# Programare concurentă în C (VII):

## *Gestiunea proceselor, partea a III-a: Semnale* UNIX

Cristian Vidrașcu

vidrascu@info.uaic.ro

# Sumar

- Introducere
- Categorii de semnale
- Tipurile de semnale predefinite ale `UNIX`-ului
- Cererea explicită de generare a unui semnal – primitiva `kill`
- Coruperea semnalelor – primitiva `signal`
- Definirea propriilor *handler*e de semnal
- Blocarea semnalelor
- Așteptarea unui semnal

# Introducere

Semnalele UNIX reprezintă un mecanism fundamental de manipulare a proceselor și de comunicare între procese, ce asigură tratarea evenimentelor asincrone apărute în sistem.

Un semnal UNIX este o *întrerupere software* generată în momentul producerii unui anumit eveniment și transmisă de sistemul de operare unui anumit proces.

# Introducere (cont.)

Un semnal este *generat* de apariția unui eveniment excepțional (care poate fi o eroare, un eveniment extern sau o cerere explicită). Orice semnal are asociat un *tip*, reprezentat printr-un număr întreg pozitiv (ce codifică cauza sa), și un proces *destinatar*.

Odată generat, semnalul este pus în *coada de semnale* a sistemului, de unde este extras și transmis procesului destinatar de către sistemul de operare.

*Transmiterea* semnalului către destinatar se face imediat după ce semnalul a ajuns în coada de semnale, cu o excepție: dacă primirea semnalelor de tipul respectiv a fost *blocată* de către procesul destinatar, atunci transmiterea semnalului se va face abia în momentul când procesul destinatar va debloca primirea acelui tip de semnal.

# Introducere (cont.)

În momentul în care procesul destinatar primește acel semnal, el își *întrerupe execuția* și va executa o anumită acțiune (*i.e.*, o funcție de tratare a acelui semnal, funcție numită *handler de semnal*) care este atașată tipului de semnal primit, după care procesul își va relua execuția din punctul în care a fost întrerupt (cu anumite excepții: unele semnale vor cauza terminarea forțată a acelui proces).

În concluzie, fiecare tip de semnal are asociat o acțiune (un *handler*) specifică acelui tip de semnal.

# Categorii de semnale

Evenimentele ce genereaza semnale se împart în trei categorii:

- erori (în procesul destinatar)

  O **eroare** înseamnă că programul a făcut o operație invalidă și nu poate să-și continue execuția. Nu toate erorile generează semnale, ci doar acele erori care pot apare în orice punct al programului, cum ar fi: împărțirea la zero, accesarea unei adrese de memorie invalide, etc.

- evenimente externe (procesului destinatar)

  **Evenimentele externe** sunt în general legate de operațiile I/O sau de acțiunile altor procese, cum ar fi: sosirea datelor (pe un *socket* sau *pipe*), terminarea unui proces fiu, expirarea intervalului de timp setat pentru o alarmă, sau suspendarea ori terminarea programului de către utilizator (prin apăsarea tastelor `^Z` ori `^C`).

- cereri explicite

  O **cerere explicită** înseamnă generarea unui semnal de către un (alt) proces, prin apelul primitivei `kill`.

# Categorii de semnale (cont.)

Semnalele pot fi generate *sincron* sau *asincron*.

- Un semnal **sincron** este generat de o anumită acțiune specifică în program și este livrat (dacă nu este blocat) în timpul acelei acțiuni.
  - Evenimente ce generează semnale sincrone: erorile și cererile explicite ale unui proces de a genera semnale pentru el însuși.
- Un semnal **asincron** este generat de un eveniment din afara zonei de control a procesului care îl recepționează; cu alte cuvinte, un semnal ce este recepționat, în timpul execuției procesului destinatar, la un moment de timp ce nu poate fi anticipat.
  - Evenimente ce generează semnale asincrone: evenimentele externe și cererile explicite ale unui proces de a genera semnale destinate altor procese.

# Categorii de semnale (cont.)

Pentru fiecare tip de semnal există o acțiune implicită de tratare a acelui semnal, specifică sistemului de operare `UNIX` respectiv. Această acțiune este denumită *handlerul implicit de semnal* atașat acelui tip de semnal.

