

2022년 2학기 운영체제실습 3주차

Linux Kernel & Kernel compile

System Software Laboratory

School of Computer and Information Engineering
Kwangwoon Univ.

Linux Kernel

- **Kernel**

- System을 구성하는 중요한 자원들의 관리
 - Memory, File, Device, etc
- OS의 핵심 역할을 수행
 - Process management, Time management, CPU scheduling, etc

- **다양한 architecture 지원**

- X86, Alpha, ARM, SPARC, MIPS, PowerPC, ia64, AMD x86-64 등
- 현존하는 다양한 architecture에 설치(porting)가 가능함

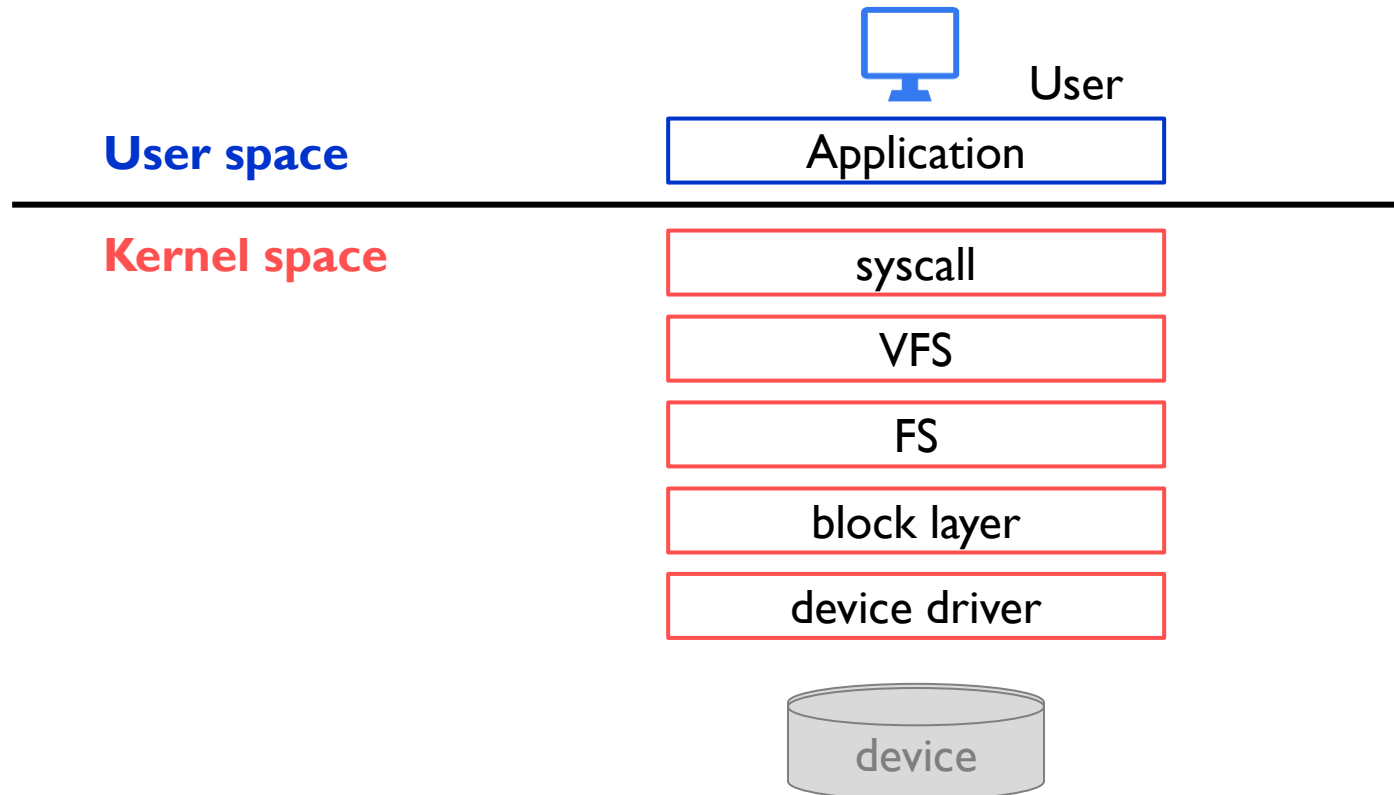
- **Kernel Code의 구성**

- Kernel code의 대부분이 C 언어로 작성됨
 - Architecture에 의존적인 code의 경우 assembly 언어로 작성
 - Inline assembly 및 Macro 포함

Linux Kernel (cont'd)

