Homework 5 - Producer-Consumer Problem

Due by 7:00 p.m. Wednesday, 4/15/15

Part 1

Update the code you submitted for homework 4 to use a one-dimensional global array to hold all of the character counts, so that all the threads increment the same array entries without using any synchronization. Then create a second version that protects the increment of the global array using a mutex. Run both versions on the same file and compare the character counts. Which are larger? Why?

Part 2

Implement a multi-threaded producer-consumer program using a bounded buffer. Use `full` and `empty` semaphores to keep track of the numbers of full and empty slots available to the consumers and producers, respectively. Use a mutex to coordinate access to the buffer (once the thread determines that there is an available slot via the appropriate semaphore). The buffer should have 16 slots total. The number of producers, the number of consumers, and the number of items each producer produces should be specified by their binary log as command-line parameters. Thus, the following command should generate 8 producers, 16 consumers, and 64 items produced by each producer:

```
./producer-consumer 3 4 6
```

The consumers need to each consume the same number of items before exiting. Your code should calculate the number of items each consumer needs to consume so that every item produced gets consumed. The main thread should parse all of the command-line parameters, print them in a message, initialize the synchronization objects, spawn all of the threads, wait for them to complete, and then print a final message. The items produced by the producer threads should be integers, obtained using the following expression:

```
thread_number * num_produced + counter
```

where `thread_number` is an integer passed to each thread ranging from `0` to `num_threads - 1`, `num_produced` is the number of items produced by each producer, and `counter` is a local variable in each thread that gets initialized to `0` and incremented every time a new item is produced. The consumer threads should consume these items by simply printing them. (Printing is the only required "consumption".)

Use the same functions listed in homework 4 for thread creation and management. Use `pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_unlock`, `sem_init`, `sem_wait` and `sem_post` for synchronization in POSIX. Use `CreateSemaphore`, `WaitForSingleObject` and `ReleaseSemaphore` for synchronization in Win32.

You should submit your source code files (one for each platform) and a short writeup in pdf format that includes a description of what you did and the compilation and execution output from each of your programs. For part 1, briefly discuss the difference, if any, between the counts produced with and without synchronization. (If you don't see a difference, try using a larger input file.) For part 2, test your code with 1 producer and 2 consumers, 2 producers and 1 consumer, 2 producers and 4 consumers, and 4 consumers and 2 producers, each time with the producers producing 32 items each. Submit everything (including the writeup) to the regular submission link on iLearn, and then submit just the writeup to the TurnItIn link to generate an originality report.