Operating System

Assignment #1

담당 교수 : 김태석 교수님
수업 일시 : 월2, 수1
학과 : 컴퓨터 공학과
학번 : 2013722095
이름 : 최 재 은

1. Introduction

1-1) 가상머신을 설치하고 그 위에 수업을 위한 Linux환경(Ubuntu)을 설치해 본다. 설치 후 Kernel을 설치하고 컴파일 해 본다.

1-2) dmesg로 kernel message를 확인하고, /init이 실행되는 지점에 본인의 학번이 찍히도록 커널 코드를 수정하여 본다. 커널 코드 내에 printk()를 이용해 수행한다

1-3) 새로운 System Call을 작성해 본다. a와 b를 더하거나 곱하는 system call을 작성하여 추가하고 이를 호출하여 결과를 출력하여 본다.

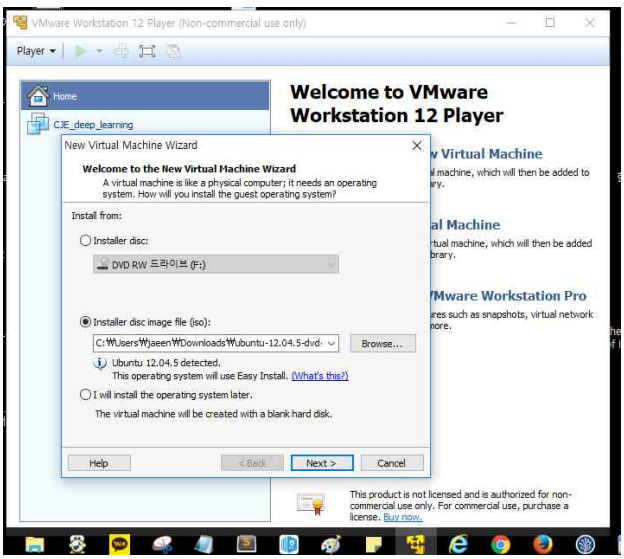
1-4) 모듈을 적재하고 적출하는 System call을 작성하여 보고 결과를 출력해 본다.

2. Reference

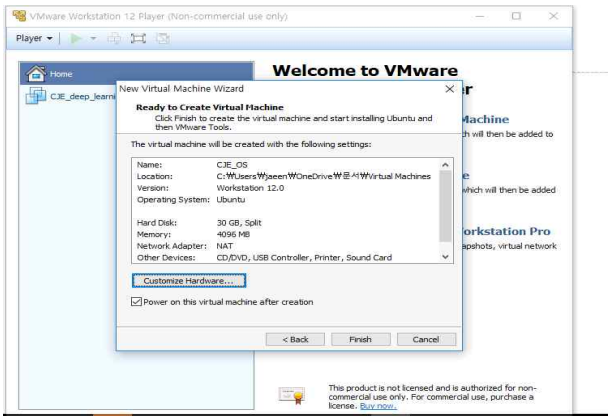
3. Conclusion

- Analysis

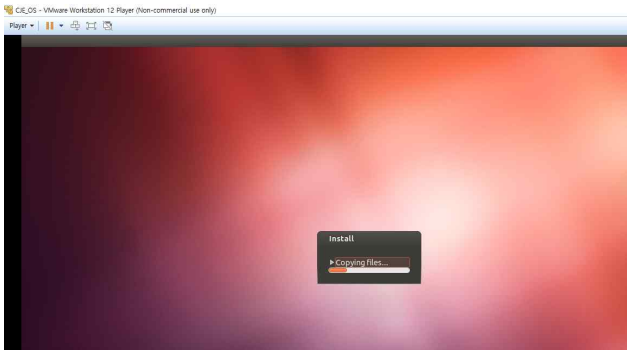
1-1)



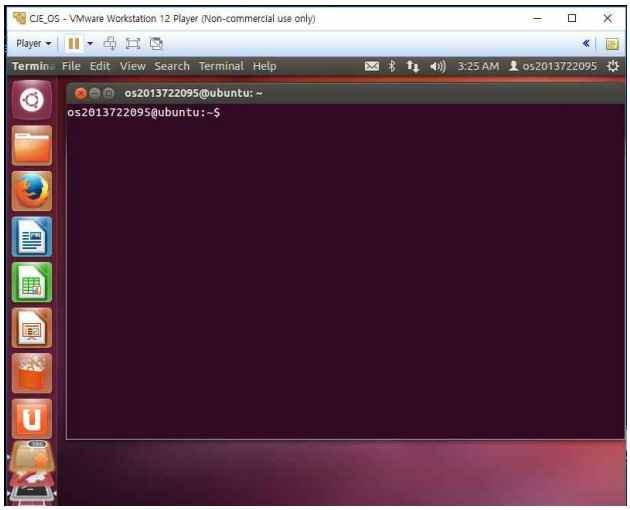
- 가상머신인 VMWARE를 설치하고 다운받은 우분투 디스크 파일을 선택하여 설치를 진행한다.



