

Exclusion mutuelle sans attente La primitive select

Luigi Santocanale

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

8 décembre 2004

Plan

- 1 L'exclusion mutuelle sans attente
 - Les primitives `sleep` et `wakeup`
 - Les sémaphores et les mutex
 - Les moniteurs
- 2 Aperçu des interfaces **SysV IPC** et **POSIX threads**
 - Les sémaphores dans le System V IPC
 - Les threads POSIX
- 3 La primitive `select`
 - La primitive `select`

Plan

- 1 L'exclusion mutuelle sans attente
 - Les primitives `sleep` et `wakeup`
 - Les sémaphores et les mutex
 - Les moniteurs
- 2 Aperçu des interfaces **SysV IPC** et **POSIX threads**
 - Les sémaphores dans le System V IPC
 - Les threads POSIX
- 3 La primitive `select`
 - La primitive `select`

sleep et wakeup

Idée :

- Au lieu de l'attente active (= consommation de la CPU)
on s'endort : primitive
`sleep`,
- Un processus peut réveiller un autre processus :
primitive
`wakeup`.

Le modèle producteur-consommateur

- n cases, chacune est libre ou pleine.
- Le producteur :
 - peut remplir une case s'il y en a libres, sinon il s'endort en attente d'être réveillé par le consommateur.
 - après avoir produit, il réveille le consommateur si une seule case est pleine.
- Le consommateur :
 - peut vider une case s'il y en a pleines, sinon il s'endort en attente d'être réveillé par le producteur.
 - après avoir consommé, il réveille le producteur si une seule case est vide.

pseudo-code

```

1 : /* compteur est une variable partagée */
2 : /* entre le producteur et le consommateur */
3 : int compteur = 10; /* Nombre de cases pleines */

1 : producteur()
2 : {
3 :   while (1)
4 :   {
5 :     if (compteur == N)
6 :       sleep();
7 :     compteur++;
8 :     if (compteur == 1)
9 :       wakeup(consommateur);
10 :  }
11 : }

1 : consommateur()
2 : {
3 :   while (1)
4 :   {
5 :     if (compteur == 0)
6 :       sleep();
7 :     compteur--;
8 :     if (compteur == N - 1)
9 :       wakeup(producteur);
10 :  }
11 : }
    
```

Problèmes

Assomption :

- un wakeup est **reçu** si le destinataire est **endormi**.

Problème :

- un wakeup est **perdu** si le destinataire n'est **pas endormi**.

Exemples de perte de messages

Si `compteur == N`,
dans l'exécution parallèle de :

```

compteur--;
if(compteur == N) sleep();
if(compteur == N-1)
    wakeup(producteur);
    
```

De même, si `compteur == 0`,
dans l'exécution parallèle de :

```

compteur++;
if(compteur == 1)
    wakeup(consommateur);
if(compteur == 0) sleep();
    
```

Solutions (I)

- un message envoyé à un processus pas endormi est mis en attente (masqué, bloqué) (Implementation : bit de réveil d'attente).
- Exécuter les codes

```
if(compteur == N) sleep();      compteur--;
                                if(compteur == N-1)
                                    wakeup(producteur);

compteur++;                    if(compteur == 0) sleep();
if(compteur == 1)              wakeup(consoммateur);

                                en exclusion mutuelle.
```

Solutions (II)

- Exécuter le code :

```
compteur--;
if(compteur == N-1)
    wakeup(producteur);
```

et le code :

```
compteur++;
if(compteur == 1)
    wakeup(consoммateur);
```

de façon non interruptible, atomique.

Plan

- 1 L'exclusion mutuelle sans attente
 - Les primitives sleep et wakeup
 - Les sémaphores et les mutex
 - Les moniteurs
- 2 Aperçu des interfaces SysV IPC et POSIX threads
 - Les sémaphores dans le System V IPC
 - Les threads POSIX
- 3 La primitive select
 - La primitive select

Les sémaphores

Un sémaphore est une variable
sem de type entier

- avec une opération down t.q. :
si sem == 0 et un processus fait down(sem),
alors le processus s'endort.
Après : sem == 0.
- avec une opération up :
si sem == 0 et un processus fait up(sem),
alors le processus réveille un autre processus à hasard,
endormi sur ce sémaphore,
Après : sem == 1.

