Signaux

François Trahay

Contents

Signaux

Rappel (CSC3102)

- Signal: mécanisme de communication inter-processus
- Message: un entier entre 1 et 31
- Ordre de réception aléatoire (différent de l'ordre d'émission)
- Une routine de réception est automatiquement invoquée chez le récepteur dès que le signal arrive

---

Envoyer un signal

- `int kill(pid_t pid, int sig);`
  - Envoie le signal `sig` au processus `pid`
  - Quelle valeur pour `sig`?
    - valeur entière (par ex: `9`): pas portable (dépend de l'OS)
    - constante (par ex: `SIGKILL`) définie dans `signal.h`

Voici un exemple de programme utilisant la fonction `kill`:

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>

int main(int argc, char**argv) {
  if(argc != 2) {
    fprintf(stderr, "usage: %s PID\n", argv[0]);
    return EXIT_FAILURE;
  }

  pid_t pid = atoi(argv[1]);
  int signo = SIGKILL;

  printf("Sending signal %d to %d\n", signo, pid);
  kill(pid, signo);

  return EXIT_SUCCESS;
}
```

Recevoir un signal

- `int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`
  - Spécifie le comportant lors de la réception du signal `signum`
  - `struct sigaction` est une structure de la forme:

```c
struct sigaction {
        void     (*sa_handler)(int); // pointeur sur la fonction à appeler
        void     (*sa_sigaction)(int, siginfo_t *, void *);
        sigset_t   sa_mask;
        int        sa_flags;
        void     (*sa_restorer)(void);
};
```

Signaux interceptables

Il est possible d'utiliser `sigaction` pour "intercepter" tout signal sauf les signaux `SIGKILL` et `SIGSTOP`

`struct sigaction`

La valeur prise par `sa_handler` est: * l'adresse d'une fonction (par ex: `void signal_handler(int signo)` * Le paramètre `signo` est le numéro du signal

reçu * la valeur `SIG_DFL` pour restaurer l'action par défaut (tuer le processus) * la valeur `SIG_IGN` pour ignorer le signal: à la réception de ce signal, aucune action ne sera effectuée

Sauf cas d'usages particuliers, les autres champs de la structure `sigaction` sont à mettre à `0`.

`oldact`

La fonction `sigaction` modifie le comportant du processus lorsqu'il reçoit le signal `signum`. Si `oldact` n'est pas `NULL`, l'ancien comportement y est stocké.

Variables globales

Si la fonction traitant le signal manipule des variables globales, il est conseillé de les déclarer `volatile`. Par exemple:

```
volatile int var;
```

Lorsqu'une variable est déclarée `volatile`, le compilateur limite les optimisations faites sur cette variable. Par exemple, le compilateur ne met pas en cache (dans un registre) la variable.

Si une fonction (par exemple `foo`) qui manipule la variable `var` non `volatile` est interrompue par un traitant de signal (`sig_handler`) modifiant `var`, la modification de la variable risque de ne pas être "vue" par `foo` qui travaille sur une copie en cache de la variable. La fonction `foo` risque donc de travailler sur une version obsolète de la variable.

Exemple

Voici un exemple de programme utilisant `sigaction` pour intercepter un signal:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/types.h>

/* Function to call upon signal reception */
void signal_handler(int signo) {
  printf("Received: signal %d\n", signo);
}

int main(int argc, char**argv) {
  if(argc != 2) {
    fprintf(stderr, "usage: %s signo\n", argv[0]);
    return EXIT_FAILURE;
  }
```

```c
  /* Initialize the sigaction structure */
  int signo = atoi(argv[1]);
  struct sigaction s;
  s.sa_handler = signal_handler;

  /* Install the signal handler */
  printf("Installing signal handler for signal %d\n", signo);
  int retval = sigaction(signo, &s, NULL);
  if(retval<0) {
    perror("sigaction failed");
    abort();
  }

  /* Wait to receive signals */
  while(1) {
    printf("[%d] Sleeping...\n", getpid());
    sleep(1);
  }

  return EXIT_SUCCESS;
}
```

Attendre un signal

- `int pause( );`
  - Attend qu'un signal (non ignoré) soit reçu

Programmer une alarme

- `int alarm(unsigned int s);`
  - Programme l'envoi de `SIGALRM` après `s` secondes

Pour programmer une alarme avec une granularité plus fine, utilisez la fonction `setitimer`. Cette fonction permet de programmer des alarmes périodiques (qui se répètent) avec une granularité de l'ordre de la microseconde.