Distributed System

Basic of distributed system Network Programmation

EL MAJJODI Abdeljalil

Distribution System and Artificial Intelligence Master's
Department of Mathematics and Informatics
University of Hassan 2 Casablanca

March 27, 2023

Introduction

In this homework, we will look at how to create a server socket with blocking and non-blocking input-output, and we will use the threads to achieve this, and finally, we will try to create a chat application as the application of that with a user interface.
Java, JavaFX, and Python are the languages we use to put this homework into practice.

Contents

List of Figures

Chapter 1

Multi Threads Blocking IO

1.1 Server Multi Threads Implementation

In this section, we'll build a multi-threads server in Java and Python and test it using telnet.

1.1.1 Implementation with Java

```java
package ma.enset;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.*;

public class MTServer implements Runnable {
    static ArrayList<String> clientNames=new ArrayList<>();

    public static void main(String[] args) {
        new Thread(new MTServer()).start();
    }

    @Override
    public void run() {
        ArrayList<Socket> clients=new ArrayList<>();
        int clientId=0;
        try {
            ServerSocket ss=new ServerSocket(123);
            Socket client;

            while (true){
                client=ss.accept();

                clients.add(client);
                new Conversation(client,++clientId,clients).start();
                System.out.println("Client "+clientId+" Joined Conversation
                    IP="+client.getRemoteSocketAddress());
```

```java
        }

    }catch (Exception e){

    }

}


public class Conversation extends Thread{
    private Socket client;
    private int clientId;
    private ArrayList<Socket> clients;

    public Conversation(Socket client,int clientId,ArrayList<Socket>
        clients){
        this.client=client;
        this.clientId=clientId;
        this.clients=clients;

    }
    @Override
    public void run() {
        try {
            String name = null;
            InputStream is = client.getInputStream();
            OutputStream os = client.getOutputStream();

            InputStreamReader isr=new InputStreamReader(is);
            BufferedReader br=new BufferedReader(isr);

            PrintWriter pw=new PrintWriter(os,true);

            //Read Msg Of Client
            String msgClient;

            while ((msgClient= br.readLine())!=null){
                if(msgClient.contains("name:")){
                    name=(msgClient.split(":"))[1];
                    clientNames.add(name);
                    pw.println("Hello "+name+" ,welcom in conversation.");
                }
                else if(msgClient.contains("=>")){
                    String to= (msgClient.split("=>"))[0];
                    msgClient= (msgClient.split("=>"))[1];
                    if(to.contains(",")){
                        String[] msgTo=to.split(",");

                        for(String x : msgTo){
```

```java
                        int index=clientNames.indexOf(x);
                        if(clients.size()>=index && index!=(clientId-1))
                            {
                            pw = new PrintWriter((
                                clients.get(index)).getOutputStream(),
                                true);
                            pw.println(clientNames.get(clientId-1) + " :
                                " + msgClient);
                        }
                    }
                }else{

                        int index=clientNames.indexOf(to);
                        if(clients.size()>=index && index!=(clientId-1)) {
                            pw = new PrintWriter((
                                clients.get(index)).getOutputStream(), true);
                            pw.println(clientNames.get(clientId-1) + " : " +
                                msgClient);
                        }
                    }

                }else {
                    for (Socket c : clients) {
                        if (c != client) {
                            pw = new PrintWriter(c.getOutputStream(), true);
                            pw.println(name + " : " + msgClient);
                        }
                    }
                }

            }

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

    }
  }
}
```

1.1.2   Telnet Test

We start by starting the server("localhost", port=2001), then send a request to connect
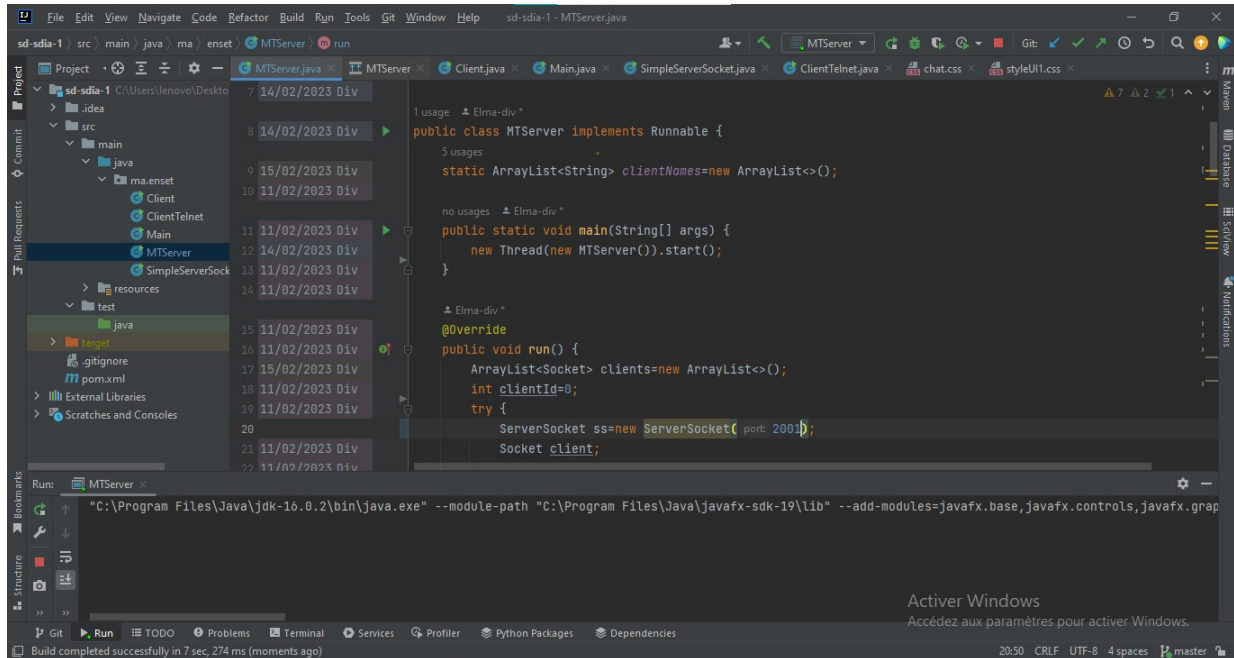to it.

