

리눅스 시스템 프로그래밍 for 서강대학교

Release 2017-07-03

커널연구회(www.kernel.bz)

정재준(rgbi3307@nate.com)

목차

내용

리눅스 시스템 프로그래밍 FOR 서강대학교.....	1
목차	2
1. 강사 및 커널연구회 소개.....	3
1.1 강사 소개	3
1.2 커널연구회 소개 (로드맵)	4
1.3 커널연구회 교육과정	5
1.4 커널연구회 최근활동	6
2. 개발환경.....	8
2.1 통합개발환경(CODEBLOCKS)	8
3. 리눅스 시스템 프로그래밍.....	11
3.1 SYSTEM CALL	12
3.2 동기화(LOCKING).....	15
3.3 메모리 맵	17
4. 리눅스 커널.....	20
4.1 커널소스 다운로드.....	21
4.2 커널소스 빌드.....	23
4.2.1 라즈베리파이 보드에서 직접 빌드.....	24
4.2.2 리눅스 Host PC에서 크로스 컴파일.....	24
4.3 커널 설치	25
4.4 커널소스 디버깅(KGDB).....	28
4.5 디바이스드라이버	36

1. 강사 및 커널연구회 소개

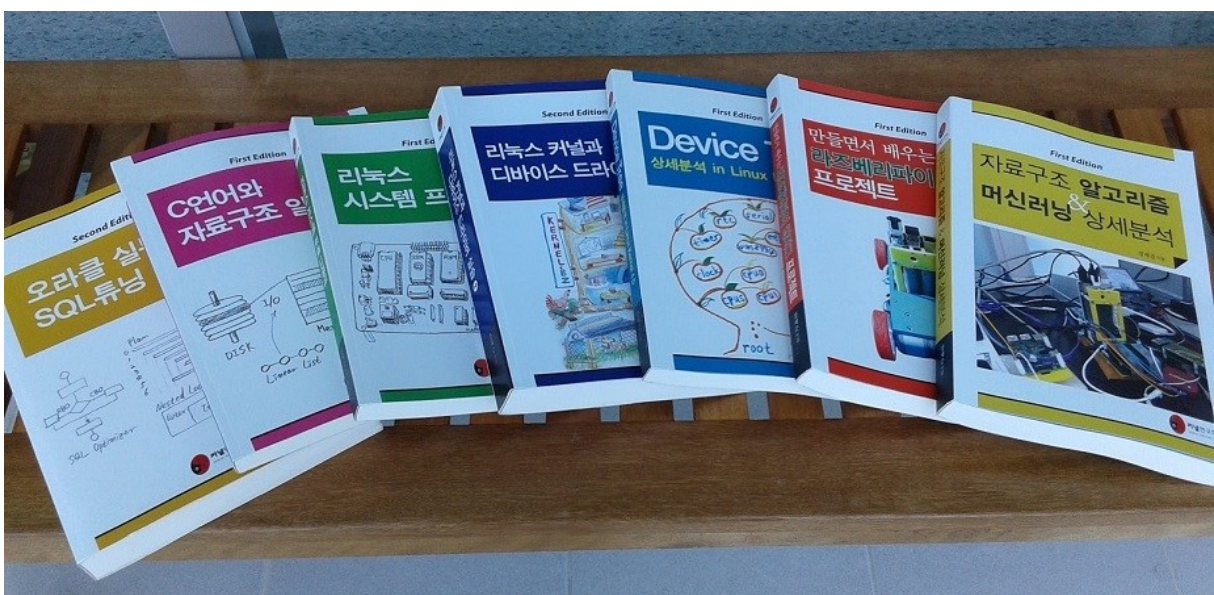
1.1 강사 소개



정재준 (rgbi3307@nate.com) / 커널연구회(www.kernel.bz)

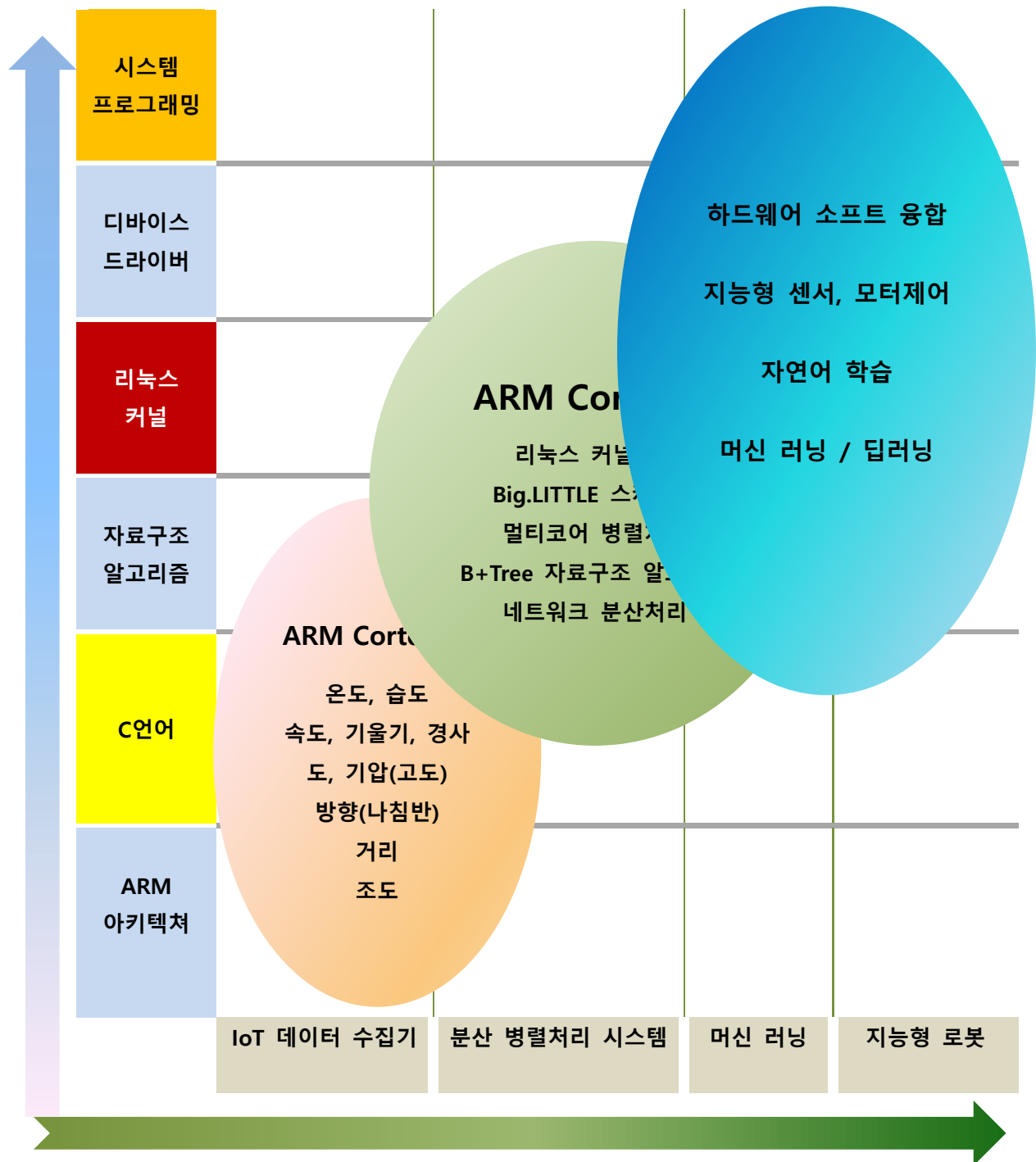
저자는 학창시절 마이크로프로세서 제어 기술을 배웠으며 리눅스 커널을 연구하고 있다. 15년 이상 쌓아온 실무 경험을 바탕으로 "C언어와 자료구조 알고리즘", "리눅스 시스템 프로그래밍", "리눅스 커널과 디바이스드라이버 실습2", "자료구조 알고리즘 & 머신러닝 상세분석"등의 책을 집필하고, 월간임베디드월드 잡지에 다수의 글을 기고 하였다. 또한 "맞춤형 문장 자동 번역 시스템 및 이를 위한 데이터베이스 구축방법 (The System for the customized automatic sentence translation and database construction method)" 라는 내용으로 프로그래밍을 하여 특허청에 특허등록 하였다. 최근에는 서울시 버스와 지하철 교통카드 요금결재 단말기에 들어가는 리눅스 커널과 디바이스 드라이버 개발 프로젝트를 성공적으로 수행했고 여러가지 임베디드 제품을 개발했다. 저자는 스탠포드대학교의 John L. Hennessy 교수의 저서 "Computer Organization and Design" 책을 읽고 깊은 감명을 받았으며, 컴퓨터구조와 자료구조 알고리즘 효율성 연구를 통한 기술서적 집필을 해오고 있다. 저자는 커널연구회(www.kernel.bz) 웹사이트를 운영하며 연구개발, 교육, 관련기술 공유 등을 위해 노력하고 있다.

