|  |  |  |
| --- | --- | --- |
| 10/14/21 |  | Memo |
|  |  |  |
| To  Dr. Otte  From  Adam C. Paquette  Jack A. Garrard  Re  CS 560 Computer Networks HW5 Observations and Explanations |  |  |

# Comments:

The single threaded server has some obvious issues when handling multiple people asking for a 3a + 1 computation. The server allows for a 100 item queue of connections which fills up quite fast. Hence the single threaded server struggles a lot to get through request from our high throughput client. While it chunks away at the connections, it takes a very long time to get through all 1000 from our stress client. That being said, it is still able to handle all 30 connections coming in from our stress test client. To preface, we have set the NUM\_CONNECTIONS variable to 100. This allows the single threaded server to take all of the connections that are being pushed at it by the stress client and buffer them.

The “regular” multi-threaded server has issues maintaining consistent communication with the client as rapid fire connections come in with no limit on the number of threads that spin up. This issue seems to be more of a client issue, as the server doesn’t ultimately crash. At times the server will either randomly drop connections from the client, or the client will terminate the connections due to a local crash of some kind. The server will also fail to hold connections from the client at a set limit defined in the listen call. This defines how many connections can be held in a queue, for the server to eventually process. It’s possible that the client is disconnecting as some of these connections that come in are just dropped, which causes issues in the client. In other words, without a thread pool, the server runs unhinged and is unable to correctly manage all of the connections coming in. As a result, the server has to drop some of the connections to prevent itself from crashing. This is also likely a hardware limitation and could be avoided if we use stronger hardware to support a rather unstable server. That being said, with the parameters we are running our server seems stable enough and is much faster than the single threaded server. This is likely due to the context switching when the treads go to sleep for 500 milliseconds, and the OS is able to switch to another thread that actually needs to get work done. Hence, while slightly more unstable, the multithreaded server is much more capable at handling the stress client.

For the thread pool server, we notice that the server runs the various queries from the client in a chunked out manner. Due to only having 30 threads to work with, the server is limited on how many concurrent jobs are allowed to be running. As a result, the server appears to be running the queries in chunks of 30, because of the 30 threads we have to work with. While this may be slower than the multithreaded server, it is more stable and as a result is less likely to drop connections to the client and vise-versa. It also limits the overall resources usage of the server and allows use of the computers hardware by other applications.