CS 470 Project 2
Kernel System Calls
Due Date: Wed March 24, 2010

In this project you will implement a new system call for the Linux/Ubuntu operating system.

The project has two parts. In the first part, we will give you detailed instructions to add a simple system call to the Linux kernel. In the second part of the project, you will add your own new system call.

You are advised to do Part 1 immediately to avoid any complications. Compiling the Linux kernel can take hours on slower machines. **Follow the instructions below very carefully.** Missing a step could cause you to start all over.

1 Part 1: Modifying the Linux Kernel

In this section we will give you step-by-step instructions on how to add a new system call into the Linux Kernel.

You will do the project on Ubuntu running on top of Virtualbox. You may run Virtualbox on any machine you like, but you must run Ubuntu (Jaunty) in the Virtualbox. Your Ubuntu system will need a minimum of 6GB of free disk space to load the Linux kernel sources and compile them.

1.1 Compiling the Linux Kernel Source Code

You will use your Ubuntu system under Virtual Box to modify the linux kernel. Before you can compile the linux kernel, you will need to load some additional software as well as the linux kernel source code.

1. All of the instructions needed to compile the kernel must be executed as root. To become root:

   ```
   # Become the root user
   sudo /bin/bash
   ```

2. To get the kernel and compile it:

   ```
   # Load additional Ubuntu packages needed to compile the kernel
   apt-get install ncurses-dev

   # Load the linux source code
   apt-get install linux-source

   # Unpack the Linux source code
   cd /usr/src
   bunzip2 linux-source-2.6.31.tar.bz2
   tar -vxf linux-source-2.6.31.tar
   ```

3. Before you compile the kernel, you will need to configure it.

```
# Change to the linux source code directory
cd /usr/src/linux-source-2.6.31

# Modify the default configuration
make menuconfig
   (Under General - change the local version to be -cs470p2)
   (You can disable any kernel options you know you will not need.)
```

4. After you have configured the kernel, you need to compile it.

```
# Compile the kernel
make bzImage
# Compile the dynamically loadable kernel modules
make modules
```

5. Now you are ready to install the kernel you just compiled:

```
# Install the Kernel modules in /lib/modules
make modules_install

# Install the Kernel in /boot
make install

# Create an initrd image
cd /boot
mkinitramfs -o initrd.img-2.6.31.9-cs470p2 2.6.31.9-cs470p2

# Update the grub boot loader to include your new kernel
update-grub

# Reboot the computer to load your new kernel
/sbin/reboot
```

If all goes well, the machine should reboot with your new kernel. It will be the default kernel. After logging in, you can run the 'uname -r' command to see that you are running your new kernel.

1.2 Adding a New System Call

The following steps will create a new "helloworld" system call and then show you how to call your new system call from an application.

1. Become the root user – sudo /bin/bash

2. Create a new file to hold your system call code. Call it `/usr/src/linux-source-2.6.31/kernel/hello.c`.
   Type the following program into your new hello.c file:

```
#include <linux/linkage.h>
#include <linux/kernel.h>
#include <asm/uaccess.h>

#define MAX_BUF_SIZE 1000

asmlinkage int sys_helloworld(char __user *buff, int len) {

    char tmp[MAX_BUF_SIZE]; /* tmp buffer to copy user's string into */

    printk(KERN_EMERG "Entering helloworld().  The len is %d\n", len);
    if ( (len < 1) || (len >  MAX_BUF_SIZE) ) {
       printk(KERN_EMERG "helloworld() failed: illegal len (%d)!", len);
       return(-1);
    }

    /* copy buff from user space into a kernel buffer */
    if (copy_from_user(tmp, buff, len)) {
       printk(KERN_EMERG "helloworld() failed: copy_from_user() error");
       return(-1);
    }
    tmp[len] = '\0';
    printk(KERN_EMERG "Hello World from %s.\n", tmp);

    printk(KERN_EMERG "Exiting helloworld().\n");
    return(0);
}
```

3. Edit the Makefile to compile your new system call code, by editing `/usr/src/linux-source-2.6.31/kernel/Makefile`.
   Add `hello.o` to the definition of `obj-y`, like:

```
obj-y     = sched.o fork.o exec_domain.o panic.o printk.o \
            cpu.o exit.o itimer.o time.o softirq.o resource.o \
            sysctl.o capability.o ptrace.o timer.o user.o \
            signal.o sys.o kmod.o workqueue.o pid.o \
            rcupdate.o extable.o params.o posix-timers.o\
            kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o\
            hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
            notifier.o ksysfs.o pm_qos_params.o sched_clock.o cred.o\
            async.o hello.o
```

4. Add your system call to the unistd kernel header files by editing `/usr/src/linux-source-2.6.31/arch/x86/include/asm/unistd_32.h`. and add (near the bottom) the line:

```
#define __NR_helloworld  337
```

   so that your system call has a unique system call number.

