Assignment 3: Linux System Calls

Introduction

The goal of this project is to get used to Linux development environment and modify Linux kernel source code. The project is split into two parts. The first part is analyzing a given user-space socket application and adding `printk()` to the system call entries in Linux kernel. The second part is to add one new system call which encrypts a given string and prints out encrypted string to kernel message.

Useful links:

- socket programming in C
- GNU make
- printk
- git diff
- GETOPT

Part 1. Adding printk() (~80 minutes)

The attached code hw3.tar.gz is a simple socket application that exchanges messages between a server and client. It consists of three files as shown in below.

```
$ cd code-socket
$ ls

client.c  Makefile  server.c
```

P1.1: Understanding source code

[24 points] Read above three source file to understand how a simple network client/server works. Once you completely understand each file, add brief comments of each line (M1-M10, S1-S8, and C1-C6) explaining what the line means. **Turn in a gzip-ed tarball**.

```
$  tar czvf hw3.tar.gz code-socket/
code-socket/
code-socket/client.c
code-socket/server.c

code-socket/Makefile
```

P1.2: Adding printk()

[6 points] Now you understand how the user-space application works. To understand how a kernel system call is called, print any message at the very beginning of the system call implementation of `accept()` in `Linux kernel v6.8`. Check if the modified kernel prints out messages you added when you run the network client/server. Then create a patch against `kernel v6.8` using git diff command and **turn in the patch** named `hw3_firstname_printk.patch`.

[2 points] Test your kernel and upload a screenshot of your kernel debug message using `dmesg` while running the network client/server. Run server, client and dmesg in **one tmux session** and **turn in a screenshot** named `hw3_firstname_printk.jpg | png`.

Part 2: Adding a new system call (~90 minutes)

Add a new super simple system call, named `sys_s2_encrypt()`, that takes two arguments, a NULL-terminated string and an encryption key, which is a positive integer between 1 and 5. The system call encrypts the given string by simply adding the given integer number (i.e., encryption key) to each character of the string. For example, it encrypts "hello" to "ifmmp" by adding 1 to each character of the string. After encryption, it prints out the encrypted string using `printk()`. The system call returns 0 when everything is okay. If the encryption key is out of bounds, it returns `-EINVAL`.

P2.1: Implementing the system call

[9 points] You should implement the system call in a separate file under the `kernel/` directory. Then create a patch against kernel `v6.8` using git diff command and **turn in the patch** named `hw3_firstname_syscall.patch`.

P2.2: Writing a test program

[7 points] Write a simple test program takes two options, `-s string -k key` and calls `sys_s2_encrypt()` with the string and key from the command line. You should implement command line argument processing using `GETOPT(3)` and the code should be able to build using make and clean using make clean. The program prints out the return value of `sys_s2_encrypt()`. **Turn in the gzip-ed tarball of the source code and Makefile**.

P2.3: Test your system call

[2 points] Upload a screenshot of your kernel debug message using dmesg while running your test program. Run your test program and dmesg in **one tmux session** and **turn in a screenshot** named `hw3_firstname_syscall.jpg | png`.