Operating Systems

Assignment #1

Professor	김태석 교수님
Department	컴퓨터공학과
Student ID	2012722028
Name	장 한 별
Date	2018. 10. 10.

Introduction.

이번 과제는 linux 설치와 kernel compile하는 과정을 파악하고, System call 및 Modules을 작성하는 것이다. Kernel을 compile 하는 과정에서 본인의 학번을 적절한 위치에 추가하여 kernel compile 과정을 이해하는 것과 system call 을 작성 후, wrapping 하는 module을 적재 및 제거함으로써 kernel 관련 지식을 파악하는데 목적이 있다.

Reference.

[Assignment 1-1]

[Assignment 1-2]

https://wiki.kldp.org/KoreanDoc/html/EmbeddedKernel-KLDP/x1942.html

http://kkamagui.tistory.com/817

http://blog.naver.com/PostView.nhn?blogId=dog9230&logNo=26104267

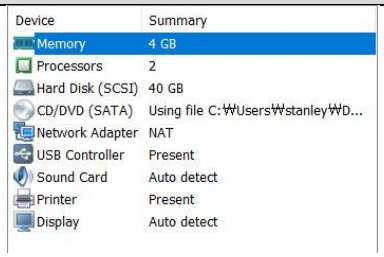
[Assignment 1-3]

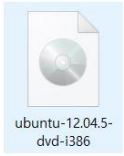
13학번 권영상 학우에게 argv[ ] 활용 관련 도움을 받음

Conclusion.

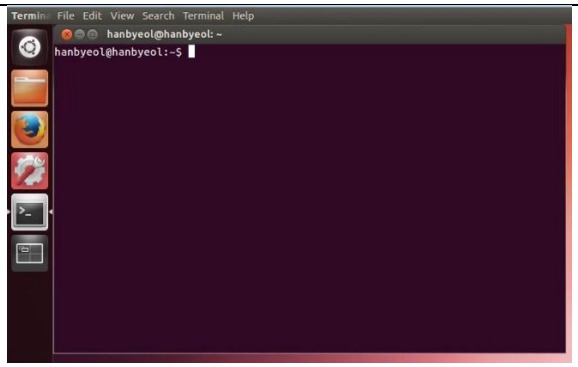
- **Assignment 1-1**

[Linux Installation]



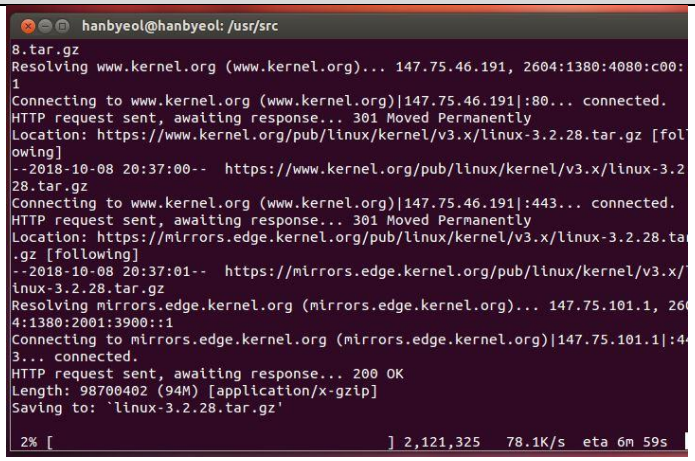


Ubuntu 12.04.5 32-bit ISO 파일을 다운받고, VMware Workstation에서 Virtual Machine을 새로 생성한다. 이때 virtual machine setting 은 위 그림과 같이 Memory는 4 GB, Processors는 2, Hard Dirsk는 40GB 로 설정했고, CD/DVD(SATA)에 다운받은 ISO 파일을 사용한다.

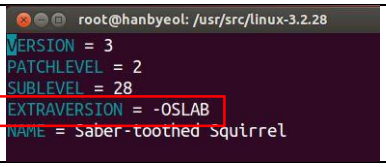


위 그림은 VMware에서 Ubuntu 설치가 모두 완료된 후 terminal을 실행시킨 화면이다.

