TD Programmation Système

Série 5 : Accès concurrent

Exercice 1 : *Exécutions atomiques*

On suppose que sur Unix on peut définir des variables `x` et `y` partagées par deux processus (x=0, y=0). Les deux processus exécutent les codes suivants :

Processus P1

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

Processus P2

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

a- Ces processus s'exécutent sur un système UNIX dont la politique d'ordonnancement est du type *round robin*. Quelles peuvent être les couples de valeurs affichées par chacun des deux processus ?

b- En utilisant un sémaphore, modifier le code pour assurer que les *`printf`* affichent toujours des valeurs identiques pour `x` et `y`.

Exercice 2 : Synchronisation (barrière)

Ecrire les procédures d'accès à un objet de type barrière dont le fonctionnement est le suivant :

1. la barrière est fermée à son initialisation,
2. elle s'ouvre lorsque N processus sont bloqués sur elle.

Définition de la barrière :

```
struct{
      Sema  Sema1 ;          /* Sémaphore de section critique    */
      Sema  Sema2 ;          /* Sémaphore de blocage sur la barrière  */
      int   Count ;          /* Compteur             */
      int   Maximum ;        /* Valeur déclenchant l'ouverture   */
} Barrière ;
```

Question 1 :

Complétez le code de la procédure d'initialisation :

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

Question 2 :
Complétez le code de la procédure d'attente donné ci-dessous :

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

      ... ;
      B→Count++ ;
      if (B→Count == B→Maximum ){
            for (I=0 ; I < (B→Maximum-1) ; I++ ) ... ;
            B→Count = 0;
            Do_Wait = ... ;
      }
      ... ;
      if (Do_Wait) ... ;
}
```

Exercice 3 : Ordonnancement

Programmer un rendez-vous entre 3 processus cycliques P_1, P_2 et P_3 en utilisant les opérations sur les ***sémaphores***. P_3 ne peut exécuter sa $2^{ème}$ partie que si P_1 et P_2 se sont complètement exécutés. P_2 ne peut exécuter sa $1^{ère}$ partie que lorsque P_1 s'est entièrement exécuté.

Processus P_1	*Processus P_2*	*Processus P_3*
Début Tantque *vrai* faire .... Fintantque Fin	Début Tantque *vrai* faire .... *// partie 1* .... *// partie_2* .... Fintantque Fin	Début Tantque *vrai* faire .... *// partie 1* .... *// partie_2* .... Fintantque Fin