Les sémaphores UNIX

L'implantation des sémaphores sous Unix est beaucoup plus riche que la simple implémentation des opérations P et V .

- Les opérations P_n et V_n sont implantées. Elles permettent d'obtenir ou de libérer n occurrences de la ressource et ceci de manière atomique.
- Unix implante les ensembles de sémaphores. Un processus peut obtenir ou libérer n occurrences de la ressource R_1 et m occurrences de la ressource R_2 . Le modèle s'en trouve considérablement enrichi.
- Cette implantation propose également une nouvelle opération appelée Z qui attend qu'un sémaphore devienne nul.

La primitive de création d'un ensemble de sémaphores est la suivante :

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

Le fonctionnement de la primitive et le rôle des paramètres `cle` et `option` sont les mêmes que pour la primitive `shm_get`. La variable `nb_sem` correspond au nombre de sémaphores souhaités dans l'ensemble.

Une opération sur un sémaphore est décrite grâce à la structure `sembuf`

```
struct sembuf {
    ushort sem_num;    /* numéro du sémaphore */
    short  sem_op;     /* opération sur le sémaphore */
    short  sem_flg;    /* option */
};
```

`sem_num` correspond au numéro du sémaphore dans l'ensemble. La numérotation commence à 0. La valeur du champ `sem_op` détermine la nature de l'opération :

- > 0 : opération V_n . La valeur du sémaphore est augmentée de `sem_op`. Au plus `sem_op` processus en attente sur le sémaphore sont réveillés.
- < 0 : opération P_n . Si l'opération n'est pas possible, le processus est bloqué. Si l'opération est possible, le sémaphore est diminué de `|sem_op|`. S'il devient nul, tout les processus en attente de la nullité sont réveillés.
- = 0 : opération Z . Si le sémaphore n'est pas nul, alors le processus est bloqué.

La variable `sem_flg` représente une conjonction des options possibles. Parmi les options, nous distinguons les deux cas suivants :

`IPC_NOWAIT` : qui rend l'opération non bloquante (*man* pour plus de détails),

`SEM_UNDO` : qui annule l'effet de l'opération en cas de terminaison intempestive du processus.

Ces opérations peuvent être utilisées grâce à la commande

```
#include <sys/sem.h>
int semop (int semid, struct sembuf *tab, int n);
```

Dans ce cas `semid` est l'identificateur de l'ensemble de sémaphores, `tab` est un tableau d'opérations et `n` le nombre des opérations de ce tableau à prendre en compte.

L'ensemble des opérations est réalisé atomiquement. Si une opération ne peut être réalisée, alors aucune opération n'est effectuée et le processus est mis en sommeil.

L'opération de contrôle sur un ou plusieurs sémaphores d'un ensemble est la suivante :

```
#include <sys/sem.h>
int semctl (int semid, int semnum, int op , ... /*arg */);
```

Nous retrouvons tout d'abord les opérations classiques sur les I.P.C. Dans ce cas la variable `op` vaut :

`IPC_RMID` : l'ensemble de sémaphores identifié par `semid` est supprimé.

`IPC_STAT` : l'adresse d'une structure de type `semid_ds` doit être passée en argument pour en extraire un contenu.

`IPC_SET` : comme précédemment, mais cette fois, l'entrée dans la table des sémaphores est ajustée. Dans ce cas et le précédent `semnum` n'a pas de signification.

Nous trouvons également des cas spécifiques aux sémaphores, dont l'initialisation des sémaphores :

`SETVAL` : le sémaphore `semnum` de l'ensemble `semid` est initialisé à `arg`.

`SETALL` : les `semnum` premiers sémaphores de l'ensemble `semid` sont initialisés selon le contenu du tableau `arg`

Exercice 1 Écrivez les primitives `creer_sem`, `destruire_sem`, `P`, `V` et `Z` en utilisant les appels système proposés par Unix.

Exercice 2

1) Implémentez l'algorithme de la simulation d'un calcul parallèle avec les sémaphores Unix.

2) De même reprenez l'exercice du problème belge (regroupement des plus petites valeurs d'un tableau dans sa première moitié) et implémentez l'algorithme.