Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Licence 3ème année

UE Parallélisme I

Corrections – Janvier 2011

Durée : 1h45 — Documents autorisés

Threads & Sémaphores – (9 points)

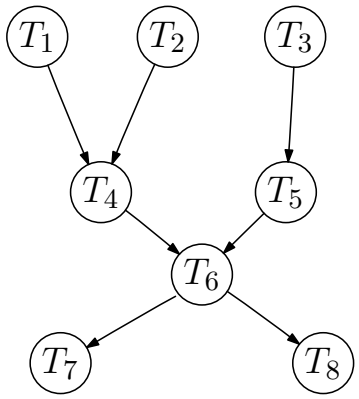
1 – 4pts Les trois *threads*, A, B et C suivantes sont exécutées de manière concurrente ; elles utilisent pour leur synchronisation deux sémaphores `S1` et `S2`.
La valeur initiale de `S1` est 1, celle de `S2` est 0 et celle de `x` est 0.

```
Thread A {              Thread B {              Thread C {
   P(s2);                  P(s1);                  P(s1);
   x = x * 2;              x = x * x;              x = x + 3;
   V(s1);  }               V(s1);  }               V(s2);  }
```

a. Quelles sont les différentes valeurs possibles de la variable `x` à la fin de l'exécution complète des trois *threads* ?
Le fait que `S1` *soit initialisée à 1 permet deux successions d'exécutions différentes pour les différentes threads :*
- ◇ *Thread B, puis Thread C et enfin Thread A, ce qui donne* $x = 6$.
- ◇ *Thread C, puis Thread A et enfin Thread B, ce qui donne* $x = 36$.

b. Qu'est-ce que cela changerait si la valeur initiale de `S1` était 2 au lieu de 1 ?
Si on initialise S1 à 2, alors les threads B & C peuvent s'exécuter simultanément et la valeur de x *deviendra imprévisible et sûrement incohérente.*

2 – 5pts Écrire un algorithme pour résoudre le problème suivant :




```
T1 {              T2 {              T3 {
   // Travail        // Travail        // Travail
   V(s4);  }         V(s4);  }         V(s5);  }

T4 {                          T5 {
      P(s4);                     P(s5);
      P(s4);                     // Travail
      // Travail                 V(s6);  }
      V(s6);  }

T6 {              T7 {              T8 {
   P(s6);            P(s7);            P(s8);
   P(s6);            // Travail }      // Travail }
   // Travail
   V(s7);
   V(s8);  }
```

Question :

a. Combien de Sémaphores faudra-t-il pour réaliser ce travail ?
5 Sémaphores sont nécessaires.

Création de processus avec fork – (3 points)

3 – Indiquez ce qui va être affiché à l'exécution du programme, dont le source est le suivant :

