Waridh ‘Bach’ Wongwandanee waridh 1603722

Objectives:

- Learning how to create and control threads.
- Use signals to prevent blocking.
- Take advantage of named pipes to connect multiple processes on the same pc.
- Using poll for input-output multiplexing. Needed to take advantage of select signals for this purpose.

Design Overview:

Part 1:

- Will read from the file specified for the designated number of lines, and then pause for the specified amount of time.
- While not reading from the file, the program will accept command line arguments and proceed to run them on bash.
- If the User specified quit, the program would exit.

Part 2:

- There are two modes to this program, server mode, in which the program will take in command and client mode, where the program will read commands from an input file and send packets to the server to do tasks.
- The assumptions here are that there will never be more than one server at a time, and there will not be more clients than the limit placed by the assignment.
- To communicate, the server process will create a select pipe, where select signals will come in from client, directing the server to look at certain connections. The client itself will create the in and out pipe allowing duplex communication. Upon quitting the client, the two pipes are destroyed by the program, however the select pipe will continue to exist until the server is closed.
- Client will read command lines from an input file and send it to the server. It will then wait for the server to give a response before moving to the next command line. The behaviour is as specified on the assignment pdf.
- The server will wait for either a user to type in the command list and quit to either show stored objects or close the program respectively or when it receives an input in the select pipe. This pipe works as the select signal in a multiplexer and will point the server to the two half-duplex pipes it must connect with.

Project Status:

Part 1:

The program works exactly as specified, tested on the CS servers. There are no know issues.

The difficulty of implementing this part using threads is stopping the thread that is taking in user command line commands. I needed to use sigkill to send an asynchronous signal to the thread to end it. Otherwise, this part was a straightforward implementation.

Part 2:

The program works exactly as specified as well. It was tested rigorously on the CS Ubuntu servers and passed concurrency testing. It matches the output provided on eclass exactly.

The main difficulty with this part is that I initially planned on purely using threads to communicate between the client and the server, (i.e., the server will create a thread to communicate with the client when given the right signal.). What happened was that when the

server created a new thread to communicate with the client, it was unable to be detached, thus concurrency was not achieved. I swapped to multiplexer after reading the assignment rubric again and created an extra pipe that served as the select channel. From here, we call a repurposed function from the thread method to communicate with the selected channel. There were also some issues with closing the server-sided connection to the pipe after the client is ready to quit, but my solution was to send a final select signal that has the value of the client id, but as a negative number instead. This allows the server to run the close function on the appropriate pipe so that it could be reused by a different client.

Testing and Results:

Part 1:

I tested the program using the given example and the command line functions I wanted to look at. The input file that I used was generated as advised by the example given on eclass, and was never changed. I tested with different numbers of lines per burst, and also different wait times. Every result seems to have returned as expected.

Part 2:

Testing was somewhat simple in this part. I had a test input that is given by the example in eclass and I had a copy which I modified by adding more commands that would cause PUT to have repeating objects. After testing, it seems that all packets are working properly. The test files are included in the submission as a2p2.data and a2p2-ex1.dat.

Acknowledgements

I did not get any collaborative help. Mostly read the textbook for understanding the concept, and geeksforgeeks as the function manual.