Réseaux. TP 2. Sockets en C et en Java

I. Sockets en C

I.1. Communication client/serveur par sockets connectées en C

Voici le code d'une socket cliente et serveur qui communiquent en mode connecté. Le client envoie un message vers le serveur puis attend la réponse de ce dernier pour l'afficher. Le serveur convertit les messages qu'il reçoit en majuscules. Compléter le code ou les commentaires aux endroits indiqués **en gras**

I.1.1. Socket cliente en mode connecté

```
/*
SOCKET CLIENTE CONNECTEE

Nombre de paramètres : 5
1. Numéro de port d'écoute local de la socket
2. nom du serveur
3. Numéro de port d'écoute du serveur
4. Message à envoyer vers le serveur
5. taille du message

La Socket est créée localement sur le port d'écoute indiqué par le paramètre 1.  puis se connecte au serveur dont le nom est donné par le paramètre 2. et le numéro de port par le paramètre 3. pour envoyer le message dont l'emplacement est donné par le paramètre 4. et la taille par le paramètre 5.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>

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

   /* 1. Récupération des paramètres d'entrée */

      short portLoc = (short) atoi(argv[1]) ;                    /* numéro de port local */
      char *nomServ;
      nomServ = argv[2];                    /* Pointeur sur le nom du serveur */
      short portServ  = (short) atoi(argv[3]) ;                  /* numéro de port serveur */
      char *msgEnvoi;
      msgEnvoi = argv[4] ;                  /* Pointeur sur le message à envoyer */
      int  tailleMsgEnvoi= atoi(argv[5]) ;              /* taille du message */

      /* 2. Déclaration des variables permettant la communication sur le réseau */

      int sockLoc;     // descripteur de la socket locale (cliente)
      struct hostent* infosServ;              /* Informations du serveur */
      struct sockaddr_in infosAddrLoc;    /* Informations d'adressage de la socket locale */
      struct sockaddr_in infosAddrServ;               /* Informations d'adressage de la socket serveur */
      int tailleInfosAddrLoc, tailleInfosAddrServ ;    /* taille des informations d'adressage de la socket locale */

      char msgRecu [1000] ;    /* Buffer qui contiendra la réponse du serveur */

      /* 3. Création de la socket en mode connecté, si création réussie retourne son descripteur dans sockLoc, sinon on quitte le programme */
      ……..
```

```
        ……..
        ……..

        /* 4. Mise à jour des informations d'adressage locales */

        /* 4.1. Mise à jour du domaine de la socket locale */
        ……..

        /* 4.2. Mise à jour du numéro de port local après avoir fait la conversion en format réseau */
        ……..

        infosAddrLoc.sin_addr.s_addr = htonl(INADDR_ANY);  /* 4.3. Que fait-on ici  ? */
        /* …….. */

        /* 4.4. Mise à jour de tailleInfosAddrLoc */
        tailleInfosAddrLoc = sizeof(infosAddrLoc);

        /* 5. Attachement de la socket locale, si erreur on quitte le programme */
        ……..
        ……..
        ……..

        /* 6. Récupère les informations du serveur à partir de son nom */
        infosServ = gethostbyname(nomServ);

        /* 7. Mise à jour des informations d'adressage du serveur */

        infosAddrServ.sin_family = infosServ->h_addrtype;       /* Mise à jour du domaine de la socket serveur */
        infosAddrServ.sin_port = htons(portServ);           /* Mise à jour du numéro de port serveur */
        memcpy(&infosAddrServ.sin_addr, infosServ->h_addr, infosServ->h_length);            /*Mise à jour de l'adresse IP du serveur*/

        printf("Adresse du serveur: %s \n", inet_ntoa(infosAddrServ.sin_addr));

        tailleInfosAddrServ = sizeof(infosAddrServ);  /* Mise à jour de tailleInfosAddrServ */

        /* 8. Connexion au serveur, si problème de connexion on quitte le programme */
        ……..
        ……..
        ……..

        /* 9. Envoi d'un message au serveur */
        ……..

        /* 10. Récupère la réponse du serveur et l'affiche */
        ……..
        ……..

        /* 11. Fermeture de la socket */
        close(sockLoc) ;
}
```

