Projet Encadré par : Mr. Khalid El Gholami

Rapport Projet
Réseaux & Protocoles

Réalisé par:

Yassir Ettoumi

Ziad Ghouzlani

Mohamed Malainine Mohammed Chakour

Youssef Ait Ouchaour

Sockets

Définition :

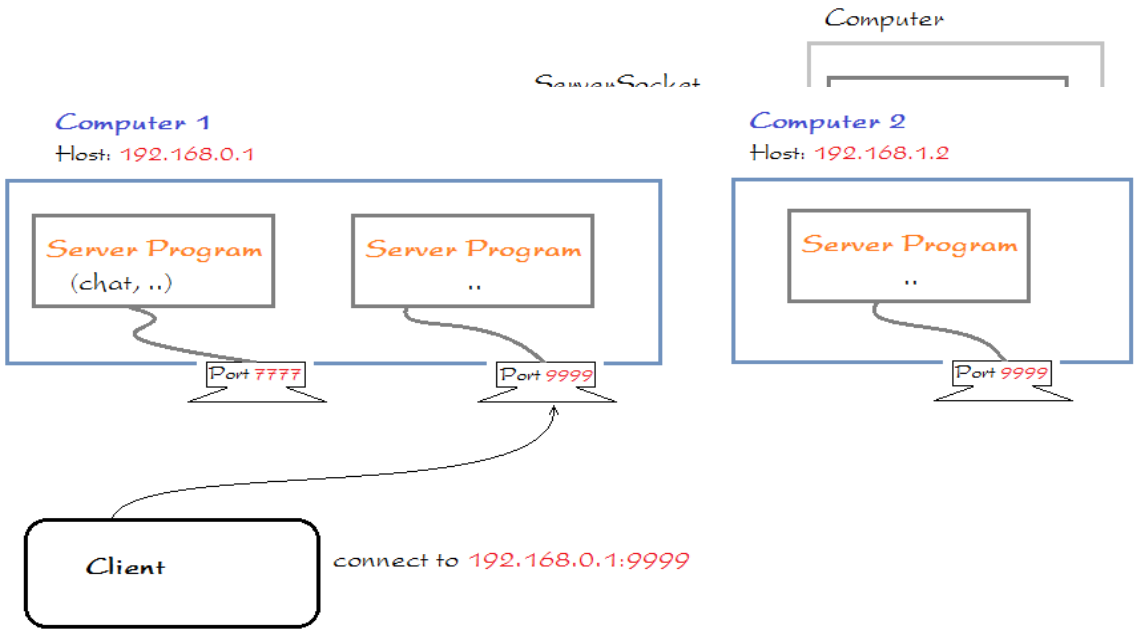
Un Socket est un point de terminaison d'une communication *bidirectionnelle*, c'est-à-dire entre un client et un serveur en cours d'exécution sur un réseau donné. Les deux sont liés par un même numéro de port TCP **(TCP Layer) de sorte que la couche puisse identifier la demande de partage de données**

À côté du serveur (server-side) :

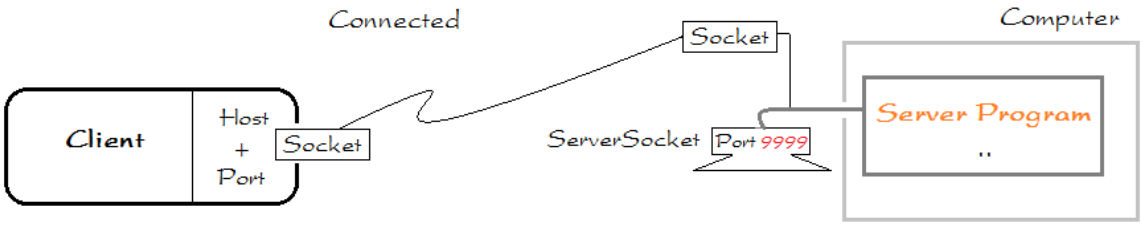
- **En générale, un serveur exécute sur un ordinateur précis et il crée une "socket serveur" (associée à un port) et se met en attente. Le serveur enregistre son service sous un numéro de port. Puis, le serveur se met en attente sur ce service. La classe ServerSocket est utilisée côté serveur : elle attend simplement les appels du ou des clients.**