- **Kernel space and User space**

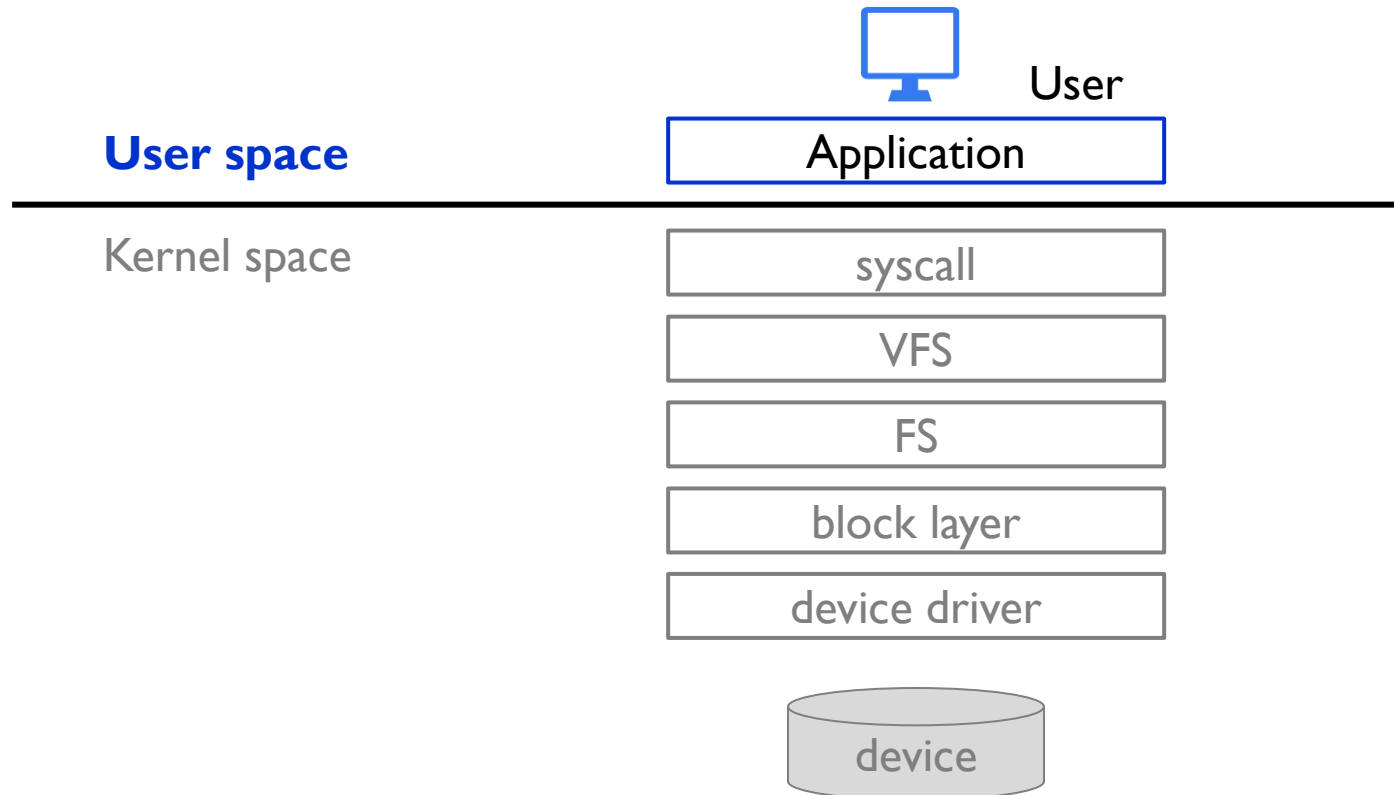
- OS와 사용자 program이 동시에 memory에 존재
- 효율적인 관리를 위해 memory를 kernel space와 user space로 분리
- 사용자 program이 kernel space를 침범 시 문제 발생
 - Kernel space에 접근할 수 있는 권한 부여



Linux Kernel (cont'd)

- **User space**

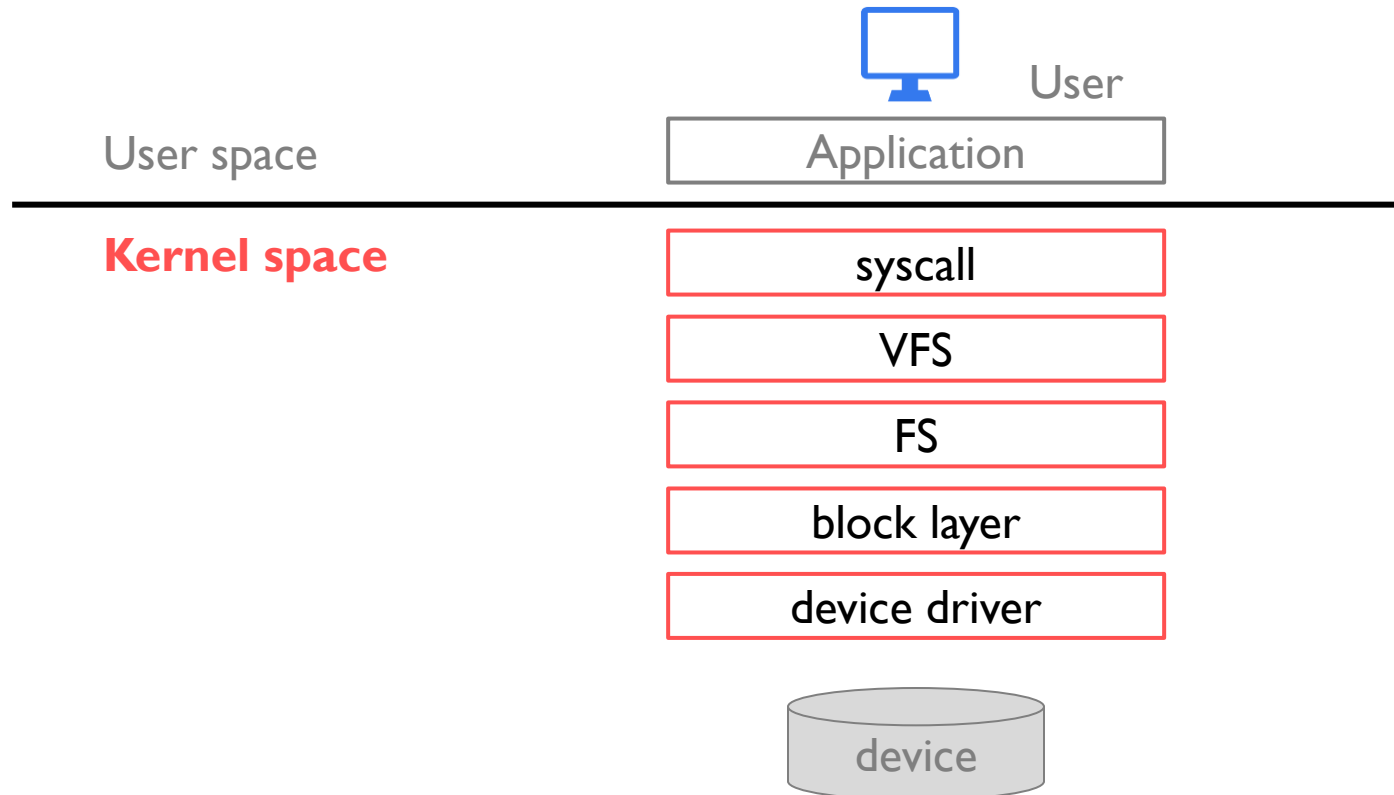
- 하나의 process가 동작하기 위해 사용되는 memory space
- System call 을 통해 kernel에 service 요청



