Gestion classique des **SIGNAUX**

SIGNAUX

- Un signal est un mécanisme de base du noyau Linux
- Un signal est une valeur envoyée par le noyau à un processus en cours d'exécution pour signaler le survenu d'un événement
- Un signal reçu par un processus interrompe son flux d'exécution. Cette interruption peut avoir lieu lors d'une exécution d'une instruction non-atomique
- Lorsqu'un processus reçoit un signal, il peut agir de différentes façons, selon l'action enregistrée ou prédéfinie pour le signal.
- Pour chaque signal, il existe une action par défaut, qui détermine ce qui arrive au processus si le programme ne spécifie pas d'autre comportement.
- Tous les noms et les fonctions qui concernent les signaux sont définis dans */usr/include/bits/signum.h ou /usr/include/sys/signal.h*

IUT DE PARIS
UNIVERSITÉ PARIS DESCARTES

Liste des SIGNAUX classiques Unix

Numéro	Nom	Description
1	***SIGHUP***	**Instruction (HANG UP) - Fin de session**
2	***SIGINT***	**Interruption**
3	***SIGQUIT***	**Instruction (QUIT)**
4	*SIGILL*	Instruction illégale
5	*SIGTRAP*	Trace trap
6	*SIGABRT* (ANSI)	Instruction (ABORT), 6 *SIGIOT* (BSD) IOT Trap
7	*SIGBUS*	Bus error
8	*SIGFPE*	Floating-point exception - Exception arithmétique
9	***SIGKILL***	**Instruction (KILL) - termine le processus immédiatement**
10	***SIGUSR1***	**Signal utilisateur 1**
11	*SIGSEGV*	Violation de mémoire
12	***SIGUSR2***	**Signal utilisateur 2**
13	*SIGPIPE*	Broken PIPE - Erreur PIPE sans lecteur

IUT DE PARIS
UNIVERSITÉ PARIS DESCARTES

Liste des SIGNAUX classiques Unix

Numéro	Nom	Description
14	*SIGALRM*	Alarme horloge
15	***SIGTERM***	**Signal de terminaison**
16	*SIGSTKFLT*	Stack Fault
17	*SIGCHLD ou SIGCLD*	Modification du statut d'un processus fils
18	*SIGCONT*	Demande de reprise du processus
19	*SIGSTOP*	Demande de suspension imbloquable
20	*SIGTSTP*	Demande de suspension depuis le clavier
21	*SIGTTIN*	Lecture terminal en arrière-plan
22	*SIGTTOU*	Ecriture terminal en arrière-plan
23	*SIGURG*	Evènement urgent sur socket
24	*SIGXCPU*	Temps maximum CPU écoulé
25	*SIGXFSZ*	Taille maximale de fichier atteinte

Liste des SIGNAUX classiques Unix

- **Numéro Nom Description**
- 26 *SIGVTALRM* Alarme horloge virtuelle
- 27 *SIGPROF* Profiling alarm clock
- 28 *SIGWINCH* Changement de taille de fenêtre
- 29 *SIGPOLL* (System V) Occurrence d'un évènement attendu, 29 *SIGIO (BSD)* I/O possible actuellement
- 30 *SIGPWR* Power failure restart
- 31 *SIGSYS* Erreur d'appel système
- 31 *SIGUNUSED* Non utilisé.

SIGNAUX temps-réel

- Les **signaux temps réel** sont des **extensions** de **SIGUSR1** et **SIGUSR2**.
- Ils sont réservés à l'utilisateur et ne sont pas déclenchés par des évènements détectés par le noyau.
- Temps-réel ⟹ les programmes pour lesquels le temps mis pour effectuer une tâche constitue un facteur important du résultat.

- Leurs caractéristiques p/r aux signaux classiques :
  - Nombre plus important de signaux utilisateur ;
  - Empilement des occurrences des signaux bloqués ;
  - Délivrance prioritaire des signaux ;
  - Informations supplémentaires fournies au gestionnaire.