I.1.2. Socket serveur en mode connecté

```
/*
SOCKET SERVEUR CONNECTEE EN MODE ITERATIF

Nombre de paramètres : 1
1. Numéro de port d'écoute local de la socket

La Socket est créée localement sur le port d'écoute indiqué par le paramètre 1.  Elle attend les demandes de connexion des sockets clientes et les traite.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```
#include <unistd.h>
#include <netdb.h>
#include <ctype.h> // pour la fonction toupper

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

	/* 1. Récupération des paramètres d'entrée */

	short portLoc = (short) atoi(argv[1]);		/* numéro de port local (serveur) */

	/* 2. Déclaration des variables permettant la communication sur le réseau */

	int sockLoc, sockLocService ;
	struct sockaddr_in infosAddrLoc;	/* Informations d'adressage de la socket locale (serveur) */
	struct sockaddr_in infosAddrCl;	/* Informations d'adressage de la socket cliente */
	int tailleInfosAddrLoc, tailleInfosAddrCl;	/* Taille des informations d'adressage de la socket locale (serveur) et cliente */
	char msgRecu [1000];	/* Buffer de réception (contiendra le message reçu du client) */

	/* 3. Création de la socket en mode connecté, si création réussie retourne son descripteur dans sockLoc, sinon quitte le
	programme */

	if ((sockLoc = socket(AF_INET, SOCK_STREAM, 0))<0){
		perror("Socket");
		exit(1);
	}

	/* 4. Mise à jour des informations d'adressage locales */

	infosAddrLoc.sin_family = AF_INET;		/* Type de la socket (TCP/IP) */
	infosAddrLoc.sin_port = htons (portLoc);		/* Affectation du port local */
	infosAddrLoc.sin_addr.s_addr = htonl(INADDR_ANY);		/* Identificateur de l'hote */

	tailleInfosAddrLoc = sizeof(infosAddrLoc);		/* Mise à jour de tailleInfosAddrLoc */

	/* 5. Attachement de la socket locale, si erreur on quitte le programme */

	if (bind(sockLoc, (struct sockaddr*) &infosAddrLoc,  tailleInfosAddrLoc)<0){
		perror("Bind");
		exit(1);
	}

	/* 6. Attente de connexion de la part de Socket clientes (maximum 5), -1 si erreur */
	……..
	……..
	……..

	while (1){
		/* 7.  Acceptation de la socket, le descripteur de la socket de service est mis dans  sockLocService */
		……..
		……..
		……..

		/* 8. Récupère le message du client */
		……..

		/*  9. Convertit le message reçu en majuscules  */
		……..
		……..

		/* 10. Renvoi du message  converti en majuscule au client */
		……..

		/* 11. Réinitialisation du buffer de réception */
		memset (msgRecu, 0, sizeof (msgRecu));

		/* 12. Fermeture de la socket de service  */
		close(sockLocService);
	}
}
```

I.2. Serveur en mode concurrent

Modifier le corps de la boucle while (while(1){...}) de la socket serveur précédente pour qu'elle traite maintenant les demandes des clients de manière concurrente.

I.3. Sockets déconnectées

Modifier le code des sockets cliente et serveur de l'exercice I. pour qu'elles communiquent maintenant en mode déconnecté (UDP).

I.4. Client multiprocessus en itération (potentiellement) infinie

On veut remplacer les deux appels sendto(...) et recvfrom(...) du client en mode déconnecté par deux boucles infinies qui tournent en (pseudo-)parallèle. Dans l'une, on envoi vers le serveur les messages saisis par l'utilisateur jusqu'à ce que celui-ci saisisse la chaîne vide, dans l'autre on récupère et on affiche les réponses du serveur. Comment procéder ? Remarque : les deux parties du programme doivent s'arrêter quand l'utilisateur saisit la chaîne vide.

II. Sockets en Java

II.1. Communication par sockets connectées en Java

Voici le code de deux sockets connectées qui communiquent. Commenter ce code. L'algorithme de la partie serveur vous paraît il bien conçu ? Pourquoi ? Comment pourrait-on améliorer ça ? Ecrire le code correspondant.