Atunci când semnalul este livrat procesului, acesta este întrerupt și are trei posibilități de comportare:

- fie să execute această acțiune implicită,
- fie să ignore semnalul,
- fie să execute o anumită funcție *handler* utilizator.

Setarea unuia dintre cele trei comportamente se face cu ajutorul apelului primitivelor `signal` sau `sigaction`.

# Tipurile de semnale predefinite ale UNIX-ului

Tipurile predefinite de semnale din UNIX se clasifică în mai multe categorii:

- semnale standard de eroare: SIGFPE, SIGILL, SIGSEGV, SIGBUS
- semnale de terminare a proceselor: SIGHUP, SIGINT, SIGQUIT, SIGTERM, SIGKILL
- semnale de alarmă: SIGALRM, SIGVTALRM, SIGPROF
- semnale asincrone I/O: SIGIO, SIGURG
- semnale pentru controlul proceselor: SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU
- alte tipuri de semnale: SIGPIPE, SIGUSR1, SIGUSR2

# Tipurile de semnale predefinite ale UNIX-ului (cont.)

Lista semnalelor UNIX predefinite, mai exact numărul întreg asociat fiecărui tip de semnal, poate fi obținută cu comanda următoare:

```
UNIX> kill -l
```

iar pagina de manual ce conține descrierea semnalelor este:

```
UNIX> man 7 signal
```

*Observație*: o parte dintre aceste tipuri de semnale depind și de suportul oferit de partea de *hardware* a calculatorului respectiv, nu numai de sistemul de operare de pe acel calculator. Din acest motiv, există mici deosebiri în modul de implementare a acestor semnale pe diferite tipuri de arhitecturi de calculatoare (adică unele semnale se poate să nu fie implementate deloc, sau să fie implementate cu mici diferențe).
Exemple de semnale ce pot diferi de la un tip de arhitectură la altul: cele generate de erori, cum ar fi SIGBUS (care nu este implementat in Linux-ul pentru *hardware*-ul i386).
În concluzie: trebuie studiată documentația tipului de calculator pe care îl utilizați pentru a vedea ce semnale aveți la dispoziție.

# Cererea explicită de generare a unui semnal – primitiva `kill`

Apelul de sistem `kill` este utilizat pentru a cere explicit generarea unui semnal. Interfața acestei funcții:

`int kill (int` *pid*`, int` *id-signal*`)`

- *pid* = `PID`-ul procesului destinatar
- *id-signal* = tipul semnalului
- valoarea returnată este `0`, în caz de reușită, sau `-1`, în caz de eroare.

Efect: în urma execuției funcției `kill` se generează un semnal de tipul specificat, destinat procesului specificat.

# Cererea explicită de generare a unui semnal – primitiva `kill`

*Observație*: prin apelul `kill(`*pid*`,0);` nu se trimite nici un semnal, dar este util pentru verificarea validității `PID`-ului respectiv (*i.e.*, dacă există un proces cu acel `PID` în momentul apelului, sau nu): se returnează `0` dacă `PID`-ul specificat este valid, sau `-1`, în caz contrar.

Pentru cererea explicită de generare a unui semnal se poate folosi și comanda `kill`:

`UNIX>` `kill -`*semnal pid*

Un proces își poate trimite semnale sie însuși folosind funcția `raise`, ce are interfața:

`int raise(int` *id-signal*`)`

Efect: este echivalent cu apelul `kill(getpid(),`*id-signal*`);` .

# Coruperea semnalelor – primitiva `signal`

Acțiunea asociată unui semnal poate fi:

- o acțiune implicită (specifică sistemului de operare respectiv),
- sau ignorarea semnalului,
- sau un *handler* propriu, definit de programator.

Se utilizează termenul de *corupere a unui semnal* cu sensul de: setarea unui *handler* propriu pentru acel tip de semnal.

*Notă*: uneori, se folosește și termenul de *tratare a semnalului*.

*Observație*: semnalele `SIGKILL` si `SIGSTOP` nu pot fi corupte, ignorate sau blocate!

# Coruperea semnalelor – primitiva `signal` (cont.)

Specificarea acțiunii asociate unui semnal se poate face cu apelurile de sistem `signal` sau `sigaction`.

