equal to the value of the macro EOF) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the undef preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```c
b = (isupper)(c);
```

`c` is the character to test. (Input)

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User and System  
Threads: Safe  
Re-entrant: Yes  
Standards: ANSI

**Library**

<ctype.h>

**See Also**

isalnum() isalpha()  
iscntrl() isdigit()  
isgraph() islower()  
isprint() ispunct()  
ispace() isxdigit()
isxdigit()

See If Parameter Is Hexadecimal

Syntax

```c
#include <ctype.h>
int isxdigit(int c);
```

Description

`isxdigit()` is a C macro that returns a non-zero value for any hexadecimal-digit character.

This routine, available as a function or a macro in `ctype.h`, does not operate well with values outside the range of an `unsigned char`.

The function checks for invalid input (values that are not representable as an `unsigned char` and not equal to the value of the macro `EOF`) and returns 0 for such input. The macro does not check for invalid input, which makes it faster than the function. The macro is the default. To use the function you must explicitly use the `undef` preprocessor directive or ensure that the macro expansion does not occur by placing the function name within parentheses.

For example:

```c
b = (isxdigit)(c);
```

c

is the character to test. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library

`ctype.h`

See Also

- `isalnum()`
- `isalpha()`
- `iscntrl()`
- `isdigit()`
- `isgraph()`
- `islower()`
- `isprint()`
- `ispunct()`
- `isspace()`
- `isupper()`
 Syntax
#include <time.h>
int _julian(
    int *time,
    int *date);

 Description
_julian() converts the time and date from Gregorian format to the Julian equivalents.

_julian() modifies the objects to which its parameters point as follows:

Figure 2-5. _julian() Output Date and Time Formats

<table>
<thead>
<tr>
<th>Time:</th>
<th>Seconds Since Midnight (0-86399)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date:</td>
<td>Julian Day Number</td>
</tr>
</tbody>
</table>

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

Be certain to pass pointers for date and time values.

<table>
<thead>
<tr>
<th>time and date</th>
</tr>
</thead>
</table>

are pointers to the Gregorian format time and date values. (Input/Output)

The following format is assumed:

Figure 2-6. _julian() Input Date and Time Formats

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time:</td>
<td>0</td>
<td>Hour (0-23)</td>
<td>Minute</td>
</tr>
<tr>
<td>Date:</td>
<td>Year (two bytes)</td>
<td>Month</td>
<td>Day</td>
</tr>
</tbody>
</table>

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Library

sys_lib.1

Example

main()
{
    int date,time,tick;
    short day;
    _sysdate(0,&time,&date,&day,&tick);
    _julian(&time,&date);
    printf("The Julian date is %d. \n    Cinderella needs to be home in %d seconds!\n",date,86400-time);
}

See Also

_archdate() (OS-9 for 68K)
_os_gregorian() (OS-9 for 68K)
_os_julian() (OS-9 for 68K)
FS_Time OS-9 for 68K Technical Manual
FS_Julian OS-9 for 68K Technical Manual
FS_TIME OS-9 Technical Manual
Syntax

#include <signal.h>
int kill(
    int pid,
    int sigcode);

Description

kill() sends a signal to a process. Both the sending and receiving process must have the same user number unless the sending process’ user number is that of the super user (0).

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

The super user can send a signal to all processes running on the system if the pid is zero.

pid
    is the process ID of the process to kill. (Input)

sigcode
    is the signal code to send. Only the lower 16 bits of sigcode are used. (Input)
    The value in sigcode is sent as a signal to the process specified by pid. You can pass any value in sigcode. The conventional code numbers are defined in the signal.h header file.

Attributes

Operating System:       OS-9 and OS-9 for 68K
State:                  User and System
Threads:                Safe
Re-entrant:             Yes

Library

sys_clib.l

See Also

_os_send()  
F$Send        OS-9 for 68K Technical Manual
F_SEND        OS-9 Technical Manual
kill command  Using OS-9 for 68K or Using OS-9
Syntax
#include <stdlib.h>
long int labs(long int num);

Description
labs() returns the absolute value of num. num and the returned value each have type long int.

num is the value to use. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also
abs()
Allocating Storage for Array (Low-Overhead)

**Syntax**

```c
#include <stdlib.h>
void *lcalloc(
    unsigned long nel,
    unsigned long elsize);
```

**Description**

`lcalloc()` allocates space for an array. The allocated memory is cleared to zeroes. `lcalloc()` calls `lmalloc()` to allocate memory.

- If the allocation is successful, `lcalloc()` returns a pointer to the area.
- If the allocation fails, `lcalloc()` returns NULL.

Use of the low-overhead allocation functions (`lcalloc()`, `lmalloc()`, and `lrealloc()` instead of the general allocation functions (`calloc()`, `malloc()`, and `realloc()`) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.

⚠️ Use extreme care to ensure that you only access the assigned memory. Modifying addresses immediately above or below the assigned memory causes unpredictable program results.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory. Note the `lfree()` and `lrealloc()` parameters.

- `nel` specifies the number of elements in the array. (Input)
- `elsize` specifies the size of each element. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

**Library**

`clib.l`

**See Also**

- `calloc()`, `free()`, `lfree()`
- `lmalloc()`, `lrealloc()`, `malloc()`, `mallocmin()`
- `_os9_srqmem()`
- `realloc()`
Syntax
#include <math.h>
double ldexp(
    double fp,
    int exp);

Description
ldexp() multiplies a floating point value by an integral power of two. ldexp() returns the value equal to \( fp \times 2^{exp} \).

\( fp \)
    is the floating point value. (Input)

\( exp \)
    is the integral power of two. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.1
ldiv()  
Compute Quotient and Remainder

Syntax
#include <stdlib.h>
ldiv_t ldiv(
    long int numer,
    long int denom);

Description
ldiv() returns the quotient and remainder of the division of the numerator by the
denominator. The parameters and member of the returned structure are all of type
long int.

numer
    is the numerator. (Input)

denom
    is the denominator. (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes
Standards:          ANSI

Library
clib.l

See Also
div()
Syntax

```c
#include <stdlib.h>
void _lfree(
    void *ptr,
    unsigned long size);
```

Description

_lfree() returns a block of memory granted by _lcalloc() or _lmalloc(). The memory is returned to a pool of memory for later re-use by calloc(), _lcalloc(), _lmalloc(), or malloc().

Use of the low-overhead allocation functions (_lcalloc(), _lmalloc(), and _lrealloc()) instead of the general allocation functions (calloc(), malloc(), and realloc()) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.

- If _lfree() is used with something other than a pointer returned by _lcalloc() or _lmalloc(), the memory lists maintained by _lmalloc() and malloc() are corrupted and programs behave unpredictably.
- _lfree() never returns an error.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory.

- **ptr**
  
  is a pointer to the memory block. (Input)

- **size**
  
  specifies the size of the memory block. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

clib.l

See Also

- calloc() 
  - free() 
  - _freemin()
- _lcalloc() 
  - _lmalloc() 
  - _lrealloc()
- malloc() 
  - _os_srtmem() 
  - realloc()
Iftocr()  
Convert Linefeed to Carriage Return

Syntax
#include <UNIX/os9def.h>
void lftocr(
    char *buf,
    int n);

Description
lftocr() converts all linefeed characters to carriage-return characters in buf for a span of n characters.

buf
    is a pointer to a buffer. (Input/Output).

n
    is the number of characters spanned. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
unix.l
Allocate Memory from Arena (Low-Overhead)

Syntax
#include <stdlib.h>
void *lmalloc(unsigned long size);

Description
_lmalloc() returns a pointer to a memory block. The pointer is suitably aligned for storing data of any type.

_lmalloc() maintains an amount of memory, called an arena, from which it grants memory requests. _lmalloc() searches its arena for a block of free memory large enough for the request and, in the process, joins adjacent blocks of free space returned by either _lfree() or free(). If sufficient memory is not available in the arena, _lmalloc() calls _srqmem() to get more memory from the system.

_lmalloc() returns NULL if no memory is available or if the arena is detected to be corrupted.

Use of the low-overhead allocation functions (_lcalloc(), _lmalloc(), and _lrealloc()) instead of the general allocation functions (calloc(), malloc(), and realloc()) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value.

Use extreme care to ensure that you only access the memory assigned by _lmalloc(). Modifying addresses immediately above or below the assigned memory or passing _lfree() a value not assigned by _lmalloc() causes unpredictable program results.

The low-overhead functions require that you keep track of the sizes of allocated spaces in memory. Note the _lfree() and _lrealloc() parameters.

size
specifies the size of the memory block in bytes. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
clib.l
localeconv()
Numeric Formatting Convention Inquiry

Syntax

```
#include <locale.h>
struct lconv *localeconv(void);
```

Description

localeconv() sets the components of an object with type `struct lconv` with values appropriate for formatting numeric quantities according to the rules of the current locale.

The members of the structure with type `char *` are pointers to strings, any of which (except `decimal_point`) can point to a NULL value to indicate that the value is not available in the current locale or is of zero length. The members with type `char` are non-negative numbers, any of which can be `CHAR_MAX` to indicate that the value is not available in the current locale. The members include the following:

- `char *decimal_point`
  The decimal point character used to format non-monetary quantities.

- `char *thousands_sep`
  The character used to separate groups of digits before the decimal point character in formatted non-monetary quantities.

- `char *grouping`
  A string whose elements indicate the size of each group of digits in formatted, non-monetary quantities. The string may contain the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CHAR_MAX</code></td>
<td>No further grouping is to be performed.</td>
</tr>
<tr>
<td><code>0</code></td>
<td>The previous element is to be repeatedly used for the remaining digits.</td>
</tr>
<tr>
<td>other</td>
<td>The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.</td>
</tr>
</tbody>
</table>

- `char *int_curr_symbol`
  The international currency symbol that applies to the current locale. The first three characters contain the alphabetic international currency symbol. The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.

- `char *currency_symbol`
  The local currency symbol applicable to the current locale.

- `char *mon_decimal_point`
  The decimal point used to format monetary quantities.
char *mon_thousands_sep
The separator for groups of digits before the decimal point in formatted monetary quantities.

char *mon_grouping
A string whose elements indicate the size of each group of digits in formatted monetary quantities. The string may contain the following:

<table>
<thead>
<tr>
<th>CHAR_MAX</th>
<th>No further grouping is to be performed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The previous element is to be repeatedly used for the remaining digits.</td>
</tr>
<tr>
<td>other</td>
<td>The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.</td>
</tr>
</tbody>
</table>

char *positive_sign
The string used to indicate a non-negative-valued formatted monetary quantity.

char *negative_sign
The string used to indicate a negative-valued formatted monetary quantity.

char int_frac_digits
The number of fractional digits (those after the decimal point) to display in an internationally formatted monetary quantity.

char p_cs_precedes
Set to one or zero if the currency_symbol respectively precedes or succeeds the value for a non-negative formatted monetary quantity.

char p_sep_by_space
Set to one or zero if the currency_symbol respectively is or is not separated by a space from the value for a non-negative formatted monetary quantity.

char n_cs_precedes
Set to one or zero if the currency_symbol respectively precedes or succeeds the value for a negative formatted monetary quantity.

char n_sep_by_space
Set to one or zero if the currency_symbol respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

char p_sign_posn
Set to a value indicating the positioning of the positive_sign for a non-negative formatted monetary quantity. The value may be:

<table>
<thead>
<tr>
<th>Table 2-11. Values for char _p_sign_posn</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
**localeconv()**

char n_sign_posn
Set to a value indicating the positioning of the negative_sign for a negative formatted monetary quantity. The values are the same as for p_sign_posn.

localeconv() returns a pointer to the filled in object. You cannot modify the structure pointed to by the return value, but a subsequent call to localeconv() can overwrite the structure. In addition, calls to setlocale() with categories LC_ALL, LC_MONETARY, or LC_NUMERIC may overwrite the contents of the structure.

**Attributes**
- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

**Library**
- clib.1

---

**Table 2-11. Values for char _p_sign_posn (Continued)**

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>The sign string immediately precedes the currency symbol.</td>
</tr>
<tr>
<td>4</td>
<td>The sign string immediately succeeds the currency symbol.</td>
</tr>
</tbody>
</table>
localtime()  
Convert Calendar Time to Local Time

Syntax

```c
#include <time.h>
struct tm *localtime(time_t *timer);
```

Description

`localtime()` converts a calendar time into a broken down time, expressed as local time.

`localtime()` returns a pointer to a static area which may be overwritten. To ensure data integrity, use the value or save it immediately.

`timer` is a pointer to the calendar time. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

c.lib

See Also

`mktime()`
`time()`


**Syntax**

```c
#include <math.h>
double log(double x);
```

**Description**

`log()` returns the natural logarithm of `x` (input). A domain error occurs, `EDOM` stored in `errno`, if `x` is negative. A range error, `ERANGE` stored in `errno`, occurs if `x` is zero. `log()` returns `HUGE_VAL` for both errors.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

`clib.l`

**Possible Errors**

- `EDOM`
- `ERANGE`
Syntax
#include <math.h>
double log10(double x);

Description
log10() returns the base-ten logarithm of x (input). A domain error, EDOM stored in errno, occurs if x is negative. A range error, ERANGE stored in errno, occurs if x is zero. log10() returns HUGE_VAL for both errors.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

Possible Errors
EDOM
ERANGE
longjmp()

Non-Local Goto

Syntax

```c
#include <setjmp.h>

void longjmp(
    jmp_buf env,
    int val);
```

Description

`longjmp()` and `setjmp()` allow program control to return directly to a higher level function. They are most useful when dealing with errors and signals encountered in a low-level routine.

`longjmp()` restores the environment saved by the most recent call to `setjmp()` in the same execution of the program, with the corresponding `jmp_buf` parameter. If there has been no such call or if the function containing the `setjmp()` call has returned in the interim, the behavior is undefined.

All accessible objects have values as of the time `longjmp()` was called, except that the value of objects of automatic storage duration that are local to the function containing the call to the corresponding `setjmp()` call and `longjmp()` call are indeterminate.

Because it bypasses the usual function call and return mechanisms, `longjmp()` executes correctly in contexts of interrupts, signals, and any of their associated functions. However, if `longjmp()` is called from a nested signal handler (that is, from a function called as a result of a signal raised during the handling of another signal), the behavior is undefined.

After `longjmp()` is completed, program execution continues as if the corresponding call to `setjmp()` had just returned the value specified by `val`. `longjmp()` cannot cause `setjmp()` to return the value zero. If `val` is zero, `setjmp()` returns the value 1.

**Do not call `longjmp()` before `env` is initialized by `setjmp()` or if the function calling `setjmp()` has already returned.**

`env`

is the program environment buffer. (Input)

`val`

is the error status value. (Input)
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also
setjmp()
Syntax

```c
#include <stdlib.h>

void *lrealloc(
    void *oldptr,
    unsigned long newsize,
    unsigned long oldsize);
```

Description

 CGSize() resizes a block of memory.

CGSize() returns a pointer to a new memory block of size newsize. The pointer is aligned to store data of any type.

If newsize is smaller than oldsize, the contents of the old block are truncated and placed in the new block. Otherwise, all of the old block’s contents begin the new block. The results of CGSize(NULL,10,0) and _lmalloc(10) are the same.

CGSize() returns NULL if the requested memory is not available or if newsize is specified as zero.

Use of the low-overhead allocation functions (lmalloc(), lmalloc(), and lrealloc()) instead of the general allocation functions (calloc(), malloc(), and realloc()) saves 8 bytes per allocation because the low-overhead functions do not save the allocation size or the 4-byte check value. The low-overhead functions require that you keep track of the sizes of allocated spaces in memory.

oldptr

is a pointer to the memory block. (Input)

oldptr should be a value returned by a previous call to lmalloc(), lcalloc(), or lrealloc().

newsize

specifies the size to use for the memory block. (Input)

oldsize

is the size of the old memory block. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Library
clib.l

See Also
calloc().free()
__freemin().__lmalloc()
__lfree().__lmalloc()
malloc().__mallocmin()
_os_srqmem().os_srtmem()
realloc()
**Syntax**

```c
#include <modes.h>
long lseek(
    int path,
    long position,
    int place);
```

**Description**

`lseek()` repositions the file pointer for the file open on `path` to the byte offset given in `position`.

The returned value is the resulting position in the file. If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

On OS-9, the file pointer is a full unsigned 32-bits.

Do not use `lseek()` on a buffered file because the buffering routines use `fseek()` to keep track of the file pointer.

- `path` is the path number of the open file. (Input)
- `position` is the new file position offset. (Input)
- `place` determines from which file position the offset is based. (Input)
  - 0 — From the beginning of the file.
  - 1 — From the current position.
  - 2 — From the end of the file.

Seeking to a location beyond end-of-file for a file open for writing and then writing to it creates a hole in the file. The hole contains data with no particular value, usually garbage remaining on the disk.

**Attributes**

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User and System
- **Threads**: Safe
- **Re-entrant**: Yes
Library
sys_lib.l

See Also
fseek()
_os_seek()
I SEEK
I SEEK

OS-9 for 68K Technical Manual
OS-9 Technical Manual
**lstat()**

Obtain Information about a File

**Syntax**

```c
#include <UNIX/stat.h>
int lstat(
    const char *path,
    struct stat *sb);
```

**Description**

The `lstat()` function obtains information about the file pointed to by `path`. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and `errno` is set to indicate the error.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`unix.l`

**See Also**

- `close()`
- `closedir()`
- `open()`
- `opendir()`
- `localtime()`
- `mktime()`
makdir()  
Create Directory

Syntax
#include <modes.h>

int mkdir(
    const char *name,
    int mode,
    int perm[, 
    int size]);

⚠️ For C++ Programmers:
The parameters for this function are not optional in C++. You must specify each; if you do not, a compilation error will occur.

Description

makdir() creates a directory file.

If successful, mkdir() returns zero. Otherwise, -1 is returned and the appropriate error code is placed in the global variable errno.

name
is a pointer to the name of the directory. (Input)

mode
specifies the file access mode. (Input)

    Only the lower 16 bits of mode are used. (Input)

perm
specifies the file attribute permissions. (Input)

    Only the lower 16 bits of perm are used. If the FAM_SIZE (or S_ISIZE) bit is set in mode, the initial size of the directory is set to size.

size
is the size of the directory. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
makdir()

Library
sys_clib.l

See Also
=os_mkdir()  OS-9 for 68K Technical Manual
I$MakDir  OS-9 Technical Manual
I_MAKDIR
Syntax
#include <regs.h>
#include <types.h>
void *make_gdesc(
    void *tbl_entry_addr,
    u_int32 offset,
    u_int16 segment_attr);

Description
make_gdesc() makes an entry in the global descriptor table. It returns a pointer to the next entry.

⚠️ make_gdesc() is only supported by the 80x86 family of processors version of the compiler.

tbl_entry_addr
is a pointer to the address of the local descriptor table or the global descriptor table. (Input)

offset
is the offset into the global descriptor table. (Input)

segment_attr
is the segment attributes. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l
#make_idesc()

## Make Interrupt Descriptor Table Entry

### Syntax

```c
#include <regs.h>
#include <types.h>

void *make_idesc(
    void *tbl_entry_addr,
    u_int32 offset,
    u_int16 segment_attr,
    u_int16 selector);
```

### Description

`make_idesc()` makes an entry in the interrupt descriptor table. It returns a pointer to the next entry.

- **tbl_entry_addr**
  - is a pointer to the interrupt descriptor table. (Input)

- **offset**
  - is an offset into the table. (Input)

- **segment_attr**
  - specifies the segment attributes. (Input)

- **selector**
  - specifies the selector entry value. (Input)

### Attributes

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

### Library

- `cpu.l`
Syntax

```c
#include <memory.h>
#include <module.h>
mh_com *make_module(
    const char *name,
    int size,
    int attr,
    int perm,
    int typelang,
    int color);
```

Description

`make_module()` creates a memory module with the specified name and attributes in the specified memory. `make_module()` is similar to `_mkdata_module()` but also allows you to specify the type/language of the module and color of memory in which to make the module.

`make_module()` returns a pointer to the beginning of the module header. If the module cannot be created, -1 is returned and the appropriate error code is placed in the global variable `errno`.

The name of the module created by `make_module()` always begins at offset $34$ within the module. This implies that program modules, trap handlers, file managers, device drivers, and device descriptors cannot be made conveniently by this call.

`mod_exec` is defined in the `module.h` header file.

`name` is a pointer to the name of the module. (Input)

`size` specifies the desired memory size of the module in bytes. (Input)

The size value does not include the module header and CRC bytes.

`size` is the amount of memory available for actual use. (Input)

The memory in the module is initially cleared to zeroes.

`attr` specifies the module’s attribute and revision level. (Input)

Only the lower 16 bits of `attr` are used.
perm
specifies the module permissions. (Input)
Only the lower 16 bits of perm are used.

typelang
specifies the type and language of the module. (Input)
Only the lower 16 bits of typelang are used.

color
indicates the specific memory type in which to load the module. (Input)
The memory.h header file contains definitions of the memory types that you can specify:

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSRAM or MEM_SYS</td>
<td>System RAM memory</td>
</tr>
<tr>
<td>VIDEO1</td>
<td>Video memory for plane A</td>
</tr>
<tr>
<td>VIDEO2</td>
<td>Video memory for plane B</td>
</tr>
<tr>
<td>MEM_ANY</td>
<td>No specific memory type</td>
</tr>
</tbody>
</table>

If color is MEM_ANY, the module is made in whatever memory the system allocates.

Attributes
Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_datmod()
F$DatMod service request  OS-9 for 68K Technical Manual
malloc()

Allocate Memory from Arena

Syntax

#include <stdlib.h>

void *malloc(size_t size);

Description

malloc() allocates space for an object. The value of the object is indeterminate. malloc() returns either a null pointer (if sufficient free memory is not available) or a pointer to the allocated space.

size is the size of the memory block to allocate. (Input)

If size is zero, NULL is returned.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

calloc()
free()
_os_srqmem()
_os9_srqmem()
realloc()
F$SRqMem
F SRQMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax
#include <stdlib.h>
void _mallocmin(unsigned size);

Description
(Encoding) sets the minimum amount of memory that allocation functions may request from the system.

(size) is the minimum allocation size in bytes. (Input)

(size) cannot be less than the system memory block size. Size must be greater than or equal to the larger of the system memory block size or 4K. If a smaller size is requested, (size) is automatically set to the system memory block size.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
clib.l

See Also
calloc()
_lcalloc()
_lm allocated
_lrealloc()
malloc()
realloc()
mblen()

Determine Number of Bytes in Multibyte Character

Syntax

```c
#include <stdlib.h>

int mblen(
    const char *str,
    size_t num);
```

Description

`mblen()` determines the number of bytes contained in the multibyte character pointed to by `str`, if `str` is a not a null pointer. `mblen()` is similar to the following; however, the shift state of `mbtowc()` is unaffected:

```c
mbtowc((wchar_t *)0, s, n);
```

If `str` is a null pointer, `mblen()` returns a non-zero value (if the multibyte character encodings have state-dependent encodings) or returns a zero value.

If `str` is not a null pointer, `mblen()` returns:

- 0 if `str` points to the null character.
- The number of bytes contained in the multibyte character if the next `num` or fewer bytes form a valid multibyte character.
- -1 if they do not form a valid multibyte character.

`mblen()` is supported for both the C and JAPAN locales.

- `str` is a pointer to a multibyte character. (Input)
- `num` is the number of bytes to search. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.l

See Also

`mbtowc()`
**Syntax**

```c
#include <stdlib.h>
size_t mbstowcs(
    wchar_t *pwcs,
    const char *str,
    size_t num);
```

**Description**

`mbstowcs()` converts a sequence of multibyte characters that begins in the initial shift state into a sequence of corresponding codes and stores not more than `num` codes in an array. No multibyte characters that follow a null character are examined or converted. A null character is converted into a code with value zero. Each multibyte character is converted as if by a call to `mbtowc()`, except that the shift state of `mbtowc()` is unaffected.

No more than `num` elements are modified in `pwcs`. If copying takes place between overlapping objects, the behavior is undefined.

If an invalid multibyte character is encountered, `mbstowc()` returns `(size_t)-1`. Otherwise, `mbstowcs()` returns the number of array elements modified, not including a terminating zero code, if any.

`mbstowcs()` is supported for both the C and JAPAN locales.

- `pwcs` is a pointer to the array into which to store the converted values. (Output)
- `str` is a pointer to the original sequence of multibyte characters. (Input)
- `num` specifies the maximum number of elements to modify. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

**Library**

`clib.l`

**See Also**

`wcstombs()`
mbtowc()  
Determine Number of Bytes in Multibyte Characters

Syntax

```c
#include <stdlib.h>

int mbtowc(
    wchar_t *pwc,
    const char *mbchar,
    size_t maxnum);
```

Description

`mbtowc()` determines the number of bytes contained in the multibyte character pointed to by `mbchar`, if `mbchar` is not a null pointer. It then determines the code for the value of type `wchar_t` that corresponds to that multibyte character. The value of the code corresponding to the null character is zero. If the multibyte character is valid and `pwc` is not a null pointer, `mbtowc()` stores the code in `pwc`. At most `maxnum` bytes of `mbchar` are examined. The returned value is never greater than `maxnum` or `MB_CUR_MAX`. `mbtowcs()` is supported for both the C and JAPAN locales.

- `pwc` is a pointer to the array in which to store the code. (Output)
- `mbchar` is a pointer to a multibyte character. (Input)

If `mbchar` is a null pointer, `mbtowc()` returns a non-zero value if multibyte character encodings have state-dependent encodings or returns a zero value.

If `mbchar` is not a null pointer, `mbtowc()` either returns:

- 0 if `mbchar` is a pointer to the null character.
- The number of bytes contained in the converted multibyte character if the next `maxnum` or fewer bytes form a valid multibyte character.
- -1 if they do not form a valid multibyte character.

- `maxnum` specifies the maximum number of bytes to examine. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI
Library
clib.l

See Also
wctomb()
memchr()
Memory Search

Syntax
#include <string.h>
void *memchr(
    const void *mem_area,
    int chr,
    size_t size);

Description
memchr() locates the first occurrence of chr, converted to an unsigned char, in the initial size characters, each interpreted as unsigned char, of mem_area.

memchr() returns a pointer to the located character, or a null pointer if the character does not occur in the object.

mem_area
    is a pointer to the memory to search. (Input)

chr
    is the character for which to search. (Input)

size
    is the size of the memory area to search. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l
Syntax

```c
#include <string.h>
int memcmp(
    const void *string1,
    const void *string2,
    size_t size);
```

Description

`memcmp()` compares the first `size` characters of `string1` to the first `size` characters of `string2`. `memcmp()` returns an integer that is:

- **Positive**, if `string1` is greater than `string2`.
- **Zero**, if `string1` equals `string2`.
- **Negative**, if `string1` is less than `string2`.

`string1`
- is a pointer to a string to compare. (Input)

`string2`
- is a pointer to a string to compare. (Input)

`size`
- is the number of characters to compare. (Input)

Attributes

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User and System
- **Threads**: Safe
- **Re-entrant**: Yes
- **Standards**: ANSI

Library

`clib.l`
memcpy()
Copy Memory

Syntax

```c
#include <string.h>
void *memcpy(
    void *dest,
    const void *source,
    size_t num);
```

Description

memcpy() copies characters from one location to another. If copying takes place between overlapping objects, the behavior is undefined. memcpy() returns the value of dest.

dest
    is a pointer to the destination in memory. (Output)

source
    is a pointer to the memory to copy into dest. (Input)

num
    is the number of characters to copy. (Input)

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes
Standards:          ANSI

Library

clib.l
memmove()

Move Memory

Syntax

```c
#include <string.h>
void *memmove(
    void *dest,
    const void *source,
    size_t num);
```

Description

`memmove()` copies characters from one location to another. Copying takes place as if the characters from the original location are first copied into a temporary array of `num` characters that does not overlap `dest` and `source`, and then the `num` characters from the temporary array are copied into `dest`. `memmove()` returns the value of `dest`.

- **dest** is a pointer to the destination in memory. (Output)
- **source** is a pointer to the memory to copy into `dest`. (Input)
- **num** is the number of characters to copy. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.l
Syntax
#include <string.h>
void *memset(
    void *mem,
    int fill,
    size_t size);

Description
memset() copies the value of fill, converted to an unsigned char, into each of the first size characters of mem. memset() returns the value of mem.

mem
    is a pointer to the memory to fill. (Output)

fill
    is the value with which to fill an area of memory. (Input)

size
    is the number of characters to fill with fill. (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes
Standards:          ANSI

Library
clib.l
_mkdata_module()

Create Data Memory Module

Syntax

```c
#include <module.h>
char *_mkdata_module(
    const char *name,
    unsigned size,
    int attr,
    int perm);
```

Description

_mkdata_module() creates a data memory module. Other processes on the system can then access the data module by modlink(). The memory in the data module is initially cleared to zeroes.

_mkdata_module() returns a pointer to the beginning of the module header.

If the data module cannot be created, -1 is returned and the appropriate error code is placed in the global variable errno.

name

is a pointer to the desired module name. (Input)

size

specifies the size of the module in bytes. (Input)

The size value does not include the module header and CRC bytes. The specified size is the amount of memory available for actual data storage.

attr

specifies the module’s attribute and revision level. Only the lower 16 bits of attr are used.

perm

specifies the module access permissions. Only the lower 16 bits of perm are used.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>OS-9 and OS-9 for 68K</td>
</tr>
<tr>
<td>State</td>
<td>User and System</td>
</tr>
<tr>
<td>Threads</td>
<td>Safe</td>
</tr>
<tr>
<td>Re-entrant</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Library
sys_clib.l

See Also
=os_datmod() OS-9 for 68K Technical Manual
F$DatMod OS-9 Technical Manual
F_DATMOD
mknod()
Create Directory

Syntax
#include <modes.h>
int mknod(
    const char *name,
    int perm);

Description
mknod() creates a directory file.
mknod() returns zero if the directory was successfully created. If the creation failed,
-1 is returned and the appropriate error code is placed in the global variable errno.
mknod() does not make UNIX-style special files as there are no such files on OS-9 for 68K and OS-9. This is a historical function and is likely to be removed in a future release. Use mkdir() in all new code.

name
    is a pointer to the name of the directory. (Input)

perm
    specifies the access permissions for the directory. (Input)
    Only the lower 16 bits of perm are used.

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes

Library
sys_clib.l

See Also
mkdir()
_os_mkdir()
I$MakDir
I_MAKDIR  OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax

```c
#include <stdio.h>
#include <stdlib.h>
int mkstemp(
    char *template);
```

Description

The `mkstemp()` function takes the given file name template and overwrites a portion of it to create a template file, mode S_IRREAD/S_IWWRITE, returning a file descriptor opened for reading and writing. This avoids the race between testing for a file’s existence and opening it for use.

This function returns -1 if no suitable file could be created. If either call fails an error code is placed in the global variable errno.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

sys_clib.l

See Also

- mktemp()
- create()
mktemp()
Create Unique File Name

Syntax
#include <stdio.h>
char *mktemp(char *name);

Description
mktemp() ensures that the name of a temporary file is unique in the system and does not clash with any other file name.

A pointer to a template string is passed to mktemp(). The template string should look like a filename with a minimum of three trailing X’s. mktemp() replaces the X’s with a letter and the current process ID. The letter is chosen so that the resulting name does not conflict with an existing file.

mktemp() returns a pointer to the file name or NULL if more than 256 mktemp() files exist.

name is a pointer to the template string. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
tmpnam()
tmpfile()
mktime()
Convert Broken-Down Time to Calendar Time

Syntax

```c
#include <time.h>

time_t mktime(struct tm *tp);
```

Description

`mktime()` converts the broken-down time, expressed as local time, into a calendar time value with the same encoding as the values returned by `time()`. The original values of the `tm_wday` and `tm_yday` fields of the structure are ignored, and the original values of the other fields are not restricted to the indicated ranges.

On successful completion, the `tm_wday` and `tm_yday` fields of the structure are set appropriately. The other fields are set to represent the specified calendar time, but with their values forced to the indicated ranges. The final value of `tm_mday` is not set until `tm_mon` and `tm_year` are determined.

`mktime()` returns the specified calendar time encoded as a value of type `time_t`. If the calendar time cannot be represented, `mktime()` returns the value `(time_t)-1.`

`tp` is a pointer to the broken-down time structure. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

clib.l

See Also

- `asctime()`
- `getenv()`
- `gmtime()`
- `localtime()`
- `time()`
modcload()

Load Module into Colored Memory

Syntax

```
#include <memory.h>
#include <module.h>

mh_com *modcload(
    const char *modname,
    int mode,
    int memtype);
```

Description

`modcload()` opens the file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached. `modname` is considered a pathlist and all modules in the specified file are loaded. A link is made to the first module loaded from the file and a pointer to it is returned.

If `memtype` is `MEM_ANY`, the module may be loaded in whatever memory the system allocates.

If the load is successful, `modcload()` returns a pointer to the module. If the load fails, -1 is returned and the appropriate error code is placed in the global variable `errno`.

- `modname` is a pointer to the path to the module. (Input)
- `mode` is the mode which opens the file to load. (Input)
  - If any access `mode` is acceptable, specify zero for `mode`.
- `memtype` indicates the specific memory type in which to load the module. (Input)

The `memory.h` header file contains definitions of the memory types that may be specified:

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSRAM or MEM_SYS</td>
<td>System RAM memory</td>
</tr>
<tr>
<td>VIDEO1</td>
<td>Video memory for plane A</td>
</tr>
<tr>
<td>VIDEO2</td>
<td>Video memory for plane B.</td>
</tr>
<tr>
<td>MEM_ANY</td>
<td>No specific memory type.</td>
</tr>
</tbody>
</table>
**Attributes**

Operating System: OS-9 for 68K  
State: User and System  
Threads: Safe  
Re-entrant: Yes  

**Library**

sys_clib.l
modf()

Return Parts of Real Number

Syntax

```c
#include <math.h>

double modf(
    double value,
    double *num);
```

Description

`modf()` breaks `value` into integral and fractional parts, each of which has the same sign as the parameter. It stores the integral part as a double in `num`.

`modf()` returns the signed fractional part of `value`.

- `value` is a real number. (Input)
- `num` is a pointer to the integer portion of `value`. (Output)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.1
**Syntax**

```c
#include <module.h>
mh_com *modlink(
    const char *modname,
    int typelang);
```

**Description**

`modlink()` searches the module directory for a module with the same name as that pointed to by `modname` and links to it, provided that `typelang` matches the respective value of Type/Language in the module. If the module is found, the module’s link count is incremented by one.

The `module.h` header file contains a structure appropriate for accessing the elements of system-defined memory modules.

If the link is successful, `modlink()` returns a pointer to the module. If the link fails, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

`modname`

is a pointer to the name of the module. (Input)

`typelang`

is the type and language value of the module. (Input)

If any module type or language is acceptable, specify zero for `typelang`. Only the lower 16 bits of `typelang` are used.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

`sys_clib.l`

**See Also**

- `_os_link()`
- `_os_load()`
- `F$Link`
- `F_LINKM`
modload()

Load and Link to Memory Module

Syntax
#include <module.h>
mh_com *modload(
    const char *modname,
    int accessmode);

Description
modload() loads all modules in the file specified by the path modname.
A link is made to the first module loaded from the file and a pointer to it is returned.
The module.h header file contains a structure appropriate for accessing the elements of system-defined memory modules.
If the load is successful, modload() returns a pointer to the module. If the load fails, -1 is returned and the appropriate error code is placed in the global variable errno.
To load a module using the shell PATH variable list, see _os_loadp() or modloadp().

modname
    is a pointer to the name of the module. (Input)

accessmode
    specifies the module’s access mode. (Input)
    If any module access mode is acceptable, specify zero for accessmode. Only the lower 16 bits of accessmode are used.

Attributes
Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library
sys_clib.l

See Also
_os_link()
_os_load()
_os_loadp()
F$Load
F_LOAD

OS-9 for 68K Technical Manual
OS-9 Technical Manual
modloadp()
Load and Link to Memory Module Using PATH

Syntax

```c
#include <module.h>
mb_com *modloadp(
    const char *modname,
    int accessmode,
    char *namebuffer);
```

Description

`modloadp()` loads all modules in the file specified by the path `modname`. The `PATH` environment variable determines alternate directories to search for the named module.

`PATH` has a list of directories to search if the program is not found in the module directory or execution directory. `PATH` is set to any number of directory path lists separated by colons: `/h0/cmds:/d0/cmds:/n0/droid/h0/special_stuff/cmds`

`modloadp()` returns a pointer to the module if the load is successful. If the load is unsuccessful, -1 is returned and the appropriate error code is placed in `errno`.

`modname`
- is a pointer to the name of the module. (Input)

`accessmode`
- is the access mode for the module. (Input)
- Only the lower 16 bits of `accessmode` are used.

`namebuffer`
- is a pointer to an array containing the pathlist of the successfully loaded module. (Input)
- Any errors other than EOS_PNNF leave the path list used for the load attempt intact. `namebuffer` can be NULL if access to the path string is not required.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_loadp()  modload()_os_load()
mq_close
Close a Message Queue

Syntax
#include <mqueue.h>
int mq_close(mqd_t mqdes);

Description
mq_close() removes the association between the message queue descriptor and its message queue. The results of using this descriptor after successful return from mq_close()—and until the return of this descriptor from a subsequent mq_open()—are undefined.

Upon successful completion mq_close() returns a value of 0. Otherwise, the function returns -1 and sets errno to indicate the error.

mqdes
is the message queue descriptor. (Input)

If the process has successfully attached a notification request to the message queue via mqdes, the attachment is removed and the message queue is available for another process to attach for notification.

Errors
EBADF The mqdes argument is not a valid message queue descriptor.
ENOSYS mq_close() not supported.

Attributes
Operating System: OS-9
State: User
Threads: Safe
Re-entrant: No
Standards: POSIX

Library
mq.l

See Also
mq_open
mq_unlink
## mq_getattr

**Get Message Queue Attributes**

### Syntax

```c
#include <mqueue.h>

int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
```

### Description

- `mqdes` specifies a message queue descriptor. (Input)

`mq_getattr()` gets status information and attributes of the message queue and the open message queue description associated with the message queue descriptor. The results are returned in the `mq_attr` structure referenced by the `mqstat` argument. Upon return, the following members shall have the values associated with the open message queue description as set when the message queue was opened and as modified by subsequent `mq_setattr()` calls: `mq_flags`.

The following attributes of the message queue shall be returned as set at message queue creation: `mq_maxmsg`, `mq_msgsize`. Upon return, the following members within the `mq_attr` structure referenced by the `mqstat` argument shall be set according to the current state of the message queue: `mq_curmsgs`—the number of messages currently on the queue.

OS-9 specifies the following user-defined flag for use with `mq_getattr()` and `mq_setattr()`:

- `_MQ_O_NOTIFY_IMMEDIATE`. When this flag is specified by the `mq_setattr()` function (mq_flags field of `mqstat` parameter) then calls to either `_mq_notify_write()` or `mq_notify()` will notify immediately if appropriate. (Message queue nonfull for `_mq_notify_write()`, and message queue non-empty for `mq_notify()`). This extension provides message queue notification consistency with other OS-9 services such the "send signal on data ready" concept. Upon successful completion `mq_getattr()` returns 0. Otherwise, the function returns -1 and sets `errno` to indicate the error.

- `mqstat` is the argument that references the `mq_attr` structure. (Output)

### Errors

- **EBADF** The `mqdes` argument is not a valid message queue descriptor.
- **ENOSYS** `mq_getattr()` not supported.
### Attributes

- **Operating System:** OS-9  
- **State:** User  
- **Threads:** Safe  
- **Re-entrant:** No  
- **Standards:** POSIX

### Library

mq.l

### See Also

- `mq_open`  
- `mq_send`  
- `mq_setattr`
mq_notify

Notify Process That a Message is Available on a Queue

Syntax

```c
#include <mqueue.h>
int mq_notify(
    mqd_t mqdes,
    const struct sigevent *notification);
```

Description

If the argument notification is not NULL, this function registers the calling process to be notified of message arrival at an empty message queue associated with `mqdes`.

If a process has registered for notification of message arrival at a message queue, and some thread is blocked in `mq.receive()` waiting to receive a message when a message arrives at the queue, the arriving message satisfies the appropriate `mq.receive()`. The resulting behavior is as if the message queue remains empty, and no notification is sent. Upon successful completion, `mq.notify()` returns a value of 0. Otherwise, the function returns -1 and sets `errno` to indicate the error.

- `mqdes` is the specified message queue descriptor. (Input)
- `notification` is the notification argument. (Input)

The notification specified by the notification argument, `notification`, is sent to the process when the message queue transitions from empty to nonempty. At any time, only one process can be registered for notification by a message queue. If the calling process or any other process has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue fail.

If notification is NULL and the process is currently registered for notification by the specified message queue, the existing registration is removed. When the notification is sent to the registered process, its registration is removed. The message queue is then available for registration.

Errors

- **EBADF** The `mqdes` argument is not a valid message queue descriptor.
- **EBUSY** A process is already registered for notification by the message queue.
- **ENOSYS** `mq.notify()` not supported.
**Attributes**

Operating System: OS-9  
State: User  
Threads: Safe  
Re-entrant: No  
Standards: POSIX

**Library**
mq.1

**See Also**

mq_open  
mq_send  
mq_setattr  
MQ_0_NOTIFY_IMMEDIATE
_mq_notify_write
Notify Process of Space Availability in Message Queue

**Syntax**

```c
#include <mqueue.h>

int _mq_notify_write(
    mqd_t                    mqdes,
    const struct sigevent    *notification);
```

**Description**

If the argument notification is not NULL, this function registers the calling process to be notified of message removal at a full message queue associated with the specified message queue descriptor, `mqdes`.

If a process has registered for notification of a message queue and some thread is blocked in `mq_send()` waiting to send a message when the message queue transitions to nonfull, the arriving message will satisfy the appropriate `mq_send()`. The resulting behavior is as if the message queue remains full, and no notification is sent.

- `mqdes` is the specified message queue descriptor. (Input)
- `notification` is the notification argument. (Input)

The notification specified by the notification argument, `notification`, is sent to the process when the message queue transitions from full to nonfull. At any time, only one process may be registered for notification by a message queue. If the calling process or any other process has already registered for notification of the specified message queue, subsequent attempts to register for that message queue will fail.

If notification is NULL and the process is currently registered for notification by the specified message queue, the existing registration is removed. The message queue is then available for registration.

**Errors**

- **EBADF** The `mqdes` argument is not a valid message queue descriptor.
- **EBUSY** A process is already registered for notification by the message queue.
- **ENOSYS** `mq_notify()` not supported.
Attributes
Operating System: OS-9
State: User
Threads: Safe
Re-entrant: No
Standards: POSIX

Library
mq.l

See Also
mq_open
mq_send
mq_setattr
_MQ_0_NOTIFY_IMMEDIATE
mq_open
Open a Message Queue

Syntax

#include <mqueue.h>

mqd_t mq_open(const char *name, int oflag, ...);

Description

mq_open() establishes the connection between a process and a message queue with a message queue descriptor. It creates an open message queue description that refers to the message queue, and it creates a message queue descriptor that refers to that open message queue description.

mq_open() does not add or remove messages from the queue.

Upon successful completion, mq_open() returns a message queue descriptor. Otherwise, the function returns (mqd_t)-1 and sets errno to indicate the error.

name

points to a string naming a message queue. (Input)

It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The name argument conforms to the construction rules for a pathname. If name begins with the slash character, then processes calling mq_open() with the same value of name shall refer to the same message queue object, as long as that name has not been removed. If name does not begin with the slash character, the effect is implementation defined. The interpretation of slash characters other than the leading slash character in name is implementation defined.

If the name argument is not the name of an existing message queue and creation is not requested, mq_open() fails and returns an error.

oflag

requests the desired receive and/or send access to the message queue. (Input)

The requested access permission to receive messages or send messages is granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.

The value of oflag is the bitwise inclusive OR of values from the following list. Applications shall specify exactly one of the first three values (access modes) below in the value of oflag:

O_RDONLY Open the message queue for receiving messages. The process can use the returned message queue descriptor with mq_receive(), but not mq_send(). A message queue may be open multiple times in the same or different processes for receiving messages.
modloadp()

O_WRONLY
Open the queue for sending messages. The process can use the returned message queue descriptor with \texttt{mq\_send()} but not \texttt{mq\_receive()}. A message queue may be open multiple times in the same or different processes for sending messages.

O_RDWR
Open the queue for both receiving and sending messages. The process can use any of the functions allowed for \texttt{O\__RDONLY} and \texttt{O\_WRONLY}. A message queue may be open multiple times in the same or different processes for sending messages.

Any combination of the remaining flags may be specified in the value of \texttt{o\_flag}:

O_CREAT
This option is used to create a message queue, and it requires two additional arguments: \texttt{mode}, which is of type \texttt{mode\_t}, and \texttt{attr}, which is a pointer to a \texttt{mq\_attr} structure. If the pathname, \texttt{name}, has already been used to create a message queue that still exists, then this flag has no effect, except as noted under \texttt{O\_EXCL}. Otherwise, a message queue is created without any messages in it.

The user ID of the message queue is set to the effective user ID of the process, and the group ID of the message queue is set to the effective group ID of the process. The "file permission bits" is set to the value of \texttt{mode}. When bits in \texttt{mode} other than file permission bits are set, the effect is implementation defined. If \texttt{attr} is \texttt{NULL}, the message queue is created with implementation-defined default message queue attributes. If \texttt{attr} is non-\texttt{NULL} and the calling process has the appropriate privilege on \texttt{name}, the message queue \texttt{mq\_maxmsg} and \texttt{mq\_msgsize} attributes are set to the values of the corresponding members in the \texttt{mq\_attr} structure referred to by \texttt{attr}. If \texttt{attr} is non-\texttt{NULL}, but the calling process does not have the appropriate privilege on \texttt{name}, \texttt{mq\_open()} fails and returns an error without creating the message queue.

O_EXCL
If \texttt{O\_EXCL} and \texttt{O\_CREAT} are set, \texttt{mq\_open()} fails if the message queue name exists. The check for the existence of the message queue and the creation of the message queue if it does not exist is atomic with respect to other processes executing \texttt{mq\_open()} naming the same name with \texttt{O\_EXCL} and \texttt{O\_CREAT} set. If \texttt{O\_EXCL} is set and \texttt{O\_CREAT} is not set, the result is undefined.

O_NONBLOCK
The setting of this flag is associated with the open message queue description and determines whether a \texttt{mq\_send()} or \texttt{mq\_receive()} shall wait for resources or messages that are not currently available, or fail with \texttt{errno} set to \texttt{[EAGAIN]}. See \texttt{mq\_send()} and \texttt{mq\_receive()} for details.
Errors

EACCES The message queue exists and the permissions specified by oflag are denied, or the message queue does not exist and permission to create the message queue is denied.

EEXIST O_CREAT and O_EXCL are set and the named message queue already exists.

EINTR The mq_open() operation was interrupted by a signal.

EINVAL The mq_open() operation is not supported for the given name. The implementation shall document under what circumstances this error may be returned.

ENFILE was specified in oflag, the value of attr is not NULL, and either mq_maxmsg or mq_msgsize was less than or equal to zero.

EMFILE Too many message queue descriptors or file descriptors are currently in use by this process.

ENAMETOOLONG The length of the name string exceeds {PATH_MAX}, or a pathname component is longer than {NAME_MAX} while {__POSIX_NO_TRUNC} is in effect.

ENOFILE Too many message queues are currently open in the system.

ENOENT O_CREAT is not set and the named message queue does not exist.

ENOSPC There is insufficient space for the creation of the new message queue.

ENOSYS mq_open() not supported.

Attributes

Operating System: OS-9
State: User
Threads: Safe
Re-entrant: No
Standards: POSIX

Library
mq.l

See Also
mq_close mq_receive
mq_send mq_setattr
mq_getattr mq_unlink
mq_receive
Receive a Message From a Message Queue

Syntax

```c
#include <mqueue.h>
size_t mq_receive(
    mqd_t        mqdes,
    char         *msg_ptr,
    size_t       msg_len,
    unsigned int *msg_prio);
```

Description

`mq_receive()` is used to receive the oldest of the highest priority message(s) from the message queue specified by `mqdes`. If the size of the buffer in bytes, specified by `msg_len`, is less than the `mq_msgsize` attribute of the message queue, the function fails and returns an error. Otherwise, the selected message is removed from the queue and copied to the buffer pointed to by `msg_ptr`.

If the specified message queue is empty and `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, `mq_receive()` blocks until a message is enqueued on the message queue or until `mq_receive()` is interrupted by a signal. If more than one thread is waiting to receive a message when a message arrives at an empty queue and the Process Scheduling option is supported, then the thread of highest priority that has been waiting the longest is selected to receive the message. Otherwise, it is unspecified which waiting thread receives the message. If the specified message queue is empty and `O_NONBLOCK` is set in the message queue description associated with `mqdes`, no message is removed from the queue, and `mq_receive()` returns an error.

Upon successful completion, `mq_receive()` returns the length of the selected message in bytes and the message is removed from the queue. Otherwise, no message is removed from the queue, the function returns a -1, and sets `errno` to indicate the error.

- `mqdes` specifies the message queue. (Input)
- `msg_ptr` points to the buffer. (Output)
- `msg_len` specifies the size of the buffer in bytes. (Input)
- `msg_prio` If `msg_prio` is not NULL, the priority of the selected message is stored in the location referenced by `msg_prio`. (Input)
Errors

EAGAIN  O_NONBLOCK was set in the message description associated with mqdes, and the specified message queue is empty.

EBADF  The mqdes argument is not a valid message queue descriptor open for reading.

EMSGSIZE  The specified message buffer size, msg_len, is less than the message size attribute of the message queue.

EINTR  The mq_receive() operation was interrupted by a signal.

ENOSYS  The mq_receive() function is not supported by this implementation.

EBADMSG  A data corruption problem with the message has been detected.

Attributes

Operating System: OS-9
State: User
Threads: Safe
Re-entrant: No
Standards: POSIX

Library

mq.l

See Also

mq_send
mq_send
Send a Message to a Message Queue

Syntax

```c
#include <mqueue.h>

int mq_send(
    mqd_t         mqdes,
    const char    *msg_ptr,
    size_t        msg_len,
    unsigned int  msg_prio);
```

Description

`mq_send()` adds the message pointed to by the argument `msg_ptr` to the message queue specified by `mqdes`. `msg_len` specifies the length of the message in bytes pointed to by `msg_ptr`. The value of `msg_len` is less than or equal to the `mq_msgsize` attribute of the message queue, otherwise `mq_send()` fails.

If the specified message queue is not full, `mq_send()` behaves as if the message is inserted into the message queue at the position indicated by the `msg_prio` argument. A message with a larger numeric value of `msg_prio` is inserted before messages with lower values of `msg_prio`. A message is inserted after other messages in the queue, if any, with equal `msg_prio`. The value of `msg_prio` is less than `{MQ_PRIO_MAX}`.

If the specified message queue is full and `O_NONBLOCK` is not set in the message queue description associated with `mqdes`, `mq_send()` blocks until space becomes available to enqueue the message, or until `mq_send()` is interrupted by a signal. If more than one thread is waiting to send when space becomes available in the message queue and the Process Scheduling option is supported, then the thread of the highest priority that has been waiting the longest is unblocked to send its message. Otherwise, it is unspecified which waiting thread is unblocked. If the specified message queue is full and `O_NONBLOCK` is set in the message queue description associated with `mqdes`, the message is not queued and `mq_send()` returns an error.

Upon successful completion, `mq_send()` returns a value of 0. Otherwise, no message is enqueued, the function returns -1, and `errno` is set to indicate the error.

- `mqdes` specifies the message queue. (Output)
- `msg_ptr` points to the buffer. (Input)
- `msg_len` specifies the size of the buffer in bytes. (Input)
- `msg_prio` If `msg_prio` is not NULL, the priority of the selected message is stored in the location referenced by `msg_prio`. (Input)
Errors

EAGAIN
The O_NONBLOCK flag is set in the message queue description associated with mqdes, and the specified message queue is full.

EBADF
The mqdes argument is not a valid message queue descriptor open for writing.

EINTR
A signal interrupted the call to mq_send().

EINVAL
The value of msg_prio was outside the valid range.

EMSGSIZE
The specified message length, msg_len, exceeds the message size attribute of the message queue.

ENOSYS
mq_send() not supported.

Attributes

Operating System: OS-9
State: User
Threads: Safe
Re-entrant: No
Standards: POSIX

Library

mq.l

See Also

mq_receive
mq_setattr
Syntax

```c
#include <mqueue.h>

int mq_setattr(
    mqd_t mqdes,  // specified the message queue. (Input)
    const struct mq_attr *mqstat,  // mq_maxmsg, mq_msgsize, and mq_curmsgs members of the mq_attr structure are ignored by mq_setattr().
    struct mq_attr *omqstat);  // omqstat is non-NULL, the function mq_setattr() stores, in the location referenced by omqstat, the previous message queue attributes and the current queue status. (Output)
```

Description

`mq_setattr()` is used to set attributes associated with the open message queue description referenced by the message queue descriptor specified by `mqdes`.

The values of the `mq_maxmsg`, `mq_msgsize`, and `mq_curmsgs` members of the `mq_attr` structure are ignored by `mq_setattr()`.

OS-9 specifies the following user-defined flag for use with `mq_setattr()` and `mq_setattr()`:

- `_MQ_O_NOTIFY_IMMEDIATE`. When this flag is specified by the `mq_setattr()` function (`mq_flags` field of `mqstat` (input) parameter) then calls to either `_mq_notify_write()` or `mq_notify()` will notify immediately if appropriate.

(Message queue nonfull for `_mq_notify_write()`, and message queue non-empty for `mq_notify()`) This extension provides message queue notification consistency with other OS-9 services such the "send signal on data ready" concept.

Upon successful completion, `mq_setattr()` returns a value of 0 and the attributes of the message queue are changed as specified. Otherwise, the message queue attributes are unchanged, and the function returns a value of -1 and sets `errno` to indicate the error.

The message queue attributes corresponding to the following members defined in the `mq_attr` structure are set to the specified values upon successful completion of `mq_setattr()`:

- `mq_flags` The value of this member is the bitwise logical OR of zero or more of `O_NONBLOCK` and any implementation-defined flags.
- `mqdes` specifies the message queue. (Input)
- `omqstat` If `omqstat` is non-NULL, the function `mq_setattr()` stores, in the location referenced by `omqstat`, the previous message queue attributes and the current queue status. (Output)

These values are the same as would be returned by a call to `mq_getattr()` at that point.
Errors
EBADF
   The mqdes argument is not a valid message queue descriptor.
ENOSYS
   mq_setattr() not supported.

Attributes
Operating System: OS-9
State: User
Threads: Safe
Re-entrant: No
Standards: POSIX

Library
mq.l

See Also
mq_open
mq_send
Syntax

```
#include <mqueue.h>
int mq_unlink(const char *name);
```

Description

`mq_unlink()` removes the message queue named by the pathname `name`.

After a successful call to `mq_unlink()` with `name`, a call to `mq_open()` with `name` fails if the flag `O_CREAT` is not set in flags. If one or more processes have the message queue open when `mq_unlink()` is called, destruction of the message queue is postponed until all references to the message queue have been closed. Calls to `mq_open()` to re-create the message queue may fail until the message queue is actually removed. However, the `mq_unlink()` call need not block until all references have been closed; it may return immediately.

Upon successful completion, `mq_unlink()` returns a value of 0. Otherwise, the named message queue is not changed by this function call, and the function returns a value of -1 and sets `errno` to indicate the error.

`name` is the pathname. (Input)

Errors

- **EACCES**  
  Permission is denied to unlink the named message queue.
- **ENAMETOOLONG**  
  The length of the name string exceeds \{NAME_MAX\} while \{_POSIX_NO_TRUNC\} is in effect.
- **ENOENT**  
  The named message queue does not exist.
- **ENOSYS**  
  `mq_unlink()` not supported.

Attributes

- Operating System: OS-9
- State: User
- Threads: Safe
- Re-entrant: No
- Standards: POSIX

Library

mq.l

See Also

`mq_close`  
`mq_open`
munlink()  
Unlink from Module

Syntax
#include <module.h>

int munlink(mh_com *module);

Description
munlink() informs the system that the process no longer requires the specified
module. The module’s link count is decremented, and the module is removed from
memory if the link count reaches zero.

module
    is a pointer to the module. (Input)

    It must have been a pointer returned by modcload(), modload(), modloadp(),
    modlink(), _os_link(), _os_load(), or _os_loadp(). On error, -1 is returned
    and the appropriate error code is placed in the global variable errno.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
modcload()
modlink()
modload()
modloadp()
_os_link()
_os_load()
_os_loadp()
_os_unlink()
F$Unlink
F_UNLINK

OS-9 for 68K Technical Manual
OS-9 Technical Manual
munload()

Unload Module

Syntax

```c
#include <module.h>
int munload(
    const char *name,
    int typelang);
```

Description

`munload()` informs the system that the process no longer requires the module `name` which has a type/language of `typelang`. The module's link count is decremented. The module is removed from memory when the link count reaches zero.

On error, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`munload()` differs from `munlink()` in that it unlinks by module name rather than module pointer. Attempting to unlink a module by performing a link to a module by name to determine its address, then unlinking it twice does not work because the first unlink removes the module from the process’ address space.

`name` is a pointer to the name of the module. (Input)

`typelang` specifies the type and language of the module. (Input)

If `typelang` is zero, any module may be unloaded. Only the lower 16 bits of `typelang` are used.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

`munlink()`  
`_os_link()`  
`_os_load()`  
`_os_unload()`  
F$UnLoad  
`OS-9 for 68K Technical Manual`

F_UNLOAD  
`OS-9 Technical Manual`
**offsetof()**

Expand Value to Integral Constant Expression

**Syntax**

```c
#include <stddef.h>
int offsetof(type, mbr_desig);
```

**Description**

`offsetof()` expands to an integral constant expression. The integral constant expression has a type of `size_t`, the value of which is the offset in bytes to the structure member from the beginning of the structure.

`mbr_desig` specifies the member name of the structure for which you want to compute the offset. If the member is a bit field, the behavior is undefined.

`type` specifies the type of the structure. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

clib.1
open()  
Open File

Syntax

```c
#include <modes.h>
int open(
    const char *name,
    int mode);
```

Description

`open()` opens an existing file.

`open()` returns a path number identifying the file when I/O is performed. If the open fails, -1 is returned and the appropriate error code is placed in the global variable `errno`.

name

is a pointer to the name of the file. (Input)

mode

specifies the access mode for the file. (Input)

The values for `mode` are defined in the `modes.h` header file. Only the lower 16 bits of `mode` are used.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

close()  
creat()  
_os_create()  
_os_open()  
I$Open  
_os_open()

OS-9 for 68K Technical Manual
OS-9 Technical Manual
opendir()
Open Directory

Syntax

```
#include <dir.h>

DIR *opendir(const char *filename);
```

Description

`opendir()` opens the specified directory and associates a directory stream with it. `opendir()` returns a pointer identifying the directory stream in subsequent operations. The pointer `NULL` is returned if `filename` cannot be accessed or if it cannot allocate enough memory to hold the entire `DIR` structure.

`filename` is a pointer to the name of the directory. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library

`sys_clib.l`

See Also

`closedir()`
`readdir()`
`rewinddir()`
`seekdir()`
`telldir()`
Acquire Ownership of Resource Lock

Syntax

```c
#include <lock.h>
#include <types.h>
error_code _os_acqlk(
    lk_desc *lock,
    signal_code *signal);
```

Description

_os_acqlk() acquires ownership of a resource lock (that is, it attempts to gain exclusive access to a resource).

If the lock is not owned by another process, the calling process is granted ownership and the call returns without error.

If the lock is already owned, the calling process is suspended and inserted into a waiting queue for the resource based on relative scheduling priority.

When ownership of the lock is released, the next process in the queue is granted ownership and is activated. The activated process returns from the _os_acqlk() call without error. If a process received a signal while waiting on a lock, the process is activated without gaining ownership of the lock. The process returns from the _os_acqlk() call with an EOS_SIGNAL error code and the signal code returned in the signal pointer.

If a waiting process receives an S_WAKEUP signal, the signal code does not register and is zero.

lock

is a pointer to the lk_desc structure of the lock that you are trying to acquire. (Input)

signal

is a pointer to the location where _os_acqlk() stores the signal that prematurely terminated the acquisition of the lock, if applicable. (Output)

Attributes

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library

lock.l
### See Also

<table>
<thead>
<tr>
<th>Function</th>
<th>OS-9 Technical Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ACQLK</td>
<td>OS-9 Technical Manual</td>
</tr>
<tr>
<td>F_CAQLK</td>
<td>OS-9 Technical Manual</td>
</tr>
<tr>
<td>F_CRLK</td>
<td>OS-9 Technical Manual</td>
</tr>
<tr>
<td>F_DDLK</td>
<td>OS-9 Technical Manual</td>
</tr>
<tr>
<td>F_DELLK</td>
<td>OS-9 Technical Manual</td>
</tr>
<tr>
<td>F_RELLK</td>
<td>OS-9 Technical Manual</td>
</tr>
<tr>
<td>F_WAITLK</td>
<td>OS-9 Technical Manual</td>
</tr>
</tbody>
</table>
_os9_alarm_atdate()

Send Signal at Gregorian Date/Time

Syntax

```c
#include <alarm.h>
#include <types.h>
error_code _os9_alarm_atdate(
    alarm_id *alrm_id,
    signal_code signal,
    u_int32 time,
    u_int32 date);
```

Description

_os9_alarm_atdate() sends a signal to the caller at a specific date and time. The alarm signal is sent anytime the system date/time becomes greater than or equal to the alarm time.

_os9_alarm_atdate() only allows you to specify the time to the nearest second. However, it does adjust if the system’s date and time have changed (using _os9_setime()).

alrm_id
is a pointer to the location where _os9_alarm_atdate() stores the alarm ID. (Output)

signal
specifies the signal code. (Input)

date
is in the format yyyyymmdd. (Input)

time
is in the format 00hmmss. (Input)

Attributes

Operating System: OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library

os_lib

See Also

F$Alarm OS-9 for 68K Technical Manual
_os_alarm_atime()
Send Signal at Specified Time

Syntax
#include <alarm.h>
#include <types.h>
error_code _os_alarm_atime(
    alarm_id *alrm_id,
    signal_code signal,
    u_int32 time);

Description
_os_alarm_atime() sends a signal when the system time reaches or passes the specified time. The time value is considered to be an absolute value in seconds since 1 January 1970 Greenwich Mean Time.

alrm_id
is a pointer to the location where _os_alarm_atime() stores the alarm ID. (Output)

signal
is the signal code of the signal to send. (Input)

time
specifies the time. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_alarm_set()
_os9_alarm_atdate()
_os9_alarm_atjul()
F$Alarm
F_ALARM

OS-9 for 68K Technical Manual
OS-9 Technical Manual
_os9_alarm_atjul()

Send Signal at Julian Date/Time

Syntax
#include <alarm.h>
#include <types.h>
error_code _os9_alarm_atjul(
    alarm_id *alrm_id,
    signal_code signal,
    u_int32 time,
    u_int32 date);

Description
_os9_alarm_atjul() sends a signal to the caller at a specific Julian date and time. The alarm signal is sent anytime the system date/time becomes greater than or equal to the alarm time.

_os9_alarm_atjul() only allows you to specify the time to the nearest second. However, it does adjust if the system’s date and time have changed (using _os9_setime()).

alrm_id
    is a pointer to the location where _os9_alarm_atjul() stores the alarm ID. (Output)

signal
    specifies the signal code. (Input)

time
    is the seconds after midnight. (Input)

date
    is the Julian day number. (Input)

Attributes
Operating System:      OS-9 for 68K
State:                User
Threads:              Safe
Re-entrant:           Yes

Library
os_lib.l

See Also
F$Alarm

OS-9 for 68K Technical Manual
_os_alarm_cycle()

Send Signal at Specified Time Intervals

Syntax