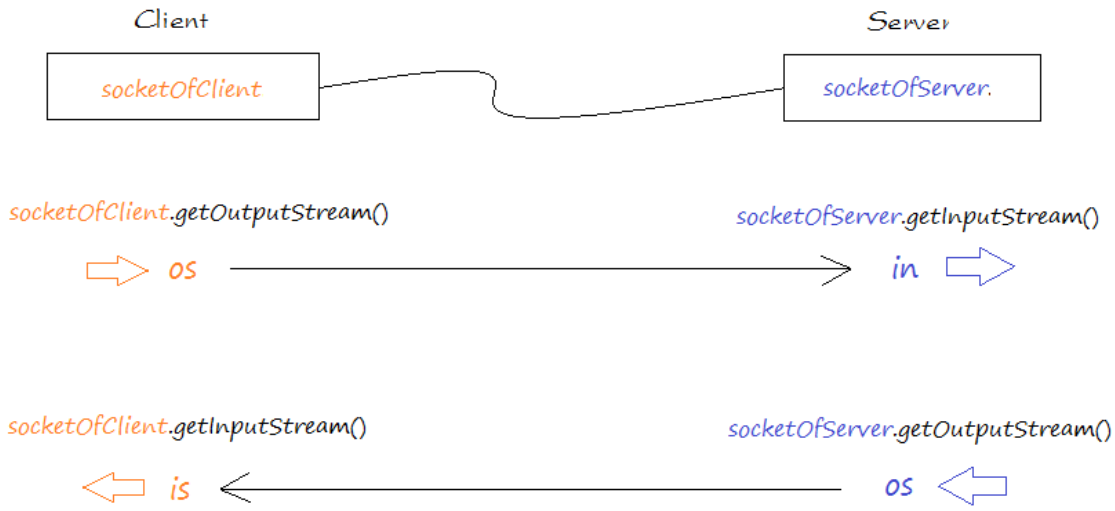
À côté du client (client-side) :

- **Les clients connaissent le nom d'hôte du serveur et le numéro du port que le serveur travaille avec. Le client peut établir une connexion avec le serveur en demandant la création d'une Socket à destination du serveur pour le port sur lequel le service a été enregistré. Le client se connecte à la socket serveur ; deux sockets sont alors crées : une "socket client", côté client, et une "socket service client" côté serveur. Ces sockets sont connectées entre elles.**



Si tout va bien, le serveur (server program) accepte la connexion du client. Au moment de l'acceptation, le serveur reçoit un nouveau socket qui est directement lié au même port local. Il a besoin d'une nouvelle prise de sorte qu'elle puisse continuer à écouter le socket d'origine (ServerSocket) pour les demandes de connexion. Tout t'en satisfaisant les besoins du client connecté. Voici comment accepter une connexion d'un client :





1- Fonctionnement général :

a – On utilisera SocketChannel et ServerSocketChannel qui implémentent la communication asynchrone sous la bibliothèque java.nio.

b – Le serveur établit premièrement le ServerSocketChannel en attendant la connexion du client.

c – Le client doit établir de sa part aussi un canal Socket correspondant pour établir la connexion avec le serveur .

d – Le serveur accepte la connexion du client et génère un socket (ou canal) pour communique avec le client .

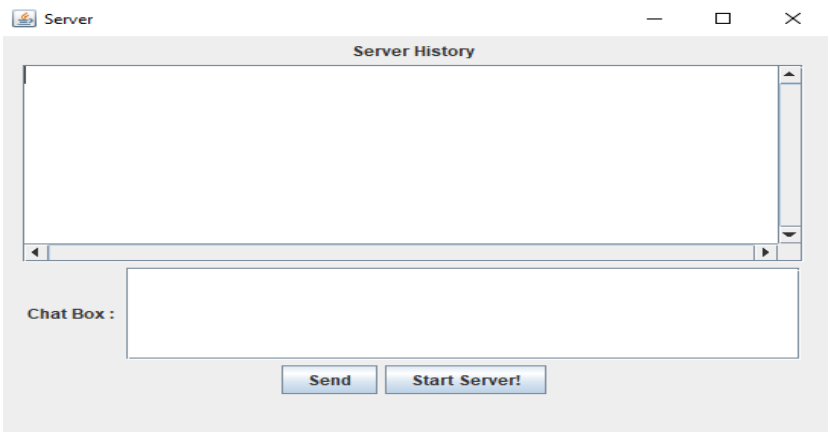
c – On utilisera ainsi les sélecteurs , pour au cas ou il y a de nombreux clients , savoir lesquels qui sont prêt a transmettre leur message .

