Fájl műveletek:

Fájlok megnyitása - kézikönyv man 2 open
Példa: megnyitás csak olvasására

```
int f1;
if((f1 = open("fajlnev.txt", O_RDONLY))<0){ <= elérési út és fájlnév, paraméterek | elvalasztva
    perror("open f1");   <= hibakezelés elkerülendő a fölösleges segmentation fault hibákat!
    exit(1)     <= hiba esetén érdemes a kilépni ha a fájl megnyitása hibával zárult minden művelet a
                    fájlon hibát fog generálni

}
```

Példa: megnyitás írásra, olvasására, ha nem létezik létrehozásra

```
if((f1 = open("/home/szabi/fajlnev.txt", O_CREAT| O_RDWR, 0666))<0){ <= utolsó paraméter a
        perror("open f1");                                                  létrehozott fájl jogosultságok
        exit(1);
}
```

Fájlok írása - kézikönyv man 2 write

```
int n;      <= beírandó byte-ok száma
if(write(f1, &buff[0], n) != n)   <= a write visszatéríti, hogy hány byte-ot tudott beírni ellenőrzéskor ezt érdemes
        perror("write");            ellenőrizni ha nem sikerült annyi byte-ot beírni ahányat akartunk akkor hibát térít vissza.
```

A write második paramétere egy const void * mutató fontos a kiírandó puffer kezdő címét megadni nem csak a nevét ez esetben hibás működésre lehet számítani.

Fájlok olvasása - kézikönyv man 2 read

```
if((n = read (f1, &buff[0], 512 ))<0)
        perror("read");
```

Az olvasás hasonló az íráshoz az f1 fájlból a buff tömb 0 címétől kezdődően megpróbál beolvasni 512 byte-ot, visszatérítési érték n a beolvasott byte-ok száma. Ha a fájl végéig akarok olvasni akkor

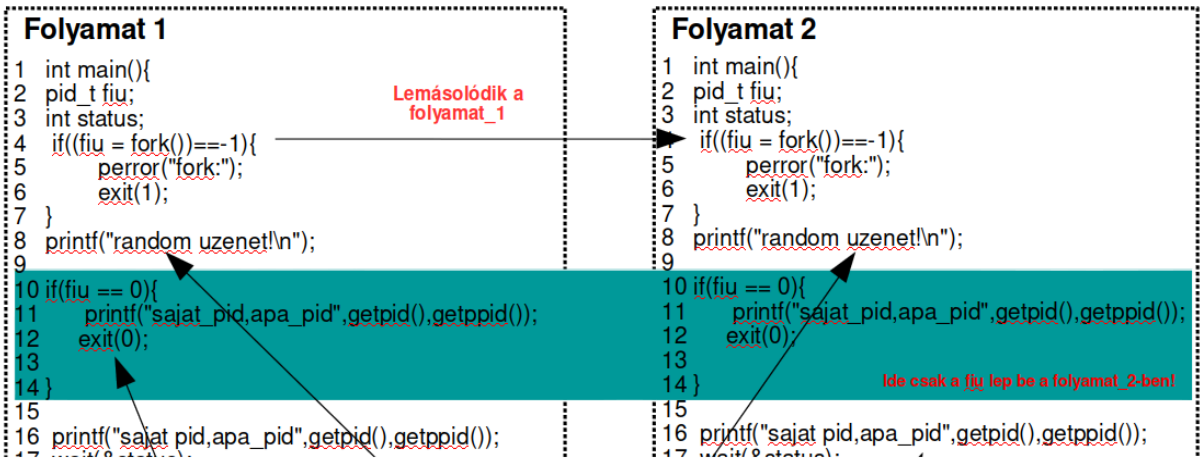
```
while ( (n = read (f1, &buff[0], 512 ) ) > 0)
```

itt is fontos hogy a read void * mutatót vár a tömb kezdő címére!

