Lucrarea de laborator Nr. 08

Semnale. Mecanisme de comunicare între procese. Mecanisme avansate de gestiune a semnalelor. Blocarea semnalelor.

Aplicație demonstrativă:

1. Ateptarea după semnal. Program care crează două procese ce așteaptă după un semnal. **(se utilizează apelurile fork(). sigsuspend(), sigprocmask, sigfillset(), sigaction(), pause())**

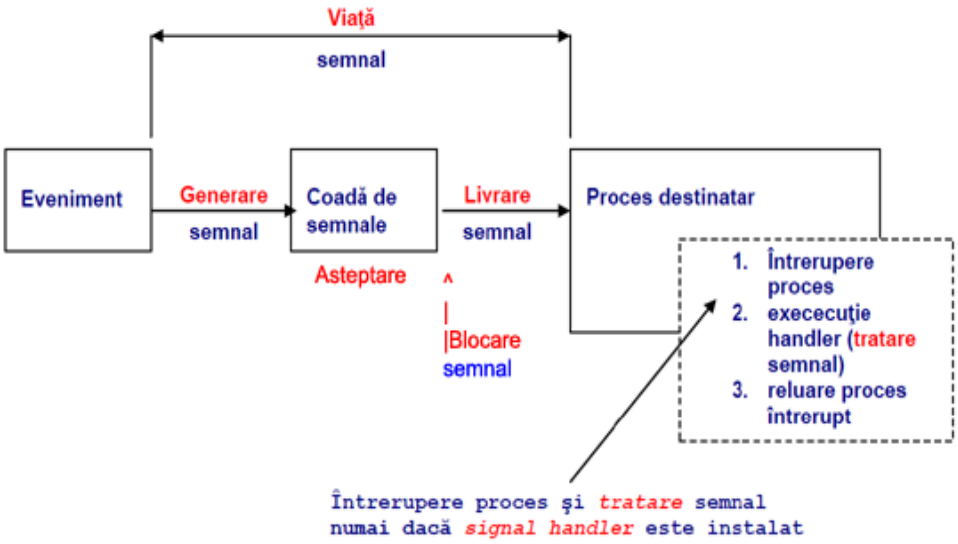
   Proces Părinte, așteaptă prin apel pause() primirea semnalului SIGUSR2 de la Copil. După primirea și tratarea semnalului SIGUSR2, Părintele trimite semnalele SIGINT şi SIGUSR1 către procesul Copil. În Copil, iniţial toate semnalele sunt blocate. După executarea unei secvenţe critice, Copilul emite semnalul SIGUSR2 către Părinte, deblocheaza SIGUSR1 şi aşteaptă numai semnalul SIGUSR1 prin apel sigsuspend(). SIGUSR1 este tratat într-o secvență de tratare, după care procesul Copil deblochează SIGINT şi tratatează semnalul. (se utilizează apelurile fork(). sigsuspend(), sigprocmask, sigfillset(), sigaction(), kill(), pause()

- **Context recapitulativ din Lucrarea de laborator nr. 07**

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).*(SIGINT, SIGTERM, SIGUSR2,...etc)
Semnalele Unix reprezintă un mecanism fundamental de comunicare între procese Unix. Un **semnal** este o notificare software transmisa unui anumit proces. Generarea unui semnal este e declanşată de apariţia unui eveniment cuprins în următoarle categorii: -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 *semna*l poate fi în următoarele stări (vezi figura de mai jos):

1. ***generat*** **(generated)** atunci când evenimentul cauzează apariţia respectivului semnal.
2. ***livrat*** (**delivered)** atunci când procesul poate executa acţiuni bazate pe semnalul respectiv.
3. ***în Viaţa*** **(lifetime)** *Viața* unui ***semnal*** reprezintă intervalul de timp dintre *generarea* şi *livrarea* sa.
4. ***în asteptare*** (**pending)**. 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.
5. ***[blocat]*** **(blocking).** Stare temporară. Un semnal *blocat* NU este *livrat* procesului. *Blocarea* unui semnal nu este acelaşi lucru cu *ignorarea* lui.
6. ***în tratare.*** 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. În caz că utilizatorul nu a instalat un handler de semnal, atuci tratarea e lăsată în seama sistemului de operare.

