

Megosztott memória

Megosztott memória segítségével megoldható, hogy különböző folyamatok ugyanazt a memória területet közösen használják. A közösen használt memóriaterület lehetőséget ad az adatok megosztására, gyors kommunikációra. A rendszer csak a hozzáférési jogosultságokat kezeli, tehát az egyidejű közös erőforrás használat következményeként versenyhelyzet alakulhat ki, ezek kezelése a felhasználói folyamatok feladata. A megosztott memória használat első lépéseként létre kell hozni a megosztott memória területet. Ezt az *shmget* függvénnyel tudjuk megtenni. A függvény által visszaadott azonosító segítségével lehet csatlakozni, leválni, adminisztrálni, törölni a megosztott memóriát. Mielőtt használni szeretnénk a megosztott memóriát, csatlakozni kell az *shmat* függvény segítségével. Az *shmat* függvény adja meg a címet, amelyen keresztül elérhetjük a megosztott memóriát. Használat után célszerű leválni a megosztott memóriáról, amelyet az *shmdt* függvény segítségével tudunk megtenni. Ennek elmulasztása esetén a folyamat kilépésekor a rendszer automatikusan leválasztja a folyamatot. Miután már egyik folyamat sem szeretné használni a megosztott memóriát az *shmctl* függvénnyel tudjuk kitörölni. A kitörlés csak, akkor történik meg, ha minden folyamat levált már a megosztott memóriáról. A rendszerben lévő IPC eszközök (megosztott memória, szemafor, üzenetsor) elérhetők az *ipcs* shell parancs segítségével.

Most nézzünk egy egyszerű példát a megosztott memóriahasználatról. A példaprogramban a folyamat létrehoz egy megosztott memóriát, csatlakozik hozzá, majd a *fork* hívással létrehoz egy gyerek folyamatot. A szülő folyamat beleír egy szöveget a megosztott memóriába, a gyerek folyamat pedig kiolvassa azt. A kiolvasás előtt a gyerek vár 1 másodpercet, hogy a szülő azt be tudja írni. A példában semmit nem teszünk a versenyhelyzet elkerülésére, csak abban bízunk, hogy 1 másodperc alatt már beírta az adatokat a szülő.



```

        sleep(1);
        printf("gyerek: %s", s);
        shmdt(s);
    }

    return 0;
}

```

System V IPC kulcs generálni az *ftok* függvénnyel tudunk. A függvény első paraméterének adjunk egy létező állománynevet, második paraméternek egy változat számot. A függvény megnyitja az állományt és tartalmából a változatszám felhasználásával hasít egy kulcsot. Ugyanazzal az állománnyal és változat számmal meghívva, ugyanazt a kulcsot generálja. Bármelyik paraméter megváltoztatására, más kulcsot fog generálni.

```

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

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

```

Paraméterek:

- *pathname*: egy létező állomány neve, amelyet megnyit és segítségével hozza létre a kulcsot
- *proj_id*: változat száma, ugyanazon *pathname* esetén ennek függvényében generál különböző kulcsot. Csak az alsó 1 byte-ot használja, amelynek értéke nem lehet 0.

Visszatérési érték:

- siker esetén a létrehozott kucs
- hiba esetén -1

Hibák:

- EACCES: jogosultsági hiba
- EBADF: hibás fájlleíró
- EFAULT: hibás cím
- ELOOP: túl sok szimbolikus link az útvonalban
- ENAMETOOLONG: a megadott név, útvonal (*pathname*) túl hosszú
- ENOENT: a megadott útvonal (*pathname*) egyik könyvtára nem létezik
- ENOMEM: nincs elég memória
- ENOTDIR: a megadott útvonal (*pathname*) egyik könyvtára nem könyvtár

Megosztott memóriát létrehozni az *shmget* függvénnyel tudunk, amelyhez az *ftok* függvénnyel létrehozott kulcsot használhatjuk. Ugyanezzel a kulccsal már létrehozott megosztott memóriának az azonosítóját is az *shmget* függvénnyel tudjuk megszerezni.

```

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

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

