**Exercice 1 (5 pts) :**

1. Quelle est la différence entre un processus lourd et un processus léger (thread) ?

Le processus lourd est créé par duplication (copie exacte du processus original) d'un processus existant. Le deux processus lourd ne partage pas le même espace mémoire. **(1 pt)**

Le thread créateur et le thread créé partagent tous deux le même espace mémoire (**1 pt)**

1. Comment est implémenté le concept de moniteur sous le langage Java. (**3pt)**

* Les données du moniteur doivent être déclarées avec le mot clé **privates** pour que seules les méthodes du moniteur accèdent à ces données,
* Les méthodes (ou procédures d'entrée) du moniteur doivent être déclarées avec le mot clé **synchronized** pour qu'elles puissent s'exécuter en exclusion mutuelle,
* La classe Object fournit les méthodes **wait**() et **notify**() pour la synchronisation des threads.

**Exercice 2** **(8 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. **(3 pts)**

Le processus 0 (le père) 2

* 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
* affiche t[0].

b- Le processus 1 (le fils) 1

* calcule le maximum du tableau dans t[0]
* affiche t[0].

2 - Y a-t-il un risque pour que les deux processus fournissent des résultats incohérents ? Expliquez. **(2pt)**

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

**3.** Réécrire le programme précédent en utilisant les threads à la place du processus lourd et la synchronisation d'accès aux sections critiques. **(3 pt)**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#define N 100

int k = 0;

float t[N];

sem\_t s;

void\* A(void\* arg){

int i;

sem\_wait(&s);

for(i=0 ; i<N ; i++)

if (t[0] < t[i])

t[0]= t[i];

printf ("Tread A, %f \n",t[0]);

sem\_post(&s);

}

void\* B(void\* arg){

int i;

sem\_wait(&s);

for(i=1;i<N;i++)

t [0]+= t[i];

printf ("Tread B, %f \n",t[0]);

sem\_post(&s);

}

int main () {

pthread\_t t1,t2;

for (int i=0;i<N;i++)

t[i] = rand ();

sem\_init(&s, 0, 1);

pthread\_create (&t1, NULL, A, NULL);

pthread\_create (&t2, NULL, B, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

return 0;

}

**Exercice 3 (7 pts):** On veut effectuer en parallèle le produit de deux matrices B et C d’ordre n (n \* n) remplis par des valeurs aléatoire et le résultat est stocké dans A.

On a :  *Pour j = 0 à n-1 A[i,j] =∑k=0,n-1 B[i,k]\*C[k,j]* ;

* Écrire le programme qui calcule A on crée 10 threads. Chaque threads se charge de calculer quelques lignes de la matrice résultat A.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <time.h>

#include <semaphore.h>

#define taille 20

int T[taille];

float A[taille][taille];

float B[taille][taille];

float C[taille][taille];

sem\_t S;

void \* produit(){

int i,j,k,ligne;

i=0;

while(i<taille){

sem\_wait(&S);

while(T[i]!=-1 & i<taille){

i++;

}

T[i]=1;

ligne=i;

sem\_post(&S);

for(j=0;j<taille;j++)

for(k=0;k<taille;k++)

A[ligne][j]+=B[ligne][k]\*C[k][j];

i++;

}

}

int main(){

pthread\_t Thread[10];

sem\_init(&S,0,1);

int i,j;

for(i=0;i<taille;i++)

for(j=0;j<taille;j++){

B[i][j]=rand()%2;//=1;

C[i][j]=rand()%2;//=1;

A[i][j]=0;

}

for(i=0;i<taille;i++)

T[i]=-1;

for(i=0;i<10;i++){

pthread\_create(&Thread[i],NULL,&produit,NULL);

}

for(i=0;i<10;i++){

pthread\_join(Thread[i],NULL);

}

for(i=0;i<taille;i++){

for(j=0;j<taille;j++)

printf("%.0f ",A[i][j]);

printf("\n");

}

return 0;

}