Operációs rendszerek

6. gyakorlat:
Processzusok közti kommunikáció
(osztott memória, üzenetsor)

A UNIX System V Release-óta minden rendszer biztosít három egyszerűabsztrakciót a processzusok közti kommunikáció megvalósítására. Ezek:

- Osztott memória szegmens: egy meghatározott méretű memória puffer, melyhez egyszerre több processzus is kapcsolódhat.
- Üzenetsor: processzusok közti ütközésektől mentes üzenettovábbítást biztosít.
- Szemafor készlet: a Dijkstra-féle szemafor kiterjesztése (ezzel csak a következő gyakorlaton foglalkozunk).

Közös tulajdonságok

Minthogy mind a három mechanizmus az operációs rendszer objektuma, így közös jellemzőiket az alábbiakban foglalhatjuk össze:

- Rendszerhívásokkal használhatjuk
- Van külső azonosítója, un. kulcs, amivel az operációs rendszer azonosítja a mechanizmust
- Van belső azonosítója, amivel a processzus azonosítja a mechanizmust
- Hasonlóan a fájlokhoz van tulajdonosa, csoportja (UID/GID), amit a létrehozó processzus effektív jogaiból örököl, és vannak hozzáférési jogok
- Hasonlítanak a rendszerhívások is amivel használhatjuk
- Kontrollja három tevékenységből áll, státusz olvasása, attribútum beállítása és mechanizmus megszüntetése

Osztott memória

Létrehozása és azonosító megszerzése

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

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

A *key* argumentum tartalmazza az osztott memória külső azonosítóját (ez a legtöbb rendszeren *long int* típusú), a *size* a lefoglalni kívánt memória méretét. Az *shmflg* a függvényhívás egyéb paramétereit tartalmazza. Ennek alsó 9 bitje a hozzáférési jogosultságokat (a fájloknál megszokott struktúrában), a további biteket logikai

műveletekkel kapcsolhatjuk be, vagy ki a megfelelő makró használatával. Itt csak az IPC_CREAT opciót fogjuk használni.
Ha a megadott *shmflg* nulla értékű, akkor nem hozunk létre memóriát, csak megszerezzük a megadott kulcsú és méretű szegmens azonosítóját.

A függvény visszatérési értéke pozitív szám, ha sikeres a művelet és ilyenkor az osztott memória azonosítóját adja. Ha sikertelen volt, akkor -1 a visszatérési érték és az *errno* tartalmazza a hiba kódját.

Használat

void shmat(int id, const void* shmaddr, int shmflg);*

int shmdt(const void shmaddr);*

Hogy egy processzus használhasson egy (már létrehozott) memóriát előbb hozzáférést kell kapnia hozzá. Erre használjunk az *shmat* hívást, aminek *id* argumentumába megadjuk az osztott memória azonosítóját, az *shmaddr* és az *shmflg* argumentumokkal megszabhatjuk, hogy milyen címre kérjük az osztott memória leképzését (ennek részleteit a man-ból olvashatjuk). Ha az *shmaddr* helyén *null* pointert adunk meg, akkor az operációs rendszerre bízzuk a cím kiválasztását. A hívás visszatérési értéke az a pointer ahol az osztott memóriához hozzáférhetünk, vagy (void*) -1, ha sikertelen, ilyenkor az *errno* tartalmazza a hiba kódját.

Használat után el kell engedni az osztott memóriát. Erre az *shmdt* szolgál, aminek egyetlen argumentuma az a pointer, amin az osztott memória elérhető. Sikeres végrehajtást a nulla visszatérési érték jelez, még hibát a -1 (hibakód az *errno*-ban ☺).

Attribútumok, kontroll

*int shmctl(int id, int cmd, struct shmid_ds *buf);*

A függvény argumentumai: *id* – az osztott memória azonosítója, *cmd* – egy utasítás, amit tenni szeretnénk az osztott memóriával, és egy *buf* pointer egy pufferre, ami az osztott memória attribútum struktúrája.

