AG Betriebssysteme FB3
Kirsten Berkenkötter

Allerlei Nützliches

1 Sockets

1.1 Allgemein

- Zum Erstellen von UDP-Sockets werden die Funtionen socket() und bind() verwendet.
- Zum Erstellen von TCP/IP-Scokets müssen zusätzlich listen() und accept() verwendet werden (s. man-Pages).
- Bei UDP können die Funtionen read() und write() nicht zum Senden und Empfangen verwendet werden (im Gegensatz zu TCP/IP), da keine permanente Verbindung besteht.

1.2 Header

Benötigt werden:

- <sys/types.h>
- <sys/socket.h>
- <unistd.h>

1.3 Socket anlegen

int socket(int domain, int type, int protocol);

- domain: Protokollfamilie, die ich verwende
  - PF_INET: IP Protokoll
  - PF_INET6: IP version 6 Protokoll
  - PF_IPX: Novell Internet Protokoll
  - PF_BLUETOOTH: Bluetooth Protokoll
  - ...
- type: Art des Socket
  - SOCK_STREAM: verbindungsorientierter Bytestream
  - SOCK_DGRAM: verbindslos, Datagramme fester Grösse
  - SOCK_SEQPACKET, verbindungsorientiert, Datagramme fester Grösse
  - ...
- protocol: ergibt sich in der Regel aus den ersten beiden Parameters, wenn mehrere Protokoll möglich sind, kann man es hier angeben, in der Regel 0

- Rückgabewert: Socket-Filedescriptor oder negativ bei Fehler

Typischer Aufruf:

```
int sfd;

if(0 > (sfd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)))
{
  perror(``Problem with socket()'');
  exit(1);
}
```

1.4 Kommandos auf Sockets

int setsockopt(int s, int level, int optname, const void *optval, socklen_t optlen);

- s: Socket-Filedescriptor
- level: üblich SOL_SOCKET, Manipulation auf Socket-Ebene, auf Protokollebene die Nummer des Protokolls
- optname: Name der Option, z.B. SO_REUSEADDR
- optval: Value, z.B. int optval = 1, Aufruf mit &optval
- optlen: Länge von optval, z.B. sizeof(optval)
- kann in <bits/socket.h> und <asm/socket.h> eingesehen werden
- Rückgabewert: -1 bei Fehler, 0 bei Erfolg

Typischer Aufruf:

```
int optval = 1;

if(0 > setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval)))
{
  perror(``Problem with setsockopt()'');
  exit(1);
}
```

1.5 Adresse an Socket binden

int bind(ind sockfd, struct sockaddr *my_addr, socklen_t addrlen);

- sockfd: Socket-Filedescriptor
- my_addr: Adresse, an die der Port gebunden wird, folgende Angaben mssen gemacht werden:
  - sin_family: Protokolltyp, z.B AF_INET, AF_INET6,usw.
  - sin_addr.s_addr: Was empfange ich, z.B. IN_ADDR_ANY, IN_ADDR_BROADCAST, IN_ADDR_NONE, INADDR_LOOPBACK, mit htonl()
  - sin_port: der zugehörige Port, will ja erreichbar sein, mit htons()

- addrlen: Länge der Adresse
- Rückgabewert: -1 bei Fehler, 0 bei Erfolg

Typischer Aufruf:

```
struct sockaddr_in saddr;
short port = 6666;

memset((void *)&saddr, 0, sizeof(saddr));
saddr.sin_familiy = AF_INET;
saddr.sin_addr.s_addr = htonl(INADDR_ANY);
saddr.sin_port = htons(port);

if(0 > bind(sfd, (struct sockaddr *),&saddr, sizeof(saddr)))
{
  perror(``Problem with bind()'');
  exit(1);
}
```

1.6 Netzwerkadressen

uint32_t htonl(uint32_t hostlong);

Host to network long, bekommt Adresse in "little endian" und konvertiert nach "big endian"

Analog dazu: htons(), ntohl(), ntohs()

Typischer Aufruf: s. bind()

struct hostent *gethostbyname(const char *name);

Ermittelt die IP-Adresse zum angegebenen Hostname, NULL, wenn nicht bekannt.