Linux Kernel (cont'd)

- **Kernel space**

- OS라는 하나의 program을 실행하기 위한 memory space
- User space에서는
 - privilege level을 부여 받아 kernel space에 접근 가능
 - system call을 이용하여 kernel space에 접근 가능



Booting Process

1. 전원 인가

2. BIOS 실행

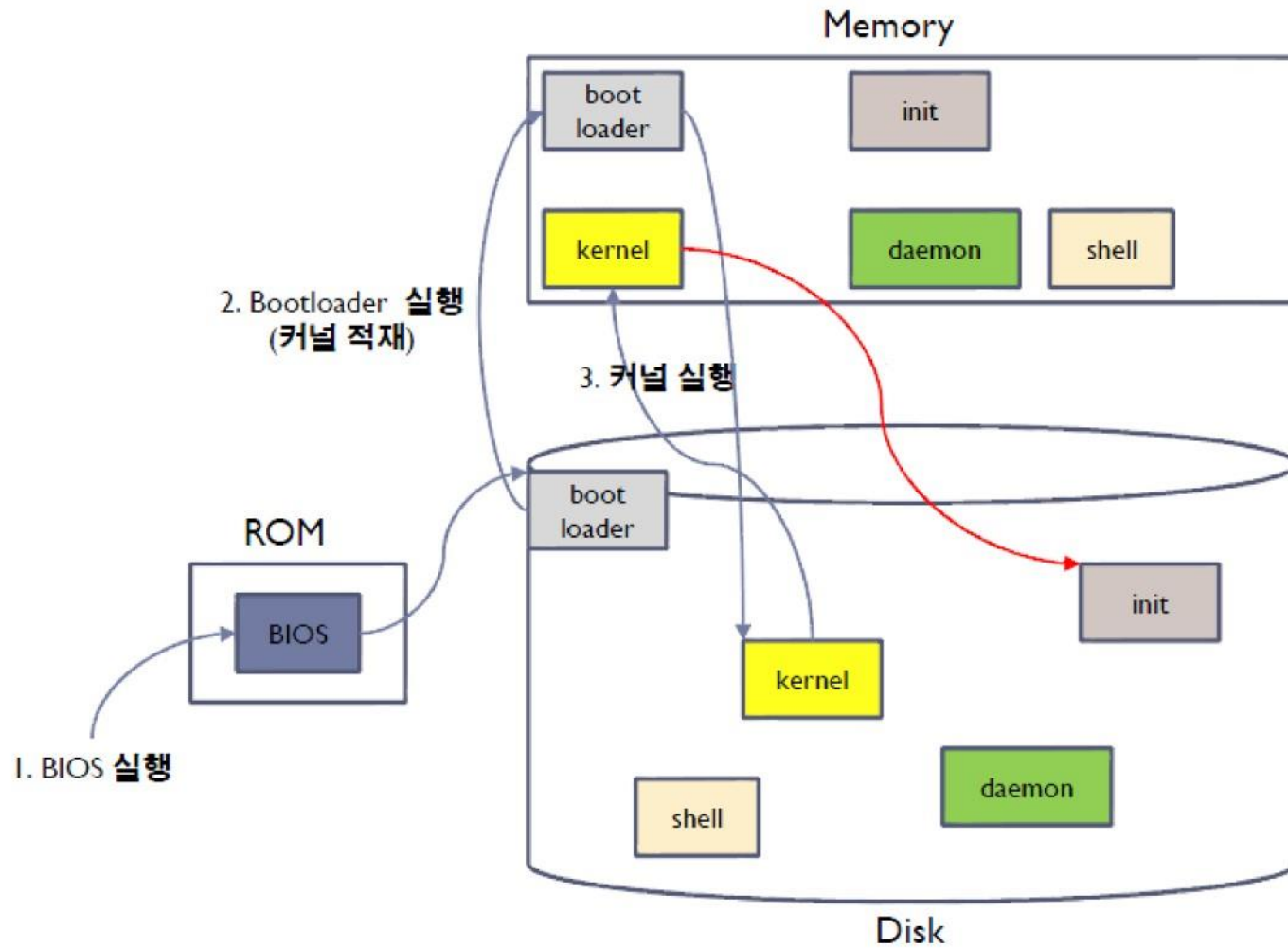
1. POST (Power On Self Test) 수행
 1. 장치 인식 및 물리적 손상 체크, 초기화 작업
2. 부팅매체 (HDD, CD-ROM, Floppy disk, etc..) 검색
3. 1번 부팅매체의 MBR(Master Boot Record)로부터 부트로더 (grub) 로드
4. 부트로더 실행

3. 부트로더 실행

1. 부팅메뉴에서 부트 커널 선택 (grub.conf)
2. 부트 커널 이미지 로드 (/boot/vmlinuz-4.19.67-OSLAB)
3. 커널 실행 (start_kernel())

4. 커널 실행

Booting Process (cont'd)



Download Kernel Source

- 참고: sudo 명령어

- 최고 관리자 (root) 권한으로 프로그램을 실행
- e.g.

```
sslab@ubuntu:~$ apt update → 권한이 없어 실패
Reading package lists... Done
W: chmod 0700 of directory /var/lib/apt/lists/partial failed - SetupAPTPartialDirectory (1
: Operation not permitted)
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
W: Problem unlinking the file /var/cache/apt/pkgcache.bin - RemoveCaches (13: Permission d
enied)
W: Problem unlinking the file /var/cache/apt/srcpkgcache.bin - RemoveCaches (13: Permissio
n denied)
sslab@ubuntu:~$
```

```
sslab@ubuntu:~$ sudo apt update → root 권한이 생겨 성공
[sudo] password for sslab: → 본인 계정 비밀번호 입력
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [99.8 kB]
Hit:2 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Get:3 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease [99.8 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease [99.8 kB]
Get:5 http://us.archive.ubuntu.com/ubuntu xenial/main amd64 Packages [61.2 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [61.2 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [61.2 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [61.2 kB]
Fetched 481 kB in 1s (481 kB/s)
Reading package lists... Done
```

```
sslab@ubuntu:~$ whoami
sslab
sslab@ubuntu:~$ sudo su → root 계정으로 전환
root@ubuntu:/home/sslab# whoami
root
root@ubuntu:/home/sslab# → 로그인 된 계정이 root로 변경 (최고 권한을 가지고 있으므로 사용에 유의)
```