2- Implémentation Serveur :

```java
public class server extends Thread {
    private static final int PORT=9999;
    private LinkedList Clients;
    private ByteBuffer ReadBuffer;
    private ByteBuffer WriteBuffer;
    public  ServerSocketChannel SSChan;
    private Selector ReaderSelector;
    private CharsetDecoder asciiDecoder;


    public server() {
        Clients=new LinkedList();
        ReadBuffer=ByteBuffer.allocateDirect(300);
        WriteBuffer=ByteBuffer.allocateDirect(300);
        asciiDecoder = Charset.forName( "UTF-8").newDecoder();
    }
```

-On a créé notre classe Server qui hérite de Thread (ce qui va nous permettre de manipuler plusieurs client dans une seul instance de notre classe) , et puis on a déclarer les objet dont nous avons besoin , tel que le port (on va exploiter le port 9999) , et une liste chainée de client dont on va stocker l'ensemble des clients communiquant avec notre programme , et ReadBuffer et WriteBuffer qui sont le conteneur des bytes , qui vont nous aider a lire le message des clients (ReadBuffer) ou envoyer un message au clients (WriteBuffer) . Ainsi que la déclaration du ServerSocketChannel qui va nous aider a écouter sur les tentatives de connections des clients, et le ReaderSelector qui va permettre de choisir lesquels des clients sont prêt pour transmettre le message, ainsi que le décodeur ascii qui va permettre de décoder les bytes transférées par le client .

L'initialisation de notre classe ce fait avec le constructeur server() qui va initialiser la liste chainée des clients et allouer aux buffers 300 bytes comme taille maximal pendant le transfert .



Notre interface graphique s'affichera en premier lieu comme suit , et en cliquant sur Start Server , le bout de code suivant s'execute :

```
Start.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        ChatServer=new server();
        ChatServer.start();

    }
});
```

Donc comme on voit, on crée une instance de notre serveur ChatServer, et on appelle la méthode start , qui va appeler elle-même la méthode run() :

```
public void run() {
    InitServer();

    while(true) {
        acceptNewConnection();

        ReadMassage();
        try {
            Thread.sleep( millis: 100);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
    }
}
```

run() est définit comme quoi qu'elle va appeler la méthode InitServer() :

```
public void InitServer() {
    try {
        SSChan=ServerSocketChannel.open();
        SSChan.configureBlocking(false);

        ServerAddress=InetAddress.getLocalHost();
        System.out.println("L'adresse du localhost : " + ServerAddress.toString());

        SSChan.socket().bind(new InetSocketAddress(ServerAddress,PORT));

        ReaderSelector=Selector.open();
        ChatBox.setText(ServerAddress.getHostName()+"<Server> Started. \n");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Cette dernier qui va ouvrir la connexion du Socket serveur avec la méthode open() , et on configure ainsi qu'il n'y a pas de blocage .

Ensuite on affiche l'adresse de notre serveur dans le terminal , pour permettre aux clients de ce connecter a l'aide de ce dernier , puis on crée

notre socket TCP qui va établir une connexion avec le client a laide de la méthode bind() , et on entend sur notre canal de sélecteur avec open() en attendant la connexion d'un client :

```
public void acceptNewConnection() {
    SocketChannel newClient;
    try {

        while ((newClient = SSChan.accept()) != null) {
            ChatServer.addClient(newClient);

            sendBroadcastMessage(newClient, messg: "Login from: " +newClient.socket().getInetAddress());

            SendMassage(newClient, messg: ServerAddress.getHostName()+"<server> welcome you !\n Note :To exit" +
                    " from server write 'quit' .\n");
        }

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

On définit ainsi la méthode sendMessage() pour envoyer le message, disant bonjour au client connecté :

```
public void SendMessage(SocketChannel client ,String messg) {
    prepareBuffer(messg);
    channelWrite(client);
}

public void SendMessage(String massg) {
    if(Clients.size()>0) {
        for(int i=0;i<Clients.size();i++) {
            SocketChannel client=(SocketChannel)Clients.get(i);
            SendMessage(client,massg);
        }
    }
}
```

