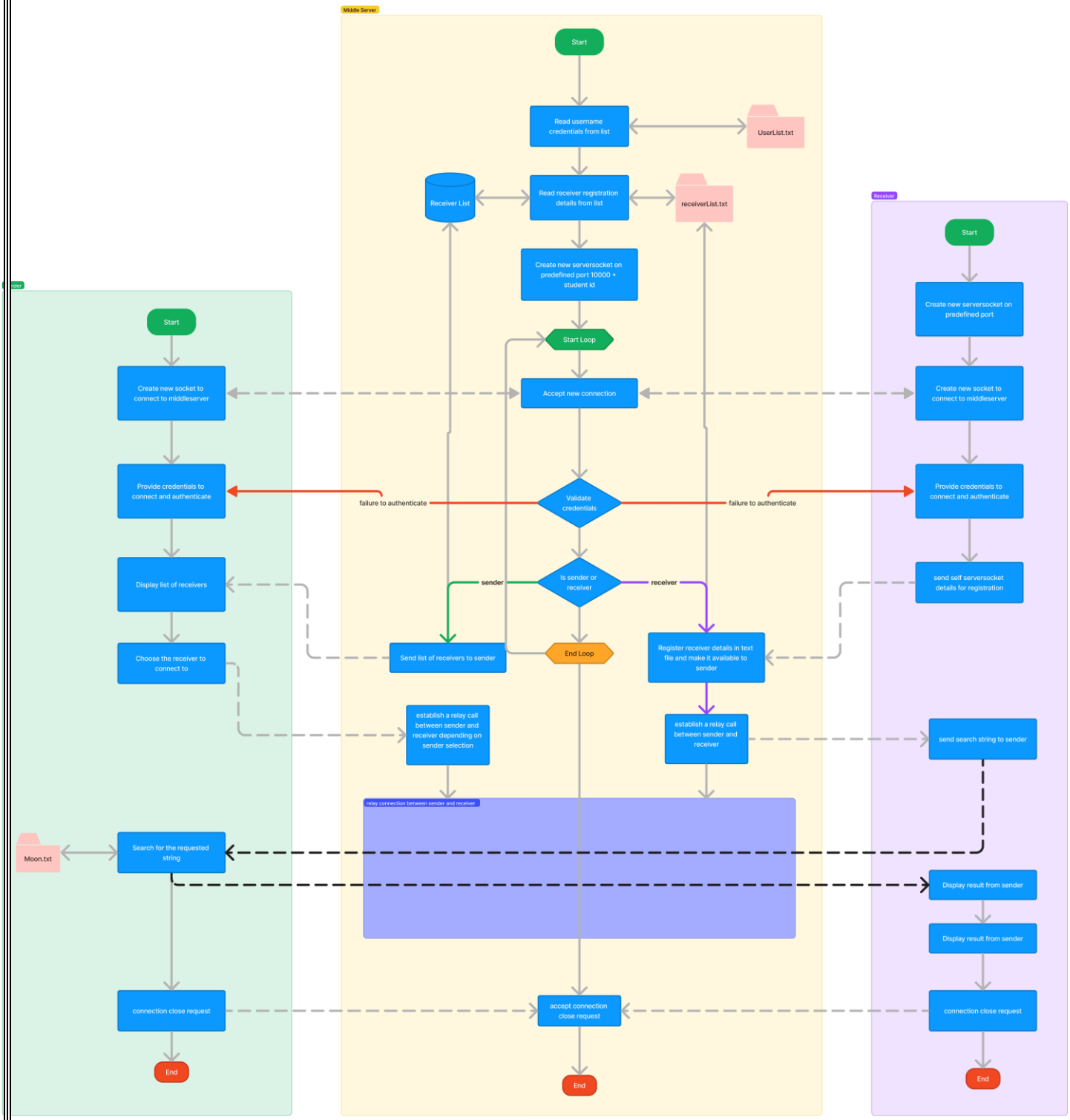
Advanced Operating System
Project-1

Submission by Kundanapreethi Surepally
(2249580)

Table of Contents


Middle Server
Start
Read username credentials from list
UserList.txt
Receiver List
Read receiver registration details from list
receiverList.txt
Create new serversocket on predefined port 10000 + student id
Start Loop
Accept new connection
Validate credentials
failure to authenticate
failure to authenticate
Is sender or receiver
sender
receiver
Send list of receivers to sender
End Loop
Register receiver details in text file and make it available to sender
establish a relay call between sender and receiver depending on sender selection
establish a relay call between sender and receiver
relay connection between sender and receiver
accept connection close request
End
Receiver
Start
Create new serversocket on predefined port
Create new socket to connect to middleserver
Provide credentials to connect and authenticate
send self serversocket details for registration
send search string to sender
Display result from sender
Display result from sender
connection close request
End
Start
Create new socket to connect to middleserver
Provide credentials to connect and authenticate
Display list of receivers
Choose the receiver to connect to
Moon.txt
Search for the requested string
connection close request
End

Objective: This project aims to learn TCP iterative client-server interaction using a socket interface in Java programming language.

