CSCE 313.506-F18:
Programming Assignment 3

Tristan Seifert

Due: Sunday, Oct. 21, 2018

---

1 Overview

The major change in this code was to use threads to first populate the `SafeBuffer` with requests, then spawn a user-defined number of threads to pop requests off of the buffer and send them to the server through a `RequestChannel`. The main thread creates a channel for each thread, then spawns the thread with a reference to that channel, so it can easily detect if the system has run out of file descriptors and continue with the already spawned threads instead of failing spectacularly.

Additionally, both the `SafeBuffer` and `RequestChannel` classes were made threadsafe by adding a `std::mutex` that protects access to the inner data structures, using a `std::lock_guard` to ensure the lock is unlocked when the function returns, or the guard goes out of scope.

Code for the `dataserver` was not modified, since it already spawns a new thread for every new request channel created, and processes data that way.

2 Performance Data

This data was gathered for varying values of w while $n = 10000$. The system used runs macOS 10.14.1 beta 2, with 64GB of RAM and dual Xeon E5-2670 v2 processors at 2.6 GHz.

A maximum of approximately 2000 threads are possible until the file descriptor limit of 4096 was reached. If the program reaches this, calls to create a new channel return with `EMFILE`; the program prints the message `Too many open files` and attempts to continue with the threads that have already been spawned.

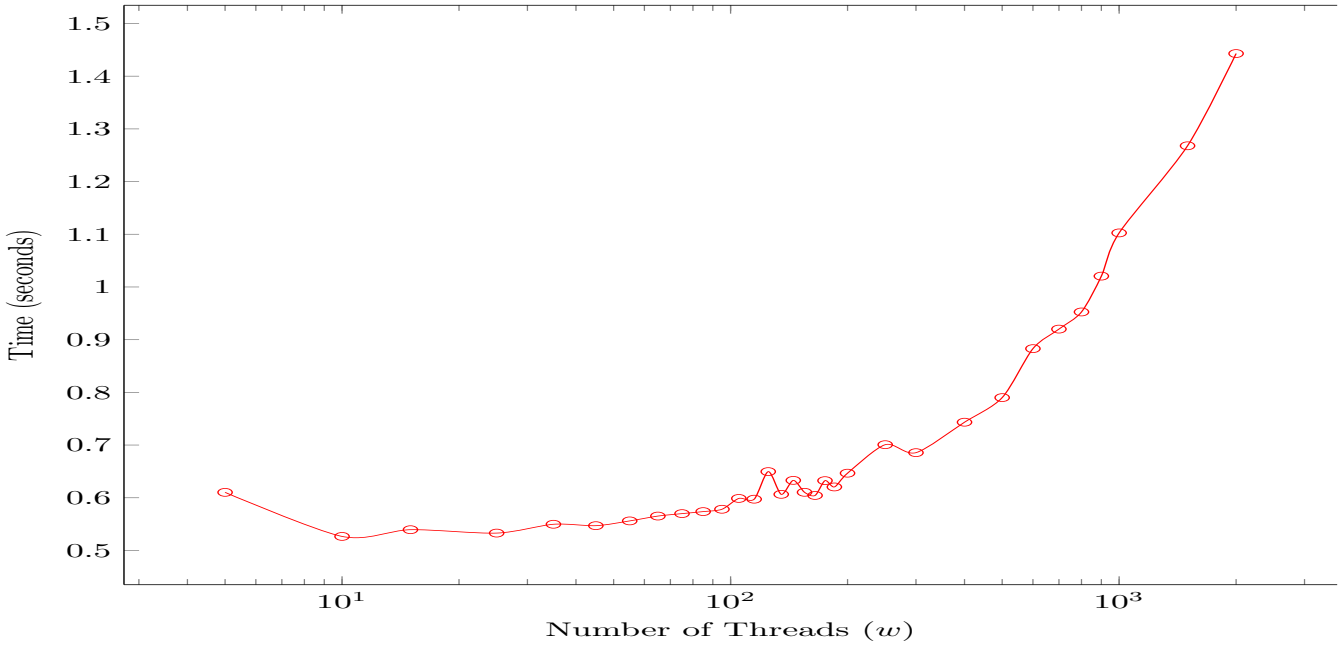

Figure 1: Runtimes for $n = 10000$

The ideal number of worker threads is around 20-25, which roughly corresponds to one thread for every CPU core on my machine. This way, threads aren't competing for CPU time, and instead can all be scheduled simultaneously. If the threads were more IO heavy, it would make sense to increase this number so other threads could get started on their work, while others are waiting for IO to complete.

Threads	**Time (sec)**
5	0.61021
10	0.52660
15	0.53936
25	0.53292
35	0.54966
45	0.54700
55	0.55604
65	0.56522
75	0.56987
85	0.57369
95	0.57813
105	0.59868
115	0.59724
125	0.64959
135	0.60645
145	0.63296
155	0.61032
165	0.60435
175	0.63245
185	0.62049
200	0.64660
250	0.70069
300	0.68557
400	0.74335
500	0.79003
600	0.88296
700	0.92002
800	0.95256
900	1.02058
1000	1.10259
1500	1.26794
2000	1.44296
2500	ERR

Compared to the original program, even just having three worker threads provides a significant improvement. For $n = 10000$, the original program took approximately 15 seconds. The sizeable improvement in performance stems from using threads to populate the buffer, as well as using multiple threads to process requests.