Linux Programozás

Folyamatok, IPC

Folyamat

- A folyamat:
  - program,
  - környezete,
  - használt erőforrások.
- Elkülönített virtuális környezetben fut.
- Szülő-gyerek struktúrába rendeződnek.
- Egymással IPC mechanizmusokon keresztül kommunikálhatnak.

Folyamat létrehozása

```
#include <unistd.h>
pid_t fork(void);
```

- A folyamatot lemásolja (COW), így alkot gyereket.
  - Szülő: A gyerek PID-jével tér vissza.
  - Gyerek: 0 a visszatérési érték
  - (Mínusz érték – hiba, nem jön létre gyerek.)

Folyamat megszüntetése

```
#include <signal.h>
kill(PID, SIGINT);
```

- Erre még visszatérünk.

PID

- Egyedi folyamat azonosító szám.
- Saját PID lekérése:

```
pid_t getpid(void);
```

- Szülő PID lekérése:

```
pid_t getppid(void);
```

Várakozás folyamatra

```
#include <sys/wait.h>
pid_t wait(int *status)
```

- Akkor tér vissza, ha bármelyik gyerek befejezte a működését.
- Visszatérési érték a gyerek PID.
- A „status” a gyerek folyamat visszatérési értéke.
- A gyerek akkor fejezi be a „zombie” létet, ha vissza adta a wait-nek a visszatérési értékét. (Ez kikapcsolható.)

Várakozás folyamatra

```
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *status, int options);
```

- pid:
  - < -1: Olyan gyerek processz, amelynek a csoport azonosítója a „pid” abszolút értéke.
  - -1: Bármelyik gyerek.
  - 0: Olyan gyerek processz, amelynek a csoportja egyezik a hívó csoportjával.
  - >0: Gyerek processz, amelynek a PID-je egyelő a „pid” értékkel.
- options:
  - 0: Megvárja, hogy a megadott gyereknek vége legyen.
  - WNOHANG: Azonnal visszatér ha nincs befejeződött gyerek.
  - WCONTINUED: Akkor is visszatér, ha a gyerek a felfüggesztett állapotból folytatódik.

Folyamat kilépés

```
void exit(int status);
```

- Konvenció:
  - 0: rendben (`EXIT_SUCCESS`),
  - nem 0: hiba (`EXIT_FAILURE`).

Program betöltése és futtatása

- Exec család.
- Betölti a kódot és az elejére ugrik.
- A hívásnál megadott paramétereket és környezeti változókat kapja meg a program.
- Sikeres végrehajtás esetén nem tér vissza. Onnan a folyamat a programot futtatja.
- Elvesznek:
  - Jelzés beállítások
  - Zárolások
- Megmaradnak:
  - Alap tulajdonságok: PID, jogosultságok, prioritás, stb.
  - Megnyitott állományok (kivéve az fcntl() FD_CLOEXEC beállítást)

Exec család

```
#include <unistd.h>
extern char **environ;

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char
*arg, ...);
int execle(const char *path, const char *arg, ...,
char * const envp[]);

int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],
char *const envp[]);
```

Tipikus alkalmazás

- Fork-al új folyamatot hozunk létre, és az exec-el felülírjuk a kódot
  - A COW miatt nincs felesleges másolás.
  - A folyamat visszatérési értéke a program visszatérési értéke.

- Használhatjuk az OS parancsértelmezőjét is (libc):

```
#include <stdlib.h>
int system(const char *command);
```

  - Szöveges parancssort adhatunk meg neki.
  - A „/bin/sh -c <command>”-ot indítja el.
  - Külön folyamatot indít, de szinkron a működése.
  - A visszatérési értéke a parancs visszatérési értéke.

Folyamatok közötti kommunikáció (IPC)

- System V IPC
  - Szemafor
  - Üzenetsor
  - Megosztott memória
- Jelzések (signal)
- Csővezeték
- Socket

System V IPC

- Közös header: `sys/ipc.h`
- Kezelő parancs: `ipcs`
- Az IPC erőforrások közösek, nem folyamathoz kötődnek:
  - Törölni kell őket, ha már nem használjuk. Manuális törlés: `ipcrm`
  - Egyedi azonosítóra van szükség:

```
key_t ftok(const char *pathname, int
proj_id);
```

System V szemafor

- Szokásos szemafor működés.
- A System V IPC szemafor tömböket használ.
- A szemafor tömb több, mint szemafor halmaz:
  - Egyszerre több szemafor műveletet kérhetünk.
  - Vagy mindent végrehajt, vagy semmit!

