Programmation Système et Réseau

Communication Inter-Processus (IPC)

2 - Les Signaux

Mohamed Maachaoui

Juan Ángel Lorenzo del Castillo

Seytkamal Medetov

Son Vu

CY Tech

ING2 - GSI

2020-2021

mohamed.maachaoui@cyu.fr

Table of Contents

Table of Contents

Introduction

- Les processus ne sont pas des entités indépendantes. Ils doivent partager les ressources de l'ordinateur.
- Dans certains cas, ils doivent communiquer entre eux pour se synchroniser ou pour communiquer de l'information.
- Il existe des nombreuses façons de communiquer. Dans ce chapitre, nous allons nous intéresser aux **signaux**.

Définitions

- Un signal est une information atomique envoyée :
  - D'un processus à un autre, à un groupe de processus, ou à lui même.
  - Du noyau du SE à un processus.
- Lorsqu'un processus reçoit un signal, le système d'exploitation l'informe : "*Tu as reçu un signal*" sans plus.
  - Exemple : le signal `SIGCHLD` permet au noyau d'avertir au processus père de la mort de son fils.
- Un signal ne transporte aucune autre information utile (forme de communication sans transport de données).
- Le processus pourra alors mettre en oeuvre une réponse décidée et pré-définie à l'avance (**`handler`**)

Définitions

- Les signaux sont un mécanisme asynchrone de communication inter-processus.
- Les signaux sont des interruptions logicielle.
- Il est assimilable à une sonnerie indiquant des événements différents pouvant donner lieu à une réaction.
- Chaque signal correspond à un événement spécifique.
- Rappel :
  - Interruption matérielle (IRQ) : traitement synchrone
  - Interruption logicielle : traitement asynchrone

Définitions

Ce mécanisme est implanté par un **moniteur**, qui scrute en permanence l'occurrence des signaux. C'est par ce mécanisme que le système communique avec les processus utilisateurs :

- Provenance interne en cas d'erreur du processus
  - violation mémoire
  - erreur d'E/S
  - segmentation fault (*core dumped*)
- à la demande de l'utilisateur lui-même via le clavier, par exemple lorsque vous tapez la commande kill ou vous appuyez sur CTRL-C.
- pour la déconnection de la ligne/terminal (provenance externe).

Définitions

- Un signal est identifié par un numéro entier et un nom symbolique décrit dans “`/usr/include/signal.h`”.
- Il existe 64 signaux différents numérotés ; ces signaux portent également des noms «normalisés» :
  - 0 : seul signal qui n’a pas de nom
  - 1 à 31 : signaux classiques
  - 32 à 63 : signaux “temps reels” (selon configuration de l’OS)
- Ces codes peuvent être trouvés dans “`/usr/include/signal.h`” ou avec la commande shell : `$ kill -l`

Table of Contents

Quelques signaux (I)

- **SIGHUP (1)** terminaison du processus leader de la session (CTRL-D interruption clavier).
- **SIGINT (2)** signal d'interruption (exemple déclenché par CTRL-C)
- **SIGKILL (9)** signal de terminaison d'un processus (non déroutable).
- **SIGILL (4)** Instruction illégale.
- **SIGFPE (8)** Erreur arithmétique.
- **SIGUSR1 (10)** Signal 1 défini par l'utilisateur.
- **SIGSEGV (11)** Adressage mémoire invalide.
- **SIGUSR2 (12)** Signal 2 défini par l'utilisateur.
- **SIGPIPE (13)** Écriture sur un tube sans lecteur.
- **SIGALRM (14)** Permet de gérer un timer (Alarme).
- **SIGTERM (15)** signal de terminaison, il est envoyé à tous les processus actifs par le programme shutdown, qui permet d'arrêter proprement un système UNIX. Terminaison normale.

Quelques signaux (II)

- **SIGCHLD (17)** Reveille le processus dont le fils vient de mourir.
- **SIGCONT (18)** Reprise du processus (non déroutable).
- **SIGSTOP (19)** Suspension du processus (non déroutable).
- **SIGTSTP (20)** Émission vers le terminal du caractère de suspension (“CTRL Z”).
- **SIGTTIN (21)** Lecture du terminal pour un processus d’arrière-plan.
- **SIGTTOU (22)** Écriture vers le terminal pour un processus d’arrière-plan.

