Principles of Computer System-2

Ale Anwesh(B22AI005), B Hitesh Shanmukha(B22AI013)

RemoteDesk: Seamless Remote Desktop Access and Control

Abstract

Operating system concepts such as **process management** and **inter-process communication** are utilized for user interaction orchestration, while **network protocols** and **socket programming** facilitate reliable data transmission between client and server.

1 Introduction

This report presents a remote desktop application developed using Java, which enables a client to remotely access and control a server's desktop. The application is a comprehensive demonstration of various operating system and computer networking concepts, including process communication, network protocols, remote procedure calls, synchronization, and multithreading. The code utilizes Java's socket programming to establish a connection between the client and server, allowing for bidirectional communication and data transfer.

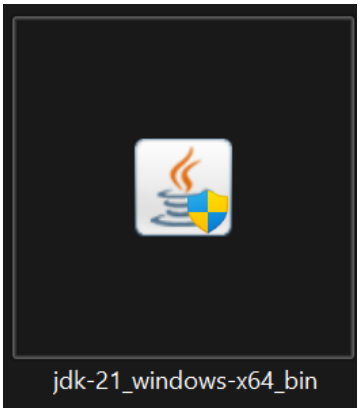
The remote desktop application also showcases the use of multithreading, where multiple threads are used to handle different tasks concurrently, such as screen transmission, user input processing, and authentication. This approach enables the application to provide a responsive and interactive user experience. Furthermore, the application demonstrates the use of synchronization mechanisms, such as locks and threads, to ensure that data is transmitted and processed correctly.

2 Prerequisites and Setup

2.1 Prerequisites

Java Runtime Environment (JRE):

- Ensure that you have Java Runtime Environment (JRE) installed on your system.
- You can download and install the latest version of JRE from the official Java website.



Operating System Compatibility:

- The JAR files are compatible with Windows, macOS, Linux, and other Unix-based operating systems.

- Ensure that your system meets the minimum requirements for running Java applications.

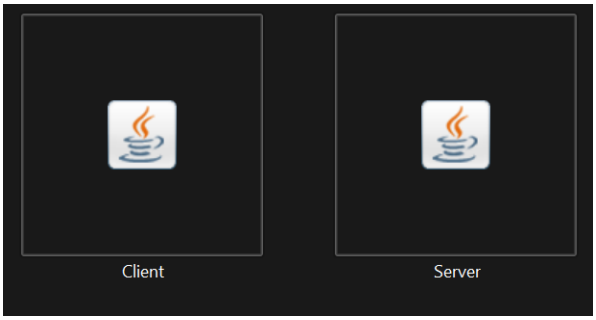
Network Connectivity:

- Both the client and server machines should be connected to the same network.
- Ensure that there are no firewall restrictions preventing communication between the client and server.

2.2 Execution

2.2.1 For Windows

1. **Download JAR Files:** Ensure you have downloaded the client.jar and server.jar files from the JAR folder in the repository.
2. **Open JAR Files:**Directly open the respective jar files that you need

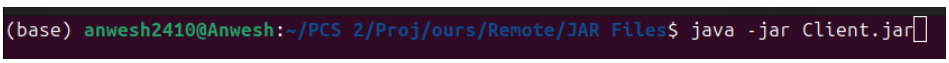


3. **Enter Server IP and Password:** On the client-side, enter the server's IP address and the password set on the server-side for authentication.
4. **Access Remote Desktop:** Once authenticated, the client will display the server's desktop, and you can start controlling it remotely.

2.2.2 For Unix-Based Systems (Linux, macOS, etc.):

1. **Download JAR Files:** Ensure you have downloaded the client.jar and server.jar files from the JAR folder in the repository.
2. **Server Execution:**
   - Open Terminal.
   - Navigate to the directory containing the server.jar file.
   - Execute the following command: "**java -jar server.jar**"

3. **Client Execution**
   - Open another Terminal window.
   - Navigate to the directory containing the client.jar file.
   - Execute the following command: "**java -jar client.jar**"



4. **Enter Server IP and Password:** On the client-side, enter the server's IP address and the password set on the server-side for authentication.
5. **Access Remote Desktop:** Once authenticated, the client will display the server's desktop, and you can start controlling it remotely.

3 Server's Overview

The server component of the Remote Desktop Application plays a crucial role in managing incoming client connections, authenticating incoming clients, capturing the server's screen, and transmitting screen updates to connected clients.

Here is an overview of the server's functionalities:

- **Connection Management:** The server listens for incoming connection requests from clients on a specified port.Upon receiving a connection request, the server creates a socket and accepts the connection, establishing a communication channel with the client.
- **Authentication:** After the connection is established, the server prompts the client to enter a password for authentication. The server verifies the password entered by the client to ensure secure access to the server's resources. Upon successful authentication, the server grants access to the client and proceeds with the screen sharing process.
- **Screen Sharing:** Once authenticated, the server captures the desktop screen using the Java Robot class. The captured screen is encoded into image data, which is then transmitted to the connected clients over the established TCP connection.The server continuously captures and transmits screen updates to ensure real-time screen sharing with minimal latency.

