

# Assignment1

운영체제

1-1: Linux Installation & kernel compile

1-2: Modify kernel code

1-3: System call wrapping

제출일: 10월 14일 금요일

담당 교수: 김태석

학 번: 2015722025

학 과: 컴퓨터정보공학부

이 름: 정용훈

## 1. Introduction

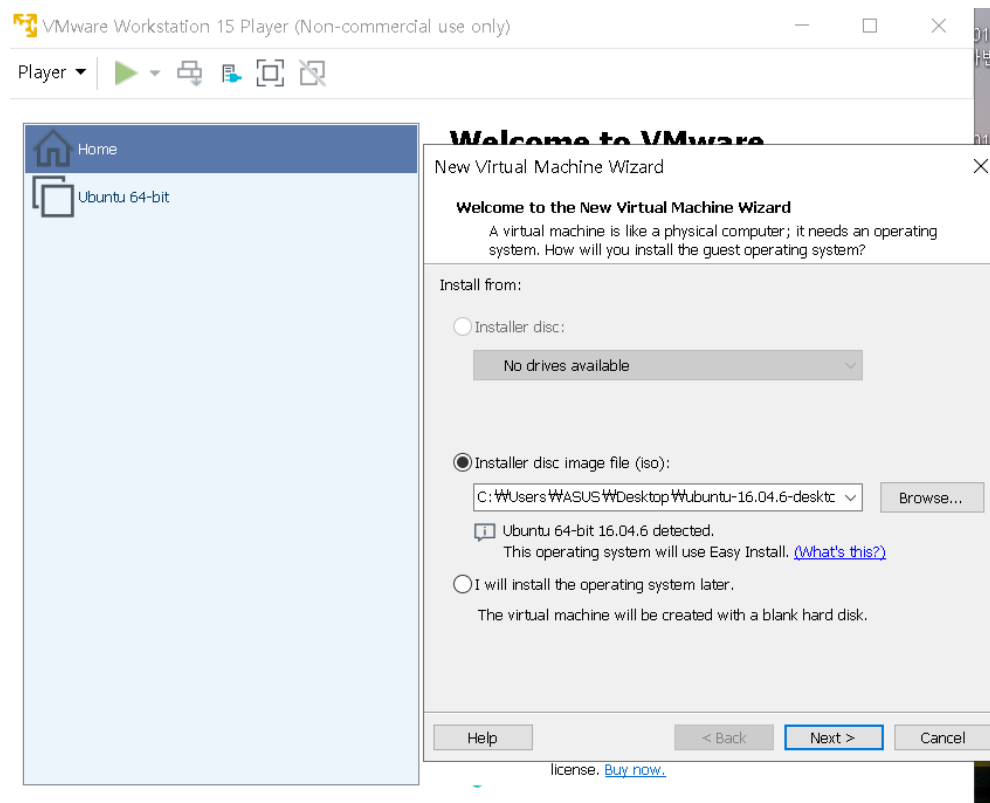
운영체제 첫 과제는 앞으로 과제를 수행하게 될 Linux환경과 kernel에 대한 설치 및 환경 설정을 익히고 이해하는 것이다. 다음으로는 kernel code를 수정함으로써 dmesg명령을 통하여 kernel message를 확인하여, code를 수정하는 방법을 익히고 마지막으로 System call을 이해하고 더 나아가 wrapping하는 skill을 익히며, module programming에 대한 개념을 익히는데 목적을 두는 과제다.

## 2. Requirements for each task(Conclusion)

### A. Assignment 1-1

#### (1) Linux Installation

전 학기 시스템프로그래밍에서 사용한 ISO파일의 버전이 달라 새로 ISO파일을 다운 받아 설치하게 되었다. Work station은 기존 설치되어있는 것을 사용하였으며, 아래 사진은 설치 과정이다.



시스템 프로그래밍에서 사용했던 Virtual machine을 대신하여 다른 버전을 사용하기 위하여 ISO파일을 선택하는 모습이다.

New Virtual Machine Wizard ×

**Easy Install Information**  
This is used to install Ubuntu 64-bit.

Personalize Linux

Full name:

User name:

Password:

Confirm:

과제 요구사항에 따라 계정은 "OS\_학번 or OS학번"에 만족하여 user name을 설정하는 부분이다.