(집필서적들)



1.2 커널연구회 소개 (로드맵)

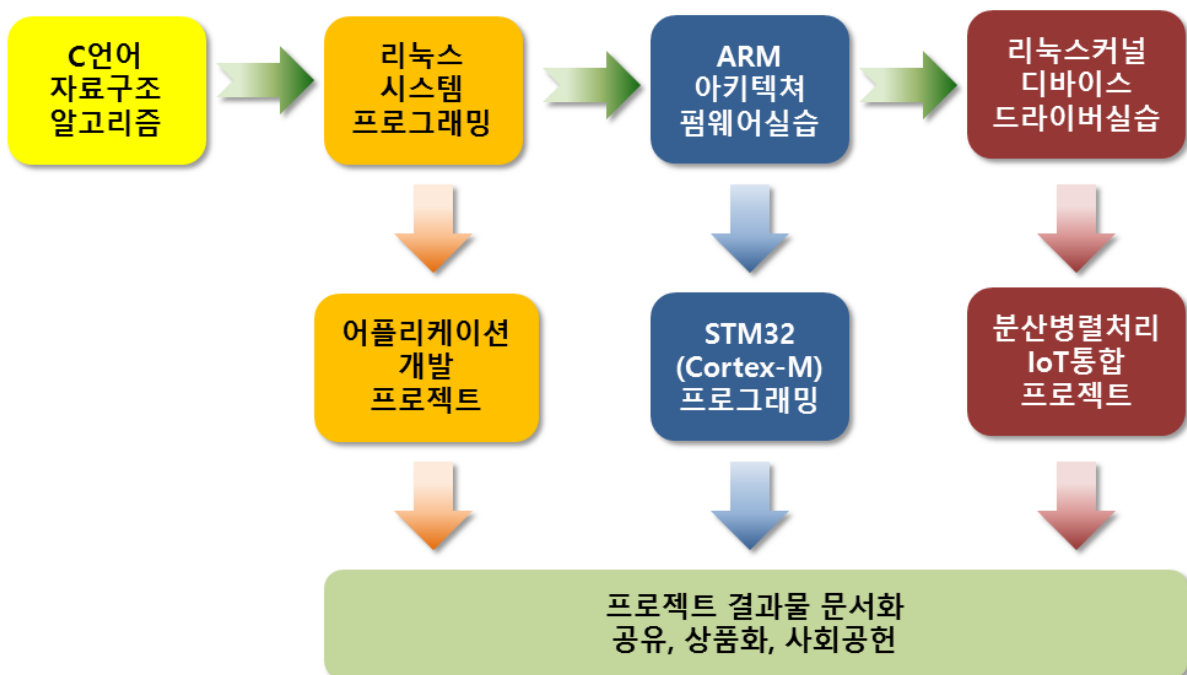
연구개발 및 교육



1.3 커널연구회 교육과정

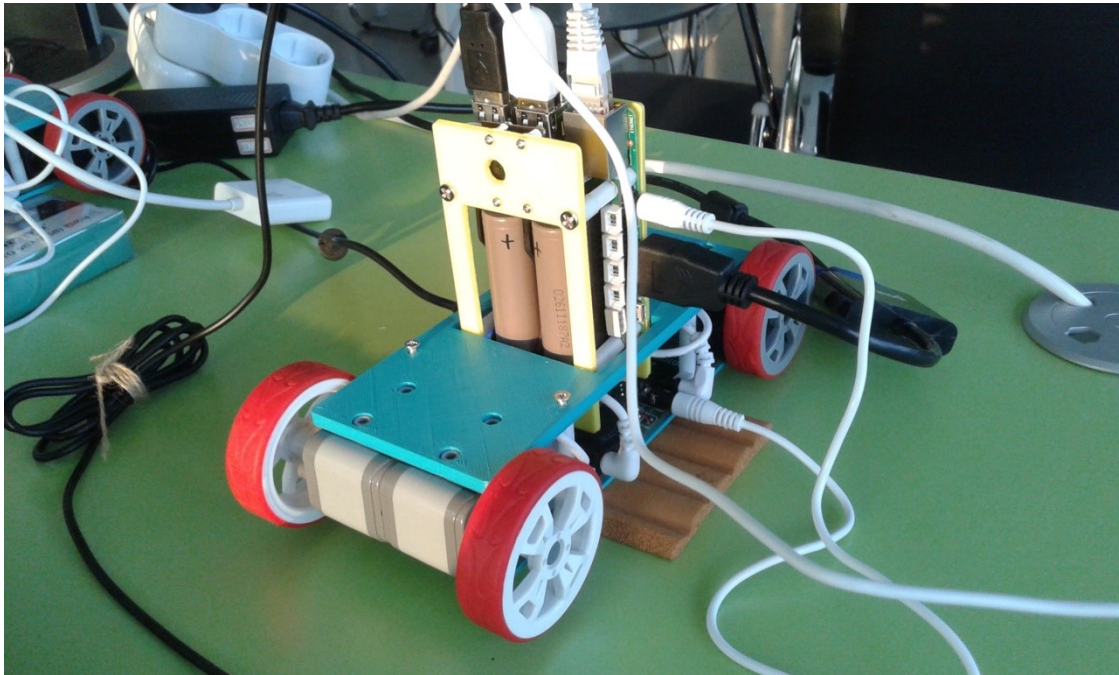


교육 로드맵



1.4 커널연구회 최근활동

만들면서 배우는 라즈베리파이 라지그 프로젝트



라지그 프로젝트 교육안내

만들면서 배우는 라즈베리파이 라지그 프로젝트는 커널연구회(www.kernel.bz)의 교육과정을 통해서 자세히 실습할 수 있다. 본 책자를 통해서 라지그 프로젝트를 수행하기 위한 기본에 대해서 학습할 수는 있으나 소프트웨어를 모두 설명한 것은 아니므로, 다음과 같은 커널연구회 교육에 참여하여 구현과정 전체를 배울 수 있다.

