20-EECE-4029-001 **Introduction to Operating Systems** **Spring 2013**

Homework Assignment 2

processes, mutex, semaphores, memory management, producer-consumer, files, deadlock, more..

Kernel module to monitor battery status

Due: January 21, 2013 (submit instructions: here)

Rationale:

Gain a little insight into the design and use of ACPI and kernel module development. Observe how the OS interacts with the hardware. Note how data structures defined in the kernel sources are crucial to the generation and handling of interrupts.

Homework Problem:

Write a linux kernel module called `battcheck` that checks battery status every second and reports an interesting result: that is, reports when either 1) the battery begins to discharge or 2) when it has begun to charge or 3) when it has reached full charge or 4) when the battery level has become low (only one report) or 5) when the battery level becomes critical (report every 30 seconds). As root, you should be able to load the module with `insmod battcheck` and unload it with `rmmod battcheck`.

Base the module on acpi_call.c using this Makefile.

You will need to find the name of the method that your computer uses to access battery information. It is likely something similar to `\_SB_.BAT0._BIF`. Use ACPI spec for help. My call returns

```
[0x1, 0x15e0, 0x1360, 0x1, 0x2b5c, 0x25a, 0x160, 0xa, 0x19,
"Primary", " ", "LION", "Hewlett-Packard"]
```

which translates to mAh/mA (discharge units), 5600 mAh capacity, 4960 mAh last full charge, rechargeable, 11100 mV design voltage, low battery at 602 mAh, 352 mAh critical battery, Primary model, no serial number, Lithium Ion type, manuactured for Hewlett Packard. There is also something like `\_SB_.BAT0._BST` for getting battery status. My call returns

```
[0x0, 0xffffffff, 0x1360, 0x31ba]
```

which translates to charged, 0 current discharge rate, 4960 mAh charge remaining, 12.7 volts.

You can print the information any way you choose. You can use some graphics, or print to a `/proc` file or print to the console.

You will probably want to create an infinite loop in either the `init` or `read` functions of your module. I do not think this will work - if you can do it, you get 50 brownie points. The sample code below shows one way to cause an action to repeat. Even that may not work but try it. Whatever way you do it, the code should examine the values returned by the battery status (`_BST`) method. If the battery level is low (a simple comparison) or if it is discharging you want to alert the user. So as not to burden the processor, the (`_BST`) method could be called, say, every 1 second.

You cannot use `sleep` from the C language to do this - you must rely on kernel functions. The unit of time in the kernel is the `jiffie`. The constant `HZ` is 1 second's worth of `jiffie`s. You can make a module that dings once a second from the following:

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/timer.h>

static struct timer_list my_timer;

void my_timer_callback( unsigned long data ) {
   printk(KERN_EMERG "my_timer_callback called\n");
   mod_timer( &my_timer, jiffies + msecs_to_jiffies(2000) );
}

int init_module( void ) {
   int ret;

   printk("Timer module installing\n");

   // my_timer.function, my_timer.data
   setup_timer( &my_timer, my_timer_callback, 0 );

   printk("Starting timer to fire in 1 sec\n");
   ret = mod_timer( &my_timer, jiffies + msecs_to_jiffies(1000) );
   if (ret) printk("Error in mod_timer\n");

   return 0;
}
```

There is enough information in the above example to complete the assignment. Note that `KERN_EMERG` results in notification to the console.