New Virtual Machine Wizard ×

**Specify Disk Capacity**  
How large do you want this disk to be?

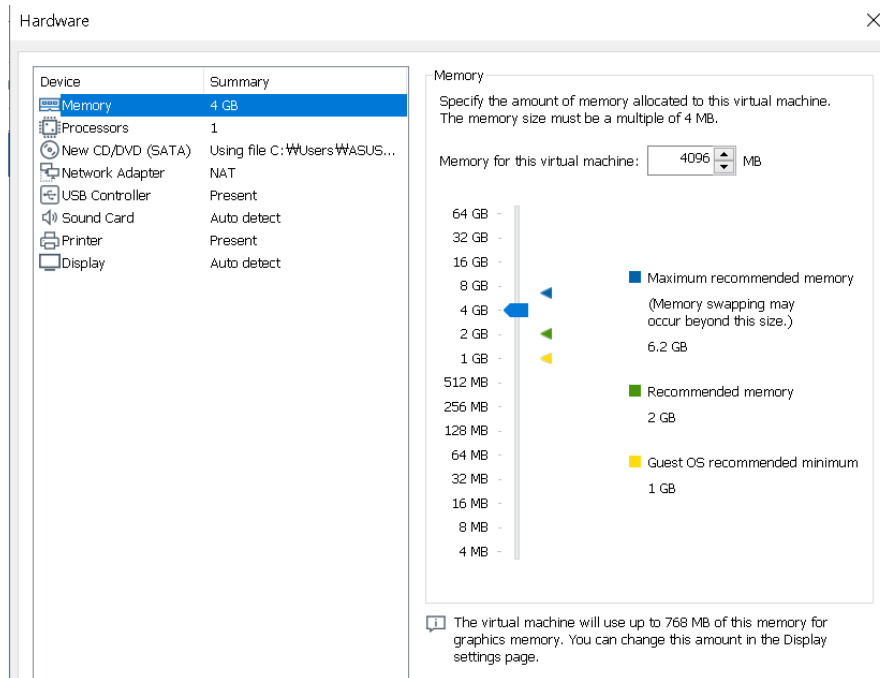
The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.

Maximum disk size (GB):

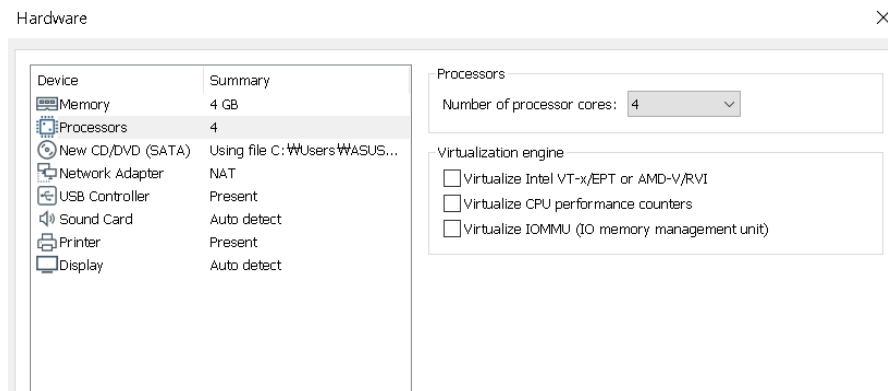
Recommended size for Ubuntu 64-bit: 20 GB

☐ Store virtual disk as a single file  
☒ Split virtual disk into multiple files  
 Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.

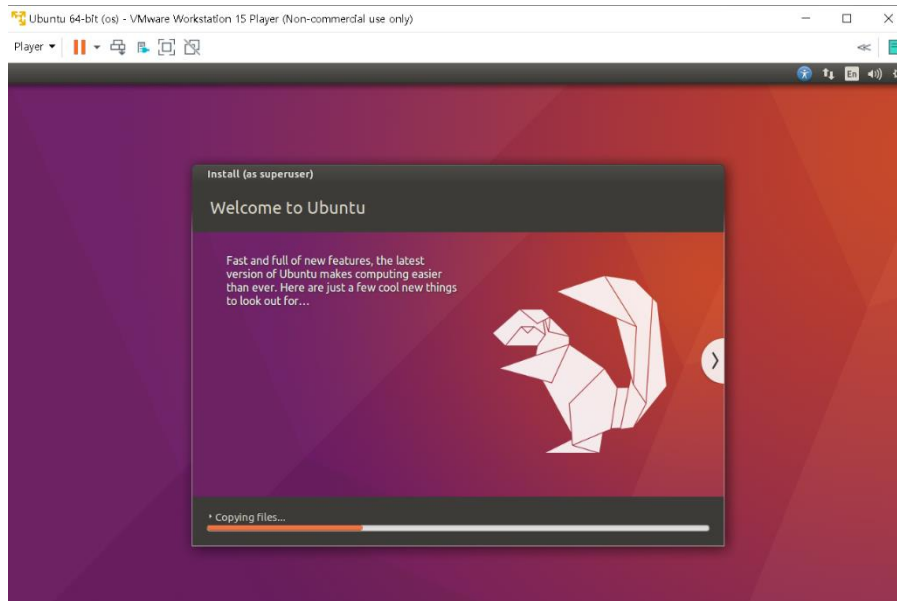
다음으로 사용하게 될 disk의 최대 size를 정하는데 기존 20GB가 아닌 60을 초과한 70을 설정하여 환경설정을 하는 과정이다.



