Lucrarea de laborator Nr. 04

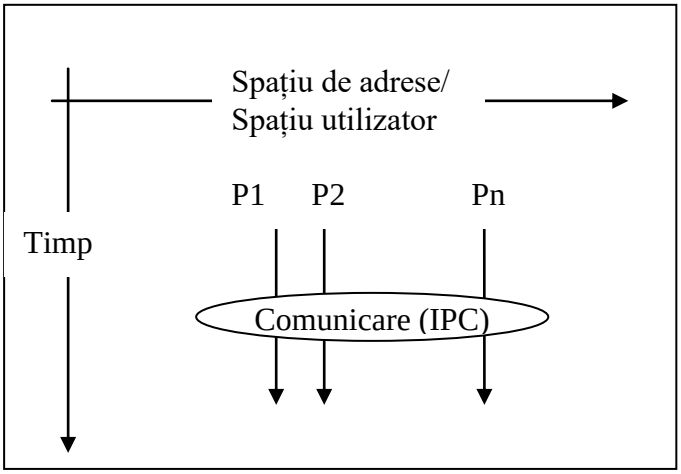
Tehnica IPC - comunicare între procese: *memorie partajată, cozi de mesaje, semafoare, fișiere partajate, semnale, bufere de comunicare (pipes), socluri (sockets).*

Aplicații demonstrative (apeluri utilizate: *ftok(), shmget(), shmat(), shmdt(), shmctl())*

1. Crearea unui proces care generează chei de identificare resursă IPC. Se pune în evidență faptul că pentru aceleași date de intrare se generează aceiași valoare de cheie. Generalizând, procese diferite generează aceeași cheie dacă au date intrare identice, lucru important în IPC.
2. Crearea a două procese P1 și P2. Procesele NU sunt în relație de părinte-copil. Procesul P1 crează și introduce date într-un segment de memorie partajată și se termină. Procesul P2, startat anume după terminarea procesului P1, citește datele din segmentul de memorie partajată, le afișează, șterge segmentul de memorie creat de P1 și se termină. Aplicația pune în evidență faptul că segmentul de memorie partajată odată creat nu mai este dependent de starea procesului creator, P1. În acest exemplu NU este necesară o sincronizare între procese.
3. Crearea unei structuri de procese în pieptene, formată dintr-un proces părinte și doua procese copil care comunică între ele prin intermediul unui segment de memorie partajată. Procesul părinte așteaptă terminarea proceselor copil. Se pune în evidență necesitatea sincronizării.

A. Comunicare inter-proces prin : *memorie partajată, cozi de mesaje, semafoare.*

Prin utilizarea tehnicii pentru comunicare IPC- *Inter-process Communication*- procesele pot comunica unele cu altele. Deoarece fiecare *proces* are propriul spațiu de adresă și un spațiu utilizator unic (vezi figura de mai jos) comunicarea inter-proces este rezolvată de către Kernel (nucleul sistemului de operare Unix) care are acces la întreaga memorie.



Prin tehnica IPC se poate solicita kernelului să aloce spațiul necesar pentru a comunica între *procese*. Comunicarea este rapidă.

Tipuri de IPC

Tehnicile IPC-uri care permit unui proces să comunice cu alte procese se realizează prin:

- *Memoria partajată* (**Shared Memory**) - Procesele pot face schimb de informații în memoria partajată. Un proces crează o porțiune de memorie pe care un alt proces o poate accesa. Este cea mai rapidă formă de inter-comunicare. Ceea ce trebuie remarcat este faptul ca o zona de memorie partajata nu poseda nici un mecanism de sincronizare. Este responsabilitatea programatorului sa asigure *accesul exclusiv* (excludere mutuală- *mutex*) la memorie. Pentru sincronizare se pot folosi semafoare.

- *Coadă de mesaje* (**Message Queue)** - este o listă structurată și ordonată a segmentelor de memorie în care *procesele* stochează sau prelucrează date. Tehnica este utilizată în multiplexarea mesajelor asincrone și este legată de utilizarea semafoarelor.
- *Semafoare* (**Semaphores)** - Oferă un mecanism de sincronizare a *proceselor* care accesează aceeași resursă. Nu se transmit date cu un *semafor*; pur și simplu coordonează accesul la resursele partajate.

NOTĂ

Toate facilitățile IPC au o cheie de identificare (KEY) precum și un identificator unic (<ipc>ID, notat *shm*ID, *msq*ID și respectiv *sem*ID) care sunt folosite pentru a identifica o facilitate IPC

Facilitățile IPC sunt create de către un *creator* și aparțin unui *proprietar*.

Diferența dintre *creator* și *proprietar* al unei intrări de facilitate IPC.

PROPRIETAR (owner)- *Login name*-ul proprietarului intrării facilității IPC.
CREATOR (creator) - *Login name*-ul creatorului intrării facilității IPC.

Creatorul este caracterizat prin *CUID* și *CGID* ale procesului ce a creat intrarea. *Creatorul* nu poate fi schimbat.
Proprietarul este indicat prin *UID* și *GID* și poate fi schimbat de către *creator* definit prin *CUID* și *CGID*.

Exemplu de intrare de facilitate IPC de *memorie partajată*:

```
------ Shared Memory Segment Creators/Owners --------
shmid      perms      cuid       cgid       uid        gid
1310720    600        student1   studenti1  student2   studenti2
```

Facilitățile IPC - ***Coadă de mesaje****/ Message queue,* **Semafoare**/ *Semaphores* ***sau Memorie partajată****/ Shared memory* – au *permisiuni de acces* (ex. *perms 600 = RW- --- ---*) de citire și scriere, dar nu și permisiuni de execuție, pentru *proprietar*, *grup* și *alții*, la fel ca fișierele obișnuite. Ca la fișiere, procesul de creare identifică *proprietarul* implicit ca fiind *creatorul*. Spre deosebire de fișiere, la facilitățile IPC, *creatorul* poate atribui altui utilizator calitatea de *proprietar* de facilitate IPC pe care ulterior o poate revoca în favoarea altui utilizator.

