



Classe LMD S5 Informatique
Année Universitaire : 2015 / 2016
Mr Benaissa Mohamed

Solution Examen N°1: Module système d'exploitation

Questions de cours (6 pts)

Q1 : Quelles critiques peut-on faire aux solutions logicielles et matérielles du problème de l'exclusion mutuelle ?

Pour la solution logicielle : perdre et gaspillage de temps de cpu

Pour la solution matérielle n'est pas applicable dans un système multiprocesseur

Q2 : Quelles sont les contraintes nécessaires pour que l'exclusion mutuelle soit satisfaite dans un système d'exploitation ?

If faut respecter les 5 contraintes suivantes :

Exclusion mutuelle

Absence de blocage

Absence de l'interblocage

Absence de famine

Le symétrique

Q3 : Comment en peut garantir l'état stable (sain) d'un système d'exploitation ?

En appliquant l'algorithme de banquier pour éviter une situation d'inteblocage

Q4 : donner deux propriétés qui différencient un thread par rapport à un processus.

-Il y a une séparation entre les processus dans l'espace d'adressage (protection).

Par contre, Tous les threads d'un même processus partagent le même espace d'adressage de leur processus principale.

-Les threads ne sont pas reconnus par le système d'exploitation, leur communication s'effectue directement sans faire l'appel au système.

Par contre, le système gère les processus. La communication entre les processus s'effectue par des appels système.

Q5 : Vous avez un programme composé par 4 threads suivants : **T1, T2, T3 et T4**

T1 : affiche le mot «née »

T2 : affiche le mot «ne an »

T3 : affiche le mot «2016 »

T4 : affiche le mot «Bon»

1-Synchronisez les 4 threads : T1, T2, T3 et T4 en utilisant les primitives de sémaphores P et V pour que le programme affiche obligatoirement la phrase suivante : « **Bonne année 2016** »

2- Donner les valeurs initiales des sémaphores utilisés.

synchronisation par les sémaphores :

P(s2) T1 V(s3)

P(s1) T2 V(s2)

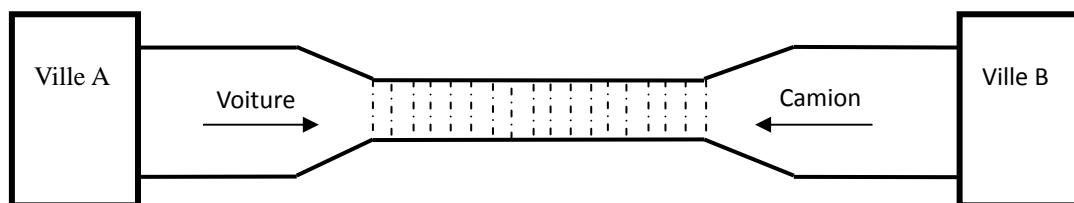
P(s3) T3

T4 V(s1)

Q2 : Initialisation les valeurs des sémaphores : $e(s1) = e(s2) = e(s3) = 0$

Exercice N°1 (5 pts)

Une route reliant deux villes A et B contient un pont. Nous supposons que les voitures vont de A vers B et les camions de B vers A. Le pont est suffisamment long pour permettre à N ($N \geq 1$) voitures de passer en même temps (l'une après l'autre), mais la largeur du pont ne permet pas aux voitures et aux camions de circuler en même temps. De plus, à cause du poids, un seul camion (à la fois) est admis sur le pont. Nous voulons synchroniser les processus « voiture » et « camion ».



Q1 : Parmi les problèmes classiques de synchronisation vus au cours, lequel correspond à ce problème (expliquer les ressemblances).

Q2 : Proposer une synchronisation des processus « voiture » et « camion » en utilisant des moniteurs (nous supposons que la priorité est aux voitures).

Solution Exercice N°1 (5 pts)

1) Ce problème correspond au problème des lecteurs rédacteurs. Les ressemblances sont :

- Les voitures = Les lecteurs
- Les camions = Les rédacteurs
- Un seul camion sur le pont = Un seul rédacteur dans la zone de donnée
- Plusieurs Voitures sur le pont en même temps (max N) = plusieurs lecteurs en même temps dans la zone de données.

