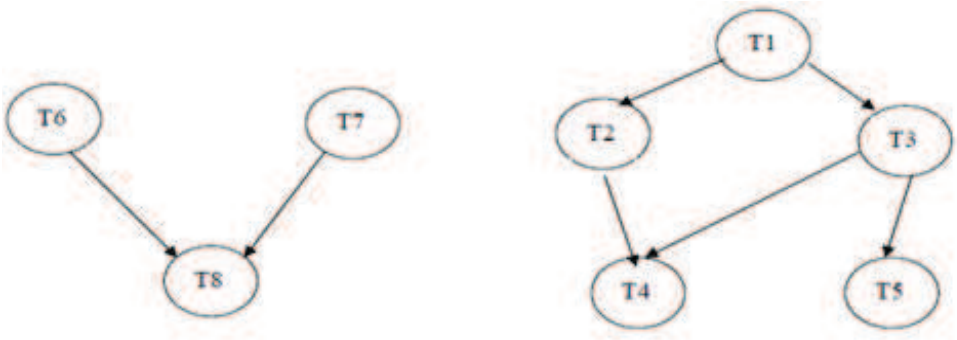
Université A. Mira Bejaia 25/ 01/2014
FSE/Département d'Informatique
L3 Académique.

Examen Systèmes d'Exploitation II

Exercice 1: (4,5 pts)

Question 1) Les mots suivants sont-ils dans le langage associé au système définit par le graphe de précédence au-dessous ? **Justifiez si la réponse est non.**

a) d1 d6 f1 d2 f6 d3 d7 f3 f2 d4 f7 d8 f4 d5 f5 f8
b) d6 d7 f6 d1 f1 f7 d2 f2 d3 f3 d4 d8 f4 d5 f5 f8
c) d1 f1 d2 d3 d7 f2 d6 d4 f3 d5 f7 f6 d8 f8 f5 f4



graphe de precedence

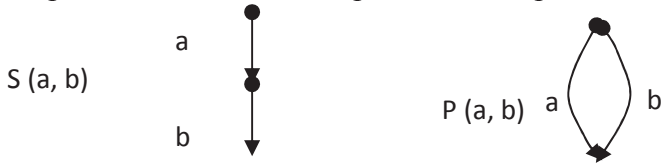
Question 2) Les expressions suivantes décrivent les relations de précédence parallèle et séquentiel entre six processus P1 à P6.

P(**S** (**P**(p3, **S**(p1, **P**(p6, p5))), p2), p4)

Transformez cette expression en un programme, en utilisant :
(a) parbegin/parend (en exploitant au maximum le parallélisme)
(b) les primitives : fork, join et quit.

Remarque :

- S (a, b) : représente l'exécution en séquentiel des processus a et b
- P (a, b) : représente l'exécution en parallèles des processus a et b



Exercice 2: (6pts)

Partie A : Trouver toutes les erreurs logiques dans le segment de code suivant, en indiquant : le numéro de la ligne erronée, ce qui est erroné puis; corriger l'erreur.

```
1. semaphore mutex = 0;
   /*  add_item_to_queue renvoie vrai si l'élément « item» a été  ajouté à la file «queue », faux sinon.*/
2. Boolean add_item_to_queue(queue_t queue, item_t item)
3. {
4.  P(mutex);
5.  if (is_full(queue)) //si la file est pleine
6.  {
7.   return false;
8.  }
9.  Else{
10.     append_to_queue(queue, item); // ajouter dans la file
11.     return true;
12.     V(mutex);
13.     }
14.}
```

Partie B: Considérons le pseudo-code du moniteur suivant:

```
   monitor m() {
     int x=10 ; y=2 ;
     condition c;

     A() {
(1)   x++;
(2)   c.signal;
(3)   y = x-2; }

     B() {
(4)   if (x>10)
(5)   x--
(6)   else  {c.wait;
(7)          x--;} }
   }
```

Supposons que, après l'initialisation du moniteur, les fonctions A et B sont appelés dans l'ordre suivant par divers processus:

m.A(); m.A(); m.B(); m.B(); m.B(); m.B(); m.A(); m.A();

Question) En utilisant les numéros de lignes dans le code, tracer la séquence d'exécution d'instruction, sous forme du tableau donné ci-après. Montrer les valeurs de x et y à la fin de chaque instruction.

