


La gestion des signaux

la gestion des signaux (1)

- ☞ attention : problèmes de portabilité (\Rightarrow norme POSIX)
- ☞ signal = interruption logicielle ; signale un événement particulier (seule information véhiculée = le numéro du signal)
- ☞ dans BCP

		3	2	1	
1/0		1/0	1/0	1/0	signaux pendants
1/0		1/0	1/0	1/0	signaux masqués
					comportements



la gestion des signaux (2)

- ☞ signal pendant : non encore pris en compte (ex : sonnerie téléphone)
 - ◆ 1 bit/signal \Rightarrow un signal émis est perdu s'il est déjà pendant
- ☞ signal délivré : pris en compte par le processus au cours de son exécution
 - ◆ bit signal pendant correspondant basculé à 0
 - ◆ déroutement du processus : exécution d'une fonction particulière et reprise de l'exécution du processus là où elle avait été interrompue

la gestion des signaux (3)

- ☞ signaux bloqués ou masqués (définis dans norme POSIX) : possibilité de différer la délivrance de certains types de signaux)
- ☞ NSIG types de signaux différents identifiés par un $n^{\circ} \in [1 .. NSIG]$
 - ♦ définis dans `<signal.h>`
 - ♦ désignables par un nom symbolique

envoi de signaux (1)

☞ la primitive kill:

```
int kill (pid_t pid, int sig);
```

- ◆ envoi du signal sig au processus désigné par pid
- ◆ rem : \exists commande kill : envoi d'un signal à un processus depuis un shell
- ◆ interprétation de pid :

>0	processus pid
0	tous les processus dans le même groupe que le processus émetteur
<-1	tous les processus du groupe pid

envoi de signaux (2)

- ♦ interprétation de sig :

<0 ou >NSIG	valeur incorrecte
0	aucun signal envoyé (test d'existence de processus)
sinon	signal de n° sig

- ♦ rem : un processus ne peut envoyer un signal que vers un processus ayant même propriétaire que lui
- ♦ retour : 0 si OK, -1 si erreur

comportement à la délivrance (1)

- ☞ 1 signal \leftrightarrow un handler par défaut (SIG_DFL)
- ☞ comportements :
 - ◆ terminaison du processus
 - ◆ terminaison du processus avec image mémoire (core)
 - ◆ signal ignoré (sans effet)
 - ◆ suspension du processus
 - ◆ continuation : reprise d'un processus stoppé, ignoré sinon
- ☞ rem : impossible d'installer un autre handler que SIG_DFL pour certains signaux (ex : SIGKILL)

comportement à la délivrance (2)

- ☞ autres types de handlers "installables"
 - ◆ SIG_IGN : pour ignorer le signal
 - ✓ rem : le signal est délivré (indicateur de signal pendant basculé à 0)
 - ◆ fonction utilisateur exécutée à la délivrance du signal
 - ✓ ie : signal capté ou capturé
 - ✓ fonction de type void
 - ✓ reçoit à l'appel, le n° du signal en paramètre

comportement à la délivrance (3)

- ☞ définition d'un handler :
 - ◆ void handler(int sig)
- ☞ rem : def d'un pointeur phandler sur handler :
 - ◆ void (* phandler)(int sig)

comportement à la délivrance (4)

- ☞ délivrance d'un signal capté :
 - ◆ bit correspondant du signal pendant basculé à 0
 - ◆ déroutement du processus vers la fonction handler
 - ◆ blocage automatique du signal en cours de délivrance pendant l'exécution du handler
 - ◆ après exécution du handler, reprise de l'exécution du processus au point où il avait été dérouté

quand a lieu la délivrance ? (1)

- ☞ signaux non bloqués : passage en mode noyau à mode utilisateur
 - ◆ après une interruption matérielle
 - ◆ après élection par le noyau
 - ◆ après exécution d'un appel système
 - ◆ ...
 - ◆ cq : un processus n'est pas interruptible lorsqu'il est en mode noyau

blocage des signaux (1)

- ☞ possibilité d'installer un masque de signaux
- ☞ manipulation d'ensemble de signaux par les fonctions
 - ◆ `int sigemptyset (sigset_t *p_ens);`
 $\Rightarrow *p_ens = \emptyset$
 - ◆ `int sigfillset (sigset_t *p_ens);`
 $\Rightarrow *p_ens = \{1, \dots, \text{NSIG}\}$
 - ◆ `int sigaddset (sigset_t *p_ens, int sig);`
 $\Rightarrow *p_ens = \{\text{sig}\} \cup *p_ens$

blocage des signaux (2)

