Operating Systems - Assignment 1

System Call

OSL 2019 - Sem. 182

Student: Nguyen Minh Nhat - ID: 1752039

---

1 Introduction

In this assignment, I am required to create a system call for the Linux Operating System. The system call is called `procmem`. The output of this system call should be the memory layout of a process specified by its process identification number (pid).

A **system call**, according to the Linux Manual [1], is the fundamental interface between an application and the Linux kernel. We can simply understand system calls as "tools" provided for applications to interact with the kernel and other low-level components of the computer. These "tools" are special in the way that they are:

- safe to use: application programmers do not have to worry about whether the application will do harm to the operating system as well as hardware components.
- easy to use: application programmers are provided with a higher-level of abstraction to interact with the operating system and hardware components through APIs.

2 Methodology

To complete this assignment, I have followed these steps:

- **Step 1**: Install a Virtual Machine with a Linux Operating System.
- **Step 2**: Figure out the logic of the system call.
- **Step 3**: Implement and test the logic of the system call through a kernel module.
- **Step 3**: Implement the system call.
- **Step 4**: Build and install the system call with a new kernel.
- **Step 5**: Write a wrapper for the system call.

Each of the steps will be covered in the following section.

2.1 Virtual Machine

In this project, we need to work with the kernel. It is a bad idea to work on an in-use machine because in case something wrong happens, the operating system may be corrupted and the machine may become unusable. Therefore, we are highly recommended to use a virtual machine.

Here is the description of the virtual machine that I used for this project:

Operating System	Ubuntu 18.04 LTS (Bionic Beaver) - 64bit
Linux Version (original)	4.15.0
Virtual Machine Software	Oracle Virtual Box 6.0.6
Storage	32GB (with 4GB for swap area)

The version of Ubuntu that I used for this assignment can be downloaded from here.

2.2 The Logic of the System Call

The system call that we were asked to implement is called `procmem`, which returns the description of the memory layout of a process.

2.2.1 Memory Layout of a Process

The memory of a process in Linux consists of these major parts:

- **Code segment**: contains the binary code of the program which is running as a process.
- **Data segment**: contains the initialized global variables and data structures.
- **Heap segment**: contains the dynamically-allocated variables.
- **Stack segment**: contains the local variables, return addresses, etc.

2.2.2 Getting information of a Process in Linux

In Linux, each process corresponds to a struct `task_struct`. In `task_struct`, there is a member struct `mm_struct` that stores the memory layout of the process:

Member of mm_struct	Data type	Description
start_code	unsigned long	where Code Segment starts
end_code	unsigned long	where Code Segment ends
start_data	unsigned long	where Data Segment starts
end_data	unsigned long	where Data Segment ends
start_brk	unsigned long	where Heap Segment starts
brk	unsigned long	where Heap Segment ends
start_stack	unsigned long	where Stack starts

The source code for `task_struct` and `mm_struct` can be found here and here, respectively.

2.2.3 The Logic of the System Call

2.3 Kernel Module

2.3.1 What are kernel modules?

According to The Linux Kernel Module Programming Guide [2], modules are pieces of code that can be loaded into and unloaded from the kernel upon demand. Without kernel modules, every time we need to add a new functionality to the kernel, we have to rebuild the whole kernel and directly add that functionality to the kernel image. The process of rebuilding the kernel and reboot takes a great amount of time and would not help much with our testing purpose.

In this assignment, we use kernel module to test the logic of the system call first before implementing it and adding it to the kernel.

2.3.2 Implement a kernel module

This is the source code of my kernel module:

3 Questions

1. Why do we need to install `kernel-package`?

   Answer: The `kernel-package` package includes dependencies required to build the kernel. More information can be found here.

2. Why do we have to use another kernel source from the server such as http://www.kernel.org, can we just compile the original kernel (the local kernel on the running OS) directly?

   Instead of downloading and compiling a new kernel, we can compile the original kernel on our machine directly. However, there are certain advantages of building a new kernel version:

   - A new kernel version often comes with bug fixes for previous kernel versions.
   - A new kernel version is often more stable and faster than the the previous versions.
   - A new kernel is better-compatible with certain hardware, such as GPU, wifi card, etc..
   - A new kernel version often offers new features and functionalities.

3. What is the meaning of each line above?

4. What is the meaning of other components, i.e. i386, procmem, and `sys_procmem`?

5. What is the meaning of these two stages, namely "make" and "make modules"?

   - `make`: compiles and links the kernel image. This is a single file named `vmlinuz` (or `bzImage`, not sure for now).
   - `make modules`: compiles individual files for each question you answered `M` during kernel config. The object code is linked against your freshly built kernel. (For questions answered `Y`, these are already part of `vmlinuz`, and for questions answered `N` they are skipped).

6. Why could this program indicate whether our system call works or not?

7. Why do we have to re-define `proc_segs` struct while we have already defined it inside the kernel?

8. Why is root privilege (e.g. adding sudo before the cp command) required to copy the header fle to `/usr/include`?

   Answer: In Linux, a normal user only has the write privilege inside his 'home' directory. To have write privilege to directories outside `home`, one needs `sudo` to run the command with the superuser's privilege. Because `/usr/include` is a directory outside of `home`, we need to use `sudo`.

9. Why must we put `-shared` and `-fpic` options into the gcc command?

   Answer: The `-shared` and `-fPIC` (or `-fpic`) flags are required to build a shared object for dynamic linking. More precisely, according to here:

   - `-shared`: Produce a shared object which can then be linked with other objects to form an executable. Not all systems support this option. You must also specify `-fpic` or `-fPIC` on some systems when you specify this option.
   - `-fPIC`: Emit position-independent code, suitable for dynamic linking and avoiding any limit on the size of the global offset table.

References

[1] "Linux Programmer's Manual - SYSCALLs(2)." http://man7.org/linux/man-pages/man2/syscalls.2.html.

[2] "The Linux Kernel Module Programming Guide." https://linux.die.net/lkmpg/x40.html.