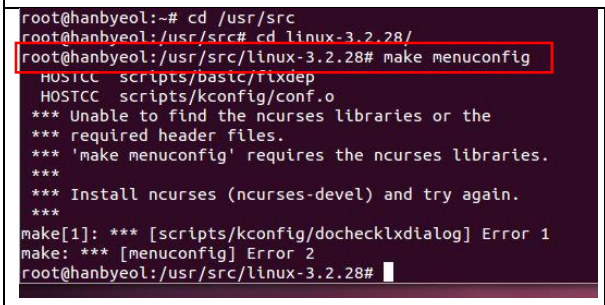
[kernel compile]

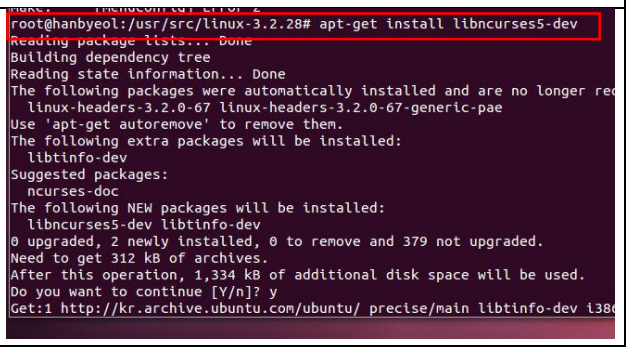


Terminal에서 # wget https://www.kernel.org/pub/linux/kernel/v3.x/linux-3.2.28.tar.gz 를 입력해 kernel source를 download 한다. Kernel version은 linux-3.2.28 이다.



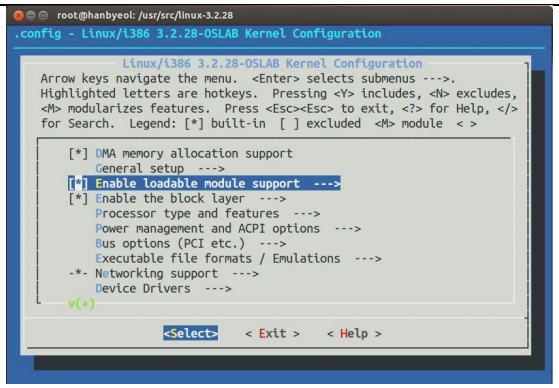
Kernel source 압축해제 후, 위 그림과 같이 kernel Extra Version을 수정한다.

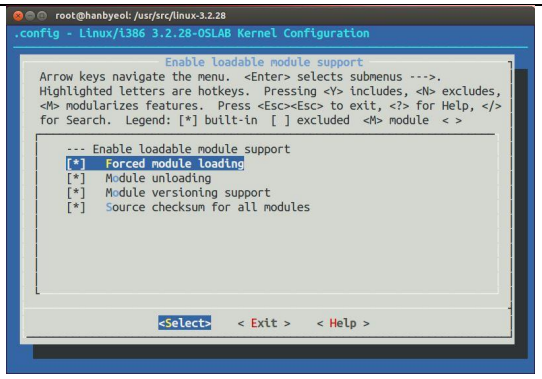


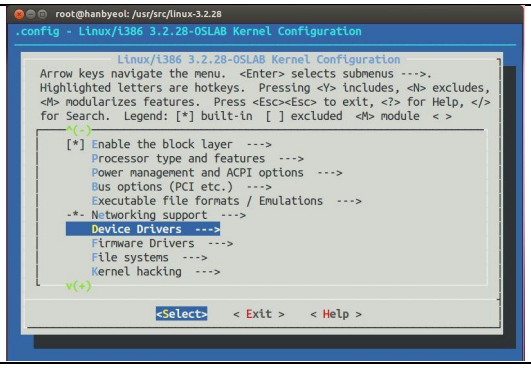


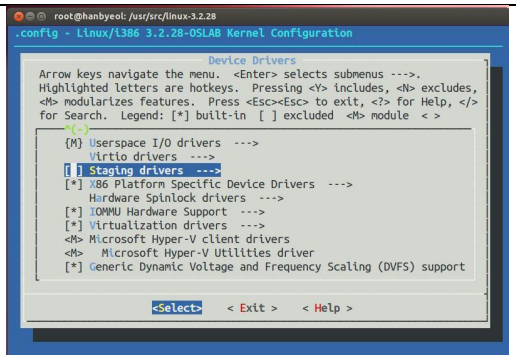
Terminal에서 # make menuconfig 을 입력해 kernel 환경 설정을 확인할 수 있다. Ncurses를 설치해야 한다. 오른쪽 그림과 같이 명령어를 입력 후 ncurses 설치를 완료한다. 그 후 다시 # make

menuconfig 를 입력해 kernel 환경설정으로 들어간다.

