USTHB 2017/2018
Faculté d'Electronique et Informatique
Département Informatique

Homework TP Système Section M1 SII

Informations concernant ce travail

- **Exercices obligatoires a rendre pour l'évaluation finale du TP système:**

L'exercice 5 de la série des sémaphores.
L'exercice du parc d'attraction (ci dessous).

- **Exercice optionnels a rendre:** l'exercice 4 de la série des processus et tout autre exercice de la série des pipes que vous avez fait.

- Travail a faire en binôme (mais avec le binôme habituel). A rendre par email au plus tard pour le 28/12/2017 a l'adresse mail suivante: malika.mehdi@gmail.com

- **Ce qu'il faut rendre:** Une archive Nom1-Nom2-groupe-SII contenant les codes sources exercices demandes et un rapport contenant les réponses aux questions de l'exercice du parc d'attraction avec captures d'écran de l'exécution de ce dernier. **Avec objet** du mail: PROJET-SII.

La notation va prendre en considération la qualité du code en plus du fonctionnement des problèmes rendus (utilisation du fichier semaphores2.h, le traitement des erreurs dans les fonctions etc..). Les exercices optionnels ouvrent droit a un bonus de points supplémentaires. Une démonstration sera faite ultérieurement sur la base des codes remis.

Le problème du parc d'attraction:

Dans ce problème, N clients dans un parc d'attraction attendent leurs tours pour monter dans une voiture du train pour faire le tour du parc. On considère ici une seule voiture de train qui contient **P** places avec **P < N**. On va résoudre le problème de synchronisation entre un processus Voiture et N processus clients avec les contraintes suivantes:

- Le processus voiture doit appeler les fonctions: **charger**(), **décharger**() et **rouler**() pour annoncer le début du chargement, déchargement et pour commencer a rouler.
- Le processus voiture ne peut démarrer que si P clients sont a bord (voiture pleine) c'est a dire P clients auraient appelé la fonction **embarquer**().
- Les processus clients doivent appeler **embarquer**() et **debarquer**() avant de monter et avant de descendre de la voiture respectivement. Les clients font la queue avant de monter et avant de descendre.
- Les clients ne peuvent descendre qu'une fois la voiture aurait appelle la fonction **décharger**().
- Quand les P clients a bord quittent la voiture, la voiture va charger a nouveau un autre groupe de clients parmi les N de départ plus d'autres qui auraient éventuellement rejoint le parc plus tard.

On vous donne la solution théorique suivante que vous devez implémenter (en utilisant semaphores+ shared memory segments):

Les declarations:

Les variables partagées et les sémaphores:

int nbEmbarques = 0 //indique le nombre de processus clients qui sont montés dans la voiture.

int nbDebarques = 0 //indique le nombre de clients qui sont descendus de la voiture.

Les semaphores:

Semaphore mutex1 ← 1 //sémaphore mutex1 1initialise a 1. Protéger la variable nbEmbarques.

Semaphore mutex2 ← 1 // mutex2 initialise a 1. Protéger la variable nbDebarques.

Semaphore semEmbarquement ← 0 // sémaphore pour bloquer les clients avant d'embarquer.

Semaphore semDebarquement ← 0 // sémaphore pour bloquer les clients avant de descendre.

Semaphore semTousAbord ← 0 //sémaphore pour signaler au processus voiture que P clients sont maintenant a bord, il peut démarrer..

Semaphore semTousDehors ← 0 //sémaphore pour signaler au processus voiture que tout les clients sont maintenant descendus, la voiture peut faire un autre tour avec un autre groupe.

