Machine Problem 4 Report

Lab members: Martin Fracker, Chris Findeisen

Date: April 18, 2015

Introduction

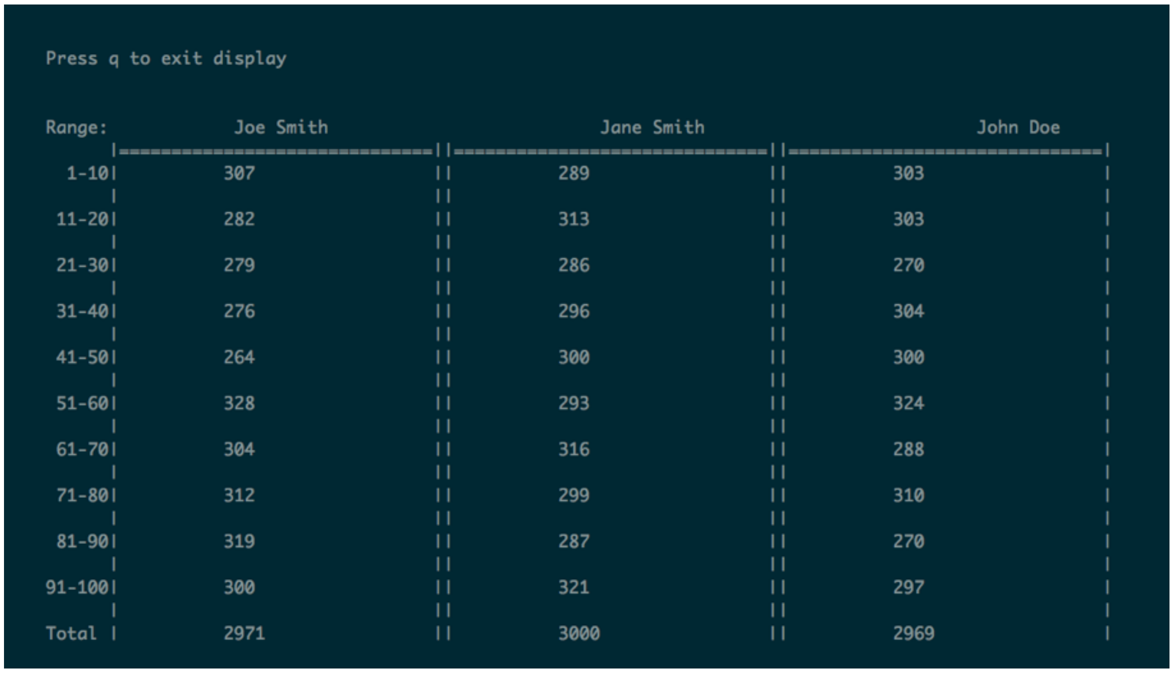
During machine problem 4, we worked with threads to extend our dataserver/client relationship from machine problem 2. We first forked off a process to load the data server. We then called a client that used three request threads and worker threads. These threads used a queue mutex and two semaphores to access a bounded buffer. Finally, we joined the threads when their work was done.

Procedures

We implemented the three different types of threads, and implemented bounded buffer. We created these in our `main.cc` and joined them at the end before sending "quit" to the data server.

