Partie 5: Interblocage

Solution 1

Question 1 :

a) Si T ≤ M, il ne peut pas y avoir d'interblocage car il y a suffisamment d'espace mémoire pour satisfaire toutes les demandes d'allocation des processus.

Par contre, si T>M, on peut atteindre une situation d'interblocage : il n'a plus d'espace mémoire disponible et tous les processus ont encore besoin d'allouer de l'espace mémoire.

b) E= 8 A = 8 – (1+1+1+3) = 2.

C = (1, 1, 1,3) *R = (2 3 3 2)*

L'état est sûr car l'ordonnancement suivant permet de terminer tous les processus :
P1 ; A = 3 ; P4 ; A = 6 ; P2 ; A=7 ; P3 ; A = 8.

c) A = 1 ; C = (1, 1, 2, 3) R = (2 3 2 2)

L'état n'est pas sûr car on ne peut satisfaire aucune requête. La demande est refusée.

Question 2 :

1) $\sum_{i=1,N} Ci = M$ et pour tout i=1 à N $Ri \geq 1$
2) Supposons que $T < N+M$ et il y a interblocage à l'état courant c'est à dire : $\sum_{i=1,N} Ci = M$ et pour tout i=1 à N on a $Ri \geq 1$

   On a donc $\sum_{i=1,N} Ri \geq N$ et $T = \sum_{i=1,N} (Ci + Ri) < N + M = N + \sum_{i=1,N} Ci$

   Par conséquent, on a $\sum_{i=1,N} Ri < N \leq \sum_{i=1,N} Ri$. D'où contradiction.

Solution 2

1) Oui, un ensemble de producteurs et le consommateur peuvent être interbloqués :

Chaque producteur Pi de l'ensemble est en attente suite à l'appel à P(vide) et il n'y a plus de cases pleines (vide=0 et plein=0).

semaphore plein =0, vide=N, mutex=1, mutex1=1 ;
char T[N] ;

int ip=0 ;

```
Producteur ( )
{       int i, M ;
        char ch[N];
        Repeter
        {       M=Lire(ch,N);
                M= min(N,M);
                P(mutex1)
                Pour i=1 à M pas 1 faire
                    P(vide) ;
               V(mutex1) ;
                P(mutex) ;
                Deposer(ch,M,ip) ; //insérer ch dans T
                ip = (ip+M)%N ;
                V(mutex) ;

                Pour i=1 à M pas 1  faire
                    V(plein) ;
        } tant que vrai ;
}
```

```
Consommateur ( )
{       int ic =0 ;
        char c ;
        Repeter
        {
                P(plein) ;
                P(mutex) ;
                c= T[ic] ;
                V(mutex) ;
                V(vide) ;
                Traiter(c) ;
                ic++ ;

         } tant que vrai ;
}
```

Solution 3

1)
État après une itération

E = [1 1 1 1] A = [**0 1 1 0**]

C = 1 0 0 0 R = 0 1 0 0

0 1 0 0 **0 0 1 0**

0 0 0 1 1 0 0 0

État après deux itérations
E = [1 1 1 1] A = [**1 1 1 0**]

C = 1 0 0 0 R = **0 1 0 0**
0 1 0 0 **0 0 1 0**
0 0 0 1 1 0 0 0

État après trois itérations
E = [1 1 1 1] A = [**1 1 1 1**]
C = 1 0 0 0 R = **0 1 0 0**
0 1 0 0 **0 0 1 0**
0 0 0 1 **1 0 0 0**

Tous les processus sont marqués… L'état est sur.

2- En numérotant les ressources et en imposant de faire les demandes par ordre croissant de leurs numéros.

Ou en allouant à la fois toutes ressources nécessaires (politique de tout ou rien).

Solution 4

1 détient D, E attend C

2 détient C attend B

3 détient A, B attend E

Oui, en utilisant l'algorithme du banquier

Solution 5

1. Oui. Par exemple, supposons deux processus P1 et P2.

P1 veut effectuer un transfert de c1 vers c2. Il a pu verrouiller c1 mais est en attente de c2. P2 veut effectuer un transfert de c2 vers c1. Il a pu verrouiller c2 mais est en attente de c1.

2. Il suffit de faire en sorte qu'il n'y est pas d'attente circulaire. Chaque processus doit verrouiller le compte le plus petit avant le compte le plus grand.

3. Oui, on connaît les besoins de chaque processus. On peut donc utiliser l'algorithme du banquier.