CS411 Operating Systems II Fall 2008

Project #1

Due 6:00 am, Saturday, 4 October 2008

Introduction:

The projects for this course will involve modifications and additions to the *linux* kernel. For most students, this is unfamiliar and intimidating territory. The goal of this project is to introduce you to the structure of the *linux* operating system, the coding standards used in *linux* kernel coding, and the development environment you will use for all of the course projects.

Part of this project involves modifications to the *noop* I/O scheduler. This is the simplest possible scheduler (***no*** fancy ***op***erations); it services I/O requests as first-come first-served, i.e., as a simple queue. You will make changes to some of the code, but not to the actual algorithm or structures. In other words, the code modifications are intended to be as easy as they seem.

Objectives:

- Learn how to
  1. upgrade a *linux* kernel
  2. use a virtual machine (VM) for testing kernel modifications
  3. use a revision control system to manage changes by various group members
  4. create/modify/insert a *linux* kernel module
  5. create a new system call in the *linux* kernel
- Review your team's development process

Assignment:

Note: The file references below are relative to the top of the *linux* source tree (i.e., the `linux-2.6.23.17` directory).

1 Follow the *VMWare* Guide on the course wiki to connect to your team's virtual machine.

2 Follow the *Subversion* Guide on the course wiki to set up your team's *Subversion* repository and initialize it with a stable copy of the linux kernel. For Fall 2008, we will use version **2.6.23.17**. Later versions will be available, but it is important that we use a common version (for using the *LXR* cross referencer, and for testing/grading).

3 Follow the Kernel Installation Guide on the course wiki to configure, compile, and install (on your VM) the kernel that you downloaded.

4 Create a new version of the *noop* I/O scheduler as described below. (**Note**: the coding should NOT be done on your VM. Each team member is required to do at least one of the sub-tasks and commit the modifications to the team's source code repository.)

4.1 Make a new directory (outside of the kernel tree) named **`proj01-iosched/`**. Copy the *noop* I/O scheduler source code (at **`block/noop-iosched.c`**) into the new directory, and name this copy **`proj01-iosched.c`**. Import the new directory into your *Subversion* repository.

4.2 Add a header block to **`proj01-iosched.c`** containing your team name and the names of your team members.

4.3 Change the names of the *noop* scheduler functions to begin with **`proj01`** (instead of **`noop`**). Similarly, change the name of the **`elevator-type`** function.

4.4 Modify the fields of the **`elevator-type`** structure so that your functions can be recognized by the kernel. Change the **`elevator-name`** field to your team's name (e.g., **`team07`**).

4.5 Change the fields of the author and description macros appropriately.

4.6 Upload your completed code to your VM (use *Subversion*).

4.7 Follow the Kernel Module Guide on the course wiki to perform the following steps. Make a *typescript* of these steps. (This typescript will be submitted as proof that you completed the module installation.)

4.7.a Compile your *proj01* I/O scheduler into a loadable kernel module on your VM.

4.7.b Insert your module into the kernel and have the kernel use your module to handle I/O scheduling for **`/dev/sdb`**.

5 Create a new system call as described below. (Note: This involves modification of existing kernel code, so be careful and precise.)

5.1 Create a new branch in your *Subversion* repository as described in the *Subversion* Guide. Name it **`linux-2.6.23.17-my_syscall`**. As you modify the code in this branch, check your changes regularly.

5.2 Within the new branch, write a new system call function in **`kernel/sys.c`** that identifies itself and then displays the system time. Name your function **`sys_team`***xx*, replacing *xx* with your team number, and provide a brief description of the function. Here's an example, showing how to do the first part of the requirement:

```
/* System call to identify the current process. */
asmlinkage long sys_team00 (void)
{
     printk ("sys_team00 called from process %d.\n",
                current->tgid);
     return 0;
}
```

5.3 Append your system call to **`arch/i386/kernel/syscall_table.S`**.

5.4 Edit **include/asm/unistd.h** to provide your system call's number. For example, if you see the lines

```
#define __NR_epoll_pwait      319

#define NR_syscalls 320
```

append your system call and change the final number, e.g.,

```
#define __NR_epoll_pwait      319
#define __NR_team00      320

#define NR_syscalls      321
```

(**Note**: These numbers are from *linux* 2.6.20.15 for *team00*. Your numbers will be different.)

5.5 Recompile the kernel. Verify that your system call is in **System.map** (use *grep*).

5.6 Install the modified kernel. Cross your fingers, toes, eyes, etc., and reboot. If the system reboots properly, your new system call will be available.

5.7 Test (and typescript) your system call with a short program, e.g. (with numbering changed appropriately):

```
#include <linux/unistd.h>
#include <sys/syscall.h>

#define __NR_team00 320

int main()
{
     syscall( __NR_team00 );
     return 0;
}
```

(**Note**: If you don't see the expected output on the screen, look at the *tail* of **dmesg**)

5.8 (Optional) Experiment with creating additional system calls. Read "Verifying the Parameters" on pp. 68 - 70 in your textbook, and have as much fun as time permits.

What to hand in:

- Create a gzipped tar file containing the items listed below, and have one team member submit it at http://engr.orst.edu/teach.
  1. Source code for your *proj01* I/O scheduler.
  2. A patch file created using **`svn diff`** that represents the changes between the vanilla 2.6.23.17 kernel and your **`2.6.23.17-my_syscall`** branch. Your patch file should be named **`my_syscall-team`***xx***`.patch`** (replace *xx* with your team number).
  3. Your compiled kernel image from the **`2.6.23.17-my_syscall`** branch. This file should be named **`vmlinuz-my_syscall-team`** *xx* (replace *xx* with your team number).
  4. Test program from section 5.7
  5. Typescripts from sections 4.7 and 5.7
  6. Your *Subversion* commit log
  7. (Optional) Similar evidence of extra work from 5.8
- Each member of your team must individually write a separate "review" document. (See the course wiki for the requirements.) This document must be submitted by 11:59 pm on Monday, October 6, at http://engr.orst.edu/teach.
- Before 9:00 am on Monday, October 6, your team must submit a signed hardcopy of your completed Credit Distribution Agreement.

Credits:

Rob Hess:

- *Subversion* Guide, *VMWare* Guide, Kernel Module Guide, Kernel Installation Guide
- Prototyping, defining, and testing parts 1 – 4.
- Setting up virtual machines, etc.

YunRim Park:

- Testing parts 1 – 4.
- Developing evaluation criteria.