IPC_STAT: Ezzel az utasítással lekérdezhető az osztott memória állapota, melyet a megadott *buf* által mutatott struktúrában jelenít meg. Az attribútum struktúra a következő:

```
struct shmid_ds {
    struct    ipc_perm shm_perm;  /* hozzáférési jogok */
```

```
    int  shm_segsz;              /* osztott mem. mérete (bytes) */
    time_t    shm_atime;         /* utolsó rákapcsolódás ideje */
    time_t    shm_dtime;         /* utolsó elengedés ideje */
    time_t    shm_ctime;         /* utolsó módosítás ideje */
    unsigned short shm_cpid; /* a létrehozó PID-je */
    unsigned short shm_lpid; /* az utolsó operációt végző PID-je */
    short     shm_nattch;        /* éppen rákapcsolt processzek száma */
    /* az alábbiakkal nem tudunk mit kezdeni */
    unsigned short   shm_npages;  /* size of segment (pages) */
    unsigned long   *shm_pages;
    struct shm_desc *attaches;    /* descriptors for attaches */
};
```

Az *ipc_perm* struktúra pedig a következő:

```
struct ipc_perm
{
  key_t  key;   /* külső azonosító */
  ushort uid;   /* tulajdonos euid, egid */
  ushort gid;
  ushort cuid;  /* létrehozó euid, egid */
  ushort cgid;
  ushort mode;  /* alsó 9 bitje a hozzáférési jogok */
  ushort seq;   /* egy sorszám */
};
```

IPC_SET: ezzel az utasítással az *ipc_perm* struktúra tartalmát módosíthatjk.

IPC_RMID: megszünteti az osztott memóriát (a *buf* lehet NULL)

Példa: http://www.iit.uni-miskolc.hu/~repasi/OS/peldak/shm/

Üzenetsor

Létrehozása és azonosító megszerzése

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

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

A függvény használata és működése hasonlít az *shmget* függvényhez, azzal a különbséggel, hogy nem kell megadni semmilyen méretet.

Használat

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

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

Üzenet küldésére az *msgsnd* függvényt használjuk. Megadjuk az üzenetsor azonosítóját, az üzenet puffert, az üzenet puffer méretét, és egy jelzést, a függvény viselkedéséről. Visszatérési értéke nulla, ha sikeres és -1, ha nem. Ha az üzenetsor tele van (nem fér több üzenet bele), akkor blokkolódik a hívás és csak akkor tér vissza, ha sikeresen elhelyezte az üzenetet a sorban. Ha megadjuk az *IPC_NOWAIT* jelzést, akkor nem fog várakozni, hanem az *errno* változó EAGAIN értéket kap.

Az *msgrcv* függvény üzenet olvasásra szolgál. Meg kell adni az üzenetsor azonosítóját, az üzenet puffert, annak maximális méretét, a kívánt üzenet típusát és a függvény viselkedését befolyásoló flageket. A függvény hívásakor a processzus blokkolódik, amíg nem áll rendelkezésre üzenet a sorban, ezt a viselkedést lehet az *IPC_NOWAIT* flaggel felülbírálni, ilyenkor az *errno* változó értéke EAGAIN lesz, ha nem volt üzenet a sorban. Visszatérési értéke –1, ha hiba lépett fel, egyébként egy pozitív szám, ami az olvasott üzenet méretét adja vissza.

Az üzenet struktúra egy tetszőleges adatstruktúra lehet, egyetlen megkötés, hogy az elején van egy *long* típusú adattag, ami az üzenet típusát jelzi. Ha az üzenet olvasásakor, az olvasandó üzenet típusának nullát adunk meg, akkor a soron következő üzenetet olvassunk. Ha egy pozitív számot adunk meg, akkor a következő megfelelő típusú üzenetet olvashatjuk, ha pedig negatív számot, akkor a megadott szám ellentettjénél kisebb típusú üzenetek kerülnek olvasásra.

Attribútumok, kontroll

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

A függvény működése megegyezik az *shmctl* működésével, csak az attribútum struktúra tér el:

```
struct msqid_ds {
        struct ipc_perm msg_perm;
        ushort msg_qnum;     /* sorban álló üzenetek száma */
        ushort msg_qbytes;   /* a sor max. mérete (bytes) */
        ushort msg_lspid;    /* az utolsó msgsnd hívó PID-je */
        ushort msg_lrpid;    /* az utolsó msgrcv hívó PID-je */
        time_t msg_stime;    /* utolsó msgsnd ideje */
```

```
        time_t msg_rtime;    /* utolsó msgrcv ideje */
        time_t msg_ctime;    /* utolsó módosítás ideje */
    }
```

Példa: http://www.iit.uni-miskolc.hu/~repasi/OS/peldak/msg/

Irodalom: Dr. Vadász Dénes: Operációs rendszerek c. tárgy jegyzete