# *Feature 1 - Client Program*

* The client program is an implementation of that can receive files from clients over a TCP/IP network
* The error function prints an error message and exits the program in case of an error
* The main function takes four parameters which are User ID, Group ID, filename and path
* The program creates a socket with the ‘socket’ system call using the ‘AF\_INET’ address family, ‘SOCK\_STREAM’ socket type and a protocol value of 0. In case of an error, the error function is called
* The program retrieves the server’s IP address using the ‘gethostbyname’ function
* The program sets up the ‘serv\_addr’ structure with the IP address and port number
* The program connects to the server using the ‘connect’ system call, passing the socket file descriptor, server address structure, and the size of server address structure
* The server sends the User ID and the Group ID to the server using the ‘write’ system call
* The system opens the file specified using the ‘open’ system call with the ‘0\_RDONLY’ flag
* The program reads the file data in chunks of 256 bytes and sends each chunk to the server using the ‘write’ system call
* The system call ‘close’ closes the file
* The system reads the server’s response using the ‘read’ system call and prints onto the console
* The program closes the socket using the ‘close’ system call

# *Feature 2 – Server Program*

* The ‘SERVER\_PORT’ constant specifies the port number on which the server listens to connections. The ‘MAX\_CLIENTS’ constant specifies the maximum number of clients that can connect to the server at one time. The ‘Client’ struct contains the socket file descriptor of the client as well as its User ID and Group ID. The ‘clients’ array is used to store information about the clients that are connected. The ‘num\_clients’ variable is used to store the number of clients connected. The ‘clients\_mutex’ is a mutex that protects access to the ‘clients’ array
* The ‘handle\_client’ function takes a client struct as an argument and is executed in a separate thread for each connected client. The function receives a filename from the client and opens said file for writing. Then, it receives file data from the client and writes it to the file. Once the file is received, a confirmation message is sent to the client and afterward it is removed from the array of connected clients

# *Feature 3 – Multithreaded connections*

Multithreaded connections are used to handle multiple clients at one time.

* The main thread creates a socket and binds it to a port. It then listens for any incoming connections using the ‘listen()’ system call
* If a client connection request is received, the ‘accept()’ call is used to accept and establish the connection and create a socket. This new socket is then sent to another thread, and the main thread once again listens for any incoming connections
* The new thread created after accepting the request reads the Used ID, Group ID and filename from the client using the ‘read’ system call
* The new thread opens the file requested by the client and reads the content, and sends it back to the client using the ‘write()’ system call
* The new thread closes the client socket and the main thread keeps listening for incoming connections

To conclude, each client connection is handled by a separate thread. This functionality allows the system to work with multiple clients simultaneously

# *Feature 4 – File Transfer*

In server.c, when the server has received User ID, Group ID and filename from the client, it creates a new file in the server’s directory with the given filename. Then, it reads the data given by the client and writes that onto the new file in chunks of 256 bytes.

In client.c, after the client has sent User ID, Group ID and filename, it opens the file specified by the filepath and reads the data which is received in chunks of 256 bytes. It then sends each chunk to the server using the ‘write()’ function. Once the server has received all data, it closes the connection and sends a transfer successful message to the client. The client prints this message to the console.

# *Feature 5 - Transfer Authentication*

The transfer authentication in the program is achieved using User ID and Group ID. Whenever a connection is established with the client, it sends the user and group IDs to the server.

On the other hand, whenever a connection is established with the client, a server reads the user and group IDs and checks if the user ID is part of the group or not. If the user is not part of the group, the server denies access and sends a message back to the client and terminates the connection. If the user is a member, the server grants access and sends the same message back to the client.

# *Feature 6 – Synchronisation (Mutex Locks)*

The synchronization between threads is achieved using mutex locks. To be more precise, the ‘pthread\_mutex\_t’ variable ‘lock’ is initialized at the beginning of the main() using the ‘pthread\_mutex\_init’.

Whenever a thread wants to access any shared resource, it has to acquire the lock using the ‘pthread\_mutex\_lock’ function. It locks the thread until it becomes available. Once a thread has finished accessing the shared resource, it must release the lock using the ‘pthread\_mutex\_unlock’ function.

This mechanism ensures that only one thread has access to writing the file at one time and prevents data corruption.

# *Conclusion*

To conclude it can be said that the code implements a server-client file transfer mechanism using sockets. It is a multi-threaded program that listens to incoming connections from clients and uses synchronization to handle multiple requests from clients.

The client can connect to the server and sends the User ID and group ID along with the filename and file path. It then reads the file and sends it to the server for storage. The server stores the file with a unique ID and sends back a confirmation message.