Concurrent file system crawler

Due at 11:59pm on Friday, 2 December 2016

1 Requirement

Disks attached to laptops and desktops have become absurdly large. This generally leads users to accumulate large numbers of files in a large number of directories – i.e. rather than manage the use of disk space, it is easier to simply create more directories, retaining older versions of files.

Some operating systems have addressed this by continuously indexing the names of files in the file system, and then providing the ability to query the index – e.g. Google desktop. Server systems generally do not continuously index file names, instead providing one or more applications for searching the file system for files whose names match the desired patterns. Linux, for example, provides the `find` application for general traversal of the file system – see the man page for more details.

The syntax for the `find` application is extremely difficult to master. You are already familiar with the filename patterns that are understood by `bash`. The goal for this assignment is to design and build a high-performance application that will recursively crawl through a set of specified directories, looking for filenames that match a specified `bash` pattern.

For very large software systems, a singly-threaded application to crawl the directories may take a very long time. The purpose of this assessed exercise is to develop a concurrent file system crawler in C, exploiting the concurrency features of PThreads, to rapidly, recursively crawl through the nominated directories, looking for filenames that match a specified pattern.

2 Specification

You are to create a file named `file_crawler.c`. Your program must understand the following arguments:

`pattern`	indicates a `bash` pattern that filenames must match
`directory`	directory to be recursively crawled for matching filenames

The usage string is: `./file_crawler pattern [directory] ...`

If a directory is not specified, the application must search in the current directory, "`.`".

The application must use the following environment variable when it runs:

`CRAWLER_THREADS` – if this is defined, it specifies the number of worker threads that the application must create; if it is not defined, then two (2) worker threads should be created.

2.1 Bash pattern refresher

When you type a command to `bash`, it splits what you have typed into words. It scans each word for the characters '*', '?', and '['. If one of these characters appears, then the word is regarded as a pattern, and replaced with an alphabetically sorted list of filenames in the specified directory matching the pattern. Your `file_crawler` will look for these patterns recursively in the specified

directories. Note that when you specify pattern to file_crawler, you will have to enclose it in single quotes to prevent bash from interpreting it – e.g., '*.c'.

Any character that appears in a pattern, other than the special characters described below, matches itself.

The special pattern characters have the following meanings:

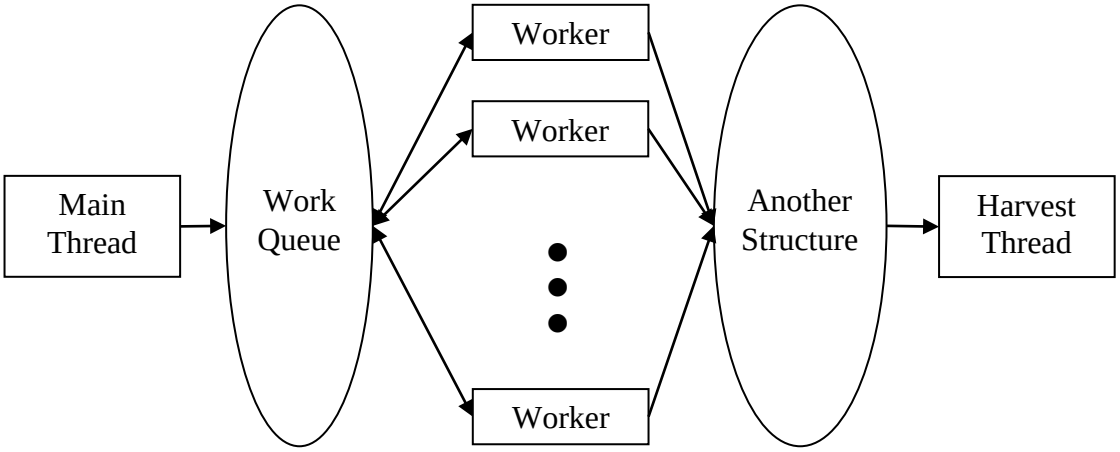
* Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by a hyphen denotes a range expression; any character that sorts between those two characters, inclusive, is matched. If the first character following the '[' is a '^', then any character **not** enclosed is matched.

