ECE 373 Assignment #5
Spring 2018

User Space stuff

Sometimes it is easier and/or better to not have more code inside the kernel space – this may be for easier debugging, access to other libraries, or other specific reasons. Using a part of the system call interface we can cheat a bit and get access to the kernel's memory space at the user level.

Those Blinking Users

This can be done in any number of user level programming languages, but we recommend you stick with C for now, at least for the PCI interface library magic.

Your user program needs to:

1) Find one of the ethernet devices in your Virtual Machine (perhaps e1000?) and map the appropriate BAR using the PCI libraries in userspace (libpci and/or libpci-devel)
2) Save the current LEDCTL value and print it for the user to read
3) Turn the LED2 and LED0 LEDs on for 2 seconds
4) Turn all LEDs off for 2 seconds
5) Loop 5 times and turn each LED (LED3, LED2, LED1, LED0) on for 1 second
6) Restore LEDCTL to initial value
7) Read and print the contents of the Good Packets Received statistics register
8) Clean up and exit

Remember that you'll need to link the pci and related libraries into your program with something like this: gcc -o eceled **-lpci -lz** eceled.c

Run this program a couple of times while e1000 is loaded in the Virtual Machine to see if you can catch any packets being sent and counted by reading the Good Packets Received register. Also run this with ledmon from Assignments 3 and 4 to verify the LED's are properly blinking. Remember that you can run this user program while the original driver is still loaded, as long as you don't do something that conflicts and confuses the driver. You can even do this on some other hardware if there's an LED to blink.

The Blink is over

Turn in these materials before or at the start of class on **Monday**, **21-May-2018:**

1. Source code to your user program and Makefile.

2. Note that you won't need to unload the existing kernel network driver to run this program. Give a short explain what this might imply.

3. A typescript of the user program running at least three times (partly to try to catch the packet counter

changing)

4. A note from someone (anyone, but our TA is a good choice) saying they actually saw it work.