Copyright and publication information

This manual reflects version 4.2 Microware OS-9. 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

### Chapter 1: System Overview

* System Modularity ................................................................. 12
  - Level 1 -- The Kernel, Clock, and Init Modules .................. 12
  - Level 2 — IOMAN ................................................................. 12
  - Level 3 — File Managers ....................................................... 12
  - Level 4 — Device Drivers ..................................................... 13
  - Level 5 — Device Descriptors .............................................. 13

* I/O Overview ........................................................................... 13

* Memory Modules ...................................................................... 14
  - Basic Module Structure ...................................................... 15
  - The CRC Value .................................................................... 16
  - ROMed Memory Modules .................................................... 16
  - Module Header Definitions .................................................. 16
  - mh_com .............................................................................. 17

### Chapter 2: The Kernel

* Kernel Functions ..................................................................... 24

* System Call Overview ............................................................ 24
  - User-State and System-State .............................................. 24
  - Installing System-State Routines ......................................... 25

* Kernel System Call Processing ............................................... 26
  - Non-I/O Calls ..................................................................... 26
  - I/O Calls ............................................................................ 27

* Memory Management ............................................................... 27
  - OS-9 Memory Map ............................................................... 28
  - System Memory Allocation .................................................. 28
  - Operating System Object Code ........................................... 29
  - System Global Memory ....................................................... 29
  - System Dynamic Memory .................................................... 29
  - User Memory ...................................................................... 29

* Memory Fragmentation ............................................................ 30

* Colored Memory ..................................................................... 30
  - Colored Memory Definition List ........................................ 30
  - SSM and Cache ................................................................. 33
  - Cache List ......................................................................... 33
  - PowerPC Processors: SSM and Cache .................................. 33
  - Colored Memory in Homogenous Memory Systems ............ 39
  - System Performance ........................................................... 39
  - Reconfiguring Memory Areas ............................................. 39

* System Initialization ............................................................... 40
  - Init: The Configuration Module ........................................... 40
  - Extension Modules ............................................................. 40
Contents

Chapter 3: Interprocess Communication

Signals ................................................................. 48
  Signal Codes ......................................................... 48
  Signal Implementation ........................................ 49
    Non-Recursive Calling ....................................... 50
    Recursive Calling ............................................. 50
Alarms ............................................................... 52
  User-State Alarms ............................................... 52
  Cyclic Alarms ..................................................... 52
  Time of Day Alarms ............................................ 53
  Relative Time Alarms ......................................... 53
  System-State Alarms .......................................... 53
Events ............................................................... 54
  ev_str/ev_infostr .................................................. 55
  Wait and Signal Operations ................................. 57
  The F_EVENT System Call ................................... 58
Semaphores ........................................................ 59
  Semaphore States ............................................... 60
  Acquiring Exclusive Access ................................. 60
  Releasing Exclusive Access ................................ 60
Usemaphores ...................................................... 61
  P and V Operations ............................................... 61
  Reset .................................................................. 63
  The F_EVENT, F_USEMA System Call ....................... 63
Pipes ................................................................. 64
  Named and Unnamed Pipes .................................. 64
Operations on Pipes ............................................ 65
  Creating Pipes ..................................................... 65
  Opening Pipes ..................................................... 65
  Read/Readln ...................................................... 66
  Write/Writeln .................................................... 67
  Close .................................................................. 67
  Getstat/Setstat .................................................... 67
  GetStat Status Codes Supported by PIPEMAN ....... 67
  SetStat Status Codes Supported by PIPEMAN ........ 68
  Pipe Directories ................................................ 68
Data Modules ...................................................... 69
  Creating Data Modules ....................................... 69
  The Link Count .................................................. 69
  Saving to Disk .................................................... 69

Chapter 4: Subroutine Libraries and Trap Handlers

Subroutine Libraries .......................................... 72
  Installing and Executing Subroutine Libraries .......... 72
<table>
<thead>
<tr>
<th>Function/Expression</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_DTLK</td>
<td>136</td>
</tr>
<tr>
<td>F_DATTACH</td>
<td>138</td>
</tr>
<tr>
<td>F_DELLK</td>
<td>139</td>
</tr>
<tr>
<td>F_DELMDIR</td>
<td>140</td>
</tr>
<tr>
<td>F_DELTSK</td>
<td>141</td>
</tr>
<tr>
<td>F_DEXEC</td>
<td>142</td>
</tr>
<tr>
<td>F_DEXIT</td>
<td>145</td>
</tr>
<tr>
<td>F_DFORK</td>
<td>146</td>
</tr>
<tr>
<td>F_DFORKM</td>
<td>148</td>
</tr>
<tr>
<td>F_EVENT</td>
<td>150</td>
</tr>
<tr>
<td>F_EVENT, EV_ALLCLR</td>
<td>152</td>
</tr>
<tr>
<td>F_EVENT, EV_ALLSET</td>
<td>154</td>
</tr>
<tr>
<td>F_EVENT, EV_ANYCLR</td>
<td>156</td>
</tr>
<tr>
<td>F_EVENT, EV_ANYSET</td>
<td>158</td>
</tr>
<tr>
<td>F_EVENT, EV_CHANGE</td>
<td>160</td>
</tr>
<tr>
<td>F_EVENT, EV_CREAT</td>
<td>162</td>
</tr>
<tr>
<td>F_EVENT, EV_CREAT</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_DELET</td>
<td>166</td>
</tr>
<tr>
<td>F_EVENT, EV_DELET</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_INFO</td>
<td>168</td>
</tr>
<tr>
<td>F_EVENT, EV_LINK</td>
<td>170</td>
</tr>
<tr>
<td>F_EVENT, EV_LINK</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_PULSE</td>
<td>174</td>
</tr>
<tr>
<td>F_EVENT, EV_READ</td>
<td>176</td>
</tr>
<tr>
<td>F_EVENT, EV_RESET</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_SET</td>
<td>179</td>
</tr>
<tr>
<td>F_EVENT, EV_SETAND</td>
<td>181</td>
</tr>
<tr>
<td>F_EVENT, EV_SETOR</td>
<td>183</td>
</tr>
<tr>
<td>F_EVENT, EV_SETR</td>
<td>185</td>
</tr>
<tr>
<td>F_EVENT, EV_SETXOR</td>
<td>187</td>
</tr>
<tr>
<td>F_EVENT, EV_SIGNL</td>
<td>189</td>
</tr>
<tr>
<td>F_EVENT, EV_SIGNL</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_TRYWAIT</td>
<td>193</td>
</tr>
<tr>
<td>F_EVENT, EV_TRYWAIT</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_TSTSET</td>
<td>197</td>
</tr>
<tr>
<td>F_EVENT, EV_UNLNK</td>
<td>199</td>
</tr>
<tr>
<td>F_EVENT, EV_UNLNK</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_WAIT</td>
<td>201</td>
</tr>
<tr>
<td>F_EVENT, EV_WAIT</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_WAITTR</td>
<td>205</td>
</tr>
<tr>
<td>F_EXIT</td>
<td>207</td>
</tr>
<tr>
<td>F_FINDPD</td>
<td>209</td>
</tr>
<tr>
<td>F_FMOD</td>
<td>210</td>
</tr>
<tr>
<td>F_FORK</td>
<td>211</td>
</tr>
<tr>
<td>F_FORKM</td>
<td>214</td>
</tr>
<tr>
<td>F_GBLKMP</td>
<td>216</td>
</tr>
<tr>
<td>F_GETMDP</td>
<td>218</td>
</tr>
<tr>
<td>F_GETSYS</td>
<td>219</td>
</tr>
<tr>
<td>F_GMODADDR</td>
<td>221</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td>F_GPRDBT</td>
<td>222</td>
</tr>
<tr>
<td>F_GPRDSC</td>
<td>223</td>
</tr>
<tr>
<td>F_ICPT</td>
<td>224</td>
</tr>
<tr>
<td>F_ID</td>
<td>226</td>
</tr>
<tr>
<td>F_INITDATA</td>
<td>228</td>
</tr>
<tr>
<td>F_IRQ</td>
<td>229</td>
</tr>
<tr>
<td>F_LINK</td>
<td>231</td>
</tr>
<tr>
<td>F_LINKM</td>
<td>233</td>
</tr>
<tr>
<td>F_LOAD</td>
<td>235</td>
</tr>
<tr>
<td>F_MKMDIR</td>
<td>237</td>
</tr>
<tr>
<td>F_MEM</td>
<td>238</td>
</tr>
<tr>
<td>F_MODADDR</td>
<td>239</td>
</tr>
<tr>
<td>F_MOVE</td>
<td>240</td>
</tr>
<tr>
<td>F_NPROC</td>
<td>241</td>
</tr>
<tr>
<td>F_PERMIT</td>
<td>242</td>
</tr>
<tr>
<td>F_PROTECT</td>
<td>244</td>
</tr>
<tr>
<td>F_PRSNAM</td>
<td>246</td>
</tr>
<tr>
<td>F_RELLK</td>
<td>248</td>
</tr>
<tr>
<td>F_RETPD</td>
<td>249</td>
</tr>
<tr>
<td>F_RTE</td>
<td>250</td>
</tr>
<tr>
<td>F_SEND</td>
<td>251</td>
</tr>
<tr>
<td>F_SETCRC</td>
<td>253</td>
</tr>
<tr>
<td>F_SETSYS</td>
<td>254</td>
</tr>
<tr>
<td>F_SIGLNGJ</td>
<td>256</td>
</tr>
<tr>
<td>F_SIGMAK</td>
<td>258</td>
</tr>
<tr>
<td>F_SIGRESET</td>
<td>260</td>
</tr>
<tr>
<td>F_SIGRS</td>
<td>261</td>
</tr>
<tr>
<td>F_SLEEP</td>
<td>263</td>
</tr>
<tr>
<td>F_SLINK</td>
<td>265</td>
</tr>
<tr>
<td>F_SLINKM</td>
<td>267</td>
</tr>
<tr>
<td>F_SPRIOR</td>
<td>268</td>
</tr>
<tr>
<td>F_SRQMEM</td>
<td>269</td>
</tr>
<tr>
<td>F_SRTMEM</td>
<td>271</td>
</tr>
<tr>
<td>F_SSPD</td>
<td>272</td>
</tr>
<tr>
<td>F_SSVC</td>
<td>273</td>
</tr>
<tr>
<td>F_STIME</td>
<td>275</td>
</tr>
<tr>
<td>F_STRAP</td>
<td>277</td>
</tr>
<tr>
<td>F_USER</td>
<td>280</td>
</tr>
<tr>
<td>F_SYSDBG</td>
<td>281</td>
</tr>
<tr>
<td>F_SYSID</td>
<td>282</td>
</tr>
<tr>
<td>F_THEXIT</td>
<td>284</td>
</tr>
<tr>
<td>F_THFORK</td>
<td>285</td>
</tr>
<tr>
<td>F_THREAD</td>
<td>287</td>
</tr>
<tr>
<td>F_TIME</td>
<td>288</td>
</tr>
<tr>
<td>F_TLINK</td>
<td>289</td>
</tr>
<tr>
<td>F_TLINKM</td>
<td>291</td>
</tr>
<tr>
<td>F_UACCT</td>
<td>293</td>
</tr>
<tr>
<td>F_UNLINK</td>
<td>294</td>
</tr>
<tr>
<td>F_UNLOAD</td>
<td>295</td>
</tr>
</tbody>
</table>
F_VMODUL ..............................................................................................................................296
F_WAIT ......................................................................................................................................298
F_WAITID ..................................................................................................................................300
F_WAITLK ..................................................................................................................................302
F_YIELD ......................................................................................................................................304
I_ALIAS .......................................................................................................................................305
I_ATTACH .....................................................................................................................................306
I_CHDIR ......................................................................................................................................308
I_CIOPROC ..................................................................................................................................310
I_CLOSE .......................................................................................................................................311
I_CONFIG .....................................................................................................................................312
I_CREATE .....................................................................................................................................313
I_DELETE .....................................................................................................................................315
I_DETACH .....................................................................................................................................316
I_DUP ..........................................................................................................................................317
I_GETDL .......................................................................................................................................319
I_GETPD .......................................................................................................................................320
I_GETSTAT ....................................................................................................................................321
I_GETSTAT, SS_COPYPD ...............................................................................................................323
I_GETSTAT, SS_CSTATS ...............................................................................................................324
I_GETSTAT, SS_DEVNAME ...........................................................................................................325
I_GETSTAT, SS_DEVOPT ...............................................................................................................326
I_GETSTAT, SS_DEVTYPE ............................................................................................................327
I_GETSTAT, SS_DISKFREE ..........................................................................................................328
I_GETSTAT, SS_DSIZE ................................................................................................................329
I_GETSTAT, SS_EDT ....................................................................................................................330
I_GETSTAT, SS_EOF ....................................................................................................................331
I_GETSTAT, SS_FD .....................................................................................................................332
I_GETSTAT, SS_FdAddr ...............................................................................................................333
I_GETSTAT, SS_FDINFO .............................................................................................................334
I_GETSTAT, SS_LUOPT ...............................................................................................................335
I_GETSTAT, SS_PARITY ...............................................................................................................336
I_GETSTAT, SS_PATHOPT ..........................................................................................................337
I_GETSTAT, SS_POS ...................................................................................................................338
I_GETSTAT, SS_READY ...............................................................................................................339
I_GETSTAT, SS_SIZE ..................................................................................................................340
I_GIOPROC ..................................................................................................................................341
I_IODEL .......................................................................................................................................342
I_IOEXIT .......................................................................................................................................343
I_IOFORK .....................................................................................................................................344
I_MAKDIR .....................................................................................................................................345
I_IOEXIT .......................................................................................................................................347
I_RDALST .....................................................................................................................................349
I_READ .........................................................................................................................................350
I_READLN ......................................................................................................................................352
I_SEEK .........................................................................................................................................354
I_SETSTAT ....................................................................................................................................355
I_SETSTAT, SS_ATTR ..................................................................................................................357
I_SETSTAT, SS_BREAK .................................................................................................................358
Appendix A: Example Code

Sysgo ............................................................... 392
Signals: Example Program ................................................................. 394
Alarms: Example Program ................................................................. 396
Events: Example Program ................................................................. 398
Semaphores: Example Program............................................................ 400
The Subroutine Library................................................................. 402
  Subroutine Module ................................................................. 405
    root psect File ................................................................. 405
    function File ........................................................................... 406
  Application Call into a Subroutine Module ........................................ 406
    Initialization ................................................................. 406
Trap Handlers ................................................................. 408
  trapc.a ................................................................. 408
  thandler.c ................................................................. 410
  tcall.c ................................................................. 411
  ttest.c ................................................................. 412
This chapter provides a general overview of OS-9® system modularity, I/O processing, memory modules, and program modules. It includes the following topics:

- **System Modularity**
- **I/O Overview**
- **Memory Modules**
System Modularity

OS-9 has five levels of modularity. These are illustrated in Figure 1-1.

Figure 1-1. OS-9 Module Organization

Level 1 -- The Kernel, Clock, and Init Modules

The kernel provides basic system services, including process control and resource management. The clock module is a software handler for the specific real-time clock hardware. The kernel uses the Init module as an initialization table during system startup.

Level 2 — IOMAN

IOMAN coordinates the input/output (I/O) system by passing I/O requests to the appropriate file managers.

For specific information about IOMAN, file managers, device drivers, and device descriptors, refer to I/O Overview, Chapter 2, The Kernel, and the OS-9 Porting Guide.

Level 3 — File Managers

File managers process I/O requests for similar classes of I/O devices. Refer to the I/O Overview in this chapter for a list of the file managers Microware currently supports for OS-9.
Level 4 — Device Drivers

Device drivers handle the basic physical I/O functions for specific I/O controllers. Standard OS-9 systems are typically supplied with a disk driver, serial port drivers for terminals and serial printers, and a driver for parallel printers. You can add customized drivers of your own design or purchase drivers from a hardware vendor.

Level 5 — Device Descriptors

Device descriptors are small tables that associate specific I/O ports with their logical name, device driver, and file manager. These modules also contain the physical address of the port and initialization data.

One important component not shown is the shell, which is the command interpreter. The shell is an application program, not part of the operating system, and is described in the Using OS-9 manual.

For a list of the specific modules comprising OS-9 for your system, use the `ident` utility on the `sysboot` file.

Although all modules can be resident in ROM, the system bootstrap module is usually the only ROMed module in disk-based systems. All other modules are loaded into RAM during system startup.

I/O Overview

The OS-9 kernel does not directly process I/O requests. Instead, the kernel passes I/O requests to the I/O manager (IOMAN), and IOMAN passes requests to the appropriate file managers. Microware includes the following file managers in the Microware OS-9 for Embedded Systems and Board Level Solution package:

Table 1-1. File Managers

<table>
<thead>
<tr>
<th>File Manager</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBF</td>
<td>The Random Block File manager handles I/O for random-access, block-structured devices such as disks and hard drives.</td>
</tr>
<tr>
<td>SCF</td>
<td>The Sequential Character File manager handles I/O for sequential-access, character-structured devices such as terminals, printers, and modems.</td>
</tr>
<tr>
<td>SBF</td>
<td>The Sequential Block File manager handles I/O for sequential-access, block-structured devices.</td>
</tr>
<tr>
<td>PIPEMAN</td>
<td>The Pipe file Manager handles I/O for interprocess communications through memory buffers called pipes.</td>
</tr>
<tr>
<td>PCF</td>
<td>The PC file manager handles reading and writing to PC-DOS disks.</td>
</tr>
</tbody>
</table>

For more information about these file managers, refer to Chapter 2, The Kernel, or the OS-9 Porting Guide.

Microware also supports additional communication file managers. Refer to the SoftStax® and Lan Communications manual sets for details.
Figure 1-2 illustrates how an OS-9 I/O request is processed.

**Figure 1-2. Processing an OS-9 I/O Request**

1. The user makes a request for data/status.

2. The kernel determines the request is an I/O request and passes it to IOMan.

3. IOMan identifies and validates the I/O request and determines the appropriate file manager, device driver, and other necessary resources. Then, IOMan passes the request to the appropriate file manager.

4. The file manager further validates the request and performs device-independent processing. It also calls the device driver for hardware interaction, as necessary.

5. The device driver performs device-specific processing and transfers the data/status back to the file manager.

6. The file manager monitors and processes the data/status.

7. The kernel and IOMan work with the file manager to return the data/status to the user.

8. The user receives the data/status.

---

**Memory Modules**

OS-9 is unique because it manages both the physical assignment of memory to programs and the logical contents of memory by using memory modules. A memory module is a logical, self-contained program, program segment, or collection of data.

OS-9 supports nine predefined module types and enables you to define your own module types. Each type of module has a different function. The predefined module types are defined in the `m_tylan` field of the module header definition.

Modules do not have to be complete programs or written in machine language. Modules simply have to be re-entrant, position independent, and conform to the basic module structure described in the next section.
OS-9 is based on a programming style called re-entrant code. That is, code that does not modify itself. This allows two or more different processes to share one copy of a module simultaneously. The processes do not effect each other, provided each process has an independent area for its variables.

Almost all OS-9 family software is re-entrant and uses memory efficiently. For example, a screen editor may require 26K of memory to load. If a request to run the editor is made while another user (process) is running it, OS-9 allows both processes to share the same copy, saving 26K of memory.

Data modules are an exception to the no-modification restriction. However, careful coordination is required for several processes to update a shared data module simultaneously.

A position-independent module is in no way dependent on, or aware of where it is loaded in memory. This enables OS-9 to load the program wherever memory space is available. In many operating systems, the user must specify a load address to place the program in memory. OS-9 determines an appropriate load address only when the program is started.

OS-9 compilers and interpreters automatically generate position-independent code. In assembly language programming, however, you must insure position independence by avoiding absolute address modes. Alternatives to absolute addressing are described in the Assembler and Linker chapters of the Using Ultra C/C++ manuals.

**Basic Module Structure**

Each module has three parts: a module header, a module body, and a CRC value as shown in Figure 1-3.

![Figure 1-3. Basic Memory Module Format](image)

The module header contains information describing the module and its use. It is defined in assembly language by a `psect` directive. The linker creates the header at link time. The information contained in the module header includes the module name, size, type, language, memory requirements, and entry point. For specific information about the structure and individual fields of the module header, refer to the Module Header Definitions section in this chapter.
The module body contains initialization data, program instructions, and constant tables. The last three bytes of the module hold a CRC (cyclic redundancy check) value used to verify the module integrity when the module is loaded into memory. The linker creates the CRC at link time.

The CRC Value

A CRC (cyclic redundancy check) value is at the end of all modules, except data modules. The CRC, which is used to validate the entire module, is an error checking method used frequently in data communications and storage systems. The CRC is also a vital part of the ROM memory module search technique. It provides a high degree of confidence that programs in memory are intact before execution and is an effective backup for the error detection systems of disk drives and memory systems.

In OS-9, a 24-bit CRC value is computed over the entire module starting at the first byte of the module header and ending at the byte just before the CRC. OS-9 compilers and linkers automatically generate the module header and CRC values. If required, a user program can use the \texttt{F_CRC} system call to compute a CRC value over any specified data bytes. For a full description of how \texttt{F_CRC} computes a CRC value, refer to the description of the \texttt{F_CRC} call in Chapter 6, OS-9 System Calls.

In the case of data modules, the CRC value is not calculated when created. The CRC must be calculated and set on a data module before that module is loaded into memory.

OS-9 cannot recognize a module with an incorrect CRC value. For this reason, you must update the CRC value of a module modified in any way, or the module cannot be loaded from disk or located in ROM. Use the OS-9 \texttt{fixmod} utility to update the CRC of a modified module.

ROMed Memory Modules

When OS-9 starts after a system reset, the kernel searches for modules in ROM. The kernel detects the modules by looking for the module header sync code (for example, \texttt{0xf00d} for PowerPC processors). When this byte pattern is detected, the header parity is checked to verify a correct header. If this test succeeds, the module size is obtained from the header and a 24-bit CRC is computed over the entire module. If the CRC is valid, the module is entered into the module directory.

OS-9 links to all of its component modules found during the search. All ROMed modules present in the system at startup are automatically included in the system module directory. This enables you to create partially or completely ROM-based systems. Any non-system module found in ROM is also included. This enables user-supplied software to be located during the start-up process and entered into the module directory.

Module Header Definitions

The structure definition for a module header is listed here, followed by a description of each field.
The module header structure is contained in the header file module.h.

**Declaration**

```c
typedef struct mh_com {
    u_int16    m_sync,     /* sync bytes */
    m_sysrev;   /* system revision check value */
    u_int32    m_size;     /* module size */
    owner_id   m_owner;    /* group/user ID */
    u_int32    m_name;     /* offset to module name */
    u_int16    m_access,   /* access permissions */
    m_tylan,    /* module type and language */
    m_attrev,   /* module attributes and revision */
    m_edit;     /* module edition number */
    u_int32    m_needs,    /* module hardware requirements flags */
    /* (reserved) */
    m_share,    /* offset of shared data in statics */
    m_symbol,   /* offset to symbol table */
    m_exec,     /* offset to execution entry point */
    m_excpt,    /* offset to exception entry point*/
    m_data,     /* data storage requirement */
    m_stack,    /* stack size */
    m_idata,    /* offset to initialized data */
    m_idref,    /* offset to data reference lists */
    m_init,     /* offset to initialization routine*/
    m_term,     /* offset to termination routine */
    m_dbias,    /* data area pointer bias*/
    m_cbias;    /* code area pointer bias */
    u_int16    m_ident;    /* linkage locale identifier */
    char       m_spare[8]; /* reserved */
    u_int16    m_parity;   /* header parity */
} mh_com, *Mh_com;
```
Fields

\( m\_sync \)
Constant bytes (for example, 0xf00d for the PowerPC) used to locate modules during the startup memory search. The value of \( m\_sync \) is processor dependent.

\( m\_sysrev \)
Identifies the format of a module.

\( m\_size \)
Overall size of the module in bytes, including header and CRC.

\( m\_owner \)
Group/user ID of the module’s owner.

\( m\_name \)
Contains the offset of the module name string relative to the start (first sync byte) of the module. The name string can be located anywhere in the module and consists of a string of ASCII characters terminated by a null (0) byte.

\( m\_access \)
Defines the permissible module access by its owner or by other users. The write permissions on memory modules only make sense for data modules. Module access permission values are located in the header file `module.h` and are defined as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP_OWNER_READ</td>
<td>$0001 = Read permission by owner</td>
</tr>
<tr>
<td>MP_OWNER_WRITE</td>
<td>$0002 = Write permission by owner</td>
</tr>
<tr>
<td>MP_OWNER_EXEC</td>
<td>$0004 = Execute permission by owner</td>
</tr>
<tr>
<td>MP_GROUP_READ</td>
<td>$0010 = Read permission by group</td>
</tr>
<tr>
<td>MP_GROUP_WRITE</td>
<td>$0020 = Write permission by group</td>
</tr>
<tr>
<td>MP_GROUP_EXEC</td>
<td>$0040 = Execute permission by group</td>
</tr>
<tr>
<td>MP_WORLD_READ</td>
<td>$0100 = Read permission by world</td>
</tr>
<tr>
<td>MP_WORLD_WRITE</td>
<td>$0200 = Write permission by world</td>
</tr>
<tr>
<td>MP_WORLD_EXEC</td>
<td>$0400 = Execute permission by world</td>
</tr>
</tbody>
</table>

All bits not defined in the preceding table are reserved.
Contains the module type (first byte) and language (second byte). The language codes indicate if the module is executable and which language the run-time system requires for execution, if any. Module type values and language codes are located in the header file `module.h` and are defined as follows:

<table>
<thead>
<tr>
<th>Module Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT_ANY</td>
<td>0 = Not used (wildcard value in system calls)</td>
</tr>
<tr>
<td>MT_PROGRAM</td>
<td>1 = Program module</td>
</tr>
<tr>
<td>MT_SUBROUT</td>
<td>2 = Subroutine module</td>
</tr>
<tr>
<td>MT_MULTI</td>
<td>3 = Multi-module (reserved for future use)</td>
</tr>
<tr>
<td>MT_DATA</td>
<td>4 = Data module</td>
</tr>
<tr>
<td>MT_CDBDATA</td>
<td>5 = Configuration Data Block data module</td>
</tr>
<tr>
<td></td>
<td>6-10 = Reserved for future use</td>
</tr>
<tr>
<td>MT_TRAPLIB</td>
<td>11 = User trap library</td>
</tr>
<tr>
<td>MT_SYSTEM</td>
<td>12 = System module</td>
</tr>
<tr>
<td>MT_FILEMAN</td>
<td>13 = File manager module</td>
</tr>
<tr>
<td>MT_DEVDRVR</td>
<td>14 = Physical device driver</td>
</tr>
<tr>
<td>MT_DEVDESC</td>
<td>15 = Device descriptor module</td>
</tr>
<tr>
<td></td>
<td>16-up = User definable</td>
</tr>
<tr>
<td>ML_ANY</td>
<td>0 = Unspecified language (wildcard in system calls)</td>
</tr>
<tr>
<td>ML_OBJECT</td>
<td>1 = Machine language</td>
</tr>
<tr>
<td>ML_ICODE</td>
<td>2 = Basic I-code (reserved for future use)</td>
</tr>
<tr>
<td>ML_PCODE</td>
<td>3 = Pascal P-code (reserved for future use)</td>
</tr>
<tr>
<td>ML_CCODE</td>
<td>4 = C I-code (reserved for future use)</td>
</tr>
<tr>
<td>ML_CBLCODE</td>
<td>5 = Cobol I-code (reserved for future use)</td>
</tr>
<tr>
<td>ML_FRTNCODE</td>
<td>6 = Fortran</td>
</tr>
<tr>
<td></td>
<td>7-15 = Reserved for future use</td>
</tr>
<tr>
<td></td>
<td>16-255 = User definable</td>
</tr>
</tbody>
</table>

Not all combinations of module type codes and languages are compatible.
m_attrev
Contains the module attributes (first byte) and revision (second byte). The attribute byte is defined in the header file module.h and as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>The module is re-entrant (sharable by multiple tasks).</td>
</tr>
<tr>
<td>6</td>
<td>The module is sticky. A sticky module is not removed from memory until its link count becomes -1 or memory is required for another use.</td>
</tr>
<tr>
<td>5</td>
<td>The module is a system-state module.</td>
</tr>
</tbody>
</table>

If two modules with the same name and type are found in the memory search or are loaded into the current module directory, only the module with the highest revision level is kept. This enables easy substitution of modules for update or correction, especially ROMed modules.

m_edit
Indicates the software release level for maintenance. OS-9 does not use this field. Whenever a program is revised (even for a small change), increase this number. Internal documentation within the source program can be keyed to this system.

m_needs
Module hardware requirements flags (reserved for future use).

m_share
Offset to any shared data the module contains within its global data area. For example, this field is used by IOMAN to locate the main statics storage structure of file managers and device drivers.

m_symbol
Reserved.

m_exec
Offset to the program starting address, relative to the module starting address.

m_excpt
Relative address of a routine to execute if an uninitialized user trap is called.

m_data
Required size of the program data area (storage for program variables).

m_stack
Minimum required size of the program’s stack area.

m_idata
Offset to an eight-byte value which precedes the initialized data area. The first four bytes contain an offset from the beginning of the program’s memory to the beginning of the initialized data area, which contains values to copy to the program data area. The linker places all constant values declared in vsects here. The second four bytes contain the number of initialized data bytes to follow.
m_idref
Offset to a table of values to locate pointers in the data area. Initialized variables in the program’s data area may contain pointers to absolute addresses. Code pointers are adjusted by adding the absolute starting address of the object code area. Data pointers are adjusted by adding the absolute starting address of the data area.

**F_FORK** automatically calculates the effective address at execution time using the tables created in the module. The first word of each table is the most significant (MS) word of the offset to the pointer. The second word is a count of the number of least significant (LS) word offsets to adjust. The adjustment is made by combining the MS word with each LS word entry. This offset locates the pointer in the data area. The pointer is adjusted by adding the absolute starting address of the object code or the data area (for code pointers or data pointers respectively). It is possible, after exhausting this first count, another MS word and LS word are given. This continues until an MS word of zero and an LS word of zero are found.

m_init
Offset to the trap handler initialization routine.

m_term
Reserved.

m_dbias
This field contains the bias value applied by the linker to the global data accesses in the module. Biasing global data accesses allows the compiler to generate efficient data accesses to a larger data space.

m_cbias
This field contains the bias value applied by the linker to the code symbols within the module. Biasing code references allows the compiler to generate efficient code references to a larger area of code.

m_ident
Linkage site identifier. This field is not currently implemented.

m_spare
Reserved.

m_parity
A complement of the exclusive-OR of the previous header words. OS-9 uses this field to check module integrity.
The Kernel

This chapter outlines the primary functions of the kernel. It includes the following topics:

- Kernel Functions
- System Call Overview
- Kernel System Call Processing
- Memory Management
- OS-9 Memory Map
- Memory Fragmentation
- Colored Memory
- System Initialization
- Extension Modules
- Process Creation
- Process Scheduling
Kernel Functions

The nucleus of OS-9 is the kernel, which manages resources and controls processing. The kernel is a ROMable, compact, OS-9 module written in C language. The primary responsibility of the kernel is to process and coordinate system calls or service requests.

OS-9 has two general types of system calls. These include I/O calls, such as reads and writes, and system function calls.

System functions include those listed below:

- Memory management
- System initialization
- Process creation and scheduling
- Exception/interrupt processing

When a system call is made, the processor is changed to privileged state. The way this is done depends on which processor is being used. The kernel determines what type of system call you want to perform. The kernel directly executes the calls that perform system functions, but does not execute the I/O calls. Instead, the I/O calls are passed to IOMAN.

System Call Overview

For information about specific system calls, refer to Chapter 6, OS-9 System Calls.

User-State and System-State

There are two distinct OS-9 environments in which you can execute object code:

user-state User-state is the normal program environment in which processes are executed. Generally, user-state processes do not deal directly with the specific hardware configuration of the system.

system-state System-state is the environment in which OS-9 system calls and interrupt service routines are executed.

Functions executing in system-state have several advantages over those running in user-state:

- A system-state routine has access to all processor capabilities. For example, on memory protected systems, a system-state routine can access memory in the system: it can mask interrupts, alter internal data structures, or take direct control of hardware interrupt vectors.

- System-state routines are never time sliced. Once a process has entered system state, no other process executes until the system-state process finishes or goes to sleep (F_SLEEP waiting for I/O). The only processing that may preempt a system-state routine is interrupt servicing.
Some OS-9 system calls are only accessible from system-state.

The characteristics of system state make it the only way to provide certain types of programming functions. For example, it is almost impossible to provide direct I/O to a physical device from user state. However, do not run all programs in system state.

Signal handler routines cannot be called for system state processes. The process must dequeue them manually.

In a multi-user environment, it is important to ensure each user receives a fair share of the CPU time. This is the basic function of time slicing.

Memory protection prevents user-state routines from accidentally damaging data structures they do not own.

A user-state process may be aborted. If a system-state routine loses control, the entire system usually crashes.

It is far more difficult and dangerous to debug system-state routines than user-state routines. You can use the user-state debugger to find most user-state problems. Generally, system-state problems are much more difficult to locate.

User programs almost never have to be concerned with physical hardware; they are essentially isolated from it. This makes user-state programs easier to write and port.

If a system call returns with an exception error code, a system-state exception has occurred. If you are getting system state exceptions, there is a bug either in the OS-9 system code, a driver, or some user developed system code. Be advised that such exceptions can leave the system and the user program in an unknown and unstable state.

Installing System-State Routines

With direct access to all system hardware, any system-state routine has the ability to take over the entire machine. It is often a challenge to keep system-state routines from crashing or hanging up the system. Increase system stability, the methods of creating routines that operate in system state are limited.

In OS-9, there are four ways to provide system-state routines:

1. Install an OS9P2 module in the system bootstrap file or in ROM.
   During cold start, the OS-9 kernel links to this module, and if found, calls its execution entry point. Typically, the OS9P2 module is used to install new system service requests.

2. Use the I/O system as an entry into system state.
   File managers and device drivers are always executed in system state. In fact, the most obvious reason to write system-state routines is to provide support for new hardware devices. It is possible to write a dummy device driver and use the I_GETSTAT or I_SETSTAT routines to provide a gateway to the driver.
3. Write a trap handler module.
   For routines of limited use that are to be dynamically loaded and unlinked, this is perhaps the most convenient method. It is often practical to debug trap handler routines as user-state subroutines and then convert the finished routines to a trap handler module. OS-9 trap handlers always execute in system state.

4. Set the supervisor state bit in the attribute/revision word for the module.
   A program executes in system state if the supervisor state bit in the module attribute/revision word is set and if the module is owned by the super user.

 Kernel System Call Processing

The kernel processes all OS-9 system calls (service requests). System call parameters are passed and returned in parameter blocks.

There are two general types of system calls:

- **Non-I/O calls (calls performing system functions)**
- **I/O calls**

System calls are identified by a function code passed in the service request parameter block. Every standard OS-9 system call has an associated symbolic name for the function code provided in the `funcs.h` C header file. The non-I/O call symbols begin with `F_` and the I/O calls begin with `I_`. For example, the system call to link a module is called `F_LINK`.

 Non-I/O Calls

There are two types of non-I/O system calls:

**User-State System Calls**
These calls perform memory management, multitasking, and other functions for user programs. These are mainly processed by the kernel.

**System-State System Calls**
These calls can only be used by system software in system-state and usually operate on internal OS-9 data structures. To preserve the modularity of OS-9, these requests are system calls rather than subroutines. User-state programs cannot access these calls, but system modules such as device drivers can use these calls.

In general, system-state routines may use any of the ordinary (user-state) system calls. However, avoid making system calls at inappropriate times. For example, an interrupt service routine should avoid I/O calls, memory requests, timed sleep requests, and other calls that can be particularly time consuming (such as `F_CRC`).

Memory requested in system-state is not recorded in the process descriptor memory list. The requesting process must ensure the memory is returned to the system before the process terminates.
I/O Calls

When the kernel receives an I/O request, it immediately passes the request to IOMAN. IOMAN passes the request to the appropriate file manager and device driver for processing.

Any I/O system call may be used in a system-state routine, with one slight difference than when executed in user state: all path numbers used in system state are system path numbers. Each user-state process has a path table used to convert its local path numbers to system path numbers. The system itself has a global path number table used to convert system path numbers into actual addresses of path descriptors. System-state I/O system calls must be made using system path numbers.

For example, a system-state OS-9 `I_WRITE` system call prints an error message on the caller’s standard error path. To do this, a system-state process may not perform output on path number two. Instead, it must use the `I_TRANPN` system call to translate the user path number to its associated system path number.

When a user-state process exits with open I/O paths, the `F_EXIT` routine automatically closes the paths. This is possible because OS-9 keeps track of the open paths in the process path table. In system state, the `I_OPEN` and `I_CREATE` system calls return a system path number that is not recorded in the process path table or anywhere else by OS-9; the system-state routine that opens an I/O path must ensure the path is eventually closed. This is true even if the underlying process is abnormally terminated.

Memory Management

If any object (such as a program and constant table) is to be loaded in memory, it must use the standard OS-9 memory module format described in `Chapter 1, System Overview`. This enables OS-9 to maintain a module directory to keep track of modules in memory. The module directory contains the name, address, and other related information about each module in memory.

After OS-9 has been booted, a single module directory exists containing all of the boot modules. You may create additional module directories and subdirectories at your discretion. Each module directory has independent access permissions. By using multiple module directories, modules with the same name can be loaded in memory and executed without conflict.

This can be extremely useful in the continuing development of existing software. When a module is loaded in memory, it is added to the process current module directory.

When a process creates a new process, the OS-9 kernel searches the current module directory for the target module. If this search fails, the kernel searches the process’ alternate module directory, initially specified in your login file. If that search fails, the kernel attempts to load the module into the current module directory.
Each module directory entry contains a link count. The link count is the number of processes using the module. When a process links to a module in memory, the link count of the module is incremented by one. When a process unlinks from a module, the link count is decremented by one. When a module’s link count becomes zero, its memory is deallocated and the module is removed from the module directory, unless the module is sticky.

A sticky module is not removed from memory until its link count becomes -1 or memory is required for another use. A module is sticky if the sixth bit of the module header’s attribute byte (first byte of the m_attrev field) is set. If several modules are merged together and loaded, you must unlink all of those modules before any are removed from the module directory.

Refer to Chapter 5 of Using OS-9 for more information on module directories.

**OS-9 Memory Map**

OS-9 uses a software memory management system in which all memory is contained within a single memory map. Therefore, all user tasks share a common address space.

A map of an example OS-9 memory space is shown in Figure 2-1. The sections shown are not required to be at specific addresses. Microware recommends you keep each section in contiguous reserved blocks arranged in an order that facilitates future expansion. It is always advantageous for RAM to be physically contiguous whenever possible.

**Figure 2-1. Example OS-9 Memory Map**

<table>
<thead>
<tr>
<th>Unused, Available for future RAM or ROM expansion.</th>
<th>Highest Memory Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM 256K minimum 1M recommended</td>
<td>Lowest Memory Address</td>
</tr>
<tr>
<td>Exception Vector area</td>
<td></td>
</tr>
</tbody>
</table>

**System Memory Allocation**

During the OS-9 start-up sequence, an automatic search function in the kernel and the boot ROM locates blocks of RAM and ROM. OS-9 reserves some RAM for its own data structures. ROM blocks are searched for valid OS-9 ROM modules.

The amount of memory OS-9 requires is variable. Actual requirements depend on the system configuration and the number of active tasks and open files. The following sections describe various parts of the OS-9 system memory.
Operating System Object Code

On disk-based systems, operating system component modules (such as the kernel, I/O managers, and device drivers) are normally bootstrap-loaded into RAM. OS-9 does not dynamically load overlays or swap system code. Therefore, no additional RAM is required for system code. Alternately, you can place OS-9 in ROM for non-disk systems.

System Global Memory

The OS-9 kernel allocates a section of RAM memory for internal use. It contains the following items:

- an exception jump table
- the debugger/boot variables
- a system global area

Variables in the system global area are symbolically defined in the `sysglob.h` library and the variable names begin with `d_`

User programs should never directly access system global variables. System calls are provided to allow user programs to read the information in this area.

System Dynamic Memory

OS-9 maintains dynamic-sized data structures (such as I/O buffers, path descriptors, and process descriptors) that are allocated from the general RAM area when needed. The system modules allocate and maintain these structures. For example, IOMAN allocates memory for path descriptors and maintains them. The system global memory area contains the pointers to these system data structures.

User Memory

All unused RAM memory is assigned to a free memory pool. Memory space is removed and returned to the pool as it is allocated or deallocated for various purposes. OS-9 automatically assigns memory from the free memory pool whenever any of the following occur:

- Modules are loaded in RAM.
- New processes are created.
- Processes request additional RAM.
- OS-9 requires more I/O buffers.
- OS-9 internal data structures must be expanded.

Storage for user program object code modules and data space is dynamically allocated from and deallocated to the free memory pool. User object code modules are also automatically shared if two or more tasks execute the same object program. User object code application programs can also be stored in ROM memory. The total memory required for user memory depends largely upon the application software that is to be run.
Memory Fragmentation

Once a program is loaded, it remains at the address where it was originally loaded. Although position-independent programs can be initially placed at any address where free memory is available, program modules cannot be dynamically relocated afterwards. This can lead to memory fragmentation.

When programs are loaded, they are assigned the first sufficiently large block of memory at the highest address possible in the address space. However, if a colored memory request is made, this may not be true. If a number of program modules are loaded, and subsequently one or more non-contiguous modules are unlinked, several fragments of free memory space will exist. The total free memory space can be large, but because it is scattered, not enough space exists in a single block to load a particular program module.

To avoid memory fragmentation, load modules at system startup. This places the modules in contiguous memory space. You can also initialize each standard device when the system is booted. This enables the devices to allocate memory from higher RAM.

If serious memory fragmentation does occur, the system administrator can kill processes and unlink modules in ascending order of importance until there is sufficient contiguous memory. The `mfree` utility can determine the number and size of free memory blocks.

Colored Memory

OS-9 colored memory allows a system to recognize different memory types and reserve areas for special purposes. For example, part of a RAM can store video images and another part can be battery-backed. The kernel allows areas of RAM like these to be isolated and accessed specifically. You can request specific memory types or colors when you allocate memory buffers, create modules in memory, or load modules into memory. If a specific type of memory is not available, the kernel returns error #237, `EOS_NORAM`.

Colored memory lists are not essential on systems whose RAM consists of one homogeneous type, although they can improve system performance and allow flexibility in configuring memory search areas.

Colored Memory Definition List

The kernel must have a description of the CPU address space in order to use the colored memory routines. This is accomplished by including a colored memory definition list in `default.des`. The list describes the characteristics of each memory region. The kernel searches each for RAM during system startup.
The following information describes a memory area to the kernel:

- Memory color (type)
- Memory priority
- Memory access permissions
- Local bus address
- Block size to be used by the kernel cold start routine to search the area for RAM or ROM
- External bus translation address (for DMA and dual-ported RAM)
- Optional name

The memory list (memlist) may contain as many regions as needed. If no list is specified, the kernel automatically creates one region describing the memory found by the bootstrap ROM.

Each line in the memory list must contain all the parameters in the following order: type, priority, attributes, blksiz, addr begin, addr end, name, and DMA-offset.

The colored memory list must end on an even address. Descriptions of the memlist fields are included below:

**Table 2-1. memlist Fields**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Size</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Type</td>
<td>word</td>
<td>Type of memory. Two memory types are currently defined in memory.h:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM_SYS 0x01 System RAM memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MEM_SHARED 0x8000 Shared memory (0x8000 - 0xffff)</td>
</tr>
<tr>
<td>Priority</td>
<td>word</td>
<td>High priority RAM is allocated first (255 - 0). If the block priority is 0,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the block can only be allocated by a request for the specific color (type)</td>
</tr>
<tr>
<td>Access Permissions</td>
<td>word</td>
<td>Memory type access bit definitions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Bit 0</strong>: B_USERRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indicates memory allocatable by user processes. (This bit is ignored if B_ROM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bit is also set.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Bit 1</strong>: B_PARITY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indicates parity memory; initialized by kernel during start-up.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Bit 2</strong>: B_ROM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indicates ROM; the kernel searches this for modules during start-up.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Bit 3</strong>: B_NVRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-volatile RAM; the kernel searches this for modules during start-up.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Bit 4</strong>: B_SHARED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shared memory; reserved for future use.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NOTE</strong>: Only B_USERRAM memory can be initialized.</td>
</tr>
</tbody>
</table>
Table 2-1. memlist Fields (Continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Size</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search Block Size</td>
<td>word</td>
<td>The kernel checks every search block size to see if RAM/ROM exists.</td>
</tr>
<tr>
<td>Low Memory Limit</td>
<td>long</td>
<td>Beginning address of the block as referenced by the CPU.</td>
</tr>
<tr>
<td>High Memory Limit</td>
<td>long</td>
<td>End address of the block as referenced by the CPU.</td>
</tr>
<tr>
<td>Description String</td>
<td>long</td>
<td>This 32-bit offset of a user-defined string describes the type of memory block.</td>
</tr>
<tr>
<td>Address Translation</td>
<td>long</td>
<td>External bus address of the beginning of the block. If zero, this field does not apply. Refer to _os_trans() for more information.</td>
</tr>
</tbody>
</table>

Refer to your OS-9 Device Descriptor and Configuration Module Reference for more information on creating a memory list in the init modules.

The complete memory list structure definitions are located in the alloc.h file and are listed below:

```c
/* initialization table (in memdefs module data area) */
typedef struct mem_table {
    u_int16
        type,      /* memory type code */
        prior,     /* memory allocation priority */
        access,    /* access permissions */
        blksiz;    /* search block size */
    u_char
        *lolim,    /* beginning absolute address for this type */
        *hilim;    /* ending absolute address +1 for this type */
    u_int32
        descr;     /* optional description string offset */
    u_int32
        dma_addr, /* address translation address for dma's, etc.*/
        rsvd2[2]; /* reserved, must be zero */
} *Mem_tbl, mem_tbl;

/* access bit definitions */
#define B_USERRAM  (0x01)   /* memory allocatable by user procs */
#define B_PARITY   (0x02)   /* parity memory; must be initialized */
#define B_ROM      (0x04)   /* read-only memory; searched for modules */
#define B_NVRAM    (0x08)   /* non-volatile RAM; searched for modules */
#define B_SHARED   (0x10)   /* shared memory (Reserved for future use.)*/
```
SSM and Cache

The SSM module provides user-state security. The cache module is used to enforce caching policy on ranges of memory. The following sections describe these modules and how they are used with OS-9 and its supported processors.

Cache List

OS-9 supports the ability to precisely define the caching modes used for regions of memory in the system. Precise definition of these modes for particular regions allows you to configure the system for optimal performance and/or system functionality. In many cases, regions of memory must be declared non-cachable so cache coherency problems do not result when processes directly reference I/O devices and memory shared with other processors.

When the SSM module is installed in the system, it provides a default cache mode of “writeback” for user-state accesses. This default mode can be over-ridden for specific regions by creating cache list entries in the `Init` module. The cache list is used to describe the caching policy enforced by the cache module.

The cache list entries must end with a longword of 0xffffffff (-1). The following table describes the CacheList parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Size</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Start</td>
<td>long</td>
<td>Start address of memory region.</td>
</tr>
<tr>
<td>Block End</td>
<td>long</td>
<td>End address (+1) of memory region.</td>
</tr>
<tr>
<td>Cache Mode</td>
<td>word</td>
<td>Cache mode (MMU specific) for region.</td>
</tr>
</tbody>
</table>

The cache mode is usually controlled by SSM, rather than the cache module. Cache mode information is located in the following location:

MWOS/OS9000/SRC/DEFS/cache.h

PowerPC Processors: SSM and Cache

This section explains how SSM and cache modules work in order to provide system security and caching policy. This information pertains to the following PowerPC processors: 603, 604, 750, 8240, and 8260.

Cacheing policies consist of the following types of cache behaviors:

- Writeback or “copyback” cache is the default cache mode of a memory region. It is the fastest type of cache, and the most problematic. With copyback cache, values written to memory are first written to cache—not to main memory. During normal operation of the processor, data values held in cache are written to main memory. Under normal circumstances this is not a problem; the cache module provides services in which a programmer can force the write of data values in cache to main memory. This ensures that the data values in the cache are coherent with the data values in main memory. Typically, this is important in multi-processor environments, especially DMA devices. Processors and DMA devices can only read main memory, and not the caches of another processor.

MWOS/OS9000/SRC/DEFS/cache.h
• Write through cache is the fastest type of cache, after Copyback. Writes to this cache are written to the cache and to the memory immediately. Reads are still cached.

• The last type of cache is no cache at all. In other words, read and writes bypass the cache and go directly into memory.

Using the cachelist in the init module, all cacheing policies can be specified on memory ranges. The 600 series PowerPC processor has two mechanisms in order to enforce cache policy: the first is the memory segment model; the second is the block address translation (BAT) mechanism.

The memory segment model is used for user-state memory protection and security, as well as cache policy. The SSM module controls this mechanism. The minimum memory range to control is 4k bytes.

The cache module controls the BAT mechanism. In addition, if the SSM module is not included within the system, the BAT mechanism will be used for system-state and user-state cache policies.

The BAT mechanism is simpler than the memory segment module; however, its options are more limited. This limitation is reflected directly onto the cachelist in the init module. Minimum memory range for a BAT is 128k bytes; regions are in the multiple of power of two from through 256M bytes. Actual BAT memory sizes are as follows: 128k, 256k, 512k 1M, 2M, 4M, 8M, 16M, 32M, 64M, 128M, and 256M. Each of these memory ranges must aligned according to their size, therefore, a 128k memory range must begin on a 128k memory address alignment, and a 32M memory range must begin on a 32M memory address alignment. There are four BAT registers on the processor; each defines only one memory range.

The cachelist for the PowerPC family of processors contains the following structure:

```c
init cachelist[0] {         /*  Initialize cache list element */
  blk_beg = 0x00000000;   /*  beginning address of memory region */
  blk_end = 0x02000000;   /*  ending address +1 */
  mode = 0;              /*  cache mode of region */
};
```

The cachelist consists of two parts: the system-state list and the user-state list. The system-state list is the cacheing policy of memory accessed from system-state, and the user-state list is the cacheing policy of memory accessed from user-state. The user-state list follows the system-state list in the array. (Each list is terminated with an entry that has the beginning address of 0xffffffff.)

Where the mode is concerned, a 0 indicates copyback cache. Macros have been defined to enforce other modes. IO_CACHE_MODE is used no cache; CM_DWTHRU is used to write through mode.
Guidelines for Defining Cachelist

In order to define a cachelist, the following guidelines should be followed:

- Minimum segment size should be 128k bytes.
- Minimum segment start address alignment should be 128k bytes.
- A segment start alignment address size must be less than or equal to the segment size itself. The 256M segment size is the exception. Multiple 256M segments can be defined as one region.
- The segment for address 0 must have sufficient memory to bring up the kernel. In other words, it must have enough memory for the kernel's globals, IRQ stack, and bootfile—if it is loaded into RAM.
- The memory definitions for different segments must not overlap.
- The SSM module must be placed before the cache module in the extension list. In other words, the SSM module must run before the cache module can run if both are in the system. The SSM or the cache module can run stand-alone as well.
- The user-state cachelist normally inherits system-state list entries. Only specific user-state cacheing differences need to be listed in the user-state cache list. For example, video memory made accessible to user-state needs to be marked as IO_CACHE_MODE, or with a data cache write-through, with the user-state driver forcing stores or flushes.

Cachelist Example for PowerPC system:

```c
/* System-state Dram */
init cachelist[0] {         /* Initialize cache list element */
    blk_beg = 0x00000000;   /* beginning address of memory region */
    blk_end = 0x10000000;   /* ending address +1 */
    mode = 0;               /* cache mode of region */
};

/* The rest of memory is considered non-cached. */
init cachelist[1] {         /* Initialize cache list element */
    blk_beg = 0x10000000;   /* beginning address of memory region */
    blk_end = 0x00000000;   /* ending address +1 */
    mode = IO_CACHE_MODE;   /* cache mode of region */
};

/* End of system-state cache list */
init cachelist[2] {         /* Initialize cache list element */
    blk_beg = 0xffffffff;   /* beginning address of memory region */
    blk_end = 0x00000000;   /* ending address +1 */
    mode = 0;               /* cache mode of region */
};
```
/* User-state list DRAM */

init cachelist[3] {         /*  Initialize cache list element           */
    blk_beg = 0x00000000;   /*  beginning address of memory region      */
    blk_end = 0x02000000;   /*  ending address +1                       */
    mode = 0;               /*  cache mode of region                    */
};

/* End of user list */

init cachelist[4] {         /*  Initialize cache list element           */
    blk_beg = 0xffffffff;   /*  beginning address of memory region      */
    blk_end = 0x00000000;   /*  ending address +1                       */
    mode = 0;               /*  cache mode of region                    */
};

With the above cachelist, there is 256M bytes of copyback cached dram; the rest of the system is considered non-cached.

**Setup for Allocatable NON-CACHED Memory**

By matching the memlist entries in the init module to the cachelist entries, a region of non-cached memory can be defined. In this way, anything in the system can use the non-cached memory as a resource. The following entries define a region of 512k bytes of non-cached memory out of 32M of total memory. You should define such regions at the end of physical dram.

/* Memlist entries */

init memlist[0] {          /*  Initialize memory list element   */
    type = MEM_SYS;         /*  memory type code                */
    prior = 0xFF;           /*  memory allocation priority      */
    access = B_USERRAM;     /*  access permissions             */
    blksiz = 0x800;         /*  search block size              */
    lolim = 0x00080000;     /*  beginning absolute address     */
    hilim = 0x01F80000;     /*  ending absolute address + 1    */
};

init memlist[1] {           /*  Initialize memory list element */
    type = MEM_NOCACHE;     /*  memory type code               */
    prior = 0x0;            /*  memory allocation priority     */
    access = B_USERRAM;     /*  access permissions             */
    blksiz = 0x800;         /*  search block size              */
    lolim = 0x01F80000;     /*  beginning absolute address     */
    hilim = 0x02000000;     /*  ending absolute address + 1    */
};
/* End of memlist */

init memlist[2] { /* Initialize memory list element */
    type = 0; /* memory type code */
    prior = 0x0; /* memory allocation priority */
    access = 0; /* access permissions */
    blksiz = 0x0; /* search block size */
    lolim = 0x0; /* beginning absolute address */
    hilim = 0x0; /* ending absolute address + 1 */
};
/* Cachelist entries */
/* DRAM */

init cachelist[0] { /* Initialize cache list element */
    blk_beg = 0x00000000; /* beginning address of memory region */
    blk_end = 0x01F80000; /* ending address +1 */
    mode = 0; /* cache mode of region */
};

init cachelist[1] { /* Initialize cache list element */
    blk_beg = 0x01F80000; /* beginning address of memory region */
    blk_end = 0x02000000; /* ending address +1 */
    mode = IO_CACHE_MODE; /* cache mode of region */
};
/* The rest of memory is considered non-cached. */

init cachelist[2] { /* Initialize cache list element */
    blk_beg = 0x10000000; /* beginning address of memory region */
    blk_end = 0x00000000; /* ending address +1 */
    mode = IO_CACHE_MODE; /* cache mode of region */
};
/* End of system-state cache list */

init cachelist[3] { /* Initialize cache list element */
    blk_beg = 0xffffffff; /* beginning address of memory region */
    blk_end = 0x00000000; /* ending address +1 */
    mode = 0; /* cache mode of region */
};

/* End of system-state cache list */
Chapter 2: The Kernel

/* DRAM */

init cachelist[4] {  /* Initialize cache list element */
    blk_beg = 0x00000000;  /* beginning address of memory region */
    blk_end = 0x01f80000;  /* ending address +1 */
    mode = 0;              /* cache mode of region */
};

init cachelist[5] {  /* Initialize cache list element */
    blk_beg = 0x01f80000;  /* beginning address of memory region */
    blk_end = 0x02000000;  /* ending address +1 */
    mode = IO_CACHE_MODE;  /* cache mode of region */
};

init cachelist[6] {  /* Initialize cache list element */
    blk_beg = 0xffffffff;  /* beginning address of memory region */
    blk_end = 0;           /* ending address +1 */
    mode = 0;              /* cache mode of region */
};

The MEM_NOCACHE type of memory is defined as memory that is not cached. Once defined in this manner, system drivers and other code can use the non-cached memory.

The memlist entry for MEM_NOCACHE has a priority of 0.

The cache module is responsible for splitting up the cache list into a BAT array list. This list is sorted, and the cache module has an exception handler that will replace BAT registers in the processor to allow access, and cache policy into the particular region of memory.

BAT register 0 contains the entry for address 0 and is never replaced. BAT registers 1, 2, and 3 are all capable of being replaced via “round robin”. This BAT array list contains all of the individual entries defined by the cachelist. Each cachelist entry may get split up into several BAT array elements as needed. For example, in the cachelist[2] entry above, the entire 3.75G byte area splits into 15 BAT array elements, each one 256M bytes in size. Because there is no 31.5M byte BAT entry size in cachelist[0], the cachelist[0] entry splits into six BAT array elements. The breakdown is as follows:

1 16M segment
1 8M segment
1 4M segment
1 2M segment
1 1M segment
1 512K segment

The cachelist[1] entry takes up the other 512K that makes up the 32M of physical memory contained by the computer.
Performance Considerations

The above BAT array scheme can have a negative impact on performance. Although the first segment is locked into BAT register 0, the other five segments in cachelist[0], the segment in cachelist[1], and other segments in cachelist[2] for I/O regions contend for the other three BAT registers. However, there are some procedures for handling this.

