Homework 4 - Word Count

Due by 7:00 p.m. Monday, 3/17/14

For this assignment, you will implement a multi-threaded word count program. Your program will receive the name of a text file as a command-line argument and output the number of words in the file. In your code, set up a 100KB buffer in global memory and read the contents of the file into the buffer (or as much of it as will fit). Then spawn a number of threads to each do a word count of a separate partition of the buffer. (Use a `define` preprocessor directive to specify the number of threads.) To count the number of words in a partition, loop through its contents and increment a local counter every time you encounter a word character (a letter or numeral) immediately following a non-word character (or at the beginning of the partition). Be sure to check for both upper- and lower-case letters. After looping throug the partition, each thread should store the output of its local word count in a global `int` array (with one entry for each thread). The partition bounds can be calculated in the main thread and passed to the worker threads using a `struct`. The partitions should be roughly equal in size, and there should be a non-word character at the boundary between each pair of partitions so words don't get double-counted. (It doesn't matter whether the non-word character is at the end of one partition or the start of the next.) You will also need to pass a thread index to each thread so that it knows where to write its local count in the global `int` array. Once the worker threads finish writing their local counts, they should exit. After spawning the worker threads, the main thread should wait for them all to complete. It should then add the partition word counts into a single overall word count, print the overall count and exit. Here is a sample execution:

```
~$ ./wordCount text_file
5280 words
```

The POSIX version of your program should use the `pthread_attr_init()`, `pthread_create()`, `pthread_exit()` and `pthread_join()` functions; follow Figure 4.9 in the book. The Win32 version should use `CreateThread()`, `WaitForSingleObject()` and `CloseHandle()` functions; follow Figure 4.10. Test each version of your program using 1, 2, 4 and 8 threads, and record the time taken by each using the `time` command for the POSIX version or the `Measure-Command` command (available in the PowerShell) for the Win32 version. Also report the number of cores on each test platform.

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. Also discuss the execution times as a function of the number of threads and number of cores on each platform. Submit everything to the regular submission link on iLearn, and then submit just the writeup to the TurnItIn link to generate an originality report.