CS 460 Programming Assignment 3 MultiThreaded Game of Life
Final Milestone Due Apr 18, 2016 11:59pm 65 points

Goal:
Learn about POSIX threads, synchronization, and mutexes.

Description:
You will implement a multithreaded version of John Conway's Game of Life. The Game of Life is a simulation of organisms living, dying, and being born on a grid as described by the four rules:

1. Any live cell with fewer than two live neighbours dies, as if by loneliness.
2. Any live cell with more than three live neighbours dies, as if by overcrowding.
3. Any live cell with two or three live neighbours lives, unchanged, to the next generation.
4. Any dead cell with exactly three live neighbours comes to life.

Each rule is applied instantaneously at a generation. That means that the result of a rule does not affect other organisms until the following generation.
http://en.wikipedia.org/wiki/Conway's_Game_of_Life

Your Program:
Your program will read command line options the determine the input and output files, as well as the number of thread and number of generations to run. See the **file format** section below. The input and output files have the same format. The height and width of the board are not necessarily equal.

Immediately before writing the game board back to a file, display the total number of births and total number of deaths that occurred over all generations. You also need to display the time, in seconds, your program took to run all the generations. This time must not include any File I/O.

Your program will need to run each generation using multiple threads. For *n* threads, each thread must do (about) *1/n* of the total work. By using more than one thread the runtime of your program must decrease if you are using an SMP machine or multicore processor.

How you divide up the work among the threads is your design decision. There are many correct ways to do this and any that achieves a speed up and produces a correct answer will be acceptable.

If the -X option is **not** given to the program, display, for each generation, the total number of births and deaths for that generation. If incorrect command line options, or a nonexistent input file is given, display a nice usage message.

```
CS460_Life inputFile outputFile #Threads #Generations [-X]
```

File Format:
The format of the file is as follows:

boardwidth
boardheight
boardwidth digits per row, boardheight rows

A 0 indicates an empty spot, a 1 indicates a spot with an organism in it.

Constraints:
You may assume the height and width of the board are always evenly divisible by the number of threads to run (the smallest board is 24x24).

Sample Output:

```
bart$ time ./CS460_Life bigTable.life bigTable.gen5.life 4 2
Generation 1:   DEATHS: 8182132 BIRTHS: 2974039
Generation 2:   DEATHS: 2601626 BIRTHS: 1571647

TOTAL DEATHS: 10783758     TOTAL BIRTHS: 4545686

Time: 15 seconds

real    0m42.809s
user    0m54.803s
sys     0m0.880s
```

Functions (and such) you will (probably) need:
pthread_create(), pthread.h, pthread_exit(), pthread_kill(), pthread_mutex_lock(), clock_gettime(), pthread_mutex_unlock(), pthread_mutex_init(), fopen(), read(), write(), close(), fcntl.h, printf(), etc.

Subversion (or how do I submit my work?):

Name your project **CS460_Life_PUNetID**. I will check out your code using the following command:

zeus$ svn co -r {2016-04-19T00:01} svn+ssh://zeus.cs.pacificu.edu/home/***login***/*repos*/CS460_Life_PUNetID

This will pull out the revision made previous to 12:01 am on April 19, 2016.

- Do not check your executable into Subversion.
- Do not check your data files into Subversion.

Makefile Targets:

CS460_Life: build the executable CS460_Life at the root of the project.
clean:
valgrind: Run the following:

```
valgrind -v --leak-check=yes ./CS460_Life smallTable.life smallTableValgrind.life 1 4
```

Testing
The lab machines are single CPU, six-core machines[1]. Xeon is a dual CPU, single-core machine[2]. Both machines implement HyperThreading. Each machine has Subversion on it and you must test on each machine.

Xeon is not currently running but will be available for milestone 2 and 3.

You can restrict your executable to one core using the following command:

```
taskset -c 1 ./CS460_Life ......
```

Running Valgrind on your code *may* produce a 100x slowdown!

Valgrind may report that pthread_create leaks memory if you don't call pthread_join.

►If your code fails to work on any of the listed machines you will lose 70% of the total points.
From the shell the command: **cat /proc/cpuinfo** will display information about the current machine's CPU.

