# 운영체제

담당 교수님: 김태석 교수님

학번: 2016722074

이름: 김영태

실습번호: Assignment2

# Assignment 2

## Introduction

특정 pid에 대해 파일에 관한 시스템 콜을 추적하는 툴을 작성한다. ftrace라는 system call을 새로 작성한다. 그 후 ftracehooking.c 파일을 만들어 ftrace system call을 hooking하는 모듈을 생성한다. Iotracehooking.c 파일을 만들어 read, write, open, close, lseek system call을 hooking하는 모듈을 생성한다. Iotracehooking을 통해 생성된 함수는 파일에대한 접근을 counting하여 ftracehooking에 보낸다.

## Reference

https://stackoverflow.com/questions/59851520/system-call-hooking-example-arguments-are-incorrect

➔ Const struct pt_regs* 에서 filename을 추출하는 방법

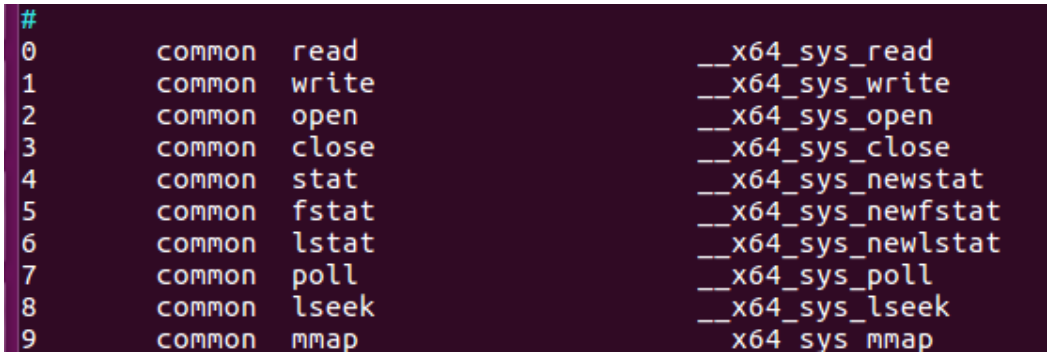
https://docs.huihoo.com/doxygen/linux/kernel/3.7/include_2linux_2string_8h.html

➔ Kernel에서 strcmp()함수를 사용하는 방법

https://stackoverflow.com/questions/19137049/hijack-syscall-access-system-call-arguments-from-struct-pt-regs-64bit-x86

➔ Struct pt_regs*에서 process name과 pid를 추출하는 방법

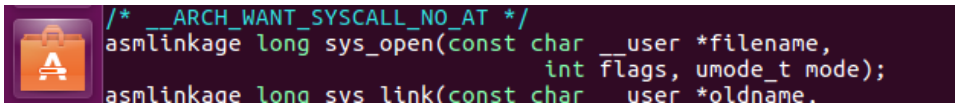
## Conclusion



먼저 커널 디렉터리의 시스템 콜 테이블을 찾아 read, write, open, close, lseek 함수를 찾는다.

```
332     common  statx                   __x64_sys_statx
333     common  io_pgetevents           __x64_sys_io_pgetevents
334     common  rseq                    __x64_sys_rseq
336     common  ftrace                  __x64_sys_ftrace
#
```

그 후 테이블에 ftrace 시스템 콜을 추가한다.

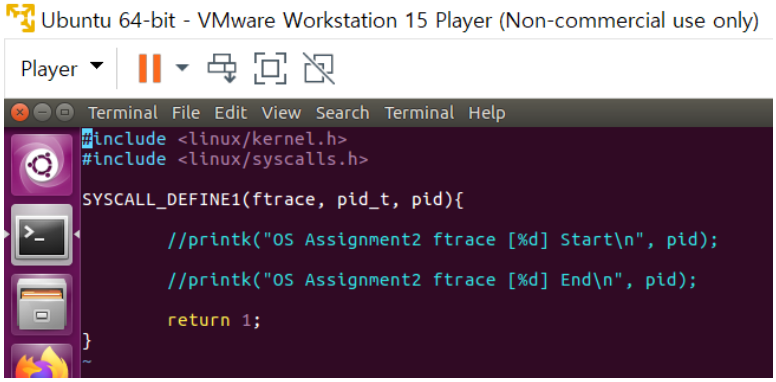


```
/* __ARCH_WANT_SYSCALL_NO_AT */
asmlinkage long sys_open(const char __user *filename,
                         int flags, umode_t mode);
asmlinkage long sys_link(const char __user *oldname,
```

