Département

d'Informatique

Industrielle

Enseignant responsable:

M. Imed Bennour

Fascicule de TPs

Programmation Parallèle Concurrente et Événementielle

Classe: 2ème Informatique Appliquée

2021-2022

Sommaire

Consignes aux étudiants

- **Tout étudiant doit être muni d'une flash USB et de l'énoncé du TP su papier.**

- **La préparation**

-*Vous devez IMPÉRATIVEMENT lire l'intégralité de l'énoncé AVANT de venir en séance.*

- Pour certains exercices des fragments de code sont déjà donnés, faites la saisie de ces fragments *AVANT de venir en séance.*

- Quand c'est indiqué, certains exercices doivent être faits avant de venir en séance.

- **Le travail en salle**

Les TPs objets de ce fascicule ne sont pas conçus pour être faits et achevés dans une séance de trois heures. Suivant le niveau de l'étudiant, chaque TP peut prendre entre trois et cinq heures. L'étudiant aura souvent à compléter chez lui le TP.

- **Les comptes-rendus**

- Remettre un seul fichier compressé nommé en votre nom et classe (exp. Gazi_Ahmed_IA2_TP1). Ce fichier compressé correspondra à un répertoire contenant les fichiers sources et les fichiers exécutables de tous les exercices et un fichier texte contenant les réponses aux questions..

- La remise des TPs se fait impérativement sur la plate-forme Moodle (*http://ent.uvt.rnu.tn/course*). La remise sur Moodle se bloque une semaine après la fin de la séance de TP.

- Le code d'accès à votre espace Moodle est **IA2XY** où X est le numéro de votre classe et Y le numéro de votre groupe de TP. Exp IA221 (classe 2 groupe 1).

- Tous vos codes doivent respecter :

  1) le style de codage décrit dans le fichier ci-après,
  2) toute fonction définie doit être précédée par un commentaire (deux lignes ou plus),
  3) les libellés de toutes les variables et fonctions doivent se terminer par vos initiaux. Exp. si votre nom est Ahmed Foulen alors les déclarations seront sous la forme:

```
int compteur_AF,
* pointeur_AF_;
void changer_AF();
```

TP N°1: Les bases de la programmation multithreadée (en POSIX C)

Objectifs du TP: Comprendre la structure d'un code multithread, savoir comparer la performance d'un code monothread de celle d'un code multithread et, faire la parallélisation d'une tâche.

Rappel :

-Argument de fonction type pointeur :

void M(void v) { int a, a= * (int*) v ; ….}

- Signature de pthread_create :

```
int pthread_create(pthread_t * thread, pthread_attr_t * attr, void
* (*Action) (void *) , void *arg)

int pthread_join(pthread_t thread, void **retval);
m
```

Exercice 1 : Analyse de la performance d'un code multithread

L'objectif de cet exercice est de comparer les temps CPU entre un code série (monothread) et un code parallèle (multithread). Pour cela, on vous demande de compléter le programme suivant (nommé *comparaisonSerialParallel.c*) qui exécute une tâche un certain nombre de fois. Ce programme paramétrable peut s'exécuter dans l'un des deux modes sériel ou parallèle:

```
> ./ comparaisonSerialParallel monoThread      nbrIterations

> ./ comparaisonSerialParallel multiThread   nbrThreads nbrIterations
```

Dans le mode `monothread`, le paramètre `nbrIterations` indique le nombre de traitements à faire en série. Dans le mode `multiThread` le paramètre `nbrThreads` indique le nombre de threads en parallèle : chaque thread exécute un nombre d'itérations égale `nbrIterations/nbrThreads`.