Un *utilizator* poate fi membru al mai multor grupuri de *utilizatori.* Pe de altă parte o resursă (fișier, ***Coadă de mesaje****/ Message queue,* **Semafoare**/ *Semaphores sau* ***Memorie partajată****/ Shared memory)* are ca *proprietar* exact un *grup* și un *utilizator*. Dacă un *utilizator* este membru al unui *grup de proprietari de resursă* atunci permisiunile de acces se aplică acelui utilizator, mai puțin cele forțate de permisiunile de acces ale *utilizatorului proprietar*.
Pentru clarificare, considerăm cazul unui fișier, pentru următoarea secvență de comenzi rulate în *root* (!):

```
# id↵
uid=0(root) gid=0(other)
```

touch fisier↵ ⇒ crearea fișierului *fisier* / este gol/ în grupul *other*

stat fisier↵ ⇒ se observă *utilizator* = root, *grup* = other

```
  File: `fisier'
  Size: 0         Blocks: 1    IO Block: 131072 regular empty file
Device: 2b90008h/45678600d      Inode: 102         Links: 1
Access: (0644/-rw-r--r--)  Uid: (  0/  root)   Gid: (  1/  other)
Access: 2017-10-30 18:35:03.230223246 +0300
Modify: 2017-10-30 18:36:34.217905730 +0300
Change: 2017-10-30 18:36:34.217905730 +0300
```

groups↵ ⇒afișează grupurile din care face parte utilizatorul root;

prima *poziție reprezintă grupul principal, adică root.*
root other bin sys adm uucp mail tty lp nuucp daemon

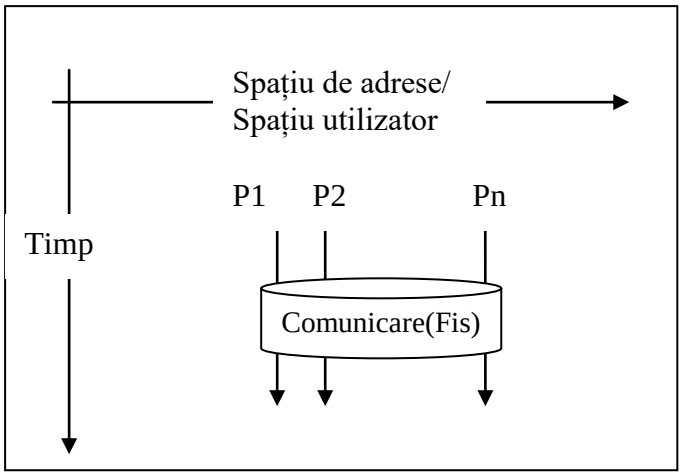
```
# newgrp root↵                ⇒ setează grupul pe grupul principal root
   # id↵
 uid=0(root) gid=0(root)

   #rm fisier; touch fisier↵   ⇒ recrearea fișierului fisier în grupul root
   # stat fisier↵              ⇒ se observă utilizator = root, grup = root
  File: `fisier'
  Size: 0         Blocks: 1    IO Block: 131072 regular empty file
Device: 2b90008h/45678600d      Inode: 112         Links: 1
Access: (0644/-rw-r--r--)  Uid: (  0/  root)   Gid: (  0/  root)
Access: 2017-10-30 29:46:50.367080263 +0300
Modify: 2017-10-30 29:46:50.367080263 +0300
Change: 2017-10-30 29:46:50.367086842 +0300
   #
```

Utilizatorul root, conform comenzii *groups*, este membru al următoarelor grupuri de *utilizatori*: *root other bin sys adm uucp mail tty lp nuucp daemon.* Grupul principal din care face parte *utilizatorul root* este *root* (este primul nume de grup listat de comanda *groups*). Creatorul fisierului *fisier* este *root* din cadrul grupului *other* (comanda *stat* afiseaza acest lucru). Ulterior creatorul fisierului *fisier* este *root* din cadrul grupului *root,* grup principal din care face parte *utilizatorul root*.

B. Alte metode de comunicare inter-procese: *fișiere partajate, semnale, bufere de comunicare (pipes), socluri (sockets), variabile de mediu, cod de retur.*

1. Prin fișiere partajate

O metodă poate mai simplă de comunicare inter-procese este utilizarea unor fișiere accesibile tuturor *proceselor.* În acest caz însă operațiile de comunicare se reduc în fapt la operații de intrare/ieșire mari consumatoare de timp.



Pentru sincronizare se pot folosi *semafoare*, dar și mecanisme de *lock* pe fișiere (fcntl)

2. Prin *semnale (*Signals*), bufere de comunicare (*Pipes*) sau socluri (*Sockets*)*.

a) *Semnale (***Signals***)- Procesele* pot dialoga între ele cu ajutorul s*emnalelor*; *semnale* ce sunt caracterizate prin *număr* şi *nume* (*i, n*). Un *proces* care primeşte un *semnal* de la alt *proces* trebuie să-l analizeze şi, funcţie de natura *semnalului,* să-l ia în considerare după cum

urmează: să-l trateze (de exemplu *procesul* trece dintr-o stare în alta sau dintr-un mod de execuţie în altul), să-l ignore, pentru a-şi continua execuţia sau să se termine (de exemplu, în cazul unui *semnal* neprevăzut).

b) *Bufere de comunicare* (**Pipes**) - Oferă un mod de comunicare inter-procese prin schimb de mesaje. Buferele de comunicare anonime oferă o cale de comunicare inter-procese care rulează pe același calculator, iar buferele de comunicare denumite (Named pipes) oferă o cale de inter-comunicare pentru procesele care rulează pe calculatoare diferite din cadrul unei rețele.

c) *Socluri* (**Sockets**) - Oferă un mod de comunicare inter-procese prin schimb de mesaje între două procese care rulează pe același calculator sau pe calculatoare diferite în cadrul unei rețele. Reprezintă o cale de comunicare inter-procese care lucrează la fel ca *Buferele de comunicare* (*pipes*), dar la nivel de rețea.

3. **Prin *variabile de mediu sau cod de retur***.

În cadrul unui program C se pot accesa variabilele de mediu, prin evidențierea celui de-al treilea parametru (opțional) al funcției main, ca în exemplul următor:

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

unde *char *environ[ ]* desemnează un vector de pointeri la șiruri de caractere, ce conțin variabilele de mediu și valorile lor. Șirurile de caractere sunt de forma VARIABILA=VALOARE. Vectorul e terminat cu NULL.

Un program C își poate manipula mediul folosind funcțiile ***getenv(), putenv(), setenv() și unsetenv().***
Prin convenție, șirurile din environ au forma „nume=valoare”. *nume* face distincție între majuscule și minuscule și nu poate conține caracterul „=".
Variabilele de mediu pot fi plasate în mediul shell-ului prin comanda *export* în din bash.
Shell-urile în stil Bourne acceptă sintaxa

NUME=valoare

pentru a crea o definiție a variabilei de mediu numai în domeniul de aplicare al procesului care execută comanda.
Mai jos sunt date câteva variabilel de mediu oferite de sistemul de operare Linux UBUNTU, shell bash (pentru a afla lista completă pe prompter se dă comanda *printenv*)

USER=xxxx - numele de logare a utilizatorului
SHELL=/bin/bash - shell-ul de lucru
HOME=/home/xxxx - directorul implicit (home directory)

Variabilele de mediu mai pot fi transmise unui proces și prin apeluri ***execle()*** respectiv ***execve().***

4. **Prin *cod de retur***.