- If you do not need non-cached memory, do not use it. Defining a small non-cached memory region results more in the necessary replacement of some segments than a larger non-cached memory region.

- Define differing priorities of memory. If segment 1 is defined alone as a high priority memory in the memlist, and the other segments grouped into a lower priority memory in the memlist, all of the beginning kernel setup and processes will run from the first segment. The other memory priorities will only be allocated when this segment has run out of memory.

Colored Memory in Homogenous Memory Systems

As previously mentioned, colored memory definitions are not essential for systems whose memory is homogenous. However, these types of systems can benefit from this feature of the kernel in terms of system performance and ease of memory list reconfiguration.

System Performance

In a homogeneous memory system, the kernel allocates memory from the top of available RAM when requests are made by \texttt{F\_SRQMEM} (loading modules). If the system has RAM on-board the CPU and off-board in external memory boards with higher addresses, the modules tend to be loaded in the off-board RAM. On-board RAM is not used for a \texttt{F\_SRQMEM} call until the off-board memory cannot accommodate the request.

Due to bus access arbitration, programs running in off-board memory execute more slowly than if they were executing in on-board memory. Also, external bus activity is increased. This may impact the performance of other bus masters in the system.

The colored memory lists can reverse this tendency in the kernel, so a CPU can not use off-board memory until all of its on-board memory is used. This results in faster program execution and less saturation of the system’s external bus. To do this, make the priority of the on-board memory higher than the off-board memory.

Reconfiguring Memory Areas

In a homogeneous memory system, the memory search areas are defined in the ROM memory list. Changes to these areas previously required new ROMs be made from source code (usually impossible for end users) or from a patched version of the original ROMs (usually difficult for end users).
The colored memory lists somewhat alleviate this situation by configuring the search areas as follows:

- the ROM memory list describes only the on-board memory.
- the colored memory lists in `default.des` define any external bus memory search areas in the Init module only.

Using colored memory in this situation enables the end user to easily reconfigure the external bus search areas by adjusting the lists in `default.des` and making a new Init module. The ROM does not require patching.

**System Initialization**

After a hardware reset, the kernel (located in ROM or loaded from disk, depending on your system configuration) is executed by the bootstrap ROM. The kernel initializes the system; this includes locating ROM modules and running the system start-up task.

**Init: The Configuration Module**

The `init` module:

- Is non-executable module of type `MT_SYSTEM`
- Contains a table of system start-up parameters
- Specifies the initial table sizes and system device names during startup
- Is always available to determine system limits
- Is required to be in memory when the system is booting and usually resides in the `sysboot` file or in ROM
- Begins with a standard module header

The `m_exec` offset in the module header is a pointer to the system constant table. The fields of this table are defined in the `init.h` header file.

Refer to the OS-9 Device Descriptor and Configuration Module Reference for a listing of the `init` module fields.

**Extension Modules**

To enhance OS-9 capabilities, you can execute additional modules at boot time. These extension modules provide a convenient way to install a new system call code or collection of system call codes, such as a system security module. The kernel calls the modules at boot time if their names are specified in the Extension list of the `init` module and the kernel can locate them.
To include an extension module in the system, you can either program the module into system memory or use the p2init utility to add it to a running system.

Refer to the Utilities Reference for information about p2init. Refer to the OS-9 Device Descriptor and Configuration Module Reference for procedures to change the init modules and your board guide for instructions on how to build a new boot file containing the desired extension modules.

When an extension module is called for initialization during coldstart, the module’s entry point is executed with its global static storage (if any) pre-initialized and set. The extension module is passed a pointer to the kernel’s global static storage as defined in the header file sysglob.h.

Process Creation

All OS-9 programs are run as processes or tasks. New processes are created by the F_FORK system call. The most important parameter passed in the fork system call is the name of the primary module that the new process is to execute initially.

The following list outlines the creation process:

1. Locate or Load the Program.
   OS-9 searches for the module in memory by means of the module directory. If OS-9 cannot locate the module, it loads a mass-storage file into memory using the requested module name as a file name.

2. Allocate and Initialize a Process Descriptor and an I/O Descriptor.
   After the primary module has been located, a data structure called a process descriptor is assigned to the new process. The process descriptor is a table containing information about the process such as its state, memory allocation, and priority. The I/O descriptor contains information about the process I/O such as the I/O paths and counts of bytes read and written. The process descriptor and I/O descriptor are automatically initialized and maintained. Processes do not need to be aware of the existence or contents of process descriptors or I/O descriptors.

3. Allocate the Stack and Data Areas.
   The primary module’s header contains a data and stack size. OS-9 allocates a contiguous memory area of the required size from the free memory space. Process memory areas are discussed in the following section.

4. Initialize the Process.
   The new process’ registers are set to the proper addresses in the data area and object code module. If the program uses initialized variables and/or pointers, they are copied from the object code area to the proper addresses in the data area.

If any of these steps cannot be performed, creation of the new process is aborted and the process that originated the fork is notified of the error. If all are completed, the new process is added to the active process queue for execution scheduling.
The new process is assigned a unique number, called a process ID, that is used as its identifier. Other processes can communicate with it by referring to its ID in various system calls. The process also has an associated group ID and user ID which identify all processes and files belonging to a particular user and group of users. The IDs are inherited from the parent process.

Processes terminate when they execute an **F_EXIT** system service request or when they receive fatal signals or errors. Terminating the process performs the following functions:

- Closes any open paths
- Deallocates the process’ memory
- Unlinks its primary module
- Unlinks any subroutine libraries or trap handlers the process may have used

**Process Memory Areas**

All processes are divided into two logically separate memory areas:

- code
- data

This division provides the modular software capabilities for OS-9.

Each process has a unique data area, but not necessarily a unique program memory module. This allows two or more processes to share the same copy of a program. This automatic OS-9 functionality results in more efficient use of available memory.

A program must be in the form of an executable memory module to be run. The program is position independent and ROMable, and the memory it occupies is considered to be read-only. It may link to and execute code in other modules.

The process data area is a separate memory space where all of the program variables are kept. The top part of this area is used for the program’s stack. The actual memory addresses assigned to the data area are unknown at the time the program is written. A base address is kept in a register to access the data area. You can read and write to this area.

If a program uses variables requiring initialization, the initial values are copied by OS-9 from the read-only program area to the data area where the variables actually reside. The OS-9 linker builds appropriate initialization tables that OS-9 uses to initialize the variables.
Process States

A process can be in one of five states:

**Table 2-3. Process States**

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>The process is active and ready for execution. Active processes are given time for execution according to their relative priority with respect to all other active processes. The scheduler uses a method that compares the ages of all active processes in the queue. All active processes receive some CPU time, even if they have a very low relative priority.</td>
</tr>
<tr>
<td>Event</td>
<td>The process is inactive until the associated event occurs. The event state is entered when a process executes an \texttt{F_EVENT} service request when the specified event condition is not satisfied. The process remains inactive until another process or interrupt service routine performs an \texttt{F_EVENT} system call that satisfies the waiting process’s condition.</td>
</tr>
<tr>
<td>Sleeping</td>
<td>The process is inactive for a specific period of time or until a signal is received. The sleep state is entered when a process executes an \texttt{F_SLEEP} service request. \texttt{F_SLEEP} specifies a time interval for which the process is to remain inactive. Processes often sleep to avoid wasting CPU time while waiting for some external event, such as completing I/O. Zero ticks specifies an infinite period of time. A process waiting on an event waits in a queue associated with the specific event, but behaves as though it was in the sleep queue.</td>
</tr>
<tr>
<td>Suspended</td>
<td>The process is inactive, unknown to the system, and not a member of any queue. The suspended state is entered when a process or system module does an \texttt{F_SSPD} call on a given process. The process can be reactivated with an \texttt{F_APROC} call.</td>
</tr>
<tr>
<td>Waiting</td>
<td>The process is inactive until a child process terminates or until a signal is received. When a process executes an \texttt{F_WAIT} system service request, it enters the wait state. The process remains inactive until one of its descendant processes terminates or until it receives a signal.</td>
</tr>
</tbody>
</table>

A separate queue (linked list of process descriptors) exists for each process state, except the suspended state. State changes are accomplished by moving a process descriptor from its current queue to another queue.
Process Scheduling

OS-9 is a multitasking operating system. This means two or more independent programs, called processes, or tasks, can execute simultaneously. Each second of CPU time is shared by several processes. Although the processes appear to run continuously, the CPU only executes one instruction at a time. The OS-9 kernel determines which process to run and for how long, based on the priorities of the active processes.

The action of switching from the execution of one process to another is called task switching. Task switching does not effect program execution.

The CPU is interrupted by a real-time clock every tick. By default, a tick is .01 second (10 milliseconds). At any occurrence of a tick, OS-9 can stop executing one program and begin executing another. The tick length is hardware dependent. Thus, to change the tick length, you must rewrite the clock driver and re-initialize the hardware.

The longest amount of time a process controls the CPU before the kernel re-evaluates the active process queue is called a slice or time slice per slice at run-time, adjust the system global variable d_tslice.

You can also change the number of ticks per slice prior to booting the system by modifying m_slice in the init modules.

Refer to the OS-9 Device Descriptor and Configuration Module Reference for information to modify this field.

To ensure efficiency, only processes on the active process queue are considered for execution. The active process queue is organized by process age, a count of how many task switches have occurred since the process entered the active queue plus the process’ initial priority. The oldest process is at the head of the queue. The OS-9 scheduling algorithm allocates some execution time to each active process.

When a process is placed in the active queue, its age is set to the process assigned priority and the ages of all other processes are incremented. Ages are never incremented beyond 0xffff.

After the time slice of the currently executing process, the kernel executes the process with the highest age.

Preemptive Task Switching

During critical real-time applications, fast interrupt response time is sometimes necessary. OS-9 provides this by preempting the currently executing process when a process with a higher priority becomes active. The lower priority process loses the remainder of its time slice and is re-inserted in the active queue.

Two system global variables affect task switching:

- d_minpty (minimum priority).
- d_maxage (maximum age).
Both variables are initially set in the Init module and are accessible by users with a group ID of zero (super users) through the `F_SETSYS` system call.

If the priority or age of a process is less than $d_{\text{minpty}}$, the process is not considered for execution and is not aged. Usually, this variable is not used and is set to zero.

If the minimum system priority is set above the priority of all running tasks, the system completely shuts down. It can only be recovered by a reset. This makes it crucial to restore $d_{\text{minpty}}$ to a normal level when the critical task(s) finishes.

$d_{\text{maxage}}$ is the maximum age to which processes can be incremented. When $d_{\text{maxage}}$ is activated, tasks are divided into high priority tasks and low priority tasks.

Low priority tasks do not age past $d_{\text{maxage}}$; high priority tasks receive all of the available CPU time and are not aged. Low priority tasks are run only when the high priority tasks are inactive. Usually, this variable is not used and is set to zero.
This chapter describes the five forms of interprocess communication supported by OS-9. The following topics are included:

- Signals
- Alarms
- Events
- Semaphores
- Usemaphores
- Usemaphores
- Operations on Pipes
- Data Modules
Signals

In interprocess communications, a signal is an intentional disturbance in a system. OS-9 signals are designed to synchronize concurrent processes, but you can also use them to transfer small amounts of data. Because they are usually processed immediately, signals provide real-time communication between processes.

Signals are also referred to as software interrupts because a process receives a signal similarly to how a CPU receives an interrupt. Signals enable a process to send a numbered interrupt to another process. If an active process receives a signal, the intercept routine is executed immediately (if installed) and the process resumes execution where it left off. If a sleeping or waiting process receives a signal, the process is moved to the active queue, the signal routine is executed, and the process resumes execution right after the call that removed it from the active queue.

⚠️ If a process does not have an intercept routine for a signal it received, the process is killed. This applies to all signals greater than 1 (wake-up signal).

Each signal has two parts:
- process ID of the destination
- signal code

Signal Codes

OS-9 supports the following signal codes.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wake-up signal. Sleeping/waiting processes receiving this signal are awakened, but the signal is not intercepted by the intercept handler. Active processes ignore this signal. A program can receive a wake-up signal safely without an intercept handler. The wake-up signal is not queued.</td>
</tr>
<tr>
<td>2</td>
<td>Keyboard abort signal. When &lt;control&gt;E is typed, this signal is sent to the last process to perform I/O on the terminal. Usually, the intercept routine performs exit(2) when it receives a keyboard abort signal.</td>
</tr>
<tr>
<td>3</td>
<td>Keyboard interrupt signal. When &lt;control&gt;C is typed, this signal is sent to the last process to perform I/O on the terminal. Usually, the intercept routine performs exit(3) when it receives a keyboard interrupt signal.</td>
</tr>
<tr>
<td>4</td>
<td>Unconditional system abort signal. The super user can send the kill signal to any process, but non-super users can send this signal only to processes with their group and user IDs. This signal terminates the receiving process, regardless of the state of its signal mask, and is not intercepted by the intercept handler.</td>
</tr>
</tbody>
</table>
You can design a signal routine to interpret the signal code word as data. For example, various signal codes could be sent to indicate different stages in a process' execution. This is extremely effective because signals are processed immediately when received.

The following system calls enable processes to communicate through signal.

### Table 3-2. System Calls

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ICPT</td>
<td>Installs a signal intercept routine.</td>
</tr>
<tr>
<td>F_SEND</td>
<td>Sends a signal to a process.</td>
</tr>
<tr>
<td>F_SIGLNGJ</td>
<td>Sets signal mask value and returns on specified stack image.</td>
</tr>
<tr>
<td>F_SIGMASK</td>
<td>Enables/disables signals from reaching the calling process.</td>
</tr>
<tr>
<td>F_SIGRESET</td>
<td>Resets process intercept routine recursion depth.</td>
</tr>
<tr>
<td>F_SLEEP</td>
<td>Deactivates the calling process until the specified number of ticks has passed or a signal is received.</td>
</tr>
</tbody>
</table>

Refer to the following for more information:

- For specific information about these system calls, refer to Chapter 6, OS-9 System Calls. The Microware Ultra C/C++ compiler also supports a corresponding C call for each of these calls.
- Refer to Appendix A, Example Code for a sample program demonstrating how you can use signals.

### Signal Implementation

For some advanced applications, it is helpful to understand how the operating system invokes a signal intercept routine when delivering a signal to a process. It may be necessary to understand the contents of the user stack when executing a process’ signal intercept routine. An application can call a signal intercept routine either non-recursively or recursively.
Non-Recursive Calling

When trying to synchronize signals, most applications call signal intercept routines for a process non-recursive. In the case of non-recursive invocation of the intercept routine, the operating system performs the following tasks to maintain the user stack for the process:

1. Save the process’ main executing context on the process’ system state stack.
2. Loads the process’ global statics pointer associated with the intercept routine (as specified when performing the F_ICPT call).
3. Loads the process’ code constant pointer.
4. Loads the process’ user stack pointer with its value at the time of the signal interruption.
5. Calls the process’ intercept routine.

In some cases, depending on the target system, the C-code application binary interface (ABI) can require the operating system allocate some additional stack space in order to call a C-code intercept routine.

Figure 3-1 shows the user stack contents as it appears in the case of a non-recursive invocation of a signal intercept routine.

Recursive Calling

Normally, the operating system prevents recursive invocation of an intercept routine by incrementing a variable associated with the process, known as the signal mask, when calling the intercept routine. The operating system then decrements the signal mask value upon returning from the intercept routine through the F_RTE system call. When the operating system sees that the signal mask of a process is non-zero, it does not attempt to invoke the intercept routine when it detects a pending signal.
The only way an intercept routine can be called recursively when a signal is pending is if the process explicitly clears its signal mask, through the `F_SIGMASK` or `F_SIGLNGJ` system calls, or implicitly via the user-state `F_SLEEL` and `F_WAIT` services, from within the context of its intercept routine. When calling an intercept routine recursively, the stack contents of the user stack are quite different from the non-recursive case. In order to keep from over consuming the system stack when saving its context, the operating system copies the saved context along with its floating-point context to the user-state stack.

Figure 3-2 shows the user-state stack contents as it appears in the case of a recursive invocation of a signal intercept routine.

Figure 3-2. Recursive Invocation of Signal Intercept Routine

<table>
<thead>
<tr>
<th>Stack information associated with the interrupted thread of execution</th>
<th>High Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long stack frame context as defined by <code>reg&lt; CPU Family&gt;.h</code></td>
<td>User Stack Pointer + sizeof(regs)</td>
</tr>
<tr>
<td>Floating-point context as defined by <code>reg&lt; CPU Family&gt;.h</code></td>
<td>User Stack Pointer + sizeof(regs) or sizeof(fpu_contents)</td>
</tr>
<tr>
<td>Previous user-stack pointer value</td>
<td>User Stack Pointer + 4</td>
</tr>
<tr>
<td>(Optional) C-code stack space as defined by ABI</td>
<td>User Stack Pointer</td>
</tr>
<tr>
<td></td>
<td>Alternate User Stack Pointer</td>
</tr>
</tbody>
</table>

The exact contents of the floating-point context shown in Figure 3-2 can vary within a given processor family, depending on whether or not the processor has hardware support for floating point calculations. If the processor has a hardware floating-point unit (FPU), the contents of the FPU context directly reflect the hardware context. If the processor does not have a hardware FPU, the FPU context area shown in Figure 3-2 contains whatever the FPU software emulation module must preserve on behalf of the process. The actual size of this area can be determined at execution time by consulting the variable `d_fpusize` in the operating system globals area (see `F_GETSYS`).

The PowerPC 6xx series processors containing a full hardware floating-point implementation are the only processors that vary from this described stack format. For this family of processors the FPU context is actually a part of the long stack frame as described in the `regppc.h` header file. The stack format resembles the format described previously with the exception that the FPU context is not separate from the long stack format.
Alarms

User-State Alarms

The user-state alarm requests enable a program to arrange for a signal to be sent to itself. The signal may be sent at a specific time of day or after a specified interval has passed. The program may also request the signal be sent periodically, each time the specified interval has passed.

<table>
<thead>
<tr>
<th>Alarm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ALARM, A_ATIME</td>
<td>Sends a signal at a specific time.</td>
</tr>
<tr>
<td>F_ALARM, A_CYCLE</td>
<td>Sends a signal at the specified time intervals.</td>
</tr>
<tr>
<td>F_ALARM, A_DELET</td>
<td>Removes a pending alarm request.</td>
</tr>
<tr>
<td>F_ALARM, A_RESET</td>
<td>Resets an existing alarm request.</td>
</tr>
<tr>
<td>F_ALARM, A_SET</td>
<td>Sends a signal after the specified time interval.</td>
</tr>
</tbody>
</table>

Cyclic Alarms

A cyclic alarm provides a time base within a program. This simplifies the synchronization of certain time-dependent tasks. For example, a real-time game or simulation might allow 15 seconds for each move. You could use a cyclic alarm signal to determine when to update the game board.

The advantages of using cyclic alarms are more apparent when multiple time bases are required. For example, suppose you are using an OS-9 process to update the real-time display of a car’s digital dashboard.

The process might perform the following functions:

- Update a digital clock display every second.
- Update the car’s speed display five times per second.
- Update the oil temperature and pressure display twice per second.
- Update the inside/outside temperature every two seconds.
- Calculate miles to empty every five seconds.

Each function the process must monitor can have a cyclic alarm, whose period is the desired refresh rate, and whose signal code identifies the particular display function. The signal handling routine might read an appropriate sensor and directly update the dashboard display. The operating system handles all of the timing details.
Time of Day Alarms

You can set an alarm to provide a signal at a specific time and date. This provides a convenient mechanism for implementing a \texttt{cron} type of utility—executing programs at specific days and times. Another use is to generate a traditional alarm clock buzzer for personal reminders.

This type of alarm is sensitive to changes made to the system time. For example, assume the current time is 4:00 and a program sends itself a signal at 5:00. The program can either set an alarm to occur at 5:00 or set the alarm to go off in one hour. Assume the system clock is 30 minutes slow, and the system administrator corrects it. In the first case, the program wakes up at 5:00; in the second case, the program wakes up at 5:30.

Relative Time Alarms

You can use this type of alarm to set a time limit for a specific action. Relative time alarms are frequently used to cause an \texttt{I\_READ} request to abort if it is not satisfied within a maximum time. This can be accomplished by sending a keyboard abort signal at the maximum allowable time and then issuing the \texttt{I\_READ} request. If the alarm arrives before the input is received, the \texttt{I\_READ} request returns with an error. Otherwise, the alarm should be cancelled. The example program \texttt{deton.c} (in \textit{Appendix A, Example Code}) demonstrates this technique.

System-State Alarms

A system-state counterpart exists for user-state alarm function. However, the system-state version is considerably more powerful than its user state equivalent. When a user-state alarm expires, the kernel sends a signal to the requesting process. When a system-state alarm expires, the kernel executes the system-state subroutine specified by the requesting process at a very high priority.

OS-9 supports the following system-state alarm functions:

\begin{table}[h]
\centering
\begin{tabular}{|l|p{10cm}|}
\hline
Alarm & Description \\
\hline\hline
\texttt{F\_ALARM, A\_ATIME} & Executes a subroutine at a specified time \\
\texttt{F\_ALARM, A\_CYCLE} & Executes a subroutine at specified time intervals \\
\texttt{F\_ALARM, A\_DELET} & Removes a pending alarm request \\
\texttt{F\_ALARM, A\_RESET} & Resets an existing alarm request \\
\texttt{F\_ALARM, A\_SET} & Executes a subroutine after a specified time interval \\
\hline
\end{tabular}
\caption{System-State Alarm Descriptions}
\end{table}

The alarm is executed by the kernel process, not by the original requester process. During execution, the user number of the system process is temporarily changed to the original requester. The stack pointer passed to the alarm subroutine is within the system process descriptor and contains about 4KB of free space.
Chapter 3: Interprocess Communication

The kernel automatically deletes the pending alarm requests belonging to a process when that process terminates. This may be undesirable in some cases. For example, assume an alarm is scheduled to shut off a disk drive motor if the disk has not been accessed for 30 seconds. The alarm request is made in the disk device driver on behalf of the I/O process. This alarm does not work if it is removed when the process exits.

The alarm has persistence if the TH_SPOWN bit in the alarm call’s flags parameter is set. This causes the alarm to be owned by the system process rather than the current process.

⚠️ If you use this technique, you must ensure the module containing the alarm subroutine remains in memory until after the alarm expires.

An alarm subroutine must not perform any function resulting in any kind of sleeping or queuing. This includes F_SLEEP, F_WAIT, F_LOAD, F_EVENT, F_ACQLK, F_WAITLK, and F_FORK (if it might require F_LOAD). Other than these functions, the alarm subroutine may perform any task.

One possible use of the system-state alarm function might be to poll a positioning device, such as a mouse or light pen, every few system ticks. Be conservative when scheduling alarms and make the cycle as large as reasonably possible. Otherwise, you could waste a great deal of the available CPU time.

💡 For a program demonstrating how alarms can be used, see Appendix A, Example Code.

Events

OS-9 events are multiple value semaphores. They synchronize concurrent processes that are accessing shared resources such as files, data modules, and CPU time. For example, if two processes need to communicate with each other through a common data module, you may need to synchronize the processes so only one process at a time updates the data module.

Events do not transmit any information, although processes using the event system can obtain information about the event, and use it as something other than a signaling mechanism.

An OS-9 event is a global data structure maintained by the system. The event structure is listed here and is defined in the header file events.h. The following section contains descriptions of each field.
Declaration

typedef struct {
    event_id     ev_id;          /* event id number */
    u_int16      ev_namsz;       /* size of memory to allocate for name */
    u_char       *ev_name;       /* pointer to event name */
    u_int16      ev_link,        /* event use count */
    ev_link;        /* event use count */
    u_int16      ev_perm;        /* event permissions */
    owner_id     ev_owner;       /* event owner (creator) */
    int16        ev_winc,        /* wait increment value */
    ev_sinc;        /* signal increment value */
    int32        ev_value;       /* current event value */
    Pr_desc      ev_quen,        /* next event in queue */
    ev_quen;        /* next event in queue */
    u_char       ev_resv[14];    /* reserved */
} ev_str, *Ev_str;

The structure used by the F_EVENT, EV_INFO request contains a subset of the standard event fields. This structure is listed here and defined in the header file events.h.

typedef struct {
    event_id     ev_id;          /* event id number */
    u_int16      ev_link,        /* event use count */
    u_int16      ev_perm;        /* event permissions */
    owner_id     ev_owner;       /* event owner (creator) */
    int16        ev_winc,        /* wait increment value */
    ev_sinc;        /* signal increment value */
    int32        ev_value;       /* current event value */
} ev_infostr, *Ev_infostr;

Description

The OS-9 event system provides the following facilities:
- To create and delete events
- To permit processes to link/unlink events and obtain event information
- To suspend operation until an event occurs
- For various means of signaling
Fields

`ev_id`
A unique ID is created from this number and the event’s array position.

`ev_namsz`
Size of the event name in bytes.

`ev_name`
The event name must be unique.

`ev_link`
The event use count.

`ev_perm`
The event’s access permissions which are used to verify that a process has access to an event when an `F_EVENT, EV_LINK` operation is performed.

`ev_owner`
The ID of the event owner (creator).

`ev_winc`
The event wait increment. `ev_winc` is added to the event value when a process waits for the event. It is set when the event is created and does not change.

`ev_sinc`
The event’s signal increment. `ev_sinc` is added to the event value when the event is signaled. It is set when the event is created and does not change.

`ev_value`
This four byte integer represents the current event value.

`ev_quen`
A pointer to the next process in the event queue. An event queue is circular and includes all processes waiting for the event. Each time the event is signaled, this queue is searched.

`ev_quep`
A pointer to the previous process in the event queue.

`ev_resv`
Reserved for future use.
Wait and Signal Operations

The two most common operations performed on events are wait and signal.

**Wait**

The wait operation performs the following three functions:
1. Suspends the process until the event is within a specified range
2. Adds the wait increment to the current event value
3. Returns control to the process just after the wait operation was called

**Signal**

The signal operation performs the following three functions:
1. Adds the signal increment to the current event value
2. Checks for other processes to awaken
3. Returns control to the process

These operations enable a process to suspend itself while waiting for an event and to reactivate when another process signals the event has occurred.

To coordinate sharing a non-sharable resource, user programs must:
- Wait for the resource to become available.
- Mark the resource as busy.
- Use the resource.
- Signal the resource is no longer busy.

Due to time slicing, the first two steps in this process must be indivisible. Otherwise, two processes might check an event and find it free. Then, both processes try to mark it busy. This would correspond to two processes using a printer at the same time. The `F_EVENT` service request prevents this from happening by performing both steps in the wait operation.

For example, you can use events to synchronize the use of a printer. You set the initial event value to 0, the wait increment to -1, and the signal increment to 1. When a process wants exclusive use of the printer, it performs an event wait call with a value range of zero and checks to see if a printer is available. If the event value is zero, it applies the wait increment (-1), causing the event value to go to -1 and marking the printer as busy; the process is allowed to use the printer. A negative event value indicates the printer is busy; the process is suspended until the event value comes into range (becomes zero in this case). When a process is finished with the printer, it performs an event signal call, the signal increment is applied causing the event value to be incremented by one, and then the process in range is activated.

For a program demonstrating how events can be used see Appendix A, Example Code.
The F_EVENT System Call

The F_EVENT system call creates named events for this type of application. The name event was chosen instead of semaphore because F_EVENT synchronizes processes in a variety of ways not usually found in semaphore primitives. OS-9 event routines are very efficient and are suitable for use in real-time control applications.

Event variables require several maintenance functions as well as the signal and wait operations. To keep the number of system calls required to a minimum, you can access all event operations through the F_EVENT system call.

Functions exist to enable you to create, delete, link, unlink, and examine events. Several variations of the signal and wait operations are also provided. Specific parameters and functions of each event operation are discussed in the F_EVENT description in Chapter 6, OS-9 System Calls. The following event functions that are supported:

Table 3-5. Event Functions

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_EVENT, EV_ALLCLR</td>
<td>Wait for all bits defined by mask to become clear.</td>
</tr>
<tr>
<td>F_EVENT, EV_ALLSET</td>
<td>Wait for bits defined by mask to become set.</td>
</tr>
<tr>
<td>F_EVENT, EV_ANYCLR</td>
<td>Wait for bits defined by mask to become clear.</td>
</tr>
<tr>
<td>F_EVENT, EV_ANYSET</td>
<td>Wait for bits defined by mask to become set.</td>
</tr>
<tr>
<td>F_EVENT, EV_CHANGE</td>
<td>Wait for any of the bits defined by mask to change.</td>
</tr>
<tr>
<td>F_EVENT, EV_CREAT</td>
<td>Create new event.</td>
</tr>
<tr>
<td>F_EVENT, EV_DELET</td>
<td>Delete existing event.</td>
</tr>
<tr>
<td>F_EVENT, EV_INFO</td>
<td>Return event information.</td>
</tr>
<tr>
<td>F_EVENT, EV_LINK</td>
<td>Link to existing event by name.</td>
</tr>
<tr>
<td>F_EVENT, EV_PULSE</td>
<td>Signal an event occurrence.</td>
</tr>
<tr>
<td>F_EVENT, EV_READ</td>
<td>Read event value without waiting.</td>
</tr>
<tr>
<td>F_EVENT, EV_SET</td>
<td>Set event variable and signal an event occurrence.</td>
</tr>
<tr>
<td>F_EVENT, EV_SETAND</td>
<td>Set event value by ANDing the event value with a mask.</td>
</tr>
<tr>
<td>F_EVENT, EV_SETOR</td>
<td>Set event value by ORing the event value with a mask.</td>
</tr>
<tr>
<td>F_EVENT, EV_SETR</td>
<td>Set relative event variable and signal an event occurrence.</td>
</tr>
<tr>
<td>F_EVENT, EV_SETXOR</td>
<td>Set event value by XORing the event value with a mask.</td>
</tr>
<tr>
<td>F_EVENT, EV_SIGNL</td>
<td>Signal an event occurrence.</td>
</tr>
<tr>
<td>F_EVENT, EV_TSTSET</td>
<td>Wait for all bits defined by mask to clear; set these bits.</td>
</tr>
<tr>
<td>F_EVENT, EV_UNLNK</td>
<td>Unlink event.</td>
</tr>
<tr>
<td>F_EVENT, EV_WAIT</td>
<td>Wait for event to occur.</td>
</tr>
<tr>
<td>F_EVENT, EV_WAITR</td>
<td>Wait for relative to occur.</td>
</tr>
</tbody>
</table>
Semaphores

Semaphores support exclusive access to shared resources. Semaphores are similar to events in the way they provide applications with mutually exclusive access to data structures. Semaphores differ from events in that they are strictly binary in nature, which increases their efficiency.

Since using C bindings is the preferred method of accessing OS-9 semaphores, F_SEMA is not documented in Chapter 8. Refer to the Ultra C/C++ Library Reference for information on the os_sema calls.

OS-9 supports the semaphore routines shown in the following table:

Table 3-6.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_os_sema_init()</td>
<td>Initialize the semaphore data structure for use.</td>
</tr>
<tr>
<td>_os_sema_p()</td>
<td>Reserve a semaphore.</td>
</tr>
<tr>
<td>_os_sema_term()</td>
<td>Terminate the use of a semaphore data structure.</td>
</tr>
<tr>
<td>_os_sema_v()</td>
<td>Release a semaphore.</td>
</tr>
</tbody>
</table>

A single semaphore system call, F_SEMA, provides all of the semaphore functionality. F_SEMA requires the following two parameters:

- One indicating which operation is being performed on the semaphore
- A pointer to the semaphore structure

Unlike events, there is no system call provided to create a semaphore; you must provide the storage for the semaphore. Because semaphores are typically used to protect specific resources, you should declare the semaphore structure as part of the resource structure. In addition, you must be certain that this storage is properly initialized (to a known value) prior to initializing the semaphore structure for use (via _os_sema_init()). In many cases this storage initialization is handled for you by OS-9 (such as inside a data module or part of global data).

For a program demonstrating how you may use semaphores, see Appendix A, Example Code.

A typical application using semaphores might create a data module containing the memory for the intended resource and its associated semaphore. By using a data module for implementing semaphores, applications can use OS-9 module protection mechanisms to protect the semaphore.

Once you have created and initialized the semaphore data module, additional processes within the application may use the semaphore by linking to the semaphore data module. You must create the semaphore data module with appropriate permissions to allow the other processes within the application to link to and use the semaphore and its resource.
Semaphore States

A semaphore has two states:

Reserved: When a semaphore is reserved, any process attempting to reserve the semaphore waits. This includes the process that has the semaphore reserved.

Free: When a semaphore is free, any process may claim the semaphore.

Acquiring Exclusive Access

To acquire exclusive access to a resource, a process may use the `_os_sema_p()` C binding to reserve the semaphore. If the semaphore is already busy, the process is suspended and placed at the end of the wait queue of the semaphore.

Releasing Exclusive Access

To release exclusive access to a resource, a process may use the `_os_sema_v()` C binding to release the semaphore. When the owner process releases the semaphore, the first process in the semaphore queue is activated and retries the reserve operation on the semaphore.

The definition for the semaphore structure can be found in the `semaphore.h` header file. Semaphores use the following data structure:

```c
/* Semaphore structure definition */
typedef struct semaphore {
    sema_val
        s_value;       /* semaphore value (free/busy status) */
    u_int32  s_lock;    /* semaphore structure lock (use count) */
    Pr_desc s_qnext,   /* wait queue for process descriptors */
        s_qprev;   /* wait queue for process descriptors */
    u_int32  s_length, /* current length of wait queue */
        s_owner,  /* current owner of semaphore (process ID) */
        s_user,   /* reserved for users */
        s_flags,  /* general purpose bit-field flags */
        s_sync,   /* integrity sync code */
        s_reserved[3]; /* reserved for system use */
} semaphore, *Semaphore;
```
Usemaphores

OS-9 usemaphores (unlocking, unlinking semaphores) binary semaphores that are automatically unlocked, if necessary, and unlinked when a process terminates. They synchronize concurrent processes that are accessing shared resources such as files, data modules, or CPU time. For example, if two processes need to access a non-sharable resource such as a printer, you may need to synchronize the processes so only one process at a time uses the printer. Further, if the process currently using the printer terminates unexpectedly you want the printer automatically freed so other processes may access it. Usemaphores have a facility that allows processes to know that a usemaphore was freed automatically by OS-9. This allows them to clean up after the terminated process. For example, if it's known that the last process using the printer terminated unexpectedly, the next process to use the printer might want to issue a formfeed to ensure that printing starts in the expected place.

OS-9 keeps track of each usemaphore that a process currently has a link to as well as the set of semaphores that a process currently owns. When a process terminates, any semaphores that are currently owned are released (and marked as needing a reset) and any semaphores that are currently linked to by the process are unlinked.

Usemaphores are implemented using specialized events. Refer to the previous section for more detailed information about events. The system calls related to usemaphores are declared in the header file `semaphore.h`.

Description

The OS-9 usemaphore system provides the following facilities:

- creates and delete usemaphores
- permits processes to link/unlink usemaphores
- suspends operation until a usemaphore is available
- checks for the availability of a usemaphore without blocking
- releases a usemaphore

P and V Operations

The two most common operations performed on usemaphores are “P” (wait for the usemaphore) and “V” (release the usemaphore).

P (Wait)

The P operation performs the following three functions:

1. suspends the process until the usemaphore is available (unowned)
2. marks the current process as the usemaphore owner
3. adds the usemaphore to the list of the process’ owned usemaphores
Chapter 3: Interprocess Communication

**V (Release)**

The P operation performs the following three functions:

1. removes the usemaphore from the list of the process’ owned usemaphores
2. marks the usemaphore as unowned
3. activates one process waiting for the usemaphore, if there is one or more waiting

These operations enable a process to suspend itself while waiting for a usemaphore and to reactivate when another process releases the usemaphore.

To coordinate sharing a non-sharable resource, user programs must:

- Wait for the resource to become available.
- Mark the resource as busy.
- Use the resource.
- Signal the resource is no longer busy.

Due to time slicing, the first two steps in this process must be indivisible. Otherwise, two processes might check a usemaphore and find it unowned. Then, both processes try to mark it owned. This would correspond to two processes using a printer at the same time. The usemaphorer service request prevents this from happening by performing both steps atomically in the P operation.

For example, you can use a usemaphore to synchronize the use of a printer. You set the initial usemaphore value to one (unowned). When a process wants exclusive use of the printer, it performs a usemaphore P call to check if the printer is available. If the event value is one, it claims ownership of the usemaphore, thus marking the printer as busy. A usemaphore value of zero indicates the printer is busy; the process is suspended until the usemaphore is released (becomes one). When a process is finished with the printer, it performs a V usemaphore call, the usemaphore is marked unowned, and then any process waiting is activated.

Also available is a “try P” operation that allows a process to check the usemaphore for being unowned. If it is unowned at the time of the call it is marked as being owned by the calling process and SUCCESS is returned. If the usemaphore is currently owned, EAGAIN is returned indicating the usemaphore could not be acquired without blocking.

For a program demonstrating how usemaphores can be used see Appendix A, Example Code.
Reset

Usemaphores sometimes need to be reset. A usemaphore needs to be reset after OS-9 automatically marks it as unowned due to the owning process having terminated. Usemaphores acted upon in this way must be reset because the status of the resource they were protected is indeterminate.

The reset operation performs these basic steps atomically:

1. ensures that the usemaphore needs to be reset
2. marks the current process as the usemaphore’s owner
3. clears the fact that a reset needs to be performed on the usemaphore

Doing these three steps atomically ensures that only one process is allowed to successfully reset a usemaphore.

After the reset is complete, use the normal V operation to release the semaphore.

Both of the claiming operations, P and try P, will return errors if used on a usemaphore that needs to be reset.

The F_EVENT, F_USEMA System Call

Variations of the F_EVENT system calls manipulate named usemaphores. OS-9 usemaphore routines are very efficient and are suitable for use in real-time control applications.

Functions exist to enable you to create, delete, link, unlink, P, try P, V, and reset usemaphores. Specific parameters and functions of each usemaphore operation are discussed in the F_EVENT, F_USEMA description in Chapter 6, OS-9 System Calls. The following usemaphore functions are supported:

Table 3-7. Supported Usemaphore Functions

<table>
<thead>
<tr>
<th>Usemaphore</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_EVENT, EV_CREAT</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_DELET</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_LINK</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_RESET</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_SIGNL</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_TRYWAIT</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_UNLNK</td>
<td>F_USEMA</td>
</tr>
<tr>
<td>F_EVENT, EV_WAIT</td>
<td>F_USEMA</td>
</tr>
</tbody>
</table>
Pipes

An OS-9 pipe is a first-in first-out (FIFO) buffer that enables concurrently executing processes to communicate data; the output of one process (the writer) is read as input by a second process (the reader). Communication through pipes eliminates the need for an intermediate file to hold data.

PIPEMAN is the OS-9 file manager supporting interprocess communication through pipes. PIPEMAN is a re-entrant subroutine package called for I/O service requests to a device named `/pipe`.

A pipe contains 128 bytes, unless a different buffer size is specified when the pipe is created. Typically, a pipe is used as a one-way data path between two processes:

- writing
- reading

The reader waits for the data to become available and the writer waits for the buffer to empty. However, any number of processes can access the same pipe simultaneously: PIPEMAN coordinates these processes. A process can even arrange for a single pipe to send data to itself. You can use this to simplify type conversions by printing data into the pipe and reading it back using a different format.

Data transfer through pipes is extremely efficient and flexible. Data does not have to be read out of the pipe in the same size sections in which it was written.

You can use pipes much like signals to coordinate processes, but with these advantages:

- Longer messages (more than 32 bits)
- Queued messages
- Determination of pending messages
- Easy process-independent coordination (using named pipes)

Named and Unnamed Pipes

OS-9 supports both named and unnamed (anonymous) pipes. The shell uses unnamed pipes extensively to construct program pipelines, but user programs can also use them. Unnamed pipes can be opened only once. Independent processes may communicate through them only if the pipeline was constructed by a common parent to the processes. This is accomplished by making each process inherit the pipe path as one of its standard I/O paths.

The use of named pipes is similar to that of unnamed pipes. The main difference is a named pipe can be opened by several independent processes, which simplifies pipeline construction. Other specific differences are noted in the following sections.
Operations on Pipes

Creating Pipes

The \texttt{I\_CREATE} system call is used with the pipe file manager to create new named or
unnamed pipe files.

You can create pipes using the pathlist /pipe (for unnamed pipes, pipe is the name
of the pipe device descriptor) or /pipe/<name> (<name> is the logical file name being
created). If a pipe file with the same name already exists, an error (\texttt{EOS\_CEF}) is
returned. Unnamed pipes cannot return this error.

All processes connected to a particular pipe share the same physical path descriptor.
Consequently, the path is automatically set to update mode regardless of the mode
specified at creation.

You can specify access permissions. They are handled similarly to permissions on
files in random block file systems.

The size of the default FIFO buffer associated with a pipe is specified in the pipe
device descriptor. To override this default when creating a pipe, set the initial file
size bit of the mode parameter and pass the desired file size in the parameter block.

If no default or overriding size is specified, a 128-byte FIFO buffer is created.

You can rename a named pipe to an unnamed pipe and an unnamed pipe to a
named pipe.

Opening Pipes

When accessing unnamed pipes, \texttt{I\_OPEN}, like \texttt{I\_CREATE}, opens a new anonymous
pipe file. When accessing named pipes, \texttt{I\_OPEN} searches for the specified name
through a linked list of named pipes associated with a particular pipe device.

Opening an unnamed pipe is simple, but sharing the pipe with another process is
more complex. If a new path to /pipe is opened for the second process, the new
path is independent of the old one.

The only way for more than one process to share the same unnamed pipe is through
the inheritance of the standard I/O paths through the \texttt{F\_FORK} call. As an example,
the following C language pseudocode outline describes a method the shell can use to
construct a pipeline for the command \texttt{dir -u ! qsort}. It is assumed paths 0 and 1
are already open.
Chapter 3: Interprocess Communication

StdInp = _os_dup(0)  save the shell’s standard input
StdOut = _os_dup(1)  save shell’s standard output
_os_close(1)        close standard output
_os_open("/pipe")  open the pipe (as path 1)
_os_fork("dir","-u") fork "dir" with pipe as standard output
_os_close(0)        free path 0
_os_dup(1)          copy the pipe to path 0
_os_close(1)        make path available
_os_dup(StdOut)     restore original standard out
_os_fork("qsort")  fork qsort with pipe as standard input
_os_close(0)        get rid of the pipe
_os_dup(StdInp)     restore standard input
_os_close (StdInp)  close temporary path
_os_close (StdOut)  close temporary path

The main advantage of using named pipes is several processes can communicate through the same named pipe without having to inherit it from a common parent process. For example, the above steps can be approximated by the following command:

$ dir -u >/pipe/temp & qsort </pipe/temp

The OS-9 shell always constructs its pipelines using the unnamed /pipe descriptor.

Read/Readln

The I_READ and I_READLN system calls return the next bytes in the pipe buffer. If not enough data is ready to satisfy the request, the process reading the pipe is put to sleep until more data becomes available.

The end-of-file is recognized when the pipe is empty and the number of processes waiting to read the pipe is equal to the number of users on the pipe. If any data was read before the end-of-file was reached, an end-of-file error is not returned. However, the returned byte count is the number of bytes actually transferred, which is less than the number requested.

The read and write system calls are faster than the readln and writeln system calls because PIPEMAN does not have to check for carriage returns and the loops moving data are tighter.
Write/Writeln

The \texttt{I\_WRITE} and \texttt{I\_WRITELN} system calls work in almost the same way as \texttt{I\_READ} and \texttt{I\_READLN}. A pipe error (\texttt{EOS\_WRITE}) is returned when all the processes with a full unnamed pipe open attempt to write to the pipe. Since there is no reader process, each process attempting to write to the pipe receives the error and the pipe remains full.

When named pipes are being used, PIPEMAN never returns the \texttt{EOS\_WRITE} error. If a named pipe becomes full before a process receiving data from the pipe has opened it, the process writing to the pipe is put to sleep until a process reads the pipe.

Close

When a pipe path is closed, its path count is decremented. If no paths are left open on an unnamed pipe, its memory is returned to the system. With named pipes, its memory is returned only if the pipe is empty. A non-empty pipe (with no open paths) is artificially kept open, waiting for another process to open and read from the pipe. This permits pipes to be used as a type of temporary, self-destructing RAM disk file.

Getstat/Setstat

PIPEMAN supports a wide range of status codes enabling the insertion of pipes as a communications channel between processes where an random block file (RBF) or serial character file (SCF) device would normally be used. For this reason, most RBF and SCF status codes are implemented to perform without returning an error. The actual function may differ slightly from the other file managers, but it is usually compatible.

GetStat Status Codes Supported by PIPEMAN

The following table shows only the supported GetStat status codes. All other codes return an \texttt{EOS\_UNKSVC} error (unknown service request).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{I_GETSTAT, SS_DEVOPT}</td>
<td>Read the default path options for the device.</td>
</tr>
<tr>
<td>\texttt{I_GETSTAT, SS_EOF}</td>
<td>Test for end-of-file condition.</td>
</tr>
<tr>
<td>\texttt{I_GETSTAT, SS_FD}</td>
<td>Read the pseudo file descriptor image for the pipe associated with the specified path.</td>
</tr>
<tr>
<td>\texttt{I_GETSTAT, SS_FDINFO}</td>
<td>Read the pseudo file descriptor sector for the pipe specified by a sector number.</td>
</tr>
<tr>
<td>\texttt{I_GETSTAT, SS_LUOPT}</td>
<td>Read the logical unit options section.</td>
</tr>
<tr>
<td>\texttt{I_GETSTAT, SS_PATHOPT}</td>
<td>Read the path options section of the path descriptor.</td>
</tr>
<tr>
<td>\texttt{I_GETSTAT, SS_READY}</td>
<td>Test whether data is available in the pipe. It returns the number of bytes in the buffer.</td>
</tr>
<tr>
<td>\texttt{I_GETSTAT, SS_SIZE}</td>
<td>Return the size of the associated pipe buffer.</td>
</tr>
</tbody>
</table>
SetStat Status Codes Supported by PIPEMAN

The table below shows the SetStat status codes supported by PIPEMAN.

Table 3-9. SetStat Status Codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_SETSTAT, SS_ATTR</td>
<td>Changes the file attributes of the associated pipe.</td>
</tr>
<tr>
<td>I_SETSTAT, SS_DEVOPT</td>
<td>Does nothing, but returns without error.</td>
</tr>
<tr>
<td>I_SETSTAT, SS_FD</td>
<td>Writes the pseudo file descriptor image for the pipe.</td>
</tr>
<tr>
<td>I_SETSTAT, SS_LUOPT</td>
<td>Does nothing, but returns without error.</td>
</tr>
<tr>
<td>I_SETSTAT, SS_PATHOPT</td>
<td>Does nothing, but returns without error.</td>
</tr>
<tr>
<td>I_SETSTAT, SS_RELEASE</td>
<td>Releases the device from the SS_SENDSIG processing before data becomes available.</td>
</tr>
<tr>
<td>I_SETSTAT, SS_RENAME</td>
<td>Changes the name of a named pipe, changes a named pipe to an unnamed pipe, and changes an unnamed pipe to a named pipe.</td>
</tr>
<tr>
<td>I_SETSTAT, SS_SIZE</td>
<td>Resets the pipe buffer if the specified size is zero. Otherwise, it has no effect, but returns without error.</td>
</tr>
<tr>
<td>I_SETSTAT, SS_SENDSIG</td>
<td>Sends the process the specified signal when data becomes available.</td>
</tr>
</tbody>
</table>

The I_MAKDIR and I_CHDIR service requests are illegal service routines on pipes. They return EOS_UNK SVC.

Pipe Directories

Opening an unnamed pipe in the Dir mode enables it to be opened for reading. In this case, PIPEMAN allocates a pipe buffer and pre-initializes it to contain the names of all open named pipes on the specified device. Each name is null-padded to make a 32-byte record. This enables utilities that normally read an RBF directory file sequentially to work with pipes.

PIPEMAN is not a true directory device; commands like chd and makdir do not work with /pipe.

The head of a linked list of named pipes is maintained in the logical unit static storage of the pipe device. If several pipe descriptors with different default pipe buffer sizes are on a system, the I/O system notices the same file manager, port address (usually zero), and logical unit number are being used. It does not allocate new logical unit static storage for each pipe device and all named pipes will be on the same list.

For example, if two pipe descriptors exist, a directory of either device reveals all the named pipes for both devices. If each pipe descriptor has a unique port address (0, 1, 2, etc.) or unique logical unit number, the I/O system allocates different logical unit static storage for each pipe device. This produces expected results.
Data Modules

OS-9 data modules enable multiple processes to share a data area and to transfer data among themselves. A data module must have a module header and a valid CRC to be loaded into memory. Data modules can be non-reentrant (modifiable). One or more processes can share and modify the contents of a data module.

OS-9 does not have restrictions as to the content, organization, or use of the data area in a data module. These considerations are determined by the processes using the data module.

OS-9 does not synchronize processes using a data module. Consequently, thoughtful programming, usually involving events or signals, is required to enable several processes to update a shared data module simultaneously.

Creating Data Modules

The `F_DATMOD` system call creates a data module with a specified set of attributes, data area size, and module name. The data area is cleared automatically. The data module is created and entered into the calling process’ current module directory. A CRC value is not computed for the data module when it is created.

It is essential the data module header and name string not be modified to prevent the module from becoming unknown to the system.

The Microware C compiler provides several C calls to create and use data modules directly. These include the `_mkdata_module()` and `_os_datmod()` calls which are specific to data modules, and the `modlink()`, `modload()`, `munlink()`, `munload()`, `_os_link()`, `_os_unlink()`, `_os_unload()`, `_os_setcrc()`, and `_setcrc()` calls that apply to all OS-9 modules.

For more information on these calls, refer to the Using Ultra C/C++ manual.

The Link Count

Like all OS-9 modules, data modules have an associated link count. The link count is a counter of how many processes are currently linked to the module. Generally, the module is taken out of memory when this count reaches 0. If you want the module to remain in memory when the link count is zero, make the module sticky by setting the sticky bit in the module header attribute byte.

Saving to Disk

If a data module is saved to disk, you can use the `dump` utility to examine the module format and contents. You can save a data module to disk with the `save` utility or by writing the module image into a file. If the data module was modified since its CRC value was created, the saved module CRC will be bad and it becomes impossible to reload the module into memory.

To allow the module to be reloaded, use the `F_SETCRC` system call or the `_setcrc()` C library call before writing the module to disk. Or, use the `fixmod` utility after the module has been written to disk.
This chapter explains how to install, execute, and terminate subroutine libraries. It also explains how to install and execute trap handlers. It includes the following topics:

- Subroutine Libraries
- Trap Handlers
Subroutine Libraries

An OS-9 subroutine library is a module containing a set of related or frequently used subroutines. Subroutine libraries enable distinct processes to share common code. Any user program may dynamically link to the user subroutine library and call it at execution time.

Although subroutine libraries reduce the size of the execution program, they do not accomplish anything that could not be done by linking the program with the appropriate library routines at compilation time. In fact, programs calling subroutine libraries execute slightly slower than linked programs performing the same function. A program can link to a maximum of sixteen subroutine libraries, numbered from zero to fifteen.

Microware provides a standard subroutine library of I/O conversions for C language programs. Subroutine library identifier zero is reserved for the Microware csl subroutine library. Identifiers one through nine are also reserved for Microware use.

Like standard OS-9 program modules, subroutine libraries have one entry point and may have their own global static storage. The module type of subroutine library modules is MT_SUBROUT and the module language is ML_OBJECT.

Subroutine functions are usually executed as though they were called directly by the main program. System calls or other operations that could be performed by the calling module can also be performed in a subroutine library.

Installing and Executing Subroutine Libraries

To install a subroutine library, a user program must use the F_SLINK system call. F_SLINK attempts to link to the subroutine library. If the link is successful, it allocates and initializes the global static storage and returns pointers to the library’s entry point and to the library’s global static storage area.

Typically, a main program’s first call to a subroutine library calls an initialization routine. The initialization routine usually has very little to do, but could be used to open files, link to additional subroutine libraries or data modules, or perform other startup activities.

The main program must save the entry pointer and static storage pointer returned by F_SLINK to enable subsequent calls to the subroutine library.

The OS-9 C library provides functions to install and call subroutine libraries. The _sliblink() function installs a specified subroutine module saving the subroutine library’s entry and global static storage pointers in the global arrays _sublibs[] and _submems[], respectively.

You can use the _subcall function to call an existing subroutine library. For example, suppose the main program reference in C is the following statement:

  my_function(p1, p2, p3, p4)
The _subcall reference in 80386 assembler would be as follows:

```assembly
    my_function: call _subcall
dc.l SUB_LIB_NUM
dc.l SUB_MY_FUNCTION
```

_subcall does the following:

- Retrieves the subroutine library and function identifiers
- Adjusts the program stack
- Dispatches to the subroutine library entry point with the correct global static storage configuration

The return from the subroutine in the subroutine library takes the flow of execution directly back to the initial function reference in the main program.

To create a subroutine library, you must create a table of _subcall calls, and subroutine library and function identifiers as previously described. In addition, some dispatch code must be written in the subroutine library. For more information, refer to the subroutine library example provided in the The Subroutine Library section of Appendix A, Example Code.

**Terminating Subroutine Libraries**

Programs using subroutine libraries do not need to explicitly terminate the use of the libraries. When a process terminates, the OS-9 kernel unlinks any subroutine libraries and releases their resources on behalf of the process. But, a program may terminate the use of a subroutine library explicitly by performing a _sliblink() call. In this case, you must specify a null string for the subroutine library name and the associated subroutine library identifier. This unlinks the subroutine library and returns its resources to the system.

These are the resources associated with the calling process’ invocation of the subroutine library and do not affect the resources of other processes using the same subroutine library.

**Trap Handlers**

Trap handlers are similar to subroutine libraries with the following exceptions:

- When a trap handler is linked, the kernel calls the trap initialization entry point. The kernel does not call an initialization entry point when the subroutine library is linked. Instead, the main program must call the initialization routine, if one exists.
- A trap handler may have more than one entry point; there is exactly one entry point in a subroutine library.
- Trap handlers only execute in system state; subroutine libraries execute in the same state as the main program.
Chapter 4: Subroutine Libraries and Trap Handlers

- There may be a termination routine for a trap handler; there is no explicit termination entry point for a subroutine library.
- Dispatching to subroutine libraries does not involve the kernel in any way.

Trap handlers have three execution entry points:

- A trap execution entry point
- A trap initialization entry point
- A trap termination entry point

Trap handler modules are of module type MT_TRAPLIB and module language ML_OBJECT.

The trap module routines are usually executed as though they were called with the standard function call instruction, except for minor stack differences. Any system calls or other operations that could be performed by the calling module are usable in the trap module.

An example C trap handler is included in Appendix A, Example Code.

Installing and Executing Trap Handlers

A user program installs a trap handler by executing the F_TLINK system request. When this is done, the OS-9 kernel performs the following functions:

- Links to the trap module
- Allocates and initializes its static storage, if any
- Executes the trap module’s initialization routine

Typically, the initialization routine has very little to do. It can open files, link to additional trap or data modules, or perform other startup activities. It is called only once per trap handler in any given program.

A trap module used by a program is usually installed as part of the program initialization code. At initialization, a particular trap number (0 - 15) is specified that refers to the trap vector. Numbers zero through nine are reserved for Microware use.

The OS-9 relocatable macro assembler has a special mnemonic (tcall) for making trap library function calls. The syntax for the tcall mnemonic is as follows:

```
tcall <trap library number>, <function code>
```

Usually, a table of tcalls with associated labels is created for calling the trap library functions from C programs. For example:

```
_asm ("
    func1: tcall T_TrapLib1, T_func1
    func2: tcall T_TrapLib1, T_func2
    .
    .
    funcN: tcall T_TrapLib1, T_funcN
"");
```
Then, the main program can call the functions in the trap library as follows:

```c
func1(param1, param2, ..., paramN);
```

The `tcall` mnemonic causes the program to dispatch the OS-9 kernel similarly to a system service request. The OS-9 kernel then uses the trap library identifier to dispatch to the associated trap handler module.

To create a trap handler library, you should create a table of `tcall` calls with trap handler and function identifiers as previously described. In addition, some dispatch and function return codes must be written in the trap handler module.

For more information, refer to the trap handler example provided in Appendix A, Example Code.

From user programs, you can delay installing a trap module until the first time it is actually needed. If a trap module has not been installed for a particular trap when the first `tcall` is made, OS-9 checks the program’s exception entry offset. The program is aborted if this offset is zero. Otherwise, OS-9 passes control to the exception routine. At this point, the trap handler can be installed, and the first `tcall` reissued.
This chapter describes the lock structure definition, lock creation, signal lock relationships, and FIFO buffer usage. It includes the following topics:

- Overview
- Preallocate Locks as Part of the Resource
- Signals and Locks
- FIFO Buffers
Overview

The OS-9 I/O system uses resource locking calls to provide exclusive access to critical regions and help ensure proper resource management. If you write file managers or drivers, review this chapter for an explanation of resource locking and implementation details.

Resource locking helps prevent data corruption by limiting process access to critical sections of code; it protects data structures from simultaneous modification by multiple processes. To manage processes waiting to enter critical areas, resource locking provides an associated queue. The queue orders lock requests according to the relative priority of the calling process.