Programme *comparaisonSerialParallel* à compléter :

```
#include      //A COMPLÉTER
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int nbr_iterations,  nbr_threads;

void *threaded_thread(void *max) {
 int i;
 double result = 0.0;

 printf("Thread %ld started\n", thread_self());
```

```
 for (i = 0; i < * (int *) max; i++) {
  result = result + sin(i) * tan(i);
 }

 printf("Thread %ld done\n", phread_self());
}
//////////////
void *serial() {
 int i;
 double result = 0.0;

 for (i = 0; i < nbr_iterations; i++) {
  result = result + sin(i) * tan(i);
 }
}

////////////////////////////
void *parallel(int nbr_threads)
{
 pthread_t thread[nbr_threads];
 int rc, max=nbr_iterations/nbr_threads;
 long t;
 for (t = 0; t < nbr_threads; t++) {
  printf("Creating thread %ld\n", t);
      //A COMPLETER
  if (rc) {
   printf("ERROR: return code from pthread_create() is %d\n", rc);
   exit(-1);
  }
 }

 for (t = 0; t < nbr_threads; t++)
     pthread_join(thread[t], NULL);

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

 if (argc == 3)   {
  printf("MonoThread\n");
  nbr_iterations = atoi(argv[2]);
  serial();
  }

 else if (argc == 4)   {
  printf("MultiThread\n");
  nbr_iterations = atoi(argv[3]);
  nbr_threads = atoi(argv[2]);
  //A COMPLETER
  }
 else
```

```
    printf("Error, Usage: %s monoThread|multiThread nbr_iterations
nbr_threads\n", argv[0]);

 printf("Main completed\n");
}
```

a) Compléter et compiler le programme *comparaisonSerialParallel .c*:

```
> gcc comparaisonSerialParallel.c -pthread -lm
```

b) Quel est le nombre de processeurs et le nombre de cœurs de votre machine? (Commande : lscpu)

c) Lancez les exécutions suivantes:

```
> time ./ comparaisonSerialParallel serial     100000000

> time ./ comparaisonSerialParallel parallel 2   100000000

 > time ./ comparaisonSerialParallel parallel 4   100000000

 > time ./ comparaisonSerialParallel parallel 5   100000000

 > time ./ comparaisonSerialParallel parallel 8  100000000
```

Remplir le tableau suivant en précisant: le temps CPU (user + system), le facteur d'accélération entre le mode série et le mode parallèle et le nombre de cœurs exploités par le programme.

	Temps réel	Temps CPU	Facteur d'accélération	Nombre de cœurs exploités
`serial 100000000`			0	
`parallel 2 100000000`				
`parallel 4 100000000`				
`parallel 5 100000000`				
`parallel 8 100000000`				

Commenter ces résultats.

d) Commande : ksysguard

Exercice 2 : Analyse de la concurrence entre threads

a) En modifiant le programme précédent, produire un programme monothread et multithread (nommé a*ffichageMultithread.c*) qui affiche les nombres entiers de 1 à une certaine constante `Iterations`.

Dans le cas d'une exécution multithread, est-ce que les entiers affichés sont ordonnés?
b) Modifier le programme de sorte que toutes les threads incrémentent une même variable globale (notée vg). Vérifier que la valeur finale de vg n'est pas toujours celle attendue.

Exercice 3 : Parallélisation d'une tâche de recherche dans un tableau

L'objectif de cet exercice est de concevoir un algorithme concurrent implémenté par un programme multithread pour rechercher un élément dans un vecteur non trié. On supposera que les éléments sont de type entier, mais le principe doit s'appliquer à tous types de données.

a) Écrire la version séquentielle du programme : une première fonction **initialise** le vecteur avec des éléments dont les valeurs sont lues à partir de l'entrée standard; la fonction **search** recherche la valeur **x** dans le vecteur **T**. Le vecteur **T** a une taille **n**. Cette fonction parcourt séquentiellement le vecteur; le programme affiche ensuite si la valeur a été trouvée dans le vecteur.

b) Écrire la version multithreadée du programme (nommé searchMultithread.c). Dans la fonction **search**: chaque thread recherchera la valeur dans une portion du vecteur qui lui aura été affectée. Un thread doit pouvoir arrêter sa recherche si un autre thread a trouvé la valeur x recherchée dans le vecteur **T**.

Votre programme doit avoir trois paramètres : nbrThreads (le nombre de threads), nbrElements (taille du tableau) et valeur (la valeur recherchée).

TP N°2: La programmation concurrente

Objectifs du TP : Exploiter les mécanismes de synchronisation pour écrire des programmes mutithread-concurrents.

Rappel : Faites la saisie des fragments de code avant l'arrivé en classe.

