운영체제

Assignment #2

Class : A

Professor : 김태석 교수님

Student ID : 2022202065

Name : 박나림

Introduction

이번 과제에서는 'os_ftrace' 시스템 콜을 생성해보고, 해당 시스템 콜을 Wrapping, hooking 을 하는 모듈을 만들어 본다. 이 과정에서 hijack 하는 방법을 익힌 뒤, 최종적으로 os_ftrace 가 특정 pid 에 대하여 파일에 관한 시스템 콜을 추적하는 것으로 만드는 것이 목표이다. 이때 추적할 시스템 콜은 openat, read, write, lseek, close 이다. 해당 과정에서 모듈간에 의존성이 있는 프로그래밍을 하기 위해 EXPORT_SYMBOL 을 사용하고 모듈 파일이 동시에 생성되도록 Makefile 을 작성하도록 한다.

결과화면

Assignment 2-1

커널 소스 폴더에 os_ftrace 폴더를 만든다.

```
root@ubuntu:/usr/src/linux-5.4.282# mkdir os_ftrace
```

System Call 테이블 등록

System call 테이블을 아래 경로에서 찾고, vi 로 들어간다.

```
root@ubuntu:/usr/src/linux-5.4.282/arch/x86/entry/syscalls# ls
Makefile  syscall_32.tbl  syscall_64.tbl  syscallhdr.sh  syscalltbl.sh
```

System call table number: 336 번으로 os_ftrace 를 등록한다.

```
root@ubuntu: /usr/src/linux-5.4.282/arch/x86/entry/syscalls
336     common  os_ftrace               __x64_sys_os_ftrace
```

Syscalls.h 헤더파일을 아래 경로로 찾아서 맨 마지막 #endif 위에 시스템 콜을 등록한다.

```
root@ubuntu:/usr/src/linux-5.4.282/include/linux# vi syscalls.h
```

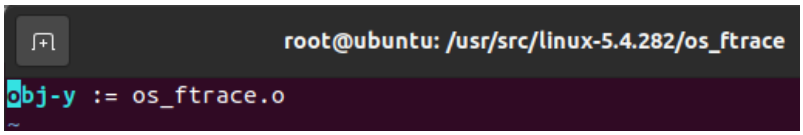
```
asmlinkage int sys_os_ftrace(pid_t pid);
#endif
```

Systeml Call 함수 구현

os_ftrace 폴더에 os_ftrace.c 시스템 콜 함수를 작성한다.

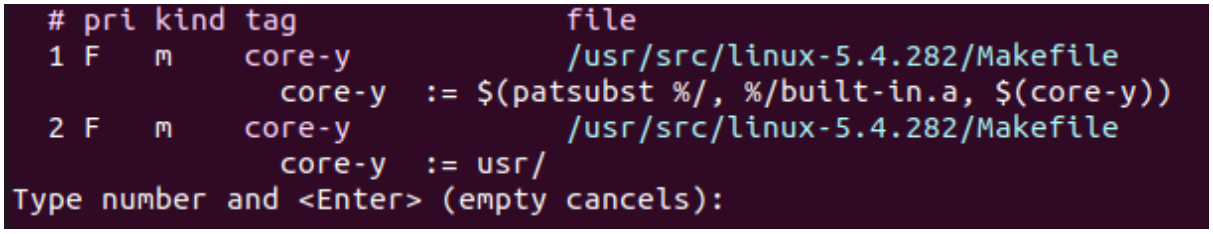
```
root@ubuntu: /usr/src/linux-5.4.282/os_ftrace
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/pid.h>

SYSCALL_DEFINE1(os_ftrace, pid_t, pid)
{
        printk("ORIGINAL ftrace() called! PID is [%d]\n", pid);
        return 0;
}
~
```

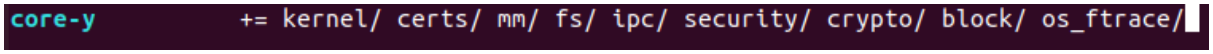
같은 폴더 내에 Makefile 을 만든다.



