Programming Assignment 2 Report

by Siyuan Yang

Design

In this assignment, we are asked to write a client program that connects to a given server which defines a communication protocol implemented by sending properly formulated messages over a communication pipe. The server hosts several electrocardiogram data points of 15 patients suffering from various cardiac diseases. Our goal is to obtain these data points by sending properly formatted messages that the server understands. Besides, the client also needs to implement the file transfer functionality such that it can collect files of arbitrary size using a series of requests/response. Finally, we need to build a client that can ask the server to create a new channel of communication. The client sends a special message, and in response, the server creates a new request channel object and returns the name back with the same channel.

Evidence

Task 1. Requesting Data Points

Description: All data points for person 1 by both ecg1 and ecg2 are saved in a file called x1.csv. By using the “diff” command, we compare both files and find out they are exactly the same. The time taken for requesting data is shown on the screen which is 78 seconds.

Task 2. Requesting Files

```
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -f 1.csv
Time Taken For Requesting Files: 14549 microseconds
Client-side is done and exited
Server terminated
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ diff received/y1.csv BIMDC/1.csv
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$
```

Description: The received file for 1.csv is saved in a file named y1.csv under the received directory and the time taken for requesting data is shown on the screen which is 14549 microseconds.

```
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -f 100.dat
Time Taken For Requesting Files: 84 microseconds
Client-side is done and exited
Server terminated
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ diff received/100.dat BIMDC/100.dat
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -f 256.dat
Time Taken For Requesting Files: 85 microseconds
Client-side is done and exited
Server terminated
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ diff received/256.dat BIMDC/256.dat
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -f 1000.dat
Time Taken For Requesting Files: 135 microseconds
Client-side is done and exited
Server terminated
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ diff received/1000.dat BIMDC/1000.dat
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$
```

Description: To treat the file as binary, we use the truncate command and transfer the file to 100.dat, 256.dat and 1000.dat and test these binary files respectively. By using diff command, the aforementioned files in two different directories are identical to each other.

```
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -f 1.csv -m 102400
Time Taken For Requesting Files: 13233 microseconds
Client-side is done and exited
Server terminated
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -f 1.csv -m 5000
Time Taken For Requesting Files: 14536 microseconds
Client-side is done and exited
Server terminated
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -f 1.csv -m 104857600
Time Taken For Requesting Files: 12342 microseconds
Client-side is done and exited
Server terminated
```

Description: By experimenting with transferring larger files, we find out that the most likely bottleneck is buffer capacity. The relationship between them is that the higher the memory capacity buffered, the faster the transfer time is.

Task 3. Requesting a New Channel

```
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -c
Test Result @new_channel: -0.475
Client-side is done and exited
Server terminated
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ./client -p 1 -t 59.00 -e 2
Requested Data Point: -0.475
Client-side is done and exited
Server terminated
```

Description: The test case for sending a message to the new channel is patient 1 at 59.00 seconds and ecg2. The result turns out to be the same, proving that the new channel can be used to speak to the server.

Task 4. Run the Server as a child process

```
// Run the server as a child process
if (fork() == 0) {
    char* argv[] = {"./server", NULL};
    execv(argv[0], argv);
} else {
```

Description: We run the server process as a child of the client process using fork() and exec(). From the diagrams above, we only open a single terminal, run the client which first runs the server and then connects to it.

```
// closing the channel
MESSAGE_TYPE m = QUIT_MSG;
chan.cwrite(&m, sizeof (MESSAGE_TYPE));
wait(NULL);
```

Description: The server does not keep running after the client dies, sending a special QUIT MSG to the server and call wait() function to wait for its finish.

Task 5. Closing Channels

```
steven@steven-desktop:~/Desktop/CODE/pa2/starter_code$ ls
BIMDC   client.cpp  common.h  FIFOreqchannel.cpp  FIFOreqchannel.o  received  server.cpp
client  common.cpp  common.o  FIFOreqchannel.h    makefile          server
```

Description: When closing the channels, there is no fifo_data* files.

Analysis

In conclusions, my client program works as expected. The client program can obtain data points by sending properly formatted messages that the server understands. It can also implement the file transfer functionality on text files and binary files with different file sizes and varying buffer capacity. Finally, the client can ask the server to create a new channel of communication. All above can be done using just one terminal and exiting successfully after everything is done.

By running several tests on requesting data points and requesting files, it's safe to say that requesting data points is less efficient than requesting files directly. We use the gettimeofday function to record the time taken for requesting data points and files. The time taken for requesting all data points in 1.csv is about 78 seconds; however, the time taken for requesting the 1.csv file is only 14549 microseconds which is about 0.01 second. The reason for that is we only get one line of data for every request we sent to request all the data points; comparatively, the server will send at most 256 bytes of data buffer back to our client. In requesting files part, the experiment with transferring larger files shows that the higher the buffer capacity, the faster the transfer time is.