3 Design and Implementation

The concurrent version of the application naturally employs the manager/worker pattern. An abstract version of this is shown in the figure below.



The Harvest Thread is often identical to the Main Thread, thus the Main Thread is the manager. It should be possible to adjust the number of worker threads to process the accumulated work queue in order to speed up the processing. Since the Work Queue and the Another Structure are shared between threads, you will need to use PThread's concurrency control mechanisms to implement appropriate [conditional] critical regions around the shared data structures.

For this assignment, the Main Thread writes the directory names from argv[] into the Work Queue; concurrently, the Worker Threads each retrieve directories from the Work Queue; for each directory retrieved, the Worker Thread scans the entries in the directory; for each such entry that is itself a directory, it writes that directory name into the Work Queue; for each such entry that is not itself the name of a directory, it attempts to match the pattern against the filename; if it matches, it places the fully-qualified filename into Another Structure.

3.1 How to proceed

The first thing most programmers do when developing a manager/worker application is to first build the application with only a single worker thread. In fact, one can design a singly-threaded program that assumes three phases:

1. populate the Work Queue
2. process the Work Queue, placing the processed data in the Another Structure
3. harvest the data in the Another Structure, printing out the results

I have provided you with a working, singly-threaded file_crawler_single.c. I have also provided extensive comments in file_crawler_single.c with respect to the design of the application, along with other modules that are needed to implement the program. Note that file_crawler_single.c recursively processes directory names found in a directory – ***this is not correct for this project***, since the specification requires that each such directory should be placed into the Work Queue for processing by other worker threads.

Starting with file_crawler_single.c, you should move to a version in which there is a single worker thread concurrently retrieving directories from the Work Queue and placing data into the Another Structure. This will require 1) thread-safe data structures, 2) that you determine an efficient way for the worker thread to determine when there are no more directories to be added to the Work Queue, and 3) for the main thread to determine when the worker thread has finished, so that it can harvest the data in the Another Structure.

After you have this version working, it should be straightforward to obtain the number of worker threads that should be created from the CRAWLER_THREADS environment variable, and create that many worker threads. The tricky part will be how the worker threads determine that there will be no more directories to be added to the Work Queue without reducing concurrency, and for the main thread to determine that all of the worker threads have finished (without busy waiting) so it can harvest the information.

4 Hints

Your efforts on Project 1 should have made you an expert on working with environment variables.

The container classes used by file_crawler_single are ***NOT*** thread safe. I have demonstrated in the lecture on threads and concurrency how to create a thread-safe container class from a non-thread-safe container class.

I have provided a sample directory tree populated with files on Canvas, as well as the correct output that one should obtain from all versions of your program when applying different patterns to that directory. The file "patterns" lists the mapping from bash pattern to output file.

5 Developing Your Code

You ***must*** develop your code in Linux running inside the virtual machine image provided to you. This gives you the benefit of taking snapshots of system state right before you do something

potentially risky or hazardous, so that if something goes horribly awry you can easily roll back to a safe state.

You should use your Bitbucket GIT repositories for keeping track of you programming work. As a reference, you can perform the command line steps below to create a new project directory and upload it to your uoregon-cis415 repository.

```
% cd /path/to/your/uoregon-cis415
% mkdir extracredit
% echo "This is a test file." >extracredit/testFile.txt
% git add extracredit
% git commit –m "Initial commit of extracredit project"
% git push –u origin master
```

Any subsequent changes or additions can be saved via the add, commit, and push commands.