Resource locking is only available in system state.

The following are the OS-9 resource locking calls. Refer to Chapter 6, OS-9 System Calls for a detailed description of each call.

Table 5-1. OS-9 Resource Locking Calls

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ACQLK</td>
<td>Acquire ownership of a resource lock.</td>
</tr>
<tr>
<td>F_CAQLK</td>
<td>Conditionally acquire ownership of a resource lock.</td>
</tr>
<tr>
<td>F_CRLK</td>
<td>Create a new resource lock descriptor.</td>
</tr>
<tr>
<td>F_DELLK</td>
<td>Delete an existing lock descriptor.</td>
</tr>
<tr>
<td>F_RELLK</td>
<td>Release ownership of a resource lock.</td>
</tr>
<tr>
<td>F_WAITLK</td>
<td>Activate the next process waiting to acquire a lock, and suspend the current process.</td>
</tr>
</tbody>
</table>

Lock Structure Definition

The lock structure definition for the kernel is as follows:

typedef struct lock_desc *lock_id;
typedef struct lock_desc {
    lock_id l_id; /* lock identifier */
    Pr_desc l_owner, /* current owner */
    l_lockqn, /* next process in lock list */
    l_lockqp; /* previous process in lock list */
} lk_desc, *Lk_desc;

Conceptually, this structure could be shown as:

Figure 5-1. Lock Structure
The next and previous boxes represent the queuing capabilities of resource locking calls. When one or more processes are waiting to acquire a lock, they work with corresponding process descriptor fields to determine which process should receive the lock next. Lock requests are queued in the order in which they are received, according to their relative priority. Higher priority processes are queued ahead of lower priority processes.

**Create and Delete Resource Locks**

OS-9 provides a call to dynamically create and initialize a resource lock. The F_CRLK call allocates data space for the lock, initializes the associated queue, and sets the lock ownership to a free state. A lock identifier is returned for subsequent use by the lock calls.

The lock identifier is the address of the lock structure.

When a lock is no longer needed, you can use the F_DELLK call to deallocate it. The data space for the lock is returned to the system. Prior to deleting a lock you must ensure any processes waiting in its queue are removed from the queue and re-activated. The F_DELLK call does not check the queue for waiting processes; it is the responsibility of the application to empty the waiting queue of the lock.

The following C language example demonstrates how to dynamically create and delete a resource lock.

```c
#include <types.h>
#include <lock.h>

Lk_desc lock;       /* declare a pointer to a lock structure */

/* dynamically allocate a new lock */
if ((error = _os_crlk(&lock)) != SUCCESS)
    return error;

/* an example use of the lock */
if ((error = _os_acqlk(lock, &signal)) != SUCCESS)
    return error;

/* delete the lock */
_os_dellk (lock);
```
Preallocate Locks as Part of the Resource

To reduce the overhead and memory fragmentation caused by dynamically created locks, you can declare the lock structure for a given resource as part of the resource structure. Prior to using the lock, you must initialize the lock structure fields.

For example:

```c
#include <types.h>
#include <const.h>
#include <lock.h>
#include <process.h>

/* Resource declaration with the lock structure included */
struct xyz {
    lk_desc lock;
    int a;
    char *b;
    unsigned c;
} resource;

/* set the lock identifier */
resource.lock.l_id = &resource.lock;

/* declare the lock free */
resource.lock.l_owner = NULL;

/* initialize the lock structure's queue pointers */
resource.lock.l_lockqp = resource.lock.l_lockqn =
    FAKEHD(Pr_desc, resource.lock.l_lockqn, p_lockqn);

// The FAKEHD initialization macro is located in the const.h header file.
```

At this point, the lock within the resource structure is ready for use. Subsequent lock calls are made by passing the address of the lock as its identifier. The following acquire lock example demonstrates this:

```c
/* use a lock declared within a resource structure */
if ((error = _os_acqlk(&resource.lock, &signal)) != SUCCESS)
    return error;
```
Signals and Locks

Locks have an associated queue used for suspending processes waiting to acquire a busy lock. If the lock is busy, the acquiring process is placed in the queue according to the relative priorities of any other waiting processes. When the owner process releases its ownership of the lock, the next process in the queue is activated and granted sole ownership of the lock. On the new owner's next time slice, the process returns from the acquire lock system call without error and continues to execute from that point. Normally, this is the proper sequence of events; the active process has ownership of the resource. But it is possible for a process to be prematurely activated prior to acquiring ownership of the lock.

If, for example, the process receives a signal while waiting in the lock queue, the process is activated without acquiring the lock and the acquire lock call returns an EOS_SIGNAL error. To avoid this error, it is critical that applications check the return value of the acquire lock calls to validate whether or not the active process has gained ownership of the lock. If a process is activated by a signal, the application writer determines how to respond to the error condition. The application may abort its operation and return with an error, or ignore the signal and attempt to re-acquire the lock. Depending on the application, either action may be appropriate.

Signal Sensitive Locks

The following example uses a lock to protect a critical section of code modifying a non-sharable resource. This example is completely sensitive to any signals a process may receive while waiting to acquire the lock. A process receiving a signal while waiting in this lock's queue is activated and the acquire lock call returns the error EOS_SIGNAL.

```
#include <lock.h>
#include <types.h>
#include <errno.h>

lk_desc lock;
signal_code signal;

/* acquire exclusive access to the resource */
if ((error = _os_acqlk(&lock, &signal)) != SUCCESS)
    return error;

/* critical section */

/* release exclusive access to the resource and activate the next process */
_os_rellk(&lock);
```
Ignoring Signals

There may be situations when a process is prematurely activated by a signal, and it is not appropriate for the application to simply return an error. In this case, the application may ignore the activating signal and error and attempt to re-acquire the lock.

The activating signal is not lost. The operating system queues it on behalf of the process. Upon return from system state, the signal is delivered to the process through its signal intercept routine.

This acquire lock example demonstrates how to use locks that ignore signals.

```c
#include <lock.h>
#include <types.h>
#include <errno.h>

lk_desc lock;
signal_code signal;
while ((error = _os_aqclk(&lock, &signal)) != SUCCESS) {
    if (error == EOS_SIGNAL)
        continue; /* signal received, ignore it */
    else
        return error; /* some other erroneous condition */
    <critical section>
    /*release exclusive access to resource and activate the next process*/
    _os_rellk(&lock);
}
```

Below is an example of a lock that is partially sensitive to signals. It ignores any non-deadly signals a process might receive, but returns an error for any deadly signal. In this case, a deadly signal is any signal with a value less than 32.

```c
#include <lock.h>
#include <types.h>
#include <errno.h>

lk_desc lock;
signal_code signal;
while ((error = _os_aqclk(&lock, &signal)) != SUCCESS) {
    if (error == EOS_SIGNAL) {
        if (signal >= 32)
            continue; /* signal greater than 32 received, ignore it */
        else
            return error; /* signal less than 32 received */
    }
    else break; /* some other erroneous condition */
    <critical section>
    /*release exclusive access to resource and activate the next process*/
    _os_rellk(&lock);
}
```
FIFO Buffers

You can use locks to synchronize the reader and writer of a FIFO buffer resource. The resource has an associated lock; any reader or writer requiring access to the resource must first acquire the resource lock. After acquiring the resource, the process may proceed to modify the buffer. If during the course of modification the reader empties the buffer or the writer fills the buffer, the `F_WAITLK` call suspends the process to wait for more data to enter or leave the buffer.

```c
#include <lock.h>
#include <types.h>
#include <errno.h>

lk_desc lock;
signal_code signal;

/* acquire exclusive access to the resource */
if ((error = _os_acqlk(&lock, &signal)) != SUCCESS) return error;

/* loop until total number of bytes is read/written */
while (bytes_read/bytes_written < bytes_to_read/bytes_to_write) {

    /* check for bytes available to read/write */
    if (bytes_available == 0) {

        /* no bytes available, so release the ownership of the lock, */
        /* activate the reader/writer if it is waiting, and unconditionally */
        /* suspend the current reader/writer */
        if ((error = _os_waitlk(&lock, &signal)) != SUCCESS)
            return error;

    } else {

        <transfer bytes>

    }

} /* number of bytes to read/write has been satisfied, so release lock */
_os_rellk(&lock);
```
Process Queuing

The diagram below is a conceptual illustration of the queuing process and the effect of various calls on the lock structure.

Figure 5-2. Effect of Various Calls on the Lock Structure

<table>
<thead>
<tr>
<th>Lock ID</th>
<th>Owner Process</th>
<th>Next</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F_ACQLK</td>
<td>Process: 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Priority: 90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this call, the owner process= 0, so the lock is available to process 1. Process 1 now owns the lock.

<table>
<thead>
<tr>
<th>Lock ID</th>
<th>Owner Process</th>
<th>Next</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F_ACQLK</td>
<td>Process: 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Priority: 100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This call places process 2 in the queue. Process 2 must wait until process 1 releases the lock before it can become the owner of the process.

<table>
<thead>
<tr>
<th>Lock ID</th>
<th>Owner Process</th>
<th>Next</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>F_ACQLK</td>
<td>Process: 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Priority: 110</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With this call, the queue is re-ordered; process 3 is of a higher priority than process 2. Process 3 will be next to acquire the lock.

<table>
<thead>
<tr>
<th>Lock ID</th>
<th>Owner Process</th>
<th>Next</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>F_CALQLK</td>
<td>Process: 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Priority: 115</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This conditional acquire call has no effect on the lock structure; it is only performed if the lock is owned by another process. In this case, it returns error EOS_NOLOCK.

<table>
<thead>
<tr>
<th>Lock ID</th>
<th>Owner Process</th>
<th>Next</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>F_RELCLK</td>
<td>Process: 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Priority: 90</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This call releases process 1. The lock is now available to process 3. Process 2 moves up in line; it can acquire the lock after process 3 is released.
The following figure shows the locking sequence with one process and with multiple processes.

**Figure 5-3. Locking Sequence**

**Single Process**

<table>
<thead>
<tr>
<th>Lock ID</th>
<th>Owner Process</th>
<th>Next</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>proc ID 1</td>
<td>priority 100</td>
<td>next</td>
</tr>
</tbody>
</table>

**Multiple Process**

<table>
<thead>
<tr>
<th>Lock ID</th>
<th>Owner Process</th>
<th>Next</th>
<th>Previous</th>
</tr>
</thead>
<tbody>
<tr>
<td>proc ID 1</td>
<td>priority 100</td>
<td>next</td>
<td>previous</td>
</tr>
</tbody>
</table>

Owner Process

| proc ID 3 | priority 110 | next | previous |
| proc ID 2 | priority 110 | next | previous |

Process Waiting for Lock

Owner Process
This chapter explains how to use OS-9 system calls and contains an alphabetized list of all OS-9 system calls. It includes the following topics:

- Using OS-9 System Calls
- System Calls Reference
Using OS-9 System Calls

System calls are used to communicate between the OS-9 operating system and C or assembly language programs. There are four general categories of system calls:

- User-state system calls
- I/O system calls
- System-state system calls
- System-state I/O system calls

All of the OS-9 system calls require a single parameter to be passed to the operating system, called the parameter block. Parameter blocks are the means by which applications and system software pass parameters to the operating system for service requests. When a system call is performed, a pointer to the associated service request parameter block is passed to the operating system. The operating system acquires the specific parameters it needs for the service request from the parameter block and returns any result parameters through the parameter block.

Every system call parameter block contains the same substructure, `syscb`. `syscb` contains:

- An identifier of the service request
- The edition number of the service request interface
- The size of the associated parameter block
- A result field for returning error status

For programming convenience, a C language system call library containing a C interface for each of the OS-9 system calls is provided. A complete description of the C language interface for each of the system calls can be found in the *Ultra C Library Reference*.

__oscall Function__

There is a single routine located in the system call library providing the gateway into the operating system. The __oscall__ function expects a parameter block pointer and uses whatever trap or software interrupt facility is available on a given hardware platform to enter into the operating system.

The __oscall() request is a common interface to the kernel and the mechanism by which all OS-9 system calls are made. __oscall() has one parameter: the address of a parameter block or structure belonging to the system call. Each OS-9 system call binding creates a parameter block that is passed to the kernel by __oscall().
For example, the C binding for the \texttt{F\_FMOD} system call fills the parameter block and passes the address of the block directly to the kernel through \_oscall():

```c
#include "defsfile"

/* _os_fmod - find module directory entry service request. */
_os_fmod(type_lang, moddir_entry, mod_name)
  u_int16   *type_lang;
  Mod_dir   *moddir_entry;
  u_char    *mod_name;
{
  register error_code error;
  f_findmod_pb pb;       /* declare parameter block of appropriate type */

  pb.cb.code = F\_FMOD;  /* fill parameter block field; 
                         fn code defined in funcs.h */
  pb.cb.param_size = sizeof f\_findmod_pb; /* fill parameter block 
                                           field */
  pb.cb.edition = _OS\_EDITION; /* fill edition number */

  pb.type_lang = *type_lang; /* fill parameter block field */
  pb.mod_name = mod_name;  /* fill parameter block field */

  if ((error = _oscall(&pb)) == SUCCESS)  { /* make _oscall */
    *type_lang = pb.type_lang; /* return value */
    *moddir_entry = pb.moddir_entry; /* return value */
  }

  return error;
}
```

For more information about installing system calls, refer to the description of the \texttt{F\_SSVC}.

A complete list of structures for OS-9 system calls is included in \textit{Chapter 1, System Overview}. 

---

For example, the C binding for the F\_FMOD system call fills the parameter block and passes the address of the block directly to the kernel through \_oscall():

```c
#include "defsfile"

/* _os_fmod - find module directory entry service request. */
_os_fmod(type_lang, moddir_entry, mod_name)
  u_int16   *type_lang;
  Mod_dir   *moddir_entry;
  u_char    *mod_name;
{
  register error_code error;
  f_findmod_pb pb;       /* declare parameter block of appropriate type */

  pb.cb.code = F\_FMOD;  /* fill parameter block field; 
                         fn code defined in funcs.h */
  pb.cb.param_size = sizeof f\_findmod_pb; /* fill parameter block 
                                           field */
  pb.cb.edition = _OS\_EDITION; /* fill edition number */

  pb.type_lang = *type_lang; /* fill parameter block field */
  pb.mod_name = mod_name;  /* fill parameter block field */

  if ((error = _oscall(&pb)) == SUCCESS)  { /* make _oscall */
    *type_lang = pb.type_lang; /* return value */
    *moddir_entry = pb.moddir_entry; /* return value */
  }

  return error;
}
```

For more information about installing system calls, refer to the description of the F\_SSVC.

A complete list of structures for OS-9 system calls is included in \textit{Chapter 1, System Overview}. 

---

For example, the C binding for the F\_FMOD system call fills the parameter block and passes the address of the block directly to the kernel through \_oscall():

```c
#include "defsfile"

/* _os_fmod - find module directory entry service request. */
_os_fmod(type_lang, moddir_entry, mod_name)
  u_int16   *type_lang;
  Mod_dir   *moddir_entry;
  u_char    *mod_name;
{
  register error_code error;
  f_findmod_pb pb;       /* declare parameter block of appropriate type */

  pb.cb.code = F\_FMOD;  /* fill parameter block field; 
                         fn code defined in funcs.h */
  pb.cb.param_size = sizeof f\_findmod_pb; /* fill parameter block 
                                           field */
  pb.cb.edition = _OS\_EDITION; /* fill edition number */

  pb.type_lang = *type_lang; /* fill parameter block field */
  pb.mod_name = mod_name;  /* fill parameter block field */

  if ((error = _oscall(&pb)) == SUCCESS)  { /* make _oscall */
    *type_lang = pb.type_lang; /* return value */
    *moddir_entry = pb.moddir_entry; /* return value */
  }

  return error;
}
```
Chapter 6: OS-9 System Calls

Using the System Calls

The typical sequence for executing an OS-9 system call would be as follows:
1. Allocate a parameter block specific to the system call.
2. Initialize the parameter block including the system sub-block.
3. Call the operating system (through _oscall).
4. Check for errors upon return.
5. Process return parameters, if applicable.

All of the predefined parameter blocks for the OS-9 are located in the srvcb.h header file. Each system call description within this chapter includes a full description of the parameter block structure specific to the system call, as well as a full summary of the functionality of the system call.

System Call Descriptions

The OS-9 Attributes field indicates the state of each call, whether the call is an I/O call, and if the call can be used during an interrupt. The characteristic for each field (for example user, system, I/O, or interrupt) is listed where appropriate. In addition, the OS-9 Attributes table indicates whether a function is thread-safe or -unsafe.

System-state system calls are privileged. They may be executed only while OS-9 is in system state (for example, when it is processing another service request or executing a file manager or device driver). System-state functions are included in this manual primarily for the benefit of those programmers who write device drivers and other system-level applications.

Some system calls generate errors themselves; these are listed as Possible Errors. If the returned error code does not match any of the given possible errors, it was probably returned by another system call made by the main call. In the system call description section, strings passed as parameters are terminated by a null byte.

If you use the system calls from assembly language, do not alter registers.

Interrupt Context

If you use any system calls in an interrupt service routine that are not listed in the following table, you can corrupt the integrity of your system.
System Calls Reference

The following section describes the system calls in detail.
F_ABORT
Emulate Exception Occurrence

Headers
#include <regs.h>

Parameter Block Structure
typedef struct f_abort_pb {
    syscbcb;
    u_int32strap_code,
    address,
    except_id;
} f_abort_pb, *F_abort_pb;

Description
F_ABORT emulates the occurrence of an exception. This service request executes the
same recovery code in the OS used to recover from exceptions occurring in the
system. The OS responds to this service just as it would if the specified exception
had actually occurred. This allows applications or system extension modules to
force an exception condition without actually triggering the exception. An
application may use this service to test its exception handlers that were installed
using the F_STRAP service.

This service is used by some of the floating-point emulation extension modules on
processors lacking hardware floating-point support to trigger floating-point
exception conditions detected during software emulation of floating-point
instructions. The service emulates the floating-point exceptions that would have
occurred if the floating-point instructions had been executed by real hardware.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

strap_code
    The associated code used in the F_STRAP service request to setup an exception
handler. It is the F_STRAP code of the exception to emulate. The F_STRAP codes
are defined in the reg<CPU>.h header file for the target CPU platform.
address
The address of where the exception is to have occurred.

ext_id
The hardware vector identifier of the exception to emulate. The exception
vector identifiers are defined in the \texttt{reg\langle CPU\rangle.h} header file for the target CPU
platform.

\textbf{See Also}

\texttt{F\_STRAP}
F_ACQLK
Acquire Ownership of Resource Lock

Headers
#include <lock.h>
#include <types.h>

Parameter Block Structure
typedef struct f_acqlk_pb {
    syscb cb;
    lock_id lid;
    signal_code signal;
} f_acqlk_pb, *F_acqlk_pb;

Description
F_ACQLK acquires ownership of a resource lock (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 system call without error. If, during the course of waiting on a lock, a process receives a signal, the process is activated without gaining ownership of the lock. The process returns from the system 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 will be zero.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

    cb
    The control block header.

    lid
    The lock identifier of the lock you are attempting to acquire.

    signal
    The signal prematurely terminating the acquisition of the lock.
Possible Errors

EOS_SIGNAL

See Also

F_CAQLK
F_CRLK
F_DELLK
F_RELLK
F_WAITLK
Headers

#include <types.h>

Parameter Block Structure

typedef struct f_salarm_pb {
    syscb cb;
    alarm_id alrm_id;
    u_int16 alrm_code;
    u_int32 time,
        flags;
    u_int32 (*function)();
    void *func_pb;
} f_salarm_pb, *F_salarm_pb;

Description

The system-state alarm requests execute a system-state subroutine at a specified time. They are provided for functions such as turning off a disk drive motor if the disk is not accessed for a period of time.

System-state alarms, as well as user-state alarms, always belong to some process. This process, for system-state alarms, is either the creating process (if the TH_SPOWN bit was 0 when the process had the operating system create the alarm) or the system process (if the TH_SPOWN bit was 1 when the process had the operating system create the alarm). For user-state alarms, they always belong to the creating process and never the system process. If a process gives ownership of an alarm to the system process, then the alarm remains in the system until either it expires, or some system-state process deletes it. In all other respects, system-state alarms behave as user-state alarms.

The time interval is the number of system clock ticks (or 1/256 second) to wait before an alarm signal is sent. If the high order bit is set, the low 31 bits are interpreted as 1/256 second. All times are rounded up to the nearest tick.

The alarm functions do not return any error code if the operating system cannot wait for the requested time due to an overflow when converting a time from 256ths-of-a-second into clock ticks. This only occurs if you specify a time in 256ths-of-a-second and the system clock ticks occur at a rate greater than 512 ticks-per-second. If an overflow occurs, the operating system waits for the longest delay possible.
The following system-state alarm functions are supported:

Table 6-1.

<table>
<thead>
<tr>
<th>Alarm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ALARM, A_ATIME</td>
<td>Executes a subroutine at a specified time.</td>
</tr>
<tr>
<td>F_ALARM, A_CYCLE</td>
<td>Executes a subroutine at specified time intervals.</td>
</tr>
<tr>
<td>F_ALARM, A_DELET</td>
<td>Removes a pending alarm request.</td>
</tr>
<tr>
<td>F_ALARM, A_RESET</td>
<td>Resets an existing alarm request.</td>
</tr>
<tr>
<td>F_ALARM, A_SET</td>
<td>Executes a subroutine after a specified time interval.</td>
</tr>
</tbody>
</table>

During an A_RESET request, the TH_SPOWN bit has the following meaning: if 0, allow the calling process to update only its own alarms; if 1, allow the calling process to update any alarm.

During an A_DELETE request, the TH_SPOWN bit has the following meaning: if 0, allow the calling process to delete only its own alarms; if 1, allow the calling process to delete any alarm. If the alarm_id field is 0 and the TH_SPOWN bit is 1, the operating system deletes all alarms belonging to the system process.

System-state alarms are run by the system process. They should not perform any function resulting in any kind of queuing, such as F_SLEEP; F_WAIT; F_LOAD; and F_EVENT, EV_WAIT. When such functions are required, the caller must provide a separate process to perform the function, rather than an alarm.

IRQ routines cannot create or delete alarms since such actions cause memory allocations/deallocations, that are illegal from an IRQ routine. The way to handle such things is to create the alarms before the IRQ routine needs them, and then have the IRQ routine use only Resets, that are legal in IRQ routines.

For non-system, process-owned alarms, the user number in the system process descriptor changes temporarily to the user number of the original process.

If an alarm execution routine suffers any kind of bus trap, address trap, or other hardware-related error, the system crashes.

Attributes

Operating System: OS-9
State: System
Threads: Safe

Parameters

cb
The control block header.

alarm_id
The alarm identifier returned by the system call. The alarm ID may subsequently be used to delete the alarm, if desired, by using the F_ALARM, A_DELET alarm call.
alarm_code
   The particular alarm function to perform.

time
   The specified time.

flags
   One of the following two alarm flags defined in <process.h>:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TH_DELPB</td>
<td>0x00000001</td>
<td>Indicates the associated function parameter block’s memory should be returned to the system after executing the alarm function.</td>
</tr>
<tr>
<td>TH_SPOWN</td>
<td>0x00000002</td>
<td>Indicates the system-state alarm should be owned by the system process and not the current process.</td>
</tr>
</tbody>
</table>

function
   The function to be executed.

cfunc_pb
   Points to the function’s parameters block.

**Possible Errors**

EOS_NOCLK
EOS_NORAM
EOS_PARAM
EOS_UNKSVC

**See Also**

F_ALARM (User-State)
F_EVENT, EV_WAIT
F_LOAD
F_SLEEP
F_WAIT
Chapter 6: OS-9 System Calls

F_ALARM (User-State)
User-State Set Alarm Clock

Headers

```c
#include <types.h>
```

Parameter Block Structure

```c
typedef struct f_alarm_pb {
    syscb cb;
    alarm_id alrm_id;
    u_int16 alrm_code;
    u_int32 time;
    signal_code signal;
} f_alarm_pb, *F_alarm_pb;
```

Description

The user-state alarm requests enable a user process to create an asynchronous software alarm clock timer. The timer sends a signal to the calling process when the specified time period has elapsed. A process may have multiple alarm requests pending.

The time interval is the number of system clock ticks (or 1/256 second) to wait before an alarm signal is sent. If the high order bit is set, the low 31 bits are interpreted as 1/256 second.

⚠️ All times are rounded up to the nearest system clock tick.

The alarm functions do not return any error code if the operating system cannot wait for the requested time due to an overflow when converting a time from 256ths-of-a-second into clock ticks. This only occurs if you specify a time in 256ths-of-a-second and the system clock ticks occur at a rate greater than 512 ticks-per-second. If an overflow occurs, the operating system waits for the longest delay possible.

The following user-state alarm functions are supported:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_ALARM, A_ATIME</td>
<td>Send signal at specified time.</td>
</tr>
<tr>
<td>F_ALARM, A_CYCLE</td>
<td>Send signal at specified time intervals.</td>
</tr>
<tr>
<td>F_ALARM, A_DELET</td>
<td>Remove pending alarm request.</td>
</tr>
<tr>
<td>F_ALARM, A_RESET</td>
<td>Reset existing alarm request to occur at a newly specified time.</td>
</tr>
<tr>
<td>F_ALARM, A_SET</td>
<td>Send signal after specified time interval.</td>
</tr>
</tbody>
</table>
Attributes
Operating System: OS-9
State: User
Threads: Safe

Parameters

`cb`
   The control block header.

`alrm_id`
   The alarm identifier returned by the system call. The alarm ID may subsequently be used to delete the alarm, if desired, by using the `F_ALARM`, `A_DELETE` alarm call.

`alrm_code`
   The particular alarm function to perform.

`time`
   The specified time.

`signal`
   The signal value originally belonging to the alarm.

Possible Errors
EOS_BPADDR
EOS_NORAM
EOS_PARAM
EOS_UNKSVC

See Also
F_ALARM (System-State)
F_ALARM, A_ATIME
Send Signal At Specified Time (User-State)
Execute Subroutine At Specified Time (System-State)

Headers
#include <types.h>

Parameter Block Structure
If OS-9 is in system state, see F_ALARM (System-State) for the parameter block structure. Otherwise, see F_ALARM (User-State) for the parameter block structure.

Description
A_ATIME sends one signal at the specified time in user state or executes a subroutine at the specified time in system state.

Attributes
Operating System:   OS-9
State: User and System
Threads: Safe

Parameters

alarm_id
The alarm identifier returned by the system call. The alarm ID may subsequently be used to delete the alarm, if desired, by using the F_ALARM, A_DELET alarm call.

signal
The signal code of the signal to send.

time
The specified time. The value is considered to be an absolute value in seconds since 1 January 1970 Greenwich Mean Time.

Possible Errors
EOS_NOCLK
EOS_NORAM
EOS_PARAM

See Also
F_ALARM, A_SET
F_ALARM (System-State)
F_ALARM (User-State)
Chapter 6: OS-9 System Calls

F_ALARM, A_CYCLE
Send Signal at Specified Time Intervals

Headers
#include <types.h>

Parameter Block Structure
If OS-9 is in system state, see F_ALARM (System-State) for the parameter block structure. Otherwise, see F_ALARM (User-State) for the parameter block structure.

Description
A_CYCLE sends a signal after the specified time interval has elapsed and then resets the alarm. This provides a recurring periodic signal.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    alrm_id
        The returned alarm ID.

    alrm_code
        The particular alarm function to perform (in this case, A_CYCLE).

    signal
        The signal code of the signal to send.

    time
        Specify the time interval between signals. 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 1/256 second. The minimum time interval allowed is two system clock ticks.

Possible Errors
EOS_NOCLK
EOS_NORAM
EOS_PARAM

See Also
F_ALARM, A_SET
F_ALARM (System-State)
F_ALARM (User-State)
Headers
#include <types.h>

Parameter Block Structure
If OS-9 is in system-state, see F_ALARM (System-State) for the parameter block structure. Otherwise, see F_ALARM (User-State).

Description
A_DELET removes a cyclic alarm, or any alarm that has not expired.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
alrm_id
Specify the alarm identification number. If alrm_id is zero, all pending alarm requests are removed.

Possible Errors
EOS_BPADDR
EOS_IBA
EOS_NORAM
EOS_PARAM

See Also
F_ALARM, A_SET
F_ALARM (System-State)
F_ALARM (User-State)
F_ALARM, A_RESET
Reset Existing Alarm Request

Headers
#include <types.h>

Parameter Block Structure
If OS-9 is in system state, see F_ALARM (System-State) for the parameter block structure. Otherwise, see F_ALARM (User-State) for the parameter block structure.

Description
A_RESET resets an existing alarm to occur at the newly specified time. It does not reset any other characteristics of the original alarm.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

alarm_id
The ID of the alarm to reset.

signal
The signal code of the signal to send.

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 1/256 second. The minimum time interval allowed is two clock ticks.

Possible Errors
EOS_NOCLK
EOS_NORAM
EOS_PARAM

See Also
F_ALARM, A_SET
F_ALARM (System-State)
F_ALARM (User-State)
F_ALARM, A_SET
Send Signal After Specified Time Interval

Headers
#include <types.h>

Parameter Block Structure
If OS-9 is in system state, see \texttt{F\_ALARM (System-State)} for the parameter block structure. Otherwise, see \texttt{F\_ALARM (User-State)} for the parameter block structure.

Description
\texttt{A\_SET} sends one signal after the specified time interval has elapsed.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
\texttt{alrm\_id}
The alarm identifier returned by the system call. The alarm ID can subsequently be used to delete the alarm, if desired, by using the \texttt{A\_DELET} alarm call.

\texttt{signal}
The signal code of the signal to send.

\texttt{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 1/256 second. The minimum time interval allowed is two system clock ticks.

Possible Errors
EOS\_BPADDR
EOS\_NORAM
EOS\_PARAM

See Also
\texttt{F\_ALARM, A\_DELET}
\texttt{F\_ALARM (System-State)}
\texttt{F\_ALARM (User-State)}
Chapter 6: OS-9 System Calls

F_ALLPRC
Allocate Process Descriptor

Headers
#include <process.h>

Parameter Block Structure
typedef struct f_allprc_pb {
    syscb cb;
    process_id proc_id;
    Pr_desc proc_desc;
}  f_allprc_pb, *F_allprc_pb;

Description
F_ALLPRC allocates and initializes a process descriptor. The address of the descriptor
is stored in the process descriptor table. Initialization consists of clearing the
descriptor and setting its process identifier.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

    cb
         The control block header.

    proc_id
         A returned value. It is the process ID of the new process descriptor.

    proc_desc
         A returned value. It points to the new process descriptor.

Possible Errors

EOS_PRCFUL
F_ALLTSK
Allocate Task

Headers
#include <process.h>

Parameter Block Structure
typedef struct f_alltsk_pb{
    syscb cb;
    Pr_desc proc_desc;
} f_alltsk_pb, *F_alltsk_pb;

Description
F_ALLTSK is called just before a process becomes active to ensure the protection hardware is ready for the process. F_ALLTSK sets the protection hardware to the map for the process pointed to by proc_desc.

F_ALLTSK is only supported on systems with a memory protection unit (for example, all 80x86). The SSM module must be present in the bootfile.

If the SSM module is not present in the system, an EOS_UNKSVC error is returned. You should ignore this error.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

cb
   The control block header.

proc_desc
   Point to the process descriptor.

Possible Errors
EOS_UNKSVC

See Also
F_DELTSK
Chapter 6: OS-9 System Calls

F_ALTMDIR
Set Alternate Working Module Directory

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_altmdir_pb {
    syscb cb;
    u_char *name;
} f_altmdir_pb, *F_altmdir_pb;

Description
F_ALTMDIR establishes an alternate working module directory for a process.
When a process performs an F_LINK or F_FORK system call, the search for the
specified target module begins in the process’ current module directory. If that
search fails, the alternate module directory is searched. This enables processes to
link to or execute modules from different locations within system memory.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
        The control block header.

    name
        Point to the name of the alternate working module directory.

Possible Errors
EOS_MNF
EOS_PERMIT

See Also
F_CHMDIR
F_FORK
F_LINK
Headers

#include <types.h>

Parameter Block Structure

typedef struct f_aproc_pb {
    syscb cb;
    process_id proc_id;
}  f_aproc_pb, *F_aproc_pb;

Description

F_APROC inserts a process into the active process queue so it may be scheduled for execution.

All processes already in the active process queue are aged. The age of the new process is set to its priority, and the process is inserted according to its relative age. If the new process has a higher priority than the currently active process, the active process gives up the remainder of its time slice and the new process runs immediately.

OS-9 does not preempt a process in system state (for example, the middle of a system call). However, OS-9 does set a bit (TIMOUT in p_state) in the process descriptor causing the process to surrender its time slice when it re-enters user state.

Attributes

Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

cb
    The control block header.

proc_id
    Specify the ID of the process to place in the active process queue.

Possible Errors

EOS_IPRCID
EOS_PERMIT

See Also

F_NPROC
F_CAQLK
Conditionally Acquire Ownership of Resource Lock

Headers
#include <lock.h>

Parameter Block Structure
typedef struct f_caqlk_pb {
    syscb cb;
    lock_id lid;
} f_caqlk_pb, *F_caqlk_pb;

Description
F_CAQLK conditionally acquires ownership of a resource lock.
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 not suspended. Instead, it returns
from the F_CAQLK call with an EOS_NOLOCK error and is not granted ownership of the
resource lock.

Refer to Chapter 6 for more information on locks.

Attributes
Operating System: OS-9
State: System and Interrupt
Threads: Safe

Parameters

cb
The control block header.

lid
The identifier of the lock you are attempting to acquire.

Possible Errors
EOS_NOLOCK

See Also
F_ACQLK
F_CRLK
F_DELLK
F_RELLK
F_WAITLK
F_CCTL (User-State)
User-State Cache Control

Headers

```c
#include <types.h>
#include <cache.h>
```

Parameter Block Structure

```c
typedef struct f_cache_pb {
    syscb cb;
    u_int32 control;
    void *addr;
    u_int32 size;
} f_cache_pb, *F_cache_pb;
```

Description

F_CCTL performs operations on the system instruction and/or data caches, if there are any.

If the C_ADDR bit of the control parameter is set, then the addr and size parameters are used to flush the specific target address from the cache. This functionality is only supported on hardware platforms with this capability.

Only system-state processes and super-group processes can perform the other precise operations of F_CCTL.

Any program that builds or changes executable code in memory should flush the instruction cache with F_CCTL before executing the new code. This is necessary because the hardware instruction cache may not be updated by data (write) accesses on certain hardware set ups and may therefore contain the unchanged instruction(s). For example, if a subroutine builds a system call on its stack, it should first use the F_CCTL system to flush the instruction cache before it executes the temporary instructions.

Attributes

- Operating System: OS-9
- State: User
- Threads: Safe

Parameters

- cb
  The control block header.
**control**

Specify the cache operation. If `control` is zero, the system instruction and data caches are flushed. Only super-group processes can perform this operation. Only three bits may be used:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 2</td>
<td>C_FLDATA</td>
<td>Flush data cache</td>
</tr>
<tr>
<td>Bit 6</td>
<td>C_FLINST</td>
<td>Flush instruction cache</td>
</tr>
<tr>
<td>Bit 8</td>
<td>C_ADDR</td>
<td>Indicates a specific target address for flush operation</td>
</tr>
</tbody>
</table>

**addr**

Specify the target address for the flush operation.

**size**

Indicate the size of the target memory area to be flushed.

**Possible Errors**

EOS_PARAM
Chapter 6: OS-9 System Calls

F_CCTL (System-State)
System-State Cache Control

Headers
#include <types.h>
#include <cache.h>

Parameter Block Structure

typedef struct f_scache_pb {
    syscb cb;
    u_int32 control;
    u_int32 (*cctl)();
    void *cctl_data;
    void *addr;
    u_int32 size;
} f_scache_pb, *F_scache_pb;

Description

F_CCTL performs operations on the system instruction and/or data caches, if there are any.

Any program that builds or changes executable code in memory should flush the instruction cache by F_CCTL prior to 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, the F_CCTL system call to flush the instruction cache must be executed prior to executing the temporary instructions.

If the C_GETCCTL bit of control is set, F_CCTL 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 F_CCTL request.

⚠️ The OS-9 kernel calls the cache extension module directly.

Attributes

Operating System: OS-9
State: System and Interrupt
Threads: Safe
Chapter 6: OS-9 System Calls

Parameters

**cb**
The control block header.

**control**
Specify the cache operation. If control is zero, the system instruction and data caches are flushed. The following bits are defined in the control parameter for precise operation:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C_ENDATA</td>
<td>If set, enables data cache.</td>
</tr>
<tr>
<td>1</td>
<td>C_DISDATA</td>
<td>If set, disables data cache.</td>
</tr>
<tr>
<td>2</td>
<td>C_FLDATA</td>
<td>If set, flushes data cache.</td>
</tr>
<tr>
<td>3</td>
<td>C_INVDATA</td>
<td>If set, invalidates data cache.</td>
</tr>
<tr>
<td>4</td>
<td>C_ENINST</td>
<td>If set, enables instruction cache.</td>
</tr>
<tr>
<td>5</td>
<td>C_DISINST</td>
<td>If set, disables instruction cache.</td>
</tr>
<tr>
<td>6</td>
<td>C_FLINST</td>
<td>If set, flushes instruction cache.</td>
</tr>
<tr>
<td>7</td>
<td>C_INVINST</td>
<td>If set, invalidates instruction cache.</td>
</tr>
<tr>
<td>8</td>
<td>C_ADDR</td>
<td>Flags a target address for flush operation.</td>
</tr>
<tr>
<td>9-14</td>
<td>Reserved</td>
<td>Reserved for future use by RadiSys.</td>
</tr>
<tr>
<td>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>16</td>
<td>C_STODATA</td>
<td>If set, stores data cache (if supported by hardware).</td>
</tr>
<tr>
<td>17-31</td>
<td>Reserved</td>
<td>Reserved for future use by RadiSys.</td>
</tr>
</tbody>
</table>

All other bits are reserved. If any reserved bit is set, an EOS_PARAM error is returned. Precise operation of F_CCTL can only be performed by system-state processes and super-group processes.

If the C_ADDR bit of the control parameter is set, then the addr and size parameters are used to flush the specific target address from the cache. This functionality is only supported on hardware platforms with this capability.

**cctl**
The returned cache routine.

**cctl_data**
The returned cache routine’s static data.

**addr**
Specify the target address for the flush operation.

**size**
Indicate the size of the target memory area to be flushed.

Possible Errors

EOS_PARAM
Chapter 6: OS-9 System Calls

F_CHAIN
Load and Execute New Primary Module

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_chain_pb {
    syscb cb;
    u_int16 priority,
        path_cnt;
    u_char *mod_name,
        *params;
    u_int32 mem_size,
        param_size;
    u_int16 type_lang;
} f_chain_pb, *F_chain_pb;

Description

F_CHAIN executes a new program without the overhead of creating a new process. It is functionally similar to a F_FORK command followed by an F_EXIT. F_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.

F_CHAIN executes as follows:
1. The process’ old primary module is unlinked.
2. The system parses the name string of the new process’ primary module (the program that is executed). Next, the current and alternate module directories are searched to see if a module with the same name and type/language is already in memory. If so, the module is linked. If not, the name string is used as the pathlist of a file to be loaded into memory. The first module in this file is linked.
3. The data memory area is reconfigured to the size specified in the new primary module’s header.
4. Intercepts and pending signals are erased.
5. The following structure definition is passed to the initial function of the new module (this is identical to \texttt{F\_FORK}).

```c
typedef struct {
    process_id proc_id; /* process ID */
    owner_id owner;   /* group/user ID */
    prior_level priority; /* priority */
    u_int16 path_count; /* of I/O paths inherited*/
    u_int32 param_size, /* size of parameters */
              mem_size; /* total initial memory allocated*/
    u_char *params,  /* parameter pointer */
            *mem_end; /* top of memory pointer */
    Mh_com mod_head; /* primary (forked) module ptr*/
} fork_params, *Fork_params;
```

The minimum overall data area size is 256 bytes.

\texttt{F\_CHAIN} never returns to the calling process. If an error occurs during the Chain, it is returned as an exit status to the parent of the process performing the Chain.

**Attributes**

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe

**Parameters**

- **cb**
  - The control block header.

- **priority**
  - The initial priority of the process.

- **path_cnt**
  - Specify the number of I/O paths to copy (inherit).

- **mod_name**
  - Point to the new program to execute.

- **params**
  - Point to the parameter string to pass to the new process.

- **mem_size**
  - Specify the additional memory size above the default specified in the primary module's module header.

- **param_size**
  - Specify the size of the parameter string.

- **type_lang**
  - Specify the desired module type/language. \texttt{type_lang} must be either program/object or zero (for any).
Possible Errors
EOS_NEMOD

See Also
F_CHAINM
F_FORK
F_FORKM
F_LOAD
Chapter 6: OS-9 System Calls

F_CHAINM
Execute New Primary Module Given Pointer to Module

Headers
#include <module.h>

Parameter Block Structure
typedef struct f_chainm_pb {
    syscb cb;
    u_int16 priority,
    path_cnt;
    Mh_com mod_head;
    u_char *params;
    u_int32 mem_size,
    param_size;
} f_chainm_pb, *F_chainm_pb;

Description
F_CHAINM executes a new program without the overhead of creating a new process. It is functionally similar to a F_FORK command followed by an F_EXIT. F_CHAINM resets the calling process’ program and data memory areas and begins executing a new primary module. Open paths are not closed or otherwise affected.

F_CHAINM is similar to F_CHAIN. However, F_CHAINM is passed a pointer to the module instead of the module name.

F_CHAINM executes as follows:
1. The process’ old primary module is unlinked.
2. The system tries to link to the module pointed to by the module header pointer.
3. The data memory area is reconfigured to the specified size in the new primary module’s header.
4. Intercepts and pending signals are erased.
5. The following structure definition is passed to the initial function of the new module (this is identical to \texttt{F\_FORK}).

```c
typedef struct {
    process_id proc_id;    /* process ID */
    owner_id   owner;      /* group/user ID */
    prior_level priority;   /* priority */
    u_int16    path_count; /* number of I/O paths inherited */
    u_int32    param_size, /* size of parameters */
                mem_size;   /* total initial memory allocated */
    u_char     *params,    /* parameter pointer */
                *mem_end;   /* top of memory pointer */
    Mh_com     mod_head; /* primary (forked) module ptr*/
} fork_params, *Fork_params;
```

The minimum overall data area size is 256 bytes.

An error is returned only if there is not enough memory to hold the parameters. If an error occurs during the \texttt{Chains}, it is returned as an exit status to the parent of the process performing the \texttt{Chains}.

**Attributes**

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe

**Parameters**

- \texttt{cb}
  
  The control block header.

- \texttt{priority}
  
  The initial priority of the process.

- \texttt{pathCnt}
  
  The number of I/O paths to copy (inherit).

- \texttt{modHead}
  
  Point to the module header.

- \texttt{params}
  
  Point to the parameter string to pass to the new process.

- \texttt{memSize}
  
  Specify the additional memory size above the default specified in the primary module’s module header.

- \texttt{paramSize}
  
  Specify the size of the parameter string.
Possible Errors
EOS_CRC

See Also
F_CHAIN
F_FORK
F_FORKM
F_LOAD
Chapter 6: OS-9 System Calls

F_CHKMEM
Check Memory Block’s Accessibility

Headers
#include <process.h>

Parameter Block Structure
typedef struct f_chkmem_pb {
  syscb cb;
  u_int32 size;
  u_int16 mode;
  u_char *mem_ptr;
  Pr_desc proc_desc;
} f_chkmem_pb, *F_chkmem_pb;

Description:
F_CHKMEM is called by system routines before accessing data at the specified address on behalf of a process to determine if the process has access to the specified memory block.

F_CHKMEM must check the process’ protection image to determine if access to the specified memory area is permitted. F_CHKMEM is called by system-state routines that can access memory (such as I_READ and I_WRITE) to determine if the user process has access to the requested memory. This software check is necessary because the protection hardware is expected to be disabled for system-state routines.

- The calling process cannot use this service to check for write-only memory because it assumes read-only as the minimum. To check for no-access to a segment of memory, the calling process can check for read access and ensure the resulting status code is EOS_BPADDR. To check for read-only access, there must be two calls to F_CHKMEM.
- F_CHKMEM is only useful on systems with an MMU and having the SSM module in their bootfile. When SSM is active, the operating system validates all arguments. On systems without SSM, the call always returns successful because every process has full access rights to the entire memory space.

Attributes
Operating System: OS-9
State: System
Threads: Safe
Parameters

cb
The control block header.

size
Specify the size of the memory area.

mode
Specify the permissions to check.

mem_ptr
Point to the beginning of the memory to check.

proc_desc
Point to the process descriptor of the target process.

Possible Errors
EOS_BPADDR
EOS_UNKSVC (from user-state, with or without SSM)

See Also
F_ALLTSK
F_DELTSK
F_PERMIT
F_PROTECT
I_READ
I_WRITE
F_CHMDIR
Change Process’ Current Module Directory

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_chmdir_pb {
    syscb cb;
    u_char *name;
} f_chmdir_pb, *F_chmdir_pb;

Description
F_CHMDIR changes a process’ current module directory.
The calling process must have access permission to the specified module directory
or an EOS_PERMIT error is returned.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

name
    Point to the new current module directory. name can be a full pathlist or
    relative to the current module directory. To change to the system’s root
    module directory, specify a slash (/) for name.

Possible Errors
EOS_BNAM
EOS_MNF
EOS_PERMIT

See Also
F_MKMDIR
**F_CLRSIGS**  
Clear Process Signal Queue

### Headers
```
#include <types.h>
```

### Parameter Block Structure
```
typedef struct f_clrsigs_pb {
    syscb cb;
    process_id proc_id;
} f_clrsigs_pb, *F_clrsigs_pb;
```

### Description
F_CLRSIGS removes any pending signals sent to the target process.

### Attributes
- **Operating System:** OS-9
- **State:** User, System, and Interrupt
- **Threads:** Safe

### Parameters
- **cb**
  - The control block header.
- **proc_id**
  - Identify the target process.

### Possible Errors
- **EOS_IPRCID**

### See Also
- **F_SIGMASK**
F_CMDPERM
Change Permissions of Module Directory

Headers
#include <module.h>

Parameter Block Structure
typedef struct f_cmdperm_pb {
    syscb cb;
    u_char *name;
    u_int16 perm;
} f_cmdperm_pb, *F_cmdperm_pb;

Description
F_CMDPERM changes the access permissions of an existing module directory. This makes it possible to restrict access to a particular module directory.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
cb
    The control block header.

name
    Point to the name of the existing module directory.

perm
    Specify the new permissions.

Possible Errors
EOS_BNAM
EOS_MNF
EOS_PERMIT

See Also
F_MKMDIR
Chapter 6: OS-9 System Calls

F_CMPNAM
Compare Two Names

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_cmpnam_pb {
sycinb cb;
    u_int32 length;
    u_char *string,
        *pattern;
    int32 result;
} f_cmpnam_pb, *F_cmpnam_pb;

Description
F_CMPNAM compares a target name to a source pattern to determine if they are equal. F_CMPNAM is not case-sensitive; it does not differentiate between upper and lower case characters.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
cb
    The control block header.

length
    Specify the length of the pattern string.

string
    Point to the target name string. The target name must be terminated by a null byte.

pattern
    Point to the pattern string. Two wildcard characters are recognized in the pattern string:
    • A question mark (?) matches any single character.
    • An asterisk (*) matches any string.
result
A returned value. It is the lexicographic result of the comparison.
- If result is zero, the target string is the same as the pattern string.
- If result is negative, the target name is greater than the pattern string.
- If result is positive, the target string is less than the pattern string.

Possible Errors
EOS_DIFFER
EOS_STKOVF
F_CONFIG
Configure an Element

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_config_pb {
    syscb cb;
    u_int32 code;
    void *param;
} f_config_pb, *F_config_pb;

Description
F_CONFIG is a wildcard call that configures elements of the operating system that may or may not be process specific. It reconfigures operating system resources at runtime. The target resources may be system-wide resources or process-specific, depending on the nature of the call.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
        The control block header.

    code
        Identify the target configuration code. Currently, no sub-codes are defined for this call.

*param
    Point to any additional parameters required by the specified configuration function.

See Also
I_CONFIG
F_CPYMEM
Copy External Memory

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_cpymem_pb {
    syscb cb;
    process_id proc_id;
    u_char *from,
        *to;
    u_int32 count;
} f_cpymem_pb, *F_cpymem_pb;

Description
F_CPYMEM uses F_MOVE to copy data from one location to another and (at present) ignores the proc_id argument (refer to the Parameters section below). The difference between F_MOVE and F_CPYMEM is the OS allows only system-state processes to use the former, while the OS allows either user- or system-state processes to use the later.

For system-state processes, the only difference between these two services is F_CPYMEM is slightly slower, since it has more routines to call before transferring control to F_MOVE.

For user-state processes, F_CPYMEM is the only choice for copying restricted memory. The OS (if the SSM is active) calls F_CHKMEM to ensure the caller has read and write access to the output. The OS allows the input address to be any existent location of memory (it allows user-state processes to copy even restricted data if it exists in RAM).

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe
Parameters

cb
The control block header.

proc_id
Specify the process ID of the owner of the external memory.

This service does not currently use the proc_id input, which was valid when OS-9 was running on the MC6809 architecture. To allow memory access beyond 64KB, OS-9 used F_CPYMEM to do bank switching in order to allow a process to copy data from a different bank of memory. The proc_id argument was nothing more than a bank selector. At this point there is no need for the proc_id argument, but it is reserved for future use.

from
The address of the external process’ memory to copy.

to
Point to the caller’s destination buffer.

count
The number of bytes to copy.

Possible Errors
EOS_BPADDR

See Also
F_MOVE
Headers

#include <types.h>

Parameter Block Structure

typedef struct f_crc_pb {
    syscb cb;
    u_char *start;
    u_int32 count,
    accum;
} f_crc_pb, *F_crc_pb;

Description

F_CRC generates or checks the CRC (cyclic redundancy check) values of sections of memory. Compilers, assemblers, and other module generators use F_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 0xffffffff before the first F_CRC call for any particular module.

To generate the CRC of an existing module, the calculation should 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 complete the following steps:

Step 1. Initialize the accumulator to -1.
Step 2. Perform the CRC over the module.
Step 3. Call F_CRC with a NULL value for start.
Step 4. Complement the CRC accumulator.
Step 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, include the byte preceding the CRC in the check and initialize this byte to zero. If a data pointer of zero is passed, the CRC is updated with one zero data byte. F_CRC always returns 0xff in the most significant byte of the accum parameter; accum can be stored (after complement) in the last four bytes of a module as the correct CRC.
Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
  The control block header.

start
  Point to the data.

count
  Specify the byte count for the data.

accum
  A returned value. It points to the CRC accumulator.

See Also

F_SETCRC
F_CRLK
Create New Resource Lock Descriptor

Headers
#include <lock.h>

Parameter Block Structure
typedef struct f_crlk_pb {
    syscb cb;
    lock_id lid;
} f_crlk_pb, *F_crlk_pb;

Description
F_CRLK creates a new resource lock descriptor. A resource lock descriptor is
allocated and initialized to a free state (not currently owned). Locks can be used to
protect resources in a multi-tasking environment. They provide a mechanism for
exclusive access to a given resource.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters
cb
    The control block header.
lid
    A returned value. It is the lock identifier for the lock descriptor. lid is used as
    a handle to perform further operations on the lock.

Possible Errors
EOS_NORAM

See Also
F_ACQLK
F_CAQLK
F_DELLK
F_RELLK
F_WAITLK
Chapter 6: OS-9 System Calls

F_DATMOD
Create Data Module

Headers

#include <module.h>

Parameter Block Structure

typedef struct f_datmod_pb {
    syscb cb;
    u_char *mod_name;
    u_int32 size;
    u_int16 attr_rev,
        type_lang,
        perm;
    void *mod_exec;
    Mh_com mod_head;
    u_int32 color
} f_datmod_pb, *F_datmod_pb;

Description

F_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 current module directory. Several processes can communicate with each other using a shared data module.

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 and CRC, and is rounded up to the nearest system memory allocation boundary.

F_DATMOD does not create a CRC value for the data module. If you load the data module into memory, you must first create the CRC value.

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Parameters

mod_name
   Point to the module name string.

size
   The size of the data portion.

attr_rev
   A returned value. The value of the module’s attribute and revision.

type_lang
   A returned value. The value of the module’s type and language.

perms
   Specify the access permissions for the module.

mod_exec
   A returned value. It points to the module data.

mod_head
   A returned value. It points to the module header.

color
   Memory color type. If color is zero, MEM_ANY is the memory type.

Possible Errors

EOS_BNAM
EOS_KWNMOD

See Also

F_SETCRC
F_DATTACH
Attach Debugger to a Running Process

Headers
#include <regs.h>

Parameter Block Structure
typedef struct f_dattach_pb {
    syscb cb;
    process_id proc_id;
    Regs reg_stack;
    Fregs freg_stack;
} f_dattach_pb, *F_dattach_pb;

Description
F_DATTACH attaches the calling debugger to an active process, enabling the debugger to assume debug control over the existing process. It establishes a debug session in the same way F_DFORK starts a new process for debug execution. Once a debugger performs the debug attach operation, the target process is suspended from execution and the debugger can then proceed to execute the target process under its control using the F_DEXEC service request. One important difference between F_DATTACH and F_DFORK is with the F_DATTACH call, the target process continues normal execution when the parent debugging process exits. The debug resources of the target process are released but the process does not terminate. However, when a process is started with the F_DFORK service request, the process is terminated when the parent debugger performs the F_DEXIT service request.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

cb
    The control block header.

proc_id
    The process identifier of the target process to attach to for debugging.

reg_stack
    Point to a register image buffer in the caller's data area where the kernel returns the current register image of the target debug process.
freg_stack
  Point to a floating-point register image buffer in the caller's data area where
  the kernel returns the current floating-point register image of the target debug
  process. Note, this floating-point image can contain an empty image since the
  target process may not be using the floating-point facilities of the system.

Possible Errors
EOS_IPRCID
EOS_PERMIT

See Also
F_DEXEC
F_DEXIT
F_DFORK
F_DDLK
Check for Deadlock Situation

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_ddlk_pb {
    syscb cb;
    process_id proc_id;
} f_ddlk_pb, *F_ddlk_pb;

Description
F_DDLK checks for a deadlock situation between processes. A search for the current process (calling process) in the linked list of potential conflicting processes is begun from the process specified by proc_id.

F_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.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

cb
    The control block header.

proc_id
    Specify the process with which to begin the search.

    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.

Possible Errors
EOS_DEADLK
F_DELLK
Delete Existing Lock Descriptor

Headers
#include <lock.h>

Parameter Block Structure
typedef struct f_dellk_pb {
    syscb cb;
    lock_id lid;
} f_dellk_pb, *F_dellk_pb;

Description
F_DELLK deletes an existing lock descriptor.
F_DELLK 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 prior to deleting it. You can corrupt the system if you do not release suspended processes prior to deletion.

Refer to Chapter 6 for more information about locks.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters
cb
    The control block header.

lid
    The lock identifier for the lock to delete.

See Also
F_ACQLK
F_CAQLK
F_CRLK
F_RELLK
F_WAITLK
F_DELMDIR
Delete Existing Module Directory

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_delmdir_pb {
    syscb cb;
    u_char *name;
} f_delmdir_pb, *F_delmdir_pb;

Description
F_DELMDIR deletes an existing module directory. If the target module directory is not empty, an EOS_DNE directory not empty error is returned.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

name
    Point to the module directory.

Possible Errors
EOS_BNAM
EOS_DNE
EOS_MNF
EOS_PERMIT
F_DELTSK
Deallocate Process Descriptor

Headers
#include <process.h>

Parameter Block Structure
typedef struct f_deltsk_pb {
    syscb cb;
    Pr_desc proc_desc;
} f_deltsk_pb, *F_deltsk_pb;

Description
F_DELTSK is called when a process terminates to return the process' protection resources. This call must release any protection structures allocated to the process, whether this be memory or any hardware resource.

F_DELTSK is only supported on systems with a memory protection unit (for example, all 80386 and 80486 systems and PowerPC systems). The SSM module must be present in the bootfile.

If the SSM module is not present in the system, an EOS_UNKSVC error is returned. You should ignore this error.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

cb
    The control block header.

proc_desc
    Point to the process descriptor.

Possible Errors
EOS_BNAM
EOS_UNKSVC

See Also
F_ALLTSK
F_CHKMEM
F_PERMIT
F_PROTECT
Headers

#include <types.h>
#include <dexec.h>

Parameter Block Structure

typedef struct f_dexec_pb {
    syscb cb;
    process_id proc_id;
    dexec_mode mode;
    u_char *params;
    u_int32 num_instr,
        tot_instr,
        except,
        addr;
    u_int16 num_bpts,
        **brk_pts;
    dexec_status status;
    error_code exit_status;
} f_dexec_pb, *F_dexec_pb;

Description

F_DEXEC controls the execution of a suspended child process created by F_DFORK. The process performing the F_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. Therefore, the parent and the child processes are never active at the same time.

When F_DEXEC is executed in DBG_M_SOFT or DBG_M_COUNT mode, it traces every instruction of the child process and checks for the termination conditions. Breakpoints are lists of addresses to check and work with ROMed object programs. Consequently, the child process being debugged runs at a slow speed.

When F_DEXEC is executed in DBG_M_HARD mode, it replaces the instruction at each breakpoint address with an illegal opcode. Next, it executes the child process at full speed (with the trace bit clear) until a breakpoint is reached or the program terminates. This can save an enormous amount of time. However, F_DEXEC cannot count the number of executed instructions.

