Homework 5 - Producer-Consumer Problem

Due by 5:00 p.m. Tuesday, 10/28/14

Part 1

Update your code submitted for homework 4 to use a single global variable to hold all of the word counts. Each thread will increment this variable every time it finds a word in its partition. Run (and time) this code without any synchronization and compare it to the results you obtained with your original code (using separate local count variables that are summed together by the main thread). Then, add code that protects the increment of the global variable using a mutex. Run and time this version as well.

Part 2

Implment a producer-consumer program using a bounded buffer. Use `full` and `empty` semaphores to keep track of the number of full and empty slots available to the consumers and producers, respectively. Use a mutex to coordinate access to the buffer once it is determined that there is an available slot via the appropriate semaphore. Command-line parameters should be used to set the number of slots in the buffer, the number of producers, the number of consumers, and the number of items each producer produces before exiting. The consumers need to consume all items produced. (The number of items each consumer will need to consume can be calculated from these parameters.) The main thread should parse all of these command-line parameters, print them in a message, initialize the buffer and synchronization objects, spawn all of the producer and consumer threads, wait for the threads to complete, and then print a final message. The producer and consumer threads should print each item they produce or consume and update the buffer accordingly. (Printing is the only required "consumption" of the items.) The items produced should be integers obtained using the following expression:

```
counter * num_threads + thread_number
```

where `counter` is a local variable in each thread that gets initialized to `0` and incremented every time a new item is produced, and `thread_number` is an integer passed to each thread ranging from `0` to `num_threads - 1`.

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 execution times as they relate to the number of threads and number of cores on each platform and the manner in which the threads maintain their counts (locally, globally without synchronization, or globally with synchronization). For part 2, run and time both versions of your code with 1 producer and 1 consumer, with 2 producers and 2 consumers, and with 4 producers and 4 consumers, each time with 4 slots in the buffer and the producers producing 1000 items each. Again, briefly discuss the execution times as they relate to the number of threads and number of cores on each platform. 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.