

OPERATING SYSTEMS



National University
of computer and emerging sciences

PROJECT REPORT

“SECURE REMOTE DESKTOP SERVER”

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES KARACHI CAMPUS

Group Members:

Muhammad Ali 23K-0730

Ayan Hussain 23K-0608

Zohair Sarosh 23K-0558

Asad Ur Rehman 23K-0902

Submitted to: Engr Abdur Rehman

Introduction

This project demonstrates a server-client communication system in C, where the server provides several functionalities, including file upload, file download, and taking screenshots. The communication is secured by an authentication mechanism that verifies the client's credentials before granting access to these functionalities. The server utilizes threads to handle multiple client connections concurrently, and semaphores are used to control the number of concurrent file uploads and downloads.

Objective

The main objective of this project is to implement a secure, multi-threaded server-client interaction system where multiple clients can connect to the server simultaneously and utilize the following features:

- **Authentication:** Clients must authenticate with a username and password before using the system.
- **File Upload/Download:** Clients can upload and download files to/from the server.
- **Screenshot Capture:** Clients can request the server to take a screenshot and send it back.
- **Message Handling:** Clients can send messages to the server, and the server responds accordingly.

The system also ensures proper synchronization using semaphores and mutexes to manage concurrent file transfers and logging operations safely.

System Architecture

The system consists of two components:

1. **Server Side (server.c):** Manages client connections, authenticates clients, handles file uploads, downloads, and screenshot requests, and sends appropriate responses.
2. **Client Side (client.c):** Allows users to input commands to interact with the server. The client requests services like file upload, file download, and screenshots, while ensuring that the authentication is successful before proceeding with other operations.

Features:

- **Authentication:** Clients must provide a valid username and password, which are checked against a file (users.txt).
- **File Upload:** The client can upload files to the server.
- **File Download:** The client can request files from the server.
- **Screenshot:** The client can request the server to take a screenshot and send it back.
- **Message Handling:** The client can send and receive messages.

Multiple Client Connections and Synchronization

The server is designed to handle multiple client connections concurrently by utilizing multi-threading. For every new client connection, the server creates a separate thread dedicated to that client. This ensures that multiple clients can interact with the server at the same time without interfering with each other's communication or operations.

To manage shared resources and maintain system stability during concurrent access, synchronization mechanisms are employed:

- **Semaphore Synchronization:**
A semaphore (file_semaphore) is used to limit the number of concurrent file uploads and downloads. Only a maximum of two clients are allowed to perform upload or download operations simultaneously. This helps prevent server overload and ensures fair resource allocation among clients.
- **Mutex Synchronization:**
A mutex (log_mutex) is used to synchronize access to shared resources, such as server logs. This prevents race conditions where multiple threads might try to write to the console or log simultaneously, preserving the integrity of the output.

Through the combination of multi-threading and careful synchronization, the server achieves scalability, consistency, and robustness, allowing real-time interaction with multiple clients efficiently and safely.

Technical Implementation

Server-Side Implementation

1. **Server Initialization:**
 - The server listens to port 8080 and accepts client connections.

- Upon each client connection, the server starts a new thread to handle that client.

2. **Client Authentication:**

- Upon connection, the server prompts the client for a username and password.
- The authentication process checks the provided credentials against entries in a users.txt file.
- If authentication is successful, the client is allowed to perform further actions like uploading files or requesting a screenshot. Otherwise, the connection is terminated.

3. **File Handling:**

- **File Upload:** The server receives a file from the client, writes it to disk, and confirms the successful transfer.
- **File Download:** The server sends a requested file to the client.

4. **Screenshot Handling:**

- The server captures a screenshot using the gnome-screenshot tool and sends the image file to the client.

Client-Side Implementation

1. **Client Initialization:**

- The client connects to the server and begins by providing the username and password for authentication.

2. **Command Input:**

- After successful authentication, the client can input commands to upload files, download files, take screenshots, or send messages to the server.

3. **Authentication Process:**

- The client receives prompts to enter the username and password and sends these credentials to the server for verification.
- If authentication fails, the client is disconnected.

4. **File Upload/Download:**

- The client can upload files by sending them to the server and can download files by requesting them from the server.

5. Screenshot Request:

- The client can request the server to take a screenshot and receive it as a file.