Quelques signaux (III)

Actions par défaut	Nom du signal
Fin du process	SIGHUP, SIGINT, SIGBUS, SIGKILL, SIGUSR1, SIGUSR2, SIGPIPE, SIGALRM, SIGTERM, SIGSTKFLT, SIGXCOU, SIGXFSZ, SIGVTALRM, SIGPROF, SIGIO, SIGPOLL, SIGPWR, SIGUNUSED
Fin du process et création `core`	SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGIOT, SIGFPE, SIGSEGV
Signal ignoré	SIGCHLD, SIGURG, SIGWINCH
Processus stoppé	SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU
Processus redémarré	SIGCONT

- **Fichier core :** C'est un fichier disque qui contient l'image mémoire du processus au moment où il s'est terminé. Cette image peut être utilisé dans un débogueur pour étudier l'état du programme au moment où il a été terminé.

Origine des signaux

- Causes internes au processus
  - Erreur d'adressage $\longrightarrow$ `SIGSEVG` (segmentation violation)
  - Division par zero $\longrightarrow$ `SIGFPE` (Floating Point Exception)
- Terminal : grâce aux caractères spéciaux
  - Intr $\longrightarrow$ SIGINT "CTRL C" (interruption)
  - Quit $\longrightarrow$ SIGQUIT "CTRL $\backslash$ "
- Déconnection du terminal
  - Envoie à l'ensemble des processus de son groupe $\longrightarrow$ SIGHUP. Hangup=décrochage (fin de session)

Envoi d'un signal

- La primitive `kill` permet au systeme d'envoyer un signal à un processus :

```
#include <sys/types.h>
#include <signal.h>

int kill(pid_t pid, int signal)
```

  - la primitive renvoie **0** en cas de succès, **-1** sinon
  - **signal** est un numéro compris entre 1 et NSIG (défini dans `<signal.h>`).
  - `pid` numero du processus destinataire du signal.

< -1	: tous les processus du groupe `|pid|`
-1	: **Si l'émetteur est root :** tous les processus du système (sauf les procesus système : 0 et 1). **Sinon :** tous les processus dont leur *real user ID* est égal au *effective user ID* du processus émetteur.
0	: tous les processus dans le même groupe que le processus émetteur
> 0	: processus du pid indiqué

Envoi d'un signal

Remarques :

1. La fonction `raise(int signal)` est un raccourci pour `kill(getpid(), signal)` qui permet à un processus de s'envoyer un signal à lui-même.
2. Un processus ne peut envoyer un signal qu'à un processus de même propriétaire.
3. La commande `kill` peut être utilisée pour envoyer des signaux. Cette commande capture ses arguments en ligne et exécute un appel système `kill()`.
4. Si `signal` est 0, kill n'envoie pas de signal, mais elle fait toujours une vérification d'erreur. Cela est utile pour vérifier l'existence d'un processus ID ou un groupe de processus ID.

Envoi d'un signal

Exemple :

```
int main() {
  pid_t pid; int statut;
  printf("Lancement du processus %d\n", getpid());
  switch (pid = fork()) {
  case -1: exit(1);
  case 0: while(1) sleep(1); exit(1);
  default:
    printf("Processus fils %d cree\n", pid); sleep(10);
    if ( kill(pid,0) == -1 ) printf("fils %d inaccessible\n", pid);
    else {
      printf("Envoi du signal SIGUSR1 au processus %d\n", pid);
      kill(pid, SIGUSR1);
    }
    pid = waitpid(pid, &statut, 0);
    printf("Statut final du fils %d : %d\n", pid, statut); } }
```

Envoi d'un signal

La commande shell `kill` :

Pour envoyer un signal à un processus, on utilise la commande appelée kill. Celle-ci prend en option le numéro du signal à envoyer et en argument le numéro du (ou des) processus destinataire(s).

```
kill [—options] pid_processus
```

Exemples :

1. $ **kill 36** : par défaut le signal 15 (`SIGTERM`) est envoyé au processus de pid 36.
2. $ **kill 0** : Envoie le signal 15 à tous les processus fils, petits-fils... tous ceux lancés depuis ce terminal.
3. $ **kill -9 36** : Envoie le signal de numéro 9 (`SIGKILL`) au processus de pid 36.
4. $ **kill -SIGKILL 36** : Envoie le signal `SIGKILL` au processus de pid 36.

