

2091

TP INF1010

CLIENT/SERVEUR

KOALAL MOHAMED AMIR ET Idrick Wilfried
Kuisseu

UQTR AUTOMNE 2019



Introduction

Aujourd'hui, l'utilisation de la technologie pour développer et renforcer notre capacité de communication est essentiel. L'évolution rapide induit un bouleversement des interactions sociales, commerciales, politiques et même personnelles. Les appareils communiquent selon plusieurs types de communication dont la communication Client/serveur.

Dans une communication client/serveur, un programme client demande un service ou une ressource à un autre programme serveur.

Objectifs et buts

Réaliser une application inter-usagés « chat » pour permettre l'échange de messages entre clients d'un même serveur, les communications se feront selon différent type de message dont (list) qui donne au clients la liste de tous les clients connectés.

Nous avons utilisé Java comme langage de programmation afin d'implémenter l'application.

Méthodologie de conception

Pour réaliser cette application la compréhension du concept Client/serveur est nécessaire en effet nous devons comprendre que nous devons faire communiquer des clients en passant par le serveur.

Pour cela un ClientHandler a été ajouté afin de pouvoir gérer la communication entre les clients, le ClientHandler est un programme qui sera créer pour chaque client qui se connecte et gèrera la communication entre le client et le serveur et entre les clients.

Le serveur sera toujours en écoute sur un port bien définit, les clients devront utiliser le même port que le serveur

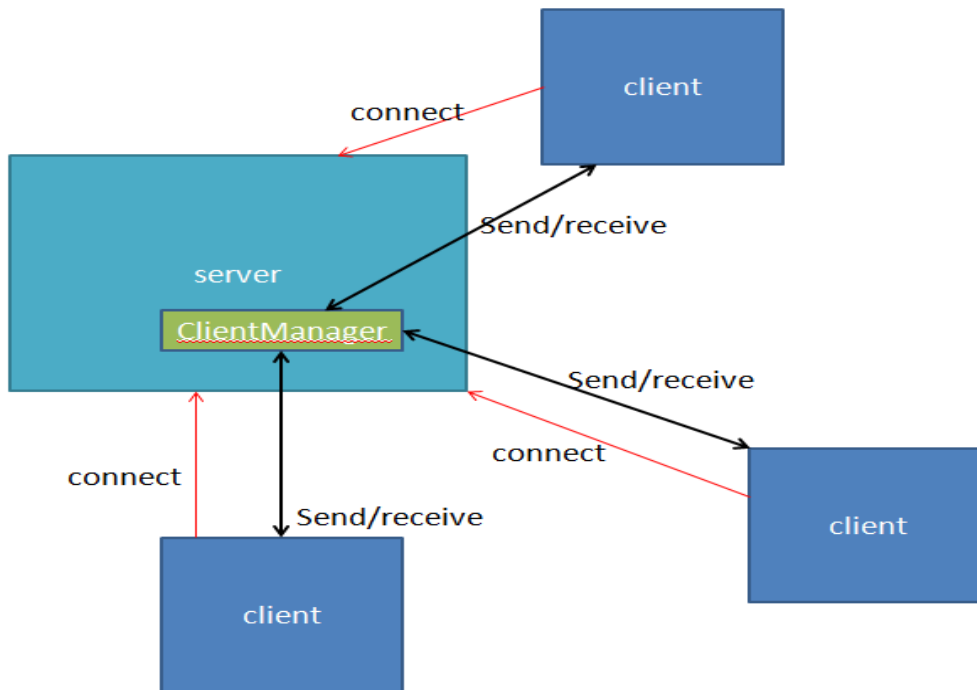
Analyse, description du programme

-Le programme implémenté en java aura une interface Client, sur cette interface l'utilisateur peut voir les messages (a qui et de qui).

-L'utilisateur doit sélectionner les clients auquel il veut envoyer le message

-Trois boutons **SEND**, **LIST** et **CLOSE** permette respectivement d'envoyer le message, demander la liste des clients et de fermer la communication.