3.1 Code Explanation:

3.1.1 Main Class:

- **Purpose:** Initiates the server-side application and handles the setup of the password for client authentication.
- **Functionality:** Provides a graphical user interface (GUI) for setting the password required for client authentication.Listens for incoming client connections using a server socket.
- **Implementation:** Utilizes Java Swing library for GUI development. Implements action listeners to handle user input for password configuration.

3.1.2 InitConnection Class:

- **Purpose:** Manages incoming client connections and authentication procedures.
- **Functionality:**
  - Accepts incoming client connections using the server socket.
  - Authenticates clients by comparing passwords provided by clients with the configured password.
  - Sends screen dimensions to authenticated clients for screen sharing purposes.
- **Implementation:**
  - Utilizes Java's ServerSocket and Socket classes for socket communication.
  - Implements data streams (DataInputStream and DataOutputStream) for sending and receiving data between server and clients.

3.1.3 SendScreen Class:

- **Purpose:** Captures the server's screen and sends screen images to connected clients.
- **Functionality:**
  - Utilizes Java's Robot class to capture the server's screen as a BufferedImage.
  - Writes the captured screen images to the output stream associated with the client socket
  - Sends screen images at regular intervals to ensure real-time screen sharing.

- **Implementation:**
  - Uses Java's ImageIO library to write images to output streams.
  - Runs as a separate thread to ensure continuous screen capturing and sending operations.

3.1.4 ReceiveEvents Class:

- **Purpose:** Listens for user input events sent by clients and simulates those events on the server's desktop.
- **Functionality:**
  - Reads user input events (mouse and keyboard events) sent by clients from the input stream associated with the client socket.
  - Utilizes Java's Robot class to simulate user input events on the server's desktop.
- **Implementation:**
  - Runs as a separate thread to continuously listen for and process user input events from clients.
  - Implements event handlers to handle different types of user input events (mouse clicks, key presses, etc.).

4 Client's Overview

The client component of the Remote Desktop Application acts as the interface through which users connect to remote servers, authenticate themselves, and interact with the server's desktop interface. Below is an overview of the client's functionalities:

- **Connection Initialization:** The client initiates a connection to the server by providing the server's IP address and port number. Upon establishing the connection, the client creates a socket and connects to the server, establishing a communication channel for data exchange.
- **Authentication:** After establishing the connection, the client is prompted to enter a password for authentication. The client securely transmits the entered password to the server for verification. Upon successful authentication, the client gains access to the server's resources and proceeds to interact with the server's desktop interface.
- **User Interface:** The client provides a user-friendly interface that allows users to view the server's desktop interface and interact with it remotely. The interface may include features such as screen display, mouse control, keyboard input, and other interactive elements to facilitate remote desktop control.
- **Screen Display:** Upon successful authentication, the client receives screen updates from the server, displaying the server's desktop interface in real-time. The client continuously receives and renders screen updates to ensure a seamless viewing experience for users.

4.1 Code Explanation:

4.1.1 Main Class:

- **Purpose:** Initiates the client-side application and prompts the user for the server's IP address.
- **Functionality:**
  - Displays a dialog box for the user to input the server's IP address.
  - Initializes the connection to the server using the provided IP address.
- **Implementation:**
  - Utilizes Java Swing library for GUI development.
  - Establishes a socket connection to the server using Java's Socket class.

4.1.2 Authenticate Class:

- **Purpose:** Handles user authentication process by verifying the password entered by the user.
- **Functionality:**
  - Provides a graphical interface for the user to input a password.
  - Sends the entered password to the server for authentication.
  - Displays appropriate messages based on the authentication result.
- **Implementation:**
  - Implements action listeners to handle user input and authentication process.
  - Utilizes data streams (DataOutputStream and DataInputStream) for sending and receiving data to/from the server.

4.1.3 CreateFrame Class Class:

- **Purpose:** Constructs the main graphical user interface for screen sharing upon successful authentication.
- **Functionality:**
  - Creates a frame to display the server's desktop shared by the server.
  - Sets up a panel for rendering the remote desktop content.
  - Initializes threads for receiving screen updates from the server and sending user input events to the server.
- **Implementation:**
  - Utilizes Java Swing components for GUI construction.
  - Implements separate threads to handle screen updates and user input events concurrently.

4.1.4 SendEvents Class:

- **Purpose:** Sends user input events (mouse and keyboard events) to the server for remote control.
- **Functionality:**
  - Implements listeners for mouse and keyboard events on the client-side panel.
  - Sends event data to the server based on user interactions.
- **Implementation:**
  - Utilizes Java's MouseListener, MouseMotionListener, and KeyListener interfaces for event handling.
  - Sends event data to the server via the socket connection established during initialization.

5 Working and Demonstration of Client-Server Remote Desktop Application

5.1 Working Mechanism

5.1.1 Client-Side Workflow:

- The client-side application initiates by prompting the user to input the server's IP address.
- Upon receiving the IP address, the client establishes a socket connection to the server.
- The authentication window appears, prompting the user to input a password.

- The entered password is sent to the server for verification.
- If the password is validated, the server sends screen dimensions to the client, and a remote desktop window is displayed.
- The client can then interact with the remote desktop by sending mouse and keyboard events to the server.

