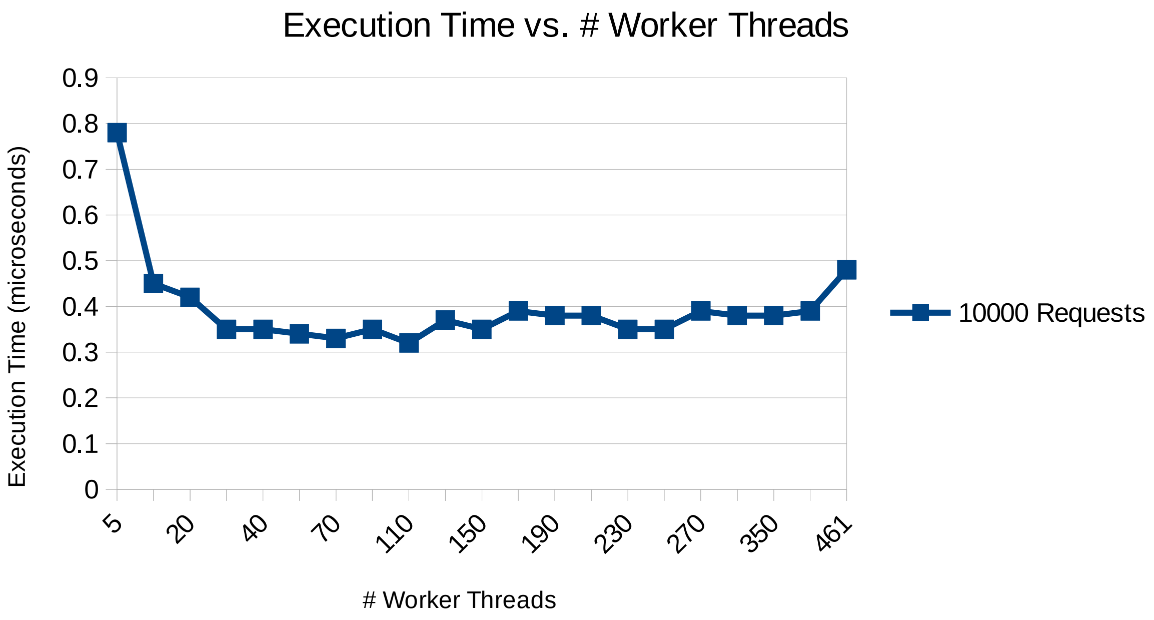
The code that I have written is different from the code that was given to me because it now runs all of the requests in parallel with threading. It does this by creating three threads for filling the request buffer with all of the requests that must be populated before processing the requests can be done. Each person (3 people) must have a request thread that can fill the request buffer. Additionally, the code is different because it creates *w* number of worker threads specified by the user to process all of the request through the server. The *w* worker threads take requests from the safe_buffer as quickly as they can process them and, because they are all able to work separately while sharing resources, the execution time of the program is greatly improved.



As shown above, the execution time lowers greatly as the number of threads increases. The kernel does not seem to be overwhelmed by the amount of threads that are open at any point, but you can see where it starts to taper off that the overhead from the threading can start to cause the execution time to increase slightly When the execution time of the program after threading is implemented is compared to the execution time of the program prior to the threading implementation, the execution time of the program with just 5 threads is around 0.8 microseconds compared to 1.4 microseconds with no threading. If we increase the number of threads up to over 30 worker threads, then the execution time of the program moves under 0.4 microseconds.   Thus, threading *greatly* decreases execution time.

I gathered all of my data and ran my programs on my Dell Precision 5510 containing a 7th generation Intel core i7 processor, 16 GB of RAM, and SSD storage. The system is running Ubuntu 18.04 Bionic Beaver. The maximum number of throws that the system allows on my system is 460 worker threads. When you reach 461 worker threads, the error that is thrown states:

```
Too many open files
fifo_data462_2: Too many open files
```

The operating system doesn't do anything other than automatically close all of those open threads and stop the program in its tracks, leaving all of the fifo files open. The client program simply stops and does not continue to receive requests from the "server."