Include/linux/syscalls.h 에서 원본 sys_open함수의 원형을 찾고,

```
asmlinkage long sys_close(unsigned int fd);
asmlinkage long sys_vhangup(void);

/* fs/pipe.c */
asmlinkage long sys_pipe2(int __user *fildes, int flags);

/* fs/quota.c */
asmlinkage long sys_quotactl(unsigned int cmd, const char __user *special,
                             qid_t id, void __user *addr);

/* fs/readdir.c */
asmlinkage long sys_getdents64(unsigned int fd,
                               struct linux_dirent64 __user *dirent,
                               unsigned int count);

/* fs/read_write.c */
asmlinkage long sys_llseek(unsigned int fd, unsigned long offset_high,
                           unsigned long offset_low, loff_t __user *result,
                           unsigned int whence);
asmlinkage long sys_lseek(unsigned int fd, off_t offset,
                          unsigned int whence);
asmlinkage long sys_read(unsigned int fd, char __user *buf, size_t count);
asmlinkage long sys_write(unsigned int fd, const char __user *buf,
                          size_t count);
```

Close, read, write, lseek의 원형도 찾는다.




그 후 커널 폴더안에 ftrace.c파일을 작성한다(이번 관제에서는 hooking될 함수이니 내부는 따로 구현하지 않았다). 원본 ftrace함수는 1을 반환하게 했다.

```
#include <linux/module.h>
#include <linux/highmem.h>
#include <linux/kallsyms.h>
#include <linux/syscalls.h>
#include <asm/syscall_wrapper.h>
#include <linux/module.h>
```

Home 디렉터리로 돌아와 hooking 디렉터리를 새로 생성 후 ftracehooking.h 파일을 작성한다. 이 파일에는 모듈 프로그래밍에 필요한 헤더파일을 작성한다.

```
#include "ftracehooking.h"

#define __NR_ftrace 336

void **syscall_table;

void *real_ftrace;

int ft_opencnt = 0;
int ft_readcnt = 0;
int ft_writecnt = 0;
int ft_lseekcnt = 0;
int ft_closecnt = 0;

EXPORT_SYMBOL(ft_opencnt);
EXPORT_SYMBOL(ft_readcnt);
EXPORT_SYMBOL(ft_writecnt);
EXPORT_SYMBOL(ft_lseekcnt);
EXPORT_SYMBOL(ft_closecnt);
/*
```

그 후 같은 디렉터리에 ftracehooking.c파일을 작성한다. System call table에 있던 원본 ftrace의 번호를 define으로 기억한다. 그 후 system call table과 원본 ftrace의 주소를 저장할 포인터 변수를 전역공간에 선언한다. 그 후 카운팅을 위한 변수를 전역공간에 선언하고 EXPORT_SYMBOL을 사용해 다른 .c 파일에서도 해당 변수를 사용할 수 있게 했다.

```
asmlinkage int (*original_ftrace)(const struct pt_regs*);
asmlinkage int new_ftrace(const struct pt_regs* regs){

        int ret = 0;
        //printk("hook! \n");
        //printk("open cnt : %d\n", ft_opencnt);

        printk("OS Assignment2 ftrace [%d] Start\n", current->pid);
        printk("[2016722074] %s file[abc.txt] stats [x] read - %d / written - %d\n",current->comm ,1 , 1);
        printk("open[%d]  close[%d]  read[%d]  write[%d]  lseek[%d]\n", ft_opencnt, ft_closecnt, ft_readcnt, ft_writecnt, ft_lseekcnt);
        printk("OS Assignment2 ftrace [%d] End\n", current->pid);

        //ret = (*original_ftrace)(regs);

        return ret;
}
```

그 후 원본 ftrace를 저장할 함수포인터와, 새 ftrace를 정의한다. 새 ftrace는 printk함수를 호출해 커널 로그를 출력한다. 이때 전역변수에 선언한 ft_opencnt, ft_closecnt, ft_readcnt, ft_lseekcnt등을 출력한다. 그 후 제대호 hooking이 되었다는 것을 알리기 위해 0을 반환한다.

```
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;
}
```

이어서 ftracehooking.c를 작성한다. System call table의 접근권한을 설정하는 두 함수 make_rw와 make_ro를 작성한다.

```
static int __init hooking_init(void){

        syscall_table = (void**) kallsyms_lookup_name("sys_call_table");

        make_rw(syscall_table);

        //real_ftrace = syscall_table[__NR_ftrace];
        original_ftrace = syscall_table[__NR_ftrace];

        //syscall_table[__NR_ftrace] = __x64_sysftrace;
        syscall_table[__NR_ftrace] = new_ftrace;

        return 0;
}

static void __exit hooking_exit(void){

        //syscall_table[__NR_ftrace] = real_ftrace;
        syscall_table[__NR_ftrace] = original_ftrace;

        make_ro(syscall_table);
}


module_init(hooking_init);
module_exit(hooking_exit);
MODULE_LICENSE("GPL");
```

모듈이 insmod 명령어를 이용해 커널에 로드 되었을 때 호출되는 함수 hooking_init을 작성한다. 이 함수는 먼저 systemcall table의 주소를 저장한다. 그 후 조금전 작성했던 make_rw함수를 호출해 이 시스템 콜 테이블의 접근권한을 바꾼다. 그 후 테이블에 있던 원본 시스템 콜을 전역변수에 저장하고 테이블의 시스템 콜을 새로 작성한 시스템 콜로 대체한다. 이렇게 하면 외부에서 시스템 콜을 호출할 때 새로 작성한 시스템 콜로 hooking이 된다.

