Sisteme de operare 1

Curs 7

Titular curs,

Dr. Dragoș Sanda Maria

Procese Unix

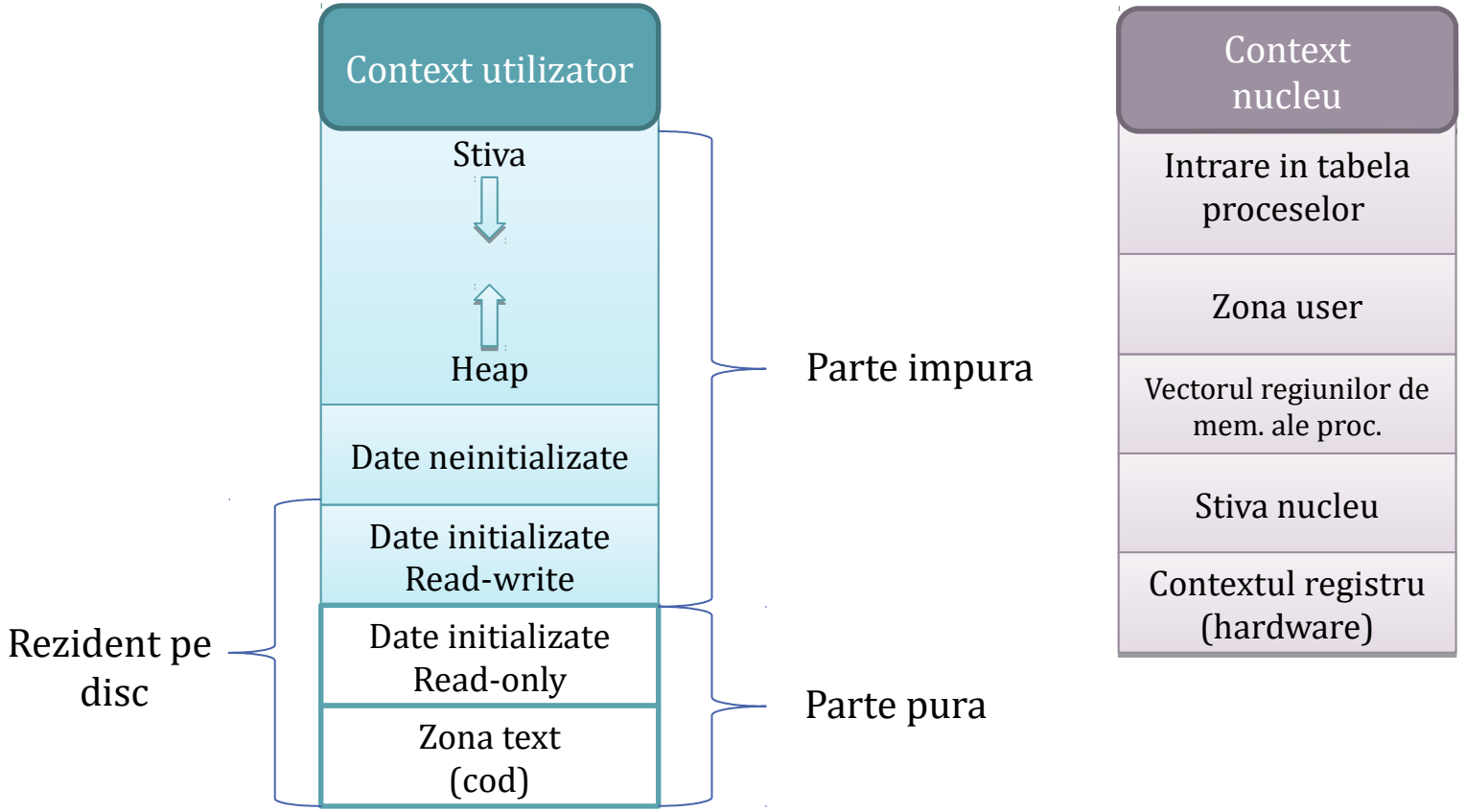
- **Proces =** un program in executie care foloseste un set de resurse ale sistemului (memorie, procesor, disc, interfata de retea, etc)

Imaginea unui proces in memorie

- **Context utilizator** – portiunea spatiului de adrese accesibile in timpul executiei de catre procesul respectiv in mod user.

