IE2060
Computer Systems Administration
3rd Year, 1nd Semester

Assignment

C Language Socket Programming Assignment: Simulating a Torrent File Exchange System

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfilment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

30/09/2024

Declaration

I certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief, it does not contain any material previously published or written by another person, except where due reference is made in text.

Registration Number: **IT22310132**

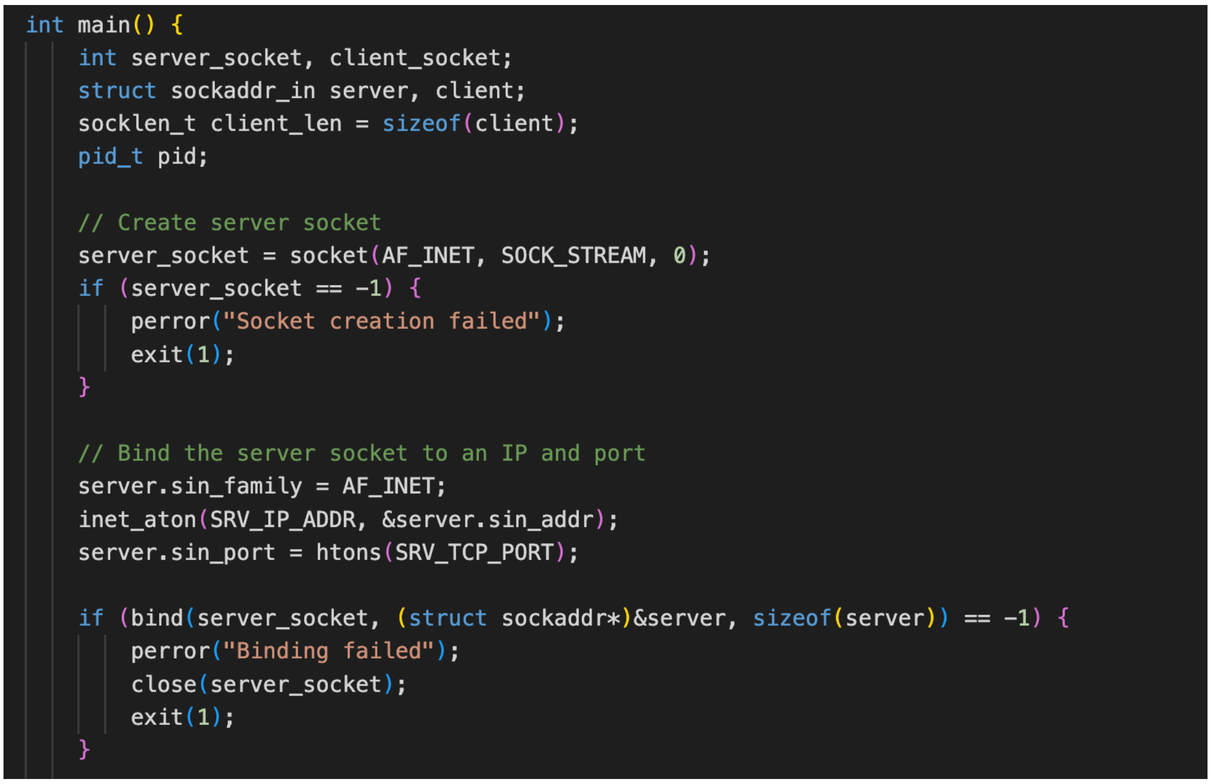
Name: **Muthukumarana T D**

Table of Contents

1. Introduction

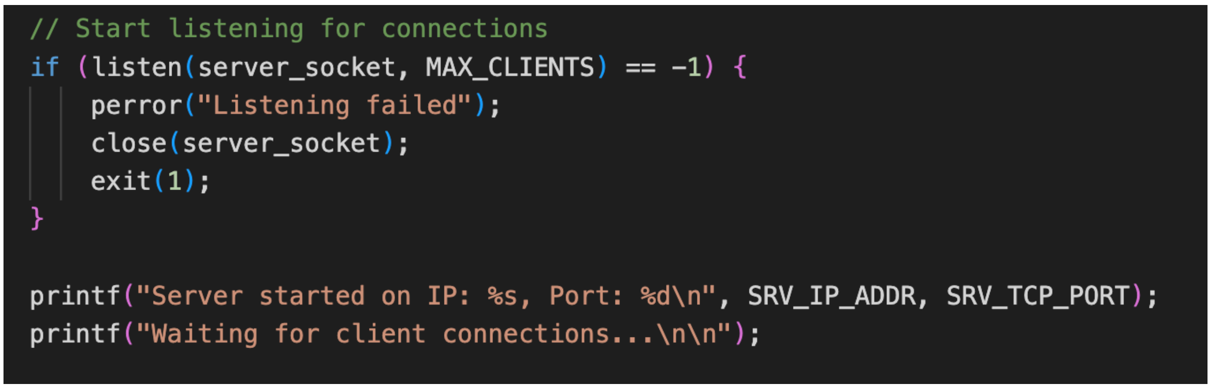
This assignment aims in understanding the behavior of a torrent-like environment, and how to implement such environment using Socket Programming with C Programming Language.

2. Server Program



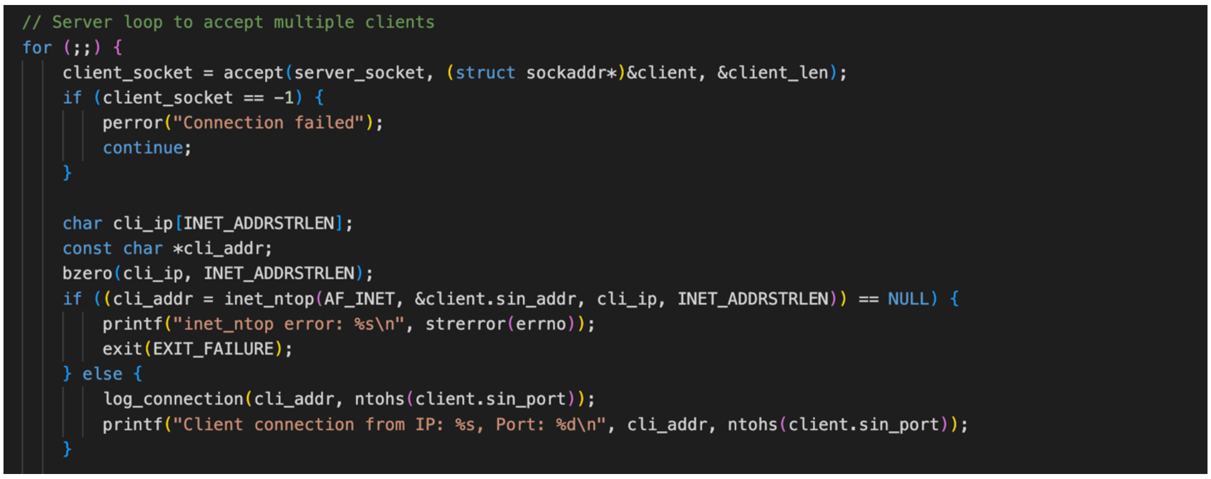
```c
int main() {
    int server_socket, client_socket;
    struct sockaddr_in server, client;
    socklen_t client_len = sizeof(client);
    pid_t pid;

    // Create server socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Socket creation failed");
        exit(1);
    }

    // Bind the server socket to an IP and port
    server.sin_family = AF_INET;
    inet_aton(SRV_IP_ADDR, &server.sin_addr);
    server.sin_port = htons(SRV_TCP_PORT);

    if (bind(server_socket, (struct sockaddr*)&server, sizeof(server)) == -1) {
        perror("Binding failed");
        close(server_socket);
        exit(1);
    }
```

Figure 2. 1 - Creating & Binding the Server Socket.



