Quan Nguyen
Homework 5 (Write-Up for Lab 4)
06/01/2018

Question 4

a. I run each `file_search` and `file_search_threaded` program 3 times to search for a same search term in a same (huge) starting directory. The average run-time of `file_search_threaded` is 848667 microseconds. The average run-time of `file_search` is 189 microseconds. Apparently, the run-time of multi-threaded version is 4490 times slower than the non-threaded version.
    i. The multi-threading with N threads not automatically lead to Nx speedup because of several reasons. Firstly, usages of threads in the program may not be efficient if the programmer does not understand clearly where and how to use them. Some threads may repeat the same works which other threads had done before them. Secondly, the workloads are not equally distributed among the threads. Some threads may end up doing more work than others. As a result, the threads with light workloads will have to wait (doing nothing) for other threads whose workloads are heavier to finishes their tasks.
    ii. Based on my test results, the performance of the multi-threaded version depends greatly on the directory structure being searched. Suppose that `file_search_threaded` program does its searching process in a starting directory that has 10 subdirectories. The first subdirectory that the thread 1 works on is 1000 times larger than the other 9 subdirectories. By the time thread 2, 3 and 4 finish their searching processes in those 9 subdirectories, the thread 1 is still working on the first subdirectory. That leads to longer run-time; and the thread 2, 3, and 4 are not efficiently used since they have a lot of free time waiting for thread 1 to finish its work.
    iii. The performance of the multi-threaded version could be improved if I managed the thread creation differently. In my program, the 4 threads are not used as efficiently as possible. Firstly, even though the threads run independently, there are overlapping works among them. Secondly, since the workloads are not distributed equally among the threads, some of them waste their time waiting for others. Thirdly, I could have use conditional variables to signal between threads so that when the threads finish all their works, they can help the thread which is currently working on a much heavier workload.
b. The `file_search_threaded` program is IO-bound. It spends most of its time waiting on the disk. Just looking at the directory/file names is a tiny fraction of time compared to what it takes to do the `opendir()` and `readdir()` operations which are very IO-intensive. The multi-threaded version should outperform the original version if the following workloads are used:
    i. The program uses all its threads to look for the search term in one subdirectory at a time. When they are done with that subdirectory, the threads will move to

another subdirectory, and so on. By doing that, there will be no thread spending its time waiting for others to finish their works.

ii. As mentioned above, when a thread initiates an IO request it goes into waiting state and does nothing productive. To improve the performance, the program must do something useful while one thread is initiating IO requests. In our program, while one thread is discovering the subdirectories and files in the starting directory (e.g. doing `opendir()` for a directory stream variable DIR), as soon as a subdirectory or a file is discovered, the other threads will immediately perform the searching process on that subdirectory/file, instead of waiting for the first thread to complete discovering all subdirectories and files.

c. Running the same program twice in quick succession (usually) result in better performance the second time because the OS integrates virtual memory pages and file system pages into a unified page cache to store important blocks. With caching, every file open does not require reads for every level in the directory hierarchy. The `file_search_threaded` program does a lot of directory opening for reading data. The first open may generate a lot of I/O traffic to read in directory inode and data, but subsequent file opens of that same file (or files in the same directory) will mostly hit in the cache and thus no I/O is needed. Therefore, the second run of `file_search_threaded` is faster than the first one.