Les opérations down et up sont atomiques.

Un mutex et un sémaphore qui
prend toujours un valeur entre 0,1.

Programme : le producteur-consommateur avec les sémaphores

```
1 : #define N 100
2 :
3 : semaphore mutex, vide = N, plein = 0;
4 :
5 : producteur()
6 : {
7 :     while (1)
8 :     {
9 :         down(vide);
10 :        down(mutex);
11 :        section_critique_producteur();
12 :        up(mutex);
13 :        up(plein);
14 :    }
15 : }
16 :
17 : consommateur()
18 : {
19 :     while (1)
20 :     {
21 :         down(plein);
22 :         down(mutex);
23 :         section_critique_consommateur();
24 :         up(mutex);
25 :         up(vide);
26 :     }
27 : }
```

Plan

- 1 L'exclusion mutuelle sans attente
 - Les primitives sleep et wakeup
 - Les sémaphores et les mutex
 - Les moniteurs
- 2 Aperçu des interfaces SysV IPC et POSIX threads
 - Les sémaphores dans le System V IPC
 - Les threads POSIX
- 3 La primitive select
 - La primitive select

Problèmes

- Pas évident à s'en servir :
- Si on échange les lignes 9-10, le système se bloque.
- Comment démontrer que le code ne se bloque jamais ?
(Dans ce cas, et dans le cas général).

Les moniteurs

Morceaux de code contenant :

- variables partagées,
- une suite de fonctions :
 - le compilateur nous assure :
 - il y a toujours aux plus un processus en train d'exécuter une de ces fonctions
 - (il assure l'exclusion mutuelle),
- variables de conditions :
 - un processus peut s'endormir
 - wait(condition) –
 - si cette variable est vraie,
 - un processus peut signaler cette variable
 - signal(condition) –
 - il réveille un processus à hasard endormi sur cette variable,
 - il sort du code du moniteur.

Problème : existence des moniteurs dépende
des langages et des compilateurs.

Plan

- 1 L'exclusion mutuelle sans attente
 - Les primitives sleep et wakeup
 - Les sémaphores et les mutex
 - Les moniteurs
- 2 Aperçu des interfaces **SysV IPC** et **POSIX threads**
 - Les sémaphores dans le System V IPC
 - Les threads POSIX
- 3 La primitive select
 - La primitive select

Programme : producteur-consommateur sous IPC

```
1 : #include <sys/sem.h>
2 : #include <unistd.h>
3 : #include <stdio.h>
4 : #include <stdlib.h>
5 : #include <time.h>
6 : #include <sys/fcntl.h>
7 :
8 : /* Gestion des sémaphores Sys V IPC*/
9 : #define NOSEM 0
10 : struct sembuf plus = { NOSEM, 1, 0 };
11 : struct sembuf moins = { NOSEM, -1, 0 };
12 :
13 : #define SEMCREATE() semget(IPC_PRIVATE,1,0600)
14 : #define DOWN(semid) semop(semid,&moins,1)
15 : #define UP(semid) semop(semid,&plus,1)
16 : union semun {
17 :     int val;
18 :     struct semid_ds *buf;
19 :     unsigned short *array;
20 : } arg;
21 : int valeur(int semid)
22 : {
23 :     return semctl(semid, NOSEM, GETVAL, arg);
24 : }
25 : int semset(int semid, int n)
26 : {
27 :     arg.val = n;
28 :     return semctl(semid, NOSEM, SETVAL, arg);
29 : }
30 :
```

Le IPC, ou « Inter Process Communication »

Permet de :

- Partager des segments de mémoire entre processus.
- Utiliser des files de messages (« Message Queues »):
Message Queue :
 - Liste chaîné de messages,
chacune d'un longueur fixé maximale.
 - Les messages sont typées.
 - Lecture et écriture de type FIFO.
- Utiliser des ensembles de sémaphores.

semget

```
#include <sys/sem.h>
int semget(key_t key, int nb_sem, int flags);
```

key : IPC_PRIVATE : permet la communication entre processus de la même famille.

Sinon, deux processus doivent se mettre d'accord sur un clé commune.

nb_sem : nombre de sémaphores dans l'ensemble.

flags : permissions lecture/écriture.