만들면서 배우는 라지그 프로젝트 교육내용

순번	교육 제목	상세 내용
1	라즈베리파이 개발환경	라즈베리파이 개발환경 설정방법 통합개발도구(Code Blocks) 설치 사운드, 비디오, USB WiFi 등글, 카메라 모듈 모뎀 GPIO 헤더핀맵 이해
2	라눅스 시스템 프로그래밍	오픈소스 사용법 설명 카메라 제어 프로세스 코딩(쓰레드, 데몬, 시그널) 카메라 동작감시 프로세스 코딩(쓰레드, 데몬, 시그널) 음악 및 동영상 재생 프로세스 코딩(쓰레드, 데몬, 시그널) 인터넷 라디오 플래이 프로세스 코딩(쓰레드, 데몬, 시그널)
3	웹서버프로그래밍	아파치, PHP, MySQL 연동방법 PHP 웹페이지와 C언어 시스템 프로그래밍 연동방법 스마트폰과 USB WiFi 무선 인터페이스 방법
4	모터 제어 프로그래밍	모터 제어 보드의 이해 PWM 인터페이스 이해 및 코딩 스마트폰으로 모터제어 인터페이스 코딩 PHP와 C언어로 로봇제어 프로그래밍 코딩
5	센서 데이터 수집 프로그래밍	라지그 센서 보드의 이해 12C 인터페이스 이해 및 코딩 스마트폰으로 센서 데이터 수집 인터페이스 코딩 PHP와 C언어로 센서 데이터 통계 및 그래프 처리 가능 코딩

*자세한 사항은 커널연구회 웹사이트(www.kernel.bz)를 참조하기 바란다.



만들면서 배우는 라즈베리파이 라지그 프로젝트

정재준 지음

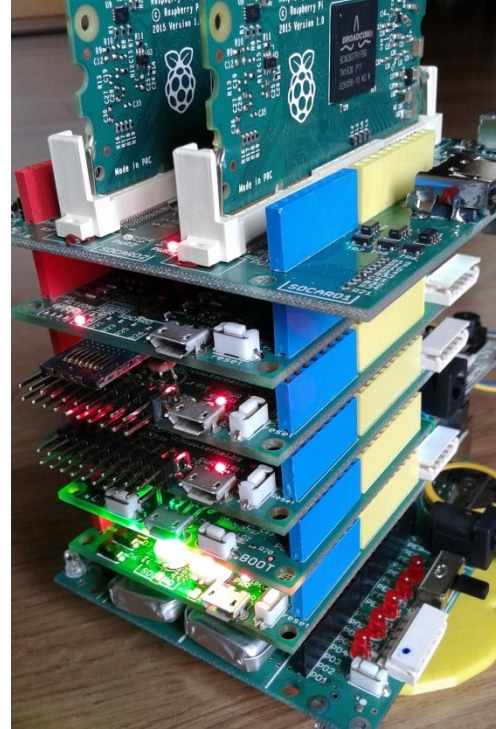
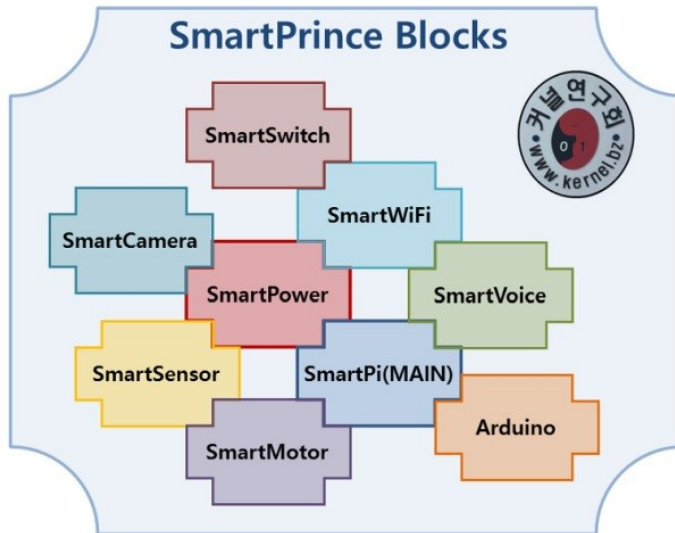
First Edition

만들면서 배우는
라즈베리파이 라지그
프로젝트

정재준 지음



개인용 머신러닝 로봇 모듈(SmartPrince, 똑똑한왕자)



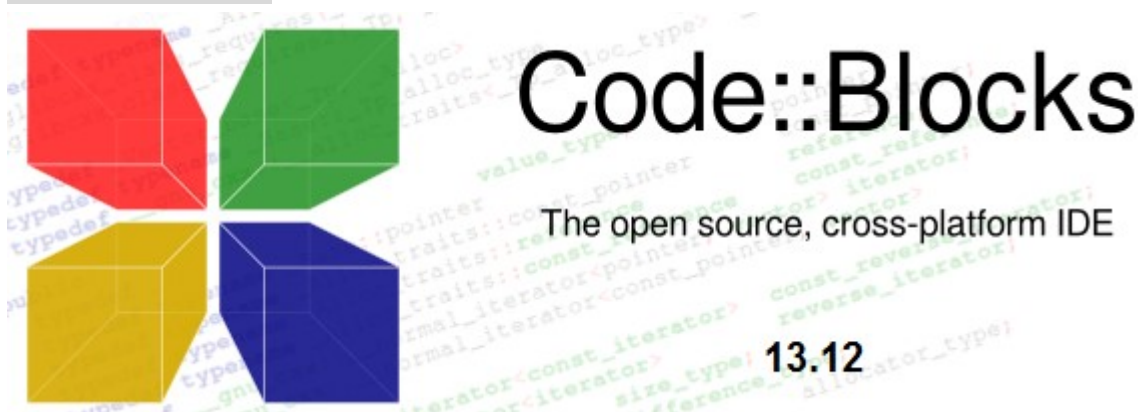
- 개인이 실생활 밀착형으로 사용할 수 있도록 저비용, 소형화, 모듈화.
- 외국의 대형서버급 기술들을 고비용으로 차용하지 않고 자체 기술로 내재화.
- 개인이 실생활에서 정서적으로 교감할 수 있는 애완동물 형태의 지능을 갖추도록 함.
- 개인 맞춤형 서비스, 정보제공(개인비서), 스마트홈과 연동하여 가정의 전자기기를 제어.
- 지능형으로 주변 데이터를 센싱하는 모듈(SmartSensor: 기울기, 방향, 거리, 속도, 온도, 습도)
- 센싱한 주변 데이터에 따라서 행동하는 모터 제어용 모듈(SmartMotor)
- 카메라영상을 시각화하여 사물을 구별하고 사물의 움직임을 추적하는 모듈(SmartCamera)
- 사운드 및 음성을 학습하여 사람의 명령에 따라서 반응하는 모듈(SmartVoice)
- 무선통신 및 인터넷에 접속하여 생활밀착형 정보들을 제공하는 모듈(SmartWiFi)
- 머신러닝/딥러닝 알고리즘을 최적화하기 위한 병렬처리 모듈(SmartPi)
- 배터리 충전 및 모듈별로 전원을 관리하고 스위칭하는 모듈(SmartPower, SmartSwitch)

2. 개발환경

2.1 통합개발환경(CodeBlocks)

GNU gcc 통합개발환경(IDE)인 Code Blocks 을 설치하면 X 윈도우 GUI 상에서 프로그램을 개발 및 디버깅할 수 있다. Qt 에 비해서 가볍게 콘솔 어플리케이션을 작성할 수 있으며 오픈소스이다.