Figure 1.1: Run server

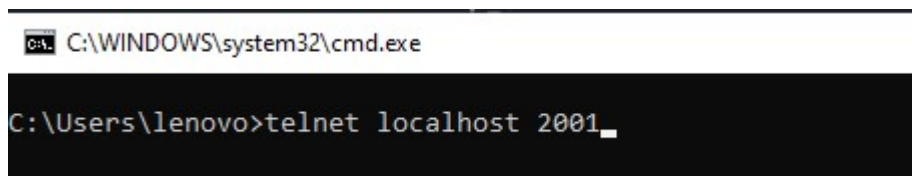Using telnet, the following command will be used to send a request:

Figure 1.2: Telnet command

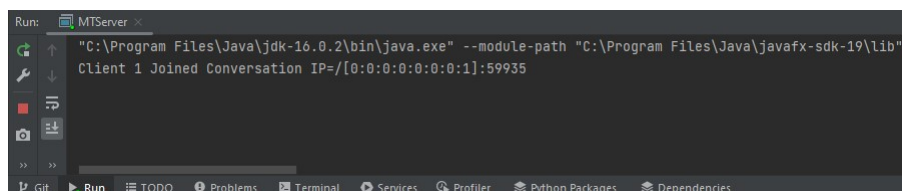The client Telnet connection was established successfully, as shown in the server con-
sole:

Figure 1.3: Client 1 connected

1.2 Create client with user interface

In this section, we'll build a client with the JavaFX user interface.
The client had the option of sending a message to a specific client or to every connected client. Additionally, we'll design an interface that makes this process simple for users.

1.2.1 Implementation with Java/JavaFX

```java
package ma.enset;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.ImageView;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;

import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

import java.io.*;
import java.net.Socket;


public class ClientTelnet extends Application {
    public static void main(String[] args) throws Exception {
        launch(args);
    }
    public void start(Stage stage) throws IOException {
        AnchorPane root2=new AnchorPane();
        VBox centring=new VBox();
        TextField name=new TextField();
        Button accept=new Button("Accepte");
        centring.setSpacing(10);

        //headText
        Text firstTxt=new Text("MyChat Application");
        Text second = new Text("broad-multi cast chat...");

        firstTxt.setFont(Font.font("Verdana", FontWeight.EXTRA_BOLD, 30));
        firstTxt.setLayoutX(135);
        firstTxt.setLayoutY(80);
        firstTxt.setFill(Color.valueOf("#FFFFFFFF"));
        root2.getChildren().add(firstTxt);
```

```java
second.setFont(Font.font("Calibri",20));
second.setLayoutX(205);
second.setLayoutY(100);
second.setFill(Color.valueOf("#FFFFFFFF"));
root2.getChildren().add(second);

name.setPromptText("Username");

centring.getChildren().add(name);
centring.getChildren().add(accept);
centring.setAlignment(Pos.CENTER);
centring.setLayoutX(220);
centring.setLayoutY(120);

Text alert=new Text();
alert.setFill(Color.RED);
centring.getChildren().add(0,alert);

root2.getChildren().add(centring);

Scene userScene=new Scene(root2);
userScene.getStylesheets().add(getClass().
getResource("/style/styleUI1.css").toExternalForm());
stage.setScene(userScene);

AnchorPane root=new AnchorPane();

TextField message=new TextField();
message.setStyle("-fx-border-color: #3469de");
message.setLayoutX(10);
message.setLayoutY(320);
message.setPrefWidth(518);
message.setPrefHeight(30);

Button send=new Button();
send.setLayoutX(528);
send.setLayoutY(317);
ImageView m=new
    ImageView(getClass().getResource("/style/send.png").toExternalForm());
m.setFitWidth(30);
m.setFitHeight(25);
send.setGraphic(m);
send.setBackground(null);

ScrollPane msgPane=new ScrollPane();
msgPane.setStyle("-fx-border-color: #3469de");
msgPane.setPrefWidth(560);
msgPane.setPrefHeight(280);
msgPane.setLayoutX(10);
msgPane.setLayoutY(35);
```

```java
msgPane.setPrefViewportWidth(560);
msgPane.setHbarPolicy(ScrollPane.ScrollBarPolicy.NEVER);
msgPane.setVbarPolicy(ScrollPane.ScrollBarPolicy.NEVER);


Text to=new Text("To :");
to.setX(10);
to.setY(13);
to.setFont(Font.font("Verdana", FontWeight.EXTRA_BOLD, 12));

TextField msgTo=new TextField();
msgTo.setStyle("-fx-border-color: #3469de");
msgTo.setPrefWidth(534);
msgTo.setLayoutX(35);
msgTo.setLayoutY(0);

AnchorPane contentPane=new AnchorPane();
contentPane.setId("contentPane");
contentPane.setStyle("-fx-background-color: WHITE");
contentPane.setLayoutX(0);
contentPane.setLayoutY(0);
contentPane.setPrefWidth(510);
contentPane.setPrefHeight(280);
msgPane.setContent(contentPane);

root.getChildren().add(message);
root.getChildren().add(send);
root.getChildren().add(to);
root.getChildren().add(msgTo);
root.getChildren().add(msgPane);

//connect to srv
Socket socket=new Socket("localhost",123);
InputStream is=socket.getInputStream();
OutputStream os=socket.getOutputStream();

InputStreamReader isr=new InputStreamReader(is);
BufferedReader br=new BufferedReader(isr);

PrintWriter printWriter=new PrintWriter(os,true);

new Thread(()->{
    String serverMsg;
    try{
        while((serverMsg=br.readLine())!=null){
            System.out.println("Here "+serverMsg);

            double y=10;
            if(contentPane.getChildren().size()-1>=0) {
                y = contentPane.getChildren()
```

```java
                .get(contentPane.getChildren().size()-
                    1).getLayoutY()+20;
        }

        Button test=new Button(serverMsg);
        test.setId("inMsg");

        StackPane spane=new StackPane();
        spane.getChildren().add(test);
        spane.setPrefWidth(550);
        spane.setPrefHeight(30);
        spane.setLayoutX(0);
        spane.setLayoutY(y+10);
        spane.setAlignment(Pos.BASELINE_RIGHT);

        Platform.runLater(()->{
            contentPane.getChildren().add(spane);
        });


    }
}catch (Exception e){
}
}).start();

send.setOnAction(actionEvent -> {
    if(!message.getText().isEmpty()){
        double y=10;
        if(contentPane.getChildren().size()-1>=0) {
            y =
                contentPane.getChildren().get(contentPane.getChildren().size()
                - 1).getLayoutY()+20;
        }

        Button test=new Button("me: "+message.getText());
        test.setLayoutX(5);
        test.setLayoutY(y+10);
        test.setId("outMsg");

        contentPane.getChildren().add(test);
        if(!msgTo.getText().isEmpty()){
            printWriter.println(msgTo.getText()+"=>"+message.getText());
        }
        else{
            printWriter.println(message.getText());
        }

        message.setText("");
    }

});
```

```java
        accept.setOnAction(actionEvent -> {
            if(!name.getText().isEmpty() ){
                printWriter.println("name:"+name.getText());
                Scene scene=new Scene(root);
                scene.getStylesheets().add(getClass().
                getResource("/style/chat.css")
                .toExternalForm());
                stage.setScene(scene);
            }
            else {
                alert.setText("Please Enter Your Username ");
            }

        });

        stage.setHeight(400);
        stage.setWidth(600);
        stage.setTitle("MyChat");
        stage.setResizable(false);
        stage.show();

    }
}
```
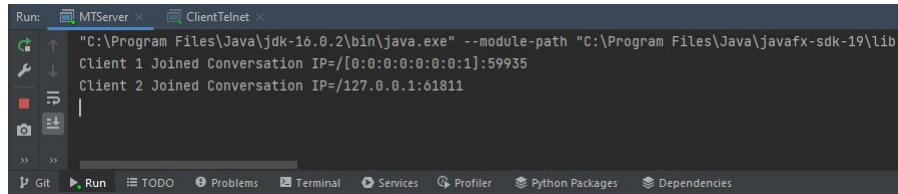
1.2.2 Test