```

Paraméterek:

- *key*: az *ftok*-kal létrehozott kulcs
- *size*: a memória terület mérete bájtban
- *shmflg* kapcsolók (állománykezelésnél használt jogok is itt adhatók meg):
 - IPC_CREAT: új megosztott memória létrehozása, ha ezt nem adjuk meg, akkor létező megosztott memóriának az azonosítóját adja meg

- IPV_EXCL: kapcsoló esetén hiba történik, ha már létezik megosztott memória a megadott kulccsal

Visszatérési érték:

- siker esetén a megosztott memória azonosítója
- hiba esetén -1

Hibák:

- EACCES: jogosultsági hiba
- ENOENT: nincs ilyen kulccsal megosztott memória, és nem volt IPC_CREAT bekapcsolva
- ENOMEM: nincs elég memória
- EEXIST: IPC_CREAT | IPC_EXCL kapcsolók esetén a megadott kulccsal már létezett megosztott memória
- EINVAL: hibás méret, vagy a létező megosztott memóriaterület kisebb, mint a megadott méret
- ENOSPC: nem lehet több megosztott memóriát létrehozni a rendszeren
- EIDRM: a megosztott memória már törlésre van jelölve

A megosztott memória területet is a „normál” memóriához hasonlóan mutató (memória cím) segítségével tudjuk elérni. A megosztott memóriához csatlakozni és címet kérni az *shmat* függvénnyel tudunk.

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

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

Paraméterek:

- shmid: az *shmget* függvény által megadott azonosító
- shmaddr: a memóriacím, amelyen keresztül szeretnénk elérni a megosztott memóriát
 - ha NULL, akkor a rendszer ad egy címet
 - ha nem NULL és a harmadik paraméterben szerepel az SHM_RND, akkor a rendszer az adott címet, ha szükséges kerekíti és ezen keresztül érhető el a megosztott memória
 - ha nem NULL és az SHM_RND nem szerepel, akkor a címnek az SHMLBA többszörösével kell megegyeznie
- shmflg kapcsolók:
 - SHM_RND: a második paraméterben adott címet szükség esetén kerekíti
 - SHM_RDONLY: csak olvasható lesz a hozzáférés

Visszatérési érték:

- siker esetén a megosztott memória címe
- hiba esetén -1

Hibák:

- EACCES: jogosultsági hiba
- ENOMEM: nincs elég memória
- EINVAL: hibás shmid vagy shmaddr

A megosztott memória területről leválni az *shmdt* függvénnyel lehet. A rendszerből csak akkor lehet törölni a megosztott memóriát, ha nincs rácsatlakozva folyamat. A folyamat kilépésekor a leválás mindenképpen megtörténik.

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

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

Paraméterek:

- shmaddr: a memóriacím, amelyet az *shmat* függvény adott

Visszatérési érték:

- siker esetén 0
- hiba esetén -1

Hibák:

- EINVAL: hibás shmaddr (nincs erre a címre leképezve megosztott memória)

Az *shmctl* függvény segítségével többféle műveletet végezhetünk a megosztott memória területtel. Lekérdezhetjük, beállíthatjuk a tulajdonságait, törölhetjük a megosztott memóriát.

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

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

Paraméterek:

- shmid: a megosztott memória azonosítója (*shmget* adta)
- cmd: parancs
 - IPC_STAT: megosztott memória adatainak a lekérdezése a harmadik paraméterben megadott címre
 - IPC_SET: megosztott memória adatainak a módosítása a harmadik paraméterben megadott tulajdonságokkal
 - IPC_RMID: törlésre jelöli a megosztott memóriát. Az utolsó csatlakozás leválása után fog csak törlődni.
 - buf: megosztott memória tulajdonságait hordozó struktúrára mutató, amelynek szerepe a második paramétertől függ.

```
struct shmid_ds {
    struct ipc_perm shm_perm;
    int      shm_segsz;          /* méret (bytes) */
    time_t   shm_atime;         /* utolsó kapcsolódás */
    time_t   shm_dtime;         /* utolsó leválás */
    time_t   shm_ctime;         /* utolsó módosítás */
    unsigned short shm_cpid;     /* létrehozó PID-je */
    unsigned short shm_lpid;     /* utolsó shmat, shmdt PID-je */
    short     shm_nattch;        /* csatlakozók kapcsolatok száma */
    ...
};

struct ipc_perm {
    key_t key;                  /* létrehozáskor használt kulcs */
    uid_t uid;                  /* tulajdonos UID */
    gid_t gid;                  /* tulajdonos GID */
    uid_t cuid;                 /* létrehozó UID */
    gid_t cgid;                 /* létrehozó GID */
    unsigned short mode;        /* jogosultságok (mint a fájlknál) + SHM_DEST and
                                SHM_LOCKED flags */
    unsigned short seq;         /* sorozatszám */
};
```

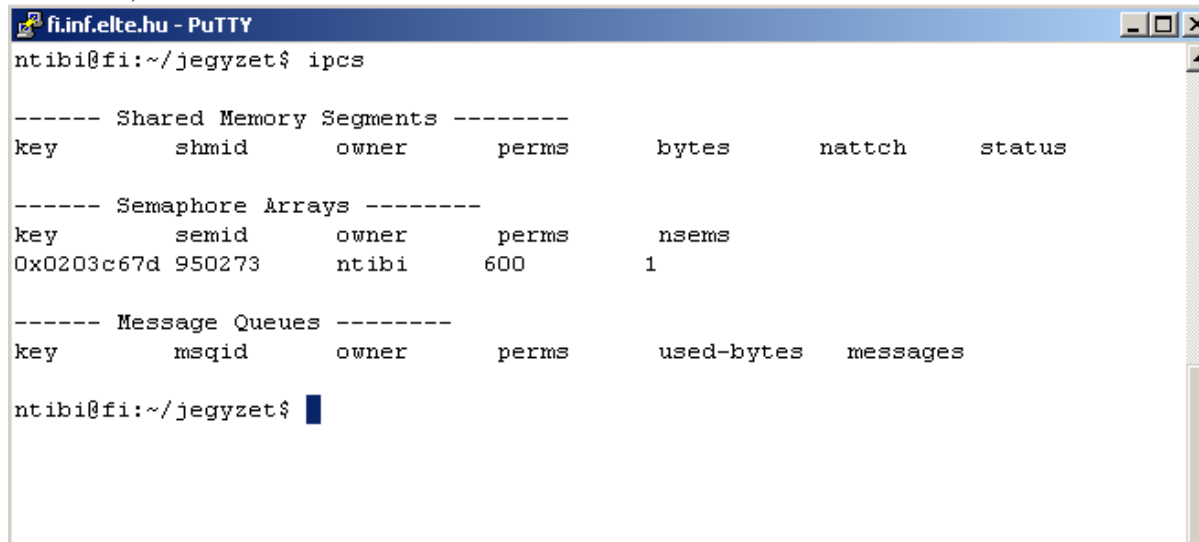
Visszatérési érték:

- siker esetén 0
- hiba esetén -1

Hibák:

- EACCESS: IPC_STAT estén, ha nincs olvasási jog
- EFAULT: cmd IPC_SET vagy IPC_STAT, de a harmadik paraméter nem megfelelő
- EINVAL: hibás az első vagy a második paraméter
- EIDRM: a megosztott memória már törölve lett
- EOVVERFLOW: IPC_STAT esetén uid vagy gid tárolására nincs elég hely a buf struktúrában
- EPERM: nincs jogunk végrehajtani a kért műveletet