Codul de retur (*exit status*) a unui proces este un număr întreg, de valoare mică, transmis de un *proces copil* la un *proces părinte* atunci când procesul copil se termină printr-un apel *exit()* sau *return. Procesul părinte* poate prelua această valoare (indicată în apelul *exit()* sau *return* al copilului) prin utilizarea unei funcții *wait()/waitpid()*

C. Apeluri utilizate în aplicațiile demonstrative:

- **Funcții IPC apelate: (detalii complete pe site): ftok(), shmget(), shnat(), shmdt(),shmctl()**

ftok () - **convertește un *identificator de cale fișier* în combinație cu un *identificator de proiect* într-o *cheie de identificare pentru oricare din resursele System V IPC*** (numită în continuare *key, respectiv shm_key*)

Sintaxă:

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

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

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

On some ancient systems, the prototype was:

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

ftok() utilizează fișierul existent și accesibil identificat prin *pathname* și *proj_id* (diferit de 0) pentru a genera o dată de tip **key_t** utilizată de funcțiile **msgget(), semget(),** sau **shmget()**. Valoarea rezultată este aceeași în cazul utilizării aceluiași *pathname* în combinație cu același *proj_id*. În caz contrar, valoarea rezultată e diferită. Algoritmul de generale are la baza ultimii 16 biți nesemnificativi ai numărului de i-node a lui *pathname* și ultimii 8 biți nesemnificativi a lui *proj_id*, rezultatul fiind pe 32 biții. Tipic pentru *proj_id* se utilizeaza un caracter ASCII. Unicitatea valorii *key* generată nu e garantată.

ftok() în caz de succes returnează valoarea *key* de tip **key_t** a cheii generate.
ftok() în caz de eșec returnează -1 și *errno* este setat pentru a identifica eroarea.

Exemplu apel:

```
// calcul KEY memorie partajata- rezultatul pe 32 biti
#define FISIER "."    // din 16 biti din numarul de i-node
                      //                    a dir. curent
#define ID_PROIECT 11 // si din ultimii 8 biti a lui 11
                      // (poate fi orice val. chiar si cod litera)

shm_key = ftok (FISIER, ID_PROIECT);
```

shmget () - alocă un *segment de memorie partajată System V* (numit în continuare *shm*)

Sintaxă:

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

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

shmget() în caz de succes returnează *identificatorul segmentului de memorie partajată* (shmID) asociat valorii argumentului *key*. Apelul este utilizat fie pentru a obține *shmID* anterior creat atunci cînd *shmflg* este 0 (**O_CREAT**) și *key* nu are valoarea de **IPC_PRIVATE**, fie pentru crearea unui nou identificator de segment. În plus, cei mai puțini semnificativi 9 biți ai *shmflg* specifică permisiunile de acces dobândite RWX RWX RWX la nivel *proprietar*, *grup* și *ceilalți*. Permisiunea X (de execuție) nu este utilizată.

shmget() în caz de eșec returnează -1 și *errno* este setat pentru a identifica eroarea. De exemplu dacă *shmflg* specifică **IPC_CREAT I IPC_EXCL** și segmentul de memorie partajată există pentru valoarea *key*, atunci **shmget**() eșuează, iar *errno* ia valoarea **EEXIST**. (analog cu efectul combinației **O_CREAT | O_EXCL** pentru **open()**.)

Exemplu apel:

```
shm_key = ftok (FISIER, ID_PROIECT);

shm_id = shmget ((key_t) shm_key, sizeof (*shm_ptr), IPC_CREAT | 0666);
```

shmat () - atașează un segment de memorie partajată System V identificat prin ***shmid*** la spațiul de adrese al procesului apelant. Adresa la care se face atașarea este specificată prin ***shmaddr***. În cazul în care ***shmaddr*** este ***NULL***, sistemul alege prima adresă liberă de segment, iar *shmflg* nu este utilizat.

Sintaxă:

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

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

shmat() în caz de succes returnează adresa zonei de memorie partajate și actualizează câmpurile din structura de control a segmentului de memorie partajată *shmid_ds* după cum urmează:

shm_atime este setat la timpul curent.
shm_lpid este setat la timpul curent process-ID a procesului apelant.
shm_nattch este incrementat cu 1.

shmat() în caz de eșec returnează *(void *)* -1 și *errno* este setat pentru a identifica eroarea.

Exemplu apel:

```
shm_key = ftok (FISIER, ID_PROIECT);

shm_id = shmget ((key_t) shm_key, sizeof (*shm_ptr), IPC_CREAT | 0666);

shm_ptr = (struct mem_partajata *) shmat (shm_id, NULL, 0);
```

shmdt () - deatașează un segment de memorie partajată System V localizat la adresa specificată prin ***shmaddr*** din spațiul de adrese al procesului apelant. Pentru a fi detașat un segment trebuie să fi fost obiectul unei atașări printr-un apel **shmat()**.

Sintaxă:

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

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

shmat() în caz de succes returnează 0 și actualizează câmpurile din structura de control a segmentului de memorie partajată *shmid_ds* după cum urmează:

shm_atime este setat la timpul curent.
shm_lpid este setat la timpul curent process-ID a procesului apelant.
shm_nattch este decrementat cu 1. Dacă devine 0 segmentul este efectiv șters, altfel el este marcat în ștergere.

shmdt() în caz de eșec returnează -1 și *errno* este setat pentru a identifica eroarea.

