BÁO CÁO THỰC HÀNH

Bài thực hành số 03: Lập trình socket

Môn học: Lập trình ứng dụng mạng

Lớp: NT109.O21.MMCL

THÀNH VIÊN THỰC HIỆN:

STT	Họ và tên	MSSV
1	Lê Hoàng Khánh	21522205

ĐÁNH GIÁ KHÁC:

Tổng thời gian thực hiện	1 tuần
Ý kiến *(nếu có)* + Khó khăn + Đề xuất, kiến nghị	Không có

MỤC LỤC

A. BÁO CÁO CHI TIẾT 3
1. TCP 3
a. TCPServer. 3
b. TCPClient 5
c. Quá trình thiết lập Socket TCP. 7
d. Kết quả log. 9
2. UDP 10
a. UDPServer. 10
b. UDPClient. 12
c. Khác biệt giữa UDP và TCP. 14
d. Các bước gửi Datagram giữa UDP Client và Server. 15
e. Kết quả log. 17
B. TÀI LIỆU THAM KHẢO 18

A. BÁO CÁO CHI TIẾT

1. TCP

a. TCPServer.

```
public class TCPServer {
    private final int port;  2 usages
    private final ExecutorService executorService;  3 usages
    private final Log log;  4 usages

    public TCPServer(int port) {  1 usage
        this.port = port;
        this.executorService = Executors.newCachedThreadPool();
        log = new Log( configPath: "configs/config.txt");
    }

    public void startServer() throws IOException {  1 usage
        try (var serverSocket = new ServerSocket(port)) {
            log.writeLog(String.format("[TCPServer] Waiting for client on port %s...%n", serverSocket.getLocalPort()));

            while (true) {
                var clientSocket = serverSocket.accept();
                log.writeLog(String.format("[TCPServer] Just connected to %s%n", clientSocket.getRemoteSocketAddress()));
                executorService.submit(() -> handleClient(clientSocket));
            }
        }
    }

    public void handleClient(Socket clientSocket)  1 usage
    {
        try (var in = new DataInputStream(clientSocket.getInputStream());
             var out = new DataOutputStream(clientSocket.getOutputStream())) {
            log.writeLog(String.format("[TCPServer] %s", in.readUTF());
            out.writeUTF(String.format("Thank you for connecting to %s \nGoodbye!", clientSocket.getLocalSocketAddress()));
            close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public void close() {  1 usage
        executorService.shutdownNow();
    }

    public static void main(String[] args) throws IOException {
        int port = 9000;
        if (args.length > 0) {
            port = Integer.parseInt(args[0]);
        }

        TCPServer server = new TCPServer(port);
        server.startServer();
    }
}
```

Thuộc tính:

- port: Cổng (port) số nguyên mà server lắng nghe.
- executorService: Đối tượng để quản lý các luồng (thread) làm nhiệm vụ xử lý kết nối.
- log: Đối tượng của lớp Log để ghi log

Constructor:

- Khởi tạo server, thiết lập cổng port và tạo một ExecutorService kiểu CachedThreadPool (có thể tái dùng luồng nếu không bận). Tạo đối tượng Log để ghi log với file cấu hình "configs/configTCPServer.txt".

Method startServer():

- Tạo ra một ServerSocket gắn với cổng được chỉ định (port).
- Ghi log thông báo server đang chờ kết nối.
- Vòng lặp chính: liên tục chấp nhận (accept()) các kết nối từ client.
    - Mỗi lần có kết nối:
    - Ghi log thông báo kết nối tới.
    - Dùng executorService để tạo luồng mới xử lý client này.

Method handleClient(Socket clientSocket):

- Tạo các luồng DataInputStream và DataOutputStream trên kênh kết nối với client.
- Đọc dữ liệu được gửi từ client.
- Gửi phản hồi thông báo kết nối thành công và lời tạm biệt.
- Đóng các nguồn dữ liệu bằng try-with-resources

Method close():

- Tắt (shutdown) executorService để hủy các luồng.