Az *ipcs* parancs segítségével tudjuk megnézni az IPC eszközöket (megosztott memória, szemafor, üzenetsor).



```
fi.inf.elte.hu - PuTTY
ntibi@fi:~/jegyzet$ ipcs

----- Shared Memory Segments -----
key          shmid      owner      perms      bytes      nattch     status

----- Semaphore Arrays -----
key          semid      owner      perms      nsems
0x0203c67d  950273    ntibi      600        1

----- Message Queues -----
key          msqid      owner      perms      used-bytes  messages

ntibi@fi:~/jegyzet$
```

Összefoglalás:

- kulcs generálás: *key_t ftok(const char *pathname, int proj_id);*
- megosztott memória létrehozása: *int shmget(key_t key, size_t size, int shmflg);*
 - shmflg paraméterek: IPC_CREATE (létrehozás, ha nincs kapcsolódás), IPC_EXCL (hiba, ha van ilyen kulccsal)
- megosztott memóriához csatlakozás: *int shmat(int shmid, const void *shmaddr, int shmflg);*
 - shmaddr paraméter: NULL (rendszer választ címet), nem NULL
 - shmflg paraméter: SHM_RND (cím kerekítés), SHM_RDONLY (read only)
- megosztott memóriáról való leválás: *int shmdt(const void *shmaddr);*
- megosztott memória lekérdezés, beállítás, törlés: *int shmctl(int shmid, int cmd, struct shmid_ds *buf);*
 - Megosztott memória akkor törölhető, ha az összes folyamat shmdt-vel levált.
 - IPC_STAT: megosztott memória adatainak a lekérdezése a harmadik paraméterben megadott címre
 - IPC_SET: megosztott memória adatainak a módosítása a harmadik paraméterben megadott tulajdonságokkal
 - IPC_RMID: törlésre jelöli a megosztott memóriát. Az utolsó csatlakozás leválása után fog csak törölődni.
 - buf: megosztott memória tulajdonságait hordozó struktúrára mutató, amelynek szerepe a második paramétertől függ.

Szemafor

A szemaforok kölcsönös kizárás megvalósítására használt versenyhelyzetmentes kezelésszerű változók. Ha a közös erőforrásokat használó folyamatokban a kritikus szakaszokat szemaforokkal védjük meg, nem alakul ki versenyhelyzet. A „klasszikus” szemafor kétállású (nyitva, zárva). Nyitott állapotban lévő szemafor esetén a kritikus szakaszba belépő folyamat lezárja „maga mögött” a szemaforot. A lezárt szemaforhoz érkező folyamatok várákoznak, amíg ki nem nyílik a szemafor, akkor közülük egy beléphet a kritikus szakaszba. Természetesen „maga mögött” ismét bezárja a szemaforot.

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

Int main(){

    /* kritikus szakasz kezdete */
    szemafor_zárás;

    /* kritikus szakasz műveletei */

    szemafor_nyitás;
    /* kritikus szakasz vége */

}
```

A kritikus szakasz elején zárjuk a szemaforot. Ha zárva lenne, akkor a folyamat várákozik (blokkolódik), míg végre nem tudja hajtani a zárás műveletét, azaz időközben ki nem nyitották a szemaforot.

A kritikus szakasz végén kinyitjuk a szemaforot. Ez a művelet nem okoz blokkolást.

A System V IPC szemaforok nem negatív egész értékű változók. A szemafor műveletek általában ennek a változónak az értékét egy egész számmal növelik vagy csökkentik. Ha a művelet eredménye negatív szám lenne, a rendszer blokkolja a folyamatot, amíg a művelet végrehajtható nem lesz, azaz a művelet eredménye nem negatív lesz. A rendszer több szemaforot is tud egyszerre kezelni (szemaforkészlet). Ebben az esetben a műveleteket csak akkor hajtja végre a szemaforokon, ha mindegyiken elvégezhető, különben várákoztatja a folyamatot.

Most nézzük egy példát egyszerű szemafor használatra. A szemaforat záráskor eggyel csökkentjük, nyitáskor eggyel növeljük. Ha kezdetben beállítjuk a szemafor értékét 1-re, akkor ez megfelel a nyitott állapotnak, mivel az első lezáró folyamat, tovább mehet és a szemafor értékét eggyel csökkenti, amelyik ezután nulla lesz. A további szemafor zárási kísérletekre (eggyel való csökkentések) a folyamatok blokkolódnak. A nyitás hatására (eggyel való növelés) a szemafor értéke ismét egy lesz.