이어서 모듈이 rmmod 명령어를 이용해 unload되었을 때 호출되는 함수 hooking_exit을 작성한다. 이 함수는 전역변수에 저장했던 원본 시스템 콜을 시스템 콜 테이블의 알맞은 위치에 저장한다.

```
#include "ftracehooking.h"
#include <uapi/linux/string.h>

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

#define FILENAME "abc.txt"

extern int ft_opencnt;
extern int ft_readcnt;
extern int ft_writecnt;
extern int ft_lseekcnt;
extern int ft_closecnt;

void **syscall_table;
```

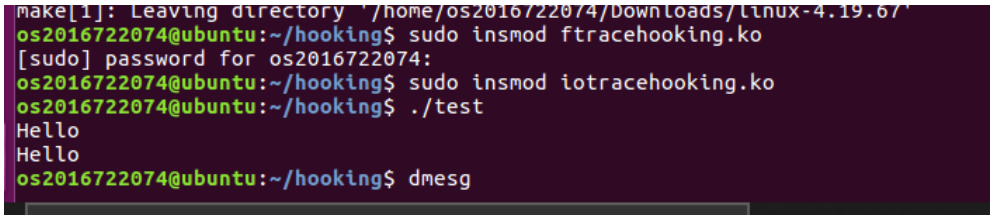
Iotracehooking.c파일을 작성한다. 두 번째 헤더파일은 커널에서 strcmp를 하기 위해 include했다. 그 후 시스템 콜 테이블에 있던 시스템 콜 번호를 define으로 정의한다. filename역시 정의해준다. Ftracehooking.c에서 선언했던 전역변수를 extern 키워드를 사용해 iotracehooking.c에서도 사용할 수 있게 한다.

```
asmlinkage long (*original_read)(const struct pt_regs*);
asmlinkage long ftrace_read(const struct pt_regs* regs){

        char __user *filename = (char*)regs->di;
        char user_filename[256] = {0};
        long copied = strncpy_from_user(user_filename, filename, sizeof(user_filename));

        if(strcmp(user_filename, FILENAME) == 0){
                ft_readcnt++;
        }
        return (*original_read)(regs);
}
```

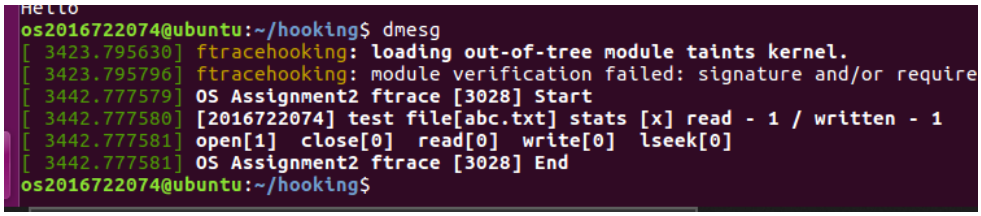
원본 read를 저장할 함수 포인터를 선언하고, 새로 대체할 read를 정의한다. 인자로 받은 레지스터 구조체에서 filename을 추출하고, 카운트를 증가시킨다. 그 후 원본 함수를 반환해 기존의 read가 제대로 동작하도록 한다.

```
asmlinkage long (*original_write)(const struct pt_regs*);
asmlinkage long ftrace_write(const struct pt_regs* regs){

        char __user *filename = (char*)regs->di;
        char user_filename[256] = {0};
        long copied = strncpy_from_user(user_filename, filename, sizeof(user_filename));

        if(strcmp(user_filename, FILENAME) == 0){
                ft_writecnt++;
        }
        return (*original_write)(regs);
}

asmlinkage long (*original_open)(const struct pt_regs*);
asmlinkage long ftrace_open(const struct pt_regs* regs){

        //ft_opencnt++;
        //printk("new_open!!!\n");

        char __user *filename = (char*)regs->di;
        char user_filename[256] = {0};
        long copied = strncpy_from_user(user_filename, filename, sizeof(user_filename));

        if(strcmp(user_filename, FILENAME) == 0){
                ft_opencnt++;
                //printk("abc has open\n");
        }

        return (*original_open)(regs);
}
```

나머지 시스템 콜도 이와 마찬가지로 원본 시스템 콜의 포인터를 저장하고 새로 정의한 시스템 콜에서 원본 카운트를 증가시키고 원본 시스템 콜을 반환했다.




Insmode를 입력하여 모듈을 올리고 실행코드를 실행시켰다.




dmesg명령어를 입력해서 로그를 확인했다.

프로세스 id와 프로세스 이름, 각 시스템 콜의 호출 횟수가 출력되었다.