Download Kernel Source (cont'd)

- **Kernel Source Download**

- <http://www.kernel.org>
- Our target kernel: 4.19.67

- **Download**

- \$ cd (/home/계정이름/Downloads)
 - 다른 디렉터리에 kernel 다운로드 가능
- \$ sudo wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.67.tar.xz # root 권한 필요

```
sslab@ubuntu:~$ cd /home/sslab/Downloads/
sslab@ubuntu:~/Downloads$ sudo wget http://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.67.tar.xz
--2022-08-29 01:21:10-- http://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.19.67.tar.xz
Resolving cdn.kernel.org (cdn.kernel.org)... 146.75.49.176, 2a04:4e42:7c::432
Connecting to cdn.kernel.org (cdn.kernel.org)|146.75.49.176|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 103291756 (99M) [application/x-xz]
Saving to: 'linux-4.19.67.tar.xz'

linux-4.19.67.tar.xz  100%[=====>] 98.51M  67.2MB/s  in 1.5s

2022-08-29 01:21:14 (67.2 MB/s) - 'linux-4.19.67.tar.xz' saved [103291756/103291756]

sslab@ubuntu:~/Downloads$
```

Compile Kernel (cont'd)

- Kernel 4.19.67 compile

- Kernel source 압축 해제
 - \$ tar -Jxvf linux-4.19.67.tar.xz

```
sslab@ubuntu:~/Downloads$ ls
linux-4.19.67.tar.xz
sslab@ubuntu:~/Downloads$ tar -Jxvf linux-4.19.67.tar.xz
```

- Kernel Extra Version 수정
 - \$ cd linux-4.19.67/
 - \$ vi Makefile 수정
 - '-본인 학번'으로 수정(assignment 1-2)

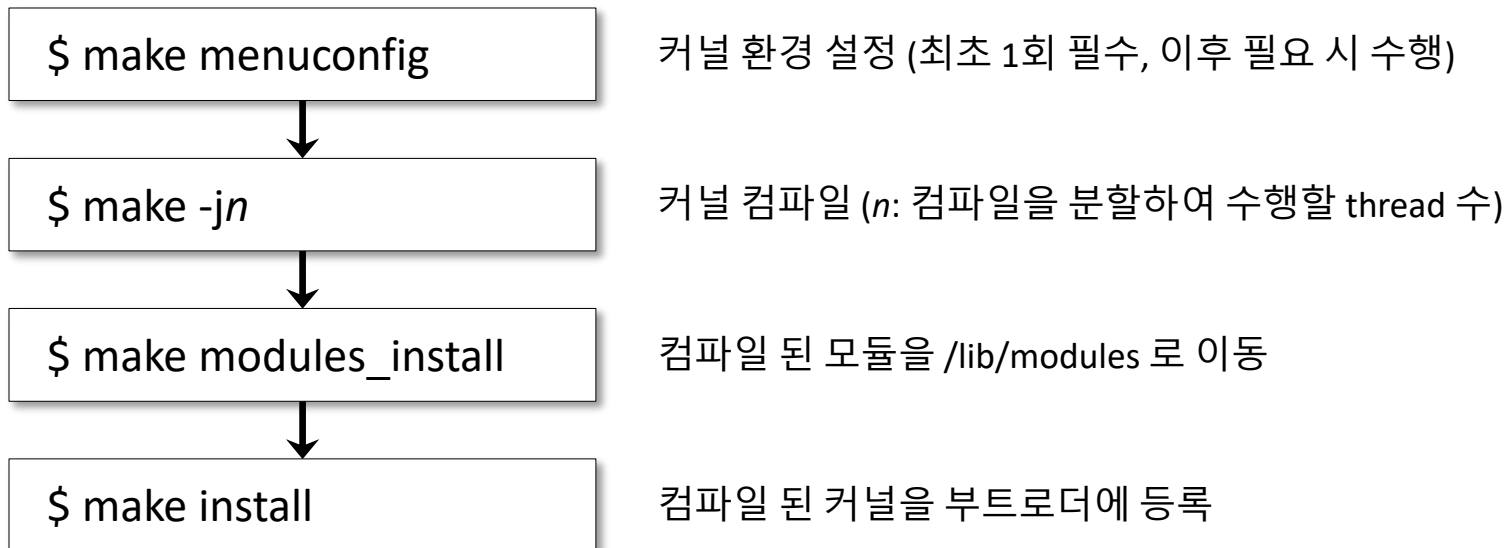
```
sslab@ubuntu:~/Downloads$ ls
linux-4.19.67 linux-4.19.67.tar.xz
sslab@ubuntu:~/Downloads$ cd linux-4.19.67/
sslab@ubuntu:~/Downloads/linux-4.19.67$ ls
arch  COPYING  Documentation  fs      ipc      kernel
block CREDITS  drivers        include Kbuild  lib
certs crypto  firmware      init    Kconfig LICENSE
sslab@ubuntu:~/Downloads/linux-4.19.67$ vi Makefile
sslab@ubuntu:~/Downloads/linux-4.19.67$
```

```
SPDX-License-Identifier: GPL-2.0
VERSION = 4
PATCHLEVEL = 19
SUBLEVEL = 67
EXTRAVERSION = -본인 학번 수정
NAME = "People's Front"
```

