PA2 Report

Jiarui Li

**DESIGN**

The main purpose of this PA2 is to learn how to manipulate the client end and be able to talk to the server. Client and server are separate process and has their own main functions. The server listens to client and client will request different messages, including data point message, file message, new channel message and quit message from the server, while the server must be able to respond to these requests and write the requested data to specified locations.

The main functionality that grants the client the ability to request messages from server are the cread() and cwrite() functions. By creating different types of messages and writing them to the server, the server will return the requested data to the pipe and be ready for user to use cread() and retrieve the data.

In this assignment, I implemented the following (briefly):

Requesting a single data point by writing a data message to the server.

Requesting all the data point (15k) for a single person and write them into a file.

Requesting an entire file by having access to small-sized windows that represents part of the file. And get the entire file by piling all the received windows together.

Requesting a new channel by writing a new channel request message to the server and the server will agree on a name and open a new communication channel.

Running the server as a child process in client executable file by using fork() and exec().

**TIMING**

There are two ways of requesting data from the server: requesting data points and requesting an entire file. Their runtime differs largely.

As for requesting data points, its runtime is linear to the number of operations it does.

Below is a chart of runtime vs num of operations for requesting data points (aka number of requests)

We can easily conclude that the more data point request is done, the more time is spent, and they are of linear relationships.

As for requesting an entire file, the runtime is affected by two factors: the file size and the window (buffer) size.

When file size is fixed, the larger the buffer size is, the shorter the runtime will be. Because with large buffers, the program is required to loop less and request less since the request time is calculated as ceil(filesize / buffercapacity).

But the buffer size has an upper limit which differs from machine to machine.

And when the buffer size is fixed, the larger the file is, the longer the runtime will be. Because larger files request more operations to get all the needed data from a relatively small window size.

Based on the data I observed throughout the testing process, for a same file, generally the runtime by requesting an entire file is significantly less than the runtime by requesting single data point many times. Even though both methods yield same consequences (i.e. the entire file is saved locally), requesting an entire file takes less operations than the other method because it is implemented in the way that it can get a chuck of data each time it send requests to the server, rather than only getting an entry of data each time.