Fájlok bezárása - kézikönyv man 2 close

```
close(f1) ;
```

Fork:



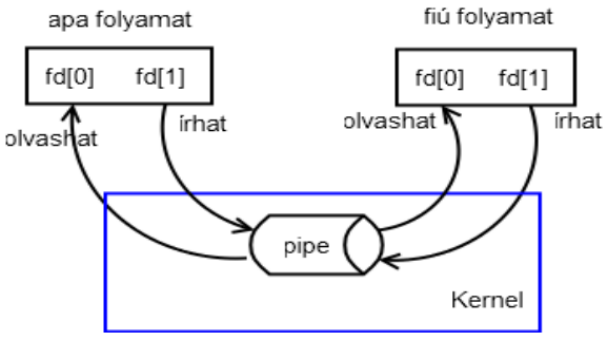

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/unistd.h>

int main (int argc, char * argv[]){
pid_t fiu;
int status;
if((fiu = fork())==-1){
                perror("fork:");
                exit(1);
}
printf("teszt uzenet ketszer\n");

if(fiu == 0){
    printf("sajat_pid: %d,apa_pid: %d\n",getpid(),getppid());
    exit(0);

}
printf("sajat_pid: %d,apa_pid: %d\n",getpid(),getppid());
wait(&status);
  exit(0);
}
```

Csővezeték:



Csővezeték létrehozása - kézikönyv man 2 pipe

int fd[2]; <= két fájlazonosítót hozunk létre fd[0] olvasható fd[1] írható

```
if(pipe(fd)<0){
        perror("pipe:"); <= hiba kiírás
        exit(1);  <= ha létrehozáskor hiba volt értelmetlen tovább menni minden írás és olvasás hibát fog adni
}
```

Írás olvasás ugyanúgy történik mint a fájlok esetében azzal a különbséggel, hogy alapértelmezetten a read() blokkolásos egy pipe esetében. Mivel a kommunikáció half-duplex a helyes működés érdekében a csővezeték nem használt végét bezárjuk! Csővezeték csak öröklődéssel hozható létre apa-fiu folyamatok közt ezért a csővezeték létrehozása a fork() előtt kell legyen.

Példe pipe apa-fiú (apa ír fiú olvas):

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/unistd.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>


int main (int argc, char * argv[])
{
int fd[2];
pid_t fiu;
int n;
int status;
```

```
        if(pipe(fd)<0){  //pipe letrehozasa fork elott!
                perror("pipe:");
                exit(1);
        }

        if((fiu=fork())<0){  //fiu letrehozasa
                perror("fork:");
                exit(1);
        }


        if(fiu==0){  //ide csak a fiu lep be
                char buff[64];
                close(fd[1]); //iras lezaras
                if((n = read (fd[0], &buff[0], 64 ))<0) //csovezetek olvasas
                        perror("pipe read");
                printf("atkuldott byte-ok szama: %d\n",n);
                printf("%s\n",buff);
                exit(0);
        }
        //ide csak az apa jut el felteve ha az if exit-el zarul!
        char buff[]="Teszt uzenet!";
        close(fd[0]);  //olvasas lezaras
        if(write(fd[1], &buff[0], strlen(buff)) != strlen(buff)) //csovezetekbe valo iras
        perror("pipe write");

        wait(&status);
        exit(0);
}
```

Üzenetsorok:

Üzenetsorok segítségével kommunikálhat két folyamat egymással a csővezetékkel ellentétben itt már mindkét irányba folyhat a kommunikáció.

Az üzeneteknek típusa és tartalma van ezt a strukturát fel kell építeni!

Példa üzenet struktúra:

```
typedef struct msgbuf {
      long    mtype;     <= üzenet típusa ugyanazon az üzenetsoron több típusú üzenet mozoghat.
      char   mtext[SIZE];  <= maga az üzenet SIZE legalább 1 kell legyen!
      } message_buf;
```

Üzenetsor létrehozása előtt egy kulcsot kell generálni ami azonosítani fogja az üzenetsort a kernelben. Ahhoz hogy két folyamat kommunikálhasson az üzenetsoron egymással ez a kulcs mindkettőben meg kell egyezzen!

