Programare de sistem în C pentru platforma Linux (VI)

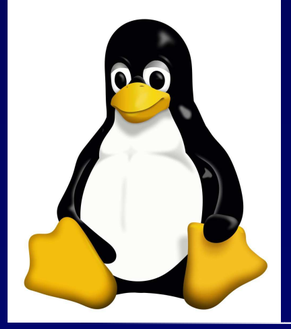
Gestiunea proceselor, partea a III-a:

Notificarea proceselor prin semnale

Cristian Vidrașcu

cristian.vidrascu@info.uaic.ro

Mai, 2024



Sumar

Introducere

Semnale POSIX

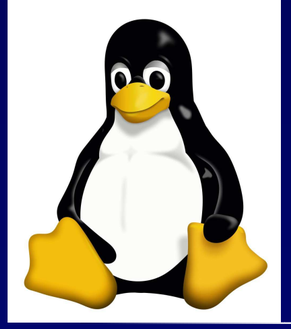
Referințe bibliografice

Introducere

Semnale POSIX

- Noțiuni generale despre semnale
- Categorii de semnale
- Primitiva `kill` pentru generarea unui semnal
- Tipurile predefinite de semnale din standardul POSIX
- Tratarea semnalelor
- Primitiva `signal` pentru configurarea tratării semnalelor
- *Demo*: exemple de cod
- Definirea propriilor *handler*e de semnal
- Blocarea semnalelor
- Așteptarea unui semnal

Referințe bibliografice



Introducere

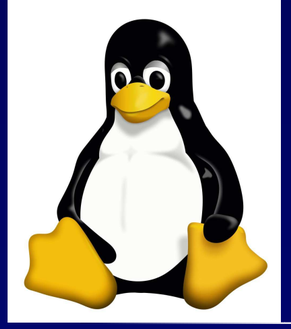
Introducere

Semnale POSIX

Referințe bibliografice

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

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



Agenda

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

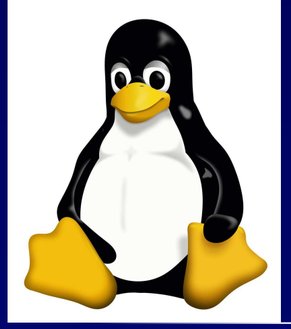
Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice



Noțiuni generale despre semnale

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

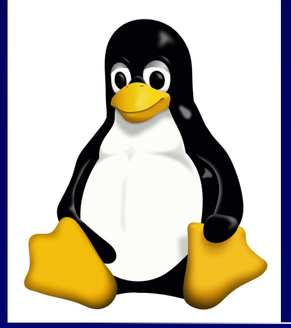
Așteptarea unui semnal

Referințe bibliografice

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.



Noțiuni generale despre semnale (cont.)

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

Tratarea semnalului de către procesul destinatar:

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

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handlere* de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

Evenimentele ce generează semnale se împart în trei categorii:

- erori (în procesul destinatar)

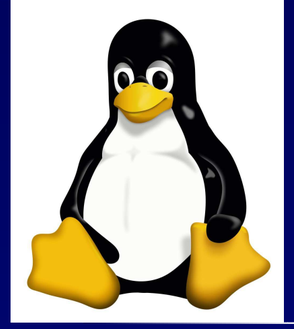
  O **eroare** înseamnă că programul a făcut o operație invalidă și nu poate să-și continue execuția. Nu toate erorile generează semnale, ci doar acele erori care pot apare în orice punct al programului, cum ar fi: împărțirea la zero, accesarea unei adrese de memorie invalide, etc.

- 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 (*e.g.*, prin apăsarea tastelor `^Z` ori `^C`).

- cereri explicite

  O **cerere explicită** înseamnă generarea unui semnal de către un (alt) proces, prin apelul uneia dintre primitivele `kill` sau `sigqueue`.



Categorii de semnale (cont.)

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handlere* de semnal

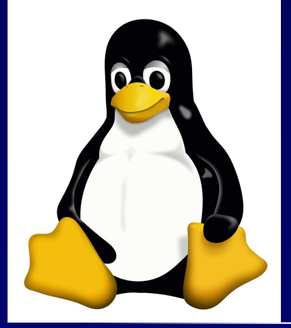
Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

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) procesului respectiv în timpul execuției 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.



Primitiva `kill` pentru generarea unui semnal

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

Un proces poate cere explicit generarea unui semnal prin apelul de sistem `kill`.