Main:

- Điểm khởi tạo chạy chương trình (entry point).
- Xử lý tham số dòng lệnh (nếu có) để xác định cổng, mặc định là cổng 9000.
- Tạo một đối tượng TCPServer và bắt đầu chạy bằng cách gọi startServer().

b. TCPClient

```java
public class TCPClient {
    private Log log;  4 usages
    private final String serverAddress;  3 usages
    private final int port;  3 usages

    public TCPClient(String serverAddress, int port) {  1 usage
        this.serverAddress = serverAddress;
        this.port = port;
        log = new Log( configPath: "configs/config.txt");
    }

    public void connect() {  1 usage
        try
        {
            log.writeLog(String.format("[TCPClient] Connecting to %s on port %s%n", serverAddress, port));
            var client = new Socket(serverAddress, port);
            log.writeLog(String.format("[TCPClient] Just connected to %s", client.getRemoteSocketAddress()));
            var out = new DataOutputStream(client.getOutputStream());
            var in = new DataInputStream(client.getInputStream());

            out.writeUTF(String.format("Hello from %s", client.getLocalSocketAddress()));
            log.writeLog(String.format("[TCPClient] Server says %s", in.readUTF()));
            client.close();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static void main(String[] args) {
        String host = "localhost";
        int port = 9000;
        var client = new TCPClient(host, port);
        client.connect();
    }
}
```

Constructor:

- Khởi tạo các thuộc tính: địa chỉ máy chủ (serverAddress), cổng (port) và đối tượng ghi log (Log).

Method connect():

- Tạo thông báo log về sự kiện đang chuẩn bị kết nối đến máy chủ, ghi lại địa chỉ và cổng.
- Sử dụng phương thức Socket(serverAddress, port) để thiết lập kết nối đến máy chủ.
- Ghi log khi kết nối vừa được thiết lập.
- Tạo các luồng DataOutputStream và DataInputStream trên socket.
- Gửi tin nhắn chào đến máy chủ.
- Đọc phản hồi từ máy chủ và ghi lại nó vào log.
- Đóng kết nối.
- Xử lý ngoại lệ tiềm ẩn.

main():

- Điểm khởi đầu của chương trình.
- Thiết lập địa chỉ máy chủ (host) và cổng (port).
- Khởi tạo TCPClient và gọi phương thức connect().

c. Quá trình thiết lập Socket TCP.

TCP Server:

Tạo ServerSocket: Sử dụng lớp java.net.ServerSocket để tạo một socket server, đồng thời ràng buộc nó với một địa chỉ IP và cổng cụ thể. Ví dụ:

```
try (ServerSocket serverSocket = new ServerSocket(9000)) {
    // Chờ kết nối từ client
} catch (IOException e) {
    e.printStackTrace();
}
```

Chấp nhận kết nối: Gọi phương thức accept() trên ServerSocket để chờ và chấp nhận kết nối từ client. Khi một client kết nối, phương thức này sẽ trả về một đối tượng Socket đại diện cho kết nối đó.

```
Socket clientSocket = serverSocket.accept();
```

Xử lý kết nối: Sử dụng đối tượng Socket để trao đổi dữ liệu với client. Ví dụ:

```
try (DataInputStream in = new
DataInputStream(clientSocket.getInputStream());
     DataOutputStream out = new
DataOutputStream(clientSocket.getOutputStream())) {
    // Đọc dữ liệu từ client
    String messageFromClient = in.readUTF();
    System.out.println("Client says: " + messageFromClient);

    // Gửi dữ liệu đến client
    String messageToServer = "Hello from server!";
    out.writeUTF(messageToServer);
} catch (IOException e) {
    e.printStackTrace();
}
```

Đóng kết nối: Gọi phương thức close() trên đối tượng Socket để đóng kết nối khi hoàn tất việc xử lý.

```
clientSocket.close();
```