[Makefile]

Compile Kernel

■ 전체 과정



Compile Kernel (cont'd)

- Kernel 환경 설정

- Download
- \$ sudo apt install build-essential libncurses5-dev bison flex libssl-dev libelf-dev

```
sslab@ubuntu:~/Downloads/linux-4.19.67$ sudo apt install build-essential libncurses5-dev bison flex libssl-dev libelf-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.1ubuntu2).
The following package was automatically installed and is no longer required:
  snapd-login-service
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libbison-dev libfl-dev libsigsegv2 libssl-doc libtinfo-dev m4 zlib1g-dev
Suggested packages:
  bison-doc ncurses-doc
The following NEW packages will be installed:
  bison flex libbison-dev libelf-dev libfl-dev libncurses5-dev libsigsegv2 libssl-dev libssl-doc
  libtinfo-dev m4 zlib1g-dev
0 upgraded, 12 newly installed, 0 to remove and 99 not upgraded.
Need to get 4,006 kB of archives.
After this operation, 15.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] y 입력 후 enter
```

Compile Kernel (cont'd)

- Kernel 환경 설정
 - \$ sudo make menuconfig

```
.config - Linux/x86 4.19.67-2017123456 Kernel Configuration

Linux/x86 4.19.67-2017123456 Kernel Configuration
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
[ ] excluded <M> module < > module capable

*** Compiler: gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609 ***
General setup --->
[*] 64-bit kernel
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
Firmware Drivers --->
[*] Virtualization --->
General architecture-dependent options --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
Executable file formats --->
Memory Management options --->
[*] Networking support --->
Device Drivers --->
File systems --->
Security options --->
-*- Cryptographic API --->
Library routines --->
Kernel hacking --->

<Select> < Exit > < Help > < Save > < Load >
```

Compile Kernel (cont'd)

Kernel 환경 설정

- (1) 커널 모듈 적재 시 발생할 수 있는 문제 해결
 - "Enable loadable module support" → "Forced module loading" 체크

```
(-)
  Binary Emulations --->
  Firmware Drivers --->
  [*] Virtualization --->
  General architecture-dependent options --->
  [*] Enable loadable module support ---> → 엔터
  [*] Enable the block layer --->
  Executable file formats --->
  Memory Management options --->
  ↓(+)

```

```
--- Enable loadable module support
[*] Forced module loading → 스페이스 입력 시 [ ] 이 [*] 로 변환 (설정 적용)
[*] Module unloading
[ ] Forced module unloading
[ ] Module versioning support
[*] Source checksum for all modules
[*] Module signature verification
[ ] Require modules to be validly signed
  ↓(+)

```

- ESC 두 번 연속 입력 시 이전 메뉴로 복귀

Compile Kernel (cont'd)

Kernel 환경 설정

- (2) 컴파일 시 문제가 될 수 있는 모듈 제거
 - "Device Drivers" → "Staging drivers" 체크 해제

```
(-)  
[*] Enable loadable module support --->  
[*] Enable the block layer --->  
    Executable file formats --->  
    Memory Management options --->  
[*] Networking support --->  
Device Drivers ---> → 엔터  
    File systems --->  
    Security options --->  
└(+)
```

```
(-)  
[*] Virtio drivers --->  
    Microsoft Hyper-V guest support --->  
    Xen driver support --->  
[ ] Staging drivers ---- → 스페이스 입력 시 [*] 이 [ ] 로 변환 (설정 해제)  
-*- X86 Platform Specific Device Drivers --->  
[ ] Platform support for Goldfish virtual devices ----  
-*- Platform support for Chrome hardware --->  
[ ] Platform support for Mellanox hardware ----  
└(+)
```

- ESC 두 번 연속 입력 시 이전 메뉴로 복귀

Compile Kernel (cont'd)

Kernel 환경 설정

- (3) 설정을 파일 (.config)에 저장
 - 방향키 중, 우측 키 세 번을 눌러 "< Save >"로 이동 및 엔터 입력

```
Linux/x86 4.19.67-2017123456 Kernel Configuration
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 [ ] excluded <M> module < > module capable

*** Compiler: gcc (Ubuntu 5.4.0-6ubuntu1~16.04.10) 5.4.0 20160609 ***
General setup --->
[*] 64-bit kernel
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
Firmware Drivers --->
⏴(+)
```

<Select> <Exit> <Help> **<Save>** <Load>

- "< Ok >" → "< Exit >" 입력으로 환경 설정 완료

Compile Kernel (cont'd)

- Kernel 환경 설정

- (4) Kernel 환경 설정 나가기

- 방향키 중, 우측 키 두 번을 눌러 "< Exit >"로 이동 및 엔터 입력

```
Linux/x86 4.19.67-SSLAB Kernel Configuration
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:

*** Compiler: gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.
General setup --->
[*] 64-bit kernel
Processor type and features --->
Power management and ACPI options --->
Bus options (PCI etc.) --->
Binary Emulations --->
Firmware Drivers --->
[*] Virtualization --->
General architecture-dependent options --->
[*] Enable loadable module support --->
└(+)
```

<Select> < Exit > < Help > < Save > < Load >

```
scripts/kconfig/mconf Kconfig
```

```
*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.
```

Compile Kernel (cont'd)

- **Kernel Compile**

- \$ make -j n
 - n : 컴파일을 나누어 수행할 thread 수
 - 가상 머신에 할당한 CPU core 수의 1.5배 ~ 2배 정도면 적합

- **Module install**

- \$ make modules_install → root 권한일 때
- \$ sudo make modules_install → root 권한이 아닐 때

- **Compile된 Kernel을 Boot Loader에 등록**

- \$ make install → root 권한일 때
- \$ sudo make install → root 권한이 아닐 때
- kernel Image(/arch/x86/boot/bzImage)를 /boot로 복사
- System Map(System.map)을 /boot로 복사
- Grub 부트 로더에 자동 등록

Compile Kernel (cont'd)

