USTHB
Faculté d'Electronique et Informatique
Département Informatique
Epreuve de Rattrapage – Durée : 1h30
(Systèmes d'exploitation- M1: HPC+IV) Le 29/01/2019 - Année 18/19

Exercice n°1 : (6 pts= 1.5*4)

Répondre aux questions suivantes :

A- Quelle est la différence entre un moniteur classique et un moniteur avec condition de Kessels ?
B- Définir une boite aux lettres privée.
C- Donner le cas précis d'utilisation des régions critiques simples.
D- Citer les deux méthodes principales qui permettent de vérifier si un système de taches est déterminé.

Exercice 2 : (7 pts=4.5+2.5)

On considère le modèle du producteur / consommateur avec un tampon de taille *N*, défini comme suit :
a- *N=C*k* cases où *C* et *k* sont deux entiers naturels,
b- Le producteur dépose au même moment et à chaque fois dans *k* cases du tampon, mais le consommateur prélève à chaque fois une seule case.
- Donner une solution à ce problème à l'aide des sémaphores en considérant un seul processus producteur et un seul consommateur,
- Généraliser la solution à plusieurs producteurs et plusieurs consommateurs.

Exercice n°3 : (7 pts= 3.5 + 2.5 + 1)

Etant donné le moniteur suivant:

```
X : Moniteur ;
Var     Dispo, Cpt, Nb : entier ;  F : file d'entier;     C : condition ;

Entry procedure P1(k: entier)
Début
  Si (Dispo<k) Alors
        Insérer (F, k) ; Nb:=Nb+1; C.Wait(); Nb:=Nb-1; Extraire (F);
        Tantque (Dispo<k) ou (non vide(F) et (k > min (F)) Faire
                Insérer (F , k) ;  Nb:=Nb+1; C.Wait(); Nb:=Nb-1 ; Extraire(F);
        Fait ;
  Fsi ;
Dispo = Dispo – k ;
Fin ;
Entry procedure P2 (m : entier) ;
        Var i : entier ;
Début
        Dispo = Dispo+ m; Cpt :=Nb ; Pour i :=1 à Cpt Faire C.Signal()  Fait ;
Fin ;
Début Dispo = N ; Nb :=0 ; F :=ϕ Fin.
```

Où vide(f): fonction qui retourne vrai si la file F est vide, Min(F) : fonction qui retourne la valeur la plus petite dans F sans modification de F, Insérer (F, k) : Insère la valeur k à la fin de la file F, et Extraire (F) : supprime la valeur en tête de la file F.

A/ Expliquer (interpréter) le fonctionnement de cette solution.
B/ Que fait cette solution ?
C/ Réécrire cette solution en adoptant la politique inverse.

Bon Courage

USTHB
Faculté d'Electronique et Informatique
Département Informatique
Correction de l'Epreuve de Rattrapage
(Systèmes d'exploitation- M1: HPC+IV) Le 29/01/2019 - Année 18/19

Exercice n°1 : (6 pts= 1.5*4)

Répondre aux questions suivantes :

A- La différence réside dans les primitives *Wait()* et *Signal ()*. Dans un moniteur classique, l'attente s'exprime sur une condition symbolique qui est une variable sans valeur et le réveil se fait explicitement avec la primitive *Signal()* sur la condition en question. Dans un moniteur avec condition de Kessels, l'attente s'exprime sur une condition booléenne réelle exprimée en fonction des variables de synchronisation et le réveil éventuel se fait en réévaluant cette même condition à la sortie d'un autre processus du moniteur ou à son blocage sur une condition. Ceci implique que la primitive *Signal ()* devient sans signification.

B- Une boite aux lettres privée à un processus est une boite aux lettres dont seulement ce processus ne peut qu'exécuter *Recevoir ()* sur cette boite aux lettres et les autres processus n'ont droit qu'à la primitive *Envoyer ()* sur cette même boite aux lettres.

C- Les régions critiques simples sont utilisées dans l'exclusion mutuelle.

D- Les deux méthodes principales qui permettent de vérifier si un système de taches est déterminé sont
- la définition du système de taches déterminé et
- le théorème qui stipule qu'un système de taches est déterminé si les taches de ce système sont deux à deux non - interférentes.