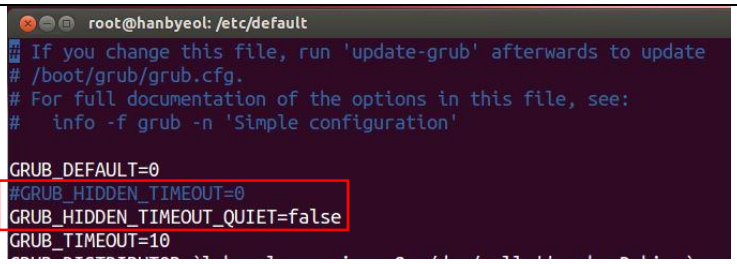




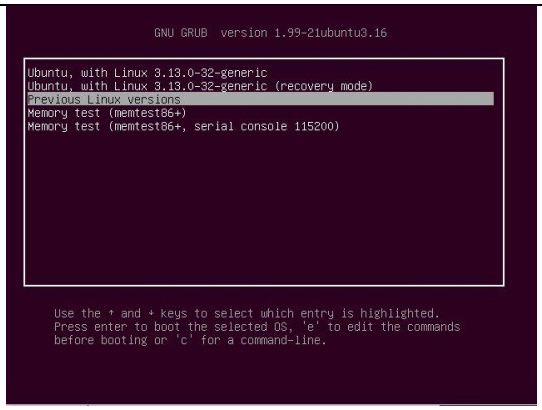


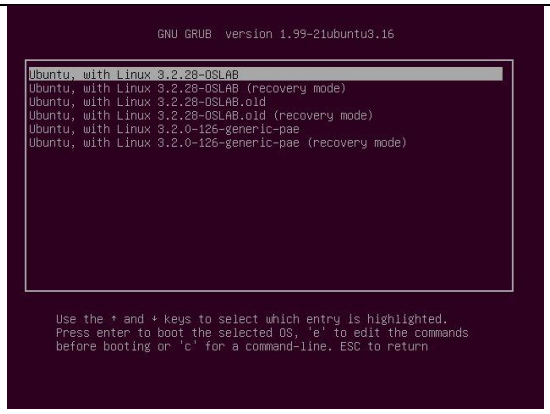


위 그림은 module을 삽입/제거 할 수 있도록 설정해준 화면이다. Enable loadable module support 로 들어가 Forced module loading 을 체크한다. 또한 device Driver의 Staging drivers에 체크를 제거 후 저장한다. 그 후 # make –j2 로 kernel을 compile 한다. 처음에 processors를 2로 등록했으므로 make –j2 로 수행한다. 이 명령어는 make dep, make clean, make bzImage, make modules 가 통합된 명령어이다. Module install 후, compile된 kernel을 boot loader에 등록후 # reboot 명령어를 통해 재부팅한다.

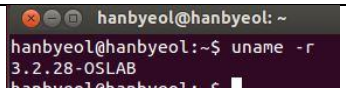


위 그림은 GRUB 설정 화면이다. GRUB_HIDDEN_TIMEOUT=0을 주석처리하고, QUIET를 false로 수정한다. Grub를 update 후 역시 # reboot 로 재부팅한다.





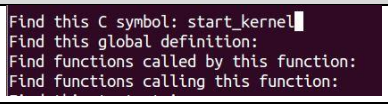
Grub bootloader에서 previous linux versions 을 선택하고, compile한 kernel을 선택한다.



