

UNIX-Domain-Sockets in LINUX

- **Allgemeines**

Die Protokollfamilie **PF_UNIX** (==**PF_LOCAL**) ist die einfachste Protokollfamilie innerhalb des Socket-APIs.

Sie stellt die **UNIX-Domain-Sockets** zur Verfügung.

Diese Sockets ermöglichen keine echte Netzwerkkommunikation sondern eine Kommunikation zwischen **beliebigen Prozessen** des gleichen Rechners (**lokale Interprozeß-Kommunikation**).

Die miteinander kommunizierenden Prozesse müssen weder miteinander verwandt sein noch die gleiche EUID besitzen.

UNIX-Domain-Sockets sind – im Unterschied zu Named Pipes - immer **verbindungsorientiert**.

Zwischen zwei Prozessen wird ein privater Kommunikationskanal eingerichtet, in den kein dritter Prozeß eindringen kann.

Ein Server-Prozeß, der gleichzeitig Verbindungen zu mehreren Clients unterhält, verwendet für jede Verbindung einen eigenen File-Deskriptor.

Gültige **Protokoll-Typen** sind : **SOCK_STREAM** und **SOCK_DGRAM**.

Dabei arbeitet **SOCK_DGRAM** ebenfalls verbindungsorientiert und gewährleistet Sequencing und eine Fehlerkontrolle (zuverlässige Verbindung).

- **UNIX-Domain-Adressen**

Die Adressen von UNIX-Domain-Sockets sind **Dateipfade** im Dateisystem (Adreß-Familie **AF_UNIX** bzw **AF_LOCAL**)
Hierfür existiert ein eigener Dateityp "Socket".

Durch das Binden einer Adresse an einen Socket (mittels **bind()**) wird ein **neuer Eintrag im Dateisystem** vom Typ "Socket" erzeugt.

Falls bereits ein Eintrag des angegebenen Pfadnamens existiert (egal welchen Dateityps), endet **bind()** mit dem Fehler **EADDRINUSE**.

Damit ein (Client-)Prozeß eine Verbindung zu einem existierenden (Server-)Socket aufnehmen kann (mittels System Call **connect()**) muß er Lese- und Schreibrechte zu der entsprechenden "Socket-Datei" besitzen.

Nach **Beendigung der Verwendung** eines Sockets muß dieser wieder **aus dem Dateisystem entfernt** werden (→ System Call **unlink()**).

- **Structure-Datentyp struct sockaddr_un**

Dieser in **<sys/un.h>** definierte Datentyp dient zur Darstellung von **UNIX-Domain-Adressen**.

```
struct sockaddr_un {
    unsigned short  sun_family;      /* Address family, hier AF_UNIX */
    char            sun_data[108];   /* Address Data : Dateipfad      */
};
```

Die für diesen Adreßtyp anzugebende **Länge** (als Parameter in System Calls) ergibt sich als Summe der Länge der Adreß-Familien-Komponente (**sun_family**) und der tatsächlichen Länge des Dateipfades.

```
Beispiel: struct sockaddr_sun my_addr;
          socklen_t addrlen;
          ...
          addrlen = sizeof(my_addr->sun_family) + strlen(my_addr->sun_path);
```

Achtung : Bei der Übergabe eines Pointers auf diese Adreß-Struktur in den System Calls ist ein Cast in einen Pointer auf den generischen Socket-Adreßtyp **struct sockaddr*** notwendig.

