Lucrarea de laborator Nr. 07

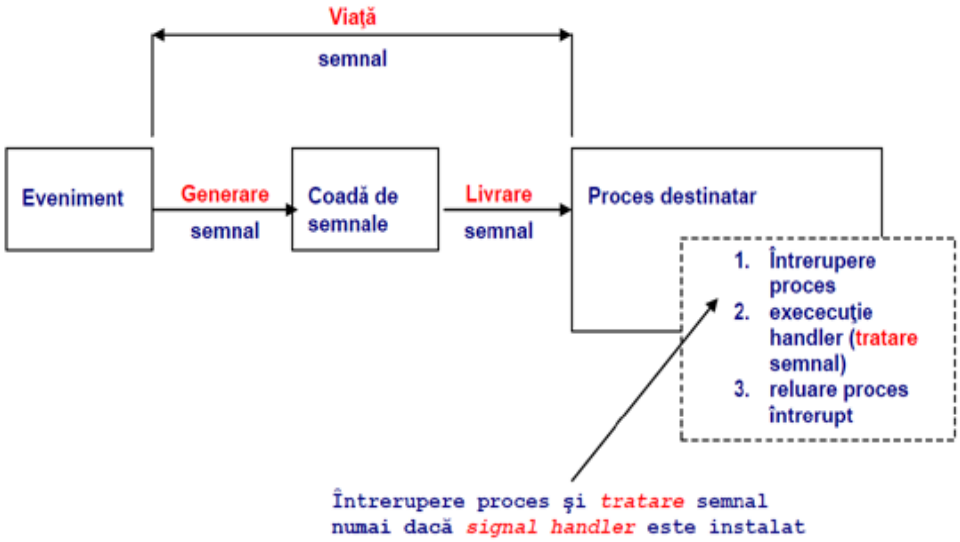
Mecanisme de comunicare între procese: Semnale. Gestiunea semnalelor.

Aplicații demonstrative:

1. Trimitere de semnal din shell către un proces (c-da kill)
2. Funcțiile kill() și raise() de trimitere semnal. Funcția signal() de definire tratare semnal

- **Context**

Procesele dialoghează între ele cu ajutorul s*emnalelor*; *semnale* ce sunt caracterizate prin *număr* şi *nume simbolic* (*i, n*). Numele de semnale sunt definite în `signal.h`. Lista semnalelor poate fi obţinută consultând fişierul *signal.h (man 7 signal* sau *kill -l).*
Semnalele Unix reprezintă un mecanism fundamental de comunicare între procese Unix. Un **semnal** este o notificare software transmisa unui anumit proces. Notificarea este declanşată de apariţia unui eveniment. Un semnal este ***generat*** **(generated)** atunci când evenimentul cauzează apariţia respectivului semnal. Un semnal este ***livrat*** (**delivered)** atunci când procesul execută acţiuni bazate pe semnalul respectiv. ***Viaţa*** **(lifetime)** a unui ***semnal*** reprezintă intervalul de timp dintre *generarea* şi *livrarea* sa. Un semnal care a fost *generat*, dar nu a fost încă *livrat* este ***în asteptare*** (**pending)**. Atunci când procesul este oprit, semnalele marcate ca *pending (în aşteptare)* vor fi trimise când procesul îşi continuă rularea. La *livrare* un proces ***tratează*** **(catches)** un semnal numai dacă este prevăzută o ***secvenţă de tratare semnal*** (**signal handler)**. Un program instalează un *signal handler* prin utilizarea apelului sistem ***signal()*** sau ***sigaction()***, iar utilizatorul scrie o secvenţă specifică de tratare semnal.



Proces destinatar

……..
……..Întrerupere ---------> Tratare semnal
……..<---------- ……
…….. | ……..
…….. |<---------- revenire în proces

Cauzele generării unui semnal pot fi grupate astfel:

- Evenimente generate de harware: execuţia unei instrucţiuni ilegale, căderea curentului, defectare memorie, etc
- Evenimente generate de sistemul de operare: încercarea de a accesa o zonă de memorie nepermisă, memorie insuficientă, timer expirat, etc.

- **Evenimente generate de procese utilizator sau de utilizatorul însuşi**: ctrl/z respectiv ctrl/c (suspendare rerspectiv terminare proces), generare de semnal de către proces prin apelul funcţiei *kill()* sau a comenzii *kill* din shell, etc .

