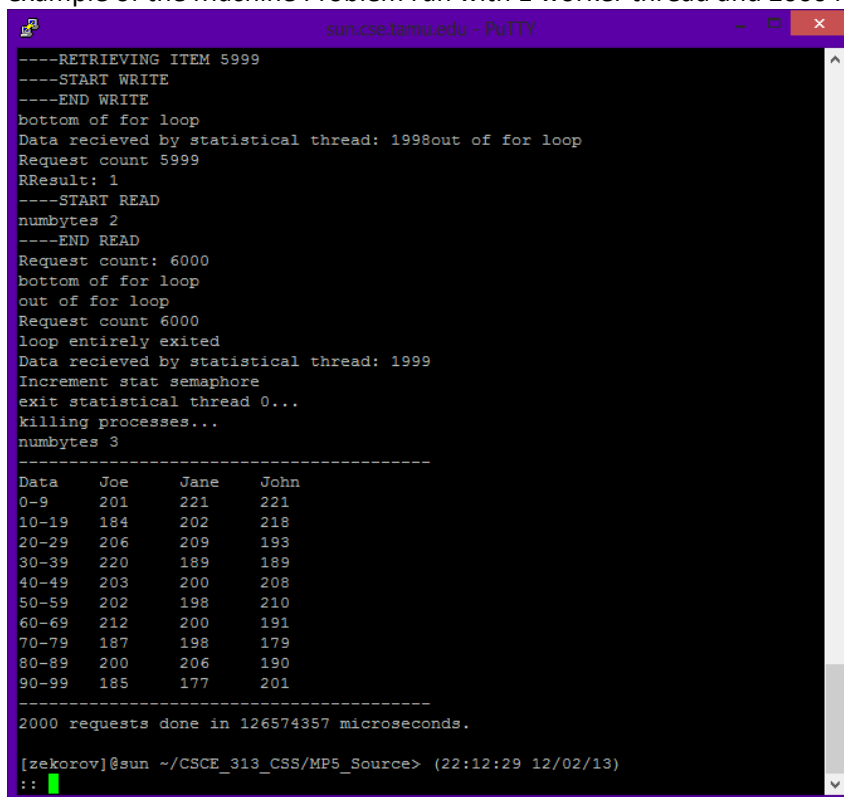


Colt Campbell
CSCE 313_504
December 2, 2013

MP5 Report

Overall Performance Evaluation:

The general performance for this machine problem seems to be rather poor, but also because I have several more usleep commands throughout the program. It is also communicating through sockets over a network rather than through local files on one machine, which perhaps is partly what makes it slower, but more useful than the previous machine problems. However, with mine in particular, there is an issue. I was trying to find a way to fix whatever was messing up with it when the client is ran with more than 5 workers, and more than about 1000 requests. I tested the program with 2000 requests and 1 worker, and it worked fine for a moment. At times, with certain combinations of inputs, the program seems to hang just before my event_handler_function gets finished processing all of the requests, and thus it never exits the main while loop in that function, and never closes all of the sockets of the existing in the client, which causes both the client, and subsequently the server to hang. At least, that is what I think is going on with the program at the moment. For some inputs, like anything with 1 worker, and a backlog of at least 5 pretty much always works for the client/server system. Because of that though, I do not know why it hangs, or where exactly it is hanging other than just before the event_handler actually gets through processing and handing everything over to the statistics thread in the client. Here is an example of the Machine Problem run with 1 worker thread and 2000 requests:



```
sun.cse.tamu.edu - PuTTY
----RETRIEVING ITEM 5999
----START WRITE
----END WRITE
bottom of for loop
Data recieved by statistical thread: 1998out of for loop
Request count 5999
RResult: 1
----START READ
numbytes 2
----END READ
Request count: 6000
bottom of for loop
out of for loop
Request count 6000
loop entirely exited
Data recieved by statistical thread: 1999
Increment stat semaphore
exit statistical thread 0...
killing processes...
numbytes 3
-----
Data      Joe      Jane      John
0-9      201      221      221
10-19    184      202      218
20-29    206      209      193
30-39    220      189      189
40-49    203      200      208
50-59    202      198      210
60-69    212      200      191
70-79    187      198      179
80-89    200      206      190
90-99    185      177      201
-----
2000 requests done in 126574357 microseconds.

[zezorov]@sun ~/CSCE_313_CSS/MP5_Source> (22:12:29 12/02/13)
::
```

You can tell it worked just fine here, but with something crazy like 20 or so worker threads, the whole thing hangs up. I tried adding usleeps all over the place to see if something just needed more time to process, but I do not think that is the case. I have checked several places in my code for possible errors, such as my dataserver functions, my networkRequestChannel functions and constructor/destructor, and I believe everything there is as it should be. But, I cannot know for sure. Whatever is wrong is certainly

not an obvious thing that can just jump out at me; otherwise, I would have found it. On the plus side, when it does work with multiple worker threads, its performance does improve greatly with each new worker, up to about 20 worker threads. After that, I have not been brave enough to test because it seems to crash more often with more workers. As far as backlog goes, I have tried out backlogs of 5, 10, and 20. With 20 backlog, I used 20 workers, and when it worked it was reasonably fast. It was not faster than my previous machine problems, but as stated, that is due to primarily the number of `usleep` commands now in the program from attempting to cure it of getting hung up near the end of processing requests and probably the fact that it is network programming rather than file programming. It makes sense to me that we would sacrifice performance with network programming so that we could have higher utility and communication possibility than with file programming.

Evaluation of Client/New Dataserver:

To my knowledge these both should work fine, as I tested them with the `simpleclient` program prior to implementing them into the client program I used for the last machine problem (the one that includes all of the thread functions and select to keep track of the file descriptors). In fact, two text files are included with the rest of the files labeled `client_test.txt` and `server_test.txt` respectively. The client test file was a test run of the dataserver along with the client program from the previous machine problem. It shows that the program hung as at the end of the file, no quit requests are ever sent. It displays the output I had going at first for the client when I first saw that it was crashing with certain combinations of inputs for the client/server. The second file, the `server_test.txt` one, was ran with the `simpleclient` given to us to start with for MP5. I just used it to test the server code that I had written, and with the output shown in the file, it does seem to work in its entirety. I ran the `simpleclient` two consecutive times in a row without closing the server and it worked both times (reflected through the text file). Thus, I believe the dataserver and the closing of the sockets works fine, so I do not know why it gets hung up sometimes. Some sort of race condition between the statistics thread and the event handler must be happening though, or something of that nature that would be hard to detect. Aside from the problems the program has, when it works, it does seem to do rather well. The server can support multiple clients running at the same time, or consecutively.

For some clarification of the runtimes with the varying times of backlog in dataserver, here is a sample (all outputs reflect only one client worker, and a buffer of 100, and 100 requests)

output when backlog = 10;

100 requests done in 11004260 microseconds.

Output when backlog = 5;

100 requests done in 11003563 microseconds.

Output when backlog = 20;

100 requests done in 11004473 microseconds.

And as you can tell, the backlog alone does not really change the performance alone. If I could get more than 10 workers to work well without being hung, I would have used as many workers as the backlog, and we probably would have seen a drastic increase in performance each time. It is very helpful to have a backlog larger than your worker amount. It helps keep everything smooth and working efficiently (nobody is waiting to be put in line, and everyone can actually be in the line if they have to be all at once even).