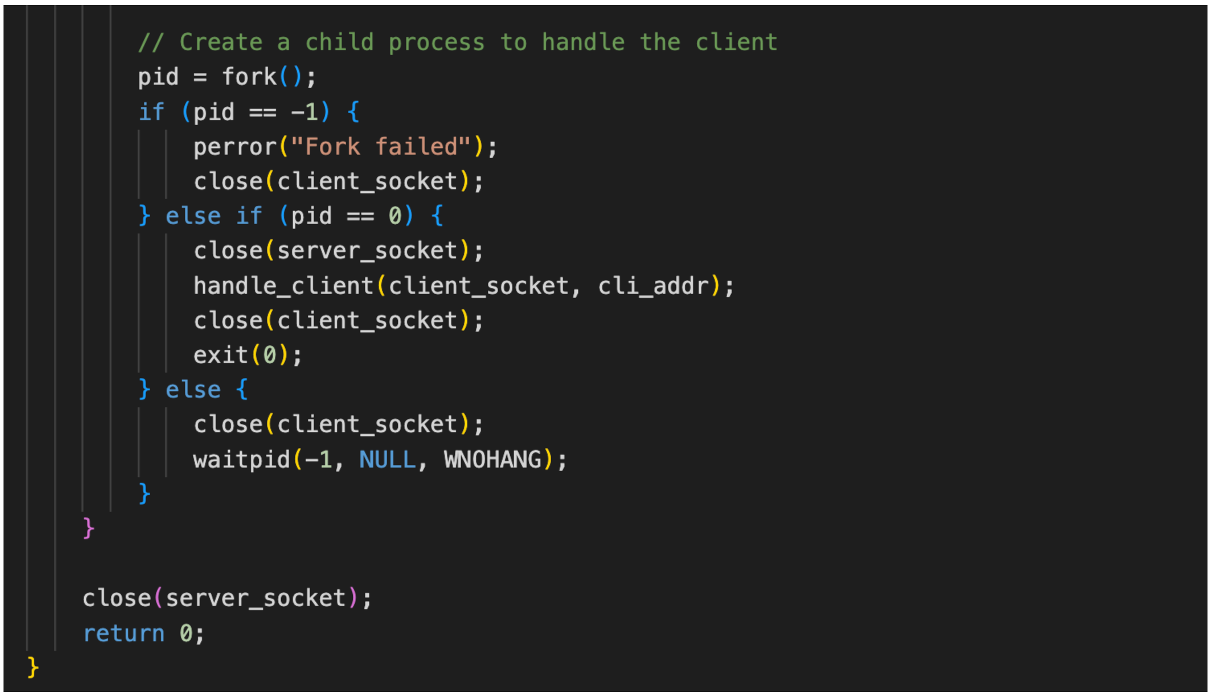
```c
// Start listening for connections
if (listen(server_socket, MAX_CLIENTS) == -1) {
    perror("Listening failed");
    close(server_socket);
    exit(1);
}

printf("Server started on IP: %s, Port: %d\n", SRV_IP_ADDR, SRV_TCP_PORT);
printf("Waiting for client connections...\n\n");
```

Figure 2. 2 - Listening for incoming connections.



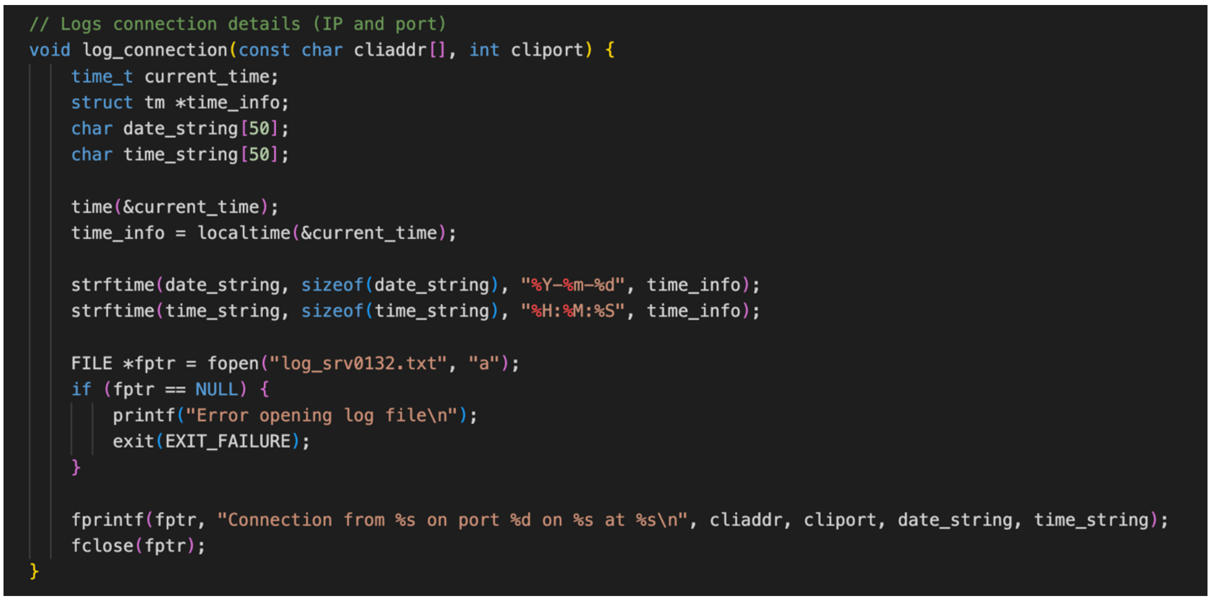
```c
// Server loop to accept multiple clients
for (;;) {
    client_socket = accept(server_socket, (struct sockaddr*)&client, &client_len);
    if (client_socket == -1) {
        perror("Connection failed");
        continue;
    }

    char cli_ip[INET_ADDRSTRLEN];
    const char *cli_addr;
    bzero(cli_ip, INET_ADDRSTRLEN);
    if ((cli_addr = inet_ntop(AF_INET, &client.sin_addr, cli_ip, INET_ADDRSTRLEN)) == NULL) {
        printf("inet_ntop error: %s\n", strerror(errno));
        exit(EXIT_FAILURE);
    } else {
        log_connection(cli_addr, ntohs(client.sin_port));
        printf("Client connection from IP: %s, Port: %d\n", cli_addr, ntohs(client.sin_port));
    }
```

Figure 2. 3 - Accepting multiple client connections.



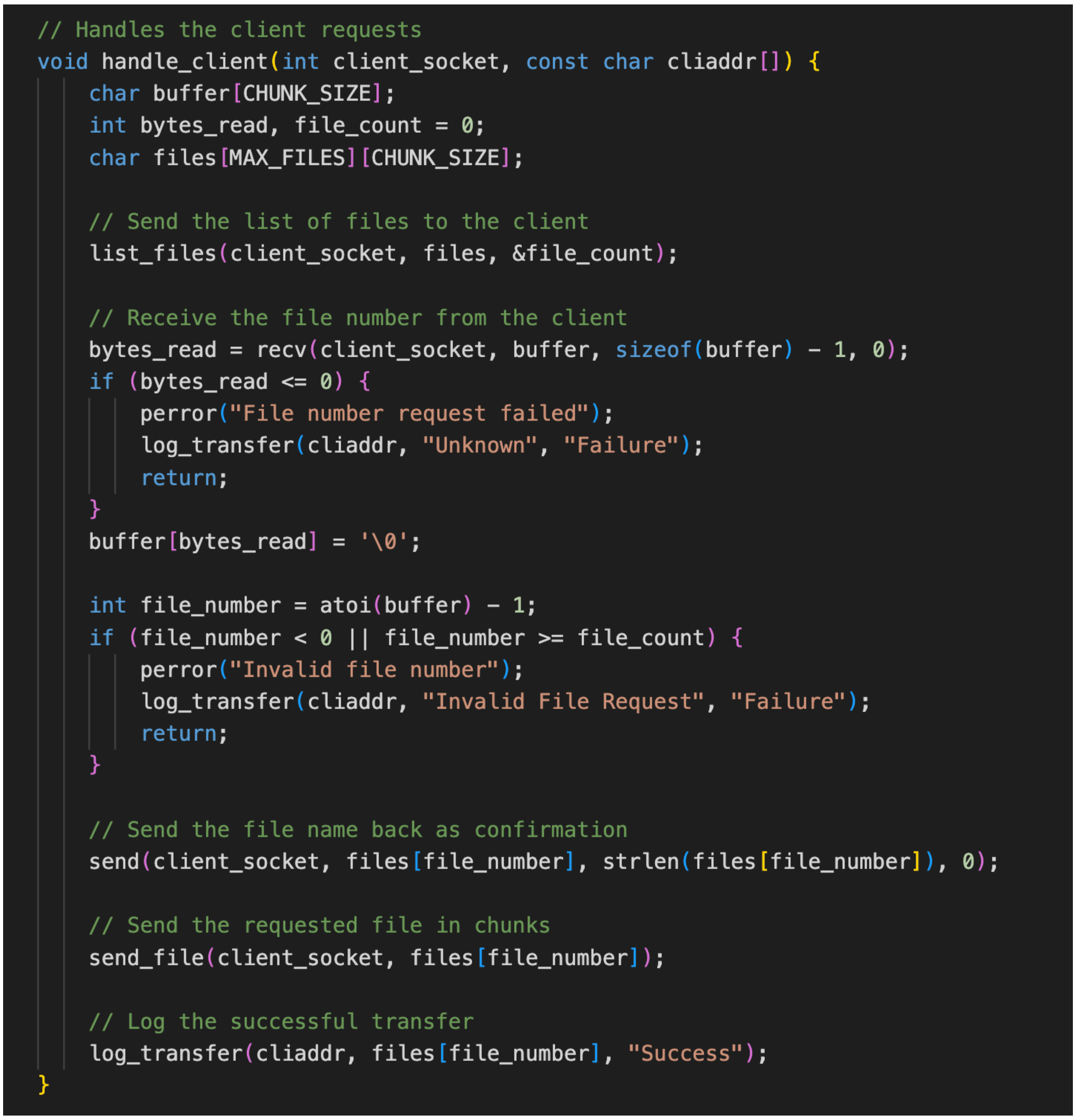
```c
            // Create a child process to handle the client
            pid = fork();
            if (pid == -1) {
                perror("Fork failed");
                close(client_socket);
            } else if (pid == 0) {
                close(server_socket);
                handle_client(client_socket, cli_addr);
                close(client_socket);
                exit(0);
            } else {
                close(client_socket);
                waitpid(-1, NULL, WNOHANG);
            }
        }

        close(server_socket);
        return 0;
}
```

Figure 2. 4 - Creating child processes to handle each connected clients.



