Contrôle Module Noyau **M1 – Master SAR**

Novembre 2006 **2 heures**

Tout document papier autorisé - Barème donné à titre indicatif

L. Arantes, P. Sens, Gaël Thomas

1. SYNCHRONISATION (55 MIN. 10 POINTS)

Considérez le code du Noyau **non préemptif** vu en TD. Nous voulons y ajouter un mécanisme de sémaphore.

Un sémaphore permet de contrôler l'accès à une ressource ayant *N* exemplaires. Cela dit, au maximum *N* processus peuvent accéder à cette ressource concurremment.

Les appels systèmes à offrir aux utilisateurs sont : *int Init_sem (int sem, int N)*, *int Lib_sem(sem)*, int *P(int sem)* et *int V(int sem)*. Ils permettent d'accéder à un objet sémaphore identifié par *sem* qui contrôle les exemplaires d'une ressource quelconque. Ces appels systèmes appellent respectivement les fonctions du noyau *int sys_Init_sem(int sem, int N)*, *int sys_Lib_sem(sem)*, int sys_*P(int sem)* et *int sys_V(int sem)* que nous voulons programmer.

Un sémaphore est un objet composé par :

(1) Un compteur *compt* qui contrôle le nombre d'exemplaires disponibles ou le nombre de processus en attente pour la ressource : si *compt* >=*0*, *compt* indique le nombre d'exemplaires courants disponibles ; si *compt* <*0*, *-compt* indique le nombre de processus en attente pour la ressource ;

(2) Une file *File* où sont mis les processus en attente pour un exemplaire de la ressource. La file *File* possède la taille **MAX_FILE** et est gérée de façon **circulaire (FIFO)**. La file est nécessaire parce que les **demandes pendantes doivent être satisfaites dans l'ordre de demande.**

Les fonctions pour manipuler un sémaphore identifié par un entier *sem* sont :

- ***int Init_sem (int sem,int N)*** : alloue le sémaphore identifié par *sem* en l'initialisant son compteur à N (nombre d'exemplaires de la ressource qui se trouvent disponibles). Une fois qu'un sémaphore a été initialisé, les autres processus qui veulent l'utiliser n'ont pas besoin (et ne doivent pas) l'initialiser (N >=0).

- **int *Lib_sem(int sem)*** **:** libère le sémaphore *sem* alloué précédemment. Pour simplifier nous considérons initialement que lorsque cette fonction est appelée, aucun exemplaire de la ressource n'est utilisé par un processus.

- ***int P(int sem)*** : permet à un processus de demander un exemplaire de la ressource gérée par *sem*. S'il y au moins un exemplaire de la ressource disponible (*compt* >*0*), un exemplaire est réservé au processus et la fonction se termine. Sinon, s'il y a de la place dans *File*, le processus est mis **à la fin** de cette file et il s'endort en attente d'un exemplaire disponible. La priorité de réveil est **PRISEM** dans ce cas. En se réveillant, le processus a le droit à un exemplaire de la ressource et la fonction alors se termine.

Code de renvoi : -1 : si le sémaphore n'avait pas été initialisé ou s'il n'y a pas de place dans *File* ;

0 : succès.

- ***int V (int sem)*** **:** libère **un** exemplaire de la ressource gérée par *sem*. De plus, s'il y a des processus dans *File* en attente pour un exemplaire, le **premier** processus de *File* est réveillé.

Code de renvoi : -1 : le sémaphore n'avait pas été initialisé ;

0 : succès.

Observation : les possibles codes de renvoi d'une fonction du noyau sont les mêmes qui ceux de son appel système correspondant.

Exemple : Trois processus P_1, P_2 et P_3 utilisent le sémaphore 0 afin de contrôler l'accès à une ressource ayant 2 exemplaires. Le processus ***Init***, qui initialise le sémaphore, est exécuté avant les 3 autres.

```
Init :
  main ( ) {
     init_sem (0,2) ;
  }
```

```
P1 :                    P2 :                    P3 :
main ( ) {              main () {               main () {
  ...                     ....                    ....
  P(0);                   P(0) ;                  P(0);
   Utiliser la ressource   Utiliser la ressource   Utiliser la ressource
  V(0);                   V(0);                   V(0);
  .....                   }                       }
 Lib_sem (0) ;
}
```

Observation : Si les deux exemplaires de la ressource sont réservés, le troisième processus sera bloqué tant qu'un exemplaire n'est pas libéré.

La valeur de *sem* est un entier entre ***0*** est ***NB_SEM -1.***

Pour implanter les sémaphores, nous avons créé la structure de données *sem* :

```
struct sem {
      int compt ; /* compteur du sémaphore */
      int File [MAX_FILE]; /* File d'attente gérée de façon circulaire */
      int first ; /* première case occupée dans File */
}
```

Nous avons aussi ajouté à la variable globale *v* du noyau le vecteur ***vect_sem*** de taille ***NB_SEM :***

```
struct var {
....
struct sem vect_sem [NB_SEM];
} v ;
```

A l'initialisation du Noyau, le champ *first* de toutes les entrées de *v.vect_sem* est initialisé à -1 pour indiquer qu'aucun sémaphore est réservé.

Le code des fonctions *sys_Init_sem (int sem, int N)*, *sys_Lib_sem (int sem)*, *sys_P(int sem)* et *sys_V(int sem)* est respectivement :

```
int sys_Init_sem (int sem, int N) {
 if ( (sem > NB_SEM-1) || (sem <0) )
      return -1;
 if (v.vect_sem[sem].first != -1)
      /*semaphore déjà en utilisation */
      return -1 ;
 v.vect_sem[sem].first=0;
 v.vect_sem[sem].compt=N;
 return 0;
}
```

```
int sys_Lib_sem (in sem) {
   v.vect_sem[sem].first=-1;
   return 0;
}
```