- 설치할 우분투의 세부 옵션을 셋팅한다. 메모리를 4GB로 상향 조정하였다.

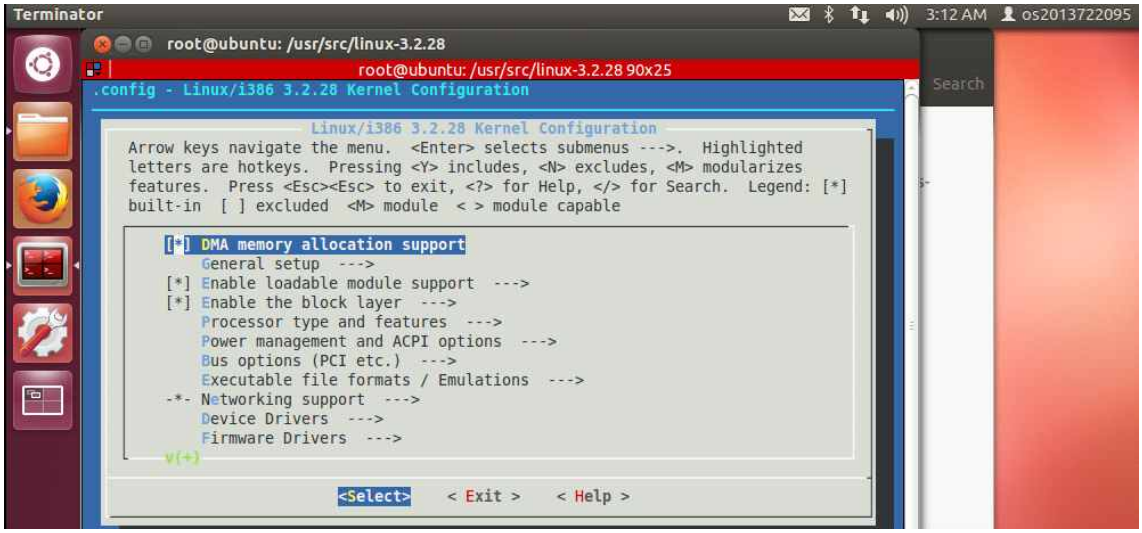


- 왼쪽과 같이 설치 과정이 진행된다.

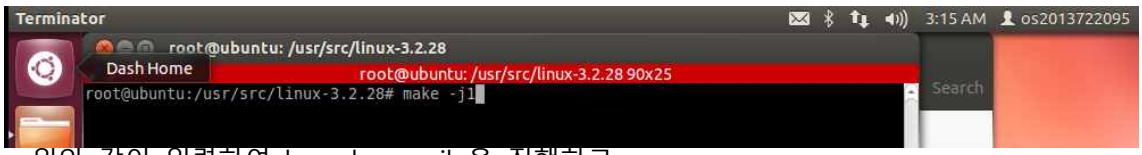


- 설치가 완료되면 왼쪽과 같이 GUI환경의 우분투를 볼 수 있다.

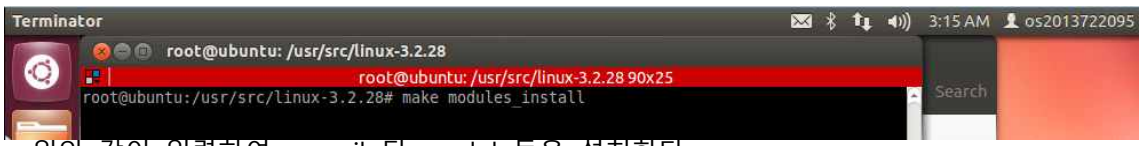
- 위와 같이 커널 파일을 다운 받고 나서 압축을 풀어준다.



