Linux Programozás

Jelzések

Jelzések

- Pozitív egész szám más-más jelentéssel.
- A küldő folytatja a futását, nem vár végeredményt.
- A fogadó futása megszakad és végrehajtódik a lekezelő függvény.
- Megszakításokhoz hasonlít
- Kommunikáció
  - Kernel és folyamat között
    - hardver esemény
    - felhasználói esemény
    - szoftver esemény
  - folyamatok között
  - szálak között
- Többszálú programoknál óvatosan használjuk.

Jelzés küldése

```
#include <signal.h>
int kill(pid_t pid, int sig);
int killpg(int pgrp, int sig);
```

- pid:
  - >0: folyamat azonosító
  - -1: minden folyamat
  - <0: csoport azonosító
- pgrp: Csoport azonosító
- sig: Jelzés (0: csak tesztel) (man 7 signal)

Valósidejű jelzés küldése

```
#include <signal.h>
int sigqueue(pid_t pid, int sig, const union sigval
value);
union sigval {
  int   sival_int;
  void *sival_ptr;
};
```

- Megőrzi a jelzések sorrendjét és ha többször küldjük, akkor ugyanannyiszor kézbesíti.
- pid: A folyamat azonosítója
- sig: Jelzés
- value: Érték

Küldés szálak esetén

```
#include <signal.h>

int pthread_kill(pthread_t thread,
int sig);

int pthread_sigqueue(pthread_t
*thread, int sig, const union
sigval value);
```

Jelzés saját folyamatnak/szálnak

```
#include <signal.h>
int raise(int sig);
```

- A hívó folyamat illetve szál kapja meg a jelzést.
- sig: Jelzés

Jelzések halmaza

- Számos függvény jelzés halmazokkal dolgozik.
- Típusa: `sigset_t`

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int
signum);
```

Jelzések tiltása/engedélyezése

```
#include <signal.h>

int sigprocmask(int how, const sigset_t
*set, sigset_t *oldset);
```

- how:
  - SIG_BLOCK: blokkolja a megadottakat
  - SIG_UNBLOCK: engedélyezi a megadottakat
  - SIG_SETMASK: csak a megadottakat blokkolja
- set: új beállítás
- oldset: régi beállítás

Függőben lévő jelzések lekérdezése

```
#include <signal.h>
int sigpending(sigset_t *set);
```

- set: a függőben lévő jelzések (pending signals)

Jelzések kezelése

- Lehetőségek
  - A program figyelmen kívül hagyja.
  - A program regisztrál egy lekezelő függvényt.
  - A Kernel alapértelmezett lekezelő rutinja fut le.
- Lekezelő függvény regisztrációja:

```
#include <signal.h>

int sigaction(int signum, const struct
sigaction *act, struct sigaction *oldact);
```

- signum: Jelzés
- act: Új beállítás
- oldact: Régi beállítás

sigaction struktúra

```
struct sigaction {
  void     (*sa_handler)(int);
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t   sa_mask;
  int        sa_flags;
  void     (*sa_restorer)(void);
};
```

- sa_handler: Normál lekezelő függvény (SIG_DFL: default, SIG_IGN: tiltás)
- sa_sigaction: Kibővített lekezelő függvény (SA_SIGINFO flag esetén)
- sa_mask: A jelzéskezelő futása alatt tiltott jelzések.
- sa_flags: Beállítások

sigaction struktúra jelzőbitjei

Parancs	Leírás
SA_NOCLDSTOP	Normál esetben, ha egy processz gyerek processze megszűnik, vagy megáll, akkor egy SIGCHLD jelzést küld. Ha ezt az opciót beállítjuk a SIGCHLD jelzésre, akkor csak a megszűnés esetén küldi.
SA_RESTART	Automatikusan újraindítja a megszakított rendszerhívásokat.
SA_NODEFER	Amikor a jelzés kezelőfüggvénye meghívódik, akkor a jelzés nem tiltódik automatikusan. (Használata nem javasolt, csak speciális esetekben.)
SA_ONSTACK	Megadhatjuk a jelzéskezelő veremterületét, amelyet a *signalstack* függvénnyel hozunk létre.
SA_RESETHAND	Akkor használjuk, ha azt akarjuk, hogy a jelzéskezelő csak egyszer hívódjon meg. Miután a kezelőfüggvény meghívódott, a beállítás alapértelmezettre állítódik. Ha ezt nem adjuk meg, akkor a jelzéskezelő addig érvényes, amíg egy további *sigaction* hívással meg nem változtatjuk.
SA_SIGINFO	A jelzéskezelő paramétereket vár.
SA_NOCLDWAIT	A *SIGCHLD* jelzésre használjuk: ilyenkor a gyermekprocesszek nem változnak zombivá.

Szinkron jelzéskezelés

- Egy jelzésre várakozunk

```
#include <unistd.h>
int pause(void);
```

- Ugyanaz maszkolással

```
#include <signal.h>
int sigsuspend(const sigset_t
*mask);
```

Szinkron jelzéskezelés

```
#include <signal.h>

int sigwaitinfo(const sigset_t
*set, siginfo_t *info);

int sigtimedwait(const sigset_t
*set, siginfo_t *info, const struct
timespec *timeout);
```

- set: A jelzések, amelyekre várakozunk.

Állománykezelés és jelzéskezelés kapcsolata

```
#include <sys/signalfd.h>

int signalfd(int fd, const sigset_t *mask,
int flags);
```

- fd:
  - -1: új leíró létrehozása
  - >=0: létező jelzés-állományleíró módosítása
- mask: Fogadott jelzések
- flags:
  - SFD_NONBLOCK: O_NONBLOCK
  - SFD_CLOEXEC: O_CLOEXEC

Hálózat kezelés

socket

- A hálózati kommunikáció reprezentációja az állományabsztrakciós interfészen keresztül.

