Discussion 2/1/19

Assignment 2

- Implement a “card device” driver
- The driver will support the read() system call
- Reading a byte will return a number 0-51, each number must be unique until all 52 cards have been served. Reset at this point.

```
read(fd, buff, 52) // Puts numbers 0-51 into buffer in random order

read(fd, buff, 1) // Puts one number from available cards numbers into buff

read(fd, buff, 0) // Does nothing
```

Assignment 2 Resources

- http://140.120.7.21/LinuxRef/LDD/LinuxDeviceDrivers-intro.html (Actual link)
- https://www.tldp.org/LDP/lkmpg/2.6/html/x569.html

Blackjack

For this assignment you need to do the following:

- Write a program that plays Blackjack
    - External from kernel
- Have the program intelligently determine if an Ace should be interpreted as a 1 or an 11.
- Gets cards to display by reading bytes from the /dev/cards file.

```
The dealer:
? + 10
You:
4 + 10 = 14
Would you like to “hit” or
“stand”? hit
The dealer:
? + 10
You:
14 + 10 = 24 BUSTED!
You busted. Dealer wins.
```

printk

- The function printk is a kernel level printing function.
- It takes two parameters, a logging level and a formatter string (like printf).

```
printk(log_level, “message”)

printk(KERN_ALERT, “URGENT!!!”)

printk(KERN_ALERT, “\tNumber %d”, 5)
```

Initializing and Managing Driver

- You will need to register a driver module
- You will need to register functions to handle system calls
- To do this, import

```
#include <linux/init.h>
#include <linux/module.h>
```

Initializing Module

```
static int __init
hello_init(void)
{
        printk("Hello, world!\n");
        return 0;
}

module_init(hello_init);
```

Removing Module

```
static void __exit
hello_exit(void)
{
        printk("Goodbye, world!\n");
}

module_exit(hello_exit);
```

Compiling

Now, to compile and run the code. Change into the directory and build the module:

```
$ cd hello_printk
$ make

$ sudo insmod ./hello_printk.ko
$ dmesg | tail

$ sudo rmmod hello_printk
$ dmesg | tail
```

Registering System Calls

```
static struct file_operations fops = {
    .read = device_read,
    .owner = THIS_MODULE
};

static struct miscdevice your_device_config {
    MISC_DYNAMIC_MINOR, // Dynamically assign driver #
    “Name”,             // Name of your device in /dev/name
    &your_file_ops
}
```

Registering System Calls

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/miscdevice.h>

static int __init
your_driver_init(void) {
    return misc_register(&your_device_config);
}

module_init(your_drivier_init);
```

Registering System Calls

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/miscdevice.h>

static int __exit
your_driver_exit(void) {
   return misc_deregister(&your_device_config);
}

module_exit(your_drivier_init);
```