CS 39006: Lab Test 1- Set B

Date: February 11, 2025

Important Note:

You have to follow the instructions and variable names given in this problem statement. Anything which is not given can be assumed; however, you should clearly write your assumptions at the beginning of the code.

Problem Statement:

You have to develop a multicast game application using stream sockets, where there will be multiple (two or more) clients connected to a server. The game works as follows. The server waits for an integer number from all the connected clients. The clients, on connecting to the server, send an integer number to clients on each round. Once the server receives the numbers from all the clients corresponding to a round number, it announces which client has sent the largest number.

Server Functionality: To do this, you have to implement a game server which will initially wait for the clients over a stream socket. It will wait for at least two game clients before the clients are ready to send/receive messages. Consider that there will be *a maximum of five clients*. The game server maintains an array of socket file descriptors, called `clientsockfd[]`, and waits over a select call for the incoming connections or messages. It can be noted that the first two instances when `select()` will exit are the two connection requests, as you need at least two game clients for the communication to begin with. After that, the `select()` can exit either because a new connection request from a new client has arrived, or some client has sent a message.

If the server receives a new connection, it adds the connection to `clientsockfd[]`, increments a counter `numclient`, and include the new client socket file descriptor `clientsockfd[numclient-1]` to the `select()` read file descriptors. The server also prints the following message at the console.

```
Server: Received a new connection from client <client IP: client Port>
```

Note that every client will be uniquely identified through the <IP, Port> pair. The server then asks for an integer number from the client corresponding to Round N (the round number is maintained as a counter value at the server) by sending the following message to each of the connected clients.

```
Server: Send your number for Round <N>
```

Note that if a client joins when the server is waiting for the numbers for say, Round i, then it considers the new client to start with the current round, that is Round i, and sends the above message after accepting the connection.

If the server receives a message from one of the clients, it checks whether the message is a duplicate for the current round (the client has already sent a number for the current round). In that case the server just ignores the message from the client and sends the following message to that client:

```
Server: Duplicate messages for Round <N> are not allowed. Please wait for the
results for Round <N> and Call for the number for Round <N+1>.
```

The server, once received the number from all the connected clients (including the newly added ones, if any), it then checks the maximum, and then sends the following message to all the clients:

```
Server: Maximum Number Received in Round <N> is: <X>. The number has been received from the client <IP>:<Port>
Server: Enter the number for Round <N+1>:
```

If the server receives a message before at least two clients have joined, it will print the following message.

```
Server: Insufficient clients, “<message>” from client <client IP: client port> dropped
```

Client Functionality: The client uses a stream socket to connect to the server. After the connection is established, it uses a `select()` call within an infinite loop to check one of the following.

1. The user has given a message (number) input through the keyboard. Note that the `STDIN` file descriptor is a standard POSIX file descriptor that can be checked to read data from the keyboard. You need to add `STDIN (STDIN_FILENO macro defined in unistd.h)` to `FD_SET` as a read file descriptor, and then use the `select()` call to wait on it. After the `select()` returns, you need to check whether something is available on `STDIN`, and if so, you can use a `read()` call to read the data from the keyboard. If the user inputs some number, then the client socket sends the number to the server and prints the following at the terminal.
   ```
   Client <Client IP: Client Port> Number <Input Number> sent to server
   ```

2. The client socket has received a message from the server. If a message is received, the client prints the following at the terminal.
   ```
   Client: Received the following message text from the server: <server_message>
   ```

Note that the client can input multiple numbers for a round. However, the server accepts only the first number sent by the client at a current round. Also, assume that the clients remain connected once they join the network. You do not need to handle connection closure.

Also you may use the function `int getpeername(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len)` to get the address for a client associated with a socket FD (sockfd returned from the `accept()` call). For details, check the man page.

Submission Instruction:

You have to submit the following two files: `chatserver.c` containing the server code and `chatclient.c` containing the client code. Put the files in a folder named `LT1_<Your Roll Number>` (for example, if your roll number is 22CS90098, then the folder name will be `LT1_22CS90098`. Compress the folder in a `zip` format and upload it on the MS Teams submission page.

You have to follow the submission instructions exactly, otherwise a 20% penalty will be imposed.