ECE391: Computer Systems Engineering **Fall 2022**
Problem Set 2 **Due: 20th Sep, 5:59:59 p.m.in Gitlab**

Work in groups of at least four people. The person submitting the code **(and NO ONE ELSE IN THE GROUP)** should list all group members' netids, including their own, in the file `partners.txt`, with each netid on a separate line. For example, `partners.txt` should contain the following if submitted by `xiangl5`:

```
xiangl5
yiruiz2
tw12
huilit2
```

Note: There is no autograder for this problem set.

WARNING: Any `partners.txt` files that conflict with any other `partners.txt` file will earn 0 pts for all concerned.

Problem 1: Reading Device Documentation (4pts)

Here is a great website for VGA related info that you might find useful: http://www.osdever.net/FreeVGA/vga/vga.htm

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: Documentation in Files (6pts)

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 (18pts)

Peppa Pig loves biscuits and apple juice, and she often can't stop herself from enjoying these delicious foods. Daddy Pig is a bit worried about that since too much juice and high sugar biscuits will cause potential health conditions... Thanks to the ECE391 students, who promised Daddy Pig to develop a tiny food machine that helps Peppa Pig to control her diet.

Actually, ECE391 students didn't promise at all. It is Prof. Lumetta and Kalbarczyk. They are the bosses xD

The food machine will dispense biscuits and juice under the control of a spinlock with the following rule:

- Peppa Pig can only consume one item at a time (i.e. one juice or one biscuit).
- If biscuit and juice are both ready for Peppa Pig, she must finish the juice first.
- The machine can only hold up to 4 juice at a time. It returns -1 if it fails to produce new juice, and returns 0 otherwise.
- The machine can hold an infinite amount of biscuits (i.e. it will keep producing biscuits no matter what).
- Peppa isn't too picky about biscuit. That means if biscuit (a) and (b) are ready at the same time, she will happily eat either (priority of biscuit does not need to be enforced).
- Peppa isn't too picky about juice. That means if juice (a) and (b) are ready at the same time, she will happily drink either (priority of juice does not need to be enforced).

When a **produce** function is called (presumably by Daddy Pig) the machine will produce the requested item. If the machine is full (4 juice in the machine) the call should return -1, but all biscuit requests should go through. Peppa will request biscuit/juice through a **consume** function.

When the function returns, the item (biscuit or juice) is dispensed and consumed. The consume function should block if the machine does not have the requested item made, or if the rules above would be broken. You may not make any assumptions about the timing of the code. All requests/code is run simultaneously. Please fill in the functions below according to the requirements:

Note: The only type of synchronization primitive you may use is spinlock t. There is no limit to the number of biscuits the machine can hold, however there may be a restriction from your code. Please state this restriction in plain text.

Also you could assume the **fd** in the function argument section will not be NULL.

P.S: There is an error in the GitLab PS2 release for problem 3 solution.c. Please use the following functions instead of those already there in the solution.c

```
// You may add up to 3 elements to this struct.
// The type of synchronization primitive you may use is SpinLock.
typedef struct biscuit_juice_lock {



} fd_lock_t;


void produce_biscuit(fd_lock_t* fd) {









}


void consume_biscuit(fd_lock_t* fd) {









}

int produce_juice(fd_lock_t* fd) {









}

void consume_juice(fd_lock_t* fd) {









}
```