Exercice n°2 : (7 pts=4.5+2.5)

1/ Il s'agit du modèle d'un producteur et d'un consommateur, donc on a besoin de deux sémaphores nvide et nplein. Cependant le modèle possède la particularité suivante :
Le producteur dépose k cases à la fois et puisque N=C*k donc il ne peut déposer que C fois avant de remplir le tampon, donc nvide doit être initialisé à C. Une fois qu'il dépose k articles, le producteur doit faire V(nplein) k fois. Par conséquent et sachant qu'il prélève d'une seule case à la fois, le consommateur ne peut faire V(nvide) qu'après avoir vidé k cases. La solution est donc comme suit :

Const k =… ; C =….
Var Sémaphore nvide :=C ; nplein :=0 ;

Processus producteur ;	**Processus** Consommateur ;
Var i : entier ;	**Var** cpt : entier :=0 ;
Début	**Début**
Répéter	**Répéter**
<Produire k-articles >;	P (nplein) ;
P (nvide) ;	<Prélever-1-article> ;
<Déposer-k-articles> ;	cpt :=cpt+1 modulo k ;
Pour i :=1 **à** k **Faire** V(nplein) **Fait** ;	**Si** cpt=0 **Alors** V(nvide) **Fsi** ;
Jusqu'à faux ;	**Jusqu'à** faux ;
Fin.	**Fin**.

2/ Généralisation

Var Sémaphore nvide :=C ; nplein :=0 ; mutexp :=1 ; mutexc :=1 ;
cpt : entier ;

Processus producteur ;	**Processus** Consommateur ;
Var i : entier ;	**Var** cpt : entier :=0 ;
Début	**Début**
Répéter	**Répéter**
<Produire k-articles> ;	P (nplein) ;
P (nvide) ;	P(mutexc)
P(mutexp) ;	<Prélever-1-article> ;
<Déposer-k-articles> ;	cpt :=cpt+1 modulo k ;
V(mutexp) ;	**Si** cpt=0 **Alors** V(nvide) **Fsi** ;

Pour i :=1 **à** k **Faire** V(nplein) **Fait** ;
Jusqu'à faux ;
Fin.

V(mutexc) ;
Jusqu'à faux ;
Fin.

Exercice n°3 : (7 pts= 3.5 + 2.5 + 1)

A/

- Explication du fonctionnement :

Remarquons tous d'abord que :

- Dans P1, le processus commence par tester si *Dispo<k*, dans le cas positif il attend, dans le cas contraire, il diminue *Dispo* de la valeur *k* et termine P1.

- Dans P2, le processus augmente *Dispo* de la valeur *k*, puis réveille tous les processus bloqués.

Donc, il s'agit de la gestion de l'allocation d'une classe de ressources à *N* exemplaires. : P1≡ Demande et P2 ≡ Libération.

Examinons de près cette solution : Quand un processus exprime une demande, il se bloque si le nombre de ressources disponibles est insuffisant *(dispo<k)* (1) après avoir inséré *k* à la fin de la liste *F*. Dans le cas contraire, on lui alloue les *k* exemplaires demandées.

Quand un processus libère *m* exemplaires (i.e exécute P2), il les ajoute à *Dispo (Dispo :=Dispo+m)* puis réveille tous les processus en attente. A chaque réveille d'un processus, il attend jusqu'à ce que le processus réveillé sort de P1 ou se bloque une autre fois.

Quand un processus est réveillé,(le traitement est le même pour les processus bloqués au niveau du premier *Wait()* et pour ceux bloqués au niveau du deuxième *Wait()*), il élimine la valeur en tête de *F* représentant le nombre d'exemplaires qu'il demande et se bloque une autre fois si le nombre de ressources disponibles est insuffisant ou si sa demande *k* n'est pas la plus petite *(k>min(F))* parmi les demandes en attente (2).

Remarquons que la file contient à tous moment et uniquement les demandes des processus en attente.

B/ Cette solution gère l'allocation d'une classe de ressources à *N* instances. D'après (1) et (2), la priorité est donnée au processus qui demande le moins d'instances. Remarquons que l'ordre relatif entre les processus exprimant la même demande n'est pas tout à fait respecté.

C/ Solution avec une politique inverse : Satisfaire en priorité celui qui demande le plus d'exemplaires de ressources.

On doit remplacer la fonction *Min()* par la fonction *Max()* qui retourne la valeur la plus grande dans *F*.

```
X : Moniteur ;
Var    Dispo, Cpt, Nb : entier ;  F : file d'entier;     C : condition ;

Entry procedure P1 (k: entier)
Début
  Si (Dispo<k) ou (non vide (F)) Alors
        Insérer (F, k) ; Nb:=Nb+1; C.Wait(); Nb:=Nb-1; Extraire (F);
        Tantque (Dispo<k) ou ((non vide(F) et (k < Max (F)) Faire
              Insérer (F, k) ;  Nb:=Nb+1; C.Wait(); Nb:=Nb-1 ; Extraire(F);
        Fait ;
  Fsi ;
Dispo = Dispo – k ;
Fin ;
Entry procedure P2 (m : entier) ;
        Var i : entier ;
Début
        Dispo = Dispo+ m; Cpt :=Nb ; Pour i :=1 à Cpt Faire C.Signal()  Fait ;
Fin ;
Début Dispo = N ; Nb :=0 ; F :=ϕ Fin.
```