Ian Stephenson
UIN: 927004123
CSCE 313-501

Youtube Link: https://youtu.be/YU1w2VcJdDA

Here, I document the design I followed to implement the client-server process for PA1. When I first began writing this program, I ran into an initial problem that forced me to rethink my design. This issue was, while using getopt, there needs to be different fundamental processes to run the same command with varying arguments. For example, the logic for getting a single data point is different from the logic used to get a thousand data points, so you have to differentiate between how many command arguments they take. To do this, I utilized argc, which is passed to main and tells how many command line arguments exist. So, when running -f, it should be taking 3 arguments, ./client, -f, and a filename. So if there are 3 arguments or fewer, the getopt function will parse arguments, and it will run the -c or -f processes depending on what it finds. Furthermore, if argc is between 3 and 7 arguments, it will be running the -p -e function regardless, so the logic to get a thousand data points is run. In the event that argc is greater than seven, it will run the -p -t -e function, so the logic is contained in there. This logic functions to decide which command line task needs to be executed.

Now onto the data. The following is a data table for the process of getting a single data point. The time is in microseconds and is calculated using the gettimeofday function.

t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	avg
4925	5916	5958	5825	7263	6336	5795	4699	4454	5848	6284.4

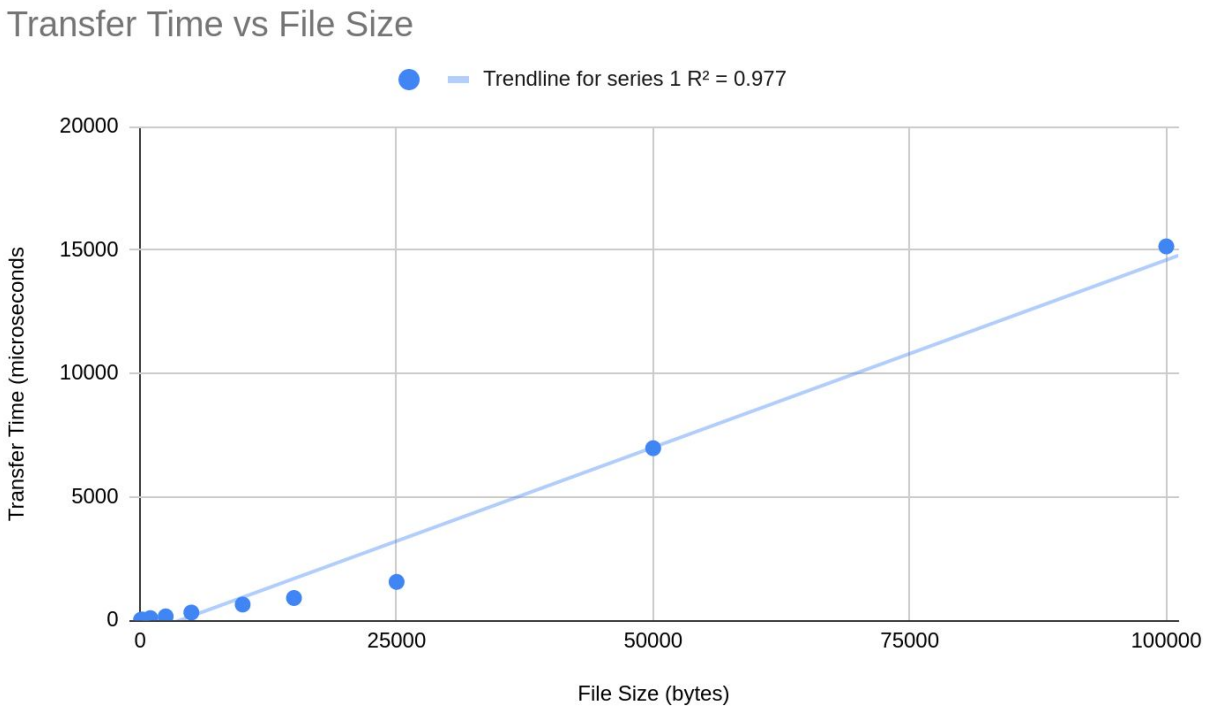
Below is a table describing the time needed to get 1000 data points, as well as how long it would take to get a single data point out of there. The single data point is calculated as the total time divided by 1000 data points. The first row is in seconds and the second row is in microseconds. The data here is truncated to two decimal places.

t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	avg
2.83	2.81	2.78	2.84	2.81	2.80	2.82	2.82	2.83	2.81	2.815
2830	2810	2780	2840	2810	2800	2820	2820	2830	2810	2815

This following data table displays the differences in the time in microseconds needed to get files of different sizes.

	t1	t2	t3	t4	t5	avg
100 Bytes	23	26	29	27	25	26
256 Bytes	27	29	26	27	29	27.6
1000 Bytes	73	74	95	73	125	88

Within the data above, there is an interesting question that arises. Why do the files of 100 bytes and 256 bytes download in the same time? This has to do with the buffer. The buffer size that was used to collect this data was 256 bytes, so the program was able to collect this data in one run in both situations. Also, as expected, the 1000 byte file was, on average, collected in between 3 to 4 times that of the 256 byte file. To see how this relationship changes over time, I'm going to create a scatterplot showing the relationship between file size and transfer time below.



As seen above, there is a clear linear relationship between transfer time and the file size, with the trendline regression statistic at a 0.977. The data above was calculated as the average transfer time for five attempts of transferring .dat files for each file size at a buffer size of 256 bytes.