Code Walkthrough

Server Code Explanation

```
int authenticate(const char* username, const char* password) { ... }
```

- The authenticate function checks the credentials against the users.txt file to verify the client's identity.
- If the username and password match a stored pair, authentication is successful; otherwise, the connection is terminated.

```
void* handleclient(void* arg) { ... }
```

- This function handles the client after successful authentication. It listens for commands from the client (e.g., upload, download, screenshot) and executes the appropriate actions.

```
void handle_upload(int clientsock, char* filename) { ... }
```

```
void handle_download(int clientsock, char* filename) { ... }
```

```
void handle_screenshot(int clientsock) { ... }
```

- These functions handle the specific operations of uploading files, downloading files, and taking screenshots.

Client Code Explanation

```
int init_client_socket() { ... }
```

- The init_client_socket function establishes a connection to the server.

```
void handle_upload(int clientfd, const char* filename) { ... }
```

- The handle_upload function sends a file from the client to the server by reading the file in binary mode and sending it over the socket.

```
void handle_screenshot(int clientfd) { ... }
```

- This function sends a screenshot request to the server and receives the screenshot image file in response.

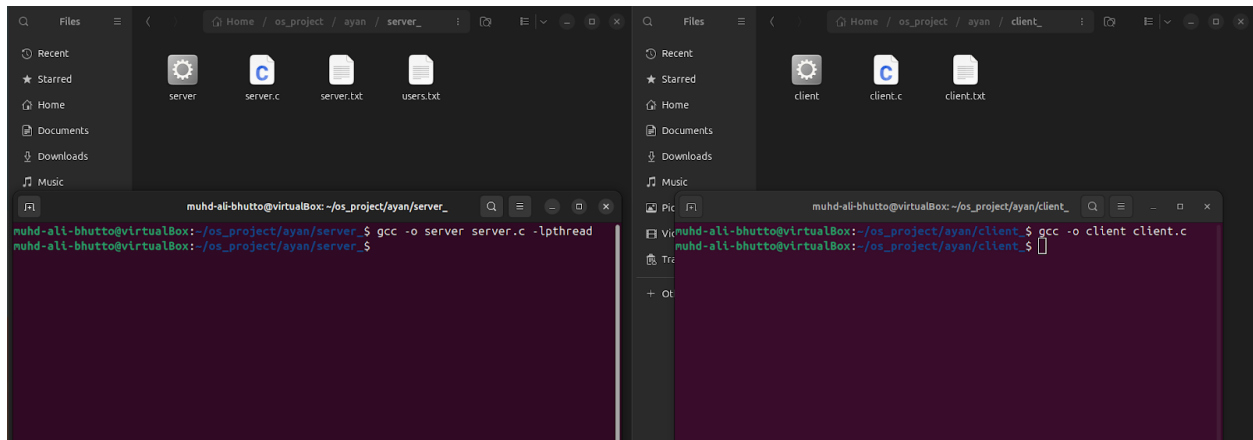
Authentication Flow:

- The client first receives prompts to enter a username and password.
- The server checks the credentials and either grants access to the system or disconnects the client based on authentication success.