When status is DBG_S_EXCEPT, the except parameter is the specific exception identifier (error) causing the child to return to the debugger.
OS-9 system calls made by the suspended program are executed at full speed and are considered one logical instruction. This is also true of system-state trap handlers. You cannot debug system-state processes.

The system uses the register buffer and floating point register buffer passed in the \texttt{F\_DFORK} call to save and restore the child's registers. Changing the contents of the register buffer alters the child process’ registers.

An \texttt{F\_DEXIT} call must be made to return the debugged process’ resources (memory).

Tracing is allowed through subroutine libraries and intercept routines. This is not a problem, but you will see code executed that is not explicitly in your sources.

**Attributes**

Operating System: OS-9  
State: User and System  
Threads: Safe

**Parameters**

\texttt{cb}  
The control block header.

\texttt{proc\_id}  
The process ID of the child to execute.

\texttt{mode}  
Specify the debug mode. These modes are defined in the header file \texttt{dexec.h}:

<table>
<thead>
<tr>
<th>Debug Modes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{DBG_M_INACTV}</td>
<td>Inactive mode (used by the kernel).</td>
</tr>
<tr>
<td>\texttt{DBG_M_HARD}</td>
<td>Hard breakpoints/full speed execution.</td>
</tr>
<tr>
<td>\texttt{DBG_M_SOFT}</td>
<td>Soft breakpoints/continuous execution.</td>
</tr>
<tr>
<td>\texttt{DBG_M_COUNT}</td>
<td>Execute count instructions.</td>
</tr>
<tr>
<td>\texttt{DBG_M_CONTROL}</td>
<td>Execute until change of control (future release).</td>
</tr>
</tbody>
</table>

\texttt{params}  
The parameter list pointer. This will be implemented in a future release.

\texttt{num\_instr}  
The number of instructions to execute. If \texttt{num\_instr} is zero, commands are executed continuously. Upon completion of the \texttt{F\_DEXEC} call, \texttt{num\_instr} is updated with a value representing the original number of instructions less the number of instructions executed.

\texttt{tot\_instr}  
A returned value. It points to the number of instructions executed so far when the child is executed in trace mode.
except
A returned value. It is the exception the child received, when the child process returned due to an exception.

addr
A returned value. It is the violation address associated with an exception condition.

num_bpts
Specify the number of breakpoints in the list.

brk_pts
Point to the breakpoint list. The breakpoint list is a list of addresses indicating which instructions are considered breakpoints.

status
The process return status. status indicates the reason the child process returned to the debugger. The following status modes are defined in the header file dexec.h:

<table>
<thead>
<tr>
<th>Status Modes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBG_S_FINISH</td>
<td>The command finished successfully.</td>
</tr>
<tr>
<td>DBG_S_BRKPNT</td>
<td>The process hit a breakpoint.</td>
</tr>
<tr>
<td>DBG_S_EXCEPT</td>
<td>An exception occurred during execution.</td>
</tr>
<tr>
<td>DBG_S_CHILDSIG</td>
<td>The process received a signal (no intercept).</td>
</tr>
<tr>
<td>DBG_S_PARENTSIG</td>
<td>The debugger received a signal.</td>
</tr>
<tr>
<td>DBG_S_CHAIN</td>
<td>The process made an F_CHAIN system call.</td>
</tr>
<tr>
<td>DBG_S_EXIT</td>
<td>The process made an F_EXIT system call.</td>
</tr>
<tr>
<td>DBG_S_CONTROL</td>
<td>The process executed a jmp or bra (future release).</td>
</tr>
<tr>
<td>DBG_S_WATCH</td>
<td>The process hit a watch point (future release).</td>
</tr>
<tr>
<td>DBG_S_FORK</td>
<td>The process made an F_FORK system call.</td>
</tr>
</tbody>
</table>

exit_status
A returned value. It is the child’s exit status, when the child performs an F_EXIT call.

Possible Errors
EOS_IPRCID
EOS_PRCABT

See Also
F_CHAIN
F_DEXIT
F_DFORK
F_EXIT
Chapter 6: OS-9 System Calls

F_DEXIT
Exit Debugged Program

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_dexit_pb {
    syscb cb;
    process_id proc_id;
} f_dexit_pb, *F_dexit_pb;

Description
F_DEXIT terminates a suspended child process created by F_DFORK. The F_EXIT done by the child process does not release the child’s resources in the case of a debugged process. This enables examination of the child after its termination. Therefore, the debugger must do an F_DEXIT to release the child’s resources after this call.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
        The control block header.

    proc_id
        The process ID of the child to terminate.

Possible Errors
EOS_IPRCID

See Also
F_DEXEC
F_DFORK
F_EXIT
F_DFORK
Fork Process Under Control of Debugger

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_dfork_pb {
    syscb cb;
    u_int16 priority,
        path_cnt;
    process_id proc_id;
    Regs reg_stack;
    Fregs freg_stack;
    u_char *mod_name,
        *params;
    u_int32 mem_size,
        param_size;
    u_int16 type_lang;
} f_dfork_pb, *F_dfork_pb;

Description:
F_DFORK creates a new process that becomes a child of the caller. It sets up the
process’ memory, MPU registers, and standard I/O paths. In addition, F_DFORK
enables 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 enables the debugger to control its execution through the
F_DEXEC and F_DEXIT system calls.

The child process is created in the DBG_M_SOFT (trace) mode and is executed with
F_DEXEC.

The register buffer is an area in the caller’s data area permanently associated with
each child process. It is set to an image of the child’s initial registers for use with
F_DEXEC.

For information about process creation, refer to the F_FORK description.
Processes whose primary module is owned by a super-user can only be debugged by
a super user. You cannot debug system-state processes.

Attributes
Operating System:       OS-9
State:                  User and System
Threads:                Safe
Chapter 6: OS-9 System Calls

Parameters

- **cb**: The control block header.
- **priority**: The priority of the new process.
- **path_cnt**: The number of I/O paths for the child to inherit.
- **proc_id**: A returned value. It is the new child process ID.
- **reg_stack**: Point to the register buffer.
- **freg_stack**: Point to the floating point register buffer.
- **mod_name**: Point to the module name.
- **params**: Point to the parameter string to pass to the new process.
- **mem_size**: Specify any additional stack space to allocate above the default specified in the primary module’s module header.
- **param_size**: Specify the size of the parameter string.
- **type_lang**: Specify the desired type and language of the primary module to be executed.

Possible Errors

- EOS_MNF
- EOS_NEMOD
- EOS_NORAM
- EOS_PERMIT
- EOS_PNNF

See Also

- F_DEXEC
- F_DEXIT
- F_DFORKM
- F_FORK
Chapter 6: OS-9 System Calls

F_DFORKM
Fork Process Under Control of Debugger

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_dforkm_pb {
    syscb cb;
    u_int16 priority,
        path_cnt;
    process_id proc_id;
    Regs reg_stack;
    Fregs freg_stack;
    Mh_com mod_head;
    u_char *params;
    u_int32 mem_size,
        param_size;
} f_dforkm_pb, *F_dforkm_pb;

Description
F_DFORKM creates a new process that becomes a child of the caller. It sets up the process’ memory, MPU registers, and standard I/O paths. In addition, F_DFORKM enables 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 enables the debugger to control its execution through the F_DEXEC and F_DEXIT system calls. F_DFORKM is similar to F_DFORK. However, F_DFORKM is passed a pointer to the module to fork rather than the module name.

For more information, refer to the description of F_DFORK.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
   cb
    The control block header.
   priority
    The priority of the new process.
path_cnt
    The number of I/O paths for the child to inherit.

proc_id
    A returned value. It is a the new child process ID.

reg_stack
    Point to the register buffer.

freg_stack
    Point to the floating point register buffer.

mod_head
    Point to the module header.

params
    Point to the parameter string to pass to the new process.

mem_size
    Specify any additional stack space to allocate above the default specified in the
    primary module’s module header.

param_size
    Specify the size of the parameter string.

Possible Errors
EOS_MNF
EOS_NEMOD
EOS_NORAM
EOS_PERMIT
EOS_PNNF

See Also
F_DEEXEC
F_DEEXIT
F_DFORK
F_FORK
Chapter 6: OS-9 System Calls

F_EVENT
Process Synchronization and Communication

Headers
Refer to the specific event for the header to include.

Parameter Block Structure
Refer to the specific event for the appropriate parameter block structure.

Description
OS-9 events are multiple-value semaphores that synchronize concurrent processes sharing resources such as files, data modules, and CPU time. The events’ functions enable processes to create/delete events, link/unlink events, get event information, and suspend operation until an event occurs. Events are also used for various means of signalling.

The following events functions are currently supported:

Table 6-3. Events Functions

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F_EVENT, EV_ALLCLR</td>
<td>Wait for all bits defined by mask to become clear.</td>
</tr>
<tr>
<td>F_EVENT, EV_ALLSET</td>
<td>Wait for all bits defined by mask to become set.</td>
</tr>
<tr>
<td>F_EVENT, EV_ANYCLR</td>
<td>Wait for any bits defined by mask to become clear.</td>
</tr>
<tr>
<td>F_EVENT, EV_ANYSET</td>
<td>Wait for any bits defined by mask to become set.</td>
</tr>
<tr>
<td>F_EVENT, EV_CHANGE</td>
<td>Wait for any bits defined by mask to change.</td>
</tr>
<tr>
<td>F_EVENT, EV_CREAT</td>
<td>Create new event.</td>
</tr>
<tr>
<td>F_EVENT, EV_DELET</td>
<td>Delete existing event.</td>
</tr>
<tr>
<td>F_EVENT, EV_INFO</td>
<td>Return event information.</td>
</tr>
<tr>
<td>F_EVENT, EV_LINK</td>
<td>Link to existing event by name.</td>
</tr>
<tr>
<td>F_EVENT, EV_PULSE</td>
<td>Signal event occurrence.</td>
</tr>
<tr>
<td>F_EVENT, EV_READ</td>
<td>Read event value without waiting.</td>
</tr>
<tr>
<td>F_EVENT, EV_SET</td>
<td>Set event variable and signal event occurrence.</td>
</tr>
<tr>
<td>F_EVENT, EV_SETAND</td>
<td>Set event value by ANDing the event value with a mask.</td>
</tr>
<tr>
<td>F_EVENT, EV_SETOR</td>
<td>Set event value by ORing the event value with a mask.</td>
</tr>
<tr>
<td>F_EVENT, EV_SETR</td>
<td>Set relative event variable and signal event occurrence.</td>
</tr>
<tr>
<td>F_EVENT, EV_SETXOR</td>
<td>Set event value by XORing the event value with a mask.</td>
</tr>
<tr>
<td>F_EVENT, EV_SIGNL</td>
<td>Signal event occurrence.</td>
</tr>
<tr>
<td>F_EVENT, EV_TSTSET</td>
<td>Wait for all bits defined by mask to clear, then set these bits.</td>
</tr>
<tr>
<td>F_EVENT, EV_UNLNK</td>
<td>Unlink event.</td>
</tr>
<tr>
<td>F_EVENT, EV_WAIT</td>
<td>Wait for event to occur.</td>
</tr>
<tr>
<td>F_EVENT, EV_WAITR</td>
<td>Wait for relative event to occur.</td>
</tr>
</tbody>
</table>
Specific parameters and functions of each event operation are discussed in the following pages. The EV_XXX function names are defined in the system definition file funcs.h. 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.

**Attributes**

Operating System: OS-9  
State: User and System  
Threads: Safe

**Possible Errors**

EOS_EVNTID

**See Also**

F_EVENT, EV_SIGNAL
F_EVENT, EV_ALLCLR
Wait for All Bits Defined by Mask to Become Clear

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evallclr_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 mask;
} f_evallclr_pb, *F_evallclr_pb;

Description
EV_ALLCLR waits until one of the event set calls occurs that clears all of the bits corresponding to the set bits in the mask. The event variable is ANDed with the value in mask. If the resulting value is not zero, the calling process is suspended in a FIFO event queue.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
        The control block header.

    ev_code
        The EV_ALLCLR event function code.

    ev_id
        Identify the event.

    value
        A returned value. It is the actual event value after the set operation that activated the suspended process.

        If the process receives a signal while in the event queue, it is activated and an EOS_SIGNAL error is returned, even though the event has not actually occurred. Also, the current event value is returned and the caller’s intercept routine is executed.
signal
Contains the returned signal code.

mask
Specify the activation mask. It indicates which bits are significant to the caller.

**Possible Errors**

EOS_EVNTID
EOS_SIGNAL
Chapter 6: OS-9 System Calls

F_EVENT, EV_ALLSET

Wait for Event to Occur

Headers

```c
#include <types.h>
```

Parameter Block Structure

```c
typedef struct f_evallset_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 mask;
} f_evallset_pb, *F_evallset_pb;
```

Description

`EV_ALLSET` waits until one of the event set calls occurs that sets all of the bits corresponding to the set bits in the mask. The event variable is ANDed with the value in `mask` and then EXCLUSIVE-ORed with it. If the resulting value is not zero, the calling process is suspended in a FIFO event queue.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe

Parameters

- `cb`  
  The control block header.
- `ev_code`  
  The `EV_ALLSET` event function code.
- `ev_id`  
  Identify the event.
- `value`  
  A returned value. It is the actual event value after the set operation that activated the suspended process.

If the process receives a signal while in the event queue, it is activated and an `EOS_SIGNAL` error is returned, even though the event has not actually occurred. Also, the current event value is returned and the caller’s intercept routine is executed.
signal
   Contains the returned signal code.

mask
   Specify the activation mask. It indicates which bits are significant to the caller.

Possible Errors
EOS_EVNTID
EOS_SIGNAL
Chapter 6: OS-9 System Calls

F_EVENT, EV_ANYCLR
Wait for Event to Occur

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evanyclr_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 mask;
} f_evanyclr_pb, *F_evanyclr_pb;

Description
EV_ANYCLR waits for an event to occur. The event variable is ANDed with the value in mask and then EXCLUSIVE-ORed with it. If the resulting value is zero, the calling process is suspended in a FIFO event queue. It waits until one of the event set calls occurs that clears any of the bits corresponding to the set bits in the mask.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
        The control block header.

    ev_code
        The EV_ANYCLR event function code.

    ev_id
        Identify the event.

    value
        A returned value. It is the actual event value after the set operation that activated the suspended process.

        If the process receives a signal while in the event queue, it is activated and an EOS_SIGNAL error is returned, even though the event has not actually occurred. Also, the current event value is returned and the caller’s intercept routine is executed.
signal
Contains the returned signal code.

mask
Specify the activation mask. It indicates which bits are significant to the caller.

Possible Errors
EOS_EVNTID
EOS_SIGNAL
Chapter 6: OS-9 System Calls

F_EVENT, EV_ANYSET
Wait for Event to Occur

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evanyset_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 mask;
} f_evanyset_pb, *F_evanyset_pb;

Description
EV_ANYSET waits until one of the event set calls occurs that sets any of the bits corresponding to the set bits in the mask. The event variable is ANDed with the value in mask. If the resulting value is zero, the calling process is suspended in a FIFO event queue.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
cb
    The control block header.

ev_code
    The EV_ANYCLR event function code.

ev_id
    Identify the event.

value
    A returned value. It is the actual event value after the set operation that activated the suspended process.

    If the process receives a signal while in the event queue, it is activated and an EOS_SIGNAL error is returned, even though the event has not actually occurred. Also, the current event value is returned and the caller’s intercept routine is executed.
signal
   Contains the returned signal code.

mask
   Specify the activation mask. It indicates which bits are significant to the caller.

Possible Errors
   EOS_EVNTID
   EOS_SIGNAL
Chapter 6: OS-9 System Calls

F_EVENT, EV_CHANGE
Wait for Event to Occur

Headers:
#include <types.h>

Parameter Block Structure
typedef struct f_evchange_pb {
  syscb cb;
  u_int16 ev_code;
  event_id ev_id;
  int32 value;
  signal_code signal;
  u_int32 mask;
  u_int32 pattern;
} f_evchange_pb, *F_evchange_pb;

Description
EV_CHANGE waits until one of the event set calls occurs that changes any of the bits corresponding to the set bits in mask. The event variable is ANDed with the value in mask. If the resulting value is not equal to the wait pattern, the calling process is suspended in a FIFO event queue.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
  cb
    The control block header.
  ev_code
    The EV_ANYCLR event function code.
  ev_id
    Identify the event.
  value
    A returned value. It is the actual event value after the set operation that activated the suspended process.
    If the process receives a signal while in the event queue, it is activated and an EOS_SIGNAL error is returned, even though the event has not actually occurred. Also, the current event value is returned and the caller’s intercept routine is executed.
signal
Contains the returned signal code.

mask
Specify the activation mask. It indicates which bits are significant to the caller.

pattern
Specify the wait pattern.

Possible Errors
EOS_EVNTID
EOS_SIGNAL
**F_EVENT, EV_CREAT**

Create New Event

---

**Headers**

#include <types.h>

**Parameter Block Structure**

typedef struct f_evcreat_pb {
    syscb cb;
    u_int16 ev_code,
    wait_inc,
    sig_inc,
    perm,
    color;
    event_id ev_id;
    u_char *ev_name;
    u_int32 value;
} f_evcreat_pb, *F_evcreat_pb;

**Description**

EV_CREAT creates events dynamically as needed. When an event is created, an initial value is specified, as well as increments to be applied each time the event is waited for or occurs. Subsequent event calls use the returned ID number to refer to the created event.

**Attributes**

Operating System: OS-9
State: User and System
Threads: Safe

**Parameters**

cb
    The control block header.

ev_code
    The EV_CREAT event function code.

wait_inc
    Specify the auto-increment value for EV_WAIT.

sig_inc
    Specify the auto-increment value for EV_SIGNAL.
perm
    Specify the access permissions.

color
    Specify the memory type for the event structure.

ev_id
    A returned value. Event identifier used for subsequent event calls.

ev_name
    Point to the event name string.

value
    Specify the initial event variable value.

Possible Errors
EOS_BNAM
EOS_EVBUSY
EOS_EVFULL
EOS_NORAM

See Also
F_EVENT, EV_DELET
F_EVENT, EV_SIGNAL
F_EVENT, EV_WAIT
F_EVENT, EV_CREAT | F_USEMA
Create New Usemaphore

Headers

```
#include <types.h>
```

Parameter Block Structure

```
typedef struct f_evcreat_pb {
    syscb cb;
    u_int16 ev_code,
        wait_inc,
        sig_inc,
        perm,
        color;
    event_id ev_id;
    u_char *ev_name;
    u_int32 value;
} f_evcreat_pb, *F_evcreat_pb;
```

Description

`EV_CREAT | F_USEMA` creates usemaphores dynamically as needed. When a
usemaphore is created an initial value is specified. Subsequent usemaphore calls use
the returned ID number to refer to the created usemaphore.

No usemaphore may have the same name as an event.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe

Parameters

- `cb`
  The control block header.
- `ev_code`
  The `EV_CREAT | F_USEMA` function code.
- `wait_inc`
  Specify the auto-increment value for `EV_WAIT`. `wait_inc` must be -1.
- `sig_inc`
  Specify the auto-increment value for `EV_SIGNAL`. `sig_inc` must be 1.
perm
   Specify the access permissions.

color
   Specify the memory type for the usemaphore structure.

ev_id
   A returned value. It is the usemaphore identifier used for subsequent
   usemaphore calls.

ev_name
   Pointer to the usemaphore name string.

value
   Specify the initial usemaphore variable value. value must be 0 or 1. If it is 0,
   the usemaphore will be created as if it was atomically created in the released
   state and then acquired by the calling process.

Possible Errors
EOS_BNAM
EOS_EVBUSY
EOS_EVFULL
EOS_NORAM

See Also
F_EVENT, EV_DELET
F_EVENT, EV_SIGNL
F_EVENT, EV_WAIT
F_EVENT, EV_DELET
Remove Event

Headers

```
#include <types.h>
```

Parameter Block Structure

```
typedef struct f_evdelet_pb {
    syscb cb;
    u_int16 ev_code;
    u_char *ev_name;
} f_evdelet_pb, *F_evdelet_pb;
```

Description

`EV_DELET` removes an event from the system event table, freeing the entry for use by another event. Events have an implicit use count (initially set to 1), which is incremented with each `EV_LINK` call and decremented with each `EV_UNLINK` call. An event may not be deleted unless its use count is zero.

OS-9 does not automatically unlink events when `EOS_EXIT` occurs.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe

Parameters

- `cb`:
  The control block header.

- `ev_code`:
  The `EV_DELET` event function code.

- `name`:
  Point to the event’s name string.

Possible Errors

- `EOS_BNAM`
- `EOS_EVBUSY`
- `EOS_EVNF`

See Also

- `F_EVENT, EV_CREAT`
- `F_EVENT, EV_LINK`
- `F_EVENT, EV_UNLINK`
F_EVENT, EV_DELET | F_USEMA
Remove Usemaphore

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evdelet_pb {
    syscb cb;
    u_int16 ev_code;
    u_char *ev_name;
} f_evdelet_pb, *F_evdelet_pb;

Description
EV_DELET | F_USEMA removes a usemaphore from the system usemaphore table, freeing the entry for use by another event or usemaphore. Usemaphores have an implicit use count (initially set to 1 by EV_CREAT | F_USEMA), which is incremented with each EV_LINK | F_USEMA call and decremented with each EV_UNLINK | F_USEMA call. A usemaphore may not be deleted unless its use count is zero.
OS-9 automatically unlocks, if necessary, and unlinks any linked usemaphores upon process termination, but does not delete them.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

ev_code
    The EV_DELET | F_USEMA function code.

name
    Pointer to the usemaphore’s name string.

Possible Errors
EOS_BNAM
EOS_EVBUSY
EOS_EVNF

See Also
F_EVENT, EV_CREAT | F_USEMA
F_EVENT, EV_LINK | F_USEMA
F_EVENT, EV_UNLINK | F_USEMA
Chapter 6: OS-9 System Calls

F_EVENT, EV_INFO

Return Event Information

Headers

#include <events.h>

Parameter Block Structure

typedef struct f_evinfo_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    u_int32 size;
    u_char *buffer;
} f_evinfo_pb, *F_evinfo_pb;

Description

EV_INFO returns event information in your buffer. This call is used by utilities needing to know the status of all active events. The information returned is defined by the ev_infostr event information structure defined in the events.h header file.

The name of the event is appended to the end of the information structure. The information buffer and size parameters must be large enough to accommodate the name of the target event.

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.

Attributes

Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

cb

    The control block header.

ev_code

    The EV_INFO event function code.

ev_id

    Specify the event index to use to begin the search. Unlike other event functions, only an event index is passed in the ev_id parameter. The index is the system event number, ranging from zero to one less than the maximum number of system events.
size
   Specify the buffer size.

buffer
   Point to the buffer containing the event information.

Possible Errors
EOS_EVNTID

See Also
ev_str/ev_infostr
Chapter 6: OS-9 System Calls

F_EVENT, EV_LINK
Link to Existing Event by Name

Headers
#include <types.h>

Parameter Block Structure

typedef struct f_evlink_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    u_char *ev_name;
} f_evlink_pb, *F_evlink_pb;

Description

EV_LINK determines the ID number of an existing event. Once an event has been linked, all subsequent references are made using the returned event ID. This permits the system to access events quickly, while preventing programs from using invalid or deleted events. The event use count is incremented when an EV_LINK is performed. To keep the use count synchronized properly, use EV_UNLINK when the event is no longer used.

The event access permissions are checked only at link time.

Attributes

Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

cb
       The control block header.

ev_code
       The EV_LINK event function code.

ev_name
       Point to the event name string.

ev_id
       The event identifier used for subsequent event calls.
Possible Errors
EOS_BNAM
EOS_EVNF
EOS_PERMIT

See Also
F_EVENT, EV_UNLNX
Chapter 6: OS-9 System Calls

F_EVENT, EV_LINK | F_USEMA
Link to Existing Usemaphore by Name

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evlink_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    u_char *ev_name;
} f_evlink_pb, *F_evlink_pb;

Description
EV_LINK | F_USEMA determines the ID number of an existing usemaphore. Once a
usemaphore has been linked, all subsequent references are made using the returned
usemaphore ID. This permits the system to access usemaphores quickly, while
preventing programs from using invalid or deleted usemaphores. The usemaphore
use count is incremented when an EV_LINK | F_USEMA is performed. To keep the use
count synchronized properly, use EV_UNLINK | F_USEMA when the usemaphore is no
longer used.

OS-9 automatically unlocks, if necessary, and unlinks any linked usemaphores upon
process termination, but does not delete them.

The usemaphore access permissions are checked only at link time.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
        The control block header.

    ev_code
        The EV_LINK | F_USEMA function code.

    ev_name
        Pointer to the usemaphore name string.

    ev_id
        The usemaphore identifier used for subsequent usemaphore calls.
Possible Errors
EOS_BNAM
EOS_EVNF
EOS_PERMIT

See Also
F_EVENT, EV_UNLNK | F_USEMA
Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evpulse_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    u_int32 actv_flag;
} f_evpulse_pb, *F_evpulse_pb;

Description
EV_PULSE signals an event occurrence. The event value is set to what is passed in value, and the signal auto-increment is not applied. Then, the event queue is searched for the first process waiting for that event value, after which the original event value is restored.

EV_PULSE with the actv_flag set executes as follows for each process in the queue until the queue is exhausted:
1.  The signal auto-increment is added to the event variable.
2.  The first process in range is awakened.
3.  The event value is updated with the wait auto-increment.
4.  The search is continued with the updated value.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters
cb  
   The control block header.

ev_code  
   The EV_PULSE event function code.

ev_id  
   Identify the event.
value
   The event value prior to the pulse operation.

actv_flag
   Specify which process(es) to activate.
   • If actv_flag is one, all processes in range are activated.
   • If actv_flag is not set, only the first process in the event queue waiting for that range is activated.

Possible Errors

EOS_EVTID
F_EVENT, EV_READ
Read Event Value Without Waiting

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evread_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
} f_evread_pb, *F_evread_pb;

Description
EV_READ reads the value of an event without waiting or affecting the event variable. This determines the availability of the event without wait.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters
cb
    The control block header.

    The EV_READ event function code.

ev_id
    Identify the event.

    The current event value.

Possible Errors
EOS_EVNTID
Chapter 6: OS-9 System Calls

F_EVENT, EV_RESET | F_USEMA
Acquire Ownership of a Usemaphore and Reset

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evwait_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 min_val,
        max_val;
} f_evwait_pb, *F_evwait_pb;

Description
EV_RESET | F_USEMA claims ownership of a usemaphore and clears the need for a reset. This is a non-blocking call. If a reset is necessary the caller will be granted ownership immediately and the need to reset the usemaphore will be cleared. If no reset is required EOS_USNORST will be returned.

The caller should use F_EVENT, EV_SIGNAL | F_USEMA to release the ownership acquired via F_EVENT, EV_RESET | F_USEMA.

Usemaphores have to be reset when the operating system releases ownership because a process terminates without manually releasing ownership.

Attributes
Operating System:   OS-9
State:       User and System
Threads:    Safe

Parameters

    cb
    The control block header.

    ev_code
    The EV_RESET | F_USEMA function code.

    ev_id
    Identify the usemaphore.
value
   A returned value. It is the actual usemaphore value prior to the application of
   the wait increment.

signal
   A returned value. Since this is a non-blocking call the value of signal is not
   defined.

min_val
   The minimum activation value. This must be 1.

max_val
   The maximum activation value. This must be 1.

Possible Errors
EOS_EVNTID
EOS_USRST

See Also
F_EVENT, EV_SIGNAL | F_USEMA
F_EVENT, EV_WAIT | F_USEMA
F_EVENT, EV_TRYWAIT | F_USEMA
F_EVENT, EV_SET
Set Event Variable and Signal Event Occurrence

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evset_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    u_int32 actv_flag;
} f_evset_pb, *F_evset_pb;

Description
EV_SET signals an event has occurred. The current event variable is set to the value passed at value, rather than updated with the signal auto-increment. Next, the event queue is searched for the first process waiting for the event value.

EV_SET with the actv_flag set executes as follows for each process in the queue until the queue is exhausted:
1. The first process in range is awakened.
2. The event value is updated with the wait auto-increment.
3. The search is continued with the updated value.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters
cb
    The control block header.
ev_code
    The EV_SET event function code.
ev_id
    Identify the event.
value
    The event value prior to the set operation.
actv_flag
   Specify which process(es) to activate.
   • If actv_flag is one, all processes in range are activated.
   • If actv_flag is not set, only the first process in the event queue waiting for that range is activated.

Possible Errors
EOS_EVNTID
Chapter 6: OS-9 System Calls

F_EVENT, EV_SETAND
Set Event Variable and Signal Event Occurrence

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evsetand_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    u_int32 mask,
        actv_flag;
} f_evsetand_pb, *F_evsetand_pb;

Description
EV_SETAND signals an event has occurred. The current event variable is ANDed with the value passed in mask rather than updated with the signal auto-increment. Next, the event queue is searched for the first process waiting for that event value.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

cb
The control block header.

ev_code
The EV_SETAND event function code.

ev_id
Identify the event.

value
The event value prior to the logical operation.

mask
The event mask. It indicates which bits are significant to the caller.
actv_flag
   Specify which process(es) to activate.
   • If actv_flag is one, all processes in range are activated.
   • If actv_flag is not set, only the first process in the event queue waiting for that range is activated.

Possible Errors

EOS_EVNTID
F_EVENT, EV_SETOR
Set Event Variable and Signal Event Occurrence

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evsetor_pb {
syscb cb;
   u_int16 ev_code;
   event_id ev_id;
   u_int32 mask,
      actv_flag;
} f_evsetor_pb, *F_evsetor_pb;

Description
EV_SETOR signals an event has occurred. The current event variable is ORed with the
value passed in mask. Next, the event queue is searched for the first process waiting
for that event value.

Attributes
Operating System:  OS-9
State:  User, System, and Interrupt
Threads:  Safe

Parameters
cb
   The control block header.

ev_code
   The EV_SETOR event function code.

ev_id
   Identify the event.

value
   The event value prior to the logical operation.

mask
   The event mask. It indicates which bits are significant to the caller.
actv_flag
   Specify which processes to activate.
   • If actv_flag is one, all processes in range are activated.
   • If actv_flag is not set, only the first process in the event queue waiting for
     that range is activated.

Possible Errors
EOS_EVNTID
F_EVENT, EV_SETR
Set Relative Event Variable and Signal Event Occurrence

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evsetr_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    u_int32 actv_flag;
} f_evsetr_pb, *F_evsetr_pb;

Description
EV_SETR signals an event has occurred. The current event value is incremented by value, rather than by the signal auto-increment. Next, 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_SETR with the actv_flag set executes as follows for each process in the queue until the queue is exhausted:
1. The first process in range is awakened.
2. The event value is updated with the wait auto-increment.
3. The search is continued with the updated value.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters
    cb
        The control block header.

    ev_code
        The EV_SETOR event function code.

    ev_id
        Identify the event.
value
The event value prior to the logical operation.

actv_flag
Specify which processes to activate.
- If actv_flag is one, all processes in range are activated.
- If actv_flag is not set, only the first process in the event queue waiting for that range is activated.

Possible Errors
EOS_EVNTID

See Also
F_EVENT, EV_SET
F_EVENT, EV_SIGNAL
Chapter 6: OS-9 System Calls

F_EVENT, EV_SETXOR
Set Event Variable and Signal Event Occurrence

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evsetxor_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    u_int32 mask,
       actv_flag;
} f_evsetxor_pb, *F_evsetxor_pb;

Description
EV_SETXOR signals an event has occurred. The current event value is EXCLUSIVE-ORed with mask rather than updated with the signal auto-increment. Next, the event queue is searched for the first process waiting for that event value.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

    cb
        The control block header.

    ev_code
        The EV_SETOR event function code.

    ev_id
        Identify the event.

    value
        The event value prior to the logical operation.

    mask
        The event mask. It indicates which bits are significant to the caller.
**actv_flag**

Specify which processes to activate.

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is not set, only the first process in the event queue waiting for that range is activated.

**Possible Errors**

`EOS_EVNTID`
Headers

```c
#include <types.h>
```

Parameter Block Structure

```c
typedef struct f_evsignl_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    u_int32 actv_flag;
} f_evsignl_pb, *F_evsignl_pb;
```

Description

`EV_SIGNL` signals an event has occurred. The current event variable is updated with the signal auto-increment specified when the event was created. Next, the event queue is searched for the first process waiting for that event value.

`EV_SIGNL` with the `actv_flag` set, executes as follows for each process in the queue until the queue is exhausted:

1. The signal auto-increment is added to the event variable.
2. The first process in range is awakened.
3. The event value is updated with the wait auto-increment.
4. The search is continued with the updated value.

Attributes

- Operating System: OS-9
- State: User, System, and Interrupt
- Threads: Safe

Parameters

- `cb`
  - The control block header.

- `ev_code`
  - The `EV_SETOR` event function code.

- `ev_id`
  - Identify the event.

- `value`
  - The event value prior to the logical operation.
**actv_flag**

Specify which processes to activate.

- If `actv_flag` is one, all processes in range are activated.
- If `actv_flag` is not set, only the first process in the event queue waiting for that range is activated.

**Possible Errors**

`EOS_EVNTID`
Chapter 6: OS-9 System Calls

F_EVENT, EV_SIGNL | F_USEMA
Release Ownership of a Usemaphore

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evsign1_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    u_int32 actv_flag;
} f_evsign1_pb, *F_evsign1_pb;

Description
EV_SIGNL | F_USEMA releases the ownership of a usemaphore. The current
usemaphore variable is updated with the signal auto-increment (plus 1), thus
releasing ownership. Next, the usemaphore queue is searched for the first process
waiting for the usemaphore.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

cb
    The control block header.

ev_code
    The EV_SIGNL | F_USEMA function code.

ev_id
    Identify the semaphore.

value
    The usemaphore value prior to the release operation. Will always be 1.

actv_flag
    Specify which processes to activate. Must be 0.
Possible Errors

EOS_EVNTID
EOS_PARAM

See Also

F_EVENT, EV_WAIT | F_USEMA
F_EVENT, EV_TRYWAIT | F_USEMA
F_EVENT, EV_TRYWAIT
Check for Event Without Blocking

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evwait_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 min_val,
    max_val;
} f_evwait_pb, *F_evwait_pb;

Description
EV_TRYWAIT checks if the current event value is in the range between the minimum and maximum activation values. If it is in range, the wait auto-increment (specified at creation) is added to the event variable and SUCCESS is returned. If the value is not in range, EAGAIN is returned.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
   cb
       The control block header.
   ev_code
       The EV_TRYWAIT event function code.
   ev_id
       Identify the event.
   value
       A returned value. It is the event value at the time of the call.
signal
   A returned value. The value of signal is undefined for EV_TRYWAIT since it
does not block.

min_val
   The minimum activation value.

max_val
   The maximum activation value. The event value is added to min_val and
   max_val, and the actual absolute values are returned to the caller. If an
   underflow or overflow occurs on the addition, the values 0x80000000
   (minimum integer) or 0x7fffffff (maximum integer) are used, respectively.

Possible Errors
EOS_EVNTID
EOS_EVPARM
EAGAIN

See Also
F_EVENT, EV_SIGNAL
F_EVENT, EV_WAIT
F_EVENT, EV_TRYWAIT | F_USEMA
Acquire Ownership of a Usemaphore Without Blocking

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evwait_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 min_val,
    max_val;
} f_evwait_pb, *F_evwait_pb;

Description
EV_TRYWAIT | F_USEMA claims ownership of the specified usemaphore if it is currently unowned. If it is unowned, the wait auto-increment (minus 1) is then applied to the usemaphore value. If it is currently owned, EAGAIN is returned. If the usemaphore needs to be reset EOS_USRST will be returned.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
cb
    The control block header.
ev_code
    The EV_TRYWAIT | F_USEMA function code.
ev_id
    Identify the usemaphore.
value
    A returned value. It is the actual usemaphore value prior to the application of the wait increment.
signal
A returned value. Since EV_TRYWAIT | F_USEMA is a non-blocking operation, the value of the signal field is undefined.

min_val
The minimum activation value. This must be 1.

max_val
The maximum activation value. This must be 1.

Possible Errors
EOS_EVNTID
EOS_USRST
EOS_PARAM

See Also
F_EVENT, EV_SIGNL | F_USEMA
F_EVENT, EV_WAIT | F_USEMA


Chapter 6: OS-9 System Calls

F_EVENT, EV_TSTSET
Wait for Event to Occur

Headers

```c
#include <types.h>
```

Parameter Block Structure

```c
typedef struct f_evtstset_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 mask;
} f_evtstset_pb, *F_evtstset_pb;
```

Description

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 EV_SIGNAL occurs clearing all of the bits corresponding to the set bits in the mask. Next, the bits corresponding to the set bits in the mask are set.

Attributes

- **Operating System**: OS-9
- **State**: User and System
- **Threads**: Safe

Parameters

- **cb**
  The control block header.
- **ev_code**
  The EV_SETOR event function code.
- **ev_id**
  Identify the event.
- **value**
  The event value prior to the logical operation.
- **mask**
  The event mask. It indicates which bits are significant to the caller.
actv_flag
Specify which processes to activate.

- If actv_flag is one, all processes in range are activated.
- If actv_flag is not set, only the first process in the event queue waiting for that range is activated.

Possible Errors
EOS_EVNTID
F_EVENT, EV_UNLNK
Unlink Event

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evunlnk_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
} f_evunlnk_pb, *F_evunlnk_pb;

Description
EV_UNLNK informs the system a process is no longer using the event. This decrements
the event use count and allows the event to be deleted with the EV_DELETE event
function when the count reaches zero.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters
cb
    The control block header.

ev_code
    The EV_UNLINK event function code.

ev_id
    Specify the event.

Possible Errors
EOS_EVTID

See Also
F_EVENT, EV_DELETE
F_EVENT, EV_LINK
Chapter 6: OS-9 System Calls

F_EVENT, EV_UNLNK | F_USEMA
Unlink Usemaphore

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evunlnk_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
} f_evunlnk_pb, *F_evunlnk_pb;

Description
EV_UNLNK | F_USEMA informs the system a process is no longer using the usemaphore. This decrements the usemaphore use count and allows the usemaphore to be deleted with the EV_DELET | F_USEMA function when the count reaches zero.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

    cb
        The control block header.

    ev_code
        The EV_UNLNK | F_USEMA function code.

    ev_id
        Specify the usemaphore.

Possible Errors
EOS_EVNTID

See Also
F_EVENT, EV_DELET | F_USEMA
F_EVENT, EV_LINK | F_USEMA
Chapter 6: OS-9 System Calls

F_EVENT, EV_WAIT
Wait for Event to Occur

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evwait_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 min_val,
        max_val;
} f_evwait_pb, *F_evwait_pb;

Description
EV_WAIT waits until an event call places the value in the range between the minimum and maximum activation values. Next, the wait auto-increment (specified at creation) is added to the event variable.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

ev_code
    The EV_WAIT event function code.

ev_id
    Identify the event.

value
    A returned value. It is the actual event value prior to the set operation that activates the suspended process.
signal
A returned value. It is the signal code, if it is activated by a signal. If a process in the event queue receives a signal, it is activated even though the event has not actually occurred. The auto-increment is not added to the event variable, and an EOS_SIGNAL error is returned. Also, the event value is returned, even though it is not in range, and the caller’s intercept routine is executed.

min_val
The minimum activation value.

max_val
The maximum activation value. The event value is added to min_val and max_val, and the actual absolute 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.

Possible Errors
EOS_EVNTID

See Also
F_EVENT, EV_SIGNAL
F_EVENT, EV_WAIT
F_EVENT, EV_WAIT | F_USEMA
Acquire Ownership of a Usemaphore

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_evwait_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 min_val,
        max_val;
} f_evwait_pb, *F_evwait_pb;

Description
EV_WAIT | F_USEMA waits until a usemaphore is unowned and then claims ownership. Then, the wait auto-increment (minus 1) is applied to the usemaphore value.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
  cb
    The control block header.

  ev_code
    The EV_WAIT | F_USEMA function code.

  ev_id
    Identify the usemaphore.

  value
    A returned value. It is the actual usemaphore value prior to the application of the wait increment.
signal
A returned value. It is the signal code, if it is activated by a signal. If a process
in the useaphore queue receives a signal, it is activated even though
ownership has not actually been granted. The auto-increment is not added to
the useaphore variable, and an EOS_SIGNAL error is returned. Also, the
event value is returned, even though it is not in range, and the process’ signal
handling code is executed.

min_val
The minimum activation value. This must be 1.

max_val
The maximum activation value. This must be 1.

Possible Errors
EOS_EVNTID
EOS_USRST

See Also
F_EVENT, EV_SIGNAL | F_USEMA
F_EVENT, EV_TRYWAIT | F_USEMA
F_EVENT, EV_WAITR
Wait for Relative Event to Occur

Headers

```c
#include <types.h>
```

Parameter Block Structure

```c
typedef struct f_evwaitr_pb {
    syscb cb;
    u_int16 ev_code;
    event_id ev_id;
    int32 value;
    signal_code signal;
    u_int32 min_val,
        max_val;
} f_evwaitr_pb, *F_evwaitr_pb;
```

Description

`EV_WAITR` 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. Next, 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 absolute 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.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe

Parameters

- `cb` The control block header.
- `ev_code` The `EV_WAIT` event function code.
- `ev_id` Identify the event.
value
A returned value. It is the actual event value prior to the set operation that activates the suspended process.

signal
A returned value. It is the signal code, if it is activated by a signal. If a process in the event queue receives a signal, it is activated even though the event has not actually occurred. The auto-increment is not added to the event variable, and an EOS_SIGNAL error is returned. Also, the event value is returned, even though it is not in range, and the caller’s intercept routine is executed.

min_val
The minimum activation value.

max_val
The maximum activation value. The event value is added to min_val and max_val, and the actual absolute values are returned to the caller. If an underflow or overflow occurs on the addition, the values 0x80000000 (minimum integer) and 0xffffffff (maximum integer) are used, respectively.

Possible Errors
EOS_EVNTID

See Also
F_EVENT, EV_SIGNAL
F_EVENT, EV_WAIT
Chapter 6: OS-9 System Calls

F_EXIT
Terminate Calling Process

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_exit_pb {
    syscall cb;
    status_code status;
} f_exit_pb, *F_exit_pb

Description
F_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.

The parent can detect the death of a child process by executing F_WAIT. This returns (to the parent) the exit status passed by the child in its exit call. The shell assumes the exit status is an OS-9 error code. The exit status can also be a user-defined status value.

Processes to be called directly by the shell should only return an OS-9 error code or zero (if no error occurred).

The parent must perform an F_WAIT or an F_EXIT before the child process descriptor is returned to free memory.

F_EXIT executes as follows:
1. Close all paths.
2. Return the memory to the system.
3. Unlink the primary module, subroutine libraries, and trap handlers.
4. Free the process descriptor of any dead child processes.
5. Free the process descriptor if the parent is dead.
6. Leave the process in limbo until the parent notices the death if the parent has not executed F_WAIT.
7. If the parent is waiting, move it to the active queue and informs it of death/status.
8. Remove the child from the sibling list and free its process descriptor memory.

Only primary modules, subroutine libraries, and trap handlers are unlinked. Other modules loaded or linked by the process should be unlinked before calling F_EXIT.

Although F_EXIT closes any open paths, it ignores errors returned by I_CLOSE. Due to I/O buffering, write errors can go unnoticed when paths are left open. However, by convention, the standard I/O paths (0, 1, and 2) are usually left open.
Chapter 6: OS-9 System Calls

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

`cb`
   The control block header.

`status`
   The status code returned to the parent process.

See Also
`F_APROC`
`F_FORK`
`F_SRTMEM`
`F_UNLINK`
`F_WAIT`
`I_CLOSE`
F_FINDPD
Find Process Descriptor

Headers
#include <process.h>

Parameter Block Structure
typedef struct f_findpd_pb {
    syscb cb;
    process_id proc_id;
    Pr_desc proc_desc;
} f_findpd_pb, *F_findpd_pb;

Description
F_FINDPD converts a process number to the absolute address of its process descriptor data structure.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters
cb
    The control block header.
proc_id
    Specify the process ID.
proc_desc
    A returned value. It is the pointer to the process descriptor.

Possible Errors
EOS_IPRCID

See Also
F_ALLPRC
F RETPD
Chapter 6: OS-9 System Calls

F_FMOD
Find Module Directory Entry

Headers
#include <moddir.h>

Parameter Block Structure
typedef struct f_findmod_pb {
    syscb cb;
    u_int16 type_lang;
    Mod_dir moddir_entry;
    u_char *mod_name;
} f_findmod_pb, *F_findmod_pb;

Description
F_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.

Attributes
Operating System: OS-9
State: System and Interrupt
Threads: Safe

Parameters
cb
    The control block header.

    type_lang
    Specify the type and language of the module.

    moddir_entry
    A returned value. It is the pointer to the module directory entry.

    mod_name
    Point to the module name.

Possible Errors
EOS_BNAM
EOS_MNF

See Also
F_LINK
F_LOAD
Chapter 6: OS-9 System Calls

F_FORK
Create New Process

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_fork_pb {
    syscb cb;
    u_int16 priority,
        path_cnt;
    process_id proc_id;
    u_char *mod_name,
        *params;
    u_int32 mem_size,
        param_size;
    u_int16 type_lang;
    u_int16 orphan;
} f_fork_pb, *F_fork_pb;

Description
F_FORK creates a new process that becomes a child of the caller. It sets up the new process’ memory, MPU registers, and standard I/O paths.

The system parses the name string of the new process’ primary module (the program that is initially executed). If the program is found in the current or alternate module directory, the module is linked and executed. If the program is not found, the name string is used as the pathlist of the file to be loaded into memory. The first module in this file is linked and executed. The module must be program object code with the appropriate read and/or execute permissions to be loaded successfully.

The primary module’s header determines the process’ initial data area size. OS-9 attempts to allocate RAM equal to the required data storage size, the size of any parameters passed, and the size specified by mem_size. The RAM area must be contiguous.

The execution offset in the module header is used to set the PC to the module’s entry point.

When the shell processes a command line, it passes a copy of the command line parameters (if any) as a parameter string. The shell appends an end-of-line character to the parameter string to simplify string-oriented processing.

If one or more of these operations is unsuccessful, the fork is aborted and the caller receives an error.
F\_FORK passes the following structure (defined in <fork.h>) as a parameter to the newly-created process:

```c
typedef struct {
    process_id proc_id;    /* process ID */
    owner_id owner;         /* group/user ID */
    priority_level priority; /* priority */
    u_int16 path_count; /* number of I/O paths inherited */
    u_int32 param_size, /* size of parameters */
    mem_size; /* total initial memory allocated */
    u_char *params,    /* parameter pointer */
    *mem_end; /* top of memory pointer */
    Mh_exec mod_head;   /* primary (forked) module ptr*/
} fork_params, *Fork_params;
```

The child and parent processes execute concurrently. If the parent executes F\_WAIT immediately after the fork, it waits until the child dies before it resumes execution. A child process descriptor is returned to free memory only when the parent performs an F\_WAIT or an F\_EXIT service request.

Modules owned by a super user can execute in system state if the system-state bit in the module’s attributes is set. This should only be done when necessary because this process is not time sliced and system protection is not enabled for this process.

**Attributes**

Operating System: OS-9  
State: User and System  
Threads: Safe

**Parameters**

- **cb**  
The control block header.

- **priority**  
Specify the priority of the new process. If priority is zero, the new process inherits the same priority as the calling process.

- **path_cnt**  
Specify the number of I/O paths for the child to inherit.

- **proc_id**  
A returned value. It is the child process ID.

- **mod_name**  
Point to the module name.
params
   Point to the parameter string to pass to the new process.

mem_size
   Specify any additional stack space to allocate above the default specified in the primary module's module header.

param_size
   Specify the size of the parameter string.

type_lang
   Specify the desired type and language. If type_lang is zero, any module, regardless of type and language, may be loaded.

orphan
   If the orphan flag is non-zero, the new process executes without a parent. If orphan is zero, the new process is the child of the calling process.

Possible Errors
EOS_NORAM
EOS_PERMIT
EOS_PNNF

See Also
F_CHAIN
F_EXIT
F_WAIT
F_FORKM
Create New Process by Module Pointer

Headers
#include <module.h>

Parameter Block Structure
typedef struct f_forkm_pb {
    syscb cb;
    u_int16 priority,
        path_cnt;
    process_id proc_id;
    Mh_com mod_head;
    u_char *params;
    u_int32 mem_size,
        param_size;
    u_int16 orphan;
} f_forkm_pb, *F_forkm_pb;

Description
F_FORKM creates a new process that 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. F_FORKM assumes the module pointer is the primary module
pointer for the new process.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
        The control block header.

    priority
        Specify the priority of the new process. If priority is zero, the new process
        inherits the same priority as the calling process.

    path_cnt
        Specify the number of I/O paths for the child to inherit.
proc_id
   A returned value. It is the child process ID.

mod_head
   Point to the module header of the module to fork.

params
   Point to the parameter string to pass to the new process.

mem_size
   Specify any additional stack space to allocate above the default specified in the primary module’s module header.

param_size
   Specify the size of the parameter string.

orphan
   If the orphan flag is non-zero, the new process executes without a parent. If orphan is zero, the new process is the child of the calling process.

Possible Errors
   EOS_MNF
   EOS_NORAM
   EOS_PERMIT

See Also
   F_FORK
Headers
#include <types.h>

Parameter Block Structure
typedef struct f_gblkmp_pb {
    syscb cb;
    Mem_list start;
    u_char *buffer;
    u_int32 size,
        min_alloc,
        num_segs,
        tot_mem,
        free_mem;
} f_gblkmp_pb, *F_gblkmp_pb;

Description
F_GBLKMP 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 in the following format:
typedef struct {
    u_char   *address;    /* pointer to block */
    u_int32   size;       /* size of block */
};

Although F_GBLKMP returns the address and size of the system’s free memory blocks,
you cannot directly access these blocks. Use F_SRQMEM to request free memory
blocks.

The address and size of free RAM changes with system use. mfree and similar
utilities use F_GBLKMP to determine the status of free system memory.

The OS suffixes the array of info structures, to which buffer points, with a sentinel
as follows:
info.address    NULL
info.size       0

The OS adds this sentinel only if at least one unused info structure occupies the
buffer after processing.
Chapter 6: OS-9 System Calls

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
  The control block header.

start
  The address to begin reporting the segments.

buffer
  Point to the buffer to use.

size
  Specify the buffer size in bytes. It is also an output containing the number of unused info structures in the buffer.
  When size is 0, the service does not validate or use buffer. It also updates the following parameters on every call.

min_alloc
  A returned value. It is the minimum memory allocation size for the system.

num_segs
  A returned value. It is the number of memory fragments in the system.

tot_mem
  A returned value. It is the total RAM found by the system at startup.

free_mem
  A returned value. It is the current total free RAM available.

See Also

F_SRQMEM
F_GETMDP
Get Current and Alternate Module Directory Pathlists

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_get_mdp_pb {
    syscb cb;
    u_char *current,
        *alternate;
} f_get_mdp_pb, *F_get_mdp_pb;

Description
F_GETMDP returns pathlists to the current module directory and the alternate module directory.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
cb
    The control block header.
current
    Point to the buffer for returning the pathlist of the current module directory.
alternate
    Point to the buffer for returning the pathlist of the alternate module directory.

See Also
F_ALTMDIR
F_CHMDIR
F_GETSYS
Examine System Global Variable

Headers
#include <types.h>
#include <sysglob.h>

Parameter Block Structure
typedef struct f_getsys_pb {
  syscb cb;
  u_int32 offset,
       size;
  union {
    u_char byt;
    u_int16 wrd;
    u_int32 lng;
  } sysvar;
} f_getsys_pb, *F_getsys_pb;

Description
F_GETSYS enables a process to examine a system global variable. Consult the sysglob.h header file for a description of the system global variables. The format and contents of the system global variables may change in future releases of OS-9.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters
  cb
    The control block header.

  offset
    The variable’s offset in the system globals.

  size
    Specify the size of the variable.

  sysvar
    A union of the three sizes of variables accessible by F_GETSYS.
byt
  A byte size variable.

wrd
  A word size variable.

lng
  A long size variable.

See Also

F_SETSYS

the DEFS files section of the OS-9 Porting Guide
F_GMODDR
Get Copy of Module Directory

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_get_moddir_pb {
    syscb cb;
    u_char *buffer;
    u_liont32 count;
} f_get_moddir_pb, *F_get_moddir_pb;

Description
F_GMODDR copies the process’ current module directory into your buffer for inspection.

F_GMODDR is provided primarily for use by mdir and similar utilities. The format and contents of the module directory may change on different releases of OS-9. Therefore, you should use the output of mdir to determine the names of modules in memory.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
    cb
        The control block header.
    buffer
        Point to the buffer.
    count
        The maximum number of bytes to copy, and upon return from F_GMODDR it is the number of bytes actually copied.

Although the module directory contains pointers to each module in the system, never access the modules directly. Instead, use F_CPYMEM to copy portions of the system’s address space for inspection.

See Also
F_CPYMEM
F_GPRDBT
Get Copy of Process Descriptor Block Table

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_get_prtbl_pb {
    syscb cb;
    u_char *buffer;
    u_int32 count;
} f_get_prtbl_pb, *F_get_prtbl_pb;

Description
F_GPRDBT copies the process descriptor block table into your buffer for inspection. The proc utility uses F_GPRDBT to determine which processes are active in the system.

Attributes
Operating System: OS-9
State: User and Interrupt
Threads: Safe

Parameters

    cb
        The control block header.

    buffer
        Point to the buffer.

    count
        The maximum number of bytes to copy and upon return from F_GPRDBT it is the number of bytes actually copied.

Although F_GPRDBT returns pointers to all process descriptors, never access the process descriptors directly. Instead, use F_GPRDSC to inspect specific process descriptors.

See Also
F_GPRDSC
Chapter 6: OS-9 System Calls

F_GPRDSC
Get Process Descriptor Copy

Headers
#include <process.h>
#include <types.h>

Parameter Block Structure
typedef struct f_gprdsc_pb {
    syscb       cb;
    process_id  proc_id;
    u_char      *buffer;
    u_int32     count;
    u_int32     type;
} f_gprdsc_pb, *F_gprdsc_pb;

Description
F_GPRDSC copies the contents of a process descriptor into the specified buffer for inspection. The procs utility uses F_GPRDSC to obtain information about an existing process.

Attributes
Operating System: OS-9
State: User
Threads: Safe

Parameters
 cb
    The control block header.

 procid
    The requested process ID.

 buffer
    Point to the buffer.

 count
    The maximum number of bytes to copy, and upon return from F_GPRDSC, it is the number of bytes actually copied.

 type
    The type of descriptor to get. The valid values for type are _OS_GET_PRDESC or _OS_GET_PRSRC. _OS_GET_PRDESC returns the specified state descriptor and _OS_GET_PRSRC returns the specified resource descriptor.

Possible Errors
EOS_IPRCID
Chapter 6: OS-9 System Calls

F_ICPT
Set Up Signal Intercept Trap

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_intercept_pb {
    syscb cb;
    u_int32 (*function)();
    void *data_ptr;
} f_intercept_pb, *F_intercept_pb;

Description
F_ICPT tells OS-9 to install a signal intercept routine.

When a process executing an F_ICPT call receives a signal, the process’ intercept routine is executed, and the signal code is passed as a parameter. A signal aborts a process that has not used F_ICPT. Many interactive programs set up an intercept routine to handle keyboard abort and keyboard interrupt signals.

The intercept routine is entered asynchronously because a signal can be sent at any time, similar to an interrupt. The signal code is passed as a parameter. The intercept routine should be short and fast, such as setting a flag in the process’ data area. You should avoid complicated system calls (such as I/O). After the intercept routine has been completed, it may return to normal process execution by executing F_RTE.

Each time the intercept routine is called, the state of the processor (such as its registers) is pushed on to the user’s system stack.

Attributes
Operating System: OS-9
State: User and Interrupt
Threads: Safe

Parameters

    cb
        The control block header.

    function
        Point to the intercept routine.
data_ptr

Point to the intercept routine’s global storage. It usually contains the address of the program’s data area. The syntax for the signal handler is as follows:

```c
void usr_sighand(sig_code, sig_count)

signal_code  sig_code;    /* signal received */
u_int32      sig_count;   /* number of signals pending */
```

**See Also**

F_RTE
F_SEND
Chapter 6: OS-9 System Calls

F_ID
Get Process ID and User ID

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_id_pb {
    syscb cb;
    process_id proc_id;
    u_int16 priority,
    age;
    int32 schedule;
    owner_id user_id;
} f_id_pb, *F_id_pb;

Description
F_ID returns the caller’s process ID number, current process priority and age, scheduling constant, and owner ID. OS-9 assigns the process ID, and each process has a unique process ID. The owner ID is defined in the system password file and is used for system and file security. Several processes can have the same owner ID.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

    cb
        The control block header.

    proc_id
        A returned value. It is the current process ID number.

    priority
        A returned value. It is the priority of the current process.

    age
        A returned value. It is the age of the current process.

    schedule
        A returned value. It is the scheduling constant of the current process.
group
A returned value. It is the group number of the current process.

user
A returned value. It is the user number of the current process.

