TD Programmation Système

Série 5 : Correction

Exercice 1 : *Exécutions atomiques*

a-L'exécution des processus en parallèle **suivant** la politique d'ordonnancement de type *round robin* partageant le temps d'exécution permet l'exécution d'une instruction de chaque *processus au pire des cas ce qui permet l'affichage de différentes valeurs de x et y :(0,0), (1,1), (2,2), (1,2), (2,0), ...*

b- Assurer que les `printf` des deux processus affichent toujours des valeurs identiques pour x et y (x=y) revient à exécuter les deux premières instructions de chaque processus sans interruption. Par conséquent, les deux premières instructions de chaque processus constituent la section critique de chaque processus. Les deux processus seront en exclusion mutuelle. Un sémaphore *mutex, avec val(mutex)=1* permet de garantir l'exclusion mutuelle

Processus P1

```
P(mutex)
x = x + 1 ;
y = y + 1 ;
V(mutex)
printf("x=%d,y=%d\n", x, y) ;
```

Processus P2

```
P(mutex)
x = x * 2 ;
y = y * 2 ;
V(mutex)
printf("x=%d,y=%d\n", x, y) ;
***
```

Exercice 2 : Synchronisation (barrière)

Question 1 :

Le sémaphore de l'exclusion mutuelle *Sema1* est toujours initialisé à 1. Le sémaphore `Sema2` pour la Synchronisation de la barrière est initialisé à 0 permettant ainsi le blocage du premier processus exécutant la fonction *Wait*. Le compteur permettant de compter le nombre de processus bloqués est initialisé à 0 au départ. Le nombre maximum de processus bloqués est initialisés à N

```
void InitBarrière (Barrière *B; int Maximum)
{
      Init (B→Sema1, 1);
      Init (B→Sema2, 0);
      B→Count    =  0  ;
      B→Maximum  =  N;
}
```

Question 2 :

Complétez le code de la procédure d'attente donné ci-dessous :

```
void Wait (Barrière *B)
{
      Boolean Do_Wait = True ;
      int I ;

      P(B→Sema1) ;
      B→Count++ ;
      if (B→Count == B→Maximum ){
            for (I=0 ; I < (B→Maximum-1) ; I++ ) V(B→Sema2); ;
            B→Count = 0;
            Do_Wait = False ;
      }
      V(B→Sema1);
      if (Do_Wait) P(B→Sema2);
}
```

Exercice 3 : Ordonnancement

Deux sémaphores sont nécessaire pour gérer les synchronisations entre P1, P2 et P3 :

- S1 pour gérer les synchronisations entre P1 et P2
- S2 pour gérer les synchronisations entre (P1, P2) et P3

Val(S1)= Val(S2)= 0

Processus P_1	*Processus P_2*	*Processus P_3*
Début Tantque *vrai* faire .... **V(S1)** Fintantque Fin	Début Tantque *vrai* faire ..... **P(S1)** *// partie 1* .... *// partie_2* ..... **V(S2)** Fintantque Fin	Début Tantque *vrai* faire .... *// partie_1* ..... **P(S2)** *// partie_2* .... Fintantque Fin