```c
#include <alarm.h>
#include <types.h>
error_code _os_alarm_cycle(
    alarm_id *alrm_id,
    signal_code signal,
    u_int32 time);
```

Description

_os_alarm_cycle() sends a signal after the specified time interval has elapsed and then resets the alarm. This provides a recurring periodic signal. The time interval may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. The minimum time interval allowed is 2 system clock ticks.

*alrm_id* is a pointer to the location where _os_alarm_cycle() stores the alarm ID. (Output)

*signal* is the signal code of the signal to send. (Input)

*time* specifies the time interval between signals. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_alarm_set()

F$Alarm OS-9 for 68K Technical Manual
F_ALARM, A_CYCLE OS-9 Technical Manual
Syntax
#include <alarm.h>
#include <types.h>

error_code _os_alarm_delete(
    alarm_id alrm_id);

Description
_os_alarm_delete() removes a cyclic alarm or any alarm that has not expired.

alrm_id specifies the alarm to remove. (Input)
If alrm_id is zero, all pending alarm requests for the owner are removed.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F$Alarm OS-9 for 68K Technical Manual
F_ALARM, F_DELET OS-9 Technical Manual
Syntax

```c
#include <alarm.h>
#include <types.h>

erreur_code _os_alarm_reset(
    alarm_id alrm_id,
    signal_code signal,
    u_int32 time);
```

Description

_os_alarm_reset() sets an existing alarm to send after the specified time interval has elapsed.

- **alrm_id**: is the alarm ID to reset. (Input)
- **signal**: is the signal code of the signal to send. (Input)
- **time**: may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. (Input)

Usually, the minimum time interval allowed is 2 clock ticks.

Attributes

- Operating System: OS-9
- State: User
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

- _os_alarm_set()
- F$Alarm
- F_ALARM, A_RESET

OS-9 for 68K Technical Manual
OS-9 Technical Manual
**Syntax**

```c
#include <alarm.h>
#include <types.h>
error_code _os_alarm_set(
    alarm_id *alrm_id,
    signal_code signal,
    u_int32 time);
```

**Description**

This function sends a signal after specified time interval has elapsed.

*alrm_id*  
is a pointer to the location where `_os_alarm_set()` stores the alarm ID.  
(Output)

*signal*  
is the signal code of the signal to send. (Input)

*time*  
may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. (Input)

Usually, the minimum time interval allowed is 2 system clock ticks.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
- Re-entrant: Yes

**Library**

`os_lib.l`

**See Also**

- `F$Alarm`
- `F_ALARM, A_SET`
  
  - *OS-9 for 68K Technical Manual*
  - *OS-9 Technical Manual*
Syntax

```c
#include <io.h>
#include <types.h>
error_code _os_alias(
    const char *alias_name,
    const char *real_name);
```

Description

_os_alias() creates an alternate name for a device pathlist. Processes can then reference a specific device pathlist with a shorter or more convenient name.

⚠️ Do not use a real device name as alias_name.

**alias_name**

is a pointer to the alternate name. (Input)

**real_name**

is a pointer to the actual device name; it must exist. (Input)

OS-9 does not validate its existence.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

I_ALIAS

OS-9 Technical Manual
_syntax

#include <memory.h>
#include <types.h>

error_code _os9_allbit(
    u_int32 bit_number,
    u_int32 count,
    void *address);

description

_os9_allbit() sets bits in a buffer that represents a bit map. Bit numbers range from 0 to n - 1, where n is the number of bits in the bit map.

bit_number
    specifies the bit number of the first bit to set. (Input)
    Only the lower 16 bits of bit_number are used.

count
    specifies the number of bits to set. (Input)
    Only the lower 16 bits of count are used.

address
    is a pointer to the base address of a bit map. (Input/Output)

attributes

Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

library

os_lib.1

see also

_os9_schbit()
_os9_delbit()
$AllBit

OS-9 for 68K Technical Manual
Allocate Fixed-Length Block of Memory

Syntax

```c
#include <process.h>
#include <types.h>
error_code _os9_allpd(
    void *table,
    u_int16 *number,
    void **ptr);
```

Description

_os9_allpd() allocates fixed-length blocks of system memory. It allocates and initializes (to zeros) a block of storage and returns its address.

- **table**: is a pointer to the process/path table. (Input)
- **number**: is a pointer to the location where _os9_allpd() stores the process/path number. (Output)
- **ptr**: is a pointer to the location where _os9_allpd() stores the pointer to the process/path descriptor. (Output)

Attributes

- Operating System: OS-9 for 68K
- State: System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

_os9_findpd()
_os9_retpd()

F$AllPD

OS-9 for 68K Technical Manual
_os_alltsk()
Allocate Task

Syntax
#include <process.h>
#include <types.h>
error_code _os_alltsk(pr_desc *proc_desc);

Description
_os_alltsk() initializes the protection hardware for a newly activated process. 
_os_alltsk() only performs a useful function on MMU/SSM systems. On non-MMU/SSM
systems, _os_alltsk() simply returns without initializing the protection hardware.

proc_desc is a pointer to the process descriptor. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F$AllTsk OS-9 for 68K Technical Manual
F_ALLTSK OS-9 Technical Manual
Allocate Process Descriptor

Syntax
#include <process.h>
#include <types.h>

error_code _os_alocproc(
    process_id *proc_id,
    pr_desc **proc_desc);

Description
_os_alocproc() allocates and initializes a process descriptor. The descriptor’s address is stored in the process descriptor table.

proc_id
    is a pointer to the location where _os_alocproc() stores the process ID. (Output)

proc_desc
    is a pointer to the location where _os_alocproc() stores the pointer to the process descriptor. (Output)

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  System
Threads:  Safe
Re-entrant:  Yes

Library
os_lib.l

See Also
F$AllPrc  OS-9 for 68K Technical Manual
Set Alternate Working Module Directory

Syntax

```c
#include <moddir.h>
#include <types.h>
error_code _os_altmdir(const char *name);
```

Description

_os_altmdir() establishes an alternate working module directory for a process.

name specifies the name of the module directory. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os.lib

See Also

F_ALTMDIR OS-9 Technical Manual
Syntax
#include <process.h>
#include <types.h>
error_code _os_aproc(process_id proc_id);

Description
_os_aproc() inserts the process specified by proc_id (input) into the active process queue so that it may be scheduled for execution.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System and User (User only for OS-9)
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_nproc()
F$AProc
F_ALLPRC
_os_suspend()
Syntax
#include <process.h>
#include <types.h>

error_code _os9_aproc(pr_desc *proc_desc);

Description
_os9_aproc() inserts a process into the active process queue so that it may be scheduled for execution.

proc_desc is a pointer to the process descriptor. (Input)

Attributes
Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1

See Also
_os_nproc()
F$AProc OS-9 for 68K Technical Manual
**_os_attach()**

**Attach New Device to System**

**Syntax**

```c
#include <io.h>
#include <modes.h>
#include <types.h>

error_code _os_attach(
    const char *name,
    u_int32 mode,
    dev_list **dev_tbl);
```

**Description**

_os_attach() causes a new I/O device to become known to the system or verifies that the device is already attached.

All available access modes are defined in the following location:
MWOS\OS\SRC\DEFS\modes.h

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the OS-9 Technical Manual.

If the descriptor name is found, _os_attach() links to the I/O device’s file manager and device driver.

If the device was already attached, _os_attach() returns a pointer to the pointer to the device table entry containing the address of the I/O device’s file manager and device driver in dev_tbl.

If the device is not already attached, _os_attach() creates a new device table entry and initializes the device. It returns a pointer to the pointer to the entry in dev_tbl.

name is a pointer to the I/O device’s name. (Input)

mode is the access mode. It can be FAM_READ, FAM_WRITE, S_IREAD, or S_IWRITE. mode may be used to verify that subsequent read and/or write operations are permitted. (Input)

Only the lower 16 bits of mode are used.

dev_tbl is a pointer to the location where _os_attach() stores the pointer to the device list entry. (Output)
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_detach()
I$Attach OS-9 for 68K Technical Manual
I_ATTACH OS-9 Technical Manual
Syntax

_For OS-9 for 68K:_
#include <cache.h>* for OS-9 for 68K systems
#include <regs.h>
#include <types.h>

error_code _os_cache(u_int32 control);

_For OS-9:_
#include <regs.h>
#include <types.h>

error_code _os_cache(
    u_int32 control,
    void *addr,
    u_int32 size);

Description

_os_cache() performs operations on the system instruction and/or data caches, if there are any.

Any program that builds or changes executable code in memory must flush the instruction cache with _os_cache() before executing the new code. This is necessary because the hardware instruction cache is not updated by data (write) accesses and may contain the unchanged instruction(s). For example, if a subroutine builds a system call on its stack, you must execute the _os_cache() call to flush the instruction cache before executing the temporary instructions.

Refer to your operating system technical manual for a description of the possible values for control.

control
If control is zero, the system instruction and data caches are flushed. (Input)
Only non-super-group, user-state processes may perform this operation.

addr
specifies the target address for the flush operation. (Input)

size
indicates the size of the target memory area to be flushed. (Input)
The cache should be flushed before trying to disable the data cache.

Only super-group processes may perform precise operations of `F$CCtl`.

**Attributes**
- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**
- `os_lib.l`

**See Also**
- `_os_scache()`
- `F$CCtl` *OS-9 for 68K Technical Manual*
- `F_CCCTL` *OS-9 Technical Manual*
Syntax

```c
#include <lock.h>
#include <types.h>
error_code _os_caqlk(lk_desc *lock);
```

Description

_os_caqlk() is used to conditionally acquire ownership of a resource lock. If another process does not own the lock, the calling process is granted ownership and the call returns without error.

If the lock is already owned, the calling process is not suspended. Instead, it returns from the _os_caqlk() call with an EOS_NOLOCK error and is not granted ownership of the resource lock.

lock

is a pointer to the lock. (Input)

Attributes

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library

lock.l

See Also

F_ACQLK OS-9 Technical Manual
F_CAQLK OS-9 Technical Manual
F_CRLK OS-9 Technical Manual
F_DDLK OS-9 Technical Manual
F_DELLK OS-9 Technical Manual
F_RELLK OS-9 Technical Manual
F_WAITLK OS-9 Technical Manual
Execute New Primary Module

Syntax

```c
#include <process.h>
#include <module.h>
#include <types.h>

error_code _os_chain(
    u_int32 priority,
    u_int32 path_cnt,
    const char *mod_name,
    const void *params,
    u_int32 param_size,
    u_int32 mem_size,
    u_int32 type_lang);
```

Description

_os_chain() executes a new program without the overhead of creating a new process. It is functionally similar to a Fork command followed by an Exit. _os_chain() effectively resets the calling process’ program and data memory areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

OS-9 for 68K: _os_chain() never returns to the calling process.

OS-9: an error is returned only if there is not enough memory to hold the parameters. If an error occurs during the chain, it is returned as an exit status to the parent of the process performing the chain.

On OS-9, a threaded process may only chain from the main() thread and may not chain if multiple threads are active in the process.

priority

is the priority of the process. (Input)

Only the lower 16 bits of priority are used.

path_cnt

specifies the number of I/O paths to copy. (Input)

Only the lower 16 bits of path_cnt are used.

mod_name

is a pointer to the new program to execute. (Input)

params

is a pointer to the parameter block. (Input)
param_size
  specifies the size of the parameter block. (Input)

mem_size
  specifies the additional memory size in bytes. (Input)

type_lang
  specifies the desired module type/language. (Input)

type_lang
  must be either program/object (MT_PROGRAM, ML_OBJECT) or zero (for any).
  Only the lower 16 bits of type_lang are used.

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes

Library
os_lib.l

See Also
_os_fork()
_os_load()

F$Chain  OS-9 for 68K Technical Manual
F_CHAIN  OS-9 Technical Manual
Execute New Primary Module Given Pointer to Module

Syntax

```c
#include <process.h>
#include <types.h>
error_code _os_chainm(
    u_int32 priority,
    u_int32 path_cnt,
    mh_com *mod_head,
    void *params,
    u_int32 param_size,
    u_int32 mem_size);
```

Description

_os_chainm() executes a new program without the overhead of creating a new process. It is functionally similar to a fork command followed by an exit. _os_chainm() effectively resets the calling process' program and data memory areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

_os_chainm() is similar to _os_chain(). However, it is passed a pointer to the module in memory to chain to instead of the module name.

An error is returned only if there is not enough memory to hold the parameters. If an error occurs during the _os_chainm(), it is returned as an exit status to the parent of the process performing the _os_chainm().

A threaded process may only chain from the main() thread and may not chain if multiple threads are active in the process.

**priority**

is the priority of the process. (Input)

Only the lower 16 bits of priority are used.

**path_cnt**

is the number of I/O paths to copy. (Input)

Only the lower 16 bits of path_cnt are used.

**mod_head**

is a pointer to the module header. (Input)

**params**

is a pointer to the parameter block. (Input)

**param_size**

specifies the size of the parameter block. (Input)
mem_size
    specifies the additional memory size in bytes. (Input)

Attributes
Operating System:   OS-9
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library
os_lib.l

See Also
F_CHAIN              OS-9 Technical Manual
F_CHAINM             OS-9 Technical Manual
F_EXIT               OS-9 Technical Manual
F_FORK               OS-9 Technical Manual
F_LOAD               OS-9 Technical Manual
Syntax

#include <modes.h>
#include <types.h>
error_code _os_chdir(
    const char *name,
    u_int32 mode);

Description

_os_chdir() changes a process’ working directory to the directory file specified by the pathlist. You may change the execution or the data directory (or both), depending on the access mode you specify. The specified file must be a directory file, and the caller must have access permission for the specified mode.

All available access modes are defined in the following location:
MWOS\OS\SRC\DEFS\modes.h
- For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the OS-9 Technical Manual.

name
    is a pointer to the pathlist. (Input)

mode
    specifies the access mode. (Input)
    Only the lower 16 bits of mode are used.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

I$ChgDir
I_CHDIR

OS-9 for 68K Technical Manual
OS-9 Technical Manual
**Syntax**

```c
#include <process.h>
#include <types.h>
error_code _os_chkmem(
    u_int32 size,
    u_int32 mode,
    void *mem_ptr,
    pr_desc *proc_desc);
```

**Description**

`_os_chkmem()` determines if a process has access to a specified memory block. `_os_chkmem()` only performs a useful function on MMU systems. On non-MMU systems, `_os_chkmem()` simply returns without checking the memory block’s accessibility.

- **size** specifies the size of the memory area. (Input)
- **mode** specifies the permissions to check. (Input)
  Only the lower 16 bits of `mode` are used.
- **mem_ptr** is a pointer to the beginning of the memory to check. (Input)
- **proc_desc** is a pointer to the process descriptor of the target process. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

`os_lib.l`

**See Also**

- F$ChkMem
- F_CHKMEM
  - OS-9 for 68K Technical Manual
  - OS-9 Technical Manual
_os_chmdir()
Change Process’ Current Module Directory

Syntax
#include <moddir.h>
#include <types.h>
error_code _os_chmdir(const char *name);

Description
_os_chmdir() changes a process’ current module directory to the directory specified by name.

name is a pointer to a full pathlist or a pathlist relative to the current module directory. (Input)

To change to the system’s root module directory, specify a slash (/) for name.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
_os_close()

Close Path to File/Device

Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os_close(path_id path);
```

Description

_os_close() terminates the I/O path specified by path.

Path

is the path ID returned from a previous call to _os_open(), _os_create(), or _os_dup(). (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_create()
_os_detach()
_os_dup()
_os_exit()
_os_open()

I$Close OS-9 for 68K Technical Manual
I_CLOSE OS-9 Technical Manual
Syntax
#include <process.h>
#include <types.h>
error_code _os_clrsigs(process_id proc_id);

Description
_os_clrsigs() removes any queued signals that have been sent to the target process.

proc_id
    specifies the target process ID number. (Input)

Attributes
Operating System:    OS-9
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library
os_lib.l

See Also
_os_sigmask()
Syntax
#include <moddir.h>
#include <types.h>
error_code _os_cmdperm(
    const char *name,
    u_int32 perm);

Description
_os_cmdperm() changes the access permissions of an existing module directory. This makes it possible to restrict access to a particular module directory.

name
is a pointer to the name of the existing module directory. (Input)

perm
specifies the new permissions. (Input)

Only the lower 16 bits of perm are used.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
**Syntax**

```c
#include <strings.h>
#include <types.h>

error_code _os_cmpnam(
    const char *pattern,
    const char *string,
    u_int32 length,
    int32 *cmp_result);
```

**Description**

_**os_cmpnam()** compares a target name to a source pattern to determine if they are equal. **os_cmpnam()** is not case-sensitive; it does not differentiate between upper and lower case characters.

**OS-9 for 68K:** If the error **EOS_PNNF** is returned, the strings differ and the integer at the address passed in **cmp_result** is not changed.

**OS-9:** If the error **EOS_DIFFER** is returned, the strings differ and the integer at **cmp_result** has the following meaning:

<table>
<thead>
<tr>
<th>Integer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>The target name is less than the pattern string.</td>
</tr>
<tr>
<td>negative</td>
<td>The target string is greater than the pattern string.</td>
</tr>
</tbody>
</table>


- **pattern**
  - is a pointer to the pattern string. (Input)
  - Two wildcard characters are recognized in the pattern string:
    - A question mark (?) matches any single character.
    - An asterisk (*) matches any string.

- **string**
  - is a pointer to the target name string. (Input)
  - The target name must be terminated by a null byte.

- **length**
  - specifies the length of the pattern string. (Input)
cmp_result

is a pointer to the location where _os_cmpnam() stores the lexicographic result
of the comparison. (Output)

_os_cmpnam()

returns zero if the target string is the same as the pattern string.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

F$CmpNam OS-9 for 68K Technical Manual
F_CMPNAM OS-9 Technical Manual
_os_config()

Configure an Element

Syntax

```c
#include <sysglob.h>
error_code _os_config(
    u_int32 code,
    void *param);
```

Description

_os_config() is a wildcard call that configures elements of the operating system which may or may not be process specific. It dynamically reconfigures operating system resources at runtime. The target resources may be system-wide resources or they may be process-specific, depending on the nature of the configuration call being made.

- **code** is the configuration command code. (Input)
- **param** is a pointer to any additional parameters required by the specified configuration function. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

See Also

_os_ioctl()
get pointer to I/O process descriptor

Syntax
#include <process.h>
#include <io.h>
#include <types.h>
error_code _os_cpy_ioproc(
    process_id proc_id,
    void *buffer,
    u_int32 count);

Description
_os_cpy_ioproc() copies the I/O process descriptor for the specified process into a buffer.

process_id
    is the process ID. (Input)

buffer
    is a pointer to the buffer in which to copy the I/O process descriptor. The format of the data copied is specified by the io_proc structure in io.h. (Output)

count
    specifies the number of bytes to copy. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
Syntax

```c
#include <process.h>
#include <types.h>

error_code _os_cpymem(
    process_id proc_id,
    void *from,
    void *to,
    u_int32 count);
```

Description

_os_cpymem() copies external memory into your buffer for inspection.

_os_cpymem() can copy portions of the system’s address space. This is especially helpful in examining modules. Any memory in the system may be viewed in this way.

proc_id
    specifies the process ID of the owner of the external memory. (Input) Using a process ID of 0 allows access to the system memory.

from
    is a pointer to the external process’ memory to copy. (Input)

to
    is a pointer to the caller’s destination buffer. (Output)

count
    is the number of bytes to copy. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

=os_move()
_os_crc()
Generate CRC

Syntax
#include <module.h>
#include <types.h>
error_code _os_crc(
    void *start,
    u_int32 count,
    int32 *accum);

Description
_os_crc() generates or checks the CRC (cyclic redundancy check) values of sections of memory. Compilers, assemblers, and other module generators use _os_crc() to generate a valid module CRC.

If the CRC of a new module is to be generated, the CRC is accumulated over the module (excluding the CRC). The accumulated CRC is complemented and stored in the correct position in the module.

The CRC is calculated over a specified number of bytes starting at the source address. It is not necessary to cover an entire module in one call, because the CRC may be accumulated over several calls. The CRC accumulator must be initialized to \(-1\) before the first _os_crc call for any particular module.

To generate the CRC of an existing module, the calculation must be performed on the entire module, including the module CRC. The CRC accumulator contains the CRC constant bytes if the module CRC is correct. The CRC constant is defined in module.h as CRCCON. The value is 0x00800fe3.

To generate the CRC for a module:
1. Initialize the accumulator to \(-1\).
2. Perform the CRC over the module.
3. Call F_CRC with a NULL value for start.
4. Complement the CRC accumulator.
5. Write the contents of the accumulator to the module.

The CRC value is three bytes long, in a four-byte field. To generate a valid module CRC, you must include the byte preceding the CRC in the check. You must initialize this byte to zero. For convenience, if a data pointer of zero is passed, the CRC is updated with one zero data byte. _os_crc always returns 0xff in the most significant byte of the accum parameter, so accum may be directly stored (after complement) in the last 4 bytes of a module as the correct CRC.

Refer to your operating system technical manual for more information on how your operating system uses CRCs for modules.
_os_crc()

start
  is a pointer to the section of memory. (Input)

count
  specifies the byte count for the section of memory. (Input)

accum
  is a returned value. It is a pointer to the CRC accumulator. (Output)

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes

Library
  os_lib.l

See Also
  F$CRC                     OS-9 for 68K Technical Manual
  F_CRC                     OS-9 Technical Manual
Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os_create(
    const char *name,
    u_int32 mode,
    path_id *path,
    u_int32 perm, ...);
```

Description

_os_create() creates a new file.

- On multifile devices, the new file is entered in the directory structure.
- On non-multifile devices, create is synonymous with open.

name

is a pointer to the path name of the new file. (Input)

mode

specifies the access mode. (Input)

If data is to be written to the file, mode must have the FAM_WRITE or S_IWRITE bit set. Only the lower 16 bits of mode are used.

All available access modes are defined in the following location:
MWOS\OS\SRC\DEFS\modes.h

path

is a pointer to the location where _os_create() stores the path number that identifies the file in subsequent I/O service requests until the file is closed. (Output)

perm

specifies the attributes to use for the new file. (Input)

Only the lower 16 bits of perm are used.

If either the FAM_SIZE bit or the S_ISIZE (initial file size) bit is set, you can also pass an initial file size estimate as a variable argument.

If either the FAM_NOCREATE bit or the S_IEXCL (exclusive) bit is set, an error is returned if name already exists. If neither FAM_NOCREATE nor S_EXCL are set, any existing file called name is deleted.
By default, if the pathlist specifies a file name that already exists, \_os\_create() recreates the file.

\* On OS-9 for 68K, the binding for this call exists for compatibility. If speed is an issue, use \_os\_9\_create().

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os\_lib.l

See Also

\_os\_attach()
\_os\_open()
\_os\_close()
\_os\_mkdir()

I\$Create
I\_CREATE

OS-9 for 68K Technical Manual
OS-9 Technical Manual
_os9_create()

Create Path to New File

Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os9_create(
    const char *name,
    u_int32 mode,
    path_id *path,
    u_int32 perm, ...);
```

Description

_os9_create() creates a new file.

- On multifile devices, the new file name is entered in the directory structure.
- On non-multifile devices, _create is synonymous with _open.

By default, if the pathlist specifies an existing file name, _os9_create() returns an error.

**name**  
is a pointer to the path name. (Input)

**mode**  
specifies the access mode. (Input)

If data is to be written to the file, _mode must have the write bit set. Only the lower 16 bits of _mode are used.

**path**  
is a pointer to the location where _os9_create() stores the path number. (Output)

**perm**  
specifies the attributes to use for the new file. (Input)

Only the lower 16 bits of _perm are used.

If either the _FAM_SIZE bit or the _S_ISIZE (initial file size) bit is set, you can also pass an initial file size estimate as a variable argument.

All available permissions are defined in the following location:  
MWOS\OS\SRC\DEFS\modes.h
Attributes
Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_attach()
_os_close()
_os_mkdir()
_os_open()
I$Create

OS-9 Technical Manual
_os_crlk()

Create New Resource Lock Descriptor

Syntax

```c
#include <lock.h>
#include <types.h>
error_code _os_crlk(lk_desc **lock);
```

Description

_os_crlk() creates a new resource lock descriptor. A resource lock descriptor is allocated and initialized to a free state (one not currently owned).

`lock`

is a pointer to the location where _os_crlk() stores the pointer to the lock descriptor. (Output)

`lock`

is used as a handle to perform further operations on the lock.

Attributes

- Operating System: OS-9
- State: System
- Threads: Safe
- Re-entrant: Yes

Library

lock.l

See Also

- F_ACQLK
- F_CAQLK
- F_CRLK
- F_DDLK
- F_DELLK
- F_RELLK
- F_WAITLK

OS-9 Technical Manual
_os_datmod()

Create Data Module

Syntax

```c
#include <module.h>
#include <types.h>
error_code _os_datmod(
    const char *mod_name,
    u_int32 size,
    u_int16 *attr_rev,
    u_int16 *type_lang,
    u_int32 perm,
    void **mod_exec,
    mh_data **mod_head);
```

Description

_os_datmod() creates a data module with the specified attribute/revision and clears the data portion of the module. The module is created and entered into the system module directory. _os_datmod() stores the module's attribute and revision at the location pointed to by attr_rev. It also stores the module's actual type and language at the location pointed to by the type_lang parameter.

Be careful not to alter the data module’s header or name string to prevent the module from becoming unknown to the system.

The created module contains at least size usable data bytes but may be somewhat larger. The module itself is larger by at least the size of the module header, CRC, and name string; it is rounded up to the nearest system memory allocation boundary.

mod_name
is a pointer to the module name string. (Input)

size
is the size of the data portion. (Input)

attr_rev
The module's attribute and revision word. (Input)

type_lang
The module's type and language word. (Output)

perm
specifies the access permissions for the module. (Input)
Only the lower 16 bits of perm are used.
mod_exec
is a pointer to the location where _os_datmod() stores the pointer to the
module data. (Output)

mod_head
is a pointer to the location where _os_datmod() stores the pointer to the
module header. (Output)

color
is the memory color type. If color is zero, MEM_ANY is the specified memory
type.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_setcrc()
=os_move()
F$_DatMod
F_DATMOD

OS-9 Technical Manual
OS-9 Technical Manual
Check for Deadlock Situation

Syntax

```
#include <process.h>
#include <types.h>

error_code _os_ddlk(process_id proc_id);
```

Description

_os_ddlk() checks for a deadlock situation between processes. A search for the current process (calling process) in the linked list of potential conflictors is begun from the process specified by proc_id.

If the calling process is already in the linked list of processes, an EOS_DEADLK error is returned to the caller.

If the process is not in the linked list, the current process is added to the list associated with proc_id.

_os_ddlk() is useful for preventing a deadlock situation when protecting multiple resources from simultaneous accesses. The deadlock list usually represents the list of processes waiting to acquire access to an associated resource.

proc_id specifies the process with which to begin the search. (Input)

Attributes

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library

lock.l

See Also

F_ACQLK  OS-9 Technical Manual
F_CAQLK  OS-9 Technical Manual
F_CRLK  OS-9 Technical Manual
F_DDLK  OS-9 Technical Manual
F_DELLK  OS-9 Technical Manual
F_RELLK  OS-9 Technical Manual
F_WAITLK OS-9 Technical Manual
Syntax

#include <memory.h>
#include <types.h>
error_code _os9_delbit(
    u_int32 bit_number,
    u_int32 count,
    void *address);

Description

_os9_delbit() clears bits in a buffer that represents a bit map. Bit numbers range from 0 to n – 1, where n is the number of bits in the bitmap.

bit_number
    specifies the bit number of the first bit to clear. (Input)
    Only the lower 16 bits of bit_number are used.

count
    specifies the number of bits to clear. (Input)
    Only the lower 16 bits of count are used.

address
    is a pointer to the base address of a bitmap. (Output)

Attributes

Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os9_allbit()
_os_cpymem()
_os9_schbit()
F$DelBit

OS-9 Technical Manual
_os_delete()

Delete File

Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os_delete(
    const char *name,
    u_int32 mode);
```

Description

_os_delete() deletes a file. The caller must have non-sharable write access to the file (the file may not already be open) or an error results.

Attempts to delete from non-multifile devices result in an error. The access mode specifies the data or execution directory (but not both) in the absence of a full pathlist. Only the lower 16 bits of mode are used.

name is a pointer to the file to delete. (Input)

mode specifies the access mode. (Input)

Possible values for mode include:

- FAM_READ
- FAM_WRITE
- FAM_EXEC
- S_IREAD
- S_IWRITE
- S_IEXEC

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

- _os_attach()
- _os_create()
- _os_detach()
- _os_open()
- I$Delete
- I_DELETE

OS-9 Technical Manual

OS-9 Technical Manual
_os_dellk()
Delete Existing Lock Descriptor

Syntax
#include <lock.h>
#include <types.h>
error_code _os_dellk(lk_desc *lock);

Description
_os_dellk() deletes an existing lock descriptor. It does not check for suspended processes still waiting to acquire the lock; an implementation using locks must release all processes waiting on a resource lock before deleting it. Failure to release suspended processes prior to deletion could result in system corruption.

lock is a pointer to the lk_desc structure for the lock to delete. (Input)

Attributes
Operating System: OS-9
State: User
Threads: Safe
Re-entrant: Yes

Library
lock.l

See Also
F_ACQLK OS-9 Technical Manual
F_CAQLK OS-9 Technical Manual
F_CRLK OS-9 Technical Manual
F_DDLK OS-9 Technical Manual
F_DELLK OS-9 Technical Manual
F_RELLK OS-9 Technical Manual
F_WAITLK OS-9 Technical Manual
Syntax

```c
#include <moddir.h>
#include <types.h>
error_code _os_delmdir(const char *name);
```

Description

_os_delmdir() deletes an existing module directory. The directory must be empty.

name is a pointer to the module directory. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`os_lib.l`

See Also

`F_DELMDIR`  
*OS-9 Technical Manual*
Syntax
#include <process.h>
#include <types.h>
error_code _os_deltsk(pr_desc *proc_desc);

Description
_os_deltsk() returns the process’ protection resources when the process terminates.
_os_deltsk() only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, _os_deltsk() simply returns without deallocating the process descriptor.

proc_desc
is a pointer to the process descriptor. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_alltsk()
_os_chkmem()
_os_protect()
_os_permit()
F$DelTsk
F_DELTSK

OS-9 for 68K Technical Manual
OS-9 Technical Manual
**_os_detach()**  
Remove Device from System

**Syntax**
```
#include <io.h>
#include <types.h>
error_code _os_detach(dev_list *dev_tbl);
```

**Description**

_os_detach() removes a device from the system device table if no other process is using the device.

**OS-9 for 68K:** If an invalid address is passed in `dev_tbl`, the system may crash or undergo severe damage.

`dev_tbl` is a pointer to the address of the device table entry. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

`os_lib.l`

**See Also**

- _os_attach()
- _os_close()

I_DETACH

OS-9 for 68K Technical Manual

OS-9 Technical Manual
Execute Debugged Program

Syntax

```c
#include <dexec.h>
#include <types.h>

error_code _os_dexec(
    process_id proc_id,
    dexec_mode mode,
    void *params,
    u_int32 *num_instr,
    u_int32 num_bpts,
    void **brk_pts,
    u_int32 *tot_instr,
    dexec_status *status,
    u_int32 *except,
    void **addr,
    error_code *exit_status);
```

Description

OS-9 for 68K:

_os_dexec() controls the execution of a suspended child process created by either _os9_dfork() or by _os_dfork(). OS-9 for 68k uses _os9_dfork() and OS-9 uses _os_dfork(). The process performing _os_dexec() is suspended, and its debugged child process is executed instead. This process terminates and control returns to the parent after the specified number of instructions have been executed, a breakpoint is reached, or an unexpected exception occurs. The parent and the child processes are never active at the same time.

An _os_dexit() call must be made for the resources (memory) of the debugged process to be returned.

Refer to your operating system technical manual for system-specific information.

Tracing is allowed through subroutine libraries, intercept routines, and OS-9 for 68K user-state trap handlers. This is not a problem, but may seem strange at times.

proc_id

is the process ID of the child to execute. (Input)

mode

specifies the debug mode. (Input)
params
  is the parameter list pointer. (Input)

num_instr
  points to the location where the number of instructions to execute is passed.
  (Input/Output)

_os_dexec()
  also stores the number of instructions not executed at the location pointed to
  by num_instr. If the integer at num_instr is zero, commands are executed
  continuously.

num_bpts
  specifies the number of breakpoints in the list. (Input)
  Maximum of 16 break points allowed.

brk_pts
  is a pointer to the pointer to the breakpoint list. (Input)

tot_instr
  points to where the number of instructions executed so far is passed.
  (Input/Output)

_os_dexec()
  also stores the updated total number of instructions at the location pointed to
  by tot_instr.

status
  is a pointer to the location where _os_dexec() stores the return status.
  (Output)

except
  is a pointer to the location where _os_dexec() stores the exception that the
  child received. (Output)

addr
  is a pointer to the location where _os_dexec() stores the violation address.
  (Output)

exit_status
  is a pointer to the location where _os_dexec() stores the exit status of the
  child. (Output)

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes
Library

os_lib.l

See Also

_os_dfork()
_os9_dfork()
_os_dexit()

F$DExec OS-9 for 68K Technical Manual
F_DEXEC OS-9 Technical Manual
F_DFORK OS-9 Technical Manual
# _os_dexit()

## Exit Debugged Program

### Syntax

```c
#include <dexec.h>
#include <types.h>
error_code _os_dexit(process_id proc_id);
```

### Description

_**os_dexit()**_ terminates a suspended child process created by _**os_dfork()**_. To allow examination of the child after its termination, normal termination by the child process does not release any resources.

**proc_id** is the process ID of the child to terminate. (Input)

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User
- **Threads:** Safe
- **Re-entrant:** Yes

### Library

- **os_lib.l**

### See Also

- _**os_exit()**_
- _**os_dfork()**_
- _**os9_dfork()**_
- _**os_dexec()**_
- F$DExit
- F_DEXIT

*OS-9 Technical Manual*
Syntax

```c
#include <dexec.h>
#include <types.h>

error_code _os_dfork(
    u_int32 priority,
    u_int32 path_cnt,
    const char *mod_name,
    const void *params,
    u_int32 param_size,
    u_int32 mem_size,
    process_id *proc_id,
    u_int32 type_lang,
    regs *reg_stack,
    fregs *fregstack);
```

Description

_os_dfork() creates a new process which becomes a child of the caller. It sets up the process’ memory, MPU registers, and standard I/O paths. In addition, _os_dfork() allows a debugger utility to create a process whose execution can be closely controlled. The created process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the _os_dexec() and _os_dexit() calls.

priority

is the priority of the new process. (Input)

Only the lower 16 bits of priority are used.

path_cnt

is the number of I/O paths for the child to inherit. (Input)

Only the lower 16 bits of path_cnt are used.

mod_name

is a pointer to the module name. (Input)

params

is a pointer to the parameter block. (Input)

param_size

specifies the size of the parameter block. (Input)
mem_size
    specifies any additional stack space to allocate. (Input)

mem_size
    is specified in bytes.

proc_id
    is a pointer to the location where _os_dfork() stores the child process ID. (Output)

type_lang
    specifies the desired type and language. (Input)
    Only the lower 16 bits of type_lang are used.

reg_stack
    is a pointer to the register buffer. (Input)

freg_stack
    is a pointer to the floating point register buffer. (Input)

Attributes
Operating System: OS-9
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_dexit()
_os_fork()
_os9_dfork()
_os_dexec()
F$DExit         OS-9 Technical Manual
F_DFORK         OS-9 Technical Manual
_os9_dfork()
Fork Process Under Control of Debugger

Syntax

```c
#include <dexec.h>
#include <types.h>
error_code _os9_dfork(
    u_int32 priority,
    u_int32 path_cnt,
    const char *mod_name,
    const void *params,
    u_int32 param_size,
    u_int32 mem_size,
    process_id *proc_id,
    u_int32 type_lang,
    void *reg_buf);
```

Description

=os9_dfork() creates a new process which becomes a child of the caller. It sets up the new process’ memory, MPU registers, and standard I/O paths. In addition, _os9_dfork() allows a debugger utility to create a process whose execution can be closely controlled. The process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the _os_dexec() and _os_dexit() system calls.

**priority**
- is the priority of the new process. (Input)
  - Only the lower 16 bits of priority are used.

**path_cnt**
- is the number of the I/O paths for a child to inherit. (Input)
  - Only the lower 16 bits of path_cnt are used.

**mod_name**
- is the module name pointer. (Input)

**params**
- is a pointer to an additional parameter block. (Input)

**param_size**
- is the size of the parameter block. (Input)

**mem_size**
- specifies any additional stack space to allocate. (Input)
mem_size
    is specified in bytes.

proc_id
    is a pointer to the location where _os9_dfork() stores the child process ID.  (Output)

type_lang
    specifies the desired type and language.  (Input)
    Only the lower 16 bits of type_lang are used.

reg_buf
    is a pointer to a copy of all child registers.  (Input)

Attributes
    Operating System:      OS-9 for 68K
    State:                 User and System
    Threads:               Safe
    Re-entrant:           Yes

Library
    os_lib.l

See Also
    _os_dexec()
    _os_dexit()
    _os_fork()
    F$DFork  OS-9 for 68K Technical Manual
Syntax

```c
#include <dexec.h>
#include <types.h>

error_code _os_dforkm(
    u_int32 priority,
    u_int32 path_cnt,
    mh_com *mod_head,
    void *params,
    u_int32 param_size,
    u_int32 mem_size,
    process_id *proc_id,
    regs *reg_stack,
    fregs *fregstack);
```

Description

_os_dforkm() creates a new process which becomes a child of the caller. It sets up the process’ memory, MPU registers, and standard I/O paths. In addition, _os_dforkm() allows a debugger utility to create a process whose execution can be closely controlled. The created process is not placed in the active queue, but is left in a suspended state. This allows the debugger to control its execution through the _os_dexec() and _os_dexit() calls. _os_dforkm() is similar to _os_dfork(). However, _os_dforkm() is passed a pointer to the module to fork rather than the module name.

priority

is the priority of the new process. (Input)

Only the lower 16 bits of priority are used.

path_cnt

is the number of I/O paths for the child to inherit. (Input)

Only the lower 16 bits of path_cnt are used.

mod_head

is a pointer to the module header. (Input)

params

is a pointer to the parameter block. (Input)

param_size

specifies the size of the parameter block. (Input)
mem_size
    specifies any additional stack space to allocate. (Input)

proc_id
    is a pointer to the location where _os_dforkm() stores the child process ID. (Output)

reg_stack
    is a pointer to the register buffer. (Input)

fregstack
    is a pointer to the floating point register buffer. (Input)

Attributes
Operating System:   OS-9
State:             User and System
Threads:           Safe
Re-entrant:        Yes

Library
os_lib.l

See Also
_os_fork()
_os_dexec()
_os_dexit()
_os_dfork()
**Syntax**

```c
#include <modes.h>
#include <types.h>
error_code _os_dup(
    path_id dup_path,
    path_id *new_path);
```

**Description**

_os_dup() returns a synonymous path number for an existing file or device.

_os_dup() always uses the lowest available path number. For example, if you perform an _os_close() on path #0 and an _os_dup() on path #4, path #0 is returned as the new path number. Therefore, service requests using either the old or new path numbers operate on the same file or device.

⚠️ It is not advised for more than one process to be performing I/O on the same path concurrently.

- **dup_path**
  - is the path number of the path to duplicate. (Input)

- **new_path**
  - is a pointer to the location where _os_dup() stores the new path number for the same path. (Output)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

- os_lib.l

**See Also**

- I$Dup
- I_DUP

*OS-9 for 68K Technical Manual*
*OS-9 Technical Manual*
_os_ev_allclr()

Wait for All Bits Defined by Mask to Become Clear

Syntax
#include <events.h>
#include <types.h>
error_code _os_ev_allclr(
    event_id ev_id,
    int32 *value,
    signal_code *signal,
    u_int32 mask);

Description
_os_ev_allclr() waits until an _os_ev_signal() occurs that clears all of the bits corresponding to the set bits in the mask.

ev_id
    identifies the event. (Input)

value
    is a pointer to the location where _os_ev_allclr() stores the actual event value. (Output)

signal
    is a pointer to the location where _os_ev_allclr() stores the signal code. (Output)

    If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.

mask
    specifies the activation mask. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F_EVENT
F_EVENT, EV_ALLCLR
F_EVENT, EV_SIGNAL
OS-9 Technical Manual
**Syntax**

```c
#include <events.h>
#include <types.h>
error_code _os_ev_allset(
    event_id ev_id,
    int32 *value,
    signal_code *signal,
    u_int32 mask);
```

**Description**

(OS-9) waits until an (OS-9) occurs that sets all of the bits corresponding to the set bits in the mask.

- **ev_id** identifies the event. (Input)
- **value** is a pointer to the location where (OS-9) stores the actual event value. (Output)
- **signal** is a pointer to the location where (OS-9) stores the signal code. (Output)
  
  If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.

- **mask** specifies the activation mask. (Input)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

- os_lib.l

**See Also**

- F_EVENT, EV_ALLSET OS-9 Technical Manual
- F_EVENT, EV_SIGNAL OS-9 Technical Manual
_os_ev_anyclr()

Wait for Event to Occur

Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_anyclr(
    event_id ev_id,
    int32 *value,
    signal_code *signal,
    u_int32 mask);
```

Description

_os_ev_anyclr() waits until an _os_ev_signal() occurs that clears any of the bits corresponding to the set bits in the mask.

- **ev_id** identifies the event. (Input)
- **value** is a pointer to the location where _os_ev_anyclr() stores the actual event value. (Output)
- **signal** is a pointer to the location where _os_ev_anyclr() stores the signal code. (Output)
  - If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.
- **mask** specifies the activation mask. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

- os_lib.l

See Also

- F_EVENT, EV_ANYCLR
- F_EVENT, EV_SIGNAL

OS-9 Technical Manual

OS-9 Technical Manual

OS-9 Technical Manual
_os_ev_anyset()

Wait for Event to Occur

Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_anyset(
    event_id ev_id,
    int32 *value,
    signal_code *signal,
    u_int32 mask);
```

Description

_os_ev_anyset() waits until an _os_ev_signal() occurs that sets any of the bits corresponding to the set bits in the mask.

**ev_id**
identifies the event. (Input)

**value**
is a pointer to the location where _os_ev_anyset() stores the actual event value. (Output)

**signal**
is a pointer to the location where _os_ev_anyset() stores the signal code. (Output)

If the process receives a signal while in the event queue, it is activated even though the event has not actually occurred.

**mask**
specifies the activation mask. (Input)

Attributes

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

Library

`os_lib.l`

See Also

- F_EVENT
- F_EVENT, EV_ANYSET
- F_EVENT, EV_SIGNAL

OS-9 Technical Manual
_os_ev_change()
Wait for Event to Occur

Syntax
#include <events.h>
#include <types.h>
error_code _os_ev_change(
    event_id ev_id,
    int32 *value,
    signal_code *signal,
    u_int32 mask,
    u_int32 pattern);

Description
_os_ev_change() waits for an event to occur. That is, it waits until an
_os_ev_signal() occurs that changes any of the bits corresponding to the set bits in
mask.

  ev_id
    identifies the event. (Input)

  value
    is a pointer to the location where _os_ev_change() stores the actual event
    value. (Output)

  signal
    is a pointer to the location where _os_ev_change() stores the signal code.
    (Output)

      If the process receives a signal while in the event queue, it is activated even
      though the event has not actually occurred.

  mask
    specifies the activation mask. (Input)

  pattern
    specifies the wait pattern. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes
_os_ev_change()

Library

os_lib.l

See Also

F_EVENT OS-9 Technical Manual
F_EVENT, EV_CHANGE OS-9 Technical Manual
F_EVENT, EV_SIGNAL OS-9 Technical Manual
Create New Event

Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_creat(
    int32 winc,
    int32 sinc,
    u_int32 perm,
    event_id *ev_id,
    const char *name,
    int32 value,
    u_int32 color);
```

Description

_os_ev_creat() allows you to create events dynamically as needed. When an event is created, an initial value is specified, as well as increments to apply each time the event is waited for or an event signal occurs. Subsequent _os_event calls use the returned ID number to refer to the created event.

- **winc** specifies the auto-increment value for _os_ev_wait(). (Input)
  Only the lower 16 bits of winc are used. (Input)

- **sinc** specifies the auto-increment value for _os_ev_signal(). (Input)
  Only the lower 16 bits of sinc are used.

- **perm** specifies the access permissions. (Input)
  Only the lower 16 bits of perm are used.
  OS-9 for 68K ignores perm.

- **ev_id** is a pointer to the location where _os_ev_creat() stores the event identifier. (Output)

- **name** is a pointer to the event name string. (Input)

- **value** specifies the initial event variable value. (Input)
color
    specifies the memory type for the event structure. (Input)
    Only the lower 16 bits of color are used.
    OS-9 for 68K ignores color.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_ev_delete()
_os_ev_signal()
_os_ev_wait()
_os9_ev_wait()
F$Event
F_EVENT
F_EVENT, EV_CREAT
F_EVENT, EV_SIGNAL
OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
_os_ev_delete()
Delete Existing Event

Syntax

```
#include <events.h>
#include <types.h>
error_code _os_ev_delete(const char *name);
```

Description

_os_ev_delete() removes an event from the system event table, freeing the entry for use by another event. An event may not be deleted unless its use count is zero.

name
is a pointer to the event’s name string. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_ev_creat()
F$Event OS-9 for 68K Technical Manual
F_EVENT OS-9 Technical Manual
F_EVENT, EV_DELET OS-9 Technical Manual
F_EVENT, EV_SIGNAL OS-9 Technical Manual
_os_ev_info()
Return Event Information

Syntax
#include <events.h>
#include <types.h>

OS-9 for 68K:
error_code _os_ev_info(
    event_id ev_id,
    u_int32 size,
    ev_infostr *buffer);

OS-9:
error_code _os_ev_info(
    event_id ev_id,
    u_int32 size,
    void *buffer);

Description
_os_ev_info() returns event information in the caller’s buffer. The returned information is defined by the ev_infostr event information structure defined in the events.h header file. On OS-9, the name of the event is appended to the end of the information structure; buffer and size must be large enough to accommodate the name of the target event.

_os_ev_info() returns the event information block for the first active event whose index is greater than or equal to this index. If no such event exists, an error is returned.

ev_id
    specifies the event index to use to begin the search. (Input)

size
    specifies the buffer size. (Input)

buffer
    is a pointer to the buffer containing the event information. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Library
os_lib.l

See Also
F$Event OS-9 for 68K Technical Manual
F_EVENT OS-9 for 68K Technical Manual
F_EVENT, EV_INFO OS-9 for 68K Technical Manual
F_EVENT, EV_SIGNL OS-9 for 68K Technical Manual
Syntax

```c
#include <events.h>
#include <types.h>
error_code _os9_ev_info(
    u_int32 *index,
    ev_infostr *buffer);
```

Description

_os9_ev_info() returns event information in the caller's buffer. The returned information is defined by the `ev_infostr` event information structure defined in `events.h`. _os9_ev_info() also stores the event index of the valid event found (if any) at the location pointed to by `index`.

`index`  
The event index to begin the search for a valid event is passed at the location pointed to by `index`. (Input)

`buffer`  
is a pointer to a buffer large enough to hold the event information. (Output)

Attributes

- Operating System: OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`os_lib.l`

See Also

`F$Event`  
OS-9 for 68K Technical Manual
**Syntax**

```c
#include <events.h>
#include <types.h>
error_code _os_ev_link(
    const char *name,
    event_id *ev_id);
```

**Description**

/os_ev_link() determines the ID number of an existing event. To keep the use count synchronized properly, you must perform an _os_ev_unlink() call when the event is longer be used.

*name*  
is a pointer to the event name string. (Input)

*ev_id*  
is a pointer to the location where _os_ev_link() stores the event identifier. (Output)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

`os_lib.l`

**See Also**

- _os_ev_unlink()
- F$Event  
- F_EVENT  
- F_EVENT, EV_LNK  
- F_EVENT, EV_UNLNK
_os_ev_pulse()

Signal Event Occurrence

Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_pulse(
    event_id ev_id,
    int32 *value,
    u_int32 actv_flag);
```

Description

_os_ev_pulse() signals an event occurrence. The event value is set to what is passed at the location `value`, and the signal auto-increment is not applied. Then, the _os_ev_signal() search routine is executed and the original event value is restored.

- **ev_id** identifies the event. (Input)
- **value** The value to pulse the event to is passed at the location pointed to by `value`. (Input/Output)
- _os_ev_pulse() also stores the event value prior to the pulse at the location pointed to by `value`.
- **actv_flag** specifies which process(es) to activate. (Input)
  - If `actv_flag` is one, all processes in range are activated.
  - If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

Library

os_lib.i
See Also

_roi_ev_signal()

F$Event
F_EVENT
F_EVENT, EV_PULSE
F_EVENT, EV_SIGNAL

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
**Syntax**

```c
#include <events.h>
#include <types.h>
error_code _os_ev_read(
    event_id ev_id,
    int32 *value);
```

**Description**

_os_ev_read() reads the value of an event without waiting for or affecting the event variable.

- **ev_id** identifies the event. (Input)
- **value** is a pointer to the location where _os_ev_read() stores the current event value. (Output)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

os_lib.l

**See Also**

- F$Event
- FEVENT
- FEVENT, EV_READ
- FEVENT, EV_SIGNAL
- OS-9 for 68K Technical Manual
- OS-9 Technical Manual
- OS-9 Technical Manual
- OS-9 Technical Manual
**Syntax**

```c
#include <events.h>
#include <types.h>
error_code _os_ev_set(
    event_id ev_id,
    int32 *value,
    u_int32 actv_flag);
```

**Description**

_*os_ev_set()* signals that an event has occurred. The current event variable is set to the value passed at *value*. Then, the event queue is searched for the first process waiting for that event.

- **ev_id** identifies the event. (Input)
- **value** is a pointer to the location where _os_ev_set() stores the event value prior to the set operation. (Output)
- **actv_flag** specifies which process(es) to activate. (Input)
  - If *actv_flag* is one, all processes in range are activated.
  - If *actv_flag* is zero, the first process in the event queue waiting for that range is activated.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

`os_lib.l`

**See Also**

- _os_ev_signal()
- F$Event
- F_EVENT
- F_EVENT, EV_SET
- F_EVENT, EV_SIGNAL
Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_setand(
    event_id ev_id,
    int32 *value,
    u_int32 mask,
    u_int32 actv_flag)
```

Description

_os_ev_setand() signals that an event has occurred. The current event variable is ANDed with the value passed in `mask`. Then, the event queue is searched for the first process waiting for that event value.

- `ev_id` identifies the event. (Input)
- `value` is a pointer to the location where _os_ev_setand() stores the event value prior to the logical operation. (Output)
- `mask` is the event mask. (Input)
- `actv_flag` specifies which process(es) to activate. (Input)
  - If `actv_flag` is one, all processes in range are activated.
  - If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`os_lib.l`

See Also

- `F_EVENT` OS-9 Technical Manual
- `F_EVENT, EV_SETAND` OS-9 Technical Manual
- `F_EVENT, EV_SIGNAL` OS-9 Technical Manual
_os_ev_setor()
Set Event Variable and Signal Event Occurrence

Syntax
#include <events.h>
#include <types.h>
error_code _os_ev_setor(
    event_id ev_id,
    int32 *value,
    u_int32 mask,
    u_int32 actv_flag);

Description
_os_ev_setor() signals that an event has occurred. The current event variable is ORed with the value passed in mask. Then, the event queue is searched for the first process waiting for that event value.

ev_id
    identifies the event. (Input)

value
    is a pointer to the location where _os_ev_setor() stores the event value prior to the logical operation. (Output)

mask
    is the event mask. (Input)

actv_flag
    specifies which process(es) to activate. (Input)
    • If actv_flag is one, all processes in range are activated.
    • If actv_flag is zero, the first process in the event queue waiting for that range is activated.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F_EVENT, EV_SETOR OS-9 Technical Manual
F_EVENT, EV_SIGNAL OS-9 Technical Manual
Set Relative Event Variable and Signal Event Occurrence

Syntax

```c
#include <events.h>
#include <types.h>

error_code _os_ev_setr(
    event_id ev_id,
    int32 *value,
    u_int32 actv_flag);
```

Description

_os_ev_setr() signals that an event has occurred. The current event value is incremented by `value` (output). Then, the event queue is searched for the first process waiting for that event value. Arithmetic underflows or overflows are set to `0x80000000` (minimum integer) or `0x7fffffff` (maximum integer), respectively.

- `ev_id` identifies the event. (Input)
  - The amount to add to the event value is passed at the location pointed to by `value`.
  - OS-9 for 68K: _os_ev_setr() stores the event value before the increment at `value`.
  - OS-9: _os_ev_setr() stores the event value after the increment at `value`.

- `actv_flag` specifies which process(es) to activate. (Input)
  - If `actv_flag` is one, all processes in range are activated.
  - If `actv_flag` is zero, the first process in the event queue waiting for that range is activated.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
See Also

=os_ev_set()
=os_ev_signal()

F$Event
F EVENT, EV_SET
F EVENT, EV_SIGNAL

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
_os_ev_setxor()
Set Event Variable and Signal Event Occurrence

Syntax
#include <events.h>
#include <types.h>
error_code _os_ev_setxor(
    event_id ev_id,
    int32 *value,
    u_int32 mask,
    u_int32 actv_flag);

Description
_os_ev_setxor() signals that an event has occurred. The current event value is
EXCLUSIVE-ORed with \texttt{mask}. Then, the event queue is searched for the first
process waiting for that event value.

\texttt{ev_id}
identifies the event. (Input)

\texttt{value}
is a pointer to the location where \_os_ev_setxor() stores the event value prior
to the logical operation. (Output)

\texttt{mask}
specifies the event mask. (Input)

\texttt{actv_flag}
specifies which process(es) to activate. (Input)

- If \texttt{actv_flag} is one, all processes in range are activated.
- If \texttt{actv_flag} is zero, the first process in the event queue waiting for that
  range is activated.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F_EVENT
OS-9 Technical Manual
F_EVENT, EV_SEXTXOR
OS-9 Technical Manual
F_EVENT, EV_SIGNL
OS-9 Technical Manual
_os_ev_signal()

Signal Event Occurrence

Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_signal(
    event_id ev_id,
    int32 *value,
    u_int32 actv_flag);
```

Description

_os_ev_signal() signals that an event has occurred. The current event variable is updated with the signal auto-increment that was specified when the event was created. Then, the event queue is searched for the first process waiting for that event value.

- **ev_id** identifies the event that has occurred. (Input)
- **value** is a pointer to the location where _os_ev_signal() stores the event value prior to the signal operation. (Output)
- **actv_flag** specifies which process(es) to activate. (Input)
  - If actv_flag is one, all processes in the event queue with a value in range are activated.
  - If actv_flag is zero, the first process in the event queue waiting for that range is activated.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

- F$Event
- _F_EVENT
- _F_EVENT, EV_SIGNL

OS-9 for 68K Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
**_os_ev_tstset_**

**Wait for Event to Occur**

**Description**

_os_ev_tstset_() waits for an event to occur. The event variable is ANDed with the value in mask. If the result is not zero, the calling process is suspended in a FIFO event queue until an _os_ev_signal_() occurs that clears all of the bits corresponding to the set bits in the mask. Then, the bits corresponding to the set bits in the mask are set.

- **ev_id**
  - identifies the event. (Input)

- **value**
  - is a pointer to the location where _os_ev_tstset_() stores the actual event value. (Output)

- **signal**
  - is a pointer to the location where _os_ev_tstset_() stores the signal code. (Output)

  If a process in the event queue receives a signal, it is activated even though the event has not actually occurred.

- **mask**
  - specifies the activation mask. (Input)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

os_lib.l
See Also

- `F_EVENT` [OS-9 Technical Manual]
- `F_EVENT, EV_TSTSET` [OS-9 Technical Manual]
- `F_EVENT, EV_SIGNAL` [OS-9 Technical Manual]
_os_ev_trywait()

Perform a Non-blocking Event Wait

Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_trywait(
    event_id ev_id,
    int32 *value,
    int32 min_val,
    int32 max_val);
```

Description

_os_ev_trywait() performs a non-blocking event wait. If the event value is currently in the range between the minimum and maximum activation values the wait auto-increment (specified at creation) is added to the event variable. If the current value is not in range, the error EAGAIN is returned.

- **ev_id**: identifies the event. (Input)
- **value**: is a pointer to the location where _os_ev_trywait() stores the actual event value before the wait increment was applied. (Output)
- **min_val**: is the minimum activation value. (Input)
- **max_val**: is the maximum activation value. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

_os_ev_signal()  _os_ev_wait()
_os9_ev_wait()

In the OS-9 Technical Manual:

F_EVENT  F_EVENT, EV_WAIT
F_EVENT, EV_TRYWAIT  F_EVENT, EV_SIGNAL
Syntax
#include <events.h>
#include <types.h>
error_code _os_ev_unlink(event_id ev_id);

Description
_os_ev_unlink() decrements the use count of an event. When the use count reaches zero, you can delete the event using _os_ev_delete().

    ev_id
        identifies the event. (Input)

Attributes
Operating System:     OS-9 and OS-9 for 68K
State:                User and System
Threads:              Safe
Re-entrant:           Yes

Library
os_lib.l

See Also
_os_ev_delete()
_os_ev_link()
F$Event
   OS-9 for 68K Technical Manual
F_EVENT
   OS-9 Technical Manual
F_EVENT, EV_UNLNK
   OS-9 Technical Manual
F_EVENT, EV_DELETE
   OS-9 Technical Manual
`_os_ev_wait()`

Wait for Event to Occur

### Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_wait(
    event_id ev_id,
    int32 *value,
    signal_code *signal,
    int32 min_val,
    int32 max_val);
```

### Description

`_os_ev_wait()` waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values. Then, the wait auto-increment (specified at creation) is added to the event variable.

- **ev_id**
  - identifies the event. (Input)

- **value**
  - is a pointer to the location where `_os_ev_wait()` stores the actual event value. (Output)

- **signal**
  - is a pointer to the location where `_os_ev_wait()` stores the signal code. (Output)
    - If a process in the event queue receives a signal, it is activated even though the event has not actually occurred.

- **min_val**
  - is the minimum activation value. (Input)

- **max_val**
  - is the maximum activation value. (Input)

### Attributes

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
Library

os_lib.l

See Also

_os_ev_signal()
_os_ev_waitr()
_os9_ev_waitr()
F_EVENT
F_EVENT, EV_WAIT
F_EVENT, EV_SIGNAL

OS-9 Technical Manual
--os9_ev_wait()
Wait for Event to Occur

Syntax

#include <events.h>
#include <types.h>
error_code _os9_ev_wait(
    event_id ev_id,
    int32 *value,
    int32 min_val,
    int32 max_val);

Description

_os9_ev_wait() waits for an event to occur. It waits until an event call places the value in range between the minimum activation value and the maximum activation value. Then, the wait auto-increment (specified at creation) is added to the event variable.

If a process in the event queue receives a signal, it is activated even though the event has not actually occurred.

ev_id
    specifies the event. (Input)

value
    is a pointer to the location where _os9_ev_wait() stores the actual event value. (Output)

min_val
    specifies the minimum activation value. (Input)

max_val
    specifies the maximum activation value. (Input)

Attributes

Operating System:   OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library

os_lib.l

See Also

F$Event    OS-9 for 68K Technical Manual
Wait for Relative Event to Occur

Syntax

```c
#include <events.h>
#include <types.h>
error_code _os_ev_waitr(
    event_id ev_id,
    int32 *value,
    signal_code *signal,
    int32 *min_val,
    int32 *max_val);
```

Description

_os_ev_waitr() waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values, where the integers at locations min_val and max_val are relative to the current event value. Then, the wait auto-increment (specified at creation) is added to the event variable.

The event value is added to min_val and max_val, and the actual values are returned to the caller. If an underflow or overflow occurs on the addition, the values 0x80000000 (minimum integer) and 0x7fffffff (maximum integer) are used, respectively.

- **ev_id** identifies the event. (Input)
- **value** is a pointer to the location where _os_ev_waitr() stores the actual event value. (Output)
- **signal** is a pointer to the location where _os_ev_waitr() stores the signal code if a signal activates the process. (Output)
- **min_val** is a pointer to location where _os_ev_waitr() stores the minimum absolute activation value. (Input/Output)
  - The minimum relative activation value is passed at the location pointed to by min_val.
- **max_val** is a pointer to the location where _os_ev_waitr() stores the maximum absolute activation value. (Input/Output)
  - The maximum relative activation value is passed at the location pointed to by max_val.
 Attributes
Operating System:   OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

 Library
os_lib.l

See Also
_os_ev_signal()
_os_ev_wait()
_os9_ev_waitr()
F_EVENT
F_EVENT, EV_SETOR
F_EVENT, EV_SIGNAL
OS-9 Technical Manual
OS-9 Technical Manual
OS-9 Technical Manual
_os9_ev_waitr()

Wait for Relative Event to Occur

Syntax

```c
#include <events.h>
#include <types.h>

error_code _os9_ev_waitr(
    event_id ev_id,
    int32 *value,
    int32 *min_val,
    int32 *max_val);
```

Description

_os9_ev_waitr() waits for an event to occur. It waits until an event call places the value in the range between the minimum and maximum activation values, where min_val and max_val are relative to the current event value. Then, the wait auto-increment (specified at creation) is added to the event variable.

The event value is added to min_val and max_val respectively, and the actual values are returned to the caller. If an underflow or overflow occurs on the addition, the values $80000000 (minimum integer) and $7fffffff (maximum integer) are used, respectively.

**ev_id**
- specifies the event. (Input)

**value**
- is a pointer to the location where _os9_ev_waitr() stores the actual event value. (Output)

**min_val**
- is a pointer to the location where _os9_ev_waitr() stores the minimum absolute value. (Input/Output)

The minimum relative activation value is passed at the location pointed to by min_val.

**max_val**
- is the pointer to the location where _os9_ev_waitr() stores the maximum absolute value. (Input/Output)

The maximum relative activation value is passed at the location pointed to by max_val.
Attributes
Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1

See Also
_os9_ev_waitr()
F$Event

OS-9 for 68K Technical Manual
Start Child Process

Syntax

```c
#include <process.h>
#include <dexec.h>  /* used when _os_dfork(),
    /* _os9_dfork(), or _os_dforkm()
    /* is specified as func */
#include <types.h>

erreur_code _os_exec(
    u_int32 (*func)(),
    u_int32 priority,
    u_int32 pathcnt,
    void *modname,
    char **argv,
    char **envp,
    u_int32 datasize, ...);
```

Description

_os_exec() prepares the parameter and environment list before creating a process. The function to create a new process passes information to the new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.

The first seven parameters are identical for _os_fork(), _os_forkm(), _os_chain(), _os_chaint(), _os_dfork(), or _os_dforkm(). Four additional parameters may be required (add1, add2, add3, and add4) and have different meaning depending on the function specified by func:

Table 2-13. _os_exec() Additional Parameters

<table>
<thead>
<tr>
<th>Call</th>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_fork()</td>
<td>add1</td>
<td>Child process ID returns to the location of the add1 pointer</td>
</tr>
<tr>
<td></td>
<td>add2</td>
<td>Type/language of the new process</td>
</tr>
</tbody>
</table>
|               | add3      | *Orphan:
|               |           | 0 = Normal child process                                                   |
|               |           | 1 = Process not associated with parent                                    |
| _os_forkm()   | add1      | The child process ID returns to the location of the add1 pointer            |
|               | add2      | *Orphan:
|               |           | 0 = Normal child process                                                   |
|               |           | 1 = Process not associated with parent                                    |
| _os_chain()   | add1      | Type/language of the new process                                           |
*On OS-9 for 68K, the orphan flag must be set to zero.

If you are using _os_exec() when func indicates a chain function, you must provide a full pathlist for the module or verify that the module is already in memory.

For OS-9, a threaded process may only chain from the main() thread and may not chain if multiple threads are active in the process.

_os_exec() returns the value of (*func)(), which returns 0 if successful. Otherwise, it returns an error number.

**func**

is a pointer to one of the following functions that create a new process. (Input)

- OS-9 for 68K: _os_fork(), _os9_dfork(), or _os_chain()
- OS-9: _os_fork(), _os_forkm(), _os_forkn(), _os_chain(), _os_chainm(), _os_dfork(), or _os_dforkm()

**priority**

is the priority value at which the new process is to run. (Input)

Only the lower 16 bits of priority are used. If priority is zero, the new process receives the priority of the calling process.

**pathcnt**

is the number of open paths to pass to the new process. (Input)

Normally, pathcnt is three, causing the three standard paths to be passed. A value of zero passes no open paths. Only the lower 16 bits of pathcnt are used.

---

Table 2-13. _os_exec() Additional Parameters (Continued)

<table>
<thead>
<tr>
<th>Call</th>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_chainm()</td>
<td></td>
<td>Does not use the extra parameters</td>
</tr>
<tr>
<td>_os_dfork()</td>
<td>add1</td>
<td>Child process ID returns to the location of the add1 pointer</td>
</tr>
<tr>
<td></td>
<td>add2</td>
<td>Type/language of the new process</td>
</tr>
<tr>
<td></td>
<td>add3</td>
<td>Pointer to register stack for child process</td>
</tr>
<tr>
<td></td>
<td>add4</td>
<td>Pointer to floating point register stack for child process</td>
</tr>
<tr>
<td>_os9_dfork()</td>
<td>add1</td>
<td>Child process ID returns to the location of the add1 pointer</td>
</tr>
<tr>
<td></td>
<td>add2</td>
<td>Type/language of the new process</td>
</tr>
<tr>
<td></td>
<td>add3</td>
<td>Pointer to register stack for child process</td>
</tr>
<tr>
<td>_os_dforkm()</td>
<td>add1</td>
<td>Child process ID returns to the location of the add1 pointer</td>
</tr>
<tr>
<td></td>
<td>add2</td>
<td>Pointer to register stack for child process</td>
</tr>
<tr>
<td></td>
<td>add3</td>
<td>Pointer to floating point register stack for child process</td>
</tr>
</tbody>
</table>
modname

is a pointer to a string naming the new primary module or a pointer to the
module in the case of the functions ending in m (_os_forkm(), _os_chainm(),
and _os_dforkm()). (Input)

argv

is a pointer to the parameter pointer list that the new process is to receive.
(Input)

By convention, argv[0] is the name of the new module. The end of the argv
list is marked by a null pointer.

envp

is a pointer to the environment pointer list containing environment variables
that the new process is to receive. (Input)

The end of the envp list is also marked by a null pointer.

datasize

is the additional memory (in bytes) to allocate to the new process’ stack
memory. (Input)

A datasize of zero indicates default stack size required by process.

Example

#include <process.h>
#include <dexec.h>
#include <errno.h>
#include <cglob.h>
#include <types.h>

/* Use the environment pointer given */
/* to you by the shell. */
char *argblk[] =
{
  0,0,0,0,0,0,0,        /* Create an array of pointers to */
};                     /* become argv[] of child process. */

main()
{
  u_int16 prior = 0;   /* Priority of Child process */
  /* 0 is same as parent */
  u_int16 paths = 3;   /* number of paths to inherit */
  /* stdin, stdout, stderr */
  u_int32 edata = 0;   /* No Extra Data Space */
  process_id child_id; /* child id number */
char orphan = 0; /* make this child a normal */ /* child not an orphan */

u_int16 type_lang; /* type language of child */

type_lang = mktypelang(MT_PROGRAM, ML_OBJECT);

argblk[0] = "dir"; /* Child process is "dir" */
argblk[1] = "-e"; /* With a -e option */
argblk[2] = "-a", /* And a -a option */
argblk[3] = 0; /* Make sure end of options is a null ptr */

if ((errno = _os_exec(_os_fork, prior, paths,
                    argblk[0], argblk, _environ,
                    edata, &child_id, type_lang,
                    orphan)) != SUCCESS)
    _os_exit(_errmsg(errno,
          "Problem Creating Child %s
", argblk[0]));

/*If we get this far, the _os_exec() was successful*/
_os_wait(0); /* Wait for child process to die */ /* die*/
    printf("The Child's ID number was %d
", child_id);
}

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_chain() _os_chainm() _os_dfork()
_os_dforkm() _os_fork() _os_forkm()
os9fork(), os9forkc() os9exec() os9kexec()
OS-9 for 68K System Call Processing

Syntax

```c
#include <process.h>

int os9exec(
    int (*func)(),
    const char *modname,
    char **argv,
    char **envp,
    unsigned int stacksize,
    int priority,
    int pathcnt);
```

Description

`os9exec()` is a high-level fork/chain interface that prepares the parameter and environment list before a process is created.

`os9forkc()` and `chainc()` are C-level hooks to the system calls. By themselves, they provide no real parameter information. This leaves the interpretation of a raw parameter string up to the `cstart` portion of the C program. Problems can occur when handling quotes and control characters.

If `func` is `os9forkc()` or `os9fork()` and the initial attempt to create the child process fails due to `EOS_PNNF` (path name not found), `os9exec()` calls `modloadp()` with `modname` to load the module from disk and attempts the fork again. This action is the same as the shell handling of executable files.

If you use `os9exec()` when `func` is `chain()` or `chainc()`, you must either:

- Ensure that the module is in memory or in your current execution directory. (See `modlink()`, `modload()`, and `modloadp()`.)

- Provide a full pathlist to the module.

For OS-9, a threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

`os9exec()` returns the value of `(*func)()`. This is the ID of the child process for `os9fork()`. `-1` is returned on error.

Any program that executes a C program must do so using the `os9exec()` interface. The `cstart` module can handle parameter strings passed by `os9fork()` and `chain()`, but no environment is available and the `argv` pointer list is separated by white space.

`os9exec()` is not automatically available if you are using ANSI or extended-ANSI modes. To use `os9exec()` in one of these modes, link with `sys_clib.l`. 
os9exec()
provides more precise control over the parameter strings and enables the environment variables to be easily inherited from the parent process. (Input)

os9exec()
closely resembles the UNIX execve() system call.

func
is a pointer to a function that creates a new process. (Input)
This function is normally chain(), chainc(), os9fork(), or os9forkc(). func passes information to a new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.

modname
is a pointer to a string naming the new primary module. (Input)

argv
is a pointer to the parameter pointer list that the new process receives. (Input)
By convention, argv[0] is the name of the new module and a null pointer marks the end of the argv list.

envp
is a pointer to the environment pointer list that the new process receives. (Input)
A null pointer marks the end of the envp list. You can set envp to point to the environment list of the current process to allow the child to inherit that environment.

stacksize
specifies the additional memory (in bytes) to allocate to the new process’ stack memory. (Input)
A stacksize of zero indicates the default stack size the process requires.

priority
is the priority at which the new process runs. (Input)
If priority is zero, the new process receives the priority of the calling process.

pathcnt
is the number of open paths to pass to the new process. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Example
The following code fragment is an example call to `os9exec()`.

The `argblk` array contains pointers to the parameters in the order that the new program is to receive them. The new process receives a copy of the parameters, not the addresses of the parameters.

This example passes the current environment by naming the global variable `_environ`.

