CS 460 Programming Assignment 4 Kernel Hacking
Due May 11, 11:59pm 60 points

Goal:
Learn about how to compile the Linux Kernel (version 2.6.30.5), add a system call, and access kernel data structures.
This project is adapted from page 74 in your book. The instructions there are not *completely* correct for the kernel we are using but are a good guide!
Also inspired by: http://www.cs.columbia.edu/~hgs/teaching/os/hw3.html and http://www.linux.org/docs/ldp/howto/Implement-Sys-Call-Linux-2.6-i386/index.html

Description:
The Linux Kernel keeps track of how many context switches each task (process/thread) makes during its lifetime. It also tracks how many voluntary and involuntary context switches are made. You will need to add two system calls (**sys_getnvcsw, sys_getnivcsw**) to give a program access to each of these pieces of data.

This data described above is stored in two variables in the task_struct struct. nvcsw is the number of voluntary context switches. nivcsw is the number of involuntary context switches. The task_struct struct is defined in: /usr/src/linux/include/linux/sched.h The task_struct for the current process is always accessed through a #define macro ***current*** which is defined in /usr/src/linux/arch/sh/include/asm/current.h. These values are updated in the __schedule() function in /usr/src/linux/kernel/sched.c

If a process's time quantum has expired, but no other process is waiting to run, Linux will not switch the process off and on the CPU,it just leaves it on the CPU. The number of times a process *actually* is swapped off the CPU is reflected in the two task_struct member variables above and does not include these *skipped* context switches. You need to add a new member variable in task_struct to track all the possible times a process could have been switched off the CPU. This new value should include skipped context switches and actual context switches. Provide a system call (**sys_getncswPossible**) to access this data. Make sure this member variable is initialized in the same place that nvcsw is.

Test Program:

Write a small program to test your system calls. It just needs to display the results to the screen as follows. Before it calls your system calls, it should run an empty for loop 100,000,000 times (just to give it a chance to context switch).

```
$ ./CS460_TestSysCalls
nvcsw 4
nivcsw 0
possible 9
```

Resources:

Linux Kernel Cross reference: **http://lxr.linux.no**
http://www.linux.org/docs/ldp/howto/Implement-Sys-Call-Linux-2.6-i386/index.html
http://www.cs.columbia.edu/~hgs/teaching/os/hw3.html
http://lxr.free-electrons.com/source/?v=2.6.30
http://users.sosdg.org/~qiyong/lxr/source/?v=2.6.30
page 74 in your book.

How do I submit my work?

You must turn in a **PAPER** copy of all the files you create/edit along with a **PAPER** copy of your answers to the questions. You do not need to print out all of the existing kernel files (sched.c, sched.h) just the sections of the files you have changes. Make sure you have comments in the code highlighting your changes.

Questions:

Use largeTable.life for the questions below. You do **not** need to turn in a paper copy of your game of life.

1. Edit your game of life project to determine how many possible and actual (and what type) context switches happen:

    During the file read
    During the generations
    During the file write

    a) 1 thread, 20 generations
    b) 2 threads, 20 generations
    c) 4 threads, 20 generations

2. What differences do you see in the above measurements (File read vs generations vs file write, 1 thread vs 2 vs 4). Discuss what you think may cause them.

3. Run two instances of your game of life at the same time and record the context switch data for each of the three sections defined in 1.

    a) 1 thread (each), 20 generations
    b) 2 threads (each), 20 generations

4. Discuss how the data from 3 and 1 are the same and how they are different.