Disk size 설정에 이어 Memory의 크기도 설정하는데 이는 초기 설정 1GB에 비하여 4 배 높은 4GB로 설정한 후 다음 과정을 진행한다.



다음으로 Core의 개수를 정의해주는데 이 또한 처음 1개로 설정되어있지만, 사용자 컴퓨터 사양에 따라 최대로 늘리게 되었고 본인은 4개까지 수용가능하기에 4개로 설정해준 모습이다.



Virtual Machine의 환경 설정을 마친 후 설치를 진행하는 과정이다.

```
os2015722025@ubuntu:~$ sudo apt update
[sudo] password for os2015722025:
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [109 kB]
Hit:2 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:4 http://security.ubuntu.com/ubuntu xenial-security/main amd64 DEP-11 Metad
ta [76.2 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu xenial-updates/main amd64 DEP-11 Metad
ata [324 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 64x64 Icons
[84.0 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 DEP-11 Me
tadata [124 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu xenial-updates/main DEP-11 64x64 Icons
[238 kB]
59% [9 icons-64x64 108 kB/238 kB 45%] [Connecting to security.ubuntu.com (91.18
```

초기 터미널을 실행한 모습이며, 명령어 'sudo apt update'를 통하여 설치되어있는 package를 모두 update해주는 모습이다.

## (2) Kernel compile

다음으로 1-1과제의 마지막 요구사항인 Kernel compile을 하는 과정을 소개하고 설명하는 부분이다.

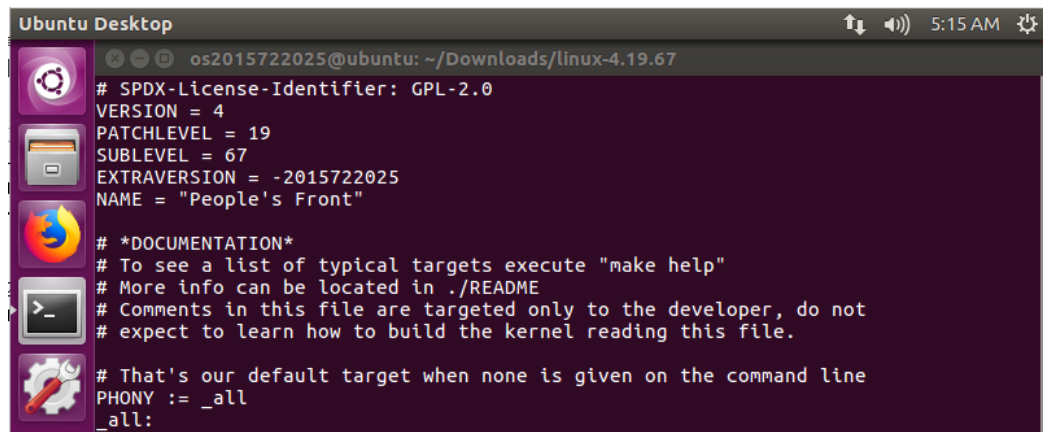
```
os2015722025@ubuntu:~/Downloads$ sudo wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.67.tar.gz
--2019-09-24 02:35:37-- https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.67.tar.gz
Resolving cdn.kernel.org (cdn.kernel.org)... 151.101.229.176, 2a04:4e42:36::432
Connecting to cdn.kernel.org (cdn.kernel.org)|151.101.229.176|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 159343779 (152M) [application/x-gzip]
Saving to: 'linux-4.19.67.tar.gz'

linux-4.19.67.tar.g 91%[=====] 139.59M 6.67MB/s eta 2s
```

Kernel version을 4.19.67을 사용하기 위해 오픈 소스에 제공하는 파일을 다운로드하기 위해 명령을 통하여 압축파일을 다운로드하고 있는 과정이다. 다운로드가 완료되면 압축을 푼 후 아래 과정을 따른다.

```
os2015722025@ubuntu:~/Downloads/linux-4.19.67$ vi Makefile
```

Vi를 실행하여 압축이 풀린 폴더의 Makefile에 접근을 한다.



```
Ubuntu Desktop
os2015722025@ubuntu: ~/Downloads/linux-4.19.67
# SPDX-License-Identifier: GPL-2.0
VERSION = 4
PATCHLEVEL = 19
SUBLEVEL = 67
EXTRAVERSION = -2015722025
NAME = "People's Front"

# *DOCUMENTATION*
# To see a list of typical targets execute "make help"
# More info can be located in ./README
# Comments in this file are targeted only to the developer, do not
# expect to learn how to build the kernel reading this file.

# That's our default target when none is given on the command line
PHONY := _all
_all:
```

다음과 같이 명령어 uname을 통하여 kernel version과 학번이 출력되어야 함으로 extraversion에 학번을 입력한다.

```
os2015722025@ubuntu:~/Downloads/linux-4.19.67$ sudo apt install build-essential  
libncurses5-dev bison flex libssl-dev libelf-dev
```