Applying the code yields the following outcome:
Our server informed us that a new client had connected.

Figure 1.4: Client 2 connected

Two sections made up the client interface:

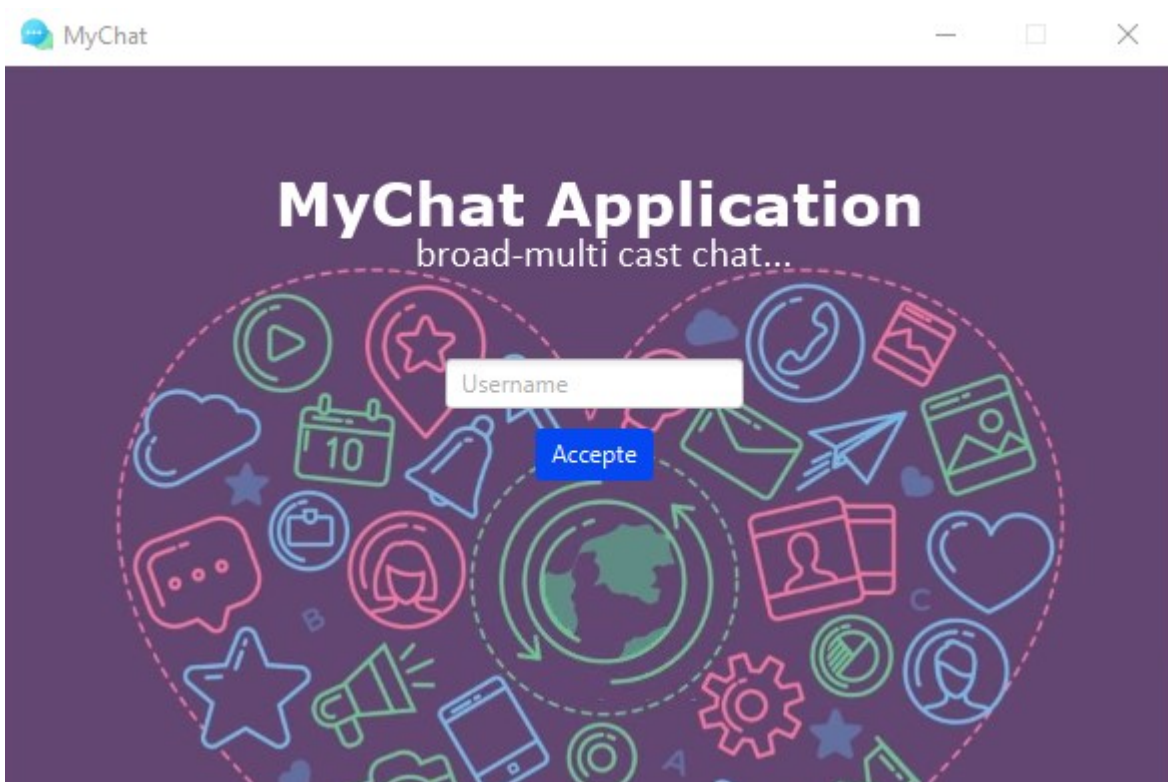- the first was for entering a username:

Figure 1.5: UserInterface 1
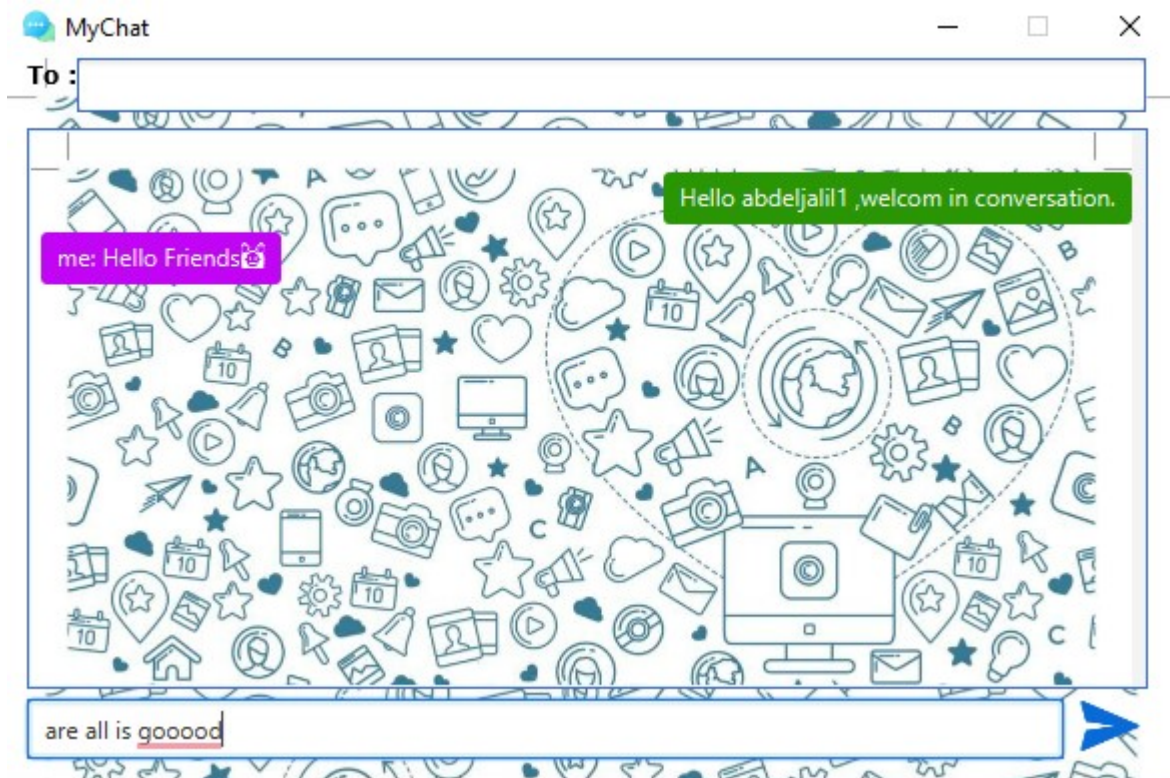
• the second to the conversation:

Figure 1.6: UserInterface 2

We will now create multiple users and test out discussions using unicast and broadcast.
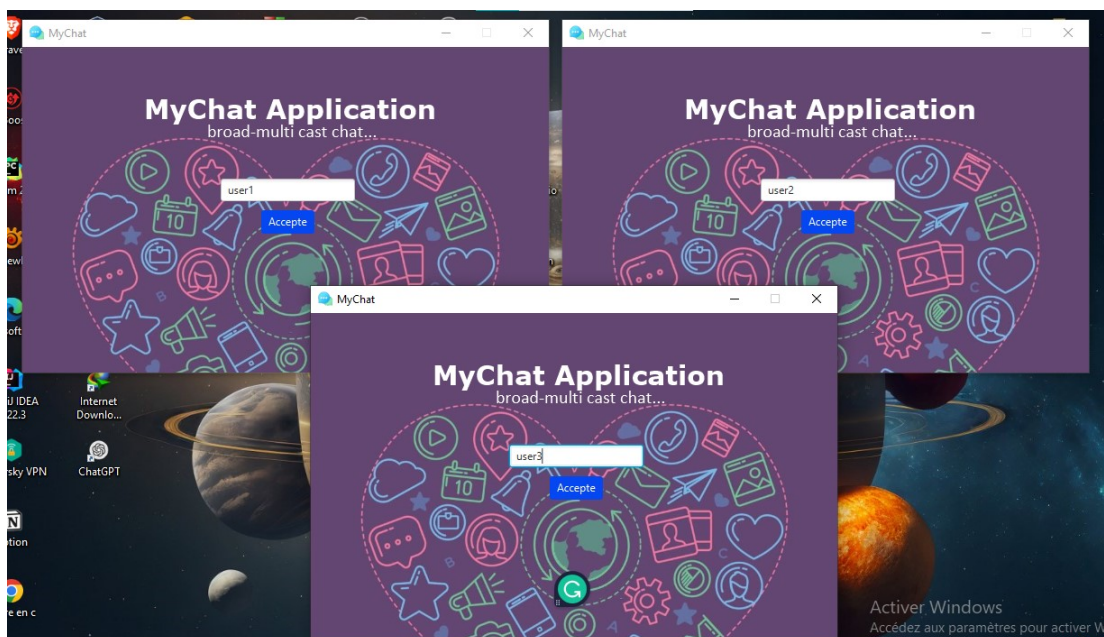
• create 3 users for testing:

Figure 1.7: Create users

• send a message from the user1 to all other users (broadcast).
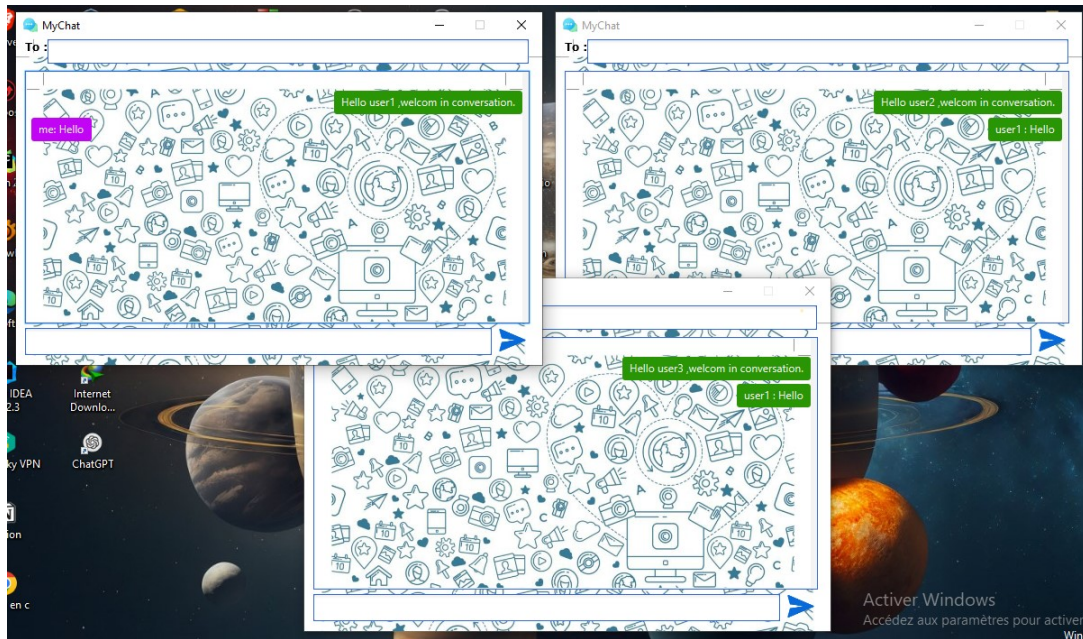

Figure 1.8: Broadcast message

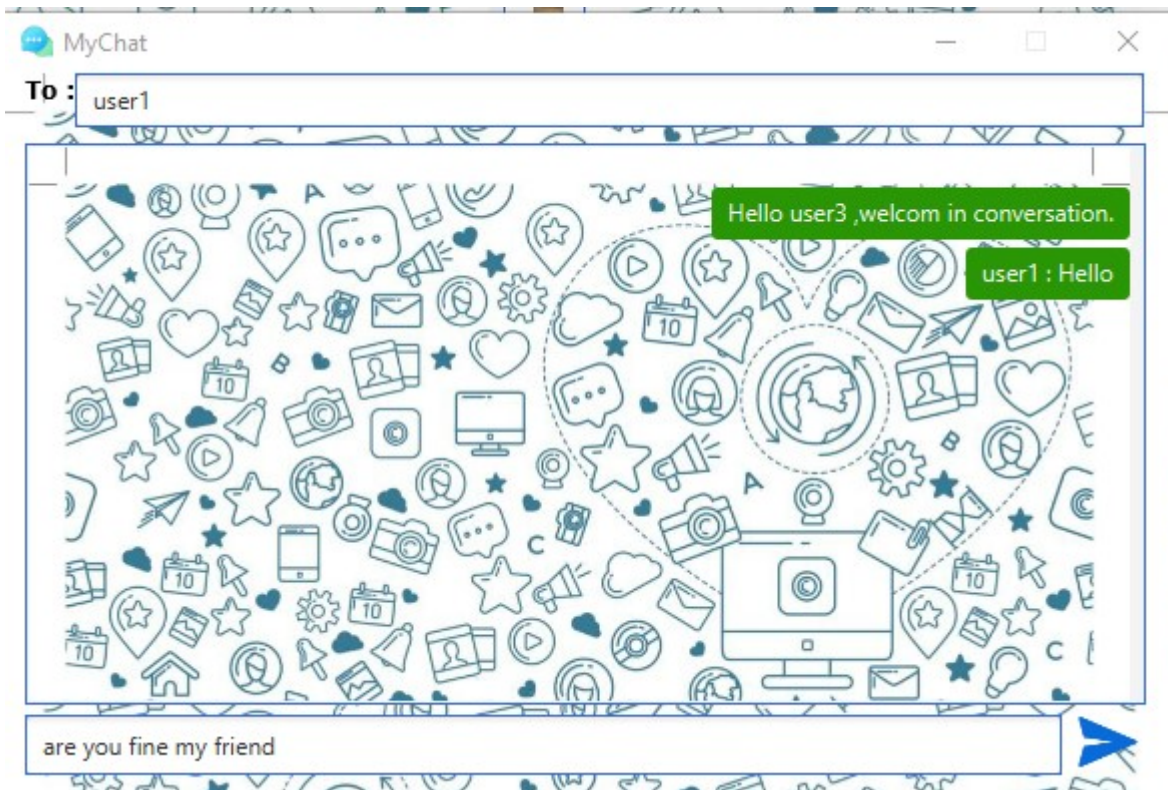• send a message from the user3 to user1 only(unicast).

Figure 1.9: Selecting the users who will receive the message
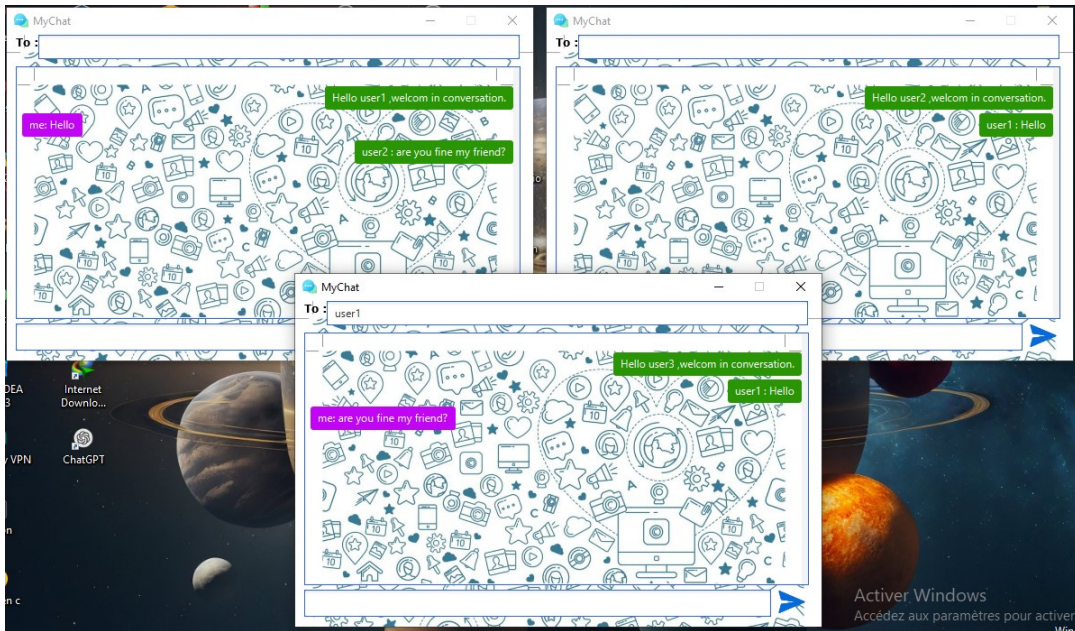
- the message was received by users select.