Appel de la procédure Du moniteur	Numéro de l'instruction	Valeur de X	Valeur de Y	Commentaire (si nécessaire)
… …	… …	… …	… …	

Exercice 3: (6pts)

Soient les deux processus P1 et P2 suivants. Ils partagent deux sémaphores, S1 et S2 initialisés à 0.

```
Processus p1
{
    Procedure A1;
    V(S2) ;
    P(S1) ;
    Procedure B1;
}
```

```
Processus p2
{
    Procedure A2;
    V(S1) ;
    P(S2) ;
    Procedure B2;
}
```

1) Quelle synchronisation a-t-on imposée sur les exécutions des procédures A1, A2, B1, B2 ?

2) Ecrire le code afin d'imposer la même synchronisation pour 3 processus P1, P2, et P3 en utilisant 3 sémaphores.

3) Ecrire le code afin d'imposer la même synchronisation pour N processus P1, P2,…, P_n en utilisant N sémaphores.

Exercice 4 : Questions de cours (3,5pts)

Question 1) Répondez par VRAI ou FAUX

a) Tant qu'un processus est bloqué dans la file d'attente d'un sémaphore, il est engagé dans l'attente active.
b) L'exclusion mutuelle peut être réalisée avec un sémaphore général dont la valeur initiale est supérieure à 1.

Question 2) Donnez les principaux inconvénients des sémaphores?

Question 3) Expliquez chacun des termes suivants en précisant la différence principale entre les deux termes.

- "*sémaphore binaire*" et " *sémaphore général*"

BON COURAGE

Corrigé de l'Examen Systèmes d'Exploitation II

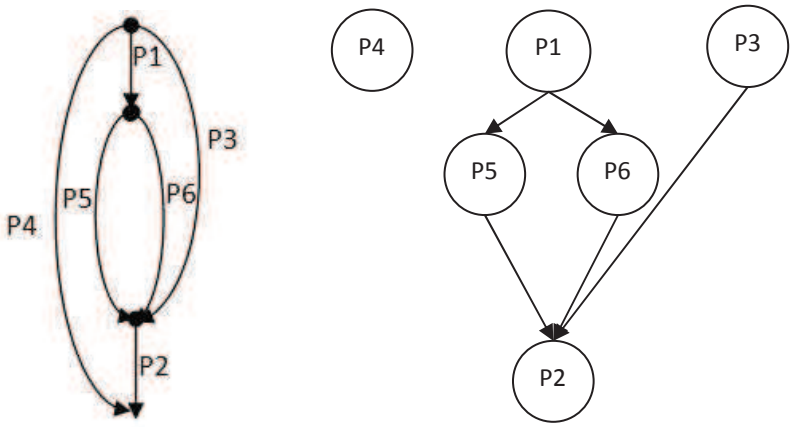
(2013/2014)

Exercice 1 :

Question 1 :

a. Oui
b. Oui
c. Non, parce que d4 < f3 alors que selon le graphe de précédence T3 < T4.

Question 2 :



a. Parbegin/Parend (1,5 pt)

```
Parbegin
    P4 ;
    Begin
            Parbegin
                    P3 ;
                    Begin
                            P1 ;
                            Parbegin
                                    P5 ;
                                    P6 ;
                            Parend
                    End
            Parend
            P2 ;
    End
Parend
```

b. Fork, Join et Quit (1,5 pt)

```
Count2 = 3 ;
        Fork L1 ;
        P4 ;
        Quit ;
L1 :    Fork L2 ;
        P1 ;
        Fork L3 ;
        P5 ;
        Goto L4 ;
L3 :    P6 ;
        Goto L4;
L2:     P3;
L4:     Join Count2;
        P2;
```