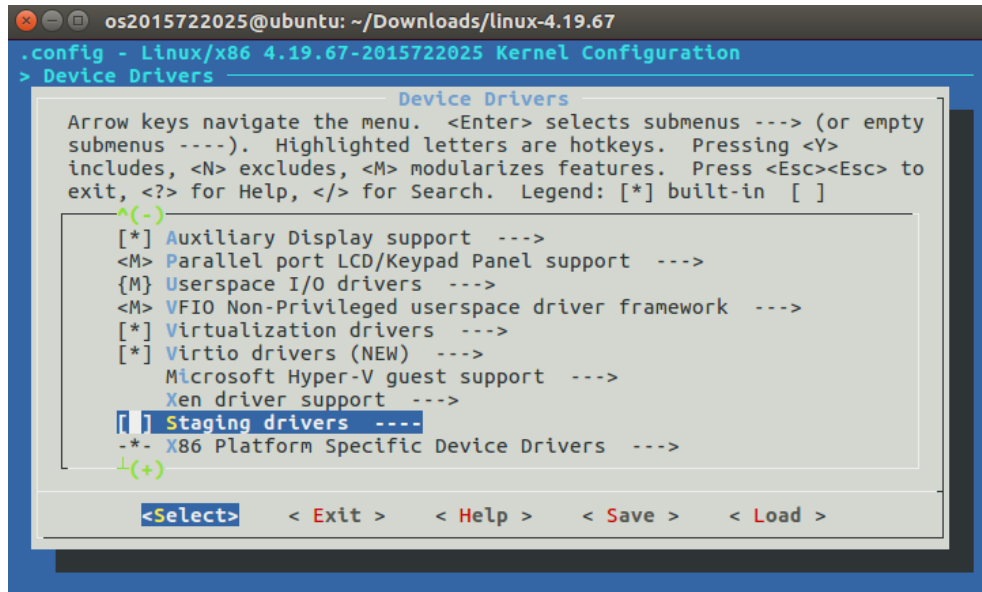
```
os2015722025@ubuntu: ~/Downloads/linux-4.19.67  
Processing triggers for install-info (6.1.0.dfsg.1-5) ...  
Processing triggers for man-db (2.7.5-1) ...  
Processing triggers for libc-bin (2.23-0ubuntu11) ...  
Processing triggers for doc-base (0.10.7) ...  
Processing 1 added doc-base file...  
Setting up libsigsegv2:amd64 (2.10-4) ...  
Setting up m4 (1.4.17-5) ...  
Setting up libfl-dev:amd64 (2.6.0-11) ...  
Setting up flex (2.6.0-11) ...  
Setting up libssl1.0.0:amd64 (1.0.2g-1ubuntu4.15) ...  
Setting up libelf1:amd64 (0.165-3ubuntu1.2) ...  
Setting up libbison-dev:amd64 (2:3.0.4.dfsg-1) ...  
Setting up bison (2:3.0.4.dfsg-1) ...  
update-alternatives: using /usr/bin/bison.yacc to provide /usr/bin/yacc (yacc) i  
n auto mode  
Setting up libelf-dev:amd64 (0.165-3ubuntu1.2) ...  
Setting up libtinfo-dev:amd64 (6.0+20160213-1ubuntu1) ...  
Setting up libncurses5-dev:amd64 (6.0+20160213-1ubuntu1) ...  
Setting up zlib1g-dev:amd64 (1:1.2.8.dfsg-2ubuntu4.1) ...  
Setting up libssl-dev:amd64 (1.0.2g-1ubuntu4.15) ...  
Setting up libssl-doc (1.0.2g-1ubuntu4.15) ...  
Processing triggers for libc-bin (2.23-0ubuntu11) ...  
Progress: [ 98%] [#####]
```

다음으로 kernel의 환경 설정을 위한 설치를 하며, 위는 그 과정을 담고 있는 모습이다.

```
os2015722025@ubuntu:~/Downloads/linux-4.19.67$ sudo make menuconfig
```

위 명령을 통해 환경설정이 가능한 interface를 띄워준다.

```
os2015722025@ubuntu: ~/Downloads/linux-4.19.67  
.config - Linux/x86 4.19.67-2015722025 Kernel Configuration  
> Enable loadable module support  
    Enable loadable module support  
    Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty  
    submenus ---). Highlighted letters are hotkeys. Pressing <Y>  
    includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to  
    exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]  
    --- Enable loadable module support  
    [*] Forced module loading  
    [*] Module unloading  
    [ ] Forced module unloading  
    [ ] Module versioning support  
    [*] Source checksum for all modules  
    [*] Module signature verification  
    [ ] Require modules to be validly signed  
    [*] Automatically sign all modules  
    Which hash algorithm should modules be signed with? (Sign m  
    ↓(+)  
    <Select> < Exit > < Help > < Save > < Load >
```