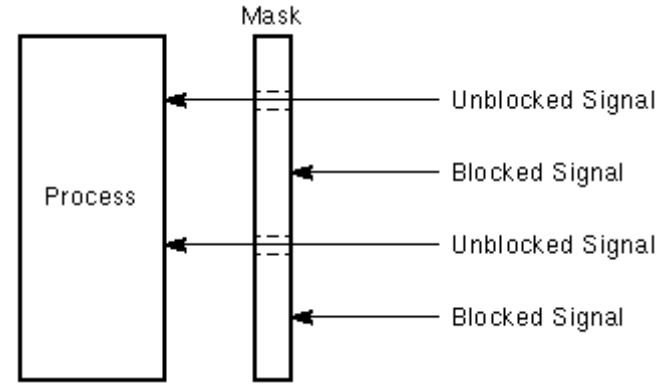


Proces destinatar

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

- **Context**

Un proces poate temporar preveni *livrarea* unui semnal prin **blocarea** lui. Până când nu sunt *livrate* semnalele *blocate* nu afectează funcţionarea procesului. **Masca de semnal (signal mask)** a unui proces furnizează **setul de semnale** care la un moment dat sunt blocate. *Masca de semnal* este o dată de tip **`sigset_t`**.
Tipul de dată **`sigset_t`** este utilizat pentru a reprezenta un set de semnale. Intern acest tip de dată poate fi implementat fie ca dată de tip intreg, fie ca o dată de tip structură.



Proces destinatar mască semnal semnale

Blocarea unui semnal nu este acelaşi lucru cu *ignorarea* lui. Un semnal *blocat* NU este *livrat* procesului. Procesului i se va *livra* semnalul după ridicarea *blocării* lui, când acesta îl va *trata*. Prin *ignorarea* unui semnal, procesului i se va *livra* semnalul şi acesta imediat îl respinge (NU îl va *trata*).
Nu este posibilă blocarea semnalelor SIGKILL sau SIGSTOP; specificarea acestor semnale în mască nu are niciun efect asupra măştii de semnal a procesului.

- **Apeluri de referință**

Utilizarea funcţiilor *sigaction(), sigsuspend(), - sigfillset(), sigadsset() sigprocmask()* și *sigdelset()*

examine and change a signal action	wait for a signal	initialize and fill a signal set
#include <signal.h> **int sigaction** **(int signum, const struct sigaction *restrict act, struct sigaction *restrict oldact);**	**#include <signal.h>** **int sigsuspend** **(const sigset_t *sigmask);**	**#include <signal.h>** **int sigfillset** **(sigset_t *set);**

examine and change blocked signals	add a signal to a signal set	delete a signal from a signal set
#include <signal.h> **int sigprocmask** **(int how, const sigset_t *restrict set, sigset_t *restrict oldset);**	**#include <signal.h>** **int sigadsset** **(sigset_t *set, int signo);**	**#include <signal.h>** **int sigdelset** **(sigset_t *set, int signo);**

Apelul `sigaction` este utilizat t pentru a schimba acțiunea (în principal secvența de cod handler semnal) întreprinsă de un proces la primirea unui semnal valid specificat prin parametrul `signum` (ex. SIGINT, SIGTERM, SIGUSR, ...etc) cu excepția semnalelor SIGKILL și SIGSTOP.
Dacă parametrul `act` este non-NULL, noua acțiune (în principal handler semnal) în cazul recepției semnalului `signum` este instalată din `act`. Dacă `oldac`t nu este NULL, acțiunea anterioară este salvată în `oldac`t.

