

## Les processus : communication par signaux

Luigi Santocanale

Laboratoire d'Informatique Fondamentale,  
Centre de Mathématiques et Informatique,  
39, rue Joliot-Curie - F-13453 Marseille

17 octobre 2004

## Plan

- 1 Introduction aux processus
  - Généralités
  - L'ordonnancement
- 2 La communication par signaux
  - Généralités
  - Les signaux System V
  - Les signaux POSIX

## Plan

- 1 Introduction aux processus
  - Généralités
  - L'ordonnancement
- 2 La communication par signaux
  - Généralités
  - Les signaux System V
  - Les signaux POSIX

### Processus :

copie en mémoire d'un fichier exécutable,  
... en train de s'exécuter.

### Système multitache :

plusieurs processus en train de s'exécuter  
au même temps sur une seule CPU.

### Protection des autres processus :

partage de la mémoire :  
chaque processus a son espace d'adressage d'adressage privé.

### 2 modes d'exécution :

- mode utilisateur :  
accès aux données privées du processus.
- mode superviseur, mode noyau :  
accès aux données globales du système.

### Passage entre les modes :

- appel système (trap) demandé par le processus
- interruption.

« plusieurs processus en train de s'exécuter ...  
... au même temps sur une seule CPU »

partage du temps de calcul :  
le noyau est aussi *ordonnneur*,  
il s'occupe de repartir le temps memoire.

## Plan

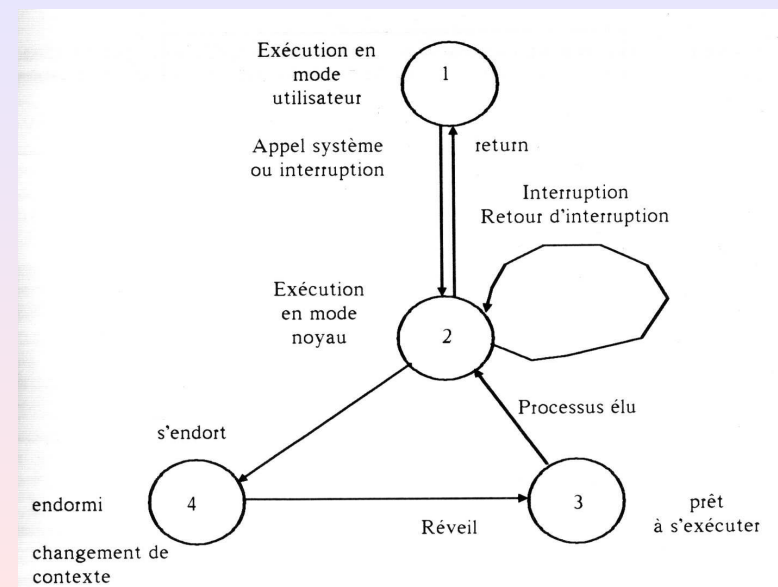
### 1 Introduction aux processus

- Généralités
- L'ordonnancement

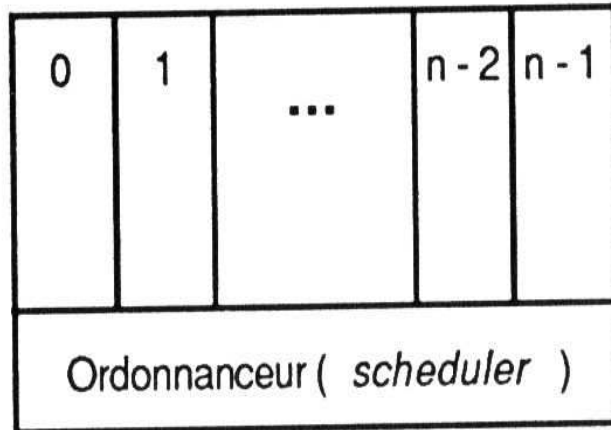
### 2 La communication par signaux

- Généralités
- Les signaux System V
- Les signaux POSIX

## Etats d'un processus (première approximation) (Bach 1989)



## Processus



## Ordonnancement pour le System V

(Bach 1989)

```

algorithme schedule_process
entrée: néant
sortie: néant
(
  while (aucun processus à prélever en vue de son exécution)
  (
    for ( chaque processus dans la file d'attente d'exécution )
      prélever le processus de plus haute priorité qui est
                                     chargé en mémoire;
    if (aucun processus éligible en vue de son exécution)
      rendre la machine oisive;
    /* une interruption sortira la machine de son état oisif */
  )
  extraire le processus choisi de la file d'attente d'exécution;
  changer de contexte pour le processus élu,
  reprendre son exécution;
)

```

## Ordonnancement circulaire (tourniquet)

Paramètres:

- Quantum.

