Multi-Client File Server Project Report

A MINI-PROJECT REPORT

Submitted by

SUDHARSAN S 231901054

RAM HAYGREV S 231901039

VENSELVAM V 231901061

in partial fulfillment of the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING SPECIALIZED IN CYBER SECURITY

RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

April 2025

Chennai

BONAFIDE CERTIFICATE

Certified that this project "A MULTI-CLIENT FILE SERVER PROJECT REPORT " is the bonafide work of "SUDHARSAN S, RAM HAYGREV S, VENSELVAM V" who carried out the project work under my supervision.

This mini project report is submitted for the viva voce examination to be held on

_____

SIGNATURE

Mrs. JANANEE V

ASSISTANT PROFESSOR

Dept. of Computer Science Engg,

Rajalakshmi Engineering College,

Chennai.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved and honorable chairman **MR. S. MEGANATHAN** and the chairperson **DR. M. THANGAM MEGANATHAN** for their timely support and encouragement. We are greatly indebted to our respected and honorable principal **Dr. S.N. MURUGESAN** for his able support and guidance. No words of gratitude will suffice for the unquestioning support extended to us by our Head Of The Department **Mr. BENIDICT JAYAPRAKASH NICHOLAS** for being an ever-supporting force during our project work. We also extend our sincere and hearty thanks to our internal guide **Mrs.V JANANEE** for her valuable guidance and motivation during the completion of this project. Our sincere thanks to our family members, friends, and other staff members of computer science engineering.

1. **SUDHARSAN S**
2. **RAM HAYGREV S**
3. **VENSELVAM V**

Table of Contents

Abstract

The Multi-Client File Server is a C-based project that allows multiple clients to connect to a server and perform secure file operations such as upload and download. The server handles multiple clients simultaneously using multithreading, ensuring smooth and efficient communication without delays.

The system includes user authentication, requiring clients to log in with valid credentials before accessing file services. Uploaded and downloaded files are organized properly and all activities are recorded to maintain security and tracking. Error handling is carefully managed to address login failures, network interruptions, and file system errors, making the server reliable even under poor connectivity.

The project structure is clear, with modules for authentication, file management, multithreading, and logging. It applies core concepts of operating systems and networking to build a simple but effective file server that is both user-friendly and dependable.

MULTI-CLIENT FILE SERVER

1. Introduction

1.1 Overview

The Multi-Client File Server is a C-based project that demonstrates Operating System concepts such as multi-threading, inter-process communication, file system management, and user authentication. It allows multiple clients to connect to a server, authenticate, and perform file operations such as upload and download over a local network or the same device.

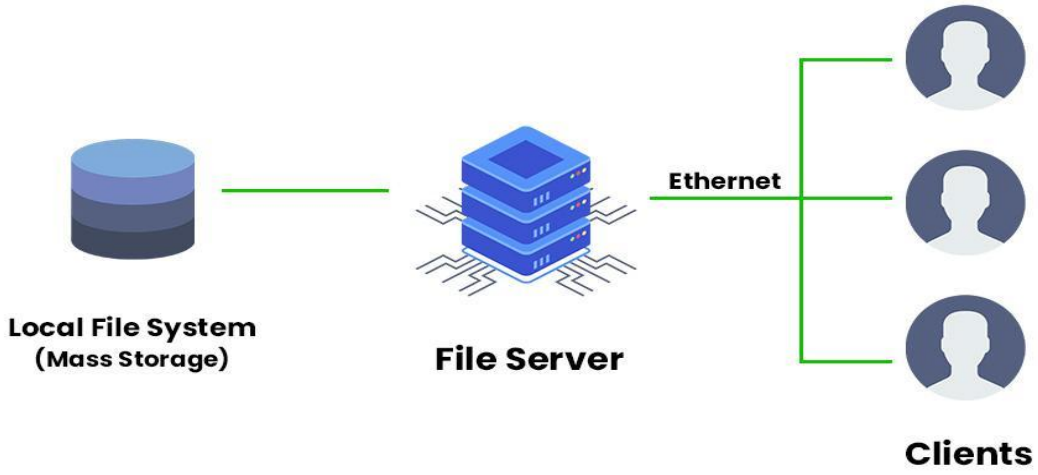
1.2 Objective

The primary objective of this project is to build a secure and efficient file server that supports multiple clients concurrently. It aims to provide user authentication, secure file handling, and robust error handling while applying OS concepts like fork, pipe, pthreads, and system calls.

2. System Design

2.1 System Architecture

The architecture includes a server that handles multiple client connections using pthreads. Each client connects through a socket, and once authenticated, can upload/download files. The server uses fork and pipe to handle logging and communication with child processes.