Exercice 2:

Partie A :

```
1. semaphore mutex =0; //erreur #1  semaphore mutex = 1;
2. Boolean add_item_to_queue(queue_t queue, item_t item)
3. {
4.  P(mutex);
5.  if (is_full(queue)) //si la file est pleine
6.  {v(mutex); //  erreur #2 (cela manquait)
7.   return false;
8.  }
9.  Else{
10.     append_to_queue(queue, item); V(mutex);
11.     return true;
12.     V(mutex); //erreur #3 cela devrait être  avant return
13.     }
14.}
```

Partie B:

Appel Proc	N° Instruction	Valeur X	Valeur Y	Commentaire
A()	(1)	11	2	
	(2)	11	2	
	(3)	11	9	
A()	(1)	12	9	
	(2)	12	9	
	(3)	12	10	
B()	(4)	12	10	
	(5)	11	10	
B()	(4)	11	10	
	(5)	10	10	
B()	(4)	10	10	
	(6)	10	10	Le processus est mis en attente sur la condition c
B()	(4)	10	10	
	(6)	10	10	Le processus est mis en attente sur la condition c
A()	(1)	11	10	
	(2)	11	10	Le processus courant est suspendu au profit du processus réveillé
B()	(7)	10	10	Le processus précédemment suspendu doit reprendre
A()	(3)	10	8	
A()	(1)	11	8	
	(2)	11	8	Le processus courant est suspendu au profit du processus réveillé
B()	(7)	10	8	Le processus précédemment suspendu doit reprendre
A()	(3)	10	8	

Exercice 3 :

1) L'exécution de la procédure B1 doit attendre la fin d'exécution de A2 et B2 doit attendre la fin de A1.

2) Pseudo-code avec 3 processus en utilisant trois sémaphores : avec deux processus, chacun attend l'autre. De même, avec trois processus chacun attend les autres. D'où le code suivant :

Sémaphores s1, s2, s3=0 ;

```
Processus p1
{
Procedure A1;
V(S2) ;v(S3)
P(S1) ; P(S1) ;
Procedure B1
}
```

```
Processus p2
{
Procedure A2;
V(S1) ;v(S3);
P(S2) ; P(S2) ;
Procedure B2
}
```

```
Processus p3
{
Procedure A3;
V(S1) ;v(S2);
P(S3) ; P(S3) ;
Procedure B3
}
```

3) Code avec N processus avec N sémaphores : s1, s2, s3,...Sn=0 ;

```
Processus p1
{
Procedure A1;
V(S2) ; V(S3)…V(Sn); P(S1); P(S1) ;
…p(S1); /*(n-1) fois*/
Procedure B1
}
```

```
Processus p2
{
Procedure A2;
V(S1) ; V(S3)…V(Sn);
P(S1); P(S2) ; …p(S2); /*(n-1) fois*/
Procedure B1
}
```

.....

```
Processus Pn
{
Procedure An;
V(S1) ; V(S2)…V(Sn-1); /*le processus n n'est pas inclus */
P(Sn); P(Sn) ; …p(Sn); /*(n-1) fois*/
Procedure B1
}
```

Exercice 4 : Questions de cours

Question 1)

a) (faux)
b) (faux)

Question 2) (

Réponse :

1) De simples algorithmes nécessitent plus d'un sémaphore.
2) Les sémaphores sont de très bas niveau
3) Il est facile de faire des erreurs de programmation (ex. P(s) suivi de p(s))
4) Le programmeur doit garder la trace de tous les appels de p et v du sémaphore.
5) ....etc.

Question 3)

1) ils sont les deux utilisés pour réglementer l'accès aux ressources partagées.
2) Un sémaphore binaire:
   - a une valeur 0 ou 1 / (exclusion mutuelle)
   - convient pour seule instance d'une ressource
3) Un sémaphore général :
   - a une valeur entière de 0 à n.
   - convient pour plusieurs ressources de même type (identiques).