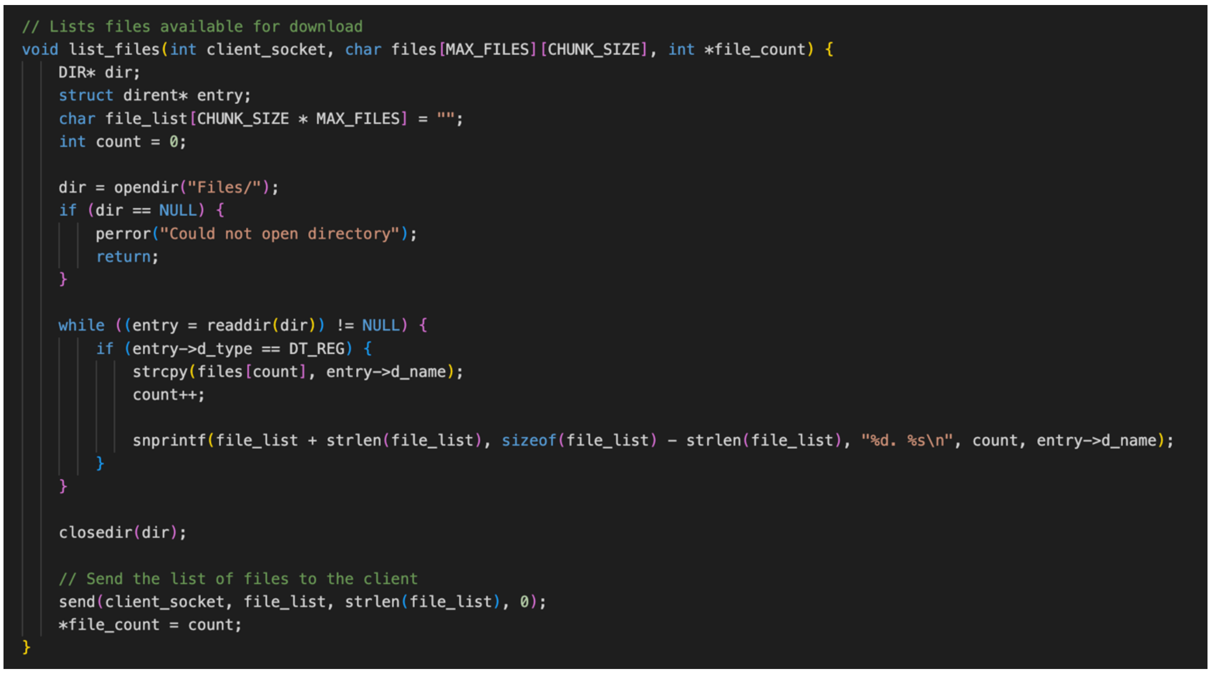
```c
// Logs connection details (IP and port)
void log_connection(const char cliaddr[], int cliport) {
    time_t current_time;
    struct tm *time_info;
    char date_string[50];
    char time_string[50];

    time(&current_time);
    time_info = localtime(&current_time);

    strftime(date_string, sizeof(date_string), "%Y-%m-%d", time_info);
    strftime(time_string, sizeof(time_string), "%H:%M:%S", time_info);

    FILE *fptr = fopen("log_srv0132.txt", "a");
    if (fptr == NULL) {
        printf("Error opening log file\n");
        exit(EXIT_FAILURE);
    }

    fprintf(fptr, "Connection from %s on port %d on %s at %s\n", cliaddr, cliport, date_string, time_string);
    fclose(fptr);
}
```

Figure 2. 5 - Log client connection details.



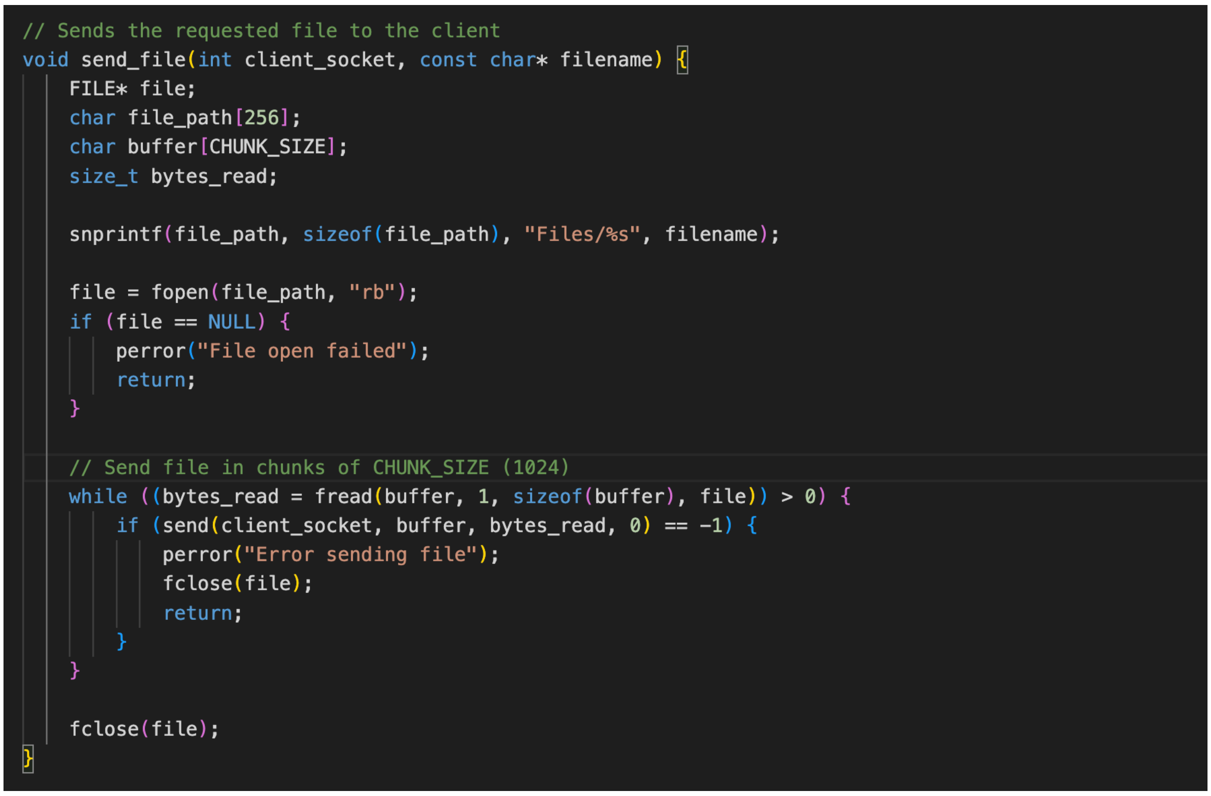
```c
// Handles the client requests
void handle_client(int client_socket, const char cliaddr[]) {
    char buffer[CHUNK_SIZE];
    int bytes_read, file_count = 0;
    char files[MAX_FILES][CHUNK_SIZE];

    // Send the list of files to the client
    list_files(client_socket, files, &file_count);

    // Receive the file number from the client
    bytes_read = recv(client_socket, buffer, sizeof(buffer) - 1, 0);
    if (bytes_read <= 0) {
        perror("File number request failed");
        log_transfer(cliaddr, "Unknown", "Failure");
        return;
    }
    buffer[bytes_read] = '\0';

    int file_number = atoi(buffer) - 1;
    if (file_number < 0 || file_number >= file_count) {
        perror("Invalid file number");
        log_transfer(cliaddr, "Invalid File Request", "Failure");
        return;
    }

    // Send the file name back as confirmation
    send(client_socket, files[file_number], strlen(files[file_number]), 0);

    // Send the requested file in chunks
    send_file(client_socket, files[file_number]);

    // Log the successful transfer
    log_transfer(cliaddr, files[file_number], "Success");
}
```

Figure 2. 6 - Handle the communication between clients.



