Programmation Système

Les signaux

Emmanuel Bouthenot (emmanuel.bouthenot@u-bordeaux.fr)

Licence professionnelle ADSILLH - Université de Bordeaux

2020/2021

Les Signaux

Définition

- Un signal est la notification à un processus qu'un evénement a eu lieu
- Les signaux (également appelés *interruptions logicielles*) sont asynchrones et on ne peut pas prédire à quel moment ils arriveront.

Description des signaux

Les signaux sont utilisés pour gérer plusieurs types d'événements :

- Généré par l'action d'un utilisateur: **Ctrl-C** (**SIGINT**), **Ctrl-Z** (**SIGSTOP**), etc.
- Fautes matérielles : division par 0 (**SIGFPE**), référence mémoire invalide (**SIGSEGV**), écriture mémoire erronée (**SIGBUS**), etc.
- Fautes logicielles : erreur d'écriture dans un tube (**SIGPIPE**), notification urgente de données disponibles (**SIGURG**), alarme (**SIGALRM**), etc.
- Envoi de signaux direct entre processus (`man 2 kill`)
- Envoi de signaux par l'utilisateur (`man 1 kill`)

Traitement des signaux

Un processus se déclare à l'écoute d'un type d'événement (signal) en initialisant un gestionnaire qui sera appelé dès que l'événement visé aura lieu.

Quand il a lieu le gestionnaire est appelé avec tous les informations associés au signal.

L'exécution normal du programme reprend là où elle s'était arrêtée à la fin de l'appel du gestionnaire de l'événement.

Comportements associés aux signaux

- **Term** : Par défaut, terminer le processus
- **Ign** : Par défaut, ignorer le signal
- **Core** : Par défaut, créer un fichier core et terminer le processus (cf `core(5)`)
- **Stop** : Par défaut, arrêter le processus
- **Cont** : Par défaut, continuer le processus s'il est actuellement arrêté

Liste des signaux (POSIX.1-1990 original)

Signal	Valeur	Action	Commentaire
SIGHUP	1	Term	Déconnexion détectée sur le terminal de contrôle ou mort du processus de contrôle
SIGINT	2	Term	Interruption depuis le clavier (`Ctrl-C`)
SIGQUIT	3	Core	Demande "Quitter" depuis le clavier (`Ctrl-\`)
SIGILL	4	Core	Instruction illégale
SIGABRT	6	Core	Signal d'arrêt depuis `abort(3)`
SIGFPE	8	Core	Erreur mathématique virgule flottante
SIGKILL	9	Term	Signal "KILL"
SIGSEGV	11	Core	Référence mémoire invalide
SIGPIPE	13	Term	Écriture dans un tube sans lecteur
SIGALRM	14	Term	Temporisation `alarm(2)` écoulée
SIGTERM	15	Term	Signal de fin
SIGUSR1	10	Term	Signal utilisateur 1
SIGUSR2	12	Term	Signal utilisateur 2
SIGCHLD	17	Ign	Fils arrêté ou terminé
SIGCONT	18	Cont	Continuer si arrêté
SIGSTOP	19	Stop	Arrêt du processus
SIGTSTP	20	Stop	Stop invoqué depuis le terminal (`Ctrl-Z`)
SIGTTIN	21	Stop	Lecture sur le terminal en arrière-plan
SIGTTOU	22	Stop	Écriture dans le terminal en arrière-plan

Liste des signaux (SUSv2 et POSIX.1-2001)

Les signaux SIGKILL et SIGSTOP ne peuvent être ni capturés ni ignorés

Signal	Valeur	Action	Commentaire
SIGBUS	7	Core	Erreur de bus (mauvais accès mémoire)
SIGPOLL		Term	Événement "pollable", synonyme de SIGIO
SIGPROF	27	Term	Expiration de la temporisation pour le suivi
SIGSYS	31	Core	Mauvais argument de fonction
SIGTRAP	5	Core	Point d'arrêt rencontré
SIGURG	23	Ign	Condition urgente sur socket
SIGVTALRM	26	Term	Alarme virtuelle
SIGXCPU	24	Core	Limite de temps CPU dépassée
SIGXFSZ	25	Core	Taille de fichier excessive
SIGWINCH	28	Ign	Fenêtre redimensionnée

Core dump

Définition

Un core dump est une image de la mémoire d'un processus prise au moment d'un plantage (*crash*).