Possible Errors

EOS_BPADDR
Chapter 6: OS-9 System Calls

F_INITDATA
Initialize Static Storage from Module

.Headers
#include <module.h>

.Parameter Block Structure
typedef struct f_init_data_pb {
    syscb cb;
    Mh_com mod_head;
    u_char *data;
} f_init_data_pb, *F_init_data_pb;

.Description
F_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). Next, it
adjusts the code and data offsets.

.Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

.Parameters
    cb
        The control block header.

    mod_head
        Point to the module header.

    data
        Point to the data area.

.Possible Errors
EOS_BMHP
EOS_BMID
F_IRQ
Add or Remove Device from Interrupt Table

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_irq_pb {
    syscb cb;
    u_int16 vector,
    priority;
    void *irq_entry;
    u_char *statics;
} f_irq_pb, *F_irq_pb;

Description
F_IRQ installs an IRQ service routine into the system polling table.
OS-9 does not poll the I/O port prior to calling the interrupt service routine. Device drivers are required to determine if their device caused an interrupt.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

cb
    The control block header.

vector
    Specify the vector number of the associated interrupt.

priority
    Specify the priority. (65535 is reserved.) IRQ service routines for the same vector are placed into a polling table for the vector according to their relative priorities:
    • If priority is 0, only this device can use the vector.
    • If priority is 1, it is polled first and no other device can have a priority of one on the vector.
    • If priority is 65534, it is polled last on the vector.
**irq_entry**
Point to the IRQ service routine entry point. If `irq_entry` is zero, the call deletes the IRQ service routine.

**statics**
Point to the global static storage. `statics` must be unique to the device.

**Possible Errors**

- **EOS_VCTBSY** signifies that the vector is busy or in use.
- **EOS_PARAM** is returned if an attempt is made to delete an IRQ routine that is not installed for that interrupt.
Chapter 6: OS-9 System Calls

F_LINK
Link to Memory Module

Headers

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

Parameter Block Structure

```c
typedef struct f_link_pb {
   syscb cb;
   u_char *mod_name;
   Mh_com mod_head;
   void *mod_exec;
   u_int16 type_lang,
       attr_rev;
} f_link_pb, *F_link_pb;
```

Description

`F_LINK` searches the current and alternate module directories for a module whose
name, type, and language match the parameters.

The module’s link count keeps track of how many processes are using the module. If
the requested module is not re-entrant, only one process may link to it at a time.

If the module’s access word does not give the process read permission, the link call
fails. `F_LINK` cannot find a module whose header has been destroyed (altered or
corrupted).

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe

Parameters

- `cb`

  The control block header.

- `mod_name`

  Point to the module name. If `mod_name` is an explicit module directory pathlist
  (for example, `/usr/tony/prog`), the `mod_name` pointer is updated to point to
  the module that was successfully linked (for example, `prog`).

- `mod_head`

  A returned value. It is the address of the module’s header.
mod_exec
A returned value. It is the pointer to the absolute address of the module’s execution entry point. The module header includes this information.

type_lang
A returned value. It is the type and language of the module.

attr_rev
A returned value. It points to the attribute and revision level of the module.

Possible Errors
EOS_BNAM
EOS_MNF
EOS_MODBSY

See Also
F_LINKM
F_LOAD
F_UNLINK
F_UNLOAD
F_LINKM
Link to Memory Module by Module Pointer

Headers
#include <module.h>

Parameter Block Structure
typedef struct f_linkm_pb {
    syscb cb;
    Mh_com mod_head;
    void *mod_exec;
    u_int16 type_lang,
        attr_rev;
} f_linkm_pb, *F_linkm_pb;

Description
F_LINKM causes OS-9 to link to the module specified by mod_head.
The module’s link count keeps track of how many processes are using the module. If
the requested module is not re-entrant, only one process can link to it at a time.
If the module’s access word does not give the process read permission, the link call
fails. Link cannot find a module whose header has been destroyed (altered or
corrupted).

Attributes
Operating System:   OS-9
State:             User and System
Threads:           Safe

Parameters

    cb
        The control block header.

    mod_head
        Point to the module.

    mod_exec
        A returned value. It points to the pointer to the absolute address of the
        module’s execution entry point.

    type_lang
        The type and language of the module. If type_lang is zero, any module can be
        linked to regardless of the type and language. Upon completion, type_lang is
        updated with the type/language value from the module’s module header.

    attr_rev
        A returned value. It is the attribute and revision level of the module.
Possible Errors
EOS_BNAM
EOS_MNF
EOS_MODBSY

See Also
F_LINK
F_LOAD
F_UNLINK
F_UNLOAD
Chapter 6: OS-9 System Calls

F_LOAD
Load Module(s) from File

Headers

#include <module.h>

Parameter Block Structure

typedef struct f_load_pb {
    syscb cb;
    u_char *mod_name;
    Mh_com mod_head;
    void *mod_exec;
    u_int32 mode;
    u_int16 type_lang,
        attr_rev,
        color;
} f_load_pb, *F_load_pb;

Description

F_LOAD loads an OS-9 memory module from a disk file or a serial device (SCF) into the current module directory. When loading from a disk file as specified by mod_name pathlist, the target file is opened and one or more memory modules are read from the file into memory until an error or end of file is reached. When loading from a serial device (SCF), the Kernel attempts to load only one memory module by first reading the header of the module and then the body of the module. In either case, the path to the disk file or serial device is closed after the loading operation.

An error can indicate an actual I/O error, a module with a bad parity or CRC, or insufficient memory of the desired type.

When a module is loaded, its name is added to the calling process’ current module directory, and the first module read is linked. The parameters returned are the same as those returned by a link call and apply only to the first module loaded.

To be loaded, the file must contain a module (or modules) with a proper module header and CRC. If the file’s access mode is S_IEXEC, the file is loaded from the current execution directory. If the file’s access mode is S_IREAD, the file is loaded from the current data directory.

If any of the modules loaded belong to the super user, the file must also belong to the super user. This prevents normal users from executing privileged service requests.
### Attributes

- **Operating System:** OS-9
- **State:** User and System
- **Threads:** Safe

### Parameters

- **cb**
  - The control block header.

- **mod_name**
  - Point to the module name (pathlist or serial device name).

- **mod_head**
  - A returned value. It is the pointer to the module.

- **mod_exec**
  - A returned value. It is the pointer to the module execution entry point.

- **mode**
  - Specify the access mode. The access modes are defined in the `module.h` header file.

- **type_lang**
  - A returned value. It is the type and language of the first module loaded.

- **attr_rev**
  - A returned value. It is the attribute and revision level of the module.

- **color**
  - Specify the type of memory in which to load the modules. Modules are loaded into the highest physical memory available of the specified type.

### Possible Errors

- **EOS_MEMFUL**
- **EOS_BMID**
Headers

```c
#include <types.h>
```

Parameter Block Structure

```c
typedef struct f_makmdir_pb {
    syscb cb;
    u_char *name;
    u_int16 perm;
} f_makmdir_pb, *F_makmdir_pb;
```

Description

F_MKMDIR creates a new module directory. The name of the new module directory is relative to the current module directory.

Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe

Parameters

- **cb**
  - The control block header.

- **name**
  - Point to the name of the new module directory.

- **perm**
  - Specify the access permissions for the new module directory.

Possible Errors

- EOS_KWNMOD
- EOS_NORAM
Chapter 6: OS-9 System Calls

F_MEM
Resize Data Memory Area

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_mem_pb {
    syscb cb;
    u_char *mem_ptr;
    u_int32 size;
} f_mem_pb, *F_mem_pb;

Description
F_MEM contracts or expands the process’ data memory area. The size requested 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.

This request cannot return all of a process’ memory or deallocate the memory at its current stack pointer.

If there is adequate free memory for an expansion request, but the memory is not contiguous, F_MEM returns an error. Memory requests by other processes may have fragmented memory resulting in small, scattered blocks that are not adjacent to the caller’s present data area.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

 cb
    The control block header.

 mem_ptr
    A returned value. It is the new end of data segment plus 1.

 size
    The memory size in bytes. The actual size of the memory is returned in size. If size is zero, F_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.

Possible Errors
EOS_DELSP
EOS_MEMFUL
EOS_NORAM
F_MODADDR
Find Module Given Pointer

Headers

#include <module.h>

Parameter Block Structure

typedef struct f_modaddr_pb {
    syscb cb;
    u_char *mem_ptr;
    Mh_com mod_head;
} f_modaddr_pb, *F_modaddr_pb;

Description

F_MODADDR locates a module given a pointer to any position with the module and returns a pointer to the module’s header.

Attributes

Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

mem_ptr
    Point to any position within the module.

mod_head
    A returned value. It is the pointer to the associated module header.

Possible Errors

EOS_MNF
Chapter 6: OS-9 System Calls

F_MOVE
Move Data (Low Bound First)

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_move_pb {
    syscb cb;
    u_char *from,
    *to;
    u_int32 count;
} f_move_pb, *F_move_pb;

Description
F_MOVE is a fast block-move subroutine that copies data bytes from one address space to another, usually from system to user or vice versa. The data movement subroutine is optimized to use long moves if possible. If source and destination buffers overlap, appropriate moves (left to right or right to left) are used to avoid data loss due to incorrect propagation.

Attributes
Operating System: OS-9
State: System and Interrupt
Threads: Safe

Parameters
cb The control block header.
from Point to the source data.
to Point to the destination data.
count The byte count to copy.
F_NPROC
Start Next Process

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_nproc_pb {
    syscb cb;
} f_nproc_pb, *F_nproc_pb;

Description
F_NPROC removes the next process from the active process queue and initiates its execution. If there is no process in the queue, OS-9 waits for an interrupt and checks the active process queue again.

F_NPROC does not return to the caller.

The process calling F_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
State: System
Threads: Safe

Parameters

cb
    The control block header.

See Also
F_APROC
F_PERMIT
Allow Access to Memory Block

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_permit_pb {
    syscb cb;
    process_id pid;
    u_int32 size;
    u_char *mem_ptr;
    u_int16 perm;
} f_permit_pb, *F_permit_pb;

Description
F_PERMIT is called when a process allocates memory or links to a module to allow
the process to access a block of memory.
F_PERMIT must update SSM (System Security Module) data structures to show a
process may access the specified memory in the requested mode. F_PERMIT must also
increment the number of links to this memory area in a corresponding block count
map to keep track of the number of times the same block(s) has been granted access.
A long word (p_spuimg) is reserved in each process descriptor for use by the SSM
code. The SSM may allocate data structures for each process and keep a pointer to
these structures in p_spuimg.

- The calling process cannot use this service to permit write-only memory or to
  permit nothing (set no permissions). This service must be used to permit at least
  read-only access.
- The only user-state processes that may permit memory are those in group zero
  (super user). All others must be system-state processes.
- On systems without SSM, the result of any F_PERMIT call is success, regardless of
  the process state since all processes have full access rights to the entire memory
  space. When SSM is not active, the operating system does not validate any of
  the arguments for this call.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Parameters

cb
   The control block header.

pid
   The target process’ process identifier.

size
   The size of the memory area.

mem_ptr
   Point to the beginning of the memory area to grant access permissions.

perm
   Specify the permissions to add.

Possible Errors

EOS_BPADDR
EOS_DAMAGE
EOS_IPRCID
EOS_NORAM
EOS_PARAM
EOS_PERMIT
F_PROTECT
Prevent Access to Memory Block

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_protect_pb {
    syscb cb;
    process_id pid;
    u_int32 size;
    u_char *mem_ptr;
    u_int16 perm;
} f_protect_pb, *F_protect_pb;

Description
F_PROTECT is called when a process deallocates memory or unlinks a module to remove a process’ permission to access a block of memory.

The counts in the block count map corresponding to the memory blocks being protected must be decremented and if any block count becomes zero, the protection image must be updated to prevent access to the corresponding memory by the process.

Note the following:
- If F_PROTECT is called for a process being debugged, the protection maps of the parent process must also be updated to remove access to the allocated memory.
- The only user-state processes that may protect memory are the ones in group zero (super user). All other processes must be system-state processes.
- On systems without SSM, the result of any F_PROTECT call is success, regardless of the process state since all processes have full access rights to the entire memory space. When SSM is not active, the operating system does not validate any of the arguments for this call.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Parameters

**cb**
- The control block header.

**pid**
- Specify the process identifier for the target process.

**size**
- The size of the memory area.

**mem_ptr**
- Point to the beginning of the memory area to protect access permissions. `size` specifies the size of memory.

**perm**
- Specify the permissions to remove.

Possible Errors

- EOS_BPADDR
- EOS_IPRCID
- EOS_NORAM
- EOS_PERMIT

See Also

- F_ALLTSK
- F_PERMIT
Chapter 6: OS-9 System Calls

F_PRSNAM
Parse Path Name

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_prsnam_pb {
    syscb cb;
    u_char *name;
    u_int32 length;
    u_char delimiter,
        *updated;
} f_prsnam_pb, *F_prsnam_pb;

Description
F_PRSNAM parses a string for a valid pathlist element and returns its size. This call parses one element in a pathname, not the entire pathname. A valid pathlist element may contain the following characters:

A - Z    Upper case letters
a - z    Lower case letters
0 - 9    Numbers
.        Periods
_        Underscores
$        Dollar signs

Other characters terminate the name and are returned as the pathlist delimiter.

Several F_PRSNAM calls are needed to process a pathlist with more than one name. F_PRSNAM terminates a name when it detects a delimiter character. Usually, pathlists must be terminated with a null byte.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Parameters

cb
  The control block header.

name
  Point to the name string.

length
  A returned value. It is the length of the pathlist element.

delimiter
  A returned value. It is the pathlist delimiter.

updated
  A returned value. It is a the pointer to the first character of name.

Possible Errors

EOS_BNAM

See Also

F_CMPNAM
F_RELLK
Release Ownership of Resource Lock

Headers
#include <lock.h>

Parameter Block Structure
typedef struct f_rellk_pb {
    syscb cb;
    lock_id lid;
} f_rellk_pb, *F_rellk_pb;

Description
F_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.

Refer to Chapter 6 for more information about resource locks.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

    cb
The control block header.

    lid
The lock identifier of the lock to release.

Possible Errors
EOS_LOCKID

See Also
F_ACQLK
F_CAQLK
F_CRLK
F_DELLK
F_WAITLK
Headers
#include <types.h>

Parameter Block Structure
typedef struct f_rtnprc_pb {
    syscb cb;
    process_id proc_id;
} f_rtnprc_pb, *F_rtnprc_pb;

Description
F_RETPD deallocates a process descriptor previously allocated by F_ALLPRC. You must ensure the process’ system resources are returned prior to calling F_RETPD.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters
cb
    The control block header.
proc_id
    Identify the process descriptor.

Possible Errors
EOS_IPRCID

See Also
F_ALLPRC
Chapter 6: OS-9 System Calls

F_RTE
Return from Interrupt Exception

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_rte_pb {
    syscb cb;
    u_int32 mode;
} f_rte_pb, *F_rte_pb;

Description
F_RTE terminates a process' signal intercept routine and continues executing the main program. However, if unprocessed signals are pending, the intercept routine is re-executed until the queue of signals is exhausted before returning to the main program.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
        The control block header.

    mode
        Currently unused. Value must be 0 for future compatibility.

See Also
F_ICPT
Chapter 6: OS-9 System Calls

F_SEND
Send Signal to Another Process

Headers

```
#include <types.h>
```

Parameter Block Structure

```
typedef struct f_send_pb {
    syscb cb;
    process_id proc_id;
    signal_code signal;
} f_send_pb, *F_send_pb;
```

Description

F_SEND sends a signal to a specific process. A process may send the same signal to multiple processes of the same group/user ID by passing 0 as the receiving process’ ID number. For example, the OS-9 shell command, `kill 0`, unconditionally aborts all processes with the same group/user ID, except the shell itself. This is an effective but dangerous tool to get rid of unwanted background tasks.

If an attempt is made to send a signal to a process with a signal pending, the signal is placed in the process’ FIFO signal queue. If the process is in the signal intercept routine when it receives a signal, the new signal is processed when F_RTE is executed by the target process.

If the destination process for the signal is sleeping or waiting, it is activated to process the signal. The signal processing intercept routine is executed, if it exists (see F_ICPT). Otherwise, the signal aborts the destination process and the signal code becomes the exit status (see F_WAIT).

The wake-up signal is an exception. It activates a sleeping process but does not invoke the signal intercept routine. The wake-up signal does not abort a process that has not made an F_ICPT call. Wake-up signals never queue and have no effect on active processes in user state. User programs should avoid using the wake-up signal since it is used by the system to activate sleeping processes. Signal codes are defined as follows:

**Table 6-4. Signal Codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_WAKE</td>
<td>1</td>
<td>Wake up process</td>
</tr>
<tr>
<td>S.Quit</td>
<td>2</td>
<td>Keyboard abort</td>
</tr>
<tr>
<td>S_INT</td>
<td>3</td>
<td>Keyboard interrupt</td>
</tr>
<tr>
<td>S_KILL</td>
<td>4</td>
<td>System abort (unconditional)</td>
</tr>
<tr>
<td>S_HANGUP</td>
<td>5</td>
<td>Hang-up</td>
</tr>
<tr>
<td>6-19</td>
<td></td>
<td>Reserved for future use by Microware (globally definable)</td>
</tr>
</tbody>
</table>
The `S_KILL` signal may only be sent to processes with the same group ID as the sender. Super users may kill any process.

**Attributes**
- **Operating System:** OS-9
- **State:** User, System, and Interrupt
- **Threads:** Safe

**Parameters**
- `cb`  
  The control block header.
- `proc_id`  
  The process ID number for the intended receiver. A `proc_id` of zero specifies all processes with the same group/user ID.
- `signal`  
  Specify the signal code to send.

**Possible Errors**
- EOS_IPRCID
- EOS_SIGNAL
- EOS_USIGP

**See Also**
- F_ICPT
- F_RTE
- F_SIGMASK
- F_SLEEP
- F_WAIT

---

**Table 6-4. Signal Codes (Continued)**

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-25</td>
<td>Reserved for Microware for specific platforms (locally definable)</td>
<td></td>
</tr>
<tr>
<td>26-31</td>
<td>User definable for specific platforms</td>
<td></td>
</tr>
<tr>
<td>32-127</td>
<td>Reserved for Microware (Ultra C)</td>
<td></td>
</tr>
<tr>
<td>128-191</td>
<td>Reserved for Microware for specific platforms (locally definable)</td>
<td></td>
</tr>
<tr>
<td>192-255</td>
<td>Reserved for Microware (globally definable)</td>
<td></td>
</tr>
<tr>
<td>256- 4294967295</td>
<td>User definable</td>
<td></td>
</tr>
</tbody>
</table>
F_SETCRC
Generate Valid CRC in Module

Headers
#include <module.h>

Parameter Block Structure
typedef struct f_setcrc_pb {
    syscb cb;
    Mh_com mod_head;
} f_setcrc_pb, *F_setcrc_pb;

Description
F_SETCRC updates the header parity and CRC of a module in memory. The module may be an existing module known to the system, or simply an image of a module that is subsequently written to a file. The module must have the correct size and sync bytes; other parts of the module are not checked.

The module image must start on a longword address or an exception may occur.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
    cb
        The control block header.

    mod_head
        Point to the module.

Possible Errors
EOS_BMID

See Also
F_CRC
F_SETSYS
Set or Examine OS-9 System Global Variables

Headers
#include <sysglob.h>

Parameter Block Structure
typedef struct f_setsys_pb {
    syscb cb;
    u_int32 offset,
        size;
    union {
        u_char byt;
        u_int16 wrd;
        u_int32 lng;
    } sysvar;
} f_setsys_pb, *F_setsys_pb;

Description
F_SETSYS changes or examines a system global variable. These variables have a d_ prefix in the system header file library sysglob.h. Consult the DEFS files for a description of the system global variables.

Only super users may change system variables. You can examine and change any system variable, but typically, only d_minpty and d_maxage are changed. Consult Chapter 1 for an explanation of these variables.

⚠️ Super users must be extremely careful when changing system global variables.

The system global variables are OS-9’s data area. They are highly likely to change from one release to another. You may need to relink programs using this system call to be able to run on future versions of OS-9.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe
Parameters

**cb**
The control block header.

**offset**
The offset to the system globals.

**size**
Specify the size of the target variable and which union variable is to be used to set the target global variable.

**sysvar**
A union of the three sizes of variables accessible by \texttt{F_SETSYS}.

**byt**
The byte size variable.

**wrd**
The word size variable.

**lng**
The long size variable.

**EXAMPLE**

```c
#include <stdio.h>
#include <sysglob.h>

main() {
    Sysglobbs sg;
    glob_buff buffer;
    error_code err;

    buffer.wrd = 100;
    if ((err = _os_setsys(OFFSET(Sysglobbs, d_minpty), sizeof(sg->d_minpty), buffer)) != 0)
        printf("Failed to set the system minimum priority (d_minpty)\n");
    else
        printf("Set the system minimum priority (d_minpty) to %d\n", buffer.wrd);

    exit(err);
}
```

**Possible Errors**

- **EOS_PARAM**
- **EOS_PERMIT**

**See Also**

- \texttt{F_GETSYS}
F_SIGLNGJ
Set Signal Mask Value and Return on Specified Stack Image

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_siglngj_pb {
    syscb cb;
    void *usp;
    u_int16 siglvl;
} f_siglngj_pb, *F_siglngj_pb;

Description
F_SIGLNGJ allows processes to perform longjump() operations from their signal
intercept routines and unmask signals in one operation.

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

The operating system takes appropriate precautions to verify the memory location
pointed to by the usp variable is accessible to the process and to ensure 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 6-1.

![Figure 6-1. F_SIGLNGJ Required Stack Image](image-url)

- High Memory
- Low Memory

processor context

FPU context

User Stack Pointer (usp variable)
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 as 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 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, containing the size of the FPU context.

**Attributes**

Operating System: OS-9  
State: User  
Threads: Safe

**Parameters**

- cb  
  The control block header.

- usp  
  Point to the new process stack image.

- siglvl  
  The new signal level value.

**Possible Errors**

EOS_PARAM

**See Also**

F_SEND  
F_SIGMASK  
F_SLEEP  
F_WAIT
F_SIGMASK
Mask or Unmask Signals During Critical Code

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_sigmask_pb {
    syscb cb;
    u_int32 mode;
} f_sigmask_pb, *F_sigmask_pb;

Description
F_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 non-zero, the signal is queued until the process mask becomes zero. The process’ signal intercept routine is executed with signals inherently masked. New processes begin with a signal mask value of zero (not masked).

Two exceptions to this rule are the S_KILL and S_WAKE signals. S_KILL terminates the receiving process, regardless of the state of its mask. S_WAKE ensures the process is active, but does not queue. When a process makes an F_SLEEP or F_WAIT system call, its signal mask is automatically cleared. If a signal is already queued, these calls return immediately to the intercept routine.

By doing additions and subtractions (instead of merely just setting a flag), this service allows the OS and the process in question to nest the masking and unmasking of multiple signals. Also, since a process may want to receive signals without nesting back out through a bunch of F_SIGMASK calls, the OS provides three ways for clearing the mask (i.e., nesting level): F_SIGMASK with a "mode" argument of zero, F_SLEEP, and F_WAIT.

This service returns the EOS_PARAM error code whenever the calling process specifies a "mode" argument other than negative one, zero, or one (i.e., -1, 0, or 1). Signals are analogous to hardware interrupts and should be masked sparingly. Keep intercept routines as short and fast as possible.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Parameters

**cb**
The control block header.

**mode**
The process signal level.

**Table 6-5.**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The signal mask is cleared.</td>
</tr>
<tr>
<td>1</td>
<td>The signal mask is incremented.</td>
</tr>
<tr>
<td>-1</td>
<td>The signal mask is decremented.</td>
</tr>
</tbody>
</table>

**Possible Errors**

EOS_PARAM

**See Also**

F_SEND
F_SLEEP
F_WAIT
F_SIGRESET
Reset Process Intercept Routine Recursion Depth

Headers
#include <signal.h>

Parameter Block Structure
typedef struct f_sigrst_pb {
    syscall cb;
} f_sigrst_pb, *F_sigrst_pb;

Description
F_SIGRESET should be used whenever a program uses a longjmp() to get out of an
intercept routine or un masks signals in an intercept service routine with the intent
of never using the F_RTE call to return.

    if(setjmp(x) != 0) {
        _os_sigreset();
        _os_sigmask(-1);
    }

Under normal circumstances, OS-9 keeps the state of the main process on the
system stack while a signal intercept routine executes. However, if the signals are
unmasked during the intercept routine, a subsequent signal causes the current state
to be stacked on the user’s stack.

This does not happen in simple cases, but if the intercept routine un masks signals or
uses a longjmp() call and then un masks signals, states are placed on the user’s stack.
There is no functional difference, and if the code actually expects to return through
the nested intercept routines with multiple F_RTE calls, the states must be left where
they are.

If the code uses a longjmp() call to leave the intercept routine it implicitly clears the
saved context off the stack. The kernel performs best if the code tells the kernel to
remove the context through a F_SIGRESET call.

Attributes
Operating System:   OS-9
State:              User
Threads:            Safe

Parameters

    cb
        The control block header.

See Also
F_ICPT
F_RTE
F_SIGRS
Resize Process Queue Block Parameter Block

Headers
#include <srvcb.h>

Parameter Block Structure
typedef struct f_sigrs_pb {
    syscb cb;
    u_int32 signals;
} f_sigrs_pb, *F_sigrs_pb;

Description
F_SIGRS allows a process to change the maximum number of signals queued on its behalf.

You can use this call 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.

This service returns EOS_PARAM if the user requests a signal-queue size of zero (while the OS has no signals pending for this process) or a signal-queue size less than the number of maximum signals (e.g., trying to resize the queue to hold only five signals when the OS has one signal pending for a process whose maximum signal count is ten).

This service returns EOS_NORAM if the process requests a queue whose size is larger than available memory.

This service does not allow the caller to set the queue's size to zero. However, the caller (if and only if there are no signals pending) can use this service to decrease the size of the queue (even down to one). If there are pending signals, however, then the value for signals must be greater than or equal to the maximum number of signals that the process' queue can hold.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Chapter 6: OS-9 System Calls

Parameters

**cb**
The control block header.

**signals**
The new maximum number of signals.

Possible Errors

EOS PARAM
EOS_NORAM
EOS_DAMAGE

See Also

F_SIGRESET
Chapter 6: OS-9 System Calls

F_SLEEP
Put Calling Process to Sleep

**Headers**

```c
#include <types.h>
```

**Parameter Block Structure**

```c
typedef struct f_sleep_pb {  
    syscb cb;  
    u_int32 ticks;  
    signal_code signal;  
} f_sleep_pb, *F_sleep_pb;
```

**Description**

F_SLEEP deactivates the calling process until the requested number of ticks have elapsed.

You cannot use F_SLEEP to time more accurately than \( \pm 1 \) tick because it is not known when the F_SLEEP request was made during the current tick.

A sleep of one tick is effectively a request to surrender the current time slice. The process is immediately inserted into the active process queue and resumes execution when it reaches the front of the queue.

A sleep of two or more \( (n) \) ticks inserts the process in the active process queue after \( (n-1) \) ticks occur and resumes execution when it reaches the front of the queue. The process is activated before the full time interval if a signal (S_WAKE) is received. Sleeping indefinitely is a good way to wait for a signal or interrupt without wasting CPU time.

The duration of a tick is system dependent and may be determined using F_TIME system call. If the high order bit of the ticks parameter is set, the low 31 bits are interpreted as \( \frac{1}{256} \) second and converted to ticks before sleeping. This allows program delays to be independent of the system’s clock rate.

This function does not return any error code if the operating system cannot wait for the requested time due to an overflow when converting a time from 256ths-of-a-second into clock ticks. This only occurs if you specify a time in 256ths-of-a-second and the system clock ticks occur at a rate greater than 512 ticks-per-second. If an overflow occurs, the operating system waits for the longest delay possible.

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.
Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
The control block header.

ticks
The length of time to sleep in ticks/second.
- 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.

signal
A returned value. It is the last signal the process received. signal is returned to the calling process at the completion of the sleep.
- If signal is zero, the process slept for the time specified by ticks.
- If signal is non-zero, the number corresponds to the signal that awoke the process.

Possible Errors
EOS_NOCLK

See Also
F_SEND
F_TIME
F_WAIT
Headers

#include <types.h>

Parameter Block Structure

typedef struct f_slink_pb {
    syscb cb;
    u_int16 sub_num;
    u_char *mod_name;
    void *lib_exec;
    u_char *mem_ptr;
    Mh_com *mod_head;
} f_slink_pb, *F_slink_pb;

Description

Subroutine libraries provide a convenient way to link to a standard set of routines at execution time. Use of subroutine libraries keeps user programs small and automatically updates programs using the subroutine code if it is changed. This is accomplished without recompiling or relinking the program itself. Most Microware utilities use one or more subroutine libraries.

F_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 address.

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.

The return value in the case of an error is -1, even though the type is a pointer and a null is more common.

Attributes

Operating System: OS-9
State: User and System
Threads: Safe
Parameters

*cb*
   The control block header.

*sub_num*
   The subroutine number.

*mod_num*
   Point to the name of the subroutine module.

*lib_exec*
   A returned value. It points to the subroutine entry point.

*mem_ptr*
   A returned value. It points to the subroutine static memory.

*mod_head*
   A returned value. It points to the module header.

Possible Errors

EOS_BPNAM
EOS_ISUB
EOS_NORAM
EOS_PERMIT

See Also

F_TLINK
Chapter 6: OS-9 System Calls

F_SLINKM
Link to Subroutine Module by Module Pointer

Headers

#include <types.h>

Parameter Block Structure

typedef struct f_slinkm_pb {
    syscb cb;
    u_int16 sub_num;
    Mh_com mod_head;
    void *lib_exec;
    u_char *mem_ptr;
} f_slinkm_pb, *F_slinkm_pb;

Description

F_SLINKM is passed a pointer to the subroutine library module to install. If a library already exists for the specified subroutine number, an error is returned. If static storage is required for the subroutine library, it is allocated and initialized.

Attributes

Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

sub_num
    The subroutine number.

mod_head
    Point to the module header.

lib_exec
    A returned value. It points to the subroutine entry point.

mem_ptr
    A returned value. It points to the subroutine static memory.

Possible Errors

EOS_NORAM
EOS_PERMIT

See Also

F_TLINKM
F_SPRIOR
Set Process Priority

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_setpr_pb {
    syscb cb;
    process_id proc_id;
    u_int16 priority;
} f_setpr_pb, *F_setpr_pb;

Description
F_SPRIOR 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.

Two system global variables affect task switching.

• d_minpty is the minimum priority a task must have for OS-9 to age or execute it.
• d_maxage is the cutoff aging point.

These variables are initially set in the Init module.

A small change in relative priorities has a tremendous effect. For example, if two processes have the priorities 100 and 200, the process with the higher priority runs 100 times before the low priority process runs at all. In actual practice, the difference may not be this extreme because programs spend a lot of time waiting for I/O devices.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

cb
    The control block header.

proc_id
    The process ID.

priority
    Specify the new priority. 65535 is the highest priority; 0 is the lowest.

Possible Errors
EOS_IPRCID
### Headers

```c
#include <types.h>
```

### Parameter Block Structure

```c
typedef struct f_srqmem_pb {
    syscb cb;
    u_char *mem_ptr;
    u_int32 size;
    u_int16 color;
} f_srqmem_pb, *F_srqmem_pb;
```

### Description

**F_SRQMEM** allocates a block of a specific type of memory.

If more than one memory area has the same priority, the area with the largest total free space is searched first. This allows memory areas to be balanced (contain approximately equal amounts of free space).

The requested number of bytes is rounded up to a system defined blocksize (currently 16 bytes). **F_SRQMEM** is useful for allocating I/O buffers and any other semi-permanent memory. The memory always begins on an even boundary.

Memory types or color codes are system dependent and may be arbitrarily assigned by the system administrator. Microware reserves values below 256 for future use.

![Warning]

Do not use **F_SRQMEM** from Interrupt Service Routines.

The byte count of allocated memory and the pointer to the block allocated must be saved if the memory is ever to be returned to the system.

### Attributes

- Operating System: OS-9
- State: User and System
- Threads: Safe

### Parameters

- `cb`
  
  The control block header.

- `mem_ptr`
  
  Point to the allocated memory block.
size
Specify the byte count of the requested memory. If \texttt{size} is \texttt{−1}, the largest block of free memory of the specified type is allocated to the calling process. Upon completion of the service request, \texttt{size} contains the actual size of the memory block allocated.

color
Specify the memory type.

- If \texttt{color} is non-zero, the search is restricted to memory areas of that color. The area with the highest priority is searched first.
- If \texttt{color} is zero, the search is based only on priority. This allows you to configure a system such that fast on-board memory is allocated before slow off-board memory. Areas with a priority of zero are excluded from the search.

Possible Errors
\texttt{EOS_MEMFUL}
\texttt{EOS_NORAM}

See Also
\texttt{F_SRTMEM}
Headers

#include <types.h>

Parameter Block Structure

typedef struct f_srtmem_pb {
    syscb cb;
    u_char *mem_ptr;
    u_int32 size;
} f_srtmem_pb, *F_srtmem_pb;

Description

F_SRTMEM deallocates memory when it is no longer needed. The returned number of bytes is rounded up to a system defined blocksize before returning the memory. Rounding occurs identically to that performed by F_SRQMEM.

In user state, the system keeps track of memory allocated to a process and all blocks not returned are automatically deallocated by the system when a process terminates.

In system state, the process must explicitly return its memory.

Attributes

Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

mem_ptr
    Point to the memory block to return.

size
    Specify the byte count of the returned memory.

Possible Errors

EOS_BPADDR

See Also

F_MEM
F_SRQMEM
F_SSPD
Suspend Process

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_sspd_pb {
    syscb cb;
    process_id proc_id;
} f_sspd_pb, *F_sspd_pb;

Description
F_SSPD temporarily suspends a process. A super user (group ID zero) may suspend any process. A non-super user can only suspend processes with the same user ID. The process may be reactivated with F_APROC.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters
    cb
        The control block header.

    proc_id
        Identify the target process.

Possible Errors
EGS_IPRCID

See Also
F_APROC
Chapter 6: OS-9 System Calls

F_SSVC
Service Request Table Initialization

Headers
#include <types.h>
#include <svctbl.h>

Parameter Block Structure
typedef struct f_ssvc_pb {
    syscb cb;
    u_int32 count;
    u_int16 state_flag;
    void *init_tbl,
        *params;
} f_ssvc_pb, *F_ssvc_pb;

Description
F_SSVC adds or replaces service requests in OS-9’s user and privileged system service request tables.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

    cb
        The control block header.

    count
        A count of the entries in the initialization table.

    state_flag
        Specify whether user or system state tables, or both, are updated.
        • If state_flag is 1, only the user table is updated.
        • If state_flag is 2, only the system table is updated.
        • If state_flag is 3, both the system and user tables are updated.
init_tbl
Point to the initialization table. An example initialization table might look like this:

```c
error_code printmsg(), service();
svctbl sysealls[] =
{
    {F_PRINT, printmsg},
    {F_SERVICE, service}
};
```

params
May be a pointer to anything, but is intended to be a pointer to global static storage. Whenever a system call is executed, the params data pointer is passed automatically.

The following structure definition of the initialization table is located in svctbl.h:

```c
#if !defined(_TYPES_H)
#include <types.h>
#endif
#define USER_State 1   /* user-state service routine flag */
#define SYSTEM_State 2  /* system-state service routine flag */
/* service routine initialization table structure. */
typedef struct {
    u_int16   fcode;        /* system call function code */
    u_int32    (*service)(); /* service routine pointer */
    u_int32   attr;         /* attributes of system call (reserved for future use) */
    u_int16   ed_low,       /* low bound of acceptable service call edition */
           ed_high       /* upper bound of edition */
} svctbl, *Svctbl;
#endif
```
F_STIME
Set System Date and Time

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_setime_pb {
    syscb cb;
    u_int32 time;
} f_setime_pb, *F_setime_pb;

Description
F_STIME sets the current system time and starts the system real-time clock to produce time slice interrupts. F_STIME puts the time in the system static storage area and links the clock module. If the link is successful, the clock initialization routine is called.

The clock module has three responsibilities:
1. Sets up hardware-dependent functions to produce system tick interrupts. This could include moving the new time into the hardware.
2. Installs a service routine to clear the interrupt when a tick occurs.
3. The interrupt service routine must call through to the kernel’s tick routine to allow the kernel to keep accurate time in software. The address to the kernel’s tick routine is provided by the kernel via the clock module’s static storage structure when the kernel initializes the clock module.

The OS-9 kernel keeps track of the current time in software, which enables clock modules to be small and simple. Some OS-9 utilities and functions expect the clock to have the correct time. Therefore, you should run F_STIME whenever the system is started. This is usually done in the system startup file.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe
Parameters

\texttt{cb}  
  The control block header.

\texttt{time}  
  Specify the time. The time is stored as the number of seconds since 1 January 1970 Greenwich Mean Time.
  
  The time is not validated in any way. If \texttt{time} is zero on systems with a battery-backed clock, the actual time is read from the real-time clock.

Possible Errors

\texttt{EOS_MNF}
\texttt{EOS_NOCLK}
\texttt{EOS_NORAM}

See Also

\texttt{F\_TIME}
Chapter 6: OS-9 System Calls

F_STRAP
Set Error Trap Handler

Headers
#include <types.h>
#include <settrap.h>
#include <regs.h>

Parameter Block Structure
typedef struct f_strap_pb {
    syscb cb;
    u_int32 *excpt_stack;
    Strap init_tbl;
} f_strap_pb, *F_strap_pb;

typedef struct strap {
    u_int32 vector;
    u_int32 (*routine)();
} strap, *Strap;

Description
F_STRAP enables user-state programs to catch exceptions such as illegal instruction or divide-by-zero. The exceptions that may be trapped are processor-dependent.

F_STRAP enters process-local error trap routine(s) into the process descriptor dispatch table. If a routine for a particular exception already exists, it is replaced.

If a user routine is not provided and one of these exceptions occurs, the program is aborted.

When a user's exception routine is executed, it is passed the following information.
void errtrap(
    u_int32 vector_errno,/* error number of the vector */
    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.
Processor-Specific Variable Arguments

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 */
  va_arg(vp, u_int32); /* exception r9 */
  ```

- **MIPS3000 and MIPS32**
  
  ```c
  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**
  
  ```c
  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**
  
  ```c
  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**
  
  ```c
  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**
  
  ```c
  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**
  
  ```c
  va_arg(vp, u_int32); /* exception stack pointer */
  va_arg(vp, u_int32); /* exception %eax */
  ```
Disable error exception handlers by calling `F_STRAP` with an initialization table specifying 0 as the address of the routine(s) to remove. For example, the table below removes user routines for data and instruction access exceptions on a PowerPC processor.

```c
strap errtab[] = {
    {STRAP_DATA, 0},
    {STRAP_INSTR, 0},
    {~0, NULL}
};
```

⚠️ Be careful not to let an exception happen in an exception handling routine. Exception handling routines are usually not re-entrant.

**Attributes**
- Operating System: OS-9
- State: User
- Threads: Safe

**Parameters**
- `cb` The control block header.
- `excpt_stack` Pointer to the stack to use if an exception occurs. If `excpt_stack` is zero, `F_STRAP` uses the stack pointer at the time of the exception.
- `init_tbl` Pointer to the service request initialization table. An initialization table might appear as follows:

```c
strap errtab[] = {
    {STRAP_DATA, errtrap},
    {STRAP_INSTR, errtrap},
    {~0, NULL}
};
```

**Possible Errors**
- `EOS_TRAP`

**See Also**
- `F_ABORT`
Headers

#include <types.h>

Parameter Block Structure

typedef struct f_setuid_pb {
    syscb cb;
    owner_id user_id;
} f_setuid_pb, *F_setuid_pb;

Description

F_SUSER changes the current user ID to user_id.

The following restrictions apply to F_SUSER:

- Users with group ID zero may change their IDs to anything.
- A primary module owned by a group zero user may change its ID to anything.
- Any primary module may change its user ID to match the module’s owner.

All other attempts to change the user ID number return an EOS_PERMIT error.

Attributes

Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

    cb
    The control block header.

    user_id
    The desired group/user ID number.

Possible Errors

EOS_PERMIT
Chapter 6: OS-9 System Calls

F_SYSDBG
Call System Debugger

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_sysdbg_pb {
    syscb cb;
    void *param1,
    *param2;
} f_sysdbg_pb, *F_sysdbg_pb;

Description
F_SYSDBG calls the system level debugger, if one exists. This allows you to debug system-state routines, such as device drivers. The caller defines the parameters to this service request to values useful in debugging. For example, a parameter could be a pointer to a critical data structure.

When the system level debugger is active, it runs in system state and effectively stops timesharing. F_SYSDBG can only be called by users in group zero. Never use this call when other users are on the system.

The break utility calls F_SYSDBG.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
    cb
    The control block header.

    param1 and param2
    Parameters passed to the debugger. These are currently not used.

Possible Errors
EOS_PERMIT
Chapter 6: OS-9 System Calls

F_SYSID
Return System Identification

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_sysid_pb {
    syscb cb;
    u_int32 oem,
        serial,
        mpu_type,
        os_type,
        fpu_type;
    int32 time_zone
    u_int32 resv1,
    resv2;
    u_char *sys_ident,
        *copyright,
        *resv3;
} f_sysid_pb, *F_sysid_pb;

Description
F_SYSID returns information about the system.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

    cb
    The control block header.

    oem
    The OEM identification number.

    serial
    The copy serial number.

    mpu_type
    The processor identifier (for example 80386).
os_type
   The kernel (OS) MPU configuration.

fpu_type
   The floating-point processor identifier (for example 80387).

time_zone
   The system time zone in minutes offset from Greenwich Mean Time (GMT).

resv1, resv2, and resv3
   Reserved pointers.

sys_ident
   Point to a buffer for the system identification message.

copyright
   Point to a buffer for the copyright message.
F_THEXIT
Exit a Thread

Headers
#include <threads.h>

Parameter Block Structure
typedef struct f_thexit_pb {
    syscb          cb;
    error_code     status;
} f_thexit_pb, *F_thexit_pb;

Description
F_THEXIT causes the calling thread to exit. If the calling program is not multi-threaded, the EOS_PERMIT error is returned.
If successful, F_THEXIT does not return to the caller.
Threads created via pthread_create() should not use this call. Doing so results in instability and loss of resources for the process.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

    cb
    System call control block.

    status
    Exit status.

See Also
F_THFORK
Chapter 6: OS-9 System Calls

F_THFORK
Fork a Thread

Headers
#include <threads.h>

Parameter Block Structure
typedef struct f_thfork_pb {
    syscb            cb;
    thread_t         thread_id;
    thread_attr_t    attr;
    void             *stack_top;
    void             *start_addr;
    void             *arg;
    void             *data;
    u_int32          stack_size;
} f_thfork_pb, *F_thfork_pb;

Description
F_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().

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
cb
    System call control block.

attr
    Input: Thread attribute structure.

*stack_top
    Input: Thread's initial stack pointer.

*start_addr
    Input: Thread's initial execution address.

*arg
    Input: Argument passed to thread.
*dataInput:
    Thread specific data pointer.

stack_size
Input/output:
    Size of stack to allocate/allocated.

See Also
F_THEXIT
F_THREAD
Chapter 6: OS-9 System Calls

F_THREAD
Set Thread Parameters

Headers
#include <funcs.h>
#include <threads.h>

Parameter Block Structure
typedef struct f_thread_pb {
    syscb cb;
    u_int32 code;
    thread_t thread_id;
    void *pb;
} f_thread_pb, *F_thread_pb;

Description
F_THREAD sets thread parameters for the thread specified by thread_id.
If code is TH_TSDATA, arg is the thread specific data pointer.
Threads created via pthread_create() should not use this call. Doing so results in
instability and loss of resources for the process.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
cbS system call control block.

codeT thread action code: TH_ORPHAN, etc.

thread_id Thread ID if applicable.

*pb Additional parameters.

See Also
F_THEXIT
F_THFORK
Chapter 6: OS-9 System Calls

F_TIME
Get System Date and Time

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_gettime_pb {
    syscb cb;
    u_int32 time,
    ticks;
} f_gettime_pb, *F_gettime_pb;

Description
F_TIME returns the current system time in the number of seconds since 1 January 1970 Greenwich Mean Time.

F_TIME returns a date and time of zero (with no error) if no previous call to F_STIME has been made. A tick rate of zero indicates the clock is not running.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

 cb
    The control block header.

time
    A returned value. It is the current time.

ticks
    Contains the following:
    • The clock tick rate in ticks per second is returned in the most significant word.
    • The least significant word contains the current tick.

See Also
F_STIME
Chapter 6: OS-9 System Calls

F_TLINK
Install System State Trap Handling Module

Headers
#include <module.h>

Parameter Block Structure
typedef struct f_tlink_pb {
    syscb cb;
    u_int16 trap_num;
    u_char *mod_name;
    void *lib_exec,
        *mod_head,
        *params;
    u_int32 mem_size;
} f_tlink_pb, *F_tlink_pb;

Description
Trap handlers enable a program to execute privileged (system state) code without running the entire program in system state. Trap handlers only run in system state. F_TLINK attempts to link or load the module specified by mod_name. If the link/load is successful, F_TLINK installs a pointer to the module in the user’s process descriptor for subsequent use in trap calls. If a trap module already exists for the specified trap code, an error is returned. If static storage is required for the trap handler, OS-9 allocates and initializes it.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

trap_num
    Specify the user trap number (1 through 15).

mod_name
    Point to the name of the trap module. If mod_name is zero or points to a null string, the trap handler is unlinked.
lib_exec
  Point to the pointer to the trap execution entry point.

mod_head
  Point to the pointer to the trap module.

params
  A reserved field.

mem_size
  Specify the additional memory size to be allocated for the trap modules static data area.

Possible Errors
EPS_ITRAP
EPS_MNF
EPS_NORAM
EPS_PERMIT

See Also
F_SLINK
**F_TLINKM**

Install User Trap Handling Module by Module Pointer

**Headers**

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

**Parameter Block Structure**

```c
typedef struct f_tlinkm_pb {
    syscb cb;
    u_int16 trap_num;
    Mh_com mod_head;
    void *lib_exec;
    void *params;
    u_int32 mem_size;
} f_tlinkm_pb, *F_tlinkm_pb;
```

**Description:**

`F_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.

**Attributes**

- Operating System: OS-9
- State: User and System
- Threads: Safe

**Parameters**

- `cb` The control block header.
- `trap_num` Specify the user trap number (0 through 15).
- `mod_head` Point to the module header.
- `lib_exec` Point to the trap execution entry point.
- `params` A reserved field.
mem_size
   Specify the additional memory size to be allocated for the trap module’s static
   data area.

Possible Errors
EOS_ITRAP
EOS_NORAM
EOS_PERMIT

See Also
F_TLINK
F_UACCT
User Accounting

Headers
#include <types.h>

Parameter Block Structure

typedef struct f_uacct_pb {
    syscb cb;
    u_int16 acct_code;
    Pr_desc proc_desc;
} f_uacct_pb, *F_uacct_pb;

Description

F_UACCT provides a means for users to set up an accounting system. The kernel calls
F_UACCT whenever it forks or exits a process. Therefore, F_UACCT provides a
mechanism for users to keep track of system operators.

To install a handler for this service request, use the F_SSVC system call 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 F_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.

Attributes

- Operating System: OS-9
- State: User, System, and Interrupt
- Threads: Safe

Parameters

cb
The control block header.

acct_code
The operation identifier. This is usually a system call function code.

proc_desc
Point to the current process descriptor.

Possible Errors

EOS_UNKSVC (This error should be ignored.)

See Also

F_SSVC
F_UNLINK
Unlink Module by Address

Headers
#include <module.h>

Parameter Block Structure
typedef struct f_unlink_pb {
    syscb cb;
    Mh_com mod_head;
} f_unlink_pb, *F_unlink_pb;

Description
F_UNLINK notifies OS-9 the calling process no longer needs a module. The module’s link count is decremented. When the link count equals zero (−1 for sticky modules), the module is removed from the module directory and its memory is deallocated. When several modules are loaded together as a group, they are only removed when the link count of all modules in the group are zero (−1 for sticky modules).

Some modules cannot be unlinked; for example, device drivers in use and all modules included in the bootfile.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
    The control block header.

mod_head
    Point to the module header.

Possible Errors
EOS_MODBSY

See Also
F_LINK
F_UNLOAD
F_UNLOAD
Unlink Module by Name

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_unload_pb {
    syscb cb;
    u_char *mod_name;
    u_int16 type_lang;
} f_unload_pb, *F_unload_pb;

Description
F_UNLOAD locates the module in the module directory, decrements its link count, and removes it from the directory if the count reaches zero. A sticky module is not removed until its link count is -1. This call is similar to F_UNLINK, except F_UNLOAD is passed the pointer to the module name instead of the address of the module header.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

cb
The control block header.

mod_name
Point to the module name.

type_lang
Specify the module’s type and language.

Possible Errors
EOS_MNF
EOS_MODBSY

See Also
F_LINK
F_UNLINK
Chapter 6: OS-9 System Calls

F_VMODUL
Verify Module

Headers
#include <module.h>

Parameter Block Structure
typedef struct f_vmodul_pb {
    syscb cb;
    Mh_com mod_head,
        mod_block;
    u_int32 block_size;
} f_vmodul_pb, *F_vmodul_pb;

Description
F_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 one with the highest revision level is retained in the module directory. Ties are broken in favor of the established module.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

cb
    The control block header.

mod_head
    Point to the module.

mod_block
    Point to the memory block containing the module.

block_size
    The size of the memory block containing the module.
Possible Errors
EOS_BMCRC
EOS_BMHP
EOS_BMID
EOS_DIRFUL
EOS_KWNMOD

See Also
F_CRC
F_LOAD
F_WAIT
Wait for Child Process to Terminate

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_wait_pb {
    syscall cb;
    process_id child_id;
    status_code status;
} f_wait_pb, *F_wait_pb;

Description
F_WAIT deactivates the calling process until a child process terminates. The child’s ID number and exit status are returned to the parent.

If the caller has several children, the caller is activated when the first child dies, so one F_WAIT call is required to detect the termination of each child.

If a child died before the F_WAIT call, the caller is reactivated immediately. F_WAIT returns an error only if the caller has no children.

The process descriptors for child processes are not returned to free memory until their parent process performs an F_WAIT system call or terminates.

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.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters

  cb
      The control block header.

  child_id
      The process ID of the terminating child.

  status
      The child process’ exit status code.
Possible Errors
EOS_NOCHLD

See Also
F_EXIT
F_FORK
F_SEND
F_WAITID
Wait for a Specified Process or Thread to Exit

Headers
#include <process.h>

Parameter Block Structure
typedef struct f_waitid_pb {
    syscb               cb;
    process_id         child_id;
    status_code        status;
    signal_code        signal;
    u_int32            wait_flag;
} f_waitid_pb, *F_waitid_pb;

Description
F_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

Waiting for a Child Process or Sibling Thread
To specify a wait related activity, wait_flag should be 0.
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.
If the call is successful, the exit code of child_id is returned in status.
If the wait is interrupted by a signal, a value of EOS_BSIG is returned by F_WAITID
and the value of the signal that caused the interruption is stored in signal.

Controlling a Signal for the Death of a Child Process or Sibling Thread
To specify a signal related activity, wait_flag should be non-zero. The valid values
for wait_flag are WT_SIGNAL and WT_RELEASE.
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 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.
signal is irrelevant in this case.
status is not modified when wait_flag is non-zero. F_WAITID returns immediately
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

Parameters

\( cb \)
- System call control block.

\( \text{child_id} \)
- Input: Child process or Thread ID

\( \text{status} \)
- Output: Status of child.

\( \text{signal} \)
- Input: Signal to send.
- Output: Signal that aborted blocking wait.

\( \text{wait\_flag} \)
- Input: wait condition flag.

See Also
F_EXIT
F_THEXIT
F_WAIT
Chapter 6: OS-9 System Calls

F_WAITLK
Activate Next Process Waiting to Acquire Lock

Headers
#include <types.h>

Parameter Block Structure
typedef struct f_waitlk_pb {
    syscb cb;
    lock_id lid;
    signal_code signal;
} f_waitlk_pb, *F_waitlk_pb;

Description
F_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. In either case, the calling process is suspended and inserted into a waiting queue for the resource based on relative scheduling priority.

If, during the course of waiting on a lock, a process receives a signal, 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 via the signal pointer.

If an S_WAKEUP signal is received by a waiting process, the signal code does not register and will be zero.

Attributes
Operating System: OS-9
State: System
Threads: Safe

Parameters

cb
    The control block header.

lid
    The lock ID on which to wait.

signal
    Point to the received signal.
Possible Errors
EOS_SIGNAL

See Also
F_ACQLK
F_CAQLK
F_CRLK
F_DELLK
F_RELLK
F_YIELD
Yield the Processor

Headers
#include <process.h>

Parameter Block Schedule
typedef struct f_yield_pb {
    syscb cb;
} f_yield_pb, *F_yield_pb;

Description
F_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, F_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 F_YIELD. The status of the process’ or thread’s signal mask remains unchanged during this system call. F_YIELD is much like F_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

Parameters
cb
    System call control block.

See Also
F_SLEEP
I_ALIAS
Create Device Alias

Headers
#include <types.h>

Parameter Block Structure
typedef struct i_alias_pb {
    syscb cb;
    u_char *alias_name,
        *real_name;
} i_alias_pb, *I_alias_pb;

Description
I_ALIAS creates an alternate name for a device pathlist. Processes can then reference a specific device pathlist with a shorter or more convenient name.
To delete an existing alias from the system, pass a NULL pointer for the real name.
Do not use a real device name as alias_name.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

    cb
        The control block header.

    alias_name
        Point to the alternate name.

    real_name
        Point to the actual device name; it must exist. OS-9 does not validate its existence of the device.

Possible Errors
EOS_BPNAM
I_ATTACH
Attach New Device to System

Headers

#include <io.h>
#include <modes.h>

Parameter Block Structure

typedef struct i_attach_pb {
    syscb cb;
    u_char *name;
    u_int16 mode;
    Dev_list dev_tbl;
} i_attach_pb, *I_attach_pb;

Description

I_ATTACH causes a new I/O device to become known to the system or verifies the device is already attached.

If the descriptor is found and the device is not already attached, I_ATTACH links to its file manager and device driver and places their addresses in a new device list entry. I_ATTACH allocates and initializes static storage memory for the file manager and device driver. After initialization, the file manager’s I_ATTACH entry point is called to allow for file manager specific initialization. In turn, the file manager calls the driver’s initialization entry point to initialize the hardware. If the driver has already been attached, the file manager usually omits calling the driver.

I_ATTACH prepares the device for subsequent use by any process, but does not reserve the device. I_ATTACH is not required to perform routine I/O.

IOMAN attaches all devices at I_OPEN and detaches them at I_CLOSE.

Attach and Detach for devices are used together like Link and Unlink for modules. However, you can improve system performance slightly by attaching all devices at startup. This increments each device’s use count and prevents the device from being reinitialized every time it is opened. If static storage for devices is allocated all at once, memory fragmentation is minimized. If a device is attached, the termination routine is not executed until the device is detached.

Attributes

Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe
Parameters

cb
The control block header.

name
Point to the I/O device. name is used to search the current module directory for a device descriptor module with the same name in memory. This is the name by which the device is known. The descriptor module contains the name of the device’s file manager, device driver, and other related information.

mode
The access mode used to verify subsequent read and/or write operations are permitted. It can be either S_IREAD or S_IWRITE.

dev_tbl
A returned value. It points to the device’s device list entry.

Possible Errors

EOS_BMODE
EOS_DEVBSY
EOS_DEVOVF
EOS_MEMFUL

See Also

I_CLOSE
I_DETACH
I_OPEN
Chapter 6: OS-9 System Calls

I_CHDIR
Change Working Directory

Headers
#include <types.h>
#include <modes.h>

Parameter Block Structure
typedef struct i_chdir_pb {
    syscb cb;
    u_char *name;
    u_int16 mode;
} i_chdir_pb, *I_chdir_pb;

Description
I_CHDIR changes a process’ working directory to the directory file specified by the pathlist. The execution or data directory (or both) may be changed, depending on the specified access mode. The file specified must be a directory file, and the caller must have access permission for the specified mode.

If the access mode is read, write, or update (read and write), the current data directory is changed. If the access mode is execute, the current execution directory is changed. You can change both simultaneously.

The shell chd directive uses update mode. This means you must have both read and write permission to change directories from the shell. This is a recommended practice.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

    cb
        The control block header.
    name
        Point to the pathlist.
mode
Specify the access mode. The following are the valid modes:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IREAD</td>
<td>Read</td>
</tr>
<tr>
<td>S_IWRITE</td>
<td>Write</td>
</tr>
<tr>
<td>S_IEXEC</td>
<td>Execute</td>
</tr>
</tbody>
</table>

Possible Errors
EOS_BMODE
EOS_BPNAM
I_CIOPROC
Get Pointer to I/O Process Descriptor

Headers
#include <io.h>

Parameter Block Structure
typedef struct i_cioproc_pb {
    syscb cb;
    process_id proc_id;
    void *buffer;
    u_int32 count;
} i_cioproc_pb, *I_cioproc_pb;

Description
I_CIOPROC copies the I/O process descriptor for the specified process into a buffer.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

cb
    The control block header.

proc_id
    The process ID of the process.

buffer
    Point to the buffer in which to copy the process descriptor.

count
    Specify the number of bytes to copy.