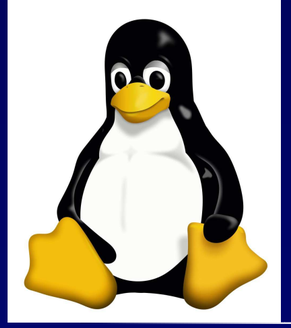
Interfața funcției `kill` ([5]):

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

- *pid* = PID-ul procesului destinatar
- *sig* = 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.

Observație: prin apelul `kill(pid,0);` nu se generează niciun 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.



Primitiva `kill` pentru generarea unui semnal (cont.)

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

Observație: Un proces își poate trimite semnale sie însuși folosind funcția `raise` din biblioteca C STANDARD LIBRARY, ce are interfața:

`int raise(int` *sig*`)`

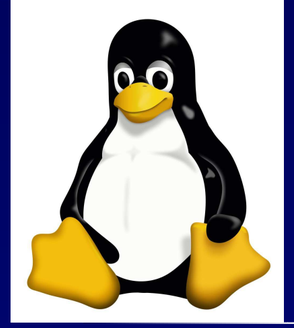
Efect: este echivalent cu apelul `kill(getpid(),`*sig*`)` .

* * *

Reamintire: pentru cererea explicită de generare a unui semnal de la linia de comandă, sau dintr-un script, se pot folosi comenzile `kill` și `killall`:

UNIX> `kill -`*sig pid*

UNIX> `killall -`*sig process_name*



Tipurile predefinite de semnale din standardul POSIX

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handlere* de semnal

Blocarea semnalelor

Așteptarea unui semnal

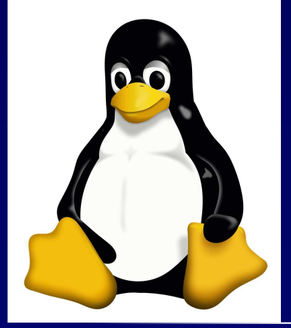
Referințe bibliografice

Tipurile predefinite de semnale din standardul POSIX pot fi clasificate astfel:

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

Toate tipurile enumerate mai sus se numesc *semnale standard*.

Mai există și categoria *semnale real-time*, formată din următoarele tipuri de semnale: `SIGRTMIN`, `SIGRTMIN+1`, ..., `SIGRTMIN+`*n*, ..., `SIGRTMAX`.
Spre deosebire de semnalele standard, semnalele în timp real nu au semnificații predefinite: întregul set de semnale în timp real poate fi utilizat în scopuri definite de aplicație. De asemenea, ele au semantici diferite – pentru mai multe detalii, consultați pagina de manual `man 7 signal` [6].



Tipurile predefinite de semnale din standardul POSIX (cont.)

Introducere

Semnale POSIX
Noțiuni generale despre semnale
Categorii de semnale
Primitiva `kill` pentru generarea unui semnal
Tipurile predefinite de semnale din standardul POSIX
Tratarea semnalelor
Primitiva `signal` pentru configurarea tratării semnalelor
Demo: exemple de cod
Definirea propriilor *handlere* de semnal
Blocarea semnalelor
Așteptarea unui semnal

Referințe bibliografice

Lista tipurilor de semnale predefinite, mai exact *numărul întreg* asociat fiecărui tip de semnal, poate fi obținută cu comanda următoare:

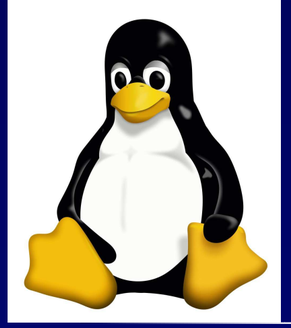
```
UNIX> kill -l
```

iar pagina de manual ce conține descrierea tipurilor de semnale este ([6]):

```
UNIX> man 7 signal
```

Observație: o parte dintre tipurile de semnale standard depind și de suportul oferit de partea de *hardware* a calculatorului utilizat. Din acest motiv, există mici deosebiri în modul de implementare a acestor semnale pe diferite tipuri de arhitecturi de calculatoare (adică este posibil ca unele semnale 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 hardware, cum ar fi `SIGBUS` (care nu este implementat în nucleul `Linux` pentru arhitectura *hardware* i386).

Concluzie: trebuie studiată documentația tipului de calculator pe care îl utilizați pentru a vedea ce semnale aveți la dispoziție.



Tratarea semnalelor

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

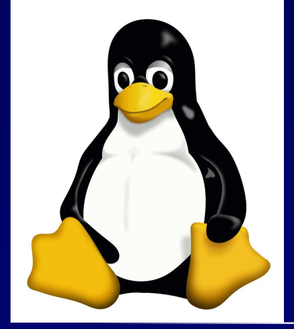
Pentru fiecare tip de semnal există o acțiune implicită de tratare a acelui semnal, definită în nucleul fiecărui sistem de operare UNIX. Această acțiune este denumită *handlerul implicit de semnal* atașat acelui tip de semnal.

