Thomas Hari Budihardjo
UIN: 126009556

PA2 REPORT

The final product of the client server is a conglomeration of the smaller parts that I was tasked to create. The smaller parts include functions to request only one data point (from which I was able to rebuild a whole CSV file), to request an entire file to be recreated, and to request a new channel to be created. The flow of the program is finally clear after running the data server as a child process. Because of this, I was able to run only one program because the data server could automatically be run from client.cpp. Getopt was the next thing that needed to be implemented. Using this, the user is able to tell the program what processes they want to run as well as pass in the relevant arguments.

In order to allow the program to run even when given faulty information, I added some default cases for the program to fall back to. It will inform the user that the value that they typed in is incorrect and give a suggestion on what the allowed inputs are. The program will run if it is given 1) a patient name, a time argument, an ecg value; 2) just a patient name; 3) just the filename to copy; 4) request to create a new server. If a combination of these information is provided, then multiple processes will run, one after the other.

In a high level, this project requires two servers to talk to each other in order to give the information that the client requests from the data server. To start the communication, a pointer needs to be sent from the client to the data server by the cwrite command and the client server can read what the data server writes back by the cread command. This is the basis of the communication between the two servers.

A regular 100MB file would normally take 70.0471 seconds. This is due to the bottleneck of having a relatively small MAX_MESSAGE size of 256 bytes. I tried increasing the MAX_MESSAGE to 104857600 bytes (100MB) and the time it takes to transfer the data is only 0.010583 seconds. The time saved is due to the fact that I only need 1 transfer with this method. Basically, this is an example of the time-memory tradeoff, where a problem can be solved in less time if only more memory is used. Here, I can improve the time it takes by increasing the size of the packets that get delivered because the bottleneck in the program is the packet size.

As evidenced in the tables below, transferring data using data points is a lot slower than transferring the whole file. This is because the packet sizes in the data point transfers are a lot smaller than the 256 byte packets that get transferred using the file transfer.

Patient	-p	-f
1	79.4711	0.076875
2	79.5208	0.086204
3	79.4461	0.077941
4	79.4707	0.072237
5	79.4913	0.082004
6	79.4867	0.089824
7	79.5131	0.077197
8	79.4835	0.084873
9	79.5522	0.118479
10	79.4988	0.071362
11	79.5309	0.069984
12	79.511	0.090227
13	79.5193	0.074175
14	79.6147	0.076821
15	79.6161	0.085131

Size	Time
100	0.00088
256	0.000803
1000	0.001613

Size	Time
100MB	295.397
100MB improved	0.010583