```c
extern int os9forkc();
extern char **_environ;

char *argblk[] = {
    "rename",
    "-x",
    "oldname",
    "newname",
    0,
};

main ()
{
    int pid;
    if ((pid = os9exec(os9forkc, argblk[0], argblk, _environ, 0, 0, 3)) > 0) wait(0);
    else printf("can’t fork %s", argblk[0]);
}
```

Library
sys_clib.l

The primary purpose of the `sys_clib.l` library is to provide a library for compatibility with the Microware K&R C compiler. For additional information, refer to Appendix A.

See Also
chainc(), chain()
modlink()
modload()
modloadp()
_os_exec()
os9fork(), os9forkc()
Syntax
#include <process.h>
#include <types.h>
error_code _os_exit(status_code status);

Description
_os_exit() allows a process to terminate itself. Its data memory area is deallocated and its primary module is unlinked. All open paths are automatically closed.

status
is the status code returned to the parent process. (Input)
The parent must perform an _os_wait() or an _os_exit() before the process descriptor is returned to free memory.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_aproc()
_os_close()
_os_findpd()
_os_fork()
_os9_retpd()
_os_srtmem()
_os_unlink()
_os_wait()

F$Exit
F_EXIT

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax
#include <process.h>
#include <types.h>
error_code _os_findpd(
    process_id proc_id,
    pr_desc **proc_desc);

Description
_os_findpd() converts a process number to the absolute address of its process
descriptor data structure.

proc_id
    specifies the process ID. (Input)

proc_desc
    is a pointer to the location where _os_findpd() stores the pointer to the
    process descriptor. (Output)

Attributes
Operating System:    OS-9 and OS-9 for 68K
State:               System
Threads:             Safe
Re-entrant:          Yes

Library
os_lib.l

See Also
_os_alocproc()
_os_rtnprc()
F$FindPD              OS-9 for 68K Technical Manual
F_FINDPD              OS-9 Technical Manual
_os9_findpd()  
Find Fixed Block of Memory

Syntax
#include <process.h>
#include <types.h>
error_code _os9_findpd(
    u_int32 number,
    void **address,
    void *table);

Description
_os9_findpd() converts a table index to the absolute address of its data structure.

number  
is the number of the table element. Only the lower 16 bits of number are used.  
(Input)

address  
is a pointer to the location where _os9_findpd() stores the table entry pointer.  
(Output)

table  
is a pointer to the table to use. (Input)

Attributes
Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os9_allpd()
_os9_retpd()
F$FindPD

OS-9 for 68K Technical Manual
Add/Remove Device from Fast IRQ Table (System-State Only)

Syntax

```c
#include <regs.h>
error_code _os_firq(
    u_int32 vector,
    u_int32 priority,
    void *irq_entry,
    void *statics);
```

Description

_os_firq() installs an IRQ service routine into or removes one from the system’s fast IRQ system.

This fast IRQ system provides a faster interrupt response context than the normal IRQ vector polling scheme (provided through _os9_irq()).

- To place a routine into the fast IRQ system, set irq_entry to a non-zero value.
- To remove a routine from the fast IRQ system, set irq_entry to zero.

Only one _os_firq() routine can be active at a time per vector. An attempt to install a second routine on a vector using _os_firq() causes an EOS_VCTBSY error. If additional devices are required to be on the same vector as an _os_firq() device, use _os9_irq() to install them.

The interrupt stack provided is a small “default” IRQ stack. The service routine must ensure that this stack is not overflowed. The init module’s “IRQ stack” value does not affect this stack; the init module value is used to control the _os9_irq() polling system stack.

vector
    specifies the vector number. (Input)

priority
    specifies the priority. (Input)

priority
    is reserved and must be zero.

irq_entry
    is a pointer to the IRQ service routine’s entry point. (Input)

    If irq_entry is zero, the IRQ service routine is deleted.

statics
    is a pointer to the global static storage. (Input)
Attributes
Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_irq()
F$FIRQ

OS-9 for 68K Technical Manual
Syntax
#include <module.h>
#include <types.h>
error_code _os_fmod(
    u_int32 *type_lang,
    mod_dir **moddir_entry,
    char *mod_name);

Description
_os_fmod() searches the module directory for a module whose name, type, and
language match the parameters. If found, a pointer to the module directory entry is
returned in moddir_entry.

_type_lang
    specifies the type and language of the module. (Input)
    Only the lower 16 bits of _type_lang are used.

_moddir_entry
    is a pointer to the location where _os_fmod() stores the pointer to the module
directory entry. (Output)

_mod_name
    is a pointer to the module name. (Input)

Attributes
Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
_os_fork()
Create New Process

Syntax
#include <process.h>
#include <types.h>
error_code _os_fork(
    u_int32 priority,
    u_int32 path_cnt,
    const char *mod_name,
    const void *params,
    u_int32 param_size,
    u_int32 mem_size,
    process_id *proc_id,
    u_int32 type_lang,
    u_int32 orphan);

Description
_os_fork() creates a new process which becomes a child of the caller. It sets up the new process’ memory, MPU registers, and standard I/O paths.

If the orphan flag is non-zero, the new process executes without a parent. If the orphan flag is set to a non-zero value, the EOS_PARAM error is returned.

priority
    specifies the priority of the new process. (Input)
    Only the lower 16 bits of priority are used. If priority is zero, the new process inherits the same priority as the calling process.

path_cnt
    specifies the number of I/O paths for the child to inherit. (Input)
    Only the lower 16 bits of path_cnt are used.

mod_name
    is the module name pointer. (Input)

params
    is a pointer to the parameter block. (Input)

param_size
    specifies the size of the parameter block. (Input)

mem_size
    specifies any additional stack space to allocate. (Input)
proc_id
  is a pointer to the location where _os_fork() stores the child process ID. (Output)

type_lang
  specifies the desired type and language. (Input)
  Only the lower 16 bits of type_lang are used. If type_lang is zero, any module may be loaded regardless of type and language.

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes

Library
os_lib.l

See Also
_os_wait()
_os_exit()
_os_chain()
F$Fork  OS-9 Technical Manual
F_FORK  OS-9 Technical Manual
os9fork(), os9forkc()
Create Process

Syntax
#include <process.h>
#include <types.h>

int os9fork(
    const char *modname,
    int parmsize,
    const char *parmptr,
    int type,
    int lang,
    int datasize,
    int prior);

int os9forkc(
    const char *modname,
    int parmsize,
    const char *parmptr,
    int type,
    int lang,
    int datasize,
    int prior,
    int pathcnt);

Description
os9forkc() creates a process that runs concurrently with the calling process. When
the new process terminates, its exit status is available to the forking (parent)
process. The new process inherits the standard paths (0, 1, and 2) plus any
additional paths indicated by pathcnt.

If the fork is unsuccessful, -1 is returned and the appropriate error code is placed in
the global variable errno. If the fork is successful, the process ID number of the new
process is returned.

os9fork() is the same as os9forkc() without a pathcnt parameter.
_os_exec() is the preferred method to fork C programs.
Beware of forking to a system module. It only makes sense to fork to a program of
type object module. os9fork() is a historical function and is likely to be removed in
a future release. Use os9forkc() instead.
modname
    is a pointer to a null-terminated module name. (Input)

parmsize
    specifies the size of the parameter list. (Input)
    Generally, you must set parmsize equal to strlen(parmptr)+1.

parmptr
    is a pointer to a null-terminated string to pass to the new process. (Input)

type
    specifies the type of module. (Input)
    If type is zero, any module may be forked, regardless of type.

lang
    specifies the module language. (Input)
    If lang is zero, any module may be forked, regardless of language.

datasize
    gives extra memory to the new program. (Input)
    If no extra memory is required, set datasize to zero.

prior
    is the new priority at which to run the program. (Input)
    Indicate zero if no priority change is desired.

pathcnt
    is the number of the parent’s open paths for the new process to inherit. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.1

The purpose of sys_clib.1 is to provide a library for compatibility with the Microware K&R C compiler. For more information, refer to Appendix A.

See Also
_os_chain()  OS-9 for 68K Technical Manual
_os_exec()   OS-9 for 68K Technical Manual
_os_fork()   OS-9 Technical Manual
F$Chain      OS-9 Technical Manual
F$Fork       OS-9 Technical Manual
F_CHAIN      OS-9 Technical Manual
F_FORK       OS-9 Technical Manual
_os_forkm()
Create New Process by Module Pointer

Syntax

```c
#include <process.h>
#include <types.h>
error_code _os_forkm(
    u_int32 priority,
    u_int32 path_cnt,
    mh_com *mod_head,
    void *params,
    u_int32 param_size,
    u_int32 mem_size,
    process_id *proc_id,
    u_int32 orphan);
```

Description

_os_forkm() creates a new process which becomes a child of the caller. It sets up the new process’ memory, MPU registers, and standard I/O paths. The new process is forked by a module pointer. _os_forkm() assumes that the module pointer is the primary module pointer for the new process.

If the orphan flag is non-zero, the new process executes without a parent. If the orphan flag is zero, the new process is the child of the calling process. Only the lower 16 bits of orphan are used.

**priority**

specifies the priority of the new process. (Input)

Only the lower 16 bits of **priority** are used. If **priority** is zero, the new process inherits the same priority as the calling process.

**path_cnt**

specifies the number of I/O paths for the child to inherit. (Input)

Only the lower 16 bits of **path_cnt** are used.

**mod_head**

is the module name pointer. (Input)

**params**

is a pointer to the parameter block. (Input)

**param_size**

specifies the size of the parameter block. (Input)
mem_size
    specifies any additional stack space to allocate. (Input)

proc_id
    is a pointer to the location where _os_forkm() stores the child process ID. (Output)

Attributes
Operating System:     OS-9
State:                User and System
Threads:              Safe
Re-entrant:           Yes

Library
os_lib.l

See Also
_os_forkm()
_os_get_blkmap()
Get Free Memory Block Map

Syntax
#include <memory.h>
#include <types.h>
error_code _os_get_blkmap(
    void *start,
    void *buffer,
    u_int32 size,
    u_int32 *min_alloc,
    u_int32 *num_segs,
    u_int32 *tot_mem,
    u_int32 *free_mem);

Description
_os_get_blkmap() copies the address and size of the system’s free RAM blocks into
your buffer for inspection. It also returns information concerning the free RAM as
noted by the parameters. A series of structures showing the address and size of free
RAM blocks is returned in your buffer.

Although _os_get_blkmap() returns the address and size of the system’s free
memory blocks, you cannot assume that these blocks are free to use. Use
_os_srqmem() to request free memory blocks.

start
    is a pointer to the address to begin reporting the segments. (Input)

buffer
    is a pointer to the buffer to use. (Input)

size
    specifies the buffer size in bytes. (Input)

min_alloc
    is a pointer to the location where _os_get_blkmap() stores the minimum
memory allocation size for the system. (Output)

num_segs
    is a pointer to the location where _os_get_blkmap() stores the number of
memory fragments in the system. (Output)

tot_mem
    is a pointer to the location where _os_get_blkmap() stores the total RAM
found by the system at startup. (Output)

free_mem
    is a pointer to the location where _os_get_blkmap() stores the current total
free RAM available. (Output)
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib

See Also
_os_mem(), _os_srqmem(), F$GBlkMap(), OS-9 for 68K Technical Manual
_F$GBLKMP, OS-9 Technical Manual
Syntax

```c
#include <io.h>
#include <types.h>
error_code _os_getdl(dev_list **dev_list_ptr);
```

Description

 `_os_getdl()` returns a pointer to the first entry in the system’s I/O device list. This pointer should never be accessed directly in user state. You should use `_os_cpymem()` to get a copy of the device list entry.

 `dev_list_ptr` is a pointer to the location where `_os_getdl()` stores the pointer to the first entry in the device list. (Output)

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`os_lib.l`
Syntax

```c
#include <time.h>
#include <types.h>
error_code _os_getime(
    time_t *time,
    u_int32 *ticks);
```

Description

_os_getime() returns the current system time in the number of seconds since a date specified in Greenwich Mean Time.

To determine the specified date, refer to the applicable manual, OS-9 for 68K Technical Manual or OS-9 Technical Manual.

On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, see _os9_getime().

_os_getime() returns a date and time of zero (with no error) if no previous call to _os_setime() has been made. A tick rate of zero indicates the clock is not running.

*time

is a pointer to the location where _os_getime() stores the current time.

(Output)

*ticks

is a pointer to the location where _os_getime() stores the clock tick rate in ticks per second in the most significant word and the current tick in the least significant word. (Output)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

_os_setime()
os9_getime()
F$Time
F_TIME

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax

```c
#include <time.h>
#include <types.h>
error_code _os9_gettime(
    u_int32 format,
    u_int32 *time,
    u_int32 *date,
    u_int16 *day,
    u_int32 *ticks);
```

Description

_os9_gettime() returns the current system date and time. In the normal (Gregorian) format, time is expressed as 00hhmms and date as yyyymmd. The Julian format expresses time as seconds since midnight and date as the Julian day number.

The following figures illustrate values returned at the pointers.

**Figure 2-7. Gregorian Formats**

<table>
<thead>
<tr>
<th>Byte MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>Hour (0-23)</td>
</tr>
</tbody>
</table>

**Figure 2-8. Julian Formats**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Time</th>
<th>0-31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>seconds since midnight (0-86399)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>Date</th>
<th>Julian day number</th>
</tr>
</thead>
</table>
_os9_gettime() returns a date and time of zero (with no error) if no previous call to _os9_gettime() is made after the system start-up. A tick rate of zero indicates the clock is not running.

format
  specifies the format of the returned date and time: (Input)

Table 2-14. _os9_gettime() Format Values

<table>
<thead>
<tr>
<th>Format</th>
<th>Date and Time Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Gregorian format</td>
</tr>
<tr>
<td>1</td>
<td>Julian format</td>
</tr>
<tr>
<td>2</td>
<td>Gregorian format with ticks</td>
</tr>
<tr>
<td>3</td>
<td>Julian format with ticks</td>
</tr>
</tbody>
</table>

time
  is a pointer to the location where _os9_gettime() stores the current system time. (Output)

date
  is a pointer to the location where _os9_gettime() stores the current system date. (Output)

day
  is a pointer to the location where _os9_gettime() stores the day of the week, where 0 is Sunday and 6 is Saturday. (Output)

ticks
  is a pointer to the location where _os9_gettime() stores the clock tick data. (Output)

  If ticks are requested, the clock tick rate in ticks per second is returned in the most significant word at the location pointed to by ticks. The least significant word contains the current tick.

Attributes
Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1

See Also
_os9_gettime() _os_julian() _os_gettime() F$Time
OS-9 for 68K Technical Manual
_os_get_ioproc()
Get Pointer to I/O Process Descriptor

Syntax
#include <io.h>
#include <types.h>
error_code _os_get_ioproc(
    process_id proc_id,
    io_proc **proc);

Description
_os_get_ioproc() returns a pointer to the I/O process descriptor for the specified process.

proc_id
    specifies the process ID of the process. (Input)

proc
    is a pointer to the location where _os_get_ioproc() stores the pointer to the I/O process descriptor. (Output)

Attributes
Operating System:    OS-9
State:               System
Threads:             Safe
Re-entrant:          Yes

Library
os_lib.l
_os_get_mdp()

Get Current and Alternate Module Directory Pathlists

Syntax

```c
#include <moddir.h>
#include <types.h>
error_code _os_get_mdp(
    char *current,
    char *alternate);
```

Description

_os_get_mdp() returns pathlists to the current and alternate module directories. current and alternate should point to buffers of at least 64 bytes to accommodate the directory names.

current

is a pointer to the location where _os_get_mdp() stores the pathlist of the current module directory. (Output)

alternate

is a pointer to the location where _os_get_mdp() stores the pathlist of the alternate module directory. (Output)

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l
_os_get_moddir()
Get Copy of Module Directory

Syntax
For OS-9 for 68K:
#include <module.h>
#include <types.h>
error_code _os_get_moddir(
  void *buffer,
  u_int32 *count);

For OS-9:
#include <moddir.h>
#include <types.h>
error_code _os_get_moddir(
  void *buffer,
  u_int32   *count);

Description
_os_get_moddir() copies the process’ current module directory into your buffer for inspection.
Although the module directory contains pointers to each module in the system, never access the modules directly. Use _os_cpymem() to copy portions of the system’s address space for inspection.

buffer
is a pointer to the buffer. (Output)

For more information on the structure of the buffer (mod_dir) refer to the appropriate header file. (For OS-9 for 68K, the header file is module.h. For OS-9, the header file is moddir.h.)

count
The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output)
_os_get_moddir() also stores the actual number of bytes copied at the location pointed to by count.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_cpymem()
_os_move()
Syntax
#include <io.h>
#include <types.h>
error_code _os_getpd(
    path_id path,
    pd_com **pd);

Description
_os_getpd() converts the path number to the absolute address of its path descriptor data structure.

path
    specifies the path number. (Input)

pd
    is a pointer to the location where _os_getpd() stores the address of the path descriptor. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os9_allpd()
_os9_retpd()
Get State Descriptor Copy

Syntax

```c
#include <process.h>
#include <types.h>
error_code _os_get_prdesc(
    process_id procid,
    void *buffer,
    u_int32 *count);
```

Description

.os_get_prdesc() copies a process' or thread's state descriptor into a buffer for inspection. The format of the returned data is a pr_desc structure.

The preprocessor macro _USE_V3_0_PROCDESC must be defined to correctly use this function.

procid is the requested process or thread ID. (Input)

buffer is a pointer to the buffer to use. (Output)

count The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output)

_os_get_prdesc() also stores the actual number of bytes copied at the location pointed to by count.

Attributes

Operating System: OS-9
State: User
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_get_prsrc()
_os_gprdesc()
_os_get_prsrc()

Get Resource Descriptor Copy

Syntax
#include <process.h>
#include <types.h>

error_code _os_get_prsrc(
    process_id procid,
    void *buffer,
    u_int32 *count);

Description
_os_get_prsrc() copies a process' resource descriptor into a buffer for inspection. All the threads within a process share the same resource descriptor. The format of the returned data is a pr_src structure.

procid is the requested process or thread ID. (Input)

buffer is a pointer to the buffer to use. (Output)

count The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output)

_os_get_prsrc() also stores the actual number of bytes copied at the location pointed to by count.

Attributes
Operating System: OS-9
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.1

See Also
_os_get_prdesc()
_os_gprdsc()
_os_get_prtbl()

Get Copy of Process Descriptor Block Table

Syntax

#include <process.h>
#include <types.h>
error_code _os_get_prtbl(
    void *buffer,
    u_int32 *count);

Description

_os_get_prtbl() copies the process descriptor block table into the caller’s buffer for inspection.

Although _os_get_prtbl() returns pointers to all process descriptors, never access the process descriptors directly. Instead, use _os_gprdsc() to inspect specific process descriptors.

buffer
  is a pointer to the buffer. (Output)

count
  The maximum number of bytes to copy is passed at the location pointed to by count. (Input/Output)

_os_get_prtbl() also stores the actual number of bytes copied at the location pointed to by count.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_alocproc()
_os_gprdsc()
_os_getstat()
Get File/Device Status

Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_getstat(
    path_id path,
    u_int32 code,
    void *pb);

Description
_os_getstat() is a wildcard call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent.

The exact operation of this call depends on the device driver and file manager associated with the path.

The mnemonics for the status codes are located in the sg_codes.h header file.

The status codes that are currently defined by Microware and the functions that they perform are described in this manual. The functions have an _os_gs_ prefix.

This function is available within OS-9 for 68K, where it resides in the conv_lib.l library. However, it can be used only with DPIO file managers and drivers, such as SoftStax.

path
    is the path number. (Input)

code
    is the GetStat code. (Input)

pb
    is a pointer to the parameter block. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
Syntax
#include <sysglob.h>
#include <types.h>
error_code _os_getsys(
  u_int32 offset,
  u_int32 size,
  glob_buff *sysvar);

Description
_os_getsys() enables a process to examine a system global variable. These variables have a d_ prefix in the system header file sysglob.h. Consult the files in the DEFS directory for a description of the system global variables.

offset
  is the variable’s offset in the system globals. (Input)

size
  specifies the size of the variable (1, 2, or 4 bytes). (Input)

sysvar
  is a pointer to the location where _os_getsys() stores the value of the requested system global. (Output)

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes

Library
os_lib.l
**Example**

This example gets the current value of the tick counter:

```c
#include <stdio.h>
#include <sysglob.h>

main() {  
    Sysglobs sg;
    glob_buff buffer;
    error_code err;

    if ((err = _os_getsys(OFFSET(Sysglobs, d_ticks), sizeof(sg->d_ticks),
        &buffer)) != 0)
        printf("Failed to get the current value of the tick counter (d_ticks)\n");
    else
        printf("Current value of the tick counter (d_ticks) = %d\n", buffer.lng);

    exit(err);
}
```

**See Also**

_os_setsys()_
Syntax
#include <process.h>
#include <types.h>

error_code _os_gprdsc(
    process_id procid,
    void *buffer,
    u_int32 *count);

Description
_os_gprdsc() copies a process descriptor into a buffer for inspection.

procid
     is the requested process ID. (Input)

buffer
     is a pointer to the buffer to use. (Output)

count
     The maximum number of bytes to copy is passed at the location pointed to by
     count. (Input/Output)

_os_gprdsc() also stores the actual number of bytes copied at the location
     pointed to by count.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_get_prtbl()
Syntax

#include <time.h>
#include <types.h>
error_code _os_gregorian(
    u_int32 *time,
    u_int32 *date);

Description

_os_gregorian() converts Julian dates to Gregorian dates. Gregorian dates are considered the normal calendar dates.

time
    is a pointer to the location where the time in seconds since midnight. (Input/Output)
    _os_gregorian() stores the time in the form 00hhmmmss in the location pointed to by time.

date
    is a pointer to the location where the Julian date is passed. (Input/Output)
    _os_gregorian() stores the date in the form yyyyymmdd in the location pointed to by date. Refer to your operating system technical manual for more information.

Attributes

Operating System:  OS-9 and OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes

Library

os_lib.l

See Also

_os_julian()
_os_getime()
Return File Descriptor

Syntax

```
#include <cdi.h>
#include <types.h>

error_code _os9_gs_cdfd(
    path_id path,
    u_int32 count,
    void *fdbuf);
```

Description

_os9_gs_cdfd() reads the file descriptor describing the path number. The file descriptor may be read for informational purposes only, as there are no user-changeable parameters.

path
    specifies the path number. (Input)

count
    is the number of bytes to copy. (Input)
    Only the lower 16 bits of count are used.

fdbuf
    is a pointer to the buffer area for the file descriptor. (Output)

Attributes

Operating System:  OS-9 for 68K
State:  User and System
Threads:  Safe
Re-entrant:  Yes

Library

os_lib.l
Copy Contents of Path Descriptor

**Syntax**

```c
#include <sg_codes.h>
#include <types.h>

error_code _os_gs_cpypd(
    path_id path,
    u_int32 *size,
    void *path_desc);
```

**Description**

 `_os_gs_cpypd()` copies the contents of the specified path’s path descriptor to your buffer.

If the `size` value is greater than the size of the target path descriptor, the actual size of the path descriptor is returned.

- **path**
  - is the path number. (Input)

- **size**
  - The number of bytes to copy from the path descriptor is passed at the location pointed to by `size`. (Input/Output)

  `_os_gs_cpypd()` also stores the actual number of bytes copied at the location pointed to by `size`.

- **path_desc**
  - is a pointer to the buffer for the path descriptor data. (Output)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`os_lib.1`
Syntax
#include <rbf.h>
#include <types.h>
error_code _os_gs_cstats(
    path_id path,
    cachestats *cache_inf);

Description
_os_gs_cstats() returns a copy of the current cachestats structure.

path
    specifies the number of a path open on the device. (Input)

cache_inf
    is a pointer to the location where _os_gs_cstats() stores the structure
    containing information about RBF caching. (Output)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
Return Device Name

Syntax

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_devnm(
    path_id path,
    char *name);
```

Description

_os_gs_devnm() returns the name of the device associated with the specified path. Some networked devices using NFM may require a buffer size of 128 or 256 bytes for the device name.

On OS-9, the buffer pointed to by `name` should be at least 64 bytes to accommodate a device name. On OS-9 for 68K, the buffer pointed to by `name` is a fixed 32 bytes.

- `path` specifies the path number. (Input)
- `name` is a pointer to the location where _os_gs_devnm() stores the device name. (Output)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
_os_gs_devtyp()

Return Device Type

Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_devtyp(
    path_id path,
    u_int16 *type,
    u_int16 *class);

Description
_os_gs_devtyp() returns the type and class of the device associated with the specified path number.

path specifies the path number. (Input)

type is a pointer to the location where _os_gs_devtyp() stores the device type. (Output)

class is a pointer to the location where _os_gs_devtyp() stores the device class. (Output)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
_os_gs_diskfree()

Return Information About RBF Disk Free Space

Syntax

```c
error_code _os_gs_diskfree(path_id  path,
    u_int32  *totblocks,
    u_int32  *blksize,
    u_int32  *avail,
    u_int32  *contig);
```

Description

_os_gs_diskfree() returns information about RBF disk free space.

path

specifies the drive’s path number. (Input)

totblocks

is set to the total number of blocks on the disk. (Output)

blksize

is set to the size of blocks used on the disk (256, 512, etc.) (Output)

avail

is set to the total number of free blocks on the disk. (Output)

contig

is set to the number of blocks in the largest contiguous area. (Output)

Attributes

Operating System: OS-9

State: User and System

Threads: Safe

Re-entrant: Yes

Library

os_lib.l
Read Device Path Options

**Syntax**

```c
#include <sg_codes.h>
#include <types.h>

error_code _os_gs_dopt(
    path_id path,
    u_int32 *size,
    void *dopts);
```

**Description**

`_os_gs_dopt()` copies the initial (default) device path options into your buffer. These options are used for initializing new paths to the device.

*path*  
specifies the path number. (Input)

*size*  
The size of the buffer is passed at the location pointed to by *size*. (Input/Output)

`_os_gs_dopt()` also stores the actual number of bytes copied at the location pointed to by *size*.

*dopts*  
is a pointer to a buffer for the device path options. (Output)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

os_lib.l
**_os_gs_dsize()**

Get Number of Blocks on Media

**Syntax**

```c
#include <rbf.h>
#include <types.h>
error_code _os_gs_dsize(
    path_id path,
    u_int32 *totblocks,
    u_int32 *blocksize);
```

**Description**

_os_gs_dsize() gets information about the number of blocks on the media.

- **path** specifies the drive’s path number. (Input)
- **totblocks** is a pointer to the location where _os_gs_dsize() stores the total number of blocks on the device. (Output)
- **blocksize** is a pointer to the location where _os_gs_dsize() stores the size of a disk block in bytes. (Output)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

- os_lib.l
_os_gs_edt()

Get I/O Interface Edition Number

Syntax
#include <ioedt.h>
#include <types.h>
error_code _os_gs_edt(
    path_id path,
    u_int32 *edition);

Description
_os_gs_edt() returns the I/O interface edition number of the driver and validates
the compatibility of drivers and file managers.

path
    specifies the driver’s path number. (Input)

description
    is a pointer to the location where _os_gs_edt() stores the driver I/O interface
definition number. (Output)

Attributes
Operating System:    OS-9
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library
os_lib.l
Syntax
```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_eof(path_id path);
```

Description

_os_gs_eof() returns the EOS_EOF error if the current position of the file pointer associated with the specified path is at the end-of-file. SCF never returns EOS_EOF with this GetStat.

path
    specifies the driver’s path number. (Input)

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library

os_lib.l
Syntax
#include <rbf.h>
#include <types.h>
error_code _os_gs_fd(
    path_id path,
    u_int32 size,
    fd_stats *fdbuf);

Description
_os_gs_fd() returns a copy of the file descriptor sector for the file associated with the specified path.

path
    specifies the driver’s path number. (Input)

size
    is the number of bytes of the file descriptor to copy. (Input)

fdbuf
    is a pointer to the buffer for the file descriptor sector. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
 Threads: Safe
 Re-entrant: Yes

Library
os_lib.l
_os_gs_fdaddr()
Get File Descriptor Block Address for Open File

**Syntax**
```
#include <rbf.h>
#include <types.h>
error_code _os_gs_fdaddr(
    path_id path,
    u_int32 *addr);
```

**Description**

_os_gs_fdaddr() returns the file descriptor block number associated with the specified path number.

- **path** specifies the driver’s path number. (Input)
- **addr** is a pointer to the location where _os_gs_fdaddr() stores the block address of the file descriptor. (Output)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

os_lib.l
_os_gs_fdinf()

Get Specified File Descriptor Sector

Syntax
#include <rbf.h>
#include <types.h>
error_code _os_gs_fdinf(
    path_id path,
    u_int32 size,
    u_int32 fdblk,
    fd_stats *fdbuf);

Description
_os_gs_fdinf() returns a copy of the specified file descriptor sector for the file
associated with the specified file descriptor sector.

Typically, _os_gs_fdinf() is used to rapidly scan a directory on a device. You do
not need to specify the path number of the file for which you want the file
descriptor in path. However, path must be an open path on the same device as the
file. path is typically the path number of the directory you are currently scanning.

! Only super users may use this call.

path
    specifies an open path on a particular device. (Input)

size
    specifies the number of bytes of the file descriptor block to copy. (Input)

fdblk
    specifies the file descriptor sector number to get. (Input)

fdbuf
    is a pointer to the buffer for the file descriptor block. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os9_gs_free(
    path_id path,
    u_int32 *free);

Description
_os9_gs_free() returns the amount of free space on a device.

_os9_gs_free() is only supported by the NRF file manager.

path
    specifies the path number of the device. (Input)

free
    is a pointer to the location where _os9_gs_free() stores the amount of free
    space. (Output)
    The amount of free space is specified in bytes.

Attributes
Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
_os_gs_luopt()
Read Logical Unit Options

Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_luopt(
    path_id path,
    u_int32 *size,
    void *luopts);

Description
_os_gs_luopt() copies the contents of the logical unit options for a path into the buffer pointed to by luopts.

path
    specifies the driver’s path number. (Input)

size
    The size of the buffer is passed at the location pointed to by size. (Input/Output)
    _os_gs_luopt() also stores the actual number of bytes copied at the location pointed to by size. size may not be less than the size of the file manager’s logical unit option section.

luopts
    is a pointer to your buffer. (Output)

Attributes
Operating System:        OS-9
State:                   User and System
Threads:                 Safe
Re-entrant:              Yes

Library
os_lib.l
Syntax
#include <rfb.h>
#include <types.h>
error_code _os_gs_parity(
    path_id path,
    fd_stats *fdbuf,
    u_int32 *parity);

Description
_os_gs_parity() uses a 32-bit vertical parity to validate file descriptor structures.

path
    specifies an open path to the file with which the file descriptor is associated. (Input)

fdbuf
    is a pointer to the file descriptor block. (Input)

parity
    is a pointer to the location where _os_gs_parity() stores the resulting parity. (Output)

Attributes
Operating System:   OS-9
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library
os_lib.l
_os_gs_popt()

Read Path Descriptor Option Section

Syntax

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_popt(
    path_id path,
    u_int32 *size,
    void *user_opts);
```

Description

_os_gs_popt() copies the option section of the path descriptor into the variable-sized area pointed to by user_opts. You must include rbf.h, sbf.h, and/or scf.h for the corresponding file managers and declare size according to the size of the rbf_opts, sbf_opts, or scf_opts.

⚠️ On OS-9 for 68K, 128 bytes are always allocated by the binding, although only the specified number of bytes are passed back to you.

**path**

specifies the driver’s path number. (Input)

**size**

The requested number of bytes is passed at the location pointed to by size. (Input/Output)

_os_gs_popt() also stores the actual number of bytes copied at the location pointed to by size.

**user_opts**

is a pointer to your buffer. (Output)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
_os_gs_pos()

Get Current File Position

Syntax

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_pos(
    path_id path,
    u_int32 *position);
```

Description

_os_gs_pos() returns the current position of the file pointer associated with the specified path.

- **path** specifies the driver’s path number. (Input)
- **position** is a pointer to the location where _os_gs_pos() stores the file position. (Output)
  - If the device is opened in block mode (S_IBLKMODE), the position parameter returns position in terms of blocks.
  - If the device is opened in non-block mode (S_IBLKMODE), the position parameter to _os_gs_pos() returns in terms of bytes.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.1
_os9_gspump()

Return Data for Specified Process’ Memory Map

Syntax

```c
#include <process.h>
#include <types.h>
error_code _os9_gspump(
    process_id proc_id,
    u_int32 size,
    void *buffer);
```

Description

_os9_gspump() returns data about a specified process’ memory map for debugging purposes. One word in this format is returned for each memory block in the system. This information is taken from the process’ SSM data structure. If the specified address space is not large enough, only the information that fits in the area is returned.

proc_id
    specifies the process ID. (Input)

size
    specifies the size of the buffer. (Input)

buffer
    is a pointer to the storage buffer. (Output)

Attributes

- Operating System: OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

F$GSPUMp OS-9 for 68K Technical Manual
Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_gs_ready(
    path_id path,
    u_int32 *incount);
```

Description

=os_gs_ready() checks for data available to be read on the specified path. RBF devices do not return the EOS_NOTRDY error because there is always information available to be read on RBF devices. =os_gs_ready() returns the number of bytes left in incount.

If the device is opened in block mode (S_IBLKMODE), =os_gs_ready returns the number of blocks available.

path
specifies the driver’s path number. (Input)

incount
is a pointer to the location where =os_gs_ready stores the number of characters available to be read. (Output)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l
_os_gs_size()

Return Current File Size

Syntax

#include <sg_codes.h>
#include <types.h>
error_code _os_gs_size(
    path_id path,
    u_int32 *size);

Description

_os_gs_size() returns the current size of the file associated with the specified path.

path
    specifies the driver’s path number. (Input)

size
    is a pointer to the location where _os_gs_size() stores the file size. (Output)

Attributes

Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library

os_lib.l
Syntax

```c
#include <process.h>
#include <types.h>
error_code _os_id(
    process_id *proc_id,
    u_int16 *priority,
    u_int16 *age,
    int32 *schedule,
    u_int16 *group,
    u_int16 *user);
```

Description

_os_id() returns the caller’s process ID number, current process priority and age, scheduling constant, and owner ID.

⚠️ On OS-9 for 68K, the binding for this call, performs conversions which allow compatibility with OS-9. If speed is a consideration, refer to _os9_id().

proc_id
is a pointer to the location where _os_id() stores the current process ID number. (Output)

priority
is a pointer to the location where _os_id() stores the priority of the current process. (Output)

age
is a pointer to the location where _os_id() stores the age of the current process. (Output)

schedule
is a pointer to the location where _os_id() stores the scheduling constant of the current process. (Output)

group
is a pointer to the location where _os_id() stores the group number of the current process. (Output)

user
is a pointer to the location where _os_id() stores the user number of the current process. (Output)
Attributes
Operating System: OS-9 and OS-9 for 68K
State: OS-9: user and system
       OS-9 for 68K: user
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os9_id()
Syntax

```c
#include <process.h>
#include <types.h>
error_code _os9_id(  
    process_id *proc_id,
    u_int16 *priority,
    u_int16 *group,
    u_int16 *user);
```

Description

_os9_id() returns the caller’s process ID number, group and user ID, and current process priority.

- **proc_id**
  - points to the location where _os9_id() stores the process ID. (Output)

- **priority**
  - is a pointer to the location where _os9_id() stores the current process priority. (Output)

- **group**
  - points to the location where _os9_id() stores the group ID. (Output)

- **user**
  - points to location where _os9_id() stores the user ID. (Output)

Attributes

- **Operating System**: OS-9 for 68K
- **State**: User and System
- **Threads**: Safe
- **Re-entrant**: Yes

Library

os_lib.l
_os_initdata()

Initialize Static Storage from Module

Syntax

```c
#include <module.h>
#include <types.h>
error_code _os_initdata(
    mh_com *mod_head,
    void *statics);
```

Description

_os_initdata() clears the uninitialized data area, copies the module header’s initialized data to the specified data area, and clears the remote data area (if it exists). It then adjusts the code and data offsets.

- **mod_head**
  is a pointer to the module header. (Input)

- **statics**
  is a pointer to the data area. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
Set Up Signal Intercept Trap

Syntax

```c
#include <signal.h>
#include <types.h>
#include <cglob.h>
error_code _os_intercept(
    void (*func)(),
    void *data_ptr);
```

Description

_os_intercept() installs a signal intercept routine. The intercept routine is entered asynchronously because a signal may be sent at any time, similar to an interrupt. The signal code is passed as a parameter to the intercept routine.

_os_rte() must be used to exit from the intercept routine (func). If the routine fails to use _os_rte() at every possible exit, unpredictable results occur.

The intercept routine should be short and fast, such as setting a flag in the process’ data area. Avoid complicated system calls (such as I/O). Signal handlers should be careful about the following:

- Calling any function in the library that may be working with static storage structures.
- Calling any function that is not re-entrant.
- Modifying any static storage structure other than a variable of type volatile sig_atomic_t

In OS-9 for 68K, data_ptr must be _glob_data to provide compatibility between OS-9 for 68K and OS-9. Also, each time the intercept routine is called, the state of the processor (such as its registers) is pushed on to the user stack.

signal(), intercept(), and the POSIX threading library use _os_intercept() internally; do not use _os_intercept() in combination with any of these.

Only the global data pointer is set up for the signal handler, the proper constant pointer, if applicable, is not automatically set up. intercept() and signal() properly initialize both the data and constant pointers before calling the signal handling function.

`func`

is a pointer to the intercept routine’s address. (Input)

`data_ptr`

is a pointer to the intercept routine’s global storage. (Input)

It usually contains the address of the program’s data area.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
intercept()
_os_rte()
_os_send()
signal()
F$Icpt
F_ICPT

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Configure an Element of Process/System I/O

**Syntax**

```
#include <modes.h>
error_code _os_ioconfig(
    u_int32 code,
    void *param);
```

**Description**

_os_ioconfig() is a wildcard call used to configure elements of the I/O subsystem which may or may not be associated with an existing path. It is intended to be used to dynamically reconfigure system I/O resources at runtime. The target I/O resources may be system-wide resources or they may be process- or path-specific, depending on the nature of the configuration call being made.

code
is the IO configuration command code. (Input)

Refer to the OS-9 Technical Manual for a list of valid command codes for your version of the operating system.

param
is a pointer to any additional parameters required by the specified configuration function. (Input)

**Attributes**

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**See Also**

_os_config()
_os_iodel()

Check for Use of I/O Module

Syntax
#include <module.h>
#include <types.h>
error_code _os_iodel(mh_com *mod_head);

Description
_os_iodel() is executed when the kernel unlinks a file manager, device driver, or device descriptor module. It is used to determine if the I/O system is still using the module.

mod_head
is a pointer to the module header. (Input)

Attributes
Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
Syntax
#include <process.h>
#include <types.h>
error_code _os_ioexit(
    process_id proc_id,
    u_int32 path_cnt);

Description
=os_ioexit() is executed when the kernel terminates or chains to a process.

If the most significant bit of path_cnt is cleared, the process’ default data and 
execution directory paths and all other open paths in the path translation table are 
closed. The I/O process descriptor is also deallocated.

If the most significant bit of path_cnt is set, the remaining bits specify the number 
of paths to leave open. The default directory paths are not closed, and the I/O 
process descriptor is not de-allocated.

proc_id
    specifies the process ID. (Input)

path_cnt
    specifies the number of I/O paths. (Input)

Attributes
Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
_os_iofork()

Set Up I/O for New Process

Syntax
#include <process.h>
#include <types.h>

error_code _os_iofork(
    process_id par_proc_id,
    process_id new_proc_id,
    u_int32 path_cnt);

Description
=os_iofork() is executed when the kernel creates a new process. _os_iofork() creates an I/O process descriptor for the new process.

par_proc_id
is the parent’s process ID. (Input)

new_proc_id
is the process ID of the new process. (Input)

path_cnt
is the number of I/O paths. (Input)

Attributes
Operating System:   OS-9
State:              User
Threads:            Safe
Re-entrant:         Yes

Library
os_lib.l
_os9_ioqueue()

Enter I/O Queue

Syntax
#include <process.h>
#include <types.h>
error_code _os9_ioqueue(process_id proc_id);

Description
_os9_ioqueue() links the calling process into the I/O queue and performs an untimed sleep.

proc_id specifies the process ID of the process currently in control of the device. (Input)

Attributes
Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_findpd()
_os_send()
_os_sleep()
_os9_sleep()
F§IOQu

OS-9 for 68K Technical Manual
Add or Remove Device from IRQ Table

Syntax

```c
#include <regs.h>
#include <types.h>
error_code _os_irq(
    u_int32 vector,
    u_int32 priority,
    const void *irq_entry,
    void *statics);
```

Description

_os_irq() installs an IRQ service routine into the system polling table.

⚠️ The called irq function must not be compiled with stackcheck because the stack is out of range at this time.

**vector**
- specifies the vector number. (Input)
- Only the lower 16 bits of `vector` are used.

**priority**
- specifies the priority. (Input)
- Only the lower 16 bits of `priority` are used.
  - If `priority` is 0, the routine is polled first on the vector.
  - If `priority` is 1, the routine is polled first and no other device can have a priority of 1 on the vector.
  - If `priority` is 255, the routine is polled last on the vector.

**irq_entry**
- is a pointer to the IRQ service routine entry point. (Input)
  - If `irq_entry` is 0, the call deletes the IRQ service routine.

**statics**
- is a pointer to the global static storage. `statics` must be unique to the device. (Input)
Attributes
Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F_IRQ  OS-9 Technical Manual
_os9_irq()

Add or Remove Device from IRQ Table

Syntax

```c
#include <regs.h>
#include <types.h>

error_code _os9_irq(
    u_int32 vector,
    u_int32 priority,
    const void *irq_entry,
    const void *statics,
    const void *port);
```

Description

_os9_irq() installs an IRQ service routine into the system polling table.

vector

- specifies the vector number. (Input)
  - Only the lower 16 bits of vector are used.

priority

- specifies the priority. (Input)
  - Only the lower 16 bits of priority are used.
  - If priority is 0, the routine is polled first on the vector.
  - If priority is 255, the routine is polled last on the vector.

irq_entry

- is a pointer to the IRQ service routine entry point. (Input)
  - If irq_entry is 0, the call deletes the IRQ service routine.

statics

- is a pointer to the global static storage. (Input)
  - statics must be unique to the device.

port

- is the port address. (Input)
Attributes

Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

F$IRQ

OS-9 for 68K Technical Manual
Syntax
#include <time.h>
#include <types.h>
error_code _os_julian(
    u_int32 *time,
    u_int32 *date);

Description
_os_julian() converts Gregorian dates to Julian dates.

time
    is a pointer to the location where the time in the form 00hhmmss is passed.
    (Input/Output)
_os_julian()
    stores the time in seconds since midnight in the location pointed to by time.

date
    is a pointer to the location where the date in the form yyyyymmdd is passed.
    (Input/Output)
_os_julian()
    stores the Julian date in the location pointed to by date.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
**Syntax**

```c
#include <process.h>
#include <types.h>

int os9kexec(
    u_int32 (*func)(),
    u_int16 priority,
    u_int16 pathcnt,
    const char *modname,
    char **argv,
    char **envp,
    u_int32 datasize,
    void *extra1,
    void *extra2,
    void *extra3,
    void *extra4);
```

**Description**

`os9kexec()` is a high-level fork/chain interface that prepares the parameter and environment list before a process is created.

`os9kexec()` closely resembles the UNIX `execve()` system call.

`func` is a pointer to one of the following functions that create a new process:

- `_os_fork()`, `_os_forkm()`, `_os_chain()`, `_os_chainm()`, `_os_dfork()`, and `_os_dforkm`. (Input)

`func` passes information to a new process as binary data specified by a pointer and size. It is up to the forked process to interpret the data.

`priority` is the priority at which the new process is to run. (Input)

If `priority` is zero, the new process receives the priority of the calling process.

`pathcnt` is the number of open paths to pass to the new process. (Input)

Normally, `pathcnt` is 3, causing the 3 standard paths to be passed. A `pathcnt` of zero passes no open paths.
modname

is a pointer to a string naming the new primary module or to a module in the
case of the functions ending in \texttt{m} (_os_forkm(), _os_chainm(), and
_os_dforkm()). (Input)

argv

is a pointer to the parameter pointer list that the new process receives. (Input)
By convention, \texttt{argv[0]} is the name of the new module. A null pointer marks
the end of the argv list.

eenvp

is a pointer to the environment pointer list that the new process receives.
(Input)
It points to environment variables. A null pointer marks the end of the envp
list.

datasize

specifies the additional memory (in bytes) to allocate to the new process’ stack
memory. (Input)
If datasize is zero, the default stack size required by the process is used.

The first seven parameters are identical for \texttt{os}_fork(), \texttt{os}_forkm(),
\texttt{os}_chain(), \texttt{os}_chainm(), \texttt{os}_dfork(), and \texttt{os}_dforkm().

The remaining four parameters have different meanings, depending on the function
specified by func:

Table 2-15. \texttt{os9kexec()} Additional Parameters

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_fork()</td>
<td>extra1</td>
<td>Type/language of the new process</td>
</tr>
<tr>
<td></td>
<td>(Input/Output)</td>
<td>Orphan: 0 = Normal child process</td>
</tr>
<tr>
<td></td>
<td>extra2</td>
<td>1 = Process not associated with parent</td>
</tr>
<tr>
<td></td>
<td>(Input)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>extra3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Input)</td>
<td></td>
</tr>
<tr>
<td>_os_forkm()</td>
<td>extra1</td>
<td>The child process ID returns to where extra1 points</td>
</tr>
<tr>
<td></td>
<td>(Input/Output)</td>
<td>Orphan: 0 = Normal child process</td>
</tr>
<tr>
<td></td>
<td>extra2</td>
<td>1 = Process not associated with parent</td>
</tr>
<tr>
<td></td>
<td>(Input)</td>
<td></td>
</tr>
<tr>
<td>_os_chain()</td>
<td>extra1</td>
<td>Type/language of the new process</td>
</tr>
<tr>
<td></td>
<td>(Input/Output)</td>
<td>Orphan: 0 = Normal child process</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Process not associated with parent</td>
</tr>
<tr>
<td>_os_chainm()</td>
<td></td>
<td>Does not use the extra parameters</td>
</tr>
</tbody>
</table>
If `func` indicates a fork function and the initial attempt to create the child process fails due to `EOS_PNNF` (path name not found), `os9kexec()` calls `modloadp()` with `modname` to load the module from disk and attempts the fork again. This action is the same as the shell handling of executable files.

If you use `os9kexec()` when `func` is `_os_chain()` or `_os_chainc()`, you must either:

- Ensure that the module is in memory or in your current execution directory (see `modlink()`, `modload()`, and `modloadp()`).
- Provide a full pathlist to the module.

A threaded process may only chain from the `main()` thread and may not chain if multiple threads are active in the process.

`os9kexec()` returns the value of `(*func)()`, which returns 0 if successful; otherwise, it returns an error number.

Any program that executes a C program should do so using the `os9kexec()` interface. The `cstart` module can handle parameter strings passed by `os9fork()` and `chain()`, but no environment is available and the `argv` pointer list is separated by white space.

**Attributes**

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
Library

sys_clib.l

The purpose of sys_clib.l is to provide a library for compatibility with the Microware K&R C compiler. For more information, refer to Appendix A, Prototyping sys_clib.l Functions.

See Also

_os_exec()  _os_fork()  _os_forkm()  _os_chain()
_os_chainm()  _os_dfork()  _os_dforkm()

Example

#include <module.h>
#include <types.h>
#include <const.h>
extern int _os_fork();
extern char **_environ;

char * argblk[] = {
    "rename",
    "oldname",
    "newname",
    "-x",
    0,
};

main()
{
    u_int16 type_lang = mktypelang (MT_PROGRAM,ML_OBJECT);
    process_id pid;

    If ((errno = os9kexec (_os_fork,
        0, * no change in priority
        3, * pass stdin, stdout, and stderr
        angblk[0], * pointer to module name
        argblk, * the argument list
        _environ, * environment list for new process
        0, * no extra memory
        &pid * pointer to the process ID
        type_lang,
        0)) * normal child process
!= SUCCESS)
        printf ("can’t fork %s\n", argblk[0]);
    else
        wait(0)
}
**Syntax**

```c
#include <module.h>
#include <types.h>
error_code _os_link(
    char **mod_name,
    mh_com **mod_head,
    void **mod_exec,
    u_int16 *type_lang,
    u_int16 *attr_rev);
```

**Description**

`_os_link()` searches the current module directory (and an alternate module directory in OS-9) for a module whose name, type, and language match the parameters.

The module name is passed at the location pointed to by `mod_name`. On OS-9, if `mod_name` is an explicit module directory pathlist (for example, `/usr/pers1/prog`), `_os_link()` also stores the pointer to the module that was successfully linked (for example, `prog`) at the location pointed to by `mod_name` (input/output).

- **mod_head**
  - is a pointer to the location where `_os_link()` stores the address of the module's header. (Output)

- **mod_exec**
  - is a pointer to the location where `_os_link()` stores the absolute address of the module's execution entry point. (Output)

- **type_lang**
  - The type and language of the module are passed at the location pointed to by `type_lang`. (Input/Output)
  - `_os_link()` also stores the actual type and language at the location pointed to by `type_lang`. If `type_lang` is zero (specifying any), any module can be linked to regardless of the type and language.

- **attr_rev**
  - is a pointer to the location where `_os_link()` stores the attribute and revision level of the module. (Output)
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_load()
_os_unlink()
_os_unload()
F$Link
F_LINK  OS-9 for 68K Technical Manual
OS-9 Technical Manual
**Syntax**

```c
#include <module.h>
#include <types.h>
error_code _os_linkm(
    mh_com *mod_head,
    void **mod_exec,
    u_int16 *type_lang,
    u_int16 *attr_rev);
```

**Description**

_os_linkm() causes OS-9 to link to the module specified by mod_head.

- **mod_head** is a pointer to the module. (Input)
- **mod_exec** is a pointer to the location where _os_linkm() stores the absolute address of the module’s execution entry point. (Output)
- **type_lang** is a pointer to where _os_linkm() stores the type and language of the module. (Output)
- **attr_rev** is a pointer to the location where _os_linkm() stores the attribute and revision level of the module. (Output)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

os_lib.l

**See Also**

_os_load()
_os_unlink()
_os_unload()
_os_load()

Load Module(s) from File

Syntax

```c
#include <module.h>
#include <modes.h>
#include <memory.h>
#include <types.h>
error_code _os_load(
    const char *mod_name,
    mh_com **mod_head,
    void **mod_exec,
    u_int32 mode,
    u_int16 *type_lang,
    u_int16 *attr_rev,
    u_int32 color);
```

Description

_os_load() opens a file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached. Then, it closes the file. Modules are loaded into the highest physical memory available of the specified color type.

When a module is loaded, its name is added to the system module directory, and the first module read is linked. The returned parameters are the same as those returned by a link call and apply only to the first module loaded.

> _os_load() does not work on SCF devices.

_mod_name

is a pointer to the module name. (Input)

_mod_head

is a pointer to the location where _os_load() stores the pointer to the module. (Output)

_mod_exec

is a pointer to the location where _os_load() stores the pointer to the module execution entry point. (Output)
mode

specifies the access mode. (Input)

Only the lower 16 bits of mode are used.

All available access modes are defined in the following location:

MWOS\OS\SRC\DEFS\modes.h

\(\text{\textbullet} \) For a list of all available access modes, refer to Chapter 3 of the OS-9 Technical Manual.

type_lang

The type and language of the module are passed at the location pointed to by type_lang. (Input/Output)

_os_load()

also stores the actual type and language at the location pointed to by type_lang. If type_lang is zero, any module can be linked to regardless of the type and language.

attr_rev

is a pointer to the location where _os_load() stores the attribute and revision level of the module. (Output)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

F$Load \hspace{1cm} OS-9 for 68K Technical Manual
F_LOAD \hspace{1cm} OS-9 Technical Manual
Syntax
```c
#include <module.h>
#include <modes.h>
#include <types.h>
error_code _os_loadp(
    const char *name,
    u_int32 mode,
    char *nameptr,
    mh_com **mod_head);
```

Description
_os_loadp() opens a file specified by the pathlist. It reads one or more memory modules from the file into memory until an error or end of file is reached; then it closes the file.

The PATH environment variable determines the alternate directories to search for the named file.

Refer to your operating system technical manual for information regarding the PATH environment variable.

name
is a pointer to the file name. (Input)

mode
specifies the access mode. (Input)

Only the lower 16 bits of mode are used.

All available access modes are defined in the following location:
MWOS\OS\SRC\DEFS\modes.h

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the OS-9 Technical Manual.

nameptr
is a pointer to a buffer for the pathlist of the successfully loaded file. (Input)

Use NULL to indicate that no complete pathlist is required.

mod_head
is a pointer to the location where _os_loadp() stores the pointer to the module. (Output)
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_load()
_os_exec()
Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os_makdir(
    const char *name,
    u_int32 mode,
    u_int32 perm, ...);
```

Description

_os_makdir() creates and initializes a new directory as specified by the pathlist. _os_makdir() is the only way to create a new directory file. _os_makdir() fails on non-multifile devices. _os_makdir() does not return a path number. You should use _os_open() to open a directory. The new directory automatically has its directory bit set in the access permission attributes. The remaining attributes are specified by the bytes passed in mode and perm.

Refer to your operating system technical manual for the available modes and attributes.

name

is a pointer to the pathlist. (Input)

mode

specifies the access mode. (Input)

Only the lower 16 bits of mode are used.

perm

specifies the access permissions. (Input)

All available access modes and permissions are defined in the following location: MWOS\<OS>\SRC\DEFS\modes.h

For a list of all available access modes, refer to the section, “Access Modes and Permissions,” in Chapter 3 of the OS-9 Technical Manual.

A variable parameter may be included to specify the initial allocation size.
Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l
Syntax
#include <moddir.h>
#include <types.h>
error_code _os_makmdir(
    const char *name,
    u_int32 perm);

Description
_os_makmdir() creates a new module directory. The name of the new module
directory is relative to the current module directory.

name
    is a pointer to the name of the new module directory. (Input)

perm
    specifies the access permissions for the new module directory. (Input)
    Only the lower 16 bits of perm are used.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
Resizing Data Memory Area

Syntax

```c
#include <memory.h>
#include <types.h>
error_code _os_mem(
    u_int32 *size,
    void **mem_ptr);
```

Description

_os_mem() contracts or expands the process' data memory area. The requested size is rounded up to an even memory allocation block. Additional memory is allocated contiguously upward (towards higher addresses) or deallocated downward from the old highest address.

-size
The desired memory size in bytes is passed at the location pointed to by size. (Input/Output)

_os_mem() also stores the actual size of the memory at the location pointed to by size. If the integer pointed to by size is zero, _os_mem() treats the call as a request for information and returns the current upper bound in mem_ptr and the amount of free memory in size.

-mem_ptr
is a pointer to the location where _os_mem() stores the pointer to the new end of data segment plus 1. (Output)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

F\$Mem
F_MEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual
_os_mkmodule()

Create Data Module of Specified Color Type

Syntax

```c
#include <module.h>
#include <memory.h>
#include <types.h>
error_code _os_mkmodule(
    const char *mod_name,
    u_int32 size,
    u_int16 *attr_rev,
    u_int16 *type_lang,
    u_int32 perm,
    void **mod_exec,
    mh_com **mod_head,
    u_int32 color);
```

Description

_os_mkmodule() creates a memory module with the specified name and attributes in the specified memory. The module is created with a valid CRC and entered into the system module directory. You can specify the type/language of the module and color of memory in which to make the module.

The created module always begins at offset $34$ within the module. This implies that this call cannot conveniently make program modules, trap handlers, file managers, device drivers, and device descriptors.

- **mod_name** is a pointer to the name of the data module. (Input)
- **size** specifies the size of the data portion to clear. (Input)
- **attr_rev** is a pointer to the location where _os_mkmodule() stores the module’s attribute and revision. (Output)
- **type_lang**
  - The type and language for the module are passed at the location pointed to by type_lang. (Input/Output)
  - _os_mkmodule() also stores the actual type and language at the location pointed to by type_lang.
perm
  specifies the access permissions for the module. (Input)
  Only the lower 16 bits of perm are used.

mod_exec
  is a pointer to the location where _os_mkmodule() stores the pointer to the
  module data. (Output)

mod_head
  is a pointer to the location where _os_mkmodule() stores the pointer to the
  module header. (Output)

color
  is the memory color type: SYSRAM, VIDEO1, or VIDEO2. (Input)
  If color is zero, no memory type is specified. Consequently, the module is
  made in whatever memory the system allocates.

Attributes
  Operating System: OS-9 and OS-9 for 68K
  State: User and System
  Threads: Safe
  Re-entrant: Yes

Library
  os_lib.l

See Also
  _os_datmod()
  _os_setcrc()
  _os_move()
Syntax

```c
#include <module.h>
#include <types.h>
error_code _os_modaddr(
    void *mem_ptr,
    mh_com **mod_head);
```

Description

_os_modaddr() locates a module given a pointer to any position within the module and returns a pointer to the module's header.