위와 같은 설정을 해준 후 저장 하고 종료하면 아래와 같은 출력을 확인할 수 있다.

```
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
os2015722025@ubuntu:~/Downloads/linux-4.19.67$
```

```
os2015722025@ubuntu:~/Downloads/linux-4.19.67$ make -j8
```

모든 과정이 끝나면 위와 같은 명령으로 compile을 실행한다. 옆 숫자는 thread의 숫자를 뜻한다.

```
os2015722025@ubuntu: ~/Downloads/linux-4.19.67
CC      init/do_mounts.o
CHK     include/generated/compile.h
CC      init/do_mounts_initrd.o
CC      init/do_mounts_md.o
HOSTCC  usr/gen_init_cpio
UPD     include/generated/compile.h
CC      init/initramfs.o
GEN     usr/initramfs_data.cpio
AS      usr/initramfs_data.o
CC      kernel/bpf/core.o
CC [M]  arch/x86/crypto/sha1-mb/sha1_mb.o
AR      usr/built-in.a
Generating X.509 key generation config
CC      certs/system_keyring.o
AS [M]  arch/x86/crypto/sha1-mb/sha1_mb_mgr_flush_avx2.o
CC      init/calibrate.o
CC      mm/filemap.o
CC      kernel/bpf/syscall.o
EXTRACT_CERTS
CC      certs/blacklist.o
CC      kernel/bpf/verifier.o
CC      kernel/bpf/inode.o
CC      init/init_task.o
```

컴파일 과정이 터미널에 보이는 것을 확인할 수 있다.



```
os2015722025@ubuntu:~/Downloads/linux-4.19.67$ sudo make modules_install
```

```
os2015722025@ubuntu: ~/Downloads/linux-4.19.67
INSTALL drivers/gpio/gpio-wcove.ko
INSTALL drivers/gpio/gpio-wm831x.ko
INSTALL drivers/gpio/gpio-wm8350.ko
INSTALL drivers/gpio/gpio-wm8994.ko
INSTALL drivers/gpio/gpio-ws16c48.ko
INSTALL drivers/gpio/gpio-xra1403.ko
INSTALL drivers/gpu/drm/amd/amdgpu/amdgpu.ko
INSTALL drivers/gpu/drm/amd/amdkfd/amdkfd.ko
INSTALL drivers/gpu/drm/amd/lib/chash.ko
INSTALL drivers/gpu/drm/ast/ast.ko
INSTALL drivers/gpu/drm/bridge/analogix-anx78xx.ko
INSTALL drivers/gpu/drm/cirrus/cirrus.ko
INSTALL drivers/gpu/drm/drm.ko
INSTALL drivers/gpu/drm/drm_kms_helper.ko
INSTALL drivers/gpu/drm/gma500/gma500_gfx.ko
INSTALL drivers/gpu/drm/hisilicon/hibmc/hibmc-drm.ko
INSTALL drivers/gpu/drm/i2c/ch7006.ko
INSTALL drivers/gpu/drm/i2c/sil164.ko
INSTALL drivers/gpu/drm/i2c/tda998x.ko
INSTALL drivers/gpu/drm/i915/gvt/kvmgt.ko
INSTALL drivers/gpu/drm/i915/i915.ko
INSTALL drivers/gpu/drm/mgag200/mgag200.ko
INSTALL drivers/gpu/drm/nouveau/nouveau.ko
```

Compile이 완료된 후 kernel을 building하기 위해 다음과 같은 명령을 입력한다.

```
os2015722025@ubuntu:~/Downloads/linux-4.19.67$ sudo make install
```

```
os2015722025@ubuntu: ~/Downloads/linux-4.19.67
run-parts: executing /etc/kernel/postinst.d/apt-auto-removal 4.19.67-2015722025
/boot/vmlinuz-4.19.67-2015722025
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 4.19.67-2015722025 /
boot/vmlinuz-4.19.67-2015722025
update-initramfs: Generating /boot/initrd.img-4.19.67-2015722025
run-parts: executing /etc/kernel/postinst.d/pm-utils 4.19.67-2015722025 /boot/vm
linux-4.19.67-2015722025
run-parts: executing /etc/kernel/postinst.d/unattended-upgrades 4.19.67-20157220
25 /boot/vmlinuz-4.19.67-2015722025
run-parts: executing /etc/kernel/postinst.d/update-notifier 4.19.67-2015722025 /
boot/vmlinuz-4.19.67-2015722025
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 4.19.67-2015722025 /b
oot/vmlinuz-4.19.67-2015722025
Generating grub configuration file ...
Warning: Setting GRUB_TIMEOUT to a non-zero value when GRUB_HIDDEN_TIMEOUT is se
t is no longer supported.
Found linux image: /boot/vmlinuz-4.19.67-2015722025
Found initrd image: /boot/initrd.img-4.19.67-2015722025
Found linux image: /boot/vmlinuz-4.15.0-45-generic
Found initrd image: /boot/initrd.img-4.15.0-45-generic
Found memtest86+ image: /boot/memtest86+.elf
Found memtest86+ image: /boot/memtest86+.bin
done
```

마지막으로 install을 통하여 파일을 복사하여 사용할 수 있도록 위 명령을 입력한다.