Kulcsgenerálás – kézikönyv man ftok
key_t key;
key = ftok(".",'a'); <= két érték kell a generáláshoz egy elérési út és egy 0-255 közötti szám a példában az 'a' betű karakterkódja. Ha az elérési út "." az azt jelenti mindkét folyamat ugyanabból a mappából kell induljon hogy a kulcsok megegyezzenek!

key = ftok("/usr/bin",65); <= elérési út "/usr/bin" érték 65.

Üzenetsor létrehozás – kézikönyv man 2 msgget
int msgid;   <= üzenetsor azonosítója

```
   if ((msgid = msgget(key, IPC_CREAT | 0660 )) < 0) { <=      üzenetsor létrehozása az előre kigenerált key
                                                               kulccsal 0660 jogokkal ha már létezik az
                                                               üzenetsor és van joga folyamatnak akkor arra
                                                               csatlakozik hanem létrehoz egy újat
      perror("msgget"); <= hiba esetén kiírja a hibát
      exit(1); <= létrehozáskor érdemes kilépni ha sikertelen minden írás és olvasás ez után sikertelen lesz
   }
```

Példa privát üzenetsorok létrehozására (csak apa-fiu folyamatok használhatják):

```
if ( (msgid = msgget (IPC_PRIVATE, IPC_CREAT | 0660))<0 ) { <=   a key kulcs helyett
                                                                 IPC_PRIVATE opciót
                                                                 használjuk
        perror("msgget");
        exit(1);
}
```

Üzenet küldés – kézikönyv man 2 msgsnd

```
   if (msgsnd(msgid, &sbuf, buf_length, 0) < 0) { <=    msgid üzenetsor azonosító, &sbuf a létrehozott msgbuff
                                                        struktúra címe!,  buf_length az msgbuff struktúra mtext
                                                        mezejének hossza ha rövidebb az üzenet mint a puffer
                                                        mérete akkor ennek a hossza nem kell a teljes puffert
                                                        hiába átküldeni, 0 jelző bitek mi nem használtuk man 2
                                                        msgsnd itt elolvasható miként paraméterezhető a
                                                        függvény.
      perror("msgsnd"); <= hiba kiírás
   }
```

Üzenet fogadás – kézikönyv man 2 msgrcv
msgrcv alapértelemezetten blokkolásos függvény!

```
  if ( msgrcv ( msgid, &sbuf, SIZE, msgtype,0 ) < 0 ) { <=    msgid üzenetsor azonosító, &sbuf a létrehozott
                                                              msgbuff struktúra címe!, SIZE az msgbuff
                                                              struktúra mtext mezejének hossza,  msgtype az
                                                              üzenet típusa az msgbuff struktúra mtype értéke
                                                              az msgrcv csak a beállított típusú üzenetet
                                                              olvassa az üzenetsorról!,  0 jelző bitek man 2
                                                              msgrcv.
      perror("msgrcv");
  }
```

Üzenetsor törlése – kézikönyv man 2 msgctl (ez a függvény nem csak törlésre használható paraméterezni is lehet az üzenetsort)

```
  if ( msgctl ( msgid, IPC_RMID, NULL ) < 0 ) <=   msgid üzenetsor azonosító,  IPC_RMID törlés parancs
                                                   az msgctl további paraméterei a man 2 msgctl
                                                   kézikönyvben, NULL egy  msqid_ds struktúra nem
                                          használjuk.
       perror ("msgctl");
```

