OS2021: Challenge #1: Linux Kernel

Jordi Mateo Fornés `jordi.mateo@udl.cat`

University of Lleida (Campus Igualada -UdL) — **From:** September 24, 2020; **To**: October 17, 2020

Scope

The Linux kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the 2, managing resources as efficiently as possible. The kernel, if appropriately implemented, is invisible to the user, working in its little world known as kernel space, where it allocates memory and keeps track of where everything is stored. System calls are commands that are executed by the operating system. There are several systems calls in Linux to manage the OS. (The man pages will show you more details among them.)

Command Line

```
$ man uname
$ man syscall
```

Objectives

- To understand the relationship between OS command interpreters (shells), system calls, and the kernel.
- Learn about modifying, compiling, debugging, and running the Linux kernel.
- To design and implement system calls.

Instructions

- This challenge can be delivered in groups between 2 or 3 students.
- The challenge must be delivered on GitHub. Please, in the submission in the Campus Virtual indicate the link to the repo.
- There are several different ways to approach these problems. It is your job to analyze them from an engineering point-of-view, determine the trade-offs, and to explain the implementation you select.
- Do not underestimate the importance of the write-up. Your project grade depends significantly on how well you understood what you were doing, and the write-up is the best way for you to demonstrate that understanding.
- Create a **README.md** or a **PDF** document with the project information about authors, workflow, design, implementation, files,... all the things you consider necessary to reproduce and understand your work.
- Append to the document the answers to the following questions: What have I learned from doing the challenge? How did I learn that? What has allowed me to improve? Why can it serve me?

Preliminaries

1. Install a Ubuntu OS in Virtual box. You can get the version 16.04 from:

```
Command Line
$   wget https://releases.ubuntu.com/16.04/ubuntu-16.04.7-desktop-amd64.iso
```

2. Install compilers and tools:

```
Command Line
$ sudo apt-get install build-essential libncurses-dev bison flex -y
$ sudo apt-get install libssl-dev libelf-dev vim -y
```

1 Working with System Calls in C — (2 points)

As you know function, procedure and library calls all refer to the same concept which is calling a block of code in user space. On the other hand, system call refers to calling an operating system service running in privileged kernel space. This separation is intentionally designed to achieve to two major goals: security and convenience.

Question 1

What is more expensive a System call or a Procedure call concerning **time**?

Question 2

How much more expensive is a System call or a Procedure call respect the other?

Question 3

Design an experiment to test and proof which call is more time expensive. Write a simple test program in C to compare the cost of a simple procedure call to a simple system call (i.e getpid()).

Hint:

- You should design and experiment with a million iterations or more doing a simple system call and another experiment with the same iterations calling some function you develop, such as merely returning an integer.
- You should use system calls such as `gettimeofday()` for time measurements.
- You should take a look in timeval struct.

2 Compiling and running the kernel — (1 points)

1. Grab the latest kernel from kernel.org
2. Verify kernel
3. Untar the kernel tarball
4. Copy existing Linux kernel config file
5. Compile and build Linux kernel
6. Install Linux kernel and modules (drivers)
7. Update Grub configuration
8. Reboot the system

Command Line

```
$ sudo apt-get update
$ sudo apt-get install git fakeroot build-essential ncurses-dev -y
$ sudo apt-get install xz-utils libssl-dev bc libelf-dev -y
$ wget https://mirrors.edge.kernel.org/pub/linux/kernel/v4.x/linux-4.7.1.tar.gz
$ tar xf linux-4.7.1.tar.gz
$ cd linux-4.7.1/
$ cp /boot/config-$(uname -r) .config
$ make menuconfig
$ nproc
$ sudo make -j 2 && sudo make modules_install -j 2 && sudo make install -j 2
$ reboot
```

Question 4

Explain the process to enter and check into the new kernel version.

Question 5

Research the roles of files in /boot/?

- vmlinuz-*
- initrd.img
- grub
- config*

3 Analysing kernel and system calls — (1.5 points)

Let's suppose that we define the following system call and we add to our kernel.

Question 6

Explain what this code is supposed to do.

kernel/sys.c

```
SYSCALL_DEFINE1(my_syscall, char *, msg)
{
  printk(KERN_INFO "my_syscall:\"%s\"\n", msg);
  return 0;
}
```

Question 7

Check or think if the execution of this code could be dangerous? If answer is yes, correct it.

4 Creating a Patch

Patch files are the normal way to distributed small changes to large source trees in the Unix/Linux development world. A patch file describes the differences between an original source tree and a modified tree. If you change only a few lines in a handful of files, the patch file will only contain a few lines. Moreover, it will be easy for someone to find, identify, and read all of your changes.

Command Line

```
$ diff -urN originalKernelSrc modifiedkernelSrc > patch1
```

A patch is created using the diff program. Another person can then apply the patch file using the patch program to the same original kernel tree and create the same result. You can read about the diff and patch programs in the online documentation for Linux.

Notice: For delivering the following activities it is essential to make a summary path file with the comparison between the modified files only and upload the information into Github.

5 Creating *GetInfoProc* System Call — (3 points)

We are going to create a system call that allows us to obtain information about a specific process such as the elapsed_time, start_time, sys_time and user_time. But feel free to return and store other information if you want.

Info: *getinfoproc(int pid, struct procinfo *pi)*

- **pid**: if pid !=–1 -> pid of process, else current process.
- **procinfo**: referenced argument to return the necessary info to user space.
- **return** the EINVAL on error or 0 on success.

Notice: **printk** is very useful for debugging messages.

Hint: To do this requires these things:

- Modify the kernel. Check: `arch/x86/entry/syscalls/syscall_64.tbl`
- Design and implement a new **system call** that will get this data back to the user application. Check: `include/linux/syscalls.h` and `kernel/sys.c`
- Write a C application that use the new system call. Remember to use: `#include<sys/syscall.h>` and `#define __NR_getinfoproc X`; where X is the number of the syscall in the file: `arch/x86/entry/syscalls/syscall_64.tbl` .

6 Hacking a system call — (2.5 points)

Hack your kernel to print " :) Your file `filename.txt` is being opened!!!" when the user opens a file.

TIP:

```
$ vi fs/open.c
```

References

- `https://www.linux.it/~rubini/docs/ksys/`
- `https://elinux.org/Debugging_by_printing`
- `https://0xax.gitbooks.io/linux-insides/content/SysCall/linux-syscall-5.html`
- `https://medium.com/@vishhvak/compiling-a-custom-linux-kernel-on-a-virtual-machine-12be9d32189b`