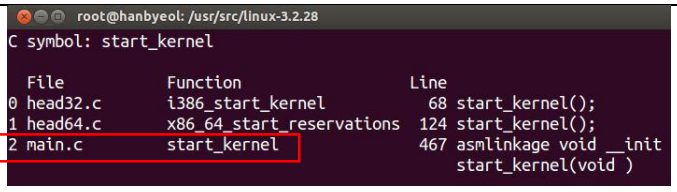
uname –r 명령어를 통해 현재 kernel version을 확인할 수 있다. 위 그림을 통해 올바르게 출력되었음을 확인할 수 있고, EXTRAVERSION에서 –OSLAB을 추가한 것까지 확인할 수 있다.

- **Assignment 1-2**

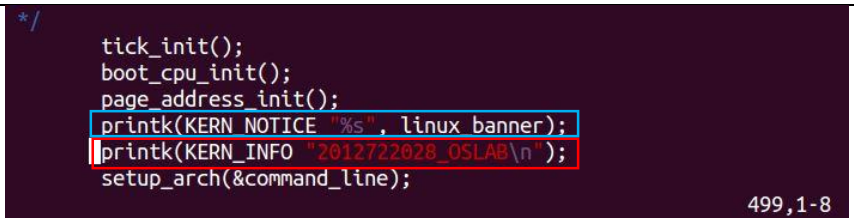
[커널 코드 수정]



위 그림은 cscope를 이용해 start_kernel을 검색한 화면이다. Cscope를 이용시 변수나 함수를 검색하는 부분에 매우 용이하다.



start_kernel을 검색한 결과 main.c에 정의되어 있는 것을 확인할 수 있다. main.c로 들어가 kernel code를 확인하도록 한다.



위 그림에서 빨간색 네모는 printk( ) 함수를 사용하여 학번을 출력하는 코드를 추가한 부분이다. 로그 레벨은 KERN_INFO를 사용했다. 파란색 네모는 linux_banner 문자열을 출력한 부분인데, kernel에 학번이 제대로 출력되었는지 확인 후 이 부분에 대해 자세히 설명하도록 한다. 또한 main.c에서

499번째 줄에 추가했음을 확인할 수 있다. Path: /usr/src/linux-3.2.28/init/main.c

```
root@hanbyeol:/usr/src/linux-3.2.28# dmesg | head -n 5
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Linux version 3.2.28-OSLAB (root@hanbyeol) (gcc version 4.6.3 (Ub
untu/Linaro 4.6.3-1ubuntu5) ) #2 SMP Tue Oct 9 03:44:39 KST 2018
[    0.000000] 2012722028_OSLAB
[    0.000000] KERNEL supported cpus:
root@hanbyeol:/usr/src/linux-3.2.28#
```

위 그림에서 빨간색 네모를 통해 kernel에 학번이 올바르게 출력되었음을 확인할 수 있다. # dmesg | head -5 명령어를 사용해 kernel message 위에서 5번째줄까지 출력하게 했다. 파란색 네모를 확인해 보면 linux version 뿐만 아니라 여러가지 정보가 출력되었음을 확인할 수 있다. 위 과정에서 printk( ) 함수로 linux_banner 를 출력하는 부분이 있었으므로 확인해보도록 한다.

```
C symbol: linux_banner

  File           Function                 Line
0 printk.h       <global>                    6 extern const char
                                               linux_banner[];
1 version.c      <global>                   41 const char linux_banner[] =
2 early_printk.c init_early_exception_vect  175 early_shadow_puts(linux_banner);
3 trace.c        show_regs                  875 pr_notice("%s", linux_banner);
4 prom_init.c    prom_init                 2838 prom_printf("Preparing to boot
                                               %s", RELOC(linux_banner));
5 main.c         start_kernel               498 printk(KERN_NOTICE "%s",
                                               linux_banner);
```

Cscope를 사용하여 linux_banner를 검색한다.

```
/* FIXED STRINGS! Don't touch! */
const char linux_banner[] =
        "Linux version " UTS_RELEASE " (" LINUX_COMPILE_BY "@"
        LINUX_COMPILE_HOST ") (" LINUX_COMPILER ") " UTS_VERSION "\n";

const char linux_proc_banner[] =
        "%s version %s"
        " (" LINUX_COMPILE_BY "@" LINUX_COMPILE_HOST ")"
        " (" LINUX_COMPILER ") %s\n";
"init/version.c" 48L, 1122C                                   48,1-8
```