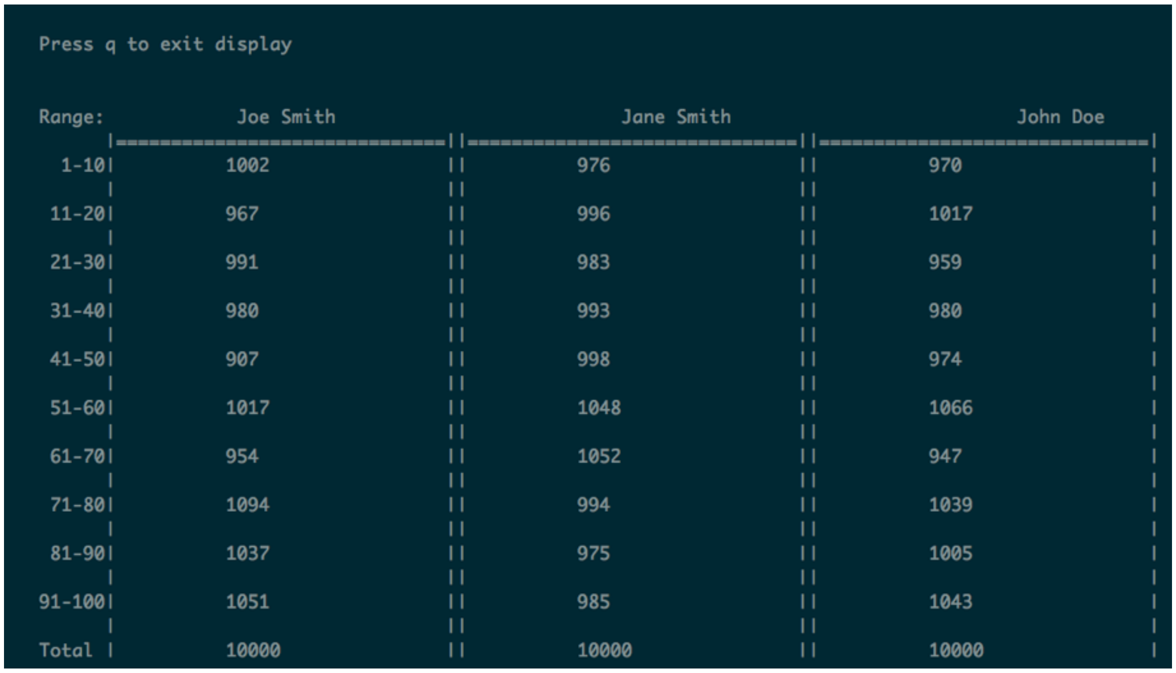
Results:

We used ascii-engine, built by Martin, to display our results, which allowed us to update in place and in real time. Below are two snapshots of it. The first is while the 10,000 requests was running and afterwards, when it was finished.



```
Press q to exit display

Range:          Joe Smith                         Jane Smith                         John Doe
       |==============================||==============================||==============================|
  1-10|         307                  ||         289                  ||         303                  |
       |                              ||                              ||                              |
 11-20|         282                  ||         313                  ||         303                  |
       |                              ||                              ||                              |
 21-30|         279                  ||         286                  ||         270                  |
       |                              ||                              ||                              |
 31-40|         276                  ||         296                  ||         304                  |
       |                              ||                              ||                              |
 41-50|         264                  ||         300                  ||         300                  |
       |                              ||                              ||                              |
 51-60|         328                  ||         293                  ||         324                  |
       |                              ||                              ||                              |
 61-70|         304                  ||         316                  ||         288                  |
       |                              ||                              ||                              |
 71-80|         312                  ||         299                  ||         310                  |
       |                              ||                              ||                              |
 81-90|         319                  ||         287                  ||         270                  |
       |                              ||                              ||                              |
91-100|         300                  ||         321                  ||         297                  |
       |                              ||                              ||                              |
Total |         2971                 ||         3000                 ||         2969                 |
```