그리고 그 전 상위 폴더에 존재하는 Makefile 에서

":tj core-y"로 패턴 검색 후 1 번째 결과에서 조금 위쪽 라인에 있는 core-y += kernel/...로 시작하는 라인에 os_ftrace 디렉토리를 추가한다.



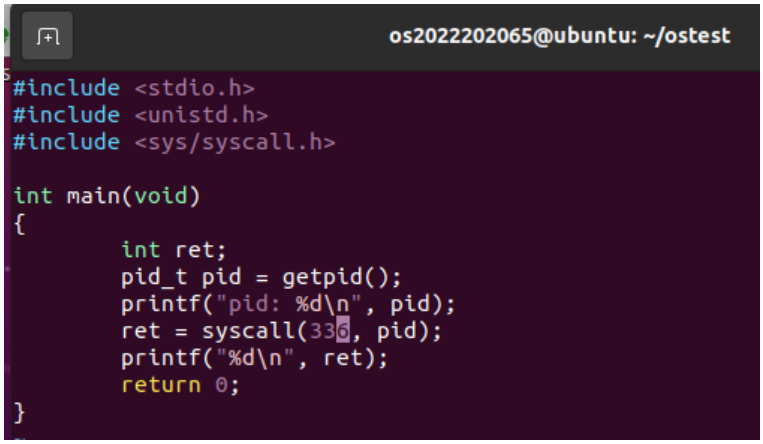


커널 컴파일 및 재부팅

sudo make -> sudo make modules_install -> sudo make install 로 커널 재컴파일 후 에러가 없는지 확인한다. 에러가 없으면 sudo reboot 로 재부팅한다.

결과 확인

Test 파일을 작성한 뒤 빌드하여 syscall 의 리턴 값을 확인한다. (-1:에러, 0:성공)



```
os2022202065@ubuntu:~/ostest$ ./a.out
pid: 2021
0
```

dmesg 를 통해 확인해보면 커널 로그에서 pid 값이 잘 전달되어 출력된 것을 확인할 수 있다.

```
os2022202065@ubuntu:~/ostest$ dmesg | tail -n 1
[  410.867315] ORIGINAL ftrace() called! PID is [2021]
```

Assignment 2-2

홈디렉토리에 osModule 디렉토리를 생성하여 os_ftacehooking.c 소스파일과 Makefile 을 작성한다.

```c
#include <linux/module.h>
#include <linux/highmem.h>
#include <linux/kallsyms.h>    // kallsyms_lookup_name()
#include <linux/syscalls.h>    // __SYSCALL_DEFINEx()
#include <asm/syscall_wrapper.h>// __SYSCALL_DEFINEx()


void **syscall_table;

void *real_os_ftrace; // Pointer to store the address of the existing os_ftrace system.


__SYSCALL_DEFINEx(1, my_ftrace, pid_t, pid)
{
        printk("os_ftrace() hooked! os_ftrace -> my_ftrace\n");
        return 0;
}

```

(os_ftracehooking.c 중 hooking 하는 함수 부분)

```
obj-m := os_ftracehooking.o

KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(CURDIR)

all:
        $(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
        $(MAKE) -C $(KDIR) M=$(PWD) clean
```

```
os2022202065@ubuntu:~/osModule$ ls
Makefile  os_ftracehooking.c
```

make 를 통해 모듈을 컴파일 한다.

```
os2022202065@ubuntu:~/osModule$ sudo make -j4
make -C /lib/modules/5.4.282-os2022202065/build M=/home/os2022202065/osModule modules
make[1]: Entering directory '/usr/src/linux-5.4.282'
  CC [M]  /home/os2022202065/osModule/os_ftracehooking.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/os2022202065/osModule/os_ftracehooking.mod.o
  LD [M]  /home/os2022202065/osModule/os_ftracehooking.ko
make[1]: Leaving directory '/usr/src/linux-5.4.282'
```