Un semnal nu furnizează informaţii suplimentare despre evenimentul care l-a produs. Semnalele pot fi transmise:

a. de către kernel unui proces
b. de către un proces altui proces
c. de către un proces lui însuşi
d. de către un utilizator unui proces (comanda kill)
e. Procesul care a recepţionat un semnal nu cunoaşte cine anume i-a transmis acel semnal.

- **Comenzi de referință**

Comanda *kill*

Trimite un semnal către un *proces*. *Procesele* dialoghează între ele cu ajutorul s*emnalelor*; aceste *semnale* sunt caracterizate prin *număr* şi *nume* (*i, n*). Lista *(i, n)* ale *semnalelor* poate fi obţinută cu ajutorul comenzii ***kill***, cu opţiunea ***-l***

$kill -l ↤⇨ listă de forma <n>;<i>

```
1) SIGHUP       2) SIGINT       3) SIGQUIT       4) SIGILL
5) SIGTRAP      6) SIGABRT      7) SIGBUS        8) SIGFPE
9) SIGKILL      10) SIGUSR1     11) SIGSEGV      12) SIGUSR2
13) SIGPIPE 14) SIGALRM     15) SIGTERM      16) SIGSTKFLT
...etc...
```

De remarcat în listă semnalele :

(1, SIGHUP)	*background*
(2, SIGINT)	<ctrl/c>
(3, SIGQUIT)	<ctrl/\>
(9, SIGKILL)	*hard kill*
(15, SIGTERM)	*soft kill*

Sintaxă:
kill signal pid(s)

unde:
- *signal*: este semnalul pe care *procesul* <*pid*> trebuie să-l analizeze; de exemplu:
  - *15* semnifică un *soft kill* (terminare întârziată *proces:*se mai fac nişte închideri de fişiere, etc.);
  - *9* semnifică un *hard kill* (terminare imediată *proces*);
- *pid(s)*: reprezintă *numărul de identificare proces* (*PID*)

- **Apeluri de referință**

Funcţia *kill()-* trimite un semnal unui proces.

Prin apelul funcţiei sistem *kill()* din interiorul unui program C, se poate trimite un semnal către un proces. Funcţia *kill()* are ca şi parametri PID-ul procesului ţintă şi numărul/numele de semnal ce se emite:

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

```
int kill(pid_t pid, int sig);
```

Dacă **pid**

> 0 atunci kill() emite semnalul specificat sig către procesul pid.
< 0 atunci kill() emite semnalul specificat sig către grupul de procese group ID egal cu |pid|.
= 0 atunci kill() emite semnalul specificat sig către grupul de procese din care face parte procesul apelant.

Funcţia *raise()-* autotrimite un semnal procesului apelant.

Pin apelul funcţiei sistem *raise()* un proces poate să emită un semnal către el însuşi. Această funcţie are un singur argument şi anume sig - numărul de semnal ce se emite.

```
#include <signal.h>

int raise(int sig);
```

Funcţia *signal()-* instalează un *signal handler* şi este utilizată pentru specificarea tratării la livrarea semnalelor; are următorul prototip:

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

sighandler_t signal (int id-signal, sighandler_t action);
```

returnează vechiul *handler signal* pentru semnalul specificat, care astfel poate fi apoi restaurat daca este nevoie. În caz de eşec funcţia returnează valoarea SIG_ERR.
id-signal este numărul/numele semnalului care startează **action**.
action este numele funcţiei ce se startează la livrarea semnalului **id-signal**. Argumentul **action** poate fi numele unei funcţii definite de utilizator, sau poate lua una dintre următoarele valori (constante simbolice):

- **SIG_DFL**: specifică acţiunea implicită (cea stabilită de către sistemul de operare) la recepţionarea semnalului.
- **SIG_IGN**: specifică faptul ca procesul va ignora acel semnal. Sistemul de operare nu permite să se ignore sau să se livreze către procese utilizator semnalele **SIGKILL** şi **SIGSTOP**, de aceea funcţia signal() va returna o eroare dacă se face o asemenea încercare.

Aplicații demonstrative:

1. **Trimitere de semnal din shell către un proces (c-da kill)**

Exemplu de trimitere de semnal din shell către un proces:

a. *$kill -l ↵* ⇒ *listă de forma <n>;<i>*

