TP1- Les sémaphores (1)

UNIX (LINUX) propose des fonctions pour travailler (initialisation, primitives P et V...) sur un **ensemble** de sémaphores **regroupés dans un tableau**.

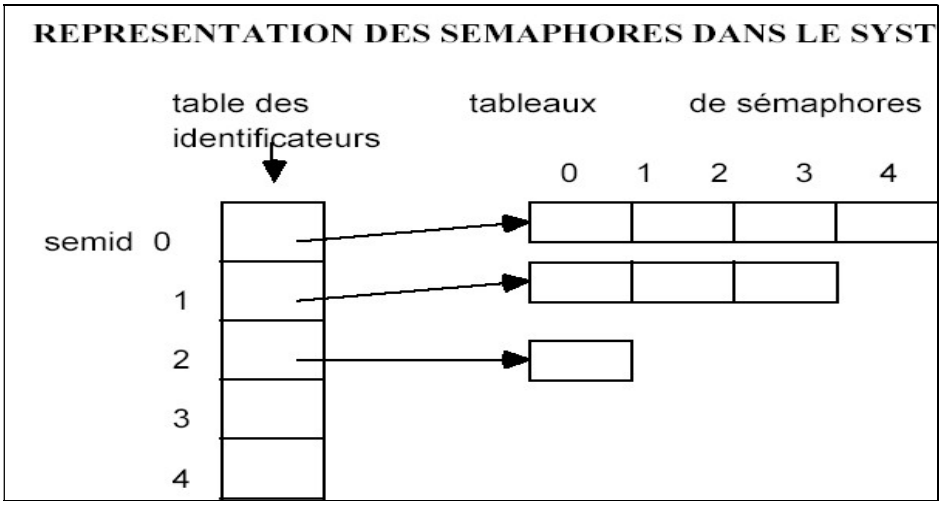
1. Création d'un tableau de sémaphores: semget

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

```
int semget (key_t  Clé, int N, int Options)
```

Cette primitive retourne l'identificateur de l'ensemble de **N** sémaphores associés à la clé **Clé** (entier ou **IPC_PRIVATE**). En cas d'échec, la fonction retourne **-1**.

Options contient:
- Les droits d'accès à l'ensemble des sémaphores créés.
- Les conditions de création:
  - **IPC_CREAT**: crée le sémaphore et retourne l'identificateur. Si le sémaphore est déjà crée par un autre processus retourne l'identificateur.
  - **IPC_ALLOC**: retourne l'identificateur d'un sémaphore déjà crée et retourne **-1** si le sémaphore n'est pas encore crée.
  - **IPC_EXCL**: avec **IPC_CREAT** permet de ne demander une création que si le sémaphore n'existe pas déjà, retourne **-1** si le sémaphore existe déjà.



Exemple: Création d'un sémaphore

```
int  semid;
semid = semget(12, 1, IPC_CREAT|IPC_EXCL|0666);
if (semid == -1)    semid = semget (12, 1, IPC_ALLOC|0666);

……..    /*initialisation du sémaphore ( voir  semctl)*/
```

Remarque: Pour s'assurer de la création d'un nouvel sémaphore, on utilise la combinaison **IPC_CREAT|IPC_EXCL|*droits*** pour le paramètre **Options** et **IPC_PRIVATE** pour le paramètre **Clé**.

Exemple: Création d'un sémaphore

```
int semid;
semid = semget (IPC_PRIVATE, 1, IPC_CREAT|IPC_EXCL|0666);
```

2. Contrôle des émaphores: semctl

```
int semctl (int semid, int semnum, int cmd, arguments de l'opération)
```

Cette primitive effectue la commande ***cmd*** sur l'ensemble des sémaphores identifié par ***semid*** (ou sur le sémaphore d'indice ***semnum***, selon l'opération). Le premier sémaphore a l'indice **0**.
Les commandes possibles sont:

- **SETVAL**: initialise le sémaphore de numéro **semnum** avec la valeur indiquée comme argument.
- **SETALL**: initialise tous les sémaphores du tableau à l'aide des valeurs indiquées comme arguments.
- **IPC_RMID**: supprime le sémaphore de **semid** donné du système.