Terminology Used:

TCP Client- Server:

A TCP server listens on a well-known port (or IP address and port pair) and accepts connections from TCP clients. A TCP client initiates a connection request to a TCP server in order to set up a connection with the server. A real TCP server can accept multiple connections on a socket.

Socket:

A socket is one endpoint of a two-way communication link between two programs running on the network.

Brokerage/Relay server:

This a middleware, which allows program calls to be made from one computer to another via a computer network.

Design Ideology:

- In this Project, we are required to make a communication between sender and receiver through the relay server which acts like a brokerage server.

- The relay server or the brokerage server creates sockets for each new client request. Here according to the requirement, a port number above 10,000 with UHCL id included needs to be considered.

- The client/ sender requests the relay server in order to communicate with the receiver via relay server.

- The relay server creates a socket for this request from the client and establishes a new connection. A connection with the corresponding server/receiver is also established.

- Proper authentication is required to make the connection.

- The receiver registers itself with the relay server to connect with the sender. The details of the receiver are sent in a text file.

- The relay server sends a list of available receivers to the sender. The sender chooses the receiver which establishes a relay call between the sender and receiver through which the communication is going to take place.

- After the connections between the sender, relay server and the receiver are connected successfully, the sender sends characters taken through user input to the relay server and waits for reply.

- The receiver consists of a story/text within the text file, the character/sub string sent from sender is fetched through simple algorithm to measure longest substring along with the length of the sub-string.

- This result will be sent to the sender from receiver through relay server.

- Example:
    - Receiver text file:
        - We are University of Houston-Clear Lake, and we have been opening doors for accessible education since 1974. As part of the University of Houston System, we offer more than 90 graduate and undergraduate programs online or face-to-face at locations in Clear Lake, Pearland and the Texas Medical Center.
    - Sender:
        - Request for the word **"University".**
    - Receiver replies with – University word occurrence, which is 2 in the above case, also length of the word which is 10.
- After the requesting part is done the connection needs to be closed if it's no longer being used.
- Here in TCP, we use Handshake mechanism: It is a three-way handshake method to establish a reliable connection.
    - Step 1: Client wants to establish a connection with server, so request for the connection.
    - Step 2: Server responses to the client about acceptance of the connection.
    - Step 3: A reliable connection is established to communicate with each other.

How to Execute:

To execute just run the MainClass.java file inside src folder.

```
> javac MainClass.java && java MainClass 19580
```

Expected output:

```
Select the mode:
1. Sender
2. Receiver
3. Middle Server
```

```
3
Running in Middle Server mode
Server socket [192.168.1.113:19580]
Press 0 Exit
1 Accept more connections
2 Print Users
3 Print Senders
```

This will start middle server in 19580 port

```
> javac MainClass.java && java MainClass 9096
```

```
Select the mode:
1. Sender
2. Receiver
3. Middle Server
1
Running in Sender mode
Enter the ip address:port of the middle server: 192.168.1.113:19580
```

This will start the program in sender mode, and it will ask for middle server ip address. Once after entering this detail, you'll have to press 1 in Middle server to accept this connection request.

```
1 Accept more connections
2 Print Users
3 Print Senders
1
Accepting more connections now...
A new client is connected : /192.168.1.113:64090
Assigning new thread for this client
Server socket [192.168.1.113:19580]
Press 0 Exit
1 Accept more connections
2 Print Users
3 Print Senders
```

Now, Sender will be prompted to enter login credentials:

```
Enter the ip address:port of the middle server: 192.168.1.113:19580
Waiting for Middle server to accept connection...
Enter credentials to login username/password: s/p
Login successful, Welcome sender!
You are registered in server.
Press Exit anytime to close this sender:
```

> *javac MainClass.java && java MainClass 9090*

Now for receiver similar process will be followed to establish connection:

```
Select the mode:
1. Sender
2. Receiver
3. Middle Server
2
Running in Receiver mode
Enter the ip address:port of the middle server: 192.168.1.113:19580
Enter credentials to login username/password: r/p
Login successful, Welcome receiver!
1. Get list of senders
2. Exit
Send to server:
```