1 http://www.newegg.com/Product/Product.aspx?Item=N82E16819117402
2 http://ark.intel.com/products/27275/Intel-Xeon-Processor-2_80-GHz-512K-Cache-533-MHz-FSB

Milestones (all milestones are due at 11:59 pm):

(15 pts) April 1: Single process (no threading at all)*. Your code must contain a comment explaining how you will divide up the work among the N threads.   The plurality of points for M1 will be given to the comment.

(15 pts) April 11: Single threaded version working (no synchronization necessary, only one thread will be run!)*

(35 pts) Apr 18: Final milestone

*If you complete a milestone before the due date and want to continue working, you need to tell me which revision of the source code satisfies the milestone. That is the revision I will checkout and test, otherwise I'll just checkout by a timestamp.

Notes:

Sample input and output files will be posted on the class schedule. You will need to link against libpthread.so, ie: **`gcc –o CS460_Life CS460_Life.o -lpthread`**

Your project needs to be well commented and broken into functions at each milestone. Minimize global variables!   Build a struct to pass to your thread! Large boards with many generations may take many tens of minutes to complete.

Using the **H** command inside the linux utility *top* will show you each thread instead of each process.

If you want to display a game board as an image (black pixel for a living cell and white for an empty spot) use the commands:

```
echo P1 | cat - board.life | display -
```

To display large boards make take many tens of seconds. man display for more info!

Questions: **Name: ______________________________**

Record real, user, and sys runtimes as reported by *time -p.* Also, you need to use the *time* functions in Linux to determine how many seconds are spent running the generations and display that time to the screen. This time must not include any file I/O.

Be sure to note in your writeup if anyone else is running a CS460_Life program on the same machine at the same time (use top or ps to determine this).

1. Fill out the following tables and hand in a hard copy of the tables with your source code. All of your data must be in seconds (to two decimal places).

ALL TIMING DATA MUST BE COLLECTED USING -X.

How long did it take your program to run 100 generations on file mediumTable.life?

Threads	Lab Machine				Lab Machine (1 core)				Xeon			
	Real	User	Sys	Non FileI/O	Real	User	Sys	Non FileI/O	Real	User	Sys	Non FileI/O
1												
2												
4												
6												
8												

How long did it take your program to run 10 generations on file largeTable.life?

Threads	Lab Machine				Lab Machine (1 core)				Xeon			
	Real	User	Sys	Non FileI/O	Real	User	Sys	Non FileI/O	Real	User	Sys	Non FileI/O
1												
2												
4												
6												
8												

How long did it take your program to run 100 generations on file largeTable.life?

Threads	Lab Machine				Lab Machine (1 core)				Xeon			
	Real	User	Sys	Non FileI/O	Real	User	Sys	Non FileI/O	Real	User	Sys	Non FileI/O
1												
2												
4												
6												
8												

2. Based on your answer to question 1, how long (non-File I/O time), in seconds, do you **expect** CS460_Life to take to run the following scenarios using **two** threads on each machine using largeTable.life? Run your program on each using two threads for each of the following scenarios and record the observed runtime. All of your data must be in seconds.

Generations	Expected Runtime (non File I/O)			Observed Runtime (non File I/O)		
	Lab	Lab (1 core)	Xeon	Lab	Lab (1 core)	Xeon
50						
150						
200						

Answer the following questions in a file named Google Doc named **CS460_Life_PUNetID** and share that with will4614@pacificu.edu

3. Discuss and explain what may cause any differences you see between expected and observed runtime in question 2.

4. Use *man time* to explain the difference between the real, user, sys times reported in the tables above.

5. If you saw any significant difference in the runtimes in question 1 explain where they came from. If you did not, explain why you did not. A *significant difference* is a change of +/-20% of the runtime.

6. Which combination of machine/thread count gives the best performance boost over a single threaded model? Explain why this pairing had the best performance boost.

BONUS:
Explain how the echo P1 | cat | display chain of commands works together to display an image from a file containing a board. Where does P1 come from? What do the dashes mean?