```
1) SIGHUP       2) SIGINT       3) SIGQUIT       4) SIGILL
5) SIGTRAP      6) SIGABRT      7) SIGBUS        8) SIGFPE
9) SIGKILL      10) SIGUSR1     11) SIGSEGV      12) SIGUSR2
13) SIGPIPE 14) SIGALRM     15) SIGTERM      16) SIGSTKFLT
...etc...
```

De remarcat în listă semnalele :

(1, SIGHUP)	*background*
(2, SIGINT)	<ctrl/c>
(3, SIGQUIT)	<ctrl/\>

(9, SIGKILL) *hard kill*
(15, SIGTERM) *soft kill*

b. terminarea (distrugerea) *hard kill* a *procesului shell*; în urma acestui lucru *utilizatorul* va fi scos din *sesiunea de lucru* (se forţează un *logout*):

$ sleep 100**&** ⇒ crearea unui proces de lungă durată și startarea lui în background (**&**)
`[1] 8177` ⇒ `[1]` este JobID, iar `8177` este PID
$ ps –f

```
UID        PID  PPID  C STIME TTY          TIME CMD
streian   8119  8117  0 Dec04 pts/0    00:00:00 -bash
streian   8177  8119  0 00:17 pts/0    00:00:00 sleep 100
streian   8178  8119  0 00:18 pts/0    00:00:00 ps -f
```

$ kill -9 **8177** ⇒ *trimitere semnal* ***SIGKILL către PID 8177***
⇒ *se putea da și așa: kill* ***SIGKILL 8177***

```
[1]+  Killed                  sleep 100
```

$ sleep 100&

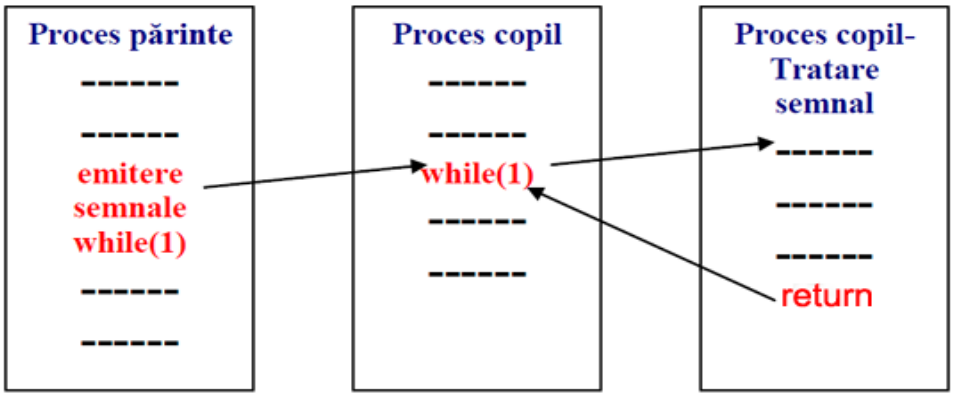
```
[1] 8178
```

$ kill -9 **%1** ⇒*trimitere semnal* ***SIGKILL către JobID 1***

```
[1]+  Killed                  sleep 100
```

2. Funcțiile kill() și raise() de trimitere semnal. Funcția signal() de definire tratare semnal

Program *semnale.c* - Demonstreaza utilizarea functiei *signal(), kill()* şi *raise().*



Notă

SIGHUP **hang up on controlling terminal**
acest semnal este generat de către sistemul de operare atunci când procesul este deconectat de la terminal (de exemplu atunci când lăsăm un proces pentru execuţie în background şi apoi facem logout; întrucât acţiunea implicită pentru SIGHUP este de terminare a procesului, dacă vrem ca procesul să supravieţuiască trebuie să blocam acest semnal, folosind utilitarul nohup

SIGINT **interactive attention signal**
program întrerupt; acest semnal este trimis procesului atunci când utilizatorul foloseşte CTRL-C

SIGQUIT **interactive termination**
similar cu SIGINT doar că este folosită combinaţia de taste CTRL-\ şi se generează şi un core dump

SIGKILL **terminate (cannot be caught or ignored)**
hard kill - procesul este imediat terminat; **acest semnal NU poate fi tratat, ignorat sau blocat de către proces**

SIGTERM **termination**
soft kill - terminare întârziată proces: se mai fac nişte închideri de fişiere, etc