Autres opérations

```
#include <sys/sem.h>
```

```
int semctl(int semid, int nb, int cmd, ...);
```

semid : quel ensemble

nb : quel sémaphore dans l'ensemble

cmd : GETVAL, SETVAL, ...

```
int semop(int semid, struct sembuf * sops, size_t nsops);
```

semid : quel ensemble

sops : tableau d'opérations à faire de façon atomique

nsops : dimension de ce tableau

Programme : prodcons.c

```
31 : /* Voici le consommateur/producteur */
32 : #define NB_CASES 10
33 : #define DELAIMAX 2
34 :
35 : void producteur(void);
36 : void consommateur(void);
37 : int mutex, vide, plein;
38 :
39 : int main(void)
40 : {
41 :     if ((mutex = SEMCREATE()) == -1 || (vide = SEMCREATE()) == -1
42 :         || (plein = SEMCREATE()) == -1)
43 :         exit(EXIT_FAILURE);
44 :
45 :     if ((semset(mutex, 1) == -1) || (semset(vide, NB_CASES) == -1)
46 :         || (semset(plein, 0) == -1))
47 :         exit(EXIT_FAILURE);
48 :
49 :     printf("Au debut : mutex %d, vide : %d, cases pleines : %d.\n",
50 :         valeur(mutex), valeur(vide), valeur(plein));
51 :
52 :     switch (fork())
53 :     {
54 :     case -1:
55 :         exit(EXIT_FAILURE);
56 :     case 0:
57 :         producteur();
58 :     default:
59 :         consommateur();
60 :     }
61 :     exit(EXIT_FAILURE);
62 : }
63 :
```

Programme : prodcons.c

```
64 : void producteur(void)
65 : {
66 :     int pid = getpid();
67 :     srand(pid * time(NULL));
68 :     while (1)
69 :     {
70 :         sleep(rand() % DELAIMAX);
71 :         DOWN(vide);
72 :         DOWN(mutex);
73 :         sleep(rand() % DELAIMAX);
74 :         UP(mutex);
75 :         UP(plein);
76 :         printf("[%d] Cases vides : %d, cases pleines : %d.\n", pid,
77 :             valeur(vide), valeur(plein));
78 :     }
79 : }
80 :
81 : void consommateur(void)
82 : {
83 :     int pid = getpid();
84 :     srand(getpid() * time(NULL));
85 :     while (1)
86 :     {
87 :         sleep(rand() % DELAIMAX);
88 :         DOWN(plein);
89 :         DOWN(mutex);
90 :         sleep(rand() % DELAIMAX);
91 :         UP(mutex);
92 :         UP(vide);
93 :         printf("[%d] Cases vides : %d, cases pleines : %d.\n", pid,
94 :             valeur(vide), valeur(plein));
95 :     }
96 : }
```