Exemplu apel:

```
shm_key = ftok (FISIER, ID_PROIECT);

shm_id = shmget ((key_t) shm_key, sizeof (*shm_ptr), IPC_CREAT | 0666);

shm_ptr = (struct mem_partajata *) shmat (shm_id, NULL, 0);

shmdt (shm_ptr);   // elibereaza segmentul de memorie partajata punctat de shm_ptr
```

shmctl () - efectuează operații de control specificate prin ***cmd*** asupra unui *segment de memorie partajată System V* a cărui indentificare este făcută printr-un ***shmid*** dat. Argumentul ***buf*** este un pointer la o structură ***shmid_ds*** definită în ***<sys/shm.h>*** și care arată astfel:

```
struct shmid_ds {
    struct ipc_perm shm_perm;    /* Ownership and permissions */
    size_t          shm_segsz;   /* Size of segment (bytes) */
    time_t          shm_atime;   /* Last attach time */
    time_t          shm_dtime;   /* Last detach time */
    time_t          shm_ctime;   /* Last change time */
    pid_t           shm_cpid;    /* PID of creator */
    pid_t           shm_lpid;    /* PID of last shmat(2)/shmdt(2) */
    shmatt_t        shm_nattch;  /* No. of current attaches */
    ...
};
```

Structura ***ipc_perm*** este definită mai jos (câmpurile evidențiate sut setabile prin utilizarea ***cmd*** cu

valoarea **IPC_SET**):

```
struct ipc_perm {
    key_t          __key;    /* Key supplied to shmget(2) */
    uid_t          uid;      /* Effective UID of owner */
    gid_t          gid;      /* Effective GID of owner */
    uid_t          cuid;     /* Effective UID of creator */
    gid_t          cgid;     /* Effective GID of creator */
    unsigned short mode;     /* Permissions + SHM_DEST and
                                SHM_LOCKED flags */
    unsigned short __seq;    /* Sequence number */
};
```

Sintaxă:

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

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

shmctl() în caz de succes de regulă returnează 0 (pentru excepții vezi site-ul) și efectuează operații de control asupra *segmentului de memorie partajată* în funcție de valoarea ***cmd***.

Câteva valori valide pentru ***cmd***:

IPC_STAT copiază informații din structura de date din kernel asociată pentru un ***shmid*** dat într-o structură ***shmid_ds*** punctată de ***buf***. Apelantul trebuie să aibă permisiunea de read pentru segmentul de memorie partajată.

IPC_SET scrie valori dintr-o structură de date ***shmid_ds*** punctată de ***buf*** într-o structură ***shmid_ds*** din kernel asociată unui ***shmid*** **.**

IPC_RMID marchează un segment pentru a fi distrus. *Segmentul* va fi efectiv distrus numai atunci când ultimul proces atașat segmentului a executat destașarea. (ex., când ***shm_nattch*** a unui membrul a structutii asociate ***shmid_ds*** devine 0).

shmdt() în caz de eșec returnează -1 și *errno* este setat pentru a identifica eroarea.

Exemplu apel:

```
struct shmid_ds shmid_ds, *shm_id_ds;          // Structura de control segment
shm_id_ds = & shmid_ds;                        //   memorie partajata
                                               //       punctata de shm_id_ds

shm_id = shmget ((key_t) shm_key, sizeof (*shm_ptr), IPC_CREAT | 0666);

shm_ptr = (struct mem_partajata *) shmat (shm_id, NULL, 0);


shmctl(shm_id, IPC_STAT , shm_id_ds);   // Citirea din structura de controlul al segmentului de
                                        //   memorie partajată
shmctl (shm_id, IPC_RMID, NULL);      // Executa operatia  IPC_RMID
                                      // (marcheaza segmentul  pentru a fi distrus)
```

D. Aplicații demonstrative:

1. Crearea unui proces care generează chei de identificare resursă IPC. Se pune în evidență faptul că pentru aceleași date de intrare se generează aceiași valoare de cheie. Generalizând, procese diferite generează aceeași cheie dacă au date intrare identice, lucru important în IPC.

- **Program ipc_shm_gen_key.c**

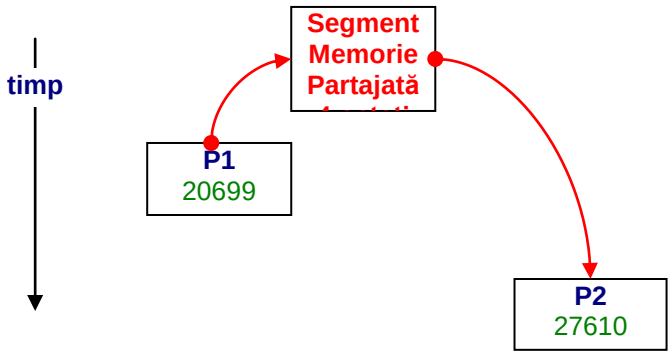
```
/** ipc_shm_gen_key.c **/
    /*  generează chei de identificare resursă IPC */
    /* Linux Microknoppix 4.12.7-64
     _POSIX_C_SOURCE= 200809
     Generare shm key
     Concluzie program:
        la date de intrare identice
        generari succesive identice
    */
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>
    //
    #include <sys/types.h>
    #include <sys/ipc.h>
    // <sys/shm.h>

    /* generare KEY segment memorie partajata- rezultatul pe 32 biti */
    #define FISIER1 "/etc/passwd"   // din 16 biti din numarul de i-node al fisierului
    #define FISIER2 "."             // din 16 biti din numarul de i-node  al directorului curent
    #define ID_PROIECT 11           // si din ultimii 8 biti a lui 11(poate fi orice val. chiar si cod litera)

    int main (void)
    {
     key_t shm_key;                 // Cheie de identificare segment  memorie partajata
     int i;
     printf("_POSIX_C_SOURCE= %ld\n\n",_POSIX_C_SOURCE);
     /* generare shm_key */
     printf("Intrare \"/etc/passwd\" si 11\n");
     for (int i = 1; i<5; i++)
     {
      shm_key = ftok (FISIER1, ID_PROIECT);
      printf("\t%2d - Cheie SHM generata (key) = 0x%08lx\n", i, shm_key);
     };
     printf("\n\n");
     printf("Intrare \".\" si 11\n");
```

```
 for (int i = 1; i<5; i++)
 {
  shm_key = ftok (FISIER2, ID_PROIECT);
  printf("\t%2d - Cheie SHM generata (key) = 0x%08lx\n", i, shm_key);
 };
 exit(0);

}
```

Operare

```
$ gcc -o ipc_shm_gen_key ipc_shm_gen_key.c
$ ./ipc_shm_gen_key

_POSIX_C_SOURCE= 200809

Intrare "/etc/passwd" si 11
    1 - Cheie SHM generata (key) = 0x0b140045
    2 - Cheie SHM generata (key) = 0x0b140045
    3 - Cheie SHM generata (key) = 0x0b140045
    4 - Cheie SHM generata (key) = 0x0b140045


Intrare "." si 11
    1 - Cheie SHM generata (key) = 0x0b142735
    2 - Cheie SHM generata (key) = 0x0b142735
    3 - Cheie SHM generata (key) = 0x0b142735
    4 - Cheie SHM generata (key) = 0x0b142735
```

2. Crearea a două procese P1 și P2. Procesele NU sunt în relație de părinte-copil. Procesul P1 crează și introduce date într-un segment de memorie partajată și se termină. Procesul P2, startat anume după terminarea procesului P1, citește datele din segmentul de memorie partajată, le afișează, șterge segmentul de memorie creat de P1 și se termină. Aplicația pune în evidență faptul că segmentul de memorie partajată odată creat nu mai este dependent de starea procesului creator, P1. În acest exemplu NU este necesară o sincronizare între procese.



