ECE391: Computer Systems Engineering **Fall 2018**
Problem Set 2 **Due: Tuesday Sep 25, 2:00pm**
3rd floor ECE391 dropbox #54

Answers to all problems must be **typed, printed, stapled** and turned in into the dropbox to receive credit. Work in groups of at least 3 and hand in one copy. You must put each and every group member's name and NetID on the problem set. Do not put your name on more than one problem set or you will get a 0.

Problem 1:

As preparation for MP2, you need to figure out how to do a couple of things with the VGA. There's some free documentation available at `http://www.osdever.net/FreeVGA/vga/vga.htm`, but you may use any documentation that you like to learn the necessary changes to registers, *etc.*

a. You must use VGA support to add a non-scrolling status bar. Figure out how to use VGA registers to separate the screen into two distinct regions. Explain the necessary register value settings, how the VGA acts upon them, and any relevant constraints that must be obeyed when setting up the status bar.

b. You must change the VGA's color palette. Figure out how to do so, and explain the sequence of register operations necessary to set a given color to a given 18-bit RGB (red, green, blue) value.

Problem 2:

As part of MP2, you will also write a device driver for the Tux controller boards in the lab. The documentation for this board can be found in the file `V:/ece391/mp2/module/mtcp.h` in the class directory under mp2. You will need to read it for the following questions.

a. For each of the following messages sent from the computer to the Tux controller, briefly explain when it should be sent, what effect it has on the device, and what message or messages are returned to the computer as a result: `MTCP_BIOC_ON`, `MTCP_LED_SET`.

b. For each of the following messages sent from the Tux controller to the computer, briefly explain when the device sends the message and what information is conveyed by the message: `MTCP_ACK`, `MTCP_BIOC_EVENT`, `MTCP_RESET`.

c. Now read the function header for `tuxctl_handle_packet` in `tuxctl-ioctl.c`—you will have to follow the pointer there to answer the question, too. In some cases, you may want to send a message to the Tux controller in response to a message just received by the computer (using `tuxctl_ldisc_put`). However, if the output buffer for the device is full, you cannot do so immediately. Nor can the code (executing in `tuxctl_handle_packet`) wait (*e.g.*, go to sleep). Explain in one sentence why the code cannot wait.

Problem 3: Synchronization

There is are two research laboratories (Lab A and Lab B) which can be occupied by both students and professors; however, the following rules must be satisfied:

- Students and professors may not occupy the same lab at any given time. To comply with fire hazard regulations, the maximum capacity of each lab is 6 people. However, there is NO limit on the number of students or professors that are waiting in line.
- At most one student or professor may enter a lab when an _**enter** function is called.
- Both students and professors will always try to enter Lab A first. If it is not available, then the person will try to enter Lab B. If both labs are unavailable, the person should wait.
- Professors have higher priority than students when entering **BOTH** Lab A and Lab B. For example, suppose Lab A is occupied by students, then in this case another student may enter only if there are NO professors waiting. Otherwise the student must wait (students already in the lab do not have to leave immediately). Note that condition 1 still applies and professors may only enter Lab A once the last student leaves. The same applies for Lab B.
- The _**exit** function will remove one professor or student from either lab.
- For either lab, priority does not need to be enforced among students or professors (ie. if student 1 arrives before student 2, student 1 does not necessarily need to enter the lab before student 2).

You are to implement a thread safe synchronization library to enforce the lab occupancy policy described above.

You may use only **ONE** spinlock in your design, and no other synchronization primitives may be used.

As these functions will be part of a thread safe library, they may be called simultaneously

Write the code for enter and exit of students/professors. A skeleton has been provided for you.

```
// You may add up to 7 elements to this struct.
// The type of synchronization primitive you may use is spinlock.
typedef struct ps_enter_exit_lock{


}ps_lock;


int professor_enter(ps_lock* ps) {


}


int professor_exit(ps_lock* ps) {


}


int student_enter(ps_lock* ps) {


}


int student_exit(ps_lock* ps) {


}
```