- **Context nucleu** – intretinut de nucleu si accesibil doar prin apeluri sistem specifice

Imaginea unui proces in memorie



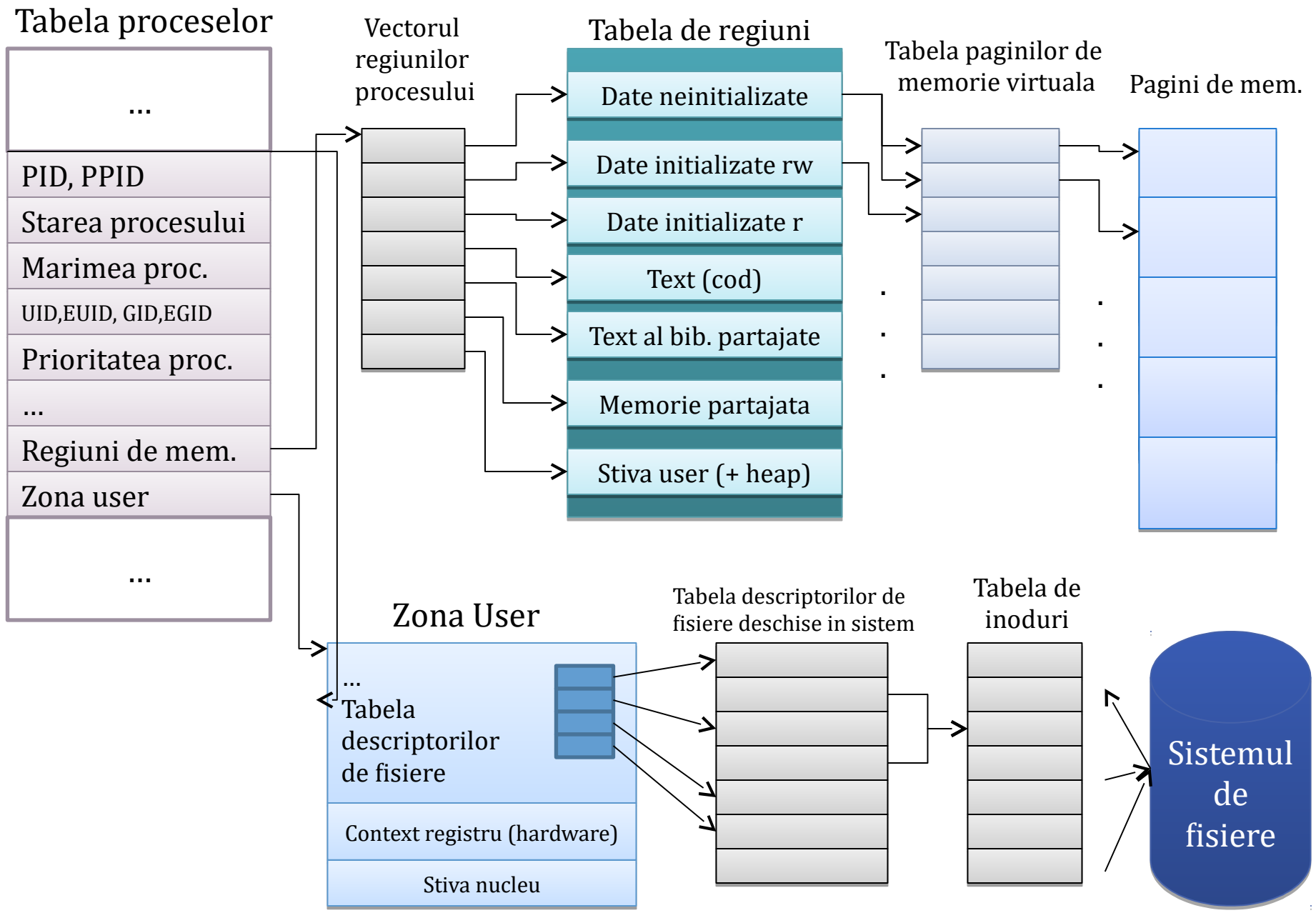


Tabela proceselor
...
PID, PPID
Starea procesului
Marimea proc.
UID,EUID, GID,EGID
Prioritatea proc.
...
Regiuni de mem.
Zona user
...
Vectorul regiunilor procesului
Tabela de regiuni
Date neinitializate
Date initializate rw
Date initializate r
Text (cod)
Text al bib. partajate
Memorie partajata
Stiva user (+ heap)
Tabela paginilor de memorie virtuala
Pagini de mem.
Zona User
...
Tabela descriptorilor de fisiere
Context registru (hardware)
Stiva nucleu
Tabela descriptorilor de fisiere deschise in sistem
Tabela de inoduri
Sistemul de fisiere