6 Submission[1]

You are to create a .tgz archive containing the following files:

- Makefile[2] – a makefile that can be used to build your file_crawler executable
- File_crawler.c – the source file as described above
- (other .[ch] files) – you should include all .c and .h files upon which your file_crawler executable depends – you do not have to include unchanged copies of linkedlist.[ch], treeset.[ch], iterator.[ch], or re.[ch] from the starting archive
- report.pdf or report.txt – a report in PDF or text file format describing the state of your solution

You will submit your solutions electronically by uploading a gzipped tar archive via Canvas.

Your TGZ archive should be named "<duckid>-extracredit.tgz", where "<duckid>" is your duckid.

These files should be contained in a folder named "<duckid>". Thus, if you upload "jsventek-extracredit.tgz", then we should see the following when we execute the following command:

```
% tar –ztvf jsventek-project2.tgz
-rw-rw-r-- joe/None      1234 2015-03-30 16:30 jsventek/Makefile
-rw-rw-r-- joe/None      5125 2015-03-30 16:37 jsventek/file_crawler.c
-rw-rw-r-- joe/None    629454 2015-03-30 16:30 jsventek/tslinkedlist.h
-rw-rw-r-- joe/None    629454 2015-03-30 16:30 jsventek/tslinkedlist.c
-rw-rw-r-- joe/None    629454 2015-03-30 16:30 jsventek/tstreeset.h
-rw-rw-r-- joe/None    629454 2015-03-30 16:30 jsventek/tstreeset.c
-rw-rw-r-- joe/None    629454 2015-03-30 16:30 jsventek/tsiterator.h
```

[1] If the TGZ archive is not in the correct format, ***I will simply not mark your submission***. See the section in the Project 0 handout for the appropriate way to create the TGZ archive if you need a refresher.

[2] You must test your submission on the Linux virtual machine. If your code does not compile there, or does not link there, ***I will simply not mark your submission***.

```
-rw-rw-r-- joe/None    629454 2015-03-30 16:30 jsventek/tsiterator.c
-rw-rw-r-- joe/None    629454 2015-03-30 16:30 jsventek/report.pdf
```

Each of your source files must start with an “authorship statement”, contained in C comments, as follows:

- state your name, your login, and the title of the assignment (CIS 415 Extra Credit)
- state either “This is my own work.” or “This is my own work except that …”, as appropriate.

Marking Scheme for CIS415, Extra Credit Project

Your submission will be marked on a 100 point scale. I place substantial emphasis upon **WORKING** submissions, and you will note that a large fraction of the points are reserved for this aspect. It is to your advantage to ensure that whatever you submit compiles and runs correctly. The information returned to you will indicate the number of points awarded for the submission.

You must be sure that your code works correctly on the VirtualBox Linux VM, regardless of which platform you use for development and testing. Leave enough time in your development to fully test on the VM before submission.

The marking scheme is as follows:

Points	Description
10	Your report – honestly describes the state of your submission
90	File_crawler (+ other classes, if provided) 36 for workable solution (looks like it should work) 4 correct argument processing 2 correct processing of CRAWLER_THREADS 2 for successful compilation 2 for successful compilation with no warnings 4 reasonable, concurrency-safe class used for Work Queue 4 reasonable, concurrency-safe class used for Another Structure 5 efficient mechanism for determination when no more directories 5 efficient mechanism for determination when worker thread has finished 4 if it works correctly with the files in the test folder 4 if it works correctly with the files in an unseen folder of files 9 runtime performance with 1 worker on test folder is similar to single threaded implementation 9 runtime performance on test folder first improves, then degrades as number of threads is increased

Several things should be noted about the marking scheme:

- Your report needs to be honest. Stating to me that everything works and then finding that it won't even compile offends me. The 10 points associated with the report are probably the easiest 10 points you will ever earn as long as you are honest.
- If your solution does not look workable, then the points associated with successful compilation and lack of compilation errors are **not** available to you.
- The points associated with "workable solution" are the maximum number of points that can be awarded. If I deem that only part of the solution looks workable, then you will be awarded a portion of the points in that category.