**CUSTOM SOCKET BASED MILITARY ALERT SYSTEM**

**1.2 PROJECT PURPOSE**:

The purpose of this project to build an alert system for military using custom socket communication.

**1.2 FUNCTIONALITIES OF THE SYSTEM:**

These functionalities have been implemented in the program:

* PUT: Transfer a file from client to server.
* GET: Transfer file from server to client.
* MGET: Transfer all requested station information from server to client.
* MPUT: Transfer all requested station information from client to server.
* If the requested station is not present, then it will show the message station not found.

**Compiling the code:**

* On Server’s terminal $ gcc server.c -o server On Client’s terminal $ gcc client.c -o client
* To run the code, First on the Client's terminal run $. /Client
* Then on the Server’s terminal $. /Server
* Commands: Enter a choice: GET PUT MGET MPUT QUIT
* Closing Client and Server Use command “QUIT server” to quit client program. This will disconnect the client.

**1.3 OPERATING ENVIRONMENT:**

Operating environment for Emulating SFTP using TCP socket are:

* Client/server system
* Operating system: Linux
* Platform: Ubuntu/C

**3. UNIT TEST:**

1. **Client:**

MAS1-UT1: Provide valid username.

MAS2-UT2: Create a valid socket to initiate the communication

MAS3-UT3: Client must pass the request

MAS4-UT4: If the connect () function fails then the client must receive an error message of connection failure.

MAS5-UT5: If read () or write () function fails then the client won’t be able to send or receive the data

1. **Server:**

MAS1-UT1: While defining the structure mention a valid element.

MAS2-UT2: The port no. in the client side and server side should be same if not connection does not establish.

MAS3-UT3: The bind () and listen () should bind the client and server if fail must return the Binding error while listen () function must listen on port number.

MAS4-UT4: Send and receive the request

Use the read () and write () file descriptor to send and receive the data from server.







No





**3. FLOW DIAGRAM**

1. Start

This is the start block which indicates the start of the program.  
which will accept the client and server credentials (like the username and password). On validation of these credentials the system will allow the client and server to further communicate else if the credentials are wrong, it will display an error message indicating “Invalid Credentials”.

1. Login Credentials

In this module the credentials entered by the server/client are then validated by the

system. If the server/client enters valid credentials, then it will move to the further step else the system will prompt the server/client with an error message.

1. Request the Station name

Once the server has validated the credentials, client will now be connected to the server’s port number. Once connected to the server’s socket the client/server will send the request.

1. Catching the station name

The server socket will be created and will be binded to the client’s port no and now the server will remain listening on the port waiting for any client connection. Once a connection is established between the client, the server/client will catch the station request.

1. Check for valid station name

Once the server/client gets the station name, it will check the format of station name, if it is valid it checks for existence otherwise displays an error message that enter valid station name.

1. Check if station name exists

Once the station name is validated for its format, it checks if the station name is present in the list or not. If the entered the station name is present in the station list, it will move to the next step else it will prompt the user with an error message that station name does not present in the list.

1. Server/Client provides or receives respective station details

When a station name corresponding to the request is found, the server/client will give or receive the respective station details to the client.

1. If station name is not present it displays error message
2. End

This ensures that the program has terminated.

**CONCLUSION:**

In this project first, Client/Server will send a join request to join the multicast group. Then whichever station it selects from the station list, it is connected to that station. Whenever receiver connects to a particular station, it starts sending and receiving information from that station.

The application has two primary parts - Client and Server - and the reliable transfer is ensured using TCP/IP with help of socket () API specifying SOCK\_STREAM.

**The Client’s flow:**

Client will send the join request to the server. Once server accept the request it will send station detail request and waits for the server response.

**The Server’s flow:**

Opens a connection and starts listening to incoming connections from client and then

binds the listening file descriptor to the specified port, and establishes the connection,

Creates a fork of its process for every incoming connection, closing listenfd for that fork, receives client’s request then sends and receives the necessary information to the client.