Session : prodcons

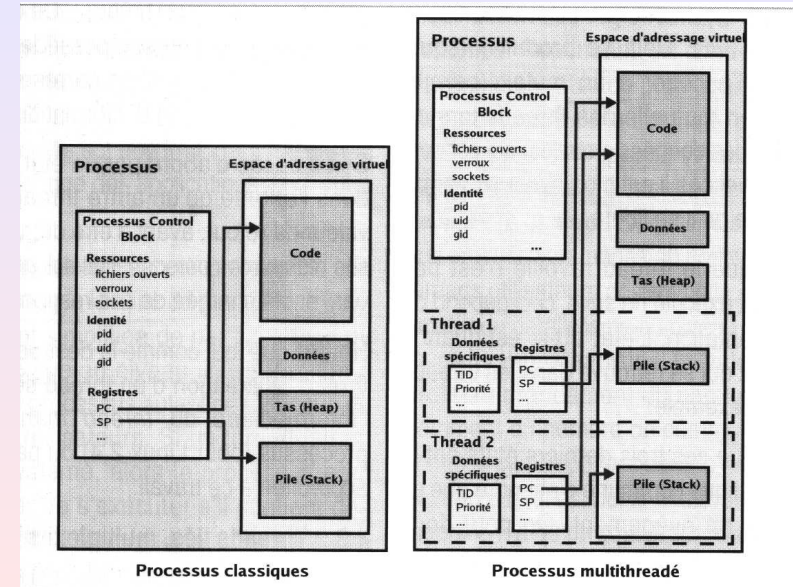
```
[lsantoca@localhost lecture10]$ a.out
Au debut : mutex 1, vide : 10, cases pleines : 0.
[7055] Cases vides : 9, cases pleines : 0.
[7055] Cases vides : 8, cases pleines : 1.
[7054] Cases vides : 9, cases pleines : 1.
[7055] Cases vides : 8, cases pleines : 2.
[7055] Cases vides : 7, cases pleines : 3.
[7054] Cases vides : 8, cases pleines : 2.
[7054] Cases vides : 8, cases pleines : 1.
[7055] Cases vides : 8, cases pleines : 1.
[7054] Cases vides : 8, cases pleines : 1.
[7055] Cases vides : 8, cases pleines : 1.
[7054] Cases vides : 8, cases pleines : 1.
[7055] Cases vides : 8, cases pleines : 2.
[7055] Cases vides : 7, cases pleines : 2.
[7054] Cases vides : 7, cases pleines : 2.
[7055] Cases vides : 7, cases pleines : 2.
[7054] Cases vides : 7, cases pleines : 2.
[7055] Cases vides : 7, cases pleines : 2.
[7054] Cases vides : 7, cases pleines : 2.
[7055] Cases vides : 7, cases pleines : 2.
[7054] Cases vides : 7, cases pleines : 2.
```

Plan

- 1 L'exclusion mutuelle sans attente
 - Les primitives sleep et wakeup
 - Les sémaphores et les mutex
 - Les moniteurs
- 2 Aperçu des interfaces **SysV IPC** et **POSIX threads**
 - Les sémaphores dans le System V IPC
 - Les threads POSIX
- 3 La primitive select
 - La primitive select

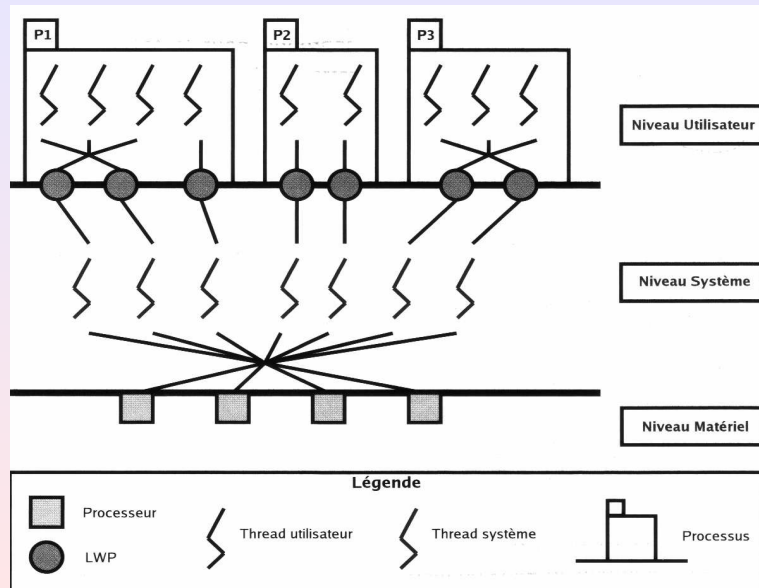
Threads vs. processus

(GNU Linux magazine)



Les threads et le système

(GNU Linux magazine)



Programme : partage_ress.c

```
1 : #include <pthread.h>
2 : #include <stdio.h>
3 : #include <stdlib.h>
4 : #include <unistd.h>
5 :
6 : #define MAX_RESOURCE 7
7 : #define NB_THREADS 3
8 :
9 : pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
10 : pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
11 :
12 : unsigned int resources = MAX_RESOURCE;
13 : unsigned int waiting = 0;
14 :
15 : void get_resources(int amount)
16 : {
17 :     pthread_mutex_lock(&mutex);
18 :     while (resources < amount)
19 :     {
20 :         waiting++;
21 :         pthread_cond_wait(&cond, &mutex);
22 :     }
23 :     resources -= amount;
24 :     pthread_mutex_unlock(&mutex);
25 : }
26 :
27 : }
28 :
```

