Master Systèmes Distribués et Intelligence Artificielle

Programmation Distribuée et Middlewares

Bases des systèmes distribués
Programmation Réseaux

Réalisé par :
Hicham EL MOUDNI

Enseignant :
Mohamed YOUSSFI

1er mars 2023

Table des matières

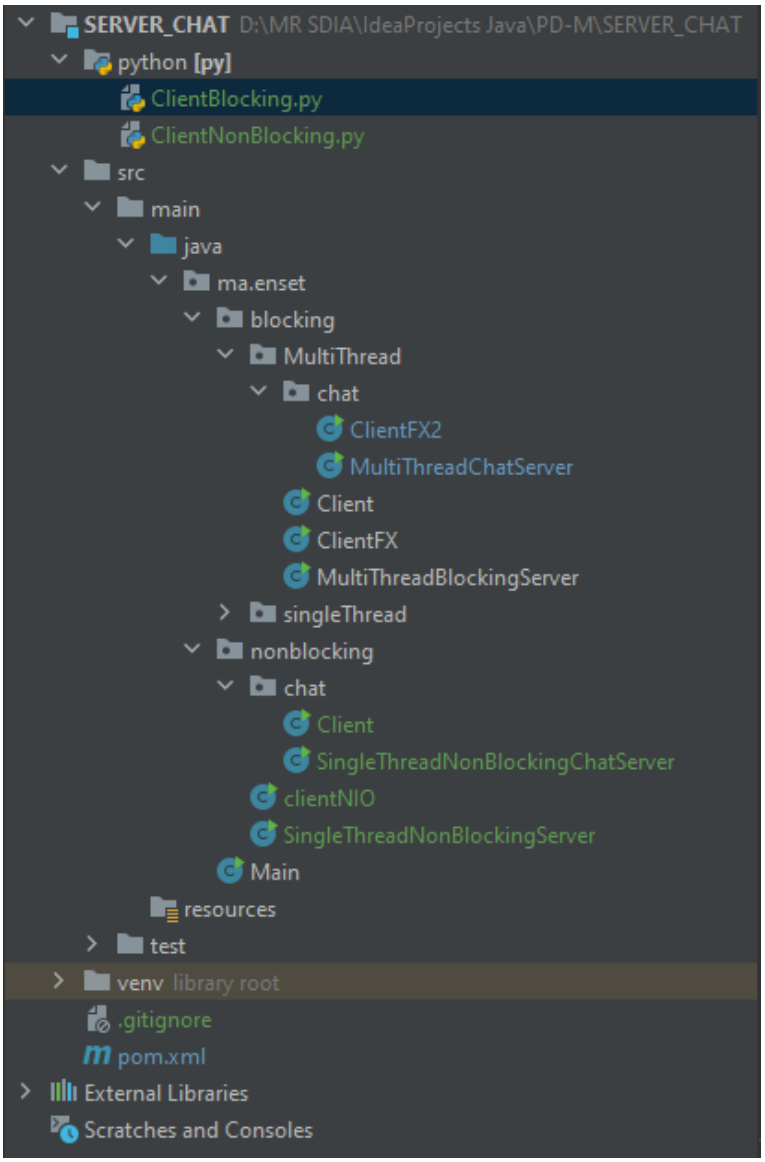
1 Introduction

Dans ce rapport, nous allons explorer deux modèles différents pour le développement d'une application client-serveur de chat. Le premier modèle utilise une approche multi-threads avec des entrées/sorties bloquantes en utilisant la bibliothèque java.io, tandis que le second modèle utilise une approche single-thread avec des entrées/sorties non bloquantes en utilisant la bibliothèque java.nio.

Nous verrons comment ces deux modèles fonctionnent, comment ils peuvent être mis en œuvre et les avantages et inconvénients de chaque approche.

2 Structure de projet

Voici la structure de projet :



3 Modèle Multi Threads Blocking IO

Dans cette partie, nous allons développer un serveur de ChatServer en utilisant le modèle Multi Threads Blocking IO qui ermet de gérer simultanément plusieurs connexions clients au serveur de manière simple, ainsi des clients pour tester la connexion.

3.1 Serveur de Multi Thread Blocking IO de ChatServer

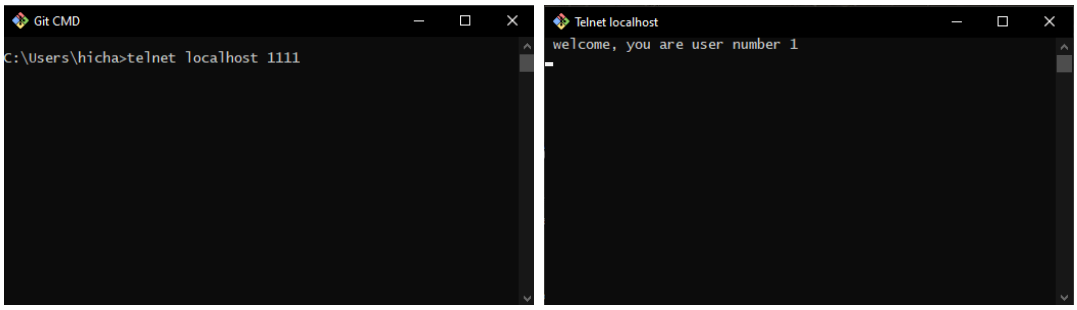
Développement d'un serveur Multi Thread Blocking IO de ChatServer.