Éléments de syntaxe du mutex et du sémaphore:

// Mutex

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mut);
int pthread_mutex_trylock(pthread_mutex_t *mut);
int pthread_mutex_unlock(pthread_mutex_t *mut);

pthread_mutex_t mut = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_lock (&mut);
pthread_mutex_unlock (&mut);
```

// Sémaphore

```
#include <semaphore.h>
int  sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);

sem_t sem;
sem_init(&sem, 0, 5);
sem_wait(&sem);
sem_post(&sem);
sem_destroy(&sem);
```

Exercice 1 : Mutexs/sémaphores pour partage de ressources

Un pont supporte une charge maximale de 15 tonnes. Ce pont est traversé par des camions dont le poids est de 15 tonnes ainsi que par des voitures dont le poids est de 5 tonnes. On vous demande de gérer l'accès au pont de sorte que la charge maximale du pont soit respectée.

On suppose que le nombre de voitures (camions) est une donnée constante notée NB_VOITURES (NB_CAMIONS). Les camions et les voitures seront modélisés par des threads.

a) Donner l'algorithme puis le programme comportant une fonction d'acquisition du pont: `void acceder_au_pont (int tonnes)`

et une fonction de libération du pont : `void quitter_le_pont(int tonnes)`

qui simulent les règles de partage du pont ci-dessus.

Cette première solution doit utiliser un sémaphore (`sem_compteur`) pour compter le poids de véhicules en cours de transit sur le pont. Lorsque ce sémaphore passe a zéro, c'est que le pont est plein et les threads sont bloqués. Un mutex (`comion_sc`) doit aussi être utilisé afin d'éviter un interblocage causé par des camions faisant des accès simultanés au sémaphore.

Le main, le thread camion et le thread voiture auront un code de cette forme:

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<semaphore.h>
#include<time.h>
…
…
pthread_t pthread_id[NB_VEHICULES];

int main(int argc, char* argv[]) {
   int i;
   pthread_t id;
   sem_init(&sem_compteur,0,3);
   for(i=0; i<NB_VEHICULES;i++){
      int*j=(int*)malloc(sizeof(int));
      *j=i;
      if (i<NB_CAMIONS)
            pthread_create(&pthread_id[i],NULL,camion,j);
      else   pthread_create(&pthread_id[i],NULL,voiture,j);
   }

  for(i = 0; i < NB_VEHICULES; i++)
  {
     pthread_join( pthread_id[i], NULL);
  }
   pthread_exit(NULL);
};
```

```
void* camion(void* arg)
{
  int pid=*((int*)arg);
 intialise_generateur_aleatoir() ;

   attendre(5);
   acceder_au_pont(15);
   printf("Camion %d : je traverse le
pont\n",pid);
```

```
void* voiture(void* arg)
{
   int pid=*((int*)arg);
     intialise_generateur_aleatoir() ;

   attendre(5);
   acceder_au_pont(5);
   printf("Voiture %d :
je traverse le pont\n",pid);
```

```
   attendre(5);
   printf("Camion %d : je quitte le
pont\n",pid);
   quitter_le_pont(15);
   pthread_exit(NULL);
}
```

```
   attendre(5;
   printf("Voiture %d :
 je quitte
 le pont\n",pid);
   quitter_le_pont(5);
   pthread_exit(NULL);
}
```

```
void attendre(in t){
sleep(valeur_aleatoire(1,t));
}
```

```
int valeur_aleatoire(int min, int max)
{
  int val;
  float f = random() * (max-min);
  f=(f / RAND_MAX) +min;
  return (f + 0.5);
}
```

```
void initialise_generateur_aleatoire()
{
  srandom(time(NULL) + getpid());
}
```

b) On souhaite améliorer la solution précédente de sorte que si un camion est en attente du 2ème ou du 3ème jetons alors il devient prioritaire sur les voitures.

c) On souhaite améliorer la solution précédente en donnant toujours la priorité aux camions: lorsqu'une voiture et un camion sont bloqués en attente d'obtenir l'accès au pont, le camion doit être réveillé en premier, sous réserve, bien sûr, que la capacité maximale du pont soit respectée. Avec cette nouvelle contrainte, il n'est plus possible d'utiliser la méthode basée sur le sémaphore commun; on utilisera alors un sémaphore privé à chaque thread.

