Design a client-server system that follows the polling type MAC protocol. There is one server and multiple clients connect to it (the server). The server polls each client one by one (in a specific sequence) by sending POLL as the keyword to the client.

If the client has a message for a server or another client, then it will send the message (keyword followed by options if present) to the server when polled (client receives the POLL message). The message is taken via the command line. If the polled client has no message for server/another client, the client responds back with keyword NONE. If a client however has some message for the server, then the client can use one of the following keywords – LIST, EXIT, and SEND <username> <message> to send the message to the server.

* When client sends LIST to the server, the server will send a list of all connected clients’ name. When a server receives LIST, it ignores any option that the user includes after the keyword LIST. (you don’t have to perform this validation) Server responds to LIST keyword in the same cycle as the request. See diagram below.
* When client sends EXIT, the server removes the client from the polling loop; the client exits its application.
* When client sends SEND <username> <message>, the server will send the <message> to the <username> client. The server does this by sending MSG <sender Client> <message> to the receiver (<username>). After sending MSG to <username>, the server continues polling. If SEND is sent with less than 2 arguments, the server ignores the message and continues with its polling. See the two examples in the figures below.

A client is always in receiving mode, i.e., it is waiting for a command from the server, until it receives the POLL keyword. When it receives the POLL keyword, it has to transmit a message (one keyword among NONE, LIST, EXIT, and SEND <username> <message>) **that it takes as a command-line input** from the user. After sending the keyword, the client switches over to the receiving mode again.

Keywords to implement:

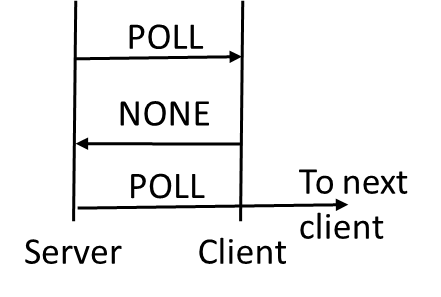
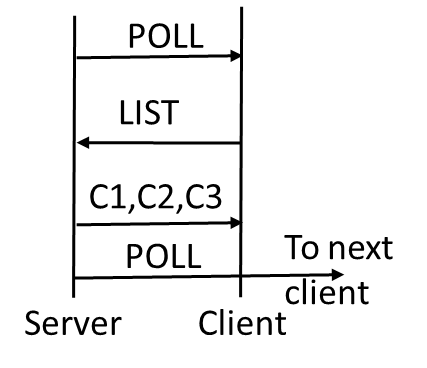
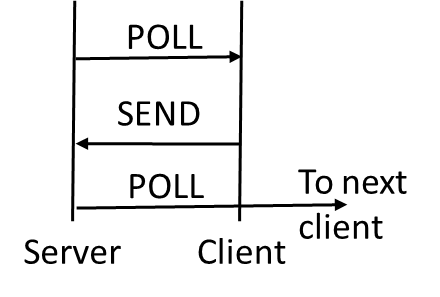
At client: NONE, LIST, SEND, EXIT

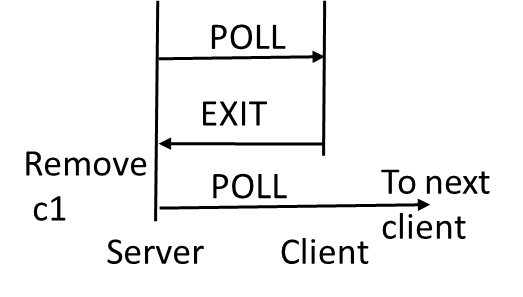
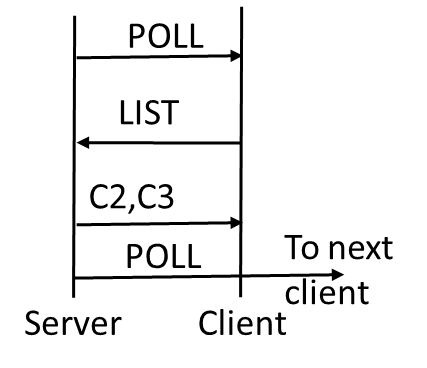
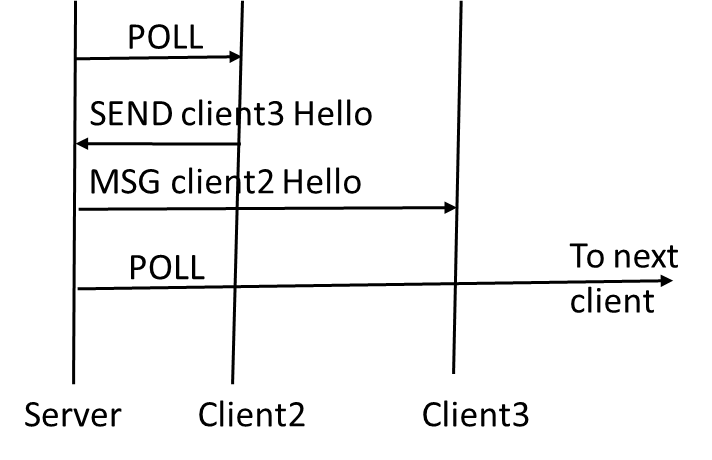
At Server: POLL, MSG

Deliverables:

1. The server, as a command-line argument, accepts the port to which it should bind (2 marks). After this step, the server does not take any command-line input.
2. The clients (start 3 clients), as a command-line argument take:
   1. The user’s username (you don’t need to perform validation in this compre. Just keep the usernames different from each other) (1 mark)
   2. The server’s IP address (1 mark)
   3. Server’s binding port number (1 mark)
3. The server stores each client’s username, IP address, port number, and any necessary details in a data structure of your choice (3 marks). Please ensure that at least 3 clients have connected with the server at the end of this step.
4. The server sends a POLL message to one client, waits for its response and then sends the message to the next client. The polling should work for at least 3 cycles (each client will be polled atleast thrice in the order client1, client2, client3, client1,…). Hint: The client polling order can be the same as the order in which they connect to the server (4 marks)
5. When client 1 receives the POLL message from the server, it responds with NONE. This (NONE) is taken as a command line input. The server, on receiving NONE from client1, then polls client2. (2 marks)
6. When client2 receives the POLL request from the server, client2 responds by sending LIST to the server (taken as command-line input). Server, in response to the LIST message, sends a list of all usernames connected to it (the server) at that moment. The server may use a specific delimiter to indicate the different clients connected. You can display the usernames in any style of your choice (4 marks). The server then POLLs client3.
7. When client3 receives the POLL request, it responds by sending a SEND message without any arguments. The server discards the message as the receiving username and message are not mentioned (3 marks).
8. When the next client (client1) receives the next POLL request, it responds by saying EXIT. The server removes it from the list and the client terminates itself. (3 marks)
9. When the next client (client 2) receives the next POLL request, it responds by sending SEND <usernameOfClient3> Hello. (Replace usernameOfClient3 with the actual name that you used). The server sends this message to client3 after modifying the message to MSG <usernameofClient2> Hello. Client3 should display this message on its terminal, indicating that the message came from client2. You may format the message in any way of your choice. (4 marks)
10. Next the server sends POLL to client3. Client3, on receiving the POLL request, sends LIST command to the server. Since client1 has already left the conversation, the server sends only the username of client2 and client3. (3 marks)

The above sequence of execution is for ensuring you capture all use cases. We will supply the keywords in a different order to test your program. Ensure that the client program takes the keywords as command line arguments.

What to submit:

The Server.c code file, the Client.c code file, a README file that should mention how to compile your code, how to execute your code, and a sample run of the code.