

# 운영체제

Assignment2

한준호

2018741035

최상호 교수님

# Assignment 2

## - Introduction

PID의 인자를 받는 'ftrace'라는 새로운 System Call을 만들어보았고 새로 만든 시스템콜과 기존에 있던 시스템콜인 open, close, read, write, close 시스템콜의 원형을 찾고 원형의 시스템콜을 새로운 커널 모듈로 대체하는 System Call Wrapping하기 위해 Module Programming 했다. 테스트용 프로그램과 작성한 커널 모듈을 추가 또는 제거 해보면서 Trace를 시작하고 종료할 때 작성한 커널 메시지를 확인했다.

## - Conclusion & Analysis

### 1. 새로운 시스템콜 'ftrace' 만들기 (시스템 콜 번호: 336)

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ vi arch/x86/entry/syscalls/syscall_64.tbl
548      common   hello           __x64_sys_hello
549      common   add             __x64_sys_add
336      common   ftrace         __x64_sys_ftrace
                                         391,38-56      Bot
```

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ vi include/linux/syscalls.h
```

```
asmlinkage long sys_hello(void);
asmlinkage long sys_add(int, int);
asmlinkage int ftrace(pid_t);

#endif
```

새로운 시스템콜인 'ftrace'를 System Call 테이블에 등록하였다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ vi ftrace/ftrace.c
```

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <linux/module.h>
#include <linux/sched.h>
#include <linux/init_task.h>

SYSCALL_DEFINE1(ftrace, pid_t, pid)
{
    return pid;
}

```
ftrace.c" 11L, 189C           1,1          All

```

'ftrace.c', System Call 함수를 구현해주었다. 이 때 PID의 인자를 하나만 받으므로 SYSCALL\_DEFINE1로 정의하였다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ vi ftrace/Makefile
```

```
obj-y = ftrace.o
```

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ vi Makefile
```

```
core-y      += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ hello/ a
dd/ ftrace/
```

'ftrace.c'을 커널 컴파일 하기 위해 Makefile을 작성하고 수정해주었다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ sudo make -j6
```

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ sudo make modules_install
```

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ sudo make install
```

커널 재 컴파일을 해주었고 새로운 System Call인 'ftrace'을 만들었다.

## 2. 새로 만든 ftrace 시스템콜을 hijack 하여 ftrace 함수로 대체하기

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ cd ftrace
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ vi ftracehooking.h

#include <linux/kernel.h>
#include <linux/syscalls.h> /* __SYSCALL_DFINRx() */
#include <linux/module.h>
#include <linux/highmem.h>
#include <linux/kallsyms.h> /* kallsyms_lookup_name() */
#include <asm/syscall_wrapper.h> /* __SYSCALL_DFIRx() */

#include <asm/uaccess.h>
#include <linux/sched.h>
#include <linux/init_task.h>
~
~
~
~
~
~
~
~
~
ftracehooking.h" 12L, 329C
```

앞으로 작성할 모듈 소스 프로그램인 'ftracehooking.c' 및 'iotracehooking.c'에서 사용하는 header를 정의해주었다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ vi ftracehooking.c
```

```
#include "ftracehooking.h"

#define __NR_ftrace 336

void **syscall_table;

unsigned int pid_n = 0;

asmlinkage int(*real_ftrace)(pid_t);
asmlinkage int my_ftrace(pid_t pid)
{
    return (real_ftrace(pid));           I
}

void make_rw(void *addr)
{
    unsigned int level;
    pte_t *pte = lookup_address((u64)addr, &level);
    if(pte->pte &~ _PAGE_RW)
        pte->pte |= _PAGE_RW;
}

void make_ro(void *addr)
{
    unsigned int level;
    pte_t *pte = lookup_address((u64)addr, &level);
    pte->pte = pte->pte &~ _PAGE_RW;
}
```

```

static int __init ftracehooking_init(void)
{
    struct task_struct *task = current;
    pid_n = task->pid;
    printk(KERN_INFO "OS Assignment 2 ftrace [%d] Start\n", pid_n);

    /* Find system call table */
    syscall_table = (void**) kallsyms_lookup_name("sys_call_table");

    /*
     * change permission of the page of system call table
     * to both readable and writable
     */
    make_rw(syscall_table);
    real_ftrace = syscall_table[__NR_ftrace];
    syscall_table[__NR_ftrace] = my_ftrace;

    return 0;
}

static void __exit ftracehooking_exit(void)
{
    printk(KERN_INFO "OS Assignment 2 ftrace [%d] End\n", pid_n);

    syscall_table[__NR_ftrace] = real_ftrace;

    /* Recover the page's permission (i.e. read-only) */
    make_ro(syscall_table);
}

module_init(ftracehooking_init);
module_exit(ftracehooking_exit);
MODULE_LICENSE("GPL");

EXPORT_SYMBOL(make_rw);
EXPORT_SYMBOL(make_ro);

