COMP 3430 Winter 2013 Assignment 2

Due Date: Wednesday, Mar 6, 11:59 PM

Notes and Style

- must be written in C
- your assignment code must be handed in electronically using the D2L drop box. Try early and re-submit later. Include a Makefile. Confusion over D2L at the last minute is your own fault.
- Use common-sense commenting. Provide function headers, comment regions, etc. No particular format or style is required.
- Error checking is extremely important in real world OS work. However, it is complex and can get in the way when trying to learn these concepts. You can assume extreme cases will not happen such as lack of memory or disk failure (and thus can assume that, e.g., read and write will never fail). Error check for things that you should report to the user or recover from, e.g., missing file.

Frogger

You will make a text version of the classic video game "frogger." (for more info see google). The rules to frogger are simple – get all the frogs home and you win. Run out of lives and you lose. The frog dies if it lands in water (the home bank is safe), and if it lands on a log, it moves with the log. To save explanation, try running a sample of the program (how yours will act) on aviary by typing the following command at the prompt

```
~young/frogger
```

`wasd` controls and `q` quits.

You have been provided with a `screen.h/screen.c` library that provides text-mode functions for you and a sample main with compilation instructions. Be sure to compile with the `–lcurses` flag to link to the curses (text graphics) library, and the `–pthread` flag to link to the posix-threads library.

- This library is simple to use. There is a screen buffer off screen. You draw to the buffer using the commands detailed in the library. To have the changes reflected to screen, use the refresh command. To *move*, e.g., the frog, you first draw a blank square where the frog WAS and then draw the frog at its new location.
- This library contains a sleep function that sleeps for the given number of ticks, where a tick is 20ms. Use this in your thread loops, e.g., to set the speed of screen redraw (1 tick?), frog animation, log movements, etc.
- note that you can set the log speeds simply by making each log sleep for longer before it moves. In my example, the rows sleep for 2, 3, 4, 5 ticks respectively (higher numbers are slower).
- **this library is not thread safe!** Create a mutex and make sure to lock it before using ANY screen calls. Not ensuring mutual exclusion will yield funny results.

Your program has the following requirements

- You must implement frogger as a highly-multithreaded program, and ensure no race conditions!

- You will create (at least) the following threads. You may find it useful to make additional ones to help you
  - a thread for each log. As a log leaves the screen, it must be destroyed, and a new thread is started to bring a new log onto the screen. Using a single thread for a row of logs is not acceptable (one log = one thread)
  - a thread for the frog. This handles the frog animation and other things.
  - a thread for the keyboard. This is very useful as you will get key input using getchar which is a blocking system call.
  - a thread to re-draw the screen / perform a refresh using the `screen_refresh` command. Do not do refreshes each time you draw, only here.
  - a main program that starts everything and then sleeps using a condition variable (`pthread_cond_wait`). When the game ends this thread is woken up (`pthread_cond_signal`) and cleans up.
- You **must** end / clean up the program cleanly by joining all threads to others until only the main one is left (who joins to who is your choice). Although in this case exit() will simply clean up, this is not acceptable in a case where your heavy threaded library may be a part of a bigger program. Every time you create a thread, you MUST think: who will join it and when?

Use posix threads and mutexes for implementation. **YOU MUST KEEP FINE-GRAINED MUTEXING**, e.g., only lock each log, frog, etc., as you need it. A single, global lock is missing the point (and extremely inefficient) and will yield very few marks (if any).

Be aware of joinable and detachable threads. (man `pthread_create`). What is the default? What happens to memory when the thread dies? What happens if you are making a lot of threads (logs) and killing them and do not clean up the memory?

Hint: get the logs working first to get practice with threads, and then get the frog moving around the screen. The worst contention will be when a log and keyboard are both moving the frog (and drawing appropriately).

HINT: stop the messy practice of working locally and uploading your files. Learn VI or pico (http://blog.interlinked.org/tutorials/vim_tutorial.html - vi is awesome when you learn tabs and split windows!!!). Working on the target machine dramatically lowers your recompile-test time and gets you done faster.

HHIINNTT:: USE GDB. seriously. I'm not joking. Your program segfaults? run it through gdb and do a backtrace and it tells you where. Your program deadlocks? GDB can let you look at where each thread is stopped so you can figure out who is contending. Printfs? Sure, if you don't mind taking hours to sift through threaded output…

```
gdb –tui ./frogger
```

Bonus (5%): Your program creates / destroys threads constantly as logs are created and destroyed – this is a huge overhead. Implement a thread pool to re-use threads once they are abandoned.