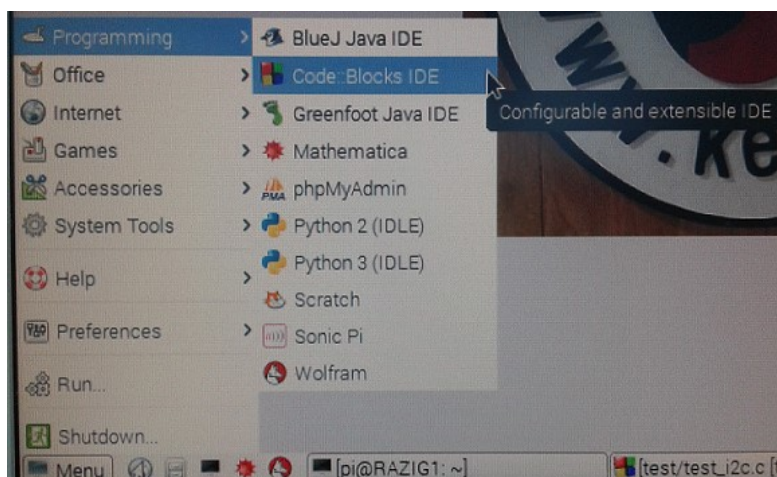
코드블락 스크린샷



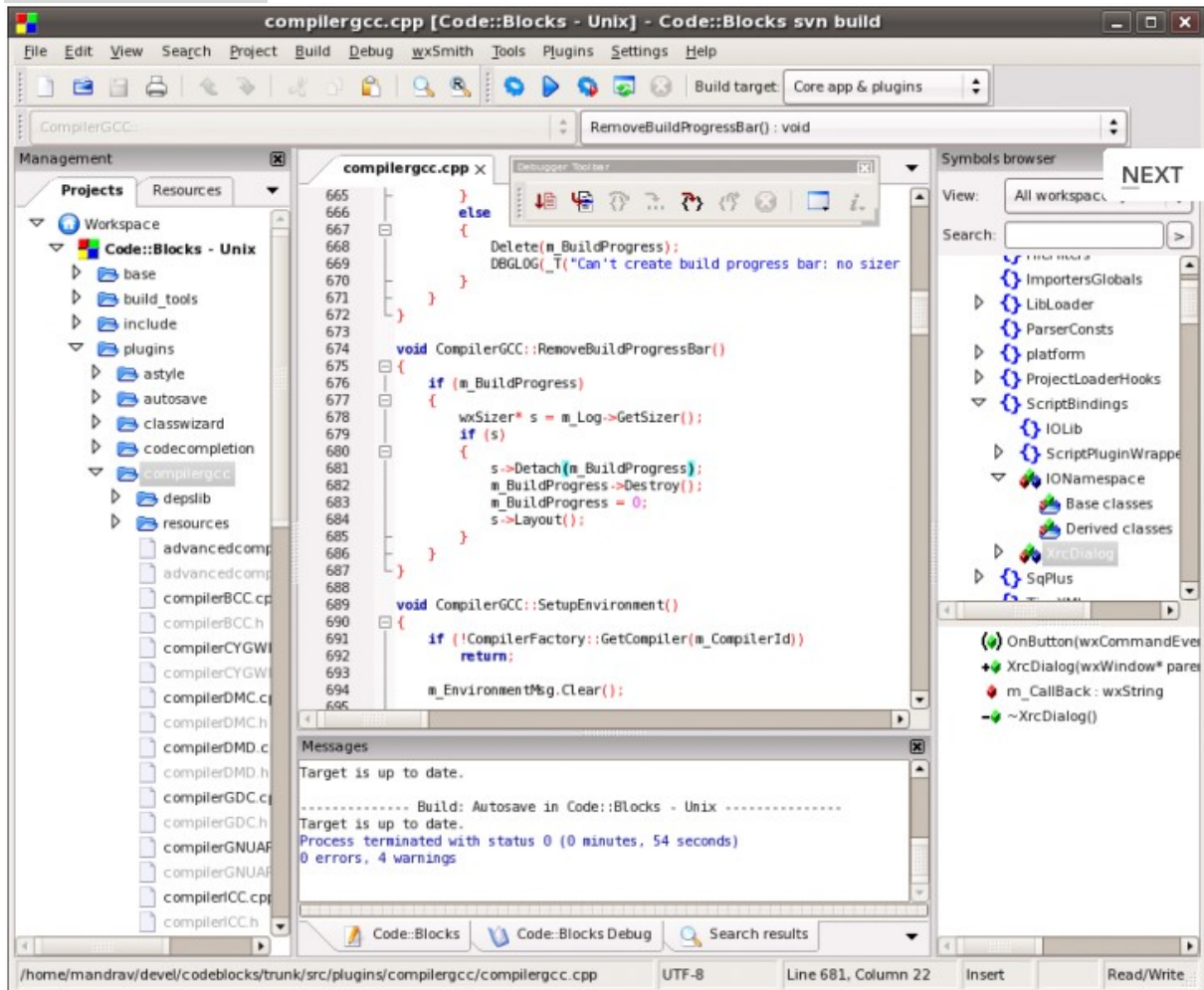
다음과 같이 apt-get 으로 설치한다.

```
$ sudo apt-get install codeblocks
```