- **Grub 설정**

- Grub
 - 대부분의 리눅스 배포판에서 사용하는 부트로더
- Grub의 실행과 역할
 - 실행과 동시에 grub 설정 파일을 읽고 boot menu를 화면에 출력
 - 사용자가 선택한 kernel image를 메모리에 적재 및 제어권을 커널에 전달
- Grub 설정 파일
 - GRUB_DEFAULT : 부팅할 기본 커널 이미지
 - GRUB_TIMEOUT : 사용자의 선택을 기다리는 시간
 - GRUB_HIDDEN_TIMEOUT_QUIET : Grub 메뉴를 숨김 ("true" 시)
 - GRUB_HIDDEN_TIMEOUT : Grub 메뉴를 숨기고 기다리는 시간

```
GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
```

Compile Kernel (cont'd)

- Grub 설정

- Grub 설정 파일 수정

- \$ sudo vi /etc/default/grub → root 권한일 때
 - \$ vi /etc/default/grub → root 권한이 아닐 때

```
GRUB_DEFAULT=0
GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
```



```
GRUB_DEFAULT=0
#GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=false
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""
```

Compile Kernel (cont'd)

- 재부팅
 - \$ reboot
- Grub 부트로더 선택 메뉴에서 컴파일한 커널 선택
- 커널 버전 확인
 - \$ uname -r

```
sslab@ubuntu:~$ uname -r  
4.19.67-SSLAB  
sslab@ubuntu:~$
```

-본인학번

- assignment 1-2 과제

2021년 2학기 운영체제실습 2주차

Programming Tools part 2

System Software Laboratory

School of Computer and Information Engineering

Kwangwoon Univ.

Contents

- **ctags /cscope**
 - 실습: ctags, cscope 사용

ctags

정의

- Generate tag file for source code
- Source code의 tag(global variable, function, macro...)들의 정보를 담고 있는 database를 생성하는 명령어

사용

- 생성된 tag DB는 vim, emacs같은 editor에서 symbol 검색에 이용
- Installation
 - \$ sudo apt install exuberant-ctags

```
sslab@ubuntu:~/Downloads/linux-4.19.67$ sudo apt install exuberant-ctags
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  snapd-login-service
Use 'sudo apt autoremove' to remove it.
The following NEW packages will be installed:
  exuberant-ctags
0 upgraded, 1 newly installed, 0 to remove and 99 not upgraded.
Need to get 126 kB of archives.
After this operation, 341 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu xenial/main amd64 exuberant-ctags amd64
1:1.5.0-20110312-11.5126-1.51
```


ctags

- 사용

- usage

- DB 생성

- \$ ctags -R

- (-R: Recurse into directories encountered in the list of supplied files)

- e.g.

```
sslab@ubuntu:~/Downloads/linux-4.19.67$ sudo ctags -R
sslab@ubuntu:~/Downloads/linux-4.19.67$ ls
arch          drivers      lib          README      virt
block         firmware    LICENSES     samples     vmlinux
built-in.a    fs          MAINTAINERS  scripts     vmlinux-gdb.py
certs         include     Makefile     security     vmlinux.o
COPYING       init        mm           sound
CREDITS       ipc         modules.builtin System.map
crypto        Kbuild     modules.order tags
cscope.out    Kconfig    Module.symvers tools
Documentation kernel      net          usr
```

ctags

- **vi(m) editor와 연동**

- tag를 이용한 symbol 탐색과 함께 vi 시작하기
- \$ vi -t [keyword]
- e.g. struct task_struct가 있는 곳부터 vi를 시작 시

```
sslslab@ubuntu: ~/Downloads/linux-4.19.67
sslslab@ubuntu:~/Downloads/linux-4.19.67$ sudo vi -t task_struct

enum perf_event_task_context {
    perf_invalid_context = -1,
    perf_hw_context = 0,
    perf_sw_context,
    perf_nr_task_contexts,
};

struct wake_q_node {
    struct wake_q_node *next;
};

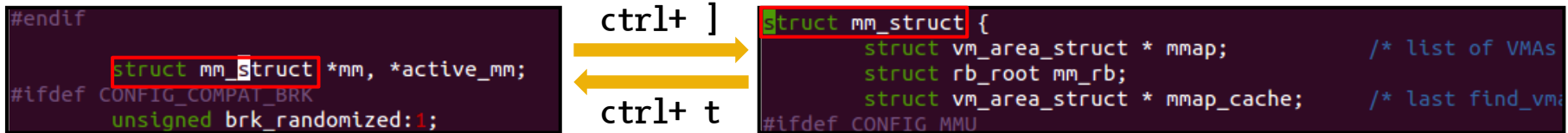
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
    /*
     * For reasons of header soup (see current_thread_info()), this
     * must be the first element of task_struct.
     */
    struct thread_info          thread_info;
#endif
    /* -1 unrunnable, 0 runnable, >0 stopped: */
    volatile long               state;

    /*
     * "include/linux/sched.h" 1911L, 53889C
     */
}
```

ctags

Tag 이동

- `ctrl +]` 를 입력하면 tags 파일에서 해당 tag 정보를 찾아서 해당 파일로 이동
- `ctrl + t` 를 입력하면 이전 위치로 이동
- e.g.