TCP Client

Tạo Socket: Sử dụng lớp java.net.Socket để tạo một socket client, đồng thời kết nối đến địa chỉ IP và cổng của server đã được biết. Ví dụ:

```
try (Socket socket = new Socket("localhost", 9000)) {
    // Giao tiếp với server
} catch (IOException e) {
    e.printStackTrace();
}
```

Gửi dữ liệu đến server: Sử dụng đối tượng Socket để gửi dữ liệu đến server. Ví dụ:

```
try (DataOutputStream out = new
DataOutputStream(socket.getOutputStream());
     DataInputStream in = new DataInputStream(socket.getInputStream())) {
    // Gửi dữ liệu đến server
    String messageToServer = "Hello from client!";
    out.writeUTF(messageToServer);

    // Đọc dữ liệu từ server
    String messageFromServer = in.readUTF();
    System.out.println("Server says: " + messageFromServer);
} catch (IOException e) {
    e.printStackTrace();
}
```

Đóng kết nối: Gọi phương thức close() trên đối tượng Socket để đóng kết nối khi hoàn tất việc xử lý.

```
socket.close();
```

d. Kết quả log.

TCPServer:

```
15-04-2024 21:40:15 - [TCPServer] Waiting for client on port 9000...


15-04-2024 21:40:33 - [TCPServer] Just connected to /127.0.0.1:34990


15-04-2024 21:40:33 - [TCPServer] Hello from /127.0.0.1:34990
```

TCPClient:

```
15-04-2024 21:40:33 - [TCPClient] Connecting to localhost on port 9000


15-04-2024 21:40:33 - [TCPClient] Just connected to localhost/127.0.0.1:9000

15-04-2024 21:40:33 - [TCPClient] Server says Thank you for connecting to /127.0.0.1:9000
Goodbye!
```

2. UDP

a. UDPServer.

```java
public class UDPServer {
    private final DatagramSocket socket;
    private final ExecutorService executorService;
    private InetAddress clientAddress;
    private int clientPort;
    private Log log;

    public UDPServer(int port) throws SocketException {
        socket = new DatagramSocket(port);
        executorService = Executors.newSingleThreadExecutor();
        log = new Log("configs/config.txt");
    }

    public void start() {
        log.writeLog(String.format("[UDPServer] Server started on port %s", socket.getLocalPort()));
        executorService.execute(this::receiveMessages);

        try (Scanner scanner = new Scanner(System.in)) {
            while (true) {
                String message = scanner.nextLine();
                sendMessage(message);
            }
        }
    }

    private void receiveMessages() {
        while (true) {
            try {
                byte[] buf = new byte[1024];
                var packet = new DatagramPacket(buf, buf.length);
                socket.receive(packet);

                clientAddress = packet.getAddress();
                clientPort = packet.getPort();
                var received = new String(packet.getData(), 0, packet.getLength());
                log.writeLog(String.format("[UDPServer] Client: " + received));
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }

    private void sendMessage(String message) {
        if (clientAddress == null) {
            return;
        }
        byte[] buf = message.getBytes();
        DatagramPacket packet = new DatagramPacket(buf, buf.length, clientAddress, clientPort);
        try {
            socket.send(packet);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static void main(String[] args) throws SocketException {
        UDPServer server = new UDPServer(9000);
        server.start();
    }
}
```

Constructor:

- Tạo một socket UDP (DatagramSocket) và liên kết socket với cổng được chỉ định.
- Khởi tạo thread pool (ExecutorService) để xử lý các gói tin đến.
- Tạo đối tượng ghi log (Log).

start():

- Ghi log để thông báo máy chủ đã khởi động
- Dùng executorService để thực thi phương thức receiveMessages (quá trình nhận tin nhắn diễn ra ở một thread riêng).
- Sử dụng vòng lặp và Scanner để đọc đầu vào từ người dùng, gọi sendMessage để gửi tin nhắn đi.

receiveMessages():