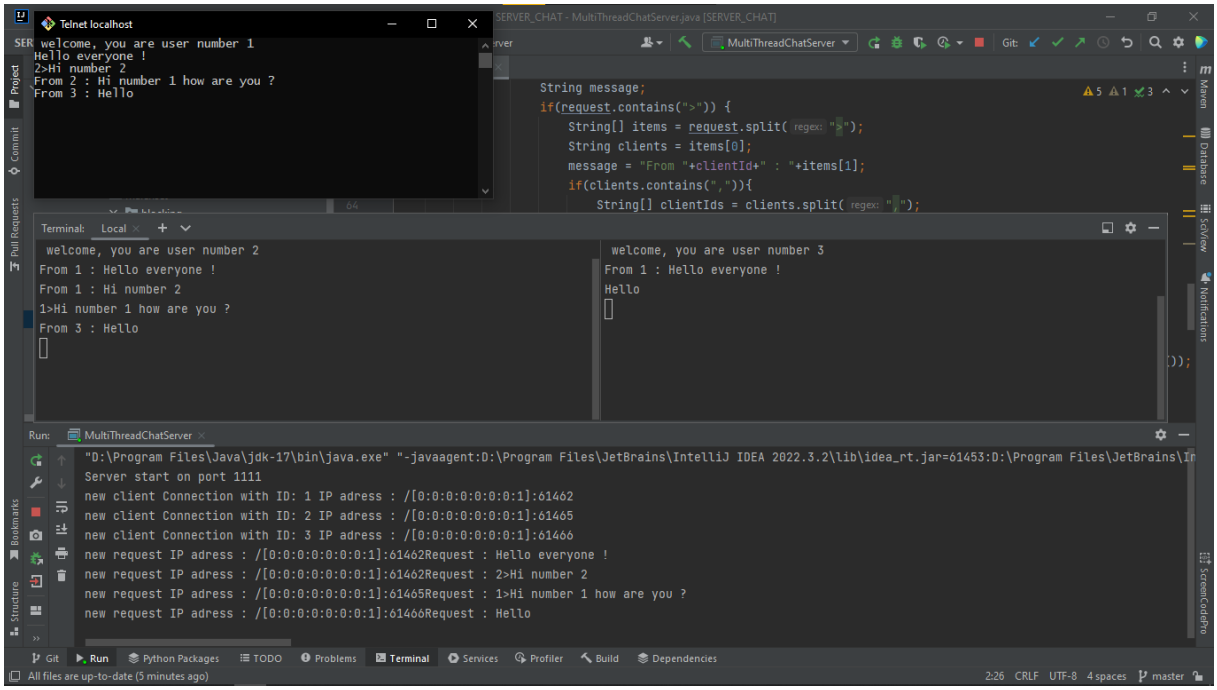
Voici le code :

```java
package ma.enset.blocking.MultiThread;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class MultiThreadChatServer extends Thread{
    private List<Conversastion> conversastionList = new ArrayList<>();
    int clientCounter;
    public static void main(String[] args) {
        new MultiThreadChatServer().start();
    }

    @Override
    public void run() {
        System.out.println("Server start on port 1111");
        try {
            ServerSocket ss = new ServerSocket(1111);
            while (true){
                Socket socket = ss.accept();
                ++ clientCounter;
                Conversastion conversastion = new Conversastion(socket,clientCounter);
                conversastionList.add(conversastion);
                conversastion.start();
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    class Conversastion extends Thread{
        private Socket socket;
        private int clientId;
        public Conversastion(Socket socket,int clientId){
            this.socket = socket;
            this.clientId = clientId;
        }
        @Override
        public void run() {
            try {
                InputStream is = socket.getInputStream();
                InputStreamReader isr = new InputStreamReader(is);
                BufferedReader br = new BufferedReader(isr);

                OutputStream os = socket.getOutputStream();
                PrintWriter pw = new PrintWriter(os,true);

                String ip = socket.getRemoteSocketAddress().toString();
                System.out.println("new client Connection with ID: "+ clientId+ " IP adress : "+ ip);
                pw.println(" welcome, you are user number "+ clientId);
                String request;
                while ((request = br.readLine()) != null){
                    System.out.println("new request IP adress : "+ ip+ "Request : "+ request);
                    List<Integer> clientsTo = new ArrayList<>();
                    String message;
                    if(request.contains(">")) {
                        String[] items = request.split(">");
                        String clients = items[0];
                        message = "From "+clientId+" : "+items[1];
                        if(clients.contains(",")){
                            String[] clientIds = clients.split(",");
                            for (String id:clientIds) {
                                clientsTo.add(Integer.parseInt(id));
                            }
                        }else{
                            clientsTo.add(Integer.parseInt(clients));
                        }
                    }else {
                        clientsTo = conversastionList.stream().map(c ->c.clientId).collect(Collectors.toList());
                        message = "From "+clientId+" : "+request;
                    }
                    broadcastMessage(message, this,clientsTo);
                }
            } catch (IOException e) {
                throw new RuntimeException(e);
            }

        }
    }

    public void broadcastMessage(String message, Conversastion from, List<Integer> clients){
        try {
            for(Conversastion conversastion:conversastionList) {
                if(conversastion != from && clients.contains(conversastion.clientId)) {
                    Socket socket = conversastion.socket;
                    OutputStream os = socket.getOutputStream();
                    PrintWriter pw = new PrintWriter(os, true);
                    pw.println(message);
                }
            }

            } catch (IOException e) {
                throw new RuntimeException(e);
            }
    }

}