Organigramme et algorithmes



Organigramme

PDU :

* P D U *

* *****

* * * *

* MSGTYPE * SENDTO * SENTFROM * MESSAGE *

* * * *

*MSGTYPE :

*GETNAME pour demander un username

* MSG pour les message

- * LIST pour demander la liste de tout les users connecter
- * LOGOUT pour se deconnecter

Croquis du code en JAVA(le code a été très simplifier pour le rapport)

Le serveur

```
while(true)
{
    Socket ClientSocket=Socket.accept();
    ClientManager newClient=new ClientManager(ClientSocket);
}
}
```

Le serveur attend en restant sur accept(), apres la connexion d'un client il créer un client manageur avec la socket du client.

Le ClientHandler

```
run()
{
    SENDLIST(AllClient);
    while(true)
    {
        String msgFromClient=new String();
        msgFromClient=dataInputStream.readUTF();
        Msg = Tokenize(msgFromClient);
        if(Msg.MsgType.equals("LOGOUT"))
        {
```

```

        remove(Client);

        SENDLIST();

        break;
    }

}

else if(Msg.MsgType.equals("MSG")) {

    Send(MSG);

}

else if(MsgType.equals("LIST")) {

    SENDLIST(Client);

}

if(Msg.MsgType.equals("LOGOUT"))

{

    break;

}

}

```

Le ClientHandler tourne tant que le client est connecté. Son rôle est de faire parvenir les messages envoyés du client A vers B, BC ou BCD ou plus. Il gère les messages de type **liste**, **getname**, **logout** et **msg**.

Le Client

```

run(){

    while(true)

    {

        msg=dataInputStream.readUTF();

        Msg = Tokenize(msg);
    }
}

```

```

        if(Msg.MsgType.equals("LIST")) {
            DESTINATIONS.add(ListDestClient)
        }

        if(MsgType.equals("MSG")) {
            jTextArea1.append( "\n" + Msg.SendFrom + " Says :" + message);
        }
    }

    ActionListener(SEND){
        SEND(msg)
    }

    ActionListener(LIST){
        LIST()
    }

    ActionListener(CLOSE){
        LOGOUT()
    }
}

```

MESSAGE TOKENARISATION

type du msg /// sendto ///FIN de DEST///// sentfrom ////////////////// message

MSGTYPE + " " + DEST + " 10110110101 " + USERNAME+ " 10110110101 " +MSG

Le Tokenizer fonctionne comme un split, il permet parcourir le String et d'extraire les informations permettant de reconnaître le type du msg a qui le message est envoyer, de qui et le message

Conclusion

Dans la réalisation de ce TP on s'aperçoit de la première difficulté à savoir comment faire pour modifier une communication Client/serveur qui est de type requête réponse en un système de messagerie, c'est-à-dire transformer la réponse du serveur au client en un message à un autre client. Après une bonne réflexion l'ajout d'une instance comme le ClientHandler permet de transformer la communication C/S en C/C, le principe ne change pas car le message passent tous par le serveur néanmoins le Client Handler permet de retourner la réponse(message) à la bonne destination, en effet si les changements apportés n'ont pas été fait le serveur retournera la réponse (message) au même client qui a envoyer le message.

La deuxième difficulté est comment penser un PDU, au début nous avons pensé à un objet ce qui rendrait la tâche facile en juste accédant à ces propriétés (avec des getters), notre choix c'est tourner de transformer un message en String en PDU en le divisant, en s'inspirant des PDU déjà existant réellement notre String a pris la forme suivante :

MSGTYPE + " " + DEST + " 10110110101 " + USERNAME+" 10110110101 " +MSG

Ainsi un Tokenizer s'occupera d'extraire le type du msg, à qui le message est envoyé, de qui et enfin le message, nous avons aussi ajouté des bits qui servent de séparateurs " 10110110101 ".