```
/* semnale.c */
 #include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

/* functii tratare semnale*/
int sigint();
int sighup();
int sigquit();
int sigterm();
int sigkill();
char c;

int main() {
  pid_t pid;

  if((pid = fork())<0) { // creare proces copil
    perror("fork esuat");
    exit(EXIT_FAILURE);
  }

  if(!pid) {
    // proces copil
    fprintf(stderr, "\n\tCopil: PID=%ld  PPID=%ld \n",(pid_t)getpid(), (pid_t)getppid());
   /* se creaza signal handler*/
   /* se armeaza/valideaza secventele de tratare a intreruperilor */
    signal(SIGHUP,(void (*))(int)sighup);    /* hang up controlling terminal */
    signal(SIGINT,(void (*))(int)sigint);      // ctrl-c    /* interactive attention signal - ctrl-c */
    signal(SIGQUIT,(void (*))(int)sigquit);   // ctrl-\    /* interactive termination - ctrl-\ */
    signal(SIGTERM,(void (*))(int)sigterm);       /* termination soft kill - terminare întârziată proces */
    signal(SIGKILL,(void (*))(int)sigkill);    /* terminate (hard kill) */

    fprintf(stderr, "\n\tCopil: PID=%ld  PPID=%ld intru in ciclu infinit\n",(pid_t)getpid(), (pid_t)getppid());
    /* bucla infinita se iese prin semnal SIGINT la nivel de parinte, care se autodistruge incluzand si copilul */
    while(1); // busy waiting/ ciclu infinit- se iese prin SIGINT emis de parinte

  } else {
      // proces parinte
     fprintf(stderr, "Parinte: PID=%ld  PPID=%ld  Copil creat =%ld\n",(pid_t)getpid(), (pid_t)getppid(), pid);

/* in parinte se emit semnale catre copil*/
     sleep(1);// intarziere pentru a da timp handler-elor din copil sa se instaleze
   //SIGINT
      fprintf(stderr, "Parinte: PID=%ld trimit un SIGINT (ctrl-c) catre copilul %ld\n",(pid_t)getpid(), pid);
      if (kill(pid,SIGINT)<0) perror("Er. kill SIGINT");
      sleep(1); // intarziere in lipsa unei sincronizari
   //SIGHUP
     fprintf(stderr, "Parinte: PID=%ld trimit un SIGHUP catre copilul %ld\n",(pid_t)getpid(),pid);
     if (kill(pid,SIGHUP)<0) perror("Er. kill SIGHUP");
     sleep(1); // intarziere in lipsa unei sincronizari
    //SIGQUIT
     fprintf(stderr, "Parinte: PID=%ld trimit un SIGQUIT  (ctrl-\\)catre copilul %ld\n",(pid_t)getpid(),pid);
     if (kill(pid,SIGQUIT)<0) perror("Er. kill SIGQUIT");
    sleep(1); // intarziere in lipsa unei sincronizari
    //SIGTERM
     fprintf(stderr, "Parinte: PID=%ld trimit un SIGTERM  catre copilul %ld\n",(pid_t)getpid(),pid);
     if (kill(pid,SIGTERM)<0) perror("Er. kill SIGTERM");
    sleep(1); // intarziere in lipsa unei sincronizari
    //SIGKILL
     fprintf(stderr, "Parinte: PID=%ld trimit un SIGKILL  catre copilul %ld\n",(pid_t)getpid(),pid);
     if (kill(pid,SIGKILL)<0) perror("Er. kill SIGKILL");
```

```
    perror("Parinte: am emis un SIGKILL cu...");
    sleep(1); // intarziere in lipsa unei sincronizari
     fprintf(stderr, "Parinte: PID=%ld un SIGKILL  nu este procesat de catre copilul %ld\n",(pid_t)getpid(),pid);
   //SIGINT
     fprintf(stderr, "Parinte: PID=%ld  imi trimit un SIGINT (ctrl-c) si ma termin\n",(pid_t)getpid());
     /* interactive attention signal - ctrl-c */
     signal(SIGINT,SIG_DFL);       /* SIG_DFL = ctrl-c sa fie tratat de sistem
                                   Procesul părinte nu are instalat niciun signal handler propriu, deci va acționa
                                   handler-ul sistemului de operare--> acest apel nu are efect, decat didactic*/
    raise(SIGINT);                 // procesul parinte isi autotrimite un SIGINT
    perror("raise SIGINT");
    fprintf(stderr, "Parinte: PID=%ld  aici nu mai ajung/ mi-am dat un ctrl/c \n",(pid_t)getpid());
   /* bucla infinita se iese prin semnal SIGINT la nivel de parinte */
     while(1); // busy waiting/ ciclu infinit- se iese prin SGINT emis mai sus
  }
} // end main

