ECE391: Computer Systems Engineering **Spring 2017**
Problem Set 2 **Due: Tuesday Feb 14, 2:00pm**
3rd floor ECE391 dropbox #5

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 `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 a laboratory which can be occupied by either zombies or scientists. Now you are hired to design a system that satisfies the following rules:

- Either zombies or scientists may stay in the lab, but NOT both at the same time. At most 5 zombies or 2 scientists can stay in the lab, but there is NO limit for the number of zombies/scientists waiting in line.
- Whenever an enter function is called, at most one scientist or zombie may enter.
- Zombies have higher priority to enter the lab. This means that when the lab is occupied by scientists, scientists may enter as long as there are NO zombies waiting. But if a zombie arrives, no more scientists should enter the lab (the scientists in the lab already dont have to leave immediately). After the last scientist leaves, the zombies may enter.
- If the lab is empty, scientists may enter if there is NO zombie waiting.
- NO need to enforce priority among zombies or among scientists (ie. if A arrives before B, A does not necessarily enter the lab before B)

Write the code for enter and exit of scientist/zombie. Assume the code for each scientist/zombie runs simultaneously.

```
// You may add up to 4 elements to this struct.
// The type of synchronization primitive you may use is SpinLock.
typedef struct zs_enter_exit_lock{

}zs_lock;


int zombie_enter(zs_lock* zs) {


}

int zombie_exit(zs_lock* zs) {


}

int scientist_enter(zs_lock* zs) {


}

int scientist_exit(zs_lock* zs) {


}
```