```
key_t kulcs;
int szemafor_id;
struct sembuf muvelet;

kulcs=ftok(argv[0],1);
szemafor_id=semget(kulcs,1,IPC_CREAT|S_IRUSR|S_IWUSR);

semctl(szemafor_id,0,SETVAL,1)

/* kritikus szakasz kezdete */
muvelet.sem_num = 0;
muvelet.sem_op = -1;
muvelet.sem_flg = 0;
semop(szemafor_id, &muvelet, 1);

/* kritikus szakasz műveletei */
```

Kulcs generálás, majd egyetlen szemaforot tartalmazó szemaforkészlet létrehozása.

Beállítjuk a szemafor kezdeti értékét 1-re, azaz „nyitott” állapotba.

Lezárjuk a szemaforot. A sembuf struktúrában sem_op értékét 1-re állítjuk. A semop függvény megkísérli hozzáadni a szemaforhoz sem_op értékét, blokkolja a folyamatot, ha az eredmény negatív lenne.

```

muvelet.sem_num = 0;
muvelet.sem_op  = 1;
muvelet.sem_flg = 0;
semop(szemafor_id, &muvelet, 1);

/* kritikus szakasz vége */

semctl(szemafor_id, 0, IPC_RMID);

```

Nyitjuk a szemaforot, azaz 1-el
növeljük a szemafor értékét.

A végén töröljük a szemaforot..

Szemaforkészletet létrehozni az *semget* függvénnyel tudunk, amelyhez az *ftok* függvénnyel létrehozott kulcsot használhatjuk. Ugyanezzel a kulccsal már létrehozott szemaforkészlet azonosítóját is a *semget* függvénnyel tudjuk megszerezni.

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

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

```

Paraméterek:

- key: az *ftok*-kal létrehozott kulcs
- nsems: a készletben lévő szemaforok száma
- semflg kapcsolók (állománykezelésnél használt jogok is itt adhatók meg)
 - IPC_CREAT: új szemafor létrehozása, ha ezt nem adjuk meg,, akkor létező szemaforkészlet azonosítóját adja meg
 - IPV_EXCL: kapcsoló esetén hiba történik, ha már létezik szemaforkészlet a megadott kulccsal

Visszatérési érték:

- siker esetén a szemaforkészlet azonosítója
- hiba esetén -1

Hibák:

- EACCES: jogosultsági hiba
- ENOENT: nem létezik a megadott kulccsal szemaforkészlet és nem írtuk elő az IPC_CREAT kapcsolót
- ENOMEM: nincs elég memória
- EEXIST: IPC_CREAT | IPC_EXCL kapcsolók esetén a megadott kulccsal már létezett szemaforkészlet
- EINVAL: hibás szemaforszám, vagy a létező szemaforkészletben lévő szemaforok száma kisebb, mint a megadott *nsems*
- ENOSPC: nem lehet több szemaforkészletet létrehozni a rendszeren

A *semop* függvénnyel tudjuk a szemaforkészlet szemaforainak az értékét változtatni, növelni vagy csökkenteni. A függvény blokkolja a folyamatot, ha a kért művelet nem hajtható végre. Ha több szemaforon végzünk műveletet, akkor minden szemaforon egyszerre végzi el a műveletet, ha az lehetséges, azaz minden szemaforon megvalósítható a kért változtatás.

```

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(int semid, struct sembuf *sops, unsigned nsops);