```
#include <sys/socket.h>

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

- domain: Protokoll család
- type: A kommunikáció típusa
- protocol: A protokoll kiválasztása (tipikusan: 0)
- Visszatérési érték: Socketleíró

Protokoll családok

Protokoll	Jelentés
PF_UNIX, PF_LOCAL	Unix domain (gépen belüli kommunikáció)
PF_INET	IPv4 protokoll
PF_INET6	IPv6 protokoll
PF_IPX	Novell IPX
PF_NETLINK	A kernel felhasználói interfésze
PF_X25	X.25 protokoll
PF_AX25	AX.25 protokoll, a rádióamatőrök használják
PF_ATMPVC	Nyers ATM-csomagok
PF_APPLETALK	AppleTalk
PF_PACKET	Alacsony szintű csomaginterfész

Kommunikáció típusa

Típus	Jelentés
SOCK_STREAM	Sorrendtartó, megbízható, kétirányú, kapcsolatalapú bájtfolyam-kommunikációt valósít meg.
SOCK_DGRAM	Datagram alapú (kapcsolatmentes, nem megbízható) kommunikáció.
SOCK_SEQPACKET	Sorrendtartó, megbízható, kétirányú, kapcsolatalapú kommunikációs vonal, fix méretű datagramok számára.
SOCK_RAW	Nyers hálózati protokoll hozzáférést tesz lehetővé.
SOCK_RDM	Megbízható datagram alapú kommunikációs réteg. (Nem sorrendtartó.)
SOCK_PACKET	Elavult opció.

Összeköttetés alapú kommunikáció

- Szerver oldal:
  - socket()
  - cím összeállítás
  - bind()
  - listen()
  - accept()
- Kommunikáció:
  - read()
  - write()
- Lezárás:
  - close()

- Kliens oldal:
  - socket()
  - cím összeállítás
  - connect()
- Kommunikáció:
  - write()
  - read()
- Lezárás:
  - close()

Címhez kötés

```
#include <sys/socket.h>

int bind(int sockfd, const struct
sockaddr *addr, socklen_t addrlen);
```

- sockfd: Socketleíró
- addr: Cím
- addrlen: Cím méret

Szerver socket

```
#include <sys/socket.h>

int listen(int sockfd, int
backlog);
```

- sockfd: Socketleíró
- backlog: Várakozó kapcsolatok listájának hossza

Várakozás a kliensekre

```
#include <sys/socket.h>

int accept(int sockfd, struct
sockaddr *addr, socklen_t
*addrlen);
```

- sockfd: Socketleíró
- addr: A kliens címe
- addrlen: A kliens címének mérete

Kapcsolódás a szerverhez

```
#include <sys/socket.h>

int connect(int sockfd, const
struct sockaddr *addr, socklen_t
addrlen);
```

- sockfd: Socketleíró
- addr: Szerver címe
- addrlen: Szerver címének hossza

Kommunikáció

- Az állománykezelésnél megismert read és write függvények.
- Mint a csővezetéknél csak kétirányú. Mindkét oldalon írhatunk és olvashatunk is.

Kapcsolat vége

- Az állománykezelésnél megismert close függvény.
  - Ha a kapcsolat egyik oldalát lezárjuk a másik is érzékeli, mint EOF.
  - Valójában állományleírót zár le. Ha dup()-al lemásoltuk, akkor nem zárja a kapcsolatot.
- Kapcsolat részleges és teljes lezárása:

```
#include <sys/socket.h>
int shutdown(int sockfd, int how);
```

  - sockfd: Socketleíró
  - how: SHUT_RD, SHUT_WR, SHUT_RDWR

Összeköttetés nélküli kommunikáció

- 1. fél
  - socket()
  - bind()
  - sendto()
  - recvfrom()
  - close()
- 2. fél
  - socket()
  - bind()
  - recvfrom()
  - sendto()
  - close()

Kommunikáció

- Küldés:

```
ssize_t sendto(int sockfd, const void *buf,
size_t len, int flags, const struct
sockaddr *dest_addr, socklen_t addrlen);
```

- Fogadás:

```
ssize_t recvfrom(int sockfd, void *buf,
size_t len, int flags, struct sockaddr
*src_addr, socklen_t *addrlen);
```

A connect() használata

- Összeköttetés nélküli kapcsolat esetén is használhatjuk.
- Nem hoz létre összeköttetést, de nem kell a csomagokat egyesével címeznünk.
- Használhatjuk a read(), write(), recv(), send() függvényeket.

Unix domain socket

- A legegyszerűbb protokoll család
- Valójában nem hálózati protokoll, csak egy gépen belüli IPC.
- A címei állománynevek (socket állomány)
- Stream és datagram is támogatott, de a stream a tipikus.

Unix domain címek

```
#include <sys/un.h>
struct sockaddr_un
{
  unsigned short sun_family;    /* AF_UNIX */
  char sun_path[UNIX_PATH_MAX]; /* eleresi ut
*/
};
```

Névtelen Unix domain socket

```
#include <sys/socket.h>

int socketpair(int domain, int
type, int protocol, int sv[2]);
```

- domain, type, protocol: lásd socket()
- sv: A két összekapcsolt socketleíró.

A Linux absztrakt névtér

- A socket állomány hátrányai:
  - Rendszeresen törölnünk kell, különben bind() hiba.
  - Ha elszáll a program, akkor megmarad.
  - Lehet, hogy nincs jogosultságunk az állomány létrehozására.
  - Lehet, hogy az állományrendszer nem támogatja.
- Absztrakt névtér használata:
  - A nevet '\0' karakterrel kezdjük. A végét a cím mérettel jelezzük.
  - Nem jön létre az állományrendszerben.