다음으로 virtual machine을 **Reboot** 후 'uname -r' 명령을 통해 kernel의 version을 확인한다.

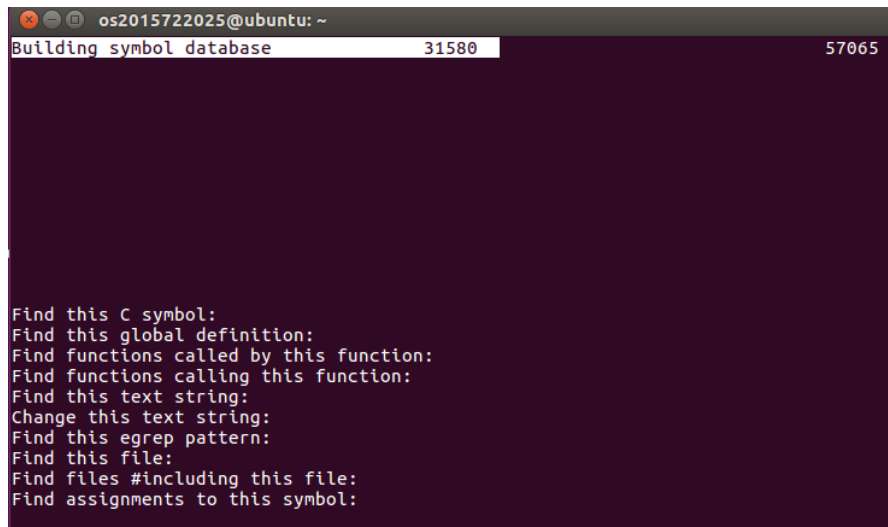
```
os2015722025@ubuntu: ~
os2015722025@ubuntu:~$ uname -r
4.19.67-2015722025
```

다음과 같이 입력한 학번과 kernel의 version을 확인할 수 있다.

## B. Assignment 1-2

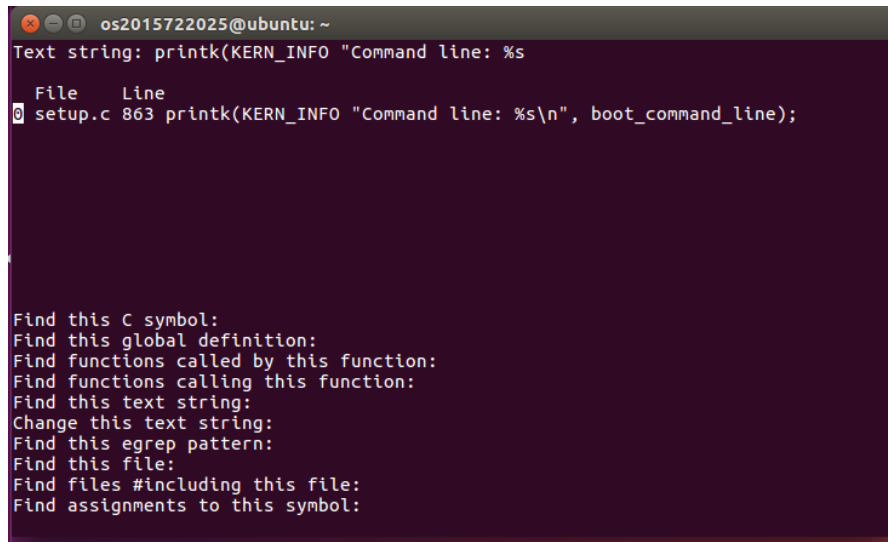
### (1) Modify kernel code

1-2과제는 dmesg를 통하여 학번이 터미널에 출력되도록 하는 과제이다. 우선 출력이 필요한 위치가 제시되어있기 때문에 검색 기능을 사용할 수 있는 명령인 cscope를 Install한 후 실행하면 된다.



```
os2015722025@ubuntu: ~  
Building symbol database 31580 57065  
  
Find this C symbol:  
Find this global definition:  
Find functions called by this function:  
Find functions calling this function:  
Find this text string:  
Change this text string:  
Find this egrep pattern:  
Find this file:  
Find files #including this file:  
Find assignments to this symbol:
```

위 그림과 같이 cscope를 실행하고, 정보를 찾기 위한 database를 준비하는 과정이다.



```
os2015722025@ubuntu: ~  
Text string: printk(KERN_INFO "Command line: %s  
File Line  
0 setup.c 863 printk(KERN_INFO "Command line: %s\n", boot_command_line);  
  
Find this C symbol:  
Find this global definition:  
Find functions called by this function:  
Find functions calling this function:  
Find this text string:  
Change this text string:  
Find this egrep pattern:  
Find this file:  
Find files #including this file:  
Find assignments to this symbol:
```

