CS 720/820 – Russell
Fall Semester, 2013
Assignment 2
Due: Monday, November 4, 2013

Modify and extend the "plcs" program you wrote for the previous assignment in order to incorporate some additional features and options. You MUST use this opportunity to correct all bugs and problems in your previous version of the program, since your new code is based on your previously submitted code.

Add 3 new features to your existing code: (i) the ability to search directories recursively, (ii) the use of threads to parallelize the search, (iii) selective processing of symbolic (soft) links. Also add 5 new options.

searching directories recursively

When processing names in the command line's `[list of input file names]`, if "plcs" encounters a name that is the name of a directory, then at that point search all the files and directories named in that directory, in the order in which they occur in that directory, except that all names in directories starting with a dot ('.), are silently ignored (but see the "-a" option below). Note that all names in the command line are always processed, whether or not they start with a dot ('.'). In addition, if any of the names in a directory are themselves directories, descend into them as you encounter them and process them recursively in the same manner. The depth of this recursion is limited only by the "-d" option (see below).

In all cases, before descending into a subdirectory, check if that subdirectory would create a directory loop in this recursive descent, and if so, do **not** descend into that subdirectory. (Repeated entries into the same directory are ok provided they occur via different descent paths). To detect loops, keep a history stack of the realpath names of all directories currently descended into. If a loop is detected, write an appropriate error message to standard error (but see the "-q" option below) that includes the descent depth (i.e., level) and full path (**not** the realpath, see below) of the subdirectory that would cause the loop. Then after that error message, for each directory in the loop print to standard error (but see the "-q" option below) one line containing its descent level and realpath, starting with the subdirectory that you were attempting to enter, and then going back through the history stack (i.e., in the reverse of the order you descended into those directories) until the same realpath is printed again. Each line in this list should be indented by 3*D spaces, where D is the descent level at which the loop was discovered. (D is initialized to 0 when processing command line names, is incremented by 1 each time a directory is descended into, and decremented by 1 each time "plcs" finishes processing a directory and resumes processing at the previous level.) The first and last line in this list should be identical. After printing the error message and the list of directory names in the loop, continue without descending into the subdirectory that caused the loop (i.e., if the name "abc" caused the loop, and this name was found in directory "xyz", continue with the name following "abc" in directory "xyz").

parallel searching with threads

Whenever a descent is made into a directory, all processing in that directory should be done by a new thread, so that in general there will be one active thread for each open directory (but see the "-t" option below). Do as much validity checking as possible before creating the new thread so that the new thread will actually have some work to do, and do not descend into that directory if any error occurs during this, including an error when actually creating the new thread. For example, make sure that the name is actually a directory that can be opened without exceeding the "-d" limit (explained below). Note that each thread must inherit its initial history stack from the thread that creates it, and then use its own history stack when descending into directories and detecting loops, independently of other threads. Different threads may have the same directory realpath on their own stacks — it is a loop only if the same directory realpath would occur twice on the history stack of a single thread.

processing symbolic links

Normally when a name in the command line or in a directory is a symbolic (soft) link, "plcs" will automatically follow that link, because it should use "fopen"() to open files, and "opendir"() to open directories, and both these functions always follow symbolic links. When the option "-f" (explained below) is present, do **not** follow symbolic (soft) links when processing names in directories. Instead, when such a name is found in a directory, write an appropriate message to standard error (but see the "-q" option below) that includes the full path (**not** the realpath, see below) of the name and then ignore that name.

-a When this option is present, names in a directory that start with a dot ('.') are processed, except for the two special names "." and "..", which are always silently ignored when found in a directory, with or without this option. Whether or not a "-a" option is present, all names in the command line are always processed, whether or not their first character is a dot ('.'), including the special files "." and "..".

-f When this option is present, "plcs" must **not** follow symbolic (soft) links when processing names in directories. Symbolic (soft) links in the command line's `[list of input file names]` are always processed, regardless of whether or not the "-f" option is present.

-q When this option is present, do **not** write an error message if a name obtained from a directory cannot be processed for any reason. You must still detect and recover from such an error in the normal manner, just do not print an error message. This includes **not** printing the error message or the realpaths of directories in the loop when a directory loop has been detected. Errors in the command line's `[list of input file names]` are always printed, regardless of whether or not the "-q" option is present.

-d **number**

where **number** must be non-negative. When this option is present, the depth of recursive descent into directories is limited to at most **number** levels. If the value of **number** is 0, "plcs" must not descend into any directories (which is the way it performed in assignment 1). If the value of **number** is 1, descend into any directories named in the command line but not into subdirectories named in those directories. If the value of **number** is 2, descend into directories named in the command line and their subdirectories, but not subsubdirectories named in those subdirectories. And so forth. If this limit is encountered during a descent, print to standard error (but see the "-q" option above) an appropriate message that includes the limit value and the full path (**not** the realpath, see below) of the directory that would exceed the limit, then continue without descending into that directory. Without this option there is no limit on the depth of recursion into directories. Regardless of the depth of recursion and the presence or absence of the "-d" option, **never** descend into a directory at any level that would cause a loop.

-t **number**

where **number** must be non-negative. This **number** limits the number of simultaneously active threads you can have at any one time (excluding the main thread). If spawning a thread to process a new directory would cause this limit to be exceeded, then do not spawn that new thread to process that new directory, but just silently have the current thread process that directory "in-line" (i.e., by having that same thread recursively descend into that new directory). Thus, if **number** is 0, never spawn any new threads, just have the single main thread process all directories "in-line" in the order they are encountered. Without this option there is no limit on the number of threads you can have simultaneously active. However, the system may impose such a limit, in which case the "pthread_create"() call will return an error. In this case, print the error message to standard error (but see the "-q" option above) and then process that directory "in-line". Note that once a newly spawned thread finishes with the directory for which it was spawned, it should exit, thus reducing the number of simultaneously active threads, which may thereby allow a new thread to be spawned for the next directory subsequently detected by any other active thread.

full path

The "full path" of a (relative) name found in a directory is the realpath of the directory followed by a slash ('/') followed by the name. The "full path" of a (absolute) name starting with a slash ('/') in the command line is just the name itself. The "full path", **not** the realpath, should always be used when printing a name in all error messages (because the realpath may not exist).

The same rules for option order, multiple options, error messages (but see the "-q" option above), error recovery, output, etc., as in the previous assignment also apply in this one. Names should be processed in the order they appear in the command line or in a directory. Because directories are processed in parallel, each by a separate thread, the order in which selected lines and/or error messages are printed may vary from run to run, and lines from different directories may be interleaved in the output. This is ok — just be sure the output is not garbled (i.e.. be sure that what should be together on one line appears together on one line, and what should be on separate lines is not run together onto the same line).

Using the gcc compiler, your program should compile and run with no changes on agate.cs.unh.edu, newton.cs.unh.edu, and topaz.cs.unh.edu. You must create a Makefile for use by gmake to build your executable on all platforms. The executable must be called "plcs". You must create a shell script called "dotest" that contains a sequence of tests to demonstrate the error detection facilities of your program as well as its correct operation.

Submit all your source files (no object, executable, or “tar” files, please), your Makefile, your “dotest” script and any test input files by typing:

```
~cs720/submit 2 file1 file2 file3 ...
```

where 2 is the number of this assignment. You MUST be on “agate.cs.unh.edu” to run submit! Please put your name in every file you submit. Note that “submit” will not process directory names.