3pts

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main(void)
5  {
6    int x = 1; int resultat;
7
8    resultat = fork();
9    if (resultat)
10   {
11       x = x * 2;
12       printf("Je suis le processus A et x vaut %d",
   x);
13   }
14   else
15   {
16       x = x + 3;
17       printf("Je suis le processus B et x vaut %d",
   x);
18       resultat = fork();
19       if (resultat)
20       {
21           x = x + 5;
22           printf("Je suis le processus C et x vaut
   %d", x);
23       }
24   }
25   x = 3 * x;
26   printf("Je suis le processus D et x vaut %d", x);
27 }
```

L'affichage est le suivant :

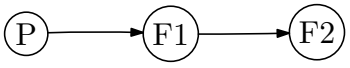
```
Je suis le processus A et x vaut 2
Je suis le processus D et x vaut 6
Je suis le processus B et x vaut 4
Je suis le processus C et x vaut 9
Je suis le processus D et x vaut 27
Je suis le processus D et x vaut 12
```

Signaux & tubes *vs* threads – (8 points)

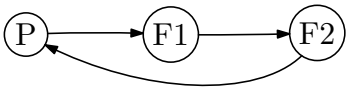
4 – Écrire le programme réalisant le travail suivant :

4pts

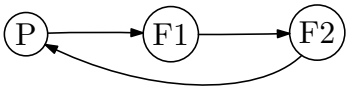
- soit le processus P :
  - ◇ il crée deux fils F1 & F2 ;
  - ◇ il communique à l'aide d'un tube avec F1 ;
- F1 communique à l'aide d'un tube avec F2 ;



- On effectue 10 fois le travail suivant :
  - ◇ P envoie à F1 la valeur 1 et attend ;
  - ◇ F1 envoie à F2 la valeur 1 et attend ;
  - ◇ F2 affiche la valeur 1 ;
  - ◇ F2 envoie un signal à P pour le faire recommencer :



  - ◇ P envoie à F1 la valeur 2 et attend ;
  - ◇ F1 envoie à F2 la valeur 2 et attend ;
  - ◇ F2 affiche la valeur 2 ;
  - ◇ F2 envoie un signal à P pour le faire recommencer :



  - ◇ *etc.*

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

#define LECTURE 0
#define ECRITURE 1

int tube_avec_F1[2];
int tube_F1_F2[2];
int nb_repetition;

void travail_F1 ()
{
   int i;
   close(tube_avec_F1[ECRITURE]);
   close(tube_F1_F2[LECTURE]);
   for(i=0; i<10; i++)
   {
      int valeur;
      read(tube_avec_F1[LECTURE], &valeur, sizeof(int));
      write(tube_F1_F2[ECRITURE], &valeur, sizeof(int));
   }
   exit(0);
}
void travail_F2 ()
{
   int i;
   close(tube_avec_F1[LECTURE]);
   close(tube_avec_F1[ECRITURE]);
   close(tube_F1_F2[ECRITURE]);
   for(i=0; i<10; i++)
   {
      int valeur;
      read(tube_F1_F2[LECTURE], &valeur, sizeof(int));
      printf("Val : %d\n", valeur);
      kill(getppid(), SIGUSR1);
   }
   exit(0);
}
void envoyer_valeur()
{
   nb_repetition++;
   write(tube_avec_F1[ECRITURE], &nb_repetition, sizeof(int));
}

int main()
{
   int i;
   pipe(tube_avec_F1);
   pipe(tube_F1_F2);

   if (fork() == 0)
      travail_F1();
   if (fork() == 0)
      travail_F2();
   /* Le travail du parent */
   close(tube_avec_F1[LECTURE]);
   close(tube_F1_F2[LECTURE]);
   close(tube_F1_F2[ECRITURE]);
   signal(SIGUSR1, envoyer_valeur);
   envoyer_valeur();
   while(1){}
}
```

5 – Donnez une solution à base de threads et de sémaphores à l'exercice 4 en vous inspirant des TDs/TP.

4pts

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>

#define LECTURE 0
#define ECRITURE 1

int tube_avec_F1[2];
int tube_F1_F2[2];

sem_t semaphore_main;

void *travail_F1 (void *arg1)
{
   int i;

   for(i=0; i<10; i++)
   {
      int valeur;
      read(tube_avec_F1[LECTURE], &valeur, sizeof(int));
      write(tube_F1_F2[ECRITURE], &valeur, sizeof(int));
   }
   exit(0);
}
void *travail_F2 (void *arg2)
{
   int i;

   for(i=0; i<10; i++)
   {
      int valeur;
      read(tube_F1_F2[LECTURE], &valeur, sizeof(int));
      printf("Val : %d\n", valeur);
      sem_post(&semaphore_main);
   }
   exit(0);
}

int main()
{
   pthread_t id_t1, id_t2;
   int i;
   int nb_repetition = 0;

   pipe(tube_avec_F1);
   pipe(tube_F1_F2);

 sem_init(&semaphore_main, 0, 1);

   pthread_create(&id_t1, NULL, travail_F1, NULL);
   pthread_create(&id_t1, NULL, travail_F2, NULL);
   for(i=0; i<10; i++)
   {
      sem_wait(&semaphore_main);
      nb_repetition++;
      write(tube_avec_F1[ECRITURE], &nb_repetition, sizeof(int));
   }
}
```