```

3.2 Teste de serveur avec un client Telnet

Après avoir développé notre serveur ChatServer, nous allons utiliser un client Telnet pour tester la connexion avec le serveur. Nous allons vérifier si les commandes envoyées par le client sont correctement reçues et traitées par le serveur comme montre les figures suivantes :





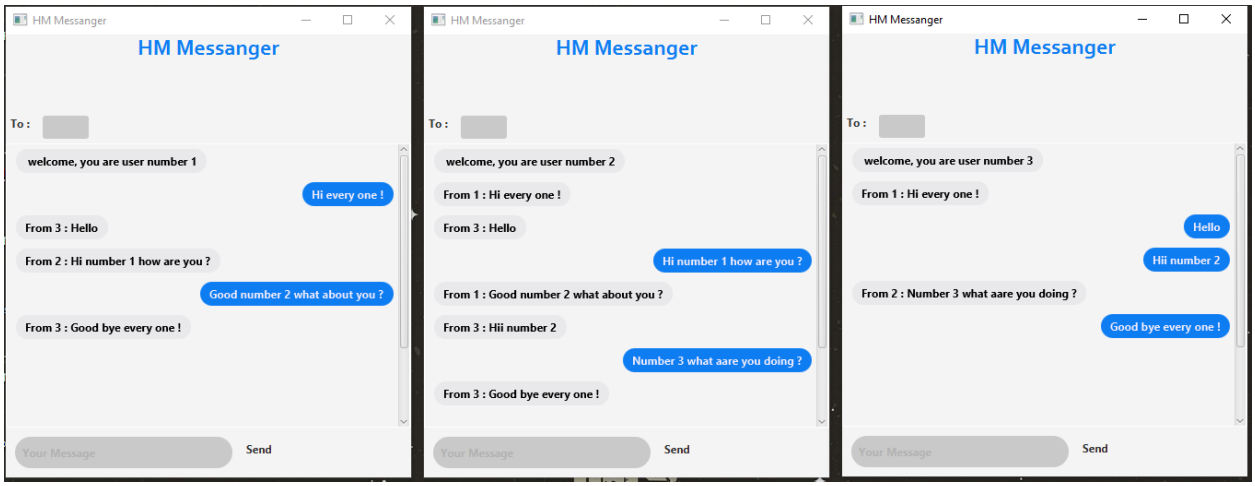
3.3 Client Java avec une interface graphique JavaFX