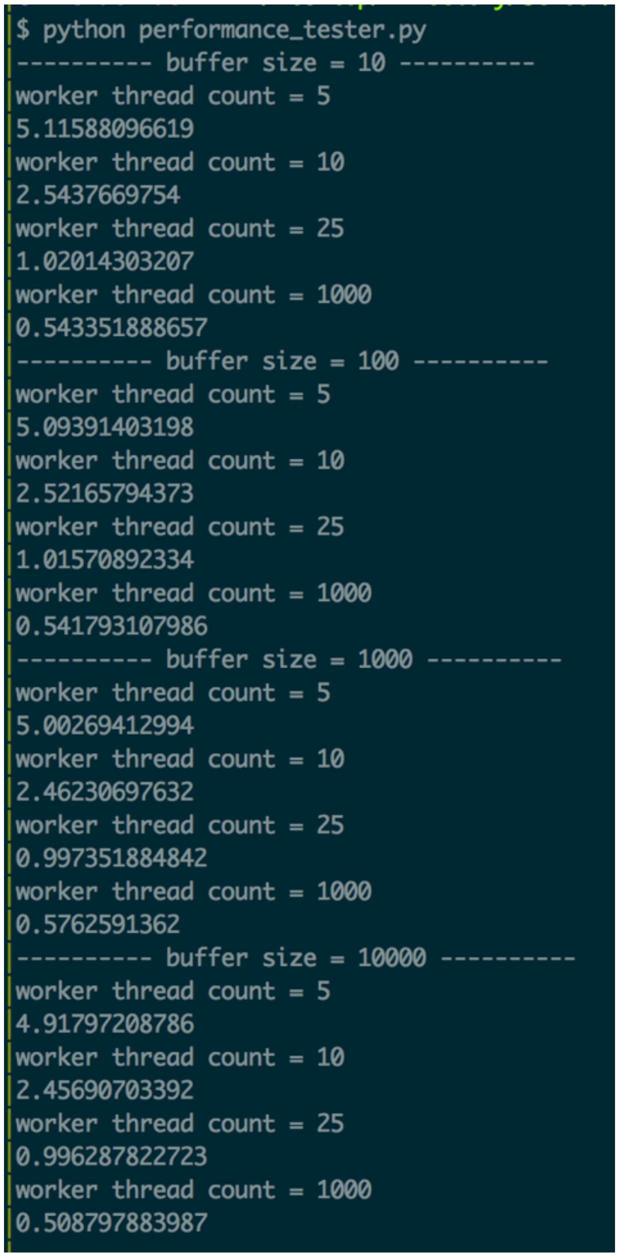
```
Press q to exit display

Range:          Joe Smith                         Jane Smith                         John Doe
       |==============================||==============================||==============================|
  1-10|         1002                 ||         976                  ||         970                  |
       |                              ||                              ||                              |
 11-20|         967                  ||         996                  ||         1017                 |
       |                              ||                              ||                              |
 21-30|         991                  ||         983                  ||         959                  |
       |                              ||                              ||                              |
 31-40|         980                  ||         993                  ||         980                  |
       |                              ||                              ||                              |
 41-50|         907                  ||         998                  ||         974                  |
       |                              ||                              ||                              |
 51-60|         1017                 ||         1048                 ||         1066                 |
       |                              ||                              ||                              |
 61-70|         954                  ||         1052                 ||         947                  |
       |                              ||                              ||                              |
 71-80|         1094                 ||         994                  ||         1039                 |
       |                              ||                              ||                              |
 81-90|         1037                 ||         975                  ||         1005                 |
       |                              ||                              ||                              |
91-100|         1051                 ||         985                  ||         1043                 |
       |                              ||                              ||                              |
Total |         10000                ||         10000                ||         10000                |
```

Performance Analysis:

The client consists of the worker threads, client threads, and the histogram threads, used for statistics.

Worker threads dequeue and client threads enqueue. Context switching when there are more threads than cores, can cause performance decreases. However, because only one thread can access the queue at one time(because of the mutex), only one thread can run at a single time during that crucial section. We identify this as the biggest performance loss, since worker threads and client threads mostly run on these instructions.



```
$ python performance_tester.py
---------- buffer size = 10 ----------
worker thread count = 5
5.11588096619
worker thread count = 10
2.5437669754
worker thread count = 25
1.02014303207
worker thread count = 1000
0.543351888657
---------- buffer size = 100 ----------
worker thread count = 5
5.09391403198
worker thread count = 10
2.52165794373
worker thread count = 25
1.01570892334
worker thread count = 1000
0.541793107986
---------- buffer size = 1000 ----------
worker thread count = 5
5.00269412994
worker thread count = 10
2.46230697632
worker thread count = 25
0.997351884842
worker thread count = 1000
0.5762591362
---------- buffer size = 10000 ----------
worker thread count = 5
4.91797208786
worker thread count = 10
2.45690703392
worker thread count = 25
0.996287822723
worker thread count = 1000
0.508797883987
```

<- This figure shows the performance of our program with differing buffer and worker sizes. The times are shown in seconds. As you can see, the worker thread count greatly increased efficiency. We found that when the worker thread count went over 10000, the efficiency returns began to be almost nil. The increased buffer size also helped, although less dramatically than increasing the worker thread count.

The reason the increase in worker threads yields such a large increase in performance is because ready threads can be scheduled while other threads are waiting for file I/O.

Conclusion

We learned a lot about threading, especially debugging concurrency issues. One take-away for us was to carefully pair-program over critical sections in our code, in order to catch silly mistakes. Likewise, when there is a problem, the method to handle it should involve 90% reading, and 10% coding.

If we were to do this again, we would have split the threads up less granularly, giving them a longer sequence of instructions. This would allow our code to benefit more greatly from our threading.