SIGNAUX

- Les événements qui peuvent générer un signal sont divers :
  - Une erreur dans le programme: division par zéro
  - La terminaison d'un processus fils (***wait*** et ***waitpid***);
  - L'échéance d'un minuteur (***timer***) ou d'un alarme
  - L'exécution de ***kill*** ou de ***raise***
  - ctrl-c ou ctrl-z … (envoyés depuis le clavier par l'utilisateur)

Actions prédéfinies

- Les actions prédéfinies pour les signaux sont les suivantes:
  - Ignorer le signal
  - Terminer le processus
  - Terminer le processus avec un fichier « Core dump[1]»
  - Suspendre le processus

- [1]*La terminaison du processus est associée à la création d'un fichier* ***Core dump*** *(dans le répertoire courant du processus) dans lequel est sauvegardée une image de la mémoire du processus. Ce fichier peut être utilisé pour examiner l'état du stack et des variables au moment de la réception du signal.*

Comportement du processus

- Les actions exécutées lors de la réception d'un signal peuvent être les suivantes:
  - Ignorer le signal
  - Capturer le signal et utiliser le gestionnaire spécifié
  - Accepter l'action prédéfinie pour ce signal

- Ces actions peuvent être spécifiées/modifiées à l'aide des fonctions ***signal*** et ***sigaction***

- ***Pour SIGKILL et SIGSTOP, il est impossible de changer l'action définie pour le signal.***

Classification des signaux

Signaux d'erreur de programme

- SIGFPE : erreur arithmétique. *L'action prédéfinie est de terminer le processus et écrire un core dump*.
- SIGSEGV : le nom dérive de « SEGment Violation », le programme essaie de lire ou d'écrire dans une zone mémoire protégée. *Idem*
- …

Classification des signaux

Signaux de terminaison

- SIGTERM : le nom dérive de TERMinate, fermeture d'un programme. Ce signal peut être ignoré, bloqué ou intercepté. *L'action prédéfinie est de terminer le processus.*
- SIGINT : dérive de INTerrupt, interruption d'un programme. Fonction kill et ctrl-c. *Idem*
- SIGQUIT : similaire à SIGINT, ctrl-\ (ctrl-AltGr-\ sur les claviers français), fichier d'image mémoire (core).

Classification des signaux

Signaux de terminaison

- SIGKILL : commande/fonction kill.
  - Il ne peut ni être capturé ni être ignoré, ni même être temporairement bloqué par un processus.
  - Il termine immédiatement n'importe quel processus.
  - init (pid = 1) ne reçoit pas ce signal.
- …

Classification des signaux

- **Signaux d'alarme**
  - SIGALRM : l'échéance d'un timer. Fonction ***alarm***
  - …
- **Signaux pour le contrôle de session**
  - SIGCHLD : ce signal est envoyé au processus père quand un fils termine ou est arrêté. *L'action prédéfinie est d'ignorer le signal.*
  - SIGSTOP : ce signal arrête un processus (il devient dans un état sleep).
  - SIGCONT : il fait repartir un programme précédemment arrêté par SIGSTOP.
  - …
- **Signaux de Input/Output asynchrone**

Classification des signaux

- **Signaux des opérations erronées**
  - SIGPIPE : erreur dans la manipulation du pipe
  - SIGXCPU : *CPU time limit exceeded*; le processus excède la limite du temps de CPU disponible.
  - SIGXFSZ : *File size limit exceeded*; le processus cherche à étendre un fichier outre à la dimension maximale consentie.
  - …
- **Autres signaux**
  - SIGUSR1 : est un signal à disposition de l'utilisateur
  - SIGUSR2 : comme SIGUSR1
  - …

La gestion des signaux

- Fonctions pour la gestion des signaux :
  - *signal, sigaction*
  - *sigprocmask*
  - *sigpending*
  - *sigsuspend, …*