```

69,1-8

Bot

앞에서 만든 ftrace 시스템콜을 hijack 하여 'my\_ftrace' 함수로 대체하기 위해 'ftracehooking.c' 를 작성하였고 'asmlinkage' 를 사용하였다. 모듈이 추가될 때 system call table에 읽기 및 쓰기 권한을 부여하고, 모듈이 제거될 때 읽기 및 쓰기 권한을 회수할 수 있도록 함수를 작성하였다.

모듈이 적재 될 때, Trace의 시작을 알수있도록 task\_struct 구조체를 이용해 현재 프로세스의 pid를 'pid\_n' 변수에 저장하고 커널 메시지로 출력할 수 있게 작성하였다. 모듈 해제 시 기존의 시스템콜을 원상 복구 하기 위해 기존 ftrace 시스템콜 주소를 'real\_ftrace' 에 저장하였고, 후킹할 'my\_ftrace' 시스템콜을 기존의 ftrace 시스템콜 대신 대체하였다.

모듈이 해제 될 때, Trace의 종료를 알수있도록 'pid\_n' 변수에 저장된 프로세스의 pid를 커널 메시지로 출력할 수 있게 작성하였다. 모듈을 적재할 때 저장해뒀던 'real\_ftrace'를 이용에 시스템콜을 원상 복구할 수 있도록 하였다.

'ftracehooking.c' 와 'iotracehooking.c' 에서 공동으로 'make\_rw', 'make\_ro' 함수를 사용하고 'ftracehooking.c' 와 'iotracehooking.c' 를 연동하기 위해 'EXPORT\_SYMBOL'을 이용하였다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ vi arch/x86/entry/syscalls/syscall_64.tbl
```

```
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The __x64_sys_*() stubs are created on-the-fly for sys_*() system calls
#
# The abi is "common", "64" or "x32" for this file.
#
0      common  read           __x64_sys_read
1      common  write          __x64_sys_write
2      common  open            __x64_sys_open
3      common  close           __x64_sys_close
4      common  stat           __x64_sys_newstat
5      common  fstat          __x64_sys_newfstat
6      common  lstat          __x64_sys_newlstat
7      common  poll            __x64_sys_poll
8      common  lseek           __x64_sys_lseek
9      common  mmap           __x64_sys_mmap
10     common  mprotect        __x64_sys_mprotect
11     common  munmap          __x64_sys_munmap
12     common  brk             __x64_sys_brk
"arch/x86/entry/syscalls/syscall_64.tbl" 390L, 15726C           1,1           Top
```

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67$ vi include/linux/syscalls.h
```

```
asm linkage long sys_open(const char __user *filename,
                           int flags, umode_t mode);
```

```
asm linkage long sys_close(unsigned int fd);
```

```
asm linkage long sys_read(unsigned int fd, char __user *buf, size_t count);
```

```
asm linkage long sys_write(unsigned int fd, const char __user *buf,
                           size_t count);
```

```
asm linkage long sys_lseek(unsigned int fd, off_t offset,
                           unsigned int whence);
```

기존의 있던 시스템콜인 open, close, read, write, lseek 을 hijack 하기 위해 open, close, read, write, lseek 시스템 콜의 원형을 찾았다.

System Call 번호: open: 2 / close : 3 / read : 0 / write :1 / lseek : 8

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ vi iotracehooking.c
```

```
#include "ftracehooking.h"

