410 Signaux

INF3173
Principes des systèmes d'exploitation

Jean Privat

Université du Québec à Montréal

Hiver 2021

Signaux

Forme d'interruption logicielle

- Analogie avec les interruptions matérielles
- Permet d'expédier à un processus une information urgente

Comportement asynchrone

- Un signal est envoyé
- Il sera reçu et traité au moment opportun

Sémantique des signaux

Liste des signaux

- Les signaux sont catalogués
- La liste est fixée
- Chacun est documenté

→ `signal(7)`

Un gestionnaire de signaux par processus

- Chaque programme gère toutefois les signaux comme il veut
- La sémantique doit être documentée dans le programme (man)
- En particulier si elle diverge du catalogue

Exemples de signaux

SIGINT

- Ctrl C génère ce signal dont le comportement par défaut est d'arrêter le processus

SIGSEV

- Une erreur de segmentation provoque l'expédition de ce signal au processus fautif

kill

- Commande `kill(1)`. Envoie signal SIGTERM par défaut.
- Appel système `kill(2)`
- **Question** `kill` est souvent une commande interne du shell, pourquoi ?

Actions possibles pour un signal

Pour chaque catégorie de signal, un processus peut

- Accepter le comportement par défaut
  En général, arrêt du processus
- Ignorer le signal (pas tous)
- Gérer le signal (pas tous)

Gestion des permissions

- Seuls les processus d'un même utilisateur peuvent s'envoyer des signaux
  Et root (RTFM pour les détails)
- Pas de kill sur le processus du voisin

Actions possibles pour un signal

Quelques signaux

Signal	Valeur	Action	Description
SIGHUP	1	T	Le terminal se ferme
SIGINT	2	T	Ctrl C au clavier
SIGKILL	9	TD	terminer le processus
SIGSEGV	11	M	erreur de segmentation
SIGCHLD	-	I	terminaison d'un enfant

- Action par défaut : T=terminer, D=défaut obligatoire, M=image mémoire, I=ignorer

Gestion classique des signaux en deux étapes

Écrire la fonction gérante (en C classique)

- Signature simple `void foo(int sig)` (pour `sa_handler`)
- Ou complète `void bar(int sig, siginfo_t* info, void* uctx)` (pour `sa_sigaction`)

Associer fonction et signal

- `sigaction(2)`
- Structure `struct sigaction` un peu pénible
  `sigemptyset(3)` et cie.

Extra

- `pause(2)` suspend l'exécution jusqu'à un signal
- `strsignal(3)` et `psignal(3)` pour le texte des signaux

sigaction.c

```c
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<signal.h>
#include<string.h>
void gere(int sig) {
  printf("Reçu %d: %s\n", sig, strsignal(sig));
  exit(1);
}
int main(void) {
  struct sigaction action;
  sigemptyset(&action.sa_mask);
  action.sa_flags = 0;
  action.sa_handler = gere;
  sigaction(SIGINT, &action, NULL);
  sigaction(SIGTERM, &action, NULL);
  pause();
}
```

Informations supplémentaires

- Pour ignorer un signal, mettre `SIG_IGN` dans `sa_handler`
- Pour l'action par défaut, mettre `SIG_DFL` dans `sa_handler`
- Les signaux d'une même catégorie ne sont pas empilés
  Une rafale d'un même signal peut activer une seule invocation de la fonction gérante

Bloquer (masquer) les signaux

- Retarde la gestion de signaux jusqu'au déblocage
- `sigprocmask(2)` pour manipuler le masque de signaux bloqués
- `sa_mask` de `sigaction(2)` permet de masquer des signaux automatiquement pendant l'exécution de la gérante

Voir les signaux

- `/proc/PID/status` montre l'état des signaux « `Sig*` »
- `sigpending(2)` voir les signaux en attente

Threads, fork et exec

pthreads

- Partagent: les gérantes, les signaux ignorés
- Copie: les signaux bloqués (masque des signaux)
- Fonctionnalités fines existent `pthread_kill(3)`, `pthread_sigmask(3)`
- Certains signaux en attente peuvent être partagés ou pas

fork

- Hérite: les gérantes, les signaux ignorés et bloqués
- Vide: les signaux en attente

execve

- Préserve: les signaux ignorés, bloqués et en attente
- Vide: les gérantes

Interruption des appels systèmes

- Un signal reçu et géré peut interrompre certains appels système
- Appels système dits « interruptibles »
- Détail dans le man de chacun des appels système (ou `signal(7)`)
- Processus dans l'état POSIX « S » selon ps

Qu'est-ce qui se passe alors

1. Processus dans appel système
2. Signal attrapé ; gérante invoquée ; gérante terminée (`return`)
3. Appel système terminé de force (interrompu)
   - Retourne `EINTR` (si pas commencé)
     - Ou autre valeur si travail partiellement réalisé
   - Sauf si `SA_RESTART` dans `sa_flag` de `sigaction(2)`
     - Mais pas pour tous les appels système
   - RTFM pour les détails

Synchronisme

L'approche asynchrone de `sigaction(2)` a des défauts

POSIX

- `sigwaitinfo(2)`, `sigtimedwait(2)`, `sigsuspend(2)`, `sigwait(3)`
- Attend des signaux
- Note: Bloquer les signaux avant avec `sigprocmask(2)` ou autre

Linux

- `signalfd(2)`
- Crée un descripteur de fichier spécial
- Permet de gérer les signaux comme des évènements («tout est fichier»)

$\rightarrow$ Attendre un signal avec `poll(2)`, `select(2)`, etc.