Kevin Funderburg
Advanced Network Programming

Homework 2 Summary

This is a summary of the homework, execution/configuration instructions are in README.md.

Problem 1

In this problem, the Nagle algorithm was disabled by setting the `TCP_NODELAY` option to 1.

In order to bypass `fgets` and `fputs`, the input is gathered from the command line using the `getchar()` function.

The goal of the Nagle's algorithm is to reduce the number of small packets on a WAN. The algorithm accomplishes this by delaying the sending of small packets of new data until there is no outstanding data (data sent but has not been acknowledged). This will result in fewer packets being sent, and ultimately slower performance time.

When Nagle's algorithm is disabled, there is no delay when sending small packets, this should result in faster performance time, but many more packets sent over the WAN.

Unfortunately, it was difficult to observe the difference between when Nagle's algorithm was disabled, this is most likely due to the speed of the WAN, so although the algorithm is disabled, it does not seem to affect the performance in this demonstration.

Problem 2

Problem 2 is fairly self-explanatory and can be inferred via the print statements and comments within `2-pthreads.c`

Problem 3

3.1 – To test the whether the value of `rl_key` variable differs, I made concurrent clients to the server, and as all the messages are sent and received from the server, it would appear the `rl_key` value remains 1 for all. This may be due to coding error, but as far as the tests would show, the `rl_key` value remains the same for all threads. However, the `rl_key` is different from the `diff_key`, which represents another thread specific data item.

3.2 - To test part 2 of problem 3, a new thread specific data item was created within the `main` function and its only job was to contain a string of characters. This was created successfully, and since this data item was created before the readline data item, the `diff_key` (which

represents the key for the new thread specific data item) had a value of 0, which while the `rl_key` had a value of 1.

Problem 4

4.1 - The `readline` function in Fig.26.12, p.693 calls `pthread_once` at line 42. What is the purpose of this function call?

> The purpose of `pthread_once` function is to initialize a key for a thread-specific data item, it guarantees that the `pthread_key_create` function is called only by the first thread.

What will happen if that function is not called?

> If `pthread_once` is not called, then each successive thread will call `pthread_key_create` which will re-initialize the key.

Does the `readline` function still work correctly without making the function call?

> When commenting out the `pthread_once` line, the following error is returned:
> `pthread_setspecific error: Invalid argument`

4.2 - Why is the return value of the `my_read` function (called by `readline`) of `static` type?

> By declaring the return value of `my_read` as static, this encapsulates the function for each thread that calls it.

What is the consequence of changing it to no-static?

> For testing purposes, I changed the return type to just `ssize_t` and ran the client and server and was unable to see any changes.