ITP 30002-01 Operating System, Spring 2020

**Homework 1**

21600212 Nam Jinwoo | 21600212@handong.edu | Demo video < https://youtu.be/4Fgo4jWLHbE >

## 1. Introduction

The problem is building Loadable Kernel Module (LKM) that can be controlled by user through user level application. In other words, user level application is also needed to be built. To develop those two, I made 'mousehole.ko' and 'jerry.c' codes. There are two specific functions that I want to address and provide through this LKM.1) Blocking a certain user from opening specified files. 2) Preventing a killing of processes created by a specific user.

Commands to control will be transferred from user level application 'jerry.c' to LKM 'mousehole.ko'. In order to create communicable structure, a proc filesystem must be made.

In this paper, I propose my approaches to tackle above goals.

## 2. Approach

***Logic Description***

Function 1. **Block Opening**

Condition 1)

*if* Input User = Current User

Condition 2)

*if* Input Filename = Opening Filename

Function 2. **Prevent Killing**

Condition 1)

*if* Input User = User of Process (selected to kill)

<Figure 1. Logic description>

*Figure 1.* describes the expected logic of functions. Each functions require user input parameters. User interface is essential to obtain input parameters.

### User Interface (jerry.c)

In order to obtain appropriate parameters, a detailed explanation is needed. I designed for users to choose between two main functions and three primary functions.

```
int opt;
printf("---This program provides five options---\n");
printf("Option 1 - Block Certain User\n");
printf("Option 2 - Prevent a Killing\n");
printf("Option 3 - Stop Option 1 Blocking\n");
printf("Option 4 - Stop Option 2 Preventing\n");
printf("Option 5 - Display Information from Kurnel\n");
printf("Select ( 1 / 2 / 3 / 4 / 5 ) : ");
scanf("%d", &opt);
```

<Figure 2.. Function Options>

As mentioned, option 1) and 2) are main functions. The main goals of this program. Option 3) and 4) exists to stop 1) and 2) functions. Option 5) displays information from kernel module.

After user choose function, I need to obtain parameters. As we can check in *figure 1*, function 1) needs two parameters and function 2) needs one parameter.

```
char filepath[100];
char name[50];
printf("Enter username and filename to Block Opening : ");
scanf("%s %s", name, filepath);
```

```
char name[50];

printf("Enter user name to prevent from process killing : ");
scanf("%s", name);
```

<Figure 3,4. Scan function 1), 2) parameters>

After getting all parameters needed, now those should be transferred to kernel module. But user level application cannot communicate with kernel level module directly. This is why the proc file system must be made.

### User-Kernel Communication (proc)

In general, User and Kernel cannot communicate. They can communicate through proc filesystem. To use proc filesystem, I need to create proc when kernel module is initializing.

```
proc_create("mousehole", S_IRUGO | S_IWUGO, NULL, &myops);
```

<Figure 5. Proc_create coded on mousehole.c>

In order to transfer user input parameters, I need to write input parameters on proc file. Though prof file is not a real file, it is possible to read and write.

```
fprintf(fd, "1 %d %s", uid, filepath);
printf("User %s(%d) is Blocked from Opening %s\n",name,uid,filepath);
```

<Figure 6. Fprintf to proc>

### Username to user id

Since "jerry.c" is user level application, I assume that users will put (string) user name, not a (int) user id.

To ease handling parameter, I transform user name to user

id. And then send user id to LKM through proc. (*figure 7*)

```
int getUid(char *name)
{
        struct passwd *p;
        if((p = getpwnam(name)) == NULL){
                perror(name);
                printf("Error: There is no user name '%s'\n",name);
                return -1;
        }
        return (int)p->pw_uid;
}
```

*<Figure 7. Uname to uid>*

**Loadable Kernel Module (mousehole.ko)**

Since "jerry.c" open and write "/proc/mousehole", mousehole_proc_write() function automatically called. Values passed from user by proc file goes to buffer. Then allocate them to appropriate variables. At this time, buf[0] means which function is selected. So if buf[0] is '1', it means Blocking function is selected, and I need to take two arguments. Otherwise if buf[0] is '2', it means Prevent Killing function is selected, and I need to take one argument.

```
if(buf[0]=='1')
        sscanf(buf,"%d %d %s",&opt, &userid_b, filepath);
else if(buf[0]=='2')
        sscanf(buf,"%d %d", &opt, &userid_p);
```

In other to build two main functions, rewriting sys_open and sys_kill is needed. System call modifying requires to access to system call table.

```
sctable = (void *) kallsyms_lookup_name("sys_call_table") ;

orig_sys_open = sctable[__NR_open] ;
orig_sys_kill = sctable[__NR_kill];
```

First, we should borrow original system call function index. Now then every opening, killing action go through my modified function (mousehole_sys_open, kill).