Des informations utiles à l'analyse du plantage sont disponibles dans les fichiers *core* :

- Copie de la mémoire au moment du plantage
- Status de terminaison du processus
- Copie des registres du processeur

```
> help ulimit
> ulimit -c
0
> ulimit -c unlimited # Création de fichiers core sans limite
```

Analyse d'un core dump

```
int main () {
    int a = 10;
    int b = 0;
    a = a / b;
}
```

```
> gcc -g -Wall sigfpe.c -o sigfpe
> ./sigfpe
> [2]    8112 floating point exception (core dumped)  ./sigfpe
> gdb sigfpe core
GNU gdb (Debian 7.11.1-2) 7.11.1
Core was generated by `./sigfpe'.
Program terminated with signal SIGFPE, Arithmetic exception.
#0  0x00000000004004ec in main () at sigfpe.c:4
4           a = a / b;
(gdb) backtrace full
#0  0x00000000004004ec in main () at sigfpe.c:4
    a = 10
    b = 0
(gdb)
```

signal

La gestion des signaux revient à changer les comportement par défaut pour les comportement souhaités pour un processus donné.

```c
#include <signal.h>

typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);

// En cas de succès, renvoi la valeur du gestionnaire
// de signal précédent. En cas d'échec SIG_ERR est renvoyé
```

- signo est le nom ou numéro du signal à modifier
- handler vaut :
  - **SIG_IGN** pour ignorer le signal
  - **SIG_DFL** pour réinitialiser le signal a sa valeur par défaut
  - **pointeur** vers une fonction de traitement du signal reçu

signal (Python)

```
import signal

signal.signal(signalnum, handler)
```

- signo est le nom du signal à modifier (`signal.SIGUSR1`, `signal.SIGSTOP`, etc.)
- handler vaut :
  - **signal.SIG_IGN** pour ignorer le signal
  - **signal.SIG_DFL** pour réinitialiser le signal a sa valeur par défaut
  - une fonction de traitement du signal reçu qui prend 2 arguments en paramètres : `signum` and `frame`
    - signum : le numéro du signal
    - frame : object stack frame

Exemple de gestion de signal

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

void sigusr1_trigger() {
    write(1, "SIGUSR1 received\n", 17);
}

int main () {
    if (signal(SIGUSR1, sigusr1_trigger) == SIG_ERR) {
        perror("Unable to catch SIGUSR1\n");
    }
    else {
        printf("SIGUSR1 is catched on process with PID=%d\n", getpid());
    }
    for(;;) {
        sleep(10);
    }
}
```

Exemple de gestion de signal (Python)

```
#!/usr/bin/python3

import os, signal, time

def sigusr1_trigger(signum, frame):
    print("SIGUSR1 received")

def main():
    try:
        signal.signal(signal.SIGUSR1, sigusr1_trigger)
    except (ValueError, AttributeError) as e:
        print("Unable to catch SIGUSR1")
    else:
        print("SIGUSR1 is catched on process with PID=%d" % (os.getpid()))

    while True:
        time.sleep(10)

main()
```

Exemple de gestion de signal - demo

```
# Terminal 1
> ./sigusr1
SIGUSR1 is catched on process with PID=10333

# Terminal 2
> kill -USR1 10333

# Terminal 1
SIGUSR1 received
```

Envoyer des signaux

Des signaux peuvent êtes envoyés à d'autres processus avec l'appel `kill` ou au processus en cours avec `raise`.

```c
#include <signal.h>

int kill(pid_t pid, int sig);
int raise(int sig);
// Renvoi 0 en cas de succès et -1 en cas d'erreur

int pause(void);
// Bloque un processus jusqu'à ce qu'un signal lui soit
// envoyé et traité
// Retourne -1 avec errno positionné à EINTR
```

Poser une alarme

```c
#include <unistd.h>

unsigned int alarm(unsigned int seconds);
// 0 (pas d'alarme programmée) ou le nombre de secondes
// depuis la précédente alarme programmée
```

Un processus peut définir une alarme qui aboutira par l'envoi du signal **SIGALRM** au bout du temps programmé.

- On ne peut définir qu'une alarme par processus
- `alarm(0)` annule l'alarme

Exercices

- **Question 1.** Écrivez un programme en Python qui exécute une boucle infinie et qui affiche le compteur de boucle lorsque qu'on appuie sur `Ctrl-C`.
- **Question 2.** Écrire un programme en Python qui crée deux processus fils dont chacun fait une pause de 60 secondes. Lorsque le père reçoit le signal `SIGTERM` il envoie le même signal à tous ces fils.

Exercices (suite)

- **Question 3.** Écrire un programme en Python qui crée un processus fils, a chaque fois que le processus père reçoit le signal `SIGUSR1` il affiche " Ping ", attend une seconde puis envoi le signal `SIGUSR2` au fils. De même le fils, dès qu'il reçoit le signal `SIGUSR2`, il affiche " Pong ", attend une seconde et envoi le signal `SIGUSR1` au père.

  Que remarquez vous ?

Bibliographie

[**APUE**] Advanced Programming in the UNIX Environment.
W. Richard Stevens and Stephen A. Rago.
Addison-Wesley Professional, 2005.

[**TLPI**] The Linux Programming Interface.
Michael Kerisk.
No Starch Press, 2010.