- **mem_ptr**
  - is a pointer to any position within the module. (Input)

- **mod_head**
  - is a pointer to the location where _os_modaddr() stores the pointer to the module header. (Output)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
Syntax
#include <memory.h>
#include <types.h>
error_code _os_move(
    void *from,
    void *to,
    u_int32 count);

Description
_os_move() is a fast “block-move” subroutine which copies data bytes from one
address space to another, usually from system to user or vice versa. This call allows
the source and destination buffers to overlap.

from
    is a pointer to the source data. (Input)

to
    is a pointer to the destination data. (Output)

count
    is the byte count to copy. (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              System
Threads:            Safe
Re-entrant:         Yes

Library
os_lib.l
**Syntax**

```c
#include <process.h>
#include <types.h>
error_code _os_nproc(void);
```

**Description**

_os_nproc() removes the next process from the active process queue and initiates its execution. _os_nproc() does not return to the caller.

The process calling _os_nproc() should already be in one of the system’s process queues. If not, the process becomes unknown to the system. This occurs even though the process descriptor still exists and is printed out by a procs command.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

os_lib.1

**See Also**

_os_aproc()
Syntax

```
#include <modes.h>
#include <types.h>
error_code _os_open(
    const char *name,
    u_int32 mode,
    path_id *path);
```

Description

_os_open() opens a path to an existing file or device as specified by the pathlist. _os_open() returns a path number which is used in subsequent service requests to identify the path. If the file does not exist, an error is returned.

A non-directory file may be opened with no mode bits set. This allows you to examine characteristics such as the attributes and size by the _os_getstat() system requests, but does not permit any actual I/O on the path.

name is a pointer to the path name of the existing file or device. (Input)

type specifies which subsequent read and/or write operations are permitted. (Input)

Only the lower 16 bits of mode are used.

All available access modes are defined in the following location:

MWOS\OS\SRC\DEFS\modes.h

path is a pointer to the location where _os_open() stores the resulting path number. (Output)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

=os_attach() =os_close() =os_create()
I$Open
I_OPEN

OS-9 for 68K Technical Manual
OS-9 Technical Manual
_os9_panic()

System Catastrophic Occurrence

Syntax

#include <process.h>
#include <types.h>

error_code _os9_panic(
    u_int32 panic_code);

Description

_os9_panic() is called when the kernel detects a disastrous, but not necessarily fatal, system condition. Ordinarily, _os9_panic() is undefined and the system dies.

_os9_panic() is called only when the kernel believes there are no processes remaining to be executed. Although the system is probably dead at this point, it may not be. Interrupt service routines or system-state alarms could cause the system to become active.

panic_code
    specifies the panic code to output. (Input)

Attributes

Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os9_setsvc()

F$Panic OS-9 for 68K Technical Manual
**Syntax**

```c
#include <process.h>
#include <types.h>
#include <modes.h>
error_code _os_permit(
    void *mem_ptr,
    u_int32 size,
    u_int32 perm,
    process_id pid);
```

**Description**

_os_permit() allows a process to access a block of memory. The _os_permit() service routine must update SSM (System Security Module) data structures to show that a process may access the specified memory in the requested mode.

_os_permit() only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, _os_permit() simply returns without allowing access to the memory block.

Only super-group users may use the _os_permit() function.

- **mem_ptr**
  - is a pointer to the beginning of the memory area to grant access permissions.
  - (Input)

- **size**
  - is the size of the memory area.
  - (Input)

- **perm**
  - is the permissions to grant.
  - (Input)
  - Only the lower 16 bits of perm are used.

- **pid**
  - is the process identifier of the target process (not used for OS-9 for 68K).
  - (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
_os_permit()

Library

os_lib.1

See Also

F$Permit

OS-9 Technical Manual

F_PERMIT

OS-9 for 68K Technical Manual
Syntax

```
#include <errno.h>
#include <types.h>
error_code _os_perr(
    path_id path,
    u_int32 errnum);
```

Description

_os_perr() is the system’s error reporting facility. It writes an error message to the standard error path. Most OS-9 for 68K systems print ERROR #mmm.nnn. Error numbers 000:000 to 063:255 are reserved for the operating system.

If an error path number is specified, the path is searched for a text description of the error encountered. The error message path contains an ASCII file of error messages. Each line may be up to 80 characters long. If the error number matches the first 7 characters in a line (that is, 000:215), the rest of the line is printed along with the error number.

Error messages may be continued on several lines by beginning each continuation line with a space. An example error file might contain lines like this:

```
000:214 (E$FNA) File not accessible.
An attempt to open a file failed. The file was found, but is not accessible to you in the requested mode. Check the file’s owner ID and access attributes.

000:215 (E$BPNam) Bad pathlist specified.
The specified pathlist is syntactically incorrect.
```

path

is the path to the error message file. (Input)

If path is zero, standard format is specified.

eralnum

is the error number. (Input)

Only the lower 16 bits of errnum are used.

Attributes

Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l
Prevent Access to Memory Block

Syntax

```c
#include <process.h>
#include <types.h>
error_code _os_protect(
    void *mem_ptr,
    u_int32 size,
    u_int32 perm,
    process_id pid);
```

Description

_os_protect() removes a process’ permission to access a block of memory.
_os_protect() only performs a useful function on MMU/SSM systems. On non-MMU/SSM systems, _os_protect() simply returns without preventing access to the memory block.

⚠️ Only super-group users may use the _os_protect function.

- **mem_ptr**
  is a pointer to the beginning of the memory area to remove access permissions. (Input)

- **size**
  specifies the size of memory. (Input)

- **perm**
  specifies the permissions to remove. (Input)
  Only the lower 16 bits of perm are used.

- **pid**
  specifies the process identifier for the target process. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`os_lib.l`

See Also

- F$Protect
- F_PROTECT
  - OS-9 for 68K Technical Manual
  - OS-9 Technical Manual
Syntax

```c
#include <strings.h>
#include <types.h>
error_code _os_prsnam(
  const char *name,
  u_int32 *length,
  char *delimiter,
  char **updated);
```

Description

_os_prsnam() parses a string for a valid pathlist element and returns its size. This call parses one element in a pathlist, not the entire pathlist. A valid pathlist element may contain the following characters:

- A - Z: Upper case letters
- .: Periods
- a - z: Lower case letters
- _: Underscores
- 0 - 9: Numbers
- $: Dollar signs

Other characters terminate the name and are returned as the pathlist delimiter. Several _os_prsnam() calls are needed to process a pathlist with more than one name. _os_prsnam() terminates a name when it reaches a delimiter character. Usually, pathlists must be terminated with a null byte.

name
is a pointer to the pathlist string. (Input)

length
is a pointer to the location where _os_prsnam() stores the length of the pathlist element. (Output)

delimiter
is a pointer to the location where _os_prsnam() stores the pathlist delimiter. (Output)

updated
is a pointer to the location where _os_prsnam() stores the pointer to the first character of name. (Output)
### Attributes
- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User and System
- **Threads**: Safe
- **Re-entrant**: Yes

### Library
- `os_lib.l`

### See Also
- `_os_cmpnam()`
Syntax

```c
#include <io.h>
#include <types.h>
error_code _os_rdalst(
    char *buffer,
    u_int32 *count);
```

Description

_con_rdalst() copies the system alias list to the caller's buffer. At most, _count bytes are copied to the buffer. Each alias entry is null terminated.

_buffer
    is a pointer to the buffer into which to copy the alias list. (Output)

_count
    The number of total bytes to copy is passed at the location pointed to by _count. (Input/Output)

_con_rdalst() also stores the actual number of bytes copied at the location pointed to by _count.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`os_lib.1`
Read Data from File or Device

Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os_read(
    path_id path,
    void *buffer,
    u_int32 *count);
```

Description

_os_read() reads the number of bytes from the specified path. The path must previously have been opened in read or update mode. The data is returned exactly as read from the file or device without additional processing or editing such as backspace and line delete. If not enough data is in the file to satisfy the read request, fewer bytes are read than requested, but an end-of-file error is not returned until the next _os_read() service request returns an end-of-file error.

If the device is opened in block mode (S_IBLKMODE), the count parameter passed must be a multiple of the block size of the device, whereas in non-block mode it is interpreted as the number of bytes to read.

- **path**
  - specifies the path number. (Input)

- **buffer**
  - is a pointer to the caller’s data buffer. (Output)

- **count**
  - The number of bytes requested to be read is passed at the location pointed to by count. (Input/Output)

  _os_read() also stores the number of bytes actually read at the location pointed to by count.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
See Also

_os_readln()

I$Read
I_READ

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax

#include <modes.h>
#include <types.h>
error_code _os_readln(
    path_id path,
    void *buffer,
    u_int32 *count);

Description

_os_readln() reads the specified number of bytes from the input file or device until an end-of-line character is encountered. On SCF-type devices, _os_readln() also causes line editing such as backspacing, line delete, echo, and automatic line feed to occur.

If the device is opened in block mode (S_IBLKMODE), the count parameter passed must be a multiple of the block size of the device, whereas in non-block mode it is interpreted as the number of bytes to read.

path
    specifies the path number. (Input)

buffer
    is a pointer to the location where the bytes to be read are placed. (Output)

count
    The number of bytes requested is passed at the location pointed to by count. (Input/Output)
    _os_readln() also stores the number of bytes actually read at the location pointed to by count.

Attributes

Operating System:       OS-9 and OS-9 for 68K
State:                  User and System
Threads:                Safe
Re-entrant:             Yes

Library

os_lib.l

See Also

_os_read()  
I$ReadLn       OS-9 for 68K Technical Manual
I_READLN        OS-9 Technical Manual
**Syntax**

```c
#include <lock.h>
#include <types.h>
error_code _os_rellk(
    lk_desc *lock);
```

**Description**

_os_rellk() releases ownership of a resource lock and activates the next process waiting to acquire the lock. The next process in the lock’s queue is activated and granted exclusive ownership of the resource lock. If no other process is waiting on the lock, the lock is simply marked free for acquisition.

`lock` is a pointer to the `lk_desc` structure of the lock to release. (Input)

**Attributes**

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

**Library**

`lock.l`

**See Also**

_os_acqlk()
_os_caqlk()
_os_crlk()
_os_dcll()
_os_dellk()
_os_waitlk()
F_RELLK

*OS-9 Technical Manual*
Syntax
#include <process.h>
#include <types.h>
error_code _os9_retpd(
    u_int32 number,
    void *table);

Description
_os9_retpd() deallocates a fixed block of memory.

number is the number of the table element. (Input)
    Only the lower 16 bits of number are used. (Input)
table is a pointer to the address of the table to use.

Attributes
Operating System:   OS-9 for 68K
State:             System
Threads:           Safe
Re-entrant:        Yes

Library
os_lib.l

See Also
_os9_allpd()
_os_findpd()
Return from Signal Interrupt Routine

Syntax

```c
#include <signal.h>
#include <types.h>
error_code _os_rte(void);
```

Description

(_os_rte()) terminates a process’ signal intercept routine and continues executing the main program.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

- os_lib.l

See Also

- _os_intercept()
- F$RTE
- F_RTE

OS-9 for 68K Technical Manual
OS-9 Technical Manual
_os_rtnprc()

Deallocate Process Descriptor

Syntax

```c
#include <process.h
#include <types.h
error_code _os_rtnprc(
    process_id proc_id);
```

Description

_os_rtnprc() deallocates a process descriptor previously allocated by _os_alocproc(). You must ensure that the process’ system resources are returned before calling _os_rtnprc().

`proc_id` identifies the process descriptor. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: System
- Threads: Safe
- Re-entrant: Yes

Library

`os_lib.l`

See Also

- _os_alocproc()
- _os_findpd()
_os9_salarm_atdate()

Execute System-State Subroutine at Gregorian Date/Time

Syntax

```c
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_atdate(
    alarm_id *alrm_id,
    u_int32 time,
    u_int32 date,
    REGISTERS *regs);
```

Description

_os9_salarm_atdate() executes a system-state subroutine at a specific date and time. The alarm subroutine executes anytime the system date/time becomes greater than or equal to the alarm time.

> _os9_salarm_atdate() only allows you to specify the time to the nearest second. However, it does adjust if the system's date and time have changed (using _os9_setime()).

> Functions installed as system-state alarm routines should be compiled without stack checking and with stand-alone constant data pointer calculation (-bepg), if appropriate for the target processor. Failure to do so can lead to a system crash or system data structure corruption.

alrm_id

is a pointer to the location where _os9_salarm_atdate() stores the alarm ID.
(Output)

time

is the time for the alarm to go off in the form 00hhmmss. (Input)

date

is the date for the alarm to go off in the form yyyymmdd. (Input)

regs

is a pointer to the register image to pass to the function. (Input)

The address of the system-state subroutine to execute should be set in regs->pc.
Attributes
Operating System: OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os9_setime()
F_ALARM

OS-9 for 68K Technical Manual
**_os_salarm_atime()**

**Execute System-State Subroutine at Specified Time**

**Syntax**

```c
#include <alarm.h>
error_code _os_salarm_atime(
    salarm_id *alrm_id,
    u_int32 time,
    u_int32 flags,
    u_int32 (*func)(),
    void *func_pb);
```

**Description**

_os_salarm_atime() executes the specified subroutine after the specified time interval has elapsed. The alarm time is an absolute time, not a relative time. The time value is considered to be a value in seconds since a date specified in Greenwich Mean Time. To determine the specified date, reference the applicable manual, *OS-9 for 68K Technical Manual* or *OS-9 Technical Manual*.

⚠️ Functions installed as system-state alarm routines should be compiled without stack checking and with stand-alone constant data pointer calculation (-bepg), if appropriate for the target processor. Failure to do so can lead to a system crash or system data structure corruption.

alrm_id

is a pointer to the location where _os_salarm_atime() stores the alarm ID. (Output)

time

specifies the time when the alarm expires. (Input)

flags

specifies the alarm flags. (Input)

func

is a pointer to a function to execute. (Input)

func_pb

is a pointer to the function’s parameter block. (Input)

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe
- Re-entrant: Yes
_os_salarm_atime()

Library

os_lib.l

See Also

_os_salarm_set()
F_ALARM

OS-9 Technical Manual
Syntax
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_atjul(
    alarm_id *alrm_id,
    u_int32 time,
    u_int32 date,
    REGISTERS *regs);

Description
_os9_salarm_atjul() executes a system-state subroutine at a specific Julian date
and time. The alarm subroutine is executed anytime the system date/time becomes
greater than or equal to the alarm time.

⚠️ Functions installed as system-state alarm routines should be compiled without
stack checking and with stand-alone constant data pointer calculation (-bepg),
if appropriate for the target processor. Failure to do so can lead to a system
-crash or system data structure corruption.

_os9_salarm_atjul() only allows the time to be specified to the nearest second.
However, it does adjust if the system’s date and time have changed (using
_os9_setime()).

alrm_id
    is a pointer to the location where _os9_salarm_atjul() stores the alarm ID.
    (Output)

time
    is the time for the alarm to go off in seconds after midnight. (Input)

date
    is the Julian day number for the alarm to go off. (Input)

regs
    is a pointer to the register image to pass to the function. (Input)
    The address of the system-state subroutine to execute should be set in
    regs->pc.
Attributes
Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os9_setselime()
F$Alarm
OS-9 for 68K Technical Manual
_os_salarm_cycle()

Set Cyclic Alarm Clock

Syntax

```c
#include <alarm.h>
#include <types.h>

error_code _os_salarm_cycle(
    alarm_id *alrm_id,
    u_int32 time,
    u_int32 flags,
    u_int32 (*func)(),
    void *func_pb);
```

Description

_os_salarm_cycle() executes the specified subroutine after the specified time interval has elapsed. Then the alarm is reset to provide recurring periodic execution of the specified subroutine.

⚠️ Functions installed as system-state alarm routines should be compiled without stack checking and with stand-alone constant data pointer calculation (-bepg), if appropriate for the target processor. Failure to do so can lead to a system crash or system data structure corruption.

- **alrm_id** is a pointer to the location where _os_salarm_cycle() stores the alarm ID. (Input)
- **time** specifies the alarm’s time interval. (Input)
  - time may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.
- **flags** specifies the alarm flags. (Input)
- **func** is a pointer to a function to execute. (Input)
- **func_pb** is a pointer to the function’s parameter block. (Input)
Attributes

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_salarm_set()
F_ALARM
_OS-9 Technical Manual
Execute System-State Subroutine Every N Ticks/Seconds

Syntax

```c
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_cycle(
    alarm_id *alrm_id,
    u_int32 time,
    REGISTERS *regs);
```

Description

_os9_salarm_cycle() executes a system-state subroutine after the specified time interval has elapsed. Then the alarm is reset to provide recurring periodic execution of the system-state subroutine.

![Warning]

Functions installed as system-state alarm routines should be compiled without stack checking and with stand-alone constant data pointer calculation (-bepg), if appropriate for the target processor. Failure to do so can lead to a system crash or system data structure corruption.

Keep cyclic system-state alarms as fast as possible. Schedule them with as long a cycle as possible to avoid consuming a large portion of available CPU time.

_alrm_id

is a pointer to the location where _os9_salarm_cycle() stores the alarm ID. (Output)

time

specifies the alarm’s time interval. (Input)

time may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.

Regs

is a pointer to the register image to pass to the function. (Input)

The address of the system-state subroutine to execute should be set in regs->pc.
Attributes
Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os9_salarm_set()
F$Alarm
OS-9 for 68K Technical Manual
_os_salarm_delete()

Remove Pending Alarm Request

Syntax

#include <alarm.h>
#include <types.h>
error_code _os_salarm_delete(alarm_id alrm_id);

Description

_os_salarm_delete() removes a cyclic alarm or any alarm that has not expired.

In a threaded environment, the alarm_id is checked to determine the owner of the thread. If a 0 is passed for the alarm_id, all pending alarm requests for the system will be removed, which are owned by the system process (forked with TH_SPOWN).

Keep cyclic system-state alarms as fast as possible. Schedule them with as long a cycle as possible to avoid consuming a large portion of available CPU time.

alrm_id specifies the alarm ID. If alrm_id is 0 (zero), all pending alarm requests are removed, which are owned by the system process (forked with TH_SPOWN).

(Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

F$Alarm OS-9 for 68K Technical Manual
F_ALARM OS-9 Technical Manual
Delete Installed Alarms

Syntax
#include <alarm.h>
#include <types.h>
error_code _os_salarm_delete_sp(alrm_id, flags);

Description
_os_salarm_delete_sp() takes a flag value to allow correct deletion of installed alarms, thus providing a workaround for a historic OS-9 bug in _os_salarm_delete.

⚠️ Functions installed as system-state alarm routines should be compiled without stack checking and with stand-alone constant data pointer calculation (-bepg), if appropriate for the target processor. Failure to do so can lead to a system crash or system data structure corruption.

alrm_id specifies the alarm ID. (Input)
If alrm_id is zero, all pending alarm requests are removed.

flags specifies that the alarm(s) to be deleted are owned by either the system process (TH_SPOWN) or by the current user process (flags=0). (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F$Alarm OS-9 for 68K Technical Manual
F_ALARM OS-9 Technical Manual
Syntax

```c
#include <alarm.h>
#include <types.h>
error_code _os_salarm_reset(
    alarm_id alrm_id,
    u_int32 time,
    u_int32 flags,
    u_int32 (*func)(),
    void *func_pb);
```

Description

_os_salarm_reset() allows you to change the parameters on an existing alarm.

⚠️ Functions installed as system-state alarm routines should be compiled without stack checking and with stand-alone constant data pointer calculation (-bepg), if appropriate for the target processor. Failure to do so can lead to a system crash or system data structure corruption.

- **alrm_id**
  - specifies the alarm ID to reset. (Input)

- **time**
  - specifies the alarm’s time interval or a time. (Input)
  - A time interval may be specified in system clock ticks or 256ths of a second.
  - The minimum time interval allowed is 2 system clock ticks.

- **flags**
  - specifies the alarm flags. (Input)

- **func**
  - is a pointer to a function to execute. (Input)

- **func_pb**
  - is a pointer to the function’s parameter block. (Input)

Attributes

- **Operating System:** OS-9
- **State:** System
- **Threads:** Safe
- **Re-entrant:** Yes
Library
os_lib.l

See Also
_os_salarm_set()
F_ALARM

OS-9 Technical Manual
Set Alarm Clock

Syntax

```c
#include <alarm.h>
#include <types.h>
error_code _os_salarm_set(
    alarm_id *alrm_id,
    u_int32 time,
    u_int32 flags,
    u_int32 (*func)(),
    void *func_pb);
```

Description

_os_salarm_set() executes the specified subroutine after the specified time interval has elapsed.

⚠️ Functions installed as system-state alarm routines should be compiled without stack checking and with stand-alone constant data pointer calculation (-bepg), if appropriate for the target processor. Failure to do so can lead to a system crash or system data structure corruption.

_alrm_id

is a pointer to the location where _os_salarm_set() stores the alarm ID. (Output)

time

specifies the alarm’s time interval or a time. (Input)

time may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second. Usually, the minimum time interval allowed is 2 system clock ticks.

flags

specifies the alarm flags. (Input)

func

is a pointer to the function to execute. (Input)

func_pb

is a pointer to the function’s parameter block. (Input)

Attributes

- Operating System: OS-9K
- State: System
- Threads: Safe
- Re-entrant: Yes
_os_salarm_set()

Library

os_lib.l

See Also

F_ALARM

OS-9 Technical Manual
Syntax
#include <regs.h>
#include <alarm.h>
#include <types.h>
error_code _os9_salarm_set(
    alarm_id *alrm_id,
    u_int32 time,
    REGISTERS *regs);

Description
_os9_salarm_set() executes a system-state subroutine after the specified time interval has elapsed.

⚠️ Functions installed as system-state alarm routines should be compiled without stack checking and with stand-alone constant data pointer calculation (-bepg), if appropriate for the target processor. Failure to do so can lead to a system crash or system data structure corruption.

alarm_id
points to the location where _os9_salarm_set() stores the alarm ID. (Output)

time
specifies the time interval. (Input)
    The time interval, time, may be specified in system clock ticks or 256ths of a second. The minimum time interval allowed is 2 system clock ticks.

regs
is a pointer to the register image to pass to the function. (Input)
    The address of the system-state subroutine to execute should be set in regs->pc.

Attributes
Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F$Alarm OS-9 for 68K Technical Manual
Syntax

```c
#include <regs.h>
#include <cache.h>
#include <types.h>
error_code _os_scache(
    u_int32 control,
    void (**cctl)(),
    void **cctl_data,
    void *addr,
    u_int32 size);
```

Description

_**os_scache()**_ performs operations on the system instruction and/or data caches.

**control** specifies the cache operation. (Input)

If **control** is zero, the system instruction and data caches are flushed. Non-super group, user-state processes may perform this operation.

The table below illustrates the bits that are defined in the **control** parameter for precise operation. Only system-state and super-group processes can use the **control** parameter for precise operation. All others are reserved. If a reserved bit is set, an **EOS_PARAM** error is returned.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>C_ENDATA</td>
<td>If set, enables the data cache.</td>
</tr>
<tr>
<td>Bit 1</td>
<td>C_DISDATA</td>
<td>If set, disables the data cache.</td>
</tr>
<tr>
<td>Bit 2</td>
<td>C_FLDATA</td>
<td>If set, flushes the data cache.</td>
</tr>
<tr>
<td>Bit 3</td>
<td>C_INVDATA</td>
<td>invalidates data cache (if set and supported by hardware).</td>
</tr>
<tr>
<td>Bit 4</td>
<td>C_ENINST</td>
<td>If set, enables the instruction cache.</td>
</tr>
<tr>
<td>Bit 5</td>
<td>C_DISINST</td>
<td>If set, disables the instruction cache</td>
</tr>
<tr>
<td>Bit 6</td>
<td>C_FLINST</td>
<td>If set, flushes the instruction cache.</td>
</tr>
<tr>
<td>Bit 7</td>
<td>C_INVINNST</td>
<td>If set, invalidates the instruction cache.</td>
</tr>
<tr>
<td>Bit 8</td>
<td>C_ADDR</td>
<td>If set, flags a target address for flush operation.</td>
</tr>
<tr>
<td>Bits 9-14</td>
<td>Reserved for future use by RadiSys.</td>
<td></td>
</tr>
<tr>
<td>Bit 15</td>
<td>C_GETCCTL</td>
<td>If set, returns a pointer to the cache control routine and cache static global data.</td>
</tr>
<tr>
<td>Bit 16</td>
<td>C_STODATA</td>
<td>stores data cache (if set and supported by hardware).</td>
</tr>
<tr>
<td>Bits 17-31</td>
<td>Reserved for future use by RadiSys.</td>
<td></td>
</tr>
</tbody>
</table>
By OS-9 convention, the terms flush, invalidate, and store are defined as follows:

Flush
Operation by which a cache line is written to main memory and marked as invalid.

Invalidate
Operation by which a cache line is marked as invalid. No write of cache line to main memory is performed.

Store
Operation by which a cache line is written to main memory. The cache line is not marked as invalid.

These specific operations may or may not be supported by a particular hardware configuration. The Flush operation is considered the base operation and will be supported by most processors.

cctl
is a pointer to the location where _os_scache() stores the cache routine’s address. (Output)

cctl_data
is a pointer to the location where _os_scache() stores the cache routine’s static storage address. (Output)

addr
is a pointer that specifies the address for the flush operation. (Input)

This parameter is used when the C_ADDR bit of control is set. Set addr to 0 when C_ADDR is not set.

size
indicates the size of the target memory area to be flushed. (Input)

This parameter is used when the C_ADDR bit of control is set. Set size to 0 when C_ADDR is not set.

Any program that builds or changes executable code in memory should flush the instruction cache with _os_scache() before executing new code. This is necessary because the hardware instruction cache is not updated by data (write) accesses and may contain the unchanged instruction(s). For example, if a subroutine builds a system call on its stack, you must use the _os_scache() system call to flush the instruction cache before executing the temporary instructions.

If the C_GETCCTL bit of control is set, _os_scache() returns a pointer to the cache control routine in the cache extension module and a pointer to that routine’s static global data. This enables drivers and file managers to call the cache routine directly rather than making a possibly time-consuming _os_scache() request. The prototype for this cache control routine is as follows:

```
error_code (*cache_cctl) (u_int32 control,
    void *addr, u_int32 size);
```
The control, addr, and size parameters are as defined earlier. The only exception is the C_GETCCTL bit in the control parameter. This obviously has no meaning in this context.

**Attributes**

Operating System: OS-9  
State: System  
Threads: Safe  
Re-entrant: Yes

**Library**

os_lib.l

**See Also**

I_SETSTAT, SS_CACHE  
F_CCTL (system-state)  
OS-9 Technical Manual
Syntax

```c
#include <memory.h>
#include <types.h>
error_code _os9_schbit(
    u_int16 *bit_number,
    u_int16 *count,
    void *address,
    void *bitmap_end);
```

Description

_os9_schbit() searches the specified buffer which represents a bit map for a free block (cleared bits) of the required length, starting at the beginning bit number. _os9_schbit() returns the offset of the first block found of the specified length.

If _os9_schbit() returns SUCCESS, you should check count to make sure that you received the requested number of bits. If count is less than the requested amount, _os9_schbit() returned the largest number of bits available, starting at bit_number.

**bit_number**

The beginning bit number to search is passed at the location pointed to by bit_number. (Input/Output)

_os9_schbit() also stores the beginning bit number found at the location pointed to by bit_number.

**count**

The number of bits needed is passed at the location pointed to by count. (Input/Output)

_os9_schbit() also stores the number of bits found at the location pointed to by count.

**address**

is a pointer to the base address of the bitmap. (Input)

**bitmap_end**

is a pointer to the end of the bitmap plus one. (Input)

Attributes

- Operating System: OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
Library
os_lib.l

See Also
=os9_allbit()
=os9_delbit()
Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os_seek(
    path_id path,
    u_int32 position);
```

Description

_os_seek() repositions the path’s file pointer. The file pointer is the 32-bit address of the next byte in the file to read or write. A seek may be performed to any value, even if the file is not large enough. Subsequent write requests automatically expand the file to the required size, if possible. read requests return an end-of-file condition.

If the device is opened in block-mode (S_IBLKMODE), the position parameter to _os_seek() is interpreted by the function as a block to seek to (rather than as a byte to seek to, which is the behavior when in non-block mode).

path
    specifies the path number. (Input)

position
    specifies the new position in the file. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

I$Seek OS-9 for 68K Technical Manual
I_SEEK OS-9 Technical Manual
Syntax

```c
#include <semaphore.h>
error_code _os_sema_init(semaphore *sema);
```

Description

OS-9 for 68K: \_os\_sema\_init() is valid in version 3.0 and greater.

\_os\_sema\_init() initializes a semaphore data structure for subsequent use. The application should call \_os\_sema\_init() before the first use of the semaphore. The semaphore may be part of a data module or it may be part of any data structure within the application.

It is necessary to clear the shared memory area before making the \_os\_sema\_init() call. In some instances, such as with driver applications or global variables, initializing the structure to zero will clear the shared memory area automatically.

**sema**

is a pointer to the semaphore data structure to initialize. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os\_lib\_l

See Also

- \_os\_sema\_p()
- \_os\_sema\_term()
- \_os\_sema\_v()
Reserve Semaphore (Acquire Exclusive Access)

**Syntax**

```c
#include <semaphore.h>
error_code _os_sema_p(semaphore *sema);
```

**Description**

OS-9 for 68K: `_os_sema_p()` is valid in version 3.0 and greater.

 `_os_sema_p()` attempts to reserve the semaphore. If the semaphore is busy, the caller is suspended until the current holder releases the semaphore.

 `_os_sema_p()` returns an error code if a signal arrives while a process is waiting for a semaphore (i.e., it signals that the process is resuming execution--because of the incoming signal--without obtaining the semaphore).

**Attributes**

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

**Library**

`os_lib.l`

**See Also**

- `_os_sema_init()`
- `_os_sema_term()`
- `_os_sema_v()`
_os_sema_term()

Terminate Use of Semaphore Data Structure

Syntax

```c
#include <semaphore.h>
error_code _os_sema_term(semaphore *sema);
```

Description

OS-9 for 68K: `_os_sema_term()` is valid in version 3.0 and greater.

`_os_sema_term()` terminates the use of the semaphore. The application should call `_os_sema_term()` before terminating the application.

⚠️ Within OS-9 and OS-9 for 68K, each call to `_os_sema_term()` invalidates the sync code of the semaphore. Therefore, the last process to use the semaphore should be the only one to terminate it.

`sema` is a pointer to the semaphore data structure to terminate. (Input)

Attributes

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User and System
- **Threads**: Safe
- **Re-entrant**: Yes

Library

`os_lib.l`

See Also

- `_os_sema_init()`
- `_os_sema_p()`
- `_os_sema_v()`
Release Semaphore (Release Exclusive Access)

Syntax
#include <semaphore.h>
error_code _os_sema_v(semaphore *sema);

Description
OS-9 for 68K: _os_sema_v() is valid in version 3.0 and greater.

_os_sema_v() releases the reservation of a semaphore. If a process is waiting on the semaphore, it is activated, allowing it to attempt to acquire the semaphore.

The _os_sema_v() call releases the semaphore, but does not give it to the next process waiting for it. Instead, if the process that issued the _os_sema_v() call issues a _os_sema_p() call before its slice is up, then it re-obtains the semaphore (i.e., the waiting process if running at the same priority was unable to obtain it).

sema

is a pointer to the semaphore to release. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_sema_init()
_os_sema_p()
_os_sema_term()
Send Signal to Another Process

Syntax

```c
#include <signal.h>
#include <types.h>
error_code _os_send(
    process_id proc_id,
    signal_code signal);
```

Description

_os_send() sends a signal to the specified process. A process may send the same signal to all processes of the same group/user ID by passing zero as the receiving process’ ID number.

If the receiving process does not have a signal handler installed, the signal will cause the receiving process to be killed. If the receiving process has a signal handler and is blocked on I/O, then a variety of error or result codes may be returned to the mainline program depending on the I/O system in use. For example, SCF in OS-9 for 68K would return the signal number send by _os_send(). SPF would return an error code of \(000:233\).

On OS-9 for 68K, the lower 16 bits of the signal value are used. The upper 16 bits are ignored.

There are times in which the kernel sends a signal to a threaded process ID. When this occurs, the kernel looks for the first thread containing a signal handler and delivers the signal to it.

proc_id

is the process ID number for the intended receiver. (Input)

signal

specifies the signal code to send. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.1

See Also

-_os_wait() _os_intercept() _os_sleep()
F$Send
F_SEND

OS-9 for 68K Technical Manual
OS-9 Technical Manual
_os_setcrc()
Generate Valid CRC in Module

Syntax
#include <module.h>
#include <types.h>
error_code _os_setcrc(mh_com *mod_head);

Description
_os_setcrc() updates the header parity and CRC of a module in memory. The module must have the correct size and sync bytes; other parts of the module are not checked.

Although _os_setcrc() and _setcrc() functions update the header and crc of a module in memory, the copy of the module header in the module directory is not affected. Therefore, you have an invalid module directory entry for the memory module with the new CRC; and an error 236 results if links to the module are attempted.

mod_head is a pointer to the module. (Input)
The module image must start on a longword address or an exception may occur.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_crc()
_os_setime()

Set System Time

Syntax

#include <time.h>
#include <types.h>
error_code _os_setime(u_int32 time);

Description

_os_setime() sets the current system time and starts the system real-time clock to produce time slice interrupts.

⚠️ Only super users may set the time.

On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, refer to _os9_setime().

The time is not validated in any way. If time is zero on systems with a battery-backed clock, the actual time is read and set from the real-time clock.

In order to get the system time/date from the real time clock or the battery-backed clock, the month field in the date parameter must be 0.

time is stored as the number of seconds since a specific date in Greenwich Mean Time. (Input)

   time is the desired current time in Greenwich Mean Time.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

gmtime() _os_link()
_os_getime() _os9_setime() F$STime
F_STIME

OS-9 for 68K Technical Manual
OS-9 Technical Manual
_os9_setime()

Set System Date/Time

Syntax

```c
#include <time.h>
#include <types.h>

error_code _os9_setime(
    u_int32 time,
    u_int32 date);
```

Description

_os9_setime() sets the current system date/time and starts the system real-time clock to produce time slice interrupts. Only super users may set the time.

Each value is one byte in length, except for the year which is 2 bytes long. Values must be in hexadecimal format.

The date and time are not checked for validity. On systems with a battery-backed clock, it is usually only necessary to supply the year to the _os9_setime() call. The actual date and time are read from the real-time clock.

In order to get the system time/date from the real-time clock or the battery-backed clock, the month field in the date parameter must be 0.

time
    is in the format 00hhmmss. (Input)
date
    is in the format yyyymmd. (Input)

Attributes

Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_link()
_os9_gettime()
F$STime

OS-9 for 68K Technical Manual
**Syntax**

```c
#include <process.h>
#include <types.h>
error_code _os_setpr(
    process_id proc_id,
    u_int32 priority);
```

**Description**

 `_os_setpr()` changes the process priority to the value specified by priority. A super user (group ID zero) may change any process’ priority. A non-super user can only change the priorities of processes with the same user ID.

- `proc_id` is the process ID. (Input)
- `priority` specifies the new priority. (Input)
- `65535` is the highest priority; `0` is the lowest.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`os_lib.l`

**See Also**

- `F$SPrior`  
  OS-9 for 68K Technical Manual
- `F_SPRIOR`  
  OS-9 Technical Manual
Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_setstat(
    path_id path,
    u_int32 code,
    void *pb);

Description
_os_setstat() is a wildcard call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent.

The exact operation of this call depends on the device driver and file manager associated with the path. The mnemonics for the status codes are located in the sg_codes.h header file. This manual describes the status codes currently defined by Microware and the functions that they perform. These calls have an _os_ss_ prefix.

/path/ is the path number. (Input)

/code/ is the SetStat code. (Input)

Only the lower 16 bits of code are used.

/pb/ is a pointer to the parameter block.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
Syntax

```c
#include <svctbl.h>
#include <types.h>
error_code _os_setsvc(
    u_int32 count,
    u_int32 state_flag,
    void *init_tbl,
    void *params);
```

Description

_os_setsvc() adds or replaces a system service in OS-9’s user and privileged system service request tables.

Use get_static() from cpu.h to acquire your current static storage pointer.

Valid service request codes range from 0 to 255.

The available service request codes are located in the funcs.h file.

count

is a count of the entries in the initialization table. (Input)

state_flag

specifies whether user or system state tables, or both, are updated. (Input)

- If state_flag is USER_STATE (1), only the user table is updated.
- If state_flag is SYSTEM_STATE (2), only the system table is updated.
- If state_flag is USER_STATE OR’ed with SYSTEM_STATE (3), both tables are updated.

init_tbl

is a pointer to the initialization table. (Input)

An example initialization table might look like this:

```c
error_code printmsg(), service();

svctbl syscalls[] =
{
    {F_PRINT, printmsg, 0, 1, 1},
    {F_SERVICE, service, 0, 1, 1}
};
```

params

may be a pointer to anything, but is intended to be a pointer to global static storage. (Input)

When a system call is used, the params data pointer is used as the global static storage pointer for the function.
Attributes

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library

os_lib.1
Syntax
#include <funcs.h>
#include <types.h>
extern void svctbl(void);    /* svctbl is in the code area */

_asm("svctbl:
dc.w %0
dc.w routine-*-2
dc.w %0+SysTrap
dc.w sys_routine-*-2
dc.w -1",
", F_DBGLOG);

Description
_os9_setsvc() adds or replaces function requests in OS-9 for 68K’s user and privileged system service request tables. To install a system-state-only system call, SysTrap is added to the function code number. Otherwise, both the system- and user-state dispatch tables are updated.
The OS-9 for 68K service request codes are located in /MWOS/OS9/SRC/DEFS.

_dc.w %0
is system-state and user-state request code.

_dc.w routine-*-2
is an offset to the routine.

_dc.w %0+SysTrap
updates system-state request code.

_dc.w sys_routine-*-2
is an offset to a routine.

Attributes
Operating System: OS-9 for 68K
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
Syntax

#include <sysglob.h>
#include <types.h>
error_code _os_setsys(
    u_int32 offset,
    u_int32 size,
    glob_buff sysvar);

Description

_os_setsys() changes a system global variable. These variables have a d_ prefix and are located in the system library sysglob.h. Consult the files in the DEFS directory for a description of the system global variables. Only super users may change system variables. Super users must be extremely careful when changing system global variables.

Refer to your operating system technical manual for the system global variables that may be changed.

offset
    is the offset to the system global variables. (Input)

size
    specifies the size of the target variable. (Input)

sysvar
    is the new value for the system global variable. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l
**Syntax**

```c
#include <process.h>
#include <types.h>
error_code _os_setuid(owner_id user_id);
```

**Description**

`_os_setuid()` changes the current user ID to `user_id`.

`user_id` is the desired group/user ID number. (Input)

The following restrictions apply to `_os_setuid()`:

- Users with group ID zero may change their IDs to anything without restriction.
- A primary module owned by a group zero user may change its ID to anything without restriction.
- Any primary module may change its user ID to match the module’s owner.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`os_lib.l`
Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_sgetstat(
    path_id spath,
    u_int32 code,
    void *pb);

Description
_os_sgetstat() makes a specific GetStat or SetStat call.

spath
    is the system path number. (Input)

code
    is the GetStat code. (Input)

pb
    is a pointer to the parameter block corresponding to the function being
    performed. (Input)
    If the function does not have a parameter block (for example,
    _os_ss_reset()), pb should be NULL.

Attributes
Operating System:    OS-9
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library
os_lib.l
**Syntax**

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_sgs_devnm(
    path_id path,
    char *namebuf);
```

**Description**

_*os_sgs_devnm()_ returns the name of the device associated with the specified system path number.

*path*  
is the system path number. (Input)

*namebuf*  
is a pointer to the location where _os_sgs_devnm() stores the device name. (Output)

**Attributes**

- Operating System: OS-9K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

os_lib.l
Set Signal Mast Value and Return on Specified Stack Image

**Syntax**

```c
#include <signal.h>
#include <types.h>
error_code _os_siglنجj(
    void *usp,
    u_int16 siglvl);
```

**Description**

_F_SIGLNGJ_ allows processes to perform _longjump_() operations from their signal intercept routines and unmask signals in one operation.

The operating system takes appropriate precautions to verify that the memory location pointed to by the _usp_ variable is accessible to the process and to ensure that the process does not attempt to make a state change.

The stack image pointed to by the _usp_ variable must have the format shown in **Figure 2-9**.

**Figure 2-9. _os_siglنجj() Required Stack Image**

The specific format of the processor context is defined by the _longstk_ structure definition found in the _reg<CPU Family>.h_ file for the associated processor. The format of the floating-point context varies depending on whether the target system has a hardware floating-point unit versus floating-point emulation software.

For floating-point hardware, the stack image is the same as that defined by the _fregs_ structure definition found in the associated _reg<CPU Family>.h_ header file.

For floating-point emulation, the floating-point context differs from the hardware implementation context in that it may contain additional context information specific to the _FPU_ module performing the emulation. The definition for the floating-point context as used by the _FPU_ module is the _fpu_context_ structure which is also defined in the associated _reg<CPU Family>.h_ header file for the target processor.
If a particular application needs to access the contents of the process context, it may use the size of these structures for indexing. Alternatively, the application can determine the size of the FPU context at runtime by accessing the kernel globals field, d_fpusize, which contains the size of the FPU context.

**usp**

is a pointer to the new process stack image.

**siglvl**

is the new signal level value.

This call is usually used by nested intercept routines to resume execution in the process at a different location from where the process was interrupted by the original signal. When this call is made, the operating system performs the following functions:

- Validates and copies the target process stack image from the memory buffer pointed to by the usp variable to the process’ system-state stack
- Sets the process’ signal mask to the value specified in the siglvl variable
- Returns to the process restoring the context copied from the user state process stack image

**Attributes**

- Operating System: OS-9
- State: User
- Threads: Safe
- Re-entrant: Yes

**Library**

- os_lib.l

**See Also**

- _os_sigmask()
- _os_send()
Syntax

```c
#include <signal.h>
#include <types.h>
error_code _os_sigmask(int32 mode);
```

The only function that _os_sigmask provides in system-state processes is to wake up that process. It does not interrupt code that is currently executing.

Description

_os_sigmask() enables signals to reach the calling process or disables signals from reaching the calling process. If a signal is sent to a process whose mask is disabled, the signal is queued until the process mask becomes enabled.

Signals are analogous to hardware interrupts and should be masked sparingly.

`mode`

is the process signal level. (Input)

<table>
<thead>
<tr>
<th>If Mode Is</th>
<th>The Mask Level Is</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cleared</td>
</tr>
<tr>
<td>1</td>
<td>Set or incremented</td>
</tr>
<tr>
<td>-1</td>
<td>Decremented</td>
</tr>
</tbody>
</table>

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`os_lib.l`

See Also

- _os_sleep()
- _os_wait()
Example

To ensure that an entire structure can be cleared without the reception of a signal one might use:

```c
_os_sigmask(1);
memset(sptr, 0, sizeof(*sptr));
_os_sigmask(-1);
```

The state of the signal mask is preserved using this method.

To avoid a race condition:

```c
_os_sigmask(1);
sig_received = FALSE;
sleep(0);           * sleep does an implied
                    * _os_sigmask(0);
```
_os_sigreset()
Reset Signal Intercept Context Stack

Syntax
#include <signal.h>
error_code _os_sigreset(void);

Description
_os_sigreset() resets the context stack for a signal intercept. Normally, the operating system keeps the state of the process on the system stack while a signal intercept routine executes. If signals are unmasked during the intercept routine and another signal occurs before the intercept routine finishes, another state is pushed on the system stack.

Similarly, if the intercept routine does a longjmp() and then unmask signals, more states are placed on the system stack by subsequent signals.

To prevent the system stack from overflowing, whenever a program uses longjmp() calls out of an intercept routine or unmask signals in an interrupt service routine with the intent of never using _os_rte() to return, it should use _os_sigreset():

if(setjmp(x) != 0) {
    _os_sigreset();
    _os_sigmask(-1);
}

If the code actually expects to return through the nested intercept routines with multiple _os_rte() calls, the states must be left where they are and _os_sigreset() should not be called.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_rte()
_os_sigrs()

Resize Current Process’ Signal Queue Block

Syntax

```c
#include <signal.h>
#include <types.h>
error_code _os_sigrs(u_int32 signals);
```

Description

_os_sigrs() allows a process to change the maximum number of signals that are queued on its behalf.

_os_sigrs() can be used to increase or decrease the number of signals queued. An error is returned (EOS_PARAM) if a request is made to reduce the number of queued signals while there are signals pending. The initial default for the system is specified in the system Init module.

```c
signals
```

is the new maximum number of signals. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
Syntax

```c
#include <signal.h>
#include <types.h>
error_code _os_sleep(
    u_int32 *ticks,
    signal_code *signal);
```

Description

_os_sleep() deactivates the calling process until the requested number of ticks have elapsed.

_os_sleep() cannot be used to time more accurately than ± 1 tick because it is not known when the _os_sleep() request was made during the current tick.

The system clock must be running to perform a timed sleep. The system clock is not required to perform an indefinite sleep or to give up a time slice.

ticks
The length of time to sleep in ticks/second is passed at the location pointed to by ticks. (Input)

- If ticks is zero, the process sleeps indefinitely.
- If ticks is one, the process gives up a time slice but does not necessarily sleep for one tick.
- _os_sleep() stores the number of ticks left to sleep when awakened prematurely at the location pointed to by ticks. If _os_sleep() completes without interruption, ticks contains the value 0.

time
may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second.

signal
is a pointer to the location where _os_sleep() stores the last signal the process received. (Output)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes
Library

os_lib.l

See Also

_os_send()
_os_wait()
_os9_sleep()

Put Calling Process to Sleep

Syntax

```c
#include <signal.h>
#include <types.h>
error_code _os9_sleep(u_int32 *ticks);
```

Description

_os9_sleep() deactivates the calling process until the requested number of ticks have elapsed.

_os9_sleep() cannot be used to time more accurately than ±1 tick because it is not known when the _os9_sleep() request was made during the current tick.

The system clock must be running to perform a timed sleep. The system clock is not required to perform an indefinite sleep or to give up a time slice.

_ticks

The requested number of ticks is passed at the location pointed to by ticks. (Input/Output)

- If ticks is zero, the process sleeps indefinitely.
- If ticks is one, the process gives up a time slice but does not necessarily sleep for one tick.
- _os9_sleep() stores the number of ticks left to sleep when awakened prematurely at the location pointed to by ticks. If _os9_sleep() completes without interruption, ticks contains the value 0.

time

may be specified in system clock ticks; or, if the high order bit is set, the low 31 bits are considered a time in 256ths of a second.

Attributes

Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_send()
_os_wait()
Syntax

#include <module.h>
#include <types.h>

error_code _os_slink(
    u_int32 sub_num,
    const char *mod_name,
    void **lib_exec,
    void **mem_ptr,
    mh_com **mod_head);

Description

_os_slink() attempts to link or load the named module. It returns a pointer to the
execution entry point and a pointer to the library’s static data area for subsequent
calls to the subroutine. The calling program must store and maintain the
subroutine's entry point and data pointer. The calling program must also set the
subroutine library’s data pointer and dispatch to the correct subroutine.

You can remove a subroutine by passing a null pointer for the name of the module
and specifying the subroutine number.

A process can link to a maximum of 16 subroutine libraries, numbered from 0 to
15.

sub_num
    is the subroutine number. (Input)

    Only the lower 16 bits of sub_num are used.

mod_name
    is a pointer to the name of the subroutine module. (Input)

lib_exec
    is a pointer to the location where _os_slink() stores the pointer to the
    subroutine entry point. (Output)

mem_ptr
    is a pointer to the location where _os_slink() stores the pointer to the
    subroutine static memory. (Output)

mod_head
    is a pointer to the location where _os_slink() stores the pointer to the
    module header. (Output)
Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_tlink()
Syntax

```c
#include <module.h>
#include <types.h>
error_code _os_slinkm(
    u_int32 sub_num,
    mh_com *mod_head,
    void **lib_exec,
    void **mem_ptr);
```

Description

_os_slinkm() is passed a pointer to the subroutine library module to install. It returns a pointer to the execution entry point and a pointer to the library’s static data area for subsequent calls to the subroutine. The calling program must store and maintain the subroutine’s entry point and data pointer. The calling program must also set the subroutine library’s data pointer and dispatch to the correct subroutine.

- **sub_num**
  - is the subroutine number. (Input)
  - Only the lower 16 bits of sub_num are used.

- **mod_head**
  - is a pointer to the module header. (Input)

- **lib_exec**
  - is a pointer to the location where _os_slinkm() stores the pointer to the subroutine library entry point. (Output)

- **mem_ptr**
  - is a pointer to the location where _os_slinkm() stores the pointer to the subroutine library static memory. (Output)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

_os_slink() _os_tlink()
Syntax
#include <memory.h>
#include <types.h>
error_code _os_srqmem(
    u_int32 *size,
    void **mem_ptr,
    u_int32 color);

Description
_os_srqmem() allocates a block of a specific memory type.

- The byte count of allocated memory and the pointer to the allocated block
  must be saved if the memory is ever to be returned to the system.

size
- The byte count of the requested memory is passed at the location pointed to
  by size. (Input/Output)

  _os_srqmem() also stores the actual number of bytes allocated at the location
  pointed to by size. If size is 0xFFFFFFFF, the largest block of free memory of
  the specified type is allocated to the calling process.

mem_ptr
  is a pointer to the location where _os_srqmem() stores the pointer to the
  allocated memory block. (Output)

color
- specifies the memory type. (Input)

  Only the lower 16 bits of color are used.
  - If color is non-zero, the search is restricted to memory areas of that color.
    The area with the highest priority is searched first.
  - If color is zero, the search is based only on priority.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
See Also

_os9_srqmem()
_os_srtmem()
_os_mem()
F$SRqCMem
F_SRQMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax
#include <memory.h>
#include <types.h>
error_code _os9_srqmem(
    u_int32 *size,
    void **mem_ptr);

Description
_os9_srqmem() allocates a memory block from the top of available RAM.

The byte count of allocated memory and the pointer to the allocated block
must be saved if the memory is ever to be returned to the system.

size
The size of the memory block is passed at the location pointed to by size.
(Input/Output)
_os9_srqmem() also stores the actual number of bytes allocated at the location
pointed to by size. If size is 0xFFFFFFFF, the largest block of free memory is
allocated to the calling process.

mem_ptr
is a pointer to the location where _os9_srqmem() stores the pointer to the
allocated memory block. (Output)

Attributes
Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_srtmem()
_os_mem()
_os_mem()
_os_srqmem()
F$SRqMem

OS-9 for 68K Technical Manual
_os_srtmem()
Return System Memory

Syntax
#include <memory.h>
#include <types.h>
error_code _os_srtmem(
    u_int32 size,
    void *mem_ptr);

Description
_os_srtmem() deallocates memory when it is no longer needed.

size
    specifies the byte count of the returned memory. (Input)

mem_ptr
    is a pointer to the memory block to return. (Input)

Attributes
Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library
os_lib.l

See Also
_os_srqmem()
_os_mem()
Syntax

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_attr(
    path_id path,
    u_int32 attr);
```

Description

_os_ss_attr() changes a file’s attributes to the new value, if possible. You cannot set the directory bit of a non-directory file or clear the directory bit of a non-empty directory.

**path**
- specifies the path number of the file. (Input)

**attr**
- specifies the new values for the file attributes. (Input)

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

Library

os_lib.l
_os_ss_break()

Send Break Conditional to Serial Line

Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_break(path_id path);

Description
_os_ss_break() sends a break conditional to the serial line.

path
    specifies the path number. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
I_SETSTAT, SS_BREAK  OS-9 Technical Manual
#os_ss_cache()

**Enable/Disable RBF Caching**

## Syntax

```c
#include <rbf.h>
#include <types.h>

error_code _os_ss_cache(
    path_id path,
    u_int32 enblflag,
    u_int32 drvcsze);
```

## Description

_os_ss_cache() enables and disables RBF caching on an RBF device.

- **path** specifies the path number. (Input)
- **enblflag** is the cache enable/disable flag. (Input)
  - If `enblflag` is zero, caching is disabled.
  - If `enblflag` is non-zero, caching is enabled.
- **drvcsze** is the memory size for the cache. (Input)

## Attributes

- Operating System: OS-9
- State: System
- Threads: Safe
- Re-entrant: Yes

## Library

- os_lib.1
_os9_ss_close()

Notify Driver that Path Has Been Closed

Syntax

#include <sg_codes.h>
#include <types.h>
error_code _os9_ss_close(path_id path);

Description

_os9_ss_close() notifies a driver that a path has been closed.

path
    specifies the path number that was closed.  (Input)

Attributes

Operating System:    OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library

os_lib.l
_os_ss_dcoff()
Set Signal to Be Sent when DCDL Goes False

Syntax
#include <scf.h>
#include <types.h>
error_code _os_ss_dcoff(
    path_id path,
    signal_code signal);

Description
_os_ss_dcoff() sets the signal code to be sent when the Data Carrier Detect line becomes false.

path specifies the path number. (Input)

signal is the signal code to send. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_ss_dcon()
Set Signal to Be Sent when DCDL Goes True

Syntax

```c
#include <scf.h>
#include <types.h>
error_code _os_ss_dcon(
    path_id path,
    signal_code signal);
```

Description

_os_ss_dcon() sets the signal code to be sent when the Data Carrier Detect line becomes true.

**path**
- specifies the path number. (Input)

**signal**
- is the signal code to send. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

_os_ss_dcoff()
Set Device Path Options

**Syntax**

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_dopt(
    path_id path,
    u_int32 size,
    void *dopts);
```

**Description**

_os_ss_dopt() sets the initial (default) device path options (dopts). These options initialize new paths to the device.

- **path**
  - specifies the path number. (Input)

- **size**
  - specifies the size of the options area. (Input)

- **dopts**
  - is a pointer to the default options for the device. (Input)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

os_lib.l
Syntax
#include <scf.h>
#include <types.h>
error_code _os_ss_dsrts(path_id path);

Description
_os_ss_dsrts() disables the RTS line on serial devices.

path
    specifies the path number. (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:             User and System
Threads:           Safe
Re-entrant:        Yes

Library
os_lib.l

See Also
I$SetStt  OS-9 for 68K Technical Manual
I_SETSTAT, SS_DSRTS  OS-9 Technical Manual
Syntax

#include <scf.h>
#include <types.h>
error_code _os_ss_enrts(path_id path);

Description

_os_ss_enrts() enables the RTS line on serial devices.

path
  specifies the path number. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State:           User and System
Threads:         Safe
Re-entrant:      Yes

Library

os_lib.1

See Also

I$SetStt              OS-9 for 68K Technical Manual
I_SETSTAT, SS_ENRTS   OS-9 Technical Manual
 Syntax
#include <sbf.h>
#include <types.h>
error_code _os_ss_erase(
    path_id path,
    u_int32 blocks);

Description
_os_ss_erase() erases a portion of the tape.

path
    specifies the tape drive’s path number. (Input)

blocks
    specifies the number of blocks to erase. (Input)
    • If blocks is 0xffffffff, SBF erases until the end-of-tape is reached.
    • If blocks is positive, SBF erases the amount of tape equivalent to that number of blocks.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
_os_ss_fd()
Write File Descriptor Sector

Syntax
#include <rbf.h>
#include <types.h>
error_code _os_ss_fd(
    path_id path,
    fd_stats *fdinfo);

Description
_os_ss_fd() changes the file descriptor sector data. The path must be open for
write. To change the file descriptor sector data, you must first read the sector, make
the changes, and then write the sector.

path
    specifies the path number. (Input)

fdinfo
    is a pointer to the file descriptor’s buffer. (Input)

Attributes
Operating System:  OS-9 and OS-9 for 68K
State:            User and System
Threads:          Safe
Re-entrant:       Yes

Library
os_lib.l
Syntax
#include <scf.h>
#include <types.h>
error_code _os_ss_fillbuff(
    path_id path,
    char *buffer,
    u_int32 size);

Description
_os_ss_fillbuff() fills the input path buffer with the data in buffer.

path
    specifies the path number. (Input)

buffer
    is a pointer to the location of your buffer. (Input)

size
    specifies the size of your buffer. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
_os_ss_flushmap()
Flush Cached Bit Map Information for RBF Device

Syntax
#include <rbf.h>
#include <types.h>
error_code _os_ss_flushmap(
    path_id path);

Description
_os_ss_flushmap() flushes the cached bit map information for an RBF device. This
is normally only performed after the bit map on the disk is changed by a utility such
as format.

path
    is the path number for any file on the device or the entire device (for example,
a path open to /h0@). (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
Syntax

```c
#include <rbf.h>
#include <types.h>
error_code _os_ss_hdlink(
    path_id path,
    char **link_path);
```

Description

_os_ss_hdlink() creates a new directory entry. This directory entry points to the file descriptor block of an open file. _os_ss_hdlink() updates the pathlist pointer.

- **path:** is the path number of an existing file. (Input)
- **link_path:** The new name for the directory entry is passed at the location pointed to by link_path. (Input/Output)

_os_ss_hdlink() also stores the updated pathlist pointer at the location pointed to by link_path.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
Syntax

```
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_link_adj()
    (path_id path,
     u_in32 fd_addr,
     int32 link_adj)
```

Description

This `I_SETSTAT` call sets the link count of a file. The link count of a file indicates the number of hard links in the file system that point to a particular file. This call can be used to correct the link count of files under corrupt file system conditions. File link counts can only be adjusted by super-user processes.

```
path         Input
fd_addr      Input
link_adj     Input
```

Attributes

- Operating System: OS-9
- State: User; System; I/O; IRQ
- Threads: Safe
- Re-entrant: Yes

Library

oslib.l

See Also

- _os_ss_hdlink()
- _os_setstat()
Syntax

```c
#include <rbf.h>
#include <types.h>
error_code _os_ss_lock(
    path_id path,
    u_int32 size);
```

Description

_os_ss_lock() locks out a section of the file from the current file pointer position up to the specified number of bytes.

path

is the file's path number.

size

is the size of the section to lockout.

- If size is zero, all locks are removed (record lock, EOF lock, and file lock).
- If size is $ffffffff$, the entire file is locked out regardless of the file pointer's location. This is a special type of file lock that remains in effect until released by an _os_ss_lock() with size set to zero, a read or write of zero bytes, or the file is closed.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l
_os_ss_luopt()  
Write Logical Unit Options

Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_luopt(
    path_id path,
    u_int32 size,
    void *luopts);

Description
_os_ss_luopt() writes the logical unit options for a path to a buffer.

path
    is the path number. (Input)

size
    specifies the buffer size of the logical unit options area. (Input)

luopts
    is a pointer to the logical unit options. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
_os9_ss_open()

Notify Driver that Path Has Been Opened

Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os9_ss_open(path_id path);
```

Description

_os9_ss_open() is an internal call for drivers.

path

specifies the path number.

Attributes

- Operating System: OS-9
- State: System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
_os_ss_popt()  
Write Option Section of Path Descriptor

Syntax

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_popt(
    path_id path,
    u_int32 size,
    void *user_opts);
```

Description

_os_ss_popt() writes the option section of the path descriptor from the options buffer pointed to by user_opts.

path

is the path number. (Input)

size

specifies the buffer size. (Input)

user_opts

is a pointer to the options buffer. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_gs_popt()  
I$SetStt  
I_SETSTAT

OS-9 for 68K Technical Manual

OS-9 Technical Manual
Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_relea(path_id path);

Description
_os_ss_relea() releases the device from any _os_ss_sendsig(), _os_ss_dcon(), or _os_ss_dcoff() request made by the calling process.

path is the path number of the device to release. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1

See Also
_os_ss_sendsig()
_os_ss_dcoff()
_os_ss_dcon()
Syntax

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_rename(
    path_id path,
    const char *newname);
```

Description

_os_ss_rename() changes the file name in the file's associated directory entry to newname.

- **path**
  - is the file's path number. (Input)

- **newname**
  - is a pointer to the file's new name. (Input)

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
_os_ss_reset()

Restore Head to Track Zero

Syntax
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_reset(
    path_id path);

Description
For RBF, _os_ss_reset() directs the disk head to track zero. It is used for formatting and error recovery. For SBF, _os_ss_reset() rewinds the tape.

path
    is the path number of the device. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
Syntax
#include <sbf.h>
#include <types.h>
error_code _os_ss_reten(path_id path);

Description
_os_ss_reten() performs a retention pass on the tape drive.

path is the path number of the tape drive. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
_os_ss_rfm()

Skip Tape Marks

Syntax

```c
#include <sbf.h>
#include <types.h>
error_code _os_ss_rfm(
    path_id path,
    int32 marks);
```

Description

_os_ss_rfm() skips the number of tape marks specified in marks.

path

is the tape drive’s path number. (Input)

marks

specifies the number of tape marks to skip. (Input)

If marks is negative, the tape is rewound the specified number of marks.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l
_os_ss_sendsig()  
Send Signal on Data Ready

Syntax

#include <sg_codes.h>
#include <types.h>
error_code _os_ss_sendsig(
    path_id path,
    signal_code signal);

Description

_os_ss_sendsig() sets up a signal to send to a process when an interactive device or pipe has data ready. _os_ss_sendsig() must be reset each time the signal is sent.

path
    specifies the path number of the process receiving the signal. (Input)

signal
    specifies the signal to send. (Input)

Attributes

Operating System:    OS-9 and OS-9 for 68K
State:           User and System
Threads:          Safe
Re-entrant:        Yes

Library

os_lib.l

See Also

_os_ss_relea()
Syntax

```c
#include <sg_codes.h>
#include <types.h>
error_code _os_ss_size(
    path_id path,
    u_int32 size);
```

Description

_os_ss_size() sets the size of the file associated with `path` to the specified size.

- **path** is the file’s path number. (Input)
- **size** is the size of the file in bytes. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`os_lib.1`
_os_ss_skip()

Skip Blocks

Syntax

```c
#include <sbf.h>
#include <types.h>
error_code _os_ss_skip(
    path_id path,
    int32 blocks);
```

Description

_os_ss_skip() skips the specified number of blocks.

- `path` is the path number of the tape drive. (Input)
- `blocks` specifies the number of blocks to skip. (Input)
  - If `blocks` is negative, the tape is rewound the specified number of blocks.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

- os_lib.l
_os_ss_skipend()

Skip to End of Tape

Syntax
#include <sbf.h>
#include <types.h
error_code _os_ss_skipend(path_id path);

Description
_os_ss_skipend() skips the tape to the end of data. This enables you to append data to tapes on cartridge-type tape drives.

path is the path number of the tape drive. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.1
_os_ss_ticks()

Wait Specified Number of Ticks for Record Release

Syntax

```c
#include <rbf.h>
#include <types.h>
error_code _os_ss_ticks(
    path_id path,
    u_int32 delay);
```

Description

_os_ss_ticks() may be used to cause an error (EOS_LOCK) to return to the user program if a file lock conflict still exists after the specified number of ticks have elapsed.

- **path**
  - is the path number. (Input)

- **delay**
  - specifies the delay interval. (Input)
    - A delay of zero (RBF's default) causes sleep until the record is released.
    - A delay of one indicates that if the record is not released immediately, an error is returned. If the high order bit is set, the lower 31 bits are converted from 256ths of a second to ticks before sleeping.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
Write Tape Marks

Syntax

```c
#include <sbf.h>
#include <types.h>
error_code _os_ss_wfm(
    path_id path,
    u_int32 marks);
```

Description

_os_ss_wfm() writes the specified number of tape marks.

- **path** specifies the tape drive’s path number. (Input)
- **marks** specifies the number of tape marks to write. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
Syntax

```
#include <rbf.h>
#include <types.h>
error_code _os_ss_wtrack(
    path_id path,
    const void *trkbuf,
    const void *ilvtbl,
    u_int32 track,
    u_int32 head,
    u_int32 interleave);
```

Description

_os_ss_wtrack() causes a format track operation (used with most floppy disks) to occur. For hard or floppy disks with a “format entire disk” command, this formats the entire media only when the track number and side number are both zero.