When opening(or killing) action occurred, modified function called to check whether current status satisfy figure 1. Condition or not.

```
if(filepath[0] != 0x0 && strstr(fname, filepath) != NULL){
        if(curr_id == userid_b){
                printk("mousehole: Fail to open file\n");
                return -1;
```

```
for_each_process(t){
        puid = t->real_cred->uid.val;
        if(pid == t->pid && puid == userid_p){
                printk("mousehole: Killing process '%d' is prevented\n",pid);
                return -1;
```

*<Figure 8 ~ 11. LKM functions>*

If passed arguments satisfy condition, file opening blocked or process killing prevented.

## 3. Evaluation

As I mentioned, this program is designed to provide five options. (*figure 2.)*

**Function 1** – Block opening

```
Enter username and filename to Block Opening : jinwoo rry
User jinwoo(1000) is Blocked from Opening rry
jinwoo@jinwoo-VirtualBox:~/hw01/lkm$ vim jerry.c
```

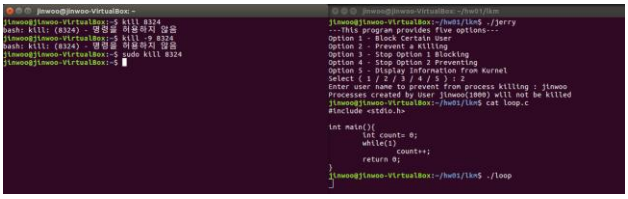
```
"jerry.c" [허용 안 됩니다]
```

vim jerry.c is blocked because 'jerry.c' include 'rry'.

**Function 2** – Prevent Killing

In order to evaluate function 2, I made loop.c which is infinite loop. And I also used two terminals.

I can check process id by 'ps -a'. loop pid was 8324.



I could see that any kind of kill command could not kill created by user jinwoo.

**Function 5** – Display Information from kernel.

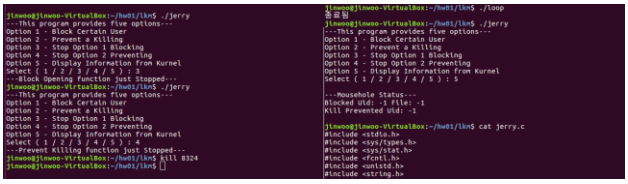
```
Select ( 1 / 2 / 3 / 4 / 5 ) : 5

---Mousehole Status---
Blocked Uid: 1000 File: rry
Kill Prevented Uid: 1000
```

This function display the mousehole variable values.

Now uid 1000(jinwoo) is blocked from opening file that contains 'rry'. And uid 1000(jinwoo) created process will not be killed by other processes.

**Function 3, 4** – Stop function 1 and 2



I could check those stop functions works well.

Kill 8324 and cat jerry.c works now.

## 4. Discussion

In order to complete this project, I needed to learn and search lots of things. Nevertheless, I'm not sure that I fully understood the codes that I used. However, it was quite fun, even though it takes time a lot and some codes were not understandable. Now finally I can handle kernel level codes and that is big change to me. Through the coding time, I felt hacking would be fun. If time allows, I want to try enhanced kernel level coding myself.

## 5. Conclusion

This assignment can be divided into three parts. 1) User Interface, 2) LKM and 3) proc file system. It is a structure in which two completely separated areas can send and receive information through the proc file system. To develop this structure, I needed to understand not only proc file system, but also system call handler, file w/r, system APIs, etc. In other words I could learn all of them at the same time. Moreover, those tasks didn't work separately. Combining them and adjusting them is very important part

to build fully working program. It seems possible to make a hacking program if enhance these knowledges.