MP4 Analysis Report

Julian Tiu | CSCE 313-504 | 19 Nov. 2017

1. README – MUST READ

All compilation is done by the makefile. To compile, first type make in the command line and press enter. This will produce two executable files, namely client and dataserver. Type make clean and press enter to remove all fifo files. Finally, type. /client –w # -n # -b # in the command line and press enter to run, where –w is the flag to determine the amount of request channels, –n is the flag that determines the number of times each request thread will send to the buffer, and –b is the size of the bounded buffer. It is recommended to run with –b 12 –n 3 –w 3 for the firs case. Although dataserver is forked from running client, it is recommended to comment out the fork command and actually run dataserver and client on separate command windows to have a cleaner command window to analyze the print statements of each run of client.

2. Report

The performance was unable to be tested fully, because the results varied from machine to machine. While using the user's local machine, the process sometimes gets aborted due to "Illegal Instruction: 4." When being debugged, it is shown that the produced binaries for requestChannel.C and dataserver.C are unable to be read by the user's current operating system. The program was also ran in multiple school servers, as to which the results varied with a segmentation fault, an infinite loop, or another "illegal instruction" error. In some cases it would run fine, but the histograms are not printed out – assuming that the histograms are not being populated correctly. Even when deleting all the fifo files after each run, all these cases can happen without even changing a single line of code. Due to this complication, it was hard to pinpoint the exact cause of the errors. When running my implementation, it is encouraged to run the program multiple times until the histogram finally prints.

Since the program ran unsuccessfully, the description of the algorithm is as follows. In main, one thread was created to run the *eventHandlerFunction(void * args)* function. When the thread enters the function, the first major protocol it does is create an array of request channels called *RCA[]*, and this array gets populated with new request channels from the *dataserver*. Then, it creates an *fd_set* called *readSet* and *FD_ZERO* gets called on *readSet.* The *readSet* is then populated with the corresponding file descriptors from the request channels in *RCA*. Once *readSet* is populated, each request channels in *RCA* uses cwrite. The *select()* call then gets the file descriptors that was written to and *cread()* each file descriptors' contents. Each reply is then sent to the proper statistic bounded buffer. After that, readSet is repopulated and more requests are then pulled from the main bounded buffer that the request threads forwarded to. If three "DONE" commands have been pulled, then the event handler thread calls "quit" and terminates. If the pulled data follows the format "data 'name'," then a request channel from *RCA* calls *cwrite()*. After this, it goes back to the top of the loop where it calls *select()* to get the reply from the dataserver.

If the program ran correctly, the assumption that would be made is that using multiple worker threads is a faster option; although using threads for very large systems would put too much pressure on the scheduler. Using the select call in the other hand, would still utilize going through each written file descriptors incrementally, but would be able to handle large systems. It is also possible to have multiple threads properly synchronized calling the select call concurrently. Increasing the amount of request channels would increase the number of file descriptors that the for loop would incrementally check with *IF_SET()*, thus increasing the time to process all the updated file descriptors.