PA2 Report – CLIENT PROCESS SPEAKING TO A SERVER

Muhammad Taha Haqqani
Section 507

Introduction

In this assignment, we implemented a client program that connects to a given server. The server
defines a communication protocol, which the client has to implement by sending properly formulated messages over a communication pipe. The server hosts several electrocardiograms
(ecg) data points of 15 patients suffering from various cardiac diseases. The client's goal is to obtain these data points by sending properly formatted messages that the server understands. In addition, the server is capable of sending raw files potentially in several segments (i.e., when the file is larger than some internal buffer size).The client must implement this file transfer functionality as well such that it can collect files of arbitrary size using a series of requests.

Tasks:

- Request Data Points
    - All ecg1 and ecg2 data points for person 1
    - Must be stored in a file "x1.csv"
    - Compare file with the original BIMDC/1.csv
    - Use diff compare tool to demonstrate that it is the same file
    - Measure the time using gettimeofday function
- Request Files
    - Request files and put them in the received directory with the same name as original file
    - Compare the files using diff Linux command
    - Measure the time for the transfer
    - File transfer must work with binary files
    - Create binary files using truncate command
    - Experiment with larger files and document required time
    - What is the main bottleneck?
    - Can transfer time be changed by varying the bottleneck?
- Request a New Channel
    - Create a new channel by sending NEWCHANNEL_MSG to server
    - Demonstrate that the channel can be used to communicate with the server
    - Send a few data point requests
- Run the Server as Child Process
    - Use fork() and exec() to run server as child process of client
    - Have one terminal instead of two
    - Make sure the server closes after the client dies by sending QUIT_MSG to server
    - Call for wait() function to wait for the server process to finish
- Closing the Channel
    - There must be no open channels
    - Close all channels using QUIT_MSG

Design

In this programming assignment, we are sending specific requests to the server through the channel. These include data message, file message, new channel message, and quit message. The cwrite function is used to send this information with the number of bytes that the message contains. The cread function is used to receive the information from the server which is limited by buffer capacity set by the server.

Data messages must include the person (1-15), time (0.000 – 59.996) and ecg value (1 or 2). If we require all the information of a patient, we must go through the entire file and make 15000 requests for those 15000 data points since each message can only ask for one data point at a time.

To request for a file, the client needs to package a message containing the file type, name of the file with the null terminator character. Before any transfer, the client requests the server for the number of bytes present in the requested file which is done by sending a file message with 0 offset and length. This is used to determine the number of times the server needs to be requested since the buffer capacity limits the number of bytes that can be transferred. Therefore, the server sends the file to the client in slices which is determined by the formula ceil (total number of bytes / maximum size of message). To change the buffer capacity, the -m command line argument -m was used.

To request for a new channel, the client needs to send a NEWCHANNEL message to the server to which the server responds by sending the name of the file. The client needs to join the channel with the same name in order to be able to communicate with the server through this channel.

Running the server as a child process of the client allows a single terminal use. This is done by making the server a child process of client by using the fork and execvp function. This step allows the server to run within the client and the server process is allowed to complete using the wait function until both the server and client die off.

The QUIT message is necessary to close any channel that is currently open in the system. This is important to make sure the server cleans up everything and then exits the system.

Experimental Data

To test the performance of the server and the factors affecting it, all the tasks were timed using the gettimeofday function. One of the factors that were expected to have a role was the size of the buffer capacity of the server which can be changed using the -m command line argument. This would only affect the file transfer process. For data messages, we are always reading a double and this would have no effect.

For data messages, random points were requested and the time taken was recorded. After averaging the results, it was seen that the average time taken to request a single data point was close to 5 ms (0.005 s). The next experiment was to check the performance of requesting all the data points for a single patient i.e 15K data points and putting them in a new file in the

received directory. Time was recorded for each patient and an average was taken. The time taken to transfer each of the patient files by data message requests is given below.

Transfer time for requesting by data messages:

Patient 1: 127.04 sec
Patient 2: 106.74 sec
Patient 3: 135.63 sec
Patient 4: 132.21 sec
Patient 5: 140.88 sec
Patient 6: 122.52 sec
Patient 7: 138.74 sec
Patient 8: 111.72 sec
Patient 9: 99.94 sec
Patient 10: 104.00 sec
Patient 11: 125.12 sec
Patient 12: 116.18 sec
Patient 13: 107.75 sec
Patient 14: 116.15 sec
Patient 15: 139.64 sec

Average = 136 sec

The next experiment was to transfer the patient files by making file message requests. Each file was between 280KB and 290KB. The time taken to transfer each file with a default buffer capacity of 256 bytes is given below.

Patient 1: 0.076 sec
Patient 2: 0.049 sec
Patient 3: 0.034 sec
Patient 4: 0.049 sec
Patient 5: 0.045 sec
Patient 6: 0.078 sec
Patient 7: 0.088 sec
Patient 8: 0.036 sec
Patient 9: 0.084 sec
Patient 10: 0.071 sec
Patient 11: 0.072 sec
Patient 12: 0.035 sec
Patient 13: 0.079 sec
Patient 14: 0.082 sec
Patient 15: 0.066 sec

Average = 0.062 sec

Two factors were expected to affect the performance of this transfer: size of the file and buffer capacity. To test the relationship, files with sizes 100B, 256B, 1KB, 500KB, 1MB, and 10MB

were created and the transfer time was recorded for a constant buffer capacity of 256B. Binary files were created using the truncate command.