Possible Errors
EOS_IPRCID
I_CLOSE
Close Path to File/Device

Headers
#include <types.h>

Parameter Block Structure
typedef struct i_close_pb {
    syscb cb;
    path_id path;
} i_close_pb, *I_close_pb;

Description
I_CLOSE terminates an I/O path.
The path number is no longer valid for OS-9 calls unless it becomes active again through an I_OPEN, I_CREATE, or I_DUP system call.
When pathlists to non-sharable devices are closed, the devices become available to other requesting processes.
If this is the last use of the path (it has not been inherited or duplicated by I_DUP), all internally managed buffers and descriptors are deallocated.
F_EXIT automatically closes any open paths. By convention, standard I/O paths are not closed unless it is desired to change the corresponding files/devices.
I_CLOSE does an implied I_DETACH call. If this causes the device use count to become zero, the device termination routine is executed.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters
cb
    The control block header.

path
    Identifies the I/O path to close.

Possible Errors
EOS_BPNUM

See Also
F_EXIT
I_DETACH  I_DUP
I_CONFIG
Configure an Element of Process/System I/O

Headers
#include <types.h>

Parameter Block Structure

typedef struct i_config_pb {
    syscb cb;
    u_int32 code;
    void *param;
} i_config_pb, *I_config_pb;

Description
I_CONFIG is a wildcard call used to configure elements of the I/O subsystem that 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.

Table 6-6. Sub-Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Parameter</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC_PATHSZ</td>
<td>param points to the number of additional paths the process wants beyond its initial 32.</td>
<td>Increase the number of paths the current process may have open beyond its initial 32. This can only be used to increase the number of paths a process may have. It cannot be used to reduce the number of available paths.</td>
</tr>
</tbody>
</table>

Attributes

Operating System: OS-9
State: User, System, Interrupt
Threads: Safe

Parameters

cb
The control block header.

code
Identify the target configuration code.

*param
Point to additional parameters required by the specified configuration function.

See Also
F_CONFIG
I_CREATE
Create Path to New File

Headers
#include <types.h>
#include <modes.h>

Parameter Block Structure
typedef struct i_create_pb {
    syscb cb;
    u_char *name;
    u_int16 mode;
    path_id path;
    u_int32 perm,
        size;
} i_create_pb, *I_create_pb;

Description
I_CREATE creates a new file. On multi-file devices, the new file name is entered in the
directory structure. On non-multi-file devices, I_CREATE is synonymous with
I_OPEN. Also, if the file already exists on a multi-file device, by default a path to the
file will be opened and the contents truncated.

mode must have the write bit set if data is to be written to the file. The file is given
the attributes passed in perm. The individual bits are defined as follows:

Table 6-7. Mode and Attribute Bits

<table>
<thead>
<tr>
<th>Mode Bits</th>
<th>Attribute Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IWRITE = write</td>
<td>S_IWRITE = owner write permission</td>
</tr>
<tr>
<td>S_IEXEC = execute</td>
<td>S_IEXEC = owner exec permission</td>
</tr>
<tr>
<td>S_IAPPEND = append to file</td>
<td>S_IAPPEND = file is non-sharable</td>
</tr>
</tbody>
</table>

If the S_IEXEC (execute) bit of the access mode byte is set, the working execution
directory is searched first, instead of the working data directory.

If the S_IEXCL mode bit is not set and the target file already exists, the file is
truncated to zero length.
If the `S_ICONTIG` mode bit is set, the space for the file is allocated from a single contiguous block.

If the `S_IAPPEND` 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 the `S_ISHARE` mode bit is set, the opening process has exclusive access to the file.

If the `S_ISIZE` mode bit is set, it is assumed the `size` parameter contains the initial file size of the target file.

File space is allocated automatically by `I_WRITE` or explicitly by an `I_SETSTAT` call.

If the pathlist specifies a file name that already exists, an error occurs. You cannot use `I_CREATE` to make directory files (see `I_MAKDIR`).

`I_CREATE` causes an implicit `I_ATTACH` call. The device’s initialization routine is executed if the device has not been attached previously.

**Attributes**

Operating System: OS-9  
State: User, System, Interrupt  
Threads: Safe

**Parameters**

`cb`  
The control block header.

`name`  
Point to the pathname of the new file.

`mode`  
Specify the access mode. If data is to be written to the file, `mode` must have the write bit set.

`path`  
A returned value. It is the path number that identifies the file in subsequent I/O service requests until the file is closed.

`perm`  
Specify the attributes to use for the new file.

`size`  
Specify the size of the new file. If the `S_ISIZE` (initial file size) bit is set, you may pass an initial file size estimate in `size`.

**Possible Errors**

EOS_BPNAM  
EOS_PTHFUL

**See Also**

`I_ATTACH`  
`I_CLOSE`  
`I_MAKDIR`  
`I_SETSTAT`  
`I_WRITE`
I_DELETE
Delete File

Headers
#include <types.h>
#include <modes.h>

Parameter Block Structure
typedef struct i_delete_pb {
    syscb cb;
    u_char *name;
} i_delete_pb, *I_delete_pb;

Description
I_DELETE deletes the file specified by the pathlist. You must have non-sharable write
access to the file (the file may not already be open) or an error results. Attempts to
delete non-multi-file devices result in an error.
The access mode is ignored if a full pathlist is specified (a full pathlist begins with a
slash (/)).

Parameters

cb
    The control block header.

name
    Point to the file to delete.

mode
    Specify the access mode. mode may be S_IREAD, S_IWRITE, or S_IEXEC. The
    access mode specifies the data or execution directory (but not both) in the
    absence of a full pathlist. If the access mode is read, write, or update (read and
    write), the current data directory is assumed. If the execute bit is set, the
    current execution directory is assumed.

Attributes
Operating System:          OS-9
State:                    User, System, and Interrupt
Threads:                  Safe

Possible Errors
EOS_BPNAM

See Also
    I_ATTACH     I_CREATE
    I_DETACH     I_OPEN
I_DETACH
Remove Device from System

Headers
#include <io.h>

Parameter Block Structure
typedef struct i_detach_pb {
    syscb cb;
    Dev_list dev_tbl;
} i_detach_pb, *I_detach_pb;

Description
I_DETACH removes a device from the system device list if the device is not in use by any other process.

If this is the last use of the device, the file manager’s I_DETACH routine is called, and in turn, the device driver’s termination routine is called and any permanent storage assigned to the file manager and driver is de-allocated. The device driver and file manager modules associated with the device are unlinked and may be lost if not in use by another process. It is crucial for the termination routine to remove the device from the IRQ system.

I_DETACH must be used to detach devices attached with I_ATTACH. Both of these attach and detach requests are used mainly by IOMAN and are of limited use to the typical user. SCF also uses attach/detach to set up its second (echo) device.

Most devices are attached at startup and remain attached while the system is up. An infrequently used device can be attached and then detached to free system resources when no longer needed.

Attributes
Operating System: OS-9
State: User, System, and Interrupt
Threads: Safe

Parameters

cb
    The control block header.

dev_tbl
    Point to the address of the device list entry.

See Also
I_ATTACH
I_CLOSE
I_DUP
Duplicate Path

Headers

```c
#include <types.h>
```

Parameter Block Structure

```c
typedef struct i_dup_pb {
    syscb cb;
    path_id dup_path,
    *new_path;
} i_dup_pb, *I_dup_pb;
```

Description

I_DUP duplicates a path. The operation of I_DUP depends on the state from which it is called.

When called from a user-state process and given an existing path number, I_DUP returns a synonymous path number for the same file or device. I_DUP always uses the lowest available path number. For example, if you perform an I_CLOSE on path 0 and an I_DUP on path 4, path 0 is returned as the new path number. In this way, the standard I/O paths may be manipulated to contain any desired paths.

When called from a system-state process, I_DUP returns the next available system path number.

The shell uses this service request when it redirects I/O. Service requests using either the old or new path numbers operate on the same file or device.

I_DUP increments the use count of a path descriptor and returns a synonymous path number. The path descriptor is NOT copied. It is usually not a good idea for more than one process to be performing I/O on the same path concurrently. On RBF files, this can produce unpredictable results.

Attributes

- Operating System: OS-9
- State: User, System, and Interrupt
- Threads: Safe

Parameters

- **cb**
  The control block header.
- **dup_path**
  The path number of the path to duplicate.
- **new_path**
  The new number for the same path.
Possible Errors
EOS_BPNUM
EOS_PTHFUL

See Also
I_CLOSE
I_GETDL
Get System I/O Device List Head Pointer

Headers
#include<io.h>

Parameter Block Structure
typedef struct i_getdl_pb{
    syscb cb;
    Dev_list dev_list_ptr;
} i_getdl_pb, *I_getdl_pb;

Description
I_GETDL returns a pointer to the first entry in the system’s I/O device list.

Attributes
Operating System: OS-9
State: User, System, I/O, and Interrupt
Threads: Safe

Parameters

  cb
    The control block header.

  dev_list_ptr
    A returned value. It points to the first entry in the device list.

!! Never access this pointer directly in user state. You should use F_CPYMEM to get
    a copy of the device list entry. This system call is used by the devs utility to
determine the presence of all of the active devices in the system.

See Also
F_CPYMEM
Chapter 6: OS-9 System Calls

I_GETPD
Find Path Descriptor

Headers
#include <types.h>
#include <io.h>

Parameter Block Structure
typedef struct i_getpd_pb {
    syscb cb;
    path_id path;
    Pd_com path_desc;
} i_getpd_pb, *I_getpd_pb;

Description
I_GETPD converts a path number to the absolute address of its path descriptor data structure.

Attributes
Operating System: OS-9
State: System, I/O, and Interrupt
Threads: Safe

Parameters

   cb
        The control block header.

   path
        Specify the path number.

   path-id
        A returned value. It points to the path descriptor.
I_GETSTAT  
Get File/Device Status

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct i_getstat_pb {
    syscb cb;
    path_id path;
    u_int16 gs_code;
    void *param_blk;
} i_getstat_pb, *i_getstat_pb;

Description
I_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. A typical use is to determine a terminal’s parameters (such as echo on/off and delete character). It is often used with I_SETSTAT, which sets the device operating parameters.

The mnemonics for the status codes are found in the header file funcs.h. Codes 0 - 127 are reserved for Microware’s use. You may define the remaining codes and their parameter passing conventions. The status codes that are currently defined and the functions they perform are described in the functions with an SS_prefix. Supported getstats include:

<table>
<thead>
<tr>
<th>Getstat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_GETSTAT, SS_COPYPD</td>
<td>Copy Contents of Path Descriptor (All)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_CSTATS</td>
<td>Get Cache Status Information (RBF)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_DEVNAME</td>
<td>Return Device Name (All)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_DEVOPT</td>
<td>Read Device Path Options</td>
</tr>
<tr>
<td>I_GETSTAT, SS_DEVTYPE</td>
<td>Return Device Type (All)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_DSIZE</td>
<td>Get Size of SCSI Devices (RBF)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_EDT</td>
<td>Get I/O Interface Edition Number (All)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_EOF</td>
<td>Test for End of File (All)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_FD</td>
<td>Read File Descriptor Sector (RBF, PIPE)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_FdAddr</td>
<td>Get File Descriptor Block Address for Open File (RBF, PCF)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_FdINFO</td>
<td>Get Specified File Descriptor Sector (RBF, Pipe)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_LUOPT</td>
<td>Read Logical Unit Options (All)</td>
</tr>
</tbody>
</table>
Table 6-8. Supported Getstats

<table>
<thead>
<tr>
<th>Getstat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_GETSTAT, SS_PARITY</td>
<td>Calculate Parity of File Descriptor (RBF)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_PATHOPT</td>
<td>Read Path Descriptor Option Section (All)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_POS</td>
<td>Get Current File Position (RBF)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_READY</td>
<td>Test for Data Ready (RBF, SCF, PIPE)</td>
</tr>
<tr>
<td>I_GETSTAT, SS_SIZE</td>
<td>Set File Size (RBF, PIPE, PCF)</td>
</tr>
</tbody>
</table>

Attributes

Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

cb
The control block header.

path
The path number.

gs_code
The get status code.

param_blk
Point to the parameter block corresponding to the function being performed. If the get status function does not require a parameter block, param_blk should be null.

Possible Errors

EOS_UNKSVC

See Also

I_SETSTAT
Chapter 6: OS-9 System Calls

I_GETSTAT, SS_COPYPD
Copy Contents of Path Descriptor (ALL)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_cpypd_pb {
    u_int32 size;
    void *path_desc;
} gs_cpypd_pb, *Gs_cpypd_pb;

Description
SS_COPYPD copies the contents of the specified path’s path descriptor to the path descriptor buffer.

Attributes
Operating System: OS-9
State: User, System, I/O, and Interrupt
Threads: Safe

Parameters
size
    The number of bytes to copy from the path descriptor. If the size value is greater than the size of the target path descriptor, size is updated with the actual size of the path descriptor.

path_desc
    Point to the buffer for the path descriptor data.

Possible Errors
EOS_BPNUM
### I_GETSTAT, SS_CSTATS
Get Cache Status Information (RBF)

#### Headers
```c
#include <rbf.h>
#include <sg_codes.h>
```

#### Parameter Block Structure
```c
typedef struct gs_cstats_pb {
    Cachestats cache_inf;
} gs_cstats_pb, *Gs_cstats_pb;
```

#### Description
`SS_CSTATS` returns a copy of the current `cachestats` structure.

#### Attributes
- Operating System: OS-9
- State: User, System, and I/O
- Threads: Safe

#### Parameters
- `cache_inf`
  - Point to a structure containing information about RBF caching.

#### Possible Errors
- `EOS_BPNUM`
Chapter 6: OS-9 System Calls

I_GETSTAT, SS_DEVNAME
Return Device Name (ALL)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_devname_pb {
    u_char *namebuf;
} gs_devname_pb, *Gs_devname_pb;

Description
SS_DEVNAME returns the name of the device associated with the specified path.

Attributes
Operating System: OS-9
State: User, System, I/O, and Interrupt
Threads: Safe

Parameters
namebuf
    Point to the buffer containing the device name.

Possible Errors
EOS_BPNUM
Chapter 6: OS-9 System Calls

I_GETSTAT, SS_DEVOPT
Read Device Path Options

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_dopt_pb {
    u_int32 dopt_size;
    void *user_dopts;
} gs_dopt_pb, *Gs_dopt_pb;

Description
SS_DEVOPT gets the initial (default) device path options. These options are used for initializing new paths to the device.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
dopt_size
    A returned value. It is the size of the option area.

user_dopts
    Point to the list of device path options buffer.

Possible Errors
EOS_BPNUM
I_GETSTAT, SS_DEVTYPE
Return Device Type (ALL)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_devtype_pb {
    u_int16 type;
    u_int16 class;
} gs_devtype_pb, *Gs_devtype_pb;

Description
SS_DEVTYPE returns the type and class of the device associated with the specified path number.
The values for the device type and device class are defined in the io.h header file.

Attributes
Operating System: OS-9
State: User, System, I/O, and Interrupt
Threads: Safe

Parameters
type
    A returned value. It is the device type.

class
    A returned value. It is the device class.

Possible Errors
EOS_BPNUM
I_GETSTAT, SS_DISKFREE
Return Information About RBF Disk Free Space

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_diskfree_pb {
    u_int32 bavail;
    u_int32 bsize;
    u_int32 blocks;
    u_int32 bcontig;
} gs_diskfree_pb, *Gs_diskfree_pb;

Description
SS_DISKFREE returns information about RBF disk free space.

Attributes
Operating System: OS-9
State: User and System
Threads: Safe

Parameters
bavail
    Set to total number of free blocks on the disk.

bsize
    Set to size of blocks used on the disk (256, 512, etc.).

blocks
    Set to total number of blocks on the disk.

bcontig
    Set to number of blocks in the largest contiguous area.

Possible Errors
EOS_READ
Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_dsize_pb {
    u_int32 totblocks,
    blocksize;
} gs_dsize_pb, *Gs_dsize_pb;

Description
SS_DSIZE gets information about the size of a SCSI disk drive.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
totblocks
    A returned value. It is the total number of blocks on the device.

blocksize
    A returned value. It is the size of a disk block in bytes.

Possible Errors
EOS_BPNUM
I_GETSTAT, SS_EDT
Get I/O Interface Edition Number (ALL)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_edt_pb {
    u_int32 edition;
} gs_edt_pb, *Gs_edt_pb;

Description
SS_EDT returns the I/O interface edition number of the driver. It validates the compatibility of drivers and file managers.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
edition
The driver I/O interface edition number.

Possible Errors
EOS_BPNUM
I_GETSTAT, SS_EOF
Test for End of File (ALL)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_eof_pb {
    u_int32 eof;
} gs_eof_pb, *Gs_eof_pb;

Description
SS_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.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

eof
The end-of-file status of the specified path. A value of 1 indicates end of file.

Possible Errors
EOS_BPNUM
EOS_EOF
I_GETSTAT, SS_FD
Read File Descriptor Sector (RBF, PIPE)

Headers
#include <types.h>
#include <rbf.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_fd_pb {
    u_int32 info_size;
    Fd_stats fd_info;
} gs_fd_pb, *Gs_fd_pb;

Description
SS_FD returns a copy of the file descriptor sector for the file associated with the specified path.

Attributes
Operating System:  OS-9
State:  User, System, and I/O
Threads:  Safe

Parameters

infosize
The number of bytes of the file descriptor to copy.

fdinfo
Point to the buffer for the file descriptor sector.

Possible Errors
EOS_BPNUM
I_GETSTAT, SS_FdAddr
Get File Descriptor Block Address for Open File (RBF, PCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_fdaddr_pb {
    u_int32 fd_blkaddr;
} gs_fdaddr_pb, *Gs_fdaddr_pb;

Description
SS_FdAddr returns the file descriptor block address associated with the specified path number.
Only super users can make this call.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
fd_blkaddr
    The block address of the file descriptor.

Possible Errors
EOS_BPNUM
EOS_PERMIT
I_GETSTAT, SS_FDINFO
Get Specified File Descriptor Sector (RBF, PIPE)

Headers
#include <rbf.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_fdinf_pb {
    u_int32 info_size,
    fd_blk_num;
    Fd_stats fd_info;
} gs_fdinf_pb, *Gs_fdinf_pb;

Description
SS_FDINFO returns a copy of the specified file descriptor sector for the file associated with the specified path.
Typically, SS_FDINFO 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. However, the path number must be an open path on the same device as the file. The path number typically represents a path to the directory you are currently scanning.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
info_size
    Specify the number of bytes of the file descriptor block to copy.

fd_blk_num
    Specify the file descriptor sector number to get.

fd_info
    Point to the buffer for the file descriptor block.

Possible Errors
EOS_BPNUM
Chapter 6: OS-9 System Calls

I_GETSTAT, SS_LUOPT
Read Logical Unit Options (ALL)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure

typedef struct gs_luopt_pb {
    u_int32 luopt_size;
    void *user_luopts;
} gs_luopt_pb, *Gs_luopt_pb;

Description
SS_LUOPT copies the contents of the logical unit options for a path into the options buffer.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
luopt_size
Size of the options section to copy. luopt_size may not be less than the size of the file manager's logical unit option section.

user_luopts
Point to the options buffer.

Possible Errors
EOS_BPNUM
EOS_BUF2SMALL
**Chapter 6: OS-9 System Calls**

**I_GETSTAT, SS_PARITY**

Calculate Parity of File Descriptor (RBF)

**Headers**

```c
#include <types.h>
#include <sg_codes.h>
```

**Parameter Block Structure**

```c
typedef struct gs_parity_pb {
    Fd_status fd;
    u_int16 parity;
} gs_parity_pb, *Gs_parity_pb;
```

**Description**

`SS_PARITY` calculates a 32 bit vertical parity for file descriptor structures. This call is used by utilities creating disk images (format disks) and utilities checking the integrity of disks.

**Attributes**

- Operating System: OS-9
- State: User, System, and I/O
- Threads: Safe

**Parameters**

- `fd`
  - Point to the file descriptor block.
- `parity`
  - The resulting parity.

**Possible Errors**

`EOS_BPNUM`
Chapter 6: OS-9 System Calls

I_GETSTAT, SS_PATHOPT
Read Path Descriptor Option Section (ALL)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_popt_pb {
    u_int32 popt_size;
    void *user_popts;
} gs_popt_pb, *Gs_popt_pb;

Description
SS_PATHOPT copies the option section of the path descriptor into the variable-sized area options buffer. You must include rbf.h, sbf.h, and/or scf.h for the corresponding file managers and to declare popt_size according to the size of the rbf_opts, sbf_opts, or scf_opts. SS_PATHOPT is typically used to determine the current settings for functions such as echo and auto line feed.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
popt_size
    The size of the path options section to copy.

user_opts
    Point to the options buffer.

Possible Errors
EOS_BFNUM
I_GETSTAT, SS_POS
Get Current File Position (RBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_pos_pb {
  u_int32 filepos;
} gs_pos_pb, *Gs_pos_pb;

Description
SS_POS returns the current position of the file pointer associated with the specified path.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
filepos
  The file position in byte-size units.

Possible Errors
EOS_BPNUM
I_GETSTAT, SS_READY
Test for Data Ready (RBF, SCF, PIPE)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_ready_pb {
    u_int32 incount;
} gs_ready_pb, *Gs_ready_pb;

Description
SS_READY checks for data available to be read on the specified path. The number of characters available to be read is returned in the incount parameter. RBF devices do not return the EOS_NRDY error. SS_READY returns the number of bytes left in the file and SUCCESS.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
incount
    The number of characters available to be read.

Possible Errors
EOS_BFNUM
EOS_NRDY
I_GETSTAT, SS_SIZE
Set File Size (RBF, PIPE, PCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct gs_size_pb {
    u_int32 filesize;
} gs_size_pb, *Gs_size_pb;

Description
SS_SIZE gets the size of the file associated with the open path to the specified filesize.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
filesize
    The new size of the file in bytes.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
Chapter 6: OS-9 System Calls

I_GIOPROC
Get Pointer to I/O Process Descriptor

Headers
#include <io.h>

Parameter Block Structure
typedef struct i_cioproc_pb {
    syscb cb;
    process_id proc_id;
    Io_proc proc_desc;
} i_cioproc_pb, *I_cioproc_pb;

Description
I_GIOPROC returns a pointer to the I/O process descriptor for the process specified.

Attributes
Operating System: OS-9
State: System and I/O
Threads: Safe

Parameters

cb
    The control block header.

proc_id
    Specify the process ID of the process.

proc_desc
    A returned value. It points to the I/O process descriptor.

Possible Errors
EOS_IPRCIDT
Chapter 6: OS-9 System Calls

I_IODEL
Check for Use of I/O Module

Headers
#include <module.h>

Parameter Block Structure
typedef struct i_iodel_pb {
    syscb cb;
    Mh_com mod_head;
} i_iodel_pb, *i_iodel_pb;

Description
I_IODEL is executed whenever 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.

Attributes
Operating System: OS-9
State: System and I/O
Threads: Safe

Parameters
    cb
        The control block header.

    mod_head
        Point to the module header.

Possible Errors
EOS_MODBSY
I_IOEXIT
Terminate I/O for Exiting Process

Headers
#include <types.h>

Parameter Block Structure
typedef struct i_ioexit_pb {
    syscb cb;
    process_id proc_id;
    u_int32 path_cnt;
} i_ioexit_pb, *I_ioexit_pb;

Description
I_IOEXIT is executed whenever the kernel terminates or chains to a process.

Attributes
Operating System: OS-9
State: System and I/O
Threads: Safe

Parameters

cb
    The control block header.

proc_id
    Specify the process ID.

path_cnt
    Specify the number of I/O paths.
    If the most significant bit of path_cnt is reset, 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 deallocated.

Possible Errors
EOS_IPRCID
Chapter 6: OS-9 System Calls

I_IOFORK
Set Up I/O for New Process

Headers

#include <types.h>

Parameter Block Structure

typedef struct i_iofork_pb {
    syscb cb;
    process_id par_proc_id,
    new_proc_id;
    u_int32 path_cnt;
} i_iofork_pb, *I_iofork_pb;

Description

I_IOFORK is executed whenever the kernel creates a new process. I_IOFORK creates an I/O process descriptor for the new process. IOMAN uses I/O process descriptors to maintain information about a process’ I/O. Each I/O process descriptor contains the user-to-system path number translation table and path numbers for the process’ default data and execution directories.

Attributes

Operating System: OS-9
State: System and I/O
Threads: Safe

Parameters

cb
    The control block header.

par_proc_id
    The parent’s process ID.

new_proc_id
    The process ID of the new process.

path_cnt
    The number of I/O paths the child is to inherit from its parent.

Possible Errors

EOS_NORAM
I_MAKDIR
Make New Directory

Headers
#include <modes.h>

Parameter Block Structure
typedef struct i_makdir_pb {
    syscb cb;
    u_char *name;
    u_int16 mode;
    u_int32 perm,
        size;
} i_makdir_pb, *I_makdir_pb;

Description
I_MAKDIR creates and initializes a new directory as specified by the pathlist.
I_MAKDIR is the only way to create a new directory file. The new directory file contains only entries for itself (.) and its parent directory (..). I_MAKDIR fails on non-multi-file devices. If the execution bit is set, OS-9 begins searching for the file in the working execution directory, unless the pathlist begins with a slash. If the pathlist begins with a slash, it is used as the pathlist.
The caller becomes the owner of the directory. I_MAKDIR does not return a path number because directory files are not opened by this request. You should use I_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 the mode and perm parameters. The individual bits for these parameters are defined as follows (if the bit is set, access is permitted):

Table 6-9. Mode and Attribute Bits for I_MAKDIR

<table>
<thead>
<tr>
<th>Mode Bits</th>
<th>Attribute Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_IREAD = read</td>
<td>$S_IREAD = owner read permission</td>
</tr>
<tr>
<td>$S_IWRITE = write</td>
<td>$S_IWRITE = owner write permission</td>
</tr>
<tr>
<td>$S_IEXEC = execute</td>
<td>$S_IEXEC = owner exec permission</td>
</tr>
<tr>
<td>$S_ITRUNC = truncate on open</td>
<td>$S IGREAD = group read permission</td>
</tr>
<tr>
<td>$S_ICONTIG = ensure contig</td>
<td>$S IGWRITE = group write permission</td>
</tr>
<tr>
<td>$S_IEXCL = do not recreate</td>
<td>$S IGEXEC = group exec permission</td>
</tr>
<tr>
<td>$S_IAPPEND = append to file</td>
<td>$S IOREAD = public read permission</td>
</tr>
<tr>
<td>$S_ISHARE = exclusive use</td>
<td>$S IOWRITE = public write permission</td>
</tr>
<tr>
<td>$S_ISIZE = set initial size</td>
<td>$S IOEXEC = public exec permission</td>
</tr>
</tbody>
</table>

$S_ISHARE = file is non-sharable
Chapter 6: OS-9 System Calls

- If the **S_IEXEC** (execute) bit of the access mode byte is set, the working execution directory is searched first instead of the working data directory.
- If the **S_IEXCL** mode bit is not set and the target file already exists, the file is truncated to zero length.
- If the **S_ICONTIG** mode bit is set, the space for the file is allocated from a single contiguous block.
- If the **S_IWRITE** mode bit is set and the target file already exists, the file is truncated to zero length.
- If the **S_IAPPEND** 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 the **S_ISHARE** mode bit is set, the opening process has exclusive access to the file.
- If the **S_ISIZE** mode bit is set, it is assumed the size parameter contains the initial file size of the target file.

**Attributes**

Operating System: OS-9
State: User and I/O
Threads: Safe

**Parameters**

cb
The control block header.

name
Point to the pathlist.

mode
Specify the access mode.

perm
Specify the access permissions.

size
Specify the initial allocation size. (optional)

**Possible Errors**

EOS_BPNAM
EOS_CEF
EOS_FULL

**See Also**

I_OPEN
Chapter 6: OS-9 System Calls

I_OPEN
Open Path to File or Device

Headers
#include <types.h>
#include <modes.h>

Parameter Block Structure
typedef struct i_open_pb {
    syscb cb;
    u_char *name;
    u_int16 mode;
    path_id path;
} i_open_pb, *I_open_pb;

Description
I_OPEN opens a path to an existing file or device as specified by the pathlist. I_OPEN returns a path number 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 bits set. This allows you to use the I_GETSTAT system requests to examine characteristics such as attributes and size, but does not permit any actual I/O on the path.

For RBF devices, use Read mode instead of Update if the file is not going to be modified. This inhibits record locking and can dramatically improve system performance if more than one user is accessing the file. The access mode must conform to the access permissions associated with the file or device (see I_CREATE).

Refer to modes.h for more information about the modes available for I_OPEN.

If the execution bit mode is set, OS-9 searches for the file in the working execution directory, unless the pathlist begins with a slash. If the pathlist begins with a slash, it uses the entire pathlist and opens the file or device with the execute bit set.

I_OPEN searches only for executables in the execution directory if the FAM_EXEC access mode is used. The execution directory is designed for the location of executable modules, not data modules. The access determination is done by

Table 6-10. Mode for I_OPEN

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IREAD</td>
<td>Read</td>
</tr>
<tr>
<td>S_IWRIITE</td>
<td>Write</td>
</tr>
<tr>
<td>S_IEXEC</td>
<td>Execute</td>
</tr>
<tr>
<td>S_ISHARE</td>
<td>Open file for non-sharable use</td>
</tr>
<tr>
<td>S_IFDIR</td>
<td>Open directory file</td>
</tr>
</tbody>
</table>

Refer to modes.h for more information about the modes available for I_OPEN.
IOMAN based on the file permissions. To override this behavior, add S_IEXEC to the
file creation permissions.

If the single user bit is set, the file is opened for non-sharable access even if the file is
sharable.

Files can be opened by several processes (users) simultaneously. Devices have an
attribute specifying whether or not they are sharable on an individual basis.

_ I_OPEN always uses the lowest path number available for the process.

Directory files may be opened only if the directory bit (S_IFDIR) is set in the access
mode.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

cb
    The control block header.

name
    Point to the path name of the existing file or device.

mode
    Specify which subsequent read and/or write operations are permitted as
    follows (if the bit is set, access is permitted).

path
    The resulting path number.

Possible Errors
EOS_BMODE
EOS_BPNAM
EOS_FNA
EOS_PNNF
EOS_PTHFUL
EOS_SHARE

See Also
_ I_ATTACH
_ I_CLOSE
_ I_CREATE
_ I_GETSTAT
I_RDALST
Copy System Alias List

Headers
#include <types.h>

Parameter Block Structure
typedef struct i_rdalst_pb {
    syscb cb;
    u_char *buffer;
    u_int32 count;
} i_rdalst_pb, *I_rdalst_pb;

Description
I_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.
The I_RDALST system call is used by the alias utility to display the list of aliases 
currently active in the system.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

    cb
        The control block header.

    buffer
        Point to the buffer into which to copy the alias list.

    count
        The total number of bytes to copy. count is updated with the total number of 
        bytes copied.

Possible Errors
EOS_BPADDR

See Also
I_ALIAS
Chapter 6: OS-9 System Calls

I_READ
Read Data from File or Device

Headers
#include <types.h>

Parameter Block Structure
typedef struct i_read_pb {
    syscb cb;
    path_id path;
    u_char *buffer;
    u_int32 count;
} i_read_pb, *I_read_pb;

Description
I_READ reads a specified number of bytes from the specified path number. The path must previously have been opened in read or update mode. The data is returned exactly as read from the file/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. After all data in a file has been read, the next I_READ service request returns an end-of-file error.

The keyboard X-ON/X-OFF characters may be filtered out of the input data on SCF-type devices unless the corresponding entries in the path descriptor have been set to zero. You may want to modify the device descriptor so these path descriptor values are initialized to zero when the path is opened. SCF devices usually terminate the read request when a carriage return is reached.

The number of bytes requested are read unless the end-of-file is reached, an end-of-record occurs (SCF only), the read times out (SCF only), or an error condition occurs.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

cb       The control block header.

path     Specify the path number.
buffer
   Point to the data buffer.

count
   The number of bytes to read. Upon completion, count is updated with the number of bytes actually read.

Possible Errors
EOS_BMODE
EOS_BPNUM
EOS_EOF
EOS_READ

See Also
I_READLN
Chapter 6: OS-9 System Calls

I_READLN
Read Text Line with Editing

Headers
#include <types.h>

Parameter Block Structure
typedef struct i_readln_pb {
    syscb cb;
    path_id path;
    u_char *buffer;
    u_int32 count;
} i_readln_pb, *I_readln_pb;

Description
I_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, I_READLN also causes line editing such as backspacing, line delete, echo, and automatic line feed to occur. Some SCF devices may limit the number of bytes read with one call.

SCF requires the last byte entered be an end-of-record character (normally carriage return). If more data is entered than the maximum specified, it is not accepted and a PD_OVF character (normally bell) is echoed. For example, an I_READLN of exactly one byte accepts only a carriage return to return without error and beeps when other keys are pressed. An I_READLN to SCF returns the number of bytes requested unless the read times out or an error occurs.

After all data in a file has been read, the next I_READLN service request returns an end of file error.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

cb
    The control block header.

path
    Specify the path number.

buffer
    Point to the data buffer.
count
   The number of bytes to read. Upon completion, count is updated with the number of bytes actually read.

Possible Errors
EOS_BMODE
EOS_BPNUM
EOS_EOF
EOS_READ

See Also
I_READ
I_SEEK
Reposition Logical File Pointer

Headers
#include <types.h>

Parameter Block Structure

typedef struct i_seek_pb {
    syscb cb;
    path_id path;
    u_int32 offset;
} i_seek_pb, *I_seek_pb;

Description
I_SEEK repositions the path’s file pointer. The file pointer is the 32-bit address of the next byte in the file to be read or written. I_SEEK usually does not initiate physical positioning of the media. You can perform a seek 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.

A seek to address zero is the same as a rewind operation. Seeks to non-random access devices are usually ignored and return without error.

On RBF devices, seeking to a new disk sector rewrites the internal sector buffer to disk if it has been modified. I_SEEK does not change the state of record locks. Beware of seeking to a negative position. RBF interprets negatives as large positive numbers.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

    cb
        The control block header.

    path
        Specify the path number.

    position
        Specify the new position.

Possible Errors
EOS_BPNUM

See Also
I_READ    I_WRITE
I_SETSTAT
Set File/Device Status

Headers

#include <types.h>
#include <sg_codes.h>

Parameter Block Structure

typedef struct i_seek_pb {
    syscb cb;
    path_id path;
    u_int16 ss_code;
    void *param_blk;
} i_seek_pb, *I_setstat_pb;

Description

I_SETSTAT is a wildcard call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent.

Typically, set status calls are used to set a terminal’s parameters for functions such as backspace character, delete character, echo on/off, null padding, and paging. I_SETSTAT is commonly used with I_GETSTAT which reads the device’s operating parameters. The mnemonics for the status codes are found in the header file funcs.h. Codes 0-127 are reserved for Microware’s use. Users may define the remaining codes and their parameter passing conventions.

Table 6-11. Supported SetStats

<table>
<thead>
<tr>
<th>Setstat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_SETSTAT, SS_ATTR</td>
<td>Set File Attributes (RBF, Pipe, PCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_BREAK</td>
<td>Break Serial Connection (SCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_CACHE</td>
<td>Enable/Disable RBF Caching (RBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_DCOFF</td>
<td>Send Signal When Data Carrier Detect Line Goes False (SCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_DCON</td>
<td>Send Signal When Data Carrier Detect Line Goes True (SCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_DEVOPT</td>
<td>Set Device Path Options (Pipe, SBF, SCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_DSRTS</td>
<td>Disable RTS Line</td>
</tr>
<tr>
<td>I_SETSTAT, SS_ENRTS</td>
<td>Enable RTS Line</td>
</tr>
<tr>
<td>I_SETSTAT, SS_ERASE</td>
<td>Erase Tape (SBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_FD</td>
<td>Write File Descriptor Sector (RBF, PCF, PIPE)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_FILLBUFF</td>
<td>Fill Path Buffer With Data (SCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_FLUSHMAP</td>
<td>Flush Cached Bit Map Information (RBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_HDLINK</td>
<td>Make Hard Link to Existing File (RBF)</td>
</tr>
</tbody>
</table>
Table 6-11. Supported SetStats

<table>
<thead>
<tr>
<th>Setstat</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I_SETSTAT, SS_LOCK</td>
<td>Lock Out Record (RBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_LUOPT</td>
<td>Write Logical Unit Options (All)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_PATHOPT</td>
<td>Write Option Section of Path Descriptor (All)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_RELEASE</td>
<td>Release Device (SCF, PIPE)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_RENAME</td>
<td>Rename File (RBF, PIPE, SCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_RESET</td>
<td>Restore Head to Track Zero (RBF, SBF, PCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_RETEN</td>
<td>Re-tension Pass on Tape Device (SBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_RFM</td>
<td>Skip Tape Marks (SBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_SENDSIG</td>
<td>Send Signal on Data Ready (SCF, PIPE)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_SIZE</td>
<td>Set File Size (RBF, PIPE, PCF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_SKIP</td>
<td>Skip Blocks (SBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_SKIPEND</td>
<td>Skip to End of Tape (SBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_TICKS</td>
<td>Wait Specified Number of Ticks for Record Release (RBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_WFM</td>
<td>Write Tape Marks (SBF)</td>
</tr>
<tr>
<td>I_SETSTAT, SS_WTRACK</td>
<td>Write (Format) Track (RBF)</td>
</tr>
</tbody>
</table>

Attributes

Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

cb
The control block header.

path
The path number.

ss_code
The set status code.

param_blk
Point to the parameter block corresponding to the function being performed.
If the set status function does not require a parameter block, param_blk
should be NULL.

Possible Errors

EOS_UNKSVC

See Also

I_GETSTAT
I_SETSTAT, SS_ATTR
Set File Attributes (RBF, PIPE, PCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_attr_pb {
    u_int32 attr;
} ss_attr_pb, *Ss_attr_pb;

Description
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.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
attr
    Specify the file attributes to change.

Possible Errors
EOS_BPNUM

See Also
I_GETSTAT
I_SETSTAT
I_SETSTAT, SS_BREAK
Break Serial Connection (SCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
This call does not use a substructure to the set status parameter block.

Description:
SS_BREAK breaks a serial connection.
The driver is responsible for implementing this call.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
I_SETSTAT, SS_CACHE
Enable/Disable RBF Caching (RBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_cache_pb {
    u_int32 enblflag,
    drvscsize;
} ss_cache_pb, *Ss_cache_pb;

Description
SS_CACHE enables and disables RBF caching on an RBF device.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
enblflag
    The cache enable/disable flag.
    • If enblflag is zero, caching is disabled.
    • If enblflag is non-zero, caching is enabled.

drvcsizex
    The memory size for the cache.

Possible Errors
EOS_CEF
EOS_PERMIT

See Also
I_SETSTAT
Chapter 6: OS-9 System Calls

I_SETSTAT, SS_DCOFF
Send Signal When Data Carrier Detect Line Goes False (SCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_dcoff_pb {
    signal_code signal;
} ss_dcoff_pb, *Ss_dcoff_pb;

Description
When a modem has finished receiving data from a carrier, the Data Carrier Detect line becomes false. SS_DCOFF sends a signal code when this happens. I_SETSTAT, SS_DCON sends a signal when the line becomes true.
The driver is responsible for implementing this call.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
signal
   The signal code to send.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT, SS_DCON
I_SETSTAT, SS_RELEASE
I_SETSTAT, SS_DCON
Send Signal When Data Carrier Detect Line Goes True (SCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_dcon_pb {
    signal_code signal;
} ss_dcon_pb, *Ss_dcon_pb;

Description
When a modem receives a carrier, the Data Carrier Detect line becomes true. SS_DCON sends a signal code when this happens. I_SETSTAT, SS_DCOFF sends a signal when the line becomes false.
The driver is responsible for implementing this call.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
signal
    The signal code to send.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT, SS_DCOFF
I_SETSTAT, SS_RELEASE
I_SETSTAT, SS_DEVOPT
Set Device Path Options (PIPE, SBF, SCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_dopt_pb {
    u_int dopt_size;
    void *user_dopts;
} ss_dopt_pb, *Ss_dopt_pb;

Description
SS_DOPT sets the initial (default) device path options. These options initialize new
paths to the device.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
dopt_size
    Specify the size of the options area to copy.

user_dopts
    Point to the default options for the device.

Possible Errors
EOS_BPNUM

See Also
I_GETSTAT
I_SETSTAT
I_SETSTAT, SS_DSRTS
Disable RTS Line

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
This call does not use a substructure to set the status parameter block.

Description
SS_DSRTS disables the RTS line.
The driver is responsible for implementing this call.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT, SS_ENRTS
I_SETSTAT, SS_ENRTS
Enable RTS Line

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_dcoff_pb {
    signal_code signal;
} ss_dcoff_pb, *Ss_dcoff_pb;

Description
SS_ENRTS asserts the RTS line.
The driver is responsible for implementing this call.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
signal is the signal code to send.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT, SS_DSRTS
I_SETSTAT, SS_ERASE
Erase Tape (SBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_erase_pb {
    u_int32 blks;
} ss_erase_pb, *Ss_erase_pb;

Description
SS_ERASE erases a portion of the tape. The amount of tape erased depends on the
hardware capabilities.
This is dependent on both the hardware and the driver.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

blks
    Specify the number of blocks to erase. If blks is -1, SBF erases until the end-
of-tape is reached. If blks is positive, SBF erases the amount of tape equivalent
to that number of blocks.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
I_SETSTAT, SS_FD
Write File Descriptor Sector (RBF, PCF, PIPE)

Headers

#include <rbf.h>
#include <sg_codes.h>

Parameter Block Structure

typedef struct ss_fd_pb {
    Fd_stats fd_info;
} ss_fd_pb, *Ss_fd_pb;

Description

SS_FD changes the file descriptor sector data. The path must be open for write.

Attributes

Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

fd_info
    Point to the file descriptor's buffer.

You can only change fd_group, fd_owner, and the time stamps fd_atime, fd_mtime, and fd_utime. These are the only fields written back to the disk.
These fields are defined in the fd_stats structure in rbf.h. Only the super
user can change the file's owner ID.

Possible Errors

EOS_BPNUM

See Also

I_GETSTAT
I_SETSTAT
I_SETSTAT, SS_FILLBUFF
Fill Path Buffer With Data (SCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_fillbuff_pb {
    u_int32 size;
    u_char *user_buff;
} ss_fillbuff_pb, *Ss_fillbuff_pb;

Description
SS_FILLBUFF fills the input path buffer with the data in buffer.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
size
    Specify the size of the buffer (amount of data to copy).

user_buff
    Point to the data buffer.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
I_SETSTAT, SS_FLUSHMAP
Flush Cached Bit Map Information (RBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
This call does not use a substructure to the set status parameter block.

Description
SS_FLUSHMAP flushes the cached bit map information for an RBF device. This normally would only be performed after the bit map on the disk is changed by a utility such as format.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
I_SETSTAT, SS_HDLINK
Make Hard Link to Existing File (RBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_link_pb {
    u_char *link_path;
} ss_link_pb, *Ss_link_pb;

Description
SS_HDLINK creates a new directory entry specified by link_path. This directory entry points to the file descriptor block of the open file specified by path in the I_SETSTAT parameter block. SS_HDLINK updates the pathlist pointer.

Attributes
Operating System:     OS-9
State:               User, System, and I/O
Threads:             Safe

Parameters
link_path
    Point to the new name for the directory entry.

Possible Errors
EOS_BPNUM
EOS_CEF
EOS_PNNF

See Also
I_SETSTAT
Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_lock_pb {
    u_int32 size;
} ss_lock_pb, *Ss_lock_pb;

Description
SS_LOCK locks out a section of the file from the current file pointer position up to the specified number of bytes.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
size The size of the section to lockout. If size is zero, all locks are removed (record lock, EOF lock, and file lock). If $ffffffff bytes are requested, 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 SS_LOCK with size set to zero, a read or write of zero bytes, or the file is closed.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
I_SETSTAT, SS_LUOPT
Write Logical Unit Options (ALL)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_luopt_pb {
  u_int32 luopt_size;
  void *user_luopts;
} ss_luopt_pb, *Ss_luopt_pb;

Description
SS_LUOPT writes the logical unit options for a path to a buffer.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
luopt_size
  Specify the buffer size of the logical unit options area.

user_luopts
  Point to the logical unit options.

Possible Errors
EOS_BPNUM
EOS_BUF2SMALL

See Also
I_GETSTAT
I_SETSTAT
**I_SETSTAT, SS_PATHOPT**

Write Option Section of Path Descriptor (ALL)

**Headers**

```c
#include <types.h>
#include <sg_codes.h>
```

**Parameter Block Structure**

```c
typedef struct ss_popt_pb {
    u_int popt_size;
    void *user_popts;
} ss_popt_pb, *Ss_popt_pb;
```

**Description**

`SS_PATHOPT` writes the option section of the path descriptor from the status packet pointed to by `user_opts`. Typically, `SS_PATHOPT` sets the device operating parameters (such as echo and auto line feed). This call is handled by the file managers, and only copies values appropriate for user programs to change.

**Attributes**

- **Operating System:** OS-9
- **State:** User, System, and I/O
- **Threads:** Safe

**Parameters**

- `popt_size`
  - Specify the buffer size.

- `user_popts`
  - Point to the options buffer.

**Possible Errors**

- EOS_BPNUM
- EOS_BUF2SMALL

**See Also**

- `I_GETSTAT`
- `I_SETSTAT`
I_SETSTAT, SSRELEASE
Release Device (SCF, PIPE)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
This call does not use a substructure to the set status parameter block.

Description
SS_RELEASE releases the device from any SS_SENDSIG, SS_DCON, or SS_DCOFF request made by the calling process.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT, SS_DCOFF
I_SETSTAT, SS_DCON
I_SETSTAT, SS_SENDSIG
I_SETSTAT, SS_RENAME
Rename File (RBF, PIPE, SCF)

Parameters

newname
Point to the file’s new name.

Possible Errors

EOS_CEF

See Also

I_SETSTAT
I_SETSTAT, SS_RESET
Restore Head to Track Zero (RBF, SBF, PCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
This call does not use a substructure to the set status parameter block.

Description
For RBF and PCF, SS_RESET directs the disk head to track zero. It is used for formatting and error recovery. For SBF, SS_RESET rewinds the tape.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Possible Errors
EoS_BPNUM

See Also
I_SETSTAT
I_SETSTAT, SS_RETEN
Re-tension Pass on Tape Drive (SBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
This call does not use a substructure to the set status parameter block.

Description
SS_RETEN performs a re-tension pass on the tape drive.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Possible Errors
EOS_BPNUM
EOS_NOTRDY

See Also
I_SETSTAT
I_SETSTAT, SS_RFM
Skip Tape Marks (SBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_rfm_pb {
    int32 cnt;
} ss_rfm_pb, *Ss_rfm_pb;

Description
SS_RFM skips the number of tape marks specified.

Parameters
cnt
   Specify the number of tape marks to skip. If cnt is negative, the tape is rewound the specified number of marks.

Attributes
Operating System:    OS-9
State:              User, System, and I/O
Threads:            Safe

Possible Errors
EOS_BPNUM
EOS_NOTRDY

See Also
I_SETSTAT
I_SETSTAT, SS_SENDSIG
Send Signal on Data Ready (SCF, PIPE)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_sendsig_pb {
    signal_code signal;
} ss_sendsig_pb, *Ss_sendsig_pb;

Description
SS_SENDSIG sets up a signal to be sent to a process when an interactive device or pipe has data ready. SS_SENDSIG must be reset each time the signal is sent. The device or pipe is considered busy and returns an error if any read request arrives before the signal is sent. Write requests to the device are allowed in this state.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
signal
    The signal to send.

Possible Errors
EOS_BMODE
EOS_BPNUM
EOS_NOTRDY

See Also
I_SETSTAT, SS_RELEASE
I_SETSTAT, SS_SIZE
Set File Size (RBF, PIPE, PCF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_size_pb {
    u_int32 filesize;
} ss_size_pb, *Ss_size_pb;

Description
SS_SIZE sets the size of the file associated with the open path to the specified filesize.
If the specified size is smaller than the current size, the data beyond the new end-of-file is lost.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
filesize
    The new size of the file in bytes.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
Chapter 6: OS-9 System Calls

I_SETSTAT, SS_SKIP
Skip Blocks (SBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_skip_pb {
   int32 blks;
} ss_skip_pb, *Ss_skip_pb;

Description
SS_SKIP skips the specified number of blocks.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
blks
   Specify the number of blocks to skip. If blks is negative, the tape is rewound
   the specified number of blocks.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
I_SETSTAT, SS_SKIPEND
Skip to End of Tape (SBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
This call does not use a substructure to the set status parameter block.

Description
SS_SKIPEND skips the tape to the end of data. This enables you to append data to tapes on cartridge-type tape drives.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Possible Errors
EOS_BPNUM
EOS_NOTRDY

See Also
I_SETSTAT
I_SETSTAT, SS_TICKS

Wait Specified Number of Ticks for Record Release (RBF)

Headers

```c
#include <types.h>
#include <sg_codes.h>
```

Parameter Block Structure

```c
typedef struct ss_ticks_pb {
    u_int32 delay;
} ss_ticks_pb, *Ss_ticks_pb;
```

Description:

Normally, if a read or write request is issued for part of a file locked out by another user, RBF sleeps indefinitely until the conflict is removed. **SS_TICKS** may be used to return an error (**EOS_LOCK**) to the user program if the conflict still exists after the specified number of ticks have elapsed.

Attributes

- Operating System: OS-9
- State: User, System, and I/O
- Threads: Safe

Parameters

delay

Specify the delay interval. The delay interval is used directly as a parameter to RBF’s conflict sleep request.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The process sleeps until the record is released. This is RBF’s default.</td>
</tr>
<tr>
<td>1</td>
<td>Returns an error if the record is not released immediately.</td>
</tr>
<tr>
<td>Other</td>
<td>Any other value specifies number of system clock ticks to wait until the conflict area is released. If the high order bit is set, the lower 31 bits are converted from 1/256 second to ticks before sleeping. This allows programmed delays to be independent of the system clock rate.</td>
</tr>
</tbody>
</table>

Possible Errors

- **EOS_BPNUM**
- **EOS_LOCK**

See Also

- **I_SETSTAT**
Chapter 6: OS-9 System Calls

I_SETSTAT, SS_WFM
Write Tape Marks (SBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct ss_wfm_pb {
    u_int32 cnt;
} ss_wfm_pb, *Ss_wfm_pb;

Description
SS_WFM writes the specified number of tape marks at the current position.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
cnt
Specify the number of tape marks to write.

Possible Errors
EOS_BPNUM

See Also
I_SETSTAT
Chapter 6: OS-9 System Calls

I_SETSTAT, SS_WTRACK
Write (Format) Track (RBF)

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure

typedef struct ss_wtrack_pb {
    void *trkbuf,
    *ilvtbl;
    u_int32 track,
    head,
    interleave;
} ss_wtrack_pb, *Ss_wtrack_pb;

Description
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. The interleave table contains byte entries of LBNs ordered to match the requested interleave offset. The path descriptor should be used with the track and side numbers to determine what density and how many blocks a certain track should have.

This function is implemented by the driver. Only super user programs are allowed to issue this command.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
trkbuf
    Point to the track buffer.

ilvtbl
    Point to the interleave table. The interleave table contains byte entries of LBNs ordered to match the requested interleave offset.

track
    The track number.
head
   The side number.

interleave
   The interleave value.

Possible Errors
EOS_FMTERR
EOS_FORMAT

See Also
I_SETSTAT
I_SGETSTAT
GetStat Call Using System Path Number

Headers
#include <types.h>
#include <sg_codes.h>

Parameter Block Structure
typedef struct i_getstat_pb{
    syscb cb;
    path_id path;
    u_init16 gs_code;
    void *param_blk;
} i_getstat_pb, *I_getstat_pb;

Description
I_SGETSTAT is a wildcard call used to handle individual device parameters that are not uniform on all devices or are highly hardware dependent. I_SGETSTAT provides the same functionality as I_GETSTAT except the path number for I_SGETSTAT is assumed to be a system path number and not a user path number.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters
cb
    The control block header.
path
    The system path number.
gs_code
    The get status code.
param_blk
    Point to the parameter block corresponding to the function being performed. If the get status function does not require a parameter block param_blk should be NULL.

Possible Errors
EOS_UNKSVC

See Also
I_GETSTAT       I_SETSTAT
I_TRANPN
Translate User Path to System Path

Headers
#include <types.h>

Parameter Block Structure
typedef struct i_tranpn_pb {
    syscb cb;
    process_id proc_id;
    path_id user_path,
    sys_path;
} i_tranpn_pb, *i_tranpn_pb;

Description
I_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).

Attributes
Operating System: OS-9
State: System and I/O
Threads: Safe

Parameters

cb
    The control block header.

proc_id
    Specify the process ID.

user_path
    Specify the user path to translate.

sys_path
    The mapped system path.

Possible Errors
EOS_BPNUM
EOS_IPRCID
Chapter 6: OS-9 System Calls

I_WRITE
Write Data to File or Device

Headers

#include <types.h>

Parameter Block Structure

typedef struct i_write_pb {
    syscb cb;
    path_id path;
    u_char *buffer;
    u_int32 count;
} i_write_pb, *I_write_pb;

Description

I_WRITE outputs bytes to a file or device associated with the specified path number. The path must have been 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.

On RBF devices, any locked record is released.

Attributes

Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

cb            The control block header.

path          The specified path number for the file or device.

buffer        Point to the data buffer.

count         The number of bytes written.

Possible Errors

EOS_BMODE
EOS_BPNUM
EOS_WRITE

See Also

I_CREATE     I_OPEN     I_WRITELN
Chapter 6: OS-9 System Calls

I_WRITELN
Write Line of Text with Editing

Headers
#include <types.h>

Parameter Block Structure
typedef struct i_writeln_pb {
    syscb cb;
    path_id path;
    u_int32 count
    u_char *buffer;
} i_writeln_pb, *I_writeln_pb;

Description
I_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. I_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. The line editing refers to functions such as auto line feed and null padding at end-of-line.

The number of bytes actually written (returned in count) does not reflect any additional bytes added by file managers or device drivers for device control. For example, if SCF appends a line feed and nulls after carriage return characters, these extra bytes are not counted.

On RBF devices, any locked record is released.

Attributes
Operating System: OS-9
State: User, System, and I/O
Threads: Safe

Parameters

cb
    The control block header.

path
    The path number of the file or device.

buffer
    Point to the data buffer.

count
    The number of bytes written.
Possible Errors
EOS_BMODE
EOS_BPNUM
EOS_WRITE

See Also
I_CREATE
I_OPEN
I_WRITE

The OS-9 Porting Guide (the SCF Drivers line editing section)
Use the examples in this section as guides for creating your own modules. These examples should not be considered the most current software. Software for your individual system may be different.

This appendix includes the following topics:

- Sysgo
- Signals: Example Program
- Alarms: Example Program
- Events: Example Program
- Semaphores: Example Program
- Usemaphores: Example Program
- The Subroutine Library
- Trap Handlers
Sysgo

Sysgo can be configured as the first user process started after the system start-up sequence. Its standard I/O is on the system console device.

Sysgo executes as follows:

1. Change to the CMDS execution directory on the system device.
2. Execute the start-up file (as a script) from the SYS directory on the root of the system device.
3. Fork a shell on the system console.
4. Wait for that shell to terminate and then fork it again. Unless Sysgo dies, a shell is always running on the system console.

The standard Sysgo module for disk systems cannot be used on non-disk systems, but is easy to customize.

```
_asm("_sysedit: equ 2");

#include <const.h>
#include "defsfile"

/* global variables and declarations */

u_int32 sighandler(), /* intercept handler */
o9fork(); /* used by os9exec */
void errexit(), /* error printing routine */
out3dec(); /* print three decimal digits */
error_code lerrmsg(); /* print the error message */
char *cmdsdir = "CMDS", /* the commands directory */
*startup = "SYS/startup", /* the startup script */
*shell = "Shell"; /* the shell command name */

/* main - main program body */
void main(argc, argv)
        register u_int32 argc; /* number of arguments */
        register u_char argv[]; /* the arguments themselves */
{
    register path_id stdid_dup; /* duped stdin ID */
    register process_id shellpid; /* the process ID */
    char *envp[1]; /* environment variables */
    static char *args[] = {
        "shell",
        "-npxt\n",
        NULL
    };

```
intercept(sighandler);  /* catch signals */
if (chdir(cmdsdir) == ERROR)
    errexit(errno, "can't change to commands directory");
if ((stdid_dup = dup(_fileno(stdin))) == ERROR)
    errexit(errno, "can't duplicate standard input path");
close(_fileno(stdin));  /* close stdin path */
if (open(startup, S_IREAD) == ERROR) {
    lerrmsg(errno, "can't open startup due to error ");
    dup(stdid_dup);  /* reset stdin path */
}
envp[0] = NULL;  /* initialize environments */
for (;;) {
    if (os9exec(os9fork, shell, args, envp, 0, 0, 3) == ERROR)
        errexit(errno, "can't fork shell");
    close(_fileno(stdin));  /* close old stdin */
    dup(stdid_dup);  /* restore initial stdin */
    wait(0);  /* wait for it to die */
    args[1] = "\n";  /* no more special options */
    envp[0] = NULL;
}
/* sighandler - ignore signals so we stay alive */

u_int32 sighandler(sigval)
register u_int32    sigval;                 /* the signal */
{
    return SUCCESS;                         /* don't quit */
}
/* errexit - print error message and leave */

void errexit(error, msg)
register error_code     error;       /* the error that caused us to quit */
register char           *msg;        /* our explanation */
{
    write(_fileno(stdout), msg, strlen(msg));
    exit(lerrmsg(error, " due to error "));
}
/* lerrmsg - print error message and number */

error_code lerrmsg(error, msg)
register error_code     error;       /* the error code */
register char           *msg;        /* the error message */
 Signals: Example Program

The following program demonstrates a subroutine that reads a \n terminated string from a terminal with a ten second timeout between the characters. This program illustrates signal usage, but does not contain any error checking.

