Chapitre 4

Les signaux

Les signaux

- Même chose que signaux en shell ! (`kill`, `trap`)
- **But initial :** prévenir un processus qu'un évènement particulier est arrivé sur la machine / sur le processus
- Certains signaux sont envoyés par le noyau
- Mais on peut en envoyer en tant qu'utilisateur
- On ne peut envoyer des signaux que si on est propriétaire du processus visé
  - Sauf root bien sûr !

Gestion des signaux par un processus

Lorsque le processus $\mathcal{P}$ reçoit le signal $\mathcal{S}$:

si $\mathcal{S}$ *est masqué* **alors**
- Ajout de $\mathcal{S}$ à la liste des signaux arrivés et ignorés

sinon
- **si** $\mathcal{S}$ *n'est pas détourné par* $\mathcal{P}$ **alors**
  - Action de $\mathcal{S}$ par défaut
- **sinon**
  - $\rightarrow$ Arrêt de l'exécution normale de $\mathcal{P}$, mémorisation de la position dans le code
  - $\rightarrow$ Exécution de la fonction de détournement installée dans $\mathcal{P}$
  - $\rightarrow$ Reprise de l'exécution là où $\mathcal{P}$ s'était interrompu

Liste des signaux POSIX

Nom	Évènement associé	Défaut
`SIGHUP`	Terminaison du processus leader de session	1
`SIGINT`	Frappe du caractère *intr* (`^C`) sur le clavier du terminal de contrôle	1
`SIGQUIT`	Frappe du caractère *quit* (`^\` ou `^4`) sur le clavier	2
`SIGILL`	Détection d'une instruction illégale	2
`SIGABRT`	Terminaison anormale provoquée par l'exécution de la fonction `abort`	2
`SIGFPE`	Erreur arithmétique (division par 0,...)	2
`SIGKILL`	Signal de terminaison	1*
`SIGSEGV`	Violation mémoire	2
`SIGPIPE`	Écriture dans un tube sans lecteur	1
`SIGALRM`	Fin de temporisation (fonction `alarm`)	1
`SIGTERM`	Signal de terminaison	1
`SIGUSR1`	Signal émis par un processus utilisateur	1
`SIGUSR2`	Signal émis par un processus utilisateur	1
`SIGCHLD`	Terminaison d'un fils	3

Liste des signaux POSIX

Nom	Évènement associé	Défaut
SIGSTOP	Signal de suspension	4*
SIGTSTP	Frappe du caractère *susp* (^Z) sur le clavier	4
SIGCONT	Signal de continuation d'un processus stoppé	5
SIGTTIN	Lecture par un processus en arrière-plan	4
SIGTTOU	Écriture par un processus en arrière-plan	4
SIGIO	Avis d'arrivée de caractères à lire	1 ou 3
SIGURG	Avis d'arrivée de caractères urgents	3
SIGWINCH	Redimensionnement de fenêtre	3

1 Terminaison
2 Terminaison + core
3 Ignoré
4 Suspendu
5 Continuation
* Signal non trappable

Envoi de signaux : `kill(2)`

- ```
  #include <sys/types.h> /* pour pid_t */
  #include <signal.h>
  ```

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

- Envoie le signal `sig` au processus `pid`
- Le processus visé doit avoir le même UID que l'émetteur (sauf si l'émetteur a l'UID 0)

- En shell : `kill(1)`
  - `kill [ -signal | -s signal ] pid ...`
  - `kill -USR1 76431 76534`

Signaux et processus

- Une table de gestion des signaux (une par processus)
- Un comportement par défaut (souvent mourir)
- Possibilité de modifier le comportement du processus si réception d'un signal (trappage = piégeage = détournement)
- Certains signaux ne peuvent pas être piégés (exemple : `SIGKILL`)

N° signal	**Bit « appelé ? » (bit pending)**	**Bit masque**	**void (*traitement)()**
SIGQUIT	0	1	`meurt_proprement`
SIGALRM	1	0	`affiche_coucou`
SIGKILL	0	0	défaut
⋮	⋮	⋮	⋮

Affectation des masques de signaux

- Masque (*sigset*) : ensemble de signaux

- ```
  int sigemptyset(sigset_t *ens);
  /* ens = {} */
  ```

- ```
  int sigfillset(sigset_t *ens);
  /* ens = {1, 2, ..., NSIG} */
  ```

- ```
  int sigaddset(sigset_t *ens, int sig);
  /* ens = ens + {sig} */
  ```

- ```
  int sigdelset(sigset_t *ens, int sig);
  /* ens = ens - {sig} */
  ```

- Toutes ces fonctions renvoie 0 si OK, -1 sinon

Quelques utilisations

- `int sigismember(const sigset_t *set, int signum);`
  - Renvoie 1 si `signum` est dans `set`
  - Renvoie 0 si `signum` n'est pas dans `set`
  - Renvoie -1 si erreur

- `int sigsuspend(const sigset_t *mask);`
  - Remplace temporairement le masque par `mask` et endort le processus jusqu'à ce qu'un des signaux non ignorés arrive.
  - Renvoie -1 si erreur.