Interfața primitivei `signal` este:

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

- *id-signal* = tipul semnalului căruia i se asociază acea acțiune
- *action* = acțiunea (*i.e.*, *handler*ul de semnal) ce se asociază semnalului; poate fi numele unei funcții definite de programator, sau poate lua una dintre valorile:
  - `SIG_DFL` : specifică acțiunea implicită (cea stabilită de către sistemul de operare) la recepționarea semnalului
  - `SIG_IGN` : specifică faptul că procesul va ignora acel semnal
- valoarea returnată este vechiul *handler* pentru semnalul specificat, sau constanta simbolică `SIG_ERR` în caz de eroare.

# Coruperea semnalelor – primitiva `signal` (cont.)

Interfața primitivei `signal` este:

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

Efect: se asociază *handler*ul specificat pentru acel tip de semnal.
Ca urmare, ulterior (pînă la o nouă recorupere), ori de câte ori procesul va recepționa semnalul *id-signal*, se va executa *handler*ul de semnal *action*.

*Observație*: în general nu este bine ca programul să ignore semnalele (mai ales pe acelea care reprezintă evenimente importante). Dacă se dorește ca programul să nu recepționeze semnale în timpul execuției unei anumite porțiuni de cod (pentru a nu fi întreruptă), soluția cea mai indicată este să se *blocheze* primirea semnalelor, nu ca ele să fie ignorate.

# Coruperea semnalelor – primitiva `signal` (cont.)

Interfața primitivei `signal` este:

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

Dacă argumentul *action* este numele unei funcții definite de utilizator, această funcție trebuie să aibă prototipul `sighandler_t`, definit astfel:

```
typedef void (*sighandler_t)(int);
```

*i.e.*, tipul "funcție ce întoarce tipul `void`, și are un argument de tip `int`".

*Notă*: la momentul execuției unui *handler* de semnal, acest argument va avea ca valoare numărul semnalului ce a determinat execuția acelui *handler*. În acest fel, se poate asigna o aceeași funcție ca și *handler* pentru mai multe semnale, în corpul ei putând ști, pe baza argumentului primit, care dintre acele semnale a cauzat apelul respectiv.

# Coruperea semnalelor – primitiva `signal` (cont.)

Exemplu: un program care să ignore întreruperile de tastatură, adică semnalul `SIGINT` (generat de tastele `CTRL+C`) și semnalul `SIGQUIT` (generat de tastele `CTRL+\`).
A se vedea programul `sig-ex1.c` fără ignorarea celor două semnale (*i.e.*, poate fi întrerupt/oprit cu `CTRL+C`, respectiv `CTRL+\`), și respectiv programul `sig-ex2.c` cu ignorarea celor două semnale (*i.e.*, va rula fără a putea fi întrerupt/oprit cu `CTRL+C`, respectiv `CTRL+\`).

Să modificăm exemplul anterior astfel: corupem semnalele să execute un *handler* propriu, care să afișeze un anumit mesaj. Iar apoi refacem comportamentul implicit al semnalelor.
A se vedea programul `sig-ex3.c`

# Definirea propriilor *handler*e de semnal

Un *handler* de semnal propriu este o funcție definită de programator, ce va fi apelată atunci când procesul recepționează semnalul căruia îi este asociată.

Strategii principale folosite în scrierea de *handler*e proprii:

- Se poate ca *handler*ul să notifice primirea semnalului prin setarea unei variabile globale și apoi să returneze imediat, urmând ca în bucla principală a programului, să se verifice periodic dacă acea variabilă a fost setată, în care caz se vor efectua operațiile dorite.
- Se poate ca *handler*ul să termine execuția procesului, sau să transfere execuția într-un punct în care procesul poate să-și recupereze starea în care se afla în momentul recepționării semnalului.

# Definirea propriilor *handler*e de semnal (cont.)

*Atenție*: trebuie luate măsuri speciale atunci când se scrie codul pentru *handler*ele de semnal, deoarece acestea pot fi apelate asincron, adică la momente de timp imprevizibile.

Spre exemplu, în timp ce se execută *handler*ul asociat unui semnal primit, acesta poate fi întrerupt prin recepția unui alt semnal (al doilea semnal trebuie să fie de alt tip decât primul; dacă este același tip de semnal, el va fi blocat până când se termină tratarea primului semnal).