```

Paraméterek:

- **semid:** a szemaforkészlet azonosítója
- **sops:** a művelet paramétereit tartalmazó tömb kezdőcíme. Egy tömbelem egy szemafor kijelölését (indexét a szemaforkészletben), a műveletet és módosító kapcsolókat tartalmaz.

```
struct sembuf {  
    ushort  sem_num;    /* a szemafor indexe a szemaforkészletben */  
    short   sem_op;     /* ezt az értéket adja hozzá a szemaforhoz */  
    short   sem_flg;    /* kapcsolók: IPC_NOWAIT, SEM_UNDO */  
};
```

- **IPC_NOWAIT:** nem blokkolja a folyamatot, még akkor sem ha nem lehet végrehajtani. Ekkor a semop hibával tér vissza.
 - **SEM_UNDO:** a műveletet visszavonja a rendszer, amikor a folyamat terminál.
- **nsops:** hány szemaforon végezzük egyszerre a műveletet, a második paraméterben szereplő tömb elemszáma

Visszatérési érték:

- siker esetén 0
- hiba esetén -1

Hibák:

- **EFAULT:** sops cím hibás
- **E2BIG:** nsops túl nagy
- **EACCES:** nincs jogunk a művelet végrehajtásához
- **EAGAIN:** IPC_NOWAIT esetén nem lehetett végrehajtani a műveletet
- **EFBIG:** a sops tömbben az egyik szemafor nem létezik a szemaforkészletben
- **EIDRM:** a szemafor törölve lett
- **EINTR:** művelet közben jelzést kaptunk
- **EINVAL:** semid vagy nsops hibás
- **ENOMEM:** nincs elég memória
- **ERANGE:** a szemaforkészlet egyik szemaforának értéke nagyobb, mint a megengedett SEMVMX

Tehát a szemafor értékét a sembuf struktúrában lévő sem_op hozzáadásával lehet módosítani.

- Ha **sem_op > 0**, akkor a szemafor értéke nő. Ez nem okoz blokkolást. Általában a kritikus szakasz végén használjuk.
- Ha **sem_op < 0**, akkor a rendszer megpróbálja a szemafor értékét csökkenteni, ha a szemafor értéke negatívvá válna, akkor nem hajtja végre, hanem blokkolja a folyamatot.
- Ha **sem_op = 0**, akkor a szemafor értéke nem változik. Ha a szemafor értéke nem nulla, akkor a folyamat blokkolódik egészen addig míg a szemafor nulla nem lesz. Ezt nullára várakozásnak is szokták nevezni.

A *semctl* függvény segítségével tudunk információt kérni a szemaforkészletről vagy annak egy szemaforáról, beállítani annak értékét anélkül, hogy blokkolódna a folyamat (kezdeti beállítás). A *semctl* függvény segítségével tudjuk a szemaforkészletet törölni.

```
#include <sys/types.h>  
#include <sys/ipc.h>  
#include <sys/sem.h>  
  
int semctl(int semid, int semnum, int cmd, ...);
```


Paraméterek:

- semid: a szemaforkészlet azonosítója (*semget* adta)
- semnum: a szemafor index a szemaforkészletben
- cmd: parancs
 - IPC_STAT, IPC_SET: a szemaforkészlet adatainak a lekérdezése vagy módosítása a negyedik paraméterben megadott struktúra segítségével.
 - SETVAL: a szemaforkészlet adatainak a módosítása a negyedik paraméterben megadott semun struktúra val mezőjére, feloldja a várakozó folyamatok blokkolását, ha szükséges.
 - GETALL, SETALL: a szemaforkészlet szemaforainak lekérdezése vagy beállítása
 - IPC_RMID: a szemaforkészlet törlése. A blokkolt folyamatok blokkolása feloldódik.
- semun: a negyedik paraméternek használható struktúra

```
union semun {
    int          val;          /* Value for SETVAL */
    struct semid_ds *buf;      /* Buffer for IPC_STAT, IPC_SET */
    unsigned short *array;     /* Array for GETALL, SETALL */
    struct seminfo *__buf;     /* Buffer for IPC_INFO
};

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 */
};

struct ipc_perm {
    key_t key;          /* létrehozáskor használt kulcs */
    uid_t uid;          /* tulajdonos UID */
    gid_t gid;          /* tulajdonos GID */
    uid_t cuid;         /* létrehozó UID */
    gid_t cgid;         /* létrehozó GID */
    unsigned short mode; /* jogosultságok (mint a fájlknál)+ SHM_DEST and
                          SHM_LOCKED flags */
    unsigned short seq;  /* sorozatszám */
};
```