위 그림을 통하여 linux_banner 은 linux version 뿐만 아니라 LINUX_COMPILE_BY, LINUX_COMPILE_HOST 등 누구에 의해 compile 되었는지 compile host는 누구인지 compiler는 무엇인지 등을 저장하는 문자열임을 확인할 수 있다. 이를 통해 위 과정에서 파란색 네모 부분으로 체크된 부분이 무슨 내용인지 확인할 수 있었고, 학번을 출력하는 코드는 이 부분 다음으로 추가해야한다.

```
root@hanbyeol:/usr/src/linux-3.2.28/init# ls
built-in.o   do_mounts_initrd.c  do_mounts_rd.c  main.c          mounts.o
calibrate.c  do_mounts_initrd.o  do_mounts_rd.o  main.o          noinitramfs.c
calibrate.o  do_mounts_md.c      initramfs.c     Makefile        version.c
do_mounts.c  do_mounts_md.o      initramfs.o     modules.builtin version.o
do_mounts.h  do_mounts.o         Kconfig         modules.order
```

위 그림을 통해 수정한 소스코드 path는 /usr/src/linux-3.2.28/init/main.c 임을 확인할 수 있다. main.c 에서 추가한 소스코드는 499번째 줄이다.

- **Assignment 1-3**

[System call]

```
#define __NR_process_vm_writev  348
#define __NR_add                349

#endif /* _ASM_X86_UNISTD_32_H */
                                                    359,1
```

위 그림은 System call 이름 및 번호 할당하는 부분이다. 덧셈에 대한 system call 이름과 번호를 할당한다. Path: /usr/include/i386-linux-gnu/asm 경로에서 unistd_32.h 파일을 찾아 추가했다.

```
        .long sys_process_vm_readv
        .long sys_process_vm_writev
        .long sys_add
-- INSERT --                                        351,15-22
```

위 그림은 system call 테이블 등록하는 부분이다. Path: /usr/src/linux-3.2.28/arch/x86/kernel 경로에서 syscall_table_32.S 파일에 할당해줬던 system call을 등록한다.

```
root@hanbyeol: /home/hanbyeol/2012722028
#include <linux/kernel.h>

asmlinkage int sys_add(int a, int b)
{
        return a+b;
}
```

위 그림은 system call 함수 구현하는 부분이다. Path: /home/hanbyeol/2012722028 경로에 my_add.c 파일을 위 그림과 같이 작성 후 kernel을 다시 컴파일 한다.

```
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 ti
            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
            hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
            notifier.o ksysfs.o sched_clock.o cred.o \
            async.o range.o my_add.o
```

위 그림은 새로 작성한 system call의 object 를 Path: /usr/src/linux-3.2.28/kernel 경로에 Makefile에 추가한 화면이다.

```
root@hanbyeol: /home/hanbyeol/2012722028
#include <linux/unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char *argv[])
{
        int a=0, b=0;
        a = atoi(argv[1]);
        b = atoi(argv[2]);
        printf("%d + %d = %d\n", a, b, syscall(__NR_add, a, b));
        return 0;
}
~
```

위 그림은 새로운 system call을 테스트할 프로그램을 작성한 화면이다. Path: /home/hanbyeol/2012722028 경로에 test_add.c 를 추가하여 vi로 작성한다. 이때 argv[1] 은 연산시

첫 번째 인자가 들어가고, argv[2] 는 두 번째 인자가 들어간다. Char* 형으로 선언했으므로 atoi( ) 함수를 사용하여 int형으로 형변환해서 변수를 저장한다.