Programme : partage_ress.c

```

29 : void free_resources(int amount)
30 : {
31 :     pthread_mutex_lock(&mutex);
32 :
33 :     resources -= amount;
34 :     if (waiting > 0)
35 :     {
36 :         waiting = 0;
37 :         pthread_cond_broadcast(&cond);
38 :     }
39 :
40 :     pthread_mutex_unlock(&mutex);
41 : }
42 :
43 : void *fonction()
44 : {
45 :     int i, tid, result;
46 :
47 :     tid = (int) pthread_self();
48 :     for (i = 3; i > 0; i--)
49 :     {
50 :         printf("thread %d : je demande %d resources\n", tid, i);
51 :         get_resources(i);
52 :         sleep(1);
53 :         printf("thread %d : je libre %d resources\n", tid, i);
54 :         free_resources(i);
55 :     }
56 :     result = 2;
57 :
58 :     pthread_exit((void *) result);
59 : }

```

Programme : partage_ress.c

```

60 :
61 : int main()
62 : {
63 :     int i;
64 :     void *retour;
65 :     pthread_t thread[NB_THREADS];
66 :
67 :     for (i = 0; i < NB_THREADS; i++)
68 :         pthread_create(&thread[i], NULL, fonction, NULL);
69 :
70 :     for (i = 0; i < NB_THREADS; i++)
71 :     {
72 :         pthread_join(thread[i], &retour);
73 :         printf("thread main : le thread %d se termine (result %d)\n",
74 :             (int) thread[i], (int) retour);
75 :     }
76 :
77 :     pthread_exit(NULL);
78 : }

```

Session : partage_ress

```

[lsantoca@localhost lecture10]$ a.out
thread 16386 : je demande 3 resources
thread 32771 : je demande 3 resources
thread 49156 : je demande 3 resources
thread 16386 : je libre 3 resources
thread 16386 : je demande 2 resources
thread 32771 : je libre 3 resources
thread 32771 : je demande 2 resources
thread 16386 : je libre 2 resources
thread 16386 : je demande 1 resources
thread 32771 : je libre 2 resources
thread 32771 : je demande 1 resources
thread 49156 : je libre 3 resources
thread 49156 : je demande 2 resources
thread 16386 : je libre 1 resources
thread 32771 : je libre 1 resources
thread 49156 : je libre 2 resources
thread 49156 : je demande 1 resources
thread main : le thread 16386 se termine (result 2)
thread main : le thread 32771 se termine (result 2)
thread 49156 : je libre 1 resources
thread main : le thread 49156 se termine (result 2)

```

Plan

- 1 L'exclusion mutuelle sans attente
 - Les primitives sleep et wakeup
 - Les sémaphores et les mutex
 - Les moniteurs
- 2 Aperçu des interfaces SysV IPC et POSIX threads
 - Les sémaphores dans le System V IPC
 - Les threads POSIX
- 3 La primitive select
 - La primitive select

select

```
#include <sys/select.h>
int select(int maxfd, fd_set * read, fd_set * write, fd_set * err,
struct timeval * t_out);
```

read : (pointeur à une structure de) ensemble de descripteurs de fichiers ouverts en lecture dont on s'en met en écoute.
Au retour cette structure est remplie avec les descripteurs prêts (non bloquantes).
NULL si on est pas intéressé à un tel ensemble.

write : ensemble de descripteurs de fichiers ouverts en lecture dont on s'en met en écoute.
Au retour cette structure est remplie avec les descripteurs prêts.
NULL si pas intéressé.

err : ensemble de descripteurs de fichiers (possiblement contenant un erreur) dont on s'en met en écoute.
Au retour cette structure est remplie avec les descripteurs prêts.
NULL si pas intéressé.

t_out : on se débloque après * *t_out*.
Si NULL on reste bloqué.

Retourne : no. descripteurs de fichier prêts, 0 si on est débloqué par *t_out*, -1 si erreur.

