CSCI415—Systems Programming

Spring 2017

Programming Assignment 5

Multithreaded Text Analysis

This project requires that you design and implement a multithreaded program that performs some rudimentary analysis of files that contain text. The program is invoked as:

`counter -b` *numlines* `-t` *maxcounters* `-d` *filedelay* `-D` *threaddelay* `file ...`

The end effect of this program's execution is the output (on `stdout`) of two word lists. The first list contains words found in the input files that have an odd number of characters and the number of times each word appears in the input files. The list must be alphabetized in lexicographic order. The second list is similar, but contains words with an even number of characters.

Required Thread Structure

You must have a **reader thread** whose job is to input lines of the designated files and to store those lines in a buffer. The reader thread is the only thread to engage in file input. Between reading two lines, you must have the reader thread sleep for *filedelay* milliseconds as specified in the `-d` option on the command line.

The buffer must contain sufficient space to store *numlines* lines of text as specified in the `-b` command line option.

You must also have at least one **counting thread**. A counting thread will (in a loop) get a line of text from the buffer, process the line word by word, and update the lists as appropriate. After a counting thread gets a line of text from the buffer, it must sleep for *threaddelay* milliseconds, as specified in the `-D` command line option, before updating any list data.

Ultimately, after all input and list updating is complete, the program (main thread) will output a nicely formatted pair of lists.

A Dynamic Pool of Counter Threads

An execution of your system will always have at least one counter thread. However, if the buffer ever becomes full, a new counter thread should be created. The rationale for this is that a full buffer is an indication of an overloaded "counting subsystem," and a new counter may help reduce the counting backlog. However, once there are *maxcounters* counter threads, as specified by the `-t` command line option, no more counter threads will be created. Once a counter thread is created, it will remain in existence for the remainder of the execution of the program.

Each counter thread should be created with a "name" that is a single lower-case alphabetic character. The initial counter thread is associated with the letter a, the second with b, etc. After processing a word by updating the lists as appropriate, a thread should output its letter on `stdout`. The sequence of letters output by the counters must precede the output of the word data.

Your Source Code

The TA will examine your source code as part of the grading procedure. He will do this to make sure that the structure of your system is in keeping with the specifications. You would be well-advised to produce neat and readable source with a level of commentary to help the TA in his task.

Notes

The following points should be considered as you design your system:

- You need not enforce that the input files be text files. We will only test your submission with files containing printable ASCII characters. You may assume that files will be structured as strings of printable ASCII characters separated by whitespace. Whitespace is defined consistent with the definition in the manual page for `isspace(3)`. Note that what we consider a "word" may include punctuation characters and need not be a valid English word; a word is just a sequence of printing characters separated from other words by whitespace.
- You may assume (and need not enforce) that a line of text is no longer than 2047 characters (2048 if you include the null byte terminator).
- You will need to devise a scheme by which the reader thread and all counter threads terminate cleanly. The exact form of that scheme is up to you.
- Command line options may appear in any order. All options, and the associated values, are required for execution of the program. You must check command line arguments for validity, aborting with an appropriate message if there is a problem. The following rules define valid values for the options:
  - *numlines* is a positive non-zero integer;
  - *maxcounters* is a positive non-zero integer no greater than 26;
  - *filedelay* is a positive integer (zero allowed); and
  - *threaddelay* is a positive integer (zero allowed).
- You may assume that a file (path) name is no longer than 4095 characters.
- All command line tokens after the required options are assumed to be file names. If a file can not be opened for reading, you should output a warning message on `stderr`. You should not process that file, but should continue processing the remaining files.

- I suggest checking out the manual page for `nanosleep(2)` as the preferred way to arrange for required intervals of sleeping for your threads.
- You are well advised to test your program on the departmental processors named `bg7`, `bg8`, and `bg9`. These are multiprocessor, hyperthreaded, systems. They will allow several of your threads to actually be in execution simultaneously.

Deadline

See the web page for the due date and time.

Preparing Your Submission

You must put your source files in a single directory; call it `counter.d`. That directory should also contain your `Makefile`. Make sure that `counter.d` is protected 0700 to prevent others from looking at your source. Make the *parent* of `counter.d` your working directory and do the following:

- `tar zcvf counter.tgz counter.d`
- `gpg -r CSCI415 --sign -e counter.tgz`
- `mv counter.tgz.gpg ~ ; chmod 0644 ~/counter.tgz.gpg`

The TA will unpack your submission and change to the directory in which your source lives to do the `make`. He will then run your program from that directory. **Note that the text files to be processed need not be in that directory.**