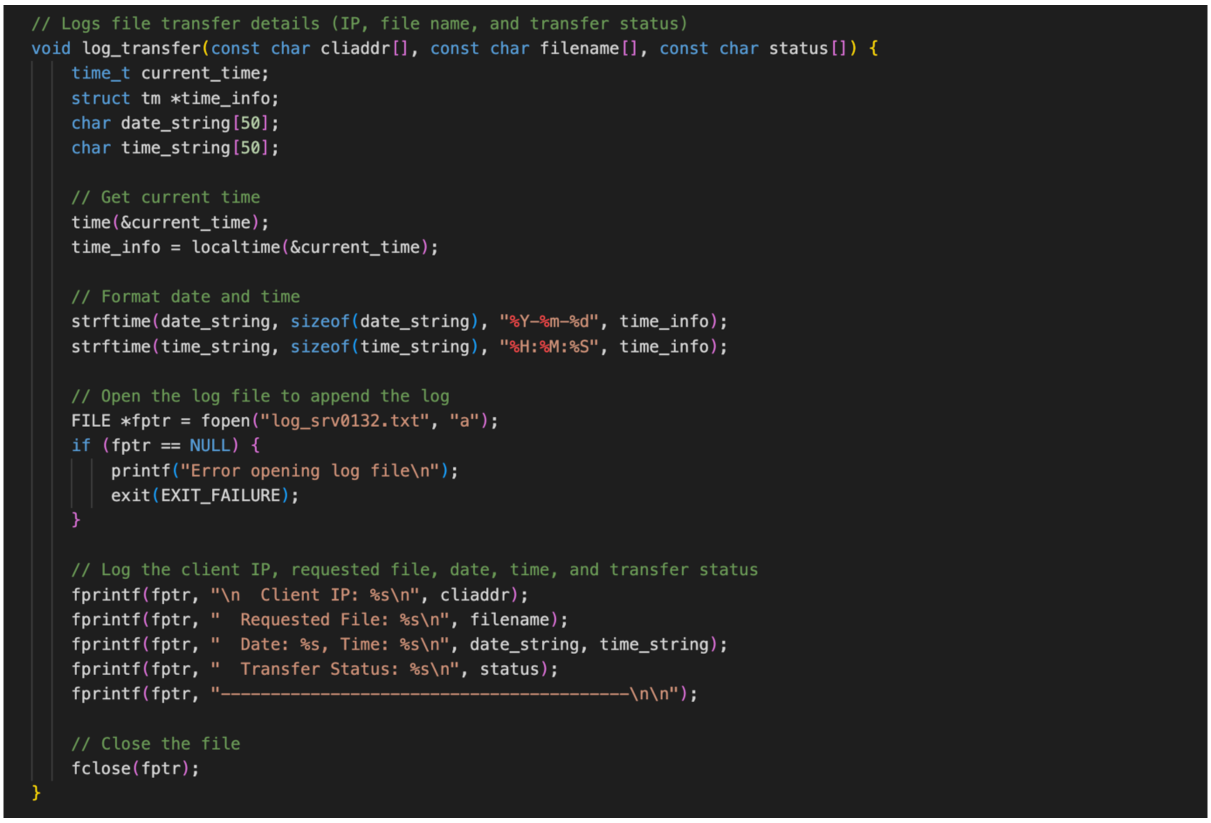
```c
// Lists files available for download
void list_files(int client_socket, char files[MAX_FILES][CHUNK_SIZE], int *file_count) {
    DIR* dir;
    struct dirent* entry;
    char file_list[CHUNK_SIZE * MAX_FILES] = "";
    int count = 0;

    dir = opendir("Files/");
    if (dir == NULL) {
        perror("Could not open directory");
        return;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (entry->d_type == DT_REG) {
            strcpy(files[count], entry->d_name);
            count++;

            snprintf(file_list + strlen(file_list), sizeof(file_list) - strlen(file_list), "%d. %s\n", count, entry->d_name);
        }
    }

    closedir(dir);

    // Send the list of files to the client
    send(client_socket, file_list, strlen(file_list), 0);
    *file_count = count;
}
```

Figure 2. 7 - Sending available files to download.



```c
// Sends the requested file to the client
void send_file(int client_socket, const char* filename) {
    FILE* file;
    char file_path[256];
    char buffer[CHUNK_SIZE];
    size_t bytes_read;

    snprintf(file_path, sizeof(file_path), "Files/%s", filename);

    file = fopen(file_path, "rb");
    if (file == NULL) {
        perror("File open failed");
        return;
    }

    // Send file in chunks of CHUNK_SIZE (1024)
    while ((bytes_read = fread(buffer, 1, sizeof(buffer), file)) > 0) {
        if (send(client_socket, buffer, bytes_read, 0) == -1) {
            perror("Error sending file");
            fclose(file);
            return;
        }
    }

    fclose(file);
}
```

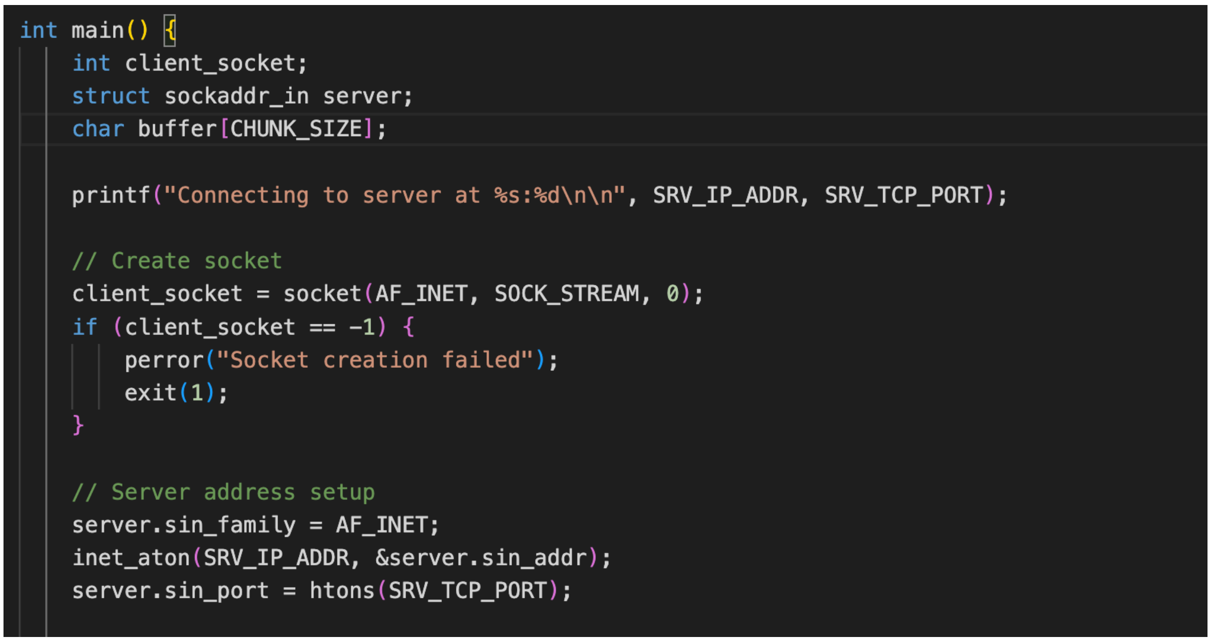
Figure 2. 8 - Send the requested file to client.



```c
// Logs file transfer details (IP, file name, and transfer status)
void log_transfer(const char cliaddr[], const char filename[], const char status[]) {
    time_t current_time;
    struct tm *time_info;
    char date_string[50];
    char time_string[50];

    // Get current time
    time(&current_time);
    time_info = localtime(&current_time);

    // Format date and time
    strftime(date_string, sizeof(date_string), "%Y-%m-%d", time_info);
    strftime(time_string, sizeof(time_string), "%H:%M:%S", time_info);

    // Open the log file to append the log
    FILE *fptr = fopen("log_srv0132.txt", "a");
    if (fptr == NULL) {
        printf("Error opening log file\n");
        exit(EXIT_FAILURE);
    }

    // Log the client IP, requested file, date, time, and transfer status
    fprintf(fptr, "\n  Client IP: %s\n", cliaddr);
    fprintf(fptr, "  Requested File: %s\n", filename);
    fprintf(fptr, "  Date: %s, Time: %s\n", date_string, time_string);
    fprintf(fptr, "  Transfer Status: %s\n", status);
    fprintf(fptr, "-----------------------------------------\n\n");

    // Close the file
    fclose(fptr);
}
```

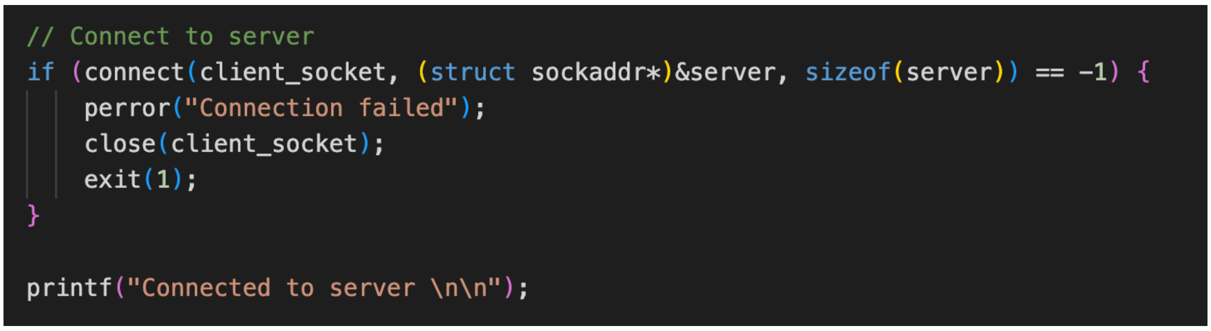
Figure 2. 9 - Log file transfer details.