Structura `sigaction` este definită astefel:

```
struct sigaction {
   void     (*sa_handler)(int);      // handler de semnal-
                                     // tratare receptie semnal
   void     (*sa_sigaction)(int, siginfo_t *, void *);
   sigset_t sa_mask;
   int      sa_flags;
   void     (*sa_restorer)(void);
}actiune_SIGUSR1;
```

Exemplu de apel: `sigaction(SIGUSR1, &actiune_SIGUSR1, NULL)`

Apelul `sigsuspend` înlocuieşte temporar *masca de semnal* curentă a procesului apelant printr-un set de semnale punctat prin `sigmask`, după care *suspendă* procesul apelant până când un semnal este *livrat.* O dată semnalul *livrat* pot avea loc două acţiuni distincte: fie execuţia unei secvenţe de tratare semnal, fie terminarea procesului apelant. Dacă acţiunea este de terminare a procesului apelant, atunci `sigsuspend()` nu returnează niciodată, procesul apelant terminîndu-se. Dacă acţiunea este execuţia unei secvenţe de tratare semnal, atunci `sigsuspend()` returnează după ce secvenţa de tratare semnal returnează, având loc şi restaurarea *măştii semnal* la setul de dinaintea apelului `sigsuspend`. În caz de eroare, pentru ambele acţiuni, `sigsuspend` returnează `-1` şi `errno` este setat pentru a indica eroarea.
În mod normal `sigsuspend()` este utilizată în conjucţie cu `sigprocmask()` pentru a preveni *livrarea* unui semnal în timpul executării unei secvenţe critice. Procesul apelant mai întâi *blochează* semnalele prin

sigprocmask(), iar atunci când secvenţa critică s-a terminat, prin **sigsuspend()** aşteaptă *livrarea* semnalului.
O secvenţă de principiu a utilizării tandemului **sigsuspend** / **sigprocmask** este dată mai jos:

```
#include<signal.h>

// int signal_received=0; // la thread-uri

sigset_t sigset;
sigset_t sigoldmak;
int signum;       // se vor seta valori din gama SIGINT, SIGTERM, SIGUSR1,..etc
signum=SIGINT;    // un exemplu de semnal
....
sigprocmask(SIG_SETMASK, NULL, &sigoldmask);  // salvează masca semnal
sigprocmask(SIG_SETMASK, NULL, &sigset);
....
/* Setare semnale temporar blocate */
sigaddset(&sigset, signum);
sigprocmask(SIG_BLOCK, &sigset, NULL);    //    blocare semnale
....
sigdelset(&sigset, signum);               // deblocare signum

// while(signal_received==0) sigsuspend(&sigset);// utilizare la thread-uri
sigsuspend(&sigset);                             // utilizare la procese
sigprocmask(SIG_SETMASK, &sigoldmask, NULL); // refacere masca semnal
```

NOTĂ:
Datorită faptului că **sigsuspend** restaurează *masca de semnal* dinaintea apelului (cu **signum** blocat) se impune refacerea *măstii de semnal* prin al doilea apel **sigprocmask** cu parametrul **&sigoldmask**.
Ciclul **while** este necesar atunci când semnalul aşteptat **signum** este generat din mai multe (sub)procese şi doar cel care pune **signal_received#0** este luat în considerare.

Apelul **sigfilleset** inițializează un set de semnale cu toate semnalele definite (SIGINT, SIGTERM,...etc). Opusul apelului **sigfilleset** este apelul **sigemptyset** care exclude toate toate semnalele definite (SIGINT, SIGTERM,...etc).
Exemplu de apel: ***sigfillset(&sigset); // introduc in sigset toate semnalele***
Apelul **sigaddset** adaugă semnalul specificat într-un set de semnale.
Exemplu de apel: ***sigaddset(&sigset, SIGINT);***
Apelul **sigdelset** șterge un anumit semnal dintr-un set de semnale. Ajută la deblocarea unui semnal dintr-un set de semnale blocat anterior prin apel ***sigprocmask(SIG_BLOCK,...)*** - blocarea unui semnal nu înseamnă ignorarea lui. La deblocare el va fi tratat.
Exemplu de apel: ***sigdelset(&sigset, SIGINT);***

Utilizarea funcţiei *pause(),*

Apelul **pause()** suspendă un proces sau un fir de execuție până când este livrarat(recepționat) un semnal
O secvenţă de principiu a utilizării apelului **pause** este dată mai jos:

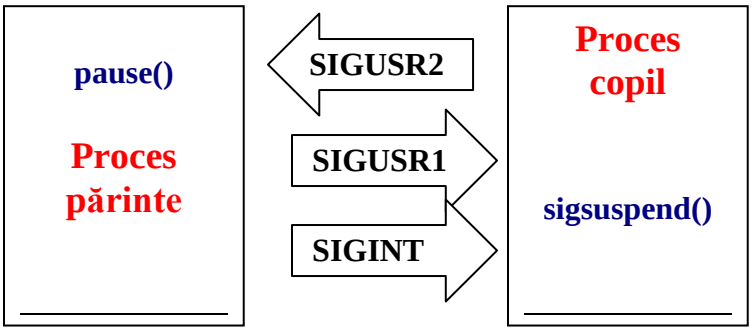
```
#include <unistd.h>

int pause(void);
```

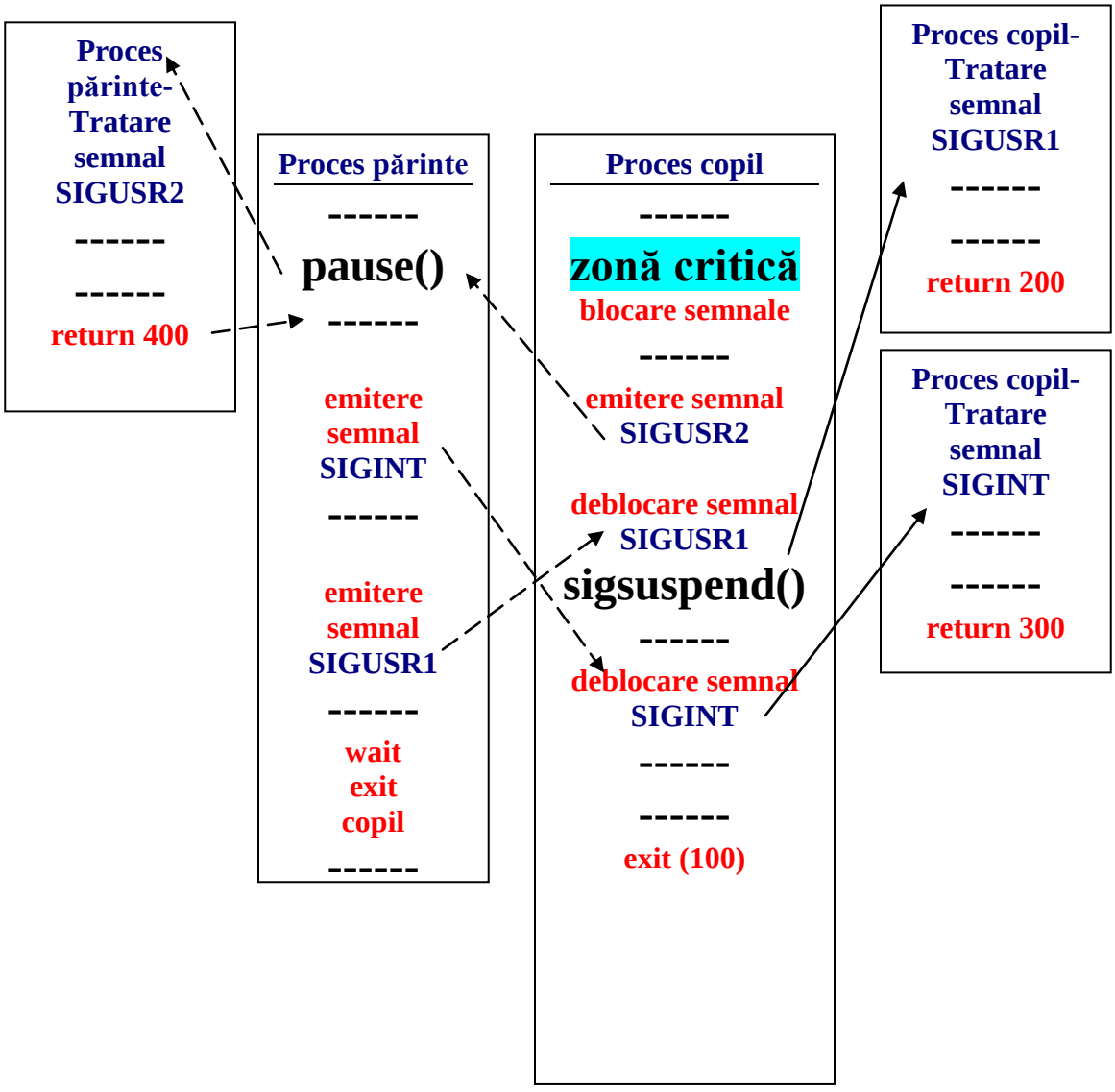
Apelul **pause()** returnează numai când un semnal a fost livrat și funcția de tratare a lui a returnat. În acest caz **pause()** returnează -1 și **errno** este setat la **EINTR**