Figure 1.10: Unicast message

1.3 Create client with Python

With Python, we will now implement the client Telent.

```python
import socket,threading

#listen to receive message
def listen_to_resp(socket):
    while True:
        data=socket.recv(1024).decode()
        print("Msg Received : "+data)


if __name__ == '__main__':
    client=socket.socket()
    client.connect(("127.0.0.1",2001))

    #Thread for read message
    thread=threading.Thread(target=listen_to_resp(client))
    thread.start()

    req=""
    while req.strip()!="exit":
        req=input("->")
        client.send(req.encode())
    client.close()
```

Chapter 2

Single Thread Non-Blocking IO

2.1   Server Single Thread Implementation with NIO

In this step, we develop a server single thread in Java using nio.

2.1.1   Implementation with Java

```java
package org.example;

import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.Iterator;
import java.util.Set;



public class Server {
    public static void main(String[] args) throws Exception {
        //create Selector in mode open
        Selector selector= Selector.open();

        //create Server Socket Channel with mode non-blocking and register in
            selector and put it in mode accept only
        ServerSocketChannel serverSocketChannel=ServerSocketChannel.open();
            //open is a method static can return new srv_sct_channel
        serverSocketChannel.configureBlocking(false);
        serverSocketChannel.bind(new InetSocketAddress("0.0.0.0",2001));
        serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);

        //get the new request in selector...after send it to server
        while(true){
            int count=selector.select(); //return how much of channel request
                0,1,2..,4
```

```java
            //if request doesn't existed
            if(count==0){
                continue;
            }
            //which keys selected
            Set<SelectionKey> selectionKey=selector.selectedKeys();
            Iterator<SelectionKey>
                selectionKeyIterator=selectionKey.iterator(); //iterator to
                read each key

            //read it
            while (selectionKeyIterator.hasNext()){
                SelectionKey key=selectionKeyIterator.next();
                //key isAcceptable or isReadable type we will analyse
                if(key.isAcceptable()){
                    //we accept the request
                    AcceptNew(key,selector);
                } else if (key.isReadable()) {
                    //read a message from client and write to him:
                    ReadWriteToClient(key,selector);
                }
                selectionKeyIterator.remove();
            }


        }

    }

    private static void AcceptNew(SelectionKey key, Selector selector) throws
        Exception {
        //get the server socket channel:
        ServerSocketChannel serverSocketChannel=(ServerSocketChannel)
            key.channel();
        //accept the request and create a socket channel of client:
        SocketChannel socketChannel =serverSocketChannel.accept();
        //mode non-blocking;
        socketChannel.configureBlocking(false);
        //socket Channel register in selector mode read only:
        socketChannel.register(selector,SelectionKey.OP_READ);
        //New Msg in the consol :
        System.out.println("New Connection From :
            "+socketChannel.getRemoteAddress());

    }

    private static void ReadWriteToClient(SelectionKey key, Selector
        selector) throws Exception {
        //get the client
        SocketChannel client=(SocketChannel) key.channel();
        //create a buffer with 1024o in storage
```

```java
        ByteBuffer buffer=ByteBuffer.allocate(256);

        int nbrOct=client.read(buffer);
        //if nbrOct==-1 that's mean client has been disconnected
        if(nbrOct==-1){
            System.out.println("The client "+client.getRemoteAddress()+" has
                been disconnected!!!");
            //close socket Client
            client.socket().close();
            //cancel the key :
            key.cancel();
        }
        else {
            //trim: delete space
            System.out.println("Msg from"+client.getRemoteAddress()+" "+new
                String(buffer.array()).trim());
            buffer.clear();
            buffer.put("Hello".getBytes());
            //change mode of buffer to read / write
            buffer.flip();
            client.write(buffer);
        }

    }

}
```