5. Add your system call to the syscalls kernel header files by editing `/usr/src/linux-source-2.6.31/arch/x86/include/asm/syscalls.h`. and add the lines

```
/* kernel/hello.c */
asmlinkage int sys_helloworld(char __user *buff, int len);
```

   in the section that is `#ifdef CONFIG_X86_32`.

6. Modify the system call initialization table by editing `/usr/src/linux-source-2.6.31/arch/x86/kernel/syscall_table_32.S`. Add

```
  .long sys_helloworld
```

   after the last line of the file (i.e., after `.long sys_inotify_init1`)

7. Now recompile and load your kernel (this can take a long time)

```
# Move to the root of the linux source code
cd /usr/src/linux-source-2.6.31
make bzImage
make install
# Move to the boot directory
cd /boot
mkinitramfs -o initrd.img-2.6.31.9-cs470p2 2.6.31.9-cs470p2
update-grub
/sbin/reboot
```

8. Login to your new kernel and create a test program called helloworld.c. Change the string "FirstName LastName" in the program to your name.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/syscall.h>

/* Replace this with your name */
#define author "FirstName LastName"
#define __NR_helloworld  337
```

```
main(){
   int retval;

   printf("About to call the new helloworld() system call\n");
   retval = syscall(__NR_helloworld, author, strlen(author));
   printf("Done calling helloworld()\n");
   printf("Your helloworld() system call returned the value %d\n", retval);
}
```

9. Compile the program from step 7 and run it

```
gcc -o helloworld helloworld.c
./helloworld
```

   You can look for the messages printed by your new system call at the end of the /var/log/syslog file (you must be root to look at this file):

```
tail /var/log/syslog
```

2 Part 2: A new system call to print the linux process table

Now that you have seen the steps needed to add a system call to the Linux operating system, you are ready to create a new system call of your own. In particular, you must add a system call that will print (to the /var/log/syslog file) information taken from the Linux process table.

Your new system call must take one parameter

```
int procnum;
```

where procnum is the process id of the process that you should print information about. If procnum is 0, then you should print information about every process in the process table. If procnum is negative, you should print an error message.

To print information from the process table, you will need to include the file "linux/sched.h" in your system call. The "sched.h" file contains the definition of a process table entry which is called a `struct task_struct` in Linux. Find the definition of a `struct task_struct` in sched.h and look at the information contained in the structure. The information you should print out for a process are:

- the process id (`pid`) of the process
- the id (`real_parent->pid`) of the process' real parent
- the state (`state`) of the process
- the priority (`prio`) of the process
- the command name (`comm`) of the process

You will also find a macro call `for_each_process(p)` in the sched.h file that will help you read through all the processes in the process table.

You must then write an application to make use of your new system call. You can do this in exactly the same manner as you wrote the helloworld application above.

3 Other Resources

There are many helpful web pages that you can find (via google) regarding compiling the linux kernel. However, the instructions given above should suffice for compiling the kernel in this project.

The area where you may need to lookup some help is the area of kernel programming. Kernel programming is different from writing user-level programs in a few important ways. First, kernel routines are best written in C (avoiding C++ features). Second, the number of library functions available is greatly reduced. Only the most basic library functions are available. Third, input/output in the kernel is not as common as it is in user-level programs. If output is needed, the printk() command is used. Third, before system calls can use the data passed in by an application, it must be copied from the user space into the kernel using `copy_from_user()`. Similarly, passing data back to the user application requires use of the `copy_to_user()` routine. Additional documentation about kernel programming can be found at

- http://www.gnugeneration.com/mirrors/kernel-api/book1.html
- http://kernelbook.sourceforge.net/kernel-hacking.html (look especially under "Common Routines"), and
- http://www.linuxhq.com/lkprogram.html
- http://lwn.net/images/pdf/LDD3/ch05.pdf (talks about synchronization in the Linux Kernel)

4 What to Submit

Note: You do not need to submit any code from Part 1.
You should submit the code for your new system call and your new application as a gzipped tar file (.tgz):

- README file - listing all the files you think you are submitting
- Documentation file - brief description of your projet including the algorithms you used. Your external documentation should also include a description of any special features or limitations of your shell. This should be a simple text file. *Do not submit MS Word, postscript, or PDF files.*
- the new .c and .h files you created inside the kernel
- all header files associated with your user-level applications
- all C/C++ files associated with your user-level applications
- a file with an example run of your user-level program–gathered with the script command (see the script man page).

Instructions on where to submit your project will be sent out via email at a later date.