**path**

is the path number. (Input)

**trkbuf**

is a pointer to the track buffer. (Input)

**ilvtbl**

is a pointer to the interleave table. (Input)

The interleave table contains byte entries of LBNs ordered to match the requested interleave offset.

**track**

is the track number. (Input)

**head**

is the side number. (Input)

**interleave**

is the interleave value. (Input)

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
Library

os_lib.l

See Also

I$SetStt OS-9 for 68K Technical Manual
I_SETSTAT OS-9 Technical Manual
set error trap handler

Syntax

```c
#include <settrap.h>
#include <types.h>
#include <regs.h>
error_code _os_strap(
    u_int32 *excpt_stack,
    strap *init_tbl);
```

Description

_os_strap() enables user programs to catch “process local” program exceptions, such as illegal instruction or divide-by-zero. If a user routine is not provided and one of these exceptions occurs, the program is aborted. An initialization table for a PowerPC platform might appear as follows:

```c
class errtab[] = {
    { STRAP_DATA, errtrap},
    { STRAP_INSTR, errtrap},
    { ~0, NULL}
};
```

The previous example is for the PowerPC platform. The STRAP_DATA and STRAP_INSTR macros are defined only for PowerPC, not necessarily for other platforms. When a user’s exception routine is executed, it is called as if it had the following prototype:

```c
void errtrap(
    u_int32 vector_err, /* vector error number for exception */
    u_int32 badpc,      /* PC where exception occurred */
    u_int32 badaddr,    /* address that caused the exception */
    ...);               /* original register contents */
```

The variable arguments contain the previous values of registers that were modified between the exception and the call to the exception handler. Generally, the variable arguments contain the previous stack pointer and the previous values of the first three parameter registers.

The following sections provide details about the variable arguments for the various processors.

- **ARMv4 and ARMv4BE**

```c
va_arg(vp, u_int32);    /* exception stack pointer */
va_arg(vp, u_int32);    /* exception r7 */
va_arg(vp, u_int32);    /* exception r8 */
```
_os_strap()

va_arg(vp, u_int32);    /* exception r9 */

- **MIPS3000 and MIPS32**
  va_arg(vp, u_int32);    /* exception stack pointer */
  va_arg(vp, u_int32);    /* exception a0 */
  va_arg(vp, u_int32);    /* exception a1 */
  va_arg(vp, u_int32);    /* exception a2 */

- **MIPS64**
  va_arg(vp, u_int64);    /* exception stack pointer */
  va_arg(vp, u_int64);    /* exception a0 */
  va_arg(vp, u_int64);    /* exception a1 */
  va_arg(vp, u_int64);    /* exception a2 */

- **PowerPC**
  va_arg(vp, u_int32);    /* exception stack pointer */
  va_arg(vp, u_int32);    /* exception r3 */
  va_arg(vp, u_int32);    /* exception r4 */
  va_arg(vp, u_int32);    /* exception r5 */

- **SH-3 and SH-4**
  va_arg(vp, u_int32);    /* exception stack pointer */
  va_arg(vp, u_int32);    /* exception r4 */
  va_arg(vp, u_int32);    /* exception r5 */
  va_arg(vp, u_int32);    /* exception r6 */

- **SH-5m**
  va_arg(vp, u_int64);    /* exception stack pointer */
  va_arg(vp, u_int64);    /* exception r2 */
  va_arg(vp, u_int64);    /* exception r3 */
  va_arg(vp, u_int64);    /* exception r4 */

- **x86/Pentium**
  va_arg(vp, u_int32);    /* exception stack pointer */
  va_arg(vp, u_int32);    /* exception %eax */

**except_stack**

is a pointer to the stack to use if an exception occurs. (Input)

If **except_stack** is zero, _os_strap() uses the stack at the time of the exception.

**init_tbl**

is a pointer to the service request initialization table. (Input)
**PowerPC Users**: When catching the STRAP_FPU exception on PowerPC processors with hardware floating-point an additional variable argument is passed. The first variable argument is a copy of the PowerPC fpscr register at the time of the trapped floating-point exception.

You can disable an error exception handler by calling _os_strap() with an initialization table that specifies zero as the address of the routine(s) to remove.

- On OS-9 for 68K, the binding for this call performs conversions which allow compatibility with OS-9. If speed is a consideration, refer to _os9_strap().
- Beware of exceptions in exception handling routines. They are usually not re-entrant.

**Attributes**
- Operating System: OS-9 and OS-9 for 68K
- State: User
- Threads: Safe
- Re-entrant: Yes

**Library**
- os_lib.l

**See Also**
- F$Strap
- OS-9 for 68K Technical Manual
- F Strap
- OS-9 Technical Manual
Syntax

```c
#include <settrap.h>
#include <regs.h>
#include <types.h>
error_code _os9_strap(
    u_int32 *excpt_stack,
    os9strap *init_tbl);
```

Description

_os9_strap() enables user programs to catch “process local” program exceptions such as illegal instructions and divide by zeroes.

*excpt_stack* is a pointer to the stack to use if an exception occurs. (Input)

If *excpt_stack* is zero, _os9_strap() uses the current stack.

*init_tbl* is a pointer to the service request initialization table. (Input)

The first field of the *init_tbl* should be the vector offset of the exception vector on which to install the handler. The second field is the offset from the next entry in *init_tbl* to the exception handling function for this vector. *init_tbl* may look like:

```c
void ovferror();
void chkerror();
extern void os9_init_tbl();

os9strap *init_tbl = (os9strap *)os9_init_tbl;
```

```asm
_asm("os9strap_init_tbl: dc.w %0,
     ovferror-*4", T_TRAPV << 2);
_asm(" dc.w %0, chkerror-*4", T_CHK << 2);
_asm(" dc.w -1, -1");
```

To access the contents of the initialization table, os9strap_init_tbl in this example, you must define os9_init_tbl as if it were a function. For example, os9strap *os9_init_tbl would not generate PC-relative references, while extern void os9_init_tbl() forces the compiler to generate references into the code area. If a user routine is not provided and one of these exceptions occur, the program is aborted. You can disable an error exception handler by calling _os9_strap() with an initialization table that specifies zero as the offset to the routine(s) to remove.

⚠️ Beware of exceptions in exception handling routines. They are usually not re-entrant.
Attributes
Operating System: OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
F$STrap
OS-9 for 68K Technical Manual
_os_suspend()
Suspend Process

Syntax
#include <process.h>
#include <types.h>
error_code _os_suspend(process_id proc_id);

Description
_os_suspend() temporarily suspends a process. A super user (group ID zero) may
suspend any process. A non-super user gets an E_PERMIT message returned.

_os_suspend() is currently not supported on OS-9 for 68K.

proc_id
identifies the target process. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_aproc()
_os_sysdbg()
Call System Debugger

Syntax

```c
#include <process.h>
#include <types.h>
error_code _os_sysdbg(
    void *param1,
    void *param2);
```

Description

_os_sysdbg() calls the system level debugger, if one exists.

When the system level debugger is active, it runs in system state and effectively stops timesharing. When used in user state, _os_sysdbg() can only be called by users in group zero. When used in system state, anyone may use _os_sysdbg(). Never use this call when other users are on the system.

You must enable the system debugger before installing breakpoints or attempting to trace instructions. If no system debugger is available, the system is reset. The system debugger handles some of the exception vectors directly. This makes it impossible to use the user debugger when the system debugger is enabled.

param1 and param2
are parameters passed to the debugger. These are currently not used. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.1
Syntax

```c
#include <regs.h>
#include <types.h>

error_code _os_sysid(
    u_int32 *oem,
    u_int32 *serial,
    u_int32 *mpu_type,
    u_int32 *os_type,
    u_int32 *fpu_type,
    int32 *time_zone,
    u_int32 *kernel_ver,
    u_int32 *resv2,
    char *sys_ident,
    char *copyright,
    char *author);
```

Description

_os_sysid() returns the following information about the system:

**Table 2-17. _os_sysid() Return Information**

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Information Stored by _os_sysid()</th>
</tr>
</thead>
<tbody>
<tr>
<td>oem</td>
<td>OEM identification number</td>
</tr>
<tr>
<td>serial</td>
<td>Copy serial number</td>
</tr>
<tr>
<td>mpu_type</td>
<td>Processor identifier (such as 68020/68030/68040/80386)</td>
</tr>
<tr>
<td>os_type</td>
<td>Kernel (OS) MPU configuration</td>
</tr>
<tr>
<td>fpu_type</td>
<td>Floating-point processor identifier (such as 68881 and 80387)</td>
</tr>
<tr>
<td>time_zone</td>
<td>System time zone in minutes offset from Greenwich Mean Time (GMT)</td>
</tr>
<tr>
<td>kernel_ver</td>
<td>The kernel's version number for OS-9 for 68K</td>
</tr>
<tr>
<td>resv2</td>
<td>A reserved pointer</td>
</tr>
</tbody>
</table>
Table 2-17. _os_sysid() Return Information  (Continued)

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Information Stored by _os_sysid()</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_ident</td>
<td>A pointer to an application buffer to copy the system identification message into. The buffer must be at least 80 bytes in length.</td>
</tr>
<tr>
<td>copyright</td>
<td>A pointer to an application buffer to copy the copyright message into. The buffer must be at least 80 bytes in length.</td>
</tr>
</tbody>
</table>
| author      | For OS-9: A pointer to an application buffer to copy the author message for OS-9. The buffer must be at least 80 bytes in length.  
For OS-9 for 68K: This parameter should be NULL. |

⚠️ On OS-9 for 68K, time_zone is always returned as -1.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User and System  
Threads: Safe  
Re-entrant: Yes

**Library**

os_lib.l
Syntax
#include <threads.h>
error_code _os_thexit(error_code status);

Description
_os_thexit() causes the calling thread to exit. If the calling program is not multi-threaded, the EOS_PERMIT error is returned.

If successful, _os_thexit does not return to the caller.

Threads created via pthread_create() should not use this call. Doing so will result in instability and loss of resources for the process.

status is the value to be returned to any thread waiting for this thread via an _os_waitid() call. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l, mt_os_lib.l

See Also
_os_thfork()
_os_thfork()
Fork a Thread

Syntax

```c
#include <threads.h>
error_code _os_thfork(
    thread_t       *thread,
    thread_attr_t  *attr,
    void           **stack_top,
    u_int32        *stack_size,
    void           *func,
    void           *arg,
    void           *data);
```

Description

_os_thfork() forks a new thread of control in the current process.

Threads created with _os_thfork() or F_THFORK are not permitted to use C library calls that have threading issues. Create threads with pthread_create().

**thread**

is a pointer to a thread_t object which is used to return the newly created thread's ID. (Input)

**attr**

is a pointer to a thread attribute structure. (Input)

The thread attribute structure is declared in threads.h.

**stack_top**

is a pointer to the top of user stack to be used by this thread—stack_top will be the initial stack pointer for the thread. (Input)

**stack_size**

If the value pointed to by stack_size is non-zero, the thread's user stack will be allocated by the system and a pointer to the allocated memory will be returned in the location pointed to by stack_top. The size of the actual stack allocated will be returned at stack_size.

If stack_size points to zero, the value pointed to by stack_top will be used as the thread's stack. (Input/Output)

**func**

is the address of the initial function to be executed by the forked thread. (Input)

**arg**

is passed as the only argument to the function specified by func. (Input)

**data**

is the address of thread's thread-specific data. (Input)

The pointer to void value pointed to by data is maintained on behalf of the thread. (Input)
Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l, mt_os_lib.l

See Also
=os_thexit()
=os_thread()
Syntax

```c
#include <funcs.h>
#include <threads.h>
error_code _os_thread(
    u_int32   code,
    thread_t  thread_id,
    void      *arg);
```

Description

_os_thread() sets thread parameters for the thread specified by thread_id (input). Threads created via pthread_create() should not use this call. Doing so will result in instability and loss of resources for the process.

code

is the subfunction to perform. (Input)

The legal values for code are TH_ORPHAN to orphan the thread and TH_TSDATA to set the thread specific data for the thread. Attempting to orphan an already orphaned thread will result in the error EOS_THRD_INVLD.

arg

If code is TH_TSDATA, arg is the thread specific data pointer. (Input)

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l, mt_os_lib.l

See Also

_os_thexit()
_os_thfork()
_os_tlink()

Install User/System State Trap Handling Module

Syntax

```c
#include <module.h>
#include <types.h>
error_code _os_tlink(
    u_int32 trap_num,
    const char *mod_name,
    void **lib_exec,
    mh_trap **mod_head,
    void *params,
    u_int32  mem_size);
```

Description

_os_tlink() attempts to link or load the specified module. If a trap module already exists for the specified trap number, an error is returned. If static storage is required for the trap handler, it is allocated and initialized.

OS-9 only allows trap handlers to run in system state. OS-9 for 68K allows trap handlers to run in either system state or user state.

- **trap_num** specifies the user trap number. (Input)
  On OS-9 for 68K valid trap numbers are in the range 1 through 15. On OS-9 valid trap numbers vary from processor to processor, but are generally in the range 0 through 15. See the include file in /MWOS/OS9000/proc/DEFS for the macros TLINK_MIN and TLINK_MAX.

- **mod_name** is a pointer to the name of the trap module. (Input)
  If mod_name is zero or points to a null string, the trap handler is unlinked.

- **lib_exec** is a pointer to the location where _os_tlink() stores the pointer to the trap execution entry point. (Output)

- **mod_head** is a pointer to the location where _os_tlink() stores the pointer to the trap module. (Output)

- **params** is a pointer to any additional parameters. (Input)
  Currently, this is not implemented.

- **mem_size** specifies the additional memory size. (Input)
_os_tlink()

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
Install User Trap Handling Module by Module Pointer

** Syntax **

```c
#include <module.h>
#include <types.h>
error_code _os_tlinkm(
    u_int32 trap_num,
    mh_com *mod_head,
    void **lib_exec,
    void *params,
    u_int32 mem_size);
```

** Description **

_-os_tlinkm()_ is passed a pointer to the module to install. If a trap module already exists for the specified trap number, an error is returned. If static storage is required for the trap handler, it is allocated and initialized.

- **trap_num** specifies the user trap number (1 through 15). (Input)
- **mod_head** is a pointer to the module header. (Input)
- **lib_exec** is a pointer to the location where _os_tlinkm()_ stores the pointer to the trap execution entry point. (Output)
- **params** is a pointer to the additional parameters, if necessary. (Input)
- **mem_size** specifies the additional memory size. (Input)

** Attributes **

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

** Library **

os_lib.l
Translate User Path to System Path

Syntax

```c
#include <io.h>
#include <types.h>

error_code _os_tranpn(
    process_id proc_id,
    path_id user_path,
    path_id *sys_path);
```

Description

_os_tranpn() translates a user path number to a system path number. System-state processes use this call to access the user paths (standard I/O paths).

**proc_id**
- specifies the process ID. (Input)

**user_path**
- specifies the user path to translate. (Input)

**sys_path**
- is a pointer to the location where _os_tranpn() stores the mapped system path. (Output)

Attributes

- Operating System: OS-9
- State: System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
Syntax

```c
#include <virtual.h>
#include <types.h>
error_code _os_transadd(
    u_int32 *size,
    u_int32 mode,
    void **blk_addr,
    void *reserved);
```

Description

_os_transadd() translates an address to or from its external bus address.

**size**

The size of the block to translate is passed at the location pointed to by `size`. (Input/Output)

_os_transadd() also stores the size of the block translated at the location pointed to by `size`.

**mode**

specifies the type of translation. (Input)

- If `mode` is zero, the local CPU address is translated to the external bus address.
- If `mode` is one, the external bus address is translated to the local CPU address.

**blk_addr**

The address of the block to translate is passed at the location pointed to by `blk_addr`. (Input/Output)

_os_transadd() also stores the block that was translated at the location pointed to by `blk_addr`. _os_transadd() is used when the external bus address must be passed to hardware devices, such as DMA-type controllers.

If the specified source block is nonlinear with respect to its destination mapping, _os_transadd() returns the maximum number of bytes accessible at the translated address.

**reserved**

is a pointer reserved for future use. (Input)
Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
Translate Memory Address

Syntax
#include <memory.h>
#include <types.h>
error_code _os9_translate(
    u_int32 *size,
    u_int32 mode,
    void **blk_addr);

Description
_os9_translate() translates an address to or from its external bus address. If the specified source block is nonlinear with respect to its destination mapping, _os9_translate() returns the maximum number of bytes accessible at the translated address.

_os9_translate() is only available if colored memory is enabled.

size
The size of the block to translate is passed at the location pointed to by size. (Input/Output)
_os9_translate() also stores the size of the block translated at the location pointed to by size.

mode
specifies the type of translation. (Input)
- If mode is zero, the local CPU address is translated to the external bus address.
- If mode is one, the external bus address is translated to the local CPU address.

blk_addr
The address of the block to translate is passed at the location pointed to by blk_addr. (Output)
_os9_translate() also stores the block that was translated at the location pointed to by blk_addr.
_os9_translate() is used when the external bus address must be passed to hardware devices, such as DMA-type controllers.

Attributes
Operating System: OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l
Syntax

```c
#include <process.h>
#include <types.h>
error_code _os_uacct(
    u_int32 acct_code,
    pr_desc *proc_desc);
```

Description

_os_uacct() provides a means for users to set up an accounting system.
_os_uacct() is called by the kernel whenever it forks or exits a process. This provides a mechanism for users to keep track of system operators.

To install a handler for this service request, the _os_setsvc() system call must be used to add the user's accounting routine to the system's service request dispatch table. This is usually done in an OS9P2 module.

You may perform your own system accounting by calling _os_uacct() with a user defined acct_code identifying the operation to perform. For example, when the kernel forks a process it identifies the operation by passing the F_FORK code to the accounting routine.

acct_code

- is the operation identifier. reserved is a pointer reserved for future use. (Input)
  
  This is usually a system call function code. Only the lower 16 bits of acct_code are used.

proc_desc

- is a pointer to the current process descriptor. reserved is a pointer reserved for future use. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l
Unlink Module by Address

Syntax

```c
#include <module.h>
#include <types.h>
error_code _os_unlink(mh_com *mod_head);
```

Description

_os_unlink() informs the system that the calling process no longer needs the module and decrements the module’s link count.

⚠️ Some modules, such as device drivers in use and all modules included in the bootfile, cannot be unlinked.

**mod_head**

is a pointer to the module header.

**reserved**

is a pointer reserved for future use. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

- _os_unload()
- F$UnLink
- F_UNLINK
  
  OS-9 for 68K Technical Manual

  OS-9 Technical Manual
Syntax
#include <module.h>
#include <types.h>

error_code _os_unload(
    const char *mod_name,
    u_int32 type_lang);

Description
_os_unload() locates the module specified by mod_name in the module directory and decrements its link count.

mod_name
    is a pointer to the module name. (Input)

type_lang
    specifies the module’s type and language. (Input)
    Only the lower 16 bits of type_lang are used.

Library
os_lib.l

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

See Also
_os_unlink()    F$UnLoad
F_UNLOAD         OS-9 for 68K Technical Manual
OS-9 Technical Manual
Create New Usemaphore

Syntax

```c
#include <semaphore.h>
#include <types.h>
error_code _os_usema_create(
    usema_id *us_id,
    const char *name,
    u_int32 value,
    u_int32perm,
    u_int32 color);
```

Description

_os_usema_create() allows you to create usemaphores dynamically as needed. When a usemaphore is created, an initial value is specified, as well as the permissions and memory color from which to allocate the usemaphore. Subsequent _os_usema calls use the returned ID number to refer to the created usemaphore. A usemaphore and an event (see _os_ev_create) may not have the same name.

us_id
is a pointer to the location where _os_usema_create() stores the usemaphore identifier. (Output)

name
is a pointer to the usemaphore name string. (Input)

value
specifies the initial usemaphore value. Only 0 or 1 may be passed. (Input)

perm
specifies the access permissions. (Input)
Only the lower 16 bits of perm are used.

color
specifies the memory type for the usemaphore structure. (Input)
Only the lower 16 bits of color are used.

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes
Library

os_lib.l

See Also

_os_usema_unlink
_os_usema_delete

F_EVENT, EV_CREAT | F_USEMA OS-9 Technical Manual
_os_usema_delete()

Delete an Existing Usemaphore

Syntax
#include <semaphore.h>
#include <types.h>
error_code _os_usema_delete(const char *name);

Description
_os_usema_delete() removes a usemaphore from the system usemaphore table, freeing the entry for use by another usemaphore. A usemaphore may not be deleted unless its use count is zero.

name is a pointer to the usemaphore's name string. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_usema_create
_os_usema_unlink
F_EVENT, EV_DELETE | F_USEMA OS-9 Technical Manual
Link to Existing Usemaphore by Name

**Syntax**

```c
#include <semaphore.h>
#include <types.h>
error_code _os_usema_link(
    usema_id *us_id);
    const char *name,
```

**Description**

_os_usema_link() determines the ID number of an existing usemaphore. Use _os_usema_unlink() to unlink from a usemaphore when it is no longer needed. The operating system will automatically unlock, if necessary, and unlink any usemaphores still linked when a process terminates.

*us_id*

is a pointer to the location where _os_usema_link() stores the usemaphore identifier. (Output)

*name*

is a pointer to the usemaphore name string. (Input)

**Attributes**

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

`os_lib.l`

**See Also**

- _os_usema_unlink
- F_EVENT, EV_LINK | F_USEMA

*OS-9 Technical Manual*
_os_usema_p()

Acquire Ownership of a Usemaphore

Syntax
#include <semaphore.h>
#include <types.h>
error_code _os_usema_p(
    usema_id us_id,
    signal_code *signal);

Description
_os_usema_p() waits for the ownership of a usemaphore. If the usemaphore is currently owned by a process this function will block until the owner releases the usemaphore and the caller is granted ownership.

If the specified usemaphore has been automatically unlocked by the operating system and not reset (refer to _os_usema_reset()) the error EOS_USRST will be returned.

us_id
    identifies the usemaphore. (Input)

signal
    is a pointer to the location where _os_usema_p() stores the signal code. (Output)

    If a process in the usemaphore queue receives a signal, it is activated even though ownership has not been granted. An error is returned in this case.

Attributes
Operating System:   OS-9
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library
os_lib.l

See Also
_os_usema_v
_os_usema_try_p
_os_usema_reset
F_EVENT, EV_WAIT | F_USEMA
Reset a Usemaphore for Normal Operation

Syntax

```c
#include <semaphore.h>
#include <types.h>
error_code _os_usema_reset(usema_id us_id);
```

Description

_os_usema_reset() waits for the ownership of a usemaphore and then clears the need for a reset. This is a non-blocking call. If the usemaphore needs to be reset the caller will immediately be granted ownership. If not, the EOS_USNORST error will be returned.

A usemaphore would need to be reset after the operating system automatically unlocks a locked usemaphore because the owning process terminated.

Release ownership of the usemaphore with _os_usema_v().

us_id
identifies the usemaphore. (Input)

signal
is a pointer to the location where _os_usema_p() stores the signal code. (Output)

If a process in the usemaphore queue receives a signal, it is activated even though ownership has not been granted. An error is returned in this case.

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.1

See Also

_os_usema_p
_os_usema_try_p
_os_usema_v
F_EVENT, EV_RESET | F_USEMA OS-9 Technical Manual
_os_usema_try_p()

Acquire Ownership of a Usempahore Without Blocking

Syntax

#include <semaphore.h>
#include <types.h>
error_code _os_usema_tryp(usema_id us_id);

Description

_os_usema_tryp() tries to acquire the ownership of a usemaphore. If the usemaphore is currently owned by a process this function returns immediately with the error EAGAIN.

If the specified usemaphore has been automatically unlocked by the operating system and not reset (refer to _os_usema_reset()) the error EOS_USRST will be returned.

us_id identifies the usemaphore. (Input)

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_usema_p
_os_usema_v
_os_usema_reset
F_EVENT, EV_TRYWAIT | F_USEMA
_os_usema_unlink()

Unlink Usemaphore

Syntax

```c
#include <semaphore.h>
#include <types.h>
error_code _os_usema_unlink(usema_id us_id);
```

Description

_**os_usema_unlink()**_ decrements the use count of a usemaphore. When the use count reaches zero, you can delete the usemaphore using _os_usema_delete()_.

```c
us_id
    identifies the usemaphore. (Input)
```

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.1

See Also

_**os_usema_create**_
_**os_usema_link**_

F_EVENT, EV_UNLINK | F_USEMA OS-9 Technical Manual
Release Usemaphore Ownership

Syntax

#include <semaphore.h>
#include <types.h>
error_code _os_usema_v(usema_id us_id);

Description

_os_usema_v() releases ownership of the usemaphore. Then, the usemaphore queue
is searched for a process waiting for the usemaphore.

Any usemaphores that are owned by a process at process termination time are
automatically released.

us_id
    specified the usemaphore to release. (Input)

Attributes

Operating System:    OS-9
State:               User and System
Threads:             Safe
Re-entrant:          Yes

Library

os_lib.1

See Also

_os_usema_p
F_EVENT, EV_WAIT | F_USEMA OS-9 Technical Manual
_os_vmodul()
Verify Module

Syntax
#include <module.h>
#include <types.h>
error_code _os_vmodul(
    mh_com *mod_head,
    mh_com *mod_block,
    u_int32 block_size);

Description
_os_vmodul() checks the module header parity and CRC bytes of an OS-9 module. If the header values are valid, the module is entered into the module directory. The current module directory is searched for another module with the same name. If a module with the same name and type exists, the module with the highest revision level is retained in the module directory. Ties are broken in favor of the established module.

mod_head
    is a pointer to the address of the module. (Input)

mod_block
    is a pointer to the memory block containing the module. (Input)

block_size
    specifies the size of the memory block containing the module. (Input)

Attributes
Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_crc()
_os_load()
_os9_vmodul()

Validate Module

Syntax

```
#include <module.h>
#include <types.h>
error_code _os9_vmodul(
    u_int32 mod_group,
    u_int32 size,
    mh_com *mod_head,
    mod_dir **dir_entry);
```

Description

_os9_vmodul() checks the module header parity and CRC bytes of an OS-9 module. If the header values are valid, the module is entered into the module directory, and a pointer to the directory entry is returned. The current module directory is searched for another module with the same name. If a module with the same name and type exists, the module with the highest revision level is retained in the module directory. Ties are broken in favor of the established module.

- **mod_group**
  - specifies the beginning of the module group (ID). (Input)

- **size**
  - is the size of the module. (Input)

- **mod_head**
  - is a pointer to the module. (Input)

- **dir_entry**
  - is a pointer to the location where _os9_vmodul() stores the pointer to the directory entry. (Output)

Attributes

- Operating System: OS-9 for 68K
- State: System
- Threads: Safe
- Re-entrant: Yes

Library

os_lib.l

See Also

_os_crc()
_os_load()
_os_wait()

Wait for Child Process to Terminate

Syntax
#include <process.h>
#include <types.h>
error_code _os_wait(
    process_id *child_id,
    status_code *status);

Description
_os_wait() deactivates the calling process until a child process terminates by executing an _os_exit() system call or is otherwise terminated. The child’s ID number and exit status are returned to the parent. If the child process died due to a signal, status contains the signal code.

If a signal is received by a process waiting for children to terminate, the process is activated. In this case, child_id contains zero, because no child process has terminated.

child_id
is a pointer to the location where _os_wait() stores the process ID of the terminating child. (Output)

status
is a pointer to the location where _os_wait() stores the child process’ exit status code. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l

See Also
_os_exit()
_os_fork()
_os_send()
F$Wait
F_WAIT

OS-9 for 68K Technical Manual
OS-9 Technical Manual
_os_waitid()
Wait for a Specified Process or Thread to Exit

Syntax

```c
#include <process.h>
error_code _os_waitid(
    process_id   *child_id,
    error_code   *status,
    signal_code  *signal,
    u_int32      wait_flag);
```

Description

_os_waitid() has two primary functions:

- waiting for a child process or sibling thread
- controlling a signal for the death of a child process or sibling thread

To specify a wait related activity, `wait_flag` should be 0.

- **child_id**
The value pointed by `child_id` specifies the process or thread to wait for. If the value of `child_id` is the ID of a thread, the caller must be a thread in the same process as `child_id`. Otherwise an EOS_IPRCID error is returned. (Input)

- **status**
If the call is successful, the exit code of `child_id` is returned in the location pointed to by `status`. (Output)

- **signal**
If the wait is interrupted by a signal, a value of EOS_BSIG is returned by _os_waitid() and the value of the signal that caused the interruption is stored in the location pointed to by `signal`. (Output)

- **wait_flag**
To specify a signal related activity, `wait_flag` should be non-zero. The valid values for `wait_flag` are WT_SIGNAL and WT_RELEASE. (Input)

  When `wait_flag` is WT_SIGNAL it specifies that the caller wants to receive a signal when the process or thread specified by `child_id` terminates. The value of the signal to be sent is pointed to by `signal`. If the process or thread specified by `child_id` has already terminated, the signal is sent immediately.

  When `wait_flag` is WT_RELEASE it specifies that the caller is no longer interested in getting a signal on the termination of the process or thread specified by `child_id`. The value pointed to by `signal` is irrelevant in this case.

  The value pointed to by `status` is not modified when `wait_flag` is non-zero.

_os_waitid() returns when `wait_flag` is non-zero; it never blocks, regardless of the state of the child.
Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
os_lib.l, mt_os_lib.l

See Also
_os_exit()
_os_thexit()
_os_wait()
Activate Next Process Waiting to Acquire Lock

Syntax

```c
#include <lock.h>
#include <types.h>
error_code _os_waitlk(
    lk_desc *lock,
    signal_code *signal);
```

Description

_os_waitlk() activates the next process waiting to acquire the lock. The next process in the lock’s queue is activated and granted exclusive ownership of the resource lock. If no other process is waiting on the lock, the lock is simply marked free for acquisition. The calling process in either case is suspended and inserted into a waiting queue for the resource based on relative scheduling priority.

**lock**
- is a pointer to the lock on which to wait. (Input)

**signal**
- is a pointer to the location where _os_waitlk() stores the received signal. (Output)
  - If a process receives a signal while waiting on a lock, the process is activated without gaining ownership of the lock.
  - The process returns from the wait lock call with an EOS_SIGNAL error code and the signal code is returned through the signal pointer.
  - If an _S_WAKEUP signal is received by a waiting process, the signal code does not register and is zero.

Attributes

Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library

lock.1

See Also

_os_acqlk() _os_caqlk()
_os_crlk() _os_ddlk()
_os_dellk() _os_rellk()
F_WAITLK OS-9 Technical Manual
Syntax
#include <modes.h>
#include <types.h>
error_code _os_write(
    path_id path,
    void *buffer,
    u_int32 *count);

Description
_os_write() outputs bytes to a file or device associated with the specified path number. The path must be opened or created in the write or update access modes.

Data is written to the file or device without processing or editing. If data is written past the present end-of-file, the file is automatically expanded.

If the device is opened in block mode (S_IBLKMODE), the count parameter passed must be a multiple of the block size of the device, whereas in non-block mode it is interpreted as the number of bytes to read.

On RBF devices, any record that was locked is released.

path
    is the specified path number for the file or device. (Input)

buffer
    is a pointer to the data buffer. (Output)

count
    The number of bytes to write is passed at the location pointed to by count. (Input/Output)

    _os_write() also stores the number of bytes written at the location pointed to by count.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:             User and System
Threads:           Safe
Re-entrant:        Yes

Library
os_lib.l

See Also
_os_open()          _os_create()
_os9_create()        _os.writeln()
_os_writeln()

Write Line of Text with Editing

Syntax

```c
#include <modes.h>
#include <types.h>
error_code _os_writeln(
    path_id path,
    void *buffer,
    u_int32 *count);
```

Description

_os_writeln() outputs bytes to a file or device associated with the specified path number. The path must have been opened or created in write or update access modes. _os_writeln() writes data until it encounters a carriage return character or count bytes. Line editing is also activated for character-oriented devices such as terminals and printers.

If the device is opened in block mode (S_IBLKMODE), the count parameter passed must be a multiple of the block size of the device, whereas in non-block mode it is interpreted as the number of bytes to read.

On RBF devices, any record that was locked is released.

path

is the path number of the file or device. (Input)

buffer

is a pointer to the location where the bytes to be written are placed. (Output)

count

the number of bytes to write is passed at the location pointed to by count. (Input/Output)

_os_writeln() also stores the number of bytes written at the location pointed to by count.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

os_lib.l

See Also

_os_open()  _os9_create()
_os_create()  _os_write()
_os_yield()  
Yield the Processor

**Syntax**

```c
#include <process.h>
error_code _os_yield(void);
```

**Description**

`_os_yield()` causes the calling process or thread to be placed back into the active queue. The active queue contents are aged and the highest aged process is given control of the processor. In other words, `_os_yield()` causes the operating system to advance to the next executable process or thread. It is possible that the next executable process or thread will be the one that called `_os_yield()`. The status of the process’ or thread’s signal mask remains unchanged during this system call. `_os_yield()` is much like `_os_sleep()` with a tick count of 1, except that signals are not implicitly unmasked.

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`os_lib.1`

**See Also**

`_os_sleep()`
outc()

Write Character to I/O Address

Syntax

#include <regs.h>
#include <types.h>

void outc(
    void *address,
    u_int32 data);

Description

outc() writes a character to an I/O address on the 80x86 processors or memory address on all other processors.

data
    specifies the character to write. (Input)
    Only the lower 8 bits are used.

address
    specifies the I/O or memory address. (Input)

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library

cpu.l

See Also

outl()
outw()
inc()
inl()
inw()
## outl()

### Write Integer to I/O Address

**Syntax**

```c
#include <regs.h>
#include <types.h>

void outl(
    void *address,
    u_int32 data);
```

**Description**

`outl()` writes a long value to the I/O address on the 80x86 or the memory address on all other processors.

- **data**: specifies the long value to write. (Input)
- **address**: specifies the I/O or memory address. (Input)

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

cpu.l

**See Also**

- `outc()`
- `outw()`
- `inc()`
- `inl()`
- `inw()`
outw()

Write Word to I/O Address

Syntax
#include <regs.h>
#include <types.h>
void outw(
    void *address,
    u_int32 data);

Description
outw() writes a word to the I/O address on the 80x86 or the memory address on all other processors.

data
    specifies the word to write. (Input)
    Only the lower 16 bits are used.

address
    specifies the I/O or memory address. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
outc()
outl()
inc()
inl()
inw()
Syntax

```c
#include <strings.h>
int _parsepath(const char *string);
```

Description

(Parsepath) parses a disk file (RBF) pathlist. (Parsepath) is useful for programs that must determine the validity of a pathlist name without actually creating or opening a file.

(Parsepath) returns the number of bytes in the valid pathlist. If RBF does not accept the pathlist as valid, (Parsepath) returns -1.

string is a pointer to the pathlist to parse. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

sys_clib.1

See Also

(Parseos_prsname)
F$PrsNam OS-9 for 68K Technical Manual
F_PRSNAM OS-9 Technical Manual
**Syntax**

```c
#include <signal.h>
int pause(void);
```

**Description**

`pause()` suspends a task until a signal is received. `pause()` always returns `-1` unless the received signal caused the process to terminate. In this case, `pause()` never returns.

The thread enabled version of this function contains a cancel point. For more information see *Using OS-9 Threads*.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

sys_clib.1

**See Also**

- `kill()`
- `_os_sleep()`
- `_os9_sleep()`
- `sleep()`
- F$Sleep
- F_SLEEP

*OS-9 for 68K Technical Manual*

*OS-9 Technical Manual*
pclose()

Close a Pipe from/to a Process

Syntax

#include <UNIX/os9def.h>
#include <stdio.h>

int pclose(FILE *stream)

Description

close() closes a pipe from or to a process that was opened by popen(). pclose() waits for the associated process to terminate and returns the exit status of the command.

stream is a pointer to the file. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library

unix.l

See Also

popen()
Syntax

```c
#include <stdio.h>
void perror(const char *chptr);
```

Description

`perror()` maps the error number in `errno` to an error message.

The contents of the error message strings are the same as those returned by `strerror()` with parameter `errno`.

`chptr` is a pointer to a character. (Input)

If `chptr` is not a null pointer and the character pointed to by `chptr` is not a null character, `perror()` writes a sequence of characters to the standard error stream. The sequence written contains `chptr`, a colon (:`), a space, and an appropriate error message string followed by a newline character.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`clib.l`

See Also

`strerror()`
pffinit()

Initialize for Float Output (Obsolete)

Syntax

```c
int pffinit(void);
```

Description

`pffinit()` was used on the 6809 C compiler to allow `printf()` to perform float and double conversions. A dummy function named `pffinit()` exists here for 6809/68000 portability.

⚠️ `pffinit()` is historical and may be removed in a future release.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

```c
sys_clib.l
```
Syntax

```c
int pflinit(void);
```

Description

`pflinit()` was used on the 6809 C compiler to allow `printf()` to perform `long int` conversions. A dummy function named `pflinit()` exists here for 6809/68000 portability.

⚠️ `pflinit()` is historical and may be removed in a future release.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`
pipe()
Create an Interprocess Communication Channel

Syntax

```c
#include <UNIX/os9def.h>
int pipe(unsigned int fd[2]);
```

Description

`pipe()` creates an I/O mechanism called a pipe and returns 2 path numbers, `fd[0]` and `fd[1]`. The standard programming model is that after the pipe has been set up, two (or more) cooperating processes (created by subsequent `_os_exec` calls) pass data through the pipe using `read` and `write` I/O calls.

If an error occurs, the `pipe()` returns -1 and sets the global variable `errno` to indicate the error.

`fd[0]` and `fd[1]` are open for `read` and `write` access. (Output)

When the pipe is written, up to 4 KB of data are buffered before the writing process is blocked. Data read from one of the paths accesses data written to either of the paths on a FIFO (first-in-first-out) basis.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

⚠️ If `pipe()` is called from system-state, `fd[0]` and `fd[1]` will refer to the same system-state path number and the use count on the path will be two.

Library

- `unix.l`

See Also

- `pclose()`
- `popen()`
**Syntax**

```c
#include <UNIX/os9def.h>
#include <stdio.h>
FILE *popen(
    const char *command,
    const char *type);
```

**Description**

`popen()` opens a pipe from or to a process. The arguments to `popen()` are pointers to null-terminated strings containing, respectively, a shell command line and an I/O mode, either `r` for reading or `w` for writing. `popen()` creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer such that you can write to the standard input of the command, if the I/O mode is `w`, by writing to the file stream. If the I/O mode was `r`, you can read from the standard output of the command by reading from the file stream. A stream opened by `popen()` should be closed with `pclose()` which waits for the associated process to terminate and returns the exit status of the command.

- `command`  
  is the command written to. (Input)

- `type`  
  is the character mode. (Input)

**Attributes**

- Operating System:  
  OS-9 and OS-9 for 68K
- State:  
  User
- Threads:  
  Safe
- Re-entrant:  
  Yes

**Library**

`unix.l`

**See Also**

- `pclose()`
- `pipe()`
- `popen()`
Syntax

#include <math.h>
double pow(
    double x,
    double y);

Description

pow() returns $x$ (input) raised to the $y$ (input) power. A domain error occurs, EDOM stored in errno, if $x$ is negative. A domain error also occurs if both $x$ and $y$ are zero. A range error, ERANGE stored in errno, occurs on underflow and overflow. pow() returns HUGE_VAL for both errors.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library

clib.l

Possible Errors

EDOM
ERANGE
_prealloc_rand()

Return No Error in Calls to Rand

Syntax

```c
#include stdlib.h
error_code _prealloc_rand(void);
```

Description

_prealloc_rand can be called to ensure that future calls to rand do not return an error.

_prealloc_rand returns SUCCESS if successful. Otherwise it returns an error number.

Attributes

- Operating System: OS-9
- State: User
- Threads: Safe
- Re-entrant: No

Library

clib.l
**Syntax**

```c
#include <stdio.h>
int prerr(
    int path,
    int errnum);
```

**Description**

`prerr()` prints an error message corresponding to an error number on the standard error path.

`prerr()` prints a message in the form “Error #ggg:sss” where ggg is the group number of the error and sss is the specific error number within the group. If path is non-zero, the opened error message file is searched for further text related to the error. If more text is available, it is printed.

- **path** is the path number of the error message file. (Input)
- **errnum** is the error number. (Input)
  
  It is divided into two numbers.
  
  - OS-9 for 68K: Error numbers are 2 bytes wide. The most significant byte is the error’s group number, and the least significant byte is the specific error within that group.
  
  - OS-9: Error numbers are 4 bytes wide. The most significant 2 bytes are the error’s group number, and the least significant 2 bytes are the specific error within that group.

**Example**

```c
prerr(0, 216) /* function call */
Error #000:216 /* output */
prerr(errmsg_path, 216) /* function call */
Error #000:216 (E$PNNF) File not found /* output line 1 */
The pathlist does not lead to any known file. /* output line 2 */
```

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_perr()
F$PErr

OS-9 for 68K Technical Manual
Syntax

#include <module.h>
char *prgname(void);

Description

_prgname() returns a pointer to the name of the calling module. Normally, the argv[0] string indicates the name of the program as called by the parent process with os9exec(). You can use _prgname() to determine the actual name of the module as it appears in the module directory.

If the code that calls _prgname() is executing in a trap handler, the name of the trap handler module is returned.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib

See Also

_errmsg()
os9exec()
printf() is a standard C library function that performs formatted output to stdout. It converts, formats, and prints any parameters as indicated by the control string. printf() returns the number of characters transmitted, or a negative value if an output error occurred.

format is a pointer to a control string. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also
fprintf()
scanf()
sprintf()
Syntax
#include <strings.h>
int _prsnam(const char *string);

Description
_prsnam() parses a path name segment.
_prsnam() returns the number of characters in a valid path name segment. If the path name segment is invalid, -1 is returned. Use _parsenam() to determine if a path is a valid disk file (RBF) pathlist.

You can use successive calls to _prsnam() to parse a complete path name.

string
is a pointer to the path name segment. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_prsnam()
F$PrsNam  OS-9 for 68K Technical Manual
F_PRSNAM  OS-9 Technical Manual
### Syntax

```c
#include <pthread.h>

int pthread_attr_destroy(pthread_attr_t *attr);
```

### Description

`pthread_attr_destroy()` tells the library that a pthread attribute, `attr`, will no longer be used. The attribute, in effect, becomes uninitialized. If successful, it returns a value of 0; otherwise, an error.

- `attr` is a pointer to an initialized pthread attribute object. (Input)

### Example

```c
er = pthread_attr_destroy(&attr);
if (err != 0)
    fprintf(stderr, “error destroying attribute - %s\n”, strerror(err));
```

### Attributes

- Operating System: OS-9
- State: User
- Compatibility: POSIX

### Library

`mt_clib.l`

### Possible Errors

- **EINVAL** An invalid `pthread_attr_t` pointer was passed.

### See Also

- `pthread_attr_init()`
- `pthread_create()`
**Syntax**

```c
#include <pthread.h>

int pthread_attr_getdetachstate(
    const pthread_attr_t  *attr,
    int                   *detachstate);
```

**Description**

`pthread_attr_getdetachstate()` gets the detach state attribute, `attr`, in the attribute object. The integer pointed to by `detachstate` (input) will be written with either `PTHREAD_CREATE_DETACHED` or `PTHREAD_CREATE_JOINABLE`.

If successful, returns a value of 0; otherwise, returns an error.

- **attr** is a pointer to an initialized pthread attribute object. (Input)

**Example**

```c
er = pthread_attr_getdetachstate(&attr, &state);
if (err != 0)
    fprintf(stderr, "error getting detach state - %s\n", strerror(err));
```

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **EINVAL** attr or detachstate is invalid or the object pointed to by attr is not properly initialized.

**See Also**

- `pthread_attr_init()`
- `pthread_attr_setdetachstate()`
- `pthread_create()`
- `pthread_detach()`
- `pthread_join()`
_pthread_attr_getinitfunction()

Get Initialization Function Attribute

Syntax
#include <pthread.h>

int _pthread_attr_getinitfunction(
    const pthread_attr_t *attr,
    int (**initfunc)(void *),
    void **initfunc_arg,
    void **initfunc_gp,
    void **initfunc_cp);

Description
(pthread_attr_getinitfunction()) returns the initialization function pointer and initialization function argument fields from an attribute objects.
attr
is a pointer to an initialized pthread attribute object. (Input)
initfunc
points to a place to store the initialization function pointer. (Output)
initfunc_arg
points to a place to store the initialization function argument. (Output)
initfunc_gp
points to a place to store the initialization function global pointer. (Output)
initfunc_cp
points to a place to store the initialization function constant pointer. (Output)

Refer to _pthread_attr_setinitfunction() for more information about these fields.

Example
err = _pthread_attr_getinitfunction(&attr, &initfunc, &initfunc_arg, &gp, &cp);
if (err != 0)
    fprintf(stderr, "error getting initialization function - %s\n",
            strerror(err));

Attributes
Operating System: OS-9
State: User
_pthread_attr_getinitfunction()

Library

mt_clib.l

Possible Errors

EINVAL       attr does not refer to an initialized attributes object.
initfunc or initfunc_arg is invalid.

See Also

pthread_attr_init()
(pthread_attr_setinitfunction()
pthread_create()
_pthread_attr_getpriority()
Get Priority Attribute

Syntax
#include <pthread.h>
int _pthread_attr_getpriority(
    const pthread_attr_t  *attr,
    u_int32               *priority);

Description
(pthread_attr_getpriority) sets the u_int32 pointed to by priority with the
current priority setting from the specified pthread attribute object pointed to by
attr. A value of 0 indicates that threads created with the specified attribute object
will adopt the priority of the creating thread. A non-zero value indicates the desired
priority for the created thread.

If successful, returns a value of 0; otherwise, returns an error.

attr is
    a pointer to an initialized pthread attribute object. (Input)

priority is a pointer to the unsigned integer. (Input)

Attributes
Operating System: OS-9
State: User

Library
mt_clib.l

Possible Errors
EINVAL attr or priority is invalid or the object pointed to by attr is
not properly initialized.

See Also
pthread_attr_init()
 pthread_attr_setpriority()
 pthread_create()

Example
er = _pthread_attr_getpriority(&attr, &pr);
if (err != 0)
    fprintf(stderr, "error getting priority - %s\n", strerror(err));
**pthread_attr_getstackaddr()**

*Get Stack Address Attribute*

**Syntax**

```c
#include <pthread.h>

int pthread_attr_getstackaddr(
    const pthread_attr_t  *attr,
    void                  **stackaddr);
```

**Description**

`pthread_attr_getstackaddr()` gets the thread stack address attribute, `attr` (input), in the attribute object.

`pthread_attr_getstackaddr()` stores the thread stack address attribute value in `stackaddr` (output) if successful.

If successful, returns a value of 0; otherwise, returns an error.

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- `EINVAL` attr or stackaddr is invalid or the object pointed to by attr is not properly initialized.

**See Also**

- `pthread_attr_init()`
- `pthread_attr_setstackaddr()`
- `pthread_create()`

**Example**

```c
err = pthread_attr_getstackaddr(&attr, &stack);
if (err != 0)
    fprintf(stderr, "error getting stack address - %s\n", strerror(err));
printf("Highest stack address is 0x%x\n", stack);
```
# pthread_attr_getstacksize()

## Syntax

```c
#include <pthread.h>

int pthread_attr_getstacksize(
    const pthread_attr_t *attr,
    size_t *stacksize);
```

## Description

`pthread_attr_getstacksize()` gets the thread stack size attribute, `attr` (input), in the attribute object.

`pthread_attr_getstacksize()` stores the thread stack size attribute value in `stacksize` (output) if successful.

If successful, returns a value of 0; otherwise, returns an error.

## Attributes

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

## Library

`mt_clib.l`

## Possible Errors

- **EINVAL**
  - `attr` or `stacksize` is invalid or the object pointed to by `attr` is not properly initialized.

## See Also

- `pthread_attr_init()`
- `pthread_attr_setstacksize()`
- `pthread_create()`

## Example

```c
err = pthread_attr_getstacksize(&attr, &size);
if (err != 0)
    fprintf(stderr, "error getting stack size - %s\n", strerror(err));
printf("Stack size will be %u\n", size);
```
**pthread_attr_init()**

Allocate Thread Creation Attribute Object

**Syntax**

```c
#include <pthread.h>
int pthread_attr_init(pthread_attr_t *attr);
```

**Description**

`pthread_attr_init()` sets default values into the pthread creation attribute object. The default values for a thread creation attribute object are shown in Table 2-18:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Size</td>
<td>PTHREAD_STACK_MIN</td>
</tr>
<tr>
<td>Stack Address</td>
<td>NULL (system allocated stack)</td>
</tr>
<tr>
<td>Detach State</td>
<td>PTHREAD_CREATE_JOINABLE</td>
</tr>
<tr>
<td>Priority</td>
<td>0 (priority of creator)</td>
</tr>
<tr>
<td>Initialization Function</td>
<td>NULL (none)</td>
</tr>
</tbody>
</table>

If successful, returns a value of 0; otherwise, returns an error.

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **ENOMEM** Insufficient memory exists to initialize the attribute.
- **EINVAL** `attr` is invalid.

**See Also**

`pthread_create()`

**Example**

```c
e = pthread_attr_init(&attr);
if (e != 0)
    fprintf(stderr, "error initializing attribute - %s\n", strerror(e));
```
pthread_attr_setdetachstate()

Set Detached State Attribute

Syntax

```c
#include <pthread.h>
int pthread_attr_setdetachstate(
   pthread_attr_t *attr,
   int         detachstate);
```

Description

`pthread_attr_setdetachstate()` sets the detach state attribute of the specified attribute object. Valid values for `detachstate` are `PTHREAD_CREATE_DETACHED` or `PTHREAD_CREATE_JOINABLE`.

Threads created as joinable retain information upon exit so that status can be returned when `pthread_join()` is used, unless `pthread_detach()` is used to detach the thread. Threads created as detached automatically free all resources upon exit and cannot be used with `pthread_join()`. These type of threads are forked as “orphan” OS-9 threads. If successful, this function returns a value of 0; otherwise, returns an error.

- `attr` is a pointer to an initialized pthread attribute object. (Output)
- `detachstate` is the detach state attribute of the specified attribute object. (Input)

Attributes

- Operating System: OS-9
- State: User
- Compatibility: POSIX

Library

mt_clib.l

Possible Errors

- `EINVAL` attr or `detachstate` is not valid.

See Also

- `pthread_attr_init()` `pthread_attr_getdetachstate()`
- `pthread_create()` `pthread_detach()`
- `pthread_join()`

Example

```c
err = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
if (err != 0)
   fprintf(stderr, “error setting to detached state - %s
”,
   strerror(err));
```
### Syntax

```c
#include <pthread.h>

int _pthread_attr_setinitfunction(
    const pthread_attr_t *attr,
    int (*initfunc)(void *),
    void *initfunc_arg,
    void *gp,
    void *cp);
```

### Description

`_pthread_attr_setinitfunction()` sets the initialization function address, argument, global data and constant pointer fields of an attribute object.

The initialization function has the following prototype:

```c
int initfunc(void *initfunc_arg);
```

The value of the argument is the value of the `initfunc_arg` parameter in the thread's creation attributes object. If the initialization function returns a non-zero value, that value is converted to a pointer to void and passed to `pthread_exit()`, thus terminating the created thread without ever calling the intended start function.

The initialization function is called in the context of the created thread before the call to `pthread_create()` returns to its caller. The function may perform application specific thread initialization as necessary. The function could be useful in eliminating any race conditions that may exist between the creating thread and created thread since it is known that the initialization code will run in the created thread prior to the return from `pthread_create()`.

Although `pthread_self()` will function correctly, the value it returns should not be communicated to any other threads. The created thread has, technically, not finished its initialization, thus it is not ready to handle all thread operations. The initialization function should not interact with any other threads.

The initialization function runs at the priority of the creating thread, instead of the priority specified for the created thread. That is, if a high priority thread is creating a low priority thread with an initialization function, the initialization function will execute at high priority in the context of the low priority thread.

- `attr` is a pointer to an initialized pthread attribute object. (Input)
- `initfunc` points to the initialization function. (Input)
initfunc_arg

is the argument to pass as the initialization functions sole argument. (Input)

gp

specifies the global data pointer that should be in place when calling initfunc. (Input)

cp

specifies the constant pointer that should be in place when calling initfunc. (Input)

If a constant pointer is not applicable for a particular processor or the code is compiled in such a way that a constant pointer is not needed, the value of cp may be NULL. Passing NULL as the initfunc parameter disables the calling of an initialization function. Passing NULL as the initfunc_arg parameter simply specifies that the value of the initialization function parameter should be NULL.

Attributes

Operating System: OS-9
State: User

Library

mt_clib.l

Possible Errors

EINVAL attr does not refer to an initialized attributes object.

See Also

pthread_attr_init()
_pthread_attr_getinitfunction()

pthread_create()

pthread_exit()

get_static()

get_const()

Example

err = _pthread_attr_setinitfunction(&attr, thread_startup, &sema);
if (err != 0)
    fprintf(stderr, "error setting initialization function - %s\n",
strerror(err));
Syntax
#include <pthread.h>

int _pthread_attr_setpriority(
    pthread_attr_t *attr,
    u_int32 priority);

Description
(pthread_attr_setpriority()) sets the priority attribute of the pthread attribute object pointed to by attr to priority. A priority of 0 indicates that created threads should adopt the priority of the creating thread. A non-zero value specifies the desired priority for the created thread.

attr
    is a pointer to an initialized pthread attribute object. (Input)

priority
    is a pointer to the unsigned integer. (Input)

If successful, returns a value of 0; otherwise, returns an error.

Attributes
Operating System: OS-9
State: User

Library
mt_clib.l

Possible Errors
EINVAL
    If attr is invalid or the object pointed to by attr is not properly initialized or the value of priority is out of range for a thread priority (0 to 65535).

See Also
pthread_attr_init()
(pthread_attr_getpriority()  
pthread_create()  
(pthread_setpr()  

Example
err = _pthread_attr_setpriority(&attr, 255);
if (err != 0)
    fprintf(stderr, "error setting priority - %s\n", strerror(err));
# pthread_attr_setstackaddr()

## Set Stack Address Attribute

### Syntax

```c
#include <pthread.h>

int pthread_attr_setstackaddr(
    pthread_attr_t *attr,
    void            *stackaddr);
```

### Description

`pthread_attr_setstackaddr()` allows a thread to specify a particular pre-allocated thread stack. The address specified is the desired stack pointer for the created thread. The specified stack must be at least `PTHREAD_STACK_MIN` in size.

There is a matrix of possibilities for the two functions `pthread_attr_setstacksize()` and `pthread_attr_setstackaddr()`. Either one can be called independent of the other one being called. The behavior depends upon the following matrix, shown in Table 2-19.

### Table 2-19. Function Behavior

<table>
<thead>
<tr>
<th>Setstackaddr()</th>
<th>Setstacksize()</th>
<th>Resultant behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not called</td>
<td>Not called</td>
<td>A system-allocated stack, of size <code>PTHREAD_STACK_MIN</code>, will be given to created threads.</td>
</tr>
<tr>
<td>Called</td>
<td>Not called</td>
<td>The specified stack address will be passed to created threads. The size will be assumed to be <code>PTHREAD_STACK_MIN</code>.</td>
</tr>
<tr>
<td>Not called</td>
<td>Called</td>
<td>A system-allocated stack of the size specified will be passed to created threads.</td>
</tr>
<tr>
<td>Called</td>
<td>Called</td>
<td>The specified stack address will be passed to created threads. The size will be assumed to be the size set by <code>pthread_attr_setstacksize()</code>.</td>
</tr>
</tbody>
</table>

Be aware of the following requirements when setting the stack address explicitly:

- The address passed to this function will be passed directly to threads created with this attribute object. Make sure that the top of the stack is passed (the highest RAM address of the stack).
- Do not create more than one thread with a given stack address.
- The stack should be "pre-loaded" with a NULL link pointer to ensure proper stack back-tracing.
If successful, returns a value of 0; otherwise, returns an error.

attr
is a pointer to an initialized pthread attribute object. (Input)

stackaddr
is a pointer to the pre-allocated thread stack. (Input)

The stackaddr parameter is rounded down to an eight-byte boundary. To get the actual stack address used for created threads with a given attribute object. Use pthread_attr_getstackaddr() to get the actual address passed to the next created thread.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EINVAL attr is invalid or attr is not properly initialized.

See Also
pthread_attr_init()
pthread_attr_getstackaddr()
pthread_create()

Example
stack = malloc(PTHREAD_STACK_MIN);
if (stack == NULL)
    fprintf(stderr, “error allocating stack - %s
”, strerror(errno));
memset(stack, 0, PTHREAD_STACK_MIN);
err = pthread_attr_setstackaddr(&attr, stack + stacksize);
if (err != 0)
    fprintf(stderr, “error setting stack address - %s
”, strerror(err));
pthread_attr_setstacksize()
Set Stack Size Attribute

Syntax
#include <pthread.h>
int pthread_attr_setstacksize(
    pthread_attr_t  *attr,
    size_t          stacksize);

Description
pthread_attr_setstacksize() sets the stack size that will be allocated for threads that are created with the specified attribute object.

The stacksize parameter is rounded down to an eight-byte boundary. To get the actual stack size used for created threads with a given attribute object. Use pthread_attr_getstacksize() to get the actual stack size attribute used to create threads.

pthread_attr_setstacksize() and pthread_attr_setstackaddr() can be called independently of one another. The behavior depends upon the following matrix:

<table>
<thead>
<tr>
<th>Setstackaddr()</th>
<th>Setstacksize()</th>
<th>Resultant behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not called</td>
<td>Not called</td>
<td>A system-allocated stack, of size PTHREAD_STACK_MIN, will be given to created threads.</td>
</tr>
<tr>
<td>Called</td>
<td>Not called</td>
<td>The specified stack address will be passed to created threads. The size will be assumed to be PTHREAD_STACK_MIN.</td>
</tr>
<tr>
<td>Not called</td>
<td>Called</td>
<td>A system-allocated stack of the size specified will be passed to created threads.</td>
</tr>
<tr>
<td>Called</td>
<td>Called</td>
<td>The specified stack address will be passed to created threads. The size will be assumed to be the size set by pthread_attr_setstacksize().</td>
</tr>
</tbody>
</table>

If successful, returns a value of 0; otherwise, returns an error.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l
**Possible Errors**

EINVAL attr is invalid, attr is not properly initialized or stacksize is less than PTHREAD_STACK_MIN.

**See Also**

pthread_attr_init()

pthread_attr_getstacksize()

pthread_create()

**Example**

err = pthread_attr_setstacksize(&attr, 4096);
if (err != 0)
    fprintf(stderr, "error setting stack size - %s\n", strerror(err));
**pthread_cancel()**

Cancel Target Thread

**Syntax**

```c
#include <pthread.h>
int pthread_cancel(pthread_t thread);
```

**Description**

`pthread_cancel()` cancels the target thread unless it is not currently cancelable. If the thread is not cancelable, the request is held pending until it reaches a cancellation point. The call to `pthread_cancel()` returns immediately regardless of the cancelability of the target thread. If the specified thread has asynchronous cancels enabled it will terminate immediately without doing any sort of cleanup.

When a thread processes a deferred cancel the cleanup routines are called, thread specific data destructors are called, and the thread is terminated with the exit status `PTHREAD_CANCELED`.

Cancelling an asynchronous cancel type thread is guaranteed to cause a loss of resources. For example, the memory allocated to implement thread safety for C library functions will be lost. Use deferred cancellation whenever possible.

In addition, cancelling an asynchronous cancel type thread that is in a queue waiting for a resource will most likely cause the process to exit with an exception.

`thread` is the target thread. (Input)

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **ESRCH**: No thread could be found corresponding to that specified by the given thread ID.

**See Also**

- `pthread_cond_timedwait()`
- `pthread_cond_wait()`
- `pthread_exit()`
- `pthread_join()`
- `pthread_setcancelstate()`
- `pthread_setcanceltype()`

**Example**

```c
err = pthread_cancel(worker);
if (err != 0)
    fprintf(stderr, "error cancelling worker - %s\n", strerror(err));
```
## pthread_cleanup_pop()

### Syntax

```c
#include <pthread.h>

void pthread_cleanup_pop(int execute);
```

### Description

`pthread_cleanup_pop()` removes the routine at the top of the cancellation cleanup stack of the calling thread and invokes the popped thread if `execute` (input) is nonzero.

`pthread_cleanup_pop()` and `pthread_cleanup_push()` have to be in the same lexical scope.

### Attributes

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

### Library

`mt_clib.l`

### Possible Errors

None

### See Also

- `pthread_cancel()`
- `pthread_cleanup_push()`
- `pthread_setcancelstate()`
- `pthread_setcanceltype()`

### Example

```c
pthread_cleanup_pop(1);  /* pop and call top cleanup function */
```
**Syntax**

```c
#include <pthread.h>
void pthread_cleanup_push(
    void (*routine)(void *),
    void  *arg);
```

**Description**

`pthread_cleanup_push()` is similar to the ANSI `atexit()` function. It allows a thread to push a series of routines that should be called if the thread is terminated by `pthread_testcancel()` or `pthread_exit()`. The routines are called in the reverse order that they were pushed onto the cleanup stack. That is, the most recently pushed routine is called first, followed by the next most recent, and so on.

Each `pthread_cleanup_push()` invocation must have an associated `pthread_cleanup_pop()` invocation in the same lexical scope. This is strictly enforced by having the `pthread_cleanup_push()` macro begin with an open brace (`{`) and the `pthread_cleanup_pop()` macro end with a close brace (`}`).

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

mt_clib.l

**Possible Errors**

None

**See Also**

- `pthread_cancel()`
- `pthread_cleanup_pop()`
- `pthread_setcancelstate()`
- `pthread_setcanceltype()`

**Example**

```c
err = pthread_mutex_lock(mutx);
if (err != 0)
    fprintf(stderr, “error locking mutex - %s\n”, strerror(err));
pthread_cleanup_push(pthread_mutex_unlock, mutx);
err = pthread_cond_wait(condvar, mutx); /* cancellation point */
if (err != 0)
    fprintf(stderr, “error during cond_wait - %s\n”, strerror(err));
pthread_cleanup_pop(1); /* unlock mutx */
```
**pthread_cond_broadcast()**

**Description**

`pthread_cond_broadcast()` releases every thread waiting on the specified condition variable.

If more than one thread is blocked on a condition variable, the OS-9 scheduler determines the order in which threads are activated. When each thread is unblocked it returns from its call to `pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread owns the mutex with which it called `pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread(s) that are unblocked contend for the mutex in the normal fashion, as if each had called `pthread_mutex_lock()`.

`pthread_cond_broadcast()` may be called by a thread whether or not that thread currently owns the mutex that threads calling `pthread_cond_wait()` or `pthread_cond_timedwait()` have associated with the condition variable during their waits. However, if predictable scheduling behavior is required, then that mutex should be locked by the thread calling `pthread_cond_broadcast()`.

`pthread_cond_broadcast()` has no effect if there are no threads currently blocked on cond.

If successful, returns a value of 0; otherwise, returns an error.

**cond**

Input

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- EINVAL: The value `cond` does not refer to an initialized condition variable.

**See Also**

- `pthread_cond_init()`
- `pthread_cond_timedwait()`
- `pthread_cond_wait()`
- `pthread_cond_signal()`
Example

err = pthread_cond_broadcast(cond);
if (err != 0)
    fprintf(stderr, "failed to signal readers - %s\n", strerror(err));
err = pthread_mutex_unlock(data_lock);
if (err != 0)
    fprintf(stderr, "failed to unlock data lock - %s\n", strerror(err));
# pthread_cond_destroy()

## Syntax

```c
#include <pthread.h>
int pthread_cond_destroy(pthread_cond_t *cond);
```

## Description

The function `pthread_cond_destroy()` destroys the given condition variable specified by `cond`; the object becomes, in effect, uninitialized.

If successful, returns a value of 0; otherwise, returns an error.

`cond` is a pointer to the given condition variable. (Input)

## Attributes

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

## Library

`mt_clib.l`

## Possible Errors

- **EBUSY**: An attempt to destroy the object referenced by `cond` while it is in use by another thread. For example, while being used in a `pthread_cond_wait()` or a `pthread_cond_timedwait()`.

