Les signaux Unix

Principe

Un processus (ou le système) peut envoyer un signal à un autre processus (commande ou appel système *kill*)

Le processus destinataire réagit instantanément (il interrompt l'exécution de son programme, traite le signal, et éventuellement reprend son exécution)

Un signal ne transporte pas d'information autre que son numéro. Il existe quelques dizaines de signaux distincts, définis par le système (kill -l pour obtenir la liste)

Exemples de signaux

SIGKILL(9) Termine le processus autoritairement
SIGSTOP (19) Met le processus en attente (sommeil)
SIGCONT(18) Reprend l'exécution d'un processus endormi

Exemples de génération de signaux :

- Lorsqu'un fils se termine, un signal SIGCHLD est envoyé à son père.
  Cependant, le père ne sait pas lequel de ses fils s'est terminé.
- Lorsqu'un processus écrit dans un pipe sans lecteur, un signal SIGPIPE lui est envoyé.
- Lorsqu'un processus écrit à une adresse mémoire invalide, un signal SIGSEGV est généré
- Une division par 0 envoie le signal SIGFPE (Floating Point Exception)
- Certaines touches dans bash entrainent l'envoi d'un signal :
  - Ctrl+C Envoie SIGINT
  - Ctrl+\ Envoie SIGQUIT

Envoi d'un signal

- Commande kill (kill -9 pid ou kill -KILL pid)
- Appel système : `int kill (pid_t pid, int signum)`
  - si pid > 0, le signal est envoyé au processus #pid
  - si pid == 0, le signal est envoyé au processus courant et à tous ceux de son groupe
  - si pid == -1, le signal est envoyé à tous les processus sauf *init*.
  - si pid < -1, le signal est envoyé au groupe de processus dont le numéro est -pid.

Envoi d'un signal

- `alarm(unsigned int nb_sec)`

  permet de programmer l'envoi d'un signal SIGALRM après un délai donné (délai approximatif)

- `pause()`

  permet de bloquer le processus appelant jusqu'à réception d'un signal

Détournement d'un signal

```
int sigaction(int signum, struct sigaction *act,
              struct sigaction *oldact);
```

modifie l'action effectuée par un processus à la réception d'un signal `signum`.

La structure `act` est définie par quelque chose comme :

```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
};
```

`sa_handler` peut être égal à SIG_DFL (action par défaut), SIG_IGN (ignorer le signal), ou un pointeur sur une fonction de gestion de signaux (*handler*).

Blocage d'un signal

- Un processus peut bloquer un signal : le signal est reçu mais ne sera pas traité tant que celui-ci sera bloqué. Ceci permet d’effectuer certaines tâches de manière atomique (en étant garanti de ne pas être interrompu par un signal)

- Un signal envoyé mais non traité par le processus récepteur est dit *pendant (pending)*

- Si un même signal est envoyé plusieurs fois, il n'est mémorisé qu'une fois

Ensembles de signaux

- Type `sigset_t`
- `int sigemptyset (sigset_t *set);`
  Initialise la variable *set* à aucun signal
- `int sigfillset (sigset_t *set);`
  Initialise la variable *set* à tous les signaux
- `int sigaddset (sigset_t *set, int num);`
  ajoute le signal *num* à l'ensemble
- `int sigdelset (sigset_t *set, int num);`
  supprime le signal *num* de l'ensemble
- `int sigismember (sigset_t *set, int num);`
  Teste si le signal *num* est dans l'ensemble

Sigprocmask

```
int sigprocmask(int how, const sigset_t* set, sigset_t *old);
```

avec:

- **how** définit l'opération à effectuer :
  - **SIG_BLOCK** : tous les signaux de *set* seront bloqués (en plus de ceux qui l'étaient éventuellement déjà). Nouveau masque : set ∪ old
  - **SIG_UNBLOCK** : tous les signaux indiqués dans *set* seront débloqués. Ceux qui étaient bloqués mais qui ne sont pas indiqués dans *set* continuent d'être bloqués. Nouveau masque : old - set
  - **SIG_SETMASK** : *set* contient directement l'état de tous les signaux du système. Nouveau masque : set
- **set** est un ensemble de signaux sur lesquels appliquer l'action *how*.
- **old**, s'il ne vaut pas NULL, reçoit l'état précédent des signaux.

Structures de données du système

Chaque entrée dans la table des processus comporte, pour chaque signal :

- un bit indiquant si le signal a été reçu et reste à traiter
- un bit indiquant si le signal est bloqué
- une structure `sigaction` indiquant :

  le comportement à adopter (ignorer, défaut ou fonction),

  diverses informations concernant le traitement, par exemple, les signaux à bloquer pendant l'exécution du gestionnaire de signal (*handler*)

Signaux et processus

- Après `fork()`, le fils a un comportement vis-à-vis des signaux identique au comportement de son père

- Après `exec()`, les signaux ignorés continuent d'être ignorés, les autres signaux reprennent le comportement par défaut (la fonction qu'ils devaient exécuter n'existe plus).