- **P1- Program ipc_shm_zone.c**

```
/* ipc_shm_zone.c */
    /*
     Procesul  crează și introduce date într-un segment de memorie partajată și se termină.
    */
    // Linux Microknoppix 4.12.7-64
    // _POSIX_C_SOURCE= 200809
    #include <stdio.h>
    #include <stdlib.h>
```

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

/* calcul KEY memorie partajata- rezultatul pe 32 biti */
#define FISIER "/etc/passwd"     // din 16 biti din numarul de i-node fisier
#define ID_PROIECT 11            // si din ultimii 8 biti a lui 11 (poate fi orice val. chiar si cod litera)

int main (void)
{
 /* secventa proces parinte */
 key_t shm_key;          // Cheie de identificare segment memorie partajata
 int shm_id;             // Identificator segment memorie  partajata

  struct mem_partajata_zona {
     int zona1;          // 4 octeti
     char zona2[10];     // 10 octeti
     char zona3[20];     // 20 octeti
  };

 struct mem_partajata_zona *shm_ptr_zona;  // pointer zona Memorie partajata

 struct shmid_ds shmid_ds, *shm_id_ds;  // Structura de control segment memorie partajata
 shm_id_ds = & shmid_ds;                //     punctata de shm_id_ds

 printf("_POSIX_C_SOURCE= %ld\n\n",_POSIX_C_SOURCE);
 printf("\nP1 (PID= %ld)\n", (long)getpid());

 shm_key=ftok (FISIER, ID_PROIECT);          // generare shm_key
 printf("\tProces P1 - cheie SHM generata (key) = 0x%08lx\n", shm_key);

 /* alocare segment  memorie partajata SHM_KEY de 36 octeti (multimplu de 4 / 34 --->36) */
 /* permisiuni de acces 0666 adica RW_RW_RW_ (0666)*/
 if ((shm_id = shmget ((key_t) shm_key, sizeof(struct mem_partajata_zona), IPC_CREAT | 0666)) == -1)
 {
  (perror("err. shmget()\n"));
  exit (EXIT_FAILURE);
 }
 printf("\tProces P1 - identificator SHM (shmID) = %ld\n", shm_id);
 shmctl(shm_id, IPC_STAT , shm_id_ds);    // citire structura de control al SHM
 printf ("\tNumar procese atasate= %u (urmeaza atasarea)\n",shm_id_ds->shm_nattch); // =0

 /* Ataseaza la segmentul de memorie partajata identificat prin
    shm_id spatiul de adrese (zona partajata) al procesului apelant. */
 printf ("\nAtasare ZONA la shmID= %ld", shm_id);
 shm_ptr_zona = (struct mem_partajata_zona *) shmat (shm_id, NULL, 0);
 if ((void *) shm_ptr_zona == (void *) -1)
 {
  perror ("err. shmat");
  shmctl (shm_id, IPC_RMID, NULL);    // Executa operatia de  control IPC_RMID
                                      // (marcheaza segmentul  pentru a fi distrus)
  exit (EXIT_FAILURE);
 };
 printf("\nAdresa ZONA  (shm_ptr_zona) = %x\n", shm_ptr_zona);

 /* Citirea din structura de controlul al segmentului de  memorie partajat*/
  shmctl(shm_id, IPC_STAT , shm_id_ds);
```

```
/* Afisarea  datelor din structura de control
   segment memorie partajata (o parte din ele) */

printf ("\nAfisarea  datelor din structura de control (o parte)\n");
printf ("\tDim.seg.mem %d octeti\n",shm_id_ds->shm_segsz);
printf ("\tPID creator= %u\n",shm_id_ds->shm_cpid);
printf ("\tNumar procese atasate= %u\n",shm_id_ds->shm_nattch); // =1
printf ("\tCheia seg.mem= 0x%08lx\n",shm_id_ds->shm_perm.__key);
printf ("\tPerm acces= %o\n",shm_id_ds->shm_perm.mode);

shm_ptr_zona->zona1=10;                    // Scriu in zona 1 din segmentul de memorie partajata
sprintf(shm_ptr_zona->zona2, "%s", "ABC");  // Scriu in zona 2 din segmentul de memorie partajata
sprintf(shm_ptr_zona->zona3, "%s", "DEF");  // Scriu in zona 3 din segmentul de memorie partajata


//shmctl (shm_id, IPC_RMID, NULL);     //   Numai la punerea la punct a programului
                                       //   executa operatia de control IPC_RMID- segmentul va fi citit in alt proces
                                       // (NU marcheaza segmentul de memorie partajata
                                       //                  pentru a fi distrus)
printf ("\nAfisarea  datelor din ZONA partajata\n");
printf("\tZONA 1 = %d \n", shm_ptr_zona->zona1);  // =10   afisez din  zona1 din shm
printf("\tZONA 2 = '%s' \n", shm_ptr_zona->zona2);  // ="ABC"   afisez din zona2 din shm
printf("\tZONA 3 = '%s' \n", shm_ptr_zona->zona3);  // ="DEF"   afisez din  zona2 din shm
exit(0);
}
```

Operare

```
$ gcc -o ipc_shm_zone ipc_shm_zone.c
$ ./ipc_shm_zone
_POSIX_C_SOURCE= 200809


P1 (PID= 20699)
   Proces P1 - cheie SHM generata (key) = 0x0b140045
   Proces P1 - identificator SHM (shmID) = 2064396
   Numar procese atasate= 0 (urmeaza atasarea)

Atasare ZONA la shmID= 2064396
Adresa ZONA  (shm_ptr_zona) = f7785000

Afisarea  datelor din structura de control (o parte)
   Dim.seg.mem 36 octeti
   PID creator= 3129
   Numar procese atasate= 1
   Cheia seg.mem= 0x0b140045
   Perm acces= 666

Afisarea  datelor din ZONA partajata
   ZONA 1 = 10
   ZONA 2 = 'ABC'
   ZONA 3 = 'DEF'