En plus de la connexion avec Telnet, nous allons également mettre en place un client Java avec une interface graphique JavaFX pour permettre aux utilisateurs d'interagir avec le serveur.

```java
package ma.enset.blocking.MultiThread.chat;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.scene.paint.Paint;
import javafx.scene.text.Text;
import javafx.scene.text.TextFlow;
import javafx.stage.Stage;

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

public class ClientFX2 extends Application {
    PrintWriter pw;
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage stage) throws Exception {
        stage.setTitle("HM Messanger");
        BorderPane borderPane = new BorderPane();

        borderPane.setStyle("-fx-font-weight: bold;");
        ////"-fx-background-color: white;"+
        Label title = new Label("HM Messanger");
        title.setAlignment(Pos.CENTER);

        title.setStyle("-fx-font-family: Corbel;"+"-fx-font-size: 24px;"+"-fx-text-fill: #0f7df2;");

        Label labelHost = new Label("");
        TextField textFieldHost = new TextField("localhost");

        Label labelPort = new Label(":");
        TextField textFieldPort = new TextField("1111");

        Button buttonConnecter = new Button("Connect");

        Label labelTo = new Label("To : ");
        TextField textFieldTo = new TextField();
        textFieldTo.setPrefWidth(50);
        textFieldTo.setStyle("-fx-background-color: #c9c9c9;");

        HBox hBox6 = new HBox();
        hBox6.setSpacing(10);
        hBox6.setPadding(new Insets(5));
        //hBox.setBackground(Background.fill(Color.AZURE));
        hBox6.getChildren().addAll(labelTo,textFieldTo);

        HBox hBox = new HBox();
        hBox.setSpacing(10);
        hBox.setPadding(new Insets(5));
        //hBox.setBackground(Background.fill(Color.AZURE));
        hBox.getChildren().addAll(labelHost,textFieldHost,labelPort,textFieldPort,buttonConnecter);
        VBox vBox5 = new VBox();
        vBox5.setAlignment(Pos.CENTER);
        vBox5.getChildren().addAll(title,hBox,hBox6);
        vBox5.setSpacing(10);
        borderPane.setTop(vBox5);

        VBox vBox = new VBox();
        vBox.setPrefSize(248,412);
        ScrollPane scrollPane = new ScrollPane();
        scrollPane.setPrefSize(256,418);
        scrollPane.setFitToWidth(true);
        scrollPane.setLayoutX(30);
        scrollPane.setLayoutY(70);
        scrollPane.setStyle("-fx-background-color: white;");
        scrollPane.setContent(vBox);
        borderPane.setCenter(scrollPane);

        //Label labelMsg = new Label("Message");
        TextField textFieldMsg = new TextField();
        textFieldMsg.setPromptText("Your Message");
        textFieldMsg.setPrefSize(236, 34);
        textFieldMsg.setStyle("-fx-background-color: #c9c9c9;"+"-fx-background-radius: 30px;");

        Button buttonMsg = new Button("Send");
        buttonMsg.setPadding(new Insets(5));
        buttonMsg.setStyle("-fx-background-color: transparent;");

        HBox hBox1 = new HBox();
        hBox1.setSpacing(10);
        hBox1.setPadding(new Insets(10));
        //hBox1.setBackground(Background.fill(Color.AZURE));
        hBox1.getChildren().addAll(textFieldMsg,buttonMsg);
        borderPane.setBottom(hBox1);


        ////
        Scene scene = new Scene(borderPane,440, 480);
        stage.setScene(scene);
        stage.show();
```

```
        buttonConnecter.setOnAction((actionEvent) -> {
            String host = textFieldHost.getText();
            int port = Integer.parseInt(textFieldPort.getText());
            try {
                Socket socket = new Socket(host,port);
                InputStream is = socket.getInputStream();
                InputStreamReader isr = new InputStreamReader(is);
                BufferedReader br = new BufferedReader(isr);

                OutputStream os = socket.getOutputStream();
                pw = new PrintWriter(os,true);

                new Thread(() -> {
                    while (true) {
                        try {
                            String response = br.readLine();
                            HBox hBox2 = new HBox();
                            hBox2.setAlignment(Pos.CENTER_LEFT);
                            hBox2.setPadding(new Insets(5, 5, 5, 10));

                            Text text = new Text(response);
                            TextFlow textFlow = new TextFlow(text);

                            textFlow.setStyle(
                                    "-fx-background-color: rgb(233, 233, 235);" +
                                            "-fx-background-radius: 20px;");

                            textFlow.setPadding(new Insets(5, 10, 5, 10));
                            hBox2.getChildren().add(textFlow);

                            Platform.runLater(new Runnable() {
                                @Override
                                public void run() {
                                    vBox.getChildren().add(hBox2);
                                }
                            });
                        }catch (IOException e) {
                                throw new RuntimeException(e);
                        }

                    }
                }).start();

            } catch (IOException e) {
                throw new RuntimeException(e);
            }
            hBox.setVisible(false);
        });

        buttonMsg.setOnAction((event) -> {
            String to = textFieldTo.getText();
            String messageToSend = textFieldMsg.getText();
            if (!messageToSend.isEmpty()) {
                HBox hBox3 = new HBox();
                hBox3.setAlignment(Pos.CENTER_RIGHT);

                hBox3.setPadding(new Insets(5, 5, 5, 10));
                Text text = new Text(messageToSend);
                TextFlow textFlow = new TextFlow(text);
                textFlow.setStyle(
                        "-fx-color: rgb(239, 242, 255);" +
                                "-fx-background-color: rgb(15, 125, 242);" +
                                "-fx-background-radius: 20px;");

                textFlow.setPadding(new Insets(5, 10, 5, 10));
                text.setFill(Color.color(0.934, 0.925, 0.996));

                hBox3.getChildren().add(textFlow);
                vBox.getChildren().add(hBox3);

                if(to.isEmpty()){
                    pw.println(messageToSend);
                } else{
                    pw.println(to+">"+messageToSend);
                }

                textFieldMsg.clear();
                textFieldTo.clear();
            }
        });
    }
}
```

Voici les résultats de test avec les clients Javafx :




```
"D:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\IntelliJ IDEA
Server start on port 1111
new client Connection with ID: 1 IP adress : /127.0.0.1:61492
new client Connection with ID: 2 IP adress : /127.0.0.1:61493
new client Connection with ID: 3 IP adress : /127.0.0.1:61494
new request IP adress : /127.0.0.1:61492Request : Hi every one !
new request IP adress : /127.0.0.1:61494Request : Hello
new request IP adress : /127.0.0.1:61493Request : 1>Hi number 1 how are you ?
new request IP adress : /127.0.0.1:61492Request : 2>Good number 2 what about you ?
new request IP adress : /127.0.0.1:61494Request : 2>Hii number 2
new request IP adress : /127.0.0.1:61493Request : 3>Number 3 what aare you doing ?
new request IP adress : /127.0.0.1:61494Request : Good bye every one !
```

3.4 Teste de serveur avec un client Python

Nous allons également explorer la possibilité de créer un client Chat en utilisant un autre langage de programmation, en l'occurrence Python.

```python
import socket
import threading

class Client:
    def __init__(self):
        try:
            self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.socket.connect(('localhost', 1111))

            self.is_ = self.socket.makefile('r')
            self.os_ = self.socket.makefile('w')

            threading.Thread(target=self.receive_messages, daemon=True).start()

            while True:
                request = input()
                self.os_.write(request + '\n')
                self.os_.flush()

        except socket.error as e:
            raise RuntimeError(e)

    def receive_messages(self):
        try:
            while True:
                request = self.is_.readline().rstrip()
                if not request:
                    raise socket.error('Disconnected')
                print(request)

        except socket.error as e:
            raise RuntimeError(e)
if __name__ == '__main__':
    Client()
```