2.1.2 Telnet Test

We first start the server with the next parameter set to "0.0.0.0", port 2001, and then connect via a Telnet request.
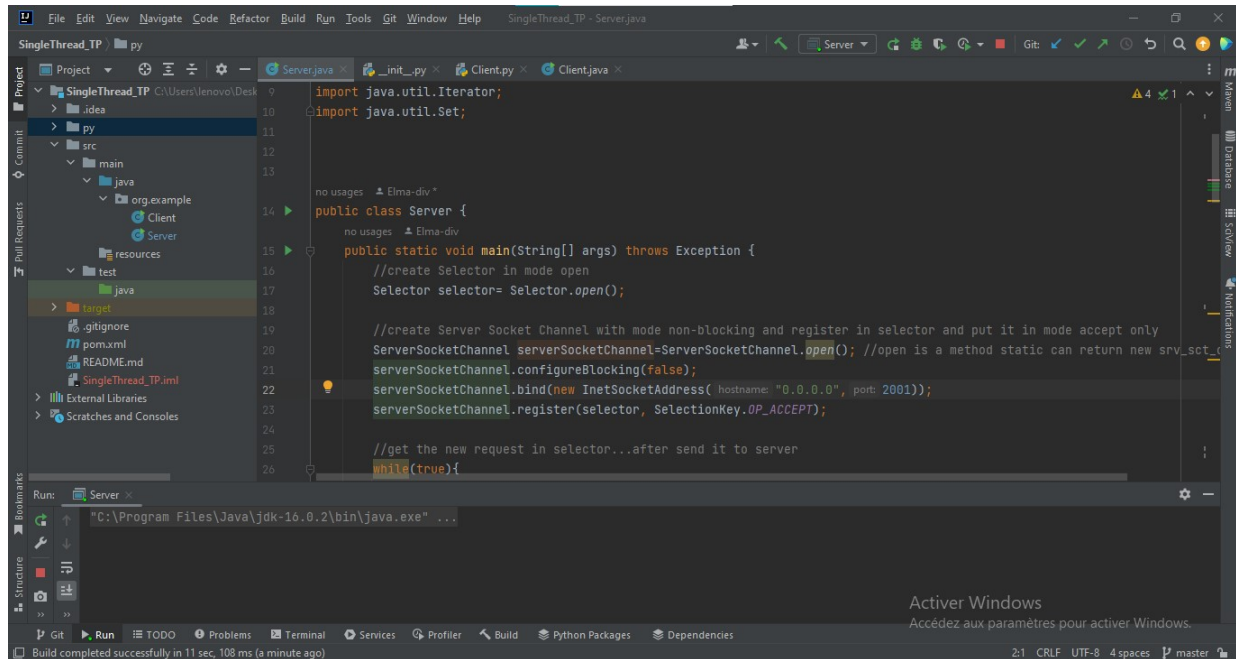
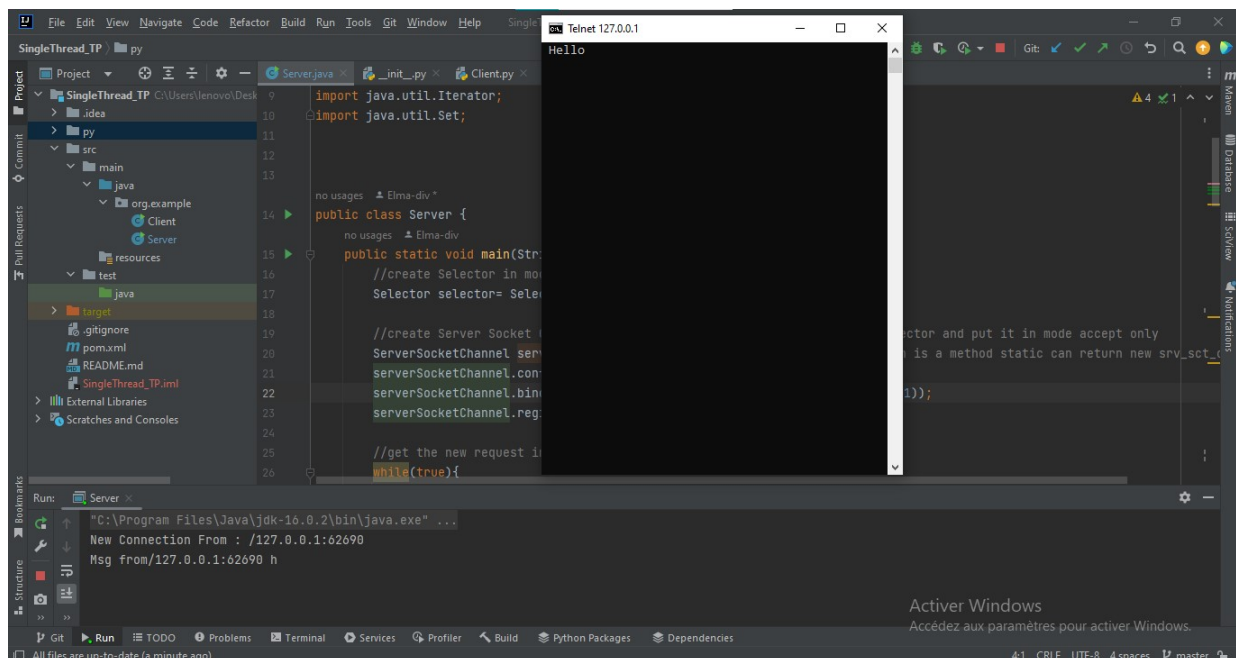

Figure 2.1: Single Thread Server Starting

Figure 2.2: Telnet Connection

2.2   Create Client Telnet

2.2.1   Java Implementation

```java
package org.example;
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SocketChannel;
import java.util.Scanner;

public class Client {
    public static void main(String[] args) throws Exception {
        //connect to srv
        SocketChannel client =SocketChannel.open(new
            InetSocketAddress("localhost",2001));

        //scanner for user input
        Scanner scanner=new Scanner(System.in);

        //Thread for read message come in srv
        new Thread(()->{
            while (true){
                ByteBuffer buffer=ByteBuffer.allocate(1024);
                try {
                    client.read(buffer);
                    if(buffer.array().toString().length()>0){
                        System.out.println("srv msg => "+new
                            String(buffer.array()).trim());
                    }
                }catch (Exception e){

                    try {
                        client.socket().close();
                    } catch (IOException ex) {
                    }
                }
            }
        }).start();

        //Write msg to srv
        while(true){
            String rsp=scanner.nextLine();
            ByteBuffer buffer=ByteBuffer.allocate(1024);
            buffer.put(rsp.getBytes());
            buffer.flip();
            client.write(buffer);
        }
    }
}
```
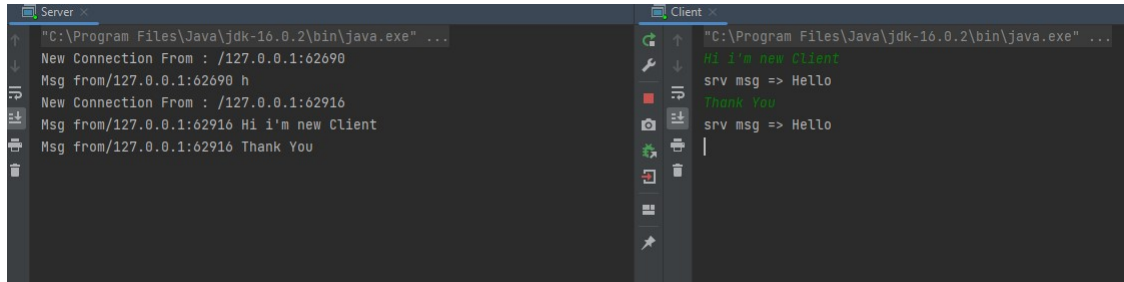
2.2.2 Test client Telnet Java