II.1.1. Socket cliente connectée

```
import java.net.*;
import java.io.*;
import java.util.*;

public class SocketConnecteeCliente {

	static String adresseIPServeur="localhost";
	static int numeroPortServeur=1234;

	public static void main(String[] args) {
		try {

			Socket socketCliente = new Socket();

			/* Connexion à la socket serveur */
			socketCliente.connect(new InetSocketAddress(adresseIPServeur, numeroPortServeur));

			while(true){
				Scanner keyboard = new Scanner(System.in);
				System.out.print ("Votre message (finir par -1) : ");
				String message=keyboard.nextLine();

				BufferedReader in = new BufferedReader(new InputStreamReader(socketCliente.getInputStream()));
				PrintStream out = new PrintStream(socketCliente.getOutputStream());

				out.println(message);

				if (message.equals("-1"))
					break;
```

```
                System.out.println("Votre message en majuscule : "+ in.readLine());

            }
            socketCliente.close();

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

II.1.2. Socket serveur connectée

```
import java.net.*;
import java.io.*;

public class SocketConnecteeServeurIteratif {

    static int numeroPortLocal=1234;
    static int delaiTimeOut=10000;

    public static void main(String[] args) {
        try {
            ServerSocket socketServeur = new ServerSocket(numeroPortLocal);

            while (true) {

                Socket socketService = socketServeur.accept();
                System.out.println("Connexion de la part de : "+(socketService.getInetAddress()).getHostAddress());

                while(true){

                    BufferedReader in = new BufferedReader(new InputStreamReader(socketService.getInputStream()));
                    String message = in.readLine();

                    if (message.equals("-1"))
                        break;

                    message=message.toUpperCase();

                    PrintStream out = new PrintStream(socketService.getOutputStream());
                    out.println(message);
                }

                socketService.close();

            }

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

II.2. Communication par sockets déconnectées en mode multicast en Java

Les deux parties de code suivantes montrent une diffusion de groupe (multicast). D'un côté, une socket publieur (émettrice) émet un message vers un groupe de diffusion (adresse multicast 234.59.0.2), de l'autre côté, une socket abonnée (réceptrice) rejoint ce groupe de diffusion et reçoit ce message. Compléter le code aux endroits indiqués **en gras**.

II.2.1. Socket publieur

```
import java.net.*;
import java.io.*;
import java.util.*;

public class SocketMulticastPublieur {

	static String adresseIPMulticast="234.59.0.2";
	static int numeroPort=1234;
	static int ttl=1; /* portée de la diffusion : le réseau local */

	public static void main(String[] args) {
		try {
			/* Création d'une socket multicast */
			MulticastSocket socketPublieur= new MulticastSocket();

			/* Demande à l'utilisateur de saisir son message au clavier */
			 Scanner keyboard = new Scanner(System.in);
			System.out.print ("Votre message : ");
			String message=keyboard.nextLine();

			/* Tranformation du message en tableau d'octets */
			byte [] buf=message.getBytes();

			/* Création du paquet d'envoi */
			……..

			/* Envoi du paquet */
			……..

			/* Fermeture de la socket */
			……..

		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

II.2.2. Socket abonnée

```
import java.net.*;
import java.io.*;
import java.util.*;

public class SocketMulticastAbonnee {

	static String adresseIPMulticast="234.59.0.2";
	static int numeroPort=1234;

	public static void main(String[] args) {
		try {
			/* Création d'une socket multicast */
			MulticastSocket socketAbonnee= new MulticastSocket(numeroPort);

			/* Joindre le groupe de diffusion à l'adresse multicast définie précédemment */
			……..

			/* Construction du datagramme de réception avec un buffer de 1024 octets */
			DatagramPacket packet= new DatagramPacket(new byte[1024],1024);

			/* Réception du datagramme */
			……..

			/* Affichage de l'adresse IP du publieur */
			……..

			/* Transformation du message reçu sous forme de tableau d'octets en chaîne */
			String message = new String(packet.getData(), packet.getOffset(), packet.getLength());
```

```
            /* Affichage du message */
            System.out.println("Message : "+message);

            /* Désabonnement (on quitte le groupe) */
            ……..

            /* Fermeture de la socket */
            socketAbonnee.close();

        }
        catch (Exception e) {
             e.printStackTrace();
        }
}
```