TD IPC : Correction

Ex1 : DS 11/2010*

une bonne sol pour la SC doit vérifier
⇒ un seul process en SC
⇒ avancement et absence de blocage : un processus hors sa section critique ne doit pas bloquer un autre processus d'entrer en SC.
⇒ attente bornée : pas de famine

1) solution incorrecte : avancement non vérifié
exemple : P2 en SC, P1 bloqué sur P(mutex2) ⇒ P1 bloque P3 alors qu'il n'est pas dans sa SC.

2) on garde le même code pour P2 et P3
pour P1 :

```
p(mutex1) ;
n=n-1 ;
v(mutex1) ;
P(mutex2) ;
out=out+1 ;
v(mutex2) ;
```

Ex2 : partage d'imprimantes*

```
int LP[3]={0,0,0} ;// si LP[i]=0 alors l'imprimante n°i est libre
semaphore impr=3 ;
semaphore mutex=1 ; // protéger LP des accès multiples

int prendre ()
{
        int i =0;
        p(impr) ;// si toutes les imprimantes sont utilisées, se bloquer
                while (i<3)
        {
                p(mutex) ; // protéger LP des accès multiples
                if (LP[i]==0) {LP[i]=getpid() ;v(mutex) ; return i;}
                v(mutex) ;
                ++i ;
        }
        return -1 ;

}

int liberer(int i)
{
        p(mutex) ;
        LP[i]=0 ;
        v(mutex) ;
        v(impr) ;
}
```

Ex3 : prod/cons (diffusion atomique)**

```
#define M 100 // taille du tampon d'objets
#define n 50 // nombre de consommateurs
objet tampon[M] ;
// on suppose que pour chaque objet, il y a une variable nbcons=n (nombre de consommateurs)
semaphore mutex=1 ; // protéger l'accès au tampon ;
semaphore plein[n] ={0} ; // 1 semaphore par consommateur
semaphore vide=M ; // indique le nombre de cases vides

Producteur()
{
        objet obj ; int k, j=0 ;
        while(1)
        {
                obj=produire_objet() ;
                p(vide) ;// s'il n'y a pas de case vides, se bloquer
                p(mutex) ;// protéger l'accès au tampon
                tampon[j]=obj ; // remplir les cases dans l'ordre, recommencer si on arrive à la fin
                p(mutex) ;
                j=(j+1)mod M ;
                for (k=0;k<n;k++)
                        v(plein[k]) ;// avertir tout les consommateurs
        }
}

consommateurCk (int k)
{
        int i=0;// pour sauvegarder l'indice de l'objet à consommer la prochaine fois
        //la consommation se fait dans l'ordre, recommencer si on arrive à la fin
        while (1) ;
        {
                p(plein[k]); //s'il n'y a pas de cases à consommer concernant k, se bloquer
                p(mutex) ;
                tampon[i].nbcons-- ;
                if ( tampon[i].nbcons==0) v(vide) ; // si tout les consommateurs ont consommé
                                                    //l'objet, déclarer l'existence d'une case vide
                v(mutex) ;
                i=(i+1) mod M ;// on calcule le prochain indice de l'objet à consommer
                // Ainsi, un consommateur ne peut pas cosommer le même objet plusieurs fois, même
s'il recommence à partir de la première case (alors que certains ne l'ont pas encore consommé) car il
sera bloqué au niveau de son sémaphore plein[k].
        }
}
```

EX4 (DS 11/2007):*

variables partagées : semaphore S1=1, S2=1, S3=1, S4=0 ; //S4 = 0 pour bloquer P3

P1 ()	P2()	P3()

While (true)
P(S1) // parallélisme ----------
	While (true)
	P(S2)
		P(S4) ; //blocage
		A4

P(S3) // synchronisation
A1
V(S2) // cooperation...
V(S3)
	P(S3)
	A2
	V(S1)
	V(S3)	

⇒ parallélisme : les deux processus peuvent commencer en parallèle
⇒ synchronisation : l'un commence et l'autre se bloque puis il sera réveillé
⇒ cooperation : losqu'un processus exécute son action Ai, il donne la main à l'autre