Donner une 2ème version de la fonction d'acquisition du pont:

```
void acceder_au_pont (int tonnes, int id)
```

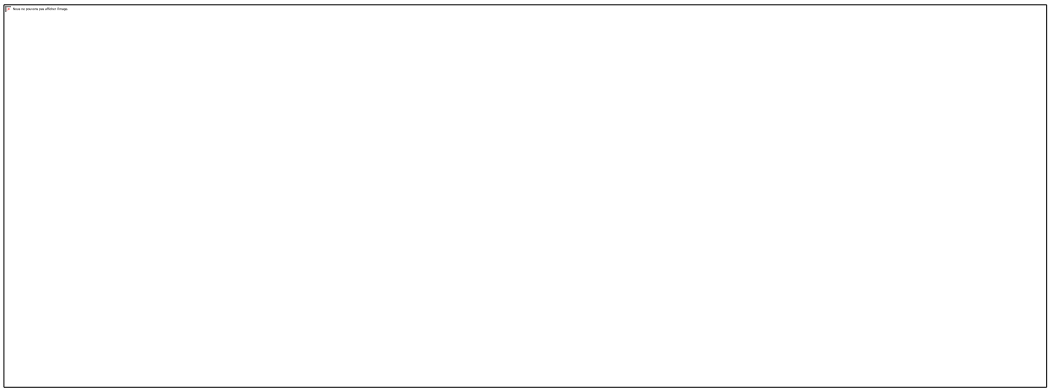
et de la fonction de libération du pont:

```
void quitter_le_pont(int tonnes, in id)
```

de sorte qu'elles prennent en compte la priorité des camions; id étant le numéro du véhicule.

Exercice 2 : Simulateur de course (Sémaphores, variables condition)

Nous nous proposons de simuler une course de véhicules sur le circuit dessiné ci-dessous. Les véhicules démarrent successivement à partir du point O et roulent dans le sens indiqué par les flèches. Le circuit est composé de 6 étapes: OA, AB, BO, OC, CD et DO. Chaque étape peut être réalisée en un temps t appartenant à un intervalle [Tmin Tmax] où Tmin et Tmax sont les temps réalisés, respectivement, par les voitures les plus rapides et les voitures les plus lentes. Les intervalles de temps pour chaque étape sont résumés dans le tableau à droite.



a) En représentant chaque véhicule par un thread, compléter le programme en C ci-dessous qui doit créer 3 véhicules au niveau du point O et leur faire effectuer 2 tours de circuit. Pour déterminer le temps mis par un véhicule pour effectuer une étape, on prendra une valeur aléatoire entière *ttrajet* dans l'intervalle de temps correspondant au trajet. Pour simuler le trajet, on endormira le processus pendant *ttrajet* secondes. Pour cela, vous pourrez utiliser la fonction *sleep (ttrajet).* Le programme devra afficher pour chaque véhicule atteignant le point O son numéro de thread, le nombre de tours effectués et le temps mis, depuis le début de la course, pour l'atteindre. En outre, il affichera le départ et la fin de course pour chaque voiture. Voici un exemple d'affichage:

Départ de la voiture 26215
Départ de la voiture 26216
Départ de la voiture 26217
Voiture 26217, tour numéro 1, temps jusqu'au point O: 7 sec
Voiture 26216, tour numéro 1, temps jusqu'au point O: 8 sec
Voiture 26215, tour numéro 1, temps jusqu'au point O: 12 sec
Voiture 26216, tour numéro 2, temps jusqu'au point O: 16 sec
Arrivée de la voiture 26216
Voiture 26217, tour numéro 2, temps jusqu'au point O: 18 sec
Arrivée de la voiture 26217
Voiture 26215, tour numéro 2, temps jusqu'au point O: 21 sec
Arrivée de la voiture 26215

Programme simulateur.c à compléter :

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/syscall.h>
#include<sys/types.h>
#include<linux/unistd.h>
#define NB_VOITURES 3
#define NB_TOURS 2
pthread_t pthread_id[NB_VOITURES];
int gettid() {
  return syscall(SYS_gettid);
}
/*** initalise le generateur aleatoire.*/
void initialise_generateur_aleatoire()
{
  srandom(time(NULL) + getpid());
}

