

Systems Software Report CA2

TU857

BSc in Computer Science

**Conor Farrell**

**C19704439**

School of Computer Science

TU Dublin – City Campus

**05/05/2023**

**Table of Contents**

Table of Contents

[*Functionality Checklist* 3](#_Toc129253098)

[*Feature 1 - Client Program* 4](#_Toc129253099)

[*Feature 2 – Server Program* 4](#_Toc129253100)

[*Feature 3 – Multithreaded connections* 4](#_Toc129253101)

[*Feature 4 – File Transfer* 5](#_Toc129253102)

[*Feature 5 - Transfer Authentication* 5](#_Toc129253103)

[*Feature 6 – Synchronisation (Mutex Locks)* 5](#_Toc129253104)

[*Conclusion* 5](#_Toc129253105)

# *Functionality Checklist*

|  |  |  |
| --- | --- | --- |
| ***Feature*** | ***Description*** | ***Implemented*** |
| F1 | Client Program | Yes |
| F2 | Server Program | Yes |
| F3 | Multithreaded Connections | Yes |
| F4 | File Transfer | Yes |
| F5 | Transfer Functionality | Yes |
| F6 | Transfer Authentication | Yes |
| F7 | Synchronisation | Yes |

Have you included a video demo as part of the assignment: Yes

Link to Video: please paste link here

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Conor Farrell

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Conor Farrell

05/05/2023

# *Feature 1 - Client Program*

The client program communicates with the server program. It will first get the current user's ID and checks if the user belongs to the manufacturing or distribution group, if not the program exits. It then creates a socket connects to the server and sends the client's username to the server.

Text

Description automatically generated

Figure 1 Successful connection from client.

Next, the program then enters a loop where it prompts the user for a command, either "upload" or "exit". If the user enters "exit" the program sends the command to the server and exits. If the user enters "upload" the program sends the command to the server and waits for the acknowledgment. If the server is ready the program prompts the user for a file path and checks if the file exists. If the file exists the program sends the file name, file size and file data to the server.

Text

Description automatically generated

Figure 2 Client command prompt.

This program uses the C standard library functions to achieve its functionality including geteuid(), getpwuid(), getgrouplist(), malloc(), free(), socket(), connect(), send(), recv(), fopen(), fread(), fclose(), and printf(). It also uses standard C header files and network related header files like arpa/inet.h, netinet/in.h, and sys/socket.h.

# *Feature 2 – Server Program*

The server program creates a server that listens on port 5000 for incoming connections from clients. It binds to the IP address of the machine it is running on and creates a new thread to handle each client connection.

Text

Description automatically generated

Figure 3 Server bootup.

The handle\_client function is responsible for handling each client and first receives the client's username. It then checks if the username belongs to either the "manufacturing" or "distribution" group by getting the user's group list and checking if the user belongs to either of these groups. If the user belongs to either of these groups, the server remembers this by setting a flag and the client can then perform certain actions on the server directory that corresponds to the group they belong to. If the user does not belong to either of these groups the server does not allow the user to perform any actions and disconnects them. The server sets permissions on the directories for the "manufacturing" and "distribution" groups to ensure that only users belonging to these groups can access them.

Text

Description automatically generated

Figure 4 Server refusing non-authorised connection.

# *Feature 3 – Multithreaded connections*

When a connection is accepted using the accept() function the server creates a new thread to handle the client using the pthread\_create() function. The thread calls the handle\_client() function passing the client's socket as an argument.

Since each user has their own thread, they can all perform the functions simultaneously. Each client is identified on the server by their username aswell as some minimal logging being implemented. This includes when a client wants to upload a file, successfully uploads a file, unsuccessfully uploads a file, or disconnects.

Text

Description automatically generated

Figure 5 Server logging.

# *Feature 4 – File Transfer*

When the client sends an "upload" message, the file transfer begins. Following this the client is prompted to enter a file path. Before the message is sent to the server file path validation is also performed on the client side. When this is complete and the server receives the file path it opens the file and reads it into a buffer. This permits the server to copy all the data into a new file in the user's preferred directory. For instance, if john a member of the manufacturing group sends a file, it will be saved to the manufacturing directory. During this process, the original filename and file extension are both preserved.

Basic logging is also implemented, this includes when a client wants to upload a file, fails to upload a file or successfully uploads a file.

Graphical user interface, application, Teams

Description automatically generated

Figure 6 File transfer process

# *Feature 5 - Transfer Authentication*

Transfer authentication is implemented through the client roles sent to the server. The server will use the getgrgid() command to compare the users groups against the authorised groups, in this case it would be “manufacturing” and “distribution”. The same method of checking groups is implemented both on server and client-side. Since we use the getgrgid() function among others, there is no need for the client to manually tell the server which group they belong to or what their current role is as it will automatically be detected.

Should a user fail these checks they are booted from the server and the client program is shutdown.

A screenshot of a phone

Description automatically generated with low confidence

Figure 7 Folders for each group

When a client uploads a file to the server, the server saves the file with the login credentials of the client who initiated the transfer. As a result the client becomes the owner of the uploaded file and only they have the rights to modify or delete the file.

By making the client who uploads the file the exclusive owner the system ensures that only they can make changes to the file reducing the risk of data loss.

Text

Description automatically generated

Figure 8 Setting file permissions.

# *Feature 6 – Synchronisation (Mutex Locks)*

Each thread handles a separate client connection and works independently. The handle\_client function does not access any shared resources that can be modified by multiple threads simultaneously, therefore there is no need to implement mutex locks and file synchronisation is not necessarily an issue.

Mutex locks are used to prevent concurrent access to shared resources, and their use can potentially impact performance in multi-threaded applications.

File synchronization may also not be necessary in this case. File synchronization is used to ensure that multiple threads or processes accessing the same file do not interfere with each other's operations. In this case, each thread is handling a separate client connection and each client is uploading a different file.

# *Conclusion*

This project met all of the requirements. It was able to build a server that can accept numerous client connections. The server can manage several client requests at the same time by employing multi-threading, which is a fast approach to handle multiple requests.

When a client uploads a file to the server the file is saved to a specified directory determined by the client's role on the server. This aids in keeping the files structured and accessible. After receiving and saving the file the server sends a response back to the client confirming the successful reception of the file.