The _ss_ssig(path, value) library call notifies the operating system to send the calling process a signal with signal code value when data is available on path. If data is already pending, a signal is sent immediately. Otherwise, control is returned to the calling program and the signal is sent when data arrives.

#include <stdio.h>
#include <errno.h>

#define TRUE 1
#define FALSE 0

#define GOT_CHAR 2001
short dataready;   /* flag to show that signal was received */

/* sighand - signal handling routine for this process */
sighand(signal)
register int signal;

{ 
    switch(signal) { 
        /* ^E or ^C? */
        case 2:
        case 3:
            _errmsg(0,"termination signal received\n");
            exit(signal);
        /* Signal we’re looking for? */
        case GOT_CHAR:
            dataready = TRUE;
            break;
        /* Anything else? */
        default:
            _errmsg(0,"unknown signal received ==> %d\n",signal);
            exit(1);
    }
}

main() { 
    char buffer[256];            /* buffer for typed-in string */
    intercept(sighand);          /* set up signal handler */
    printf("Enter a string:\n"); /* prompt user */

    /* call timed_read, returns TRUE if no timeout, -1 if timeout */
    if (timed_read(buffer) == TRUE)
        printf("Entered string = %s\n",buffer);
    else
        printf("\nType faster next time!\n");
}

int timed_read(buffer) 
    register char *buffer; 
    { 
        char c = '\0';            /* 1 character buffer for read */
        short timeout = FALSE;    /* flag to note timeout occurred on read */
        int pos = 0;              /* position holder in buffer */

        /* loop until <return> entered or timeout occurs */
        while ( (c != '\n') && (timeout == FALSE) ) { 
            _os_sigmask(1);            /* mask signals for signal setup */
            _ss_ssig(0,GOT_CHAR);      /* set up to have signal sent */
            sleep(10);                /* sleep for 10 seconds or until signal */

            })
    
    }}

Appendix A: Example Code

OS-9 Technical Manual 395
/* NOTE: we had to mask signals before doing _ss_ssig() so we did not get the signal between the time we _ss_ssig()’ed and went to sleep. */
/* Now we’re awake, determine what happened */
if (!dataready)
    timeout = TRUE;
else {
    read(0,&c,1); /* read the ready byte */
    buffer[pos] = c; /* put it in the buffer */
    pos++; /* move our position holder */
    dataready = FALSE; /* mark data as read */
    }
    }
/* loop has terminated, figure out why */
if (timeout)
    return -1; /* there was a timeout so return -1 */
else {
    buffer[pos] = '\0'; /* null terminate the string */
    return TRUE;
    }
}

Alarms: Example Program

The following example program can be compiled with this command:

$ cc deton.c

The complete source code for the example program is as follows:

/*------------------------------------------------------------*|
|          Psect Name:deton.c                                  |
|          Function: demonstrate alarm to time out user input  |
|-------------------------------------------------------------*/
@_sysedit: equ 1
#include <stdio.h>
#include <errno.h>
#include <const.h>
#define TIME(secs) ((secs << 8) | 0x80000000)
#define PASSWORD "Ripley"
/*-------------------------------------------------------------*/
sighand(sigcode)
{
    /* just ignore the signal */
}
/*-------------------------------------------------------------*/
main(argc,argv)
int argc;
char **argv;
{
    register int secs = 0;
    register int alarm_id;
    register char *p;
    register char name[80];

    intercept(sighand);
    while (--argc)
    if (*p = *((++argv)) == '-') {
        if (*++p == '?')
            printuse();
        else if (secs == 0)
            secs = atoi(p);
        else exit(_errmsg(1, "unknown arg - \"%s\"\n", p));
    } else if (secs == 0)
        secs = atoi(p);
    else exit(_errmsg(1, "unknown arg - \"%s\"\n", p));

    secs = secs ? secs : 3;
    printf("You have %d seconds to terminate self-destruct...\n", secs);

    /* set alarm to time out user input */
    if ((errno = _os_alarm_set(&alarm_id, 2, TIME(secs))) != SUCCESS)
        exit(_errmsg(errno, "can’t set alarm - ");

    if (gets(name) != 0)
        _os_alarm_delete(alarm_id);   /*remove alarm; it didn’t expire */
    else printf("\n");

    if (_cmpnam(name, PASSWORD, 6) == 0)
        printf("Have a nice day, %s.\n", PASSWORD);
    else printf("ka BOOM\n");
    exit(0);
}

/*---------------------------------------------*/
/* printuse() - print help text to standard error */
printuse()
{
    fprintf(stderr, "syntax: %s [seconds]\n", _prgname());
    fprintf(stderr, "function: demonstrate use of alarm to time out I/O\n");
    fprintf(stderr, "options: none\n");
    exit(0);
}
Events: Example Program

The following program uses a binary semaphore to illustrate the use of events. To execute this example, complete the following steps:

Step 1. Enter or copy the code into a file called sema1.c.
Step 2. Copy sema1.c to sema2.c.
Step 3. Compile both programs.
Step 4. Run both programs using this command: sema1 & sema2.

The program completes the following tasks:
1. Creates an event with an initial value of 1 (free), a wait increment of -1, and a signal increment of 1.
2. Enters a loop that waits on the event.
3. Prints a message.
4. Sleeps.
5. Signals the event.
6. Unlinks itself from the event after ten times through the loop
7. Deletes the event from the system.

```c
#include <module.h>
#include <stdio.h>
#include <memory.h>
#include <errno.h>
#include <const.h>

void main()
{
    char         *ev_name = "semaevent";   /* name of event to be used */
    event_id     ev_id;                /* ID that is used to access event */
    u_int16      perm = MP_OWNER_READ | MP_OWNER_WRITE; /* access perms for event */
    u_int32      value;                    /* returned event value */
    signal_code  signal;                   /* returned signal value */
    int          count = 0;                /* loop counter */

    /* create to link to the event */
    if ((errno = _os_ev_link(ev_name, &ev_id)) != SUCCESS)
        if ((errno = _os_ev_creat(1,-1,perm,&ev_id,ev_name,1,MEM_ANY)) != SUCCESS)
            exit(_errmsg(errno,"error getting access to event - "));
```
while (count++ < 10) {

    /* wait on the event */
    if ((errno = _os_ev_wait(ev_id, &value, &signal, 1, 1)) != SUCCESS)
        exit(_errmsg(errno, "error waiting on the event - ");

    _errmsg(0,"entering "critical section \"\n");

    /* simulate doing something useful */
    sleep(2);

    _errmsg(0,"exiting "critical section \"\n");

    /* signal event (leaving critical section) */
    if ((errno = _os_ev_signal(ev_id, &value, 0)) != SUCCESS)
        exit(_errmsg(errno, "error signalling the event - ");

    /* simulate doing something other than critical section */
    sleep(1);
}

/* unlink from event */
if ((errno = _os_ev_unlink(ev_id)) != SUCCESS)
    exit(_errmsg(errno, "error unlinking from event - ");

/* delete event from system if this was the last process to unlink from it */
if ((errno = _os_ev_delete(ev_name)) != SUCCESS && errno != EOS_EVBUSY)
    exit(_errmsg(errno, " error deleting event from system - ");

    _errmsg(0, terminating normally\n");
Semaphores: Example Program

The following example shows how to use semaphores.

```c
#include <stdio.h>
#include <stdlib.h>
#include <types.h>
#include <module.h>
#include <modes.h>
#include <semaphore.h>

#define DMNAME "hplaserjet"

semaphore *printerSema;

error_code main(int argc, char **argv){
    mh_data *modptr;
    u_int16 attrev, typlang;
    u_int32 perm;
    char *dm_name = DMNAME;
    error_code error;

    /* prepare parameters */
    attrev = mkattrevs(MA_REENT,0);
    typlang = mktypelang(MT_DATA,ML_ANY);
    perm = MP_OWNER_READ|MP_OWNER_WRITE|MP_GROUP_READ|MP_GROUP_WRITE;
    /* first try to create the module */
    error = _os_datmod(dm_name, sizeof(*printerSema),
                        &attrev, &typlang, perm,
                        (void**)&printerSema, &modptr);
    if(error){
        /* then try to link to it */
        error = _os_link(dm_name, (mh_com**)&modptr,
                         (void**)&printerSema, &typlang, &attrev);

        if(error){
            fprintf(stderr,"Couldn’t link or create! Error#%d\n",error);
            _os_exit(error);
        }
    }
}
```
/* initialize semaphore */
_os_sema_init(printerSema);

/* compete for the resource */
_os_sema_p(printerSema);
/* print the file */
printfile(argv[1]);

/* release the semaphore */
_os_sema_v(printerSema);

/* terminate semaphore */
_os_sema_term(printerSema);
/* unlink the data module */
_os_unlink(modptr);

_os_exit(0);
}

#define PRINTER "/p"
#define BUFSIZE 256
error_code printfile(char *filename){
    path_id prnpath, dskpath;
    char buffer[BUFSIZE];
    u_int32 size;
    error_code error;

    /* open path to printer */
    error = _os_open(PRINTER, FAM_WRITE, &prnpath);
    if(error != 0){
        return(error);
    }

    /* open path to disk file */
    error = _os_open(filename, FAM_READ, &dskpath);
    if(error != 0){
        _os_close(prnpath);
        return(error);
    }

    /* print the file */
    printfile(argv[1]);

    /* release the semaphore */
    _os_sema_v(printerSema);

    /* terminate semaphore */
    _os_sema_term(printerSema);
    /* unlink the data module */
    _os_unlink(modptr);

    _os_exit(0);
}
/* until the last byte, read from disk and write to printer */
do{
    size = BUFSIZE;
    error = _os_read(dskpath, buffer, &size);
    if(error == 0 && size > 0){
        _os_write(prnpath, buffer, &size);
    }
}while(size > 0);

_os_close(dskpath);
_os_close(prnpath);
return(0);

Usemaphores: Example Program

The following example, usemademo.c, shows how to use usemaphores.

#define _OPT_PROTOS
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <cglob.h>
#include <types.h>
#include <module.h>
#include <modes.h>
#include <semaphore.h>
#include <memory.h>
#include <const.h>

void main(int, char **);
error_code printfile(char *, char *);

void main(int argc, char **argv)
{
    char *us_name;
    usema_id us_id;
    error_code err;
    signal_code sig;
/* must supply printer name and filename to print */
if (argc != 3) {
    printf("usage: %s <printer device> <filename>\n", _modname);
    exit(EXIT_FAILURE);
}

us_name = argv[1];
if (*us_name++ != '/') {
    printf("%s: printer device should begin with '/\n", _modname);
    exit(EXIT_FAILURE);
}

if (strchr(us_name, '/') != NULL) {
    printf("%s: printer device should not contain path components\n", _modname);
    exit(EXIT_FAILURE);
}

/* create/link usemaphore */
err = _os_usema_create(&us_id, us_name, 1, MP_WORLD_ACCESS, MEM_ANY);
if (err == EOS_EVBUSY)
    err = _os_usema_link(&us_id, us_name);
if (err != SUCCESS)
    exit(_errmsg(err, "failed to create/link %s usemaphore - ",
           (u_int32)us_name, 0, 0));

/* compete for the resource */
for (;;) {
    err = _os_usema_p(us_id, &sig);
    if (err == EOS_OSTOSRT) {
        err = _os_usema_reset(us_id);
        if (err == SUCCESS)
            break;
        else if (err != EOS_USNORST)
            exit(_errmsg(err, "failed to reset %s usemaphore - ",
                        (u_int32)us_name, 0, 0));
    }
    else if (err == SUCCESS)
        break;
    else
        exit(_errmsg(err, "failed to wait for %s usemaphore - ",
                        (u_int32)us_name, 0, 0));
}
/ * print the file */
err = printfile(argv[1], argv[2]);
if (err != SUCCESS)
    exit(_errmsg(err, "failed to print the file %s on %s - ",
         (u_int32)argv[2], (u_int32)argv[1], 0));

/* release the semaphore */
err = _os_usema_v(us_id);
if (err != SUCCESS)
    exit(_errmsg(err, "failed to release the %s usemaphore - ",
         (u_int32)us_name, 0, 0));

/* terminate semaphore */
_os_usema_unlink(us_id);
exit(EXIT_SUCCESS);

#define BUFSIZE 256
error_code printfile(char *printer, char *filename)
{
    path_id prnpath, dskpath;
    u_int8 buffer[BUFSIZE];
    u_int32 size;
    error_code error;

    /* open path to printer */
    error = _os_open(printer, FAM_WRITE, &prnpath);
    if (error != SUCCESS)
        return error;

    /* open path to disk file */
    error = _os_open(filename, FAM_READ, &dskpath);
    if (error != SUCCESS) {
        _os_close(prnpath);
        return error;
    }
/* until the last byte, read from disk and write to printer */
do {
    size = BUFSIZE;
    error = _os_read(dskpath, buffer, &size);
    if (error == SUCCESS && size > 0)
        _os_write(prnpath, buffer, &size);
} while(size > 0);

_os_close(dskpath);
_os_close(prnpath);

return SUCCESS;

The Subroutine Library

Subroutine Module

Below is an example of a subroutine module with two entrypoints. The subroutine
module consists of two files: the root psect file (in assembly language), and the
subroutine module function file (in C language).

root psect File

*substart.a: Generic psect for OS-9 subroutine modules use <oskdefs.d>

Edition equ 1

Typ_Lang set (Sbrtn<<8)+Objct
Attr_Rev set ((ReEnt+SupStat)<<8)+0
psect hcstart_a,Typ_Lang,Attr_Rev,Edition,0,0,exec
* This portion lists the functions that exist in the subroutine module.
* This is essentially an array of function pointers.
exec:
dc.l sub_init-btext
dc.l sub_function_1-btext
dc.l 0
ends
function File
/
** This is a "C" file that implements sub_init and sub_function_1. Remember to turn off stack checking (-r) and compile with -bapg. Global variables are ok to access in this file, since the calling routine will set up your global pointer. */
error_code sub_init(int init_param)
{
    ...
    return SUCCESS;
}
error_code sub_function_1(u_int32 *ticks)
{
    *ticks = ...;

    return SUCCESS;
}

Application Call into a Subroutine Module
This section describes how an application calls into a subroutine library. It also describes how to initialize a subroutine module and how to discontinue using it.

Initialization
Before you can begin using a subroutine library, you must first link to it using the _os_slink call, as several values returned from _os_slink must be stored. Below is an example of how to link to a subroutine module and store the necessary returned values.

#include <module.h>
#include <types.h>
#include <errno.h>
#include <stdlib.h>
#include <regs.h>

typedef struct {
    u_int32 *exec;     /* execution point for hcsb */
    void    *gp;   /* global (static storage) pointer for hcsb */
    mh_com  *mod_head; /* module header */
    int     subnum;    /* subroutine number (0..15) */
} SUB_DATA;
/* defines array index of subroutine module function */
#define SUB_INIT 0
#define SUB_FUNCTION_1 1

/* this links to the subroutine module called 'mysub' and stores ** the module head, execution offset, global pointer, and ** subroutine number in the SUB_DATA structure.*/
error_code _sub_link(SUB_DATA *hc, int subnum)
{
    hc->subnum = subnum;
    return _os_slink(subnum,"mysub",(void**)&hc->exec,&hc->gp,&hc->mod_head);
}

/* this unlinks from the subroutine module. Only call this when ** you are all done using 'mysub'. */
error_code _sub_unlink(SUB_DATA *hc)
{
    void *ptr;
    error_code err;

    err = _os_slink(hc->subnum,NULL,&ptr,&ptr,(mh_com**)&ptr);

    return err;
}

Calling into the subroutine module: The following shows functions that setup and call into the subroutine module. These functions may be placed into a library if several applications are to access the subroutine module.

/* This calls the 'sub_init' entrypoint into the subroutine ** module. This will pass one parameter to the init function: **'init_param'. */
error_code _sub_init(SUB_DATA *hc, int init_param)
{
    error_code (*func)();
    error_code err;
    void *oldstatics;

    func = (error_code(*))((u_int8*)hc->mod_head + hc->exec[SUB_INIT]);
oldstatics = change_static(hc->gp);
err = func();
(void) change_static(oldstatics);

return err;
}

/*
 ** This calls into the "sub_function_1' entrypoint of the subroutine module.
 ** This passes one parameter: 'ticks'.
 */
error_code _sub_function_1(SUB_DATA *hc, u_int32 *ticks)
{
    error_code (*func)(u_int32*);
    error_code err;
    void *oldstatics;

    func = (error_code(*)(u_int32*))((u_int8*)hc->mod_head + hc->exec[SUB_FUNCTION_1]);
    oldstatics = change_static(hc->gp);
    err = func(ticks);
    (void) change_static(oldstatics);

    return err;
}

Trap Handlers

The following example trap handler consists of four files: trapc.a, thandler.c, tcall.c, and ttest.c.

trapc.a

name OS-9000 80386 Example System State Trap Handler

use <oskdefs.d>

type equ (TrapLib<<8)+Objct
revs equ ((ReEnt+Ghost+SupStat)<<8)
edit equ 1
stack equ 1024

psect Trap_9000,type,revs,edit,stack,_trap_entry
_m_init: equ _trap_init      * Trap Handler initialization entry point
_m_term: equ _trap_term       * Trap Handler termination entry point

_sysedit: equ edit edition number of module

E_ILLFNC equ $40    Illegal trap handler function code error

vsect
_caller_eip: ds.l 1         caller's return pc
_caller_statics: ds.l 1     caller's static storage pointer (%ebx)
ends

******************************************************************
* _trap_entry - trap handler entry point code.
*
* input:  0(%esp) = caller's static storage pointer (%ebx)
*        4(%esp) = trap number
*        6(%esp) = function code
*        8(%esp) = return address
*
_trap_entry: push.l %eax save registers
            push.l %esi
stacked set 2*4
            sub.l %eax,%eax sweep register
            mov.w 6+stacked(%esp),%eax get function code
            cmp.l trap_max(%ebx),%eax function code in range?
            jge.b _bad_trap branch if not
            lea trap_dsptable(%ebx),%esi get trap dispatch table
            mov.l (%esi,%eax*4),%eax get routine address
            mov.l %eax,4+stacked(%esp) set routine address
            pop.l %esi restore registers
            pop.l %eax
            pop.l _caller_statics(%ebx) save caller's static storage
* call trap handler function

OS-9 Technical Manual 409
ret

_bad_trap pop.l %esi restore registers
    pop.l %eax
    lea 2*4(%esp),%esp pop stack
    mov.l #E_ILLFNC,%eax return error code
    ret

ends

thandler.c

#include <const.h>
/* pre-declare trap handler functions */
int func1(), func2(), func3();

/* initialize maximum function count variable */
int trap_max = 3;

/* initialize trap handler dispatch table */
(* trap_dsptable[])() = { 
    func1,
    func2,
    func3
};

/* _trap_init - trap handler initialization routine. */
_trap_init(trapnum, memsize, statics)
    register int trapnum;           /* trap handler number */
    register int memsize;      /* addtional trap handler memory size */
    register void *statics;         /* caller’s static storage pointer */
{
    return SUCCESS;
}

/* _trap_term - trap handler termination routine. */
_trap_term(trapnum, statics)
    register int trapnum;           /* trap handler number */
    register void *statics;         /* caller’s static storage pointer */
Appendix A: Example Code

```c
{    
    return SUCCESS;
}

/* func1 - first trap handler function. */
func1()
{
    return 1;
}

/* func2 - second trap handler function. */
func2()
{
    return 2;
}

/* func3 - third trap handler function. */
func3()
{
    return 3;
}

tcall.c

_asm ("*******
* tcall - macro definition
* tcall trap, function
* tcall macro
   dc.w $fece
   dc.w 00
   dc.w 01
   dc.w 02
   ret
   dc.b $00
   endm
   trap_func1: tcall 8,0
   trap_func2: tcall 8,1
   trap_func3: tcall 8,2
");
```
Appendix A: Example Code

test.c

#include <stdio.h>
#include <errno.h>

#ifndef SUCCESS
#define SUCCESS 0
#endif

char *libexec;
char *modhead;

/* _trapinit - trap handler exception routine, install trap handler. */
_trapinit(trapnum, funcode)
register int trapnum;
register int funcode;
{
    register int err;

    /* validate trap number */
    if (trapnum != 8) return errno = EOS_ITRAP;

    /* install the trap handler */
    if ((err = _os_tlink(8, "trap9000", &libexec, &modhead, 0, 0)) != SUCCESS)
        return errno = err;

    return SUCCESS;
}

main()
{
    printf("calling function %d\n", trap_func1());
    printf("calling function %d\n", trap_func2());
    printf("calling function %d\n", trap_func3());
}
OS-9 Error Codes

This section lists OS-9 error codes in numerical order. The first three numbers indicate a group of messages. Processor-specific error messages can also be added with each processor family port. If this manual has not been updated to include the messages for your processor, see the errmsg file in the OS9000/SRC/SYS/ERRMSG directory. This appendix includes the following topics:

- Error Categories
- Errors
## Error Categories

OS-9 error codes are grouped in the following categories:

**Table B-1. OS-9 Error Code Categories**

<table>
<thead>
<tr>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:001 - 000:031</td>
<td>Miscellaneous Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-2.</td>
</tr>
<tr>
<td>000:032 - 000:047</td>
<td>Ultra C Related Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-3.</td>
</tr>
<tr>
<td>000:060 - 000:069</td>
<td>Miscellaneous Program Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-4.</td>
</tr>
<tr>
<td>000:080 - 000:089</td>
<td>Miscellaneous OS Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-5.</td>
</tr>
<tr>
<td>000:102 - 000:132</td>
<td>Reserved Errors</td>
</tr>
<tr>
<td>000:134 - 000:163</td>
<td>Refer to Table B-6.</td>
</tr>
<tr>
<td>000:133</td>
<td>Uninitialized User Trap (1-15) Error</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-6.</td>
</tr>
<tr>
<td>000:164 - 000:239</td>
<td>Operating System Errors</td>
</tr>
<tr>
<td></td>
<td>(usually generated by the kernel or file managers)</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-7.</td>
</tr>
<tr>
<td>000:240 - 000:255</td>
<td>I/O Errors (generated by device drivers or file managers)</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-8.</td>
</tr>
<tr>
<td>000:256</td>
<td>ANSI C math out-of-range error</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-9.</td>
</tr>
<tr>
<td>001:000 - 001:099</td>
<td>Compiler Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-10.</td>
</tr>
<tr>
<td>006:100 - 006:206</td>
<td>RAVE Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-11.</td>
</tr>
<tr>
<td>007:001 - 007:029</td>
<td>Internet Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-12.</td>
</tr>
<tr>
<td>008:257 - 008:288</td>
<td>IEEE1394 Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-13.</td>
</tr>
<tr>
<td>008:321 - 008:339</td>
<td>SVLAN Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-14.</td>
</tr>
<tr>
<td>020:001 - 020:022</td>
<td>POSIX Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-15.</td>
</tr>
<tr>
<td>100:000 - 100:999</td>
<td>PowerPC Processor-specific Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-16.</td>
</tr>
<tr>
<td>102:000 - 102:032</td>
<td>MIPS Processor-specific Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-17.</td>
</tr>
<tr>
<td>103:000 - 103:008</td>
<td>ARM Processor-specific Errors</td>
</tr>
<tr>
<td></td>
<td>Refer to Table B-18.</td>
</tr>
<tr>
<td>104:002 - 104:009</td>
<td>SuperH Processor-specific Errors</td>
</tr>
<tr>
<td>104:011 - 104:115</td>
<td>Refer to Table B-19.</td>
</tr>
<tr>
<td>104:200 - 104:204</td>
<td></td>
</tr>
</tbody>
</table>
Errors

The following OS-9 error codes are defined in the `errno.h` file.

### Table B-2. Miscellaneous Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:001</td>
<td>Process has aborted.</td>
<td></td>
</tr>
<tr>
<td>000:002</td>
<td>S_Abort signal</td>
<td>Keyboard quit (^E) typed.</td>
</tr>
<tr>
<td>000:003</td>
<td>S_Intrpt signal</td>
<td>Keyboard interrupt (^C) typed.</td>
</tr>
<tr>
<td>000:004</td>
<td>S_HangUp signal</td>
<td>Modem hangup.</td>
</tr>
</tbody>
</table>

### Table B-3. Ultra C Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:032</td>
<td>EOS_SIGABRT</td>
<td>An abort signal was received.</td>
</tr>
<tr>
<td>000:033</td>
<td>EOS_SIGFPE</td>
<td>An erroneous math operation signal was received.</td>
</tr>
<tr>
<td>000:034</td>
<td>EOS_SIGILL</td>
<td>An illegal function image signal was received.</td>
</tr>
<tr>
<td>000:035</td>
<td>EOS_SIGSEGV</td>
<td>A segment violation (bus error) signal was received.</td>
</tr>
<tr>
<td>000:036</td>
<td>EOS_SIGTERM</td>
<td>A termination request signal was received.</td>
</tr>
<tr>
<td>000:037</td>
<td>EOS_SIGALRM</td>
<td>An alarm time elapsed signal was received.</td>
</tr>
<tr>
<td>000:038</td>
<td>EOS_SIGPIPE</td>
<td>A write to pipe with no readers signal was received.</td>
</tr>
<tr>
<td>000:039</td>
<td>EOS_SIGUSR1</td>
<td>A user signal #1 was received.</td>
</tr>
<tr>
<td>000:040</td>
<td>EOS_SIGUSR2</td>
<td>A user signal #2 was received.</td>
</tr>
<tr>
<td>000:041</td>
<td>EOS_SIGCHECK</td>
<td>A machine check exception signal was received.</td>
</tr>
<tr>
<td>000:042</td>
<td>EOS_SIGALIGN</td>
<td>An alignment exception signal was received.</td>
</tr>
<tr>
<td>000:043</td>
<td>EOS_SIGINST</td>
<td>An instruction access exception signal was received.</td>
</tr>
<tr>
<td>000:044</td>
<td>EOS_SIGPRIV</td>
<td>A privilege violation exception signal was received.</td>
</tr>
</tbody>
</table>

### Table B-4. Miscellaneous Program Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:064</td>
<td>EOS_ILLFNC</td>
<td>Illegal function code.</td>
</tr>
<tr>
<td>000:065</td>
<td>EOS_FMTERR</td>
<td>ASCII to numeric format conversion error.</td>
</tr>
<tr>
<td>000:066</td>
<td>EOS_NOTNUM</td>
<td>Number not found.</td>
</tr>
<tr>
<td>000:067</td>
<td>EOS_ILLARG</td>
<td>Illegal argument.</td>
</tr>
<tr>
<td>000:067</td>
<td>EINVAL</td>
<td>Invalid argument (POSIX).</td>
</tr>
</tbody>
</table>
### Table B-5. Miscellaneous Operating System Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:080</td>
<td>EOS_MEMINUSE</td>
<td>Memory already in use.</td>
</tr>
<tr>
<td>000:081</td>
<td>EOS_UNKADDR</td>
<td>Do not know how to translate.</td>
</tr>
</tbody>
</table>

### Table B-6. Operating System Reserved Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:102</td>
<td>EOS_BUSERR</td>
<td>A bus trap error occurred.</td>
</tr>
<tr>
<td>000:103</td>
<td>EOS_ADRERR</td>
<td>An address trap error occurred.</td>
</tr>
<tr>
<td>000:104</td>
<td>EOS_ILLINS</td>
<td>An illegal instruction exception occurred.</td>
</tr>
<tr>
<td>000:105</td>
<td>EOS_ZERDIV</td>
<td>A zero divide exception occurred.</td>
</tr>
<tr>
<td>000:106</td>
<td>EOS_CHK</td>
<td>A <code>chk</code> or <code>chk2</code> instruction trap occurred.</td>
</tr>
<tr>
<td>000:107</td>
<td>EOS_TRAPV</td>
<td>A <code>trapv</code> or <code>trapcc</code> instruction occurred.</td>
</tr>
<tr>
<td>000:108</td>
<td>EOS_VIOLAT</td>
<td>A privileged instruction violation occurred.</td>
</tr>
<tr>
<td>000:109</td>
<td>EOS_TRACE</td>
<td>An uninitialized Trace exception occurred.</td>
</tr>
<tr>
<td>000:110</td>
<td>EOS_1010</td>
<td>A 1010 instruction exception occurred.</td>
</tr>
<tr>
<td>000:111</td>
<td>EOS_1111</td>
<td>A 1111 instruction exception occurred.</td>
</tr>
<tr>
<td>000:112</td>
<td>EOS_RESRVD</td>
<td>An invalid exception occurred (#12).</td>
</tr>
<tr>
<td>000:113</td>
<td>EOS_CPROTO</td>
<td>Coprocessor protocol violation.</td>
</tr>
<tr>
<td>000:114</td>
<td>EOS_STKFMT</td>
<td>System stack frame format error.</td>
</tr>
<tr>
<td>000:115</td>
<td>EOS_UNIRQ</td>
<td>An uninitialized interrupted.</td>
</tr>
<tr>
<td>000:116 - 000:123</td>
<td>An invalid exception occurred (#16 - #23).</td>
<td></td>
</tr>
<tr>
<td>000:124</td>
<td>EOS_TRAP</td>
<td>Spurious Interrupt occurred.</td>
</tr>
<tr>
<td>000:133</td>
<td>EOS_CHIP</td>
<td>An uninitialized user TRAP (1-15) was executed.</td>
</tr>
<tr>
<td>000:148</td>
<td>EOS_FPUNORDC</td>
<td>Floating point coprocessor unordered condition.</td>
</tr>
<tr>
<td>000:149</td>
<td>EOS_FPINXACT</td>
<td>Floating point coprocessor inexact result.</td>
</tr>
<tr>
<td>000:150</td>
<td>EOS_FPDIVZER</td>
<td>Floating point coprocessor divide by zero.</td>
</tr>
<tr>
<td>000:151</td>
<td>EOS_FPUOVRFL</td>
<td>Floating point coprocessor overflow.</td>
</tr>
<tr>
<td>000:152</td>
<td>EOS_FPOVRFL</td>
<td>Floating point coprocessor overflow.</td>
</tr>
<tr>
<td>000:153</td>
<td>EOS_FPNOTNUM</td>
<td>Floating point coprocessor not a number.</td>
</tr>
<tr>
<td>000:155</td>
<td>EOS_FPUDATA</td>
<td>FP Unsupported data type.</td>
</tr>
<tr>
<td>000:156</td>
<td>EOS_MMUCONF</td>
<td>PMMU Configuration exception.</td>
</tr>
<tr>
<td>000:157</td>
<td>EOS_MMUILLEG</td>
<td>PMMU Illegal Operation exception.</td>
</tr>
<tr>
<td>000:158</td>
<td>EOS_MMUACCESS</td>
<td>PMMU Access Level Violation exception.</td>
</tr>
<tr>
<td>000:159 - 000:163</td>
<td>An invalid exception occurred (#59 - #63).</td>
<td></td>
</tr>
</tbody>
</table>
# Table B-7. OS-9-Specific Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:164</td>
<td>EOS_PERMIT</td>
<td>No permission. A user process has attempted something that can only be done by a super user.</td>
</tr>
<tr>
<td>000:164</td>
<td>EACCESS</td>
<td>POSIX access denied.</td>
</tr>
<tr>
<td>000:165</td>
<td>EOS_DIFFER</td>
<td>The arguments to F_CHKNam do not match.</td>
</tr>
<tr>
<td>000:166</td>
<td>EOS_STKOVF</td>
<td>System stack overflow. F_ChkNam can return this error if the pattern string is too complex.</td>
</tr>
<tr>
<td>000:167</td>
<td>EOS_EVNTID</td>
<td>Invalid or Illegal event ID number.</td>
</tr>
<tr>
<td>000:168</td>
<td>EOS_EVNF</td>
<td>Event name not found.</td>
</tr>
<tr>
<td>000:169</td>
<td>EOS_EVBUSY</td>
<td>The event is busy (and can’t be deleted).</td>
</tr>
<tr>
<td>000:170</td>
<td>EOS_EVPARAM</td>
<td>Impossible event parameters supplied.</td>
</tr>
<tr>
<td>000:171</td>
<td>EOS_DAMAGE</td>
<td>System data structures have been damaged.</td>
</tr>
<tr>
<td>000:172</td>
<td>EOS_BADREV</td>
<td>Module revision is incompatible with operating system.</td>
</tr>
<tr>
<td>000:173</td>
<td>EOS_PTHLOST</td>
<td>Path became lost because network node was down.</td>
</tr>
<tr>
<td>000:174</td>
<td>EOS_BADPART</td>
<td>Bad (disk) partition data, or no active partition.</td>
</tr>
<tr>
<td>000:175</td>
<td>EOS_HARDWARE</td>
<td>Hardware damage has been detected.</td>
</tr>
<tr>
<td>000:176</td>
<td>EOS_NOTME</td>
<td>Not my device. Error returned by an interrupt service routine when it is polled for an interrupt its device did not cause.</td>
</tr>
<tr>
<td>000:177</td>
<td>EOS_BSIG</td>
<td>Fatal signal or no intercept routine. Process received a fatal signal or did not have an intercept function.</td>
</tr>
<tr>
<td>000:178</td>
<td>EOS_BUF2SMALL</td>
<td>The buffer passed is too small. A routine was passed a buffer too small to hold the data requested.</td>
</tr>
<tr>
<td>000:179</td>
<td>EOS_ISUB</td>
<td>Illegal/used subroutine module number.</td>
</tr>
<tr>
<td>000:180</td>
<td>EOS_EVTFUL</td>
<td>Event descriptor table full.</td>
</tr>
<tr>
<td>000:196</td>
<td>EOS_SYMLINK</td>
<td>Symbolic link found in path list. A link was attempted that would have caused recursion in the file structure. You may not link to a directory containing the real directory.</td>
</tr>
<tr>
<td>000:197</td>
<td>EOS_EOLIST</td>
<td>End of alias list.</td>
</tr>
<tr>
<td>000:198</td>
<td>EOS_LOCKID</td>
<td>Illegal I/O lock identifier specified. Usually this error occurs because a user has initialized a device for use with more than one file manager.</td>
</tr>
<tr>
<td>000:199</td>
<td>EOS_NOLOCK</td>
<td>Lock not obtained.</td>
</tr>
<tr>
<td>000:200</td>
<td>EOS_PTHFUL</td>
<td>The user’s (or system) path table is full. Usually this error occurs because a user program has tried to open more than 32 I/O paths simultaneously. It might also occur if the system path table becomes full and can not be expanded.</td>
</tr>
</tbody>
</table>
Table B-7. OS-9-Specific Error Codes  (Continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:201</td>
<td>EOS_BPNUM</td>
<td>Bad path number. An I/O request has been made with an invalid path number, or one not currently open.</td>
</tr>
<tr>
<td>000:201</td>
<td>EBADF</td>
<td>Bad file descriptor (POSIX).</td>
</tr>
<tr>
<td>000:203</td>
<td>EOS_BMODE</td>
<td>Bad I/O mode. An attempt has been made to perform I/O on a path incapable of supporting it. For example, writing to a path open for input.</td>
</tr>
<tr>
<td>000:204</td>
<td>EOS_DEVOVF</td>
<td>The system’s device table is full. To install another device descriptor, one must first be removed. The system init module can be changed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to allow more devices.</td>
</tr>
<tr>
<td>000:205</td>
<td>EOS_BMID</td>
<td>Bad module ID. An attempt has been made to load a module without a valid module header.</td>
</tr>
<tr>
<td>000:206</td>
<td>EOS_DIRFUL</td>
<td>The module directory is full. No more modules can be loaded or created unless one is first unlinked. Although OS-9 automatically expands</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the module directory when it becomes full, this error may be returned because the there is not enough memory or the memory is too fragmented to use.</td>
</tr>
<tr>
<td>000:207</td>
<td>EOS_MEMFUL</td>
<td>Memory full. This error is returned from the F_SRqMem service call when there is not enough system RAM to fulfill the request, or if a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>process has already been allocated the maximum number of blocks permitted by the system.</td>
</tr>
<tr>
<td>000:208</td>
<td>EOS_UNKSVC</td>
<td>Unknown service code. An OS-9 call specified an unknown or invalid service code, or a getstat/setstat call was made with an unknown status code.</td>
</tr>
<tr>
<td>000:209</td>
<td>EOS_MODBSY</td>
<td>The module is busy. An attempt has been made to access (through F_Link) a non-sharable module or non-sharable device already in use.</td>
</tr>
<tr>
<td>000:210</td>
<td>EOS_BPADDR</td>
<td>Bad page address. A memory de-allocation request has been given a buffer pointer or size that is invalid, often because it references</td>
</tr>
<tr>
<td></td>
<td></td>
<td>memory that has not been allocated to the caller. The system detects trouble when the buffer is returned to free memory or if it is used</td>
</tr>
<tr>
<td></td>
<td></td>
<td>as the destination of a data transfer, such as I_Read.</td>
</tr>
<tr>
<td>000:210</td>
<td>EFAULT</td>
<td>Bad address (POSIX).</td>
</tr>
<tr>
<td>Number</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>000:211</td>
<td>EOS_EOF</td>
<td>The end of file has been reached. An end of file condition was encountered on a read operation.</td>
</tr>
<tr>
<td>000:211</td>
<td>EPIPE</td>
<td>Broken pipe (POSIX).</td>
</tr>
<tr>
<td>000:212</td>
<td>EOS_VCTBSY</td>
<td>IRQ vector is busy. A device has tried to install itself in the IRQ table to handle a vector claimed by another device.</td>
</tr>
<tr>
<td>000:213</td>
<td>EOS_NES</td>
<td>Non-existing segment. A search was made for a disk file segment that cannot be found. The device could have a damaged file structure.</td>
</tr>
<tr>
<td>000:214</td>
<td>EOS_FNA</td>
<td>File not accessible. An attempt to open a file failed. The file was found, but is inaccessible in the requested mode. Check the file’s owner ID and access attributes.</td>
</tr>
<tr>
<td>000:214</td>
<td>EPERM</td>
<td>Operation not permitted (POSIX).</td>
</tr>
<tr>
<td>000:215</td>
<td>EOS_BPNAM</td>
<td>Bad pathlist specified. The specified pathlist has a syntax error, for example, an illegal character.</td>
</tr>
<tr>
<td>000:216</td>
<td>EOS_PNNF</td>
<td>File not found. The specified pathlist does not lead to any known file.</td>
</tr>
<tr>
<td>000:216</td>
<td>ENOENT</td>
<td>No such file or directory (POSIX).</td>
</tr>
<tr>
<td>000:217</td>
<td>EOS_SLF</td>
<td>File segment list is full. A file has become too fragmented to accommodate further growth. This can occur on a nearly full disk, or one whose free space has become scattered. The simplest way to solve the problem is to copy the file, which should move it into more contiguous space.</td>
</tr>
<tr>
<td>000:218</td>
<td>EOS_CEF</td>
<td>Tried to create an existing file. The specified filename already appears in the current directory.</td>
</tr>
<tr>
<td>000:218</td>
<td>EEXIST</td>
<td>File exists (POSIX).</td>
</tr>
<tr>
<td>000:219</td>
<td>EOS_IBA</td>
<td>Illegal memory block specified. The system was called to return memory, but was passed an invalid pointer or block size.</td>
</tr>
<tr>
<td>000:220</td>
<td>EOS_HANGUP</td>
<td>Telephone (modem) connection terminated. This error is returned when an I/O operation is attempted on a path after irrecoverable line problems have occurred, such as data carrier lost. It may be returned from network devices, if the network connection is lost.</td>
</tr>
<tr>
<td>000:221</td>
<td>EOS_MNF</td>
<td>Module not found. An F_Link call was made to a module not in memory. Modules with corrupted or modified headers will not be found.</td>
</tr>
</tbody>
</table>
### Table B-7. OS-9-Specific Error Codes (Continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:222</td>
<td>EOS_NOCLK</td>
<td>No system clock. A request was made requiring a system clock, but one is not running. For example, a timed F_Sleep call has been requested, but the clock was not running. The <code>setime</code> utility is used to start the system clock.</td>
</tr>
<tr>
<td>000:223</td>
<td>EOS_DELSP</td>
<td>Deleting stack memory. A process tried to return the memory containing it's current stack pointer to the system. This is also known as a suicide attempt.</td>
</tr>
<tr>
<td>000:224</td>
<td>EOS_IPRCID</td>
<td>Illegal process ID. A system call was passed a process ID to a non-existent or inaccessible process.</td>
</tr>
<tr>
<td>000:225</td>
<td>ESRCH</td>
<td>No such process (POSIX).</td>
</tr>
<tr>
<td>000:226</td>
<td>EOS_PARAM</td>
<td>Bad parameter. A system call was passed an illegal or impossible parameter.</td>
</tr>
<tr>
<td>000:226</td>
<td>EOS_NOCHLD</td>
<td>No children. An F_Wait call was made with no child processes to wait for.</td>
</tr>
<tr>
<td>000:226</td>
<td>ECHILD</td>
<td>No child process (POSIX)</td>
</tr>
<tr>
<td>000:227</td>
<td>EOS_ITRAP</td>
<td>Invalid trap number. An F_Tlink call was made with an invalid user trap code or one already in use.</td>
</tr>
<tr>
<td>000:228</td>
<td>EOS_PRCABT</td>
<td>The process has been aborted.</td>
</tr>
<tr>
<td>000:229</td>
<td>EOS_PRCFUL</td>
<td>Too many active processes. The system's process table is full. (Too many processes are currently running.) The kernel automatically tries to expand the process table, but returns this error if there is not enough contiguous memory to do so.</td>
</tr>
<tr>
<td>000:230</td>
<td>EOS_IFORKP</td>
<td>Illegal fork parameter (not currently used)</td>
</tr>
<tr>
<td>000:231</td>
<td>EOS_KWNMOD</td>
<td>Known module. A call was made to install a module that is already in memory.</td>
</tr>
<tr>
<td>000:232</td>
<td>EOS_BMCRC</td>
<td>Bad module CRC. A CRC calculation is performed on every module when it is installed in the system module directory. Only modules with good CRCs are accepted. To generate a valid CRC value in an intentionally altered module, use the <code>fixmod</code> utility.</td>
</tr>
<tr>
<td>000:233</td>
<td>EOS_SIGNAL</td>
<td>Signal error (replaces EOS_USIGP)</td>
</tr>
<tr>
<td>000:234</td>
<td>EOS_NEMOD</td>
<td>Non executable module.</td>
</tr>
<tr>
<td>000:235</td>
<td>EOS_BNAM</td>
<td>Bad name. This error is returned by the F_PrsNam system call if there is a syntax error in the name.</td>
</tr>
</tbody>
</table>
Appendix B: OS-9 Error Codes

Table B-7. OS-9-Specific Error Codes (Continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:236</td>
<td>EOS_BMHP</td>
<td>Bad module header parity.</td>
</tr>
<tr>
<td>000:237</td>
<td>EOS_NORAM</td>
<td>No RAM available. A process has made an F_Mem request to expand its memory size. F_Mem is no longer supported and F_SrqMem should be used. This error may also be returned if there is not enough contiguous memory to process a fork request or if a device driver does not specify any static storage requirements.</td>
</tr>
<tr>
<td>000:237</td>
<td>ENOMEM</td>
<td>Insufficient RAM (POSIX).</td>
</tr>
<tr>
<td>000:238</td>
<td>EOS_DNE</td>
<td>The directory is not empty. The directory attribute of a file cannot be removed unless the directory is empty. This prevents accidental loss of disk space.</td>
</tr>
<tr>
<td>000:239</td>
<td>EOS_NOTASK</td>
<td>No available task number. All of the task numbers are currently in use and a request was made to execute or create a new task. This error could be returned by a system security module (SSM).</td>
</tr>
</tbody>
</table>

Table B-8. OS-9-Specific I/O Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:240</td>
<td>EOS_UNIT</td>
<td>Illegal unit (drive) number.</td>
</tr>
<tr>
<td>000:241</td>
<td>EOS_SECT</td>
<td>Bad disk sector number.</td>
</tr>
<tr>
<td>000:242</td>
<td>EOS_WP</td>
<td>Media is write protected.</td>
</tr>
<tr>
<td>000:243</td>
<td>EOS_CRC</td>
<td>Bad module cyclic redundancy check value. A CRC error occurred on read or write verity.</td>
</tr>
<tr>
<td>000:244</td>
<td>EOS_READ</td>
<td>Read error. A data transfer error occurred during a disk read operation, or an SCF (terminal) input buffer overrun.</td>
</tr>
<tr>
<td>000:244</td>
<td>EIO</td>
<td>POSIX I/O error.</td>
</tr>
<tr>
<td>000:245</td>
<td>EOS_WRITE</td>
<td>A hardware error occurred during a disk write operation.</td>
</tr>
<tr>
<td>000:246</td>
<td>EOS_NOTRDY</td>
<td>Device not ready.</td>
</tr>
<tr>
<td>000:246</td>
<td>ENODEV</td>
<td>No such device (POSIX).</td>
</tr>
<tr>
<td>000:246</td>
<td>ENXIO</td>
<td>No such device or address (POSIX).</td>
</tr>
<tr>
<td>000:247</td>
<td>EOS_SEEK</td>
<td>Seek error. A physical seek operation was unable to find the specified sector.</td>
</tr>
<tr>
<td>000:248</td>
<td>EOS_FULL</td>
<td>Media has insufficient free space.</td>
</tr>
<tr>
<td>000:249</td>
<td>EOS_BTYP</td>
<td>Bad type (incompatible media). A read operation was attempted on incompatible media. For example, a read operation for a double-sided disk was tried on a single-sided disk.</td>
</tr>
</tbody>
</table>
### Table B-8. OS-9-Specific I/O Error Codes (Continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:250</td>
<td>EOS_DEVBSY</td>
<td>A non-sharable device is in use.</td>
</tr>
<tr>
<td>000:250</td>
<td>EBUSY</td>
<td>POSIX resource busy.</td>
</tr>
<tr>
<td>000:251</td>
<td>EOS_DIDC</td>
<td>Disk ID change. RBF copies the disk ID number (from sector zero) into the path descriptor of each path when it is opened. If it does not agree with the driver's current disk ID, this error is returned. The driver updates the current disk ID only when sector zero is read; it is possible to swap disks without RBF noticing; this prevents this possibility.</td>
</tr>
<tr>
<td>000:252</td>
<td>EOS_LOCK</td>
<td>Another process is accessing the record. Normal record locking routines wait for a record in use by another user to become available. However, RBF may be told (through a <code>SetStat</code> call) to wait for a finite amount of time. If the time expires before the record becomes free, this is returned.</td>
</tr>
<tr>
<td>000:253</td>
<td>EOS_SHARE</td>
<td>Non-sharable file/device is busy. The requested file or device has the single user bit set or was opened in single user mode and another process is accessing the file. This is returned when an attempt is made to delete an open file.</td>
</tr>
<tr>
<td>000:254</td>
<td>EOS_DEADLK</td>
<td>I/O deadlock error, returned when two or more processes are waiting for each other to release I/O resources before they can proceed. One must release control to enable the other to proceed.</td>
</tr>
<tr>
<td>000:255</td>
<td>EOS_FORMAT</td>
<td>Device is format protected. This error occurs when an attempt is made to format a format-protected disk. A bit in the device descriptor may be changed to allow the device to be formatted. Formatting is usually inhibited on hard disks to prevent accidental erasure.</td>
</tr>
</tbody>
</table>

### Table B-9. OS-9-Specific ANSI C Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000:256</td>
<td>ERANGE</td>
<td>ANSI C math out of range error.</td>
</tr>
</tbody>
</table>

### Table B-10. OS-9-Specific Compiler Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>001:000</td>
<td>ERANGE</td>
<td>ANSI C Number out of range error.</td>
</tr>
<tr>
<td>001:001</td>
<td>EDOM</td>
<td>ANSI C Number Not in Domain.</td>
</tr>
</tbody>
</table>
### Table B-11. OS-9-Specific RAVE Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>006:000</td>
<td>EOS_ILLPRM</td>
<td>An illegal parameter was passed to a function.</td>
</tr>
<tr>
<td>006:001</td>
<td>EOS_IDFULL</td>
<td>Identifier (ID) table full.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An ID table could not be expanded.</td>
</tr>
<tr>
<td>006:002</td>
<td>EOS_BADSIZ</td>
<td>Bad size error.</td>
</tr>
<tr>
<td>006:003</td>
<td>EOS_RGFULL</td>
<td>Region definition full (overflow).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The region is too complex.</td>
</tr>
<tr>
<td>006:004</td>
<td>EOS_UNID</td>
<td>Unallocated identifier number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An attempt was made to use an ID number for an object (drawmap, action region, etc.) that was not allocated.</td>
</tr>
<tr>
<td>006:005</td>
<td>EOS_NULLRG</td>
<td>Null region.</td>
</tr>
<tr>
<td>006:006</td>
<td>EOS_BADMOD</td>
<td>Bad drawmap/pattern mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An illegal mode was passed to create a drawmap or pattern.</td>
</tr>
<tr>
<td>006:007</td>
<td>EOS_NOFONT</td>
<td>No active font.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No font was activated when an attempt to output text was made.</td>
</tr>
<tr>
<td>006:008</td>
<td>EOS_NODM</td>
<td>No drawmap.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No character output drawmap was available when attempting an __os_write or __os_writeln call.</td>
</tr>
<tr>
<td>006:009</td>
<td>EOS_NOPLAY</td>
<td>No audio play in progress.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An attempt was made to stop an audio play when none was in progress.</td>
</tr>
<tr>
<td>006:010</td>
<td>EOS_ABORT</td>
<td>Asynchronous operation aborted.</td>
</tr>
<tr>
<td>006:011</td>
<td>EOS_QFULL</td>
<td>Audio queue is full.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The driver queue could not handle the number of soundmaps you were attempting to output.</td>
</tr>
<tr>
<td>006:012</td>
<td>EOS_BUSY</td>
<td>Audio processor is busy.</td>
</tr>
</tbody>
</table>