System V szemafor függvényei

- Header: `sys/sem.h`
- Létrehozás:

```
int semget(key_t key, int nsems, int semflg);
```

  - `semflg` (bites VAGY): `0, IPC_CREAT, IPC_EXCL`
- Törlés:

```
semctl(semid, 0, IPC_RMID, 0);
```

- Művelet: `semop()`
- Kontrol: `semctl()`

System V szemafor művelet

```
int semop(int semid, struct sembuf *sops, unsigned
nsops);
int semtimedop(int semid, struct sembuf *sops,
unsigned nsops, struct timespec *timeout);


struct sembuf
{
  ushort  sem_num; /*a szemafor indexe a tömbben
*/
  short   sem_op;  /*foglalás: -1, elengedés: 1 */
  short   sem_flg; /* jelzőbitek: IPC_NOWAIT,
SEM_UNDO*/
};
```

System V szemafor kontrol

```
int semctl(int semid, int semnum, int cmd, ...)
```

- Műveletek:

Művelet	Leírás
IPC_STAT	Szemaforinformáció lekérdezése. Olvasási jogosultság szükséges.
IPC_SET	Jogosultság, felhasználó- és csoportazonosítók megváltoztatása.
IPC_RMID	Szemafortömb megszüntetése, felébresztve a várakozó processzeket.
GETALL	A szemafortömb elemeinek értékét adja vissza.
GETNCNT	Egy szemaforra várakozó processzek száma.
GETPID	A szemafortömb **utolsó** módosítójának processzazonosítóját kérdezi le.
GETVAL	Egy szemafor értékét adja vissza.
GETZCNT	Egy szemafor nulla (foglalt) értékére várakozó processzek száma.
SETALL	A szemafortömb összes elemének értékét állítja be.
SETVAL	Egy szemafor értékét állítja be.

System V szemafor kontrol

- Ha van utolsó paraméter, akkor típusa:

```
union semun {
  int              val;    /* SETVAL */
  struct semid_ds *buf;    /* IPC_STAT, IPC_SET */
  unsigned short  *array; /* GETALL, SETALL */
  struct seminfo  *__buf; /* IPC_INFO (Linux) */
};

struct semid_ds {
  struct ipc_perm sem_perm;  /*Ownership and permissions */
  time_t          sem_otime; /*Last semop time */
  time_t          sem_ctime; /*Last change time */
  unsigned short  sem_nsems; /*No. of semaphores in set */
};
```

Üzenetsor

- FIFO a programozó által definiált típusú üzenetekkel.
- Általános üzenet formátum:

```
struct msgbuf
{
  long mtype;       /* uzenettipus */
  char mtext[1];    /* adat */
};
```

- Fizikailag egy láncolt listában tárolódik.

Üzenetsor saját üzenet típusa

- Az általános struktúrát lecserélhetjük, például:

```
struct studentinfo
{
  char nev[40];           /* Nev */
  char cim[80];          /* Cim */
  char igszam[20];       /* Igazolvanyszam */
  char megj[80];    /* Megjegyzes */
};

struct studentinfo_msg
{
  long mtype;
  struct studentinfo data;
};
```

Üzenetsor függvényei

- Header: `sys/msg.h`
- Létrehozás:

```
int msgget(key_t key, int msgflg);
```

  - `msgflg` (bites VAGY): `0, IPC_CREAT, IPC_EXCL`
- Törlés:

```
msgctl(mqid, IPC_RMID, 0);
```

- Küldés: `msgsnd()`
- Fogadás: `msgrcv()`
- Kontrol: `msgctl()`

Üzenetsor – küldés

```
int msgsnd(int msqid, struct msgbuf
*msgp, size_t msgsz, int msgflg);
```

- `msqid`: Üzenetsor leíró.
- `msgp`: Üzenet mutató castolva `struct msgbuf*`-re.
- `msgsz`: Az üzenet „`mtext`” része. (A `long` nem számít.)
- `Msgflg`: `0, IPC_NOWAIT`

Üzenetsor – fogadás

```
ssize_t msgrcv(int msqid,struct
msgbuf*msgp, size_t msgsz, long
msgtype, int msgflg);
```

- `msqid`: Üzenetsor leíró.
- `msgp`: Üzenet fogadó buffer.
- `msgsz`: Fogadó buffer „mtext” részének mérete.
- `msgtype` és `msgflg`: lásd következő oldal.

Üzenetsor – fogadás

- `msgtype = 0`: Következő üzenet.
- `msgtype > 0`: A legelső msgtype üzenet.
- `msgtype > 0` és `MSG_EXCEPT`: Az első nem msgtype típusú üzenet.
- `msgtype < 0`: Az első üzenet kisebb, vagy egyenlő típus azonosítóval, mint abs(msgtype).