Tag 검색

- vi 명령모드에서 `:ta [keyword]` 로 이동
- e.g. `:ta schedule` 또는 `:tag schedule`

```
        unsigned long pgoff, unsigned long flags);
void (*unmap_area) (struct mm_struct *mm, unsigned long addr);
#endif
    unsigned long mmap_base;           /* base of mmap area */
    unsigned long task_size;           /* size of task vm space */
:ta schedule
```

- `:ta` : 첫 번째로 검색된 결과로 이동
- `:tn` : 두 번째로 일치하는 결과로 이동
- `:tp` : 이전 결과로 이동

ctags

- Tag 목록

- 검색 결과를 한 눈에 보고 원하는 결과를 선택 → [:ts \[keyword\]](#)
 - E.g. :ts schedule 입력 시,

```
      unsigned long mmap_base;          /* base of mmap area */
      unsigned long task_size;          /* size of task vm space */
# pri kind tag file
1 F m schedule drivers/usb/host/isp116x.h
    struct:isp116x_ep typeref:struct:isp116x_ep::list_head
    struct list_head schedule;
2 F m schedule drivers/usb/host/isp1362.h
    struct:isp1362_ep typeref:struct:isp1362_ep::list_head
    struct list_head schedule; /* list of all EPs that need processing */
3 F m schedule drivers/usb/host/sl811.h
    struct:sl811h_ep typeref:struct:sl811h_ep::list_head
    struct list_head schedule;
4 F m schedule include/net/ip_vs.h
    struct:ip_vs_scheduler typeref:struct:ip_vs_scheduler::schedule
    struct ip_vs_dest* (*schedule)(struct ip_vs_service *svc,
5 F v schedule kernel/sched.c
    EXPORT_SYMBOL(schedule);
6 F f schedule kernel/sched.c
    asmlinkage void __sched schedule(void)
7 FS m schedule drivers/scsi/ncr53c8xx.c
    struct:launch typeref:struct:launch::link
    struct link schedule; /* Jump to scheduler point */
Type number and <Enter> (empty cancels):
```

- q 입력 후, 해당 index 입력

ctags

ctags 사용

- vi와의 연동 1 : tags 생성 후
 - \$ ctags -R 명령어를 수행한 디렉토리에서 vi를 실행
 - 즉, tags 파일이 생성되어 있는 디렉토리

vi와의 연동 2 : tag 이동

명령	설명
: ta [g] { ident }	{ ident } 가 정의된 위치로 이동
<Ctrl+>	': ta' 와 동일, 커서가 위치한 keyword 에 대하여 이동
visual <Ctrl+>	': ta' 와 동일, 선택된 keyword 에 대하여 이동

vi와의 연동 3 : tag 스택

명령	설명
<Ctrl+T>	이전 Tag 로 이동
: [count] po [p]	[count] 만큼 이전 Tag 로 이동
: [count] ta [g]	[count] 만큼 다음 Tag 로 이동
: tags	Tag 스택의 목록 출력
: Otag	마지막 사용한 Tag 로 이동

vi와의 연동 4 : 기타 명령어

명령	설명
: ts [elect] [ident]	[ident] 와 일치하는 Tags 목록을 출력하고 선택 후 이동
g]	': ts' 와 동일, 커서가 위치한 keyword 에 대하여 이동
visual g]	': ts' 와 동일, 선택된 keyword 에 대하여 이동
: tj [ump] [ident]	': ts' 와 동일, 단일 Tag 일 경우 ': ta' 와 동일
g <Ctrl+>	': tj' 와 동일, 커서가 위치한 keyword 에 대하여 이동
visual g <Ctrl+>	': tj' 와 동일, 선택된 keyword 에 대하여 이동
: [count] tn [ext]	일치하는 다음 Tag 로 이동
: [count] tp [revious]	일치하는 이전 Tag 로 이동
: [count] tN [ext]	': tp' 와 동일
: [count] tr [ewind]	[count] 번째 위치로 이동, [count] 가 없으면 첫번째로 일치하는 Tag 로 이동
: [count] tf [irst]	': tr' 와 동일
tl [ast]	마지막으로 일치하는 Tag 로 이동
: sts [elect] [ident]	': ts' 와 동일, 선택된 Tag 는 윈도우 분할
: stj [ump] [ident]	': tj' 와 동일, 선택된 Tag 는 윈도우 분할

cscope

정의

- ctags 보다 다양한 검색 타입 지원
- ctags 만으로 검색하기 힘든 변수나 함수를 찾기 위해 이용
- cscope 실행 시 database(cscope.out) 구축

사용

설치

- `$ apt install cscope`

```
sslab@ubuntu:~/Downloads/linux-4.19.67$ sudo apt install cscope
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer
required:
  snapd-login-service
Use 'sudo apt autoremove' to remove it.
```