/*** Retourne une valeur aleatoire entre
* min (inclus) et max (inclus).*/
int valeur_aleatoire(int min, int max)
{
  int val;
  float f = random() * (max-min);
  f=(f / RAND_MAX) +min;
  return (f + 0.5);
}
/*** Procedure faisant effectuer les tours de pistes
* a une voiture.*/
void * threadVoiture ( void * nbTour ) {
/* VARIABLES A DEFINIR */
 initialise_generateur_aleatoire () ;
/* A COMPLETER */ }
 }
int main()
{
  int i, nb_tours;
  nb_tours = NB_TOURS;
  for(i = 0; i < NB_VOITURES; i++)
  {
       if( pthread_create( &pthread_id[i], NULL, threadVoiture,
&nb_tours) == -1)
       {
        fprintf(stderr, " Erreur de creation du pthread
numero %d", i);
       }
  }
  for(i = 0; i < NB_VOITURES; i++)
  {
       pthread_join( pthread_id[i], NULL);
```

```
  }
        return EXIT_SUCCESS;

}
```

b) Que risque-t-il de se passer au point O?

c) Nous décidons d'ajouter deux feux bicolores au point O pour gérer la circulation au niveau du carrefour. Le premier feu régule le passage sur la voie DA. Le second feu régule le passage sur la voie BC. Chaque feu est alternativement soit vert soit rouge. Le feu sera modélisé par une variable globale de type entier feu qui vaudra 0 si le feu est vert sur la voie DA (donc rouge sur la voie BC), et 1 si le feu est vert sur la voie BC (donc rouge sur la voie DA). Chaque feu change de couleur toutes les 2 secondes. Le feu étant un objet indépendant, il sera représenté par un thread. Modifier le programme précédent tel que la variable feu est testée à chaque passage DA ou BC: si le feu est rouge, le thread Voiture attend 1 seconde, et puis réessaie.

d) Donner une autre solution à base de variable de condition et qui ne fait pas recours à l'attente systématique d'une seconde par les voitures.

e) Ajouter des instructions avec code permettant de vérifier que les voitures n'entrent pas en collision au point O.

TP N°3: Programmation réseau avec les sockets

Exercice 1

Exécuter le code monoclient-serveur donnée en annexe 1.

Quel est le numéro du port du serveur ?

Quel est le numéro du port du client ?

Combien de sockets sont crées par le client et par le serveur respectivement ?

Changer le numéro du port serveur à 8082 et retester votre code.

Exercice 2

Exécuter le code multiclient-serveur donnée en annexe 2.

Combien de sockets sont crées par le client et par le serveur respectivement ?

Exercice 3

Écrire une version client-serveur du simulateur de course (simulateur_serveur.c, simulateur_client.c): A chaque voiture serait associé un programme distinct de type client. Les feux sont régulés au niveau du serveur, ainsi que l'affichage des résultats.

Annexe 1

///Client

```
#include<string.h>  //strlen
#include<sys/socket.h>      //socket
#include<arpa/inet.h>       //inet_addr

int main(int argc , char *argv[])
{
        int sock;
        struct sockaddr_in server;
        char message[1000] , server_reply[2000];

        //Create socket
        sock = socket(AF_INET , SOCK_STREAM , 0);
        if (sock == -1)
        {
                printf("Could not create socket");
        }
        puts("Socket created");

        server.sin_addr.s_addr = inet_addr("127.0.0.1");
        server.sin_family = AF_INET;
        server.sin_port = htons( 8888 );

        //Connect to remote server
        if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
        {
                perror("connect failed. Error");
                return 1;
        }

        puts("Connected\n");

        //keep communicating with server
        while(1)
        {
                printf("Enter message : ");
                scanf("%s" , message);

                //Send some data
                if( send(sock , message , strlen(message) , 0) < 0)
                {
                        puts("Send failed");
                        return 1;
                }

                //Receive a reply from the server
                if( recv(sock , server_reply , 2000 , 0) < 0)
                {
```

```
                puts("recv failed");
                break;
            }

            puts("Server reply :");
            puts(server_reply);
        }

        close(sock);
        return 0;
}
```

////////**Serveur**

```
#include<stdio.h>
#include<string.h>  //strlen
#include<sys/socket.h> //socket libery
#include<arpa/inet.h>        //inet_addr
#include<unistd.h>  //write

int main(int argc , char *argv[])
{
        int socket_desc , client_sock , c , read_size;
        struct sockaddr_in server , client;
        char client_message[2000],message[2000];


        //Create socket
        socket_desc = socket(AF_INET , SOCK_STREAM , 0);
        if (socket_desc == -1)
        {
                printf("Could not create socket");
        }
        puts("Socket created");

