La gestion des signaux

O. Aktouf

Définitions

Un signal correspond à un événement.

Occurrence de l'événement  émission du signal

Signaux : mécanisme de base de communication

-entre le système et un processus

-entre processus

Définitions

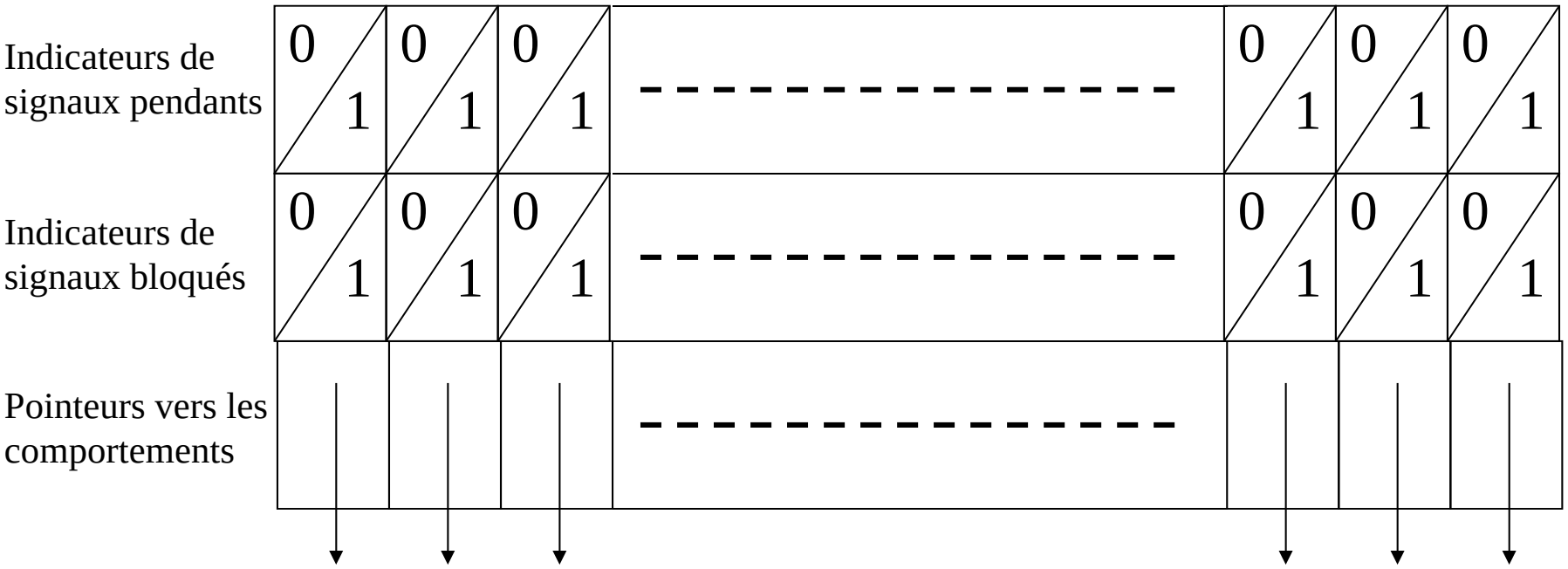
Signal pendant : signal envoyé à un processus mais non encore pris en compte

Signal délivré : le processus le prend en compte au cours de son exécution

Signal bloqué ou masqué : signal dont la prise en compte est volontairement retardée (version BSD et norme Posix)

Définitions

Structure de gestion des signaux dans le PCB d'un processus



Identification des signaux

Fichier `signal.h` : constantes et structures relatives à la gestion des signaux

NSIG types de signaux différents ⟶ signaux identifiés par un entier allant de 1 à `NSIG`

Nom symbolique d'un signal :

`SIG...`

Préfixe Suffixe

Utilisation des signaux

Un événement associé à un signal peut être :

- extérieur au processus
- intérieur au processus

Utilisation des signaux

Exemple :

- Exécution d'une division par 0
- Le processus reçoit le signal `SIGFPE`

(terminaison du processus avec sauvegarde d'une image mémoire)

Utilisation des signaux

Exemples de signaux standards requis par la norme Posix :

`SIGHUP` : terminaison du processus leader de session
`SIGFPE` : erreur arithmétique
`SIGKILLL` : signal de terminaison
`SIGCHLD` : terminaison d'un processus fils

La commande `kill -l` donne la liste des signaux du système.

Les appels système

Appel kill()

```
#include <signal.h>
int kill(pid_t pid, int sig) ;
```

Valeur de pid	Interprétation
0	signal envoyé à tous les processus du groupe de l'appelant
-1	signal envoyé à tous les processus du système sauf 0 et 1 (système V.4)
>0	signal envoyé au processus de PID indiqué
<-1	tous les processus du groupe |pid|

Les appels système

Appel kill()

Valeur de sig	Interprétation
∈[1..NSIG]	signal indiqué par le numéro sig
0	demande d'information (test d'existence de processus, …)

Retour de l'appel : 0 ou -1

Les appels système

Appel raise()

```
#include <signal.h>
int raise(int sig) ;
```

Le processus s’envoie à lui-même un signal.