Typischer Aufruf: s. sendto()

1.7 Senden und Empfangen

ssize_t sendto(int s, const void *msg, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);

- s: Socket File-Descriptor
- msg: Zeiger auf die zu sendende Nachricht
- len: Länge der Nachricht
- flags: Flags, z.B. MSG_DONTWAIT (nichtblockieren)
- to: Adresse des Empfängers
  - sin_family: Protokollfamilie, z.B. AF_INET
  - sin_port: der zugehörige Port, mit htons()
  - sin_addr.s_addr: der Host
- tolen: Länge der Adresse
- Rückgabewert: -1 bei Fehlern, sonst die Anzahl der gesendeten Zeichen

Typischer Aufruf

```
char *dummy = ``Hallo Welt'';
struct sockaddr raddr;
int port = 5555;
struct hostent *hostinfo = gethostbyname(auenland);

memset((void *)&raddr, 0, sizeof(raddr));
raddr.sin_family = AF_INET;
raddr.sin_port = htons(port);
memcpy(&raddr.sin_addr.s_addr, hostinfo->h_addr, hostinfo->h_length);

if(0 > sendto(sfd, dummy, sizeof(dummy), 0, (struct sockaddr *) & raddr,
   sizeof(raddr)))
{
  perror(``Problem with sendto()'');
  exit(1);
}
```

ssize_t recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);

- s: Socket File-Descriptor
- buf: Puffer fr die empfangene Nachricht
- len: Länge des Puffers
- flags: z.B. MSG_WAITALL
- from: Addresse des Empfangssockets, wie bei sendto()
- fromlen: Länge des Empfangssockets
- Rückgabewert: -1 bei Fehlern, sonst die Anzahl der empfangenen Zeichen

Typischer Aufruf:

```
char recvbuffer[4000];
int messagelenght = sizeof(raddr);

if(0 > recvfrom(sfd, recvbuffer, sizeof(recvbuffer), 0,
                (struct sockaddr *)&raddr, &length))
{
  perror(``Problem with recvfrom()'');
  exit(1);
}
```

1.8 Socket schließen

int close(int fd);

- fd: Filedescriptor
- Rückgabewert: -1 bei

Typischer Aufruf

```
if(0 > close(sfd))
{
  perror(``Problem with close()'');
  exit(1);
}
```

2 Kindprozesse

2.1 Header

- <unidst.h>
- <sys/types.h>

pid_t fork();

- Rückgabewert: 0 bei Kind, Kind-PID bei Vater, im Fehlerfall -1

2.2 Kindprozesse erzeugen

Typischer Aufruf:

```
int childpid;

if(0 > (childpid = fork()))
{
  perror(``Problem with fork()'');
  exit(1);
}

if(childpid)
{
  //alles, was der Vater macht
}
else
{
  //alles, was das Kind macht
}
```

3 Misc

3.1 Header

- <string.h> für memset()
- <sys/types.h> für getpid() und waitpid()
- <unistd.h> für getpid()
- <wait.h> für waitpid()

3.2 Speicherbereich initialisieren

void *memset(void *s, int c, size_t n);

- s: Zeiger auf Speicherbereich
- c: Zeichen, mit dem gesetzt werden soll
- n: Anzahl der zu setzenden Bytes
- Rückgabewert: Zeiger auf s

Typischer Aufruf: s. bind()

3.3 PID ermitteln

pid_t getpid();

- Rckgabewert: eigene PID

Typischer Aufruf:

```
pid_t pid;

if(0 > (pid = getpid()) )
{
  perror(``Problem with getpid()'');
  exit(1);
}
```

3.4 Auf Terminierung von Kindern warten

pid_t waitpid(pid_t pid, int *status, int options);

- pid: pid auf die gewartet werden soll, -1 meint alle Kindprozesse
- status: Statusinformationen, können hinterher ausgelesen werden
- options: WNOHANG (nonblocking), WUNTRACED (stopped, not traced)
- Rückgabewert: > 0 ist PID des beendeten Kindprozesses, 0 bei Verwendung von WNOHANG, -1 bei Fehler

Typischer Aufruf:

```
if(0 > waitpid(childpid, NULL, 0))
{
  perror(``Problem with waitpid()'');
  exit(1);
}
```