Intrarea in tabela proceselor

- starea procesului
- pointer la zona User si la Tabela regiunilor de memorie ale proceselor
- dimensiunea procesului in memorie
- PID, PPID
- UID, EUID, GID, EGID
- prioritatea procesului (numar intre 1 si 39)
- semnalele trimise procesului
- statistici de timp (ex: folosirea procesorului)
- statusul memoriei procesului (daca imaginea procesului se afla in memoria principala sau in memoria swap)
- pointer la urmatorul proces din coada de procese in starea READY
- descriptori de evenimente care au avut loc cat timp procesul a fost in starea SLEEP

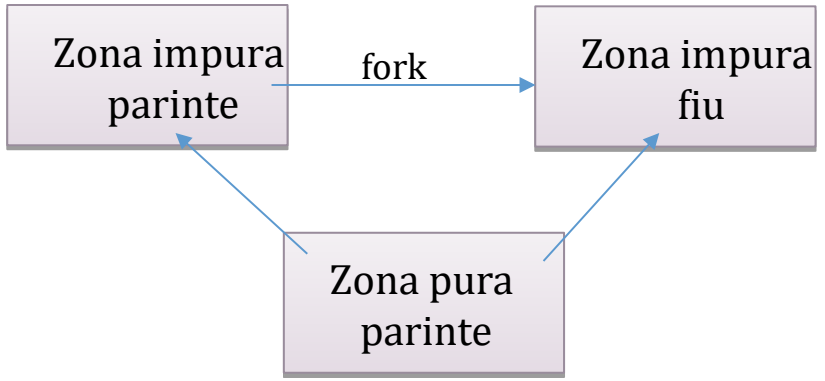
Zona User

- pointer la intrarea in tabela proceselor care corespunde acestei zone user
- UID, EUID, GID, EGID - folosite la determinarea drepturilor de acces ale procesului
- timpul cat acest proces a fost in mod de executie user si cat a fost in mod de executie nucleu
- vectorul de actiunilor de tratarea a semnalelor
- terminalul de control al procesului - cel de la care a fost lansat
- valorile returnate si posibile erori in urma efectuarii unor apeluri sistem
- directorul curent si directorul radacina
- zona variabilelor de mediu
- posibile restrictii impuse de nucleu procesului (ex: dim. procesului in mem.)
- tabela descriptorilor de fisiere (date despre toate fisierele deschise de proces)
- umask - masca drepturilor de acces pentru fisierele nou create de catre acest proces

Crearea unui proces. Apelul sistem *fork*

#include <unistd.h>

pid_t fork();