3. Client Program



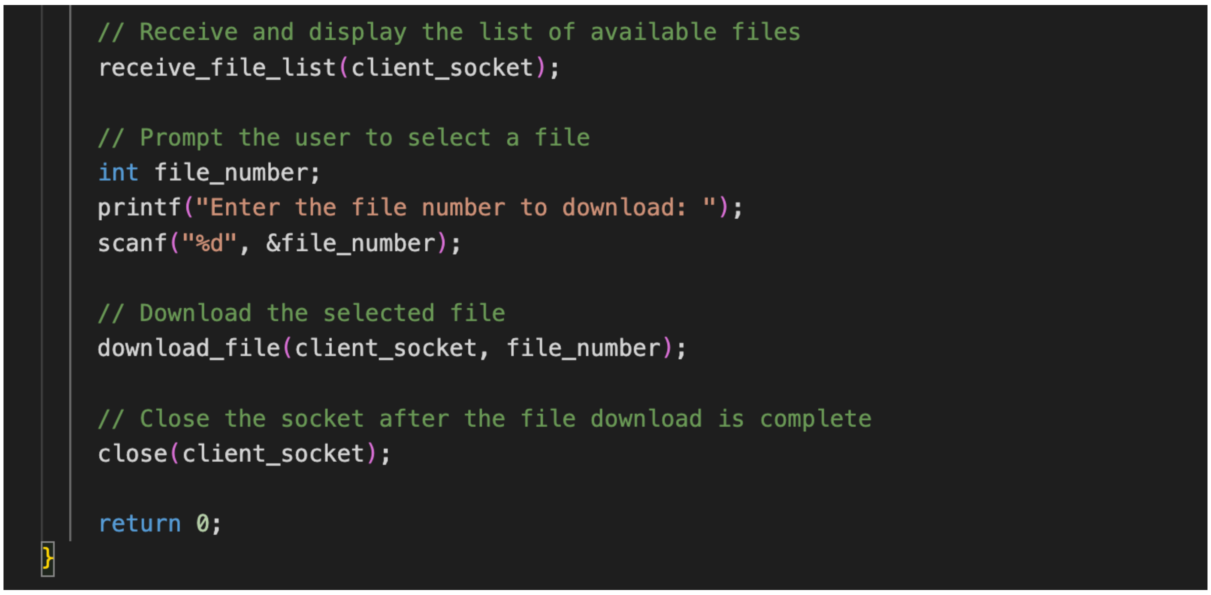
```c
int main() {
    int client_socket;
    struct sockaddr_in server;
    char buffer[CHUNK_SIZE];

    printf("Connecting to server at %s:%d\n\n", SRV_IP_ADDR, SRV_TCP_PORT);

    // Create socket
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Socket creation failed");
        exit(1);
    }

    // Server address setup
    server.sin_family = AF_INET;
    inet_aton(SRV_IP_ADDR, &server.sin_addr);
    server.sin_port = htons(SRV_TCP_PORT);
```

Figure 3. 1 - Creating and Setting up the client socket.



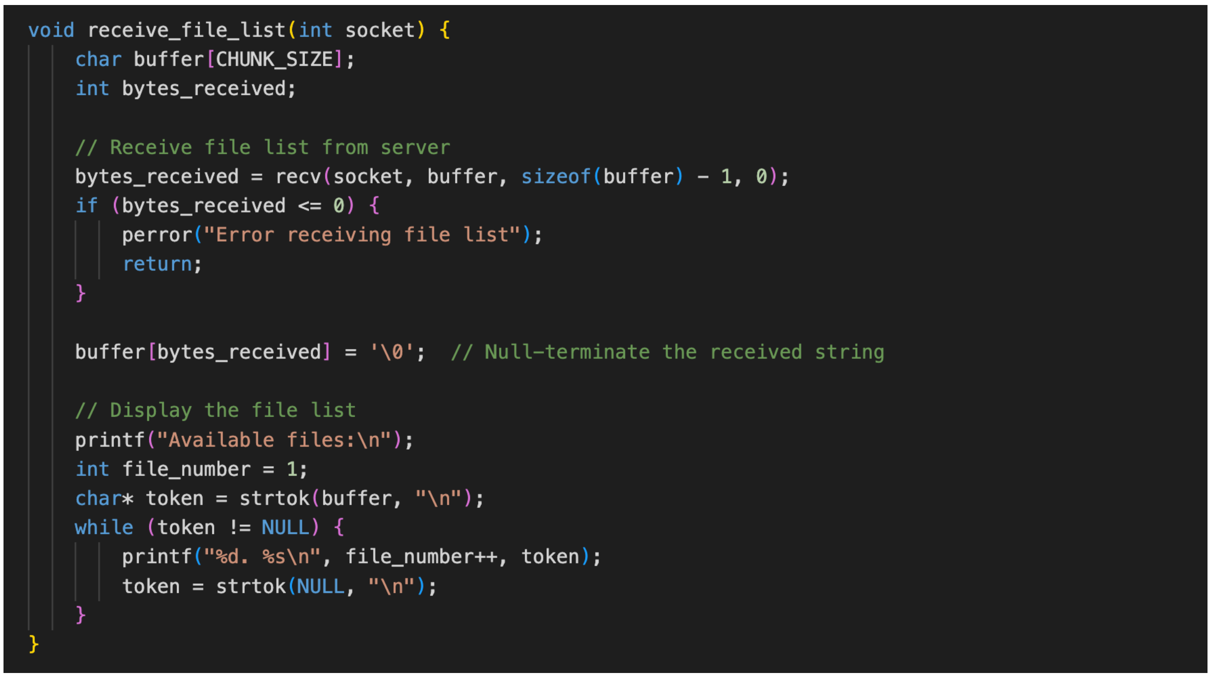
```c
// Connect to server
if (connect(client_socket, (struct sockaddr*)&server, sizeof(server)) == -1) {
    perror("Connection failed");
    close(client_socket);
    exit(1);
}

printf("Connected to server \n\n");
```

Figure 3. 2 - Connecting to the server.



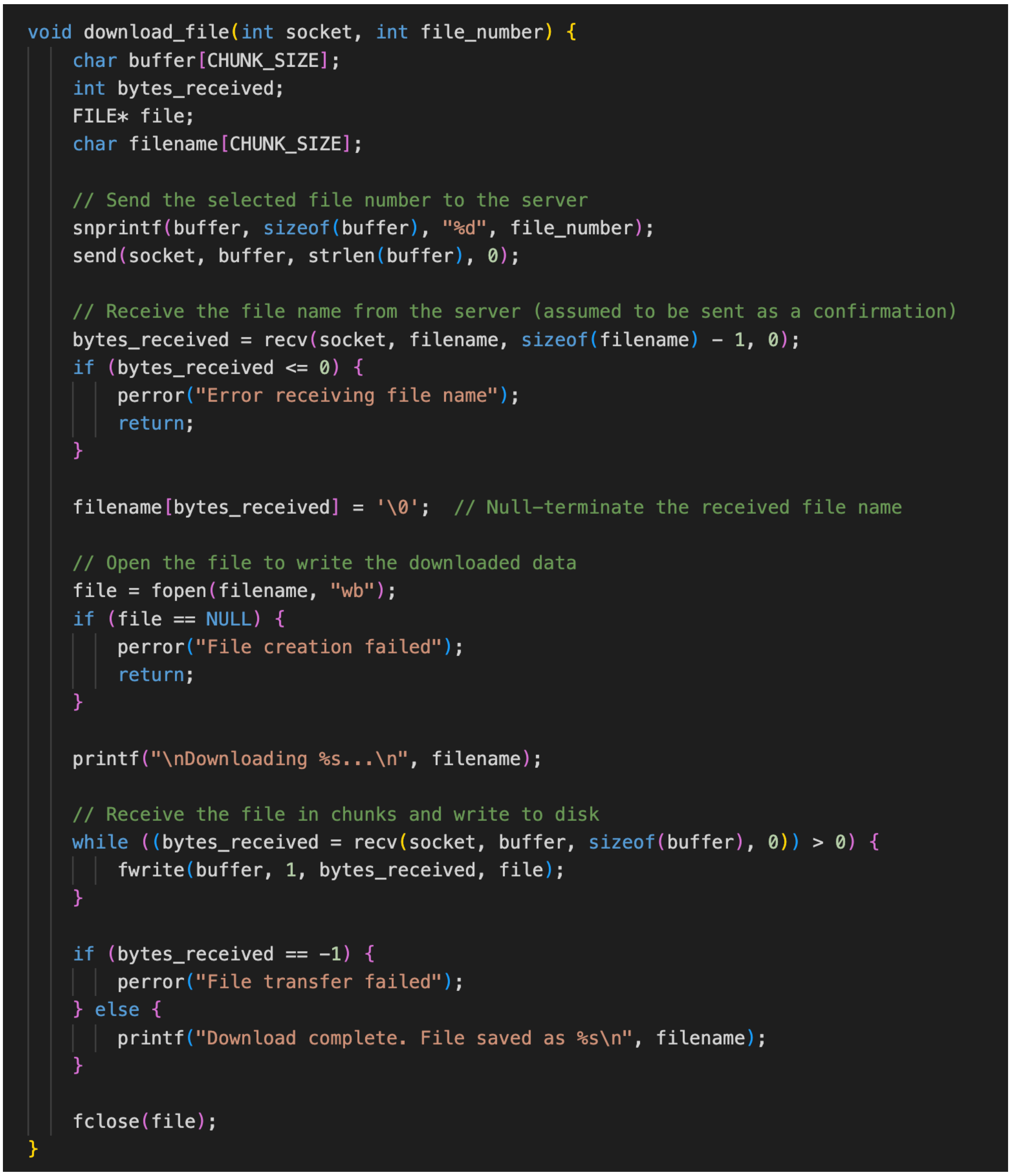
```c
    // Receive and display the list of available files
    receive_file_list(client_socket);

    // Prompt the user to select a file
    int file_number;
    printf("Enter the file number to download: ");
    scanf("%d", &file_number);

    // Download the selected file
    download_file(client_socket, file_number);

    // Close the socket after the file download is complete
    close(client_socket);

    return 0;
}
```

