Institute of Technology
University of Washington – Tacoma
Instructor: Wes Lloyd

Assignment 1

File Reporter

Due Date: Monday October 24th, 2016 @ 11:59 pm, tentative

Objective

The purpose of this assignment is to write a Linux C program that uses multiple threads to launch Linux utilities to run in parallel to produce a comprehensive report about an input file. To implement this assignment use pthread_create() and pthread_join().

Additionally, for this program, please use popen() to invoke external programs. See the man pages for more details. "popen()" automatically creates a pipe for interprocess communication while calling fork() to invoke the shell to run external programs. Though it would be possible to implement this assignment using fork(), wait(), and exec(), use of popen() is probably a better choice.

The three programs that will be run include:

"wc"	to report the line count, word count, and character count
"md5sum"	to generate a unique 128-bit md5 (checksum) hash message digest
"grep –c"	to count the number of occurrences of a given word

The motivation is that writing a program to perform these tasks in parallel on a multi-core machine will allow the sum of all of the operations to complete is less time. Multi-threading can exercise a multi-core CPU to perform unrelated tasks (*embarrassingly parallel*) simultaneously to achieve a performance speedup versus performing the tasks separately.

This assignment provides an introducing to threading programming.

Input

The input format is as follows:

```
$./fileReporter {filename} {term_to_count_occurenences_of}
```

Where {filename} is the name of the file to report on, and {term_to_count_occurrences_of} is the term which will be passed to "grep –c" to count occurrences of in {filename}.

The output is:
Line 1 - A title line describing the name of the file being processed
(Lines 2-4 can be returned in any order):
For these lines the function is described in parenthesis followed by a ":", a space, and the report information.
Line 2 – Reports the line count, word count, and character count of {filename}
Line 3 – Reports the md5sum has of {filename}

Line 4 – Provides a count of {term_to_count_occurrences_of} for {filename}

Output

Here is a sample input output sequence:

```
# sequential version
$ time ./fileReporter /var/log/syslog print
File Report on: /var/log/syslog
(WC):lines=9233647 words=200024597 chars=1472447331
(MD5SUM): md5sum=e459e5dee94fd8be39810af467712fac
(WRDCNT): COUNT OF 'print'=84256

real  0m22.241s
user  0m21.277s
sys   0m0.884s

# parallel version
$ time ./fileReporter /var/log/syslog print
File Report on: /var/log/syslog
(WC):lines=9233649 words=200024637 chars=1472447793
(MD5SUM): md5sum=60aa1e4e4ce38900a3a0ed39e771a486
(WRDCNT): COUNT OF 'print'=84256

real  0m16.588s
user  0m19.953s
sys   0m0.608s
```

(the md5sum above does not match because /var/log/syslog changed between the two runs)

IF SOME FUNCTIONALITY IS MISSING, PLEASE FOLLOW THE OUTPUT FORMAT AND USE PLACEHOLDERS.

For example, use a placeholder like:
"`(WRDCNT): not implemented`", if the (WRDCNT) feature is not implemented.

As a recommendation, it *may be* best to work on the wrapper functions separately starting with "wc", then "md5sum", and finally "grep –c".

To encourage good programming practices, ***ALL*** wrapper functions for wc, md5sum, and grep –c should perform no console I/O. These wrappers should be standalone. All data required for the popen shell commands should be passed in. The wrappers should process (a.k.a. PARSE) output from popen to produce output text which is returned as results. Results display to the console should be in int main() or in another output function. Decoupling display I/O (view) from the popen "business" wrapper routines (model) provides an implementation of the common Model-View-Controller software architectural pattern.

MVC

Popen wrappers	model
Display routine (*optional*) or int main()	view
int main()	controller

Parallel Speedup

We see that running the fileReporter program on a large /var/log/syslog file (In this case 1.4 GB) provides a 25.4% speedup.

When implementing the fileReporter program it is possible to write two implementations: one- where every pthread_create() is proceeded by a pthread_join() for sequential operation of each task, and two- where all of the pthread_create() functions are spawned in order, and all pthread_join() calls are grouped at the end to allow processing to occur in parallel.

For this assignment, write a wrapper function for each of the three primary operations: wc, md5sum, and grep –c. Or alternatively, write a generic routine which performs the operation to minimize code duplicate since wrapper functions for these three routines will share a lot in common.

Grading

This assignment will be scored out of 90* points. (90/90)=100%

* *If necessary the total points scored from may be lowered, while the total available points will remain 100.*

Rubric:

100 possible points: (Currently 10 extra credit points are available)

Report Toal:	75 points
10 points	Output the first line, which reports the {filename} for the report
15 points	Output from wc – word count
>>>	*5 points for the number of lines*
>>>	*5 points for the number of words*
>>>	*5 points for the number of characters*
15 points	Output of the md5sum hash
15 points	Output from grep -c
>>>	*5 points for providing the name of the term being counted*
>>>	*10 points for providing the number of occurrences of the term*
10 points	Decoupling I/O from the wrapper functions (MVC)
10 points	Proper use of pthread_create and pthread_join for parallel execution

Miscellaneous:	25 points
5 points	Program compiles, and does not crash upon testing
5 points	Coding style, formatting, and comments
5 points	Makefile with valid “all” and “clean” targets
10 points	Output format matches the example provided (even if a portion doesn’t work!)

WARNING!	
10 points	Automatic deduction if program is not named “fileReporter”

***** THE PROGRAM SHOULD BE CALLED “fileReporter.c” *****

FAILURE TO USE THIS NAME WILL RESULT IN A 10 point deduction.

What to Submit

For this assignment, submit a tar gzip archive as a single file upload to Canvas.

Package up all of the files into the single tar gzip archive.
This should include a makefile with "all" and "clean" targets.
The "-pthread" option should be included in the makefile to support compilation with the thread libraries.

Tar archive files can be created by going back one directory from the kernel module code with "`cd ..`", then issue the command "`tar czf <lastname_firstname>_A1.tar.gz my_dir`". Name the file with your last name underscore firstname underscore A1 for assignment 1. "my_dir" would be the directory that contains the source code and makefile. No other files should be submitted.

Pair Programming (not supported)

Assignment 1 programs must be designed and implemented individually by students in TCSS 422 – Fall 2016.