Példa üzenetsor:
A következő kód létrehoz egy üzenetsort ha nem létezik írás és olvasás joggal minden felhasználónak /bin elérési út 44 azonosító, küld egy Teszt üzenetet aminek a típusa 1 és várakozik egy 3 típusú üzenetre aztán bezárja az üzenetsort és kilép.

```
#include <stdio.h>
#include <string.h>
#include <sys/unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define SIZE     64
typedef struct msgbuf {
        long    mtype;
        char   mtext[SIZE];
        } message_buf;

int main()
{
    int msgid;
    message_buf sbuf;
    key_t key;
    size_t buf_length;

   key = ftok("/bin",44);

   if ((msgid = msgget(key, IPC_CREAT | 0666 )) < 0) {
        perror("msgget");
        exit(1);
   }
   (void) strcpy(sbuf.mtext, "Teszt");
   buf_length = strlen (sbuf.mtext) + 1 ;
   sbuf.mtype=1;
   if (msgsnd(msgid, &sbuf, buf_length, 0) < 0) {
        perror("msgsnd");
   }
   if ( msgrcv ( msgid, &sbuf, SIZE, 3,0 ) == -1 ) {
        perror("msgrcv");
   }
   printf("%s", sbuf.mtext);
   if ( msgctl ( msgid, IPC_RMID, NULL ) < 0 ){
          perror ("msgctl");
   }
   exit(0);
}
```

Shell parancsok:

ipcs -q	listázás man ipcs
ipcrm -q msgid	eltávolítás man ipcrm

Szemaforok:

Fogalmak: kritikus terület (critical section)
osztott erőforrás (shared resource)
2 alapművelet (UP és DOWN)

Létrehozása azonos az üzenetsorhoz. Itt egy szemafor tömböt kell létrehozni ha csak egy szemaforra van szükségünk akkor egy 1 elemű tömböt hozunk létre. Up növelem a szemafor értéket ha lehetséges ha nem akkor várakozok, Down csökkentem a szmeafor értékét ha lehet ha nem akkor várakozok.

Kulcsgenerálás – kézikönyv man ftok
key_t key;
key = ftok(".",'a'); <= két érték kell a generáláshoz egy elérési út és egy 0-255 közötti szám a példában az 'a' betű karakterkódja. Ha az elérési út "." az azt jelenti mindkét folyamat ugyanabból a mappából kell induljon hogy a kulcsok megegyezzenek!
key = ftok("/usr/bin",65); <= elérési út "/usr/bin" érték 65.

Szemafor létrehozás – kézikönyv man 2 semget
int semid; <= szemafor azonosítója

```
if ((semid = semget(key, 2,IPC_CREAT | 0660 )) < 0) { <=   szemafor létrehozása az előre kigenerált key kulccsal 0660 jogokkal ha már létezik a szemafor és van joga folyamatnak akkor arra csatlakozik hanem létrehoz egy újat. A második paraméter határozza meg a szemafor tömb hosszát hány darab szemafort akarunk létrehozni jelen estebe 2.
    perror("semget"); <= hiba esetén kiírja a hibát
    exit(1); <= létrehozáskor érdemes kilépni ha sikertelen minden up és down művelet ez után sikertelen lesz
}
```

Példa privát szemafor létrehozására (apa-fiu):

```
if ( (semid = semget (IPC_PRIVATE, 2, IPC_CREAT | 0660))<0 ) { <=   a key kulcs helyett IPC_PRIVATE opciót használjuk
        perror("semget");
        exit(1);
}
```

Az up, down illetve a nullára való várakozás műveleteket egy sembuf típusú változóban kell deklarálni.
Műveletek deklarálása – kézikönyv man 2 semop

```
struct sembuf down = {0, -1, 0};  <=  első paraméter a szemafor amelyen végrehajtom a műveletet, második paraméter a művelet (up +1,down -1,wait_zero 0),utolso paraméter flag (IPC_NOWAIT,SEM_UNDO) man 2 semop
struct sembuf up = {0, 1, 0};
struct sembuf wait_zero = {0, 0, 0};
```