2.2 Modules

Authentication Module

File Upload/Download Module

Client Handler (Multithreaded)

Logging and Error Handling Module

3. Implementation

3.1 Tools and Libraries Used

Programming Language: C

Libraries: pthread, socket

OS Concepts: fork, pipe, exec, file system calls

Platform: Linux (Ubuntu)

3.2 Workflow

1. Server initializes and listens for incoming client connections.
2. Each client is handled in a separate thread.
3. Client authenticates using a predefined credential system.
4. Once authenticated, client can upload or download files.
5. Server logs each action and maintains a secure file directory.

4. Programs

4.1 server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <limits.h>
#define PORT 9000
#define MAX_CLIENTS 5
#define MAX_BUFFER 8192
#define AUTH_FILE "./common/auth.txt"
```

```
// Structure to hold client data
typedef struct {
    int socket;
    char username[50];
} client_data_t;

// Function to check authentication
int authenticate_user(int client_socket, char *username) {
    char recv_data[MAX_BUFFER], user_pass[4096], line[100];
    FILE *fp = fopen(AUTH_FILE, "r");
    if (!fp) return 0;
    send(client_socket, "Username: ", strlen("Username: "), 0);
    recv(client_socket, username, 50, 0);
    username[strcspn(username, "\n")] = 0;
    send(client_socket, "Password: ", strlen("Password: "), 0);
    recv(client_socket, recv_data, MAX_BUFFER, 0);
    recv_data[strcspn(recv_data, "\n")] = 0;
  int written = snprintf(user_pass, sizeof(user_pass), "%s:%s", username, recv_data);
    if (written >= sizeof(user_pass))
        fprintf(stderr, "[!] Warning: user_pass was truncated!\n");
    while (fgets(line, sizeof(line), fp)) {
        line[strcspn(line, "\n")] = 0;
```

```
        if (strcmp(line, user_pass) == 0) {
            fclose(fp);
            return 1;
        }
    }
    fclose(fp);
    return 0;
}
// Function to log client actions
void log_action(const char *action, const char *username) {
    pid_t pid = fork();
    if (pid == 0) {
        FILE *logfile = fopen("./server/logs/server_log.txt", "a");
        if (logfile) {
            fprintf(logfile, "User: %s | Action: %s\n", username, action);
            fclose(logfile);
        }
        exit(0);
    } else {
        wait(NULL);
    }
}
```

```
void *client_handler(void *arg) {
    client_data_t *client_data = (client_data_t *)arg;
    char buffer[MAX_BUFFER], filepath[MAX_BUFFER], filename[256];
    int bytes_read;
    log_action("Client connected", client_data->username);
    while (1) {
        bzero(buffer, MAX_BUFFER);
        if ((bytes_read = recv(client_data->socket, buffer, MAX_BUFFER, 0)) <= 0) break;


        buffer[strcspn(buffer, "\n")] = 0;


        if (strcmp(buffer, "upload") == 0) {
            recv(client_data->socket, filename, sizeof(filename), 0);
            filename[strcspn(filename, "\n")] = 0;
            snprintf(filepath, sizeof(filepath), "./server/server_files/%s_%s", client_data->username, filename);
            int fd = open(filepath, O_WRONLY | O_CREAT | O_TRUNC, 0644);
            if (fd < 0) continue;
            while ((bytes_read = recv(client_data->socket, buffer, MAX_BUFFER, 0)) > 0) {
                buffer[bytes_read] = '\0';
                if (strcmp(buffer, "EOF") == 0) break;
                write(fd, buffer, bytes_read);
```

```
            }
            close(fd);
            log_action("File uploaded", client_data->username);
        } else if (strcmp(buffer, "download") == 0) {
            DIR *dir = opendir("./server/server_files");
            struct dirent *entry;
            char file_list[MAX_BUFFER] = "Available files:\n";
            if (dir) {
                while ((entry = readdir(dir)) != NULL) {
                    // Skip . and ..
                    if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
                        continue;
                    char fullpath[PATH_MAX];
                    snprintf(fullpath, sizeof(fullpath), "./server/server_files/%s", entry->d_name);
                    struct stat path_stat;
                    stat(fullpath, &path_stat);
                    if (!S_ISDIR(path_stat.st_mode)) {
                        strcat(file_list, entry->d_name);
                        strcat(file_list, "\n");
                    }
                }
                closedir(dir);
```

```
        } else {
            strcpy(file_list, "Could not list files.\n");
        }
        send(client_data->socket, file_list, strlen(file_list), 0);
        send(client_data->socket, "Enter filename to download:\n", 29, 0);

        recv(client_data->socket, filename, sizeof(filename), 0);
        filename[strcspn(filename, "\n")] = 0;

        snprintf(filepath, sizeof(filepath), "./server/server_files/%s", filename);
        struct stat path_stat;
        if (stat(filepath, &path_stat) < 0 || S_ISDIR(path_stat.st_mode)) {
            send(client_data->socket, "Error: File does not exist or is a directory.\n", 46, 0);
            continue;
        }
        int fd = open(filepath, O_RDONLY);
        if (fd < 0) {
            send(client_data->socket, "Error opening file.\n", 21, 0);
            continue;
        }
      while ((bytes_read = read(fd, buffer, MAX_BUFFER)) > 0){
            send(client_data->socket, buffer, bytes_read, 0);
```