- **EINVAL**: The value specified by `cond` is invalid.

## See Also

- `pthread_cond_broadcast()`
- `pthread_cond_init()`
- `pthread_cond_signal()`
- `pthread_cond_timedwait()`
- `pthread_cond_wait()`

## Example

```c
err = pthread_cond_destroy(&cond);
if (err != 0)
    fprintf(stderr, "failed to destroy convar - %s\n", strerror(err));
```
pthread_cond_init()
Allocate Condition Variable Object

Syntax

```c
#include <pthread.h>

int pthread_cond_init(
    pthread_cond_t *cond,
    const pthread_condattr_t *attr);
```

Description

The function `pthread_cond_init()` initializes the condition variable referenced by `cond` with attributes referenced by `attr`. If `attr` is NULL, the default condition variable attributes are used; the effect is the same as passing the address of a default condition variable attributes object. Upon successful initialization, the state of the condition variable becomes initialized.

If successful, returns a value of 0; otherwise, returns an error.

- `attr` is a pointer to an initialized pthread attribute object. (Input)
- `cond` is a pointer to the condition variable. (Input)

Attributes

- Operating System: OS-9
- State: User
- Compatibility: POSIX

Library

`mt_clib.l`

Possible Errors

- **EAGAIN**: The system lacked the necessary resources (other than memory) to initialize another condition variable.
- **ENOMEM**: Insufficient memory exists to initialize the condition variable.
- **EBUSY**: An attempt to reinitialize the object referenced by `cond` (a previously initialized, but not yet destroyed, condition variable) has been detected.
- **EINVAL**: The value specified by `cond` or `attr` is invalid.
See Also

pthread_cond_broadcast()
pthread_cond_destroy()
pthread_cond_signal()
pthread_cond_timedwait()
pthread_cond_wait()
 pthread_condattr_init()

Example

err = pthread_cond_init(&cond, NULL);
if (err != 0)
    fprintf(stderr, "failed to initialize condvar - %s\n", strerror(err));
**pthread_cond_signal()**

Release Thread Waiting for Condition Variable

**Syntax**

```c
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t *cond);
```

**Description**

`pthread_cond_signal()` releases one thread waiting on the specified condition variable.

When the thread is unblocked it returns from its call to `pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread owns the mutex with which it called `pthread_cond_wait()` or `pthread_cond_timedwait()`. The thread that is unblocked contends for the mutex in the normal fashion, as if it had called `pthread_mutex_lock()`.

`pthread_cond_signal()` may be called by a thread whether or not that thread currently owns the mutex that threads calling `pthread_cond_wait()` or `pthread_cond_timedwait()` have associated with the condition variable during their waits. However, if predictable scheduling behavior is required, then that mutex should be locked by the thread calling `pthread_cond_signal()`.

`pthread_cond_signal()` has no effect if there are no threads currently blocked on `cond`.

If successful, returns a value of 0; otherwise, returns an error.

`cond` is a pointer to the condition variable. (Input)

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- EINVAL The value `cond` does not refer to an initialized condition variable.
See Also

pthread_cond_init()
pthread_cond_timedwait()
pthread_cond_wait()
pthread_cond_broadcast()

Example

err = pthread_cond_signal(cond);
if (err != 0)
    fprintf(stderr, “failed to signal worker - %s\n”, strerror(err));
err = pthread_mutex_unlock(work_que_lock);
if (err != 0)
    fprintf(stderr, “failed to unlock work queue - %s\n”, strerror(err));
pthread_cond_timedwait() (C Function)

Wait on Condition Variable for Specified Interval

**Syntax**

```c
#include <pthread.h>

int pthread_cond_timedwait(
    pthread_cond_t         *cond,
    pthread_mutex_t        *mutex,
    const struct timespec  *abstime);
```

**Description**

pthread_cond_timedwait() is used to block on a condition variable until an absolute time is reached. It must be called with mutex locked by the calling thread or EINVAL will be returned.

mutex is a pointer to the mutex. (Input)

This function releases the mutex and causes the calling thread to block on the condition variable cond. If another thread is able to acquire the mutex after the about-to-block thread has released it, then a subsequent call to pthread_cond_signal() or pthread_cond_broadcast() in that thread behaves as if it were issued after the about-to-block thread has blocked.

Upon return, the mutex is locked and is owned by the calling thread. When using condition variables, there is always a boolean predicate involving shared variables associated with each condition wait that is true if the thread should proceed. Spurious wakeups from the pthread_cond_timedwait() may occur. Since the return from pthread_cond_timedwait() does not imply anything about the value of this predicate, the predicate should be re-evaluated upon each return.

The effect of using more than one mutex for concurrent pthread_cond_wait() or pthread_cond_timedwait() operations on the same condition variable will result in EINVAL errors being returned. That is, a condition variable becomes bound to a unique mutex when a thread waits on the condition variable, and this dynamic binding ends when the last concurrent wait returns.

A condition wait is a cancellation point. When the cancelability enable state of a thread is set to PTHREAD_CANCEL_DEFERRED, a side effect of acting upon a cancellation request while in a condition wait is that the mutex is re-acquired before calling the first cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up to the point of returning from the call to pthread_cond_timedwait(), but at that point notices the cancellation request and instead of returning to the caller of pthread_cond_timedwait(), starts the thread cancellation activities, which includes calling cancellation cleanup handlers.

A thread that has been unblocked because it has been canceled while blocked in a call to pthread_cond_timedwait() does not consume any condition signal that may be directed concurrently at the condition variable if there are other threads blocked on the condition variable.
pthread_cond_timedwait()

The timespec pointed to by abstime specifies an absolute time in GMT that the call should return if the thread is not awakened by a pthread_cond_signal() or pthread_cond_broadcast().

abstime is a pointer to the absolute time. (Input)

The Microware Pthread implementation supports the concept of interruption as it relates to condition variable waits. If a thread has a pending interruption or is interrupted while blocked, pthread_cond_timedwait() will return EINTR. The mutex will be re-acquired prior to return.

If successful, returns a value of 0; otherwise, returns an error.

This function contains a cancel point.

cond is a pointer to the condition variable. (Input)

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
ETIMEDOUT The time specified by abstime to pthread_cond_timedwait() has passed.
EINVAL The value specified by cond, mutex, or abstime is invalid. Different mutexes are supplied for concurrent
                        pthread_cond_wait() or pthread_cond_timedwait() operations on the same condition variable. The mutex is not
                        owned by the current thread at the time of the call.
EINTR _pthread_interrupt() was called with this thread as the target prior to or during this call.

See Also
pthread_cond_broadcast()
pthread_cond_signal()
pthread_cond_wait()
Example

```c
err = _os_gettime(&tspec->tv_sec, &ticks);
if (err != SUCCESS)
    fprintf(stderr, "error getting GMT - %s\n", strerror(err));

/* give up after 5 seconds */
tspec->tv_sec += 5;
tspec->tv_nsec = 0;
err = pthread_cond_timedwait(cond, mutx, tspec);
switch (err) {
  case EINTR:
    fputs("timed wait interrupted\n", stderr);
    break;
  case ETIMEDOUT:
    fputs("timed wait timed out\n", stderr);
    break;
  default:
    fprintf(stderr, "error on timed wait - %s\n", strerror(err));
    break;
}
```
**pthreadCondWait()**

Wait on Condition Variable

**Syntax**

```c
#include <pthread.h>

int pthreadCondWait(
    pthreadCond_t   *cond,
    pthread_mutex_t  *mutex);
```

**Description**

`pthreadCondWait()` is used to block on a condition variable. It must be called with `mutex` locked by the calling thread or `EINVAL` will be returned.

`mutex` is a pointer to the mutex. (Input)

This function releases the mutex and causes the calling thread to block on the condition variable `cond`. If another thread is able to acquire the mutex after the about-to-block thread has released it, then a subsequent call to `pthreadCondSignal()` or `pthreadCondBroadcast()` in that thread behaves as if it were issued after the about-to-block thread has blocked.

Upon return, the mutex is locked and is owned by the calling thread. When using condition variables, there is always a boolean predicate involving shared variables associated with each condition wait that is true if the thread should proceed. Spurious wakeups from the `pthreadCondWait()` may occur. Since the return from `pthreadCondWait()` does not imply anything about the value of this predicate, the predicate should be re-evaluated upon each return.

The effect of using more than one mutex for concurrent `pthreadCondWait()` or `pthreadCondTimedWait()` operations on the same condition variable will result in `EINVAL` errors being returned. That is, a condition variable becomes bound to a unique mutex when a thread waits on the condition variable, and this dynamic binding ends when the last concurrent wait returns.

A condition wait is a cancellation point. When the cancelability enable state of a thread is set to `PTHREAD_CANCEL_DEFERRED`, a side effect of acting upon a cancellation request while in a condition wait is that the mutex is re-acquired before calling the first cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up to the point of returning from the call to `pthreadCondWait()`, but at that point notices the cancellation request and instead of returning to the caller of `pthreadCondWait()`, starts the thread cancellation activities, which includes calling cancellation cleanup handlers.

A thread that has been unblocked because it has been canceled while blocked in a call to `pthreadCondWait()` does not consume any condition signal that may be directed concurrently at the condition variable if there are other threads blocked on the condition variable.
pthread_cond_wait()

The Microware Pthread implementation supports the concept of interruption as it relates to condition variable waits. If a thread has a pending interruption or is interrupted while blocked, `pthread_cond_wait()` will return `EINTR`. The mutex will be re-acquired prior to return.

If successful, returns a value of 0; otherwise, returns an error.

⚠ This function contains a cancel point.

cond

is a pointer to the condition variable. (Input)

Attributes

Operating System: OS-9
State: User
Compatibility: POSIX

Library

`mt_clib.l`

Possible Errors

`EINVAL`

The value specified by `cond` or `mutex` is invalid. Different mutexes are supplied for concurrent `pthread_cond_wait()` or `pthread_cond_timedwait()` operations on the same condition variable. The mutex is not owned by the current thread at the time of the call.

`EINTR`

`pthread_interrupt()` was called with this thread as the target prior to or during this call.

See Also

`pthread_cond_broadcast()`
`pthread_cond_signal()`
`pthread_cond_timedwait()`

Example

```c
err = pthread_cond_wait(cond, mutex);
if (err != 0)
    fprintf(stderr, "failed to wait on condvar\n", strerror(err));
```
pthread_condattr_destroy()
Free Condition Variable Attributes Object

Syntax
#include <pthread.h>
int pthread_condattr_destroy(
    pthread_condattr_t *attr);

Description
pthread_condattr_destroy() destroys a condition variable attributes object; the object becomes, in effect, uninitialized.
If successful, returns a value of 0; otherwise, returns an error.

attr
    is a pointer to an initialized pthread attribute object. (Input)

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EINVAL
The value specified by attr is invalid.

See Also
pthread_cond_init()
pthread_condattr_init()

Example
err = pthread_condattr_destroy(attr);
if (err != 0)
    fprintf(stderr, “failed to destory condattr - %s\n”, strerror(err));
**pthread_condattr_getpshared()**

**Get Condition Variable Process-Shared Attribute**

### Syntax

```c
#include <pthread.h>

int pthread_condattr_getpshared(
    const pthread_condattr_t *attr,
    int *pshared);
```

### Description

`pthread_condattr_getpshared()` obtains the value of the process-shared attribute from the attributes object referenced by `attr`.

If successful, returns 0 and stores the value of the process-shared attribute of `attr` into the object referenced by the `pshared` parameter. Otherwise, an error number is returned.

- **attr** is a pointer to an initialized pthread attribute object. (Input)
- **pshared** is a pointer to the process-shared attribute. (Output)

This facility is not currently supported in Microware’s Pthreads implementation. The process-shared attribute can be changed, but both values behave like `PTHREAD_PROCESS_PRIVATE`. `_POSIX_THREAD_PROCESS_SHARED` is not currently defined.

### Attributes

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

### Library

`mt_clib.l`

### Possible Errors

- **EINVAL** The value specified by `attr` is invalid.

### See Also

- `pthread_condattr_init()`
- `pthread_condattr_setpshared()`
Example

```c
err = pthread_condattr_getpshared(attr, &pshare);
if (err != 0)
    fprintf(stderr, "failed to get pshared attribute - %s\n",
            strerror(err));
```
**pthread_condattr_init()**

Allocate Condition Variable Attributes Object

### Syntax

```c
#include <pthread.h>

int pthread_condattr_init(pthread_condattr_t *attr);
```

### Description

`pthread_condattr_init()` initializes a condition variable attributes object `attr` with the default value for all of the attributes.

If successful, returns a value of 0; otherwise, returns an error.

`attr` is a pointer to an initialized pthread attribute object. (Output)

The default values of the attributes are shown in Table 2-21.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process-shared</td>
<td>PTHREAD_PROCESS_PRIVATE</td>
</tr>
</tbody>
</table>

### Attributes

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

### Library

`mt_clib.l`

### Possible Errors

- **ENOMEM** Insufficient memory exists to initialize the condition variable attributes object.
- **EINVAL** `attr` is an invalid value.

### See Also

- `pthread_cond_init()`
- `pthread_condattr_destroy()`

### Example

```c
err = pthread_condattr_init(&condattr);
if (err != 0) {
    fprintf(stderr, “failed to init condattr - %s\n”, strerror(err));
}
```
pthread_condattr_setpshared()
Set Condition Variable Process-Shared Attribute

Syntax
#include <pthread.h>
int pthread_condattr_setpshared(
   pthread_condattr_t *attr,
   int pshared);

Description
pthread_condattr_setpshared() sets the process-shared attribute in an initialized attributes object referenced by attr. Valid values for pshared are PTHREAD_PROCESS_SHARED or PTHREAD_PROCESS_PRIVATE.

If successful, returns a value of 0; otherwise, returns an error.

attr
   is a pointer to an initialized pthread attribute object. (Input)

pshared
   is a pointer to the process-shared attribute. (Input)

This facility is not currently supported in the Microware Pthreads implementation. The process-shared attribute can be changed, but PTHREAD_PROCESS_SHARED behaves exactly like PTHREAD_PROCESS_PRIVATE. _POSIX_THREAD_PROCESS_SHARED is not currently defined.

Attributes
Operating System:    OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EINVAL The value specified by attr is invalid. The new value specified for the attribute is outside the range of legal values for that attribute.

See Also
pthread_condattr_init()
pthread_condattr_getpshared()
Example

err = pthread_condattr_setpshared(&condattr, PTHREAD_PROCESS_PRIVATE);
if (err != 0)
    fprintf(stderr, "failed to set to private - %s\n", strerror(err));
**Syntax**

```c
#include <pthread.h>

int pthread_create(
    pthread_t *thread,
    const pthread_attr_t *attr,
    void *(*start_routine) (void *),
    void *arg);
```

**Description**

`pthread_create()` is used to create a new thread, with attributes specified by `attr`, within a process. If `attr` is `NULL`, the default attributes are used. If the attributes specified by `attr` are modified later, the attributes of the thread are not affected. Upon successful completion, `pthread_create()` stores the ID of the created thread in the location referenced by `thread`.

The thread starts by executing `start_routine` with `arg` as its sole argument. If the `start_routine` returns, the effect is as if there was an implicit call to `pthread_exit()` using the return value of `start_routine` as the exit status.

If `pthread_create()` fails, no new thread is created, and the contents of the location referenced by `thread` are undefined.

The `pthread_attr` structure is used when threads are created. If successful, returns a value of 0; otherwise, returns an error.

- `attr` is a pointer to an initialized pthread attribute object. (Input)
- `thread` is a pointer to the target thread. (Output)
- `start_routine` is a pointer to the start thread routine. (Input)
  - The thread in which `main()` was originally invoked differs from this. When this thread returns from `main()`, the effect is as if there was an implicit call to `exit()` using the return value of `main()` as the exit status.
- `arg` is the sole argument for execution. (Input)
### Attributes
- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

### Library
*mt_clib.l*

### Possible Errors
- **EAGAIN**
  The system lacked the necessary resources to create another thread, or the system-imposed limit on the total number of threads in a process `PTHREAD_THREADS_MAX` would be exceeded.

- **EINVAL**
  The value specified by `attr` is invalid.

### See Also
- `_os_thfork()`
- `pthread_exit()`
- `pthread_join()`
- `pthread_detach()`

### Example
```c
err = pthread_create(&tid, &worker_attr, worker_loop, NULL);
if (err != 0)
  fprintf(stderr, "error creating worker - %s\n", strerror(err));
```
**pthread_detach()**

Orphan Target Thread

---

**Syntax**

```c
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

**Description**

`pthread_detach()` orphans the designated thread. Any thread within the caller's process can be detached unless it is already in detached state.

The `pthread_detach()` function is used to indicate that storage for the thread can be reclaimed when that thread terminates. If thread has not terminated, `pthread_detach()` does not cause it to terminate. Multiple `pthread_detach()` calls on the same target thread result in `EINVAL` being returned.

If successful, returns a value of 0; otherwise, returns an error.

`thread` is the target thread. (Input)

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **EINVAL** The value specified by `thread` does not refer to a thread that can be joined.
- **ESRCH** No thread could be found corresponding to that specified by the given thread ID.

**See Also**

`pthread_join()`

**Example**

```c
err = pthread_detach(io_thread);
if (err != 0)
    fprintf(stderr, "error detaching I/O thread - %s\n", strerror(err));
```
Syntax
#include <pthread.h>
int pthread_equal(pthread_t t1, pthread_t t2);

Description
pthread_equal() tests whether two thread IDs are the same.
pthread_equal() returns a nonzero value if the two thread IDs are equal; otherwise 0 is returned.
t1 and t2
are two thread IDs. (Input)

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
None

See Also
pthread_self()

Example
if (pthread_equal(worker[0], dead))
    fputs("worker #0 died", stderr);

pthread_equal()
Compare Thread Identifiers
# pthread_exit()

## Terminate Thread

### Syntax

```c
#include <pthread.h>

void pthread_exit(void *value_ptr);
```

### Description

`pthread_exit()` terminates the calling thread. If any thread is waiting on a join on this thread, they are released and passed `value_ptr` as the exit status.

`pthread_exit()` terminates the calling thread and makes the value `value_ptr` available to any successful join with the terminating thread. Any cancellation cleanup handlers that have been pushed and not yet popped, shall be popped in the reverse order that they were pushed and then executed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, appropriate destructor functions are called in an unspecified order. Thread termination does not release any application visible process resources (e.g. allocated memory, open paths, etc.). Nor does it perform any process level cleanup actions like calling any `atexit()` routines that may exist.

An implicit call to `pthread_exit()` is made when a thread other than the thread in which `main()` was first invoked returns from the start routine that was used to create it. The return value of the function serves as the exit status of the thread.

`pthread_exit()` returns immediately without doing anything if called from a cancellation cleanup handler or destructor function that was invoked as a result of either an implicit or explicit call to `pthread_exit()`.

After a thread has terminated, the result of access to local (auto) variables of the thread is undefined. Thus, references to local variables of the exiting thread should not be used for the `pthread_exit()` `value_ptr` parameter value.

The process exits with an exit status of 0 after the last thread has been terminated. The behavior is as if the implementation called `exit()` with a zero argument at the time of thread termination.

Calling `pthread_exit` from the thread in which `main` was first invoked does not necessarily cause the process to exit. The process will continue to run until all threads have terminated or an exit call is made.

`value_ptr`

is a pointer to the value. (Input)

### Attributes

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX
Library

mt_clib.l

Possible Errors

None

See Also

exit()

pthread_create()

pthread_join()

pthread_detach()

Example

pthread_exit((void *)SUCCESS);
pthread_getspecific()
Get Thread-Specific Data Pointer

Syntax
#include <pthread.h>
void *pthread_getspecific(pthread_key_t key);

Description
pthread_getspecific() returns the value currently bound to the specified key on behalf of the calling thread.

pthread_getspecific() returns the thread-specific data value associated with the given key. If no thread-specific data value is currently associated with key, then the value NULL is returned.

key is the key associated with the calling thread. (Input)

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
None

See Also
pthread_key_create()
pthread_setspecific()

Example
thread_data = pthread_getspecific(thread_data_key);
if (thread_data == NULL) {
    thread_data = malloc(sizeof(thread_data_t));
    if (thread_data == NULL)
        fprintf(stderr, "memory allocation error - %s\n", strerror(errno));
    pthread_setspecific(thread_data_key, thread_data);
}
**Syntax**

```c
#include <pthread.h>

int _pthread_getstatus(
    pthread_t          thread,
    _pthread_status_t  *status);
```

**Description**

 `_pthread_getstatus()` returns various pieces of information related to the target thread in the structure pointed to by `status`.

If successful, returns a value of 0; otherwise, returns an error.

`thread` is the target thread. (Input)

`status` is a pointer to the status. (Output)

The following table describes the fields of the `_pthread_status_t` structure:

**Table 2-22. `_pthread_status_t` Structure Fields**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u_int32 (bit masks follow)</td>
<td>status</td>
<td>Bits for various boolean pieces of information:</td>
</tr>
<tr>
<td>_PT_DETACHED</td>
<td></td>
<td>0 = joinable thread</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = detached thread</td>
</tr>
<tr>
<td>_PT_EXIT</td>
<td></td>
<td>0 = thread has terminated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = thread has not yet terminated</td>
</tr>
<tr>
<td>_PT_CSTATE</td>
<td></td>
<td>0 = cancels enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = cancels disabled</td>
</tr>
<tr>
<td>_PTCTYPE</td>
<td></td>
<td>0 = deferred cancels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = asynchronous cancels</td>
</tr>
<tr>
<td>_PTCPENDING</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2-22. _pthread_status_t Structure Fields (Continued)

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = no cancel request pending</td>
<td>_PT_SSTATE</td>
<td>0 = suspendable</td>
</tr>
<tr>
<td>1 = cancel request pending</td>
<td></td>
<td>1 = unsuspendable</td>
</tr>
<tr>
<td>0 = no suspend request pending</td>
<td>_PT_SPENDING</td>
<td>0 = no suspend request pending</td>
</tr>
<tr>
<td>1 = suspend request pending</td>
<td></td>
<td>1 = suspend request pending</td>
</tr>
<tr>
<td>0 = not suspended</td>
<td>_PT_SFLAG</td>
<td>0 = not suspended</td>
</tr>
<tr>
<td>1 = suspended</td>
<td></td>
<td>1 = suspended</td>
</tr>
<tr>
<td>0 = not priority boosted</td>
<td>_PT_BOOSTED</td>
<td>0 = not priority boosted</td>
</tr>
<tr>
<td>1 = priority boosted</td>
<td></td>
<td>1 = priority boosted</td>
</tr>
<tr>
<td>0 = no interruption pending</td>
<td>_PT_IPENDING</td>
<td>0 = no interruption pending</td>
</tr>
<tr>
<td>1 = interruption pending</td>
<td></td>
<td>1 = interruption pending</td>
</tr>
</tbody>
</table>

| thread_t | tid | OS-9 thread identifier                               |
| thread_t | creator | OS-9 thread identifier of thread's creator         |
| void *   | stack | stack base (highest address)                        |
| size_t   | stack_size | Stack size in bytes                             |
| u_int16  | priority | Thread’s priority                                 |
| u_int16  | bpriority | Thread’s boosted priority                         |
| u_int32 [2] | resv | Reserved space for future additional status information |

Attributes

Operating System: OS-9
State: User

Library

mt_clib.l
Possible Errors

EINVAL          The passed thread or status pointer is NULL.
ESRCH           The specified target thread is not valid.

Example

err = _pthread_getstatus(child, &stats);
if (err != 0)
    fprintf(stderr, "failed to get status for child - %s", strerror(err));
printf("child's OS-9 thread ID is %u\n", stats.tid);
**_pthread_global_slot()**

**Read or Write Global Variables**

**Syntax**

```c
u_int32    *_pthread_global_slot(
    int32 slot)
```

**Description**

This function is used when reading or writing global versions of “core” C run-time variables. `_mainid`, the process ID of a thread’s host process, is the only example of such a variable.

`_pthread_global_slot()` is used to get the address of the global version of `_mainid`. 

`_pthread_global_slot()` returns the address of the storage for a specific slot number. This makes it equally easy to read or write the variable. In addition, `_pthread_global_slot()` automatically saves and restores any modified registers except the return value. This makes it easier to call from assembly language.

A module’s global data pointer and `_pthread` value must be valid prior to calling `_pthread_global_slot()`.

`slot` is the slot number. (Input)

Slot numbers are defined in `MKOS/SRC/DEFS/pthread.h`. Once a slot number has been assigned to a variable, it will not change in a subsequent release.

**Attributes**

Operating System: OS-9

State: User

**Library**

`mt_clib.l`
pthread_interrupt()
Interrupt Target Thread

Syntax

```c
#include <pthread.h>

int _pthread_interrupt(pthread_t thread);
```

Description

pthread_interrupt() interrupts any pthread_cond_wait() or pthread_cond_timedwait() being done by the specified thread. If the thread is not currently blocked in pthread_cond_wait() or pthread_cond_timedwait(), _pthread_interrupt() makes the interruption pending.

pthread_interrupt() is implemented as if the target thread can atomically check for a pending interrupt and then block in pthread_cond_timedwait() or pthread_cond_wait() if none is pending. That is, there is no window between when a thread checks for a pending interrupt and when the thread actually blocks where an interruption request could be missed.

If successful, returns a value of 0; otherwise, returns an error.

thread is the target thread. (Input)

Attributes

Operating System: OS-9
State: User

Library

mt_clib.l

Possible Errors

EINVAL  thread is invalid.
ESRCH   thread is not a valid thread.

See Also

pthread_cond_timedwait()  pthread_cond_wait()
(pthread_getstatus()  pthread_interrupt_clear()

Example

```c
err = _pthread_interrupt(wait_thread);
if (err != 0)
    fprintf(stderr, "failed to interrupt waiter - %s\n", strerror(err));
```
_pthread_interrupt_clear()

Clear Interrupt Request for Target Thread

Syntax

```c
#include <pthread.h>

int _pthread_interrupt_clear(
    pthread_t   thread,
    int         *old_status);
```

Description

(pthread_interrupt_clear) clears any pending interrupt for the specified thread. This function might be useful if other interruptible operations are defined for a particular application. Refer to _pthread_getstatus() for more information on determining if a particular thread has an interruption pending.

If successful, returns a value of 0; otherwise, returns an error.

thread  
is the target thread. (Input)

old_status  
The value of the interruption status for the target thread is returned at the integer pointed to by old_status. (Output)

If the old status is not required, NULL may be passed for old_status.

Attributes

Operating System: OS-9

State: User

Library

mt_clib.l

Possible Errors

EINVAL  
thread is invalid.

ESRCH  
thread is not a valid thread.

See Also

pthread_cond_timedwait()  
pthread_cond_wait()

(pthread_interrupt)  
(pthread_getstatus)

Example

```c
err = _pthread_interrupt_clear(pthread_self());
if (err != 0)
    fprintf(stderr, “failed to clear interruption - %s\n”, strerror(err));
```
**pthread_join()**  
Wait for Target Thread to Terminate

**Syntax**

```c
#include <pthread.h>
int pthread_join(
    pthread_t thread,
    void **value_ptr);
```

**Description**

The `pthread_join()` function suspends execution of the calling thread until the target thread terminates, unless the target thread has already terminated. On return from a successful `pthread_join()` call with a non-NULL `value_ptr` (output) argument, the value passed to `pthread_exit()` by the terminating thread (input) is stored in the location referenced by `value_ptr`.

When a `pthread_join()` returns successfully, the target thread has been terminated. Multiple simultaneous calls to `pthread_join()` specifying the same target thread results in one thread successfully getting the exit status and the remainder getting `EOS_NOCHLD` as the result of `pthread_join()`.

Exited but remaining unjoined threads count against the maximum number of threads a process may have, `PTHREAD_THREADS_MAX`.

If successful, returns a value of 0; otherwise, returns an error.

This function contains a cancel point.

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **EINVAL** The value specified by `thread` does not refer to a thread that can be joined.
- **ESRCH** No thread could be found corresponding to that specified by the given thread ID.
- **EDEADLK** A deadlock was detected, or the value of `thread` specifies the calling thread.
See Also

pthread_create()
pthread_detach()
pthread_exit()

Example

err = pthread_join(child, &status);
if (err != 0)
    fprintf(stderr, “error waiting for child - %s\n”, strerror(err));
printf(“Child’s exit status was %u\n”, status);
### Syntax

```c
u_int32  *pthread_local_slot(
    int32 slot)
```

### Description

This function is used when reading or writing thread-specific versions of “core” C run-time variables. `errno` is an example of a local slot. For threaded applications, there exists one `errno` per thread. `_pthread_local_slot()` is used to get the address of the calling thread’s version of `errno`.

`_pthread_local_slot()` returns the address of the storage for a specific slot number. This makes it equally easy to read or write the variable. In addition, `_pthread_local_slot()` automatically saves and restores any modified registers except the return value. This makes it easier to call from assembly language.

This might be used in the dispatcher during function exit to copy the version of `errno` generated by the code within a subroutine module back to the application’s version of `errno`.

A module’s global data pointer and `_pthread` value must be valid prior to calling `_pthread_local_slot()`.

- **slot** is the slot number. (Input)

  Slot numbers are defined in `MKOS/SRC/DEFS/pthread.h`. Once a slot number has been assigned to a variable, it will not change in a subsequent release.

### Attributes

- **Operating System:** OS
- **State:** User

### Library

`mt_clib.l`
pthread_key_create()
Create Thread-Specific Data Key

Syntax
#include <pthread.h>
int pthread_key_create(
    pthread_key_t *key,
    void (*destructor) (void *));

Description
pthread_key_create() creates a thread-specific data key visible to all threads in the process. Key values provided by pthread_key_create() are opaque objects used to locate thread-specific data. Although the same key value may be used by different threads, the values bound to the key by pthread_setspecific() are maintained on a per-thread basis and persist for the life of the calling thread.

If successful, pthread_key_create() stores the newly created key value at *key and returns 0. Otherwise, an error number is returned.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EAGAIN The system lacked the necessary resources to create another thread-specific data key, or the limit on the total number of keys per process, PTHREAD_KEYS_MAX, has been exceeded.
EINVAL The key value is invalid.
ENOMEM Insufficient memory exists to create the key.

See Also
pthread_getspecific()
pthread_key_delete()
pthread_setspecific()

Example
err = pthread_key_create(&thread_data_key, free_thread_data);
if (err != 0)
    fprintf(stderr, "failed to create key - %s\n", strerror(err));
pthread_key_delete()
Delete Thread-Specific Data Key

Syntax
#include <pthread.h>
int pthread_key_delete(pthread_key_t key);

Description
pthread_key_delete() deletes a thread-specific data key (input) previously returned by pthread_key_create(). The thread-specific data values associated with key need not be NULL at the time pthread_key_delete() is called. It is the responsibility of the application to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after pthread_key_delete() is called.

If successful, returns a value of 0; otherwise, returns an error.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EINVAL The key value is invalid.

See Also
pthread_key_create()
pthread_getspecific()
pthread_setspecific()

Example
err = pthread_key_delete(thread_data_key);
if (err != 0)
    fprintf(stderr, “error deleting key - %s\n”, strerror(err));
pthread_mutex_destroy() Free Mutex Object

Syntax
#include <pthread.h>
int pthread_mutex_destroy(pthread_mutex_t *mutex);

Description
pthread_mutex_destroy() destroys the mutex object referenced by mutex (input);
the mutex object becomes, in effect, uninitialized. If successful, returns a value of 0;
otherwise, returns an error.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EBUSY Attempt to destroy the object referenced by mutex while it is
locked or referenced. For example, while being used in a
pthread_cond_wait() or pthread_cond_timedwait() by
another thread.

EINVAL The value specified by mutex is invalid.

See Also
pthread_mutex_init()

Example
err = pthread_mutex_destroy(mutx);
if (err != 0)
    fprintf(stderr, "error destroying mutex - %s\n", strerror(err));
# pthread_mutex_getprioceiling()

## Get Mutex Priority Ceiling

### Syntax

```c
#include <pthread.h>
int pthread_mutex_getprioceiling(
    const pthread_mutex_t      *mutex,
    int                        *prioceiling);
```

### Description

`pthread_mutex_getprioceiling()` obtains the value of the priority ceiling value from the mutex object referenced by `mutex` (input).

The value stored at `prioceiling` (output) will be the current value of the priority ceiling for the mutex. Valid priority ceilings are in the range 0 to 65535 (0xffff).

If successful, returns 0 and stores the value of the priority ceiling of `mutex` into the integer referenced by the `prioceiling` parameter. Otherwise, returns an error number.

### Attributes

- Operating System: OS-9
- State: User
- Compatibility: POSIX

### Library

`mt_clib.l`

### Possible Errors

- `EINVAL` The value specified by `mutex` is invalid.

### See Also

- `pthread_mutexattr_init()`
- `pthread_mutex_setprioceiling()`

### Example

```c
err = pthread_mutex_getprioceiling(mutex, &pc);
if (err != 0)
    fprintf(stderr, "error getting priority ceiling - %s\n",
            strerror(err));
```
pthread_mutex_init()

Allocate Mutex Object

Syntax
#include <pthread.h>

int pthread_mutex_init(
    pthread_mutex_t    *mutex,
    const pthread_mutexattr_t  *attr);

Description
The pthread_mutex_init() function initializes the mutex referenced by mutex with attributes specified by attr.
If successful, returns a value of 0; otherwise, returns an error.

mutex
    is a pointer to the mutex. (Input)

attr
    is a pointer to an initialized pthread attribute object. (Input)

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EAGAIN  The system lacked the necessary resources (other than memory) to initialize another mutex.
EBUSY   An attempt to reinitialize the object referenced by mutex (a previously initialized, but not yet destroyed, mutex).
EINVAL  The value specified by attr or mutex is invalid.

See Also
pthread_mutex_lock()  pthread_mutex_trylock()
pthread_mutex_unlock()  pthread_mutex_destroy()

Example
err = pthread_mutex_init(&glob_mutex, NULL);
if (err != 0)
    fprintf(stderr, “error initializing mutex - %s\n”, strerror(err));
Syntax
#include <pthread.h>
int pthread_mutex_lock(pthread_mutex_t *mutex);

Description
The mutex object referenced by \texttt{mutex} (input) is locked by calling \texttt{pthread_mutex_lock()}. If the mutex is already locked, the calling thread blocks until the mutex becomes available. This operation returns with the mutex object referenced by \texttt{mutex} in the locked state with the calling thread as its owner. An attempt by the current owner of a mutex to relock the mutex results in an \texttt{EDEADLK} error.

If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread resumes waiting for the mutex as if it was not interrupted. If priority inheritance is enabled for the specified mutex and a thread with a lower priority already owns the mutex then the owning thread's priority will be raised to the level of calling thread.

After the lock is acquired, if priority protection is enabled for the specified mutex and the specified ceiling priority is greater than the thread's current priority the thread's priority will be raised.

If successful, returns a value of 0; otherwise, returns an error.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
\texttt{mt_clib.l}

Possible Errors
\begin{itemize}
\item \texttt{EINVAL} The value specified by \texttt{mutex} does not refer to an initialized mutex object.
\item \texttt{EDEADLK} The current thread already owns the mutex.
\end{itemize}

See Also
\texttt{pthread_mutex_trylock()} \hspace{1cm} \texttt{pthread_mutex_unlock()}

Example
\begin{verbatim}
err = pthread_mutex_lock(&glob_mutex);
if (err != 0)
    fprintf(stderr, “error locking mutex - %s
", strerror(err));
\end{verbatim}
pthread_mutex_setprioceiling()

Set Mutex Priority Ceiling

Syntax
#include <pthread.h>

int pthread_mutex_setprioceiling(
    pthread_mutex_t      *attr,
    int                  ceiling);

Description
pthread_mutex_setprioceiling() is used to set the priority ceiling value in an initialized mutex object referenced by mutex.

If successful, returns a value of 0; otherwise, returns an error.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EINVAL The value specified by mutex is invalid. The new value specified for the attribute is outside the range of legal values for that attribute.

See Also
pthread_mutexattr_init()
pthread_mutex_getprioceiling()

Example
err = pthread_mutex_setprioceiling(mutex, 255);
if (err != 0)
    fprintf(stderr, “error setting priority ceiling - %s\n”,
            strerror(err));
pthread_mutex_trylock()

Lock Mutex Object (Non-Blocking)

**Syntax**

```c
#include <pthread.h>
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

**Description**

`pthread_mutex_trylock()` is a non-blocking mutex lock operation. If `mutex` (input) is currently unowned, the calling thread is made the owner. If `mutex` is currently owned (by any thread, including the calling thread), `EBUSY` is returned.

Returns 0 if a lock on the mutex object referenced by `mutex` is acquired; otherwise, returns an error number.

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **EBUSY**: The mutex could not be acquired because it was already locked.
- **EINVAL**: The value specified by `mutex` does not refer to an initialized mutex object.

**See Also**

- `pthread_mutex_lock()`
- `pthread_mutex_unlock()`

**Example**

```c
err = pthread_mutex_trylock(&glob_mutex);
if (err != 0 && err != EBUSY)
    fprintf(stderr, "error trying to lock glob_mutex - %s\n",
            strerror(err));
```
pthread_mutex_unlock()
Unlock Mutex Object

Syntax
#include <pthread.h>
int pthread_mutex_unlock(pthread_mutex_t *mutex);

Description
pthread_mutex_unlock() is called by the owner of the mutex object referenced by mutex (input) to release it. A pthread_mutex_unlock() call by a thread that is not the owner of the mutex results in an EPERM error. Calling pthread_mutex_unlock() when the mutex object is unlocked also results in an EPERM error.

If there are threads blocked on the mutex object referenced by mutex when pthread_mutex_unlock() is called, the mutex becomes available, and is given to the next waiting thread.

If successful, returns a value of 0; otherwise, returns an error.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EINVAL The value specified by mutex does not refer to an initialized mutex object.
EPERM The current thread does not own the mutex.

See Also
pthread_mutex_lock()
pthread_mutex_trylock()

Example
err = pthread_mutex_unlock(&glob_mutex);
if (err != 0)
    fprintf(stderr, “error unlocking glob_mutex - %s\n”, strerror(err));
**pthread_mutexattr_destroy()**

Free Mutex Attributes Object

**Syntax**

```c
#include <pthread.h>

int pthread_mutexattr_destroy(
    pthread_mutexattr_t *attr);
```

**Description**

`pthread_mutexattr_destroy()` destroys a mutex attributes object; the object becomes, in effect, uninitialized.

If successful, returns a value of 0; otherwise, returns an error.

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **EINVAL** The value specified by `attr` is invalid.

**See Also**

- `pthread_mutex_init()`
- `pthread_mutexattr_init()`

**Example**

```c
err = pthread_mutexattr_destroy(mutex_attr);
if (err != 0)
    fprintf(stderr, "error destroying attr - %s\n", strerror(err));
```
**pthread_mutexattr_getprioceiling()**

Get Priority Ceiling Attribute

**Syntax**

```c
#include <pthread.h>
int pthread_mutexattr_getprioceiling(
    const pthread_mutexattr_t *attr,
    int *prioceiling);
```

**Description**

`pthread_mutexattr_getprioceiling()` obtains the value of the priority ceiling attribute from the mutex attributes object referenced by `attr` (input).

The value stored at `prioceiling` (output) will be the current value of the priority ceiling attribute. Valid priority ceilings are in the range 0 to 65535 (0xffff).

If successful, returns 0 and stores the value of the priority ceiling attribute of `attr` into the integer referenced by the `prioceiling` parameter. Otherwise, returns an error number.

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **EINVAL** The value specified by `attr` is invalid.

**See Also**

- `pthread_mutexattr_init()`
- `pthread_mutexattr_setprioceiling()`

**Example**

```c
er = pthread_mutexattr_getprioceiling(mutex_attr, &pc);
if (err != 0)
    fprintf(stderr, “error getting priority ceiling - %s\n”,
            strerror(err));
```
# pthread_mutexattr_getprotocol()

## Get Protocol Attribute

### Syntax

```c
#include <pthread.h>

int pthread_mutexattr_getprotocol(
    const pthread_mutexattr_t *attr,
    int *protocol);
```

### Description

`pthread_mutexattr_getprotocol()` obtains the value of the protocol attribute from the mutex attributes object referenced by `attr` (input).

The value stored at `protocol` will be one of `PTHREAD_PRIO_NONE`, `PTHREAD_PRIO_INHERIT`, or `PTHREAD_PRIO_PROTECT`.

If successful, returns 0 and stores the value of the protocol attribute of `attr` into the integer referenced by the `protocol` (output) parameter. Otherwise, returns an error number.

### Attributes
- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

### Library

`mt_clib.l`

### Possible Errors

- **EINVAL**: The value specified by `attr` is invalid.

### See Also

- `pthread_mutexattr_init()`
- `pthread_mutexattr_setprotocol()`

### Example

```c
err = pthread_mutexattr_getprotocol(mutex_attr, &prot);
if (err != 0)
    fprintf(stderr, "error getting protocol - %s\n", strerror(err));
```
pthread_mutexattr_getpshared()

Get Mutex Process-Shared Attribute

Syntax

```
#include <pthread.h>

int pthread_mutexattr_getpshared(
    const pthread_mutexattr_t *attr,
    int *pshared);
```

Description

`pthread_mutexattr_getpshared()` obtains the value of the process-shared attribute from the attributes object referenced by `attr` (input).

The value stored at `pshared` will be either `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

If successful, returns 0 and stores the value of the process-shared attribute of `attr` into the object referenced by the `pshared` (output) parameter. Otherwise, returns an error number.

This facility is not currently supported in Microware’s Pthreads implementation. The process-shared attribute can be changed, but both values behave like `PTHREAD_PROCESS_PRIVATE`. `_POSIX_THREAD_PROCESS_SHARED` is not currently defined.

Attributes

- Operating System: OS-9
- State: User
- Compatibility: POSIX

Library

`mt_clib.l`

Possible Errors

- `EINVAL`: The value specified by `attr` is invalid.

See Also

- `pthread_mutexattr_init()`
- `pthread_mutexattr_setpshared()`

Example

```
err = pthread_mutexattr_getpshared(mutex_attr, &pshared);
if (err != 0)
    fprintf(stderr, “error getting pshared - %s
”, strerror(err));
```
Allocate Mutex Attributes Object

Syntax

```c
#include <pthread.h>
int pthread_mutexattr_init(
    pthread_mutexattr_t *attr);
```

Description

`pthread_mutexattr_init()` initializes a mutex attributes object `attr` with a default value for all of the attributes.

Returns 0 if successful or an error code if unsuccessful.

- `attr` is a pointer to an initialized pthread attribute object. (Input)
- The default values for the attributes are shown in Table 2-23.

### Table 2-23. Default Attribute Values for Mutex

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process-shared</td>
<td>PTHREAD_PROCESS_PRIVATE</td>
</tr>
<tr>
<td>Protocol</td>
<td>PTHREAD_PRIO_NONE</td>
</tr>
<tr>
<td>Priority Ceiling</td>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

Attributes

- Operating System: OS-9
- State: User
- Compatibility: POSIX

Library

`mt_clib.l`

Possible Errors

- **EINVAL** The key value is invalid.
- **ENOMEM** Insufficient memory exists to initialize the mutex attributes object.

See Also

`pthread_mutex_init()`  `pthread_mutexattr_destroy()`

Example

```c
err = pthread_mutexattr_init(mutex_attr);
if (err != 0)
    fprintf(stderr, "error initializing attr - %s\n", strerror(err));
```
**pthread_mutexattr_setprioceiling()**

## Syntax

```c
#include <pthread.h>

int pthread_mutexattr_setprioceiling(
    pthread_mutexattr_t  *attr,
    int                  ceiling);
```

## Description

`pthread_mutexattr_setprioceiling()` is used to set the priority ceiling attribute in an initialized attributes object referenced by `attr` (input).

`ceiling` (output) must be a valid OS-9 priority value; it must be in the range 0 to 65535 (0xffff).

If successful, returns a value of 0; otherwise, returns an error.

## Attributes

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

## Library

`mt_clib.l`

## Possible Errors

- **EINVAL**
  
  The value specified by `attr` is invalid. The new value specified for the attribute is outside the range of legal values for that attribute.

## See Also

- `pthread_mutexattr_init()`
- `pthread_mutexattr_getprioceiling()`

## Example

```c
err = pthread_mutexattr_setprioceiling(mutex_attr, 255);
if (err != 0)
    fprintf(stderr, “error setting priority ceiling - %s\n”,
            strerror(err));
```
pthread_mutexattr_setprotocol()
Set Protocol Attribute

Syntax
#include <pthread.h>
int pthread_mutexattr_setprotocol(
    pthread_mutexattr_t  *attr,
    int                  protocol);

Description
pthread_mutexattr_setprotocol() is used to set the protocol attribute in an
initialized attributes object referenced by attr (input).
protocol (output) must be either PTHREAD_PRIO_NONE, PTHREAD_PRIO_INHERIT, or
PTHREAD_PRIO_PROTECT.
If successful, returns a value of 0; otherwise, returns an error.

Attributes
Operating System: OS-9
State: User
Compatibility: POSIX

Library
mt_clib.l

Possible Errors
EINVAL The value specified by attr is invalid. The new value specified
for the attribute is outside the range of legal values for that
attribute.

See Also
pthread_mutexattr_init()
pthread_mutexattr_getprotocol()

Example
err = pthread_mutexattr_setprotocol(mutex_attr, param);
if (err != 0)
    fprintf(stderr, “error setting protocol attr - %s
”, strerror(err));
**Syntax**

```c
#include <pthread.h>

int pthread_mutexattr_setpshared(
    pthread_mutexattr_t *attr,
    int pshared);
```

**Description**

`pthread_mutexattr_setpshared()` is used to set the process-shared attribute in an initialized attributes object referenced by `attr` (input).

`pshared` (input) must be either `PTHREAD_PROCESS_SHARED` or `PTHREAD_PROCESS_PRIVATE`.

If successful, returns a value of 0; otherwise, returns an error.

This facility is not currently supported in Microware’s Pthreads implementation. The process-shared attribute can be changed, but both values behave like `PTHREAD_PROCESS_PRIVATE`. `_POSIX_THREAD_PROCESS_SHARED` is not currently defined.

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

`EINVAL` The value specified by `attr` is invalid. The new value specified for the attribute is outside the range of legal values for that attribute.

**See Also**

`pthread_mutexattr_init()`

`pthread_mutexattr_getpshared()`

**Example**

```c
err = pthread_mutexattr_setpshared(mutex_attr, PTHREAD_PROCESS_PRIVATE);
if (err != 0)
    fprintf(stderr, “error setting pshared attr - %s\n”, strerror(err));
```
**pthread_once()**

Execute Routine Once per Process

**Syntax**

```c
#include <pthread.h>

int pthread_once(  
    pthread_once_t *once_control,  
    void (*init_routine) (void));
```

**Description**

The first call to `pthread_once()` by any thread in a process with a given `once_control` (input) calls the `init_routine()` with no arguments. Subsequent calls of `pthread_once()` (input) with the same `once_control` will not call the `init_routine()`. On return from `pthread_once()` by any thread, it is guaranteed that `init_routine()` has completed. The `once_control` parameter is used to determine whether the associated initialization routine has been called.

`pthread_once()` is not a cancellation point. However, if `init_routine()` is a cancellation point and is canceled, the effect on `once_control` is as if `pthread_once()` was never called.

The behavior of `pthread_once()` is undefined if `once_control` has automatic storage duration or is not initialized by `PTHREAD_ONCE_INIT`.

If successful, returns a value of 0; otherwise, returns an error.

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- **EINVAL**
  - `once` is an invalid pointer to a `pthread_once_t` object. `once` does not point to an initialized object. `init_routine` is an invalid address.

**Example**

```c
err = pthread_once(get_key_once, create_data_key);  
if (err != 0)  
    fprintf(stderr, "error creating data key - %s\n", strerror(err));
```
**_pthread_resume()**

**Decrement Suspension Counter**

**Syntax**

```c
#include <LIB/pthread.h>
int _pthread_resume(
    pthread_t thread,
    int *status);
```

**Description**

`_pthread_resume()` decrements the suspension counter for the specified target thread. The suspension status of the target thread is returned at the `int` pointed to by `status`. The `int` is as follows:

- 0 if the target thread was not suspended
- 1 if the target thread went from suspended to not suspended
- > 1 if the target thread remained suspended

A suspension counter is used to support multiple suspension requests with the same target thread. An equal number of resume requests must be made for the thread to continue execution.

If successful, returns a value of 0; otherwise, returns an error.

- `thread` is the target thread. (Input)
- `status` is a pointer to the suspension status of the target thread. (Output)

**Attributes**

- Operating System: OS-9
- State: User

**Library**

`mt_clib.l`

**Possible Errors**

- `EINVAL` thread or status is NULL.
- `ESRCH` thread is not a valid thread.

**See Also**

- `_pthread_setsuspendable()`
- `_pthread_setunsuspendable()`
- `_pthread_suspend()`
Example

err = pthread_resume(worker, &level);
if (err != 0)
    fprintf(stderr, "error resuming worker - %s\n", strerror(err));
**Syntax**

```
#include <pthread.h>
pthread_t pthread_self(void);
```

**Description**

`pthread_self()` returns the calling thread’s thread ID.

**Attributes**

- **Operating System:** OS-9
- **State:** User
- **Compatibility:** POSIX

**Library**

`mt_clib.l`

**Possible Errors**

None

**See Also**

`pthread_equal()`

**Example**

```c
if (pthread_self() == worker[0])
    fputs("thread is worker #0\n", stdout);
```
**pthread_setcancelstate()**

**Set Cancel State**

**Syntax**

```c
#include <pthread.h>
int pthread_setcancelstate(
    int state,
    int *oldstate);
```

**Description**

`pthread_setcancelstate()` sets a thread’s cancel state. `state` (input) can be either `PTHREAD_CANCEL_ENABLE` or `PTHREAD_CANCEL_DISABLE`. The previous value of the thread’s cancel state is returned at `oldstate` (output).

Any cancel requests made against a thread while its state is `PTHREADCANCEL_DISABLE` will be held pending until the state is changed to `PTHREAD_CANCEL_ENABLE`.

If successful, returns a value of 0; otherwise, returns an error.

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

- `EINVAL` state is neither `PTHREAD_CANCEL_ENABLE` nor `PTHREAD_CANCEL_DISABLE`. `oldstate` is an invalid address.

**See Also**

- `pthread_setcanceltype()`
- `pthread_cancel()`
- `pthread_cleanup_push()`
- `pthread_cleanup_pop()`
- `pthread_testcancel()`

**Example**

```c
int err = pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &oldstate);
if (err != 0)
    fprintf(stderr, “error setting cancel state - %s\n”, strerror(err));
```
# pthread_setcanceltype()

## Set Cancel Type

### Syntax

```c
#include <pthread.h>
int pthread_setcanceltype(
    int type,
    int *oldtype);
```

### Description

`pthread_setcanceltype()` sets a thread's cancel type. `type` (input) can be either `PTHREAD_CANCEL_DEFERRED` or `PTHREAD_CANCEL_ASYNCHRONOUS`. The previous value of the thread's cancel type is returned at `oldtype` (output).

When a thread's cancel type is `PTHREAD_CANCEL_DEFERRED` cancel requests against it wait to take effect until the next call to `pthread_testcancel()`. When a thread's cancel type is `PTHREAD_CANCEL_ASYNCHRONOUS` cancel requests are acted upon when they are made. That is, when a thread calls `pthread_cancel()` with a target thread that has cancellation enabled and asynchronous, the target thread will immediately cancel.

If successful, returns a value of 0; otherwise, returns an error.

Cancelling an asynchronous cancel type thread is guaranteed to cause a loss of resources. For example, the memory allocated to implement thread safety for C library functions will be lost. Use deferred cancellation whenever possible.

### Attributes

- **Operating System**: OS-9
- **State**: User
- **Compatibility**: POSIX

### Library

`mt_clib.l`

### Possible Errors

- `EINVAL` type is neither `PTHREAD_CANCEL_DEFERRED` nor `PTHREADCANCEL_ASYNCHRONOUS`.

### See Also

- `pthread_setcancelstate()`  
- `pthread_cancel()`  
- `pthread_cleanup_push()`  
- `pthread_cleanup_pop()`  
- `pthread_testcancel()`

### Example

```c
err = pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, &oldtype);
if (err != 0)
    fprintf(stderr, "error setting cancel type - %s\n", strerror(err));
```
**_pthread_setpr()**

Set Priority for Target Thread

**Syntax**

```c
#include <pthread.h>

int _pthread_setpr(
    pthread_t thread,
    u_int32 priority);
```

**Description**

 `_pthread_setpr()` sets the OS-9 priority of `thread` (input) to `priority` (input). This call must be used by threaded applications instead of `_os_setpr()` to ensure that priority inversion avoidance is properly supported for mutexes. Calling `_os_setpr()` directly results in undefined behavior as it relates to priority inversion.

Use `_pthread_getstatus()` to determine the priority of a thread.

If successful, returns a value of 0; otherwise, returns an error.

**Attributes**

- Operating System: OS-9
- State: User

**Library**

`mt_clib.l`

**Possible Errors**

- `EINVAL` thread is invalid. `priority` is out of range. Valid range of `priority` is 0-65538.
- `ESRCH` thread is an invalid thread ID.

**See Also**

- `_pthread_getstatus()`
- `pthread_mutex_destroy()`
- `pthread_mutex_destroy()`
- `_pthread_attr_setpriority()`
- `_pthread_attr_getpriority()`

**Example**

```c
err = _pthread_setpr(reactor_shutdown, HIGH_PRIORITY);
if (err != 0)
    fprintf(stderr, "failed to set priority - %s", strerror(err));
```
Set Range of Signal Values

Syntax

#include <pthread.h>

int pthread_setsignalrange(
    signal_code low,
    signal_code high);

Description

pthread_setsignalrange() is used to specify the set of signal values that the
Pthread library uses internally. Using this function will cause the Pthread library to
use signals in the range low to (high - 1).

Use this function if your application uses the same set of signal values as the Pthread
library. By default, the Pthread library will use signals in the range 40,000 to 49,999
inclusive.

A minimum of 1000 signal values must be specified. The Pthreads library uses
about 5 signals per thread as well as 1 per timed condition variable wait.

The new set of signals may not overlap the current set of signal values. This is to
ensure integrity of any already allocated signal numbers.

pthread_setsignalrange() returns 0 if successful or an error code if not.

low and high
    are the low and high ranges of signals used in the Pthread library. (Input)

Attributes

Operating System:    OS-9
State:               User

Library

mt_clib.l

Possible Errors

EINVAL If less than 1000 signal values are in the range or high is less
than low or the specified range overlaps with the signal range
currently in use.

Example

err = pthread_setsignalrange(2000, 3500);
if (err != SUCCESS)
    fprintf(stderr, "error setting signal range - %s\n", strerror(err));
pthread_setspecific()
Set Thread-Specific Data Pointer

Syntax
#include <pthread.h>

int pthread_setspecific(
    pthread_key_t key,
    const void *value);

Description

pthread_setspecific() function associates a thread-specific value with a key (input) obtained via a previous call to pthread_key_create(). Different threads may bind different values to the same key. These values are typically pointers to blocks of dynamically allocated memory that have been reserved for use by the calling thread. value is an input.

If successful, returns a value of 0; otherwise, returns an error.

Attributes

Operating System: OS-9
State: User
Compatibility: POSIX

Library

mt_clib.l

Possible Errors

ENOMEM Insufficient memory exists to associate the value with the key.
EINVAL The key value is invalid.

See Also

pthread_key_create()
pthread_getspecific()

Example

err = pthread_setspecific(thread_data_key, thread_data);
if (err != 0)
    fprintf(stderr, "error setting thread data - %s\n", strerror(err));
_pthread_setsuspendable()

Decrement Suspendability Counter

**Syntax**

```c
#include <pthread.h>
int _pthread_setsuspendable(void);
```

**Description**

`_pthread_setsuspendable()` decrements the suspendability counter for the calling thread. When this counter is at 0, the thread is suspendable.

This call would be used by applications that contain thread suspension and resource locking. Before taking a common lock a thread would set itself unsuspendable. This prevents the thread from holding a common lock while it is in the suspended state. Holding a common lock while suspended could cause deadlock for the remaining unsuspended threads. After unlocking the common lock the thread would call this function to return itself to the suspendable state.

Calling this function from a suspendable thread yields no change in state.

**Attributes**

- Operating System: OS-9
- State: User

**Library**

`mt_clib.l`

**Possible Errors**

None

**See Also**

- `_pthread_resume()`
- `_pthread_setunsuspendable()`
- `_pthread_suspend()`
Example

The following example illustrates how to execute a semaphore protected critical section. Using the mechanisms shown below, a thread calling _pthread_suspend() can be assured that glob_lock will not be claimed by the suspended thread.

```c
pthread_setsuspendable();

ec = _os_sema_p(&glob_lock);
if (ec != SUCCESS) {
    fprintf(stderr, "failed to get semaphore\n");
    pthread_exit((void *)ec);
}

/* critical section code */

ec = _os_sema_v(&glob_lock);
if (ec != SUCCESS) {
    fprintf(stderr, "failed to release semaphore\n");
    pthread_exit((void *)ec);
}

pthread_setsuspendable();
```

_pthread_setunsuspendable()

Description

(pthread_setunsuspendable() increments the suspendability counter for the calling thread. When this counter is greater than 0, the thread is unsuspendable. This call does not return until the unsuspendable state is achieved.

This call is used by applications that contain thread suspension and resource locking. Before taking a common lock a thread would use this call to set itself unsuspendable. This prevents the thread from holding a common lock while it is in the suspended state. After unlocking the common lock the thread would call pthread_setsuspendable() to return itself to the normal suspendable state.

Calling this function from an unsuspendable thread simply increases the suspendability counter. It is expected that each pthread_setunsuspendable() call has a matching pthread_setsuspendable() call.

Calling this function more than 0xffffffff times without any intervening pthread_setsuspendable() calls results in undefined behavior. Fewer pthread_setsuspendable() calls than pthread_setunsuspendable() calls will be required to return to the normal suspendable state.

Attributes

Operating System: OS-9
State: User

Library

mt_clib.l

Possible Errors

Errors from memory allocation and getting a process descriptor if called when signals are masked.

See Also

pthread_resume()
pthread_setsuspendable()
pthread_suspend()

Example

Refer to the example provided for pthread_setunsuspendable().

Syntax

#include <pthread.h>
int _pthread_setunsuspendable(void);
_pthread_suspend()

Increment Suspension Counter

Syntax

```
#include <pthread.h>

int _pthread_suspend(
    pthread_t thread,
    unsigned int *count);
```

Description

(pthread_suspend()) increments the suspension counter for the target thread specified by thread (input). The target thread's suspension counter prior to the suspension request is returned at the unsigned integer pointed to by count (output). A counter is used to support multiple suspension requests on the same target thread. An equal number of resume requests must be made before the target thread will resume execution.

This call does not return until the target thread has been successfully suspended. That is, if the target thread has set itself unsuspendable then this call will poll until the target sets itself back to suspendable.

Refer to the section on Thread Suspension for more information on what services are guaranteed while threads are suspended.

Returns 0 if the thread's suspension counter was successfully incremented or an error number if not.

Attributes

Operating System: OS-9
State: User

Library

mt_clib.l

Possible Errors

EINVAL The specified thread or count pointer is NULL.
ESRCH The specified thread is invalid or has terminated.
EDEADLK The specified thread is the calling thread and there is only one thread in the process.

See Also

pthread_resume()
pthread_setsuspendable()
pthread_setunsuspendable()
Example

err = _pthread_suspend(child, &count);
if (err != 0) {
    fprintf(stderr, "failed to suspend child\n");
    pthread_exit((void *)err);
}

/* do some activity with child suspended */

err = _pthread_resume(child, &status);
if (err != 0) {
    fprintf(stderr, "failed to resume child\n");
    pthread_exit((void *)err);
}
**Syntax**

```c
#include <pthread.h>
void pthread_testcancel(void);
```

**Description**

`pthread_testcancel()` checks for a pending, deferred cancel request. If there is one, cancellation cleanup handlers are called in the reverse order in which they were pushed, thread specific data destructors are called in an unspecified order, and the thread is terminated with `PTHREAD_CANCELED` as its status. If the cancel state of the thread is `PTHREAD_CANCEL_DISABLE`, this call has no effect.

`pthread_testcancel()` does not return if a cancel is pending.

**Attributes**

- Operating System: OS-9
- State: User
- Compatibility: POSIX

**Library**

`mt_clib.l`

**Possible Errors**

None

**See Also**

- `pthread_cancel()`
- `pthread_setcancelstate()`
- `pthread_setcanceltype()`

**Example**

```c
pthread_testcancel();
```
putc(), putchar()

Put Next Character to File, Standard Out

Syntax

```c
#include <stdio.h>

int putc(
    int chr,
    FILE *stream);

int putchar(int chr);
```

Description

putc() converts a character to an unsigned char. It writes the character to the
output stream at the position indicated by the associated file position indicator for
the stream, if defined, and advances the indicator appropriately. If the stream
cannot support positioning requests or if the stream was opened with append mode,
the character is appended to the output stream.

If it is implemented as a macro, it may evaluate the stream more than once.
Therefore, the parameter should never be an expression with side effects.

putc() returns the character written. If a write error occurs, the error indicator for
the stream is set and putc() returns EOF.

putchar() is similar to putc() with the second parameter stdout.

chr
    is the character to write. (Input)
    Only the lower 16 bits of chr are used.

stream
    is a pointer to the C I/O FILE structure. (Input)

Attributes

Operating System:    OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          No
Standards:           ANSI

Library

clib.l

See Also

fopen()
ftputc()
getc(), getchar()
putenv()  
Change or Add Value to Environment

Syntax

#include <UNIX/os9def.h>
int putenv(const char *string)

Description

putenv() sets the value of an environment variable.

putenv() manipulates the environment pointed to by _environ, and can be used in conjunction with getenv(). However, envp (the third argument to main) is not changed. This routine uses malloc to enlarge the environment.

putenv() returns non-zero if it was unable to obtain enough space using malloc for an expanded environment, otherwise zero.

string is in the form “name=value” where name is the name of the environment variable to set and value is the new value for the environment variable.

(Input)

If the variable named exists, its value is changed to value. If the variable name does not exist, a new environment list entry is created and set to the value passed. putenv() makes a copy of string so subsequent modifications to string do not affect the environment.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

unix.l
puts()

Output String to File

Syntax

```c
#include <stdio.h>
int puts(const char *strptr);
```

Description

puts() writes strptr to the stream pointed to by stdout and adds a newline character to the output. The terminating null character is not written.

puts() returns EOF if an error occurs. Otherwise, it returns a non-negative value.

strptr is a pointer to a string to write to stdout. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library
clib.l

See Also

fgets()  
gets()
Syntax

```c
#include <stdio.h>
int putw(
    int num,
    FILE *stream);
```

Description

`putw()` writes an integer, converted to a word, to a file. `putw()` neither assumes nor causes any special alignment in the file.

`putw()`, like `getw()`, is machine-dependent because the size of the integer it outputs varies with the integer size of the machine on which it resides. This compiler defines `int` values as 4-byte quantities, but `putw()` actually outputs 2 bytes.

If successful, `putw()` returns the value it has written. Otherwise, it returns -1.

`num` is the integer to write to the structure `FILE`. (Input)

`stream` is a pointer to the file. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

`getw()`
Syntax

```c
#include <stdlib.h>
void qsort(
    void *base,
    size_t nmemb,
    size_t size,
    int (*compar)
        (const void *, const void *));
```

Description

`qsort()` sorts an array.

- **base**
  - is a pointer to the initial array element. (Input)

- **nmemb**
  - is the number of elements in the array. (Input)

- **size**
  - is the size of each array element. (Input)

- **compar**
  - is a pointer to a comparison function. (Output)

  The contents of the array are sorted into ascending order according to `compar`. `compar` is called with two parameters that point to the objects being compared. `compar` returns an integer that is:

  - Negative, if the first parameter is less than the second.
  - Zero, if the first parameter is equal to the second.
  - Positive, if the first parameter is greater than the second.