If receiver presses 1 he'll get list of sends to connect to:

```
1. Get list of senders
2. Exit
Send to server: 1
192.168.1.113:9096
Choose the sender you wanna connect to: 0
```

Receiver can choose a sender by pressing a number representing the position of the sender starting from 0. So here 1 sender is at 1st position, so we'll select 0.

```
Preparing to connect to 192.168.1.113:9096.
Enter string to search in sender 192.168.1.113:9096 or Exit:
```

This prompt means that we have successfully created RMI connection with the sender. Type any string you want to search in lowercase

```
Enter string to search in sender 192.168.1.113:9096 or Exit: timmy
Match found at position 75 on line 1
Match found at position 43 on line 2
Match found at position 1 on line 5
Match found at position 15 on line 9
Match found at position 68 on line 9
Match found at position 75 on line 9
Match found at position 62 on line 10
Match found at position 13 on line 14
Enter string to search in sender 192.168.1.113:9096 or Exit:
```

We can type Exit at any time to go back to sender selection menu.

```
Enter string to search in sender 192.168.1.113:9096 or Exit: Exit
1. Get list of senders
2. Exit
Send to server:
```

Classes:

MainClass

This class acts as a controller for three different modes: Sender, Receiver, and Middle Server. It prompts the user to select a mode and based on the user's input, it creates an instance of the corresponding class and starts its execution.

It performs following operations:

1. The program imports the necessary classes, including Scanner for user input, and the classes for Sender, Receiver, and MiddleServer.

2. The MainClass class contains the main method, which serves as the entry point of the program.

3. Inside the main method, a Scanner object scanner is created to read user input.

4. The program displays a menu of options for the user to select a mode: Sender, Receiver, or Middle Server.

5. The user's input is read using scanner.nextInt(), and stored in the variable mode.

6. After reading the input, scanner.nextLine() is called to consume the newline character.

7. The program uses a switch statement to determine the selected mode based on the value of mode.

8. If the mode is 1, it creates an instance of Sender and passes the value of args[0] (presumably a command-line argument) to the constructor. Then it calls the startExecution() method on the sender object.

9. If the mode is 2, it creates an instance of Receiver and passes the value of args[0] to the constructor. Then it calls the startExecution() method on the receiver object.

10. If the mode is 3, it creates an instance of MiddleServer and passes the value of args[0] to the constructor. Then it calls the startExecution() method on the middleServer object.

11. If none of the above cases match (invalid mode), it displays an error message.

12. Finally, the scanner is closed to release system resources.

MiddleServer

The MiddleServer class extends the common.Parent class and represents a middle server in a network communication system. It has the following attributes:

userList: A list of User objects that represents the users in the system.
senderList: A set of strings that stores the sender information.
serverSocket: An instance of ServerSocket that represents the server socket for accepting client connections.
The class has the following methods:

MiddleServer(String startingPort): The constructor that takes a starting port as a parameter. It initializes the userList as an empty ArrayList, senderList as an empty TreeSet, and calls the readUsers() and startServerSocket() methods.

readUsers(): A private method that reads user information from a file (MyConst.USERS_FILE) and populates the userList with User objects.

printUsers(): A private method that prints the information of users in the userList in a formatted manner.

printSenders(): A private method that prints the receiver IP and port information stored in the senderList in a formatted manner.

startServerSocket(): A private method that creates a ServerSocket object using the provided starting port.

startExecution(): A public method that represents the main execution logic of the middle server. It continuously accepts client connections, reads user input to perform different actions (such as accepting more connections, printing users, or printing senders), and creates a new thread (MiddleServerThread) to handle each client connection.

getUser(String username, String password): A public method that takes a username and password as parameters and checks if there is a matching user in the userList. If a match is found, it returns the corresponding User object; otherwise, it returns null.

MiddleServerThread

The MiddleServerThread class extends the SocketThread class and implements the Runnable interface. It represents a thread that handles client connections in the middle server mode of the network communication system.

Here's an overview of the class:

middleServer: A reference to the MiddleServer object that creates this thread.
receiverStr: A string variable to store the list of sender connections.
The class has the following methods:

MiddleServerThread(Socket s, DataInputStream dis, DataOutputStream dos, MiddleServer middleServer): The constructor that takes a Socket object, DataInputStream, DataOutputStream, and a MiddleServer object. It calls the constructor of the superclass (SocketThread) and assigns the middleServer parameter to the corresponding attribute.

run(): The run method is the entry point of the thread. It handles the client connection logic. Inside the method:

1. The method prompts the client to enter their credentials (username/password) and checks if the middle server has a matching user.