```
            }
            send(client_data->socket, "EOF", 3, 0);
            close(fd);
            log_action("File downloaded", client_data->username);
        }
        else {
            send(client_data->socket, "Invalid command\n", 17, 0);
        }
    }

    close(client_data->socket);
    log_action("Client disconnected", client_data->username);
    free(client_data);
    pthread_exit(NULL);
}

int main() {
    int server_fd, client_fd;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len = sizeof(client_addr);
    pthread_t tid;

    mkdir("./server", 0777);
```

```
    mkdir("./server/server_files", 0777);
    mkdir("./server/logs", 0777);

    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (server_fd == -1) exit(1);

    bzero(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    if (bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) exit(1);
    if (listen(server_fd, MAX_CLIENTS) < 0) exit(1);

    printf("Server listening on port %d...\n", PORT);

    while (1) {
        client_fd = accept(server_fd, (struct sockaddr *)&client_addr, &client_len);
        if (client_fd < 0) continue;

        client_data_t *data = malloc(sizeof(client_data_t));
        data->socket = client_fd;
```

```
        if (!authenticate_user(client_fd, data->username)) {
            send(client_fd, "Authentication failed\n", 23, 0);
            close(client_fd);
            free(data);
            continue;
        }
        else {
        send(client_fd, "Authentication successful\n", 27, 0);
        }
        pthread_create(&tid, NULL, client_handler, (void *)data);
        pthread_detach(tid);
    }
    close(server_fd);
    return 0;
}
```

4.2 client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#define SERVER_IP "127.0.0.1"
#define PORT 9000
#define MAX_BUFFER 8192

void error_exit(const char *msg) {
    perror(msg);
    exit(EXIT_FAILURE);
}

void recv_prompt(int sock, char *prompt, size_t size) {
    int len = recv(sock, prompt, size - 1, 0);
```

```
    if (len <= 0) {
        fprintf(stderr, "[!] Server closed connection or timeout\n");
        exit(EXIT_FAILURE);
    }
    prompt[len] = '\0';
    printf("%s", prompt);
}

int main() {
    int sock;
    struct sockaddr_in server_addr;
    char buffer[MAX_BUFFER], input[256], filename[256], full_filename[512];
    char username[50], password[50];

    sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) error_exit("Socket creation failed");

    // Set timeout
    struct timeval timeout;


timeout.tv_sec = 10;
    timeout.tv_usec = 0;
```

```
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    inet_pton(AF_INET, SERVER_IP, &server_addr.sin_addr);

    printf("Connecting to server...\n");
    if (connect(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        error_exit("Connection failed");
    }

    recv_prompt(sock, buffer, sizeof(buffer)); // "Username:"
    fgets(username, sizeof(username), stdin);
    username[strcspn(username,"\n")] = 0;
    send(sock, username, strlen(username), 0);

    recv_prompt(sock, buffer, sizeof(buffer)); // "Password:"
    fgets(password, sizeof(password), stdin);
    password[strcspn(password,"\n")] = 0;
    send(sock, password, strlen(password), 0);

    int len = recv(sock, buffer, sizeof(buffer) - 1, 0);
```

```
if (len <= 0) {
    fprintf(stderr, "[!] No response after login\n");
    close(sock);
    return 1;
}
buffer[len] = '\0';

if (strstr(buffer, "failed")) {
    printf("[!] Authentication failed.\n");
    close(sock);
    return 1;
}

printf("[+] Login successful. You can upload, download or exit.\n");

while (1) {
    printf("\nChoose option:\n1. Upload\n2. Download\n3. Exit\nChoice: ");
    fgets(input, sizeof(input), stdin);

    if (strncmp(input, "1", 1) == 0) {
        send(sock, "upload", strlen("upload"), 0);
        printf("Enter filename to upload: ");
        fgets(filename, sizeof(filename), stdin);
```

```
    filename[strcspn(filename, "\n")] = 0;

    if (strlen(filename) > 200) filename[200] = '\0';

    send(sock, filename, strlen(filename), 0);

    FILE *fp = fopen(filename, "rb");
    if (!fp) {
        perror("File open failed");
        continue;
    }

    while (!feof(fp)) {
        int n = fread(buffer, 1, MAX_BUFFER, fp);
        if (n > 0)
            send(sock, buffer, n, 0);
    }
    fclose(fp);
    send(sock, "EOF", 3, 0);
    printf("[+] Upload complete.\n");

} else if (strncmp(input, "2", 1) == 0) {
    send(sock, "download", strlen("download"), 0);
```

