



Systeme d'exploitation avancé

Signaux UNIX

Pierre LEROY – leroy.pierre1@gmail.com

Sommaire

- I. Principes
 - II. Mise en œuvre
 - III. Essentiel
 - IV. Conclusion
-

Principes

- Forme **simplifiée** de communication / dialogue entre processus :

NOTIONS

- Un processus a la possibilité d'envoyer un signal à un autre processus
 - ✓ Réalisable la commande **kill** ou un appel système
- Un signal transporte uniquement son numéro
 - ✓ Aucune autre métadonnée n'est disponible ni véhiculée
 - ✓ Le listing des signaux est obtenu via la commande `kill -l`
- Le processus destinataire réagit de façon asynchrone et réalise la séquence suivante:
 - ✓ Interruption du traitement courant
 - ✓ Traitement du signal
 - ✓ Reprise du contexte nominal

Exemple

- Signaux implicites rencontrés en utilisation nominale :

SIGNAUX USUELS

- Les quatre identifiants les plus connus sont :
 - ✓ **SIGKILL(9)** Termine le processus autoritairement.
 - ✓ **SIGTERM(15)** Termine le processus nominalelement.
 - ✓ **SIGSTOP(19)** Met le processus en attente (sommeil).
 - ✓ **SIGCONT(18)** Reprend l'exécution d'un processus endormi.
- Des signaux sont générés implicitement lors de l'utilisation de l'OS :
 - ✓ **SIGCHLD(20)** est envoyé au processus père lorsqu'un de ses fils se termine.
 - ✓ **SIGPIPE(13)** est envoyé au processus écrivain lorsqu'il écrit dans un pipe sans lecteur.
 - ✓ **SIGSEGV(11)** est généré lorsqu'un processus écrit à une adresse mémoire invalide.
 - ✓ **SIGFPE(8)** est généré lors d'une division par 0.

Certaines séquences de touches dans bash provoque l'envoi d'un signal :

- ✓ – **Ctrl+C** envoie **SIGINT(2)**.
- ✓ – **Ctrl+** envoie **SIGQUIT(3)**.

Sommaire

- I. Principes
 - II. Mise en œuvre
 - III. Essentiel
 - IV. Conclusion
-

Manipulations

- Il existe plusieurs manipulations réalisables sur les signaux :

LISTING

- Un processus a la possibilité de réaliser les actions suivantes :
 - ✓ Emettre un signal vers un autre processus.
 - ✓ Détourner un signal reçu d'un autre processus.
 - ✓ Inhiber un signal perçu d'un autre processus.

Envoi d'un signal

- Manipulation d'émission d'un signal :

MISE EN OEUVRE

- Il existe trois possibilités pour émettre un signal vers un processus
 - ✓ Une séquence de touches au sein d'un **shell**
 - ✓ Via la commande **kill** : `kill -SIGNUMBER -PID`
 - ✓ Via un appel système **pause** / **alarm** / **kill**

Envoie d'un signal

- Manipulation d'émission d'un signal :

MISE EN OEUVRE

- Appel système *kill* :

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

↑ Destinataire

↑ Numéro du signal

- ✓ $\text{pid} > 0$: signal est envoyé au processus #pid.
- ✓ $\text{pid} == 0$: signal est envoyé au processus courant et à tous ceux de son groupe.
- ✓ $\text{pid} == -1$: signal est envoyé à tous les processus sauf init.
- ✓ $\text{pid} < -1$: signal est envoyé au groupe de processus dont le numéro est -pid.

Envoie d'un signal

- Manipulation d'émission d'un signal :

MISE EN OEUVRE

- Appel système ***alarm***:

✓ Programme l'envoi d'un signal SIGALRM après un délai donné (délai approximatif).

```
alarm(unsigned int nb_sec)
```

↑ Délais avant déclenchement

- Appel système ***pause***:

✓ Programme le blocage d'un processus appelant jusqu'à réception d'un signal.

```
pause()
```

Détournement de signal

- Redéfinition de l'action exécutée lors de la réception d'un signal :

MISE EN OEUVRE

- Appel système ***sigaction***:

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

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

- ✓ **`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).

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

Détournement de signal

- Redéfinition de l'action exécutée lors de la réception d'un signal :

MISE EN OEUVRE

- Exemple:

```
void sig_hand(int sig){
    printf ("signal reçu %d \n",sig);
}

int main(int argc, char **argv) {
    sigset_t sig_proc;
    struct sigaction action;

    sigemptyset(&sig_proc);

    /* changer le traitement */
    action.sa_mask=sig_proc;
    action.sa_flags=0;
    action.sa_handler = sig_hand;
    sigaction(SIGINT, &action,0);
```

```
/* masquer SIGINT */
    sigaddset (&sig_proc, SIGINT);
    sigprocmask (SIG_SETMASK,
    &sig_proc, NULL);

    /* attendre le signal SIGINT */
    sigfillset (&sig_proc);
    sigdelset (&sig_proc, SIGINT);
    sigsuspend (&sig_proc);

    return EXIT_SUCCESS;
}
```

Ensemble de signaux

- Un ensemble de signaux doit être défini pour une prise en compte par le processus :

MISE EN OEUVRE

- Type **sigset_t** :
- Appels systèmes ***sigXXXset*** :

- ✓ `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

Ensemble de signaux

- Un ensemble de signaux doit être défini pour une prise en compte par le processus :

MISE EN OEUVRE

- Appels système **sigprocmask** :

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

Opération à effectuer

Ensemble de signaux sur
lesquels appliquer **how**

Etat des précédents signaux
reçois si **old** != NULL

- ✓ **SIG_BLOCK** : tous les signaux de *set* seront bloqués.
- ✓ **SIG_UNBLOCK** : tous les signaux indiqués dans *set* seront débloqués.
- ✓ **SIG_SETMASK** : *set* contient directement l'état de tous les signaux du système.
- ✓ Il faut raisonner de façon ensembliste

Blocage de signal

- Un signal peut être inhibé par un processus lors de sa réception :

MISE EN ŒUVRE

- Le signal est reçu mais ne sera pas traité tant que celui-ci sera bloqué.
 - ✓ Ceci permet d'effectuer certaines tâches de manière atomique.
 - ✓ Si un même signal est envoyé plusieurs fois, il n'est mémorisé qu'une fois.
 - ✓ Un signal envoyé mais non traité par le processus récepteur est dit pendant (pending).

Prise en charge OS

- Implémentation & spécificités au sein de l'OS :

MISE EN OEUVRE

- 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.
- Après ***fork()***, le fils a un comportement vis-à-vis des signaux identique au comportement du père
- Après ***exec()***, les signaux ignorés continuent d'être ignorés, les autres signaux reprennent le comportement par défaut.

Sommaire

I. Principes

II. Mise en œuvre

III. Essentiel

IV. Conclusion

Essentiel

- Toutes les notions abordées dans ce chapitre sont fondamentales



Conclusion



Annexes

Annexes

- Liens annexes :