        //Prepare the sockaddr_in structure
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = INADDR_ANY;
        server.sin_port = htons( 8888 );

        //Bind
        if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0) //bind new
concxion ma3a client
        {

                perror("Error mfcmh cnx");
                return 1;
        }
        puts("bind done");

        //ytsanet 3al resaux lin yconnctewa7ed
        listen(socket_desc , 3);

        //Accept  incoming connection
        puts("Waiting  connections...");
        c = sizeof(struct sockaddr_in);

        //accept connection from  client
        client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
///accepte le demande
        if (client_sock < 0)
        {
                perror(" failed mfch client ");
```

```
            return 1;
        }
        puts("Connection accepted ");

        //Receive a message from client
        while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
        {
            //Send the message back to client
            puts( client_message);
            printf("Enter message ");
            scanf("%s" , message);
            //printf ("%s",client_message);
            write(client_sock ,message  , strlen(client_message));
        }

        if(read_size == 0)
        {
            puts("Client disconnected ");
            fflush(stdout);
        }
        else if(read_size == -1)
        {
            perror("recving failed");
        }

        return 0;
}
```

Annexe 2.

///////Serveur multiclient

```
#include<stdio.h>
#include<string.h>  //strlen
#include<stdlib.h>  //strlen
#include<sys/socket.h>
#include<arpa/inet.h>       //inet_addr
#include<unistd.h>  //write
#include<pthread.h> //for threading , link with lpthread

//the thread function
void *connection_handler(void *);

int main(int argc , char *argv[])
{
        int socket_desc , client_sock , c , *new_sock;
        struct sockaddr_in server , client;
        int nbrConnection =0;

        //Create socket
        socket_desc = socket(AF_INET , SOCK_STREAM , 0);
        if (socket_desc == -1)
        {
                printf("Could not create socket");
        }
        puts("Socket created");

        //Prepare the sockaddr_in structure
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = INADDR_ANY;
        server.sin_port = htons( 8888 );

        //Bind
        if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
        {
                //print the error message
                perror("bind failed. Error");
                return 1;
        }
        puts("bind done");

        //Listen
        listen(socket_desc , 3);

        //Accept and incoming connection
        puts("Waiting for incoming connections...");
        c = sizeof(struct sockaddr_in);
```

```
	//Accept and incoming connection
	puts("Waiting for incoming connections...");
	c = sizeof(struct sockaddr_in);
	while( (client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c)) )
	{
		nbrConnection++	;
		puts("New Connection accepted");
		printf("Le nombre de connections en cours est %d \n", nbrConnection);
		pthread_t sniffer_thread;
		new_sock = malloc(1);
		*new_sock = client_sock;

		if( pthread_create( &sniffer_thread , NULL ,  connection_handler , (void*) new_sock) < 0)
		{
			perror("could not create thread");
			return 1;
		}

		//Now join the thread , so that we dont terminate before the thread
		//pthread_join( sniffer_thread , NULL);
		puts("Handler assigned");
	}


	if (client_sock < 0)
	{
		perror("accept failed");
		return 1;
	}

	return 0;
}

/*
 * This will handle connection for each client
 * */
void *connection_handler(void *socket_desc)
{
	//Get the socket descriptor
	int sock = *(int*)socket_desc;
	int read_size;
	char *message , client_message[2000];

	//Send some messages to the client
	message = "Greetings! I am your connection handler\n";
	write(sock , message , strlen(message));

	message = "Now type something and i shall repeat what you type \n";
	write(sock , message , strlen(message));
```

```
	//Receive a message from client
	while( (read_size = recv(sock , client_message , 2000 , 0)) > 0 )
	{
		puts(client_message);
		//Send the message back to client
		write(sock , client_message , strlen(client_message));
	}

	if(read_size == 0)
	{
		puts("Client disconnected");
		fflush(stdout);
	}
	else if(read_size == -1)
	{
		perror("recv failed");
	}