Voici les résultats de test avec les clients Python, Java et Telnet :


```
"D:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\IntelliJ
Server start on port 1111
new client Connection with ID: 1 IP adress : /127.0.0.1:60266
new client Connection with ID: 2 IP adress : /127.0.0.1:60269
new client Connection with ID: 3 IP adress : /[0:0:0:0:0:0:0:1]:60270
new request IP adress : /127.0.0.1:60266Request : hi
new request IP adress : /127.0.0.1:60269Request : hello I'm python
new request IP adress : /127.0.0.1:60266Request : 2>hii python
new request IP adress : /127.0.0.1:60269Request : 1>hi client java !
```

4 Modèle Single Thread avec Non Blocking IO

Dans cette partie, nous allons développer un serveur de ChatServer en utilisant le modèle Single Thread avec Non Blocking IO qui ermet de gérer simultanément plusieurs connexions clients au serveur de manière simple, ainsi des clients pour tester la connexion.

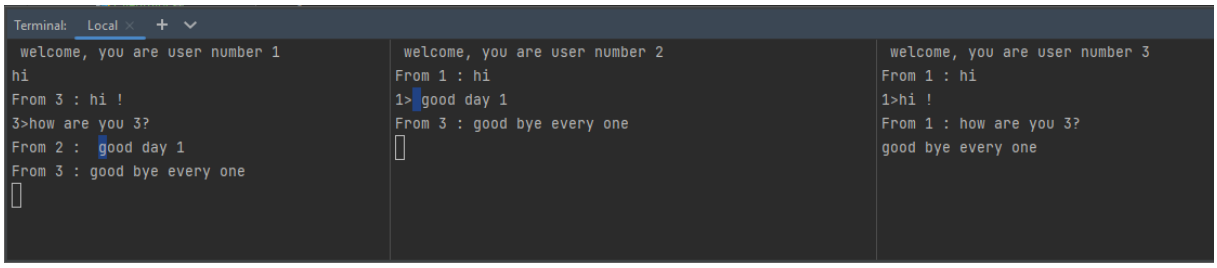
4.1 Serveur de Single Thread Non Blocking IO de ChatServer

Développement d'un serveur Single Thread Non Blocking IO de ChatServer

```java
package ma.enset.nonblocking.chat;

import java.io.IOException;
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.*;

public class SingleThreadNonBlockingChatServer {
    private Map<SocketChannel,Integer> socketChannels=new HashMap<>();
    private int clientsCount;
    public static void main(String[] args) throws Exception {
        new SingleThreadNonBlockingChatServer();
    }
    public SingleThreadNonBlockingChatServer(){
        this.startServer();
    }
    public void startServer(){
        try {
            Selector selector=Selector.open();
            ServerSocketChannel serverSocketChannel=ServerSocketChannel.open();
            serverSocketChannel.bind(new InetSocketAddress("0.0.0.0",1111));
            serverSocketChannel.configureBlocking(false);
            serverSocketChannel.register(selector, SelectionKey.OP_ACCEPT);
            while (true){
                int readyChannels = selector.select();
                if (readyChannels==0) continue;
                Set<SelectionKey> selectionKeys = selector.selectedKeys();
                Iterator<SelectionKey> iterator = selectionKeys.iterator();
                while (iterator.hasNext()){
                    SelectionKey selectionKey = iterator.next();
                    if(selectionKey.isAcceptable()){
                        handleForAccept(selector,selectionKey);
                    } else if(selectionKey.isReadable()){
                        handleForRead(selector,selectionKey);
                    }
                    iterator.remove();
                }
            }
        } catch (Exception e) { e.printStackTrace(); }
    }

    private void handleForAccept(Selector selector, SelectionKey selectionKey) throws IOException {
        ServerSocketChannel serverSocketChannel= (ServerSocketChannel) selectionKey.channel();
        SocketChannel socketChannel = serverSocketChannel.accept();
        ++clientsCount;
        socketChannels.put(socketChannel,clientsCount);
        socketChannel.configureBlocking(false);
        socketChannel.register(selector,SelectionKey.OP_READ);
        System.out.println("New Connection from : "+clientsCount);
        sendMessage(String.format("Welcome you are client number %s",clientsCount),socketChannel);
    }
    private void handleForRead(Selector selector, SelectionKey selectionKey) throws IOException {
        SocketChannel socketChannel = (SocketChannel) selectionKey.channel();
        ByteBuffer byteBuffer = ByteBuffer.allocate(1024);
        if (socketChannel.isConnected()){
            int read = socketChannel.read(byteBuffer);
        if (read == -1) {
            System.out.println("Client disconnected ....");
            socketChannels.remove(socketChannel);
            socketChannel.close();
            socketChannel.keyFor(selector).channel();
        } else {
            String request = new String(byteBuffer.array()).trim();
            if (request.length() > 0) {
                String message = request;
                List<Integer> destinationList = new ArrayList<>();
                String[] requestItems = request.split(">");
                if (requestItems.length == 2) {
                    String destination = requestItems[0];
                    message = requestItems[1];
                    if (destination.trim().contains(",")) {
                        String[] destinations = destination.trim().split(",");
                        for (String d : destinations) {
                            destinationList.add(Integer.parseInt(d));
                        }
                    } else {
                        destinationList.add(Integer.parseInt(destination));
                    }
                }
                broadCastMessage(message, socketChannel, destinationList);
            }
        }
    }
    }
    private void broadCastMessage(String message, SocketChannel from, List<Integer> destinations) throws IOException {
        for (SocketChannel socketChannel:socketChannels.keySet()){
            int clientId=socketChannels.get(socketChannel);
            boolean all=destinations.size()==0;
            if(!socketChannel.equals(from) && (destinations.contains(clientId) || all)){
                ByteBuffer byteBufferResponse=ByteBuffer.allocate(1024);
                int fromId=socketChannels.get(from);
                String formattedMessage=String.format("New message from user number : %s with content : %s",fromId,message);
                byteBufferResponse.put(formattedMessage.getBytes());
                byteBufferResponse.flip();
                socketChannel.write(byteBufferResponse);
            }
        }
    }
    private void sendMessage(String message, SocketChannel socketChannel) throws IOException {
        ByteBuffer byteBufferResponse=ByteBuffer.allocate(1024);
        byteBufferResponse.put(message.getBytes());
        byteBufferResponse.flip();
        socketChannel.write(byteBufferResponse);
    }
}