$ ipcs -m

------ Shared Memory Segments --------
key            shmid      owner      perms      bytes      nattch     status
..... etc ...
0x00000000     557061     knoppix    600        393216     2          dest
0x00000000     589830     knoppix    600        524288      2         dest
0x0b140045     2064396    knoppix    666        36         0
... etc ...
```

```
$
```

- **P2- Program ipc_shm_cit_zone.c**

```
/* ipc_shm_cit_zone.c */
    /*
     Procesul citește datele din segmentul de memorie partajată creat de P1, le afișează, șterge segmentul de
    memorie creat de P1 și se termină
    */

    // Linux Microknoppix 4.12.7-64
    // _POSIX_C_SOURCE= 200809
    #include <stdio.h>
    #include <stdlib.h>
    #include <errno.h>
    #include <unistd.h>
    //
    #include <sys/types.h>
    //
    #include <sys/ipc.h>
    #include <sys/shm.h>

    // calcul KEY memorie partajata- rezultatul pe 32 biti
    #define FISIER "/etc/passwd"    // din 16 biti din numarul de i-node al fisierului
    #define ID_PROIECT 11           //     si din ultimii 8 biti a lui 11


    int main (void)
    {
     key_t shm_key;                  // Cheie de identificare segment  memorie partajata
     int shm_id;                     // Identificator segment memorie

     /** structura folosita de memoria partajata din cadrul segmentului **/
    struct mem_partajata_zona {
        int zona1;                  // 4 octeti
       char zona2[10];              // 10 octeti
        char zona3[20];             // 20 octeti
    };

     struct mem_partajata_zona *shm_ptr_zona;   // pointer zona Memorie partajata

     struct shmid_ds shmid_ds, *shm_id_ds;  // Structura de control segment memorie partajata
     shm_id_ds = & shmid_ds;                //       punctata de shm_id_ds

     printf("_POSIX_C_SOURCE= %ld\n\n",_POSIX_C_SOURCE);
     printf("\nProces P2 (PID= %ld)\n", (long)getpid());
     shm_key = ftok (FISIER, ID_PROIECT);           // generare shm_key
     printf("\tProces P2 - cheie SHM generata (key) = 0x%08lx\n", shm_key);
     // utilizarea unui shm existent
     // controleaza daca shm_key exista
     if ((shm_id = shmget ((key_t) shm_key, 0, IPC_CREAT)) == -1)
     { // non EEXIST-- eroare
      perror("err. shmget() - probabil nu exista shm\n");
      exit (EXIT_FAILURE);
     }

     printf("\tProces P2 - identificator SHM (shmID) = %ld\n", shm_id);
     shmctl(shm_id, IPC_STAT , shm_id_ds);
     printf ("\tNumar procese atasate= %u (urmeaza atasarea)\n",shm_id_ds->shm_nattch);
```

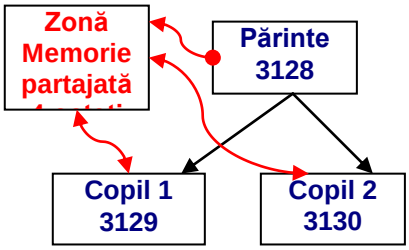
```
        // Ataseaza la segmentul de memorie partajata identificat prin
        //   shm_id la spatiul de adrese (zona partajata) al procesului apelant.

        shm_ptr_zona = (struct mem_partajata_zona *) shmat (shm_id, NULL, 0);
        if ((void *) shm_ptr_zona == (void *) -1)
        {
         perror ("err. shmat() - esuare atasare zona");
         exit (EXIT_FAILURE);
        };
        printf("\nAdresa ZONA (shm_ptr_zona) = %x\n", shm_ptr_zona);

        // Citirea din structura de controlul al segmentului de
        //   memorie partajat巢
         shmctl(shm_id, IPC_STAT , shm_id_ds);

        // Afisarea  datelor din structura de control
        //   segment memorie partajata (o parte din ele)

        printf ("\nAfisarea  datelor din structura de control (o parte)\n");
        printf ("\tDim.seg.mem %d octeti\n",shm_id_ds->shm_segsz);
        printf ("\tPID creator= %u\n",shm_id_ds->shm_cpid);
        printf ("\tNumar procese atasate= %u\n",shm_id_ds->shm_nattch);
        printf ("\tCheia seg.mem= 0x%08lx\n",shm_id_ds->shm_perm.__key);
        printf ("\tPerm acces= %o\n",shm_id_ds->shm_perm.mode);

        printf ("\nAfisarea  datelor din ZONA partajata\n");
        printf("\tZONA 1 = %d \n", shm_ptr_zona->zona1);  // =10   afisez din  in zona1 din segmentul de memorie
      partajata
        printf("\tZONA 2 = '%s' \n", shm_ptr_zona->zona2);  // ="ABC"   afisez din  in zona2 din segmentul de
      memorie partajata
        printf("\tZONA 3 = '%s' \n", shm_ptr_zona->zona3);  // ="DEF"   afisez din  in zona2 din segmentul de
      memorie partajata

          shmctl (shm_id, IPC_RMID, NULL);   // executa operatia de control IPC_RMID
                                             // (marcheaza segmentul de memorie partajata
                                             //                  pentru a fi distrus)

          exit(0);

        }
```

Operare

```
$ gcc -o ipc_shm_cit_zone ipc_shm_cit_zone.c
$ ./ipc_shm_cit_zone
_POSIX_C_SOURCE= 200809


Proces P2 (PID= 27610)
   Proces P2 - cheie SHM generata (key) = 0x0b140045
   Proces P2 - identificator SHM (shmID) = 2064396
   Numar procese atasate= 0 (urmeaza atasarea)

Adresa ZONA (shm_ptr_zona) = f76de000

Afisarea  datelor din structura de control (o parte)
   Dim.seg.mem 36 octeti
   PID creator= 3129
   Numar procese atasate= 1
   Cheia seg.mem= 0x0b140045
   Perm acces= 666
```

```
Afisarea  datelor din ZONA partajata
   ZONA 1 = 10
   ZONA 2 = 'ABC'
   ZONA 3 = 'DEF'
$
```

3. Crearea unei structuri de procese în pieptene, formată dintr-un proces părinte și două procese copil care comunică între ele prin intermediul unui segment de memorie partajată. Procesul părinte așteaptă terminarea proceselor copil. Se pune în evidență necesitatea sincronizării.



- **Program ipc_shm.c**

