

1. PRINCIPE DES SEMAPHORES SOUS UNIX

Pour créer un sémaphore, l'utilisateur doit lui associer **une clé**. le système lui renvoie un identificateur de sémaphore auquel sont attachés **n** sémaphores numérotés de 0 à n-1. Pour spécifier un sémaphore parmi les n, l'utilisateur indique l'identificateur et le numéro du sémaphore.

A chaque identificateur de sémaphore sont associés des **droits d'accès**. Ces droits sont nécessaires pour effectuer des opérations sur les sémaphores, mais inopérants pour :

- la destruction d'un identificateur de sémaphore,
- la modification des droits d'accès.

Seul le créateur, le propriétaire ou le super-utilisateur peuvent réaliser ces opérations.

1.1 Le fichier <sys/sem.h>

A chaque ensemble de sémaphores sont associées les structures `semid_ds` (une par ensemble de sémaphores), `__sem` et `sembuf` (une par sémaphore) décrites dans `<sys/sem.h>`

```
struct __sem    /* correspond à la structure d'un sémaphore dans le noyau */
{
    unsigned short semval;    /* valeur du sémaphore */
    unsigned short sempid;    /* pid du dernier processus utilisateur */
    unsigned short semcnt;
    /* nombre de processus attendant l'incrément du sémaphore */
    unsigned short semzcnt;
    /* nombre de processus attendant que semval soit nul */
};

struct sembuf   /* correspond à une opération sur un sémaphore */
{
    unsigned short sem_num;    /* n° de sémaphore : 0,... */
    short sem_op;             /* opération sur le sémaphore */
    short sem_flg;           /* option */
};

struct semid_ds
/* correspond à la structure d'une entrée dans la table des sémaphores */
{
    struct ipc_perm sem_perm; /* les droits */
    struct __sem * sem_base; /* pointeur sur le premier sémaphore de l'ensemble */
    time_t sem_otime;        /* date dernière opération par semop */
    ushort sem_nsems;        /* nombre de sémaphores de l'ensemble */
    time_t sem_ctime;        /* date dernière modification par semctl */
};
```

1.2 La fonction semget

int semget (key_t cle, int nb_sem, int option)

nb_sem : nombre de sémaphores de l'ensemble

cle : si **cle** = IPC_PRIVATE, un nouvel ensemble de sémaphores est créé sans clé d'accès

sinon, il y a appel à **ftok**.

si **cle** ne correspond pas à un ensemble de sémaphores existant

si IPC_CREAT n'est pas positionné : erreur et retour de -1

sinon, un nouvel ensemble de sémaphores est créé et associé à cette clé. Les droits d'accès sont ceux mentionnés; le propriétaire et créateur est le propriétaire effectif du processus; le groupe propriétaire et créateur est le groupe propriétaire effectif du processus

sinon si IPC_CREAT **et** IPC_EXCL sont tous deux positionnés :

erreur et retour de -1

sinon la fonction retourne l'identificateur de l'ensemble de sémaphores

option : on peut combiner par | des droits d'accès et les constantes suivantes :

SEM_A : permission de modification

SEM_R : permission en lecture

IPC_CREAT , IPC_EXCL

La fonction : **key_t ftok (char * path , int id)** utilise une chaîne de caractères (en principe un nom de fichier unique dans le système), la combine à **id** pour générer une clé unique dans le système, sous forme d'un **key_t** (équivalent à un entier long).

Exemple : récupération de l'identificateur d'un ensemble de 5 sémaphores

```
num = semget (ftok (path, cle), 5, 0)
```

Exemple : /* création de 4 sémaphores associés à la clé 456 */

```
#include <errno.h>
#define CLE 456
main ()
{
int semid ; /* identificateur des sémaphores */
char *path = "nom_de_fichier_existant";
if (( semid = semget (ftok (path, (key_t) CLE) , 4, IPC_CREAT | IPC_EXCL | SEM_R |
SEM_A)) == -1)
{
perror ("échec de semget\n");
exit (1);
}
printf (" identificateur de l'ensemble : %d\n", semid);
printf ("clé de l'ensemble : %d\n", ftok (path, (key_t) CLE));
}
```

1.3 La fonction semctl

int semctl (int semid, int semnum, int cmd, arg)

semid : identificateur de l'ensemble de sémaphores

semnum : numéro du sémaphore traité

```

union semun
{
    int val;
    struct semid_ds *buf;
    ushort array [ <nb de sémaphores>];
} arg;

```

cmd : paramètre de commande pouvant prendre 10 valeurs :

GETVAL : retourne la valeur du sémaphore de n° **semnum**

SETVAL : le sémaphore de n° **semnum** reçoit la valeur **arg.val**

GETPID : retourne le pid du processus qui est intervenu en dernier lieu

GETNCNT : retourne le nombre de processus en attente d'une incrémentation de la valeur du sémaphore de n° **semnum**

GETZCNT : retourne le nombre de processus en attente du passage à 0 du sémaphore de n° **semnum**

GETALL : range les valeurs de tous les sémaphores dans le tableau pointé par

arg.array

SETALL : positionne les valeurs de tous les sémaphores à l'aide des valeurs du tableau pointé par **arg.array**