```

4.2 Teste de serveur avec un client Telnet

Après avoir développé notre serveur ChatServer, nous allons utiliser un client Telnet pour tester la connexion avec le serveur. Nous allons vérifier si les commandes envoyées par le client sont correctement reçues et traitées par le serveur comme montre le figure suivante :



4.3 Teste de serveur avec un client Java

En plus de la connexion avec Telnet, nous allons également mettre en place un client Java qui permettre aux utilisateurs d'interagir avec le serveur.

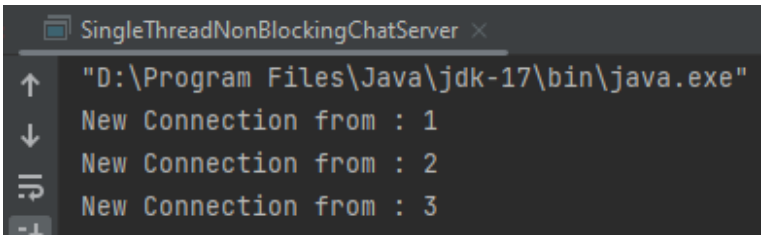
```java
package ma.enset.nonblocking.chat;

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 IOException, InterruptedException {
        SocketChannel socketChannel = SocketChannel.open(new InetSocketAddress("localhost",1111));
        Scanner scanner=new Scanner(System.in);
        new Thread(()->{
            while (true){
                ByteBuffer byteBuffer=ByteBuffer.allocate(1024);
                try {
                    socketChannel.read(byteBuffer);
                    String receivedMessage=new String(byteBuffer.array()).trim();
                    if(receivedMessage.length()>0){
                        System.out.println(receivedMessage);
                    }
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
        }).start();

        Thread.sleep(500);

        while(true){
            System.out.println("Your Message : ");
            String message = scanner.nextLine();
            ByteBuffer byteBuffer=ByteBuffer.allocate(1024);
            byteBuffer.put(message.getBytes());
            byteBuffer.flip();
            socketChannel.write(byteBuffer);

        }
    }
}
```

4.4 Teste de serveur avec un client Python

Nous allons également explorer la possibilité de créer un client Chat en utilisant un autre langage de programmation, en l'occurrence Python.

```python
import socket
import threading

listen = True

def listen_to_response(client_socket):
    while listen:
        response = client_socket.recv(1024).decode()
        print("Response : " + response.strip())

def start_client():
    host = "localhost"
    port = 1111
    client_socket = socket.socket()
    client_socket.connect((host, port))
    thread = threading.Thread(target= listen_to_response, args= (client_socket,))
    thread.start()

    request = ""
    while request.lower().strip() != 'exit':
        request = input("")
        client_socket.send(request.encode())
    client_socket.close()
    listen = False

if __name__ == '__main__':
    start_client()

