UNIVERSITÉ HASSAN II DE CASABLANCA
FACULTÉ DES SCIENCES AÏN CHOCK

Système d'exploitation II
TD 5 : Sémaphore.
SOLUTION

Exercice 1 :

Le problème de rédacteur et lecteurs consiste à un écrivain qui écrit et met le résulta dans un mémoire partagé. Cette mémoire est par la suite lue via plusieurs lecteurs. Ce qui revient à un problème de synchronisation, vu que la lecture doit être bloquante, de même pour l'écriture.

> **Question 1** : écrire un pseudo code en utilisant les sémaphores qui permettra la synchronisation entre les processus, dans un premier cas on suppose la présence d'un seul rédacteur et un seul lecteur.

Pour répondre à cette question nous devons répondre aux questions suivantes (i) La section critique, (ii) le nombre de processus qui peut accéder à la section critique, (iii) le nombre de fonctions à utiliser :

(i) Pour la section critique c'est l'accès au fichier, ainsi nous aurons besoin d'un seul sémaphore.

(ii) Le nombre de processus un seul ainsi le sémaphore sera initié à un 1.

(iii) Le nombre de fonction à utiliser c'est trois (lecteur, écrivain et init).

Il faut noter que c'est pseudocode c'est-à-dire un algorithme.

```
semaphore mutex ;

void lecteur()

{

char * buf ;

 printf(« je veux accéder au fichier pour lire ») ;

down(&mutex) ;  //réserver l'accès à la ressource

lire_fichier(buf) ;

up(&mutex) ; // libérer la ressource.

afficher(buf) ;

}

void ecrivain (char * data)
```

```
{
printf(«je veux écrire dans le fichier » ) ;
down(&mutex) ;  //réserver l'accès à la ressource
ecrire_data(data) ;
up(&mutex) ; // libérer la ressource.
}
Void init()
{mutex=1 ;}
```

Question 2 : généraliser la solution pour plusieurs lecteurs.

Dans ce cas nous avons plusieurs lecteurs, c'est-à-dire plusieurs lecteurs peuvent lire en même temps mais un lecteur et un écrivant ne peuvent pas accéder au document simultanément. Donc, nous devons définir le processus lecteur qui bloquera l'écriture et celui qui libérera la lecture, c'est-à-dire le premier lecteur qui accédé au fichier doit bloquer l'écrivain et le dernier lecteur doit libérer l'accès. Pour se faire nous devons utiliser un compteur. Ce compteur sera partagé entre tos les lecteurs, ainsi c'est une section critique donc il faut utiliser un deuxième sémaphore.

```
semaphore mutex ;
semaphore mutex1 ; //pour l'accès au compteur
int cpt ;
void lecteur()
{
char * buf ;
 printf(« je veux accéder au fichier pour lire ») ;
 down(&mutex1) ;
cpt++ ;

if(cpt==1) // je suis le premier lecteur je doit bloquer l'écrivain
{    down(&mutex) ;  //réserver l'accès à la ressource
```

```
    lire_fichier(buf) ;
     cpt-- ;
      if(cpt==0) // je suis le dernier lecteur dc je peux libérer l'accès au fichier.
           { up(&mutex) ;}
     afficher(buf) ;
}
else
// je ne suis pas le premier lecteur donc je n'ai pas à réserver l'accès au fichier, car le premier a déjà réservé.
{ lire_fichier(buf) ;
up(&mutex) ; // libérer la ressource.
cpt-- ;
       if(cpt==0) // je suis pas le premier mais ça se peut que je suis le dernier lecteur dc je peux libérer l'accès au fichier.
            {up(&mutex) ;}
afficher(buf) ;
}
up(&mutex1) ;
}
void ecrivain (char * data)
{
printf(«je veux écrire dans le fichier » ) ;
down(&mutex) ;  //réserver l'accès à la ressource
ecrire_data(data) ;
up(&mutex) ; // libérer la ressource.
}
void init()
```

```
{mutex=1 ;
mutex2=1 ;
cpt=0 ;}
```

Exercice 2 :

Dans une commune rurale, les gens en mis un dispositif pour pouvoir traverser la route, un petit pond qui permet de faire passer une seule personne à la fois, c'est-à-dire deux personnes ne peuvent pas traverser vers deux directions différentes.

> **Question 1** Au moyen de sémaphores, écrivez un programme pour éviter l'inter blocage. Ne traitez pas le cas d'un groupe infini personnes se déplaçant d'un côté et interdisant tout Passage à ceux qui se déplacent vers l'autre côté.

La section critique est l'accès au pond. Une seule personne peut accéder dans un sens donc nous aurons besoin d'un seul sémaphore.

```
Semaphor mutex ;
void traver_gauche()
{ printf(« je veux traverser vers la gauche » ) ;
  down(&mutex) ;
    traverser() ;
 up(&mutex) ;
}
void traver_droite()
{ printf(« je veux traverser vers la droite » ) ;
  down(&mutex) ;
    traverser() ;
 up(&mutex) ;
}
void init()
{mutex=1;}
```

Question 2 Reprenez la question précédente, mais en évitant la privation de ressources (famine). Lorsqu'une personne qui souhaite traverser le pont vers l'est arrive à la corde et trouve une personne qui traverse vers l'ouest, il attend jusqu'à ce que la corde soit vide, mais Aucune personne se déplaçant vers l'ouest n'est autorisé à démarrer jusqu'à ce qu'au moins une personne ait traversé dans l'autre sens.