Démarche : Sans boucle, on doit exécuter A1 suivie de A2 ou A2 suivie de A1 donc on aura : S3=1 et le code suivant :

P1 ()	P2()	P3()

P(S3) // synchronisation
A1
V(S3)
	P(S3)
	A2
	V(S3)
		P(S4) ; //blocage
		A4

Mais lorsqu'on va boucler, un processus qui s'exécute une fois (semaphore initialisé à 1) doit attendre que l'autre le débloque pour recommencer la compétition sur S3. Il se bloque donc sur son propre sémaphore

EX5 (DS 11/2007):*

1) deux processus peuvent obtenir la même machine . Exemple : commutation avant « dispo[i]=0 ; »
2) solution : ajouter un verrou (semaphore) avant « for » ou dans la boucle « for » et dans la fonction liberer

```
#define NMACHINES 5
Semaphore nlibre = 5 ;
int dispo[NMACHINES] =(1,1,1,1,1) ;
Semaphore mutex=1 ;
int Allouer()
{       Int i ;
        P(nlibre) ;
        for (i=0; i < NMACHINES; i++)
        {       p(mutex) ;
                if (dispo[i] != 0) {
                        dispo[i] = 0;
                        v(mutex) ;
                        return i;
                }
                else v(mutex) ;
        }
}
---------------------------------------
liberer(int machine) {
P(mutex) ;
dispo[machine] = 1 ;
v(mutex) ;
V(nlibre) ; }
```

EX6 :**
1) prob des lecteurs/rédacteurs avec priorité des lecteurs. Rédacteur= 1 seul camion ; lecteurs : 3 voitures

2) semaphore voit=3 ;
semaphore cam=1 ;
semaphore mutexvoit=1 ;
semaphore mutexprio=1 ;// pour la priorité des voitures par rapport aux camions
int nbvoit=0 ; // nécessaire pour pouvoir réveiller un camion si le nombre de vitures devient 0

```
processus voiture()
{
        p(voit) ;// s'il y a déjà 3 voitures, se bloquer
        p(mutexvoit) ;// protège la varible nbvoitures
        nbvoit++ ;
        if(nbvoit==1) // la première voiture arrive : elle peut trouver un camion ou non
        {
                v(mutexvoit) ;
                p(cam) ;// bloquer les camions (s'il n'y a pas encore de camions) ou se bloquer
        }
        else v(mutexvoit) ;
        Traverser_pont() ;
        p(mutexvoit) ;
        nbvoit-- ;
        if (nbvoit==0)  v(cam) ;
        v(mutexvoit) ;
        v(voit) ;
}

processus camion()
{
        p(mutexprio) ; // priorité des voitures : les camions traversent deux barrières (2 semaphores)
                        // de telle façon à avoir un seul camiom bloqué au niveau de p(cam).
        p(cam) ;
        Traverser_pont() ;
        v(cam) ;
        v(mutexprio) ;
}
```

Ex7(DS 11/2005) :*

1) semaphore mutex=1 ;
        processus pi() // i=1,2,3
        debut
        cycle
        p(mutex)
        Ai
        v(mutex)
        fincycle
2) semapore mutex1=1, mutex2=0, mutex3=0 ;

P1	P2	P3
debut cycle p(mutex1) Ai v(mutex2) fincycle	debut cycle p(mutex2) Ai v(mutex3) fincycle	debut cycle p(mutex3) Ai v(mutex1) fincycle

3) semaphore mutex1=1, mutex23=1 ;

P1	P2	P3
debut cycle p(mutex1) Ai v(mutex23) fincycle	debut cycle p(mutex23) Ai v(mutex1) fincycle	debut cycle p(mutex23) Ai v(mutex1) fincycle

Partie II : Moniteurs

Ex1(exam 06 /2010)

1) l'erreur est dans if qui doit avoir une condition de blocage et non pas de passage car nous devons garantir le même comportement pour un processus qui passe directement et pour un processus qui se bloque puis continue lorsqu'il sera réveillé