Les appels système

Appel signal()

```
#include <signal.h>
void(*signal(int sig, void(*action)(int)))(int) ;
```

- `sig` : numéro du signal
- `action` : nom de la fonction à réaliser à la réception du signal

- Spécifie le comportement du processus à la réception d'un signal
- Installe le comportement défini par la fonction `action` pour le signal donné
- Renvoie un pointeur sur le comportement antérieur en cas de succès
- Renvoie la valeur `SIG_ERR` en cas d'échec

Les appels système

Appel signal()

Valeurs possibles du paramètre action :

`SIG_DFL` : comportement par défaut. Peut être :

- *exit* : le processus se termine avec éventuellement la production d'un core
- *ignore* : le processus ignore le signal
- *pause* : le processus est suspendu
- *continue* : reprise d'un processus suspendu

Les appels système

Appel signal()

Valeurs possibles du paramètre action :

`SIG_IGN` : le signal est ignoré

`HANDLER` : une fonction au choix du programmeur

- fonction de type `void` à un seul paramètre entier
- reçoit en paramètre le numéro du signal

Les appels système

Appel signal()

Remarque :

Pour certains signaux (`SIGKILL`, `SIGSTOP`,…), le seul comportement possible est le comportement par défaut.

Les appels système

Appel pause()

```
#include <signal.h>
int pause (void) ;
```

- Attend qu'un signal qui n'est pas ignoré soit délivré au processus.
- Si le comportement par défaut est la terminaison, le processus se termine.
- Si le signal reçu est dérouté, `pause()` renvoie -1 et le déroutement est exécuté.

Exemples

Exemple 1 : rendre un programme insensible au signal `SIGINT`

Exemple 2 : un processus père envoie un signal `SIGUSR1` à son processus fils après avoir testé son existence

Fonctions de la norme Posix

La norme Posix permet

- de manipuler des ensembles de signaux
- de masquer des signaux

Fonctions de la norme Posix

Manipulation d'ensembles de signaux

Fonction	**Action**
int sigemptyset(sigset_t *p_ens)	*p_ens = {}
int sigfillset(sigset_t *p_ens)	*p_ens={1, …, NSIG}
int sigaddset(sigset_t *p_ens, int sig)	*p_ens=p_ens + {sig}
int sigdelset(sigset_t *p_ens, int sig)	*p_ens=p_ens - {sig}
int sigismember(sigset_t *p_ens, int sig)	teste si sig appartient à *p_ens

Fonctions de la norme Posix

Manipulation d'ensembles de signaux

Retour de ces fonctions :

- -1 en cas d'erreur,
- pour la fonction sigismember(), 1 ou 0 selon que le signal sig appartient ou non à l'ensemble *p_ens,
- les autres fonctions renvoient 0 en cas de succès.

Fonctions de la norme Posix

Blocage des signaux

```
#include <signal.h>
int    sigprocmask(int    op,    const    sigset_t    *p_ens,
sigset_t *p_ens_ancien) ;
```

Retour de la fonction : 0 (succès) ou 1(échec)

Le nouveau masque est construit en fonction de la valeur du paramètre **op** à partir de l'ensemble pointé par **p_ens** et de l'ancien masque pointé par **p_ens_ancien** si ce pointeur n'est pas NULL.

Fonctions de la norme Posix

Blocage des signaux

Valeur du paramètre op	**Nouveau masque**
SIG_SETMASK	*p_ens
SIG_BLOCK	*p_ens ∪ *p_ens_ancien
SIG_UNBLOCK	*p_ens_ancien - *p-ens

Fonctions de la norme Posix

Obtention de la liste des signaux pendants bloqués

```
#include <signal.h>
int sigpending(sigset_t *p_ens) ;
```

Retour de la fonction : 0 (succès) ou –1 (échec)

p_ens : adresse de rangement de la liste des signaux recherchés

Fonctions de la norme Posix

La structure sigaction

```
struct sigaction {

        void (*sa_handler)() ; /* comportement à adopter
                               à la réception du signal */
        sigset_t sa_mask ;     /*signaux supplémentaires à bloquer */
        int sa_flag ;          /* options */
}
```

Fonctions de la norme Posix

L'appel sigaction()

```
#include <signal.h>
int sigaction(int sig, const struct sigaction
*p_action, struct sigaction *p_action_anc) ;
```

Paramètre p_action : pointeur sur la structure définissant le comportement à installer pour le signal sig

Paramètre p_action_ancien : indique l'ancien comportement au retour de la fonction