Atunci când semnalul este livrat procesului, execuția acestuia este întreruptă și există trei posibilități de tratare a acelui semnal de către procesul respectiv:

- fie să execute *handler*ul implicit asociat tipului acelui semnal,
- fie să ignore semnalul,
- fie să execute o anumită funcție *handler* utilizator (*i.e.*, definită de programatorul care a scris programul respectiv).

Notă: termenul de *corupere a unui semnal* se utilizează atunci când programatorul își setează un *handler* propriu pentru acel tip de semnal.

Configurarea unui anumit *comportament de tratare a semnalelor* dintre cele trei de mai sus, se poate face apelând una dintre primitivele `signal` sau `sigaction`.



Primitiva `signal` pentru configurarea tratării semnalelor

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

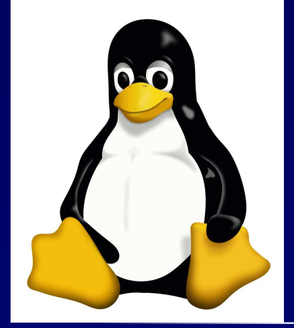
Interfața funcției `signal` ([5]):

`sighandler_t signal (int` *signum*`, sighandler_t` *handler*`)`

- *signum* = tipul semnalului căruia i se asociază acea acțiune de tratare
- *handler* = 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.

Efect: acelui tip de semnal i se asociază *handler*ul specificat.

Ca urmare, ulterior (*i.e.*, până la o nouă reconfigurare), ori de câte ori procesul va recepționa semnalul *signum*, se va executa *handler*ul de semnal *handler*.



Primitiva `signal` pentru configurarea tratării semnalelor (cont.)

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handlere* de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

Observație: dacă argumentul *handler* este numele unei funcții definite de programator, această funcție trebuie să aibă prototipul `sighandler_t`, definit astfel:

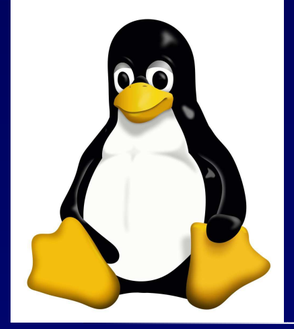
```
typedef void (*sighandler_t)(int);
```

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*. Astfel, 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.

Observație: în general nu este recomandat 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 recomandată este să se *blocheze* primirea semnalelor, nu ca ele să fie ignorate.

Important: semnalele `SIGKILL` și `SIGSTOP` nu pot fi corupte, ignorate sau blocate !



Demo: exemple de cod

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handlere* de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

Exemplu: un program care să ignore întreruperile de tastatură, adică semnalul `SIGINT` (generat de tastele `CTRL+C`) și semnalul `SIGQUIT` (generat de tastele `CTRL+\`).

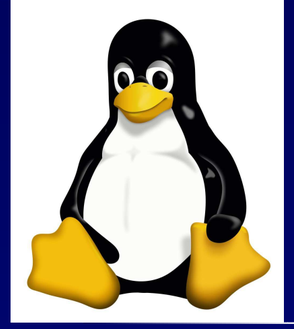
A se vedea programul `sig_ex1.c` ([2]), fără ignorarea celor două semnale (*i.e.*, poate fi întrerupt / oprit cu `CTRL+C`, respectiv `CTRL+\`), și respectiv programul `sig_ex2.c` ([2]), cu ignorarea celor două semnale (*i.e.*, va rula fără a putea fi întrerupt / oprit cu `CTRL+C`, respectiv `CTRL+\`).

* * *

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` ([2]).

Demo: pentru explicații mai detaliate, a se vedea [FirstDemo], [SecondDemo] și [ThirdDemo] prezentate în suportul de laborator #13.



Definirea propriilor *handlere* de semnal

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handlere* de semnal

Blocarea semnalelor

Așteptarea unui semnal

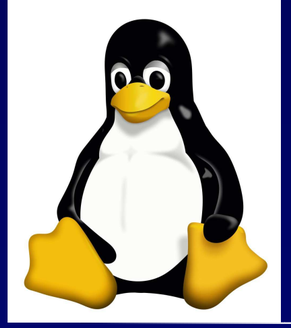
Referințe bibliografice

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

Demo: exemplul anterior, *i.e.* programul `sig_ex3.c`, ilustrează folosirea unui *handler* de semnal propriu.

Strategii principale folosite în scrierea de *handlere* 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 *handlere* de semnal (cont.)