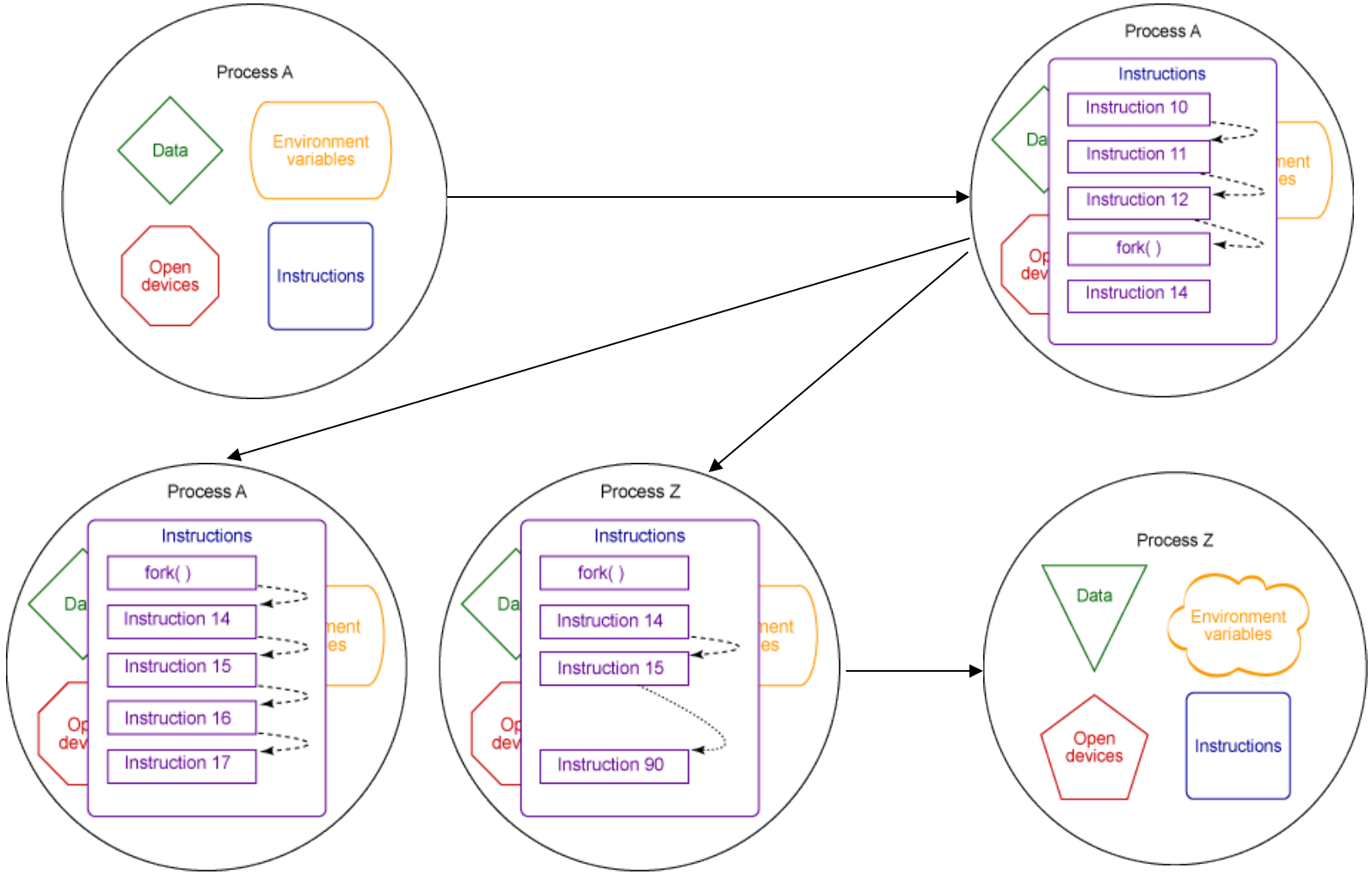


Returneaza:

- -1, daca operatia nu s-a putut efectua (eroare)
- 0, in codul fiului
- pid, in codul parintelui, unde pid este identificatorul de proces al fiului nou-creat.

man fork

Crearea unui proces



Folosire fork - propuneri

```
...
if( ( pid=fork() ) < 0) {
        perror("Eroare");
        exit(1);
}
if(pid==0) {
        /* codul fiului */
        ...
        exit(0)
}
/* codul parintelui */
...
wait(&status) ;
```

```
if (fork() == 0)
{       /* codul fiului */          }
else
{       /* codul parintelui */      }
```

```
...
pid=fork();
switch(pid) {
  case -1:
        fprintf(stderr,"ERROR: %s\n", sys_errlist[errno]);
        exit(1);
        break;
  case 0:
        /* codul fiului */
        break;
  default:
        /* codul parintelui */
        break;
}
...
```

Executia unui program extern. Apelurile sistem *exec**

```
#include <unistd.h>

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

man 3 exec

Apelurile sistem *wait()* si *waitpid()*

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

#include <sys/wait.h>


pid_t wait(int *status);

pid_t waitpid(pid_t pid, int *status, int options);
```

man 2 wait

Alte functii pentru lucrul cu procese

#include <sys/types.h>

#include <unistd.h>

pid_t getpid(); - returneaza PID-ul procesului curent

pid_t getppid(); - returneaza PID-ul parintelui procesului curent

uid_t getuid(); - returneaza identificatorul utilizatorului care a lansat procesul curent

gid_t getgid(); - returneaza identificatorul grupului utilizatorului care a lansat procesul curent

