Examen système d'exploitation L3-SI Durée : 1h00

Exercice 1 (4 pts) :

1. Quelles sont les inconvénients des solutions de l'**attente active** et **Peterson** du problème de l'exclusion mutuelle ?

2. **Attente active :** **(2 pt)**
3. 1- Cette méthode n'assure pas l'exclusion mutuelle
4. 2- Susceptible de consommer du temps en bouclant inutilement.
5. **Peterson :** **(2 pt)**
6. 1- La généralisation de cette solution aux cas de plusieurs processus est bien complexe.
7. 2- Susceptible de consommer du temps en bouclant inutilement.

Exercice 2 (5 pts) :

On cherche à évaluer l'expression suivante

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

1. Réalise un découpage en tâches de cette expression sans l'ajout de variables intermédiaires. 1pt

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

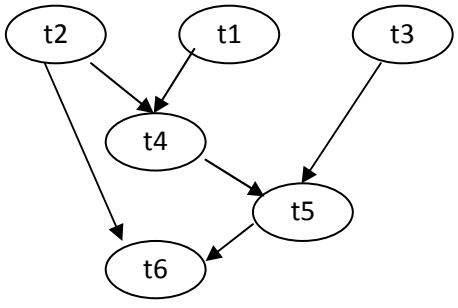
2. En vous servant de la définition de la condition de Bernstein, donner le graphe de précédence correspondant. 2 pt

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 :

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

t1 // t2
t1 // t3
t1 → t4 : t1 précéde t4 puisque le b de t4 est celui calculé par t1
t1 ↛ t5 : t1 ne précéde pas t5 puisque le b de t5 ($W(t_1) \cap W(t_5) = \{b\}$) ce n'est pas celui calculé par t1 mais c'est celui de t4
t1 ↛ t6 : même chose
t2 // t3
t2 → t4
t2 // t5
t2 → t6
t3 // t4
t3 → t5
t3 ↛ t6



t4 → t5
t4 ↛ t6
t5 → t6

Examen système d'exploitation **L3-SI** Durée : 1h00

3. 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 (){          Pt2 (){          Pt3 (){          Pt4 (){          Pt5 (){          Pt6 (){
t1: b = b - d;   t2: a = a + c;   t3: e = e * f;   P(S1); P(S1);    P(S2);P(S2);     P(S3);P(S3);
V(S1);           V(S1); V(S3);    V(S2);           t4: b = b * a    t5: b = b + e;   t6: e = b / a;
}                }                }                V(S2)            V(S3)
                                                   }                }
```

Exercice 3 (6 pts) :

Soit une route reliant la ville A et la ville B. Les règles de circulation sur cette route sont les suivantes:

1. Des voitures peuvent y circuler ensemble dans le sens A → B ou dans le sens B → A (circulation alternée),
2. La route ne doit jamais être empruntée simultanément par deux voitures allant en sens inverse.
3. La priorité d'accès à la route est la même pour les deux sens.

On considère donc deux classes d'utilisateurs (processus) : voitures de A vers B (V_AB) et voitures de B vers A (V_BA).

Question:

En utilisant les sémaphores, écrivez les algorithmes des fonctions de demande d'accès aux tronçons acces_AB(), acces_BA() et des fonctions de sortie des tronçons AB_sort () et BA_sort() de façon à ce que les processus V-AB et V-BA respectent les trois règles de circulation sur la route. **Précisez clairement vos déclarations et initialisations**.

Solution:

Initialisation ***2pts***

int nbAB=0; //Compte le nombre de voitures allant de A vers B

int nbBA =0; entier (init à 0) Compte le nombre de voitures allant de B vers A

Sempahore route=1; // pour permettre que des voitures de sens contraire ne s'engagent pas dans la voie en même temps.

Sempahore mutexAB=1;//pour protéger la mise à jour en exclusion mutuelle de la variable protégée nbAB

Sempahore mutexBA=1; //pour protéger la mise à jour en exclusion mutuelle de la variable protégée nbBA

Code des fonctions **1pt/fonction**

```
Fonction acces_AB()  {
    P(mutexAB);
    nbAB++;
    si (nbAB == 1) P(route);
    V(mutexAB);}
Fonction AB_sort ()
    P(mutexAB);
    nbAB--;
    si (nbAB == 0) V(route);
    P(mutexAB);}
```

```
Fonction acces_BA ()  {
    P(mutexBA);
    nbAB++;
    si (nbAB == 1) P(route);
    V(mutexBA);}
Fonction BA_sort ()  {
    (mutexBA);
    nbBA--;
    si (nbBA == 0) V(route);
    (mutexBA);}
```

Exercice 4 (5 pts) :

Écrire un programme qui crée 5 processus fils, chaque fils affiche son PID et se termine. Le père devra attendre la fin de ses 5 fils et afficher quel a été le premier et dernier processus qui a terminé.

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <sys/types.h>
#include <unistd.h>
int main(){
pid_t pid_premier, pid_dernier;
int i,pid[5],status;
  for(i=0;i<5;i++){
    pid[i]=fork();
    if (pid[i] == -1) {
              // ERREUR
          fprintf(stderr, "Impossible de créer un fils (%d)\n", i);
    } else if (pid[i]==0) {
                    // FILS
                    printf("Fils %2d (PID=%d): Activé\n", i, getpid());
                    exit(0);
    } else {
                     printf("Père : Activation du fils %2d\n", i);
          }
  }
pid_premier = wait(&status);
wait(&status);
wait(&status);
wait(&status);
pid_dernier = wait(&status);
printf("Premier processus a finir : %d\n", pid_premier);
printf("Dernier processus a finir : %d\n", pid_dernier);
return 0;
}
```