Donc ceci informe le client connecté, que le serveur est en écoute et est prêt pour recevoir le message et le lire par la méthode ReadMessage()

```java
public void ReadMessage() {
    try {
        ReaderSelector.selectNow();
        Set readkeys=ReaderSelector.selectedKeys();
        Iterator iter=readkeys.iterator();
        while(iter.hasNext()) {
            SelectionKey key=(SelectionKey) iter.next();
            iter.remove();

            SocketChannel client=(SocketChannel)key.channel();
            ReadBuffer.clear();
            long num=client.read(ReadBuffer);

            if(num==-1) {
                client.close();
                Clients.remove(client);
                sendBroadcastMessage(client, mesg: "logout: " +
                        client.socket().getInetAddress());

            } else {
                StringBuffer str=(StringBuffer)key.attachment();
                ReadBuffer.flip();
                String data= asciiDecoder.decode(ReadBuffer).toString();
                ReadBuffer.clear();
                str.append(data);
                String line = str.toString();
```

```java
                if ((line.indexOf("\n") != -1) || (line.indexOf("\r") != -1)) {
                    line = line.trim();
                    System.out.println(line);

                    if (line.endsWith("quit")) {
                        client.close();
                        Clients.remove(client);
                        ChatBox.append("Logout: " + client.socket().getInetAddress());
                        sendBroadcastMessage(client, mesg: "Logout: "
                                + client.socket().getInetAddress());
                        ChatBox.append(""+'\n');
                    } else {
                        ChatBox.append(client.socket().getInetAddress() + ": " + line);
                        sendBroadcastMessage(client, mesg: client.socket().getInetAddress()
                        + ": " + line);
                        ChatBox.append(""+'\n');
                        str.delete(0,str.length());
                    }
                }
            }
        }

    } catch (IOException ex) {
        ex.printStackTrace();
    }
```

Cette fonction choisit les clés des sockets clients qui sont prêts pour effectuer un opération I/O , pour lire le message diffusé par chaque client séparément a l'aide de l'iterator (iter.next()) , et donc on lit le message diffuse sur le canal dans notre ReadBuffer , et donc on decode les bytes lus sur notre ReadBuffer pour reconstruire notre message .

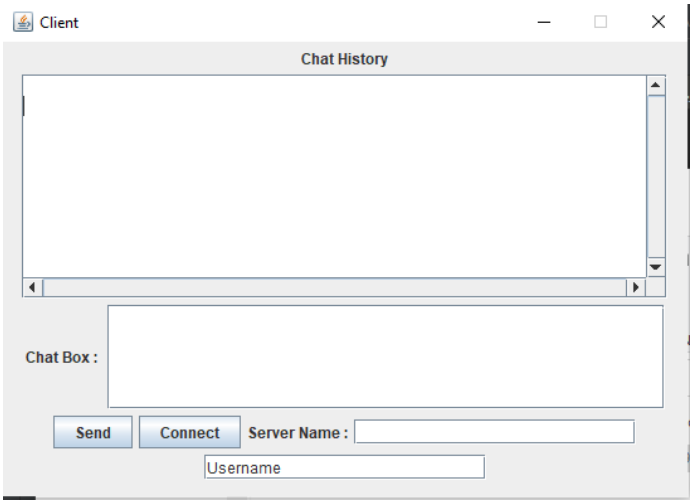
Le message doit est affiché normalement si il ne contient pas la chaine « quit » a la fin , sinon ça déclare que le client désire quitter le chat et on affiche un message informant les autres utilisateurs qu'un utilisateur a quitté .

3- Implémentation Client :

-Le client est une version légère du Serveur , car elle va contenir seul les méthodes pour connecter son propre socket au socket Serveur , et d'autres pour écrire le message au serveur , donc il ya autant de fonction redefinit qu'on a déjà vu dans la partie Serveur.

```
public class Client extends Thread {
    private static final int PORT=9999;
    private LinkedList Clients;
    private ByteBuffer ReadBuffer;
    private ByteBuffer writeBuffer;
    private SocketChannel SChan;
    private Selector ReadSelector;
    private CharsetDecoder asciiDecoder;

    public Client() {
        Clients=new LinkedList();
        ReadBuffer=ByteBuffer.allocateDirect(300);
        writeBuffer=ByteBuffer.allocateDirect(300);
        asciiDecoder = Charset.forName( "US-ASCII").newDecoder();
    }
```

Et donc l'interface client apparait comme suit , avec le champ chat box ou on peut envoyer notre message et le champ Server Name ou on spécifie l'adresse du serveur , qui est dans ce cas l'adresse du localhost (cette adresse est affichée par le serveur après son démarrage), et un champ Username , ou le client spécifie son nom d'utilisateur :