```

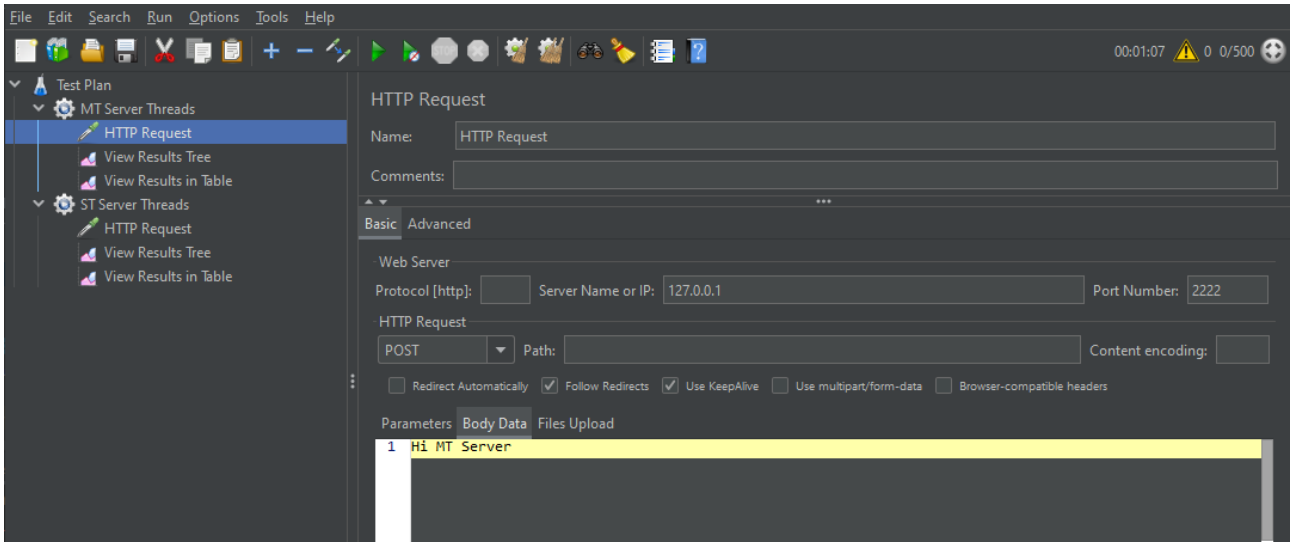
Voici les résultats de test avec les deux clients Java et Python :

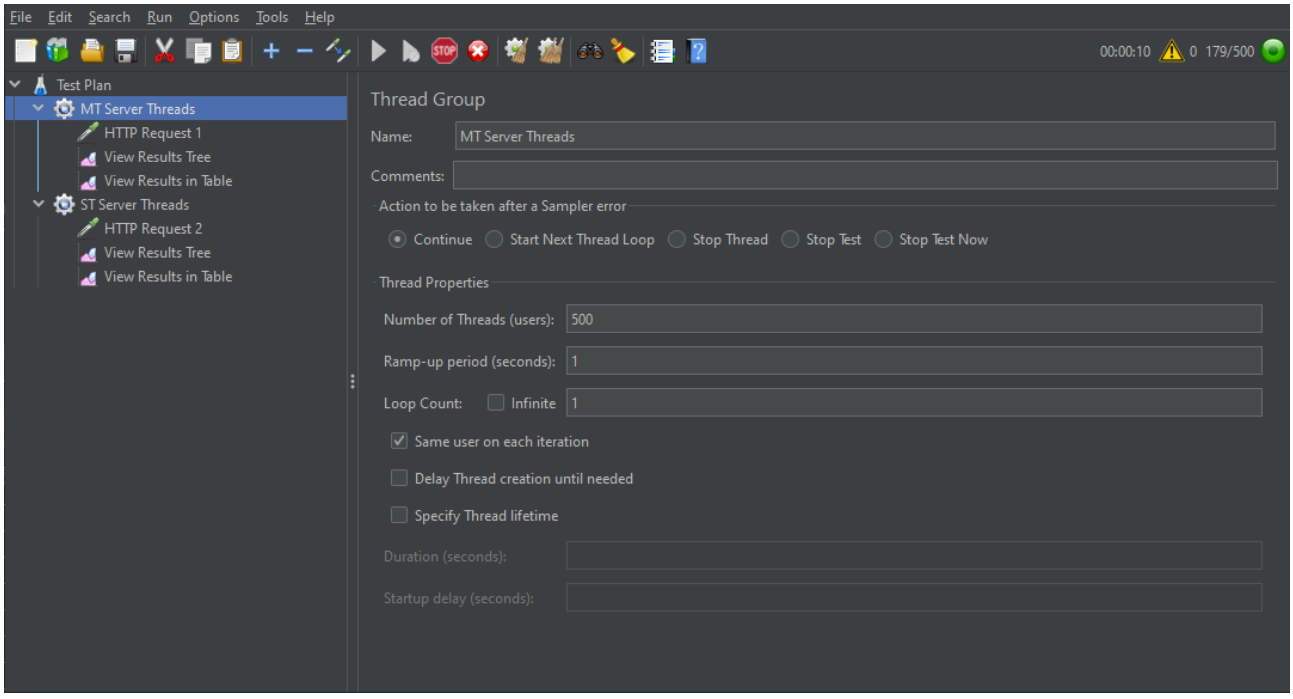


5 Teste des performances des serveurs avec JMeeter

Voici les résultats de test de server Multi Thread :

