الجامعة الدولية للرباط ⵜⴰⵙⴷⴰⵡⵉⵜ ⵜⴰⵅⵔⵕⵕⵉⵜ ⵏ ⵕⵕⴱⴰⵟ Université Internationale de Rabat	*Module : Système d'Exploitation et Programmation Système* ***TP N°6 : Threads et Synchronisation*** *Intervenants : Pr M. Bakhouya, A. Kharbouch, H. El Khoukhi*	**Année universitaire** 2019/2020

Rapport

Fait Par : EL HANAFI Maha

Objectifs :

- Apprendre la programmation multithreading.
- Savoir reconnaître les données partagées entre différents threads.
- Être capable de gérer la synchronisation de threads au moyen des primitives de l'exclusion mutuelle.

Exercice1 :

a)

```c
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <inttypes.h>
#define SIZE 4   //Taille du matrice (Carrée)
int num_thrd;    //Nombre de Threads (argv[1] par l'utilisateur)
 //Matrice C = MAtrice A * MAtrice B
int A[SIZE][SIZE], B[SIZE][SIZE], C[SIZE][SIZE];

//Fonction d'initiatlisation d'une matrice
void init_matrix(int m[SIZE][SIZE])
{
  int i, j, val = 0;
  for (i = 0; i < SIZE; i++)
    for (j = 0; j < SIZE; j++)
      m[i][j] = val++;
}
 //Fonction d'affichage de contenu d'une Matrice
void print_matrix(int m[SIZE][SIZE])
{
  int i, j;
  for (i = 0; i < SIZE; i++)
  {
    printf("\n\t| ");
for (j = 0; j < SIZE; j++)
      printf("%3d ", m[i][j]);

    printf("|");
  }
}
//Fonction de multiplication executée par le thread
//arguments : Slice (morceau de la matrice)
void* multiply(void* slice)
```

```c
void* multiply(void* slice)
{
  //slice est de type int. Une reconversion est obligatoir
  //pour recupérer la valeur depuis Void
  int s = (uintptr_t) slice;
  int from = (s * SIZE) / num_thrd;
  int to = ((s + 1) * SIZE) / num_thrd;
  int i,j,k;
 printf("Thread %d : Calcul du morceau %d (de la ligne %d au %d)\n",s, s, from, to-1);
  for (i = from; i < to; i++)
  {
      for (j = 0; j < SIZE; j++)
      {
        C[i][j] = 0;
        for ( k = 0; k < SIZE; k++)
          C[i][j] += A[i][k]*B[k][j];
      }
  }
  printf("Thread %d à terminé le morceau %d\n",s, s);
  return 0;
}

int main(int argc, char* argv[])
{
  //Pointeur vers un groupe de Threads
  pthread_t* thread;
  int i;
  //tstart : Debut de clacul de matrice. tend : fin de calcul
  struct timeval tstart, tend;
  //Exectime : le temps ecoulé entre tstart et tend
  double exectime;
```

```c
  //Recupérer le nombre de Threads depuis la commande
  if(argc != 2)
  {
    printf("Usage: %s number_of_threads\n",argv[0]);
    exit(-1);
  }
  //Nombre de threads utilisé
  //ex : ./nomprogramme 2
  //(pour deux threads)
  num_thrd = atoi(argv[1]);
 //Initilisation de la matrice A et B par des données quelconque
  init_matrix(A);
  init_matrix(B);
 //Allocation dynamique d'un tableau de taille num_thrd pour les IDs des threads
  thread = (pthread_t*) malloc(num_thrd * sizeof(pthread_t));
 //Top chrono : départ de calcul de temps ecoulé
  gettimeofday( &tstart, NULL );
 for (i = 1; i < num_thrd; i++)
  {
    //Creation de threads. pthread_create (Tableau des IDs avec indice, Attrubuts, Fonction à exécuter, (voi
    //Void* est un pointeur de type générique. la valeur pointer peut etre recupérer par une affectation ver
    //ex : (void*) i -> multiply : int s = i;
    if (pthread_create (&thread[i], NULL, multiply, (void*) (intptr_t) i) != 0 )
    {
      perror("Erreur de création de thread");
      free(thread);
      exit(-1);
    }
  }
// Thread principal traite le premier morceau de la matrice, et donc tous le programme traite quelque chose.
  // Lle programme principale fera tous si le nombre de threads est 1
  multiply(0);
```

```
 98
 99    // Thread principal attends les autres threads pour terminer
100    for (i = 1; i < num_thrd; i++)
101      pthread_join (thread[i], NULL); // pthread_join (ID thread, Retour de fonction);
102
103
104    //End chrono, arreter le calcul de temps
105    gettimeofday( &tend, NULL );
106
107    //Calculer le temps ecoulé a partir de tend et tstart
108    exectime = (tend.tv_sec - tstart.tv_sec) * 1000.0; // sec to ms
109    exectime += (tend.tv_usec - tstart.tv_usec) / 1000.0; // us to ms
110
111    printf("\n\n");
112    print_matrix(A);
113    printf("\n\n\t       * \n");
114    print_matrix(B);
115    printf("\n\n\t       = \n");
116    print_matrix(C);
117    printf("\n\n");
118
119    printf("Nombre de threads: %d\t Temps d'execution :%.3lf sec\n", num_thrd, exectime/1000.0);
120
121    //libérer l'espace du tableau des IDs des threads
122    free(thread);
123
124    return 0;
125  }
```