Choix de la durée du quantum:

Quantum bref : overhead.

Quantum long : temps de réponse important.

## Ordonnancement avec priorités

Paramètres:

- Quantum.
- Priorité.

Un processus utilisateur peut être préempté.

Le recalcul de la priorité se fait chaque second selon la formule:

$$\text{priorité} = (\text{utilisation cpu})/2 + (\text{priorité de base})$$

Le processus avec la priorité plus basse est choisi.

## Plan

- 1 Introduction aux processus
  - Généralités
  - L'ordonnancement
- 2 La communication par signaux
  - Généralités
  - Les signaux System V
  - Les signaux POSIX

## Origine des signaux

- Frappes de caractères:

touche	signal
CTRL-C	SIGINT
CTRL-\	SIGQUIT
CTRL-Z	SIGTSTP

Utilisateur à la console → processus en avant-plan.

- Violation de mémoire, SIGSEGV  
« erreur du programme », noyau → programme même
- Commande kill → un processus « quelconque »

## La communication par signaux

Forme primitive de communication.

Un signal est

- envoyé par un processus,
- reçu par un autre processus,
- véhiculé par le noyau.

Le processus qui reçoit le signal

- a un comportement par défaut, en général:  
termination anormale du processus †.
- il peut modifier le comportement par défaut:  
exécuter une tâche sans terminer.
- il peut ignorer le signal.

## Liste des signaux

```
[lsantoca@localhost lecture4]$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
5) SIGTRAP     6) SIGABRT     7) SIGBUS      8) SIGFPE
9) SIGKILL     10) SIGUSR1    11) SIGSEGV    12) SIGUSR2
13) SIGPIPE    14) SIGALRM    15) SIGTERM    17) SIGCHLD
18) SIGCONT    19) SIGSTOP    20) SIGTSTP    21) SIGTTIN
22) SIGTTOU    23) SIGURG     24) SIGXCPU   25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO
30) SIGPWR     31) SIGSYS     32) SIGRTMIN   33) SIGRTMIN+1
34) SIGRTMIN+2 35) SIGRTMIN+3 36) SIGRTMIN+4 37) SIGRTMIN+5
38) SIGRTMIN+6 39) SIGRTMIN+7 40) SIGRTMIN+8 41) SIGRTMIN+9
42) SIGRTMIN+10 43) SIGRTMIN+11 44) SIGRTMIN+12 45) SIGRTMIN+13
46) SIGRTMIN+14 47) SIGRTMIN+15 48) SIGRTMAX-15 49) SIGRTMAX-14
50) SIGRTMAX-13 51) SIGRTMAX-12 52) SIGRTMAX-11 53) SIGRTMAX-10
54) SIGRTMAX-9  55) SIGRTMAX-8  56) SIGRTMAX-7  57) SIGRTMAX-6
58) SIGRTMAX-5  59) SIGRTMAX-4  60) SIGRTMAX-3  61) SIGRTMAX-2
62) SIGRTMAX-1  63) SIGRTMAX
```

## Signaux fréquents

1	SIGHUP	terminaison du processus leader	T
2	SIGINT	frappe de intr (CTRL-C)	T
3	SIGQUIT	frappe de quit (CTRL-\)	A
4	SIGILL	instruction illégale	A
6	SIGABRT	terminaison anormale	A
7	SIGBUS	accès à portion de mémoire non définie	A
8	SIGFPE	erreur arithmétique	A
9	SIGKILL	signal de terminaison (non intercepté)	T
10	SIGUSR1	signal utilisateur	T
11	SIGSEGV	violation écriture mémoire	A
12	SIGUSR2	signal utilisateur	T
13	SIGPIPE	écriture dans tube sans lecteur	T
14	SIGALRM	fin de temporisation	T
15	SIGTERM	signal de terminaison	T
17	SIGCHLD	terminaison (arrêt) d'un fils	I
18	SIGCONT	continuation	C
19	SIGSTOP	sig. de suspension (pas intercepté)	S
20	SIGTSTP	frappe caractère susp (CTRL-Z)	S

## Plan

- 1 Introduction aux processus
  - Généralités
  - L'ordonnancement
- 2 La communication par signaux
  - Généralités
  - Les signaux System V
  - Les signaux POSIX

## kill, raise

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

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

*pid* : > 0 : pid du destinataire.

0 : tous les processus du groupe.

-1 : non défini.

< -1 : tous les processus du groupe abs (pid).

*sig* : le signal à envoyer.