Aplicație demonstrativă:

1. Ateptarea după semnal. Program care crează două procese ce așteaptă după un semnal (se utilizează apelurile fork(). sigsuspend(), sigprocmask, sigfillset(), sigaction(), pause(), kill().
   Procesul Părinte, așteaptă prin apel pause() primirea semnalului SIGUSR2 de la Copil. După primirea și tratarea semnalului SIGUSR2, Părintele trimite semnalele SIGINT şi SIGUSR1 către procesul Copil. În Copil, iniţial toate semnalele sunt blocate. După executarea unei secvenţe critice, Copilul emite semnalul SIGUSR2 către Părinte, deblocheaza SIGUSR1 şi aşteaptă numai semnalul SIGUSR1 prin apel sigsuspend(). SIGUSR1 este tratat într-o secvență de tratare, după care procesul Copil deblochează SIGINT şi tratatează semnalul..

Program *sigsuspend.c* - Așteptarea după un semnal





```
/* sigsuspend.c */
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h> // pt. exit()
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

int main()
{
pid_t pid;      // memoreaza PID copil
sigset_t sigset, sigoldmask;

if ((pid = fork()) == 0) {
  /* cod copil */

    struct sigaction actiune_SIGUSR1;
    int TratareSemnalSIGUSR1();

    struct sigaction actiune_SIGINT;
    int TratareSemnalSIGINT();

  /* salvare masca semnal curenta in sigoldmask */
    sigprocmask(SIG_SETMASK, NULL, &sigoldmask);  // probabil in sigildmask SIGINT nu e blocat
  /* setare masca semnal curenta in sigset */
    sigfillset(&sigset);  // introduc in sigset toate semnalele
    sigprocmask(SIG_SETMASK, &sigset, NULL);
    fprintf(stderr, "C: Intru in zona critica - blochez toate semnalele inclusiv SIGINT si  SIGUSR1 \n");
  /* ++++++ Zona critica  se blocheaza toate semnalele */
  /*  in aceasta zona Parintele nu trebuie sa emita semnale.
      Acum Parintele  sta suspendat prin apel in pause()
      pentru ca Copilul nu e pregatit sa primeasca semnale de la el   */
    sigprocmask(SIG_BLOCK, &sigset, NULL); // se blocheaza toate semnalele,

  /* in copil se creaz冲ignal handler*/
  //SIGUSR1
    actiune_SIGUSR1.sa_flags = 0;
    /*set noul handler semnal SIGUSR1*/;
    actiune_SIGUSR1.sa_handler = (void (*))(int)TratareSemnalSIGUSR1;

    if (sigaction(SIGUSR1, &actiune_SIGUSR1, NULL) == -1){  // instalez actiune SIGUR1
     perror("Eroare: sigaction");
     exit(1);
    }
  //SIGINT
    actiune_SIGINT.sa_flags = 0;
    /*set noul handler semnal SIGINT*/;
    actiune_SIGINT.sa_handler = (void (*))(int)TratareSemnalSIGINT;

    if (sigaction(SIGINT, &actiune_SIGINT, NULL) == -1){   // instalez actiune SIGINT
     perror("Eroare: sigaction");
     exit(1);
    }
     fprintf(stderr, "C: deblochez SIGUSR1 \n");
     sigdelset(&sigset, SIGUSR1);   // se deblocheaza numai semnal SIGUSR1/

    /* --------- Sfarsit zona critica */
```

```
            fprintf(stderr, "C: emit SIGUSR2 catre parinte pentru a incepe sa emita semnale \n");
            kill(getppid(), SIGUSR2);        //  emit semnal SIGUSR2 catre parinte - reluare parinte suspendat
            /* astept numai semnal SIGUSR1 - restul semnalelor sunt blocate */
             sigsuspend(&sigset);