- 압축 해제한 경로로 sudo –s 권한으로 접근하여 'make menuconfig'를 입력하여 위와 같은 창을 띄우고 강의 자료에 명시된 대로 옵션을 바꿔준다.

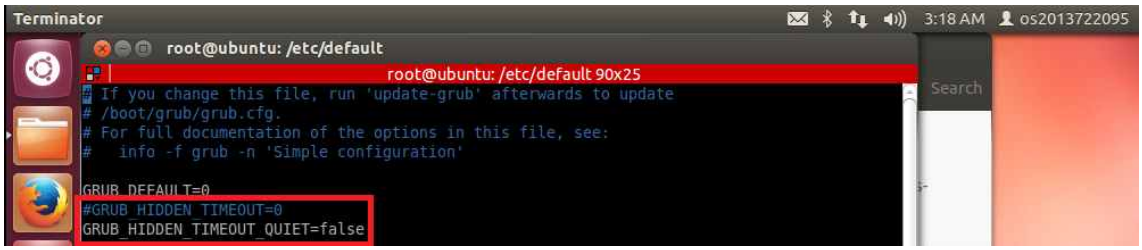


- 위와 같이 입력하여 kernel compile을 진행하고



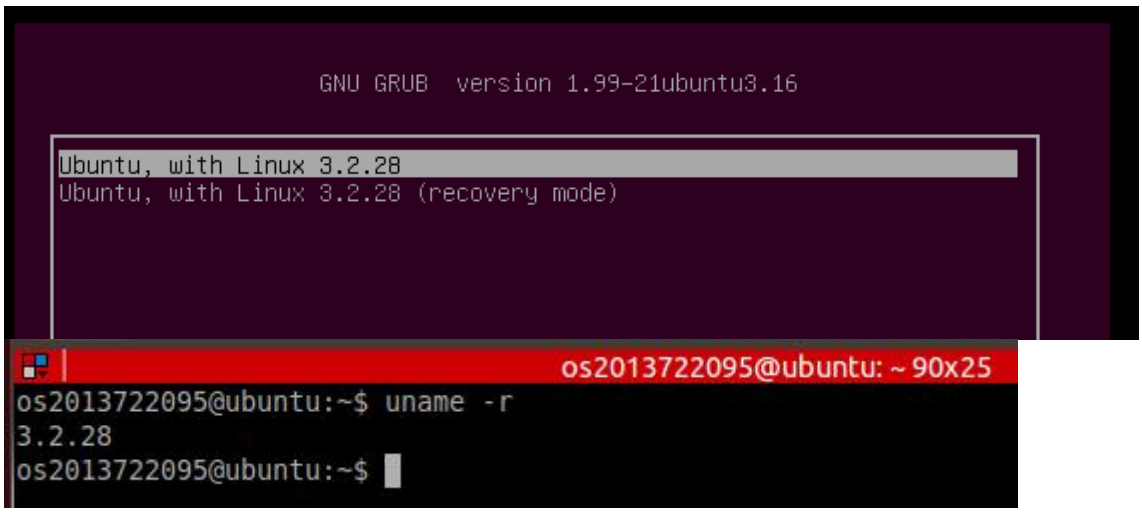
- 위와 같이 입력하여 compile된 module들을 설치한다.

- 위와 같이 입력하여 compile된 kernel을 boot loader에 등록한 뒤 재부팅을 진행



```
root@ubuntu:/etc/default# update-grub
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-3.13.0-32-generic
Found initrd image: /boot/initrd.img-3.13.0-32-generic
```

- 위와 같이 /etc/default 경로의 grub 파일을 수정한 뒤 update하고 재부팅 진행



- 3.2.28 버전의 kernel을 선택하면 커널이 설치, 변경 된 것을 확인할 수 있다.