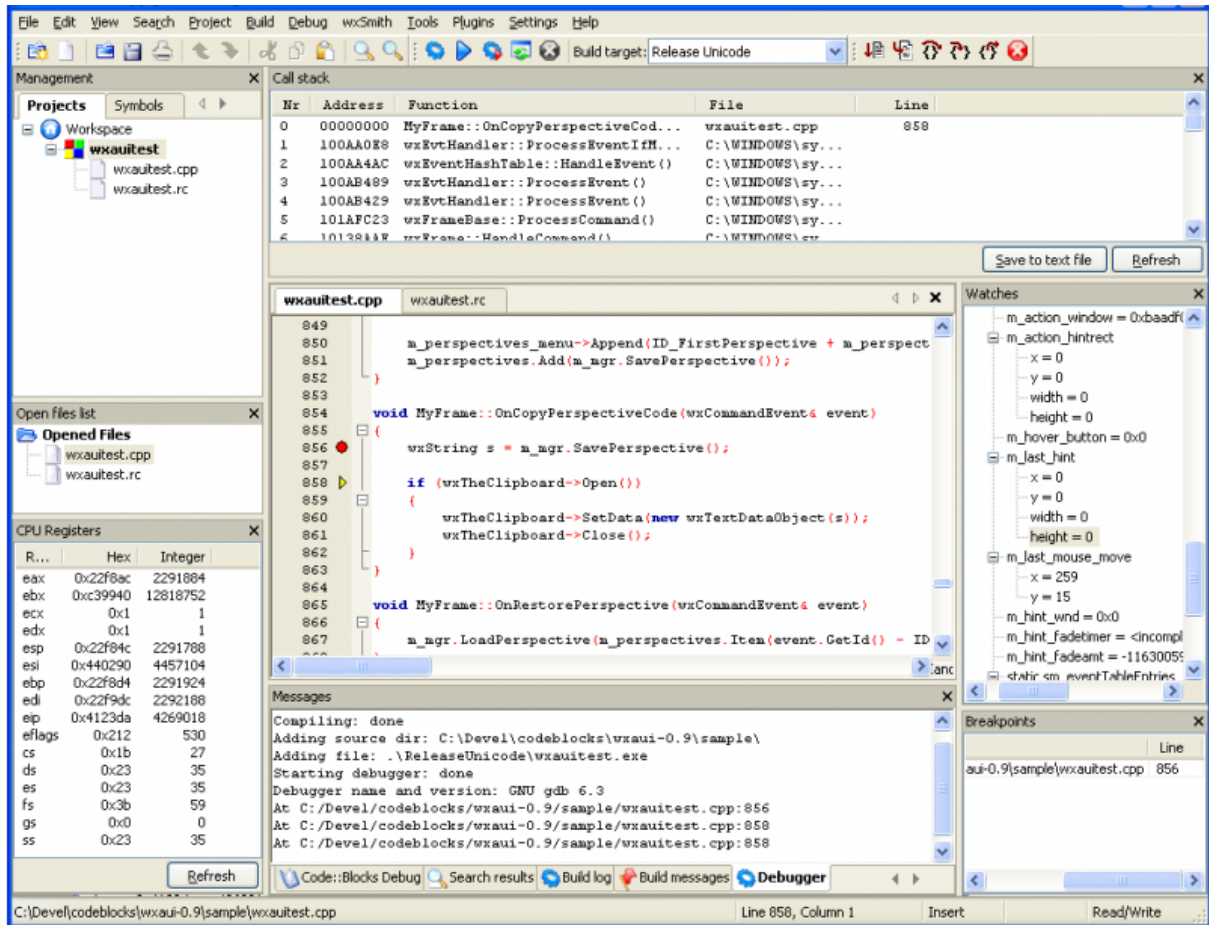
설치완료되면 다음과 같이 Menu → Programming → Code::Blocks IDE 을 실행한다.