*Important*: prin urmare, primirea unui semnal poate întrerupe nu doar execuția programului respectiv, ci chiar execuția *handler*ului unui semnal anterior primit, sau poate întrerupe execuția unui apel de sistem efectuat de program în acel moment.

# Blocarea semnalelor

*Blocarea semnalelor* înseamnă că procesul spune sistemului de operare să nu îi transmită anumite semnale (ele vor rămâne în coada de semnale, până când procesul va debloca primirea lor).

*Notă*: nu este recomandabil ca un program să blocheze semnalele pe tot parcursul execuției sale, ci numai pe durata execuției unor porțiuni critice ale codului său. Astfel, dacă un semnal ajunge în timpul execuției acelei porțiuni de program, el va fi livrat procesului abia după terminarea execuției acesteia și deblocarea acelui tip de semnal.

# Blocarea semnalelor

*Blocarea semnalelor* înseamnă că procesul spune sistemului de operare să nu îi transmită anumite semnale (ele vor rămâne în coada de semnale, până când procesul va debloca primirea lor).

Blocarea semnalelor se realizează cu funcția `sigprocmask`, ce utilizează structura de date `sigset_t` (care este o mască de biți, cu semnificația de set de semnale ales pentru blocare).
Cu primitiva `sigpending` se poate verifica existența, în coada de semnale, a unor semnale blocate (deci care așteaptă să fie deblocate pentru a putea fi livrate procesului).

Exemplu: a se vedea fișierul sursă `sig-ex4.c`

# Așteptarea unui semnal

Dacă aplicația este influențată de evenimente externe, sau folosește semnale pentru sincronizare cu alte procese, atunci ea nu trebuie să facă altceva decât să aștepte semnale.

Se poate folosi în acest scop funcția `pause`, ce are prototipul:

```
int pause()
```

Efect: suspendarea execuției programului până la sosirea unui semnal.

*Observație*: simplitatea acestei funcții poate ascunde erori greu de detectat. Deoarece programul principal nu face altceva decât să apeleze `pause()`, înseamnă că cea mai mare parte a activității utile în program o realizează *handler*ele de semnal. Însă, codul acestor *handler*e nu este indicat să fie prea lung, deoarece poate fi întrerupt de alte semnale.

# Așteptarea unui semnal (cont.)

Modalitatea cea mai indicată, pentru așteptarea unui anumit semnal (*i.e.*, așteptarea primului semnal primit, dintr-o mulțime fixată de semnale), este de a folosi funcția `sigsuspend`, ce are prototipul:

```
int sigsuspend(const sigset_t *set)
```

Efect: se înlocuiește masca de semnale curentă a procesului cu cea specificată de parametrul *set* și apoi se suspendă execuția procesului până la recepționarea unui semnal, de către proces (deci un semnal care nu este blocat, adică nu este cuprins în masca de semnale curentă).
Masca de semnale rămâne la valoarea setată (*i.e.*, valoarea lui *set*) numai până când funcția `sigsuspend()` returnează, moment în care este reinstalată, în mod automat, vechea mască de semnale.

# Așteptarea unui semnal (cont.)

Modalitatea cea mai indicată, pentru așteptarea unui anumit semnal (*i.e.*, așteptarea primului semnal primit, dintr-o mulțime fixată de semnale), este de a folosi funcția `sigsuspend`, ce are prototipul:

```
int sigsuspend(const sigset_t *set)
```

Valoarea returnată: `0`, în caz de succes, respectiv `-1`, în caz de eșec (iar variabila `errno` este setată în mod corespunzător: `EINVAL`, `EFAULT` sau `EINTR`).

Exemplu: un program care își suspendă execuția în așteptarea semnalului `SIGQUIT` (generat de tastele `CTRL+\`), fără a fi întrerupt de alte semnale. A se vedea fișierul sursă `sig-ex5.c`

# Bibliografie obligatorie

Cap.4, §4.5 din manualul, în format PDF, accesibil din pagina disciplinei “Sisteme de operare”:

- `https://profs.info.uaic.ro/~vidrascu/SO/books/ManualID-SO.pdf`

Programele demonstrative amintite pe parcursul acestei prezentări pot fi descărcate de la adresa următoare:

- `https://profs.info.uaic.ro/~vidrascu/SO/cursuri/C-programs/signal/`