### Table B-12. OS-9-Specific Internet Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>007:001</td>
<td>EWOULDBLOCK</td>
<td>I/O operation would block.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An operation was attempted that would cause a process to block on a socket in non-blocking mode.</td>
</tr>
<tr>
<td>007:001</td>
<td>EAGAIN</td>
<td>POSIX item temporarily available.</td>
</tr>
<tr>
<td>007:002</td>
<td>EINPROGRESS</td>
<td>I/O operation now in progress.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An operation taking a long time to complete was performed, such as a connect() call, on a socket in non-blocking mode.</td>
</tr>
</tbody>
</table>
### Table B-12. OS-9-Specific Internet Error Codes (Continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>007:003</td>
<td>EALREADY</td>
<td>Operation already in progress. An operation was attempted on a non-blocking object that already had an operation in progress.</td>
</tr>
<tr>
<td>007:003</td>
<td>EINTR</td>
<td>Interrupted function call (POSIX).</td>
</tr>
<tr>
<td>007:004</td>
<td>EDESTADDRREQ</td>
<td>Destination address required. The attempted socket operation requires a destination address.</td>
</tr>
<tr>
<td>007:005</td>
<td>EMSGSIZE</td>
<td>Message too long. A message sent on a socket was larger than the internal message buffer or some other network limit.</td>
</tr>
<tr>
<td>007:006</td>
<td>EPROTOTYPE</td>
<td>Protocol wrong type for socket. A protocol was specified that does not support the semantics of the socket type requested.</td>
</tr>
<tr>
<td>007:007</td>
<td>ENOPROTOOPT</td>
<td>Bad protocol option. A bad option or level was specified in a getsockopt() or setsockopt() call.</td>
</tr>
<tr>
<td>007:008</td>
<td>EPROTONOSUPPORT</td>
<td>Protocol not supported. The requested protocol is not available or not configured for use.</td>
</tr>
<tr>
<td>007:009</td>
<td>ESOCKNOSUPPORT</td>
<td>Socket type not supported. The requested socket type is not supported or not configured for use.</td>
</tr>
<tr>
<td>007:010</td>
<td>EOPNOTSUPP</td>
<td>Operation unsupported on socket.</td>
</tr>
<tr>
<td>007:011</td>
<td>EPFNOSUPPORT</td>
<td>Protocol family not supported.</td>
</tr>
<tr>
<td>007:012</td>
<td>EAFNOSUPPORT</td>
<td>Address family unsupported by protocol.</td>
</tr>
<tr>
<td>007:013</td>
<td>EADDRINUSE</td>
<td>Address already in use. Only one use of each address is normally permitted. Wildcard use and connectionless communication are exceptions.</td>
</tr>
<tr>
<td>007:014</td>
<td>EADDRNOTAVAIL</td>
<td>Cannot assign requested address. Normally results when an attempt is made to create a socket with an address not on the local machine.</td>
</tr>
<tr>
<td>007:015</td>
<td>ENETDOWN</td>
<td>Network is down.</td>
</tr>
<tr>
<td>007:016</td>
<td>ENETUNREACH</td>
<td>Network is unreachable. This is usually caused by network interface hardware that is operational, but not physically connected to the network. This error is also returned when the network has no way to reach the destination address.</td>
</tr>
<tr>
<td>007:017</td>
<td>ENETRESET</td>
<td>Network lost connection on reset. The host crashed and rebooted.</td>
</tr>
<tr>
<td>007:018</td>
<td>ECONNABORTED</td>
<td>Software caused connection abort. The local (host) machine caused a connection abort.</td>
</tr>
<tr>
<td>Number</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>007:019</td>
<td>ECONNRESET</td>
<td>Connection reset by peer. A peer forcibly closed the connection. This normally results from a loss of connection on the remote socket due to a time out or reboot.</td>
</tr>
<tr>
<td>007:020</td>
<td>ENOBUFS</td>
<td>No buffer space available. A socket operation could not be performed because the system lacked sufficient buffer space or queue was full.</td>
</tr>
<tr>
<td>007:021</td>
<td>EISCONN</td>
<td>Socket is already connected. The connection request was made for an already connected socket. Sending a <code>sendto()</code> call to an already connected destination could cause this error.</td>
</tr>
<tr>
<td>007:022</td>
<td>ENOTCONN</td>
<td>Socket is not connected. A request to send or received data was rejected because the socket was not connected or no destination was given for a datagram socket.</td>
</tr>
<tr>
<td>007:023</td>
<td>ESHUTDOWN</td>
<td>Cannot send after socket shutdown. Additional data transmissions are not allowed after the socket was shut down.</td>
</tr>
<tr>
<td>007:024</td>
<td>ETOOMANYREFS</td>
<td>Too many references.</td>
</tr>
<tr>
<td>007:025</td>
<td>ETIMEDOUT</td>
<td>Connection timed out. A <code>connect()</code> or <code>send()</code> request failed because the connected peer did not properly respond after a set period of time. The time out period depends on the protocol used.</td>
</tr>
<tr>
<td>007:026</td>
<td>ECONNREFUSED</td>
<td>Connection refused by target. No connection could be established because the target machine actively refused it. This usually results from trying to connect to an inactive service on the target host.</td>
</tr>
<tr>
<td>007:027</td>
<td>EBUFTOOSMALL</td>
<td>Buffer too small for F_MBuf operation. The specified buffer cannot be used to support F_MBUF(SysMbuf) calls.</td>
</tr>
<tr>
<td>007:028</td>
<td>ESMODEXISTS</td>
<td>Socket module already attached. An attach was requested of an already attached socket.</td>
</tr>
<tr>
<td>007:029</td>
<td>ENOTSOCK</td>
<td>Path is not a socket. A socket function was attempted on a path that is not a socket.</td>
</tr>
<tr>
<td>007:030</td>
<td>EHOSTUNREACH</td>
<td>No route to host.</td>
</tr>
<tr>
<td>007:031</td>
<td>EHOSTDOWN</td>
<td>Host is down.</td>
</tr>
<tr>
<td>008:001</td>
<td>EOS_LNKDOWN</td>
<td>Layer 1 link down (driver).</td>
</tr>
<tr>
<td>008:002</td>
<td>EOS_CONN</td>
<td>Connection error (driver).</td>
</tr>
<tr>
<td>008:003</td>
<td>EOS_RXTHREAD</td>
<td>Error in receive thread.</td>
</tr>
</tbody>
</table>
### Appendix B: OS-9 Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008:004</td>
<td>EOS_ME</td>
<td>Management entity error.</td>
</tr>
<tr>
<td>008:005</td>
<td>EOS_SAPI</td>
<td>Unrecognized SAPI</td>
</tr>
<tr>
<td>008:006</td>
<td>EOS_TEI</td>
<td>TEI not found.</td>
</tr>
<tr>
<td>008:007</td>
<td>EOS_MAX_TEI</td>
<td>Maximum number of TEI.</td>
</tr>
<tr>
<td>008:008</td>
<td>EOS_TSTATE</td>
<td>Illegal TEI state.</td>
</tr>
<tr>
<td>008:009</td>
<td>EOS_TEI_DENIED</td>
<td>TEI request denied by network.</td>
</tr>
<tr>
<td>008:010</td>
<td>EOS_PRIMITIVE</td>
<td>TEI request denied by network.</td>
</tr>
<tr>
<td>008:011</td>
<td>EOS_L2IN</td>
<td>Layer 2 error.</td>
</tr>
<tr>
<td>008:012</td>
<td>EOS_PEER_BUSY</td>
<td>Peer receiver busy condition</td>
</tr>
<tr>
<td>008:013</td>
<td>EOS_K</td>
<td>Maximum number of outstanding I frames.</td>
</tr>
<tr>
<td>008:014</td>
<td>EOS_MAXCREF</td>
<td>Maximum number of call references.</td>
</tr>
<tr>
<td>008:015</td>
<td>EOS_CREF</td>
<td>Call reference does not exist.</td>
</tr>
<tr>
<td>008:016</td>
<td>EOS_CALLPROG</td>
<td>Error on call progress state.</td>
</tr>
<tr>
<td>008:017</td>
<td>EOS_RXVR</td>
<td>Receiver previously assigned</td>
</tr>
<tr>
<td>008:018</td>
<td>EOS_REQDENIED</td>
<td>Request denied by far end.</td>
</tr>
<tr>
<td>008:019</td>
<td>EOS_RXSTART</td>
<td>Receive thread did not start.</td>
</tr>
<tr>
<td>008:020</td>
<td>EOS_NOSTACK</td>
<td>Last driver on path’s stack.</td>
</tr>
<tr>
<td>008:021</td>
<td>EOS_BTMSTK</td>
<td>Attempt to pop last driver.</td>
</tr>
<tr>
<td>008:022</td>
<td>EOS_NPBNULL</td>
<td>Notify param block is null.</td>
</tr>
<tr>
<td>008:023</td>
<td>EOS_PPS_NOTFND</td>
<td>Per path storage not found.</td>
</tr>
<tr>
<td>008:024</td>
<td>EOS_STKFULL</td>
<td>Path’s stack array is full.</td>
</tr>
<tr>
<td>008:025</td>
<td>EOS_MBNOTINST</td>
<td>Sysmbuf is not installed.</td>
</tr>
<tr>
<td>008:026</td>
<td>EOS_TMRNTFND</td>
<td>Timer not found.</td>
</tr>
<tr>
<td>008:027</td>
<td>EOS_GETIME</td>
<td>Get time error.</td>
</tr>
<tr>
<td>008:028</td>
<td>EOS_TIMERINT</td>
<td>Timer interrupt.</td>
</tr>
<tr>
<td>008:029</td>
<td>EOS_RXMB_NODEENTRY</td>
<td>No device entry in mbuf.</td>
</tr>
<tr>
<td>008:030</td>
<td>EOS_PGM_TBLBSY</td>
<td>PSI/SI table is in use.</td>
</tr>
<tr>
<td>008:031</td>
<td>EOS_PGM_LOVF</td>
<td>Too many tables being read.</td>
</tr>
<tr>
<td>008:032</td>
<td>EOS_PGM_TBLNFND</td>
<td>Table not found.</td>
</tr>
<tr>
<td>008:033</td>
<td>EOS_PGM_NFND</td>
<td>Program not found.</td>
</tr>
<tr>
<td>008:034</td>
<td>EOS_PGM_NOPLAY</td>
<td>No program is currently playing.</td>
</tr>
<tr>
<td>008:035</td>
<td>EOS_NODNDRVR</td>
<td>No down driver.</td>
</tr>
<tr>
<td>Number</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>008:257</td>
<td>EOS_MW1394_BUSRESET</td>
<td>Bus reset. An IEEE 1394 Bus is reset whenever the cable is plugged or unplugged or a device forces a bus reset (such as when driver software is initialized).</td>
</tr>
<tr>
<td>008:258</td>
<td>EOS_MW1394_NOIRM</td>
<td>No Isochronous Resource Manager found.</td>
</tr>
<tr>
<td>008:259</td>
<td>EOS_MW1394_NNF</td>
<td>Requested node not found on the bus.</td>
</tr>
<tr>
<td>008:260</td>
<td>EOS_MW1394_TIMEDOUT</td>
<td>Timed out waiting for response. The remote device is not present, does not have memory mapped at the specified location, or does not allow data transfer to that location. This is a general error indicating communication was not successful.</td>
</tr>
<tr>
<td>008:261</td>
<td>EOS_MW1394_FAILED</td>
<td>Internal failure.</td>
</tr>
<tr>
<td>008:262</td>
<td>EOS_MW1394_BADSIZE</td>
<td>Bad size. The length of the packet (or response) specified is either greater than the maximum allowable payload or the allocated Asynchronous transmit buffer size.</td>
</tr>
<tr>
<td>008:263</td>
<td>EOS_MW1394_ADDRINUSE</td>
<td>Address in use. The address could not be mapped because some portion of the requested range has already been mapped.</td>
</tr>
<tr>
<td>008:264</td>
<td>EOS_MW1394_ADDRNOTFND</td>
<td>Address not found. The address is not currently mapped, so it need not be unmapped.</td>
</tr>
<tr>
<td>008:265</td>
<td>EOS_MW1394_ADDRNOTAVAIL</td>
<td>Address not available. The desired address is not available for mapping.</td>
</tr>
<tr>
<td>008:266</td>
<td>EOS_MW1394_INVCHANNEL</td>
<td>Invalid channel. The channel value specified is out of range.</td>
</tr>
<tr>
<td>008:267</td>
<td>EOS_MW1394_CHNLINUSE</td>
<td>Channel in use. The specific channel requested is currently in use.</td>
</tr>
<tr>
<td>008:268</td>
<td>EOS_MW1394_NOCFGREG</td>
<td>No free Isochronous configuration registers.</td>
</tr>
<tr>
<td>008:269</td>
<td>EOS_MW1394_CHNLNOTFND</td>
<td>Channel not found. The specified channel was not found in the isochronous configuration registers.</td>
</tr>
<tr>
<td>008:270</td>
<td>EOS_MW1394_CHNL_STOPPED</td>
<td>Channel is stopped. Operations on this channel have been stopped by a call to ms1394IsochStop().</td>
</tr>
<tr>
<td>008:271</td>
<td>EOS_MW1394_NO_FREECHNL</td>
<td>Channel deallocation failed.</td>
</tr>
</tbody>
</table>
Table B-13. IEEE 1394 Error Codes (Continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008:272</td>
<td>EOS_MW1394_NOXMIT</td>
<td>Could not transmit. Data may not be transmitted because the FIFO is full or a hardware condition is prohibiting transmit.</td>
</tr>
<tr>
<td>008:273</td>
<td>EOS_MW1394_INVTOPOMAP</td>
<td>Topology Map invalid. The Topology Map on the Bus Manager is invalid.</td>
</tr>
<tr>
<td>008:274</td>
<td>EOS_MW1394_INVSPEDMAP</td>
<td>Invalid speedmap.</td>
</tr>
<tr>
<td>008:275</td>
<td>EOS_MW1394_BUSMGR_EXISTS</td>
<td>Bus Manager already present. There is already a Bus Manager present.</td>
</tr>
<tr>
<td>008:276</td>
<td>EOS_MW1394_BEBUSMGRFAIL</td>
<td>IRM refused request. The request to be a bus manager failed because the Isochronous Resource Manager did not honor request.</td>
</tr>
<tr>
<td>008:277</td>
<td>EOS_MW1394_BUSMGR_ALRDY</td>
<td>Already the Bus Manager. The requesting node is already the Bus Manager.</td>
</tr>
<tr>
<td>008:278</td>
<td>EOS_MW1394_ALLOCHNLFAIL</td>
<td>Channel allocation failed.</td>
</tr>
<tr>
<td>008:279</td>
<td>EOS_MW1394_FREECHNLFAIL</td>
<td>Channel deallocation failed.</td>
</tr>
<tr>
<td>008:280</td>
<td>EOS_MW1394_CHNLFREEALRDY</td>
<td>Channel already free. The specified channel is already free.</td>
</tr>
<tr>
<td>008:281</td>
<td>EOS_MW1394_ALLOCBDTHFAIL</td>
<td>Failed to allocate bandwidth.</td>
</tr>
<tr>
<td>008:282</td>
<td>EOS_MW1394_FREEBDTHFAIL</td>
<td>Failed to deallocate bandwidth.</td>
</tr>
<tr>
<td>008:283</td>
<td>EOS_MW1394_BANDWIDTH_NOTAVAIL</td>
<td>Bandwidth is not available. The amount of bandwidth requested is not available. The caller may reduce the desired maxbytesPerFrame and try again or retry after some delay.</td>
</tr>
<tr>
<td>008:284</td>
<td>EOS_MW1394_INV_BANDWIDTH_HNDL</td>
<td>Invalid handle. No bandwidth handle is found with the specified handleID.</td>
</tr>
<tr>
<td>008:285</td>
<td>EOS_MW1394_NOBUFFERS_ATTCHD</td>
<td>No buffers attached. No buffers are attached for the specified channel.</td>
</tr>
<tr>
<td>008:286</td>
<td>EOS_MW1394_RCODE_ERROR</td>
<td>Remote response code was not RCODE_COMPLETE. A response code other than RCODE_COMPLETE was received.</td>
</tr>
<tr>
<td>008:287</td>
<td>EOS_MW1394_NOBUSMGR</td>
<td>No Bus Manager found.</td>
</tr>
<tr>
<td>008:288</td>
<td>EOS_MW1394_INVACCESSTYPE</td>
<td>Invalid access type. No valid access type was specified.</td>
</tr>
</tbody>
</table>
### Table B-14. SVLAN Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>008:321</td>
<td>EOS_INVALID_BRIDGE</td>
<td>Bridge identifier is out of range.</td>
</tr>
<tr>
<td>008:322</td>
<td>EOS_INVALID_PORT</td>
<td>Port is out of range.</td>
</tr>
<tr>
<td>008:323</td>
<td>EOS_VTM_NOT_STARTED</td>
<td>The table manager has not been started.</td>
</tr>
<tr>
<td>008:324</td>
<td>EOS_INSUFFICIENT_MEMORY</td>
<td>The buffer is too small for data.</td>
</tr>
<tr>
<td>008:325</td>
<td>EOS_INVALID_MAC_ADDRESS</td>
<td>Invalid MAC address specified.</td>
</tr>
<tr>
<td>008:326</td>
<td>EOS_NO_LEARNING</td>
<td>Port does not support learning.</td>
</tr>
<tr>
<td>008:327</td>
<td>EOS_INVALID_PORTMAP</td>
<td>The specified portmap is invalid.</td>
</tr>
<tr>
<td>008:328</td>
<td>EOS_INVALID_VID</td>
<td>VLAN identifier value is out of range.</td>
</tr>
<tr>
<td>008:329</td>
<td>EOS_INVALID_PORT_NUMBER</td>
<td>Port number is out of range.</td>
</tr>
<tr>
<td>008:330</td>
<td>EOS_NO_TRANSFORMED_VID</td>
<td>No transformed VID has been specified.</td>
</tr>
<tr>
<td>008:331</td>
<td>EOS_INVALID_MAX_AGE</td>
<td>Maximum age value is out of range and/or does not conform to the set rules.</td>
</tr>
<tr>
<td>008:332</td>
<td>EOS_INVALID_FORWARD_DELAY</td>
<td>Forward delay is out of range.</td>
</tr>
<tr>
<td>008:333</td>
<td>EOS_INVALID_MAX_ADDRESS</td>
<td>Maximum address value is out of range.</td>
</tr>
<tr>
<td>008:334</td>
<td>EOS_INVALID_HELLO_TIME</td>
<td>Hello time value is out of range.</td>
</tr>
<tr>
<td>008:335</td>
<td>EOS_INVALID_TIMEOUT</td>
<td>Timeout value is out of range.</td>
</tr>
<tr>
<td>008:336</td>
<td>EOS_NO_BRIDGE</td>
<td>There is no such bridge.</td>
</tr>
<tr>
<td>008:337</td>
<td>EOS_NO_PORT</td>
<td>There is no such port.</td>
</tr>
<tr>
<td>008:338</td>
<td>EOS_PORT_EXISTS</td>
<td>The specified port is already part of the bridge.</td>
</tr>
<tr>
<td>008:339</td>
<td>EOS_ENF</td>
<td>BAT entry not found in table.</td>
</tr>
</tbody>
</table>

### Table B-15. POSIX Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>020:001</td>
<td>E2BIG</td>
<td>Argument list too long.</td>
</tr>
<tr>
<td>020:003</td>
<td>EBADMSG</td>
<td>Bad message.</td>
</tr>
<tr>
<td>020:004</td>
<td>ECANCELED</td>
<td>Operation cancelled.</td>
</tr>
<tr>
<td>020:005</td>
<td>EDEADLK</td>
<td>Resource deadlock avoided.</td>
</tr>
<tr>
<td>020:006</td>
<td>EFBIG</td>
<td>File too big.</td>
</tr>
<tr>
<td>020:007</td>
<td>EISDIR</td>
<td>Is a directory.</td>
</tr>
<tr>
<td>020:008</td>
<td>EMFILE</td>
<td>Too many process open files.</td>
</tr>
<tr>
<td>020:009</td>
<td>EMLINK</td>
<td>Too many links.</td>
</tr>
<tr>
<td>020:010</td>
<td>ENAMETOOLONG</td>
<td>Filename too long.</td>
</tr>
<tr>
<td>020:011</td>
<td>ENFILE</td>
<td>Too many system open files.</td>
</tr>
<tr>
<td>020:012</td>
<td>ENOEXEC</td>
<td>Exec format error.</td>
</tr>
</tbody>
</table>
### Table B-15. POSIX Error Codes (Continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>020:013</td>
<td>ENOLCK</td>
<td>No locks available.</td>
</tr>
<tr>
<td>020:014</td>
<td>ENOSPC</td>
<td>No space left on device.</td>
</tr>
<tr>
<td>020:015</td>
<td>ENOSYS</td>
<td>Function not implemented.</td>
</tr>
<tr>
<td>020:016</td>
<td>ENOTDIR</td>
<td>Not a directory.</td>
</tr>
<tr>
<td>020:017</td>
<td>ENOTEMPTY</td>
<td>Directory not empty.</td>
</tr>
<tr>
<td>020:018</td>
<td>ENOTSUP</td>
<td>Not supported.</td>
</tr>
<tr>
<td>020:019</td>
<td>ENOTTY</td>
<td>Bad I/O control operation.</td>
</tr>
<tr>
<td>020:020</td>
<td>EROFS</td>
<td>Read-only file system.</td>
</tr>
<tr>
<td>020:021</td>
<td>ESPIPE</td>
<td>Invalid seek.</td>
</tr>
<tr>
<td>020:022</td>
<td>EXDEV</td>
<td>Improper link.</td>
</tr>
</tbody>
</table>

### Table B-16. OS-9-Specific PowerPC Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100:002</td>
<td>EOS_PPC_MACHCHK</td>
<td>Machine check exception.</td>
</tr>
<tr>
<td>100:003</td>
<td>EOS_PPC_DATAACC</td>
<td>Data access exception.</td>
</tr>
<tr>
<td>100:004</td>
<td>EOS_PPC_INSTACC</td>
<td>Instruction access exception.</td>
</tr>
<tr>
<td>100:005</td>
<td>EOS_PPC_EXTINT</td>
<td>External interrupt.</td>
</tr>
<tr>
<td>100:006</td>
<td>EOS_PPC_ALIGN</td>
<td>Alignment exception.</td>
</tr>
<tr>
<td>100:007</td>
<td>EOS_PPC_PROGRAM</td>
<td>Program exception.</td>
</tr>
<tr>
<td>100:008</td>
<td>EOS_PPC_FPUUNAV</td>
<td>FPU unavailable exception.</td>
</tr>
<tr>
<td>100:009</td>
<td>EOS_PPC_DEC</td>
<td>Decrementer exception.</td>
</tr>
<tr>
<td>100:010</td>
<td>EOS_PPC_IOCONT</td>
<td>I/O controller exception.</td>
</tr>
<tr>
<td>100:012</td>
<td>EOS_PPC_SYSCALL</td>
<td>System call exception.</td>
</tr>
<tr>
<td>100:032</td>
<td>EOS_PPC_TRACE</td>
<td>Trace exception.</td>
</tr>
</tbody>
</table>

### Table B-17. OS-9-Specific MIPS Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>102:000</td>
<td>EOS_MIPS_EXTINT</td>
<td>External interrupt.</td>
</tr>
<tr>
<td>102:001</td>
<td>EOS_MIPS_MOD</td>
<td>TLB Modification exception.</td>
</tr>
<tr>
<td>102:002</td>
<td>EOS_MIPS_TLBL</td>
<td>TLB Miss exception (load or instruction fetch).</td>
</tr>
<tr>
<td>102:003</td>
<td>EOS_MIPS_TLBS</td>
<td>TLB Miss exception (store).</td>
</tr>
<tr>
<td>102:004</td>
<td>EOS_MIPS_ADEL</td>
<td>Address Error exception (load or instruction fetch).</td>
</tr>
<tr>
<td>102:005</td>
<td>EOS_MIPS_ADES</td>
<td>Address Error exception (store).</td>
</tr>
<tr>
<td>102:006</td>
<td>EOS_MIPS_IBE</td>
<td>Bus Error exception (instruction fetch).</td>
</tr>
<tr>
<td>102:007</td>
<td>EOS_MIPS_DBE</td>
<td>Bus Error exception (load or store).</td>
</tr>
</tbody>
</table>
### Table B-17. OS-9-Specific MIPS Error Codes (Continued)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>102:008</td>
<td>EOS_MIPS_SYS</td>
<td>SYSCALL exception.</td>
</tr>
<tr>
<td>102:009</td>
<td>EOS_MIPS_BP</td>
<td>Breakpoint exception.</td>
</tr>
<tr>
<td>102:010</td>
<td>EOS_MIPS_RI</td>
<td>Reserved Instruction exception.</td>
</tr>
<tr>
<td>102:011</td>
<td>EOS_MIPS_CPU</td>
<td>CoProcessor Unusable exception.</td>
</tr>
<tr>
<td>102:012</td>
<td>EOS_MIPS_OVF</td>
<td>Arithmetic Overflow exception.</td>
</tr>
<tr>
<td>102:013</td>
<td>EOS_MIPS_TR</td>
<td>Trap exception.</td>
</tr>
<tr>
<td>102:023</td>
<td>EOS_MIPS_WATCH</td>
<td>Watch exception.</td>
</tr>
<tr>
<td>102:032</td>
<td>EOS_MIPS_UTLB</td>
<td>User State TLB Miss exception.</td>
</tr>
</tbody>
</table>

### Table B-18. OS-9-Specific ARM Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>103:001</td>
<td>EOS_ARM_UNDEF</td>
<td>Undefined instruction exception.</td>
</tr>
<tr>
<td>103:003</td>
<td>EOS_ARM_PFABORT</td>
<td>Instruction pre-fetch abort exception.</td>
</tr>
<tr>
<td>103:004</td>
<td>EOS_ARM_DTABORT</td>
<td>Data abort exception.</td>
</tr>
<tr>
<td>103:008</td>
<td>EOS_ARM_ALIGN</td>
<td>Alignment exception.</td>
</tr>
</tbody>
</table>

### Table B-19. OS-9-Specific SuperH Error Codes

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>104:002</td>
<td>EOS_SH_TLBMISSLD</td>
<td>TLB miss on a load</td>
</tr>
<tr>
<td>104:003</td>
<td>EOS_SH_TLBMISSST</td>
<td>TLB miss on a store</td>
</tr>
<tr>
<td>104:004</td>
<td>EOS_SH_INITPG</td>
<td>Initial page write</td>
</tr>
<tr>
<td>104:005</td>
<td>EOS_SH_TLBPROTLD</td>
<td>TLB protection violation on a load</td>
</tr>
<tr>
<td>104:006</td>
<td>EOS_SH_TLBPROTST</td>
<td>TLB protection violation on a store</td>
</tr>
<tr>
<td>104:007</td>
<td>EOS_SH_ADDRLD</td>
<td>Address error on a load</td>
</tr>
<tr>
<td>104:008</td>
<td>EOS_SH_ADDRST</td>
<td>Address error on a store</td>
</tr>
<tr>
<td>104:009</td>
<td>EOS_SH_FPU</td>
<td>FPU exception</td>
</tr>
<tr>
<td>104:011</td>
<td>EOS_SH_TRAPA</td>
<td>TRAPA instruction</td>
</tr>
<tr>
<td>104:012</td>
<td>EOS_SH_RSRVINSTR</td>
<td>Reserved instruction</td>
</tr>
<tr>
<td>104:013</td>
<td>EOS_SH_ILLSLOT</td>
<td>Illegal slot instruction</td>
</tr>
<tr>
<td>104:015</td>
<td>EOS_SH_BRKPT</td>
<td>User break point</td>
</tr>
<tr>
<td>104:200</td>
<td>EOS_SH_FPUINEXACT</td>
<td>FPU inexact error</td>
</tr>
<tr>
<td>104:201</td>
<td>EOS_SH_FPUUNDERFLOW</td>
<td>FPU underflow error</td>
</tr>
<tr>
<td>104:202</td>
<td>EOS_SH_FPUOVERFLOW</td>
<td>FPU overflow error</td>
</tr>
<tr>
<td>104:203</td>
<td>EOS_SH_FPUDIVIDEO</td>
<td>FPU divide-by-zero error</td>
</tr>
<tr>
<td>104:204</td>
<td>EOS_SH_FPUINVALID</td>
<td>FPU invalid error</td>
</tr>
</tbody>
</table>
Index

Symbols
  _os_sema_init()
    semaphore initialization 59
  _os_sema_p()
    semaphore
      reservation 59
  _os_sema_term()
    semaphore terminate 59
  _os_sema_v()
    semaphore release 59
  oscall() 88
  _sliblink()
    to unlink a subroutine library 73
  _subcall
    to call a subroutine library 72

A
A_ATIME 101
A_ATIME, F_ALARM
  system-state alarms 53
A_CYCLE 102
A_CYCLE, F_ALARM
  system-state alarms 53
  user-state alarm 52
A_DELET 103
A_DELET, F_ALARM
  system-state alarms 53
  user-state alarm 52
A_RESET 104
A_RESET, F_ALARM
  system-state alarms 53
  user-state alarm 52
A_SET 105
A_SET, F_ALARM
  system-state alarms 53
  user-state alarm 52
access
  permission
    change for module directory 125
accounting system
  user 293
acquire
  conditional ownership of resource lock 94, 110
  acquire lock
    C example 80
  check return value 81
  activate next process waiting to acquire lock 302
  active process queue
    insert process into 109
    remove process 241
  add device to IRQ table 229
alarms
  A_ATIME 101
  A_CYCLE 102
  A_DELET 103
  A_RESET 104
  A_SET 105
  cyclic 52
  F_ALARM 99
    call reference 96
flags
  TH_DELPB 98
  TH_SPOWN 98
relative time
  defined 53
  remove pending request 52, 53, 103
  reset request 104
  system state
    defined and listed 53
  time of day
    defined 53
  user state
    defined and listed 52
alias
  copy system alias list 349
allocate
  process descriptor 106
  resource lock descriptor 133
  task 107
allow access to memory block 242
alternate module directory 27
anonymous pipes
  unnamed 64
ARM
  errors
    list of 414
attach
device 306

B
B_NVRAM 31
B_PARITY 31
B_ROM 31
B_USERRAM 31
bit map
flush cached information 368
blocks
skip 380
SS_SKIP 380
break serial connection 358
breakpoints
defined 142
hard 143
soft 143

C
C_ADDR 114
C_DISDATA 114
C_DISINST 114
C_ENDATA 114
C_ENINST 114
C_FLDATA 112, 114
C_FLINST 112, 114
C_GETCCTL 113, 114
C_INVDATA 114
C_STODATA 114
cache
cache control 111, 113
disable
data 114
instruction 114
RBF caching 359
enable
data 114
instruction 114
RBF caching 359
F_CCTL 111
flush
data 112, 114
instruction 112, 114
get status information 324
invalidate data 114
invalidate instruction 114
SS_CACHE 359
SS_CSTATS 324
calculate parity of file descriptor 336
call system debugger 281
change
file name 374
module directory permissions 125
process’ current module directory 123
system global variables 254
working directory 308
check
CRC 131
for deadlock situation 138
for use of I/O module 342
memory block’s accessibility 121
color codes 269
colored memory 30, 39
definition list 30
list 31, 40
command interpreter 13
compare names 126
Compiler errors 414
configuration module 40
contract data memory area 238
copy external memory 129
copy system alias list 349
CRC 16
check 131
F_CRC 131
F_SETCRC 253
F_VMODUL 296
generate 131, 253
verify module 296
CRCCON 131
create
data module 134
device alias 305
directory 345
event 162, 164
module directory 237
new descriptor 133
new process 211, 214
new resource lock descriptor 79
path 313
critical regions
locking 78
cyclic alarm 52

D
d_maxage 44, 268
d_minpty 44, 45, 268
d_tslice 44
data
modules
create 134
F_DATMOD 134
ready
send signal on 378
SS_SENDSIG 378
data modules
  creating 69
  DBG_M_CONTROL 143
  DBG_M_COUNT 142, 143
  DBG_M_HARD 142, 143
  DBG_M_INACTV 143
  DBG_M_SOFT 142, 143
  DBG_S_BRKPNT 144
  DBG_S_CHAIN 144
  DBG_S_CHILDLSIG 144
  DBG_S_CONTROL 144
  DBG_S_EXCEPT 142, 144
  DBG_S_EXIT 144
  DBG_S_FINISH 144
  DBG_S_PARENTSIG 144
  DBG_S_WATCH 144
de-allocate process descriptor 249
deallocate process descriptor 141
debug mode 143
debugged program
  debug modes 142
  execute 142
  exit 145
  F_DEXEC 142
  F_DEXIT 145
  F_DFORK 146
  F_DFORKM 148
  fork process 146, 148
debugger
  attach to a running process 136
  call system debugger 281
  F_SYSDBG 281
delete
  event 166, 167
  existing lock descriptor 139
  existing module directory 140
  file 315
  pending alarm request 103
device
  add to IRQ table 229
  attach 306
  close 311
  F_IRQ 229
  get
    size of SCSI device 329
    status 321
  I_DETACH 316
  I_GETSTAT 321
  I_OPEN 347
  I_READ 350
  I_READLN 352
  I_SETSTAT 355
  I_WRITE 388
  L_WRITELN 389
  open path to 347
  read
    data from 350, 352
    path options 326
  release 373
  remove 316
    from IRQ table 229
  return
    name 325
    type 327
  set
    path options 362
    status 355
  SS_DEVNAME 325
  SS_DEVOPT 326
  SS_DEVTYPE 327
  SS_DOPT 362
  SS_DSIZE 329
  SS_RELEASE 373
  write data to 388, 389
Device Alias
  create 305
device descriptor
  defined 13
device descriptors
  pipe 64
device driver
  defined 13
dexec.h 143, 144
directory
  access mode 345
  attribute bits 345
  attributes 345
  change 308
    permission 125
    create 345
  F_DELMDIR 140
  F_FMOD 210
  F_GETMDP 218
  F_GMODDR 221
  F_MKMDIR 237
  I_MAKDIR 345
  mode bits 345
  module directory 218
    change current 123
    create 237
    delete 140
    find entry 210
    get copy of 221
    remove 140
    set alternate 108
  disable
data cache 114
instruction cache 114
RBF caching 359
drivers
  resource locking 78
dump utility
  using to examine
data modules 69
duplicate path 317
dynamic-sized data structures 29

E
enable
  data cache 114
  instruction cache 114
  RBF caching 359
end of file
test for 331
EOS_SIGNAL
  no lock acquired 81
erase tape 365
error
  EOS_SIGNAL
    no lock acquired 81
error handling
    F_STRAP 277
errors
  list of
    ARM 414
    MIPS 414
    PowerPC 414
trapping 277
EV_ALLCLR 58, 152
EV_ALLSET 58, 154
EV_ANYCLR 58, 156
EV_ANYSET 58, 158
EV_CHANGE 58, 160
EV_CREAT 58, 63, 162, 164
EV_DELET 58, 63, 166, 167
ev_id
  EV_INFO field
    ID 56
EV_INFO 58, 168
ev_infostr structure 55
EV_LINK 58, 63, 170, 172
ev_link 56
EV_LNK 170, 172
ev_name
  EV_INFO field
    name 56
ev_namsz
  EV_INFO field
    name size 56
ev_owner 56
ev_perm 56
EV_PULSE 58, 174
ev_quen 56
ev_quep 56
EV_READ 176
EV_SET 58, 179
EV_SETAND 58, 181
EV_SETOR 58, 183
EV_SETXOR 58, 185
EV_SETXOR 58, 187
EV_SIGNL 58, 63, 189, 191
ev_sinc 56
ev_str structure 55
EV_TSTSET 58, 197
EV_UNLNK 58, 63, 199, 200
ev_value 56
EV_WAIT 58, 63, 177, 193, 195, 201, 203
EV_WAITR 58, 205
ev_winc 56
events
  create 162, 164
  delete 166, 167
  EV_ALLCLR 152
  EV_ALLSET 154
  EV_ANYCLR 156
  EV_ANYSET 158
  EV_CHANGE 160
  EV_CREATE 162, 164
  EV_DELET 166, 167
  EV_INFO 168
  EV_INFO fields 56
  EV_LNK 170, 172
  EV_PULSE 174
  EV_READ 176
  EV_SET 179
  EV_SETAND 181
  EV_SETOR 183
  EV_SETR 185
  EV_SETXOR 187
  EV_SIGNL 189, 191
  EV_TSTSET 197
  EV_UNLNK 199, 200
  EV_WAIT 177, 193, 195, 201, 203
  EV_WAITR 205
example
  synchronization 57, 62
F_EVENT 150
F_EVENT call
  using to synchronize resources 58, 63
link to existing 170, 172
read 176
remove event 166, 167
return information 168
set event variable 179, 181, 183, 187
signal event occurrence 174, 179, 181, 183, 185, 187, 189, 191
unlink 199, 200
using to coordinate non-sharable resources 57, 62
wait and signal operations 57
wait for event 152, 154, 156, 158, 160, 177, 193, 195, 197, 201, 203
relative 205
events.h 54, 61
examine system global variables 254
example code
alarms 396
events 398
semaphores 400
signals 394
tcall.c 411
thandler.c 410
trap handlers 408
trapc.a 408
ttest.c 412
exception jump table 29
execute
module 115
subroutine
     after interval 53
     at intervals 53
     at time 53
exit debugged program 145
expand data memory area 238
extension modules 40
external memory
copy 129
F_CPYMEM 129
F_ACQLK 94
      resource locking 78
F_ALARM
     A_ATIME 101
     A_CYCLE 102
     A_DELET 103
     A_RESET 104
     A_SET 105
     call reference 96
     system state 96
     user state 99
F_ALARM, A_ATIME
     system-state alarms 53
F_ALARM, A_CYCLE
     system-state alarms 53
     user-state alarm 52
F_ALARM, A_DELET
     system-state alarms 53
     user-state alarm 52
F_ALARM, A_RESET
     system-state alarms 53
     user-state alarm 52
F_ALARM, A_SET
     system-state alarms 53
     user-state alarm 52
F_ALLPRC 106
F_ALLTSK 107
F_ALTMDDIR 108
F_APROC 109
F_CAQLK 94, 110
      resource locking 78
F_CCTL 111
     system state 113
     user state 111
F_CHAIN 115
F_CHAINM 118
F_CHKMEM 121
F_CHMDDIR 123
F_CLRSIGS 124
F_CMDPERM 125
F_CMPNAM 126
F_CONFIG 128
      configure 128
F_CPYMEM 129
F_CRC 131
F_CRLK 79, 133
      resource locking 78
F_DATMOD 134
F_DATTACH 136
      attach Debugger to a Running Process 136
F_DDLK 138
F_DELMDIR 79, 139
      resource locking 78
F_DELTSK 140
F_DFORK 146
      system state 148
F_DFORKM 148
F_EVENT 150
     EV_ALLCRL 152
     EV_ALLSET 154
     EV_ANYCLR 156
     EV_ANYSET 158
     EV_CHANGE 160
     EV_CREAT 162, 164
     EV_DELET 166, 167
EV_INFO 168
EV_LINK 170, 172
EV_PULSE 174
EV_READ 176
EV_SET 179
EV_SETAND 181
EV_SETOR 187
EV_SETXOR 189
EV_SIGNAL 189
EV_TSTSET 197, 200
EV_UNLINK 199, 200
EV_WAIT 193, 201, 203
EV_WAITR 205
F_EXIT 207
F_FINDPD 209
F_FMODO 210
F_FORK 211
F_FORKM 214
F_GBLKMP 216
F_GETMDP 218
F_GETSYS 219
F_GMODDR 221
F_GPRDBT 222
F_GPRDSC 223
F_ICPT 224
F_ID 226
F_INITDATA 228
F_IRQ 229
F_LINK 231
F_LINKM 233
F_LOAD 235
F_MEM 238
F_MKMDIR 237
F_MODADDR 239
F_MOVE 240
F_NPROC 241
F_PERMIT 242
F_PROTECT 244
F_PRSNAM 246
F_RELLK 248
F_RETPD 249
F_RTE 250
F_SEMA 252
F_SEND 251
F_SETCRC 253
F_SETSYS 254
F_SIGLNGJ 256
F_SIGMASK 258
F_SIGRESET 260
F_SIGRS 261
F_SLEEP 263
F_SLINK 265
F_SLINKM 267
F_SPRIOR 268
F_SRQMEM 269
F_SRTMEM 271
F_SSPD 272
F_SSVC 273
F_STIME 275, 288
F_STRAP 277
F_USER 280
F_SYSDBG 281
F_SYSID 282
F_TIME 288
F_TLINK 289
F_TLINKM 291
F_UACCT 293
F_UNLINK 294
F_UNLOAD 295
F_VMODUL 296
F_WAIT 298
F_WAITLK 302
resource locking 78
FIFO buffer synchronization 83
FIFO buffer resource
synchronize reader and writer 83
file
access modes 313
attribute bits 313
attributes 313
change name 374
close 311
calculate parity of 336
descriptor
get
calculate parity of 336
descriptor
get
close 311
current position 338
status 321
I_CREATE 313
I_DELETE 315
I_GETSTAT 321
I_OPEN 347
I_READ 350
I_READLN 352
I_SETSTAT 355
I_WRITE 388
I_WRITELN 389
load module from 235
make hard link to 369
managers
defined 12
PCF 13
PIPEMAN 13, 64
RBF 13
SBF 13
SCF 13
mode bits 313
open path to 347
pointer
I_SEEK 354
reposition 354
read data from 350, 352
rename 374
set
attributes 357
size 340, 379
status 355
SS_ATTR 357
SS_HDLINK 369
SS_POS 338
SS_RENAME 374
SS_SIZE 340, 379
write data to 388, 389
file managers
resource locking 78
fill path buffer with data 367
find
module 239
module directory entry 210
process descriptor 209
fixmod 16
flush
cached bit map information 368
data cache 112, 114
instruction cache 112, 114
fork process under control of debugger 146, 148
fork_params structure 116, 119, 212
format
track 384
free memory pool 29
G
generate CRC 131, 253
get
cache status information 324
current file position 338
device status 321
file
descriptor block address 333
descriptor sector 334
status 321
free memory block map 216
I/O interface edition number 330
pointer to I/O process descriptor 310
process
descriptor copy 223
ID 226
size of SCSI device 329
system
date/time 288
global variable 219
user ID 226
Get Current File Position (RBF) 338
GetStat
using system path number 386
ghost bit
see sticky bit. 69
global
path number table 27
variables 268
change 254
examine 219, 254
set 254
group ID 42
H
hang-up signal 49
header files
dexec.h 143, 144
events.h 54, 61
module.h 17, 18, 131
svctbl.h 274
I
I/O 26, 27
attach device 306
close path 311
descriptor 41
device list
get 319
I_GETDL 319
module
check for use of 342
I_IODEL 342
process descriptor
  get pointer to 341
I_CIOPROC 310
I_GIOPROC 341
terminate for exiting process 343
I/O errors 414
I_ALIAS 305
I_ATTACH 306
I_CHDIR 308
I_CIOPROC 310
I_CLOSE 311
I_CONFIG 312
  configure 312
I_CREATE 313
I_DELETE 315
I_DETACH 316
I_DUP 317
I_GETDL 319
I_GETPD 320
I_GETSTAT 321
  SS_COPYPD 323
  SS_CSTATS 324
  SS_DEVDNAME 325
  SS_DEVOPT 326
  SS_DSIZET 327
  SS_EDT 330
  SS_EOF 331
  SS_FD 332
  SS_FdAddr 333
  SS_FDINFO 334
  SS_LUOPT 335
  SS_PARITY 336
  SS_PPATHOPT 337
  SS_POS 338
  SS_READY 339
  SS_SIZE 340
I_GETSTAT, SS_DEVOPT
  get status
    pipes 67
I_GETSTAT, SS_FD
  get status
    pipes 67
  set status
    pipes 68
I_GETSTAT, SS_FDINFO
  get status
    pipes 67
I_GETSTAT, SS_LUOPT
  get status
    pipes 67
I_GETSTAT, SS_PPATHOPT
  get status
    pipes 67
get status
pipes 67
I_GETSTAT, SS_SIZE
get status
pipes 67
I_GIOPROC 341
I_IODEL 342
I_IOEXIT 343
I_IOFORK 344
I_MAKDIR 345
I_OPEN 347
I_RDALST 349
I_READ 350
I_READLN
  difference between
    for pipes 66
I_READLN 352
I_READ
  difference between
    for pipes 66
ISEEK 354
I_SETSTAT 355
  SS_ATTR 357
  SS_BREAK 358
  SS_CACHE 359
  SS_DCOW 360
  SS_DCON 361
  SS_DOPT 362
  SS_DSRTS 363
  SS_ENRTS 364
  SS_ERASE 365
  SS_Fd 366
  SS_FILLBUFF 367
  SS_FLUSHMAP 368
  SS_HDLINK 369
  SS_LOCK 370
  SS_LUOPT 371
  SS_PPATHOPT 372
  SS_REN 373
  SS_REN 374
  SS_RESET 375
  SS_RETEN 376
  SS_RFM 377
  SS_SENDSIG 378
  SS_SIZE 379
  SS_SKIP 380
  SS_SKIPEND 381
  SS_TICKS 382
  SS_WFM 383
  SS_WTRACK 384
I_SETSTAT, SS_ATTR
  set status
    pipes 68
I_SETSTAT, SS_DEVOPT
  set status
  pipes 68
I_SETSTAT, SS_LUOPT
  set status
  pipes 68
I_SETSTAT, SS_PATHOPT
  set status
  pipes 68
I_SETSTAT, SS_RELEASE
  set status
  pipes 68
I_SETSTAT, SS_RENAME
  set status
  pipes 68
I_SETSTAT, SS_SENDSIG
  set status
  pipes 68
I_SETSTAT, SS_SIZE
  set status
  pipes 68
I_GETSTAT 386
I_TRANPN 387
I_WRITE 388
  _WRITE
    _WRITELN
      difference between
      for pipes 66
  _WRITELN 389
  _WRITE
    _WRITELN
      difference between
      for pipes 66
ident 13
Init module 40
init.h 40
initialization table
  example 274
  structure table 274
  svcbl 274
initialize
  directory 345
  process descriptor 106
  resource lock descriptor 133
  static storage 228
insert process
  in active process queue 109
install
  system state trap module 289
user
  trap handling module 291
intercept routine
  install 224
Internet errors 414
interprocess communication
A_RESET 104
  clear process signal queue 124
  create new event 162, 164
  delete existing event 166, 167
  link to existing event 170, 172
  read event value without waiting 176
  remove pending alarm request 103
reset
  alarm request 104
return event information 168
send signal 102, 104, 251
set
  alarm 96
  event variable 179, 181, 183, 187
  relative event variable 185
  signal intercept trap 224
  signal event occurrence 174, 179, 181, 183, 185, 187, 189, 191
unlink event 199, 200
wait for
  bits to clear 152
  events to occur 154, 156, 158, 160, 197, 201, 203
  relative event to occur 205
interrupt manipulation functions
  add device to IRQ table 229
  F_IRQ 229
  F_RTE 250
remove device to IRQ table 229
return from interrupt exception 250
invalidate
  data cache 114
  instruction cache 114
IOMAN 29
  defined 12
IRQ
  add device to table 229
  F_IRQ 229
remove device from table 229
K
kernel 13, 27, 30, 39, 40, 44, 54
  lock structure definition 78
keyboard signal
  abort 48, 251
  interrupt 48, 251
L
limit process access
  with resource locking 78
link
count
  for data modules 69
make hard link 369
subroutine libraries 265
to existing event 170, 172
to memory module 231, 233
to subroutine module 267
load module 235
lock
  acquire
    C example 80
  activate next 302
conditionally acquire ownership 94, 110
create 133
dealocate 79
delete descriptor 139
dynamic creation
    C example 79
F_CRLK 79, 133
F_DDLK 138
F_DELlk 139
F_RELLK 248
F_WAITlk 302
    FIFO buffer synchronization 83
identifier 79
initialize fields 80
out record 370
preallocate in resource 80
queue,
    using signals to 81
release ownership 248
request queueing 79
signal sensitive 81
synchronize FIFO reader and writer 83
lock structure
definition 78
logical
  unit
    read options 335
SS_LUOPT 335, 371
write options 371
m_ident 21
m_idref 21
m_init 21
m_name 18
m_owner 18
m_parity 21
m_share 20
m_size 18
m_slice 44
m_stack 20
m_symbol 20
m_sync 18
m_sysrev 18
m_term 21
m_tylan 14, 19
make
  hard link to existing file 369
  new directory 345
Mask/Unmask Signals During Critical Code 258
MEM_SHARED 31
MEM_SYS 31
memlist 31
memory
  assigning 29
  avoiding fragmentation 30
  block
    allow access to 242
    check accessibility 121
    F_GBLKMP 216
    F_PERMIT 242
    F_PROTECT 244
    map 216
    prevent access to 244
colored 30
definitions 39
  lists 40
copy external 129
F_CPYMEM 129
F_MEM 238
F_SRQMEM 269
fragmentation 30
  list 31, 39
management functions
  allocate task 107
  allow access to memory block 242
  check memory block’s accessibility 121
deallocate process descriptor 141
F_ALLTSK 107
F_CHKMEM 121
F_DELTSK 141
F_MEM 238
F_PERMIT 242
F_PROTECT 244
prevent access to memory block 244
resize data memory areas 238
map 28
modules
  ROMed 16
protection calls
deallocate process descriptor 141
  F_DELTSK 141
  F_PERMIT 242
  F_PROTECT 244
required 28
resize data area 238
system memory request 269
memory.h 31
mfree 30
mh_com structure 17
MIPS
  errors
    list of 414
Miscellaneous errors 414
module
  basic structure 15
  check
    CRC 296
    for use of 342
    header parity 296
directory 27
  alternate 27
  change
    permission 125
    create 237
    defined 27
    delete 140
  F_CMDPERM 125
  F_DELMDIR 140
  F_MOD 210
  F_GETMDP 218
  F_GMOD 221
  F_MKMDIR 237
  find entry 210
  get
    alternate pathlist 218
    copy of 221
    current pathlist 218
    remove 140
    set alternate 108
execute 115
  F_LINK 231
  F_LINKM 233
  F_LOAD 235
  F_MODADDR 239
  F_SETCRC 253
  F_UNLINK 294
  F_UNLOAD 295
  F_VMODUL 296
  find 239
  generate CRC 131, 253
  I_IODEL 342
  link to 231, 233
  load 115, 235
  manipulation functions
    change
      permissions of module directory 125
      process’ current module directory 123
    create
      data module 134
      new module directory 345
    delete
      existing module directory 140
    execute
      new primary module 118
    find
      module directory entry 210
      module given pointer 239
    install
      system state trap handling module 289
      user trap handling module 291
    set alternate working module directory 108
    position-independent 14
    re-entrant 14
    unlink 294, 295
    verify 296
module.h 17, 18, 131
move data 240
MT_SYSTEM 40

N
name
  valid characters 246
non-sharable
  coordinating resources
    with events 57, 62

O
open path 347
Operating system errors 414
OS9P2 25

P
p_spuing 242
p_state 109
parse path name 246
path
  access modes 313
attributes 313
close 311
create path to file 313
descriptor
  copy contents 323
  find 320
  I_GETTPD 320
  read option section 337
  SS_PATHOPT 337, 372
  write option section 372
duplicate 317
F_PRSNAM 246
fill buffer with data 367
I_DUP 317
I_OPEN 347
I_TRANPN 387
open 347
options
  set 362
  SS_DOPT 362
parse name 246
SS_FILLBUFF 367
terminate 311
translate user to system 387
PC file manager 13
PCF 13
permissions
  change module directory 125
pipe file manager 13, 64
pipelines
  process communications
    shell 64
PIPEMAN 13, 64
  status codes
    listed 67
pipes 13
  closing 67
  creating 65
device descriptors 64
directories 68
named 64
opening
  explanation 65
  returning data from 66
  unnamed 64
  writing data 67
PowerPC errors
  list of 414
prevent access to memory block 244
priority
  F_SPRIOR 268
  set 268
proc_id 310
process 44
  active 43
  clear process signal 124
  create 211, 214
  data area 42
descriptor 41
  allocate 106
de-allocate 249
deallocate 141
F_DELTSK 141
F_FINDPD 209
F_GPRDBT 222
F_GPRDSC 223
F_RETPD 249
find 209
generate 223
table 222
  pointer to 310, 341
I/O 310
I_GIOPROC 341
initialize 106
return resources 141
end 207
ensure protection hardware is ready 107
F_DFORK 146
F_DFORKM 148
F_EXIT 207
F_FORK 211
F_FORKM 214
F_SEND 251
F_SPRIOR 268
F_SPSPD 272
I_IOEXIT 343
I_IOFORK 344
ID 42, 226
inactive 43
insert in active process queue 109
insert process in active process queue 109
manipulation functions
  create new functions 211
  create new process 214
deallocate process descriptor 249
F_APROC 109
F_CHAIN 115
F_DFORK 146
F_DFORKM 148, 214
F_FINDPD 209
F_FORK 211
F_GPRDBT 222
F_GPRDSC 223
F_ID 226
F_NPROC 241
find process descriptor 209
fork process under control of debugger 146, 148
get
  copy of process descriptor block table 222
    process descriptor copy 223
    process ID/user ID 226
insert process in active process queue 109
load and execute module 115
set process priority 268
start next process 241
suspend process 272
wait for child process to terminate 298
memory area 42
F_DFORK 146
F_DFORKM 148
F_EXIT 207
F_FORK 211
F_FORKM 214
F_NPROC 241
F_SEND 251
F_SSPD 103, 272
I_IOEXIT 343
I_IOFORK 344
priority 268
scheduling 109
priority 268
ready for execution 43
scheduling 44
send signal to 251
set
  priority 268
  up I/O 344
start next process 241
state
  active 43
  event 43
  sleeping 43
  suspended 43
  waiting 43
states 43
stop 343
suspend 272
terminate 42, 207
I/O 343
wait for child to terminate 298
process queueing 84
protect critical section of code 81
Put Calling Process to Sleep 263
queue
  active process
  insert process 109
  clear process signal 124
queue a process 84
random block file manager 13
RAVE errors 414
RBF 13
read
data 350
  device path options 326
  event value 176
  file descriptor sector 332
  logical unit options 335
  path descriptor option section 337
record
  lock out 370
  SS_LOCK 370
  SS_TICKS 382
  wait for release 382
release
  device 373
  ownership of resource lock 248
remove
  device 316
    from IRQ table 229
    event information 166, 167
    pending alarm request 52, 53, 103
rename file 374
reposition logical file pointer 354
reset
  alarm request 52, 53, 104
resize
data memory area 238
resource lock
  acquire
    C example 80
  activate 302
  conditionally acquire ownership 94, 110
  create 133
  deadlock situation 138
  delete descriptor 79, 139
  dynamic creation
    C example 79
F_CRLK 79, 133
F_DDLK 138
F_DELKK 139
F_RELLK 248
F_WAITLK 302
  FIFO buffer synchronization 83
  for creating file managers/drivers 78
  initialize fields 80
  preallocate in resource 80
  release ownership 248
  synchronize FIFO buffer reader and writer 83
restore head to track zero 375
retool pass on tape drive 376
return
device
  name 325
  type 327
event information 168
  from interrupt exception 250
system
  identification 282
Return System Memory 271
RTS line
  disable 363
  enable 364
S
S_HANGUP 251
S_IAPPEND 313, 345
S_ICONTIG 313, 345
S_IEXEC 309, 313, 345
S_IFDIR 347
S_IEXEC 313, 345
S_IGREAD 313, 345
S_IGWRITE 313, 345
S_INT 251
S_IOEXEC 313, 345
S_IORD 313, 345
S_IOWRITE 313, 345
S_IREAD 309, 313, 345, 347
S_ISHARE 313, 345, 347
S_ISIZE 313, 345
S_ITRUNC 313, 345
S_IWRITE 309, 313, 345, 347
S_KILL 251
S_QUIT 251
S_WAKE 251
save utility
  using to save to disk
data module 69
SBF 13
SCF 13
SCSI device
  get size of 329
semaphore.h
structure 60
semaphores
  acquiring access 60
  application example .59
  binary event synchronization .59
defined .59
  example code 400
  header file structure 60
  initialization
    _os_sema_init() .59
  releasing
    _os_sema_v() .59
  releasing access 60
  reservation
    _os_sema_p() .59
  states 60
  structure definition 60
  terminate
    _os_sema_term() .59
send signal
  after specified time interval .52
  at specific time .52
  at specified time intervals 52, 102
  on data ready 378
  to another process 251
sequential block file manager 13
sequential character file manager 13
service request codes
  range of valid 273
  service request table initialization 273
set
  alternate working module directory 108
device
  path options 362
  status .355
  event variable 179, 181, 183, 187
  file
    attributes .357
    size 340, 379
    status .355
  process priority 268
  relative event variable 185
  system
    date/time 275
    global variables 254
    up I/O for process 344
    up signal intercept trap 224
    user ID number 280
SHARED 31
shell 13
signal
  clear queue 124
codes 251
communications
using F_SEND 49
event occurrence 174, 179, 181, 183, 185, 187, 189, 191
F_SEND 251
for events
defined 57
for lock queuing 81
handler
syntax 225
hang-up 49, 251
intercept routine installing
using F_ICPT 49
keyboard
abort 48, 251
interrupt 48, 251
remove 124
reset
F_SIGRESE 260
resize
F_SIGRS 261
S_HANGUP 251
S_INT 251
S_KILL 251
S_QUIT 251
S_WAKE 251
send
after specified time 52, 105
at specific time 52
at specified time 52, 101
at specified time intervals 102
on data ready 378
to another process 251
when DCD line goes false 360
when DCD line goes true 361
sensitive locks 81
set mask value
return on stack image
F_SGLN 256
SS_SENDSIG 378
system abort 251
unconditional system abort 48
wake up process 251
wake-up 48
skip
blocks 380
tape marks 377
to end of tape 381
software interrupts 48
SS_ATTR 357
SS_BREAK 358
SS_CACHE 359
SS_CopyPD 323
SS_CSTATS 324
SS_DCOFF 360
SS_DCON 361
SS_DEVNAME 325
SS_DEVOPT 326
SS_DEVTYPE 327
SS_DOPT 362
SS_DSIZE 329
SS_DSRTE 363
SS_EDT 330
SS_ENRTS 364
SS_EOF 331
get status
pipes 67
SS_ERASE 363
SS_FD 332, 366
SS_FdAd 333
SS_FINFO 334
SS_FILLBUFF 367
SS_FLUSHMAP 368
SS_HDLINK 369
SS_LOCK 370
SS_LUOPT 333, 371
SS_OPT 67
SS_PARITY 336
SS_PPATHOPT 337, 372
SS_POS 338
SS_READY 339
get status
pipes 67
SS_REDEA 368
SS_REUSE 373
SS_RNAME 374
SS_RESET 375
SS_RENRT 376
SS_RFMT 377
SS_SENDSIG 378
SS_SIZE 67, 68, 340, 379
SS_SKIP 380
SS_SKIPEND 381
SS_SSIG 68
SS_TICKS 382
SS_WFM 383
SS_WTRACK 384
standard I/O functions
attach new device to system 306
change working directory 308
check for use of I/O module 342
close path to file/device 311
create
device alias 305
path to new file 313
delete file \texttt{315}
duplicate path \texttt{317}
get
  pointer to I/O process descriptor \texttt{310, 341}
system I/O device list head pointer \texttt{319}
I_ALIAS \texttt{305}
I_ATTACH \texttt{306}
I_CHDIR \texttt{308}
I_CIOPROC \texttt{310}
I_CLOSE \texttt{311}
I_CREATE \texttt{313}
I_DELETE \texttt{315}
I_DETACH \texttt{316}
I_DUP \texttt{317}
I_GETDL \texttt{319}
I_GIOPROC \texttt{341}
I_IODEL \texttt{342}
I_IOEXIT \texttt{343}
I_IOFORK \texttt{344}
I_MAKDIR \texttt{345}
I_OPEN \texttt{347}
I_RDALST \texttt{349}
I_READ \texttt{350}
I_READLN \texttt{352}
I_WRITE \texttt{358}
I_WRITELN \texttt{359}
make new directory \texttt{345}
open path to file or device \texttt{347}
read
  data from file or device \texttt{350}
  text line with editing \texttt{352}
remove device from system \texttt{316}
set up I/O for new process \texttt{344}
terminate I/O for exiting process \texttt{343}
write
  data to file/device \texttt{388}
  line of text with editing \texttt{389}
start next process \texttt{241}
static storage
F_INITDATA \texttt{228}
initialize \texttt{228}
status functions
break serial connection \texttt{358}
calculate parity of file descriptor \texttt{336}
copy contents of path descriptor \texttt{323}
disable RBF caching \texttt{359}
enable RBF caching \texttt{359}
erase tape \texttt{365}
execute system global variable \texttt{219, 254}
F_GETSYS \texttt{219}
F_SETSYS \texttt{254}
fill path buffer with data \texttt{367}
flush cached bit map information \texttt{368}
get
  cache status information \texttt{324}
  current file position \texttt{338}
  file descriptor for open file \texttt{333}
  file/device status \texttt{321}
  I/O interface edition number \texttt{330}
  size of SCSI devices \texttt{329}
  specified file descriptor sector \texttt{334}
GetStat call using system path number \texttt{386}
I_GETSTAT \texttt{321}
I_SETSTAT \texttt{355}
I_SGETSTAT \texttt{386}
lock out record \texttt{370}
make hard link to existing file \texttt{369}
read
  device path options \texttt{326}
  file descriptor sector \texttt{332}
  logical unit options \texttt{335}
  path descriptor option section \texttt{337}
release device \texttt{373}
rename file \texttt{374}
restore head to track zero \texttt{375}
retension pass on tape drive \texttt{376}
return
  device
    name \texttt{325}
    type \texttt{327}
send
  signal on data ready \texttt{378}
  signal when DCD line goes
    false \texttt{360}
    true \texttt{361}
  wait for record release \texttt{382}
set
  device path options \texttt{362}
  file attributes \texttt{357}
  file size \texttt{340, 379}
  file/device status \texttt{355}
OS-9 system global variables \texttt{254}
skip
  blocks \texttt{380}
  tape marks \texttt{377}
  to end of tape \texttt{381}
SS_ATTR \texttt{357}
SS_BREAK \texttt{358}
SS_CACHE \texttt{359}
SS_COPYPD \texttt{323}
SS_CSTATS \texttt{324}
SS_DCOFF \texttt{360}
SS_DCON \texttt{361}
SS_DEVNAME \texttt{325}
SS_DEVOPT \texttt{326}
SS_DEVTYPE \texttt{327}
terminating 73
link to 267
subroutine libraries
link 265
suspend process 272
with F_WAITLK 83
tsvctbl structure 274
svctbl.h 274
Symbols
F_CRLK 79
F_DELLK 79
symbols
F_WAITLK
FIFO buffer synchronization 83
sysboot 13
sysglob.h 29, 41
Sysgo 392
system
debugger
call 281
F_SYSDBG 281
F_STIME 275
F_SYSID 282
F_TIME 288
functions of 24
get date/time 288
global
area 29
variables
change 254
examine 219, 254
F_GETSYS 219
F_SETSYS 254
set 254
memory request 269
path numbers 27
remove device table entry 316
return identification 282
set date/time 275
state 24
advantages 24
installing routines 25
system-state
alarms
defined and listed 53
systype.des 40
systype.h 30

T
tape
erase 365
retension 376
skip
tape marks 377
to end 381
SS_ERASE 365
SS_RETEN 376
SS_RFM 377
SS_SKIPEND 381
SS_WFM 383
write marks 383
task 44
allocate 107
switching 44
pre-emptive 44
system global variables 44
variables affecting 268
tcall
dispatching 75
relocatable macro assembler
mnemonic for trap calls 74
terminate
calling process 207
I/O for process 343
test for
data ready 339
end of file 331
TH_DELPB 98
TH_SPOWN 54, 98
time
functions
F_STIME 275
F_TIME 288
get system date and time 288
set system date and time 275
slice 44
track
SS_WTRACK 384
write 384
translate user path to system path 387
trap
handler 26
calling 74
creating library 75
described 289
execution entry points 74
explained 73
F_TLINK 289
F_TLINKM 291
installing 74
system state module 289
user number 74
program exceptions 277
vector 74

U
Ultra C related errors 414
unconditional system abort signal 48
unlink
event 199, 200
module 294, 295
user
ID 42, 226
set 280
state 24
user accounting system 293
user-state
alarms
defined and listed 52

V
valid pathlist characters 246
verify module 296

W
wait
for child to terminate 298
for event 152, 154, 156, 158, 160, 177, 193, 195, 197, 201, 203
for events
defined 57, 61
for record release 382
for relative event 205
wake-up signal 48
write
data 388
file descriptor sector 366
line of text 389
logical unit options 371
option section of path descriptor 372
tape marks 383
track 384