More data was recorded and the graph below was obtained.

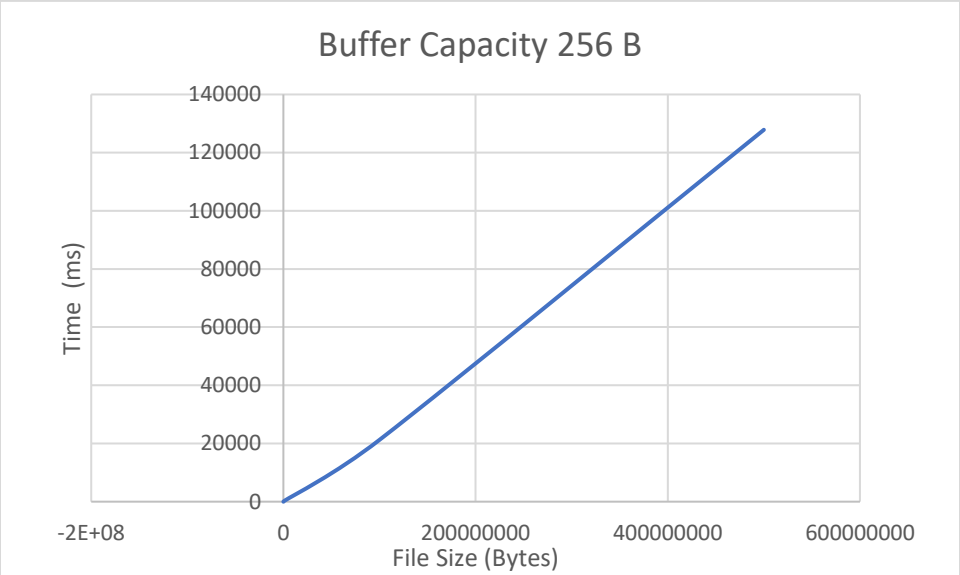

Fig: Graph for transferring file sizes (100 B – 500MB) for buffer capacity 256B

The next experiment was done to find the relationship between buffer capacity and file size. Buffer capacities were increased in the order 100 B, 250 B, 500B, 1KB, 250KB, 500 KB, and 1 MB to transfer a file of size 100 MB. The following graph was obtained after taking more readings.

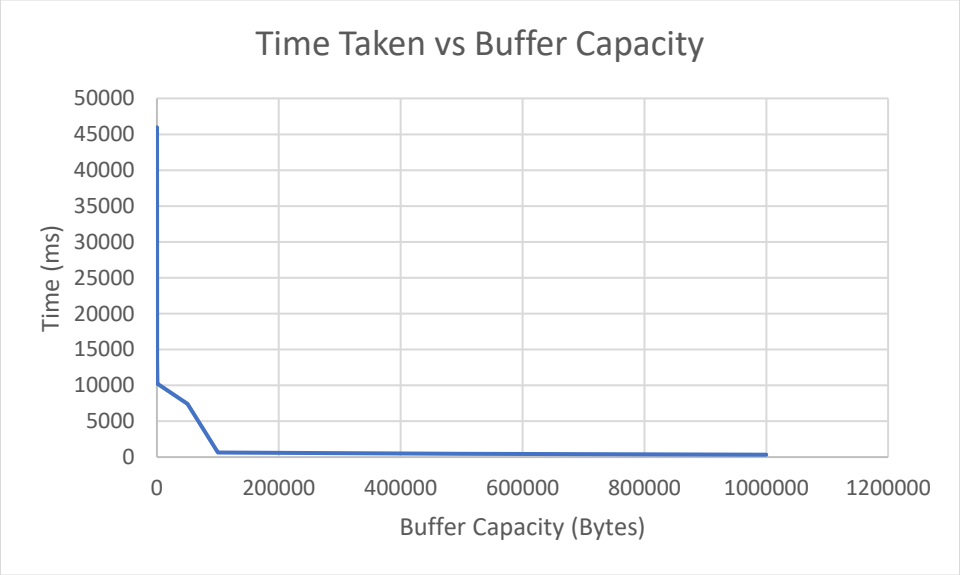

Fig: Graph for Examining the affect of increasing buffer capacity to transfer a 100MB file

Conclusions

After conducting the experiments, the following conclusions can be made:

- It takes longer to transfer 15K data points than to transfer the whole file
- If the buffer capacity is kept constant, there is a direct relationship between file size and time which means as file size increases, transfer time increases
- If the file size is kept constant, there is an indirect relationship between buffer size and time, which means that as buffer capacity increases, the transfer time decreases
- For very large files (> 100 MB), it is more efficient to use larger buffer capacity
- The bottleneck for this experiment is the existence of just one buffer
- An improvement to this would be using several buffers
- Another improvement is to have a system where the buffer size will alter automatically depending on the file size to keep the transfer time constant