- Fonctions pour la transmission/émission de signaux :
  - *raise*
  - *kill*
  - *killpg*

La gestion des signaux

- ***signal***, ***sigaction*** :
  - Spécifier le comportement d'un processus (fonction à appeler) lors de la réception d'un signal

- ***sigprocmask*** :
  - Bloquer et débloquer des signaux
  - Examiner et modifier le masque (***mask***, la liste des signaux bloqués)
  - Il est impossible de bloquer SIGKILL et SIGSTOP

La gestion des signaux

- ***sigpending*** : renvoie l'ensemble des signaux bloqués en attente de livraison au processus en cours d'exécution.

- ***sigsuspend (int sigsuspend(const sigset_t *mask))*** :
  I. remplace temporairement le masque de signaux du processus en cours avec le masque fourni dans *mask*
  II. suspend le processus jusqu'à la livraison d'un signal
    - invoquer un gestionnaire de signal
    - Terminer le processus

La gestion des signaux

- ***raise :*** envoyer un signal au processus courant (en cours d'exécution)
- ***kill :*** envoyer un signal à un processus ou un groupe de processus
- ***killpg :*** envoyer un signal à un groupe de processus

Lors de la réception d'un signal

- fonction écrite par l'utilisateur
- fonction invoquée par le système (pas de retour fonctionnel)
- fonction exécutée de manière asynchrone par rapport au processus courant
- fonctions prédéfinies:
  - SIG_DFL : traitement par défaut
    1) terminaison sans ou avec image mémoire (core dump)
    2) suspension
    3) Ignorance
  - SIG_IGN : ignorance

raise et kill

- `#include <signal.h>`
- *`int`* ***`raise`*** *`(int num_signal)`* : envoie le signal num_signal au processus courant. Valeur de retour = 0 si succès et -1 si erreur.
- `#include <signal.h>`
- `#include <sys/types.h>`
- `int` ***`kill`*** `(pid_t proc_pid,int num_signal)`
  - Le premier paramètre (*proc_pid*) est l'identifiant du processus cible
  - Le second (*num_signal*) est le numéro du signal
  - Valeur de retour = 0 si succès et -1 si erreur.
  - Si *proc_pid* contient l'identifiant du processus fils, on peut utiliser la fonction *kill* pour terminer un processus fils depuis le père en l'appelant comme ceci :
    - *`kill(child_pid,SIGTERM) ;`*

raise et kill

- proc_pid > 0 => envoyer le signal au processus dont le pid est proc_pid.
- proc_pid = 0 => envoyer le signal à chaque processus membre d'un groupe de processus (groupe auquel appartient le processus courant).
- proc_pid = -1 => envoyer le signal à tous les processus (exclus init). Sauf l'administrateur peut envoyer un signal à n'importe quel processus.
- proc_pid < -1 => envoyer le signal à chaque processus membre d'un groupe de processus dont le pid du groupe est|proc_pid|.

- raise(*num_signal*) == kill(getpid(), *num_signal)*

killpg

- `#include <signal.h>`
- `int killpg(pid_t pidgrp,int num_signal)`
  - Le premier paramètre (*pidgrp*) est l'identifiant d'un groupe de processus
  - Le second (*num_signal*) est le numéro du signal
  - Valeur de retour = 0 si succès et -1 si erreur.

- *`killpg(pidgrp, num_signal) == kill(- pidgrp, num_signal)`*

IUT DE PARIS
UNIVERSITÉ PARIS DESCARTES

Commande kill - exemples

- `kill  -9     pid_proc`
- `kill  -KILL  pid_proc`
- `kill  -15     pid_proc`
- `kill  -TERM  pid_proc`
- `kill  -SIGTERM  pid_proc`
- …

Le type sigset_t

- ***sigset_t*** a été considéré successivement par le noyau Linux comme :
  - Un unsigned long
  - Un tableau de 2 unsigned long
  - Une structure …

