Howard Cheng
Jason Narvaez
Sam Lu

Machine Problem 4

Introduction

This machine problem expands on the client-server protocol and requests from the previous machine problem. However, the responses are to be in integer form instead of strings. Through this machine problem, we understand thread behavior and these threads' order of processes. We also understand how semaphores can be used as "locks" to control thread execution by implementing "mutual exclusion" on a code's critical section.

Procedure

Similarly to the previous machine problem, we fork off a process, load the data server, and send requests and expect responses. Each person making requests have their own "request thread," which help load the requests into a "request" bounded buffer instead of sending the requests directly to the data server. The bounded buffer consists of three parameters to hold number of request threads, the size of the buffer, and a worker thread. Worker threads are created to continuously pull requests from the request buffer. Each worker thread communicates with the data server through a request channel. There is also a thread named "control" to signify a connection between the client and server. The worker thread function enters an infinite loop because of the unsure number of worker threads processing. When the data server receives the requests and sends them back to the worker threads, the worker threads send these responses to response buffers. Each person has their own response buffer which is evaluated with a statistics thread. The statistics thread counts the response frequency of each integer from the server and outputs the respective person's frequencies when the requests terminate. To terminate the client requests, we load "stop" requests into the request buffer. If a worker thread receives this request, it breaks the while loop and consequently stops pulling from the request buffer. To regulate thread execution, we used semaphores to give mutual exclusion by having threads wait on each other finish executing a function based on the semaphore's value. The semaphore operates with two atomic operations which either decrement or increment the semaphore value. Consequently, the semaphore may sleep.

Results

After testing with different numbers of worker threads, we observed that increasing worker threads improves performance. However, after reaching a certain number of worker threads, the program enters a deadlock. Conversely, decreasing worker threads hinders performance. Compared to having 40 worker threads, having 30 worker threads hindered performance by nearly one second for 10,000 requests per person. We also observed that increasing the buffer size does not affect the performance.

Conclusion

From this lab, we learned how semaphores can be used to regulate thread execution and exclusively have one thread use a function. Through the bounded buffer, we also learned how threads should wait on others to finish before executing.