학번을 출력하기 위한 string을 검색하여 string이 포함되어있는 c file을 찾으며, 위 명령 창에서 바로 파일로 접근이 가능한데 접을 하면 아래와 같다.

```
os2015722025@ubuntu: ~
/*
 * Note: Quark X1000 CPUs advertise PGE incorrectly and require
 * a cr3 based tlb flush, so the following __flush_tlb_all()
 * will not flush anything because the cpu quirk which clears
 * X86_FEATURE_PGE has not been invoked yet. Though due to the
 * load_cr3() above the TLB has been flushed already. The
 * quirk is invoked before subsequent calls to __flush_tlb_all()
 * so proper operation is guaranteed.
 */
__flush_tlb_all();
#else
    printk(KERN_INFO "2015722025_OSLAB\n");
    printk(KERN_INFO "Command line: %s\n", boot_command_line);
    boot_cpu_data.x86_phys_bits = MAX_PHYSMEM_BITS;
#endif

/*
 * If we have OLPC OFW, we might end up relocating the fixmap due to
 * reserve_top(), so do this before touching the ioremap area.
 */
olpc_ofw_detect();

idt_setup_early_traps();
:wq
```

위와 같이 출력이 필요한 위치에 printk함수를 사용함으로써 학번이 찍힐 수 있도록 code를 추가해준다. 해당 위치의 코드를 추가한 이유는 cscope 명령을 통하여 dmesg에서 제공받은 정확한 위치와 파일을 찾았기 때문이고 간단하게 printk 함수를 추가하고 dmesg를 다시 입력하면 학번이 출력된 모습을 확인할 수 있다.

<wnloads/linux-4.19.67/arch/x86/kernel/setup.c" 1312L, 33193C 863,9 66%

다음은 해당 code의 path다.

```
os2015722025@ubuntu: ~
os2015722025@ubuntu:~$ dmesg grep | head
[ 0.000000] Linux version 4.19.67-2015722025 (os2015722025@ubuntu) (gcc versi
on 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1-16.04.11)) #2 SMP Thu Sep 26 07:55:59 P
DT 2019
[ 0.000000] 2015722025_OSLAB
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-4.19.67-2015722025 root=UU
ID=edb695ec-44a4-4fd5-ae7c-01b9c6a65056 ro find_preseed=/preseed.cfg auto noprom
pt priority=critical locale=en_US quiet
[ 0.000000] KERNEL supported cpus:
[ 0.000000] Intel GenuineIntel
[ 0.000000] AMD AuthenticAMD
[ 0.000000] Centaur CentaurHauls
[ 0.000000] Disabled fast string operations
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point regi
sters'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
```

위 출력과 같이 요구 사항 위치에 학번이 출력된 것을 확인할 수 있다.

## C. Assignment 1-3

### (1) System call wrapping

```
os2015722025@ubuntu:~/Downloads/linux-4.19.67$ vi arch/x86/entry/syscalls/syscall_64.tbl
```

546	x32	preadv2	__x32_compat_sys_preadv64v2
547	x32	pwritev2	__x32_compat_sys_pwritev64v2
549	common	add	__x64_sys_add
			384,1

Table 등록

```
os2015722025@ubuntu:~/Downloads/linux-4.19.67$ vi include/linux/syscalls.h
```

```
    if (personality != 0xffffffff)
        set_personality(personality);

    return old;
}
asmlinkage long sys_add(int, int);
#endif
```

System call 함수 구현

```
os2015722025@ubuntu: ~/Downloads
#include <linux/kernel.h>
#include <linux/syscalls.h>

SYSCALL_DEFINE2(add, int, a, int, b)
{
    long ret;

    ret = a + b;

    return ret;
}
```

요구사항 대로 덧셈 동작을 하는 system call을 code작성해준다.

```
os2015722025@ubuntu:
obj-y := add.o
```

```

objtool_target := tools/objtool FORCE
else
SKIP_STACK_VALIDATION := 1
export SKIP_STACK_VALIDATION
endif
endif

ifeq ($(KBUILD_EXTMOD),)
core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/ add/

```

Kernel이 수정되었으므로 kernel 컴파일 진행

```

os2015722025@ubuntu: ~/working
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/syscall.h>

int main(int argc, char *argv[])
{
    long ret;
    int a, b;

    a=atoi(argv[1]);
    b=atoi(argv[2]);

    ret = syscall(549, a, b);
    printf("%d op. %d = %ld\n", a, b, ret);

    return 0;
}

```

System call test를 위한 파일 생성

```

os2015722025@ubuntu:~/working$ gcc -o test -g test.c
os2015722025@ubuntu:~/working$ ls
test  test.c
os2015722025@ubuntu:~/working$ ./test 10 12
10 op. 12 = 22
os2015722025@ubuntu:~/working$