- Vòng lặp vô hạn để lắng nghe các gói tin đến:
    - Tạo một buffer để lưu dữ liệu.
    - Tạo một DatagramPacket để nhận dữ liệu.
    - Sử dụng phương thức socket.receive() để chờ và nhận một gói tin.
    - Ghi lại địa chỉ/port của client và nội dung gói tin vào file log.

sendMessage(String message):

- Kiểm tra clientAddress để đảm bảo đã có client kết nối trước đó.
- Chuyển đổi tin nhắn thành mảng byte.
- Tạo một DatagramPacket chứa dữ liệu, địa chỉ client, và port.
- Gửi DatagramPacket thông qua socket.send().
- Xử lý ngoại lệ nếu có.

main():

- Tạo một thể hiện UDPServer và gọi phương thức start().

b. UDPClient.

```java
package io.github.tuilakhanh.Lab02.UDP;

import io.github.tuilakhanh.log.Log;

import java.io.IOException;
import java.net.*;
import java.util.Scanner;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class UDPClient {
    private final InetAddress serverAddress;
    private final int serverPort;
    private final DatagramSocket socket;
    private final ExecutorService executorService;
    private final Log log;

    public UDPClient(String serverAddress, int port) throws UnknownHostException, SocketException {
        this.serverAddress = InetAddress.getByName(serverAddress);
        this.serverPort = port;
        this.socket = new DatagramSocket(5001);
        this.executorService = Executors.newCachedThreadPool();
        log = new Log("configs/config.txt");
    }

    public void connect() {
        log.writeLog(String.format("[UDPClient] Connected to server on port %s", serverPort));
        executorService.execute(this::receiveMessages);

        try (Scanner scanner = new Scanner(System.in)) {
            while (true) {
                String message = scanner.nextLine();
                sendMessages(message);
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    private void receiveMessages() {
        while (true) {
            try {
                var buf = new byte[1024];
                var packet = new DatagramPacket(buf, buf.length);
                socket.receive(packet);

                var received = new String(packet.getData(), 0, packet.getLength());
                log.writeLog(String.format("[UDPClient] Server: %s", received));
            } catch (IOException e) {
                log.writeLog(String.format("[UDPClient] Error receiving message: %s", e.getMessage()));
                break;
            }
        }
    }

    private void sendMessages(String message) {
        try {
            var buf = message.getBytes();
            var packet = new DatagramPacket(buf, buf.length, serverAddress, serverPort);
            socket.send(packet);
        } catch (IOException e) {
            log.writeLog(String.format("[UDPClient] Error receiving message: %s", e.getMessage()));
        }
    }

    public static void main(String[] args) throws IOException {
        var client = new UDPClient("localhost", 9000);
        client.connect();
    }
}
```

Constructor:

- Lấy ra đối tượng InetAddress đại diện cho địa chỉ máy chủ.
- Khởi tạo socket UDP (DatagramSocket) và gán port 5001 để client tự sử dụng.
- Khởi tạo thread pool (ExecutorService) để xử lý các gói tin đến.
- Tạo đối tượng ghi log (Log).

connect():

- Ghi log để thông báo kết nối tới server đã được tạo.
- Tạo một thread bằng cách gọi executorService.execute() để thực thi phương thức receiveMessages (quá trình nhận tin diễn ra ở một thread riêng).
- Sử dụng vòng lặp và Scanner để đọc đầu vào từ người dùng, gọi sendMessages để gửi tin nhắn đi.

receiveMessages():

- Vòng lặp vô hạn để lắng nghe các gói tin đến:
- Tạo một buffer để lưu dữ liệu.
- Tạo một DatagramPacket để nhận dữ liệu.
- Sử dụng phương thức socket.receive() để chờ và nhận một gói tin.
- Ghi lại nội dung gói tin vào file log.
- Xử lý ngoại lệ nếu có.

sendMessages(String message):

- Chuyển đổi tin nhắn thành mảng byte.
- Tạo một DatagramPacket chứa dữ liệu, địa chỉ server, và port.
- Gửi DatagramPacket thông qua socket.send().
- Xử lý ngoại lệ nếu có.

main():

- Tạo một thể hiện UDPClient và gọi phương thức connect().

c. Khác biệt giữa UDP và TCP.