  If two elements compare as equal, their order in the sorted array is unspecified. `qsort()` returns no value.

Attributes

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User and System
- **Threads**: Safe
- **Re-entrant**: Yes
- **Standards**: ANSI

Library

clib.l
Syntax
#include <signal.h>
int raise(int sig);

Description
raise() sends a signal to the executing program.

sig
    specifies the signal to send. (Input)
raise() returns zero if successful. Otherwise, it returns a non-zero value.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l
Syntax
#include <stdlib.h>
int rand(void);

Description
rand() computes a sequence of pseudo-random integers in the range zero to RAND_MAX and returns a pseudo-random integer.

rand() uses the following algorithm:
static unsigned long int next = 1;
int rand(void)    * RAND_MAX assumed to be 32767
{
    next = next * 1103515245 + 12345;
    return (unsigned int) (next/65536) % 32768;
}

next contains the current internal value for calculating pseudo-random numbers. Because the value of next is divided by 65536 before returning the value, this algorithm avoids the common problem of non-random, low-order bits.

By storing the value in an int, this algorithm also allows the period of the sequence to be greater than RAND_MAX.

Attributes
Operating System:     OS-9 and OS-9 for 68K
State:               User and System
Threads:             Safe
Re-entrant:          No
Standards:           ANSI

Library
clib.1
read()
Read Bytes from Path

Syntax
#include <modes.h>
int read(
    int path,
    char *buffer,
    unsigned count);

Description
read() reads bytes from a path.
read() essentially performs a raw read; that is, the read is performed without translating characters. The characters are passed to the program as read.
read() returns the number of bytes actually read (0 indicates the end of the file). If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

End-of-file is returned as zero bytes being read, not an error indication.

path specifies the path number of the path from which to read. (Input)
The path number is an integer indicating one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to open(), creat(), create(), or dup().

buffer is a pointer to a space with at least count bytes of memory into which read() puts the data read from the path. (Output)

count is the minimum buffer size. (Input)
It is guaranteed that at most, count bytes are read, but often less are read, either because the path serves a terminal and input stops at the end of a line or the end-of-file has been reached.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
read()

Library
sys_clib.1

See Also
creat()
create()
dup()
open()
read()
readln()
I$Read
I_READ

OS-9 for 68K Technical Manual
OS-9 Technical Manual
### Syntax

```c
#include <dir.h>
struct direct *readdir(DIR *dirp);
```

### Description

`readdir()` returns a pointer to a structure containing the next directory entry. `readdir()` returns `NULL` upon reaching the end of the directory or detecting an invalid `seekdir()` operation.

- **dirp**
  - is a pointer to the directory. (Input)

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No

### Library

`sys_clib.l`

### See Also

- `closedir()`
- `opendir()`
- `rewinddir()`
- `seekdir()`
- `telldir()`
Syntax

```c
#include <modes.h>
int readln(
    int path,
    char *buffer,
    unsigned count);
```

Description

`readln()` reads bytes from a path.

`readln()` causes line-editing, such as echoing, to take place and returns once a \n is read in the input or the number of bytes requested has been read. `readln()` is the preferred call for reading from the terminal.

`readln()` returns the number of bytes actually read (0 indicating end of file). If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

End-of-file is returned as zero bytes being read, not an error indication.

The thread enabled version of this function contains a cancel point. For more information see Using OS-9 Threads.

`path`

specifies the path number of the file from which to read. (Input)

The path number is an integer indicating one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

`buffer`

is a pointer to a space with at least `count` bytes of memory into which `read()` puts the data read from the path. (Output)

`count`

is the minimum buffer size. (Input)

It is guaranteed that at most, `count` bytes are read. Often less are read, either because the path serves a terminal and input stops at the end of a line or the end-of-file has been reached.
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
creat()
create()
dup()
open()
_os_readln()
read()
I$ReadLn OS-9 for 68K Technical Manual
I_READLN OS-9 Technical Manual
Syntax
#include <UNIX/os9def.h>
int readv(
    unsigned int fd,
    struct iovec *iov,
    unsigned int iovcnt);

Description
readv() reads input, the same action as read(). The difference is that readv() scatters the input data into the iovcnt buffers specified by the members of the iov array:
    iov[0], iov[1], ...iov[iovcnt-1].
If iovcnt is zero, the value of the iov_len member in an iovec structure is less than zero, or all the iov_len members of the iovec structures are zero, readv() returns -1 and sets the global variable errno.
For readv(), the iovec structure is defined as:
struct iovec{
    caddr_t iov_base;
    int iov_len
};
Each iovec entry specifies the base address and length of an area in memory where data should be placed. readv() always fills an area completely before proceeding to the next.
Upon successful completion, readv() returns the number of bytes actually read and placed in the buffers.

fd                Input
iov               Output
iovcnt            Input

Attributes
Operating System: OS-9 and OS-9 for 68K
State:            User and System
Threads:          Safe
Re-entrant:       No

Library
unix.l
realloc() changes the size of a memory block. The contents of the object are unchanged up to the lesser of the new and old sizes. If the new size is larger, the value of the newly allocated portion of the object is indeterminate.

realloc() returns either a null pointer or a pointer to the possibly moved allocated space.

**ptr**

is a pointer to the old block of memory. (Input)

If **ptr** is a null pointer, realloc() behaves like malloc() for the specified size. Otherwise, if **ptr** does not match a pointer earlier returned by calloc(), malloc(), or realloc(), or if the space has been deallocated by a call to free(), _lfree(), realloc(), or _lrealloc(), the behavior is undefined.

If the space cannot be allocated, **ptr** is unchanged.

**size**

is the size of the new block of memory. (Input)

If **size** is zero and **ptr** is not a null pointer, the object **ptr** points to is freed.
Syntax
#include <UNIX/regexp.h>
regexp *regcomp(const char *str);

Description
regcomp() returns a pointer to the compiled regular expression. Otherwise a string containing an error message is returned. If an error occurs, the function regerror() is called with a character string for the error and NULL is returned.

When the compiled regular expression is no longer needed, it can be passed to free().

strregcomp() compiles a string (str) into an internal form suitable for pattern matching with regex(). regex() matches a compiled regular expression against a string.

(Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
unix.l

See Also
regerror()
regex(), regexec()
Syntax

#include <UNIX/regexp.h>
void regerror (const char *msg);

Description

If printing a message is inappropriate for the application using regular expressions, the programmer may define a regerror() function to override the unix.l version.

msgregerror() is called by the functions regex() and regcomp() when various errors occur. regerror() prints the string "regexp: " and the string pointed to by msg.

(Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library

unix.l

See Also

regcomp()
regex(), regexec()
Syntax
#include <UNIX/regexp.h>
int regex(
    regexp *re,
    char const *str);

Description
regex() returns one if the strings match and zero if the strings don’t match. If an error occurs, regerror() is called with the appropriate error message and zero is returned to indicate the strings did not match.

re and strregex()
    compare a string (str) with a compiled regular expression (re). (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         No

Library
unix.l

See Also
regcomp()  
regerror()
Syntax
#include <stdio.h>
int remove(const char *filename);

Description
remove() deletes a file. Subsequent attempts to open a file using that name fail, unless the file is recreated.

remove() returns zero if the operation succeeds. Otherwise, it returns a non-zero value.

filename is a pointer to the file to delete. (Input)

If the file is open, remove() returns -1 with errno set to EOS_SHARE.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.1
rename() Rename File

Syntax
#include <stdio.h>
int rename(
    const char *old,
    const char *new);

Description
rename() renames a file.
The file named old is no longer accessible by that name. If a file named new exists prior to the call to rename(), it is overwritten.
rename() returns zero if the operation succeeds. Otherwise, it returns a non-zero value and the file name does not change.

old is a pointer to the name of an existing file. (Input)

new is a pointer to the new name for the file. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l
Syntax

#include <stdio.h>

void rewind(FILE *stream);

Description

rewind() sets the file position indicator for the stream pointed to by stream to the beginning of the file. It is equivalent to the following except that the error indicator for the stream is also cleared:

(void) fseek(stream, 0L, SEEK_SET);

stream is a pointer to the C I/O file structure. (Input)

rewind() returns no value.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

fseek()
rewinddir()

Reset Position of Directory Stream

Syntax

```c
#include <dir.h>
void rewinddir(DIR *dirp);
```

Description

rewinddir() resets the position of the named directory stream to the beginning of the directory.

`dirp`

is a pointer to the directory stream. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `closedir`
- `opendir`
- `readdir`
- `seekdir`
- `telldir`
rindex()

Search for Character in String

Syntax

```c
#include <strings.h>
char *rindex(
    const char *ptr,
    int ch);
```

Description

rindex() returns a pointer to the last occurrence of a character in a string. If the character is not found, rindex() returns NULL.

- **ptr** is a pointer to the string to search. (Input)
- **ch** is the character for which to search. (Input)
  Only the lower 8 bits of ch are used.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

sys_clib.l

See Also

- index()

Example

The following example code fragment looks for a period (.) in the string and sets ptr to point to it. The search starts from the end of the string and progresses to the front of the string. Note that ch is a character, not a pointer to a character:

```c
func()
{
    char *ptr,*string;
    if((ptr = rindex(string,'.')) {
        process(ptr);
    } else {
        printf("No ‘.’ found!\n");
    }
}
```
Syntax

```c
#include <stdlib.h>
char *sbrk(unsigned size);
```

Description

`sbrk()` allocates memory from the top of the data area upwards.

![Figure 2-10. sbrk() Memory Allocation Table](image)

`sbrk()` grants memory requests by calling the \texttt{F\_MEM} (for OS-9 for 68K) or the \texttt{F\_MEM} (for OS-9) system call. This method resizes the data area to a larger size. The new memory granted is contiguous with the end of the previous data memory.

On systems without an MMU, `sbrk()` fails quickly because it may keep growing in size until the data area reaches other allocated memory. At this point, it is impossible to increase in size and an error is returned. A program may increase its data size by only 20K, even if 200K is available elsewhere.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`size` specifies the size of the memory block. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

`sys_clib.l`
See Also

_os_mem()
_os_srqmem()
FS$Mem
F_MEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual
**scanf()**

**Input Strings Conversion**

**Syntax**

```c
#include <stdio.h>
int scanf(const char *format, ...);
```

**Description**

`scanf()` reads input from stdin.

`scanf()` returns `EOF` if an input failure occurs before any conversion. Otherwise, `scanf()` returns the number of input items assigned, which can be fewer than provided for, or zero in the event of an early matching failure.

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `scanf()` function. Refer to the Using Ultra C/C++ manual for more information about the small library functions.

**format**

is a pointer to the control string. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No
- **Standards:** ANSI

**Library**

`clib.l`

**See Also**

`fscanf()`

`sscanf()`
Syntax

```c
#include <sched.h>
int sched_yield(void);
```

Description

`sched_yield()` gives up a thread’s timeslice, allowing the kernel to reschedule the active threads. If there are other active threads at the same priority as the calling thread, one of them will be given the processor. The standard aging process occurs at every `sched_yield()` call.

`sched_yield()` works much like `_os_sleep()` with a ticks value of 1, except that it does not implicitly unmask signals.

If successful, 0 is returned, otherwise an error number is returned.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `_os_sleep()`
- `tsleep()`

Example

```c
while (not_ready)
    sched_yield(); /* allow other threads to run */
```
seekdir()

Set Position of Next Readdir

Syntax

```c
#include <dir.h>

void seekdir(
    DIR *dirp,
    long loc);
```

Description

`seekdir()` sets the position of the next `readdir()` operation on the directory stream to `loc`.

Values returned by `telldir()` are valid only for the lifetime of the associated `dirp` pointer. If the directory is closed and then reopened, the `telldir()` value may no longer be valid. It is safe to use a previous `telldir()` value immediately after a call to `opendir()` and before any calls to `readdir()`.

- `dirp` is a pointer to the directory. (Input)
- `loc` is the location for the next `readdir()` operation. (Input)
  - `loc` is a specific position in the directory stream returned by a previous call to `telldir()`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

sys_clib.l

See Also

- `closedir()`
- `opendir()`
- `readdir()`
- `rewinddir()`
- `telldir()`
Syntax

```
#include <UNIX/os9types.h>
#include <types.h>

int select (  
    unsigned int width,  
    fd_set *readfds,  
    fd_set *writefds,  
    fd_set *exceptfds,  
    struct timeval *timeout);
```

Description

The `select()` function examines the path sets whose addresses are passed in `readfds`, `writefds`, and `exceptfds` to see if some of their paths are ready for reading, ready for writing, or have an exceptional condition pending.

`select()` returns a non-negative value on success. A positive value indicates the number of ready paths in the path sets. Zero indicates that the time limit referred to by `timeout` has expired. On failure, `select()` returns -1, sets `errno` to indicate the error, and the path sets are not changed.

User applications using `select()` cannot use the `_os_intercept()` function.

No paths in the `exceptfds` are ever returned since OS-9 for 68K does not support the concept of “exceptional condition.”

`width` is the number of bits to be checked in each bit mask that represent a path. The paths from 0 through `width-1` in the descriptor sets are examined. `width` is in the range 1-FD_SETSIZE. The total number of ready paths in all the sets is returned. The macros `FD_SET`, `FD_CLR`, `FD_ISSET`, and `FD_ZERO` have been defined in `os9types.h` to manipulate these `fd_set` structures.

`timeout` If `timeout` is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If `timeout` is a NULL pointer, the `select` blocks indefinitely. To effect a poll, the `timeout` argument should be a non-NULL pointer, pointing to a zero-valued `timeval` structure.

`readfds`, `writefds`, `exceptfds` Any `readfds`, `writefds`, and `exceptfds` may be given as NULL pointers if no paths are of interest. (Input)
Attributes
Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: No

Library
unix.l
Syntax
#include <getset.h>
#include <types.h>

void _set_<name>(u_int32 val);

Description
The _set_<name>() class of functions can be used to set the value of Special Purpose Registers (SPRs), Device Control Registers (DCRs), Processor Registers (PREG), and Time-Based Registers (TBs).

All functions are provided. It is the programmer's responsibility to avoid using system-state only registers from user-state and accessing registers that do not exist on the target processor. All registers are accessible from system-state. Only those noted are available from user-state. Attempting to use system state registers from user state causes unpredictable results.

For information on which _get_<name>() functions are available for your processor, refer to the getset.h header file in the following location:

MWOS\OS9000\<PROCESSOR>\DEFS\getset.h

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
_get_<name>()
Syntax

```c
#include <stdio.h>
void setbuf(
    FILE *stream,
    char *buf);
```

Description

`setbuf()` sets up buffering for an I/O stream. It may be used only after the stream has been associated with an open file and before any other operation is performed on the stream. It is similar to `setvbuf()` called with `_IOFBF` for `mode` and `BUFSIZ` for `size`, or, if `buf` is a null pointer, with `_IONBF` for `mode`.

- `stream` is a pointer to the C I/O `FILE` structure. (Input)
- `buf` is a pointer to the file’s buffer. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

clib.l

See Also

- `fopen()`
- `getc(), getchar()`
- `putc(), putchar()`
- `setvbuf()`
_setcrc()
Re-Validate Module CRC

Syntax

#include <module.h>

int _setcrc(mh_com *module);

Description

_setcrc() updates the header parity and CRC of a module in memory. The module must have the correct size and sync bytes; other parts of the module are not checked. If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

Although _os_setcrc() and _setcrc() functions do update the header and CRC of a module in memory, the copy of the module header in the module directory is not affected. Therefore, you have an invalid module directory entry for the memory module with the new CRC and an error 236 results if links to the module are attempted.

module
is a pointer to the module to update. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

_os_setcrc()
F$SetCRC  OS-9 for 68K Technical Manual
F_SETCRC  OS-9 Technical Manual
setenv()  Insert or Reset Environment Variable Name

Syntax
#include <stdlib.h>
int setenv(
   const char *name,
   const char *value,
   int overwrite);

Description
The setenv() function inserts or resets the environment variable name in the current environment list. If the variable name does not exist in the list, it is inserted with the given value. If the variable does exist, the argument overwrite is tested; if overwrite is zero, the variable is not reset, otherwise it is reset to the given value.

setenv() and returns zero if successful; otherwise the global variable errno is set to indicate the error and a -1 is returned.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
unix.l
set_excpt_base()
Set Exception Table Base Address

Syntax
#include <regs.h>
void set_excpt_base(
    u_int32   *dsreg);

Description
set_excpt_base() sets the exception table base address. It is a system state only system call.

set_excpt_base() is supported only on the 80x86 family of processors version of the compiler.

dsreg
    is a pointer to the base address of the exception table. (Input)

Attributes
Operating System: OS-9
State: System
Threads: Safe
Re-entrant: Yes

Library
cpu.l
set_gdtr()

Set Global Descriptor Pointer

Syntax

```c
#include <regs.h>
void set_gdtr(
    u_int32 *base);
```

Description

`set_gdtr()` sets the global descriptor pointer. It is a system state only system call.

⚠️ `set_gdtr()` is supported only on the 80x86 family of processors version of the compiler.

`base` is a pointer to the global descriptor. (Input)

Attributes

- Operating System: OS-9
- State: System
- Threads: Safe
- Re-entrant: Yes

Library

`cpu.l`
setgid()
Set Global Descriptor Pointer

Syntax
#include <UNIX/os9def.h>
int setgid(
    unsigned int gid);

Description
setgid() sets the user’s group identification of the current process as specified by
 gid (input). setgid() returns -1 on error and sets the global variable errno to
 indicate the error.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:       User and System
Threads:       Safe
Re-entrant:       Yes

Library
unix.l
setime()

setime()
Set System Time

Syntax

```c
#include <time.h>

int setime(struct sgtbuf *timebuf);
```

Description

`setime()` sets the system time from a time buffer. The time units are defined in the `time.h` header file. This function uses the year 1900 as a base, and the year passed to the function must be an offset of 1900. For example, to use the `setime()` function and set to the year 2011, you need to pass 111 in the year member of the `tm` structure. If successful, `setime()` returns a pointer to a buffer. Otherwise, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Only super users may set the time.

timebuf

is a pointer to the time buffer. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `getime()`
- `_os_setime()`
- `_os9_setime()`

OS-9 for 68K Technical Manual
OS-9 Technical Manual
setitimer()
Set a Timer to Specified Value

Syntax
#include <UNIX/os9time.h>
int setitimer(
    int which,
    const struct itimerval *value,
    struct itimerval *ovalue);

Description
The `setitimer()` call sets a timer to the specified value (returning the previous value of the timer if `ovalue` is non-nil). A timer value is defined by the `itimerval` structure:

```c
struct itimerval {
    struct timeval it_interval;    /* timer interval */
    struct timeval it_value;       /* current value */
};
```

The first parameter must be `ITIMER_REAL`; the third must be `NULL`. If `it_value` is non-zero, it indicates the time to the next timer expiration. If `it_interval` is non-zero, it specifies a value to be used in reloading `it_value` when the timer expires. If `it_interval` is non-zero, it must be the same value as `it_value`. Setting `it_value` to 0 disables a timer. Setting `it_interval` to 0 causes a timer to be disabled after its next expiration (assuming `it_value` is non-zero).

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
unix.l

See Also
_os_alarm_cycle()
_os_alarm_set()
_os_alarm_delete()
Syntax
#include <setjmp.h>
int setjmp(jmp_buf env);

Description
setjmp() is a macro which saves its calling environment in its env parameter for later use by a call to longjmp().

If the return is from a direct call, setjmp() returns zero.

If the return is from a call to longjmp(), setjmp() returns a non-zero value.

setjmp() may only be used as follows:

- The entire expression of an expression statement. This may be cast to void. For example:
  (void) setjmp(env);
- The entire controlling expression of a selection or iteration statement. For example:
  while(setjmp(env))
- One expression of a relational or equality operator with the operand an integral constant expression, with the resulting expression being the entire controlling expression of a selection or iteration statement. For example:
  while(1<=setjmp(env))
- The operand of the unary operator (!) with the resulting expression being the entire controlling expression of a selection or iteration statement. For example:
  while(!setjmp(env))

env
  is a program environment structure. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.1

See Also
longjmp()
Syntax

#include <locale.h>
char *setlocale(
    int category,
    const char *locale);

Description

setlocale() selects the appropriate portion of the program’s locale as specified by category and locale. You can use setlocale() to change or query the program’s entire current locale or portions of it.

Currently, the C locale is fully supported.

If locale (input) is C, the minimal environment for C translation is specified.

If locale is “”, the value of the corresponding environment variable, LC_TIME for category LC_TIME, and so forth, is checked for validity. If the environment variable contains a valid locale string, locale is set accordingly. If it does not contain a valid string, setlocale() behaves as if passed an invalid locale string.

If the environment variable is not set or is set to the empty string, the existence and validity of the environment variable LANG is checked and locale is set from these. If LANG does not exist, locale is set to C.

At program startup, the equivalent of the following is executed:

setlocale(LC_ALL, “C”);

If locale is a pointer to a string and the selection can be honored, setlocale() returns a pointer to the string associated with the specified category for the new locale. If the selection cannot be honored, setlocale() returns a null pointer and the program’s locale is not changed.

A null pointer for locale() causes setlocale() to return a pointer to the string associated with the category for the program’s current locale; the program’s locale is not changed.

The pointer to the string returned by setlocale() is such that a subsequent call with that string value and its associated category restores that part of the program’s locale. The program does not modify the string, but it may be overwritten by a subsequent call to setlocale().
category can be one of the values shown in Table 2-24. (Input)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_ALL</td>
<td>Names the program’s entire locale. The other values for category name only a portion of the program’s locale.</td>
</tr>
<tr>
<td>LC_COLLATE</td>
<td>Affects the behavior of strcoll() and strxfrm()</td>
</tr>
<tr>
<td>LC_CTYPE</td>
<td>Affects the behavior of the character handling functions and the multibyte functions</td>
</tr>
<tr>
<td>LC_MONETARY</td>
<td>Affects the monetary formatting information returned by localeconv()</td>
</tr>
<tr>
<td>LC_NUMERIC</td>
<td>Affects the decimal point character for the formatted I/O functions and the string conversion functions, as well as the non-monetary formatting information returned by localeconv()</td>
</tr>
<tr>
<td>LC_TIME</td>
<td>Affects the behavior of strftime()</td>
</tr>
</tbody>
</table>

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

**Library**

clib.l

**See Also**

atof() atoi()
atol() fprintf()
fscanf() isalnum()
isalpha() iscntrl()
isdigit() isgraph()
islower() isprint()
ispunct() isspace()
isupper() isxdigit()
_issjis1() _issjis2()
_isjis_kana() mblen()
mbstowcs() mbtowc()
printf() scanf()
sprintf() sscanf()
vfprintf() vprintf()
vscanf() wctomb()
wctombs() strtod()
strtol() strtoul()
strcoll() strftime()
strxfrm()
Syntax
#include <process.h>
int setpr(
    int pid,
    int prior);

Description
setpr() sets the priority of a process. The lowest priority is zero; the highest is 65535.

If an error occurs, such as the process not having the same user ID as the caller, -1 is returned and the appropriate error code is placed in the global variable errno.

pid
    is the process ID. (Input)

prior
    specifies the priority. (Input)
    Only the lower 16 bits of prior are used.

Attributes
  Operating System:   OS-9 and OS-9 for 68K
  State:              User and System
  Threads:            Safe
  Re-entrant:         Yes

Library
sys_clib.l
Syntax

```c
#include <UNIX/pwd.h>
int setpwent(void);
```

Description

`setpwent()` has the effect of rewinding the password file to allow for repeated searches.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

`unix.l`

See Also

- `endpwent()`
- `fgetpwnent()`
- `getpwnent()`
- `getpwnam()`
- `getpwuid()`
setstat()

Set File Status

Syntax

```c
#include <sg_codes.h>
#include <sgstat.h>
int setstat(
    int code,
    int path,
    char *buffer); /* code = 0 */
int setstat(
    int code,
    int path,
    long size); /* code = 2 */
```

Description

`setstat()` sets the path options or the file size of the file open on `path`. If `code` is 0, the buffer is copied to the path descriptor options section. The `sgstat.h` header file contains the definitions for the path options.

If `code` is 2, `size` should be an `int` specifying the new file size. Only the lower 16 bits of `code` are used.

If an error occurs, both forms of the call return -1 and place the appropriate error code in the global variable `errno`.

This call exists for 6809 portability. The `_os_ss` functions are the preferred versions of these function calls.

`path`

is the path number of the open file. (Input)

`buffer`

is a pointer to the buffer containing the path descriptor options. (Input)

`size`

is the size of the new file. (Input)

Attributes

- Operating System: OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
### Library

`sys_clib.l`

### See Also

- `getstat()`
- `_os9_ss_close()`
- `_os9_ss_dcoff()`
- `_os9_ss_enrts()`
- `_os9_ss_fd()`
- `_os9_ss_popt()`
- `_os9_ss_reset()`
- `_os9_ss_sendsig()`
- `_os9_ss_skip()`
- `_os9_ss_xfm()`
- `_os9_ss_open()`

---

*OS-9 for 68K Technical Manual*
Syntax

```
#include <sysglob.h>
int _setsys(
    int glob,
    int size[,
    int value]);
```

OS-9 for 68K Users:
Include `setsys.h` in addition to `sysglob.h`.

For C++ Programmers:
The parameters for this function are not optional in C++. You must specify each; if you do not, a compilation error will occur.

Description

`_setsys()` changes or examines a system global variable. These variables are defined in `sys1`. The same values are defined in the `setsys.h` header file for C program use. These variables begin with a `D_` prefix.

`_setsys()` returns the value of the variable on an examination request. On a change request, `_setsys()` returns the value of the variable before the change.

If the examine or change request fails, -1 is returned and the appropriate error code is placed in the global variable `errno`.

Use `_setsys()` to examine variables without the possibility of accidental change.

- `glob` is the offset to the intended variable. (Input)
  Only the lower 16 bits of `glob` are used.

- `size` is the size of the variable. (Input)
  `size | 0x80000000` is used to examine the variable.

- `value` is an optional parameter used only when changing a variable. (Input)
Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

_setsys()
_os_setsys()
Syntax

#include <process.h>
int setuid(int uid);

Description

setuid() sets the group/user ID of the process to uid. The following restrictions govern the use of setuid():

- User number 0.0 may change his/her ID to anything without restriction.
- A primary module owned by user 0.0 may change the process’ user ID to anything without restriction.
- Any primary module may change the process’ user ID to match the module’s owner.

If the call fails, -1 is returned and the appropriate error code is placed in the global variable errno.

uid

is the user identification. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

getuid()
**Syntax**

```c
#include <stdio.h>
int setvbuf(
    FILE *stream,
    char *buf,
    int mode,
    size_t size);
```

**Description**

`setvbuf()` sets up buffering for an I/O stream. It may be used only after the `stream` has been associated with an open file and before any other operation is performed on the stream.

`setvbuf()` returns zero if successful. If an invalid value is given for `mode` or if the request cannot be honored, `setvbuf()` returns a non-zero value.

- **stream** is a pointer to the C I/O `FILE` structure. (Input)
- **buf** is a pointer to a buffer. (Input)
  
  If `buf` is not a null pointer, the library may use the array it points to instead of a buffer allocated by `setvbuf()`.
- **mode** determines how `stream` is buffered. (Input)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IOFBF</td>
<td>I/O is fully buffered</td>
</tr>
<tr>
<td>_IOLBF</td>
<td>I/O is line buffered</td>
</tr>
<tr>
<td>_IONBF</td>
<td>I/O is unbuffered</td>
</tr>
</tbody>
</table>

- **size** specifies the array size. (Input)
  
  The contents of the array at any time are indeterminate.
Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l
**Syntax**

```c
#include <signal.h>
int sigmask(int level);
```

**Description**

`sigmask()` controls the process’ signal mask. A signal mask is an internal variable in each process descriptor which determines the signal handling for the process.

The process’ signal level is not decremented below zero.

If a signal is received by a process whose signal mask is zero, normal program flow is interrupted and the signal is processed by executing the program’s intercept routine.

If a signal is received by a process whose signal mask is non-zero, the signal is placed in a queue of signals waiting to be processed. The queued signals become active only when the process’ signal mask becomes zero.

Regardless of the signal mask value, a process wakes up from event waits and I/O operation when signals arrive, but the signal handling routine is not executed until the signals are unmasked.

The process’ signal mask is automatically incremented while the intercept routine is executing. This prevents the intercept routine from being accidentally re-entered if a new signal arrives. The process may use `sigmask()` within its intercept routine to allow re-entrant signals or to force the signal mask to remain non-zero when normal program execution resumes.

When a process makes an `F$Sleep` or `F$Wait` (for OS-9 for 68K) or `F_SLEEP` or `F_WAIT` (for OS-9) system call, its signal mask is automatically cleared. If any signals are pending, the process returns to the intercept routine without sleeping.

The `S_KILL` and `S_WAKE` signals ignore the state of the signal mask and are never queued. `S_KILL` terminates the receiving process, and `S_WAKE` ensures that the receiving process is active.

If an error occurs, `-1` is returned and the appropriate error code is placed in the global variable `errno`. If errors do not occur, `sigmask()` returns zero.

I/O operations using the `cio` library should not be performed by both the main program and the intercept routine.

If an intercept routine is exited with `longjmp()`, the signal mask is still set to one. Generally, the destination of the long jump should unmask signals immediately.

**OS-9 for 68K:** The depth to which signals may be queued is limited only by available memory.

**OS-9:** There is a set limit.
level
    specifies the signal level. (Input)
    • 0 specifies clear
    • 1 specifies increment
    • −1 specifies decrement

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.1

See Also
 oslo_intercept()
 oslo_send()
 oslo_sigmask()
 F$SigMask OS-9 for 68K Technical Manual
 F_SIGMAK OS-9 Technical Manual
**signal()**

**Specify Signal Handling**

**Syntax**

```c
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

**Description**

`signal()` chooses one of three ways in which receipt of the signal number `sig` is handled.

- If `func` is `SIG_DFL`, default handling for that signal occurs.
- If `func` is `SIG_IGN`, the signal is ignored.
- Otherwise, `func` is a pointer to a function to call when that signal occurs. Such a function is called a signal handler.

When a signal occurs, signals are masked and `signal(sig, SIG_DFL)` is executed. Then, the specified type of handling is executed. Signals are blocked for the duration of the signal handler, unless the program takes some action that unmask the signal.

There are times in which a thread receives a signal it is not expecting. When this occurs, the thread forwards the signal to an arbitrary thread that is expecting it.

A program will not receive an abort or quit signal from the keyboard (usually `^C` and `^E`) unless the program performs output to the terminal. This is because the operating system sends the abort/quit signals to the last process to perform I/O to the terminal. If you run the program from the shell and type `^E` before the program performs I/O to the terminal, the shell receives the signal and kills the running program. If a program requires immediate control of the terminal, perform some I/O to one of the standard paths such as printing a program banner or getting the terminal options with `_gs_opt()`.

> For each signal recognized by `signal()`, the default handling is program termination (with an exit status related to the exception or the signal number for non-exception related signals) and the program startup condition is `SIG_DFL`.

The default handling is reset on all signals.

**Default Handling**

If you specify default handling, the process terminates and the termination status is set to the signal number that occurred. This occurs if no `signal()` has been called for the signal or `signal()` was called with `SIG_DFL` as `func`. If the program has used `intercept()` the signal handler assigned to the call is called and execution resumes at the point of interruption instead of terminating.

**Do not use _os_intercept() with signal().**
Ignore Handling

If a signal occurs that is to be ignored, the program resumes at the point of interruption.

If the occurring signal is exception-related, the process terminates with the appropriate vector as the exit status. Effectively, exception-related signals cannot be ignored.

Available Signals

The set of signals for the signal function varies with the operating system and the processor. A set of signals is also available regardless of the operating system.

ANSI “Core” Signals

Table 2-26. ANSI Core Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABRT</td>
<td>Abnormal termination, such as initiated by abort().</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>An erroneous arithmetic operation, such as zero divide or an operation resulting in overflow. This signal is generated on processor-specific conditions.</td>
</tr>
<tr>
<td>SIGILL</td>
<td>Detection of an invalid function image, such as illegal instruction. This signal is generated on processor-specific conditions.</td>
</tr>
<tr>
<td>SIGINT</td>
<td>Receipt of an interactive attention signal generated by a &lt;control&gt;C on OS-9 for 68K and OS-9.</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>An invalid access to storage. This signal is generated on processor-specific conditions.</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>A termination request sent to the program.</td>
</tr>
</tbody>
</table>

OS-9 for 68K/OS-9 “Core” Signals

Table 2-27. OS-9 for 68K and OS-9 Core Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGKILL</td>
<td>A death request sent to the program. This signal cannot be ignored or handled.</td>
</tr>
<tr>
<td>SIGWAKE</td>
<td>A wakeup request sent to the program. If a process is active when this signal arrives, it is ignored. Otherwise, the process is activated, but the signal handling function is not called.</td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>Receipt of an interactive abort signal generated by a &lt;control&gt;E on OS-9 for 68K and OS-9.</td>
</tr>
<tr>
<td>SIGHUP</td>
<td>Modem hangup signal. This signal is sent to a process when a modem connection is lost.</td>
</tr>
<tr>
<td>SIGNALRM</td>
<td>Not automatically sent. Programs could use this signal for an alarm call.</td>
</tr>
<tr>
<td>SIGPIPE</td>
<td>Not automatically sent.</td>
</tr>
</tbody>
</table>
Table 2-27. OS-9 for 68K and OS-9 Core Signals (Continued)

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGUSR1</td>
<td>Condition is user defined.</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>Condition is user defined.</td>
</tr>
</tbody>
</table>

680x0: OS-9 for 68K, OS-9

Table 2-28. OS-9 for 68K and OS-9 Core Signals for 680x0

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGADDR</td>
<td>Address error</td>
</tr>
<tr>
<td>SIGCHK</td>
<td>CHK instruction</td>
</tr>
<tr>
<td>SIGTRAPV</td>
<td>TRAPV instruction</td>
</tr>
<tr>
<td>SIGPRIV</td>
<td>Privilege violation</td>
</tr>
<tr>
<td>SIGTRACE</td>
<td>Trace exception</td>
</tr>
<tr>
<td>SIG1010</td>
<td>1010 line-A exception</td>
</tr>
<tr>
<td>SIG1111</td>
<td>1111 line-F exception</td>
</tr>
</tbody>
</table>

680x0: OS-9

Table 2-29. OS-9 Core Signals for 680x0

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGCOPRCV</td>
<td>Coprocessor protocol violation</td>
</tr>
<tr>
<td>SIGFMTERR</td>
<td>Format error</td>
</tr>
<tr>
<td>SIGUNIRQ</td>
<td>Uninitialized interrupt</td>
</tr>
</tbody>
</table>

80x86: OS-9

Table 2-30. OS-9 Core Signals for 80x86

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGGPROT</td>
<td>General protection</td>
</tr>
<tr>
<td>SIGSTACK</td>
<td>Stack exception</td>
</tr>
<tr>
<td>SIGSEGNP</td>
<td>Segment not present</td>
</tr>
<tr>
<td>SIGINVSS</td>
<td>Invalid TSS</td>
</tr>
<tr>
<td>SIGDBLFLT</td>
<td>Double fault</td>
</tr>
<tr>
<td>SIGBNDCHK</td>
<td>Boundary check</td>
</tr>
<tr>
<td>SIGBRKPT</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>SIGNMI</td>
<td>Non-maskable interrupt</td>
</tr>
<tr>
<td>SIGDBG</td>
<td>Debug exception</td>
</tr>
</tbody>
</table>
PowerPC: OS-9

Table 2-31. OS-9 Core Signals for PowerPC

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGALIGN</td>
<td>Alignment</td>
</tr>
<tr>
<td>SIGCHECK</td>
<td>Machine check</td>
</tr>
<tr>
<td>SIGINST</td>
<td>Instruction access</td>
</tr>
<tr>
<td>SIGPRIV</td>
<td>Privilege violation</td>
</tr>
</tbody>
</table>

Function Handling

If a signal which has an associated function occurs, the equivalent of (*func)(sig) is executed. func may terminate by executing a return statement or by calling abort(), exit(), or longjmp(). If func executes a return statement and sig was exception-related, the program exits with the appropriate vector for a status. Otherwise, the program resumes execution at the point it was interrupted.

Asynchronous Signals

Signals that occur by means other than abort() or raise() are called asynchronous. Signal handlers dealing with asynchronous signals should be careful about the following:

- Calling any function in the library (except signal() with the occurring signal) that may be working with static storage structures.

- Calling any function that is not re-entrant.

- Modifying any static storage structure other than a variable of type volatile sig_atomic_t.

If the signal() call can be honored, it returns func for the most recent call to signal() for the specified signal sig. Otherwise, SIG_ERR is returned and a positive value is stored in errno.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library
clib.l

See Also
aborted, exit(), intercept(), _os_intercept(), raise()
**Syntax**

```c
#include <math.h>

double sin(double x);
```

**Description**

`sin()` returns the sine of `x` (input) as a double float. `x` is in radians.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

clib.1
**Syntax**

```c
#include <math.h>
double sinh(double x);
```

**Description**

The `sinh()` function returns the hyperbolic sine of \( x \) (input) as a double float. \( x \) is in radians. A range error occurs, `ERANGE` stored in `errno`, if the magnitude of \( x \) is too large. In this case, the appropriately signed `HUGE_VAL` is returned.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

**Library**

`clib.l`

**Possible Error**

`ERANGE`
Syntax
#include <UNIX/stringlist.h>
StringList * sl_init();
int sl_add(
               StringList *sl,
               char *item);
void sl_free(
               StringList *sl,
               int freeall);
char * sl_find(
               StringList *sl,
               char *item);

Description
The stringlist functions manipulate stringlists, which are lists of strings that extend automatically if necessary. The StringList structure has the following definition:

typedef struct _stringlist {
    char    **sl_str;
    size_t    sl_max;
    size_t    sl_cur;
} StringList;

sl_str
    is a pointer to the base of the array containing the list.

sl_max
    is the size of sl_str.

sl_cur
    is the offset in sl_str of the current element.

The following stringlist manipulation functions are available:

sl_init()
    Create a stringlist. Returns a pointer to a StringList, or NULL in case of failure.

sl_free()
    Releases memory occupied by sl and the sl->sl_str array. If freeall is non-zero, then each of the items within sl->sl_str is released as well.
**sl_add()**

Add item to `sl->sl_str` at `sl->sl_cur`, extending the size of `sl->sl_str`. Returns zero upon success, -1 upon failure.

**sl_find()**

Find item in `sl`, returning NULL if it is not found.

**Attributes**

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: No

**Library**

`unix.l`

**See Also**

`free()`

`malloc()`
sleep()  
Suspend Execution for Specified Time

Syntax

```c
#include <signal.h>
int sleep(unsigned seconds);
```

Description

`sleep()` suspends the calling process for the specified time. A sleep time of zero seconds sleeps indefinitely. `sleep()` returns the number of ticks remaining to sleep if awakened prematurely by a signal.

![Warning]
The thread enabled version of this function contains a cancel point. For more information see *Using OS-9 Threads*.

`seconds`

specifies the length of time for the process to sleep. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K  
State: User and System  
Threads: Safe  
Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `_os_sleep()`  
- `_os9_sleep()`
- F$Sleep  
- F_SLEEP

OS-9 for 68K Technical Manual  
OS-9 Technical Manual
**Syntax**

```c
#include <module.h>
mh_com *_sliblink(
    int slib_num,
    const char *slib_name);
```

**Description**

`_sliblink()` links or loads the named module. Then, it places the subroutine module’s execution entry pointer and static storage data pointer in the global arrays `_sublibs` and `_submems`, respectively. You can use `_subcall()` to make subsequent calls the subroutine library.

For more information on creating, installing, and calling subroutine library modules, refer to the *OS-9 Technical Manual*.

You can remove a subroutine module by passing a null pointer for the name of the module and specifying the subroutine number. A process can link to a maximum 16 subroutine libraries, numbered from 0 to 15.

- **slib_num**
  - specifies the subroutine number. (Input)

- **slib_name**
  - is a pointer to the subroutine name string. (Input)

**Attributes**

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**See Also**

- `_subcall()`
- F_SLINK  
  *OS-9 Technical Manual*
**Syntax**

```c
#include <stdio.h>
int snprintf(
    char *str,
    size_t size,
    const char *format, ...);
```

**Description**

`snprintf()` writes to the character string `str`. The function writes to the at most `size - 1` of the characters printed into the output string (the `size`'th character then gets the terminating `\0`); if the return value is greater than or equal to the size argument, the string was too short and some of the printed characters were discarded. If size is zero, nothing is written and `str` may be a NULL pointer.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

sys_clib.l

**See Also**

- `sprintf()`
- `memcpy()`
- `memset()`
sprintf()

Formatted Output

Syntax

```c
#include <stdio.h>

int sprintf(
    char *str,
    const char *format, ...);
```

Description

`sprintf()` is a standard C library function that performs formatted output to the array specified by `str`. `sprintf()` converts, formats, and prints any parameters as indicated by the control string. A null character is written at the end of the characters written; it is not counted as part of the returned sum. If copying takes place between overlapping objects, the behavior is undefined.

`sprintf()` returns the number of characters written in the array, not counting the terminating null character.

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `sprintf()` function. Refer to the `Using Ultra C/C++` manual for more information about the small library functions.

`str`

is a pointer to an array into which to write the generated output. (Output)

`format`

is a pointer to a control string. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

`printf()`
`fprintf()`
sqrt()  
Square Root Function

Syntax

```c
#include <math.h>

double sqrt(double x);
```

Description

`sqrt()` returns the square root of `x` (input). `x` must not be negative. If the parameter is negative, `sqrt()` sets a domain error `EDOM` in `errno` and returns `HUGE_VAL`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`clib.l`

Possible Error

`EDOM`
srand()

Set Seed for Random Number Generator

**Syntax**

```c
#include <stdlib.h>
void srand(unsigned int seed);
```

**Description**

`srand()` uses the parameter as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to `rand()`. If `srand()` is then called with the same seed value, the sequence of pseudo-random numbers is repeated. If `rand()` is called before any calls to `srand()` have been made, the same sequence is generated as when `srand()` is first called with a seed value of one.

*seed* is used as a seed for a new sequence of pseudo-random numbers. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No
- **Standards:** ANSI

**Library**

clib.l

**See Also**

`rand()`
srqcmem()
Allocate Colored Memory

Syntax
#include <memory.h>
char *srqcmem(  
    int bytecnt,
    int memtype);

Description
srqcmem() is a direct hook to the F$SRqCMem (for OS-9 for 68K) or _os_srqmem() (for OS-9000) system call.

srqcmem() is identical to _srqmem() with the exception of the additional color parameter.

If successful, a pointer to the memory granted is returned. The returned pointer always begins on an even byte boundary. If the request was not granted, -1 is returned and the appropriate error code is placed in the global variable errno.

bytecnt
    is rounded to a system-defined block size. (Input)
    The size of the allocated block is stored in the global integer variable _srqcsiz. If bytecnt is 0xffffffff, the largest contiguous block of free memory in the system is allocated.

memtype
    indicates the specific type of memory to allocate. (Input)
    If memtype is MEM_ANY, any available system memory may be allocated.

memory.h
    contains definitions of the memory types that you may specify:

Table 2-32. srqcmem() Memory Types

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSRAM or MEM_SYS</td>
<td>System RAM memory</td>
</tr>
<tr>
<td>VIDEO1</td>
<td>Video memory for plane A</td>
</tr>
<tr>
<td>VIDEO2</td>
<td>Video memory for plane B</td>
</tr>
<tr>
<td>MEM_ANY</td>
<td>No memory type specified</td>
</tr>
</tbody>
</table>
**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User and System  
Threads: Safe  
Re-entrant: Yes

**Library**

sys_clib.l

**See Also**

- free()  
- malloc()  
- _os_srqmem()  
- _os9_srqmem()  
- _os_srtmem()

F$SRqCMem  
F__SRQMEM  

*OS-9 for 68K Technical Manual*  
*OS-9 Technical Manual*
_srqmem()

System Memory Request

Syntax

```
#include <memory.h>

char * _srqmem(unsigned size);
```

Description

 hỏng _srqmem() and the complementary function _srtmem() are provided to request and return system memory when tight control over memory allocation is required.

 hỏng _srqmem() is a direct hook into the F$SRqMem (for OS-9 for 68K) or F_SRQMEM (for OS-9) system call.

If successful, a pointer to the memory granted is returned. If the request was not granted, -1 is returned and the appropriate error code is placed in the global variable errno.

The returned pointer always begins on an even byte boundary. Care should be taken to preserve the value of the pointer if the memory is to be returned with _srtmem() or _os_srtmem().

size

specifies the number of bytes to request. (Input)

The specified size is rounded to a system-defined block size. A size of 0xffffffff obtains the largest contiguous block of free memory in the system. The global unsigned variable _srqsiz may be examined to determine the actual size of the allocated block.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

See Also

_-os_srqmem()
_-os9_srqmem()
_-os_srtmem()
F$SRqMem
F_SRQMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual
**Syntax**

```c
#include <memory.h>
int _srtmem(
    unsigned size,
    char *ptr);
```

**Description**

_srtmem() is a direct hook into the F$SRtMem (OS-9 for 68K) or _os_srtmem() (OS-9) system call. It is used to return memory granted by _srqmem() or _os_srqmem(). size and ptr must the same as those returned by _srqmem() or _os_srqmem(). If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

- **size**: specifies the number of bytes to return. (Input)
- **ptr**: is a pointer to the memory to return. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

sys_clib.l

**See Also**

- _os_srqmem()
- _os9_srqmem()
- _os_srtmem()
- F$SRtMem
- F_SRTMEM

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Set File Attributes

Syntax

```c
#include <sg_codes.h>
int _ss_attr(
    int path,
    int attr);
```

Description

`_ss_attr()` changes a disk file’s attributes. `_os_gs_fd()` determines the current attributes of a file. Only the owner of the file or the super user can change a file’s attributes.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`_ss_attr()` is effective even if the owner or super user does not have write permission to the path. You cannot set the directory bit of a non-directory file or clear the directory bit of a directory that is not empty.

`path` is the path number of a disk file. (Input)

`attr` specifies the file attributes to set. (Input)

Only the lower 16 bits of `attr` are used.

The attributes selected in `attr` are set in the file open on `path`. The `modes.h` header file defines the valid mode values.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `_os_gs_fd()`
- `_os_ss_attr()`
- `_os_ss_fd()`
- `I$SetStt`
- `I_SETSTAT, SS_ATTR`

OS-9 Technical Manual
OS-9 for 68K Technical Manual
**Syntax**

```c
#include <stdio.h>
int sscanf(
    const char *input,
    const char *format, ...);
```

**Description**

`sscanf()` reads input from the string specified by `input`. Reaching the end of the string is equivalent to encountering end-of-file for `fscanf()`. If copying takes place between overlapping objects, the behavior is undefined.

`sscanf()` returns `EOF` if an input failure occurs before any conversion. Otherwise, `sscanf()` returns the number of input items assigned. This can be fewer than provided for or zero, in the event of an early matching failure.

The `sclib.l` and `sclib.il` libraries contain a smaller version of the `sscanf()` function. Refer to the *Using Ultra C/C++* manual for more information about the small library functions.

`input` is a pointer to the input string. (Input)

`format` is a pointer to the control string. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
-Threads: Safe
- Re-entrant: No
- Standards: ANSI

**Library**

clib.l

**See Also**

- `scanf()`
- `fscanf()`
- `printf()`
**ss_dcoff()**

Send Data Carrier Lost Signal to Process

**Syntax**

```c
#include <scf.h>
int _ss_dcoff(
    int path,
    int sigcode);
```

**Description**

`_ss_dcoff()` sends a signal to the calling process when the “data carrier detect line” associated with the device is lost.

Only the lower 16 bits of `sigcode` are used on OS-9 for 68K. If `_ss_dcoff()` cannot register the signal, `-1` is returned and the appropriate error code is placed in the global variable `errno`.

- **path**
  - is the path number of the device. (Input)

- **sigcode**
  - is the signal code to send. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`sys_clib.l`

**See Also**

- `_os_ss_dcoff()`
- `I$SetStt`, `I_SETSTAT, SS_DCOFF`

- OS-9 for 68K Technical Manual
- OS-9 Technical Manual
_ss_dcon()

Send Data Carrier Present Signal to Process

**Syntax**

```c
#include <scf.h>
int _ss_dcon(
    int path,
    int sigcode);
```

**Description**

/ss_dcon() sends a signal to the calling process when the “data carrier detect line” associated with the device is present.

If _ss_dcon() cannot register the signal, -1 is returned and the appropriate error code is placed in the global variable errno.

**path**

is the path number of the device. (Input)

**sigcode**

is the signal code to send. (Input)

Only the lower 16 bits of sigcode are used on OS-9 for 68K.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

sys_clib.l

**See Also**

- _os_ss_dcon()
- I$SetStt
- I_SETSTAT, SS_DCON

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Disable RTS Line

Syntax

```c
#include <scf.h>
int _ss_dsrts(int path);
```

Description

`_ss_dsrts()` disables the RTS line for an open device.

If `_ss_dsrts()` cannot disable the RTS line, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`path` is the path number of the device. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `os_ss_dsrts()`
- `I$SetStt` OS-9 for 68K Technical Manual
- `I_SETSTAT, SS_DSRTS` OS-9 Technical Manual
Enable RTS Line

Syntax
#include <scf.h>
int _ss_enrts(int path);

Description
_ss_enrts() enables the RTS line for an open device.

If _ss_enrts() cannot enable the RTS line, -1 is returned and the appropriate error code is placed in the global variable errno.

path is the path number of the device. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
_os_ss_enrts()
I$SetStt
I_SETSTAT, SS_ENRTS

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax
#include <rbf.h>
int _ss_lock(
    int path,
    unsigned locksize);

Description
/ss_lock() locks out a section of the file open on path from the current file position up to the number of bytes specified by locksize.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

Refer to your operating system technical manual for information on RBF record locking.

path
is the path number of the file. (Input)

locksize
specifies the number of bytes in the file to lock. (Input)

If locksize is zero, all locks (record-lock, EOF lock, and file lock) are removed.

If locksize is 0xffffffff, the entire file is locked out regardless of the file pointer's location. This is a special type of file lock that remains in effect until released by _ss_lock(path, 0), a read or write of zero bytes, or the file is closed.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.1

See Also
_os_ss_lock()
I$SetStt
I_SETSTAT, SS_LOCK

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Set Path Options

Syntax

```c
#include <sg_codes.h>
#include <sgstat.h>
int _ss_opt(
    int path,
    struct sgbuf *buffer);
```

Description

(_ss_opt()) copies a buffer into the options section of the path descriptor open on path.

Generally, a program gets the options with _gs_opt(), changes the appropriate values, and updates the path options with _ss_opt(). The structure sgbuf declared in the sgstat.h header file provides a convenient means to access the individual option values.

If the path was invalid, -1 is returned and the appropriate error code is placed in the global variable errno.

It is common practice to preserve a copy of the original options so the program can restore them before exiting. The option changes take effect on the currently open path (and any paths created with I$Dup or I_DUP to the same).

If you use _ss_popt() on OS-9, you must use the OS-9 for 68K header file. _ss_luopt() is provided on OS-9 solely for compatibility with OS-9. If the OS-9 for 68K header file is unavailable, change the source code to use I_SETSTAT, SS_PATHOPT or I_SETSTAT, SS_LUOPT.

path
    specifies the path number of the path descriptor. (Input)

buffer
    is a pointer to a buffer containing the path descriptor information. (Input)

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library

sys_clib.l
See Also

_os_gs_popt()
_os_ss_popt()
I$SetStt
I_SETSTAT
tmode

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Using OS-9 for 68K or Using OS-9
**Syntax**

```c
#include <rbf.h>
#include <direct.h>
int _ss_pfd(
    int path,
    struct fd *buffer);
```

**Description**

/ss_pfd/() copies certain fields from the file descriptor information pointed to by `buffer` into the file descriptor sector of the file open on `path`. The buffer is usually obtained from `_gs_gfd()`. Only the owner ID, the modification date, and creation date fields are changed.

The structure `fd` declared in the `direct.h` header file provides a convenient means to access the file descriptor information.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

The buffer must be at least 32 bytes long or unanticipated data could be written into the file descriptor sector. Only the super user can change the owner ID field.

You cannot change the file attributes with this call. Instead, use `_ss_attr()`.

⚠️ You must use the OS-9 for 68K `direct.h` header file. This call is provided on OS-9 in order to provide compatibility with OS-9 for 68K. For faster speed, use `_os_ss_fd`.

`path` is the path number of the file. (Input)

`buffer` is a pointer to the buffer containing the file descriptor information. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

sys_clib.1
See Also

_os_ga_fd()
_os_ss_fd()
I$SetStt
I_SETSTAT

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Syntax

```c
#include <sg_codes.h>
int _ss_rel(int path);
```

Description

`_ss_rel()` cancels the signal to send from a device on data ready.

If an error occurs, −1 is returned and the appropriate error code is placed in the
global variable `errno`.

The signal request is also cancelled when the issuing process dies or closes the path
to the device. This feature exists only on SCF and PIPEMAN devices.

`path` is the device’s path number. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

- `_os_ss_relea()`
- `_os_ss_sendsig()`
- `I_SETSTAT, SS_RELEASE`

`OS-9 for 68K Technical Manual`

`OS-9 Technical Manual`
**Syntax**

```c
#include <sg_codes.h>
int _ss_rest(int path);
```

**Description**

_**ss_rest()**_ causes an RBF device to restore the disk head to track zero. It is usually used for disk formatting and error recovery.

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`path` is the device’s path number. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`sys_clib.l`

**See Also**

- `_os_ss_reset()`
- I$SetStt (OS-9 for 68K Technical Manual)
- I_SETSTAT, SS_RESET (OS-9 Technical Manual)
**Syntax**

```c
#include <sg_codes.h>
int _ss_size(
    int path,
    int size);
```

**Description**

`_ss_size()` changes the size of an open file. This call is effective only on RBF devices. The size change is immediate. If the file size is decreased, the freed sectors are returned to the system. If the file size is increased, sectors with undefined contents are added to the file.

If the path is invalid or the device is not an RBF device, -1 is returned and the appropriate error code is placed in the global variable `errno`.

- `path` is the file's path number. (Input)
- `size` specifies the new file size in bytes. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`sys_clib.l`

**See Also**

- `_os_ss_size()`
- I$SetStt
- I_SETSTAT, SS_SIZE

`OS-9 for 68K Technical Manual`

`OS-9 Technical Manual`
Send Signal on Data Ready

Syntax

```c
#include <sg_codes.h>
int _ss_ssig(
    int path,
    int sigcode);
```

Description

/ss_ssig() sets up a signal to be sent to the calling process when an interactive device has data ready. This feature exists only on SCF devices and pipes. When data is received on the device indicated by `path`, a signal is sent to the calling process.

/ss_ssig() must be called each time the signal is sent if it is to be used again.

The device is considered busy and returns an error if any read requests arrive before the signal is sent. Write requests to the device are allowed while in this state.

If an error occurs, -1 is returned and the appropriate error value is placed in the global variable `errno`.

`path`

is the device’s path number. (Input)

`sigcode`

specifies the signal code to send. (Input)

Only the lower 16 bits of `sigcode` are used on OS-9 for 68K.

Attributes

<table>
<thead>
<tr>
<th>Operating System:</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td>User and System</td>
</tr>
<tr>
<td>Threads:</td>
<td>Safe</td>
</tr>
<tr>
<td>Re-entrant:</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Library

`sys_clib.l`

See Also

- `os_ss_relea()`
- `os_ss_sendsig()`
- `I$SetStt`
- `I_SETSTAT, SS_SENDSIG`

OS-9 for 68K Technical Manual

OS-9 Technical Manual
ss_tiks()

Wait for Record Release

Syntax

```c
#include <rbf.h>
int ss_tiks(
    int path,
    int tickcnt);
```

Description

If a read or write request is issued for a part of a file that is locked out by another user, RBF normally sleeps indefinitely until the conflict is removed. This feature exists only on RBF devices. `ss_tiks()` may be used to return the error EOS_LOCK to the program if the conflict still exists after a specified number of ticks have elapsed.

If an error occurs, -1 is returned and the appropriate error value is placed in the global variable `errno`.