Exemple: Initialisation à **1** (la commande **SETVAL**) du sémaphore crée dans l'exemple précédent.

```
semctl (semid, 0, SETVAL, 1);
```

3. Opérations sur les sémaphores: semop

Les deux primitives **P** et **V** sur les sémaphores sont réalisées par la fonction **semop**.

```
int semop(int semid, struct sembuf *Op, unsigned int N)
```

Elle est utilisée pour réaliser un tableau **Op** de **N** opérations (P ou V) sur un ensemble de sémaphores indiqué par **semid**. **Op** est un tableau de **N** structures **sembuf**.

La structure **sembuf**

```
struct sembuf
    {  unsigned short int  sem_num;  /* numéro du sémaphore*/
       short    sem_op;              /* opération du sémaphore:  -1 (ou < 0) → l'opération P
                                                                  1 (ou > 0) → l'opération V  */
       short  sem_flg;  /*options */  }
```

Sem_flg vaut: - **IPC_NOWAIT**: permet au processus de ne pas se bloquer sur une "ressource" indisponible.
- **SEM_UNDO** : si un processus est tué dans sa partie critique le système libère la "ressource".

Les **N** opérations placées dans le tableau **Op** sont réalisées **atomiquement** (Le noyau gère l'atomicité).

Exemple: Effectuer l'opération **P** sur le sémaphore créé et initialisé dans les exemples précédents.

```
struct sembuf   operation;  // Une seule opération

operation.sem_num = 0;     // Le sémaphore visé est celui d'indice 0
operation.sem_op = -1;     // Pour faire l'opération P
operation.sem_flg = 0;     // Ou  operation.sem_flg = SEM_UNDO

semop (semid, &operation, 1);
```

Opération P sur le premier sémaphore du tableau de sémaphores dont l'identifiant est id

```
void   P (int   id)   // id est l'identifiant d'un tableau de sémaphores
{
struct  sembuf   operation;  // Une seule opération
operation.sem_num = 0;     // Le sémaphore visé est celui d'indice 0
operation.sem_op = -1;     // Pour faire l'opération P
operation.sem_flg = 0;     // Ou  operation.sem_flg = SEM_UNDO
semop (id, &operation, 1);
}
```

Remarques:

1) Pour plus d'informations sur les primitives de sémaphores, utilisez la commande d'aide d'UNIX: man
   Exemple: **man semop**

2) Pour consulter la liste des ensembles de sémaphores crées, utilisez la commande **ipcs** d'UNIX.

Exercices

1) Soit le programme suivant :

```
void msg1( )
{
  printf ("Salut je suis le processus père \n");
  sleep (3);
  printf ("C'est encore moi, le processus père \n");
}
void msg2( )
{
  printf ("Salut je suis le processus fils \n");
  sleep(3);
  printf ("C'est encore moi, le processus fils \n");
}
main( )
{
    int semid, pid;
    pid = fork ( );
    if (pid == 0)
                    msg2 ( );
   else
                    msg1 ( );
}
```

Compiler et exécuter le programme.

2) Modifier le programme afin que le processus père termine l'exécution de la fonction msg1( ), et le processus fils celle de msg2( ) sans interruption.

Indications :

- Utiliser un sémaphore initialisé à **1** (un seul processus doit exécuter sa fonction).
- Chaque processus doit effectuer :
    - L'opération **P** avant d'exécuter sa fonction.
    - L'opération **V** après l'exécution.

3) Modifier le programme (2) pour trois processus : le père et deux fils.

4) Modifier le programme (3) afin que les trois processus s'exécutent selon un ordre bien déterminé, par exemple fils 2, fils 1, père.

5)

A. Créer un programme dans lequel :

- On crée un sémaphore binaire initialisé à 0 en utilisant la clé 12345
- On affiche un premier message
- On applique la primitive P sur le sémaphore crée
- On affiche un deuxième message

B. Créer un autre programme dans lequel :

- On récupère l'identifiant du sémaphore dont la clé est 12345
- On affiche un message
- On applique un V sur le sémaphore récupérée.
- On fait un sleep(2)
- On affiche un message

C. Compiler et lancer le premier programme dans terminal

D. Compiler et lancer le deuxième programme dans un autre terminal.