Linux Programozás

Állománykezelés

I/O multiplexálás

I/O multiplexálás

- Blokkolt I/O
  - „Elhanyagolt” állományleírók
- Nem blokkolt I/O
  - „Adathiány” lekezelése

Nem blokkolt I/O

- open() + O_NONBLOCK
- Ha van adat:
  - read() visszatér az adattal
- Ha nincs adat:
  - read() hibát ad
  - errno értéke EAGAIN
- A hibakezelésben foglalkozunk kell az EAGAIN hibával
- Ha nem rakunk várakozást a ciklusba, akkor „megeszi” a CPU-t

select

```
int select(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, struct timeval *timeout);

int pselect(int nfds, fd_set *readfds, fd_set *writefds,
fd_set *exceptfds, const struct timespec *timeout,
const sigset_t *sigmask);


void FD_CLR(int fd, fd_set *set);

int  FD_ISSET(int fd, fd_set *set);

void FD_SET(int fd, fd_set *set);

void FD_ZERO(fd_set *set);
```

select

- BSD (select), POSIX(pselect)
- „Takarékosabb”
- Szintvezérelt:

  Ha a meghívás előtt is volt a bufferben tartalom, akkor azonnal visszatér.
- Hordozható
- Nem optimális a belső működése
- Korlátos a leírók száma (bitmap-eket használ)
- Módosítja a bemenő paramétereit

poll

```
int poll(struct pollfd *fds, nfds_t nfds, int
timeout);


struct pollfd {
  int fd;
  short events;
  short revents;
};
```

poll

- System V
- Mindazt tudja, mint a select, de
  - Optimálisabb (a kernel a poll-t implementálja)
  - Kevésbé hordozható
  - Nincs korlátozva a leíró szám
  - A bemenete újra felhasználható

epoll

#include <sys/epoll.h>

- Virtuális állományban tárolja a leírókat:
  - epoll_create()
  - epoll_ctl()
- Várakozás
  - epoll_wait()
- Szintvezérelt és élvezérelt is tud lenni
- Csak azokat az elemeket adja vissza, ahol esemény van (max 20)
- Privát adatokat rendelhetünk az elemekhez
- Linux specifikus

Aszinkron olvasás

Példa: „asyncread.c”

Állomány leképezése memóriába

mmap

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

void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

int munmap(void *addr, size_t length);
```

- Leképezést végez
- Az igény szerinti lapozás tölti be.
- Csak egész lapokat kezelhetünk, és laphatárra igazít.
- Ha az eszköz támogatja, akkor a driverrel beszélhetünk.

Védelmi beállítások

- PROT_NONE – semmi
- PROT_EXEC – futtatás
- PROT_READ – olvasás
- PROT_WRITE – írás

- Ne ütközzön az open()-nél használtakkal!

Flags

- MAP_SHARED
  - Lementődik és a többi folyamat is láthatja a módosításokat.
  - Frissítése: msync(), munmap()
- MAP_PRIVATE
  - Más folyamatok nem láthatják a módosítást.
- MAP_ANON, MAP_ANONYMOUS
  - Nincs mögötte állomány.
- MAP_FIXED
  - A címet nem csak javaslatnak veszi.
  - Pontos, laphatárra eső címet igényel.
- MAP_LOCKED
  - A leképezett lapokat a fizikai memóriában tartja.

msync

```
int msync(void *addr, size_t length, int flags);
```

- Flags
  - MS_ASYNC – azonnal visszatér
  - MS_SYNC – szinkronizálás után tér vissza
  - MS_INVALIDATE – a többi leképezés frissítését kérelmezi

Állomány zárolás

Fajtái

- Tájékoztató (advisory lock)
  - Csak konvenció
- Kötelező (mandatory lock)
  - A kernel betartatja
  - Csökkenti a rendszer teljesítményét.

Zároló állományok

- Ellenőrzés és zárolás:

  open() + O_CREAT | O_EXCL
- Feloldás:

  unlink()
- Tájékoztató zárolás
- Egyszerre egy folyamat férhet hozzá az állományhoz, nincs párhuzamos olvasás.
- Csak lokális állományrendszernél
- Elszállás esetén beragad a lock

Rekord zárolás

- System V: lockf(), flock()
- POSIX: még egy metódus fcntl() használatával
- Tájékoztató zárolás
- Területeket zárolhatunk
- Olvasási és írási zárolás (megosztott és kizárólagos)
- Nem tárolódik az állományrendszeren. (Nem csak lokálisan működik.)
- A folyamat leállásakor felszabadul.

Rekord zárolás

```
int fcntl(int fd, int cmd, struct flock *lock);

cmd: F_SETLK, F_SETLKW, F_GETLK

struct flock {
  ...
  short l_type;    /* Type of lock: F_RDLCK, F_WRLCK, F_UNLCK */
  short l_whence;  /* SEEK_SET, SEEK_CUR, SEEK_END */
  off_t l_start;   /* Starting offset for lock */
  off_t l_len;     /* Number of bytes to lock */
  pid_t l_pid;     /* PID of process blocking our lock (F_GETLK only) */
  ...
};
```

- Példa: recordlock.c

Kötelező zárolás

- 1. lépés - Partíció beállítás
  - mand: engedélyezés
  - nomand: tiltás
- 2. lépés - Állomány jelölése
  - setgid + csoport futtatás tiltása
- A root sem férhet hozzá, ha zárolva van.
- Csak lokális állományrendszernél használjuk, mert hálózatinál nagyon lassít.

Kapcsolat a magas szintű állománykezeléssel

Állomány leírók

- Magas szintű:

  struct FILE*

- Alacsony szintű:

  int

- Magasból alacsony szintű:

  int fileno(FILE *stream);

- Alacsonyból magas szintű:

  FILE *fdopen(int fd, const char *mode);

- Párhuzamosan ne használjuk a kettőt, vagy fflush()-el ürítsük ki mindig a magas szintű buffert!