5.1.2 Server-Side Workflow:

- The server-side application waits for incoming client connections using a server socket.
- Upon connection, the server prompts the client to enter a password for authentication.
- The server compares the entered password with the configured password for validation.
- If the password is authenticated, the server sends screen dimensions to the client and begins capturing and sending screen images.
- The server also listens for user input events sent by the client and simulates those events on the server's desktop.

This workflow ensures seamless communication and interaction between the client and server, enabling remote desktop access and control.

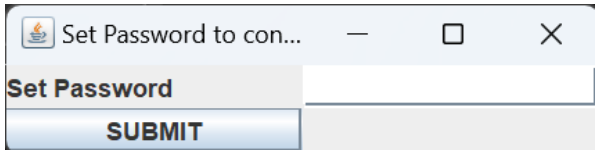
5.2 Demonstration

5.2.1 Preparation:

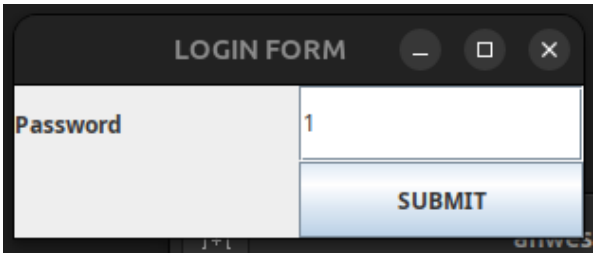
- Ensure that both the client and server applications are installed and configured properly on their respective machines.
- Verify network connectivity between the client and server machines.

5.2.2 Execution Steps:

- **Server-Side Setup:**
  - Launch the server-side application on the designated server machine.
  - Set the password for client authentication through the provided GUI interface.
  - Start the server socket to listen for incoming client connections.



- **Client-Side Setup:**
  - Launch the client-side application on the client machine.
  - Input the IP address of the server when prompted.
  - Enter the password for authentication in the provided dialog box.



- **Remote Desktop Interaction:**
  - Upon successful authentication, observe the appearance of the remote desktop window on the client's screen.
  - Interact with the remote desktop by moving the mouse cursor and typing on the keyboard.
  - Notice how the actions performed on the client's side reflect on the server's desktop in real-time.

Figure 1: Server's screen

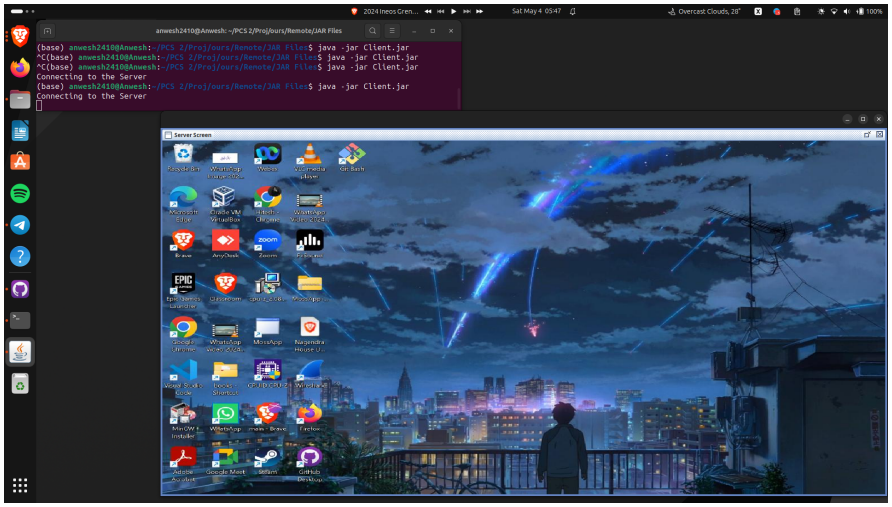


Figure 2: Client's screen

- **Termination:**
  - Close the client and server applications to terminate the connection and end the demonstration.

6 Conclusion

In conclusion, the Client-Server Remote Desktop Application, named "RemoteDesk," demonstrates the powerful capabilities of networked systems in facilitating remote access and control over desktop interfaces. Through the collaborative effort of both client and server components, users can securely connect to remote servers, authenticate themselves, and interact with the server's desktop interface in real-time.

This project exemplifies the seamless integration of various technologies, including socket programming, graphical user interface (GUI) development, and event handling, to create a robust and user-friendly remote desktop solution. By leveraging Java's Swing library for GUI development and network communication capabilities, the application provides a smooth and intuitive user experience for both clients and servers.

Furthermore, the project highlights the importance of security measures, such as password authentication, in ensuring secure access to remote resources. The implementation of authentication mechanisms adds an extra layer of protection, safeguarding sensitive data and preventing unauthorized access to the server's desktop interface.

Overall, "RemoteDesk" offers a reliable and efficient solution for remote desktop access and control, with potential applications in various domains, including remote technical support, collaborative work environments, and remote access to resources. With further refinement and optimization, this project has the potential to become a valuable tool in enhancing productivity and connectivity in networked environments.