Dans cette question plusieurs personnes peuvent accéder dans un sens c'est le même exemple de la question précédente, donc il faut ajouter deux ompteur , chaque compteur dans un sens, et pour chaque compteur il faut ajouter un sémaphore donc nous arons besoins de :

- **Deux compteurs cptd et cptg**
- **Trois sémaphores mutex1, mutexg et mutexp ;**

```
Semaphore mutex ; //pour le pond
Semaphore mutxg ; // pour le compteur gauche
Semaphore mutexd ; // pour le commpteur droite

void traverser_g()
{
 printf(« je veux traverser vers la gauche ») ;
 // je dois vérifier est ce que je suis le premier
down(&mutexg);
cptg++ ;
if(cptg==1)
 {  down(&mutex);
    traverser();
   cptg--;
   if(cptg ==0) { up(&mutex);}
}
else // je suis pas le premier
{     traverser();
   cptg--;
   if(cptg ==0) { up(&mutex);}
}
up(&mutexg) ;
}

void traverser_d()
{
 printf(« je veux traverser vers la droite») ;
 // je dois vérifier est ce que je suis le premier
down(&mutexd);
cptd++ ;
if(cptd==1)
 {  down(&mutex);
    traverser();
   cptd--;
```

```
   if(cptd ==0) { up(&mutex);}
}
else // je suis pas le premier
{     traverser();
   cptd--;
   if(cptd ==0) { up(&mutex);}
}
up(&mutexd) ;
}

void init()
{
mutex=1;
mutexd=1;
mutexg=1;
cptd=0;
cptg=0;
}
```

Exercice : 3

Dans cet exercice nous voulons proposer une solution pour l'accès à un bus via une seule porte. C'est-à-dire qu'un bus comporte une seule porte et qu'on veut synchroniser entre les personnes qui veulent monter et ceux qui veulent descendre. Bien sur une seule personne peut passer à la fois dans un sens différent.

L'objectif consiste à éviter l'état de famine c'est-à-dire une personne attend à l'infini pour descendre ou monter, ainsi que l'état d'inter blocage c'est-à-dire deux personnes veulent accéder dans un sens différent en même temps.

Question 1 : C'est quoi la section critique dans ce cas d'utilisation ?

La ressource partagée est l'accès la porte ainsi la section critique est l'accès à la porte.

Question 2 : Dans un premier temps on suppose qu'une seule personne peut descendre ou monter à la fois. En utilisant les sémaphores :

a) Ecrire le pseudo code de la fonction descendre ().

```
semaphore mutex ;
void descendre()
{ printf(« je veux descendre » ) ;
  down(&mutex) ;
  utiliser_porte() ;
  up(&mutex) ;
}
```

b) Ecrire le pseudo code de la fonction monter ().

```
void monter()
{ printf(« je veux monter » ) ;
  down(&mutex) ;
  utiliser_porte() ;
  up(&mutex) ;
}
```

c) Ecrire un programme C qui permet la création de deux processus fils. Le premier essaye de descendre et le deuxième essaye de monter. Utiliser les deux fonctions des questions b) et c)

```
#include<sys/signal.h>
int main()
{ int f1=fork();
  Int f2=fork();
 if(f1==0) {monter();}
if(f2==0) {descender();}
 return 0;
}
```

Questions 3 : Dans cette question on suppose que plusieurs personnes peuvent descendre ou monter en même temps à condition que ça soit dans le même sens. En utilisant les sémaphores :

a) Ecrire le pseudo code de la fonction descendre1 ().

```
semaphore mutexd ;
int cptd ;
void descendre1()
{ printf(« je veux descendre » ) ;
// est ce que je suis le premier qui descends
down(&mutexd) ;
cptd++ ;
if(cptd==1)
{    down(&mutex);
     utiliser_porte();
```

```
    cptd-- ;
   if(cptd==0) {up(&mutex) ;}
}
else // je suis as le premier
{    utiliser_porte();
    cptd-- ;
   if(cptd==0) {up(&mutex) ;}
}
 up(&mutexd) ;
}
```

b) Ecrire le pseudo code de la fonction monter2().

```
semaphore mutexm ;
int cptm ;
void monter1()
{ printf(« je veux monter » ) ;
// est ce que je suis le premier qui descends
down(&mutexm) ;
cptm++ ;
if(cptm==1)
{    down(&mutex);
     utiliser_porte();
    cptm-- ;
   if(cptm==0) {up(&mutex) ;}
}
else // je suis as le premier
{    utiliser_porte();
    cptm-- ;
   if(cptm==0) {up(&mutex) ;}
}
 up(&mutexm) ;
}
```

c) Ecrire un programme C qui permet la création de 4 processus fils. Les deux premiers essayent de descendre et les deux derniers essayent de monter. Utiliser les deux fonctions des questions b) et c)

```
#include<sys/signal.h>
int main()
{ int f1=fork();
  int f2=fork();
```

```
  int f3=fork();
  int f4=fork();
 if(f1==0) {monter();}
if(f2==0) {descender();}
if(f3==0) {monter();}
if(f4==0) {descender();}
 return 0;

}
```