CS 3100 Lab 5: Threads (DRAFT)
Concurrency and Locks

In Lab 5, you will convert a single-threaded application into a multithreaded one. The libraries you will be introduced to are `ncurses` and `pthreads`. You will not be required to actually code any `ncurses` APIs.

Learning Objectives

- Install the `ncurses` library and documentation
- Modify a small program to use threads and the resulting program "thread-safe"

Assignment

Begin by installing the tools you will need for this assignment:

```
mkdir -p ~/cs3100/lab5
cd ~/cs3100/lab5
sudo apt-get update
sudo apt-get install ncurses-dev ncurses-doc
scp USER@icarus.cs.weber.edu:/var/classes/cs3100/lab5/lab5.tar .
tar xvf lab5.tar
```

The file lab5.tar file will create, when extracted, a folder called `template` which will contain these files:

```
window.c
Makefile
```

You are to modify `window.c` and enable it to be a multi-threaded application. `Window.c` creates two tiled windows with a horizontal line between them, and continuously writes to both screens simultaneously until the user hit a key on the keyboard. You will create a separate thread for each of the two windows but keep in mind that `ncurses` is NOT thread-safe, meaning it is not reentrant. You are required to use a mutex to prevent multiple simultaneous calls to any ncurses function. To find out which functions are `ncurses`, use the `man` command for every function called in the program. As delivered, `window.c` just creates the two windows and the horizontal bar, then exits. It is your job to get it to display the counts.

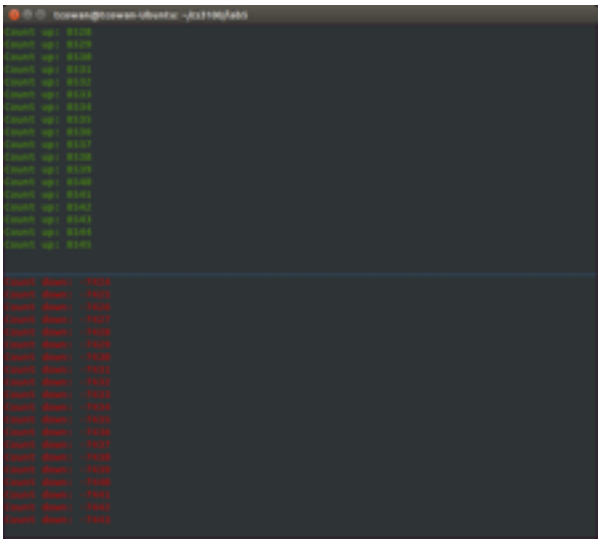
What you must do to `window.c` to make it multi-threaded:

Make these changes to `window.c`:

- Locate the code in `window.c` that initializes `ncurses`, creates the top and bottom windows and initialize color support. Just after the color support, create a mutex for controlling access to `ncurses`. Name your mutex variable "`ncurses`".
- Locate for two functions called `countUp()` and `countDown()` that prints to the appropriate window and performs the counting. These two functions should be identical except for the direction they count and the window they use for output. In `countUp()` and `countDown()` you should surround all of the calls to any `ncurses` function with your mutex, creating a critical section.

- Just after the comment "`// Thread code goes here!`", create two threads with the `pthreads` API, one for each window, calling `countUp()` and `countDown()` as appropriate. There are examples in the text and also in Advanced Programming in the Unix Environment. Of course, you may also research this on the Internet.
- Also add code to wait for each of the two threads to terminate.
- Compile and test. If one of your windows seems to lock up, look to see if you are forgetting to unlock the mutex. If the horizontal line disappears and it seems as if the threads are writing into the wrong window, look to see if you properly protected all of the `ncurses` function calls.

Sample Output



Your output should look something like this. The top window should always be green on black, the bottom red on black. There is a blue line between the two windows. There should never be any window corruption, the blue line should never disappear or change and it should take TWO keystrokes to end the program. We will discuss why in class.

Let your program run for a minute or so to ensure your solution is completely thread-safe. You will know almost immediately if it is not.

Files

Here is a list of the files created or modified in this lab.

`window.c`

Please add your name, lab # (5) and class # (CS 3100) as comments at the top of `window.c`.

Additional Requirements

- Please modify this program as simply as possible. Do not add any prompts, output or any file I/O except as specified for each program. Do not remove or rearrange any code from `window.c`.
- Do not modify `countUp()` or `countDown()` except to add the critical sections as required, along with any local variables you may need.
- Do not modify `main()` except for the thread creation and synchronization code and any local or global variables you may need.
- Upload only `window.c` to Canvas for grading as you turn in this assignment.

Grading

This assignment is a bit too complex for `cucumber` grading. I will compile and run your `window.c`, review your source code and give you points based on my observation.

- Upload your `window.c` file for this assignment to Canvas when you are ready for me to grade your work.
- It is not necessary to upload any files to icarus for this assignment.
- As always, you are free to deviate from the requirements for this assignment and you agree that if you do, you will receive a zero for the assignment.

Here is how you earn points for this assignment:

FEATURES	POINTS
Must-Have Features	
Only the following file is uploaded to Canvas: `window.c`	10
`window.c` compiles without errors or warnings	10
`window` implements pthread APIs to create two threads, one executing `countUP()`, the other `countDown()` and waits for both threads to complete.	
`window` correctly creates two equally sized windows and immediately begins counting up in the top window in green and counting down in the bottom window, with a dashed line between the two windows	10
`window` causes one of the two windows to stop counting on a single keystroke	10
`window` causes the other window to stop counting on the second keystroke and terminates the program after a three second pause.	10
`window` runs without creating any screen artifacts, corrupting or removing the dashed line during execution or crashing during or at the end of execution	50
Grand Total	100