```plaintext
path
    specifies the file’s path number. (Input)

tickcnt
    specifies the number of ticks to wait if a record-lock conflict occurs while the file is open on `path`. (Input)

    • A `tickcnt` of zero (RBF’s default) causes a sleep until the record is released.
    • A `tickcnt` of one returns an error if the record is not released immediately.
```

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`sys_clib.lib`

See Also

`_os_ss_ticks()`
I$SetStt
I_SETSTAT, SS_TICKS

OS-9 Technical Manual
OS-9 for 68K Technical Manual


**Syntax**

```c
#include <rbf.h>
int _ss_wtrk(
    int path,
    int trkno,
    int siden,
    int ilvf,
    const char *trkbuf,
    const char *ilvptr);
```

**Description**

`_ss_wtrk()` performs a write-track operation on a disk drive. It is essentially a direct hook into the driver’s write-track entry point.

If an error occurs, -1 is returned and the appropriate error value is placed in the global variable `errno`.

This feature exists only on RBF devices. You can obtain additional information on actual use of this call by examining the `format` utility and/or a device driver.

- **path**: is the path on which the device is open. (Input)
- **trkno**: is the track number on which to write. (Input)
- **siden**: is the side of the track to write. (Input)
- **ilvf**: is the interleave factor. (Input)
- **trkbuf**: is a pointer to the track buffer image.(Input)
- **ilvptr**: is a pointer to the interleave table. (Input)

**Attributes**

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User and System
- **Threads**: Safe
- **Re-entrant**: Yes
Library
sys_clib.1

See Also
__os_ss_wtrack()  
I$SetStt
I_SETSTAT, SS_WTRACK
format

OS-9 for 68K Technical Manual
OS-9 Technical Manual
Using OS-9 for 68K or Using OS-9
stacksiz(), _stacksiz()

Get Size of Stack Used

Syntax

```c
#include <stdlib.h>
int stacksiz(void);
int _stacksiz(void);
```

Description

`stacksiz()` returns the maximum number of bytes of stack used at the time of the call if stack checking code is in effect. You can use `stacksiz()` to determine the stack size a program requires. `stacksiz()` is not available when compiling in strictly conforming ANSI mode.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

cstart.r
Stat()
Get File Status

Syntax

#include <UNIX/stat.h>

int stat(
    const char *path,
    struct stat *buf);

Description

Stat() obtains information about the file named by path (input). Read, write, or execute permission of the named file is not required, but all directories leading to the file must be searchable.

If an error occurs, stat() returns -1 and sets the global variable errno to indicate the error.

buf is a pointer to a struct stat structure into which information about the file is placed.

(Output)

A struct stat structure includes the following members:

- unsigned short st_mode; /* attributes */
- unsigned short st_nlink; /* nhard links */
- unsigned short st_uid; /* user ID of owner */
- unsigned short st_gid; /* group ID of owner */
- unsigned long st_size; /* file size */
- time_t st_atime; /* last access time */
- time_t st_mtime; /* last modify time */
- time_t st_ctime; /* last status change time */
- long st_ino; /* undefined */
- long st_dev; /* undefined */
- long st_rdev; /* undefined */

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

unix.l

See Also

fstat()
_strass()

Structure Assignment

Syntax

#include <strings.h>
void _strass(
    void *to,
    void *from,
    int count);

Description

_strass() is available to maintain source code compatibility. count bytes are copied from the memory pointed to by from to the memory pointed to by to, regardless of contents.

_strass() can move at most 65536 bytes. No regard is given to overlapping moves.

to
    is a pointer to the destination for the copy. (Output)

from
    is a pointer to the structure to copy. (Input)

count
    is the number of bytes to copy. (Input)

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes

Library

sys_clib.l
strncasecmp(), strcasecmp()

Compare Null-Terminated Strings

Syntax
#include <strings.h>
int strcasecmp(
    const char *s1,
    const char *s2);
int strncasecmp(
    const char *s1,
    const char *s2,
    size_t len);

Description
The strcasecmp() and strncasecmp() functions compare the null-terminated strings s1 and s2 and return an integer greater than, equal to, or less than 0, according as s1 is lexicographically greater than, equal to, or less than s2 after translation of each corresponding character to lowercase. The strings themselves are not modified. The comparison is done using unsigned characters, so that \200' is greater than \0'.

The strncasecmp() compares at most len characters.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
tolower()
**strcat()**

String Catenation

**Syntax**

```c
#include <string.h>
char *strcat(
  char *orig,
  const char *append);
```

**Description**

`strcat()` appends a copy of `append`, including the terminating null character, to the end of `orig`. The initial character of `append` overwrites the null character at the end of `orig`. If copying takes place between overlapping objects, the behavior is undefined.

- `orig` is a pointer to the original string. (Input/Output)
- `append` is a pointer to the string to append to `orig`. (Input)
- `strcat()` returns the value of `orig`.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

clib.l
**Syntax**

```c
#include <string.h>
char *strchr(
    const char *src,
    int chr);
```

**Description**

`strchr()` locates the first occurrence of `chr`, converted to a `char`, in string `src`. The terminating null character is considered part of the string.

`strchr()` returns a pointer to the located character, or a null pointer if the character does not occur in the string.

- **src**
  - is a pointer to the string to search. (Input)
- **chr**
  - specifies the character for which to search. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

**Library**

`clib.l`
**strcmp()**

**String Comparison**

**Syntax**

```c
#include <string.h>
int strcmp(
    const char *string1,
    const char *string2);
```

**Description**

`strcmp()` compares string1 to string2.

`strcmp()` returns an integer that is:

- Positive, if string1 is greater than string2
- Zero, if string1 equals string2
- Negative, if string1 is less than string2

*string1* is a pointer to a string to compare. (Input)

*string2* is a pointer to a string to compare. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

clib.1
**strcoll()**

String Comparison

**Syntax**

```c
#include <string.h>
int strcoll(
    const char *string1,
    const char *string2);
```

**Description**

`strcoll()` compares two strings, both interpreted as appropriate to the `LC_COLLATE` category of the current locale.

When both `string1` and `string2` are interpreted as appropriate to the current locale, `strcoll()` returns an integer that is:

- Positive, if `string1` is greater than `string2`.
- Zero, if `string1` equals `string2`.
- Negative, is `string1` is less than `string2`.

`string1` is a pointer to a string to compare. (Input)

`string2` is a pointer to a string to compare. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

**Library**

`clib.l`
Syntax

```c
#include <string.h>
char *strcpy(
    char *dest,
    const char *src);
```

Description

`strcpy()` copies a string, including the terminating null character, into an array. If
copying takes place between overlapping objects, the behavior is undefined.

`strcpy()` returns the value of `dest`.

- `dest` is a pointer to the destination array. (Output)
- `src` is a pointer to the string to copy into `dest`. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`clib.l`
**Syntax**

```c
#include <string.h>
size_t strcspn(
    const char *src,
    const char *delim);
```

**Description**

`strcspn()` returns the length of the segment of `src` up to the first occurrence of any character from `delim`.

- `src` is a pointer to a source string. (Input)
- `delim` is a pointer to a string of characters. (Input)

`strcspn()` returns the length of the segment.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

**Library**

clib.1
Syntax
#include <string.h>
char *
strdup(
    const char *str);

Description
The strdup() function allocates sufficient memory for a copy of the string str, does the copy, and returns a pointer to it. The pointer may subsequently be used as an argument to the function free().
If insufficient memory is available, NULL is returned.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
sys_clib.l

See Also
memcpy()
strlen()
Syntax
#include <string.h>
char *strerror(int errnum);

Description
strerror() maps an error number to an error message string. strerror() returns a pointer to the string. The program cannot modify the array, but a subsequent call to strerror() overwrites the array.

Error messages are generally in the form:
<major>:<minor> <detail>
<major> is the family of error. <minor> is the error number within the family. <detail> is the information from the system errmsg file, if available. The system errmsg file (*/dd/SYS/errmsg, /h0/SYS/errmsg, or /d0/SYS/errmsg) contains the error messages generated by perror().

errnum specifies the error number. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l

See Also
perror()
**Syntax**

```c
#include <time.h>
size_t strftime(
    char *dest,
    size_t maxsize,
    const char *format,
    const struct tm *timeptr);
```

**Description**

`strftime()` places characters into an array as controlled by `format`. The format is a multibyte character sequence, beginning and ending in its initial shift state.

All ordinary multibyte characters, including the terminating null character, are copied unchanged into the array. If copying takes place between overlapping objects, the behavior is undefined. No more than `maxsize` characters are placed in the array.

Each conversion specifier is replaced by appropriate characters as described in the following list. These characters are determined by the `LC_TIME` category of the current locale and by the values contained in the structure pointed to by `timeptr`.

**Table 2-33. `strftime()` Conversion Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Replaced By</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>The locale’s abbreviated weekday name.</td>
</tr>
<tr>
<td>%A</td>
<td>The locale’s full weekday name.</td>
</tr>
<tr>
<td>%b</td>
<td>The locale’s abbreviated month name.</td>
</tr>
<tr>
<td>%B</td>
<td>The locale’s full month name.</td>
</tr>
<tr>
<td>%c</td>
<td>The locale’s appropriate date and time representation.</td>
</tr>
<tr>
<td>%d</td>
<td>The day of the month as a decimal number (01 - 31).</td>
</tr>
<tr>
<td>%H</td>
<td>The hour (24-hour clock) as a decimal number (00 - 23).</td>
</tr>
<tr>
<td>%I</td>
<td>The hour (12-hour clock) as a decimal number (01 - 12).</td>
</tr>
<tr>
<td>%j</td>
<td>The day of the year as a decimal number (001 - 366).</td>
</tr>
<tr>
<td>%m</td>
<td>The month as a decimal number (01 - 12).</td>
</tr>
<tr>
<td>%M</td>
<td>The minute as a decimal number (00 - 59).</td>
</tr>
<tr>
<td>%p</td>
<td>The locale’s equivalent of the AM/PM designations associated with a 12-hour clock.</td>
</tr>
<tr>
<td>%S</td>
<td>The second as a decimal number (00 - 61).</td>
</tr>
<tr>
<td>%U</td>
<td>The week number of the year (the first Sunday as the first day of week 1) as a decimal number (00 - 53).</td>
</tr>
<tr>
<td>%w</td>
<td>The weekday as a decimal number (0 - 6)--Sunday is 0.</td>
</tr>
</tbody>
</table>
If a conversion specifier is not one of the above, the behavior is undefined. If the total number of resulting characters including the terminating null character is not more than \texttt{maxsize}, \texttt{strftime()} returns the number of characters placed into array \texttt{dest} not including the terminating null character. Otherwise, zero is returned and the contents of the array are indeterminate.

\texttt{dest} is a pointer to the array into which to place the characters. (Output)

\texttt{maxsize} specifies the maximum number of characters to place in the array. (Input)

\texttt{format} is a pointer to a control string. (Input)

\texttt{format} consists of zero or more conversion specifiers and ordinary multibyte characters. A conversion specifier consists of a percent (\%) sign followed by a character that determines the behavior of the conversion specifier.

\texttt{timeptr} is a pointer to a time structure. (Input)

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No
- **Standards:** ANSI

### Library

clib.1
strncpy()

Copy Old OS-9 for 68K Strings

Syntax

```c
#include <strings.h>

char *strncpy(
    char *dest,
    const char *src);
```

Description

strncpy() makes a copy of src in dest. strncpy() copies the bytes from src to dest until it reaches the null terminator. strncpy() then removes the high-bit from the last character and appends a null byte to the output string.

strncpy() is used primarily for copying file names from the directory entries on RBF disks. strncpy() returns its first parameter.

strncpy() assumes that dest contains adequate space for the copy. The calling routine is responsible for verifying the space.

dest
    is a pointer to the destination string. (Output)

src
    is a pointer to the source string to copy. (Input)

src
    is assumed to have the high-order bit set on the character byte indicating the last character.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l
strlcat()

Concatenate String src

Syntax

```c
#include <string.h>
size_t strlcat(
    char *dst,
    const char *src,
    size_t size);
```

Description

The `strlcat()` function appends the NULL-terminated string `src` to the end of `dst`. It will append at most `size - strlen(dst) - 1` bytes, NULL-terminating the result.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

`strlen()`
Syntax
#include <string.h>
size_t strlcpy(
    char *dst,
    const char *src,
    size_t size);

Description
The strlcpy() function copies up to size - 1 characters from the NUL-terminated string src to dst, NULL-terminating the result.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l
Syntax
#include <string.h>
size_t strlen(const char *str);

Description
strlen() determines the length of str.
strlen() returns the number of characters that precede the terminating null character.

str is a pointer to a string. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l
Syntax

```c
#include <string.h>
char *strncat(
    char *orig,
    const char *append,
    size_t count);
```

Description

`strncat()` appends characters (a null character and characters that follow it are not appended) from `append` to the end of `orig`. The initial character of `append` overwrites the null character at the end of `orig`. A terminating null character is always appended to the result. If copying takes place between overlapping objects, the behavior is undefined.

`strncat()` returns the value of `orig`.

- **orig**
  - is a pointer to the original string. (Input/Output)

- **append**
  - is a pointer to the string to append to `orig`. (Input)

- **count**
  - specifies the maximum number of characters to append. (Input)

Attributes

- **Operating System**: OS-9 and OS-9 for 68K
- **State**: User and System
- **Threads**: Safe
- **Re-entrant**: Yes
- **Standards**: ANSI

**Library**

c.lib
strncmp()  
String Comparison

Syntax
#include <string.h>
int strncmp(
   const char *string1,
   const char *string2,
   size_t count);

Description
strncmp() compares not more than count characters from string1 to string2. Characters following a null character are not compared. The strings do not need to be null terminated. strncmp() returns an integer that is:

- Positive, if string1 is greater than string2.
- Zero, if string1 equals string2.
- Negative, if string1 is less than string2.

string1  
is a pointer to a string to compare. (Input)

string2  
is a pointer to a string to compare. (Input)

count  
specifies the maximum number of characters to compare. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l
Syntax
#include <string.h>
char *strncpy(
    char *dest,
    const char *src,
    size_t count);

Description
strncpy() copies characters from src to dest. Characters following a null character are not copied. If copying takes place between overlapping objects, the behavior is undefined.

If src is shorter than count characters, null characters are appended to the copy in dest, until count characters in all have been written.

strncpy() returns the value of dest.

dest is a pointer to the destination string. (Output)

src is a pointer to the source string to copy. (Input)

count specifies the maximum number of characters to copy. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib1
strpbrk()

Locate First Occurrence of String

Syntax

```
#include <string.h>
char *strpbrk(
    const char *src,
    const char *delim);
```

Description

strpbrk() locates the first occurrence in src of any character from delim. strpbrk() returns a pointer to the character or a null pointer, if no character from delim occurs in src.

src

is a pointer to a string in which to search for any character from delim. (Input)

delim

is a pointer to a string containing characters to locate in src. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

clib.1
strptime()
Convert Character String Stored in tmf

Syntax
#include <time.h>
char *
strptime(
    const char *buf,
    const char *format,
    struct tm *tm);

Description
The strptime() function converts the character string pointed to by buf to values which are stored in the tm structure pointed to by tm, using the format specified by format.

The format string consists of zero or more conversion specifications, whitespace characters as defined by isspace(), and ordinary characters. All ordinary characters in format are compared directly against the corresponding characters in buf; comparisons which fail will cause strptime() to fail. Whitespace characters in format match any number of whitespace characters in buf, including none.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
sys_clib.l

See Also
strcasicmp(), strncasicmp()
strlen()
strftime()
strrchr()

Locate Last Occurrence of String

Syntax

```c
#include <string.h>
char *strrchr(
    const char *src,
    int chr);
```

Description

`strrchr()` locates the last occurrence of `chr`, converted to `char`, in `src`. The terminating null character is considered part of the string.

`strrchr()` returns a pointer to the character or a null pointer, if `chr` does not occur in the string.

- `src` is a pointer to a string. (Input)
- `chr` is the character to search for in `src`. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.1
strsep()

Replace character in *stringp

Syntax

```c
#include <string.h>
char *
strsep(
    char **stringp,
    const char *delim);
```

Description

The `strsep()` function locates, in the null-terminated string referenced by `*stringp`, the first occurrence of any character in the string `delim` (or the terminating `\0` character) and replaces it with a `\0`. The location of the next character after the delimiter character (or NULL, if the end of the string was reached) is stored in `*stringp`. The original value of `*stringp` is returned.

An empty field (one caused by two adjacent delimiter characters) can be detected by comparing the location referenced by the pointer returned in `*stringp` to `\0`.

If `*stringp` is initially NULL, `strsep()` returns NULL.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`sys_clib.l`
Syntax
#include <string.h>
char *strspn(
    const char *src,
    const char *pattern);

Description
strspn() returns the length of the maximum initial segment of src which consists entirely of characters from pattern.

src
    is a pointer to a string. (Input)

pattern
    is a pointer to a string. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l
**Syntax**

```c
#include <string.h>
char *strstr(
    const char *src,
    const char *pattern);
```

**Description**

`strstr()` locates the first occurrence in `src` of the sequence of characters (excluding the terminating null character) in `pattern`. `strstr()` returns a pointer to the located string or a null pointer, if the string is not found. If `pattern` is a pointer to a string with zero length, `strstr()` returns `src`.

- `src` is a pointer to a string. (Input)
- `pattern` is a pointer to a string containing a sequence of characters to search for in `src`. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

`clib.l`
Syntax

#include <stdlib.h>

double strtod(
    const char *begin,
    char **end);

Description

strtod() converts the initial portion of begin to double representation. strtod() separates the input string, attempts to convert the subject sequence to a floating point number, and then returns the result.

strtod() expects the subject sequence to have the following form:

- An optional plus or minus sign.
- A non-empty sequence of digits optionally containing a decimal point character.
- An optional exponent part, but no floating suffix.

If the subject sequence has the expected form, the character sequence starting with the first digit or the decimal point character (whichever occurs first) is interpreted as a floating constant. Currently, only the c locale is supported. Therefore, the decimal point character is a period. If neither an exponent part nor a decimal point character appears, a decimal point is assumed to follow the last digit in the string. If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in end, unless end is a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed. begin is stored in end, unless end is a null pointer.

strtod() returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, plus or minus HUGE_VAL is returned, according to the sign of the value, and ERANGE is stored in errno. If the correct value would cause underflow, zero is returned and ERANGE is stored in errno.
begin
  is a pointer to the beginning of a string. (Input)
end

  is a pointer to the pointer to the first character after the converted string. (Input)

The input string is separated into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).
2. A subject sequence resembling a floating point constant. The subject sequence is the longest initial subsequence of the input string starting with the first non-white space character having the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign, a digit, or a decimal point character.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

`clib.1`

**See Also**

`isspace()`  
`strtoll()`  
`strtoul()`

**Possible Error**

`ERANGE`
**Syntax**

```c
#include <string.h>
char *strtok(
    char *src,
    const char *delims);
```

**Description**

`strtok()` breaks `src` into a series of tokens when a sequence of calls is made to `strtok()`. A character from `delims` separates each token. When the first call is made, `src` is the first parameter. A null pointer is the first parameter for subsequent calls.

The first call in the sequence searches `src` for the first character that is not contained in the current `delims`. If no such character is found, then there are no tokens in `src`, and `strtok()` returns a null pointer. If a character is found, it starts the first token.

`strtok()` searches from that location for a character that is contained in the current `delims`. If the character is not found, the current token extends to the end of `src`, and subsequent searches for a token return a null pointer. If the character is found, it is overwritten by a null character to terminate the current token.

The search for the next token starts with the next character. `strtok()` saves a pointer to this location. Each subsequent call behaves in the same manner.

`strtok()` returns a pointer to the first character of a token, or a null pointer if there is no token.

- `src` is a pointer to the string to break into tokens. (Input)
- `delims` is a pointer to a string containing characters to separate each token. (Input)

`delims` may be different from call to call.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No
- **Standards:** ANSI
Library
clib.l

Example
#include <string.h>
static char str[] = "?a??b,,,#c";
char *t;

  t = strtok(str, "?"); /* t points to the token “a” */
  t = strtok(NULL, ","); /* t points to the token “??b” */
  t = strtok(NULL, ",,"); /* t points to the token “#c” */
  t = strtok(NULL, "?"); /* t is a null pointer */
strtol()
String to Long Conversion

Syntax

```
#include <stdlib.h>
long strtol(
    const char *begin,
    char **end,
    int base);
```

Description

`strtol()` converts the initial portion of begin to “long int” representation.

- **begin** is a pointer to the beginning of the string. (Input)
- **end** is a pointer to the pointer to the first character after the converted string. (Input)
- **base** is the base of the number string. (Input)

`strtol()` separates the input string into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).
2. A subject sequence resembling an integer represented in some radix determined by base. The subject sequence is the longest initial subsequence of the input string, starting with the first non-white space character of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign or a permissible letter or digit.
3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

`strtol()` then attempts to convert the subject sequence to an integer and returns the result.

- If `base` is zero, the subject sequence is expected to be an integer constant, optionally preceded by a plus or minus sign. An integer suffix is not included. If the subject sequence is an integer constant, the character sequence starting with the first digit is interpreted as an integer constant.
- If `base` is between two and 36, the subject sequence is expected to be a sequence of letters and digits representing an integer. `base` specifies the radix, optionally preceded by a plus or minus sign. An integer suffix is not included. The letters from A through Z (regardless of case) are assigned the values 10 to 35. Only letters with assigned values less than base are permitted. If `base` is 16, the characters 0x or 0X may optionally precede the sequence, following the sign if present. If the subject sequence has the expected form, it is used as the base for conversion, ascribing to each letter its value.
If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in end, unless end is a null pointer. If the subject sequence is empty or does not have the expected form, no conversion is performed. begin is stored in end, unless end is a null pointer.

`strtol()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LONG_MAX` or `LONG_MIN` is returned (according to the sign of the value) and `ERANGE` is stored in `errno`.

**Attributes**

Operating System: OS-9 and OS-9 for 68K  
State: User and System  
Threads: Safe  
Re-entrant: Yes  
Standards: ANSI

**Library**

clib.l

**See Also**

`isupper()`  
`strtod()`  
`strtoll()`  
`strtoul()`  

**Possible Error**

`ERANGE`
Syntax

```c
#include <stdlib.h>
long long int strtoll(
const char *nptr,
char **endptr,
int base);
```

Description

The `strtoll()` function converts the string in `nptr` to a “long long int” value.

The string may begin with an arbitrary amount of white space (as determined by `isspace(3)`) followed by a single optional `+` or `-` sign. If `base` is zero or 16, the string may then include a `0x` prefix, and the number will be read in base 16; otherwise, a zero base is taken as 10 (decimal) unless the next character is `0`, in which case it is taken as 8 (octal).

The remainder of the string is converted to a long long value in the obvious manner, stopping at the first character which is not a valid digit in the given base. (In bases above 10, the letter `A` in either upper or lower case represents 10, `B` represents 11, and so forth, with `Z` representing 35.)

If `endptr` is non nil, `strtoll()` stores the address of the first invalid character in `*endptr`. If there were no digits at all, however, `strtoll()` stores the original value of `nptr` in `*endptr`. (Thus, if `*nptr` is not `\0` but `**endptr` is `\0` on return, the entire string was valid.)

The `strtoll()` function returns the result of the conversion, unless the value would underflow or overflow. If an underflow occurs, `strtoll()` returns `LONG_LONG_MIN`. If an overflow occurs, `strtoll()` returns `LONG_LONG_MAX`. In both cases, `errno` is set to `ERANGE`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`sys_clib.l`
See Also

- `strtod()`  
- `strtol()`  
- `strtoul()`  

Possible Error

**ERANGE**  

The given string was out of range; the value converted has been clamped.
**Syntax**

```c
#include <stdlib.h>
#include <limits.h>

unsigned long strtoul(
    char *begin,
    char **end,
    int base);
```

**Description**

`strtoul()` converts the initial portion of `begin` to `unsigned long int` representation.

`strtoul()` separates the input string into three parts:

1. An initial, possibly empty, sequence of white space characters (as specified by `isspace()`).

2. A subject sequence resembling an unsigned integer represented in some radix determined by `base`. The subject sequence is the longest initial subsequence of the input string, starting with the first non-white space character of the expected form. The subject sequence contains no characters if the input string is empty or consists entirely of white space, or if the first non-white space character is other than a sign or a permissible letter or digit.

3. A final string of one or more unrecognized characters, including the terminating null character of the input string.

`strtoul()` then attempts to convert the subject sequence to an unsigned integer and returns the result.

- If `base` is zero, `strtoul()` expects the subject sequence to be an integer constant, optionally preceded by a plus or minus sign. An integer suffix is not included.
  
  If the subject sequence is an integer constant, the character sequence starting with the first digit is interpreted as an integer constant.

- If `base` is between 2 and 36, the subject sequence is expected to be a sequence of letters and digits representing an integer. `base` specifies the radix, optionally preceded by a plus or minus sign. It does not include an integer suffix. The letters `a` through `z` (regardless of case) are assigned the values 10 to 35. Only letters whose assigned values are less than `base` are permitted. If `base` is 16, the characters `0x` or `0X` may optionally precede the sequence of letters and digits, following the sign if present.

  If the subject sequence has the expected form, it is used as the base for conversion, ascribing to each letter its value.
If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. A pointer to the final string is stored in end, unless end is a null pointer.

If the subject sequence is empty or does not have the expected form, no conversion is performed. begin is stored in end, unless end is a null pointer.

=strtoul() returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, ULONG_MAX is returned and ERANGE is stored in errno.

begin
   is a pointer to the beginning of the string. (Input)

end
   is a pointer to the pointer to the first character after the converted string. (Input)

base
   is the base of the number string. (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes
Standards:          ANSI

Library
clib.l

See Also
isspace()
strtol()
strtol()
Syntax

```c
#include <string.h>
size_t strxfrm(
    char *dest,
    const char *trans,
    size_t count);
```

Description

`strxfrm()` transforms `trans` and places the result in `dest`. You can then use `strcmp()` to compare two transformed strings. The result of the `strcmp()` on the transformed strings is the same as the result of `strcoll()` applied to the two original strings.

If copying takes place between overlapping objects, the behavior is undefined.

`strxfrm()` returns the length of the transformed string, not including the terminating null character. If the value returned is `count` or more, the contents of `dest` are indeterminate.

- `dest` is a pointer to the destination string.
- `trans` is a pointer to the string to transform.
- `count` specifies the maximum number of characters to place in `dest`. `count` includes the terminating null character. If `count` is zero, `dest` can be a null pointer.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

`clib.l`

See Also

- `strcmp()`  
- `strcoll()`
Syntax
#include <module.h>
void _subcall(void);

Description
_subcall() calls a subroutine module previously linked by _sliblink().
_subcall() expects the first two longwords in the code area following it to contain, respectively:
  • The subroutine library number.
  • The function number to call.
_subcall() retrieves these identifiers, adjusts the program stack, and dispatches to the subroutine library entry point with the correct global static storage configuration.

A process can link to a maximum of sixteen subroutine libraries, numbered from 0 to 15.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
clib.l

See Also
_subcall()
sysctl()

Get and Set System Information

Syntax

```c
#include <sys/param.h>
#include <sys/sysctl.h>

int sysctl(int *name,
    u_int namelen,
    void *oldp,
    size_t *oldlenp, void *newp,
    size_t newlen);
```

Description

`sysctl()` retrieves system information and allows processes with appropriate privileges to set system information. Unless explicitly noted below, `sysctl()` returns a consistent snapshot of the data requested.

- **name**: array of integers specifying the system variable to get/set
- **namelen**: number of integers in the `name` array
- **oldp**: pointer to a buffer that contains the current value of the system variable
  The required size can be determined by calling `sysctl()` with a NULL parameter for `oldp`. The required size will then be returned in `oldlenp`.
- **oldlenp**: The location pointed to by `oldlenp` specifies the size of the `oldp` buffer on input; on output it specifies the number of bytes copied into the `oldp` buffer.
  If `oldlenp` is NULL, no information is returned. If the `oldp` buffer is too small, as much data as possible is copied into it and an ENOMEM error is returned.
- **newp**: pointer to a buffer that contains the new value of the system variable specified by `name`
  If the system variable is not changed, `newp` should be NULL.
- **newlen**: specifies the size of the `newp` buffer
  If the system variable is not changed, `newlen` should be 0.
The system variables controlled by `sysctl()` are grouped according to functionality and organized in a logical tree. The `name` parameter is an array of integers that identifies a particular variable. The first entry of this array is the top level group; the next entry is the second level group. Currently, the only top level group supported is `CTL_NET`, which contains variables related to the networking subsystem.

**CTL_NET**

The string and integer information available for the `CTL_NET` level is detailed in the table below. The “Changeable” column details whether or not a process with appropriate privilege may change the value.

**Table 2-34.**

<table>
<thead>
<tr>
<th>Second Level Name</th>
<th>Type</th>
<th>Changeable</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF_ROUTE</td>
<td>routing messages</td>
<td>no</td>
</tr>
<tr>
<td>PF_INET</td>
<td>IPv4 values</td>
<td>yes</td>
</tr>
<tr>
<td>PF_INET6</td>
<td>IPv6 values</td>
<td>yes</td>
</tr>
</tbody>
</table>

**PF_ROUTE**

Return the entire routing table or a subset of it. The data is returned as a sequence of routing messages. The length of each message is contained in the message header.

The third level name is a protocol number, which is currently always 0. The fourth level name is an address family, which may be set to 0 to select all address families. The fifth and sixth level names are as follows:

- **Fifth Level:**
- **Sixth Level:**
  - NET_RT_FLAGS   rtflags
  - NET_RT_DUMP    None
  - NET_RT_IFLIST  None

**PF_INET**

Get or set various global information about IPv4 (Internet Protocol version 4). The third level name is the protocol. The fourth level name is the variable name. The currently defined protocols and names include:

- `ip forwarding`: A changeable integer. This returns 1 when IP forwarding is enabled for the host, meaning that the host is acting as a router.
PF_INET6

Get or set various global information about IPv6 (Internet Protocol version 6). The third level name is the protocol. The fourth level name is the variable name. The currently defined protocols and names include:

- **ip6 forwarding**: A changeable integer. Returns 1 when IPv6 forwarding is enabled for the node, meaning that the node is acting as a router. Returns 0 when IPv6 forwarding is disabled for the node, meaning that the node is acting as a host. IPv6 specification defines node behavior for “router” case and “host” case quite differently, and changing this variable during operation may cause serious trouble. It is recommended to configure the variable at bootstrap time, and bootstrap time only.

- **ip6 accept rtadv**: A changeable integer. If set to non-zero, the node will accept ICMPv6 router advertisement packets and autoconfigures address prefixes and default routers. The node must be a host (not a router) for the option to be meaningful.

**Attributes**

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

**Library**

unix.l

**Example**

This example retrieves the current value for IPv4 forwarding.

```c
size_t len;
int mib[4];
int ipforwarding = 0;

len = sizeof(ipforwarding);
mib[0] = CTL_NET;
mib[1] = PF_INET;
mib[2] = IPPROTO_IP;
mib[3] = IPCTL_FORWARDING;
sysctl (mib, 4, &ipforwarding, &len, 0, 0);
return (ipforwarding);
```
sys_mktime()

Convert Local Time to Greenwich Mean Time

Syntax

```
#include <time.h>

time_t sys_mktime(struct tm *tp);
```

Description

sys_mktime() converts the local time values in a broken-down time structure into a
Universal Coordinated Time value.

sys_mktime() ignores the input values of tm_wday and tm_yday. It does not require
the other fields of tm to be within the ranges specified in the structure. Instead, it
normalizes the fields and sets tm_wday, tm_yday, and tm_isdst. This allows you to
easily do temporal arithmetic without having to worry about mixed-base
operations.

If (time_t)-1 returns to indicate an error, there is a one second interval that appears
to be non-representable. The encoding of times for time_t only represents times
ranging from approximately 1902 to 2038.

tp

is a pointer to a broken-down time structure. (Input)

Broken-down time has the following format:

```
struct tm {
    int tm_sec;     /* seconds after the minute: [0,59] */
    int tm_min;     /* minutes after the hour: [0,59] */
    int tm_hour;    /* hours after midnight: [0,23] */
    int tm_mday;    /* day of the month: [1,31] */
    int tm_mon;     /* months after January: [0,11] */
    int tm_year;    /* years after 1900 */
    int tm_wday;    /* days after Sunday: [0,6] */
    int tm_yday;    /* days after January 1: [0,365] */
    int tm_isdst;   /* Daylight Savings Time (DST) flag */
                         /* 0 = no, 1 = yes, -1 = unknown */
};
```

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: No

Library

sys_clib.l

See Also

mktime()
_sysdate()

Get Current System Date/Time

Syntax

#include <time.h>
int _sysdate(
    int format,
    int *time,
    int *date,
    short *day,
    int *tick);

Description

_sysdate() obtains the current time, date, day of week, and clock tick from the
system. Note that all the parameters except format are pointers to the receiving
locations.

format
    specifies the format of the date and time to return. (Input)
    format can be any of the following:
    • 0 = Gregorian
    • 2 = Gregorian with ticks
    • 1 = Julian
    • 3 = Julian with ticks

time
    is a pointer to the location where _sysdate() stores the time value. (Output)

date
    is a pointer to the location where _sysdate() stores the date value. (Output)

day
    is a pointer to the location where _sysdate() stores the day of week value.
    (Output)
tick is a pointer to the location where _sysdate() stores the tick value. (Output)

The values are returned in the following format:

**Figure 2-11. _sysdate() Return Format**

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time:</td>
<td>0</td>
<td>Hour (0-23)</td>
<td>Minute</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Seconds Since Midnight (0-86399)</td>
<td></td>
</tr>
<tr>
<td>Date:</td>
<td></td>
<td>Year (2 bytes)</td>
<td>Month</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Julian Day Number</td>
<td></td>
</tr>
<tr>
<td>Day:</td>
<td></td>
<td>Day of the Week</td>
<td>0 = Sunday, 1 = Monday, ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tick:</td>
<td></td>
<td>Ticks per Second</td>
<td>Current Clock Tick</td>
</tr>
</tbody>
</table>

If an error occurs, -1 is returned and the appropriate error code is placed in the global variable errno.

Be careful when passing pointers to the date, time, and day values. Also, be sure day is declared to be a short or the value appears as day + 65536.

**Attributes**

<table>
<thead>
<tr>
<th>Operating System:</th>
<th>OS-9 and OS-9 for 68K</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td>User and System</td>
</tr>
<tr>
<td>Threads:</td>
<td>Safe</td>
</tr>
<tr>
<td>Re-entrant:</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Library**

sys_clib.1
Example

main()
{
    int date,time,tick;
    short day;
    ...
    ...
    ...
    _sysdate(0,&time,&date,&day,&tick);
    ...
    ...
}

See Also

_os_getime()
_os_julian()
F$Julian OS-9 for 68K Technical Manual
F$Time OS-9 for 68K Technical Manual
F_TIME OS-9 Technical Manual
**_sysdbg()**

**Call System Debugger**

**Syntax**

```c
#include <process.h>
int _sysdbg(void);
```

**Description**

`_sysdbg()` starts the system-level debugger, if one exists. This allows you to debug system state routines, such as drivers. The system-level debugger runs in system state and effectively stops timesharing whenever it is active. Only the super user can make this call.

If the system does not have a system-level debugger, the system is reset.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

`sys_clib.l`

**See Also**

- `_os_sysdbg()`  
- `F$SysDbg`  
- `F_SYSDBG`  
  
  *OS-9 for 68K Technical Manual*  
  *OS-9 Technical Manual*
system()
Shell Command Execution

Syntax

```c
#include <stdlib.h>
int system(const char *string);
```

Description

`system()` passes a string to the shell for execution. A null pointer may be used for the string to inquire whether a command processor exists.

The shell used to execute the string is either the current value of the `SHELL` environment variable or `shell`, if `SHELL` is unavailable.

- The thread enabled version of this function contains a cancel point. For more information refer to Using OS-9 Threads.

`string` is a pointer to the string to pass to the shell. (Input)

The string may be any valid shell command line.

- If `string` is a null pointer, `system()` returns non-zero only if a command processor is available.
- If `string` is not a null pointer, `system()` returns the exit status of the forked shell.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.l
system_addr()

Address Translation Function

Syntax

```c
#include <regs.h>
void *system_addr(void *addr);
```

Description

`system_addr()` returns the system-state version of the specified address. If the passed address is already in the system-state address space or address translation is not applicable on the host processor, it is returned untouched.

`system_addr()` returns the system-state version of the specified address.

`addr` is the address to be converted to its system-state address (Input).

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`cpu.l`

See Also

`user_addr()`
Syntax
#include <math.h>
double tan(double x);

Description
tan() returns the tangent of \( x \) (input). \( x \) is presumed to be in radians.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.1
tanh()
Hyperbolic Tangent Function

Syntax
#include <math.h>
double tanh(double x);

Description
tanh() returns the hyperbolic tangent of x (input).

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.1
## telldir()

### Syntax

```c
#include <dir.h>
long telldir(DIR *dirp);
```

### Description

`telldir()` returns the current location associated with the named directory stream.

- **dirp** is a pointer to the directory stream. (Input)

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

### Library

`sys_clib.l`

### See Also

- `closedir()
- opendir()
- readdir()
- readdir()
- rewinddir()
- seekdir()`
tempnam()

Create Names for Temporary Files

**Syntax**

```c
#include <UNIX/os9def.h>
char *tempnam(
    char *dir,
    char *pfx);
```

**Description**

tempnam() generates names that can safely be used for temporary files. tempnam() allows the user to control the choice of a directory.

tempnam() uses malloc() to get space for the constructed file name, and returns a pointer to this area. Thus any pointer value returned from tempnam() may serve as an argument to free(). If tempnam() cannot return the expected result for any reason, that is malloc() failed, or none of the above mentioned attempts to find an appropriate directory was successful, a NULL pointer is returned.

dir

points to the name of the directory in which the file is to be created. (Input)

If dir is NULL or points to a string which is not a name for an appropriate directory, “.” is used. This entire sequence can be preempted by setting the environment variable TMPDIR, whose value is the name of the desired temporary-file directory.

pfx

Many applications prefer their temporary files to have certain favorite initial letter sequences in their names. Use the pfx argument for this. (Input)

This argument may be NULL or point to a string to be used as the first few characters of the temporary-file name.

**Attributes**

Operating System: OS-9 and OS-9 for 68K

State: User and System

Threads: Safe

Re-entrant: No

**Library**

unix.l
Syntax

\#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetent(
    char *bufptr,
    const char *name);

Description

tgetent() extracts the entry for the specified terminal type. The entry is placed in a buffer. The buffer size must be at least 1,024 characters and must remain intact for all subsequent calls to tgetnum(), tgetflag(), and tgetstr().

If the termcap file cannot be opened, -1 is returned. If the terminal name cannot be found, 0 is returned. If the terminal name is found, 1 is returned and the data is placed in the buffer.

By defining the shell environment variables TERMCAP and TERM, you can modify the behavior of tgetent(). If TERMCAP is found in the environment and its value string begins with a slash (/), that string is used as the path to the file to use instead of the default termcap file. If the value string does not begin with a slash and name is the same as the environment string for TERM, tgetent() uses this file to search for the terminal entry. This mode is useful for testing or for custom termcap definitions. If the TERMCAP value string does not begin with a slash, the string is used as the termcap string instead of reading a file.

tgetent() must be called before any of the other termcap library functions.

bufptr

is a pointer to the buffer in which to place the entry. (Output)

name

is a pointer to the name of the terminal type. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
termlib.l

See Also
tgetflag() tgetnum() tgetstr()
tgetflag()
Check Terminal Capability Presence

Syntax
#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetflag(const char *id);

Description
tgetflag() returns 1 if the capability ID was specified for the terminal type and 0 if not specified for the terminal type.
tgetent() must be called before any of the other termcap library functions.

id is a pointer to the capability ID. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
termlib.l
**Syntax**

```c
#include <termcap.h>
extern char *BC;
extern char *UP;
int tgetnum(const char *id);
```

**Description**

`tgetnum()` returns the numeric value given for the capability ID.

`tgetent()` must be called before any of the other `termcap` library functions.

`id` is a pointer to the capability ID. (Input)

If `id` was not specified for the terminal type, -1 is returned.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No

**Library**

termlib.l
tgetstr()

Get Terminal Capability

Syntax
#include <termcap.h>
extern char *BC;
extern char *UP;
char *tgetstr(
    const char *id,
    char **strptr);

Description
tgetstr() places the string given for the capability ID into a buffer.
tgetent() must be called before any of the other termcap library functions.
tgetstr() returns null or 0 upon error.

Cursor motion and padding information are interpreted when the string is actually
output with tgoto() and tputs().

id
    is a pointer to the capability ID. (Input)

strptr
    is a pointer to the pointer to the buffer for the capability string. (Output)
    strptr is advanced past the last character of the returned data.

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         No

Library
termlib.l

See Also
tgoto()
tputs()


**Syntax**

```c
#include <termcap.h>
extern char *BC;
extern char *UP;
char *tgoto(
    const char *motion_string,
    int column,
    int line);
```

**Description**

tgoto() returns a string suitable for positioning a cursor on the terminal.
tgetent() must be called before any other termcap library functions.
The extern variables UP and BC (set to the up and bc capability strings, respectively) are used to avoid placing a null or newline character into the output string.
tgoto() returns a pointer to the translated motion string if the motion string was successfully created. Otherwise, the string "OOPS" is returned.

- **motion_string**
  - is a pointer to the string given by the cm capability. (Input)

- **column**
  - specifies the column destination for the cursor. (Input)

- **line**
  - specifies the line destination for the cursor. (Input)

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No

**Library**

termlib.l
Syntax
#include <time.h>
time_t time(time_t *timer);

Description

time() determines the current calendar time. The encoding of the value is unspecified.

(time()) returns the implementation’s best approximation to the current calendar time. The value (time_t)-1 is returned if the calendar time is unavailable. If timer is not a null pointer, the return value is also assigned to the object to which it points.

timer is a returned value. It is a pointer to the calendar time. (Output)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also
getenv()
_os_getime()
_os9_getime()
times()  
Get Process Times

Syntax
#include <UNIX/times.h>
int times(struct tms *buffer);

Description
	 times() returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/60 seconds.

buffer (output) points to the following structure:

struct tms {
    clock_t   tms_utime;   /* user time */
    clock_t   tms_stime;   /* system time */
    clock_t   tms_cutime;  /* user time, children */
    clock_t   tms_cstime;  /* system time, children */
} ;

This information comes from the calling process.

tms_utime is the CPU time used while executing instructions in the user space of the calling process.
tms_stime is the CPU time used by the system on behalf of the calling process.
tms_cutime and tms_cutimes are not implemented and set to zero by times().

Attributes

Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         No

Library

unix.l
Syntax

```c
#include <stdio.h>
FILE *tmpfile(void);
```

Description
tmpfile() creates a temporary binary file that is automatically removed when it is closed or at program termination. If the program terminates abnormally, temporary files are not removed. The file is opened for update with "wb+" mode.
tmpfile() returns a pointer to the stream of the file that it created. If the file cannot be created, tmpfile() returns a null pointer.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library
clib.l

See Also
fopen()
tmpnam()
Generate Unique Valid Filename

Syntax

#include <stdio.h>
char *tmpnam(char *filename);

Description
tmpnam() generates a string. The string is a valid file name and is not the same as the name of an existing file. tmpnam() generates a different string each time it is called up to TMP_MAX times. If it is called more than TMP_MAX times, it returns a null pointer.

filename is a pointer to the generated file name. (Output)

- If filename is a null pointer, tmpnam() leaves its result in an internal static object and returns a pointer to that object. Subsequent calls to tmpnam() modify the same object.
- If filename is not a null pointer, it is assumed to point to an array of at least L_tmpnam chars. tmpnam() writes its result in that array and returns the parameter as its value.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.l
**Syntax**

```c
#include <ctype.h>
int toascii(int c);
int _toascii(int c);
```

**Description**

_toascii() returns a character with all bits that are not part of a standard ASCII character turned off. It is used for compatibility with other systems. Only _toascii() is available in strictly conforming ANSI mode.

_c

is the returned character. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

**Library**

clib.l

**See Also**

getc(), getchar()
_tolower()

Convert Character to Lowercase

Syntax
#include <ctype.h>
int _tolower(int c);

Description
_tolower() is a macro that changes the uppercase parameter to lowercase. The parameter must be uppercase or the results are unpredictable and useless. Use tolower() if the parameter is not guaranteed to be uppercase.

_tolower() is hard-coded to the ASCII character set. Changing the locale does not cause _tolower() to recognize a non-ASCII character. You must use tolower() if you are not using ASCII characters.

\( c \)

is the character to convert. (Input)

Only the lower 8 bits of \( c \) are used.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
ctype.h.

See Also
tolower()
toupper()
_isascii(), isascii()
**tolower()**

Convert Character to Lowercase

**Syntax**

```c
#include <ctype.h>
int tolower(int c);
```

**Description**

tolower() converts an uppercase letter to the corresponding lowercase letter.

If the parameter `c` is such that `islower(c)` returns a non-zero value (is uppercase), `tolower()` returns the corresponding character. Otherwise, `c` is returned unchanged.

- `c` is the character to convert. (Input)

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

**Library**

clib.l

**See Also**

`islower()`
**_toupper()**

Convert Character to Uppercase

### Syntax

```c
#include <ctype.h>
int _toupper(int c);
```

### Description

`_toupper()` is a macro that changes the lowercase parameter to uppercase. The parameter *must* be lowercase or the results are unpredictable and useless. Use `toupper()` if the parameter is not guaranteed to be lowercase.

`_toupper()` is hard-coded to the ASCII character set. Changing the locale does not cause `_toupper()` to recognize a non-ASCII character. You must use `toupper()` if you are not using ASCII characters.

- `c` is the character to convert. (Input)
  - Only the lower 8 bits of `c` are used.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

### Library

`ctype.h`

### See Also

- `tolower`
- `toupper`
- `_isascii`, `isascii()`
toupper()

Convert Character to Uppercase

Syntax

```c
#include <ctype.h>
int toupper(int c);
```

Description
toupper() converts a lowercase letter to the corresponding uppercase letter. If the parameter `c` is such that `isupper(c)` returns a non-zero value (is uppercase), `toupper()` returns the corresponding character. Otherwise, `c` is returned unchanged.

- `c` is the character to convert. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.l

See Also

isupper()
tputs()

Output Capability String

Syntax

```
#include <termcap.h>
extern char *BC;
extern char *UP;
void tputs(
    const char *str,
    int lines_affected,
    int (*outfunc)(char));
```

Description

`tputs()` is used to output the capability strings to the terminal. `tputs()` decodes leading padding information from the `str` string.

`tgetent()` must be called before any of the other `termcap` library functions.

- **str**
  - is a pointer to the string. (Input)

- **lines_affected**
  - is the number of lines the operation affects. (Input)
  - This should be set to 1 if not applicable.

- **outfunc**
  - is a pointer to a function called by `tputs()` to output each successive character of the `str` string. (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

termlib.l

See Also

`_gs_opt()`
tsleep()
Sleep for Specified Interval

Syntax
#include <signal.h>
int tsleep(unsigned svalue);

Description
tsleep() deactivates the calling process for a specified interval. For values greater than one, \texttt{svalue} is considered a tick count to sleep, if the high bit is clear. If the high bit of \texttt{svalue} is set, the remaining 31 bits are considered the number of 256ths of a second to sleep. If the sleeping process is awakened prematurely by a signal, tsleep() returns the number of ticks remaining to sleep.

\texttt{svalue} specifies the interval below. (Input)
- If \texttt{svalue} is 0, the process sleeps indefinitely.
- If \texttt{svalue} is 1, the process gives up the current time slice.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
sys_clib.l

See Also
\_os\_sleep()
\_os9\_sleep()
F\$Sleep
F\_SLEEP

The thread enabled version of this function contains a cancel point. For more information see \textit{Using OS-9 Threads}.
ungetc()

Unget Character

Syntax

```c
#include <stdio.h>
int ungetc(
    int c,
    FILE *stream);
```

Description

`ungetc()` pushes a character, converted to an `unsigned char`, back onto the input stream. The pushed-back character is returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call with the stream pointed to by `stream` to a file positioning function (`fseek()`, `fsetpos()`, or `rewind()`) discards any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

One character of pushback is guaranteed. If `ungetc()` is called too many times on the same stream without an intervening read or file positioning operation on that stream, the operation may fail.

A successful call to `ungetc()` clears the end-of-file indicator for the stream. The value of the file position indicator for the stream after reading or discarding all pushed-back characters is the same as it was before the characters were pushed back. For a text stream, the value of the file position indicator after a successful call to `ungetc()` is unspecified until all pushed-back characters are read or discarded. For a binary stream, the file position indicator is decremented by each successful call to `ungetc()`. If the value was zero before a call, it is indeterminate after the call.

`ungetc()` returns the character pushed back after conversion. If the conversion fails, `EOF` is returned.

`c`  
is the character to push back. (Input)

If `c` is `EOF`, the operation fails and the input stream is unchanged.

`stream`  
is a pointer to the C I/O `FILE` structure. (Input)

Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** No
- **Standards:** ANSI
Library
clib.l

See Also
getc(), getchar()
fopen()
fsellk()
fsetpos()
rewind()
setbuf()
**link()**

Unlink (Delete) File

**Syntax**

```c
#include <modes.h>
int unlink(const char *name);
```

**Description**

`unlink()` decrements the link count of a file. When the link count reaches zero (0), the space occupied by the file on the disk is freed and the file no longer exists.

**name**

is a pointer to the name of the file. (Input)

If successful, `unlink()` returns 0. Otherwise, -1 is returned and the appropriate error code is placed in the global variable `errno`.

**Attributes**

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes

**Library**

sys_clib.l

**See Also**

- `_os_delete()`
- I$Delete
- I_DELETE

OS-9 for 68K Technical Manual
OS-9 Technical Manual

OS-9 for 68K does not yet support multiple links to a file. `unlink()` always causes the file to be removed from the disk. Attempting to delete an open file by the calling (or another) process results in an `EOS_SHARE` error.
Unlink (Delete) File

Syntax

```c
#include <modes.h>
int unlinkx(
    const char *name,
    int mode);
```

Description

`unlinkx()` decrements the link count of a file. If successful, `unlinkx()` returns 0. Otherwise, -1 is returned and the appropriate error code is placed in the global variable `errno`.

> OS-9 for 68K does not support multiple links to a file. `unlinkx()` always causes the file to be removed from the disk. Attempting to delete an open file by the calling (or another) process results in an `EOS_SHARE` error.

name

is a pointer to the name of the file. (Input)

mode

specifies the file’s access mode. (Input)

If the execution bit (`FAM_EXEC` or `S_IEXEC`) in the mode is set, the path is assumed to be based in the current execution directory. The header file `modes.h` defines the legal `mode` values. When the link count reaches zero, the space occupied by the file on the disk is freed and the file no longer exists. Only the lower 16 bits of `mode` are used.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library

`sys_clib.l`

See Also

_-os_delete()
I$Delete
I_DELETE

OS-9 for 68K Technical Manual
OS-9 Technical Manual
user_addr()  
Address Translation Function

Syntax
#include <regs.h>
void *user_addr(void *addr);

Description
user_addr() returns the user-state version of the specified address. If the passed address is already in the user-state address space or address translation is not applicable on the host processor, it is returned untouched.

user_addr() returns the user-state version of the specified address.

addr
is the address to be converted to its user-state address. (Input)

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Re-entrant: Yes

Library
cpu.l

See Also
system_addr()
utimes()
Change Access and Modification Times

Syntax
#include <sys/time.h>
int utimes(
    const char *path,
    const struct timeval *times);

Description
The access and modification times of the file named by path or referenced by fd are changed as specified by the argument times.

If times is NULL, the access and modification times are set to the current time. The caller must be the owner of the file, have permission to write the file, or be the super-user.

If times is non-NULL, it is assumed to point to an array of two timeval structures. The access time is set to the value of the first element, and the modification time is set to the value of the second element. The caller must be the owner of the file or be the super-user. In either case, the inode-change-time of the file is set to the current time.

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and errno is set to indicate the error.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes

Library
unix.l

See Also
_os_close()
_os_gs_devtyp()
_os_gs_fd()
_os_open()
_os_ss_fd()
localtime()
mktime()
time()
va_arg()  
Get Parameter in Variable Parameter List

Syntax
#include <stdarg.h>

type va_arg(va_list ap, type);

Description
va_arg() is a macro associated with the stdarg.h header file. stdarg.h provides a method for stepping through a list of function parameters whose length and type are unknown.

va_arg() expands into an expression. The expression returns a value that has the type and value of the next unnamed parameter. It also modifies ap such that the next call to va_arg() returns the next parameter. ap must have been previously initialized using va_start(). If there is no actual next parameter or if type is not compatible with the type of the actual next parameter (as promoted according to the default parameter promotions), the behavior is undefined.

ap     is an object representing the next unnamed parameter. (Input)
type   is a type name. (Input)

Attributes
Operating System:   OS-9 and OS-9 for 68K
State:              User and System
Threads:            Safe
Re-entrant:         Yes
Standards:          ANSI

Library
clib.1

See Also
va_start()
va_end()

End References to Current Variable Parameter List

Syntax
#include <stdarg.h>
void va_end(va_list ap);

Description
va_end() is a macro associated with the stdarg.h header file. stdarg.h provides a method for stepping through a list of function parameters whose length and type are unknown.

va_end() expands into an expression. The expression facilitates a normal return from the function whose variable parameter list was referred to by the expansion of the va_start() call that initialized ap.

va_end() may modify ap so that you can no longer use it without an intervening call to va_start(). If there is no corresponding call to va_start() or if va_end() is not called before the return, the behavior is undefined. va_end() returns no value.

ap
is an object representing the next unnamed parameter. (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library
clib.l

See Also
va_start()
# va_start()

## Initialize Variable Parameter List

### Syntax

```c
#include <stdarg.h>

void va_start(va_list ap, parmN);
```

### Description

`va_start()` is a macro associated with the `stdarg.h` header file. `stdarg.h` provides a method for stepping through a list of function parameters whose length and type are unknown.

- **ap**
  - is an object representing the next unnamed parameter. (Input)

- **parmN**
  - identifies the right-most parameter in variable parameter list in the function definition (the parameter just before the `, ...`). (Input)

  If `parmN` is declared with one of the following types, the behavior is undefined:
  - The register storage class
  - A function type
  - An array type
  - A type that is not compatible with the type that results after applying the default parameter promotions. For example, `char`, `short`, and `float`.

### Attributes

- **Operating System:** OS-9 and OS-9 for 68K
- **State:** User and System
- **Threads:** Safe
- **Re-entrant:** Yes
- **Standards:** ANSI

### Library

`clib.l`
Syntax

```c
#include <UNIX/err.h>

void verr(int eval, const char *fmt, va_list args);
void verrx(int eval, const char *fmt, va_list args);
```

Description

The `err()` family of functions, to which these functions belong, displays a formatted error message on the standard error output. In all cases, the last component of the program name, a colon character, and a space are output. If the `fmt` argument is not NULL, the formatted error message is output. In the case of the `verr()` function, the error message string affiliated with the current value of the global variable `errno` is output next, preceded by a colon character and a space if `fmt` is not NULL. In all cases, the output is followed by a newline character.

These functions do not return, rather they exit with the value of the argument `eval`.

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

Library

`unix.l`

See Also

`err(), errx()`
Syntax
#include <stdarg.h>
#include <stdio.h>
int vfprintf(
    FILE *stream,
    const char *format,
    va_list arg);

Description
vfprintf() performs formatted output to a stream. It converts, formats, and prints any parameters as indicated by the control string. vfprintf() is similar to fprintf(). However, arg replaces the variable parameter list.

vfprintf() does not call va_end(). vfprintf() returns the number of characters transmitted. If an output error occurs, a negative value is returned.

The sclib.i and sclib.il libraries contain a smaller version of the vfprintf() function. Refer to the Using Ultra C/C++ manual for more information about the small library functions.

stream
    is a pointer the C I/O FILE structure. (Input)

format
    is a pointer to a control string. (Input)

arg
    is initialized by va_start() (and possibly subsequent va_arg() calls). (Input)

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library
clib.i
**Example**

The following example shows the use of `vfprintf()` in a general error-reporting routine.

```c
#include <stdarg.h>
#include <stdio.h>

void error(char *function_name, char *format, ...)
{
    va_list args;
    va_start(args, format); /* print out name of function */
    /* causing error */
    fprintf(stderr, "ERROR in %s ", function_name);
    /* print out remainder of message */
    vfprintf(stderr, format, args);
    va_end(args);
}
```

**See Also**

- `fprintf()`
- `va_arg()`
- `va_end()`
- `va_start()`
vprintf()
Print to Standard Output

Syntax

```c
#include <stdarg.h>
#include <stdio.h>
int vprintf(
    const char *format,
    va_list arg);
```

Description

vprintf() performs formatted output to stdout. It converts, formats, and prints any parameters as indicated by the control string. vprintf() is similar to printf(). However, arg replaces the variable parameter list.

vprintf() does not call va_end().

vprintf() returns the number of characters transmitted. If an output error occurs, a negative value is returned.

format
is a pointer to a control string. (Input)

arg
is initialized by va_start() (and possibly subsequent va_arg() calls). (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No
Standards: ANSI

Library

clib.l

See Also

fprintf() printf() va_arg() va_end() va_start()
vsnprintf()

Write Formatted Output to Character Array

Syntax
#include <stdarg.h>
#include <stdio.h>

int vsnprintf( char *buf,
               size_t count,
               const char* format,
               va_list arg );

Description
The vsnprintf() function is equivalent to snprintf(), with a variable argument list. It writes formatted output to a character array, up to a given maximum number of characters.

vsnprintf() returns the number of characters generated by expanding the format string, regardless of whether or not the number generated exceeds that specified by count.

buf is the location in which vsnprintf() stores the data controlled by format (Output)

count specifies the maximum number of characters allowed in the array, including a terminating null character (Input)

format is the control string under which vsnprintf() formats data (Input)

arg is a variable argument list (Input)

For details on the format string, refer to the fprintf() function.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library
sys_clib.l
Examples
The following shows the use of \texttt{vsnprintf()} in a general error message routine:

```c
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

char msgbuf[80];

cchar *fmtmsg( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    strcpy( msgbuf, "Error: " );
vsnprintf( &msgbuf[7], 80-7, format, arglist );
    va_end( arglist );
    return( msgbuf );
}

int main( void )
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s\n", msg );

    return 0;
}
```

See Also
\texttt{fprintf()}
\texttt{printf()}
\texttt{snprintf()}
\texttt{sprintf()}
\texttt{va_arg()}
\texttt{va_end()}
\texttt{va_start()}
\texttt{vfprintf()}
\texttt{vprintf()}
\texttt{vsprintf()}
vsprintf()
Print to String

Syntax

```c
#include <stdarg.h>
#include <stdio.h>
int vsprintf(
    char *str,
    const char *format,
    va_list arg);
```

Description

vsprintf() performs formatted output to a string. It converts, formats, and prints any parameters as indicated by the control string. vsprintf() is similar to sprintf(). However, arg replaces the variable parameter list.

dsprintf() does not call va_end(). If copying takes place between overlapping objects, the behavior is undefined.

dsprintf() returns the number of characters written in the array, not counting the terminating null character.

The sclin.l and scil il libraries contain a smaller version of the vprintf() function. Refer to the Using Ultra C/C++ manual for more information.

str
    is a pointer to a string. (Output)

format
    is a pointer to a control string. (Input)

arg
    is initialized by va_start() (and possibly subsequent va_arg() calls). (Input)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No
- Standards: ANSI

Library

clib.l

See Also

fprintf() sprintf()
va_arg() va_end() va_start()
Syntax

#include <UNIX/err.h>

void vwarn(const char *fmt, va_list args);
void vwarnx(const char *fmt, ...);

Description

The `vwarn()` family of functions displays a formatted error message on the standard error output. In all cases, the last component of the program name, a colon character, and a space are output. If the `fmt` argument is not NULL, the formatted error message is output. In the case of the `vwarn()` function, the error message string affiliated with the current value of the global variable `errno` is output next, preceded by a colon character and a space if `fmt` is not NULL. In all cases, the output is followed by a newline character.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: No

Library

unix.l

See Also

warn(), warnx()
wait()

Wait for Process Termination

Syntax

```c
#include <process.h>
int wait(unsigned int *status);
int wait(0);
```

Description

`wait()` suspends the current process until a child process has terminated.

`wait()` returns the process ID of the terminating child process and places the exit status of that process in the `unsigned int` pointed to by `status`. If `status` is passed as zero, no child status is available to the caller.

The lower 16 bits of the status value contains the parameter of the `exit()` or `_exit()` call as executed by the child process, or the signal number if it was interrupted with a signal. A normally terminating C program with no explicit call to `exit()` in `main()` returns a status equal to the returned value of `main()`.

`wait()` returns:

- `-1` if there is no child process for which to wait.
- `0` if a signal was received before a child process terminated.

The status codes used in the operating system may not be compatible with other operating systems. A wait must be executed for each child process created.

Status

is a pointer to a storage location where the exit status of the terminating child process is stored. (Output)

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes

Library

`sys_clib.l`

See Also

waitpid()

Syntax

```c
#include <sys/wait.h>
pid_t waitpid(
pid_t     pid,
int       *stat_loc,
int       options);
```

Description

waitpid() allows a thread to wait for a process and get its exit status.

If or when a child’s exit status is available waitpid() returns the process ID and, if
stat_loc (output) is non-NULL, writes the exit status at stat_loc. If a blocking
waitpid() is interrupted by a signal it sets errno to EINTR and returns -1. If options
is WNOHANG and the calling process has children, but none of the specified children
has available exit status, 0 is returned. Otherwise errno is set to the error number
and -1 is returned.

pid and options

The pid and options arguments determine how the wait is done. (Input)

For values of pid less than or equal to 0, the next available child process’
status is returned. For values of pid greater than 1 (1 is an invalid process ID)
the exit status of the specified process is returned. The options parameter can
either be 0 to indicate a blocking wait or WNOHANG to indicate a non-blocking
wait.

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User
Threads: Safe
Re-entrant: Yes

Library

sys_clib.l

Possible Errors

ECHLD The calling process has no children or not the specific child requested.
EINTR A blocking waitpid() was interrupted by a signal.
EINVAL The value of options is invalid.
Example

while (waitpid(child, &status, 0) == -1 && errno == EINTR)
    ;

See Also

wait()

pthread_join()

pthread_exit()
**Syntax**

```c
#include <UNIX/err.h>

void warn(const char *fmt, ...);
void warnx(const char *fmt, ...);
```

**Description**

The `warn()` family of functions displays a formatted error message on the standard error output. In all cases, the last component of the program name, a colon character, and a space are output. If the `fmt` argument is not NULL, the formatted error message is output. In the case of the `warn()` function, the error message string affiliated with the current value of the global variable `errno` is output next, preceded by a colon character and a space if `fmt` is not NULL. In all cases, the output is followed by a newline character.

**Attributes**

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: No

**Library**

`unix.lib`

**See Also**

`vwarn()`, `vwarnx()`
wcstombs()

Convert Sequence of Wide Chars to Multibyte Chars

Syntax

```c
#include <stdlib.h>
size_t wcstombs(
    char *str,
    const wchar_t *pwcs,
    size_t maxnum);
```

Description

`wcstombs()` converts a sequence of codes that correspond to multibyte characters from an array into a sequence of multibyte characters that begins in the initial shift state. `wcstombs()` stores these multibyte characters, stopping if a multibyte character would exceed the limit of `maxnum` total bytes or if a null character is stored. Each code is converted as if by a call to `wctomb()`, except that the shift state of `wctomb()` is not affected.

No more than `maxnum` bytes are modified in `str`. If copying takes place between overlapping objects, the behavior is undefined.

If a code is encountered that does not correspond to a valid multibyte character, `wcstombs()` returns `size_t - 1`. Otherwise, `wcstombs()` returns the number of bytes modified, not including a terminating null character, if any.

`wcstombs()` is supported for both the C and JAPAN locales.

`str` is a pointer to the string in which to store the multibyte characters. (Output)

`pwcs` is a pointer to the array from which to read. (Input)

`maxnum` specifies the maximum number of bytes to convert. (Input)

Attributes

Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Standards: ANSI

Library

clib1

See Also

`wctomb()`
Syntax

```c
#include <stdlib.h>

int wctomb(
    char *str,
    wchar_t wchar);
```

Description

`wctomb()` determines the number of bytes needed to represent the multibyte character corresponding to the code whose value is `wchar`. This includes any change in shift state. `wctomb()` stores the multibyte character representation in `str`, if `str` is not a null pointer. At most `MB_CUR_MAX` characters are stored.

- `wchar` is the multibyte character to be represented. (Input)
  - If `wchar` is zero, `wctomb()` is left in the initial shift state.

The returned value is never greater than `MB_CUR_MAX`.

- `wctomb()` is supported for both the C and JAPAN locales.
- `str` points to the string that stores a multibyte character representation. (Output)
  - If `str` is a null pointer, `wctomb()` returns:
    - a non-zero value if multibyte character encodings have state-dependent encodings
    - zero if multibyte character encodings do not have state-dependent encodings
  - If `str` is not a null pointer, `wctomb()` returns:
    - `-1` if the value of `wchar` does not correspond to a valid multibyte character
    - the number of bytes contained in the multibyte character corresponding to the value of `wchar`

Attributes

- Operating System: OS-9 and OS-9 for 68K
- State: User and System
- Threads: Safe
- Re-entrant: Yes
- Standards: ANSI

Library

clib.l
write()
Write Bytes to Path

Syntax
#include <modes.h>
int write(
    int path,
    const char *buffer,
    unsigned count);

Description
write() writes bytes to a path.
write() essentially performs a raw write; that is, the write is performed without translating characters. The characters are passed to the file (or device) as transmitted by the program.
write() returns the number of bytes actually written. If an error occurred, -1 is returned and the appropriate error code is placed in the global variable errno.
write() performs a "raw" write to the terminal; no linefeeds after returns are transmitted.

path
is the path number. (Input)
The path number is an integer which specifies one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to open(), creat(), create(), or dup().

buffer
is a pointer to space with at least count (input) bytes of memory from which write() obtains the data to write to the path. (Output)
It is guaranteed that at most count bytes are written. If fewer bytes are written, an error has occurred.

Attributes
Operating System: OS-9 and OS-9 for 68K
State: User and System
Threads: Safe
Re-entrant: Yes
Library
sys_clib.l

See Also
creat()
create()
dup()
open()
_os_write()
writeln()
I$Write
I_WRITE

OS-9 for 68K Technical Manual
OS-9 Technical Manual
writeln()

Write Bytes to Path

Syntax

```
#include <modes.h>
int writeln(
    int path,
    const char *buffer,
    unsigned count);
```

Description

`writeln()` writes bytes to a path.

`writeln()` causes output-filtering to take place such as outputting a linefeed after a carriage return or handling the page pause facility. `writeln()` writes, at most, one line of data. A carriage return indicates the end of a line. `writeln()` is the preferred call for writing to the terminal.

`writeln()` returns the number of bytes actually written. If an error occurred, -1 is returned and the appropriate error code is placed in the global variable `errno`.

`writeln()` stops when the carriage return is written, even if the byte count has not been exhausted.

![Important Note]

The thread enabled version of this function contains a cancel point. For more information refer to *Using OS-9 Threads*.

**path**

is the path number. (Input)

The path number is an integer which specifies one of the standard path numbers (0, 1, or 2) or a path number returned from a successful call to `open()`, `creat()`, `create()`, or `dup()`.

**buffer**

is a pointer to space with at least `count` (input) bytes of memory from which `writeln()` obtains the data to write to the path. (Output)

It is guaranteed that at most `count` bytes are written.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>OS-9 and OS-9 for 68K</td>
</tr>
<tr>
<td>State</td>
<td>User and System</td>
</tr>
<tr>
<td>Threads</td>
<td>Safe</td>
</tr>
<tr>
<td>Re-entrant</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Library
sys_clib.l

See Also
creat()
create()
dup()
open()
_os_writeln()
write()
I$Writeln
I_WRITELN

OS-9 for 68K Technical Manual
OS-9 Technical Manual
writev()
Write Output from iovcnt Buffers

Syntax
#include <UNIX/os9def.h>
int writev(
    unsigned int fd,
    struct iovec *iov,
    unsigned int iovcnt);

Description
writev() performs the same action as write(), but gathers the output data from the
iovcnt buffers specified by the members of the iov array: iov[0], iov[1], ..., iov[iovcnt-1]. If iovcnt is zero, any iov_len member of a passed iovec structure
is less than zero, or all the iov_len members of the passed iovec structures are zero,
writev() returns -1 and sets the global variable errno.

For writev(), the iovec structure is defined as follows.
struct iovec {
    caddr_t     iov_base;
    int iov_len;
};

Each iovec entry specifies the base address and length of an area in memory from
which data should be written. writev() always writes a complete area before
proceeding to the next.

Attributes
    Operating System:    OS-9 and OS-9 for 68K
    State:               User and System
    Threads:             Safe
    Re-entrant:          No

Library
    unix.l
Prototyping sys_clib.1 Functions

Before this topic is discussed in depth, it is important to note that the primary purpose of sys_clib.1 is to provide a library for compatibility with the Microware K&R C compiler. This allows people who wrote code for that compiler to use the Ultra C/C++ ANSI C compiler without altering their original code.

- It is strongly recommended that when writing new code, a programmer not use the sys_clib library functions.
- Table 1-37 provides a table to look up equivalent functions for the sys_clib functions.
Declarations and K&R Compiler vs. Prototypes and ANSI C

In the past, the concept of a function prototype did not exist. The closest equivalent concept was a "function declaration". The purpose of the function declaration was to "pre-define" what the return type of a function was so that some degree of type checking could be done on its use. An example of a function declaration is:

```c
struct direct *readdir();
```

This shows that the function "readdir" returns a pointer to a structure of the type `direct` (found in `dir.h`).

There is nothing in the declaration which mentions what type, or even how many, arguments the readdir function takes. As a result, no type checking was ever done on these function parameters. The "standard promotion" of arguments was always assumed to have occurred.

The "standard promotion" rules are quite simple, all types being passed in that are "narrower" than an int type, are "widened" to an int type, and all arguments that are of floating point type are "widened" to a double precision. As a natural consequence, all functions which are compiled in K&R mode (for instance, all the `sys_clib.l` functions) assume that the calling function has already promoted the function arguments to their widened forms. ANSI C prototypes are somewhat different. For example:

```c
int os9fork(char *, int, char *, short, short, int short);
```

This shows that the function "os9fork" returns an `int` and accepts the parameters specified. Of particular interest is the three "short" parameters. In ANSI mode, the compiler assumes that the prototype was visible to the calling program, and as a result, the values were NOT widened to `int`. If the prototype was not visible to the calling function, the values would have been widened. This could have potentially caused a problem (especially if the parameters were passed on the stack and NOT in registers). As a result, the Ultra C compiler has an option (`-cw`) which warns the user if a function is being called without a prototype for that function being visible.
Making Code and Function Prototypes ANSI Compliant

It has been a common practice as of late to take code and switch it over to ANSI-compliant code. This often entails prototyping many of the functions which used to be compiled with K&R C compilers. If this were done strictly as in the prototype from the "os9fork" function mentioned above, and the os9fork function was not recompiled with an ANSI compiler, things would not work since the function itself would be expecting the short arguments to be widened to ints and the calling program, after having seen the prototype, would send shorts.

There is really only one way to handle this in a manner that causes everything to still work and that is to prototype the functions to their widened types, as follows:

```c
int os9fork(char *, int, char *, int, int, int, int);
```

Unfortunately, this has the undesired side effect of implying that the fourth, fifth and seventh arguments are valid integers, when indeed only their lower 16 bits are valid. As a result, Microware has decided to optionally prototype these functions in system header files.

sys_clib.l functions affected by this include:

- os9exec()
- os9fork()
- os9forkc()
- os9kexec()