Fonctionnement du processus voiture (implémenté dans un fichier: voiture.c):

//mettre dans ce fichier la création et initialisation de tout les sémaphores nécessaires.

```
while (true){
chargement ();  //afficher un message, nouveau chargement tournée numéro i

pour I allant de 1 a p faire
        {
        V  (semEmbarquement )
        } //la il va débloquer p processus client bloques dans la file d'attente du sémaphore semEmbarquement
        P (semTousAbord)   //ici le processus  voiture se bloque et attend d'être libéré par le dernier client      qui monte, pour ensuite rouler ….
rouler()  //afficher un message et sleep(3).
Decharger() //afficher message, processus voiture prêt pour le déchargement
pour j allant de 1 a p faire
        {
        V(semDebarquement) //il va libérer tout les processus clients qui font la queue pour descendre (c'est a dire bloques dans l'opération P(semDebarquement)).
        }
P(semTousDehord) //le processus voiture se bloque ici et attend que le dernier client qui descent le reveil
}
```

Fonctionnement des processus clients (créer n processus fils qui vont jouer le rôle des clients dans un fichier clients.c) :

```
while(true){
P(semEmbarquement) //faire la chaine pour monter, attendre d'être libéré par le processus voiture
Embarquement () //affiche un message : "client  pid: je vais monter"
P (mutex1)
        nbEmbarques+=1
        si nbEmbarques==p faire:
        {
        V(semTousAbord) // le dernier client qui monte va réveiller le processus voiture pour qu'il démarre
        nbEmbarques = 0  //il remet le compteur a zero
        }
V (mutex1)  //libere la SC
enBalade() //affiche un message "client pid: je suis en balade " et sleep(1)
P( semDebarquement ) //se met en attente du debarquement  (il fait la chaine) pour descendre
Debarquement ()  //affiche un message "client pid: FIN de la balade je descend"
P(mutex2)
nbDebarques+=1;
si (nbDebarques==p) alors :
        {  V(semTousDehord) //réveiller le processus voiture, quand tout le monde est descendu pour qu'il
        charge a nouveau.
        NbDebarques=0;
        }
V(mutex2)
sleep(aleatoire 1 ou 2).
}
```

Les Questions:

Implémenter la solution précédente puis tester les cas suivants :

Fonctionnalités de base :

1- Fixer P dans les deux programmes et passer N au clavier. Lancer les deux programmes dans deux terminaux différents. En prenant le soin de toujours lancer le processus voiture d'abord (car c'est lui qui va créer tout ce qui est nécessaire pour le problème, les sémaphores par exp).

2- Qu'est ce qui se passera si on envoie un signal kill a un des processus client pendant que la voiture est en tournée (essayez en ligne de commande: kill -9 pid).

Est ce qu'on peut régler ce problème en utilisant le flag SEM_UNDO (avec le P). Expliquez le principe de ce flag dans ce cas précis.

3- On souhaite maintenant limiter le nombre de tournées pour chaque client a un nombre déterminé par le processus voiture (et qu'il peut changer lui même bien sur). Modifier le code des processus client pour prendre en considération cette limitation (la condition de la boucle while(true)) en respectant les contraintes suivantes :

- Le processus voiture lui doit rester toujours en vie en attente de nouveau clients qui arrivent (des processus clients qu'on lance a partir d'un autre terminal par exemple).
- Réaliser un teste de cette situation.

4- Que se passera t il si il n'y a pas p client pour monter dans la voiture? (donnez n<p). Expliquer la raison du problème remarqué.

- Pour régler ce problème utilisez la fonction **semtimedop** au lieu de **semop** (créez une nouvelle fonction appelée Ptimed par exp). Expliquer le principe de la fonction **semtimedop** et comment l'utiliser pour régler le problème précédant. (man 2 **semtimedop).**

Fonctionnalités avancées :

- **Implémentation de la question 4** : Dans ce cas on adopte le comportement suivant: après l'opération Ptimed(semEmbarquement), la voiture va vérifier si le nombre disponible est > C/3, si oui elle va quand même faire la tournée sinon la tournée est annulée (il faut donc afficher un message d'annulation et décharger directement). Il faut aussi penser a utiliser Ptimed a la place de P lors du déchargement.

5- Implémenter une barrière de synchronisation (ou RDV) réutilisable entre un groupe de N processus a ajouter a votre bibliothèque. Le principe : tout le monde dors en appelant P sur un sémaphore et le dernier arrivé appellera V en boucle (N-1) pour réveiller tout les processus bloqués. Inspirez vous du code fournie.

Bon courage.