Ben Williams
CPRE 308
Lab 5

The purpose of this lab was to introduce Computer Engineering Students to creating a print spooler server that can support multiple printers. It was very challenging and asked a lot of our knowledge of the C language, core concepts of operating systems, and different data structures. Before I talk about the print driver itself, there were some Pre Lab questions I answered below.

What is the expected output?
Identical expected sum, averages, and standard devs
e_sum: 6846.000000
e_avg: 6.846000
e_sum: 18906.314453
e_std: 4.348139

What is the calculated output?
Slightly different than the expected output due to a lack of mutex locks
sum: 6846.000000
avg: 6.846000
sum: 18906.296875
std: 4.348137

What caused the discrepancy between the expected and calculated values?
The lack of thread safe memory protections put into place

Did this fix the issue with the original code?
Yes
e_std: 4.348139 std: 4.348139

What is the minimum number of conditions needed for the example to work as intended?
1: A variable that is incremented when each thread accesses the queue. When the producer see that the number is higher than the number of threads, it knows that starvation is occuring and can access the queue while resetting threadAccesses to 0.
2: A variable indicating that the thread is locked or not, basically a standard mutex

What would those conditions be, and which thread(producer or consumer) should wait on that condition?
1 - The producer waits on the condition being equal to the number of consumer threads, and then the producer can fill the queue with more data
2 - Consumer should wait on the producer to finish filling the queue with data

I began programming the assignment by completeing queue.c. This file contained the methods and member variables for a queue data structure, and I used semaphores and mutexes together to lock and hold the queue when popping or pushing was occuring. This is necessary due to the fact that several spooler threads will be attempting to read from the queue at once and it is important that jobs are distributed in complete and integral manner.

After completing queue.c, I moved on to print_job.c, where I simply finished the print_job_to_string()

function by adding in a few extra parameters to the printf() statement.

Finally, I moved on to main.c. Here, I filled in additional arguments for the command line argument parser, competed the print job spooler function, and made the main body of execution. I simply followed the step by step instructions provided by the comments from the professor.

To start, I created an arguments structure and parsed the incoming arguments from the command line into the program. I also created 2 queues, one for each type of printer I will be driving. I then created an array of print driver objects according to the number specified by the user. I installed them all using the given install_driver() function.

I then created n1 and n2 number of PRINTER_SPOOLER_PARAM structs, and gave them the necessary input information (queue, log file, and the associated printer driver.)

Finally, I began multithreading. I created n1 and n2 number of threads using the printer_spooler() function I defined earlier in the code.

These threads are the consumers in the producer-consumer relationship. They wait for a new item to be added to the queue, and exit if it is NULL. They then print to the log file using the print_job_to_string() function and fopen()/fclose(). They then finally call the print method on the print driver object, and destroy the print job.

While this is all happening, the main thread is constantly prompting the user for input on another postscript file name. The program opens the postscript file, scans through it and initializes different attributes of the current print job (using strncmp() and fgets()), and then pushes that new job to the queue.

When #exit is read in the input prompt, all of the threads are joined and terminated by calling queue_release_threads and then joining all of the threads, making sure they all properly terminate. The queue data is then finally released using queue destroy and the producer thread exits.

My program only partially works, unfortunately. There have been setbacks and time constraints on this lab, and I have only had so many hours to allocate. My printer spooler will print out the first job as expected, but after that things get complicated. I know there is a memory leak, as the program eventually crashes after so many seconds/minutes. There are also occasional segfaults when there are too many jobs in the queue. I wish this weren't the case but I figured I would be honest and say that I will work twice as hard to remedy these shortcomings as well as complete lab 6 in the coming week, now that I don't have 2 tests and projects in every class.