Sommaire : Envoie un signal à un processus.

```
int raise ( int sig );
```

Sommaire : Envoie le signal à soi-même : kill(getpid(),sig)

Remarques : bibliothèque standard C

## signal

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

*sig* : le signal à intercepter

*handler* : (l'adresse d'une) procédure qui gère le signal. Le  
constantes SIG\_DFL : comportement par défaut  
SIG\_IGN : ignorer le signal

Retourne : L'adresse de la procédure qui a géré ce signal jusqu'à maintenant.

## Programme : exsignal2.c

```
1 : #include <signal.h>
2 : #include <limits.h>
3 : #include <unistd.h>
4 : #include <stdio.h>
5 : #include <stdlib.h>
6 :
7 : static void handler(int signum);
8 : static int no_bonjour = 1;
9 :
10 : int main(void)
11 : {
12 :     unsigned int i;
13 :     signal(SIGINT, handler);
14 :     signal(SIGALRM, handler);
15 :
16 :     while (1)
17 :     {
18 :         for (i = UINT_MAX; i; i--);
19 :         kill(getpid(), SIGALRM);
20 :     }
21 :     exit(EXIT_SUCCESS);
22 : }
```

## Programme : exsignal1.c

```
1 : #include <signal.h>
2 : #include <stdio.h>
3 : #include <stdlib.h>
4 :
5 : void handler(int signum)
6 : {
7 :     printf("Merci, et à la prochaine.\n");
8 :     exit(EXIT_SUCCESS);
9 : }
10 :
11 : int main(void)
12 : {
13 :     signal(SIGINT, handler);
14 :     while (1);
15 :     exit(EXIT_SUCCESS);
16 : }
```

## Programme : exsignal2.c

```
23 :
24 : void handler(int signum)
25 : {
26 :     signal(SIGALRM, handler);
27 :     switch (signum)
28 :     {
29 :         case SIGINT:
30 :             printf("Merci, et A+.\n");
31 :             exit(EXIT_SUCCESS);
32 :         case SIGALRM:
33 :             printf("Bonjour no. %d.\n", no_bonjour++);
34 :             return;
35 :     }
36 : }
```

## alarm, pause

```
#include <unistd.h>
```

```
unsigned alarm ( unsigned no );  
no : attente de no secondes
```

Retourne : nombre de secondes restant si un alarme était déjà prévu.

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

Retourne : -1 si erreur

Sommaire : le processus s'endorme en attente d'un signal quelconque.

Remarques : évite l'attente active.

## Programme : exalarm.c

```
21 :  
22 : void handler(int signum)  
23 : {  
24 :     signal(SIGALRM, handler);  
25 :     switch (signum)  
26 :     {  
27 :     case SIGINT:  
28 :         printf("Merci, et A+.\n");  
29 :         exit(EXIT_SUCCESS);  
30 :     case SIGALRM:  
31 :         printf("Bonjour no. %d.\n", no_bonjour++);  
32 :         return;  
33 :     }  
34 : }
```

## Programme : exalarm.c

```
1 : #include <signal.h>  
2 : #include <unistd.h>  
3 : #include <stdio.h>  
4 : #include <stdlib.h>  
5 :  
6 : static void handler(int signum);  
7 : static int no_bonjour = 1;  
8 :  
9 : int main(void)  
10 : {  
11 :     signal(SIGINT, handler);  
12 :     signal(SIGALRM, handler);  
13 :  
14 :     while (1)  
15 :     {  
16 :         alarm(5);  
17 :         pause();  
18 :     }  
19 :     exit(EXIT_SUCCESS);
```

## Plan

- ① Introduction aux processus
  - Généralités
  - L'ordonnancement
- ② La communication par signaux
  - Généralités
  - Les signaux System V
  - Les signaux POSIX

## Motivations

Problèmes avec les signaux System V :

- ne sont pas fiables, les signaux pouvant être perdus.
- leur sémantique est différente de la sémantique des signaux BSD.

POSIX reprends le mécanisme des signaux BSD.

## Manipulation des ensembles de signaux

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

```
struct sigset_t;
```

Sommaire : Un ensemble de signaux

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

Sommaire : p.ens = ∅.

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

Sommaire : p.ens = tous les signaux.

```
int sigaddset ( sigset_t * p_ens, int sig );
```

Sommaire : Ajoute sig à p.ens.

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

Sommaire : Enlève sig de p.ens.

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

sig : cucu

Retourne : -1/0/1.