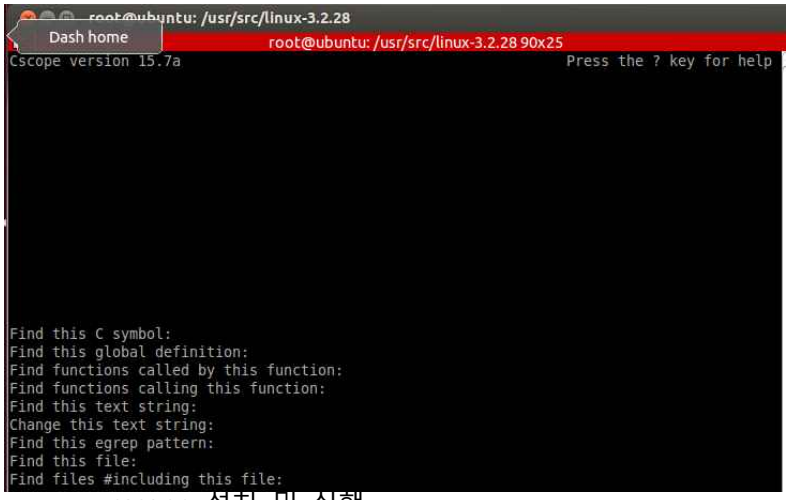
1-2)

```
root@ubuntu:~# apt-get install exuberant-ctags
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  exuberant-ctags
0 upgraded, 1 newly installed, 0 to remove and 376 not upgraded.
Need to get 134 kB of archives.
After this operation, 306 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu/ precise-updates/main exuberant-c
~svn20110310-3ubuntu0.1 [134 kB]
87% [1 exuberant-ctags 116 kB/134 kB 87%]
```

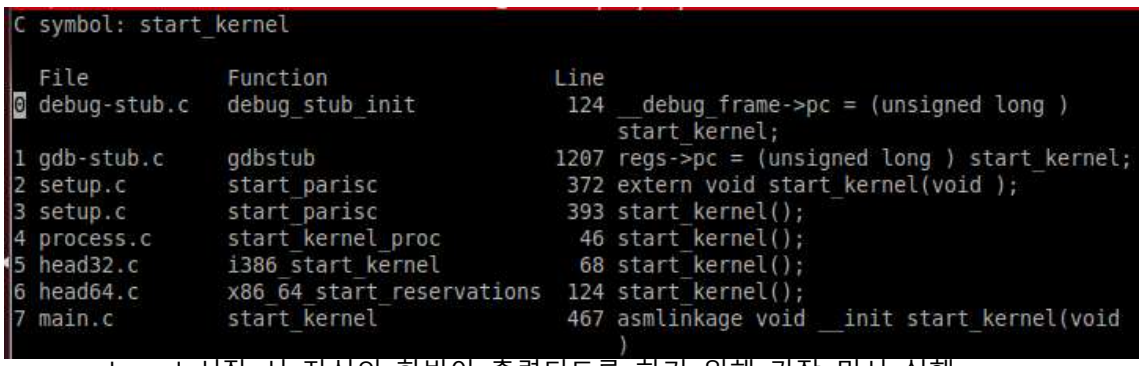
- ctag 설치

```
os2013722095@ubuntu:/usr/src/linux-3.2.28$ sudo ctags -R
^Cos2013722095@ubuntu:/usr/src/linux-3.2.28$ ls
arch            drivers   Kbuild       mm              REPORTING-BUGS  tags
block           firmware  Kconfig      modules.builtin  samples         tools
COPYING         fs        kernel       modules.order   scripts         usr
CREDITS         include   lib          Module.symvers  security        virt
crypto          init      MAINTAINERS  net             sound           vmlinux
Documentation   ipc       Makefile     README          System.map      vmlinux.o
os2013722095@ubuntu:/usr/src/linux-3.2.28$
```