Comportements possibles du processus

- Un *handler* définit le comportement par défaut du processus ou la procédure à exécuter à la réception du signal donné.
- À chaque type de signal est associé à un *handler* par défaut `SIG_DFL`.
- Les différents comportements gérés par ce handler sont :
  - terminaison du processus, avec ou sans une image mémoire (fichier core),
  - rien : signal ignoré, (SIGKILL et SIGSTOP ne peuvent être ignorés)
  - suspension (SIGSTOP) du processus (le père est prévenu),
  - continuation (SIGCONT) : reprise du processus stoppé et ignoré sinon.

Comportements par défaut des signaux

- Rien :
  - SIGCHLD (Terminaison d'un processus fils),
  - SIGPWR,
  - SIGCONT...
- Fin :
  - SIGHUP(Fin de session),
  - SIGINT,
  - SIGKILL...
- Génération d'une image mémoire (CORE) :
  - SIGQUIT,
  - SIGILL (Instruction illégale),
  - SIGSEGV (Violation de mémoire)...
- Arrêt :
  - SIGSTOP,
  - SIGSTP (Demande de suspension depuis le terminal)...

Détournement (ou déroutement) d'un signal

- Pour certains signaux, on peut détourner l'action par défaut.
- Le caractère non modifiable de certains signaux assure la stabilité du système (SIGKILL, SIGCONT, SIGSTOP).

L'association signaux/handlers

- Un **handler** est une fonction qui décrit la suite des instructions à effectuer lors de la reception d'un signal.
- Tout processus peut installer pour les autres signaux un nouveau *handler*.
- Il existe deux interfaces de manipulation permettant l'installation d'un handler :
  - L'une, historique (ATT) et **rendu obsolète** est simplifiée ("**signal()**"), mais avec un comportement incertain.
  - ("**sigaction()**"), POSIX, plus complexe que la première garantit un comportement plus sûr et des programmes plus portables.

L'association `Sigaction`

- Sigaction permet de determiner ou de modifier l'action associée à un signal particulier.

```
int sigaction ( int num_sig, // le signal a dérouter
                const struct sigaction *nouv_action,
                struct sigaction *anc_action);
```

- Si le pointeur `nouv_action` est différent de NULL, alors le système modifie l'action du signal `num_sig` avec celle de `nouv_action`.
- Si le pointeur `anc_action` est différent de NULL, alors le système sauvegarde sur `anc_action` l'action précédente qui était prévue pour `num_sig`.
- Si les pointeurs `nouv_action` et `anc_action` sont NULL, l'appel système teste uniquement la validité du signal.
- La valeur renvoyée par `sigaction()` est :
  - 0 si tout s'est bien passé
  - -1 si une erreur est survenue, l'appel à sigaction() est ignorée

L'association `Sigaction`

La structure `sigaction` :

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

- Certaines architectures emploient une union. Il ne faut pas utiliser ou remplir simultanément `sa_handler` et `sa_sigaction`.

L'association `Sigaction`

La structure `sigaction` :

- **void (*sa_handler)(int)** est le handler (l'action) associé au signal `num_sig`, qui peut être :
  - Un pointeur vers une fonction de traitement du signal. Cette fonction reçoit le numéro de signal comme seul argument.
  - SIG_IGN pour ignorer le signal
  - SIG_DFL pour restaurer la réaction par défaut
- **void (*sa_sigaction)(int, siginfo_t *, void *)** : Pointeur sur une structure de type sigaction. Si le flag `SA_SIGINFO` est indiqué en `sa_flags`, `sa_sigaction` spécifiera le handler pour `num_sig` (à la place de `sa_handler`). Cette fonction reçoit le numéro de signal comme premier argument, un pointeur vers siginfo_t comme second argument et un troisième argument qui normalement n'est pas utilisé.
- **sa_mask** est un ensemble/vecteur de signaux qui seront bloqués avec celui passé à l'appel de sigaction(), lors de l'exécution du handler associé
- **sa_flags** permet de passer des drapeaux, et indique les options liées à la gestion du signal.
- L'élément **sa_restorer** est obsolète et ne doit pas être utilisé, POSIX ne mentionne pas de membre **sa_restorer**.

