CITS3002 Project: Socket Programming Report

By Shane Monck (22501734)

I. *How would you scale up your solution to handle many more clients?*

The current solution is capable of handling large quantities of clients, each client process is assigned a different port number to a different client, meaning the port represents a unique connection so we can handle hundreds or even thousands of simultaneous connections. So, it can have as many connections as possible; the issue is the client experience, the solution only runs the game for four clients. This means that there will be a large stack of clients that are spectators, and due to the clients being selected at random to play the game some clients could be waiting long times to play. To solve this, I would update the code so that multiple games are running at once, communication from client to server would still essentially stay the same, but I would keep track in the server of what clients are in what game and what that games board state is. Communication would therefore only be between clients that in the same specific game together and the server. Clients that are in a different game would never communicate with other clients in a separate game.

II. *How could you deal with identical messages arriving simultaneously on the same socket?*

The messages that are sent to the socket are in a packet, the packet can be read and the messages within the packages can be deciphered, therefore you can determine the type of message sent. If identical messages appear at the same time and are going to be sent to a socket, the server can look at the messages being sent and decide to only send one of the messages and ignore the other. I would do this with the aid of timestamps, the server would look at the messages and if they are the exact same type and sent at the exact same time then they are identical, and then it would ignore any identical duplicates and send only one message to the client socket. If the timestamps are different then the message type could still be the same, but the messages are essentially not identical and then you would have the socket handle the simultaneous arriving messages and process them in order of the timestamps.

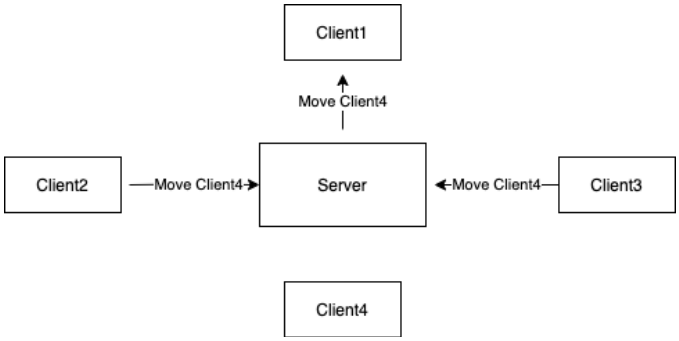

Figure 1- Identical messages sent at same time, Client1 only gets one of the messages

III. *With reference to your project, what are some of the key differences between designing network programs, and other programs you have developed?*

In other programs I've developed they have never communicated to other terminals i.e., different clients. They have always been processed on one terminal and an output comes from that one terminal. In this project, communication was down to multiple terminals (clients), as well in other projects I've never done multiple processes at once, processes were always done sequentially. In this network program, two processes were running at the same time; listening and accepting new connections, and the other was handling the connected clients.

IV. *What are the limitations of your current implementation?*

The limitations of my implementation become clear after running the test script that was provided too us, unfortunately my implementation struggled with updating clients board within a certain time leading to "not all clients boards equal" error from the test script, meaning the speed at which clients receive messages from the server is too slow, however manually running the program. It is never obvious to the client of this latency, the game runs smoothly and their board is updated correctly when events occur.

The implementation will also struggle with scalability, it will perform adequality with small number of clients but with a thousand clients it might fail to meet response time requirements due to the complexity increasing. A potential solution could be implementing more multi-threaded processes, as of right now there's only three threads; accepting connections, playing game, and a thread for when a client joins mid game. More threads that handle processes in the playing game thread could lead to a better performance.

V. *Is there any other implementations outside the scope of this project you would like to mention?*

For Tier 3 – Connection issues, the only disconnection scenario to fix was when a client exits mid game. My implementation however checks to see if a client has disconnected at any stage, i.e., before a game, after a game, middle of game, and for when they are a spectator during game. This is done with *is_socket_closed(connection)* this function is called whenever the server needs to send data about the board to the client. The function will "peek" into the socket to check if it is still active, if there is a *socket.error or ConnectionResetError* the client has therefore disconnected and so that socket will be closed and removed from server variables so they no longer get updates.

VI. *Any other notable things to discuss.*

As stated in **IV** my program will fail the test scripts tests, however sometimes it will pass them with SUCCESS, this is a strange occurrence and therefore the tests could be producing fake negatives, however manually the program runs smoothly. Potentially these failures could be due to the fact I implemented a countdown before a game begins, so that there is enough time for clients to join the server.
Also, I have implemented in the program the case where one client is connected when a game starts, so that the program will still run with one client and they are allowed to play a solo game.