- La définition réelle du type ***sigset_t*** peut varier selon les machines, les versions du noyau et le type de fichier d'en-tête utilisé (noyau ou bibliothèque C)
- Les routines suivantes (définies par SUSv4 (Single UNIX Specification-version 4)) permettent de modifier les ensembles de signaux de type sigset_t :
  - int ***sigemptyset*** (sigset_t * ens);
  - int ***sigfillset*** (sigset_t * ens);
  - int ***sigaddset*** (sigset_t * ens, int num_sig);
  - int ***sigdelset*** (sigset_t * ens, int num_sig);
  - int ***sigismember*** (const sigset_t * ens, int num_sig);

Masquage des Signaux (*signal mask*)

- Tous les systèmes ‘‘Unix-like’’ modernes permettent de bloquer temporairement (ou d’ignorer complètement avec SIG_ING) la livraison d’un signal à un processus.
- ***sigprocmask*** : modifie la liste des signaux **bloqués** (*signal mask*) du processus en cours
- `#include <signal.h>`
- `int sigprocmask(int how, const sigset_t *newset, sigset_t *oldset)`
- `Valeur de retour = 0 si succès et -1 si erreur.`
- ***oldset*** sera rempli avec le masque actuel (avant l’exec. de *sigprocmask*) de blocage des signaux.
- L’utilité principale d’un blocage des signaux est la protection des sections critiques de code.

Masquage des Signaux (*signal mask*)

- `int` ***`sigprocmask`***`(int how, const sigset_t *newset, sigset_t *oldset)`
- Le comportement de ***sigprocmask*** dépend de la valeur de ***how***, avec les conventions suivantes :
  - SIG_BLOCK : l'ensemble des signaux bloqués est l'union de l'ensemble actuel (***oldset***) et de l'argument ***newset.***
  - SIG_UNBLOCK : les signaux dans l'ensemble ***newset*** sont supprimés de la liste des signaux bloqués; il est possible de débloquer un signal non bloqué.
  - SIG_SETMASK : l'ensemble des signaux bloqués est égal à l'argument ***newset***.

sigprocmask - exemple

```
typedef struct {
    double X;
    double Y;
} point_t;
point_t centre, pointeur;


void gestionnaire_sigusr1
(int inutilise)
{
  centre.X = pointeur.X;
  centre.Y = pointeur.Y;
}
```

```
int main (void)
{
    sigset_t ensemble, ancien;
    sigemptyset(& ensemble);
    sigaddset(& ensemble, SIGUSR1);

    sigprocmask(SIG_BLOCK,&ensemble,&ancien);
    X1 = centre.X * zoom;/* voici la section*/
    Y1 = centre.Y * zoom;/* critique protegee*/
    sigprocmask(SIG_SETMASK,&ancien,NULL);
    ...
    ...
    cercle( centre.X, centre.Y, rayon);
    ...
    return EXIT_SUCCESS;
}
```

Masquage des Signaux (*signal mask*)

- ***sigpending*** **:** renvoie l'ensemble des signaux bloqués en attente de livraison au processus courant. Le masque des signaux en attente est stocké dans ***set***
- ***sigsuspend*** : modifie temporairement la liste des signaux bloqués (***mask***) **et** demande la mise en sommeil du processus jusqu'à la délivrance d'un signal.

- `#include <signal.h>`
- `int sigpending(sigset_t *set)`
- `int sigsuspend(const sigset_t *mask)`

sigpending - exemple

```
int main (void)
{
    int i;
    sigset_t ensemble, ...;
    sigfillset(& ensemble); /* on remplit ensemble avec tous les signaux */
    sigdelset(& ensemble, SIGINT); /* sauf SIGINT */
    sigprocmask(SIG_BLOCK,&ensemble, NULL);
    ...
    ...

    /* Voyons maintenant qui est en attente */
    sigpending(& ensemble);
    for(i=1; i < NB_SIG_CLASSIQUES; i++)
        if ( sigismember(& ensemble, i) )
            fprintf(stdout,`` en attente %d (%s)\n``, i,sys_siglist[i]);
    ...
    ...
    return EXIT_SUCCESS;
}
```