L'association Sigaction

La structure siginfo_t :

```
siginfo_t {
   int      si_signo;     /* Signal number */
   int      si_errno;     /* An errno value */
   int      si_code;      /* Signal code */
   int      si_trapno;    /* Trap number that caused hw-generated signal */
   pid_t    si_pid;       /* Sending process ID */
   uid_t    si_uid;       /* Real user ID of sending process */
   int      si_status;    /* Exit value or signal */
   clock_t  si_utime;     /* User time consumed */
   clock_t  si_stime;     /* System time consumed */
   sigval_t si_value;     /* Signal value */
   int      si_int;       /* POSIX.1b signal */
   void    *si_ptr;       /* POSIX.1b signal */
   int      si_overrun;   /* Timer overrun count; POSIX.1b timers */
   int      si_timerid;   /* Timer ID; POSIX.1b timers */
   void    *si_addr;      /* Memory location which caused fault */
   long     si_band;      /* Band event */
   int      si_fd;        /* File descriptor */
   short    si_addr_lsb;  /* Least significant bit of address  */
   void    *si_call_addr; /* Address of system call instruction  */
   int      si_syscall;   /* Number of attempted system call */
   unsigned int si_arch;  /* Architecture of attempted system call  */
  }
```

L'association Sigaction

Manipulation de sa_mask :

Les fonctions suivantes permettent de manipuler la masque sa_mask

- **int sigemptyset (sigset_t * ens_signaux)** : vide l'ensemble des signaux du masque
- **int sigaddset (sigset_t * ens_signaux, int num_sig) :** ajout d'un signal au masque
- **int sigdelset (sigset_t * ens_signaux, int num_sig)** : retire un signal du masque
- **int sigprocmask (int action, const sigset_t ens_signaux, sigset_t *ancien_signaux)** : bloque ou débloque l'ensemble des signaux du masque
- **int sigismember(sigset_t * ens_signaux, int num_sig)** : voir si le signal appartient à ens_signaux

Signaux et héritage

- Un fils n'hérite pas des signaux pendants du père, mais bien des associations signaux-handler faites par le père.
- Lors d'un `fork()`, suivi par un `exec()` dans le fils, toutes les associations signaux-handler sont réinitialisées dans le fils avec les handlers par défaut.

Exemple `sigaction()`

```c
#include <stdio.h>  /* Example of using sigaction() to setup */
#include <unistd.h> /* a signal handler with 3 arguments including siginfo_t. */
#include <signal.h>
#include <string.h>

static void hdl (int sig, siginfo_t *siginfo, void *context){
        printf ("Sending PID: %d, UID: %d\n", (long)siginfo->si_pid, (long)siginfo->si_uid);
}

int main (int argc, char *argv[]){
        struct sigaction act;

        memset (&act, '\0', sizeof(act));

        /* Use the sa_sigaction field because the handles has two additional parameters */
        act.sa_sigaction = &hdl;

        /* The SA_SIGINFO flag tells sigaction() to use the sa_sigaction field, not sa_handler. */
        act.sa_flags = SA_SIGINFO;

        if (sigaction(SIGTERM, &act, NULL) < 0) {
                perror ("sigaction");
                return 1;
        }

        while (1)
                sleep (10);

        return 0;
}
```

Source : https://www.linuxprogrammingblog.com/code-examples/sigaction

Exemples

Exemple de blockage des signaux avec sigprocmask() :

- https://www.linuxprogrammingblog.com/code-examples/blocking-signals-with-sigprocmask

Exemple d'addition d'un signal :

- https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_71/apis/sigaset.htm

Exemples (II)

Ignorer tous les signaux sauf SIGQUIT :

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

void main(void)
{
   int i;
   struct sigaction action;
   action.sa_handler = SIG_IGN; //ignore le signal

   for(i=1;i<NSIG;i++)
         sigaction(i, &action, NULL);
   action.sa_handler = SIG_DFL;  //remise a la valeur
                                 //par defaut du signal
   sigaction(SIGQUIT, &action, NULL);
}
```