#define __NR_open 2
#define __NR_read 0
#define __NR_write 1
#define __NR_lseek 8
#define __NR_close 3

extern void make_rw(void *addr);
extern void make_ro(void *addr);

void **syscall_table;

unsigned int read_bytes = 0;
unsigned int write_bytes = 0;

unsigned int open_count = 0;
unsigned int close_count = 0;
unsigned int read_count = 0;
unsigned int write_count = 0;
unsigned int lseek_count = 0;

asmlinkage long(*real_open)(const char*, int, umode_t);
asmlinkage long(*real_close)(unsigned int);
asmlinkage long(*real_read)(unsigned int, char*, size_t);
asmlinkage long(*real_write)(unsigned int, const char*, size_t);
asmlinkage long(*real_lseek)(unsigned int, off_t, unsigned int);

asmlinkage long ftrace_open(const char __user *filename, int flags, umode_t mode)
{
    open_count += 1;
    return(real_open(filename, flags, mode));
}
asmlinkage long ftrace_close(unsigned int fd)
{
    close_count += 1;
    return(real_close(fd));
}
asmlinkage long ftrace_read(unsigned int fd, char __user *buf, size_t count)
{
    read_count += 1;
    read_bytes = sizeof(real_read(fd, buf, count));
    return(real_read(fd, buf, count));
}
asmlinkage long ftrace_write(unsigned int fd, const char __user *buf, size_t count)
{
    write_count += 1;
    write_bytes = sizeof(real_write(fd, buf, count));
    return(real_write(fd, buf, count));
}
asmlinkage long ftrace_lseek(unsigned int fd, off_t offset, unsigned int whence)
{
    lseek_count += 1;
    return(real_lseek(fd, offset, whence));
}
```

```

static int __init iotracehooking_init(void)
{
    /* Find system call table */
    syscall_table = (void**) kallsyms_lookup_name("sys_call_table");

    /*
     * change permission of the page of system call table
     * to both readable and writable
     */
    make_rw(syscall_table);
    real_open = syscall_table[__NR_open];
    real_close = syscall_table[__NR_close];
    real_read = syscall_table[__NR_read];
    real_write = syscall_table[__NR_write];
    real_lseek = syscall_table[__NR_lseek];
    syscall_table[__NR_open] = ftrace_open;
    syscall_table[__NR_close] = ftrace_close;
    syscall_table[__NR_read] = ftrace_read;
    syscall_table[__NR_write] = ftrace_write;
    syscall_table[__NR_lseek] = ftrace_lseek;

    return 0;
}

```

```

static void __exit iotracehooking_exit(void)
{
    printk(KERN_INFO "[2018741035] ./test file[abc.txt] stats [x] read - %d / written - %d\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);

    syscall_table[__NR_open] = real_open;
    syscall_table[__NR_close] = real_close;
    syscall_table[__NR_read] = real_read;
    syscall_table[__NR_write] = real_write;
    syscall_table[__NR_lseek] = real_lseek;

    read_bytes = 0;
    write_bytes = 0;

    open_count = 0;                                I
    close_count = 0;
    read_count = 0;
    write_count = 0;
    lseek_count = 0;

    /* Recover the page's permission (i.e. read-only)*/
    make_ro(syscall_table);
}

module_init(iotracehooking_init);
module_exit(iotracehooking_exit);
MODULE_LICENSE("GPL");

```

111,0-1

Bot

앞에서 찾은 시스템콜의 원형을 이용하여 ftrace 시스템콜을 hijack 해줬던 방법과 동일하게 기존의 있던 시스템콜인 open, close, read, write, lseek 을 hijack 해주었다.

'iotracehooking' 모듈이 해제될 때, 과제 조건에 맞게 커널 메시지를 출력하기 위해 hijack 하는 과정에서 횟수를 해당 시스템콜이 사용될 때 count 값이 +1씩 증가하도록 작성하였고, sizeof() 함수를 이용해 read, write 시스템콜의 bytes를 저장해주었다.

```
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;
}
```

28,1

Bot

강의자료실에 첨부된 'abc.txt' 파일 생성 후 open/read/write/lseek/close를 사용하고 trace 종료 시점에 ftrace 시스템콜에 0값을 넣고 호출하는 테스트용 프로그램인 'test.c'를 사용하였다.

'ftracehooking.ko' 파일과 'iotracehooking.ko' 파일이 동시에 생성되도록 Makefile을 작성해주었다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ sudo make
make -C /lib/modules/4.19.67-2018741035/build SUBDIRS=/home/os2018741035/Downloads/linux-4.19.67/ftrace modules
make[1]: Entering directory '/home/os2018741035/Downloads/linux-4.19.67'
  CC [M]  /home/os2018741035/Downloads/linux-4.19.67/ftrace/iotracehooking.o
Building modules, stage 2.
MODPOST 2 modules
  CC      /home/os2018741035/Downloads/linux-4.19.67/ftrace/ftracehooking.mod.o
  LD [M]  /home/os2018741035/Downloads/linux-4.19.67/ftrace/ftracehooking.ko
  CC      /home/os2018741035/Downloads/linux-4.19.67/ftrace/iotracehooking.mod.o
  LD [M]  /home/os2018741035/Downloads/linux-4.19.67/ftrace/iotracehooking.ko
make[1]: Leaving directory '/home/os2018741035/Downloads/linux-4.19.67'
gcc -o test test.c
```