Donc quand on clique pour ce connecter on déclenche cette partie du code :

```
Start.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ChatClient=new Client();
        ChatClient.start();
    }
});
```

```java
public void run() {
    ServerName=Server.getText();
    System.out.println(ServerName);
    UserName=User.getText();
    Connect(ServerName);
    myRead.start();
    while (true) {
        ReadMassage();
        try {
            Thread.sleep( millis: 30);
        } catch (InterruptedException ie){
        }
    }
}
```

On appelle donc la méthode connect() qui sert a connecter le socket client au socket serveur, on va ainsi lire le message que l'utilisateur souhaite transférer sur notre Buffer , et au moment ou on clique Send sur l'interface on appelle :

```java
Send.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(ChatClient!=null) {

            System.out.println(UserText.getText());
            ChatClient.SendMessage(UserText.getText());
        }
    }
});
```

Qui envoie le message lu, et le transfère dans le canal spécifié a la connexion pour être lu par le serveur.

```java
public void ReadMessage() {
    try {
        ReadSelector.selectNow();
        Set readyKeys = ReadSelector.selectedKeys();
        Iterator i = readyKeys.iterator();

        while (i.hasNext()) {
            SelectionKey key = (SelectionKey) i.next();
            i.remove();
            SocketChannel channel = (SocketChannel) key.channel();
            ReadBuffer.clear();
            long nbytes = channel.read(ReadBuffer);
            if (nbytes == -1) {
                ChatBox.append("You logged out !\n");
                channel.close();
            } else {
                StringBuffer sb = (StringBuffer)key.attachment();

                ReadBuffer.flip( );
                String str = asciiDecoder.decode( ReadBuffer).toString( );
                sb.append(str);
                ReadBuffer.clear( );
                String line = sb.toString();
                if ((line.indexOf("\n") != -1) || (line.indexOf("\r") != -1)) {
                    line = line.trim();
                    ChatBox.append("> "+ line);
                    ChatBox.append(""+'\n');
                    sb.delete(0,sb.length());
                }
            }
        }
    } catch (IOException ioe) {
    } catch (Exception e) {
    }
}
```

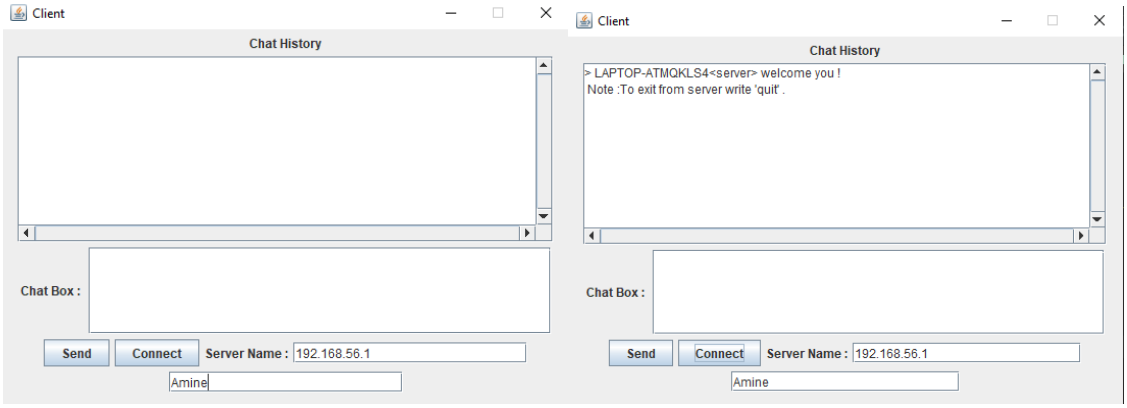
```java
public void SendMessage(SocketChannel client ,String messg) {
    prepareBuffer(messg);
    channelWrite(client);
}

public void SendMessage(String massg) {
    if(Clients.size()>0) {
        for(int i=0;i<Clients.size();i++) {
            SocketChannel client=(SocketChannel)Clients.get(i);
            SendMessage(client,massg);
        }
    }
}
```

4- Démonstration :

Quand on demare le serveur , il s'affiche comme ceci ainsi que l'adresse du serveur .

```
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaag
L'adresse du localhost : LAPTOP-ATMQKLS4/192.168.56.1
```

Le client entre son nom et ladresse du serveur essaye de ce connecter :



Le serveur envoie donc un message de bienvenu , et le client envoie un message disant salut , et on voit que ca s'affiche sur le chat box du serveur