Figure 3. 3 - Handling the communication between the server.



```c
void receive_file_list(int socket) {
    char buffer[CHUNK_SIZE];
    int bytes_received;

    // Receive file list from server
    bytes_received = recv(socket, buffer, sizeof(buffer) - 1, 0);
    if (bytes_received <= 0) {
        perror("Error receiving file list");
        return;
    }

    buffer[bytes_received] = '\0';  // Null-terminate the received string

    // Display the file list
    printf("Available files:\n");
    int file_number = 1;
    char* token = strtok(buffer, "\n");
    while (token != NULL) {
        printf("%d. %s\n", file_number++, token);
        token = strtok(NULL, "\n");
    }
}
```

Figure 3. 4 - Retrieving available files from the server.



```c
void download_file(int socket, int file_number) {
    char buffer[CHUNK_SIZE];
    int bytes_received;
    FILE* file;
    char filename[CHUNK_SIZE];

    // Send the selected file number to the server
    snprintf(buffer, sizeof(buffer), "%d", file_number);
    send(socket, buffer, strlen(buffer), 0);

    // Receive the file name from the server (assumed to be sent as a confirmation)
    bytes_received = recv(socket, filename, sizeof(filename) - 1, 0);
    if (bytes_received <= 0) {
        perror("Error receiving file name");
        return;
    }

    filename[bytes_received] = '\0';  // Null-terminate the received file name

    // Open the file to write the downloaded data
    file = fopen(filename, "wb");
    if (file == NULL) {
        perror("File creation failed");
        return;
    }

    printf("\nDownloading %s...\n", filename);

    // Receive the file in chunks and write to disk
    while ((bytes_received = recv(socket, buffer, sizeof(buffer), 0)) > 0) {
        fwrite(buffer, 1, bytes_received, file);
    }

    if (bytes_received == -1) {
        perror("File transfer failed");
    } else {
        printf("Download complete. File saved as %s\n", filename);
    }

    fclose(file);
}
```

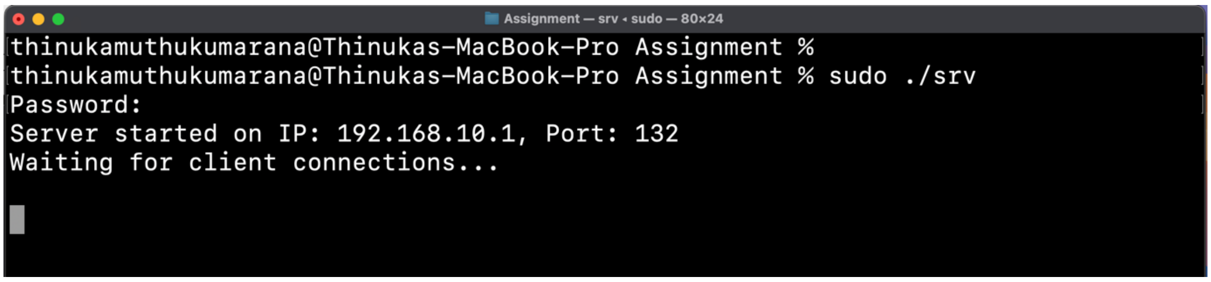
Figure 3. 5 - Downloading a file from the server.

4. Test Cases

The purpose of this test case is to confirm that both client and server programs executes error-free ensuring smooth communication between client and server.

The server program is executed in a host machine running Unix-like operating system, and the client program is executed in a virtual machine running Linux Operating System.

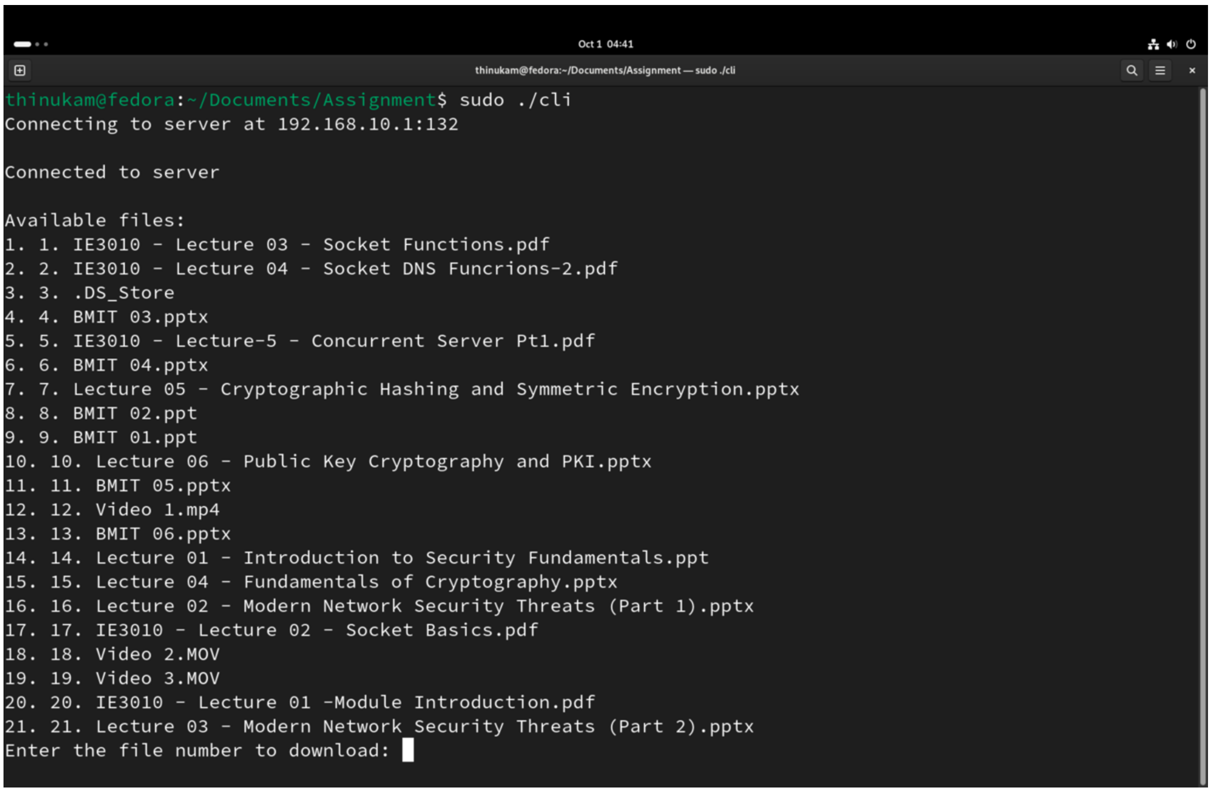
4.1. Launch the Server



```
thinukamuthukumarana@Thinukas-MacBook-Pro Assignment %
thinukamuthukumarana@Thinukas-MacBook-Pro Assignment % sudo ./srv
Password:
Server started on IP: 192.168.10.1, Port: 132
Waiting for client connections...
```

Figure 4. 1 - Launching the server.