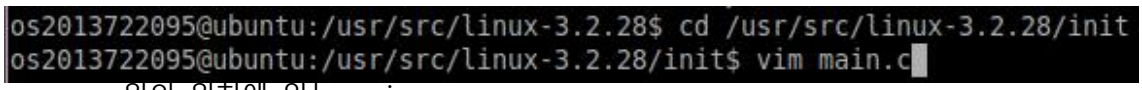
- DB 생성



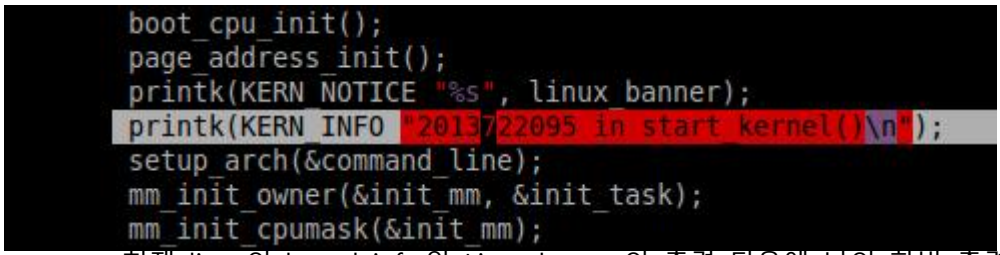
- cscope 설치 및 실행



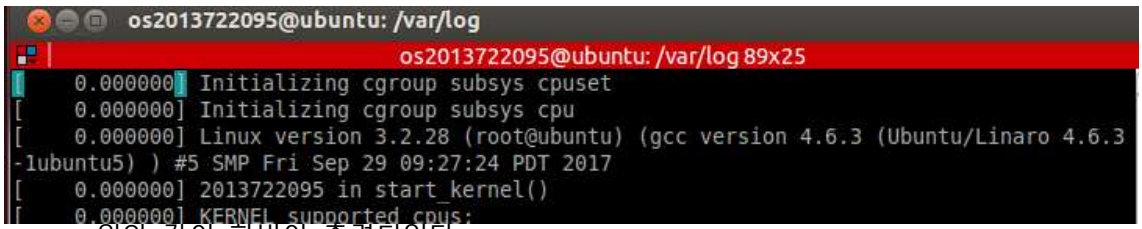
- kernel 시작 시 자신의 학번이 출력되도록 하기 위해 가장 먼저 실행되는 'start_kernel'의 초기화 함수를 찾아들어감



- 위의 위치에 있는 main.c



- 현재 linux의 kernel info.인 Linux_banner의 출력 다음에 나의 학번 출력



-위와 같이 학번이 출력되었다.

1-3)

```
root@ubuntu:/usr/include/i386-linux-gnu/asm# vim unistd_32.h
#define __NR_add                 349
#define __NR_multiple            350
```

- 위의 경로로 이동하여 System call 이름 및 번호 할당

```
root@ubuntu:/usr/src/linux-3.2.28/arch/x86/kernel# vim syscall_table_32.S
        .long sys_add
        .long sys_multiple
:wq
```

- 위의 경로로 이동하여 새로 만든 System call을 테이블에 등록

```
root@ubuntu:/usr/src/linux-3.2.28/kernel#
root@ubuntu: /usr/src/linux-3.2.28/kernel 89x25
#include <linux/kernel.h>

asmlinkage int sys_add(int a, int b)
{
        return a+b;
}
~
~
~
root@ubuntu: /usr/src/linux-3.2.28/kernel 89x25
#include <linux/kernel.h>
asmlinkage int sys_multiple(int a , int b){
        return a * b;
}
```

- 위의 경로로 이동하여 생성한 system call의 함수 구현

```
obj-y     = sched.o fork.o exec_domain.o panic.o printk.o \
            cpu.o exit.o itimer.o time.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \
            signal.o sys.o kmod.o workqueue.o pid.o \
            rcupdate.o extable.o params.o posix-timers.o \
            kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
            hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
            notifier.o ksysfs.o sched_clock.o cred.o \
            async.o range.o 2013722095_add.o 2013722095_multiple.o my_init.o my_exit.o
obj-y += groups.o
```

- 새 system call의 생성될 실행 object 추가 후 kernel 컴파일 진행

```c
#include <stdio.h>
#include <linux/unistd.h>

int main(int argc, char ** argv)
{
        int a = 5;
        int b = 6;
        printf("%d + %d = %d\n", a, b, syscall(__NR_add, a, b));
        printf("%d * %d = %d\n", a, b, syscall(__NR_multiple, a, b));
}
```

add_multiple.c

```
os2013722095@ubuntu:~/os/Mon2Tue1_1_2013722095/1_3$ gcc add_multiple.c
os2013722095@ubuntu:~/os/Mon2Tue1_1_2013722095/1_3$ ./a.out
5 + 6 = 11
5 * 6 = 30
os2013722095@ubuntu:~/os/Mon2Tue1_1_2013722095/1_3$
```

- 위의 코드를 통해 시스템콜을 호출하여 다음과 같은 결과가 도출됨

1-4)

```
#define __NR_my_init            351
#define __NR_my_exit            352
-- INSERT --
```

```
        .long sys_my_init
        .long sys_my_exit
```

os2013722095@ubuntu: /usr/src/linux-3.2.28/kernel 90x25

```c
#include <linux/kernel.h>
asmlinkage void sys_my_init(void)
{
        printk("init [2013722095]\n");
}
```

root@ubuntu: /usr/src/linux-3.2.28/kernel 90x25

```c
#include <linux/kernel.h>

asmlinkage void sys_my_exit(void)
{
        printk("exit [2013722095]\n");
}
```