코드블락 스크린샷



코드블락 디버깅 화면

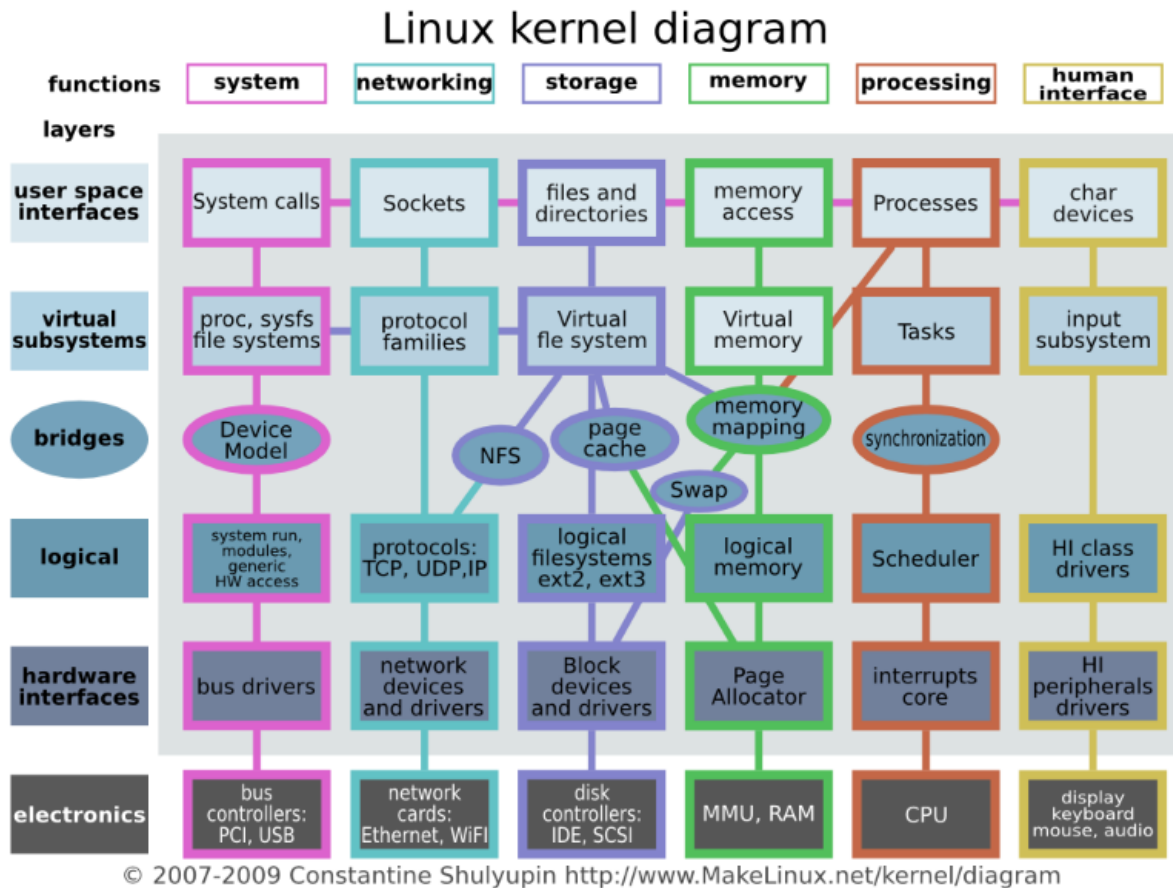


코드 블록을 root 권한으로 실행하기

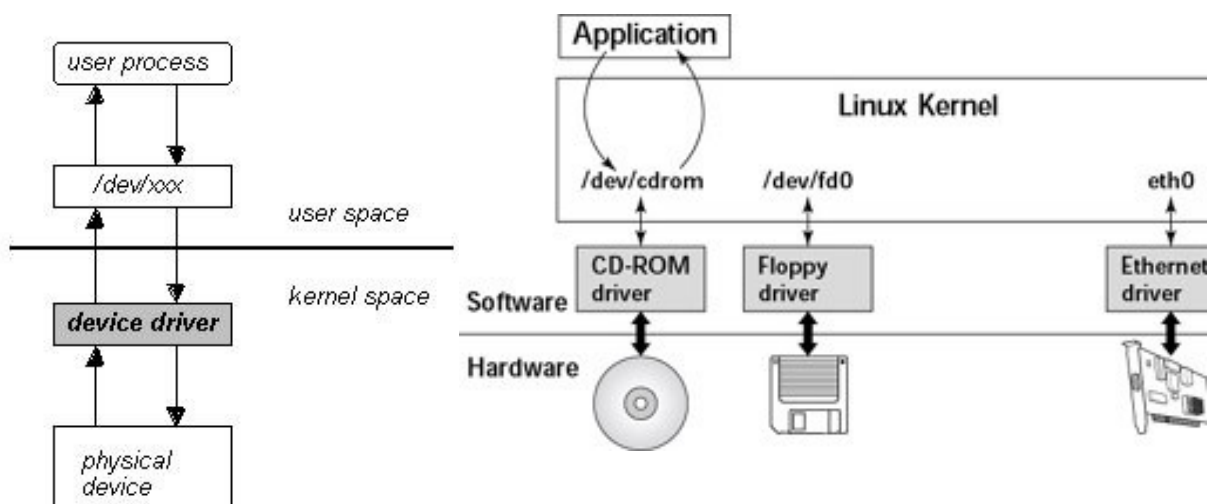
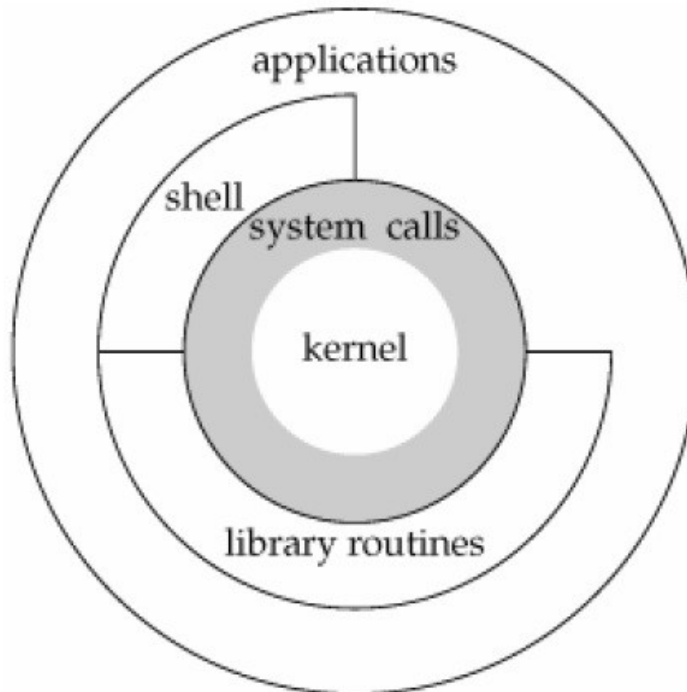
```
$ sudo codeblocks
```

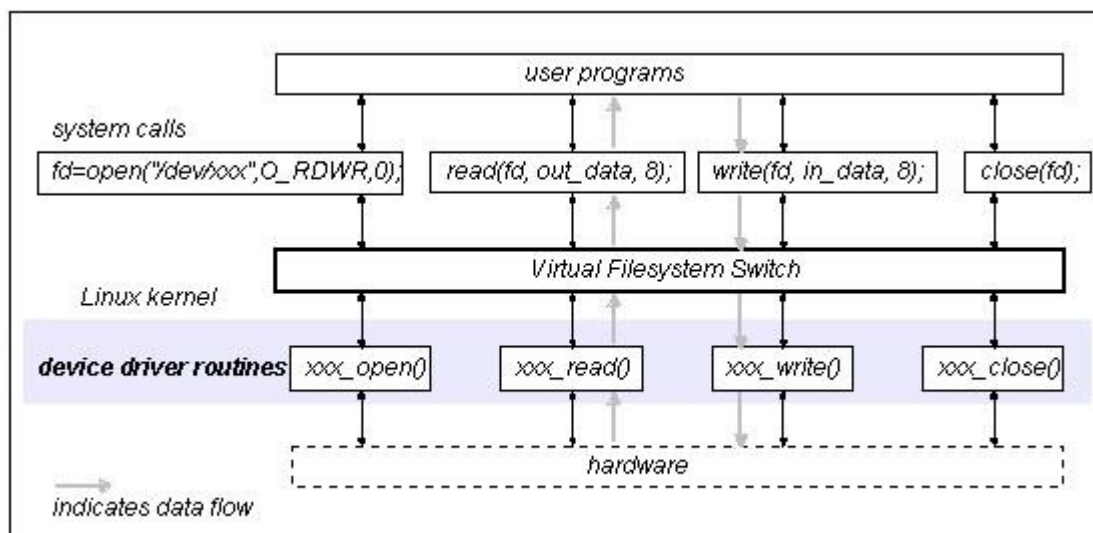
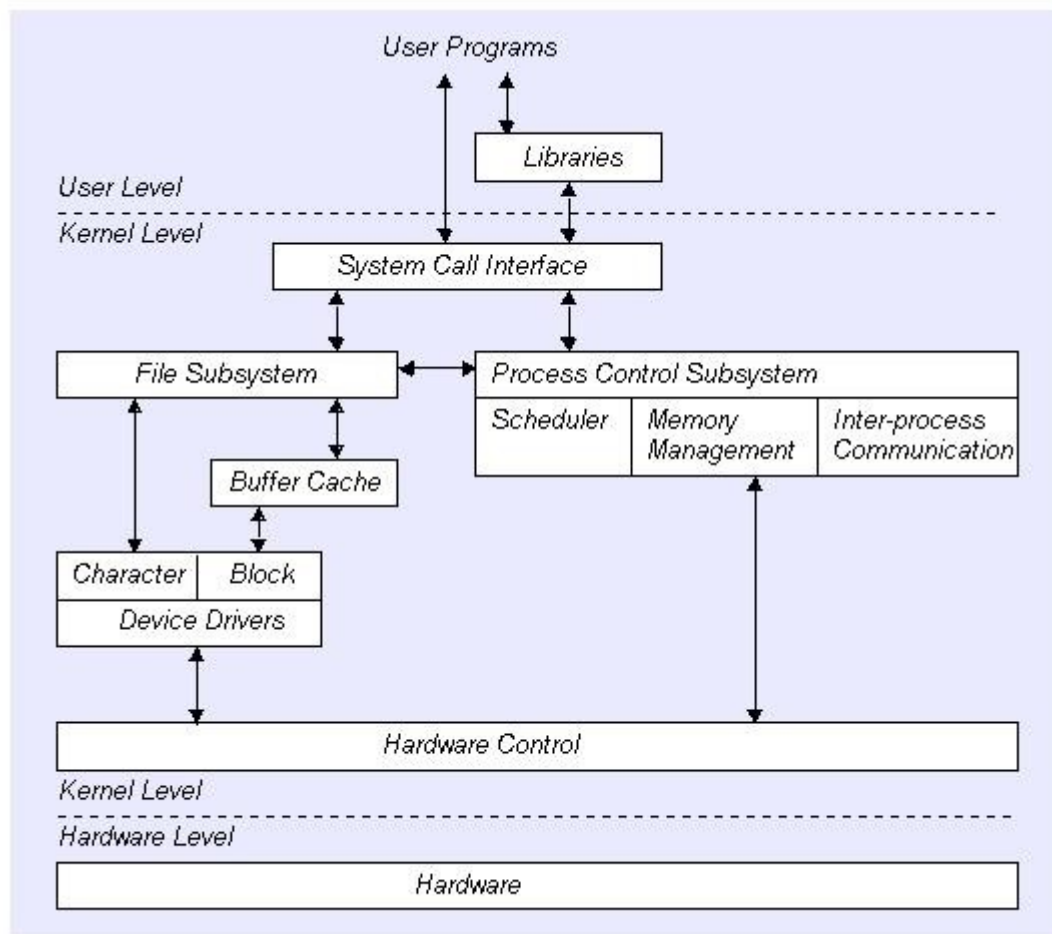
3. 리눅스 시스템 프로그래밍