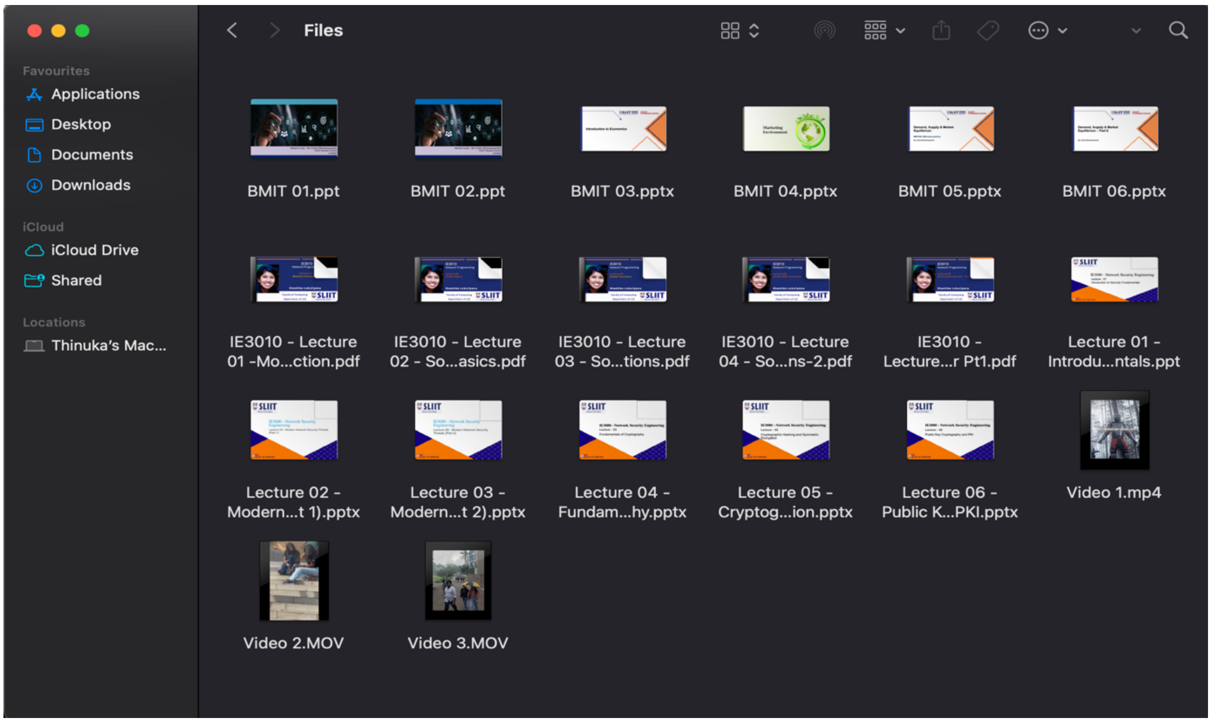
4.2. Launch the Client



```
thinukam@fedora:~/Documents/Assignment$ sudo ./cli
Connecting to server at 192.168.10.1:132

Connected to server

Available files:
1. 1. IE3010 - Lecture 03 - Socket Functions.pdf
2. 2. IE3010 - Lecture 04 - Socket DNS Funcrions-2.pdf
3. 3. .DS_Store
4. 4. BMIT 03.pptx
5. 5. IE3010 - Lecture-5 - Concurrent Server Pt1.pdf
6. 6. BMIT 04.pptx
7. 7. Lecture 05 - Cryptographic Hashing and Symmetric Encryption.pptx
8. 8. BMIT 02.ppt
9. 9. BMIT 01.ppt
10. 10. Lecture 06 - Public Key Cryptography and PKI.pptx
11. 11. BMIT 05.pptx
12. 12. Video 1.mp4
13. 13. BMIT 06.pptx
14. 14. Lecture 01 - Introduction to Security Fundamentals.ppt
15. 15. Lecture 04 - Fundamentals of Cryptography.pptx
16. 16. Lecture 02 - Modern Network Security Threats (Part 1).pptx
17. 17. IE3010 - Lecture 02 - Socket Basics.pdf
18. 18. Video 2.MOV
19. 19. Video 3.MOV
20. 20. IE3010 - Lecture 01 -Module Introduction.pdf
21. 21. Lecture 03 - Modern Network Security Threats (Part 2).pptx
Enter the file number to download: 
```

Figure 4. 2 - Launching the client

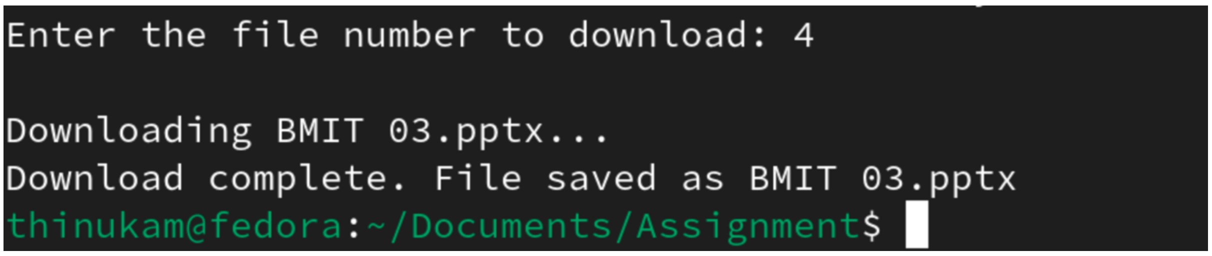
4.3. Ensure all the files in the directory are sent and displayed



```
BMIT 01.ppt
BMIT 02.ppt
BMIT 03.pptx
BMIT 04.pptx
BMIT 05.pptx
BMIT 06.pptx
IE3010 - Lecture 01 -Mo...ction.pdf
IE3010 - Lecture 02 - So...asics.pdf
IE3010 - Lecture 03 - So...tions.pdf
IE3010 - Lecture 04 - So...ns-2.pdf
IE3010 - Lecture...r Pt1.pdf
Lecture 01 - Introdu...ntals.ppt
Lecture 02 - Modern...t 1).pptx
Lecture 03 - Modern...t 2).pptx
Lecture 04 - Fundam...hy.pptx
Lecture 05 - Cryptog...ion.pptx
Lecture 06 - Public K...PKI.pptx
Video 1.mp4
Video 2.MOV
Video 3.MOV
```

Figure 4. 3 - List of files in the server shared directory.

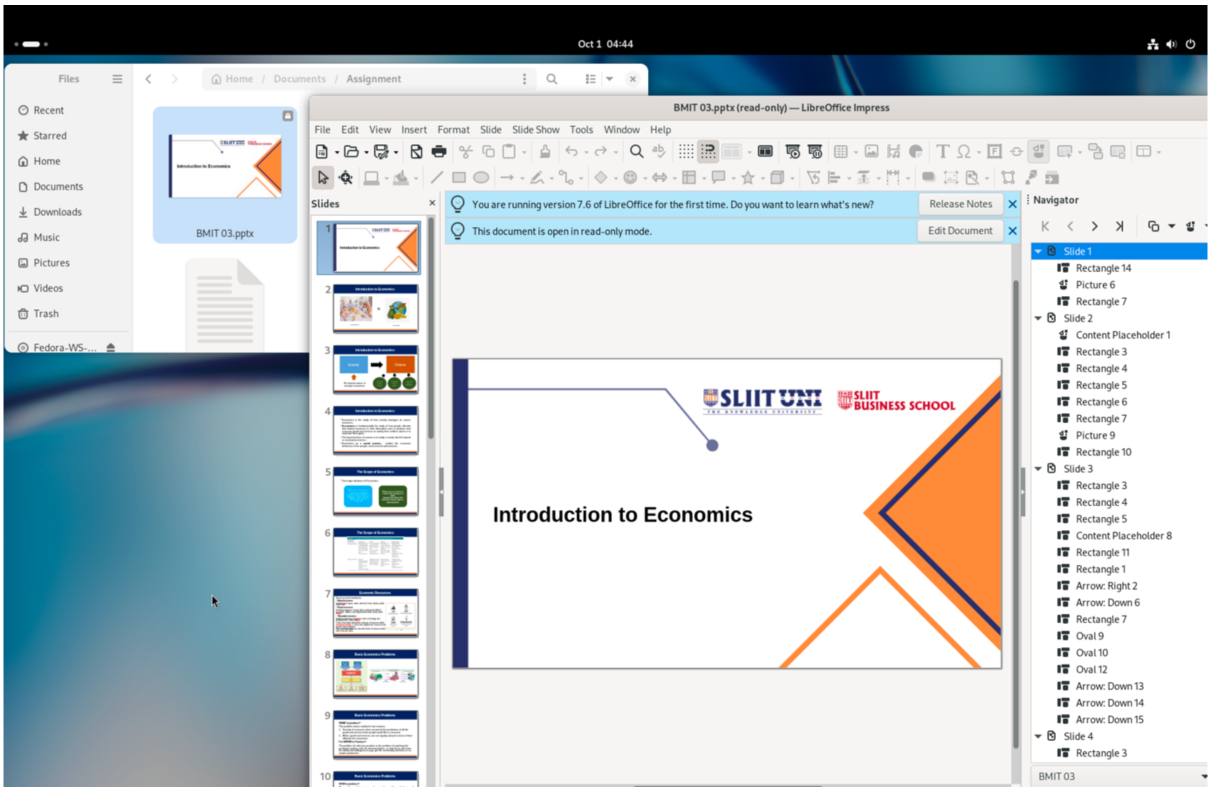
4.4. Download from the server



```
Enter the file number to download: 4

Downloading BMIT 03.pptx...
Download complete. File saved as BMIT 03.pptx
thinukam@fedora:~/Documents/Assignment$
```

Figure 4. 4 - Downloading a file.