Visszatérési érték:

- siker esetén 0
- hiba esetén -1

Hibák:

- EACCES: nincs jogosultságunk a művelet végrehajtásához
- EFAULT: arg.buf, arg.array nem elérhető
- EINVAL: hibás a cmd vagy a semid paraméter
- EIDRM: a szemaforkészlet törölve lett
- ERANGE: a szemafor új értéke érvénytelen
- EPERM: nincs jogunk végrehajtani a kért műveletet IPC_RMID vagy IPC_SET esetén

Most használjunk a szemafor a megosztott memória témakörben látott példa esetén. A szülő folyamat a megosztott memóriába ír egy szöveget a gyerek folyamat pedig azt kiolvassa. Korábban a gyerek folyamatban a kiolvasás elé raktunk egy másodperces várakozást, ami nem biztosít elegendő védelmet. Használjunk szemafor. Kezdetben állítsuk a szemafor nulla értékre, azaz zárt állapotba. A szülő folyamat a megosztott memória feltöltése után nyitja csak ki a szemafor. A gyerek folyamat a megosztott memória kiolvasás előtt lezárja a szemafor, ha zárva lenne (a szülő még nem töltötte fel),

akkor blokkolódik a folyamat. Miután a szemafor kinyílik, a gyerek folyamat kiolvassa a megosztott memória tartalmát.

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <wait.h>

#define MEMSIZE 1024

int szemafor_letrehozas(const char* pathname,int szemafor_ertek){
    int semid;
    key_t kulcs;

    kulcs=ftok(pathname,1);
    if((semid=semget(kulcs,1,IPC_CREAT|S_IRUSR|S_IWUSR))<0)
        perror("semget");

    if(semctl(semid,0,SETVAL,szemafor_ertek)<0)
        perror("semctl");

    return semid;
}

void szemafor_muvelet(int semid, int op){
    struct sembuf muvelet;

    muvelet.sem_num = 0;
    muvelet.sem_op = op;
    muvelet.sem_flg = 0;

    if(semop(semid,&muvelet,1)<0)
        perror("semop");
}

void szemafor_torles(int semid){
    semctl(semid,0,IPC_RMID);
}

int main (int argc,char* argv[]) {

    pid_t child;
    key_t kulcs;
    int sh_mem_id,semid;
    char *s;

    kulcs=ftok(argv[0],1);
    sh_mem_id=shmget(kulcs,MEMSIZE,IPC_CREAT|S_IRUSR|S_IWUSR);
    s = shmat(sh_mem_id,NULL,0);

    semid = szemafor_letrehozas(argv[0],0);

    child = fork();
    if(child>0){

        char buffer[] = "Hello!\n";
        sleep(1);
        strcpy(s,buffer);
        szemafor_muvelet(semid,1);
    }
}
```

Szemaforkészlet létrehozása és kezdeti értékének a beállítása.

Szemafor nyitása vagy zárása op értékétől függően.

Szemaforkészlet létrehozása nulla kezdeti értékkel, azaz zárt állapotban.

A szülő folyamat beírja a „hello!\n” szöveget a megosztott memóriába, majd kinyitja a szemafort.