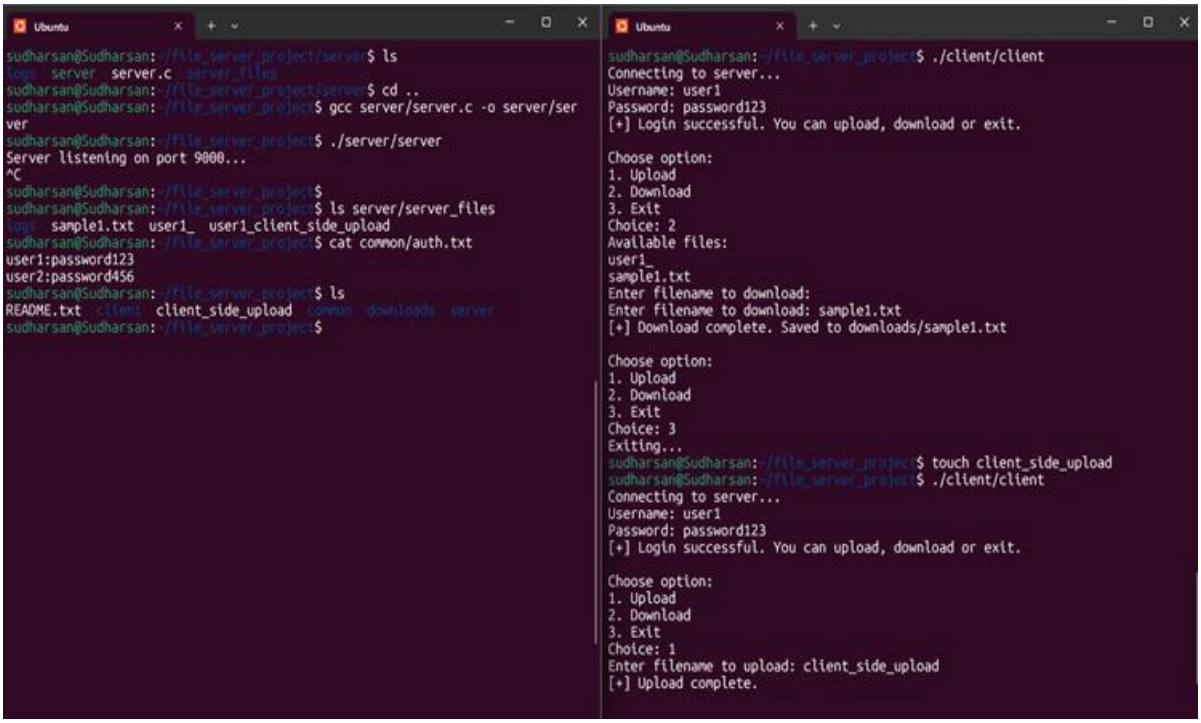
```
        recv_prompt(sock, buffer, sizeof(buffer));
        printf("Enter filename to download: ");
        fgets(filename, sizeof(filename), stdin);
        filename[strcspn(filename, "\n")] = 0;

        if (strlen(filename) > 200) filename[200] = '\0';
        send(sock, filename, strlen(filename), 0);
        mkdir("downloads", 0777);
        snprintf(full_filename, sizeof(full_filename), "downloads/%s",
filename);
        FILE *fp = fopen(full_filename, "wb");
        if (!fp) {
            perror("File create failed");
            continue;
        }
        while ((len = recv(sock, buffer, MAX_BUFFER, 0)) > 0) {
            if (strncmp(buffer, "EOF", 3) == 0)
                break;
            fwrite(buffer, 1, len, fp);}
        fclose(fp);
        printf("[+] Download complete. Saved to downloads/%s\n", filename);
    } else if (strncmp(input, "3", 1) == 0) {
        printf("Exiting...\n");
```

```
            break;
        } else {
            printf("[!] Invalid choice. Try again.\n");
        }
    }
    close(sock);
    return 0;
}
```

5. Screenshots

Screenshots of terminal-based client/server interactions



6. Features

Secure authentication system
Multi-client support using pthreads
File upload and download
Path sanitization
Logging of client activities

7. Limitations

No encryption for file transfer
Hardcoded user credentials
No concurrent file access handling
Basic GUI with limited features

8. Conclusion

The Multi-Client File Server demonstrates core Operating System principles in a practical setting. It allows users to understand system-level programming, threading, and file handling through a functional project. While it serves as a robust academic prototype, it can be enhanced further with security features and scalability improvements.

9. References

1. W. Richard Stevens, 'UNIX Network Programming, Volume 1: The Sockets Networking API', Pearson Education.

2. Beej's Guide to Network Programming

3. The GNU C Library Documentation

4. POSIX Threads Programming