Művelet végrehajtása – kézikönyv man 2 semop

```
if( semop (semid, &up, 1 )<0) <=   semid azonosítóval rendelkező szemaforon egy up művelet előzőleg deklarálva mint struct sembuf 1-hány szemaforon végzem el.
      perror("down1"); <= hibát kiírom
```

Szemafor törlése - kézikönyv man 2 semctl (ez a függvény nem csak törlésre használható paraméterezni is lehet a szemafort)

```
if ( semctl ( semid, IPC_RMID, NULL ) < 0 ) <=
        perror ("semctl");
```

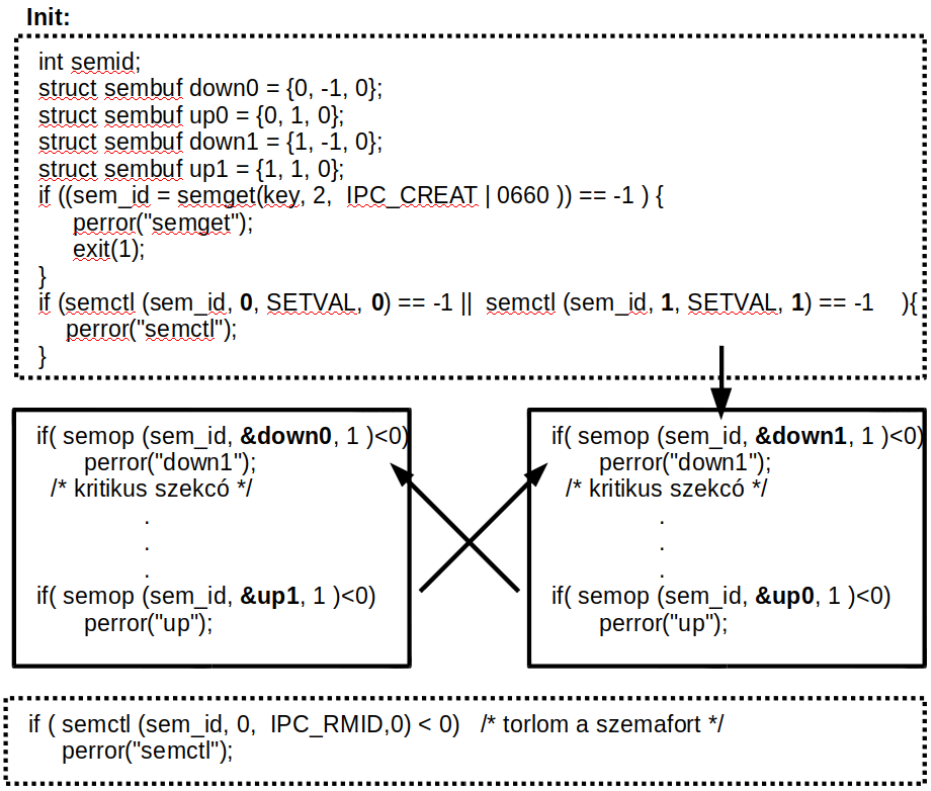
semid szemafor azonosító, **IPC_RMID** törlés parancs a semctl további paraméterei a man 2 semctl kézikönyvben, NULL egy semid_ds struktúra nem használjuk.

Szemafor kezdő értékének beállítása - kézikönyv man 2 semctl

```
if (semctl (semid, 0, SETVAL, 1) == -1 ){ <=
    perror("semctl");
}
```

a 0 szemafor értéket 1-re állítom tehát elvégezhető rajta egy down művelet

Példa:



Shell parancsok:

ipcs -s listázás man ipcs
ipcrm -s semid eltávolítás man ipcrm

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

int main()
{

int sem_id;
key_t key;
pid_t fiu;