Sommaire : Teste sig ∈ p.ens

## Le gestionnaire des signaux

1	2	3	...	253	254	255
0/1	0/1	0/1	...	0/1	0/1	0/1
0/1	0/1	0/1	...	0/1	0/1	0/1
↓	↓	↓	...	↓	↓	↓

signaux pendant  
signaux bloqués  
comportements

Un signal est :

- *pendant* : pas encore pris en compte par le processus.
- *délivré* : pris en compte par le processus.
- *bloqué* ou *masqués* : le processus veut retarder la délivrance d'un tel signal.

Comportement:

- la routine à utiliser à la prise en compte du signal.

## La structure sigaction

```
struct sigaction {
    void (*sa_handler)(int); /* Pointeur à l'handler, ou SIG_DFL, SIG_IGN
    sigset_t sa_mask; /* signaux supplémentaires à bloquer */
    int sa_flags; /* options */
    ...
};
```

sa\_flags : SA\_RESETHAND, pour réinstaller SIG\_DFL.



## sigaction

```
int sigaction(int sig, const struct sigaction * new,  
struct sigaction * old);
```

*sig* : le signal à intercepter

*new* : le nouvelles coordonnées de l'handler

*old* : ou sauver les anciennes coordonnées. NULL si on ne veut pas sauver ces coordonnées.

Retourne : 0/-1

## Programme : exsigaction.c

```
25 :  
26 : void handler(int signum)  
27 : {  
28 :     /* sigaction(SIGALRM,&action,NULL); */  
29 :     switch (signum)  
30 :     {  
31 :     case SIGINT:  
32 :         printf("Merci, et A+.\n");  
33 :         exit(EXIT_SUCCESS);  
34 :     case SIGALRM:  
35 :         printf("Bonjour no. %d.\n", no_bonjour++);  
36 :         return;  
37 :     }  
38 : }
```

## Programme : exsigaction.c

```
1 : #include <signal.h>  
2 : #include <unistd.h>  
3 : #include <stdio.h>  
4 : #include <stdlib.h>  
5 :  
6 : static void handler(int signum);  
7 : struct sigaction action;  
8 : static int no_bonjour = 1;  
9 :  
10 : int main(void)  
11 : {  
12 :     action.sa_handler = handler;  
13 :     action.sa_flags = 0;  
14 :     sigemptyset(&action.sa_mask);  
15 :     sigaction(SIGINT, &action, NULL);  
16 :     sigaction(SIGALRM, &action, NULL);  
17 :  
18 :     while (1)  
19 :     {  
20 :         alarm(5);  
21 :         pause();  
22 :     }  
23 :     exit(EXIT_SUCCESS);  
24 : }
```

## sigprocmask

```
#include <signal.h>  
int sigprocmask(int how, const sigset_t * new, const  
sigset_t * old);
```

*how* : SIG\_SET\_MASK : \*new  
          SIG\_BLOCK : \*old ∪ \*new  
          SIG\_UNBLOCK : \*old \ \*new

*new* : ensemble à copier dans le gestionnaire.

*old* : pointeur à un ensemble où sauver le contenu courant du gestionnaire.

Retourne : 0/code erreur.

Sommaire : examine et change les signaux bloqués.

## sigpending

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

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

Retourne : 0/-1.

Sommaire : copie dans la structure ens l'ensemble des signaux bloqués et pendants.

## Programme : exsigprocmask.c

```
1 : #include <stdio.h>
2 : #include <signal.h>
3 : #include <stdlib.h>
4 : #include <unistd.h>
5 :
6 : sigset_t ens1, ens2;
7 : int sig;
8 :
9 : int main()
10 : {
11 :     /* Initialisation de la masque */
12 :     sigemptyset(&ens1);
13 :     sigaddset(&ens1, SIGINT);
14 :     sigaddset(&ens1, SIGQUIT);
15 :     sigprocmask(SIG_SETMASK, &ens1, NULL);
16 :
17 :     sleep(15);
18 :
```

## Programme : exsigprocmask.c

```
19 :     /* Affichaghe des signaux bloqués */
20 :     sigpending(&ens2);
21 :     printf("Signaux pendants : ");
22 :     for (sig = 1; sig < NSIG; sig++)
23 :         if (sigismember(&ens2, sig))
24 :             printf("%d ", sig);
25 :     putc('\n', stdout);
26 :
27 :     sleep(15);
28 :
29 :     /* Debloquage des signaux */
30 :     sigemptyset(&ens1);
31 :     printf("Debloquage.\n");
32 :     sigprocmask(SIG_SETMASK, &ens1, NULL);
33 :
34 :     /* Fin normale */
35 :     printf("Fin normale !!!\n");
36 :     exit(0);
37 : }
```