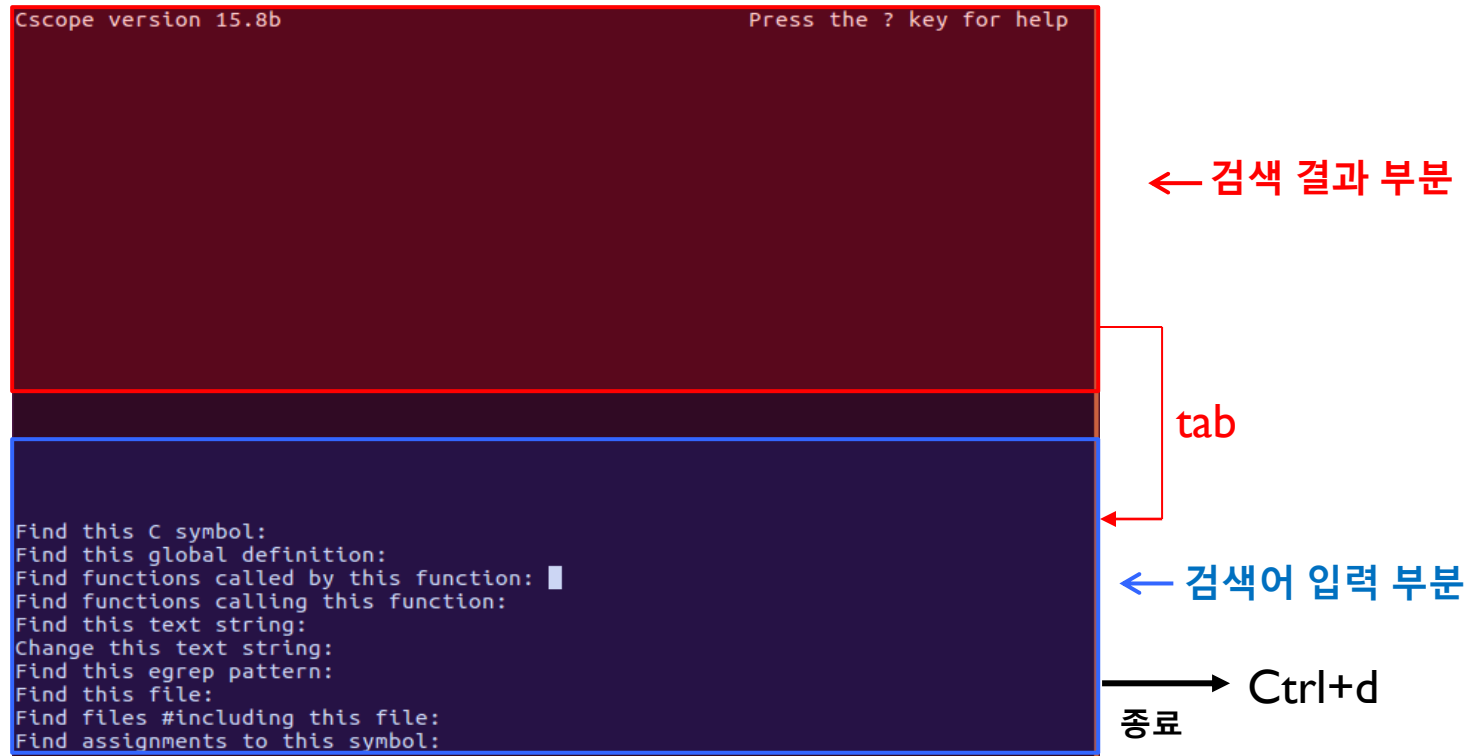
사용

- DB 구축 및 실행
 - `$ cscope -R`
 - -R : Recurse subdirectories during search for source files
 - DB파일을 cscope가 직접 생성

cscope

- cscope의 사용 (cont'd)

- cscope 실행 후 원하는 검색어 입력 가능



- 해당 소스를 선택하면 vim을 사용해 해당 코드로 이동
- vim에서 ctags 명령어 및 단축키 사용 가능
- vim 종료 시 cscope로 돌아오며, cscope 종료 단축키는 ctrl+d

cscope

- vim 명령모드 내에서 cscope 명령 형식

- :cs find '질의 종류' '심볼'

- 질의 종류

질의 종류	의미
0 or s	Find this C symbol
1 or g	Find this definition
2 or d	Find function called by this function
3 or c	Find function calling this function
4 or t	Find assignments to
6 or e	Find this egrep pattern
7 or f	Find this file
8 or i	Find files #including this file

- e.g. :cs find s printf

E567: no cscope connections

- :cscope add cscope.out
 - :cs find s printf

ctags & cscope 실습(1/2)

- start_kernel() 을 찾아 커널 메시지로 학번 출력

step 1. cscope 실행 후 "start_kernel"검색

```
Find this C symbol: start_kernel
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:
```

```
C symbol: start_kernel

File           Function           Line
0 process.c     <global>           16 extern void start_kernel(void );
1 start_kernel.h <global>           11 extern asmlinkage void __init start_kernel(void );
2 bootp.c       start_kernel       135 start_kernel(void )
3 bootpz.c      start_kernel       263 start_kernel(void )
4 main.c        start_kernel       152 void start_kernel(void )
5 relocate.c    relocate_kernel    305 void *kernel_entry = start_kernel;
6 relocate.c    relocate_kernel    399 kernel_entry = RELOCATED(start_kernel);
7 setup.c       start_parisc       425 start_kernel();
8 setup_32.c    sparc32_start_kernel 294 start_kernel();
9 setup_64.c    start_early_boot   386 start_kernel();
a process.c     start_kernel_proc   28 start_kernel();
b head32.c      i386_start_kernel   56 start_kernel();
c head64.c      x86_64_start_reservations 472 start_kernel();
d main.c        start_kernel       531 asmlinkage __visible void __init start_kernel(void )
```

ctags & cscope 실습(1/2)

- start_kernel() 을 찾아 커널 메시지로 학번 출력

step 2. printk 코드 입력 후 수정

printk(KERN_INFO "학번 in start_kernel()\n");

```
asmlinkage __visible void __init start_kernel(void)
{
    char *command_line;
    char *after_dashes;

    set_task_stack_end_magic(&init_task);
    smp_setup_processor_id();
    debug_objects_early_init();

    cgroup_init_early();

    local_irq_disable();
    early_boot_irqs_disabled = true;

    printk(KERN_INFO "SSLAB in start_kernel()\n");
}
```

본인 학번

ctags & cscope 실습(2/2)

- `start_kernel()` 을 찾아 커널 메시지로 학번을 출력

step 3. module compile 후 reboot

```
$ make
```

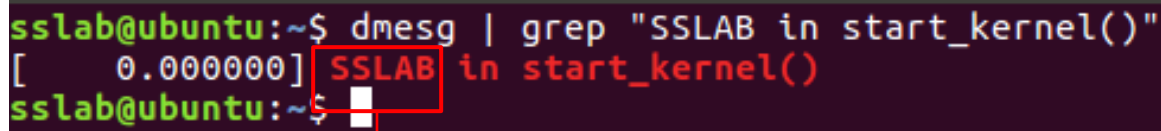
```
$ make modules_install
```

```
$ make install
```

```
$ reboot
```

step 4. dmesg 로 확인

```
$ dmesg | grep "학번 in start_kernel()"
```



```
sslab@ubuntu:~$ dmesg | grep "SSLAB in start_kernel()"  
[ 0.000000] SSLAB in start_kernel()  
sslab@ubuntu:~$
```

본인 학번

Assignment 1

- Assignment #1
 - 제출 기한: 2022. 09.01(목) ~ 2022.09.22(목) 23:59:59
 - Delay 없음
 - 업로드 양식에 어긋날 경우 감점 처리
 - Hardcopy 제출하지 않음