작성한 소스를 컴파일 해주었다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ ls
built-in.a      ftracehooking.mod.c  iotracehooking.c~      modules.builtin
ftrace.c        ftracehooking.mod.o  iotracehooking.ko      modules.order
ftracehooking.c  ftracehooking.o    iotracehooking.mod.c  Module.symvers
ftracehooking.c~ ftracehooking.z~   iotracehooking.mod.o  test
ftracehooking.h  ftrace.o        iotracehooking.o       test.c
ftracehooking.ko iotracehooking.c  Makefile
```

정상적으로 'ftracehooking.o' 파일과 'iotracehooking.ko' 파일이 동시에 생성된 것을 확인하였다.

### 3. 결과 확인하기

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ sudo insmod ftracehooking.ko
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ sudo insmod iotracehooking.ko
```

'ftracehooking' 과 'iotracehooking' 이 연동되어있기 때문에 'ftracehooking' -> 'iotracehooking' 순서로 모듈을 적재해주었다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ ./test
```

open/read/write/lseek/close를 사용하는 'test' 를 실행해주었다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ sudo rmmod iotracehooking.ko
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ sudo rmmod ftracehooking.ko
Killed
```

'ftracehooking' 과 'iotracehooking' 이 연동되어있기 때문에 'iotracehooking' -> 'ftracehooking' 순서로 모듈을 해제해주었다.

```
os2018741035@ubuntu:~/Downloads/linux-4.19.67/ftrace$ dmesg
```

커널 log 메시지를 확인하기 위해 'dmesg'를 입력해주었다.

```
[ 60.321188] OS Assignment 2 ftrace [2032] Start
[ 76.540492] [2018741035] ./test file[abc.txt] stats [x] read - 8 / written - 8

[ 76.540494] open[231] close[254] read[1739] write[1032] lseek[52]
[ 81.292621] OS Assignment 2 ftrace [2032] End
```

결과 커널 log 메시지.

### - 고찰

처음 ftrace 시스템콜을 구성할 때, SYSCALL\_DEFINE1() 함수의 내부를 어떻게 작성해야 할지 몰라서 과제 진행이 어려웠지만, 계속 과제를 진행하면서 모듈 프로그래밍을 통해 ftrace 시스템콜을 hooking 할 것이기 때문에 ftrace 시스템콜 내부는 크게 중요하지 않다는 것을 알 수 있었다.

ftrace 시스템콜을 구성하고 컴파일 할 때, 시스템콜을 잘못 만들어 제대로 컴파일이 되지 않았는지 커널이 부팅이 되지 않는 문제가 있었는데 초기 버전(4.15)의 커널로 부팅하여 시스템콜을 맞게 수정하였고 다시 컴파일하여 해결할 수 있었다.

'ftracehooking.c' 와 'iotracehooking.c' 를 작성하여 기존의 시스템콜을 hijack하는 과정에서 'asmlinkage', 'EXPORT\_SYMBOL' 와 같은 몰랐던 것이 있었지만 검색을 통해 해결할 수 있었다.

open/read/write/lseek/close 시스템콜을 hijack하고 hijack한 해당 시스템콜을 사용될 때, count가 1씩 증가하게 작성하였지만, 예시 결과와 달리 너무 큰 결과값이 출력되었다.

모듈을 적재할 때, 가끔 적재하는 과정에서 멈추는 현상(무한 로딩)이 있었는데 이는 가상머신을 사용하기 때문에 메모리의 과다 사용이 원인일 것 같다는 생각이 들었고 가상머신을 재 부팅하여 해결하였다.

### - Reference

- linux kernel 2.6 모듈 (module) 프로그래밍 (4), <https://blog.naver.com/kkn2988/138441227>
- 리눅스 현재 실행중인 전체 프로세스의 task\_struct의 값 읽어내기, [https://m.blog.naver.com/bmw\\_rad/70176211621](https://m.blog.naver.com/bmw_rad/70176211621)