/* struktura: hanyadik szemafor, muvelet, flag=0 vagy IPC_NOWAIT vagy SEM_UNDO*/
   struct sembuf down = {0, -1, 0};
   struct sembuf up = {0, 1, 0};
   struct sembuf down1 = {1, -1, 0};
   struct sembuf up1 = {1, 1, 0};

key = ftok(".",'a');
   if ((sem_id = semget(key, 2,  IPC_CREAT | 0660 )) == -1 ) {
      perror("semget");
                        exit(1);
   }

   if (semctl (sem_id, 0, SETVAL, 0) == -1 ||  semctl (sem_id, 1, SETVAL, 1) == -1    ){
   /* az 1-es szemafornak van 1 erteke, tehat eloszor a down1 hajthato vegre */
      perror("semctl");
   }
            if((fiu = fork())<0)
                        perror("fork");

            if (fiu == 0){
                        if( semop (sem_id, &down, 1 )<0) /* probalkozas*/
         perror("down1");

      printf("masodik folyamat a kritikus szekcioban!\n");   /* kritikus resz */

      if( semop (sem_id, &up1, 1 )<0) /* probalkozas*/
         perror("up");
            exit(0);
            }

            sleep(2);

   if( semop (sem_id, &down1, 1 )<0) /* probalkozas*/
      perror("down");

   printf("elso folyamat a kritikus szekcioban!\n");   /* kritikus resz */

   if( semop (sem_id, &up, 1 )<0) /* probalkozas*/
      perror("up1");

            wait(NULL);
    if ( semctl (sem_id, 0,  IPC_RMID,0) < 0)   /* torlom a szemafort */
         perror("semctl");

exit(0);
}
```

Osztott memória

Az osztott memória több folyamat által használt közös memória tartomány, a folyamatok közti adatcsere leggyorsabb formája.

Létrehozása azonos az üzenetsoréhoz illetve a szemaforéhoz.

Kulcsgenerálás – kézikönyv man ftok

```
key_t key;
key = ftok(".",'a'); <=   két érték kell a generáláshoz egy elérési út és egy 0-255 közötti szám a példában az 'a'
                          betű karakterkódja. Ha az elérési út "." az azt jelenti mindkét folyamat ugyanabból a
                          mappából kell induljon hogy a kulcsok megegyezzenek!
key = ftok("/usr/bin",65);  <= elérési út "/usr/bin" érték  65.
```

Osztott memória szegmens létrehozás – kézikönyv man 2 shmget

```
int shmid;    <= osztott memória szegmens azonosítója

if ( (shmid = shmget (key, sizeof(int), 0660 | IPC_CREAT)) == -1) { <= osztott memória szegmens
                                                                      létrehozása az előre kigenerált key
                                                                      kulccsal 0660 jogokkal ha már létezik
                                                                      foglalás és van joga folyamatnak akkor
                                                                      arra csatlakozik hanem létrehoz egy
                                                                      újat. A második paraméter határozza
                                                                      meg a memória zóna méretét byte-ban
                                                                      esetünkben sizeof(int) tehát 4 byte.
  perror("shmget");<= hiba esetén kiírja a hibát
  exit(1); <= létrehozáskor érdemes kilépni ha sikertelen minden osztott memória művelet ez után sikertelen lesz
}
```

Példa privát osztott memória szegmens létrehozására (apa-fiu):

```
if ( (shmid = shmget (IPC_PRIVATE, sizeof(int), 0660 | IPC_CREAT)) == -1) { <=     a key kulcs helyett
                                                                                   IPC_PRIVATE opciót
                                                                                   használjuk
        perror("shmget");
        exit(1);
}
```

Osztott memória szegmens kezdő címének egy folyamattérbeli címmel való összekötése – kézikönyv man 2 shmat