- UDP là giao thức connectionless: Client không cần thiết lập một kết nối rõ ràng với server, các gói tin được gửi đến server mà không cần xác nhận đồng ý trước.
- Không đảm bảo thứ tự gói tin và gói tin có thể thất lạc: Server không có cách nào để đảm bảo thứ tự các gói tin đã nhận, hoặc để xác nhận đã nhận được gói tin; do đó gói tin có thể đến theo thứ tự khác so với khi gửi hoặc bị mất mát.
- Không đảm bảo thứ tự và tính tin cậy của gói tin: UDP không đảm bảo gói tin client gửi tới server theo thứ tự, server cũng không có cơ chế xác nhận đã nhận được gói tin. Điều này dẫn đến khả năng gói tin bị thất lạc.

d. Các bước gửi Datagram giữa UDP Client và Server.

UDPClient:

Bước 1: Tạo UDPClient và DatagramSocket

Khởi tạo đối tượng UDPClient với địa chỉ IP và cổng server.

Tạo socket UDP (DatagramSocket) để gửi và nhận datagram.

```
UDPClient client = new UDPClient("localhost", 9000);
DatagramSocket socket = client.getSocket();
```

Bước 2: Chuẩn bị dữ liệu cho Datagram

Chuyển đổi tin nhắn cần gửi thành chuỗi byte.

```
String message = "Hello from UDPClient!";
byte[] data = message.getBytes();
```

Bước 3: Tạo DatagramPacket và gửi Datagram

Tạo DatagramPacket với dữ liệu, địa chỉ server và cổng.

Gửi DatagramPacket đến server bằng phương thức socket.send().

```
DatagramPacket packet = new DatagramPacket(data, data.length,
client.getServerAddress(), client.getServerPort());
socket.send(packet);
```

UDPServer:

Bước 1: Chờ nhận Datagram

Tạo buffer để lưu trữ dữ liệu nhận được.

Tạo DatagramPacket để nhận dữ liệu.

Sử dụng phương thức socket.receive() để chờ và nhận một datagram.

```
byte[] buf = new byte[1024];
DatagramPacket packet = new DatagramPacket(buf, buf.length);
socket.receive(packet);
```

Bước 2: Xử lý Datagram nhận được

Lấy ra dữ liệu từ DatagramPacket.

Chuyển đổi dữ liệu byte thành chuỗi.

Ghi log nội dung tin nhắn nhận được.

```
String received = new String(packet.getData(), 0, packet.getLength());
log.writeLog("[UDPServer] Client: " + received);
```

Bước 3: Gửi Datagram phản hồi (tùy chọn)

Tạo DatagramPacket với dữ liệu phản hồi, địa chỉ client và cổng.

Gửi DatagramPacket phản hồi đến client bằng phương thức socket.send().

String response = "Hello from UDPServer!";

```
byte[] responseData = response.getBytes();
DatagramPacket responsePacket = new DatagramPacket(responseData,
responseData.length, packet.getAddress(), packet.getPort());
socket.send(responsePacket);
```

e. Kết quả log.

UDPClient:

```
16-04-2024 04:16:22 - [UDPClient] Connected to server on port 9000

Hello
Iam Client
16-04-2024 04:16:40 - [UDPClient] Server: Hello

16-04-2024 04:16:46 - [UDPClient] Server: Iam Server

16-04-2024 04:16:49 - [UDPClient] Server: test 12345

test tes
```

UDPServer:

```
16-04-2024 04:16:12 - [UDPServer] Server started on port 9000

16-04-2024 04:16:27 - [UDPServer] Client: Hello

16-04-2024 04:16:33 - [UDPServer] Client: Iam Client

Hello
Iam Server
test 12345
16-04-2024 04:16:53 - [UDPServer] Client: test tes
```

B. TÀI LIỆU THAM KHẢO