Résultat d'exécution :

```
uir_student@ubuntu:~$ gcc -o ex1tp6  ex1tp6.c -lpthread
uir_student@ubuntu:~$ ./ex1tp6 2
Thread 0 : Calcul du morceau 0 (de la ligne 0 au 1)
Thread 0 à terminé le morceau 0
Thread 1 : Calcul du morceau 1 (de la ligne 2 au 3)
Thread 1 à terminé le morceau 1


        |    0    1    2    3 |
        |    4    5    6    7 |
        |    8    9   10   11 |
        |   12   13   14   15 |

               *

        |    0    1    2    3 |
        |    4    5    6    7 |
        |    8    9   10   11 |
        |   12   13   14   15 |

               =

        |   56   62   68   74 |
        |  152  174  196  218 |
        |  248  286  324  362 |
        |  344  398  452  506 |

Nombre de threads: 2     Temps d'execution :0.001 sec
```

b)

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

#define SIZE 4  // la taille des matrices
int num_thrd;   // nombre des threads

int A[SIZE][SIZE], B[SIZE], C[SIZE];

// initialiser une matrice
void init_matrix(int m[SIZE][SIZE])
{
  int i, j, val = 0;
  for (i = 0; i < SIZE; i++)
    for (j = 0; j < SIZE; j++)
      m[i][j] = val++;
}


// initialiser une matrice
void init_vecteur(int m[SIZE])
{
  int i, val = 0;
  for (i = 0; i < SIZE; i++)
      m[i] = val++;
}

void print_matrix(int m[SIZE][SIZE])
{
  int i, j;
  for (i = 0; i < SIZE; i++) {
    printf("\n\t| ");
    for (j = 0; j < SIZE; j++)
      printf("%2d ", m[i][j]);
    printf("|");
  }
}


void print_vecteur(int m[SIZE])
{
  int i;
  for (i = 0; i < SIZE; i++) {
    printf("\n\t| ");
      printf("%2d ", m[i]);
    printf("|");
  }
}

// la fonction de la multiplication des deux matrices
void* multiply(void* slice)
{
  int s = (int)slice;   // récupérer les informations de tranche
  int from = (s * SIZE)/num_thrd;
  int to = ((s+1) * SIZE)/num_thrd;
  int i,j,k;

  printf("computing slice %d (from row %d to %d)\n", s, from, to-1);
  for (i = from; i < to; i++)
  {
    C[i] = 0;
  }
    for (i = from; i < to; i++)
```

