

# CSCE 313 Programming Assignment 2

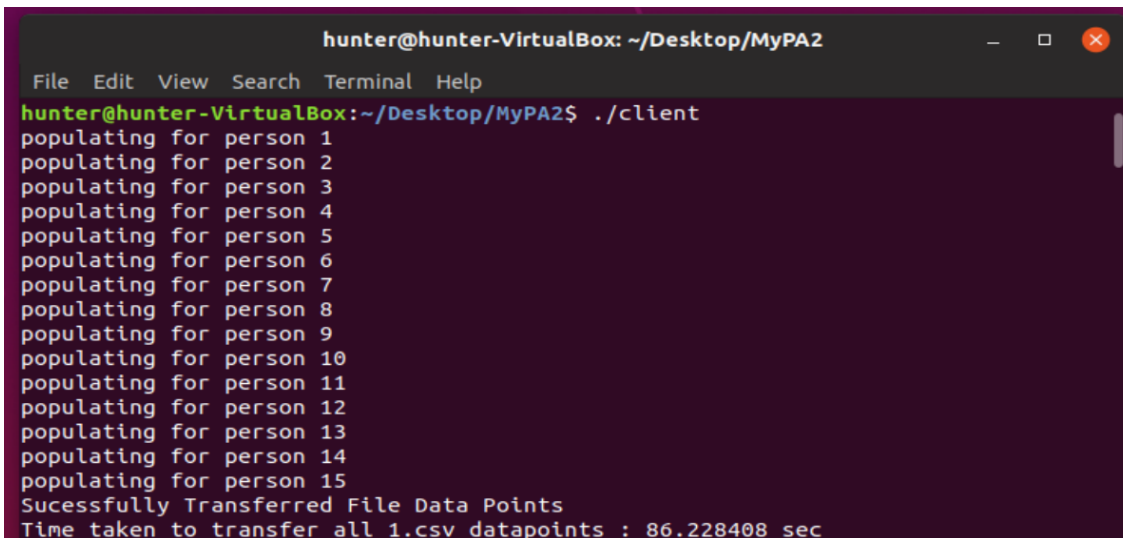
Hunter Cleary - hncleary - 625001547

February 2019

Client-Server Interaction

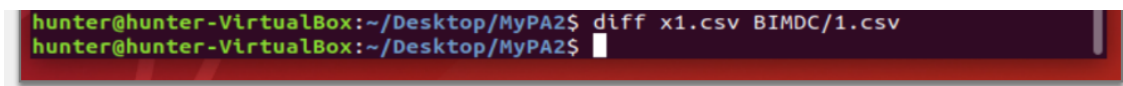
## 1 Requesting Data Points

Data points were requested from "1.csv" by sending a datamsg for every entry in the csv. The signal was written as a buffer, then read as a casted double once the data point had been retrieved from the original file. This process was lengthy and took 88.228 seconds.



```
hunter@hunter-VirtualBox: ~/Desktop/MyPA2
File Edit View Search Terminal Help
hunter@hunter-VirtualBox:~/Desktop/MyPA2$ ./client
populating for person 1
populating for person 2
populating for person 3
populating for person 4
populating for person 5
populating for person 6
populating for person 7
populating for person 8
populating for person 9
populating for person 10
populating for person 11
populating for person 12
populating for person 13
populating for person 14
populating for person 15
Sucessfully Transferred File Data Points
Time taken to transfer all 1.csv datapoints : 86.228408 sec
```

The file was then checked against the original using the diff command in the terminal to ensure that it was equivalent.



```
hunter@hunter-VirtualBox:~/Desktop/MyPA2$ diff x1.csv BIMDC/1.csv
hunter@hunter-VirtualBox:~/Desktop/MyPA2$
```

## 2 Requesting Files

Files were requested in a similar method used in the data point request. A filemsg was generate, then placed into a buffer with length allocated for the message, the file name, and the null delimiter. (filemsg + "filename" + '\0'). The data server processes the command to retrieve the correct data.

Because there is a limit on the size of the max message, the file had to be requested in smaller chunks. A filemsg was sent with parameters (0,0), allowing for the size of the file to be returned. Using this length, the file was divided into segments of 256 or less. The file was requested and the subsequently recieved/ written in these chunks in order to make the complete file.

```
282700 , 256
Server received request for file BIMDC/1.csv
282957 , 256
Server received request for file BIMDC/1.csv
283214 , 256
Server received request for file BIMDC/1.csv
283471 , 256
Server received request for file BIMDC/1.csv
283728 , 256
Server received request for file BIMDC/1.csv
283985 , 256
Server received request for file BIMDC/1.csv
284242 , 137
Time taken to transfer file 1.csv : 2.744349 sec
hunter@hunter-VirtualBox:~/Desktop/MyPA2$
```

This process was much faster than the individual data points request, due to the decrease in total requests to the server required. More data could be received and written at once, expediting the transfer. This method worked for both text and binary files.

### 3 Requesting a New Channel

Requesting a new channel was as simple a command as any. A new class constructor for message type NEWCHANNELMSG was added into the common.h file. This allowed the message to be generated and sent in the buffer. The would request a new channel to be created by the dataserver. This channel was automatically named to "data1\_" , and the integer would increase to 2 if another channel were to be created, 3 if yet another, and so on. The channel could be connected to through a simple request to create the other end on the client side.

```
FIFORequestChannel channel2 ("data1_", FIFORequestChannel::CLIENT_SIDE);
```

After this request, the channel behaved exactly like the original base "control" channel. The channel was tested with sample data messages.

```
Channel :data1_ created.
Testing Data Point Request on New Channel ...
(2,.008,1)
-0.450000
Testing Data Point Request on New Channel ...
(1,.012,2)
-0.450000
hunter@hunter-VirtualBox:~/Desktop/MyPA2$
```

## 4 Running the Server as a Child Process

```
17  int main(int argc, char *argv[]){
18      int pid = fork();
19      if ( pid == 0 ) {
20          char *args[] = { "./dataserver", NULL};
21          execvp(args[0], args);
22      }
23      else {
24          int n = 100;    // default number of requests
```

The data server was ran as a `fork()` process within the client. The child process `dataserver` starts, then the client parent process runs and connects to the server. It then executes all of the client requests, then waits, and then finally sends a quit message to the server to end the child process.

```
224
225  quitmsg q = quitmsg();
226  char* quitbuffer = new char [sizeof(q)];
227  *(quitmsg*)quitbuffer = q;
228  chan.cwrite( buffer, sizeof(quitmsg) );
229
230
```

## 5 Requesting Large Files - Bottlenecks

I was unable to get the function to work correctly on the large generated files. But, it would be expected that to transfer the file it would take a linearly proportional amount of time due to the `MAX_MESSAGE` size limitation. If larger chunks of data were allowed to be transferred over the channel, the process could be expedited. The rate of data moved through the channel would increase, which is the primary source of any bottlenecking.