2. If the login is successful, a welcome message is sent to the client.

3. If the user has the role "sender", the thread expects the sender to send their connection details, which are then added to the middleServer.senderList. A response is sent back to the sender indicating the success or failure of the registration.

4. If the user has the role "receiver", the thread expects the receiver to send a request for the list of senders. The thread responds by sending the receiverStr, which contains the list of sender connections.

Sender

The Sender class extends the common.Parent class and represents a sender in the network communication system. It has the following attribute:

registry: An instance of the Registry class from the RMI (Remote Method Invocation) framework.
The class has the following methods:

Sender(String startingPort): The constructor that takes a starting port as a parameter. It calls the constructor of the superclass (Parent) and initializes the registry attribute.

startExecution(): A public method that represents the main execution logic of the sender. It prints a message indicating that it is running in sender mode, and then calls the registerRMI() and connectToSocket() methods.

registerRMI(): A private method that registers the sender as an RMI object in the RMI registry. It creates a registry on the provided starting port, exports a PatternFinderRemote object as an RMI remote object, and binds it to the registry with a specified name (MyConst.REGISTRY_NAME).

connectToSocket(): A private method that establishes a connection with the middle server. It prompts the user to enter the IP address and port of the middle server, creates a socket connection with the middle server, and sets up input and output streams for communication.

- The method handles the login process by reading login prompts from the middle server and sending the user's credentials.
- After successful login, the method sends the connection details of the sender (IP address and starting port) to the middle server for self-registration.
- The method then enters a loop where it prompts the user for input. If the user enters "Exit", the connection is closed, and the loop is terminated.
- Finally, the method closes resources, including the socket connection, input and output streams, and unbinds the RMI object from the registry.

MatchAPattern

This MatchAPattern class provides a method called perform that performs pattern matching on a predefined text file. Here's a breakdown of what the class does:

The perform method takes a pattern (p_pattern) as input and returns a string containing the result of the pattern matching.

- It initializes a LinkedList called res to store the positions of the pattern matches.
- It creates a BufferedReader to read the predefined text file (specified by MyConst.MOON_FILE).
- It reads the file line by line and performs pattern matching on each line.
- For each line, it searches for the pattern (case-insensitive) using the indexOf method.
- If a match is found, it creates a LinkedList called item to store the position of the match (index + 1) and the line number (lineNumber).
- It adds the item to the res list.
- It continues searching for the pattern in the line until no more matches are found.
- After processing all the lines in the file, it returns the result of the pattern matching by calling the printPatternResult method with the res list as the parameter.
- The printPatternResult method takes a list of lists (resultFound) containing the positions and line numbers of the pattern matches and generates a formatted string representing the result. It iterates over the resultFound list, formats each match's position and line number into a string, and appends it to the res string. If no matches are found, it sets the res string to indicate that no matches were found.

Receiver

The Receiver class extends the common.Parent class and represents a receiver in the network communication system. It has the following attributes:

registry: An instance of the Registry class from the RMI (Remote Method Invocation) framework.
patternFinderRemote: A static attribute of type PatternFinderRemote used to access the remote methods provided by the sender.
The class has the following methods:

Receiver(String startingPort): The constructor that takes a starting port as a parameter. It calls the constructor of the superclass (Parent) and initializes the registry attribute.

startExecution(): A public method that represents the main execution logic of the receiver. It prints a message indicating that it is running in receiver mode and calls the connectToSocket() method.

connectToSocket(): A private method that establishes a connection with the middle server. It prompts the user to enter the IP address and port of the middle server, creates a socket connection with the middle server, and sets up input and output streams for communication.

- The method handles the login process by reading login prompts from the middle server and sending the user's credentials.
- After successful login, the method enters a loop where it prompts the user for input. If the user enters "1", the receiver requests the list of senders from the middle server and displays it. The user can then select a sender to connect to by entering the corresponding number.
- If the user enters "2", the connection is closed, and the loop is terminated.
- The method also handles the case where the user enters an invalid input.
- connectToRMI(String connectionURL): A public method that connects to the RMI registry of a specific sender. It takes the connection URL (IP address and port) of the sender as a parameter.
- The method uses the LocateRegistry class to get the RMI registry of the sender based on the provided connection URL.
- It then looks up the remote object with the name MyConst.REGISTRY_NAME in the registry, which corresponds to the PatternFinderRemote object exported by the sender.

- The method enters a loop where it prompts the user to enter a string to search in the sender's data. If the user enters "Exit", the loop is terminated. Otherwise, the method calls the findPattern method on the patternFinderRemote object and prints the result.