2) exemple : P1 occupe la SC donc token=false
P2 arrive et se bloque (wait)
P1 quitte la SC et remet token à true et réveille P2 qui passe dans la SC
P3 arrive et trouve token à true et passe dans la SC

⇒ P3 et P2 se trouvent tous les deux dans la section critique ⇒ exclusion mutuelle non satisfaite

2) solution :

```
Public void EntreeSC()
{
        while (!token)
        wait(sc) ;
        token=False
}
```

Ex II.2 (exam 2006)

1) Puisqu'on ne peut jamais avoir plus qu'un processus actif dans le moniteur, cette solution garantit l'exclusion mutuelle.

2) Mais, le moniteur sera monopolisé durant toute la durée d'utilisation de la ressource.

3) une solution comprend un prologue et un epilogue :

```
Monitor SC {
        int occupe ; // 0 : ressource libre sinon occupée
        cond prise ;

        public void entreeSC{
        if (occupe) wait (prise) ;
        occupe=1 ;
        }

        public void SortieSC{
        occupe=0 ;
        signal (prise) ;
        }
        { occupe=0 ;//initialisation du moniteur}
}

Processus Pi
{       ….
        SC.EntreeSC()
        // utiliser ressource
        SC.SortieSC()
        ….
}
```

Ex II.3 (DS 11/2006)

1) Si un consommateur C1 arrive en premier lieu, il sera bloqué sur la condition vide et il libère le moniteur. Puis, un producteur produit un objet et réveille C1 (qui n'occupera pas le moniteur pour le moment) mais entre temps un consommateur C2 arrive et passe (il retire l'objet et libère le moniteur) ⇒ C1 peut être ramené à retirer un objet inexistant ⇒ C1 doit retester si le tampon est vide ou pas ⇒ **remplacer if par while**

2)
```
Public void retirer (objet x){
        If (is_prioritary()) {
                Nbprio++ ;
                While (tampon_vide)
                wait(vide) :
                Nb_prio--;
        }
        Else
        {
                while (tampon_vide || nbprio>0)
                {
                        Wait(vide);
                        If ( nbprio>0) // si on me réveille(il y a des objets déposés), s'il y a des
                                       // consommateur prioritaire, je réveille l'un d'entre eux
                        signal(vide) ;
                }
        }
        Retirer_objet(x) ;
        Signal(plein) ;
 }
```

3)

<table>
<tr>
<td rowspan="2">Monitor PC_TI {
int nbobj=0 ; // nombre d’objets déposé
cond pasvide ;

public void déposer(objet x){
deposer_objet(x) ;
nbobj++ ;
signal(pasvide) ;
}

public void retirer(objet x){
while(nbobj==0) wait (pasvide)
retirer_objet(x) ;
nbobj-- ;
}
}</td>
<td>Producteur(){
….
produire_objet (x) ;
PC_TI.deposer(x) ;
...
}</td>
</tr>
<tr>
<td>Consommateur() {
….
PC_TI.retirer(x) ;
consommer_objet(x) ;
….
}</td>
</tr>
</table>

Ex II.4 (DS 11/2010)

Trois clubs A, B et C. Dans le stade il doit y avoir au Max 2 clubs et N athlètes par club.
Soit nbA, nbB et nbC le nombre d’athlètes des clubs A, B et C ayant accédé au stade (en vérifiant la condition précédente)
Un athlète d’un club X (X=A ou Bou C) sera bloqué lorsqu’il y a déjà N athlètes de son club ou des athlètes des deux autres clubs

```
Monotor Stade {
        int nbA=0, nbB=0, nbC=0 ;
        cond Astop, Bstop, Cstop ;

        public void entrerA(){
                while(nbA==N || nbB*nbC!=0)
                        wait(Astop) ;
                nbA++ ;
        }

        public void sortieA(){
                nbA-- ;
                signal(Astop)
                if (nbA==0) { signal (Bstop) ; signal(Cstop);}

        }
        // de même pour B et C

}
```