```
64        {
65          for ( j = 0; j < SIZE; j++)
66          C[i] += A[i][j]*B[j];
67        }
68
69      printf("finished slice %d\n", s);
70      return 0;
71    }
72
73    int main(int argc, char* argv[])
74    {
75      pthread_t* thread;  // pointeur vers un groupe de threads
76      int i;
77      struct timeval tstart, tend;
78      double exectime;
79
80      if (argc!=2)
81      {
82        printf("Usage: %s number_of_threads\n",argv[0]);
83        exit(-1);
84      }
85
86      num_thrd = atoi(argv[1]);
87      init_matrix(A);
88      init_vecteur(B);
89      thread = (pthread_t*) malloc(num_thrd*sizeof(pthread_t));
90
91       gettimeofday( &tstart, NULL );
92      // cette boucle on l'utilise pas si le numéro de thread est spécifié comme 1
93      for (i = 1; i < num_thrd; i++)
94      {
95        // créer chaque thread qui va travailler sur sa propre tranche de i
96        if (pthread_create (&thread[i], NULL, multiply, (void*)i) != 0 )
97        {
98          perror("Can't create thread");
99          free(thread);
100         exit(-1);
101       }
102     }
103
104     // le thread principal fonctionne sur la tranche 0
105     // donc tout le monde est occupé
106     // le thread principal fait tout, si le numéro de thread est spécifié comme 1
107     multiply(0);
108
109     // le thread principal en attente d'un autre thread pour terminer
110     for (i = 1; i < num_thrd; i++)
111   pthread_join (thread[i], NULL);
112      gettimeofday( &tend, NULL );
113
114   exectime = (tend.tv_sec - tstart.tv_sec) * 1000.0; // sec à ms
115     exectime += (tend.tv_usec - tstart.tv_usec) / 1000.0; // us à ms
116
117     printf("\n\n");
118     print_matrix(A);
119     printf("\n\n\t       * \n");
120     print_vecteur(B);
121     printf("\n\n\t       = \n");
122     print_vecteur(C);
123     printf("\n\n");
124
125      printf( "Number of threads: %d\tExecution time:%.3lf sec\n", num_thrd, exectime/1000.0);
126
```

```
124
125     printf( "Number of threads: %d\tExecution time:%.3lf sec\n", num_thrd, exectime/1000.0);
126
127     free(thread);
128
129     return 0;
130
131 }
```

Résultat de l'exécution :

```
uir_student@ubuntu:~/Desktop/test$ gedit multivecteur.c
uir_student@ubuntu:~/Desktop/test$ gcc -o multivecteur multivecteur.c -lpthread
uir_student@ubuntu:~/Desktop/test$ ./multivecteur
computing slice 0 (from row 0 to 1)
finished slice 0
computing slice 1 (from row 2 to 3)
finished slice 1
|   0   1   2   3 |
|   4   5   6   7 |
|   8   9  10  11 |
|  12  13  14  15 |

          *

|   0 |
|   1 |
|   2 |
|   3 |

          =

|  14 |
|  38 |
|  62 |
|  86 |
Number of threads: 2    Execution time:0.001 sec
```

Exercice2 :

a.

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

int somme = 0 ; // somme ( shared )
int N=30000; // N
int M=5; //thread number

void * processus ( void * arg)
{
  int localsum;
  int i ;
  int iproc=(int) arg;
  printf("thread number %d\n", iproc);

  int debut = iproc*N/M;
  int fin = (iproc+1)*N/M;

  for (i=debut ; i<fin; i++)
  {
      somme+=i;
  }
  return (0);
}

int main ( int argc , char *argv[] )
{
    int j;
    pthread_t tid[M];
    for(j=0;j<M;j++)
    {
        pthread_create(&tid[j],NULL,processus, (void*) j);
    }
```

```
34
35      for(j=0;j<M;j++)
36      {
37        pthread_join(tid[j],NULL);
38      }
39      printf ( "Somme ( partagee ) = %d\n" , somme ) ;
40
41      int sum=0.0;
42      for (j=0;j<N;j++)
43      {
44        sum = sum + j;
45      }
46
47      printf("Check Sum= %d\n",sum);
48      return 0 ;
49  }
```