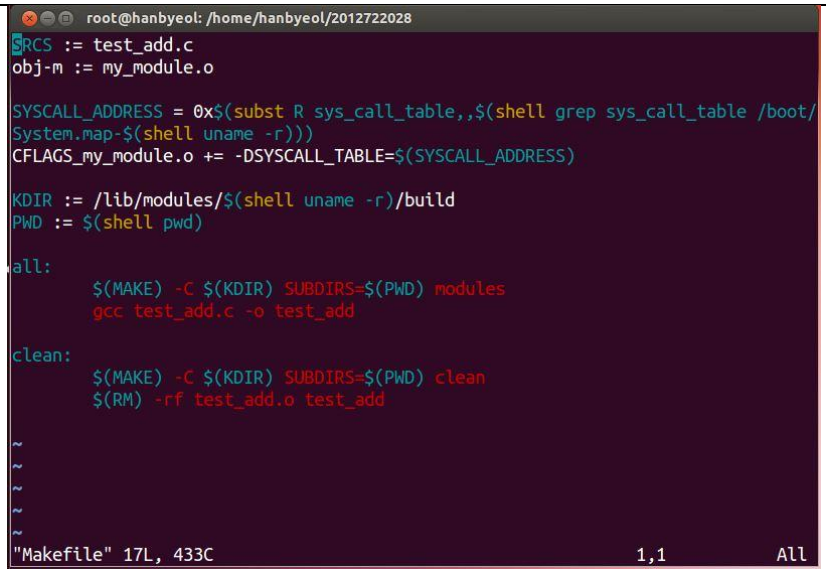
[Modules]

```
int hooking_init(void)
{
        printk(KERN_INFO "init_[2012722028]");
        make_rw(syscall_table);
        real_add = syscall_table[__NR_add];
        syscall_table[__NR_add] = sub;
        return 0;
}
```

```
void hooking_exit(void)
{
        printk(KERN_INFO "exit_[2012722028]");
        syscall_table[__NR_add] = real_add;
        make_ro(syscall_table);
}
```

위 그림은 module을 적재 및 제거시, kernel에 message를 출력하게 하는 코드를 추가한 화면다. Path: /home/hanbyeol/2012722028 경로에서 my_module.c 파일 중 hookin_init 이 정의된 부분은 moulde 을 적재 시에 이루어지는 코드이므로, 이 부분에 printk( ) 함수를 사용하여 module이 적재되었을 때, KERN_INFO 와 "init_[2012722028]" 이 출력하도록 추가했다.

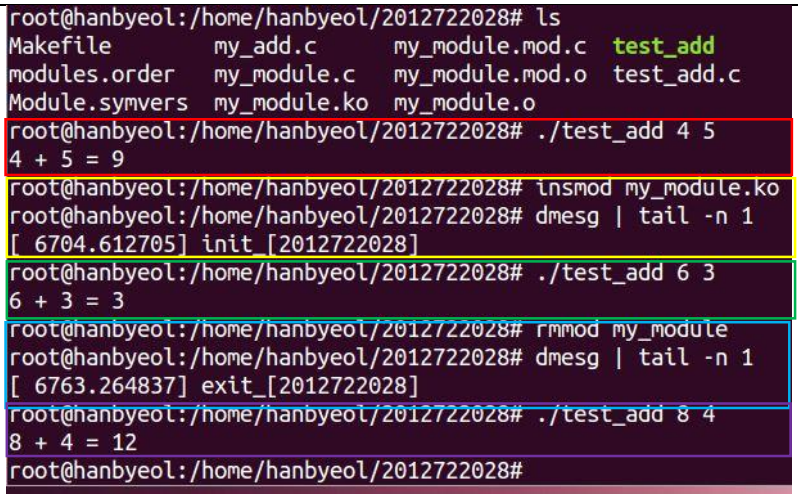
마찬가지로 module이 제거 시 이루어지는 코드인 hooking_exit 에 module이 제거되었을 때 메시지를 출력하게 하기 위해 printk( ) 함수를 사용하여 KERN_INFO 와 "exit_[2012722028]" 을 출력하도록 추가했다.



위 그림은 Makefile을 작성한 화면이다.

```
root@hanbyeol:/home/hanbyeol/2012722028# make
make -C /lib/modules/3.2.28-OSLAB/build SUBDIRS=/home/hanbyeol/2012722028 modules
make[1]: Entering directory `/usr/src/linux-3.2.28'
  CC [M]  /home/hanbyeol/2012722028/my_module.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/hanbyeol/2012722028/my_module.mod.o
  LD [M]  /home/hanbyeol/2012722028/my_module.ko
make[1]: Leaving directory `/usr/src/linux-3.2.28'
gcc test_add.c -o test_add
```

Makefile을 올바르게 작성했다면, # make 명령어를 입력한다.



