Kernel Modules and Character Device to list all processes

Accessing your Virtual Machines

Instructions to access your virtual machine remotely

This assignment will be carried out on a Linux virtual machine. Each student has been assigned a VM with a unique IP address. The login credentials have been emailed to you. You can either use the VM assigned to you or you can use your own Linux VM on your personal computer.

Assignment description

1. Implement a kernel module that creates a `/dev/process_list` character device. The character device should support the `read()` operation. When the `read()` system call is invoked on your character device from a user space process, your kernel module should return to the following information for all currently running processes:
    1. process ID
    2. parent process ID
    3. the CPU on which the process is running
    4. its current state.

    For example, the output could be as follows:

```
PID=1 PPID=0 CPU=4 STATE=TASK_RUNNING
PID=2 PPID=0 CPU=2 STATE=TASK_INTERRUPTIBLE
PID=10 PPID=2 CPU=0 STATE=TASK_UNINTERRUPTIBLE
PID=16434 PPID=16424 CPU=10 STATE=TASK_DEAD
PID=14820 PPID=16424 CPU=8 STATE=TASK_WAKEKILL,TASK_UNINTERRUPTIBLE
...and so forth
```

    Note that the state field in task_struct can be -1, 0 or greater than 0. A state value of 1 or more indicates a combination (bitwise OR) values listed here.

    So you will have to parse the `task->state` value to figure out which of the states apply to a process. You can pass the raw state value (as a `long` value) from kernel to user space and decode the state values in your user space program.

2. Also, provide a user-space C program that open your character device and outputs the list of processes retrieved from your character device.

   One such application could be written as follows (please fill in the missing code):

```
char *buffer;

/* allocate memory for character buffers HERE before you use them */

fd = open("/dev/process_list", O_RDONLY);
/* check for errors HERE */

while(!some termination condition)
{
  bytes_read = read(fd, buffer, buffer_length);
  /* check for errors HERE. Exit loop if all processes have been
retrieved. */
  /* print the output you have read so far. */
}

close(fd);
```

BEWARE that bugs in kernel code may either crash your kernel immediately or may have no immediate visible effect, but may have a delayed effect that is disastrous. Therefore, you cannot assume that the thing you did most recently is necessarily the cause of a crash.

Assignment Deadlines

```
       Mar 13: Submit any partial code you have written so far and a plan for
completing rest of the assignment.
       Mar 27: Final submission
```

Grading Guidelines

```
       Kernel Module- 60
       User-level code - 30
       Error checks and coding style - 10

       Total = 100
```

References

- Example kernel modules hello.c and hellon.c
- Kernel Cross Reference
- for_each_process() macro can help you iterate over all processes in the system.
- `struct task_struct` definition can be found here

- Process states are defined here
- The function task_cpu() can help you extract the currently assigned CPU for a process.
- Introductory material on Linux Kernel
    - Chapters 1 and 2 of the following online book provide a good introduction to the kernel, though with a bias towards device-driver development.
      http://lwn.net/Kernel/LDD3/
    - For more kernel programming help, just google "Linux Kernel" and you'll get lots more.