```
/**     ipc_shm.c   **/
        // Linux Microknoppix 4.12.7-64
        // _POSIX_C_SOURCE= 200809
         /* IPC - Tehnica de comunicare intre procese   **
         ** program scris in scop didactic              **
         ** autor: Virgiliu Streian                     **
         ** exemplu de cod : comunicare prin memorie partajata        **
         ** Programul creaza un pieptene doua procese copil C1 si C2  **
         ** care comunica intre ele printr-o zona  memorie partajata. **
         ** Segmentul de memorie partajata este creat in procesul parinte   **
         ** Ceea ce trebuie remarcat este faptul ca o zona de memorie partajata nu poseda
         **    niciun mecanism de sincronizare.
         **    Este responsabilitatea programatorului sa asigure accesul exclusiv la memorie.
         **    Pentru sincronizare se pot folosi semafoare, variabile mutex dar si mecanisme de lock pe fisiere
        (fcntl).

         compilare:
           $gcc -o ipc_shm ipc_shm.c
         executie:
           $./ipc_shm
        Alte informatii:
           $ipcs -lm
            ------ Shared Memory Limits --------
            max number of segments = 4096
            max seg size (kbytes) = 32768
            max total shared memory (kbytes) = 8388608
            min seg size (bytes) = 1

        */

        #include <stdio.h>
        #include <stdlib.h>
```

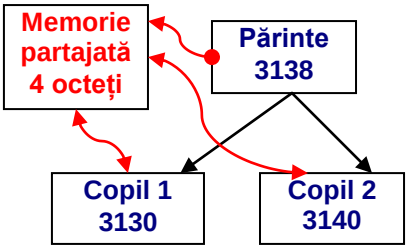
```
#include <unistd.h>
//
#include <sys/types.h>
#include <sys/wait.h>
//
#include <sys/ipc.h>
#include <sys/shm.h>

// date pentru calcul KEY memorie partajata- rezultatul pe 32 biti
#define FISIER "."    // din 16 biti din numarul de i-node   a dir. curent
#define ID_PROIECT 11 // si din ultimii 8 biti a lui 11
                      // (poate fi orice val. chiar si cod litera)

/* structura folosita la crearea segmentului de memorie partajata */
struct mem_partajata {  // 4 octeti- aici era bine de utilizat o sincronizare mutex
   int test1;
};
/*
Structura asociata cu segmentul de memorie partajata.
O data creat un segment de memorie partajata, sistemul de operare mentine informaii despre el in
cadrul unei structuri de date de tipul shmid_ds, ale careii câmpuri sunt descrise mai jos:
        struct shmid_ds {
           struct ipc_perm shm_perm;    // Ownership and permissions
           size_t         shm_segsz;   // Size of segment (bytes)
           time_t         shm_atime;   // Last attach time
           time_t         shm_dtime;   // Last detach time
           time_t         shm_ctime;   // Last change time
           pid_t          shm_cpid;    // PID of creator
           pid_t          shm_lpid;    // PID of last shmat()/shmdt()
           shmatt_t       shm_nattch;  // No. of current attaches
           ...
        };
        struct ipc_perm {
        key_t          __key;  // Key supplied to shmget()
        uid_t          uid;    // owner user ID
        gid_t          gid;    // owner group ID
        uid_t          cuid;   // creator group ID
        gid_t          cgid;   // creator user ID
        unsigned short mode    // r/w permissions+SHM_DEST and
                               SHM_LOCKED flags
        unsigned short __seq;  // Sequence number
         };
*/
/*****************************************************/
/**********************M A I N************************/
/*****************************************************/
int main (void)
{
 /* secventa proces parinte */
 key_t shm_key;              // Cheie de identificare segment memorie partajata
 int shm_id;                 // Identificator segment memorie partajata
 pid_t childpid;             // PID proces copil
 int status;
 struct mem_partajata *shm_ptr;    // zona Memorie partajata  punctata de shm_ptr
 struct shmid_ds shmid_ds, *shm_id_ds;  // Structura de control  segment memorie partajata
 shm_id_ds = & shmid_ds;               //  punctata de shm_id_ds

 printf("_POSIX_C_SOURCE= %ld\n\n",_POSIX_C_SOURCE);
 printf("\nProces PARINTE (PID= %ld) voi crea un pieptene de doua procese COPIL\n",
(long)getpid());
 printf("\tseg size (bytes) - sizeof (*shm_ptr)= %ld\n", sizeof (*shm_ptr));
```

```
// generare shm_key

if ((shm_key = ftok (FISIER, ID_PROIECT)) == -1)
{
 perror ("err. ftok\n");
 exit (EXIT_FAILURE);
}
printf("\tshm_key generat= 0x%08lx\n", shm_key);

/* alocare segment  memorie partajata SHM_KEY de sizeof() octeti
   permisiuni de acces 0666 adica RW_RW_RW_ */

if ((shm_id = shmget ((key_t) shm_key, sizeof (*shm_ptr), IPC_CREAT | 0666)) == -1)
{
 (perror("err. shmget()\n"));
 exit (EXIT_FAILURE);
}
printf("\tshm_id= %ld\n", shm_id);

/* Ataseaza segmentul de memorie partajata identificat prin
   shm_id la spatiul de adrese al procesului apelant.
   NULL semnifica faptul ca atasarea se face la  urmatoarea adresa care e libera.
   In caz de succes shmat() returneaza adresa segmentului de   memorie partajata.
   In caz de insucces shmat() returneaza (void *) -1;
*/
shm_ptr = (struct mem_partajata *) shmat (shm_id, NULL, 0);
if ((void *) shm_ptr == (void *) -1)
{
 perror ("err. shmat");
 shmctl (shm_id, IPC_RMID, NULL);   // Executa operatia de  control IPC_RMID
                                    // (marcheaza segmentul   pentru a fi distrus)
 exit (EXIT_FAILURE);
};

/* Citirea din structura de controlul al segmentului de  memorie partajat王/
 shmctl(shm_id, IPC_STAT , shm_id_ds);

