CS342 Operating Systems - Spring 2016
Project 3: Synchronization

Assigned: March 12, 2016, Saturday
Due date: March 28, 2016, Monday, 23:55

Objective: *Practice synchronization, practice use of Pthteads mutex andcondition variables, practice multithreaded programming, locking, readers/writer locks, and C/Linux.*

Part A:

In this project you will develop again a multi-threaded index generation program that will be similar to project 1-A and project 2-A, but this time your program will use bounded buffers (BBs) and synchronization. Again there will a main thread and several worker threads. But this time, for each worker thread, there will be a bounded buffer (BB) of size M to pass information from main thread to that worker thread. There will be N worker threads, hence N BBs. The main thread will read words from the input file as in project 1, and will partition the words (together with their line numbers) into the BBs. A BB can store at most N items. An item is a structure containing a word and its line number. When the main thread obtains an item from the input file, it puts the item into the corresponding BB. If the corresponding BB is full, the main thread should be blocked (wait) until that BB has space for a new item. Then main thread can continue. A worker thread will read the items arriving through its corresponding BB. If the corresponding BB has no item, the worker thread will block (wait) until the main tread puts an item into the respective BB. Then the worker thread can continue. When a worker thread retrieves an item from its BB, it will insert the item into a linked list (LL) in sorter order with respect to the word stored in the item. Hence a worker thread will maintain a sorted LL. This is insertion sort. Each worker thread will work similarly. Hence there will be N sorted LLs maintained in this manner. When all input is processed and all worker threads finish their work, then the main thread will access those LLs and generate the index to an output file.

You can implement your BB as an array of M items.

The maximum number of worker threads, MAXTHREADS, is 8. Minimum is 1. MAXBUFSIZE is 1000 items. You can assume that the max line size (including newline character) is 4096 in an input file. Maximum word length is 64 including the NULL character at the end of a string (word).

The program executable will be named as **ts_indexgen** and will have the following parameters:

ts_indexgen <n> <m> <infile> <outfile>

Here <n> is the worker count, <m> is size of a BB, <infile> is input text file (ascii file) and <outfile> is output text file. An example invocation can be:

ts_indexgen 5 50 in.txt out.txt

Some initial files to start with are given in github.com. You can start with them if you wish.

You will do the project in the way described here so that you can practice synchronization, mutex and condition variables. Otherwise, the project could be implemented in a much simpler way, of course. You will use POSIX Pthreads mutex and condition variables to solve the concurrency (synchronization) problems that you will face (mutual exclusion, sleeping, waking up, etc.). Make sure you study the related POSIX Pthreads API very well and write some small exercise programs to make sure that you learned them. Especially, the following API functions must be learned very well: functions related to creation and initialization of mutex and condition variables, functions to lock and unlock mutex variables, functions to wait, signal/broadcast on a condition variable. Study the design patterns that use mutex/condition variables for solving some synchronization problems. Try to use these patterns in your solution.

Please read the project 1 part A specification for remembering what indexgen program, the main process, and worker processes were doing.

Report:

Design and do some timing experiments and plot/tabulate the results. For example, you can measure the running time of your program for various values of parameters <n>, <m>, and the input file size. You will also upload your report, in PDF form.

Part B:

In this part you will implement a simple readers-writer lock library (**librwlock.a**) that will allow applications to use readers-writer locks. That means an application may use such a lock for reading or writing. Multiple readers can get the lock (in a shared manner), but only one writer (in an exclusive manner). Your library should define (in its header file) a structure "struct rwlock" to be the type of rw locks. In this way, an application should be able to define locks of this type. Below is the set of functions you will implement. You should not change this interface (function prototypes).

- void **rw_init** (struct rwlock *L). Initialize the lock.
- void **rw_reader_lock** (struct rwlock *L). Try to get a reader lock (shared lock). Wait until you get it. When lock is acquired, the function returns and the calling thread can start reading data.
- void **rw_reader_unlock** (struct rwlock *L). Release a reader lock.
- void **rw_writer_lock** (struct rwlock *L). Try to get a writer lock (exclusive lock). Wait until you get it. When lock is acquired, the function returns and the calling thread can start writing data.
- void **rw_writer_unlock** (struct rwlock *L). Release a writer lock.
- void **rw_destroy** (struct rwlock *L). Destroy the lock.

A multithreaded application will be able to use such a lock. Some threads may be readers and may want to have a shared lock. Some threads may be writers and may

want to have an exclusive lock. You have already seen a solution of this problem in the classroom and textbook using semaphores. Now you will use POSIX mutex variables (and condition variables if you find necessary).

A skeleton of the solution is given in github.com. There you also see a **sample application skeleton** that is using the rwlock library. Github also contains a Makefile that shows you how to compile and obtain a C library. Note that you should not change the library interface: rwlock.h file. An application will include it to use your library. We will write test applications that will include that header file. Hence you should not change the interface and implement functions with the expected behavior according to the specification.

Submission:

Put the following files into a project directory, tar the directory (using **tar xvf**), zip it (using **gzip**) and upload it to Moodle. You will also upload to PAGS (programming assignment grading system) if the teaching assistant requests it.

- ts_indexgen.c: contains your C program.
- rwlock.h: contains your library header file.
- rwlock.c: contains your library implementation.
- Makefile: Compiles the program and library.
- Report.pdf: Your report.
- README: Your name and ID and any additional information that you want to put.

Additional Information and Clarifications:

- The project will be done individually.
- There is documentation available on the course website and on Internet about Pthreads mutex and condition variables.
- More clarifications can be put later to the webpage of the course, just near the project assignment.
- Note that in this project you will use mutex and condition variables. Not semaphores; not monitors. C does not have monitor construct. Pthreads library has mutex and condition variables. And you will use them to protect critical sections (via mutexes) and wait for some event (via conditions variables). Note that condition variables need to be used in association with mutex variables. This is very important. Study the examples. Learn the patterns and apply them.
- Make sure you wait on a condition variable in a while loop. Don't forgot to lock a mutex variable before accessing a shared variable.
- The following web page contains some files that may be helpful. You can clone it into your local machine, if you wish.
    - https://github.com/korpeoglu/cs342-spring2016-p3