Examen système d'exploitation Département d'informatique, Université de Bouira

1ère Année Master ISIL (2017/2018) Durée : 1h30 **Documents non autorisés**

Exercice 1 (3 pts) :

1. Qu'est-ce qu'un PCB et de quoi est-il composé ? **(0,5 pt)**

PCB est une structure de données particulière appelée bloc de contrôle de processus (PCB : Process Control Bloc) et dont le rôle est de reconstituer le contexte du processus.

PCB est composé de l'ensemble des informations dynamiques qui représente l'état d' exécution d'un processus. Il contient aussi des informations sur l'ordonnancement du processus (priorité du processus, les pointeurs sur les files d'attente)

2. L'appel système fork() est le moyen pour créer des processus, par duplication d'un processus existant. Expliquer comment distinguer entre le processus père et le processus fils. **(0,5 pt)**

Pour distinguer le processus père du processus fils on regarde la valeur de retour de fork(), qui peut être:

- La valeur 0 dans le processus fils.
- Positive pour le processus père et qui correspond au PID du processus fils.
- Négative si la création de processus fils a échoué ;

3. Quelles critiques peut-on faire aux solutions de l'**attente active** et **Peterson** du problème de l'exclusion mutuelle ?

Attente active : **(1 pt)**

1- Cette méthode n'assure pas l'exclusion mutuelle

2- Susceptible de consommer du temps en bouclant inutilement.

Peterson : **(1 pt)**

1- La généralisation de cette solution aux cas de plusieurs processus est bien complexe.

2- Susceptible de consommer du temps en bouclant inutilement.

Exercice 2 (6 pts) :

On cherche à évaluer l'expression suivante

e := ((b-d) * (a+c) + (e*f)) / (a+c)

1. Réaliser un découpage en tâches de cette expression sans l'ajout de variables intermédiaires. (2 pts)

t1: b = b - d
t2: a = a + c
t3: e = e * f
t4: b = b * a
t5: b = b + e
t6: e = b / a

2. En vous servant de la définition de la condition de Bernstein, donner le graphe de précédence correspondant.

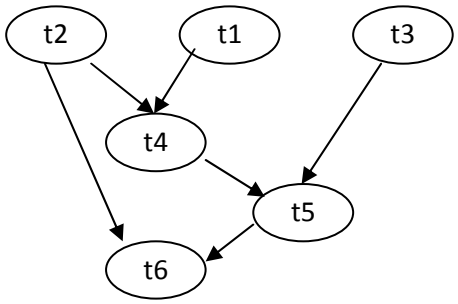
t_1: {R(t_1) = d, W(t_1) = b};

t_2: {$R(t_2) = c$, $W(t_2) = a$};
t_3: {$R(t_3) = f$, $W(t_3) = e$}
t_4: {$R(t_4) = a$, $W(t_4) = b$};
t_5: {$R(t_5) = e$, $W(t_5) = b$};
t_6: {$R(t_6) = b, a$, $W(t_6) = e$}

On dira que deux tâches (instructions) t_i et t_j peuvent s'exécuter en parallèle si les conditions suivantes sont satisfaites : **(2pts)**

a) $R(t_i) \cap W(t_j) = \varnothing$
b) $W(t_i) \cap R(t_j) = \varnothing$
c) $W(t_i) \cap W(t_j) = \varnothing$

Le graphe est le suivant



En se basant sur le graphe obtenu, réaliser la synchronisation des tâches (processus) en utilisant trois (03) sémaphores S1, S2 et S3. **(2pts)**

Sémaphore S1=0, S2=0, S3=0

```
Pt1 (){
t1: b = b - d;
V(S1);
}
```

```
Pt2 (){
t2: a = a + c;
V(S1); V(S3);
}
```

```
Pt3 (){
t3: e = e * f;
V(S2);
}
```

```
Pt4 (){
P(S1); P(S1);
t4: b = b * a
V(S2)
}
```

```
Pt5 (){
P(S2);P(S2);
t5: b = b + e;
V(S3)
}
```

```
Pt6 (){
P(S3);P(S3);
t6: e = b / a;
}
```

Exercice 3 **(7 pts) :**

On considère le problème du producteur consommateur. Le processus **producteur**, délivre des messages à un processus **consommateur**. Le **producteur** produit le message dans la *ZoneP*, puis le dépose dans le buffer. Le **consommateur** prélève un message du buffer et le place dans la ZoneC où il peut le consommer. Ce problème induit plusieurs types de contraintes de synchronisation qui sont:

1. Un producteur ne doit pas produire, ni déposer le message dans le buffer quand il n y a plus de place pour déposer ce qu'il a produit.
2. Un consommateur ne doit pas prélever de messages quand le buffer est vide.
3. Producteurs et consommateurs ne doivent pas accéder en même temps au buffer.

- Écrire l'algorithme des processus producteurs et consommateurs en utilisant les sémaphores.

Semaphore Mutex = 1, **Plein = 0, Vide=n ;** Message tampon[];

```
Producteur (){
Message m ;
Tantque Vrai faire
   P(Vide);
      m = creermessage() ;
      P(Mutex) ;
         EcritureTampon(m);
      V(Mutex) ;
   V(Plein);
FinTantque
}
```

```
Consommateur (){
Message m ;
Tantque vrai faire
   P(Plein);
      P(Mutex);
            m = LectureTampon();
      V(Mutex);
   V(Vide);
Fin Tantque
}
```

Exercice 4 (4 pts) : Soit le programme ci-dessous :

```
# define N 100
int k = 0;
int main () {
float t[N];
int i;
pid_t P
for (i=0;i<N;i++)
        t[i] = rand ();
   P = fork ();
if (P ==0) {
k=1;
for(i=0 ; i<N ; i++)
        if (t[0] < t[i])
                t[0]= t[i];
}
else {
        for(i=1;i<N;i++)
        t [0]+= t[i];
        waitpid (P ,NULL ,0);
}
printf ("processus%d, %f \n",k, t[0]);
return 0;
}
```

1- Que fait le "processus 0" et le "processus 1" dans le code ci-dessus? Expliquez. **(3pts)**

1- Le processus 0 (le père) initialise un tableau aléatoirement, exécute l'appel système fork, calcule la somme des éléments du tableau dans t[0], attend la terminaison de son fils et affiche t[0].
Le processus 1 (le fils) calcule le maximum du tableau dans t[0] qui il affiche.

2- Y a-t-il un risque pour que les deux processus fournissent des résultats incohérents ? Expliquez. **(1pt)**
Remarque : les processus s'annoncent à la fin du code avant le return 0

2 Non, car les deux processus ont des espace de données différents (pas de variable partagée)