Mise en place des ‘‘pilotes’’ de signaux

- ***signal*** : assignation d’une fonction ***handler*** (ou un gestionnaire) au traitement de la prochaine occurrence d’un signal ***signum***
  - `#include <signal.h>`
  - `sighandler_t` ***`signal`***`(int signum, sighandler_t handler)`
  - Valeur de retour = le gestionnaire *précedent* si succès et SIG_ERR si erreur.
  - *handler* peut prendre les 2 valeurs *SIG_IGN* et *SIG_DFL*
  - Utilisez plutôt ***sigaction*** qui est plus flexible et robuste (mais plus complexe).

Mise en place des ‘‘pilotes’’ de signaux

sigaction **:** assignation d’une fonction et d’un masque au traitement d’un signal (une version plus complète de ***signal***)

- `#include <signal.h>`
- `int` ***`sigaction`***`(int signum, const struct sigaction *newact, struct sigaction *oldact)`
- ***sigaction*** sert à modifier l’action effectuée par un processus à la réception d’un signal spécifique.
- Cette fonction renvoie 0, si succès, et -1 si erreur.
- Le 1er pointeur (***newact***) est le nouveau comportement à programmer, et le 2ème pointeur (***oldact***) sert à sauvegarder l’ancienne action.
- Les signaux *SIGKILL* et *SIGSTOP* ne peuvent être ni ignorés, ni interceptés !

IUT DE PARIS
UNIVERSITÉ PARIS DESCARTES

sigaction() plus en détail

int ***sigaction*** (int signum, const struct sigaction *newact, struct sigaction *oldact)

- ***signum*** indique le signal concerné, à l'exception de SIGKILL et SIGSTOP.
- Si ***newact*** est non nul, la nouvelle action pour le signal *signum* est définie par *newact*. Sinon (*newact* est NULL), aucune modification n'a lieu.
- Si ***oldact*** est non nul, l'ancienne action est sauvegardée dans *oldact*.
- La structure ***sigaction*** est définie dans <sigaction.h>, qui est à son tour inclus par <signal.h>.

sigaction() plus en détail

- La **structure *sigaction*** est définie par :

```
struct sigaction {
    void  (* sa_handler)   (int); // signal(int signum, void (*handler) (int))
    sigset_t    sa_mask;
    int         sa_flags;
    void     (* sa_restorer)  (void);
}
```

  - ***sa_handler*** peut prendre une des trois valeurs suivantes : SIG_DFL, SIG_IGN, et un pointeur vers une fonction de gestion de signal.
  - ***sa_mask*** est du type *sigset_t* (c.à.d. un ens. de signaux) et sert à indiquer la liste des signaux qui seront bloqués durant l'exécution de la fonction de gestion de signal. A cette liste, est ajouté le signal qui a déclenché cet appel sauf dans le cas où un comportement divers est spécifié avec *sa_flags*.
  - ***sa_flags*** contient un OU binaire entre différentes constantes permettant de configurer le comportement du gestionnaire du signal.
  - ***sa_restorer*** n'est pas utilisé

sigaction - exemple

```
Void gestionnaire (int num_sig)
{
    fprintf(stdout,''%d (%s) recu \n'', num_sig,
sys_siglist[num_sig]);
}
int main (void)
{
    int i;
    struct sigaction action;
    sigset_t ensemble, ...;

    action.sa_handler = gestionnaire;
    sigfillset(& (action.sa_mask) );
    action.sa_flags = 0; /* Pas de SA_RESTART */

    for(i=1; i < NSIG; i++)
        if ( sigaction(i, & action, NULL) != 0 )
            fprintf(stderr,`` %ld : %d pas capture\n``, (long)
getpid(),i);
    return EXIT_SUCCESS;
}
```