빌드된 모듈을 insmod 로 커널에 적재한 뒤 lsmod 로 확인해보면 정상적으로 올라간 것을 확인할 수 있다.

```
os2022202065@ubuntu:~/osModule$ sudo insmod os_ftracehooking.ko
os2022202065@ubuntu:~/osModule$ lsmod |grep os_ftracehooking
os_ftracehooking       16384  0
```

2-1 때 만들었던 os_ftrace 테스트 코드를 실행한 뒤 dmesg 로 확인하면 기존 os_ftrace 시스템 콜이 my_ftrace 로 대체된 것을 볼 수 있다.

```
os2022202065@ubuntu:~/ostest$ ./a.out
pid: 88470
0
```

```
os2022202065@ubuntu:~/ostest$ dmesg | tail -n 1
[ 5106.939361] os_ftrace() hooked! os_ftrace -> my_ftrace
```

Assignment 2-3

Assignment2-3 디렉토리에 ftracehooking.c, iotracehooking.c, ftracehooking.h, Makefile, test.c 파일을 작성한다.

ftracehooking.c 는 2-2 에서 작성했던 os_ftracehooking.c 를 수정해서 작성하였다. 기존에 2-2 에서 생성했던 모듈은 rmmod 로 삭제하고, 2-3 에서 새로 my_ftrace 로 생성하여 대체하도록 만든다. 이때 iotracehooking.c 에서 openat, read, write, lseek, close 시스템콜을 hijack 하여 측정한 값들은 printfInfo()로 보낼 것이기 때문에 extern 으로 전달받는다. 또한 측정을 위해 target_pid 값은 my_ftrace 에서 받아서 추적이 시작될 때 setPid()로 보내야 한다. 이때 두 파일에서 EXPORT_SYMBOL()을 사용하면 모듈 적재할 때 에러가 발생하므로 ftracehooking.c 는 extern 으로 받기만 하도록 작성한다.

```c
extern void printInfo(void);
extern void setPid(pid_t pid);


static asmlinkage int my_ftrace(const struct pt_regs *regs) {
    pid_t pid = (pid_t)regs->di;  // first argument

    if (pid > 0) {
        printk(KERN_INFO "OS Assignment 2 ftrace [%d] Start\n", pid);
        target_pid = pid;
        setPid(target_pid);
    } else if (pid == 0) {
        printInfo();
        printk(KERN_INFO "OS Assignment 2 ftrace [%d] End\n", target_pid);
    }

    // return sys_os_ftrace
    return ((asmlinkage int (*)(const struct pt_regs *))real_os_ftrace)(regs);
}
```

Iotracehooking.c 에서는 setPid(), printInfo() 함수를 작성해서 EXPORT_SYMBOL 로 넘긴다. 또한 openat 과 같은 시스템콜들을 hijack 하여 현재 pid 가 setPid 에서 받은 tartget_pid 일때만 count, byte 들을 기록한다.

```c
int open_count;
int read_count;
size_t read_bytes;
int write_count;
size_t write_bytes;
int lseek_count;
int close_count;
pid_t callPid = -1;

asmlinkage long (*real_openat)(const struct pt_regs *);
asmlinkage long (*real_read)(const struct pt_regs *);
asmlinkage long (*real_write)(const struct pt_regs *);
asmlinkage long (*real_lseek)(const struct pt_regs *);
asmlinkage long (*real_close)(const struct pt_regs *);

void setPid(pid_t pid) {
    callPid = pid;
}
EXPORT_SYMBOL(setPid);

static asmlinkage long ftrace_openat(const struct pt_regs *regs) {
    pid_t pid = current->pid;

    if (pid == callPid) {
        open_count++;
    }
    return real_openat(regs);
}
```

```c
void printInfo(void) {
        printk(KERN_INFO "[2022202065] a.out file[abc.txt] stats [x] read - %zu / written - %zu\n", read_bytes, write_bytes);
        printk(KERN_INFO "open[%d], close[%d], read[%d], write[%d], lseek[%d]\n", open_count, close_count, read_count, write_count, lseek_count);
}
EXPORT_SYMBOL(printInfo);
```

ftracehooking.h 는 두 c 파일에서 사용할 수 있도록 작성한다.

```
#ifndef FTRACEHOOKING_H
#define FTRACEHOOKING_H

#include <linux/module.h>
#include <linux/syscalls.h>
#include <linux/kallsyms.h>
#include <linux/types.h>
#include <linux/limits.h>


#endif
```