- ◆ `int sigdelset(sigset_t *p_ens, int sig);`

$\Rightarrow *p_ens = *p_ens - \{sig\}$

- ◆ `int sigismember(sigset_t *p_ens, int sig);`

$\Rightarrow \{sig\} \in *p_ens$

☞ retour : -1 en cas d'erreur

blocage des signaux (3)

☞ installation d'un masque de blocage :

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

```
int sigprocmask(int op, const sigset_t  
    *p_ens, sigset_t *p_ens_ancien);
```

✓ interprétation de op :

valeur du paramètre	nouveau masque
SIG_SETMASK	*p_ens
SIG_BLOCK	*p_ens U *p_ens_ancien
SIG_UNBLOCK	*p_ens_ancien - *p_ens

✓ retour :

- 0 ou -1
- masque antérieur récupéré au retour dans p_ens_ancien (si ≠ NULL)

blocage des signaux (3)

- ☞ récupération des signaux pendants bloqués :

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

```
int sigpending(sigset_t *p_ens);
```

blocage des signaux - exemple (1)

```
#include <stdio.h>
#include <signal.h>
sigset_t ens1, ens2;
int sig;
main()
{
    /* ens1 = {SIGINT, SIGQUIT} */
    sigemptyset(&ens1);
    sigaddset(&ens1, SIGINT);
    sigaddset(&ens1, SIGQUIT);
    /* installation du masque ens1 */
    sigprocmask(SIG_SETMASK, &ens1, NULL);
    sleep(10);
}
```


blocage des signaux - exemple (2)

```
/* extraction des signaux pendants masqués */
sigpending(&ens2);
printf("signaux pendants : ");
for (sig = 1; sig < NSIG; sig++)
    if (sigismember(&ens2, sig))
        printf("%d ", sig);
putchar('\n');

/* déblocage des signaux */
sigemptyset(&ens1);
printf("deblocage des signaux\n");
sigprocmask(SIG_SETMASK, &ens1 , NULL);
printf("fin du processus\n");
}
```

manipulation des handlers (1)

- ☞ structure sigaction : décrit le comportement général lors de la délivrance d'un signal

```
struct sigaction {  
    void (*sa_handler) (); /* SIG_DFL ou SIG_IGN  
                           ou pointeur sur handler */  
    sigset_t sa_mask ; /* signaux à bloquer  
                       pendant l'exécution du handler */  
    int sa_flags /* options permettant de  
                modifier comportement cf man */  
};
```

- ☞ rem : un signal en cours de délivrance est automatiquement bloqué pendant l'exécution du handler

manipulation des handlers (2)

- ☞ primitive `sigaction` : permet d'installer un handler pour un signal donné
- ☞ le nouvel handler reste installé jusqu'à une demande explicite de changement

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

```
int sigaction (int sig, const struct sigaction  
              *paction, struct sigaction *p_action_anc);
```

- ◆ si `paction ≠ NULL` : délivrance du signal `sig` ⇒
 - ✓ exécution de `paction->sa_handler`
 - ✓ masquage des signaux de l'ensemble `paction->sa_mask ∪ {sig}` pendant exécution du handler

manipulation des handlers (3)

- ◆ si `paction = NULL`, comportement non modifié
- ◆ si `paction_anc ≠ NULL` \Rightarrow obtention au retour de l'ancien comportement associé à sig
- ☞ exemple : compter le nombre de SIGINT délivrés à un proc et terminaison quand `nb = NMAX`

manipulation des handlers (4)

```
#include <stdio.h>
#include <signal.h>
#define NMAX 3

struct sigaction action;
int nb_int = 0;

void hand(int sig) {
    if (sig == SIGINT) {
        printf("nb_int = %d\n", ++nb_int);
        if (nb_int == NMAX) exit(SIGINT);
    }
}
```

manipulation des handlers (5)

```
else {  
    fprintf(stderr, "bizarre !\n");  
    exit(-1);  
}  
  
}  
  
main() {  
    action.sa_handler = hand;  
    sigaction(SIGINT, &action, NULL);  
    while (1) sleep(1);  
}
```

attente d'un signal

```
int pause (void);
```

- ➡ processus mis en attente de l'arrivée de signaux
- ➡ ne permet pas d'attendre l'arrivée d'un signal de type donné (ni de savoir quel signal a réveillé le processus)