<http://www.makelinux.net/kernel/diagram>



3.1 System Call



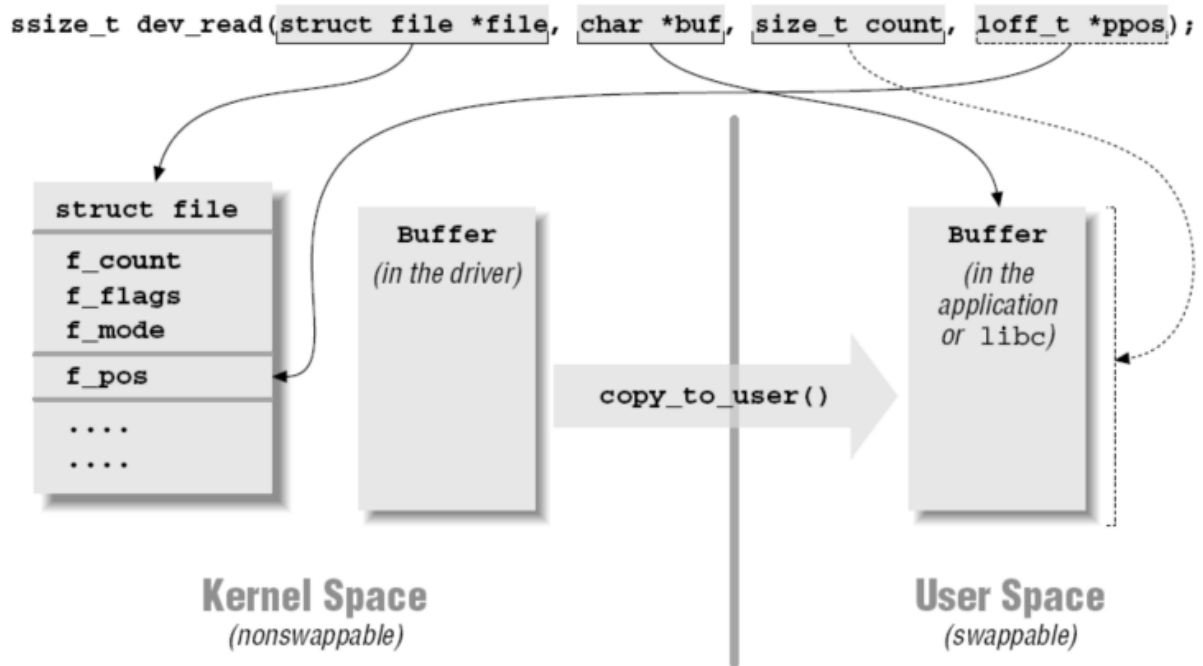


```

ret = read( int fd, void *buf, size_t count);

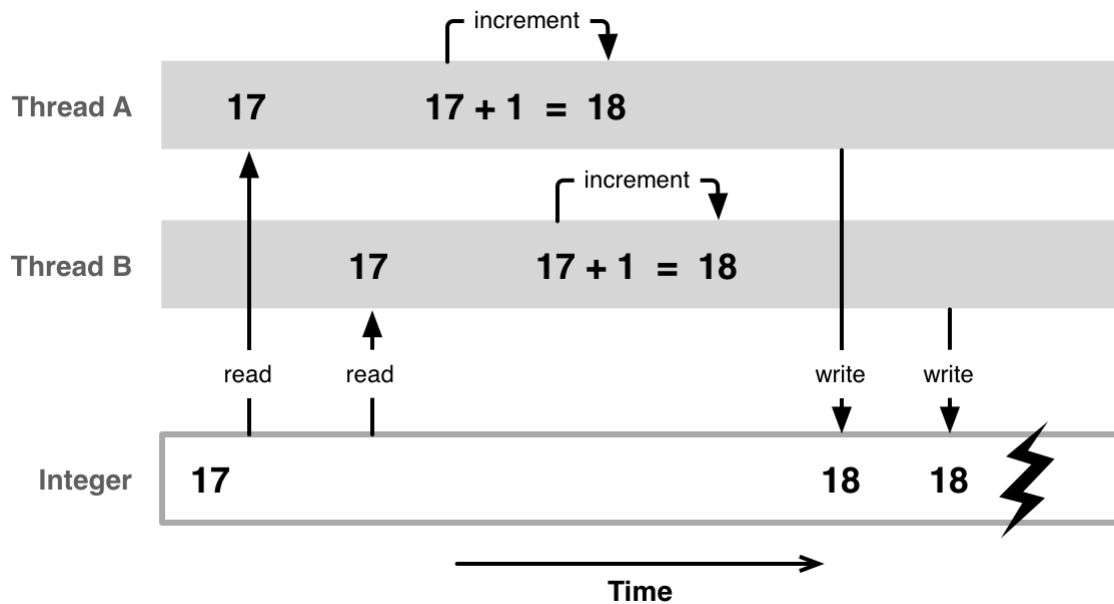
      ↓      ↓      ↓      ↓
ssize_t xxx_read( struct file *filp, char *buf, size_t count, loff_t *f_pos)
{
    return ret;
}