```
char * shmem ; <= az osztott memória szegmens kezdő címét mindig eltároljuk külön egy char * változóban!
int * valt;

if ( (shmem = ( char* ) shmat ( shmid, 0, 0)) == (void*) -1){<=  attach művelet shmid az osztott memória
                                                                 szegmens azonosítója a második paraméter ha
                                                                 nulla akkor a rendszer keres egy szabad címet és
                                                                 oda csatolja a memória szegmenst a harmadik
                                                                 paraméter a flagek (man 2 shmat).
      perror ("shmat");
}

valt = (int *) shmem <= cast a szükséges típusra az shmem nem kerül direkt felhasználásra.
```

Osztott memória szegmens folyamattérből való leválasztása – kézikönyv man 2 shmdt

```
if ( shmdt ( shmem ) < 0) <= detach
     perror("shmdt");
```

Osztott memória szegmens törlése - kézikönyv man 2 shmctl (ez a függvény nem csak törlésre használható paraméterezni is lehet a memória szegmenst)

```
  if ( shmctl ( shmid, IPC_RMID, NULL ) < 0 ) <=
```

shmid memória szegmens azonosító, **IPC_RMID** törlés parancs a shmctl további paraméterei a man 2 shmctl kézikönyvben, NULL egy shmid_ds struktúra nem használjuk.

```
   perror ("shmctl");
```

Shell parancsok:

ipcs -m listázás man ipcs
ipcrm -m shqmid eltávolítás man ipcrm

Példa osztott memória:

Az osztott memóriába történő írás illetve olvasás szinkronizálására szemaforokat használunk két folyamat ugyanabban az időben nem írhatja vagy olvashatja ugyanazt a memória szegmenst.

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

int main(){

int sem_id;
int shmid;
int * ip;
char * shmem ; /* ez a mutato mutat az osztott memoriara, !!! char* */
key_t key;
pid_t fiu;

/* struktura: hanyadik szemafor, muvelet, flag=0 vagy IPC_NOWAIT vagy SEM_UNDO*/
   struct sembuf down = {0, -1, 0};
   struct sembuf up = {0, 1, 0};
   struct sembuf down1 = {1, -1, 0};
   struct sembuf up1 = {1, 1, 0};

key = ftok(".",'a');
if ((sem_id = semget(key, 2,  IPC_CREAT | 0660 )) == -1 ) {
   perror("semget");
   exit(1);
}
if ( (shmid = shmget (key, sizeof(int), 0660 | IPC_CREAT)) == -1) {
   perror("shmget");
```

```
    exit(1);
}

if (semctl (sem_id, 0, SETVAL, 0) == -1 ||  semctl (sem_id, 1, SETVAL, 1) == -1    ){
   /* az 1-es szemafornak van 1 erteke, tehat eloszor a down1 hajthato vegre */
   perror("semctl");
}

if ( (shmem = ( char* ) shmat ( shmid, 0, 0)) == (void*) -1){  /* attach */
     perror ("shmat");
}

ip = (int*) shmem; //cast
if((fiu = fork())<0){
   perror("fork");
   exit(1);
}
if(fiu==0){
          if( semop (sem_id, &down1, 1 )<0) /* probalkozas*/
                    perror("down1");
          *ip=2457; /* ez az ertek az osztott memoriaba kerul */

          if( semop (sem_id, &up, 1 )<0) /* probalkozas*/
                    perror("up");
          if ( shmdt ( shmem ) < 0)
                    perror("shmdt"); /* detach fiu*/
          exit(0);
}

if( semop (sem_id, &down, 1 )<0) /* probalkozas*/
   perror("down");

printf("az i erteke: %d\n",*ip);   /* ertek olvasasa*/

if( semop (sem_id, &up1, 1 )<0) /* probalkozas*/
   perror("up1");

if ( shmdt ( shmem ) < 0)
    perror("shmdt"); /* detach apa*/

wait(NULL);

if ( semctl (sem_id, 0,  IPC_RMID,0) < 0)   /* torlom a szemafort */
    perror("semctl");

if ( shmctl ( shmid , IPC_RMID, 0) <0 )   /* torlom az osztott memoria szegmenst */
    perror("shmctl");

exit(0);
}
```