	//Free the socket pointer
	free(socket_desc);

	return 0;
}
```

TP N°4: La

programmation événementielle (sous Swing Java)

Objectif du TP: Pratiquer la programmation multithread événementielle et analyser un code événementiel

Exercice 1 : GUI avec Swing

a) Dessiner le diagramme de classes du code de BoucingBall.java donné en annexe.

- Lister les attributs et les fonctions de chaque classe.

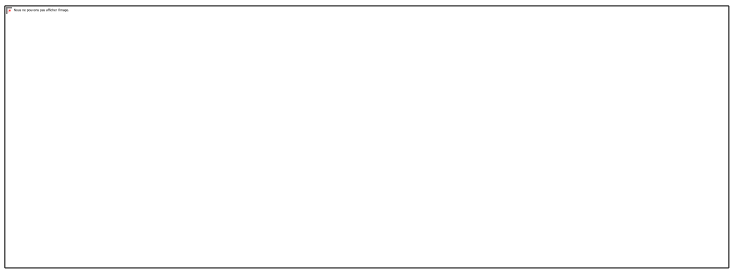
- Quels sont les objets créés?

- Combien de Jframe sont créés?

- Combien de Jpanel sont créés?

- Quels sont les Composants auxquels sont attachés des Listener?

b) Exécuter ce code.



Faites augmenter la vitesse du ballon et simuler de nouveau.

c) Ajouter un deuxième ballon avec ses propres boutons *start/stop.*

d) Ajouter le code nécessaire qui permet d'arrêter un ballon lorsqu'on le clique avec la souris.

Exercice 2 : GUI et multithread

a) Copier le code BoucingBall.java dans BoucingBallMultiThread.java.

b) On souhaite ajouter le bloc suivant au programme BoucingBallBis.java. L'exécution de ce code doit être sous le contrôle du bouton *lancer-processus.*

```
          System.out.println("Début traitement long");

           float s=0;

           for (int i = 0; i < 1000000; i++) {

                s+=Math.sin(i);

     System.out.println(" en cours");

   }

          System.out.println(" Fin traitement long");
```

Ajouter le bloc, simuler et noter le comportement. Est-ce que l'application fonctionne d'une façon performante? Expliquer?

c) Mettre le bloc sous la forme d'un thread. Puis simuler et noter le comportement. Est ce que l'application fonctionne d'une façon performante?

d) Mettre le bloc sous la forme d'un SwingWork Thread.

Est-ce que l'application fonctionne maintenant d'une façon performante?

```
package bouncingball;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Graphics;
import java.awt.event.ActionListener;
import javax.swing.*;
import java.awt.event.*;

//////////////////////////////////////////////////////////
public class Bouncingball {
   public static void main(String[] args) {
      JFrame win = new JFrame("Bouncing Ball Demo");
      win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      win.setContentPane(new BBPanel());
      win.pack();
      win.setVisible(true);
   }
}//endclass BBDemo

class BBPanel extends JPanel {

   BallInBox m_bb;
   // The bouncing ball panel

   //================================================== constructor
   /**
    * Creates a panel with the controls and bouncing ball display.
    */
   BBPanel() {

      //... Create components
      m_bb = new BallInBox();

      // theBallBox= m_bb;
      JButton startButton = new JButton("Start");

      JButton stopButton = new JButton("Stop");

      //... Add Listeners
      startButton.addActionListener(new StartAction());

      stopButton.addActionListener(new StopAction());

      //... Layout inner panel with two buttons horizontally
      JPanel buttonPanel = new JPanel();

      buttonPanel.setLayout(new FlowLayout());
```

```
        buttonPanel.add(startButton);
        buttonPanel.add(stopButton);

        //... Layout outer panel with button panel above bouncing ball
        this.setLayout(new BorderLayout());

        this.add(buttonPanel, BorderLayout.NORTH);

        this.add(m_bb, BorderLayout.CENTER);

    }//end constructor

////////////////////////////////// inner listener class StartAction
    class StartAction implements ActionListener {

        public void actionPerformed(ActionEvent e) {
            m_bb.setAnimation(true);

        }

    }

    ///////////////// inner listener class StopAction
    class StopAction implements ActionListener {

        public void actionPerformed(ActionEvent e) {

            m_bb.setAnimation(false);

        }

    }
}//endclass BBPanel
////////////////////////////////// BouncingBall

class BallInBox extends JPanel {

    //================================================ fields
    //... Instance variables representing the ball.
    private Ball m_ball = new Ball(0, 0, 1, 1); //... Instance variables for the animiation

    private int m_interval = 35;  // Milliseconds between updates.

    private Timer m_timer;  // Timer fires to anmimate one step.