Figure 2.3: Client Telnet Java

2.2.3 Python Implementation

```python
import socket,threading

#listen to receive message
def listen_to_resp(socket):
    while 1:
        data=socket.recv(1024).decode()
        print("Msg Received : "+data)


if __name__ == '__main__':
    client=socket.socket()
    client.connect(("127.0.0.1",2001))

    #Thread for read message
    thread=threading.Thread(target=listen_to_resp(client))
    thread.start()

    req=""
    while req.strip()!="exit":
        req=input("->")
        client.send(req.encode())
    client.close()
```

Chapter 3

Performance testing using Apache JMeter

We may evaluate each server's performance using JMeter. First, we use JMeter to send a 500 request to test the Multi Threads Server. Next, we attempt with the Single Thread Server and will obtain the following results:
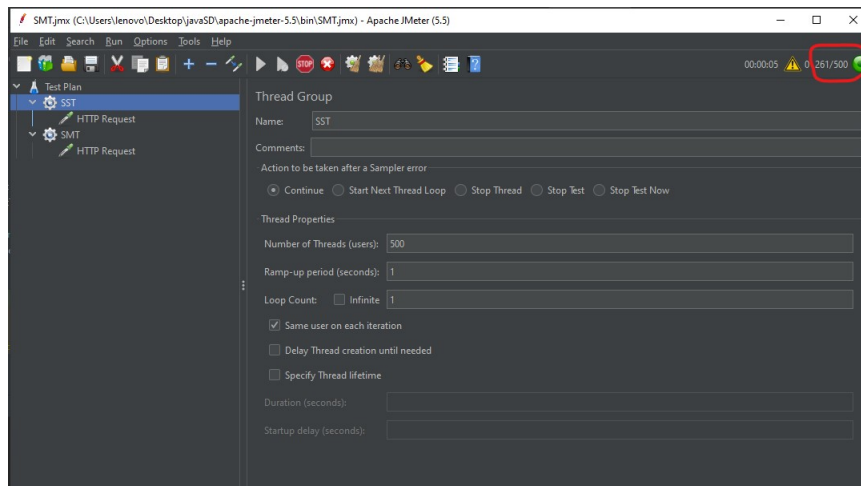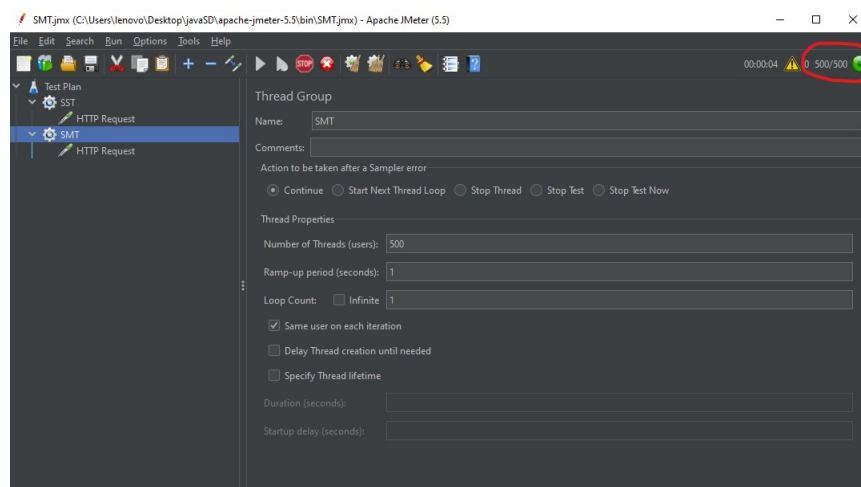

Figure 3.1: JMeter request to SM-Threads

Figure 3.2: JMeter request to SS-Thread

The result shows that the server with many threads accepts more requests than the one with a single thread. This outcome is not what we were expecting.

Conclusions

Input/Output (I/O) operations are an essential part of any program that interacts with external resources like files, network sockets, databases, etc. I/O operations can be performed in two ways: blocking and non-blocking.

Blocking I/O: In blocking I/O, when a program issues an I/O operation (like reading data from a file or sending data over a network), the program execution is halted until the operation completes. During this time, the program cannot do any other task.

Non-blocking I/O: In non-blocking I/O, the program issues an I/O operation, but the execution is not blocked. Instead, the program continues to execute and checks later whether the operation has completed or not. If the operation is not yet completed, the program can perform some other tasks, and check the operation status again later.

Blocking I/O is straightforward to use and can be suitable for simple programs. However, in a more complex system where multiple I/O operations need to be performed simultaneously, blocking I/O can cause significant delays, and the program's performance can suffer. Non-blocking I/O, on the other hand, allows a program to perform multiple I/O operations simultaneously without waiting for one to complete before starting the next. It can improve program performance and scalability in systems that need to handle many I/O operations simultaneously.

In summary, the key difference between blocking and non-blocking I/O is that blocking I/O blocks program execution until the operation is complete, while non-blocking I/O does not block program execution and allows the program to perform other tasks while waiting for I/O operations to complete.