Andrew Hoskins

811939260

CSCI 4730

I have added to this project the required semaphores of full and empty along with the mutex lock (used binary semaphore). I also adjusted the functions of listener and thread_control, while adding another function of ConsumerThread. To implement and hold the consumer-producer model, I decided to use an array that is essentially acts as a buffer for our consumers and producers. To keep track of the overall status of the buffer, we use global counter variables of incoming and outgoing to help manage it. My listener function gets requests from clients and puts it in the array buffer. Every time this function gets requests from clients, it puts the request in the array buffer and adds one to the incoming counter. This is a critical section so we have to lock down this area properly which is why we use the binary semaphore mutex. Using this mutex allows my program to avoid the concurrency problem of race conditions. It is not only important to lock down this critical section with a mutex, but we must also check to make sure that the buffer is within its bounds and not going over. This is why I use sem_wait(&sem_empty) before we lock the critical section. The job of the empty semaphore is to try and add the request to the buffer, but if the buffer is full then we halt. Once these two semaphores have passed, our other semaphore full is called to make sure that our buffer is up to date. The next function is thread_control which address the consumer-producer model. This function creates the listener thread that will communicate to the clients with pthread_create. The parameters of this command also allow us to pass the listener function into it thus giving us a functional listener. The same method is used for our client/consumers with the added help of a counting variable to make sure that enough threads are made. We also don't pass listener with this command because we are dealing with the clients so I passed the ConsumerThread. My final function of ConsumerThread has a job of actually doing the tasks. This function deals with many threads trying to access it so we know that we are dealing with a critical section. A critical section means we have to avoid concurrency problems such as race conditions and we do this with the help of our mutex. Specifically, our critical section within this function is incrementing the outgoing variable of our buffer array. The mutex lock is first put before and after our critical section which lock and unlocks it. Then we put our full semaphore before the mutex with wait. This checks to make sure the buffer is empty. Once we have made it to the last mutex, we update semaphore empty. Lastly, we finish managing the request by calling process. Need to destroy the semaphores once main function is done.

An example of a test I did was ./client 127.0.0.1 6223 10 with ./webserver 6223 .This test took around 10 seconds. A followup was done with ./client 127.0.0.1 5666 10 with ./webserver_multi 5666 10 which took around 3 seconds.

Notes: The submission tarball example says to submit README, makefile, and webserver_multi.c. However, one requirement listed to include all source files needed to compile. Also, the provided Makefile has instructions for webserver.c, net.c, and webserver.h, so to make sure everything is delivered right I included those c files in.