Résultat d'exécution :

```
uir_student@ubuntu:~$ gcc -o ex62  ex62.c -lpthread
uir_student@ubuntu:~$ ./ex62
thread number 0
thread number 3
thread number 1
thread number 2
thread number 4
Somme ( partagee ) = 449985000
Check Sum= 449985000
uir_student@ubuntu:~$
```

Exercice3 :

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

#define N 5 //les philosophes
#define G ((i-1+N)%N) //les fourchettes gauches
#define libre 1
#define occupe 0

int fourchette[N] = {libre,libre,libre,libre,libre};
sem_t mutex;


void* philosophe ( void* arg )
{
/******* Completer la fonction du philosophe********/

int i =*(int*)arg;
int nb = 3;

//..../
while (nb){
  sem_wait(&mutex);
  if(fourchette[i] && fourchette[G]){ //il prendra sa fourchette et la fourchette de la

      fourchette[i]=0;
      fourchette[G]=0;
      printf("philosophe %d mange \n",i );

      sem_post(&mutex);

      nb--;

      sleep(1);

      sem_wait(&mutex);
```

```c
  sem_wait(&mutex);
  if(fourchette[i] && fourchette[G]){ //il prendra sa fourchette et la fourchette de la

      fourchette[i]=0;
      fourchette[G]=0;
      printf("philosophe %d mange \n",i );

      sem_post(&mutex);

      nb--;

      sleep(1);

      sem_wait(&mutex);

      fourchette[i]=1;
      fourchette[G]=1;

      printf("philosophe [%d] a fini de manger \n",i );

      sem_post(&mutex);

  }else{
    sem_post(&mutex);
  }
}


}


int main(int argc, char const *argv[]) {

  int NumPhi[N] = {0,1,2,3,4};
  int i;
  pthread_t ph[N];
  sem_init(&mutex,0,1);

  //ecration des N PHILOSOPHES
```

```
57
58  int main(int argc, char const *argv[]) {
59
60    int NumPhi[N] = {0,1,2,3,4};
61    int i;
62    pthread_t ph[N];
63    sem_init(&mutex,0,1);
64
65    //ecration des N PHILOSOPHES
66    for (i = 0; i < N; i++) {
67
68      pthread_create(&ph[i],NULL,philosophe,&(NumPhi[i]));
69  }
70      // à la fin des threads
71
72      i=0;
73      while (i<N && (pthread_join(ph[i++],NULL)==0)) {
74        printf("fin des threads\n");
75      }
76
77    return 0;
78  }
```

Résultat de l'execution :

```
philosophe 4 mange
philosophe 2 mange
philosophe [4] a fini de manger
philosophe 4 mange
philosophe [2] a fini de manger
philosophe 2 mange
philosophe [4] a fini de manger
philosophe 4 mange
philosophe [2] a fini de manger
philosophe 2 mange
philosophe [4] a fini de manger
philosophe 0 mange
philosophe [2] a fini de manger
philosophe 3 mange
philosophe [0] a fini de manger
philosophe 0 mange
philosophe [3] a fini de manger
philosophe 3 mange
philosophe [0] a fini de manger
philosophe 0 mange
philosophe [3] a fini de manger
philosophe 3 mange
philosophe [0] a fini de manger
fin des threads
philosophe 1 mange
philosophe [3] a fini de manger
philosophe [1] a fini de manger
philosophe 1 mange
philosophe [1] a fini de manger
```

```
philosophe [4] a fini de manger
philosophe 0 mange
philosophe [2] a fini de manger
philosophe 3 mange
philosophe [0] a fini de manger
philosophe 0 mange
philosophe [3] a fini de manger
philosophe 3 mange
philosophe [0] a fini de manger
philosophe 0 mange
philosophe [3] a fini de manger
philosophe 3 mange
philosophe [0] a fini de manger
fin des threads
philosophe 1 mange
philosophe [3] a fini de manger
philosophe [1] a fini de manger
philosophe 1 mange
philosophe [1] a fini de manger
philosophe 1 mange
philosophe [1] a fini de manger
fin des threads
fin des threads
fin des threads
fin des threads
```