그림을 통해 make가 제대로 수행되었음을 확인할 수 있었고, 실행파일인 test_add 역시 제대로 생성됨을 확인할 수 있다. 빨간색 네모는 4 와 5를 인자로 하여 test_add를 실행시킨 결과화면이다. 덧셈 연산이 올바르게 수행되었음을 확인할 수있다.

노란색 네모는 module을 삽입 후 메시지를 출력한 화면이다. Init_2012722028 이 올바르게 출력됨을 확인할 수 있다. 이때 tail –n 1 은 출력하고자하는 메시지의 마지막줄에서 1줄만 출력하게 하는 dmesg option이다.

초록색 네모는 module을 적재 후 test_add 실행파일을 6 과 3을 인자로 실행시킨 결과화면이다. 뺄셈을 수행하는 module이 적재되었으므로 6 과 3의 연산 결과인 3이 올바르게 출력되었음을 확인할 수있다.

파란색 네모는 적재된 module을 제거후, 메시지를 출력한 화면이다. Exit_2012722028 이 올바르게 출력됨을 확인할 수 있다.

그 후 보라색 네모에서 8 과 4 를 인자로 test_add 를 실행한 결과, 덧셈 연산이 올바르게 수행되었으므로 module이 올바르게 제거되었음을 확인할 수 있다.

[고찰]

이번 과제는 linux 설치부터 kernel compile하는 과정을 파악하고, kernel code 분석 후, 새로운 system call 및 module 작성하는 것이다. 초반부터 쉽지않았다. workstation이나 iso 파일이 다른 version이어도 실습하는데 문제가 없을 것이라 판단해 기존에 설치되어 있는 version 으로 진행했지만 계속 error 가 나서 홀로 독단적으로 판단했다는 생각을 했다. 다시 iso 나 kernel source version 을 pdf 에 맞춰 내려 받아 문제가 없이 수행되었다. 또한 실습 강의를 수강하지 않는 필자는 전체적인 kernel system 흐름을 파악하는데 많은 시간이 걸렸다. 운영체제 이론 강의로만 배웠던 부분을 홀로 실습하려 하니 감이 잘 오지 않았다. 처음에는 주어진 pdf를 그냥 따라하는 것에만 집중했지만

과정을 나아가도 이해가 되지않아 먼저 Booting Process 가 어떻게 이루어지는지 파악하는데 중점을 뒀다. BIOS가 실행된 후 boot loader가 실행되어 memory에 적재되고, 그 후 kernel이 실행된다는 과정을 알고 난 후 pdf 에서 무엇을 요구하는지 보이기 시작했다.

먼저 assignment 1-2 에서 kernel 에 필자의 학번을 출력하는 부분은 start_kernel 에서 printk( ) 로 linux_banner 문자열을 출력하는 것을 발견함으로 시작했다. 이 문자열은 linux의 compiler 가 무엇인지 누구에 의해 compile 되었는지 등 각종 정보가 담겨있는 문자열인데 dmesg 를 통해 출력했을 때 확인할 수 있어서 필자의 학번을 이 문자열이 출력되는 다음에 추가하여 해결할 수 있었다.

System call을 작성하는 것은 pdf 를 통해 어렵지 않게 구현했으나 module을 적재하거나 제거할 때 messge를 출력하는 부분을 어디에 추가할지 파악하는 것이 까다로웠다. Hooking_init 과 hooking_exit 함수가 각각 module이 적재되고 제거됨에 수행되는 코드라는 것을 이해하고 printk( ) 를 추가해서 init_[학번] 과 exit_[학번] 이 출력되는 것을 확인했을 때, 필자가 kernel 구조에 대해 어느정도 배경지식이 생겼고, 이해도가 향상되었다는 것을 느꼈다. 이렇게 사용하는 것은 kernel의 size 를 최소화 할 수 있고, 유연성 면에서 아주 좋은 점이 나타날 수 있는 좋은 방법이라고 생각한다.

이번 과제를 통해 OS의 전반적인 부팅과정이나 kernel 의 동작 방식을 조금이나마 파악할 수 있게 되었고, 과거 전세계에서 컴퓨터공학의 길을 걸었던 선배님들의 도움으로 현재 편하게 프로그래밍할 수 있는 환경이 구축되었다는 생각이 들어 감사함을 느꼈다.