    //================================================constructor
    /**
     * Set panel size and creates timer.
     */
```

```
    public BallInBox() {

        setPreferredSize(new Dimension(400, 120));
        setBorder(BorderFactory.createLineBorder(Color.BLACK));

        m_timer = new Timer(m_interval, new TimerAction());
    }

    public void actionPerformed(ActionEvent ae) {
        this.setAnimation(false);
        System.out.println("a click");
    }

    //================================================ setAnimation
    /**
     * Turn animation on or off.
     *
     * @param turnOnOff Specifies state of animation.
     *
     */
    public void setAnimation(boolean turnOnOff) {

        if (turnOnOff) {
            m_timer.start();
            // start animation by starting the timer.
        } else {

          m_timer.stop();
// stop timer
        }
    }

//==================== paintComponent
    public void paintComponent(Graphics g) {
        super.paintComponent(g);  // Paint background, border
        m_ball.draw(g);  // Draw the ball.
    }

////////////////////////////////   inner   listener   class  ActionListener
    class TimerAction implements ActionListener {

//============================================== actionPerformed
        /**
         * ActionListener of the timer. Each time this is called,
          * the ball's position is updated, creating the appearance of
             * movement. .
         *
         * @param e This ActionEvent parameter is unused.
         *
         */
```

```
        public void actionPerformed(ActionEvent e) {
            m_ball.setBounds(getWidth(), getHeight());
            m_ball.move();  // Move the ball.
            repaint();    //   Repaint   indirectly   calls paintComponent.
        }
    }
}//endclass

////////////////////////////////// BallModel
class Ball {
//... Constants
    final static int DIAMETER = 21; //... Instance variables
    int m_x;  // x and y coordinates upper left
    int m_y;
    private int m_velocityX;  //Pixels to move each time move()  is called.
    private int m_velocityY;
    private int m_rightBound;  // Maximum permissible x, y values.
    private int m_bottomBound;

//=========================================== constructor
    public Ball(int x, int y, int velocityX, int velocityY) {
        m_x = x;
        m_y = y;
        m_velocityX = velocityX;
        m_velocityY = velocityY;
    }

//====================================== setBounds
    public void setBounds(int width, int height) {
        m_rightBound              = width              - DIAMETER;
        m_bottomBound = height - DIAMETER;

    }

//================================================ move
    public void move() {

//... Move the ball at the give velocity.
        m_x += m_velocityX;
        m_y += m_velocityY;  //... Bounce the ball off the walls if necessary.
        if (m_x < 0) {  // If at or beyond left side
            m_x = 0;  // Place against edge and
            m_velocityX = -m_velocityX; // reverse direction.
        } else if (m_x > m_rightBound) { // If at or beyond right side
            m_x = m_rightBound;  // Place against right edge.
            m_velocityX = -m_velocityX;  // Reverse direction.
        }
        if (m_y < 0) {  // if we're at top
            m_y = 0;
            m_velocityY = -m_velocityY;
```

```
        } else if (m_y > m_bottomBound) { // if we're at bottom
            m_y = m_bottomBound;
            m_velocityY = -m_velocityY;

        }
    }

//=============================================================
draw
    public void draw(Graphics g) {
        g.fillOval(m_x, m_y, DIAMETER, DIAMETER);
    }

//=====================================   getDiameter, getX, getY
    public int
            getDiameter() {
        return DIAMETER;
    }
   public int getX() {
        return m_x;
    }
    public int getY() {
        return m_y;
    }

//============================== setPosition
    public void setPosition(int x, int y) {
        m_x = x;
        m_y = y;
    }

}
```

TP **N°5** Initiation à la programmation parallèle sous OPEN-MP et sous CUDA

Objectifs du TP: Écriture un code parallèle basique sous OPEM-MP et sous CUDA

Exercice 1 : Parallélisation d'une tâche de recherche dans un tableau avec OPEN-MP

- Dans le TP1 vous avez écrit un programme multithreadé en POSIX faisant la recherche d'un élément dans un tableau. On vous demande de récrire le même programme en OPEN-MP.

S’assurer qu'un thread doit pouvoir arrêter sa recherche dés qu’un autre thread trouve la valeur x recherchée dans le vecteur T.

Exercice 2 : Parallélisation avec CUDA

Écrire un code parallèle sous CUDA faisant le produit de deux matrices**.**