            /* aici se ajunge. Return din  TratareSemnalSIGUSR1 */
            fprintf(stderr, "C: am iesit din asteptare semnal. SIGUSR1 a fost livrat\n");
            fprintf(stderr, "C: refac masca de semnal initiala cu semnal SIGINT probalil deblocat\n");
            /* Restaureaza masca de semnal in care SIGINT (probabil) nu e blocat*/

              sigprocmask(SIG_SETMASK, &sigoldmask, NULL);
}
else {
  /* cod parinte */
  int stat;
  int TratareSemnalSIGUSR2();
  signal(SIGUSR2,(void (*))(int)TratareSemnalSIGUSR2);
  pause();   /* se asteapta un semnal SIGUSR2  de la copil atunci cand copilul asteapta semnale */
  fprintf(stderr,
      "P: Am primit SIGUSR2, deci Copilul este pregatit pentru a receptiona SIGINT si SIGUSR1\n");
  fprintf(stderr, "P: emit un semnal SIGINT - inca blocat de Copil, catre Copil\n");
  kill(pid, SIGINT);          //  emit semnal SIGINT  ctrl-c (fara efect- semnalul este blocat in Copil)
  fprintf(stderr, "P: emit un semnal SIGUSR1 - deblocat de Copil, catre Copil\n");
  kill(pid, SIGUSR1);       //  emit semnal SIGUSR1 (cu efect- semnalul nu este blocat in Copil)
  fprintf(stderr, "P: Astep terminarea Copilului\n");
  pid=wait(&stat);
  /* Macro WEXITSTATUS(stat)   returneaza codul de retur al procesului copil dat prin exit(100) */
   fprintf(stderr,"P: Copilul a iesit cu exit status = %d\n", WEXITSTATUS(stat));

  }
/* secventa comuna parinte/copil */
  exit(100);    // cod retur proces = 100
}

/* Functii tratare semnale /  sunt activate numai pentru semnale neblocate */

int TratareSemnalSIGUSR1 (int signo)
{
  fprintf(stderr, "\tC-TS:  tratare semnal SIGUSR1 - Semnal %d - emis din parinte\n", signo);
  fflush(stderr);   // golire fortata bufer
  return 200;
}
int TratareSemnalSIGINT (int signo)
{
  fprintf(stderr, "\tC-TS:  tratare semnal SIGINT - Semnal %d - emis din parinte\n", signo);
  fflush(stderr);   // golire fortata bufer
  return 300;
}
int TratareSemnalSIGUSR2 ()
{
  signal(SIGUSR2,(void (*))(int)TratareSemnalSIGUSR2);
  fprintf(stderr, "\tP-TS:  tratare semnal SIGUSR2 - Semnal %d - emis din copil- porneste parintele\n", SIGUSR2);
  fflush(stderr);   // golire fortata bufer
  return 400;
}
```

Compilare linkeditare şi rulare program *sigsuspend.c*

$gcc -o sigsuspend sigsuspend.c¶

$./sigsuspend¶

Mesaje afişate în urma rulării

C: Intru in zona critica - blochez toate semnalele inclusiv SIGINT si SIGUSR1
C: deblochez SIGUSR1
C: emit SIGUSR2 catre parinte pentru a incepe sa emita semnale
  P-TS: tratare semnal SIGUSR2 - Semnal 12 - emis din copil- porneste parintele
P: Am primit SIGUSR2, deci Copilul este pregatit pentru a receptiona SIGINT si SIGUSR1
P: emit un semnal SIGINT - inca blocat de Copil, catre Copil
P: emit un semnal SIGUSR1 - deblocat de Copil, catre Copil
P: Astep terminarea Copilului
  C-TS: tratare semnal SIGUSR1 - Semnal 10 - emis din parinte
C: am iesit din asteptare semnal. SIGUSR1 a fost livrat
C: refac masca de semnal initiala cu semnal SIGINT probalil deblocat
  C-TS: tratare semnal SIGINT - Semnal 2 - emis din parinte
P: Copilul a iesit cu exit status = 100

Observaţii:

1. .....