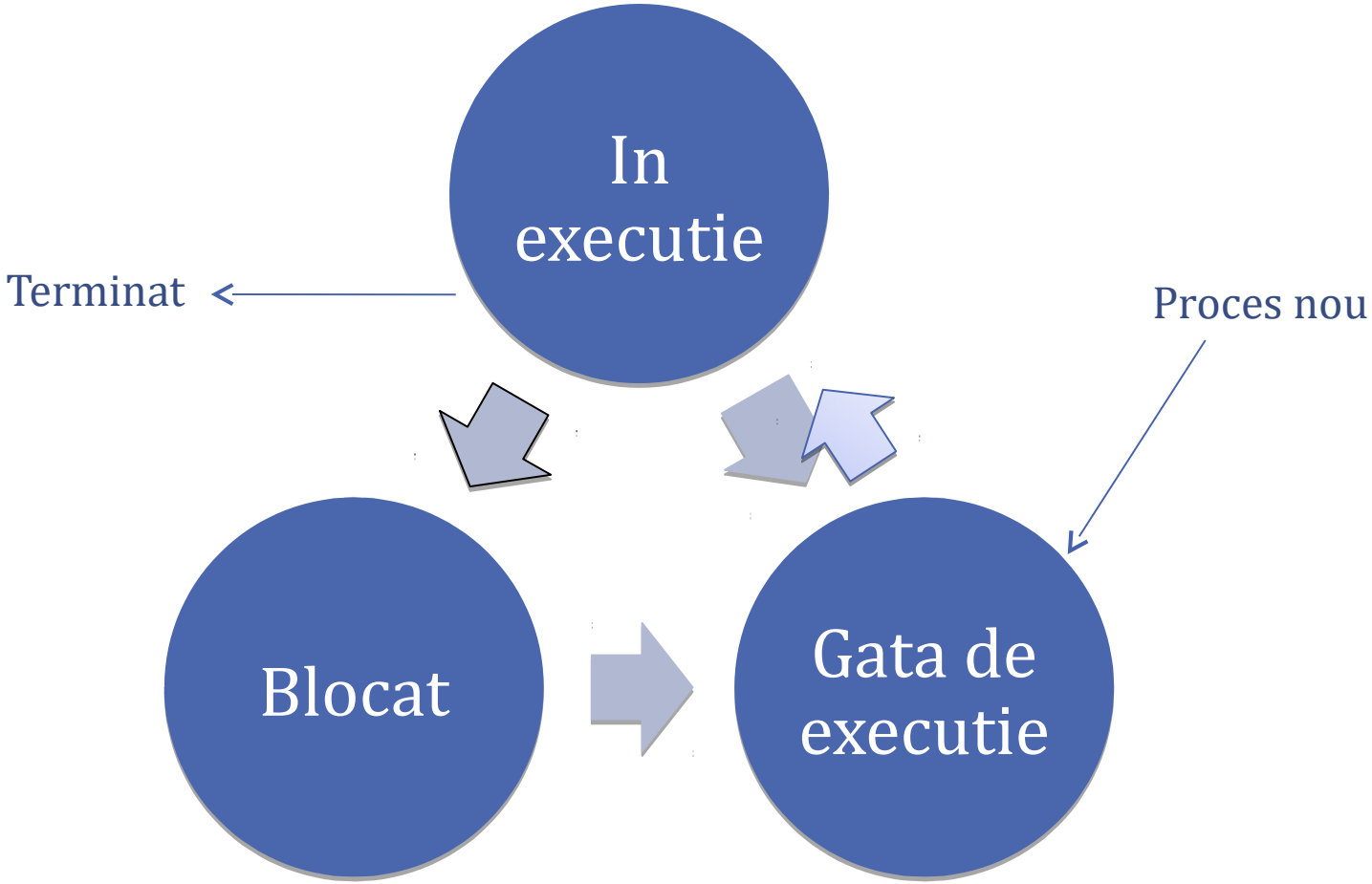
man getpid; man getuid; man getgid

Exercitiu

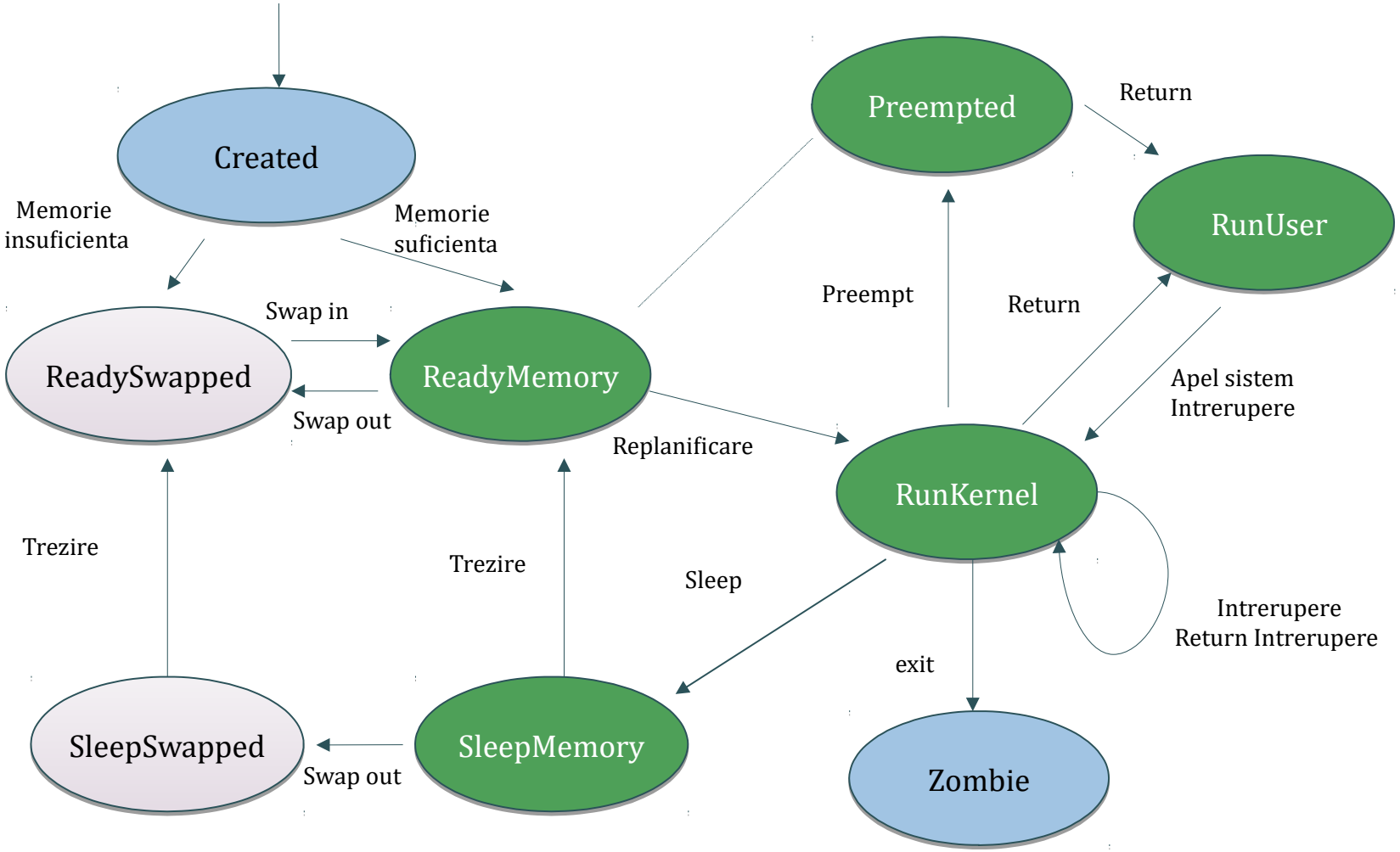
Explicaţi efectul urmatoarei secvenţe de cod:

```
int i;
for (i = 1; i <= 10; i++)    fork();
```

Starile unui proces



Starile unui proces



Starile unui proces

- **Created** – procesul este creat prin intermediul unui apel sistem *fork*
- **ReadyMemory** – procesul ocupa loc in memoria RAM si este gata pentru a fi executat; nucleul decide momentul intrarii lui in executie.
- **ReadySwapped** – procesul este gata pentru a fi executat; pentru a intra in ciclul executiei trebuie sa intre mai intain in starea **ReadyMemory**
- **RunKernel** – procesului ii sunt executate instructiuni in mod nucleu (ex: apeluri sistem, handler de intreruperi)
- **RunUser –** sunt executate instructiunile programului

Starile unui proces (cont.)

- **Preemted** – aproape identica cu **ReadyMemory**. Dupa executia unui apel sistem procesul trebuie trecut in starea **RunUser** sau in starea **ReadyMemory**. Daca in starea **ReadyMemory** exista un proces mai prioritar, procesul curent este trecut in starea **Preemted** iar procesul mai prioritar trece in starea **RunUser**, altfel procesul curent trece in starea **RunUser**. Un proces aflat in starea **Preemted** se muta in starea **RunUser** cand este cel mai prioritar.

- **SleepMemory** – procesul se afla in memorie dar nu poate fi executat pana se produce un eveniment care sa-l scoata din aceasta stare

- **SleepSwapped** - procesul este evacuat pe disc, el nu poate fi executat pana se produce un eveniment care sa-l scoata din acasta stare

- **Zombie** – Procesul nu mai exista insa informarea faptului ca s-a terminat, trimisa spre procesul parinte, nu a fost inca preluta de acesta