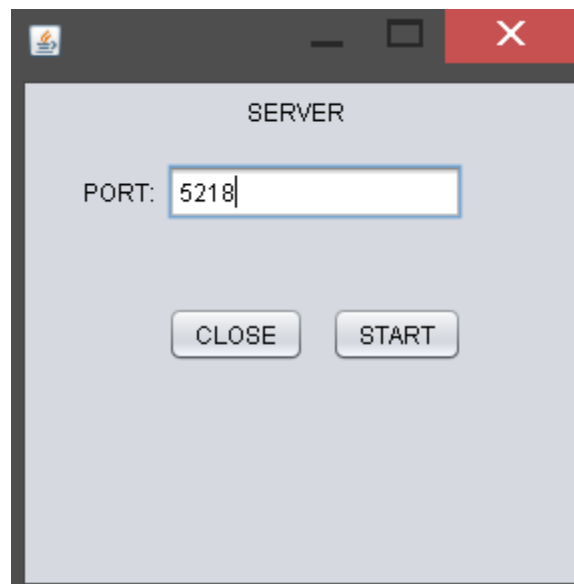
Le Client handler considère le username du destinataire pour choisir la bonne socket de celui-ci pour lui envoyer le message. Dans le cas de Type LIST le client handler considère le USERNAME du client qui a envoyé le message pour choisir sa socket et lui

retourner la liste des clients, dans le cas d'un logout aussi le username et pris en compte pour enlever la socket du clients.

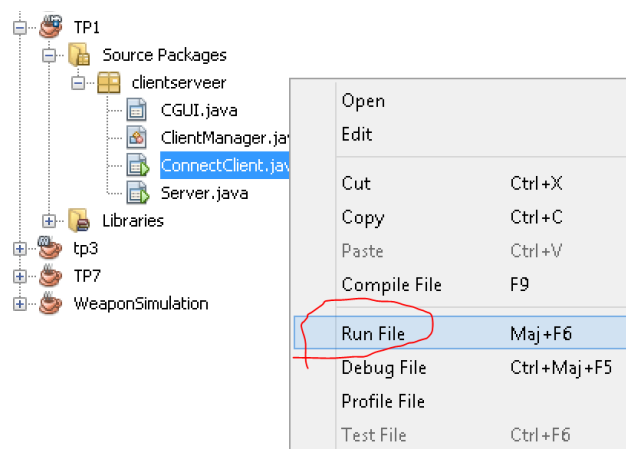
D'autres points peuvent s'ajouter tell que la sécurité comme le hachage a fin de mieux protéger la vie prive des utilisateurs.

Mode d'emplois :

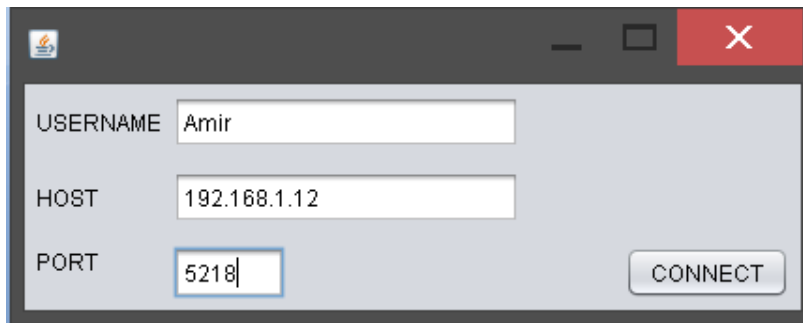
-1 démarrer le server en choisissant un port et cliquer sur Start



2- lancer ConneClient.java



3- entrer les informations de connexion pour vous connecter au serveur



A screenshot of a Java Swing window titled with a standard icon. The window contains three text input fields and a button. The first field is labeled 'USERNAME' and contains the text 'Amir'. The second field is labeled 'HOST' and contains the IP address '192.168.1.12'. The third field is labeled 'PORT' and contains the number '5218'. To the right of these fields is a button labeled 'CONNECT'.

USERNAME	Amir
HOST	192.168.1.12
PORT	5218

CONNECT

-Appuyer sur **CONNECT**



A screenshot of a Java Swing window titled 'amir0'. The window has a large text area at the top containing the text 'Users : amir0'. Below this is a text field labeled 'jTextField1'. At the bottom, there is a table with two columns: 'SELECT' and 'CLIENT'. The 'CLIENT' column contains the text 'amir0'. To the right of the table are three buttons: 'SEND', 'LIST', and 'CLOSE'.

SELECT	CLIENT
<input type="checkbox"/>	amir0

SEND
LIST
CLOSE