Makefile 은 ftracehooking.ko 와 iotracehooking.ko 파일이 동시에 생성되도록 작성한다.

```
obj-m += ftracehooking.o
obj-m += iotracehooking.o

PWD = $(CURDIR)

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

test.c 파일에서 os_ftrace 시스템콜을 호출한다. (336) 이후 파일 시스템콜들을 호출하여 abc.txt 파일을 수정한 뒤, 마지막으로 os_ftrace 에 pid 값을 0 으로 설정하여 추적을 종료하도록 작성한다.

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/syscall.h>

int main()
{
        syscall(336, getpid());
        int fd = 0;
        char buf[50];
        fd = open("abc.txt", O_RDWR);
        for (int i=1; i<=4; i++)
        {
                read(fd, buf, 5);
                lseek(fd, 0, SEEK_END);
                write(fd, buf, 5);
                lseek(fd, i*5, SEEK_SET);
        }
        lseek(fd, 0, SEEK_END);
        write(fd, "HELLO", 6);
        close(fd);

        syscall(336, 0);

        return 0;
}
```

작성한 디렉토리에서 Make 한다.

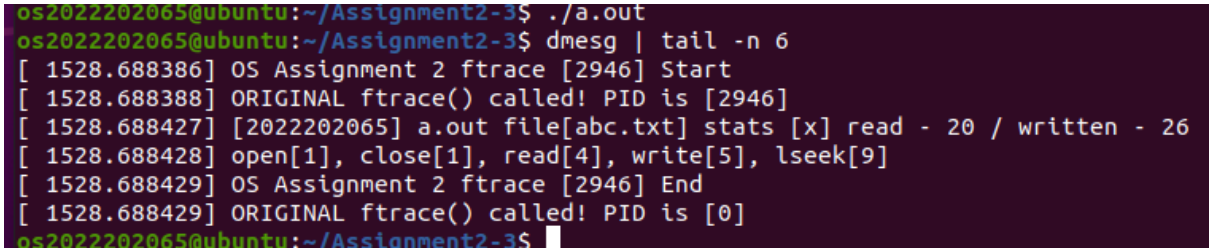
```
os2022202065@ubuntu:~$ ls Assignment2-3/
abc.txt  ftracehooking.c  iotracehooking.c  test.c
a.out    ftracehooking.h  Makefile
```

```
os2022202065@ubuntu:~/Assignment2-3$ make
make -C /lib/modules/5.4.282-os2022202065/build M=/home/os2022202065/Assignment2
-3 modules
make[1]: Entering directory '/usr/src/linux-5.4.282'
  CC [M]  /home/os2022202065/Assignment2-3/ftracehooking.o
  CC [M]  /home/os2022202065/Assignment2-3/iotracehooking.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC [M]  /home/os2022202065/Assignment2-3/ftracehooking.mod.o
  LD [M]  /home/os2022202065/Assignment2-3/ftracehooking.ko
  CC [M]  /home/os2022202065/Assignment2-3/iotracehooking.mod.o
  LD [M]  /home/os2022202065/Assignment2-3/iotracehooking.ko
make[1]: Leaving directory '/usr/src/linux-5.4.282'
```

모듈을 차례대로 적재한다. Iotracehooking.c 에서 SYMBOL 들을 보내기 때문에 iotracehooking.ko 부터 적재해야 한다. 반대로 하면 ftracehooking.c 를 적재할 때 Unknown Symbol 에러가 난다.

```
os2022202065@ubuntu:~/Assignment2-3$ sudo insmod iotracehooking.ko
[sudo] password for os2022202065:
os2022202065@ubuntu:~/Assignment2-3$ sudo insmod ftracehooking.ko
```

테스트 파일을 실행시킨뒤 dmesg 로 확인해보면, 커널 로그로 abc.txt 파일을 추적한 결과가 출력되는 것을 확인할 수 있다.