- 위와 같이 모듈 적재, 적출 시 사용할 system call 작성

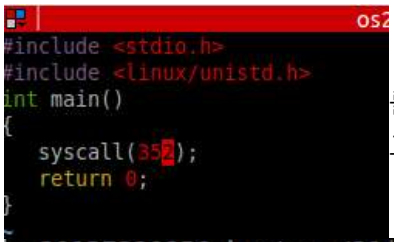
```
obj-y     = sched.o fork.o exec_domain.o panic.o printk.o \
            cpu.o exit.o itimer.o time.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o timer.o user.o \
            signal.o sys.o kmod.o workqueue.o pid.o \
            rcupdate.o extable.o params.o posix-timers.o \
            kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
            hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
            notifier.o ksysfs.o sched_clock.o cred.o \
            async.o range.o add.o multiple.o my_init.o my_exit.o
```

- Makefile에 대상 추가

```c
int hooking_start(void)
{
        make_rw(syscall_table);
        backup = syscall_table[__NR_my_exit];
        syscall_table[__NR_my_exit] = syscall_table[__NR_my_init];
        return 0;
}

void hooking_exit(void)
{
   syscall_table[__NR_my_exit] = backup;
   make_ro(syscall_table);
```

- 강의자료의 module hooking 부분의 코드를 그대로 인용하였고 적재, 적출 시 사용 함수에 대해서만 위와 같이 hooking을 사용하도록 수정



- 왼쪽과 같이 my_exit syscall을 호출하도록 main script를 작성하고 아래와 같이 컴파일을 진행하면 모듈이 적재되고 적출되는 것을 확인할 수 있다

```
os2013722095@ubuntu:~/21$ make run
make -C /lib/modules/3.2.28/build SUBDIRS=/home/os2013722095/21 modules
make[1]: Entering directory `/usr/src/linux-3.2.28'
  CC [M]  /home/os2013722095/21/hooking.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/os2013722095/21/hooking.mod.o
  LD [M]  /home/os2013722095/21/hooking.ko
make[1]: Leaving directory `/usr/src/linux-3.2.28'
cc    -c -o hooking_test.o hooking_test.c
cc   hooking_test.o   -o hooking_test
sudo insmod hooking.ko
[sudo] password for os2013722095:
./hooking_test
sudo rmmod hooking
./hooking_test
os2013722095@ubuntu:~/21$ dmesg | grep "2013722095"
[    0.000000] 2013722095 in start_kernel()
[  110.902698] init_[2013722095]
[  110.926952] exit_[2013722095]
```

- 학번을 통해서 커널 메시지를 탐색하면 위와 같이 모듈이 적재/적출 되면서 찍힌 메시지를 확인할 수 있다.

- 고찰

  모듈을 적재하고 적출하는 부분에서 굉장히 애를 먹었다. 새로 만든 모듈에서 system call 함수를 호출하면 없는 함수라고 에러가 속출했고 아무리 검색해도 이에 대한 해결책이 되지는 않았다. 왜 강의 자료의 뒷부분을 보려하지 않았는지 이해가 가질 않는다. sys_exit를 후킹하는 방식으로 모듈을 만들었고, 이를 호출하도록 하였다. 소스가 되는 메인스크립트를 돌리게 되면 my_exit system call을 호출하게 되는데 이 과정에서 모듈이 적재되며 my_exit대신 my_init 이 호출된다. 적출 과정에서는 init에서 hooking하기 전 미리 백업해둔 변수를 통해 다시 my_exit 함수가 호출되게 하였으며 그 결과는 위에서 보는바와 같다. 다만 아직도 이해가 가지 않는 부분은 왜 강의 자료에서 모듈이 load/unload 부분처럼 작성한 코드가 왜 컴파일이 안 되는지 모르겠다. 처음에는 add를 init이 후킹하고 exit가 후킹한 뒤 복구를 안 해줬는데 그렇게 하니 add syscall이 망가지는 사태가 벌어져 버렸다. 결국 다시 커널 컴파일을 진행하여 초기화를 해주고, 나중에 호출되는 my_exit system call을 후킹하고 되돌려 주도록 코드를 수정하였다.