2) Proposition d'un moniteur

Nous utiliserons le moniteur qui sert à résoudre le problème des lecteurs/rédacteurs avec priorité aux lecteurs et le nombre de lecteurs limité à N ($N \geq 1$).

```

#define N 10                // nb max de lecteurs en accès partagés

monitorPont
{
    int camion ;    // camion=1    camion sur le pont
    int nv1 ;       // nombre de voitures sur le pont
    int nv2 ;       // nombre de voitures qui ont demandé de passer sur le pont (nv2>=nv1)
    condition voitureOk, camionOK ;

    void voiture_demande_passage ( )
    {
        nv2++ ;
        if ((camion==1) || (nv2>N))
            { wait (voitureOk) ;
              nv1++ ;
              if (nv1<N) signal (voitureOk);
            }
        else nv1++ ;
    }
    Void voiture_fin_passage ( )
    {
        nv1--; nv2--;
        if (nv2>nv1) signal (voitureOk);
        if (nv2==0) signal (camionOk);
    }

    Void camion_demande_passage ( )
    {
        if ((camion==1) || (nv2>0)) wait (camionOk);
        camion = 1;
    }
    voidcamion_fin_passage ( )
    {
        camion = 0;
        if (nv2>0)    signal (voitureOk);
        else          signal (camionOk);
    }
    {
        // bloc d'initialisation
        nv1 = nv2 = camion = 0;
    }
} // fin du moniteur

```

Exercice N°2 (4 pts)

Dans un système il y a 4 processus (P_i , $i=0..4$) qui partagent 6 types de ressources (R_i , $i=0..6$). Le système possède une seule ressource de chaque type.

L'état courant du système est décrit par les matrices d'allocation (ALLOCATION) et de besoin suivantes, ainsi que le vecteur de disponibilité instantanée

Disponible = [0 1 0 0 0 0]

| | R1 | R2 | R3 | R4 | R5 | R6 |
|----|----|----|----|----|----|----|
| P1 | 1 | 0 | 1 | 0 | 0 | 0 |
| P2 | 0 | 0 | 0 | 1 | 0 | 0 |
| P3 | 0 | 0 | 0 | 0 | 1 | 0 |
| P4 | 0 | 0 | 0 | 0 | 0 | 1 |

Allocation

| | R1 | R2 | R3 | R4 | R5 | R6 |
|----|----|----|----|----|----|----|
| P1 | 0 | 1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 1 | 1 | 0 | 0 | 0 |
| P3 | 1 | 1 | 0 | 0 | 0 | 0 |
| P4 | 0 | 1 | 1 | 1 | 0 | 0 |

Besoin

Q1 : L'état courant est-il sûr (sain) ? Justifier.

Réponse :

En appliquant l'algorithme de vérification de l'état sain, on trouve que le vecteur Finsh est entièrement égal à Vrai :

| | | | |
|---|---|---|---|
| V | V | V | V |
|---|---|---|---|

L'état du système est donc sain.

Q2 : Si P3 demande une ressource R2 peut on lui accorder immédiatement sa requête? Justifier votre réponse en appliquant l'algorithme de vérification de l'état sain. .

Réponse :

On applique l'algorithme du banquier :

Etape 1 : Request3 <= besoin3 ? Oui

Etape2 : Request3 <= disponible ? Oui

Etape3 : Sauvegarde de l'état du système.

Modification des matrices

Algorithme de vérification de l'état sain :

Algorithme de vérification de l'état sain :

| Work | | | | | |
|------|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |

| Finish | | | |
|--------|---|---|---|
| F | F | F | F |

Le vecteur Finish final n'est pas entièrement égal à Vrai .

L'état du système n'est pas sain. Annuler les modifications sur les matrices. On ne pas satisfaire la requête immédiatement.

Q3 : Si P1 demande une ressource R2 peut on lui accorder immédiatement sa requête?
Justifier votre réponse en appliquant l'algorithme de vérification de l'état sain.