고찰

2-1)

커널 소스 폴더를 상위 폴더가 아니라 그 하위에 있는 kernel 폴더로 잘못 이해하여, 아래 사진처럼 os_ftrace 폴더를 만들고 진행하였다가 컴파일 때 Makefile 에서 osftrace 폴더를 찾을 수 없다는 에러가 떴었다.

```
root@ubuntu:/usr/src/linux-5.4.282/kernel# mkdir os_ftrace
```

처음엔 Makefile 에서 잘못 작성한 줄 알았으나, 폴더 위치가 잘못되었음을 깨닫고 상위 커널 소스 폴더인 linux-5.4.282 에 바로 os_ftrace 폴더를 만들었더니 해당 에러는 해결할 수 있었다.

Makefile 에서 Ctag 를 통해 "core-y"를 찾을 때, 검색 결과에서 한번에 안 떠서 잘못된 곳에다 작성하였었다. 아래처럼 Makefile 25~35% 라인에 위치한 core-y 에 썼다가 컴파일 때 에러가 났다. 다시 수정하려 해도 해당 위치는 검색을 할 수 없다는 에러가 추가적으로 떴었다.

```
core-y          := usr/ os_ftrace/
```

그래서 Makefile 을 직접 검사하여 해당 부분을 수정하다가 첫번째 검색 결과에서 몇 라인 위쪽에 찾고자하는 core-y 가 위치한 것을 발견하였다. 그 부분에 os_ftrace/를 추가하니 정상적으로 컴파일 될 수 있었다.

위 오류들을 해결한 뒤 다시 컴파일 했으나, main()에서 테스트 코드를 실행할 때 시스템 콜에서 -1 이 리턴되었다. [syscall failed: Function not implemented]라는 오류여서 테이블을 잘못 등록한 줄 알고 코드를 여러 번 수정하였으나 같은 오류가 발생하였다. 이에 커널 설정이 잘못된 것 같아 make menuconfig 를 통해 실습 자료에 나와있는대로 다시 환경설정을 맞추고 컴파일 하니 정상적으로 0 이 리턴되며 dmesg 도 확인할 수 있었다.

2-3

커널 모듈을 적재하는 과정에서, 처음에 바로 make 로 컴파일 후 insmod 로 적재를 시도하였으나 Unknown SYMBOL 에러가 발생하였다. ftracehooking.c 와 iotracehookin.g 두 파일에서 서로 EXPORT_SYMBOL()을 사용할 때 발생하는 문제였다. 두 모듈 간에 의존성이 있는 경우에는 첫번째 모듈을 적재할 때 그 중에 모르는 코드가 있으면 안되는데, 서로 EXPORT 를 하니 extern 을 하는 과정에서 알 수 없는 심볼을 받았다고

떴던 것이다. 따라서 한쪽에서만 보내는 방향으로 수정하였다. 추적 결과를 printInfo()로 iotracehooking.c -> ftracehooking.c 방향으로 전달하는 것으로 작성하였는데, 그렇게 하니 ftrace 에서 받은 target_pid 값을 전달하는 방법에 여러 시행 착오가 있었다. 파일 시스템콜 후킹 함수에서 리턴 값을 다르게 하거나 static 전역변수로 수정하는 등의 일을 하다가 iotracehooking.ko 모듈을 적재할 때 커널이 터지는 일이 여러 번 있었다. 이는 전역 변수 문제인 것 같다. 그래서 EXPORT_SYMBOL()로 setPid()함수를 인자를 통해 전달하는 것으로 구현하였더니 정상적으로 작동할 수 있었다. 해당 과제를 통해 모듈간에 의존성이 있는 모듈 프로그래밍 방법을 더 이해할 수 있었다.

Reference

강의자료 참고