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handlere* de semnal

Blocarea semnalelor

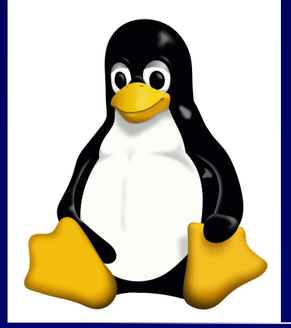
Așteptarea unui semnal

Referințe bibliografice

Atenție: trebuie luate măsuri speciale atunci când se scrie codul pentru *handler*ele de semnal, deoarece acestea pot fi apelate asincron pe parcursul execuției programului, 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

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handlere* de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

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).

Observație importantă: 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 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 (*i.e.*, semnale care așteaptă să fie deblocate pentru a putea fi livrate procesului).

Demo: a se vedea programul `sig_ex4.c` ([2]).
(Pentru explicații mai detaliate, a se vedea [ForthDemo] prezentat în suportul de laborator #13).

Așteptarea unui semnal

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

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 primirea de semnale.

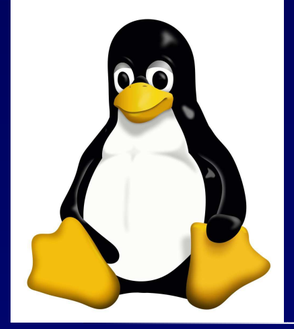
Se poate folosi în acest scop funcția `pause`, ce are prototipul următor:

```
int pause()
```

Efect: suspendă execuția 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.

Demo: a se vedea prima soluție dată în exercițiul rezolvat ['Ping-pong' pattern #2], prezentat în suportul online de laborator ([3]), care ilustrează un alt motiv de eroare: semnalul așteptat ar putea fi primit înaintea apelului `pause()`.



Așteptarea unui semnal (cont.)

Introducere

Semnale POSIX

Noțiuni generale despre semnale

Categorii de semnale

Primitiva `kill` pentru generarea unui semnal

Tipurile predefinite de semnale din standardul POSIX

Tratarea semnalelor

Primitiva `signal` pentru configurarea tratării semnalelor

Demo: exemple de cod

Definirea propriilor *handler*e de semnal

Blocarea semnalelor

Așteptarea unui semnal

Referințe bibliografice

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 următorul prototip:

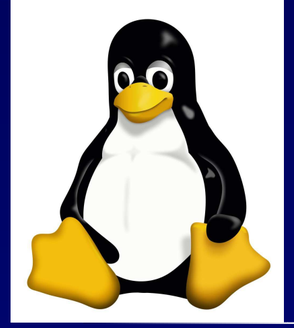
`int sigsuspend(const sigset_t *`*set*`)`

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 apelul funcției returnează, moment în care este reinstalată, în mod automat, vechea mască de semnale.

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`).

Demo: a se vedea programul `sig_ex5.c` ([2]), ce își suspendă execuția în așteptarea semnalului `SIGQUIT` (generat de tastele `CTRL+\`), fără a fi întrerupt de alte semnale.
(Pentru explicații mai detaliate, a se vedea [FifthDemo] prezentat în suportul de laborator #13).

Alt exemplu: a se vedea a doua soluție dată în exercițiul rezolvat ['Ping-pong' pattern #2], prezentat în suportul online de laborator ([3]).



Bibliografie obligatorie

Introducere

Semnale POSIX

Referințe bibliografice

[1] Cap. 4, §4.5 din cartea “*Sisteme de operare – manual pentru ID*”, autor C. Vidrașcu, editura UAIC, 2006. *Notă*: este accesibilă, în format PDF, din pagina disciplinei “Sisteme de operare”:

- https://edu.info.uaic.ro/sisteme-de-operare/SO/books/ManualID-SO.pdf

[2] Programele demonstrative amintite pe parcursul acestei prezentări pot fi descărcate de la:

- https://edu.info.uaic.ro/sisteme-de-operare/SO/lectures/Linux/demo/signals/

[3] Suportul de laborator online asociat acestei prezentări:

- https://edu.info.uaic.ro/sisteme-de-operare/SO/support-lessons/C/suport_lab13.html

Bibliografie suplimentară:

[4] Cap. 20, 21 și 22 din cartea “The Linux Programming Interface : A Linux and UNIX System Programming Handbook”, autor M. Kerrisk, editura No Starch Press, 2010.

- https://edu.info.uaic.ro/sisteme-de-operare/SO/books/TLPI1.pdf

[5] POSIX API : `man 2 kill` , `man 2 signal` , `man 2 sigaction` , ... ș.a.

[6] Sumar : `man 7 signal`