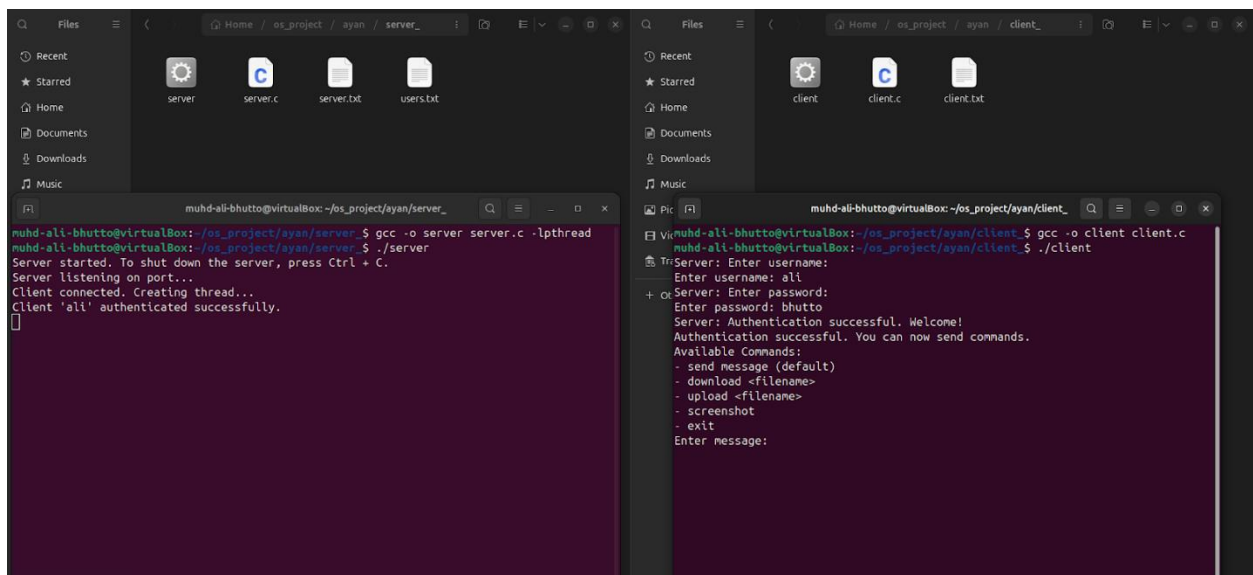
Screenshots

Below are the screenshots of the system in action. Insert these screenshots at the respective places in the report.

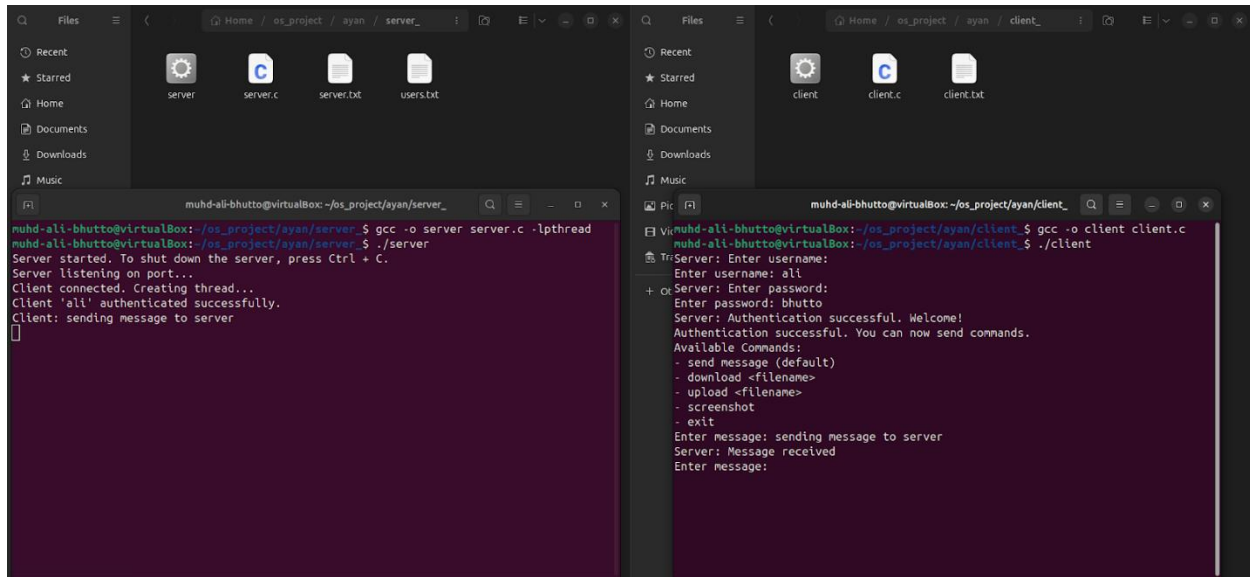
1. Server Startup:



2. Client Authentication Process:



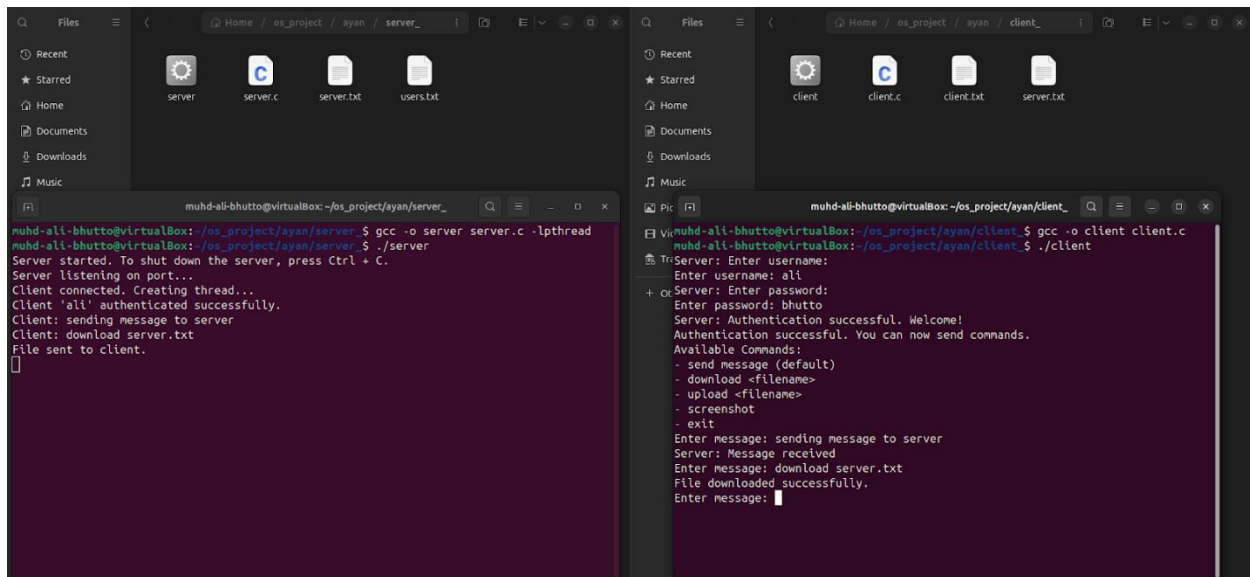
3. Messaging



```
muhd-ali-bhutto@virtualBox: ~/os_project/ayan/server_$ gcc -o server server.c -lpthread
muhd-ali-bhutto@virtualBox: ~/os_project/ayan/server_$ ./server
Server started. To shut down the server, press Ctrl + C.
Server listening on port...
Client connected. Creating thread...
Client 'ali' authenticated successfully.
Client: sending message to server
[]

muhd-ali-bhutto@virtualBox: ~/os_project/ayan/client_$ gcc -o client client.c
muhd-ali-bhutto@virtualBox: ~/os_project/ayan/client_$ ./client
Server: Enter username:
Enter username: ali
Server: Enter password:
Enter password: bhutto
Server: Authentication successful. Welcome!
Authentication successful. You can now send commands.
Available Commands:
- send message (default)
- download <filename>
- upload <filename>
- screenshot
- exit
Enter message: sending message to server
Server: Message received
Enter message:
```

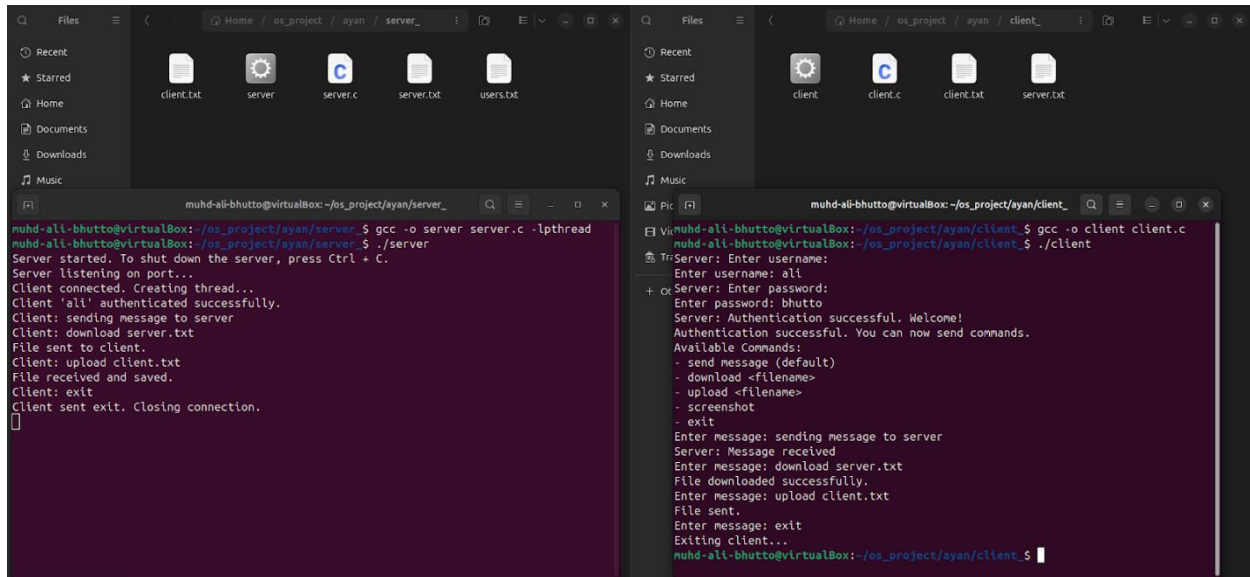
4. File Download Command:



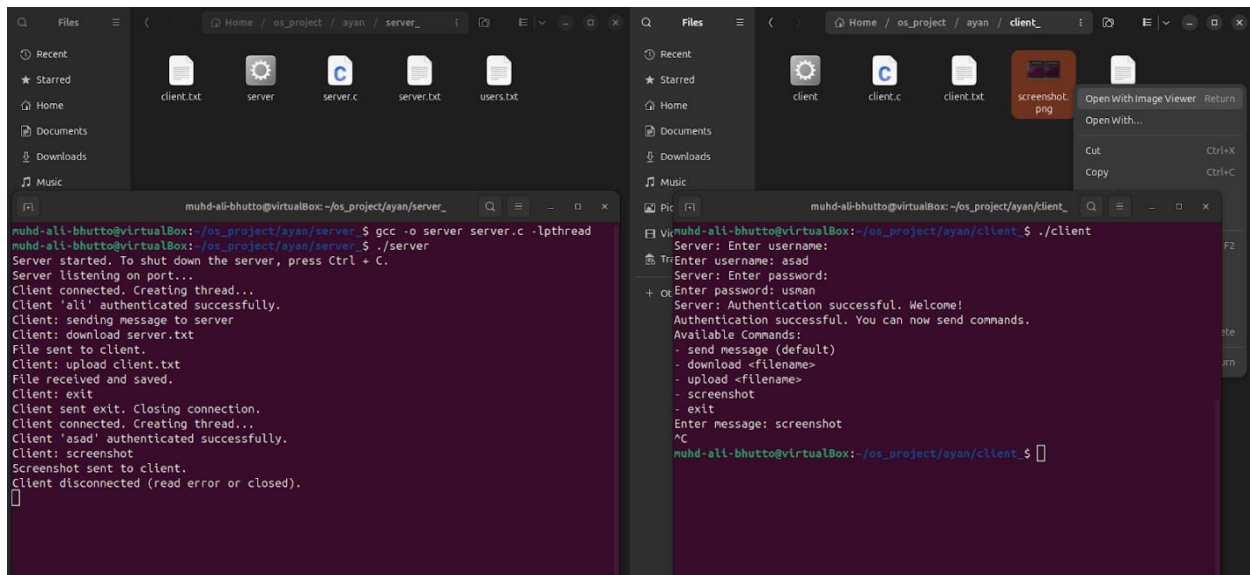
```
muhd-ali-bhutto@virtualBox: ~/os_project/ayan/server_$ gcc -o server server.c -lpthread
muhd-ali-bhutto@virtualBox: ~/os_project/ayan/server_$ ./server
Server started. To shut down the server, press Ctrl + C.
Server listening on port...
Client connected. Creating thread...
Client 'ali' authenticated successfully.
Client: sending message to server
Client: download server.txt
File sent to client.
[]

muhd-ali-bhutto@virtualBox: ~/os_project/ayan/client_$ gcc -o client client.c
muhd-ali-bhutto@virtualBox: ~/os_project/ayan/client_$ ./client
Server: Enter username:
Enter username: ali
Server: Enter password:
Enter password: bhutto
Server: Authentication successful. Welcome!
Authentication successful. You can now send commands.
Available Commands:
- send message (default)
- download <filename>
- upload <filename>
- screenshot
- exit
Enter message: sending message to server
Server: Message received
Enter message: download server.txt
File downloaded successfully.
Enter message:
```

5. File Upload Command:



6. Client Screenshot Request:



Testing and Evaluation

The system was tested with various scenarios, including:

1. **Single Client Authentication:** The system correctly authenticates the client based on the username and password.
2. **Single Client Upload/Download:** The system successfully handles file uploads and downloads.

3. **Multiple Client Handling:** The server can handle multiple clients concurrently with the use of threads and semaphores.
4. **Screenshot Functionality:** The server successfully sends screenshots to the client upon request.
5. **Authentication Failure:** The system disconnects clients who fail to authenticate.

Conclusion

This project successfully implements a secure server-client communication system with authentication. The system provides basic file transfer functionalities and supports taking and sending screenshots, demonstrating key concepts in network programming, such as socket communication, multi-threading, and file handling. The authentication mechanism ensures that only authorized users can access the services provided by the server.