Beispiel zu UNIX-Domain-Sockets (1) : Serverprozeß

```
/* ----- */
/* Programm locserv */
/* ----- */
/* Beispiel fuer Unix Domain Sockets */
/* Einfacher Serverprozeß, */
/* - der auf einen Verbindungswunsch durch einen Clientprozeß wartet, */
/* - einen Verbindungswunsch akzeptiert */
/* - und die ihm vom Clientprozeß geschickten Daten an die Standardausgabe ausgibt */
/* Wenn der Clientprozeß die Kommunikation durch Senden von EOF (=CTRL-D) beendet, */
/* wartet er auf den nächsten Verbindungswunsch */
/* ----- */

#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <string.h>

#define SOCKET_PATH "/home/thomas/mysocket"
#define FD_STDOUT 1

void errorExit(char* msg);
void copyData(int iDest, int iSrc);

int main(int argc, char *argv[])
{
    struct sockaddr_un strAddr;
    socklen_t lenAddr;
    int fdSock;
    int fdConn;

    if ((fdSock=socket(PF_UNIX, SOCK_STREAM, 0)) < 0)
        errorExit("socket");

    unlink (SOCKET_PATH); /* Sicherstellung, daß SOCKET_PATH nicht existiert */
    strAddr.sun_family=AF_UNIX; /* Unix Domain */
    strcpy(strAddr.sun_path, SOCKET_PATH);
    lenAddr=sizeof(strAddr.sun_family)+strlen(strAddr.sun_path);

    if (bind(fdSock, (struct sockaddr*)&strAddr, lenAddr) != 0)
        errorExit("bind");

    if (listen(fdSock, 5) != 0)
        errorExit("listen");

    while ((fdConn=accept(fdSock, (struct sockaddr*)&strAddr, &lenAddr)) >= 0)
    {
        printf("\nConnection !!! receiving data ...\n");
        copyData(FD_STDOUT, fdConn);
        printf("\n... finished\n");
        close (fdConn);
    }

    close(fdSock);

    return 0;
}
```

Beispiel zu UNIX-Domain-Sockets (2) : Clientprozeß

```
/* ----- */
/* Programm loccli */
/* ----- */
/* Beispiel fuer Unix Domain Sockets */
/* Einfacher Clientprozeß, */
/* - der versucht, eine Socket-Verbindung zu einem Serverprozeß herzustellen */
/* - und bei Erfolg fortlaufend Zeichen von der Standardeingabe einliest */
/* - und diese über den Socket an den Serverprozeß sendet. */
/* Der Prozeß endet, wenn EOF (==CTRL-D) eingegeben wird */
/* ----- */

#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <string.h>

#define SOCKET_PATH "/home/thomas/mysocket"
#define FD_STDIN 0

void errorExit(char* msg);
void copyData(int iDest, int iSrc);

int main(int argc, char *argv[])
{
    struct sockaddr_un strAddr;
    socklen_t lenAddr;
    int fdSock;

    if ((fdSock=socket(PF_UNIX, SOCK_STREAM, 0)) < 0)
        errorExit("socket");

    strAddr.sun_family=AF_UNIX; /* Unix domain */
    strcpy(strAddr.sun_path, SOCKET_PATH);
    lenAddr=sizeof(strAddr.sun_family)+strlen(strAddr.sun_path);

    if (connect(fdSock, (struct sockaddr*)&strAddr, lenAddr) !=0 )
        errorExit("connect");

    printf("\nConnected to Server ... sending data ...\n");
    copyData(fdSock, FD_STDIN);
    printf("\nready !\n");
    close(fdSock);

    return 0;
}
```

Beispiel zu UNIX-Domain-Sockets (3) : Hilfsfunktionen für Server und Client

```
/* ----- */
/* C-Quell-Modul com_util.c */
/* ----- */
/* Hilfsfunktionen, die in den Programmen für locserv und loccli
/* eingesetzt werden */
/* ----- */

#include <unistd.h>
#include <stdio.h>

/* ----- */
/* Ausgabe einer Meldung msg, gefolgt von der Fehlermeldung, die dem in errno */
/* gespeicherten Fehlercode entspricht */
/* anschließende Beendigung des Programms */
/* ----- */

void errorExit(char* msg)
{
    perror(msg);
    exit(1);
}

/* ----- */
/* Kopieren von Bytefolgen von einem Quell-File-Deskriptor zu einem Ziel-File- */
/* Deskriptor */
/* Das Kopieren endet, wenn keine Bytes mehr vom Ziel-Deskriptor gelesen werden */
/* ----- */

#define BUFFLEN 1024

void copyData(int iDest, int iSrc)
{
    char acBuff[BUFFLEN];
    int iCount;

    while ((iCount=read(iSrc, acBuff, sizeof(acBuff)))>0)
    {
        if (write(iDest, acBuff, iCount)!=iCount)
            errorExit("write");
    }

    if (iCount<0)
        errorExit("read");
    return;
}

/* ----- */
```