- `int sigpending(sigset_t *set);`
  - Écrit dans `set` la liste des signaux bloqués.
  - Renvoie 0 si OK, -1 si erreur

Bloquage des signaux

- `int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);`
- `sigprocmask` permet de bloquer (temporairement si l'on veut) des signaux
- Quand le masquage (= bloquage) est levé, les signaux éventuellement bloqués sont alors délivrés (ils n'ont pas disparu)
- `how` :
  - `SIG_SETMASK` : remplace `oldset` par `set`
  - `SIG_BLOCK` : bloque les signaux de `set` et ceux de `oldset`
  - `SIG_UNBLOCK` : débloque les signaux de `set`
- `oldset` : si non NULL, reçoit l'ancien masque de bloquage
- Renvoie 0 si OK, -1 sinon

Bloquage des signaux : exemple

```
sigset_t signaux ;
int sig;

/* signaux = {SIGQUIT, SIGUSR1} */
sigemptyset(&signaux);
sigaddset(&signaux, SIGQUIT);
sigaddset(&signaux, SIGUSR1);

/* Installation du masque */
sigprocmask(SIG_SETMASK, &signaux, NULL);

/* On joue la cible */
sleep(20);
printf("Signaux envoyes pendant le sleep : ");
sigpending(&signaux);
for (sig=1; sig<NSIG; sig++)
    if (sigismember(&signaux, sig))
        printf("%d ", sig);
puts("\n");

/* Deblocage des signaux */
sigemptyset(&signaux);
return sigprocmask(SIG_SETMASK, &signaux, NULL);
```

Pointeurs de fonctions en C

- Déclaration :

```
double (*maFonction)(double, int);
```

- Affectation (vérification du typage) :

```
double puissance(double d, int n)
{ /* code de "puissance" */ }

maFonction = puissance;
```

- Utilisation :

```
resultat = (*mafonction)(1.5, 2);
```

Exemple avec la fonction `atexit()`

- ```
  #include <stdlib.h>
  int atexit(void (*function)(void));
  ```
- `atexit(3)`
  - enregistre une fonction à appeler juste avant la fin d'un programme
  - `function` : argument de type « fonction sans argument qui ne renvoie rien » désignant la fonction à appeler
- Exemple :

```
void c_est_la_fin() {
  puts("Adieu monde cruel");
}

int main() {
  atexit(c_est_la_fin);
  puts("J'existe");
  return 0;
}
```
```

struct sigaction

- ```
  struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
  }
  ```
- `sa_handler` ou `sa_sigaction` : fonction à exécuter (ou `SIG_DFL` ou `SIG_IGN`)
- `sa_mask` : liste des signaux à bloquer pendant l'exécution du handler
- `sa_flags` (morceaux choisis) :
  - `SA_NOCLDSTOP` : ignorer `SIGCHLD` quand le fils meurt
  - `SA_RESETHAND` : remettre le handler par défaut (non POSIX)
  ```

Détournement : mise en place

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

- `signum` : numéro du signal à détourner
- Si `act` est non-nul : nouveau sigaction à mettre en place
- Si `oldact` est non-nul : écrit l'ancien sigaction dans `oldact`
- Renvoie 0 si OK, -1 sinon
  ```

Détournement : exemple

```
void coucou(int sig) {
    printf("Coucou, sig %d\n", sig);
}

int main() {
    struct sigaction action, save;

    sigemptyset(&action.sa_mask);
    action.sa_handler = coucou;
    action.sa_flags = 0;

    if (sigaction(SIGQUIT, &action, &save)) {
        perror("Installation coucou"); exit(1);
    }

    sleep(20);
    printf("Fin pause\n");
    if(sigaction(SIGQUIT,&save,NULL)) {
        perror("Restauration signaux"); exit(1);
    }

    return 0;
}
```