/* Afisarea  datelor din structura de control segment memorie partajata (o parte din ele) */
printf ("\nAfisarea  datelor din structura de control (o parte)\n");
printf ("\tDim.seg.mem %d octeti\n",shm_id_ds->shm_segsz);
printf ("\tPID creator= %u\n",shm_id_ds->shm_cpid);
//printf ("\tPID ultim care a accesat seg.mem= %u\n",   shm_id_ds->shm_lpid);
printf ("\tNumar procese atasate= %u\n",shm_id_ds->shm_nattch);
printf ("\tCheia seg.mem= 0x%08lx\n",shm_id_ds->shm_perm.__key);
printf ("\tPerm acces= %o\n",shm_id_ds->shm_perm.mode);

shm_ptr->test1=10;                  // Scriu in segmentul de memorie partajata

childpid= fork();                   // Crearea proces Copil 1
if (childpid == -1)
{
 /* eroare */
 perror("fork creare Copil1 esuat");
 shmdt (shm_ptr);                   // elibereaza segmentul de memorie partajata punctat de shm_ptr
 exit (EXIT_FAILURE);
}
if (childpid == 0)
{
 /* secventa proces copil 1*/
 /* Dupa fork() procesul copil mosteneste segmentele de memorie partajata */
 printf("\nAcesta este procesul COPIL 1 (PID= %ld) cu PID parinte= %ld\n", (long)getpid(),
(long)getppid());
```

```
   printf ("\tCopil 1 la intrare test1= %ld\n", shm_ptr->test1);  // val. asteptata = 10
  //  sleep(2);                          // [1] forteaza o intarziere pt. a arata necesitatea unei sincronizari
   shm_ptr->test1++; // se aduna 1- Scriu in segmentul de memorie partajata

   printf ("\tCopil 1 la iesire  test1= %ld\n", shm_ptr->test1);  // val. asteptata = 11
                                          // respectiv 12 daca sleep(2) activ
   exit(1);
  }
  else
  {
 /* din nou secventa proces parinte */
    childpid=fork();                      // Crearea proces Copil 2
    if (childpid == -1)
    {
     /* eroare */
     perror("fork creare Copil2 esuat");
     shmdt (shm_ptr);           // elibereaza segmentul de memorie partajata punctat de shm_ptr
     exit (EXIT_FAILURE);
    }
    if (childpid == 0)
    {
     /* secventa proces copil 2*/
     /* Dupa fork() procesul copil mosteneste segmentele de memorie partajata */
     printf("\nAcesta este procesul COPIL 2 (PID= %ld) cu PID parinte= %ld\n", (long)getpid(),
 (long)getppid());
     printf ("\tCopil 2 la intrare test1= %ld\n", shm_ptr->test1);  // val. asteptata = 11
     shm_ptr->test1++;                    // se aduna 1 - Scriu in segmentul de memorie partajata
     printf ("\tCopil 2 la iesire  test1= %ld\n", shm_ptr->test1);  // val. asteptata = 12
     exit(2);
    }
    /* din nou secventa proces parinte */

     shmctl(shm_id, IPC_STAT , shm_id_ds);              // Citirea din structura de controlul

     printf ("\n1Numar procese atasate= %u\n",shm_id_ds->shm_nattch);  // =3

     wait(&status);   // astept ca COPILUL1 sa faca   exit si afisez exit status
     printf("\nCopil  %ld terminat cu exit code: %ld\n", WEXITSTATUS(status), WEXITSTATUS(status));


     shmctl(shm_id, IPC_STAT , shm_id_ds);              // Citirea din structura de controlul

     printf ("\n2Numar procese atasate= %u\n",shm_id_ds->shm_nattch);  // =2

     wait(&status);   // astept ca COPILUL2 sa faca exit si afisez exit status

     printf("Copil  %ld terminat cu exit code: %ld\n", WEXITSTATUS(status), WEXITSTATUS(status));

     shmctl(shm_id, IPC_STAT , shm_id_ds);              // Citirea din structura de controlul

     printf ("\n3Numar procese atasate= %u\n",shm_id_ds->shm_nattch);  // =1

     printf ("\tPARINTE la iesire  test1= %ld\n", shm_ptr->test1);

     shmctl (shm_id, IPC_RMID, NULL);  // executa operatia de control IPC_RMID
                                       // (marcheaza segmentul de memorie partajata
                                       //                pentru a fi distrus)
   exit(0);
  }
}
```

Operare

```
$ gcc -o ipc_shm ipc_shm.c
$ ./ipc_shm
_POSIX_C_SOURCE= 200809
```




```
Proces PARINTE (PID= 3138) voi crea un pieptene de doua
procese COPIL
   seg size (bytes) - sizeof (*shm_ptr)= 4
   shm_key generat= 0x0b142772
   shm_id= 1966093

Afisarea  datelor din structura de control (o parte)
   Dim.seg.mem 4 octeti
   PID creator= 3138
   Numar procese atasate= 1
   Cheia seg.mem= 0x0b142772
   Perm acces= 666

1Numar procese atasate= 3

Acesta este procesul COPIL 1 (PID= 3139) cu PID parinte= 3138
   Copil 1 la intrare test1= 10
   Copil 1 la iesire  test1= 11  < ---- Aici apare 12 daca se activeaza sleep(2) din Copil 1

Copil  1 terminat cu exit code: 1


2Numar procese atasate= 2
Acesta este procesul COPIL 2 (PID= 3140) cu PID parinte= 3138
   Copil 2 la intrare test1= 11
   Copil 2 la iesire  test1= 12
Copil  2 terminat cu exit code: 2

3Numar procese atasate= 1
   PARINTE la iesire  test1= 12
$


/*
se dezcomenteaza linia [1]  din COPIL 1 ce contine un sleep(2)
si se constata necesitatea unei sincronizari inter-proces.
*/
$ gcc -o ipc_shm ipc_shm.c
$ ./ipc_shm
_POSIX_C_SOURCE= 200809


Proces PARINTE (PID= 3153) voi crea un pieptene de doua procese COPIL
   seg size (bytes) - sizeof (*shm_ptr)= 4
   shm_key generat= 0x0b142772
   shm_id= 2195467

Afisarea  datelor din structura de control (o parte)
   Dim.seg.mem 4 octeti
   PID creator= 3153
   Numar procese atasate= 1
   Cheia seg.mem= 0x0b142772
   Perm acces= 666

1Numar procese atasate= 3
```

```
Acesta este procesul COPIL 1 (PID= 3154) cu PID parinte= 3153
   Copil 1 la intrare test1= 10

Acesta este procesul COPIL 2 (PID= 3155) cu PID parinte= 3153
   Copil 2 la intrare test1= 10
   Copil 2 la iesire  test1= 11

Copil  2 terminat cu exit code: 2

2Numar procese atasate= 2
   Copil 1 la iesire  test1= 12< ---- Aici trebuia sa apara 11 - (cu sleep(2) din Copil 1 activa)
Copil  1 terminat cu exit code: 1

3Numar procese atasate= 1
   PARINTE la iesire  test1= 12
```