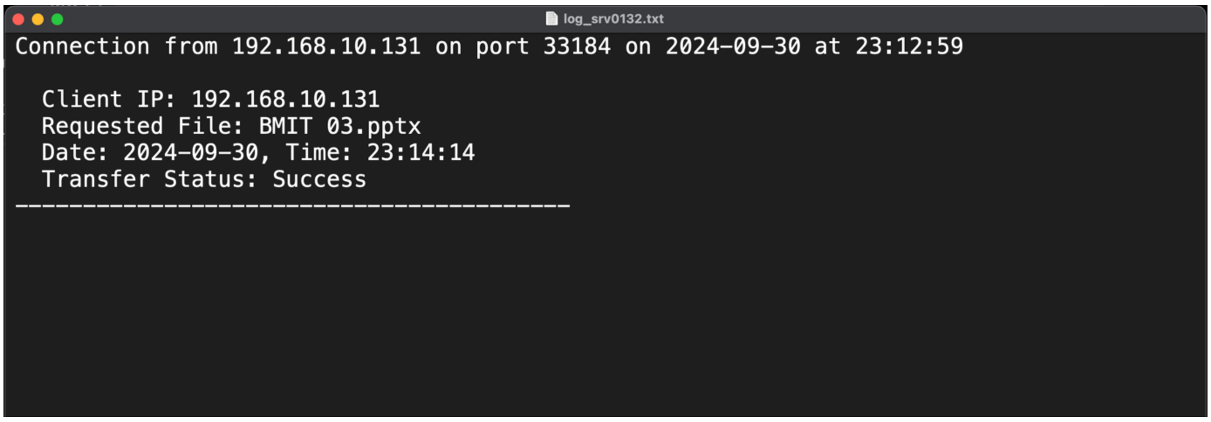
4.5. Ensuring the correct file have been downloaded



Oct 1 04:44
Files
Home / Documents / Assignment
Recent
Starred
Home
Documents
Downloads
Music
Pictures
Videos
Trash
BMIT 03.pptx
BMIT 03.pptx (read-only) — LibreOffice Impress
File Edit View Insert Format Slide Slide Show Tools Window Help
Slides
You are running version 7.6 of LibreOffice for the first time. Do you want to learn what's new?
Release Notes
This document is open in read-only mode.
Edit Document
Introduction to Economics
Navigator
Slide 1
Slide 2
Slide 3
Slide 4
BMIT 03

Figure 4. 5 - File downloaded on Client and working.

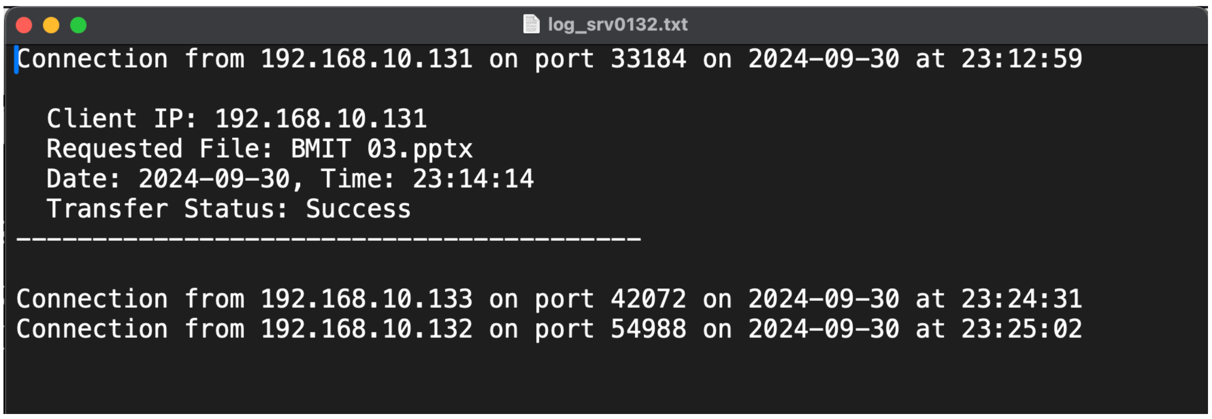
4.6. Ensuring all connection and transfer details are logged



```
Connection from 192.168.10.131 on port 33184 on 2024-09-30 at 23:12:59

  Client IP: 192.168.10.131
  Requested File: BMIT 03.pptx
  Date: 2024-09-30, Time: 23:14:14
  Transfer Status: Success
-----------------------------------------
```

Figure 4. 6 - Connection and Transfer Log.

4.7. Multi-client connection



```
Connection from 192.168.10.131 on port 33184 on 2024-09-30 at 23:12:59

  Client IP: 192.168.10.131
  Requested File: BMIT 03.pptx
  Date: 2024-09-30, Time: 23:14:14
  Transfer Status: Success
-----------------------------------------

Connection from 192.168.10.133 on port 42072 on 2024-09-30 at 23:24:31
Connection from 192.168.10.132 on port 54988 on 2024-09-30 at 23:25:02
```

Figure 4. 7 - Multi-client connection log

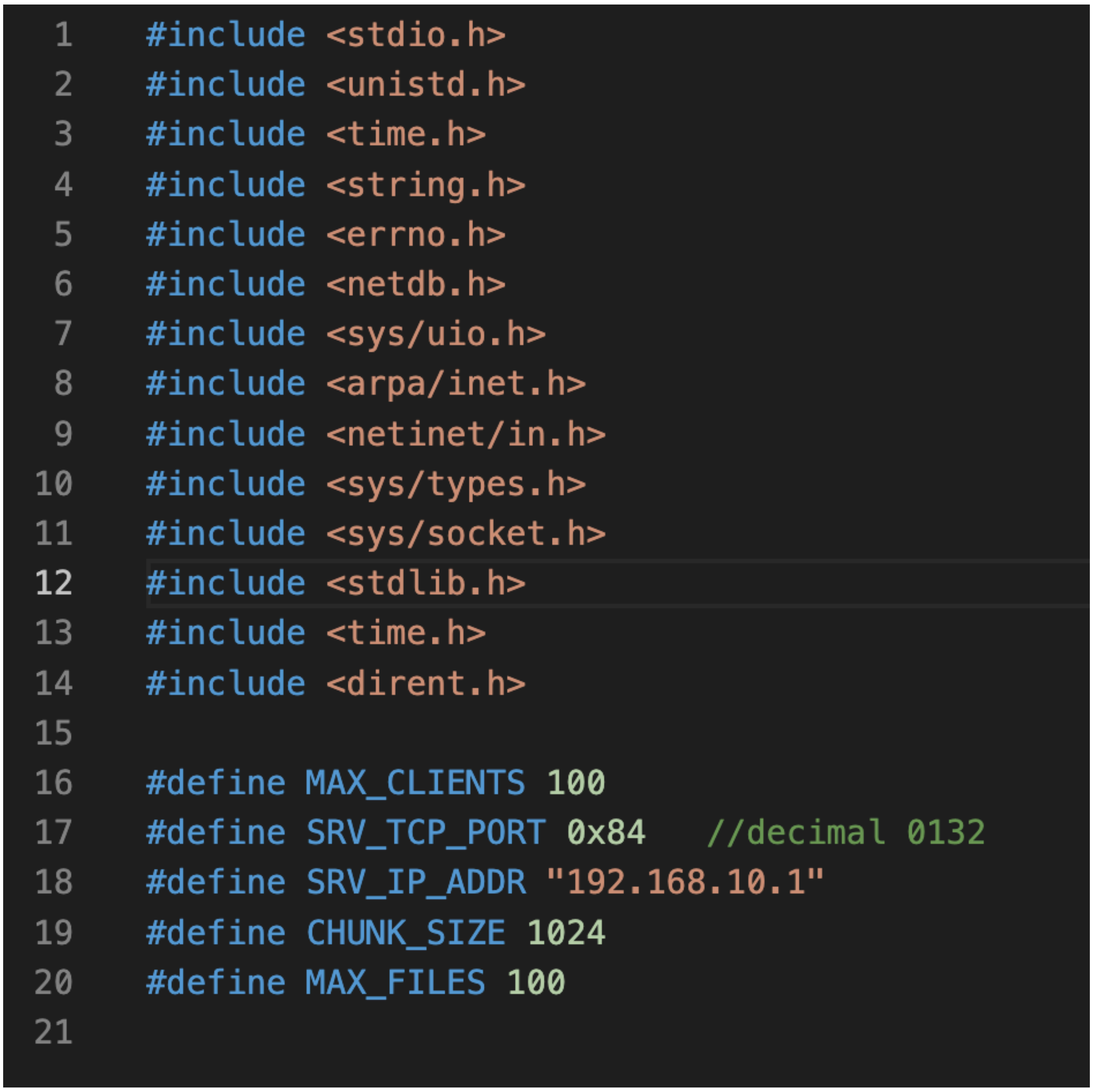
5. Conclusion

This hands-on-experience gives a strong foundation on understanding of how a torrent-like environment works, step-by-step methods in implementing such environment where multiple clients can connect to the server and download available files error-free.

6. References

i. https://www.geeksforgeeks.org/socket-programming-cc/
ii. https://youtube.com/playlist?list=PLPyaR5G9aNDvs6TtdpLcVO43_jvxp4emI&si=_aBTNsW5k2LWNO-o
iii. https://youtube.com/playlist?list=PL9IEJIKnBJjH_zM5LnovnoaKlXML5qh17&si=pku3kAT5RNQETBm1
iv. https://www.sanfoundry.com/c-program-list-files-directory/
v. https://www.w3schools.com/c/c_files_write.php
vi. https://stackoverflow.com/questions/11952898/c-send-and-receive-file
vii. https://www.youtube.com/watch?v=7d7_G81uews

7. Appendix



```c
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include <string.h>
#include <errno.h>
#include <netdb.h>
#include <sys/uio.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <time.h>
#include <dirent.h>

#define MAX_CLIENTS 100
#define SRV_TCP_PORT 0x84    //decimal 0132
#define SRV_IP_ADDR "192.168.10.1"
#define CHUNK_SIZE 1024
#define MAX_FILES 100

```

Figure 7. 1 - "headers.h" header file.