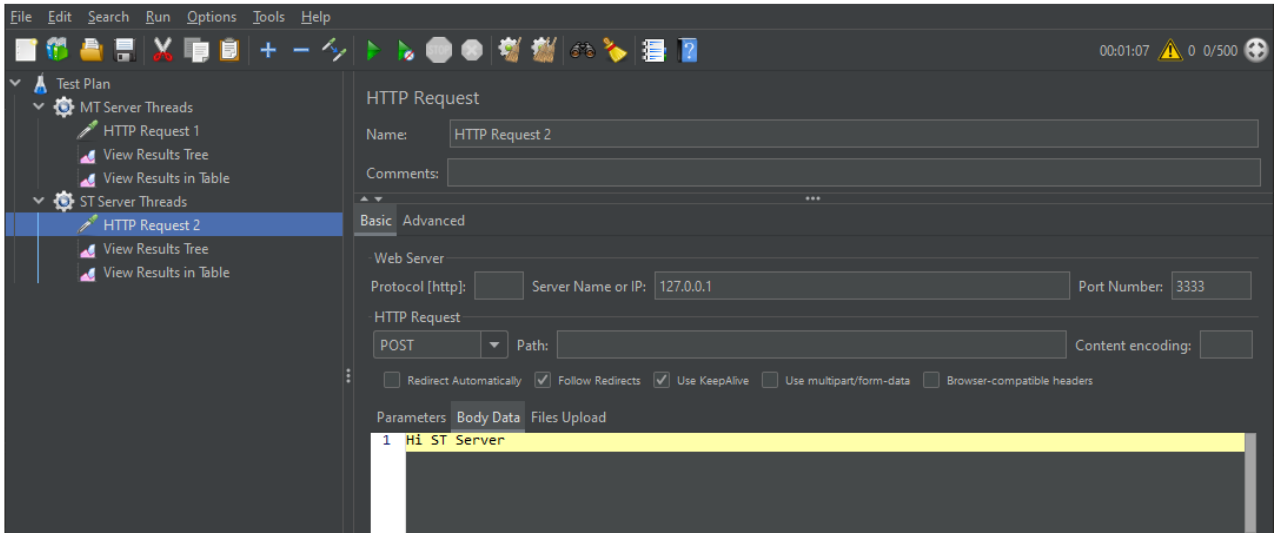
Server MT accepte 179 Thred sur 500

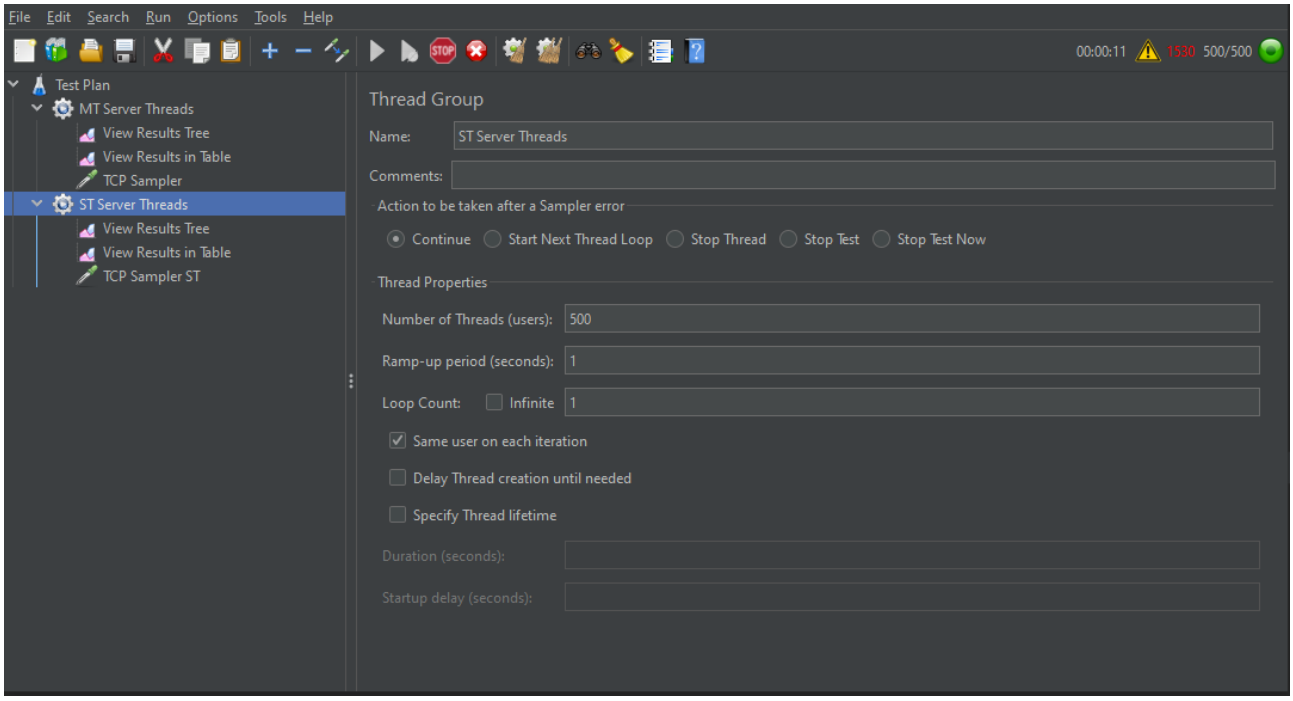




Voici les résultats de test de server single Thread :

Server ST accepte 500 Thred sur 500





6 Conclusion

En conclusion, nous avons développé une application client-serveur de chat en utilisant deux modèles différents : le modèle multi-threads blocking IO (java.io) et le modèle single-thread avec non-blocking IO (java.nio).

Le modèle multi-threads blocking IO offre une approche simple pour traiter plusieurs connexions simultanément. Cependant, cette approche peut être coûteuse en termes de ressources système si le nombre de clients augmente.

En revanche, le modèle single-thread avec non-blocking IO utilise une seule thread pour gérer toutes les connexions, ce qui réduit la consommation de ressources système.

Code source in Github :

github.com/el-moudni-hicham/ChatServer