PA2 Report
Sahil Palnitkar

The design of the program was mainly receiving input_args using getopt() and then executing parts of code depending on the included inputs. For the single data point call, the client creates a datamsg and writes it to the FIFO channel. The dataserver receives this datamsg, analyses it and writes the corresponding data point back into the FIFO channel, which can then be printed into the terminal output. Similarly, for copying the entire file point by point, the client sends 30,000 datamsg requests to the dataserver, as there are 15000 time points and 2 ecg values for each. These are then written to a copy of the original .csv file. The -f call performs the same function except it sends filemsg requests, and receives 256B of data at a time, instead of a single point and writes it to a copy of the file. Therefore it takes only filesize/256 requests to transfer the entire file. To create the new channel, the original channel creates the channel as a thread of itself and then makes it independent.

The dataserver is run as a child process of the client, so that the dataserver is only started up when a client is running. The dataserver terminates as soon as the client terminates and all channels are closed.

I am copying the .csv files in the BIDMC folder to the received folder using multiple methods. First I am copying the files using ./client -p. Secondly, I am copying them using ./client -f. These two methods have drastically varying times.

FileName	Time taken for all data points	Time taken for entire file
1.csv	79.437650 sec	0.379389 sec
2.csv	80.788921 sec	0.417158 sec
3.csv	81.162838 sec	0.494726 sec
4.csv	80.980552 sec	0.471813 sec
5.csv	81.553642 sec	0.378265 sec
6.csv	81.685758 sec	0.438473 sec
7.csv	81.483393 sec	0.357352 sec
8.csv	81.058438 sec	0.378632 sec
9.csv	81.184186 sec	0.288548 sec
10.csv	80.858446 sec	0.308997 sec
11.csv	81.141733 sec	0.358369 sec
12.csv	81.437654 sec	0.331625 sec
13.csv	81.118828 sec	0.387008 sec
14.csv	81.787336 sec	0.345167 sec
15.csv	83.766427 sec	0.355496 sec

The binary files I tested were 100B 256B, 1000B and 100MB

FileName	Time taken for entire file
100.dat	0.324874 sec
256.dat	0.379928 sec
1000.dat	0.310963 sec
100MB.dat	13.483304 sec

The 100MB file takes a while to transfer because it requires a large number of 256B file messages to be sent. This can be sped up by using multiple channels to send the file message requests concurrently.