/* se trateaza semnale*/

 /* Functia se executa dupa receptia semnalului SIGINT. */
int sigint() {
  signal(SIGINT,(void (*))(int)sigint);                  // rearmeza tratarea SIGINT
  fprintf(stderr, "\n\tCopil %ld: Am primit un SIGINT adica CTRL-C\n\n",(pid_t)getpid());
  return 0;
}
/* * Functia se executa dupa receptia semnalului  SIGHUP. */
int sighup() {
signal(SIGHUP,(void (*))(int)sighup);
fprintf(stderr,"\n\tCopil %ld: Am primit un SIGHUP\n\n",(pid_t)getpid());
fflush(stderr);   // golire fortata bufer
  return 0;
}

/* * Functia se executa dupa receptia semnalului  SIGQUIT. */
int sigquit() {
  signal(SIGQUIT,(void (*))(int)sigquit);
  fprintf(stderr, "\n\tCopil %ld: Am primit un SIGQUIT adica CTRL-\\\n\n",(pid_t)getpid());
  fflush(stderr);   // golire fortata bufer
   return 0;
}

/* * Functia se executa dupa receptia semnalului  SIGTERM.
   termination soft kill - terminare întârziată proces */
int sigterm() {
  signal(SIGTERM,(void (*))(int)sigterm);
  fprintf(stderr, "\n\tCopil %ld: Am primit un SIGTERM\n\n",(pid_t)getpid());
  fflush(stderr);   // golire fortata bufer
  return 0;
}

/* * Functia se executa dupa receptia semnalului SIGKILL.
 * Demonstreaza ca semnalul SIGKILL nu se livreaza. */
int sigkill() {
  signal(SIGKILL,(void (*))(int)sigkill);
  fprintf(stderr,"\n\tCopil %ld: Terminare prin SIGKILL!\n\n", (pid_t)getpid());
  fflush(stderr);   // golire fortata bufer
     return 0;
  //exit(EXIT_SUCCESS);
}
```

Compilare linkeditare şi rulare program *semnale.c*

$gcc -o semnale semnale.c

$./semnale

Mesaje afişate în urma rulării

Parinte: PID=3967 PPID=3809 Copil creat =3968

Copil: PID=3968 PPID=3967

Copil: PID=3968 PPID=3967 intru in ciclu infinit
Parinte: PID=3967 trimit un SIGINT (ctrl-c) catre copilul 3968

Copil 3968: Am primit un SIGINT adica CTRL-C

Parinte: PID=3967 trimit un SIGHUP catre copilul 3968

Copil 3968: Am primit un SIGHUP

Parinte: PID=3967 trimit un SIGQUIT (ctrl-\)catre copilul 3968

Copil 3968: Am primit un SIGQUIT adica CTRL-\

Parinte: PID=3967 trimit un SIGTERM catre copilul 3968

Copil 3968: Am primit un SIGTERM

Parinte: PID=3967 trimit un SIGKILL catre copilul 3968
Parinte: am emis un SIGKILL cu...: Success
Parinte: PID=3967 un SIGKILL nu este procesat de catre copilul 3968
Parinte: PID=3967 imi trimit un SIGINT (ctrl-c) si ma termin
$

Observaţii:

1. În roşu sunt mesajele emise de procesul părinte, iar în albastru cele emise de către procesul copil.
2. Se observă că după afişarea ultimului mesaj:

Parinte: PID=3967 imi trimit un SIGINT (ctrl-c) si ma termin

apare prompterul ***….$,*** adică procesul părinte s-a autoterminat. Procesul părinte nu are instalat niciun *signal handler* propriu, deci va acționa *handler*-ul sistemului de operare.