- Flag értékek (bites VAGY): `0, IPC_NOWAIT, MSG_EXCEPT, MSG_NOERROR`

Üzenetsor kontroll

```
int msgctl(int msqid, int cmd,
struct msqid_ds *buf);
```

Parancs	Leírás
IPC_STAT	Információt másol a *buf* argumentum által mutatott struktúrába.
IPC_SET	A *buf* által mutatott struktúra némely tagja alapján átállítja az üzenetsor tulajdonságait. A figyelembe vett tagok a következők: *msg_perm.uid* *msg_perm.gid* *msg_perm.mode* (az alsó 9 bit) *msg_qbytes*
IPC_RMID	Az üzenetsor megszüntetése.

Üzenetsor kontrol

- Ha van utolsó paraméter, akkor típusa:

```
struct msqid_ds
{
  struct ipc_perm msg_perm; /* Hozzaferesi jogosultsagok */
  struct msg *msg_first;    /* Az elso uzenet a uzenetsor lancolt listajaban */
  struct msg *msg_last;     /* Az utolso uzenet az uzenetsor lancolt listajaban */
  time_t msg_stime;         /* A legutolso kuldes ideje */
  time_t msg_rtime;         /* A legutolso olvasas ideje */
  time_t msg_ctime;         /* A legutolso valtoztatas ideje */
  ushort msg_cbytes;        /* Az uzenetsorban levo bajtok szama (ossszes uzenet) */
  ushort msg_qnum;          /* Az eppen az uzenetsorban levo uzenetek szama */
  ushort msg_qbytes;        /* Az uzenetsorban levo bajtok maximalis szama  */
  ushort msg_lspid;         /* A legutolso kuldo processz azonositoja */
  ushort msg_lrpid;         /* A legutolso olvaso processz azonositoja */
};
```

Megosztott memória

- A megosztott memóriát leképezzük a folyamat virtuális címterébe.
- Gyorsabb, mint bármilyen más mechanizmus, mert csak memória hozzáférés.
- Szinkronizálni kell, mert azt nem biztosítja.
- Az egyes folyamatoknál más-más címen lehetnek a lapok.

Megosztott memória függvényei

- Header: `sys/shm.h`
- Létrehozás:

```
int shmget(key_t key, int size, int shmflg);
```

  - `shmflg` (bites VAGY): `0, IPC_CREAT, IPC_EXCL`
- Törlés:

```
shmctl(shmid, IPC_RMID, 0);
```

- Leképezés: `shmat()`
- Leképezés lezárása: `shmdt()`
- Kontrol: `shmctl()`

Megosztott memória leképezése

```
void *shmat(int shmid, const void
*shmaddr, int shmflg);
```

- `shmid`: Megosztott memória leíró.
- `shmaddr`: Kért cím (laphatár!)
- `shmflg`:
  - `SHM_RND`: laphatárra kerekíti a címet (nem nekünk kell)
  - `SHM_RDONLY`: csak olvasható
- visszatérés: a cím, ahova leképezte

Megosztott memória leképezés megszüntetése

```
int shmdt(const void *shmaddr);
```

- `shmaddr`: A leképezés kezdőcíme

Megosztott memória kontroll

```
int shmctl(int shmid, int cmd,
struct shmid_ds *buf);
```

Parancs	Leírás
IPC_STAT	Információ a megosztott memóriáról.
IPC_SET	Hozzáférési jogosultságok és azonosítók megváltoztatása.
IPC_RMID	A megosztott memória megszüntetése.
SHM_LOCK	A megosztott memória mindvégig a fizikai memóriában marad. Linux specifikus.
SHM_UNLOCK	Engedélyezi a swappinget.

Megosztott memória kontroll

- Ha van utolsó paraméter, akkor a típusa:

```
struct shmid_ds
{
  struct ipc_perm shm_perm; /* Hozzaferes es azonositok beallitasa */
  int shm_segsz;    /* A memoriatartomany merete (bajtokban) */
  time_t shm_atime; /* A legutolso felcsatolas ideje */
  time_t shm_dtime; /* A legutolso lecsatolas ideje */
  time_t shm_ctime; /* Az utolso valtozas ideje */
  unsigned short shm_cpid; /* A letrehozo processz azonositoja */
  unsigned short shm_lpid; /* Az utolso muvelet vegrehajtojanak azonositoja */
  short shm_nattch; /* Az aktualis felcsatolasok szama */
  /*---------- A tobbi tagot a rendszer hasznalja ---------*/
  unsigned short shm_npages; /* A tartomany merete (lapok szama) */
  unsigned long *shm_pages;
  struct vm_area_struct *attaches; /* A felcsatolasok leiroja  */
};
```