CITS3002 - Computer Networks - Project Report

Ethan Chin - 22248878 & Daphne Yu - 22531975

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

Ensuring that there is enough memory space to hold all the clients data.
Vertical scaling can be used to handle more clients by upgrading the hardware such as memory and cpu to hold more memory and process information faster.
Three tier architecture can be used, storing player data in a separate server from the server that is processing each players data.
Having multiple servers in different locations that interact with different groups of clients that are in the same area or region. This will decrease latency for clients that are far away from the servers such as different countries.

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

A player's client can send two "INIT" packets and pretend to play as two players. This can be prevented by ensuring that there is only one player per socket connection. ie. each connection to the server will only be allow one player to be connected.

A client sends two of the same packet by editing the client code. This can be prevented by only reading what the client sends once. The second packet will just act as the next action or be ignored.

A malicious client pretends to be another client by changing the client id on the packet they send to the server. This malicious client can send an invalid packet using someone else's client id to get the removed from the game. This can be solved by having the server store the client's id that is sent to the client and ensuring that any packet sent from that client matches the one that the client was given. This way the server only kicks the player with malicious intent and the targeted player isn't affected.

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

Communication between different programs (client and server) that use different languages (python and c).

Time component. While for other programs, we execute and prefer it to finish as quickly as possible, this network program requires looking after details such as arriving sequence of the message, life cycle of the message delivered via pipe (read and write by whichever party), slotting in time interval using sleep() command if necessary.

Inter-process communication. Using techniques such as piping to transfer messages between processes, and shared memory to hold stored memory for all processes, especially new processes, to read. In this project, two pipes were used. p1 child data to host and p2 host data to child. Where each child is responsible for a player's client and the host is the process that is not responsible for any clients which handles the overall game state (number of players).

Multiple processes can be used to connect multiple clients to the server. The program need to make sure its child processes run concurrently/ at the same speed, so to keep the communication portal e.g. pipe clean with only correct data flowing.

Process synchronisation, where multiple processes reach agreement and have identical local data, e.g. number of players, stored in each process.

Connections to the server are unreliable where players can suddenly disconnect from a game. The server must not fail when a player disconnects and is able to function normally for the other players. The game must continue to run even if some players lose connection.

The concepts used in this project are the same as what is used in modern day multiplayer gaming. Having a client that the player has to be able to send packets of data to a server that can be in a different networks. The server does all of the data storage and processing, while the client only sends its actions and receives its own results.
Lots of multiplayer games sometimes don't consider that people will try to cheat by editing the client code to gain an unfair advantage. As a result, many games will have "hackers" that make the game unfair for others. In recent years, game developers have started implementing anti-cheat detection in their servers to be able to detect and ban hackers. They consider ways that the player can attempt to cheat and then find solutions to prevent that from happening while maintaining functionality for normal players.
This was considered when processing the packets received from the client by using the function `watch_dog` which processes and validates that each packet from the client is a valid packet.

The skills learned here can be used to develop actual networking for games as sockets can connect different computers on different networks such as a client to a server.

What are the limitations of your current implementation (e.g. scale, performance, complexity)?
There is a limited amount of memory available in the pipe for each time data from the host process is sent to the child processes. Also, new memory is allocated for each player that connects to the server. When scaling for more players, the memory that is used will increase and won't be able to handle more players due to limited memory usage. This can easily be fixed by upgrading hardware to have more memory available on the server and optimize the amount of memory used in the code.

Python input limitation. This program uses request mode to interact with users in the real-time, although timeouts is introduced to prevent the server side from hanging when not receiving response from the player, player cannot escape request mode until input is entered or interrupt signal is caught; therefore, there is a chance of missing important message update sent by the server.

The dice is rolled in separately in each child process instead of just one dice roll. If there is a delay between the host and the child process, the dice rolls can be different for each player.

Notes:
If there is only one player, the game will go into single player mode where you can play by yourself.
Edgecase - if 2 or more is eliminated on the same round, then all that was eliminated that round wins.
Players start with 3 lives, this can be changed on line 292.

TODO list / contributions
Tier 1 - Done
Server side: - Ethan Done

- reads the players move
- rolls a dice and calculates the score,
    - if the same srand value is used then each client will get the same result
- then sends the information to the player

Client side: - Daphne Done

- Add pass, fail, elim, vict, reject, cancel

- Time out on moves - Daphne Done
- Comments to understand the code - Daphne Done
- End of game, tear down - Daphne Done

Tier 2 - Ethan Done

- Multiple processes to handle multiple players

Tier 3 - Ethan Done

- Handle players joining mid game, reject, not just an if statement - Done
- Handle exiting mid-game

Tier 4

- Anti-Cheat detection - Ethan Done
- The server must wait for some reasonable amount of time (~30s) before starting a game. If the lobby does not fill in time (>= 4 players connect) the match is cancelled and connections are torn-down gracefully - Daphne Done

Extras

- Function tear down, end of the game, send FIN - Daphne
- Single player mode - Daphne Done
- Separate into small functions - Daphne
- Edgecase - Ethan Done
- Dedicated host process, to roll dice, calculate new player count - Ethan Done