La structure timeval

```
struct timeval {
    time_t      tv_sec ;      /* Seconds. */
    suseconds_t tv_usec ;     /* Microseconds. */
}
```

Les ensembles de descripteurs

```
#include <sys/select.h>
```

`void FD_ZERO(fd_set * ens);`
Sommaire : *ens* devient l'ensemble vide

`void FD_SET(int fd, fd_set * ens);`
Sommaire : add *fd* à *ens*

`void FD_CLEAR(int fd, fd_set * ens);`
Sommaire : efface *fd* de *ens*

`int FD_ISSET(int fd, fd_set * ens);`
Sommaire : vérifie si *fd* appartient à *ens*

Programme : exempleselect.c

```
1 : #include <stdio.h>
2 : #include <stdlib.h>
3 : #include <unistd.h>
4 : #include <sys/stat.h>
5 : #include <sys/fcntl.h>
6 : #include <sys/select.h>
7 : #include <signal.h>
8 :
9 : #define TUBE "fifo"
10 : #define ERR 1
11 : #define OK 0
12 : #define NOTUBES 3
13 : #define TIMEOUT 60
14 : #define POURTOUTTUBE(i) for(i=0;i<NOTUBES;i++)
15 : #define MAX(n,m) ((n < m)?m:n)
16 :
17 : int fd[NOTUBES];
18 : char nontube[NOTUBES][256];
19 :
20 : void sortir(char *message, int err)
21 : {
22 :     int i;
23 :
24 :     POURTOUTTUBE(i) unlink(nontube[i]);
25 :
26 :     if (err)
27 :     {
28 :         perror(message);
29 :         exit(EXIT_FAILURE);
30 :     }
31 :     printf("Terminaison du serveur : %s\n", message);
32 :     exit(EXIT_SUCCESS);
33 : }
```

Programme : exempleselect.c

```

34 :
35 : void handler(int sig)
36 : {
37 :     sortir("Signaled", ERR);
38 : }
39 :
40 : void traiter(fd_set * readfds)
41 : {
42 :     int i, n;
43 :     char tampon[256];
44 :
45 :     POURTOUTTUBE(i)
46 :     {
47 :         if (FD_ISSET(fd[i], readfds))
48 :         {
49 :             n = read(fd[i], tampon, sizeof(tampon) - 1);
50 :             tampon[n] = '\0';
51 :             printf("Lu %d caractères de %s : \n", n, nomtube[i]);
52 :             printf("%s\n", tampon);
53 :         }
54 :     }
55 : }
56 :

```

Programme : exempleselect.c

```

57 : int main(void)
58 : {
59 :     int i, n, maxfd = 0;
60 :     mode_t mode;
61 :     fd_set read;
62 :     struct timeval tv;          /* Cette structure définie dans select.h */
63 :
64 :     for (i = 1; i < NSIG; i++)
65 :         signal(i, handler);
66 :
67 :     POURTOUTTUBE(i) sprintf(nomtube[i], "%s%d", TUBE, i + 1);
68 :
69 :     mode = S_IRUSR | S_IWUSR | S_IWGRP | S_IWOTH;
70 :
71 :     POURTOUTTUBE(i) if (mkfifo(nomtube[i], mode) == -1
72 :                         || chmod(nomtube[i], mode) == -1)
73 :         sortir("mkfifo/chmod", ERR);
74 :
75 :     POURTOUTTUBE(i) if ((fd[i] = open(nomtube[i], O_RDONLY | O_NONBLOCK))
76 :                         == -1)
77 :         sortir("open", ERR);
78 :     else
79 :     {
80 :         maxfd = MAX(fd[i], maxfd);
81 :         /* Prochaine ligne : rendre la lecture bloquante */
82 :         fcntl(fd[i], F_SETFL, fcntl(fd[i], F_GETFL) & !O_NONBLOCK);
83 :     }
84 :

```

Programme : exempleselect.c

```

85 :     for (;;)
86 :     {
87 :         tv.tv_sec = TIMEOUT;      /* Secondes */
88 :         tv.tv_usec = 0;          /* Microsecondes */
89 :         FD_ZERO(&read);
90 :         POURTOUTTUBE(i) FD_SET(fd[i], &read);
91 :
92 :         n = select(maxfd + 1, &read, NULL, NULL, &tv);
93 :         switch (n)
94 :         {
95 :             case -1:
96 :                 sortir("select", ERR);
97 :             case 0:
98 :                 sortir("attente trop longue", OK);
99 :             default:
100 :                 traiter(&read);
101 :         }
102 :     }
103 : }

```