IPC_STAT : la structure associée à semid est rangée à l'adresse contenue dans

arg.buf

IPC_SET : positionne les valeurs gid, uid et mode à partir de la structure pointée par **arg.buf**

IPC_RMID : détruit le sémaphore

1.4 La fonction semop

int semop (int semid, struct sembuf (* sops) [], int nbops)

retourne semval du dernier sémaphore manipulé ou -1 en cas d'erreur

semid : identificateur de l'ensemble de sémaphores

sops : pointeur vers un tableau de **nbops** structures de type sembuf (cf. page 1)

Pour chaque sémaphore de n° **sem_num** (0 à nbops-1), on indique dans le champ **sem_op** l'opération à effectuer avec un code entier :

si **sem_op** > 0 : **semval** = **semval** + **sem_op**

si **sem_op** = 0 : si **semval** = 0, rien n'est effectué

sinon le processus est endormi en attendant la nullité de **semval**

si **sem_op** < 0 : si **semval** >= |**sem_op**|, alors **semval** ← **semval** - |**sem_op**|

sinon, le processus est endormi jusqu'à : **semval** >= |**sem_op**|

Pour le champ **sem_flg** :

s'il vaut **IPC_NOWAIT**, évite le blocage de processus et retourne un code d'erreur

s'il vaut **SÉM_UNDO**, on évite de bloquer indéfiniment des processus sur des sémaphores après la mort accidentelle d'un processus. Très coûteux en temps CPU et en mémoire

2. LES OPERATEURS P ET V (WAIT et SIGNAL au sens de DIJKSTRA)

```
/* Appelons ce fichier par exemple : "Dijkstra.h"
   Il est évidemment à inclure dans le programme utilisateur*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
/*****/
int sem_create (key_t CLEF, int initval)
/* crée un ensemble d'un seul sémaphore dont la clé est reliée à CLEF, de valeur initiale initval
   retourne l'identifiant associé au sémaphore si OK, sinon retourne -1
   Si l'ensemble associé à CLEF existe déjà, la fonction part en échec.
   Les droits sont positionnés de sorte que tout processus peut modifier le sémaphore */
{
    union semun {
        int val;
        struct semid_ds *buf;
        ushort *array;
    } arg_ctl;
    int semid;
    semid = semget ( flock ("Dijkstra.h-<votre login>", CLEF), 1, IPC_CREAT | IPC_EXCL | 0666);
    if (semid == -1) return -1;
    arg_ctl.semval = initval;
    if (semctl (semid, 0, SETVAL, arg_ctl) == -1) return -1;
    return semid;
}
/*****/
int sem_delete ( int semid)
/* supprime l'ensemble de sémaphores identifiés par semid */
{
    return semctl (semid , 0 , IPC_RMID , 0) ;
}
/*****/
int P (int semid)
{
    struct sembuf sempar;
    sempar.sem_num = 0;
    sempar.sem_op = -1;
    sempar.sem_flg = SEM_UNDO;
    return semop (semid , &sempar , 1);
}
/*****/
int V (int semid)
{
    struct sembuf sempar;
    sempar.sem_num = 0;
    sempar.sem_op = 1;
    sempar.sem_flg = SEM_UNDO;
    return semop (semid , &sempar , 1);
}
}
```

En outre, la commande shell **ipcs** permet d'obtenir des informations sur les objets de type ensemble de sémaphores (et plus généralement tous les IPC), avec l'option :

- s pour se limiter aux sémaphores
- m pour se limiter aux segments de mémoire
- q pour se limiter aux files de messages
- c pour faire apparaître l'identité de l'utilisateur créateur

Elle fournit :

T : type de l'objet (q, m ou s)
ID : identification interne de l'objet
KEY : clé de l'objet en hexadécimal
MODE : droits d'accès à l'objet. Les 2 premiers des 11 caractères indiquent des informations spécifiques à une file de messages ou à un segment de mémoire partagée
OWNER : identité du propriétaire
GROUP : identité du groupe propriétaire

3. EXEMPLE DE PROGRAMMATION CONCURRENTE

```
main ()
{
    int i, CLE = 33, mutex = 0;
    if ((mutex = sem_create ( CLE, 1)) == -1)
    {
        perror ("problème de création du sémaphore ");
        exit (-1);
    }
    printf ("création du sémaphore d'identifiant %d\n", mutex);
    if (fork () == 0)
    {
        printf ("je suis le processus %d, fils de %d\n", getpid (), getppid ());
        for (i = 0; i < 3 ; i++)
        {
            P (mutex);
            printf ("le fils entre dans sa section critique\n");
            sleep (15);
            printf ("le fils sort de sa section critique\n");
            V (mutex);
        }
        exit (0);
    }
    else {
        for (i = 0 ; i < 4 ; i++)
        {
            P (mutex);
            printf ("le pere entre dans sa section critique\n");
            sleep (10);
            printf ("le pere sort de sa section critique\n");
            V (mutex);
        }
        wait (0);
        printf ("FIN !\n");
        sem_delete (mutex);
    }
}
```