CS 460 Programming Assignment 4 MultiThreaded Game of Life
Final Milestone Due May 1, 2018 1:00pm 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 to determine the input and output files, as well as the number of threads 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 nanoseconds, your program took to run all the generations. This time must not include any File I/O. Use clock_gettime() with the CLOCK_REALTIME clk_id to use the high resolution timer.

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 a 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. **You must always** display the time your program took to run all the generations

If the -F option is given to the program, do not write data to the output file.

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

File Format:

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:

```
coffee$ ./CS460_Life testcases/smallGame.life testcases/smallGame.gen3.life 4 3
Generation 1:      DEATHS: 81756      BIRTHS: 29611
Generation 2:      DEATHS: 25944      BIRTHS: 15480
Generation 3:      DEATHS: 14429      BIRTHS: 12456

TOTAL DEATHS: 122129      TOTAL BIRTHS: 57547

Time: 70,372,179 nanoseconds

coffee$ ./CS460_Life testcases/smallGame.life testcases/smallGame.gen3.life 4 3 -X

TOTAL DEATHS: 122129      TOTAL BIRTHS: 57547

Time: 70,372,179 nanoseconds
```

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.

Revision Control

Name your project **CS460_Life_PUNetID**.

Don't forget the **bin/** directory!

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 --track-origins=yes --leak-check=full --show-leak-kinds=all ./CS460_Life testcases/smallTable.life testcases/smallTableValgrind.life 1 4
```

Milestones

(10 pts) April 10, **11:59 pm**: **Correctness**. 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. The #Threads command line argument is ignored in this milestone. Do not call pthread_create.

(10 pts) April 19, **11:59 pm**: **Single threaded** version working (no synchronization necessary, only one thread will be run!). Call pthread_create exactly once. The #Threads command line argument is ignored in this milestone.

(45 pts) May 1: Final milestone **DUE IN CLASS**

Notes:

nanoseconds generally require an unsigned long long.

Total births and deaths also require an unsigned long long.

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.

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

If Valgrind reports that various pthread functions leak memory:
 Be sure you call pthread_join()
 Be sure you destroy any pthread datatype you init.

unsigned long long:

```
#include <locale.h>

unsigned long long timer = SOME_LARGE_NUMBER;

setlocale(LC_ALL, "");
printf("Time: %'llu nanoseconds\n",timer);
```

Performance Analysis

For the Final Milestone you'll need to take performance measurements on various machines with various number of threads. The class will pool their results to see how threading performs in different situations.