```

확인결과 덧셈이 system call에 의한 덧셈이 잘되는 것을 확인할 수 있다.

```

os2015722025@ubuntu: ~/moduleTest
#include <linux/module.h>
#include <linux/highmem.h>
#include <linux/kallsyms.h>
#include <linux/syscalls.h>
#include <asm/syscall_wrapper.h>

#define __NR_add 549

void **syscall_table;
void *real_add;

__SYSCALL_DEFINE2(mul, int, a, int, b)
{
    return a*b;
}

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

    make_rw(syscall_table);

    real_add = syscall_table[__NR_add];
    syscall_table[__NR_add] = __x64_sysmul;
    printk(KERN_INFO "init_[2015722025]\n");
    return 0;
}

static void __exit hooking_exit(void)
{
    syscall_table[__NR_add] = real_add;

    make_ro(syscall_table);
    printk(KERN_INFO "exit_[2015722025]\n");
}

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

```

다음으로 모듈을 적재하게 되면 동작하는 code를 작성하면 된다. 이번 과제에서는 549번 system call인 덧셈을 곱셈으로 바꾸기 때문에 return 값이 곱셈인 것을 확인할 수 있다. 그 후 적재 시 kernel 메시지에 학번이 출력되어야 하므로 init함수에 printk를 통해 학번을 입력해주고 마찬가지로 exit제거하였을 경우에도 학번이 출력되어야 하므로 exit함수에 학번출력 code를 추가해준다.

```

os2015722025@ubuntu:~/moduleTest$ make
make -C /lib/modules/4.19.67-2015722025/build SUBDIRS=/home/os2015722025/moduleTest modules
make[1]: Entering directory '/home/os2015722025/Downloads/linux-4.19.67'
  CC [M]  /home/os2015722025/moduleTest/my_wrapper.o
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/os2015722025/moduleTest/my_wrapper.ko
make[1]: Leaving directory '/home/os2015722025/Downloads/linux-4.19.67'

```

그 후 module code를 make명령을 통하여 사용가능 하도록 compile한다.

```

os2015722025@ubuntu:~/moduleTest$ sudo insmod my_wrapper.ko
[sudo] password for os2015722025:
os2015722025@ubuntu:~/moduleTest$ cd ..
os2015722025@ubuntu:~$ cd working
os2015722025@ubuntu:~/working$ ./test 5 4
5 op. 4 = 20
os2015722025@ubuntu:~/working$

```

다음으로 sudo insmod를 통해 compile된 code를 정재하고 미리 작성해두었던, test code를 통해 system call명령이 덧셈에서 곱셈으로 wrapping되는 것을 확인한다.

```

os2015722025@ubuntu:~/moduleTest$ dmesg | tail -n 1
[ 886.879664] init_[2015722025]

```

위 출력은 모듈이 적재되었을 때, 정상적으로 학번이 출력되는 것을 확인한 결과이다.

```

os2015722025@ubuntu:~/working$ ./test 5 4
5 op. 4 = 20

```

곱셈으로 wrapping되어 원래 덧셈을 담당했던 system call이 곱셈으로 바뀐 모습이다.

```

os2015722025@ubuntu:~/moduleTest$ sudo rmmod my_wrapper

```

Sudo rmmod를 통하여 wrapping됐던 모듈을 제거한다.

```

os2015722025@ubuntu:~/moduleTest$ dmesg | tail -n 1
[ 1025.288383] exit_[2015722025]

```

그 후 메시지를 출력하면, "exit\_학번"을 확인할 수 있으며, 이는 exit이 잘 되었다는 것을 알 수 있다.

```

os2015722025@ubuntu:~/working$ ./test 10 20
10 op. 20 = 30

```

Test 실행파일을 통해 다시 덧셈으로 system call이 동작하는 것을 확인할 수 있다.



### 3. Reference

제공된 강의 자료를 통해 충분히 수행할 수 있었습니다.

### 4. Consideration

우선 저번학기 시스템프로그래밍에서 사용하였던 환경과 다른 환경에서 작업해야 했으므로 가상 머신을 조건에 맞춰 다시 설치하였다. 설치 과정 자체는 가이드에 따라 순조롭게 진행되었다. 그리고 이론에서 배웠던 system call을 직접 구현하는데 있어 코드가 들어가는 이유와 프로그램을 실행시키는 코드 자체는 어렵지 않지만 compile 과정에 시간이 많이 걸리고, 처음 써보는 명령이 있었기에 앞으로는 오타에 유의해야 한다. 마지막으로 wrapping을 통해 기존 실행을 변경하는 과제를 진행하였는데, wrapping에 필요성과 실제 사용 사례를 찾아볼 필요가 있을 것 같다.

각 명령과 코드에 대한 분석 결과는 위 항목(conclusion)에서 자세히 다루었습니다.