# 1 Log of Commands

# 2 Concurrency Solution

We create ten threads, five producer threads, and five comsumer threads. The producer thread will call the function producer_start that will run forever. In the function we get a random value and sleep for that amount of time. We then get two random values one of which we use to determine the amount of time that the consumer will sleep for and the other is the value that the consumer will print out after the consumption time. We then use a semaphore to keep track of the space availible in the buffer. If the space becomes full this will block the producer until another space becomes avalible. Next we use a mutex to lock the other treads out while we put the item into the buffer. Finally we will increment the index of the buffer, unlock the buffer, and increment an items semaphore that will keep track of wether or not there are any items are in the buffer and if none are it will block. When we create the consumer threads we call the consumer_start. This function will first decrement the items semaphore, lock the mutex, get the item out of the buffer, decrement the index of the buffer, unlock the mutex and increment the space semaphore, and finally sleep for the consumption time and prints the value and frees the memory. We also have an infinite loop at the end of the main function so the program will not end.

# 3 List of Commands

-nographic disables the graphical output and redirects serial I/Os the console

-kernel uses a 'bzImage' as the kernel image.

-drive uses a 'file' as a drive image.

-enable enable KVM full virtualization support.

-net none use it alone to have zero network devices.

-usb enable the USB driver.

-localtime

–no-reboot exit instead of rebooting.

-append cmdline use 'cmdline' as kernel command line

# 4 Point of the assignment

To understand how multiple threads working at the same time need to use synchronization when they are sharing data the can be manipulated by each of the threads indivually.

# 5 how was the problem approached

We first read how to solve the problem in the Little Book Of Semaphores. we then looked up the semaphores syntax in c. We also looked pthread syntax. We first got the random numbers to generate using one of the options avalible. We then created the struct and the pthreads. We got the producer function working and then we got the consumer function working.

## 6 How did we test the solution

We added in print statements and minipulated the number of producers and consumeres to make sure that the semaphores were properly blocking. We made an abundent amount of producers and a small amount of consumers. Initially the program had a lot of statements being printed by all of the produce threads and eventually the number of print statements decreased and got to a point where they only printed after a consumer message printed proving that the producer was being blocked until a consumer removed something from the buffer. To test that the semaphore was blocking when there was nothing in the buffer we preformed the same test except we had an abundant amount of consumers and a small amount of producers.

## 7 What did we learn

That the semaphores are very useful tools for helping with synchronization across multiple threads. Type declarations can be very important. We accidentally had the get random number fuction returning the wrong type and it caused our whole program to not run the way we expected.