Réponse :

On applique l'algorithme du banquier :

Etape 1 : Request1 <= besoin1 ? oui

Etape2 : Request1 <= disponible ? oui

Etape3 : Sauvegarde de l'état du système.

Modification des matrices

Algorithme de vérification de l'état sain :

| | | |
|-------------|---------------|---------------|
| Work | Finish | |
| 0 0 0 0 0 0 | F F F F | |
| I=0 | Work | Finish |
| | 1 1 1 0 0 0 | V F F F |
| I=2 | Work | Finish |
| | 1 1 1 1 0 0 | V V F F |
| I=3 | Work | Finish |
| | 1 1 1 1 1 0 | V V V F |
| I=4 | Work | Finish |
| | 1 1 1 1 1 1 | V V V V |

Le vecteur Finish final est entièrement égal à Vrai.

L'état du système est sain. La requête peut être accordée.

Exercice N°3 (5 pts)

Un train est constitué de N wagons. Chaque wagon est doté d'une porte automatique. Devant chaque porte se trouve un bouton qui permet au passager de l'ouvrir en poussant dessus. Pour la sécurité des passagers les règles suivantes doivent être strictement respectées :

- Un passager ne peut pas ouvrir une porte si le train est en marche.
- Le train ne peut pas démarrer si au moins l'une des portes est ouverte.
- Une porte se ferme automatiquement après 30 secondes de son ouverture

Le programme qui gère l'ouverture/fermeture des portes s'exécute sur un système d'exploitation multitâche dans le système informatique du train. Il est composé des fonctions suivantes :

- ***Démarrer_train()*** : fonction exécutée par le conducteur de train. Son rôle est de démarrer le train.
- ***Arrêter_train()*** : fonction exécutée par le conducteur de train. Son rôle est d'arrêter le train.
- ***Ouvrir_porte()*** : fonction exécutée par le passager en appuyer sur un bouton. Son rôle est d'ouvrir une porte.
- ***Fermer_auto_porte()*** : fonction exécutée automatiquement 30 secondes après l'ouverture d'une porte.

Ces fonctions s'exécutent simultanément et modifient de façon concurrentielle les variables état_train et nbr_portes_ouvertes :

Etat_train = 'M' si le train est en marche

Etat_train= 'A' si le train est en arrêt.

Nbr_portes_ouvertes : nombre de portes ouvertes à un instant donné.

Initialement etat_train= 'A', nbr_portes_ouvertes =0 ;

Q1 : proposer une solution à base de sémaphores d'exclusion mutuelle et de sémaphores de synchronisation pour commander l'ouverture/fermeture des portes ainsi que le démarrage du train en écrivant le code des fonctions démarrer_train(), arreter_train(), ouvrir_porte(), fermer_auto_porte()

Remarque : écrire la solution à base de primitives P et V et expliquer le rôle de chaque sémaphore utilisé (il n'est pas demandé d'utiliser le langage c)

Solution exercice

```
Etat_train='A' ;  
Nb_Portes_Ouvertes=0;  
Semaphore mutex=1 ;  
Semaphore sync=0;
```

Démarrer_train()

```
{  
P(mutex)  
If (Nb_Portes_Ouvertes > 0) {           // Il existe au moins un porte ouverte  
P(sync)                                // Se bloquer  
}  
Démarrer ...  
Etat_train='M'  
V(mutex)  
}
```

Arrêter_train()

```
{  
P(mutex)  
Arrêter ...  
Etat_train='A' ;  
V(mutex)  
}
```

Ouvrir_porte()

```
{  
P(Mutex)  
If (Etat_train=='A') { // Ouvrir une porte seulement si le train est en arrêt  
Nb_Portes_Ouvertes= Nb_Portes_Ouvertes+1 ;  
}  
V(Mutex)  
}
```

Fermer_auto_porte()

```
{  
P(Mutex)  
Nb_Porte_Ouvertes= Nb_Porte_Ouvertes-1 ;  
If (Nb_Porte_Ouvertes == 0) { // La dernière porte a été fermée  
V(sync) // Débloquer le démarrage du train  
}  
V(Mutex)  
}
```