```

shmdt(s);
wait(NULL);
szemafor_torles(semid);
shmctl(sh_mem_id, IPC_RMID, NULL);

} else if ( child == 0 ) {

    /* kritikus szakasz kezdete */
    szemafor_muvelet(semid, -1);
    printf("gyerek: %s", s);
    szemafor_muvelet(semid, 1);
    /* kritikus szakasz vége */

    shmdt(s);
}

return 0;
}

```

Megvárjuk a gyerek folyamatot, majd töröljük a szemaforot.

A kritikus szakasz kezdetén lezárjuk a szemaforot, majd a végén kinyitjuk.

Javasolt feladat gyakorlatra:

Egy megosztott memóriaterületen tároljunk egy szöveget. Az egyik folyamat 3. másodpercenként olvassa ki a tartalmát és írja ki a stdout-ra, a másik folyamat olvasson be egy szöveget az stdin-ről és rakja a megosztott területre, amíg az "exit" szöveget nem kapja. A folyamatok kritikus szakasza a megosztott memória olvasása ill. írása, ezt szemaforral védjük.

Összefoglalás:

- kulcs generálás: *key_t* *flok(const char *pathname, int proj_id);*
- szemaforkészlet létrehozása: *int semget(key_t key, int nsems, int semflg);*
 - **IPC_CREAT:** új szemafor létrehozása, ha ezt nem adjuk meg, akkor létező szemaforkészlet azonosítóját adja meg
 - **IPV_EXCL:** kapcsoló esetén hiba történik, ha már létezik szemaforkészlet a megadott kulccsal
- Művelet a szemaforkészlet szemaforaival: *int semop(int semid, struct sembuf *sops, unsigned nsops);*
 - A szemafor értéke: 0..SEMVMX, szemafor művelet ezt csökkenti vagy növeli, hozzáadódik a *sem_op* értéke
 - Ha a művelet nem hajtható végre (szemafor értéke negatív lenne), akkor a rendszer blokkolja a folyamatot.
 - Több szemaforon egyszerre hajtja végre a műveletet, ha mindegyiken végrehajtható.
 - A szemafor értékét a *sembuf* struktúrában lévő *sem_op* hozzáadásával lehet módosítani.
 - Ha *sem_op* > 0, akkor a szemafor értéke nő. Ez nem okoz blokkolást. Általában a kritikus szakasz végén használjuk.
 - Ha *sem_op* < 0, akkor a rendszer megpróbálja a szemafor értékét csökkenteni, ha a szemafor értéke negatívvá válna, akkor nem hajtja végre, hanem blokkolja a folyamatot.
 - Ha *sem_op* = 0, akkor a szemafor értéke nem változik. Ha a szemafor értéke nem nulla, akkor a folyamat blokkolódik egészen addig míg a szemafor nulla nem lesz. Ezt nullára várakozásnak is szokták nevezni
 - *sembuf* struktúra *semflg*:
 - **IPC_NOWAIT:** nem blokkolja a folyamatot, még akkor sem ha nem lehet végrehajtani. Ekkor a *semop* hibával tér vissza.
 - **SEM_UNDO:** a műveletet visszavonja a rendszer, amikor a folyamat terminál
- Szemaforkészlet adatainak lekérdezése, beállítása, szemaforkészlet törlés: *int semctl(int semid,*

int semnum, int cmd, ...);

- cmd: parancs
 - IPC_STAT, IPC_SET: a szemaforkészlet adatainak a lekérdezése vagy módosítása a negyedik paraméterben megadott struktúra segítségével.
 - SETVAL: a szemaforkészlet adatainak a módosítása a negyedik paraméterben megadott semun struktúra val mezőjére, feloldja a várakozó folyamatok blokkolását, ha szükséges.
 - GETALL, SETALL: a szemaforkészlet szemaforainak lekérdezése vagy beállítása
 - IPC_RMID: a szemaforkészlet törlése. A blokkolt folyamatok blokkolása feloldódik.