```

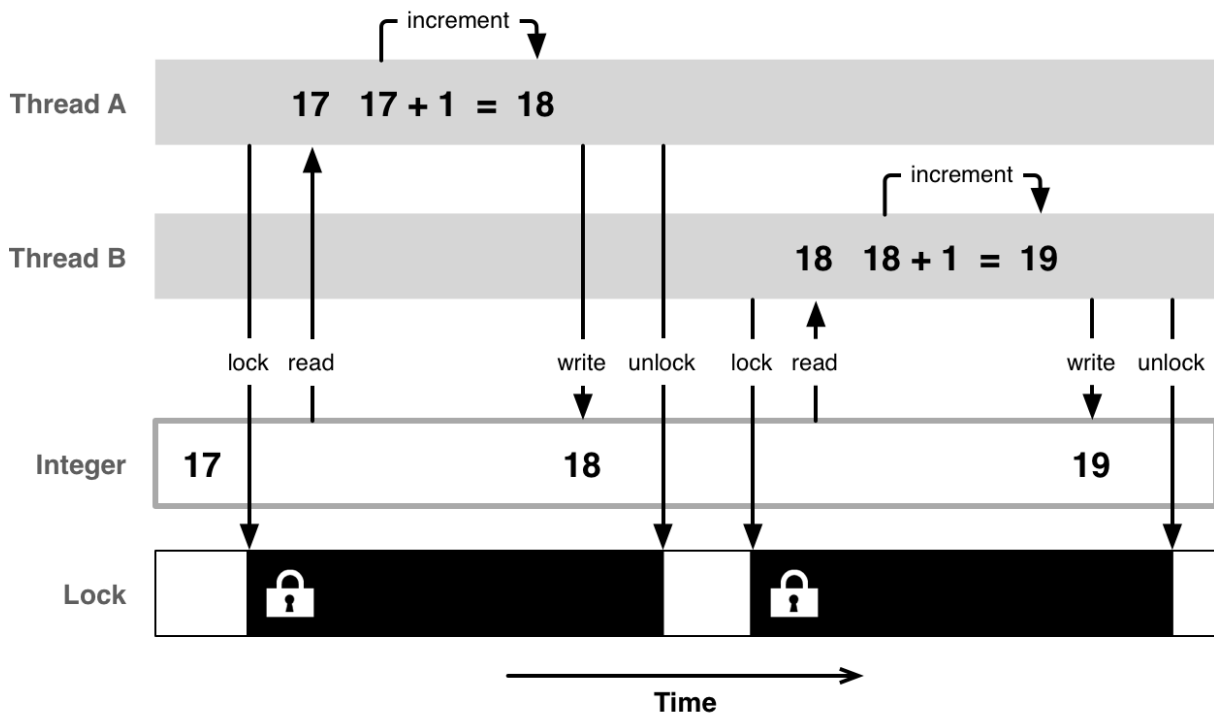


3.2 동기화(Locking)

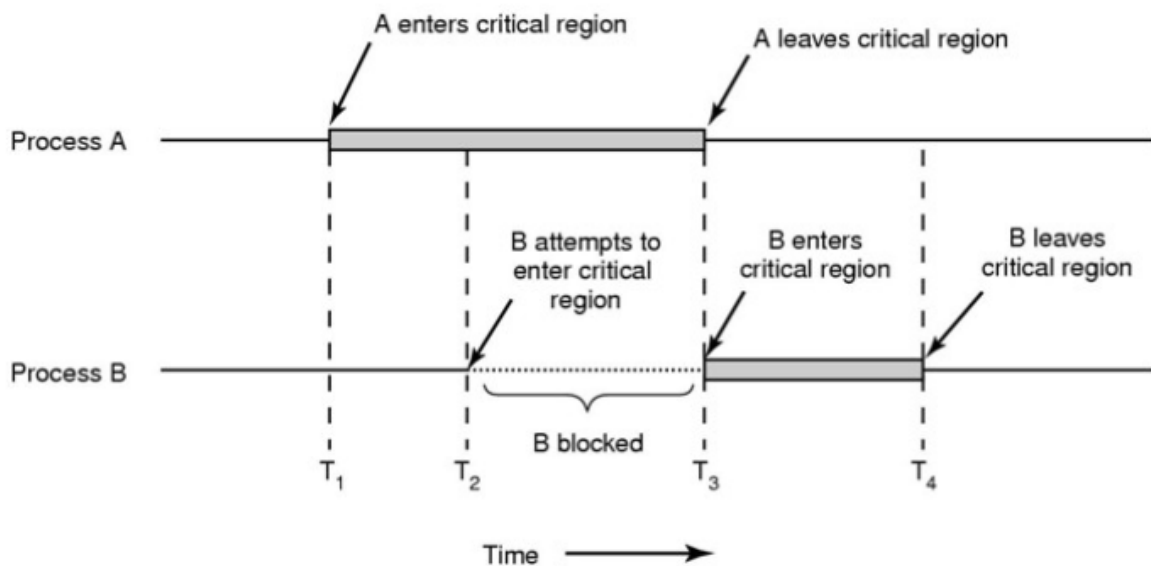
Sharing of Resources



Mutual Exclusion(Mutex Lock)



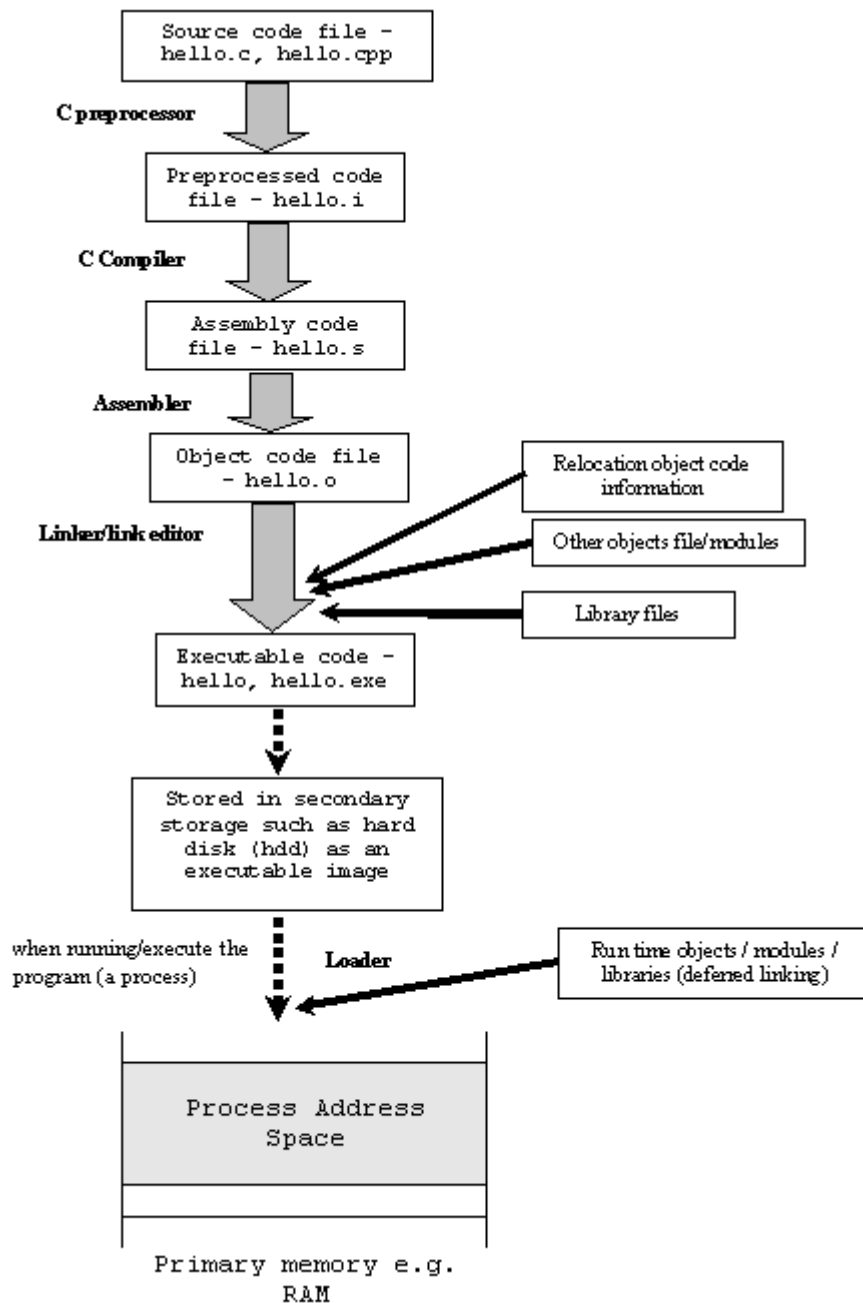
Critical sections with mutual exclusion



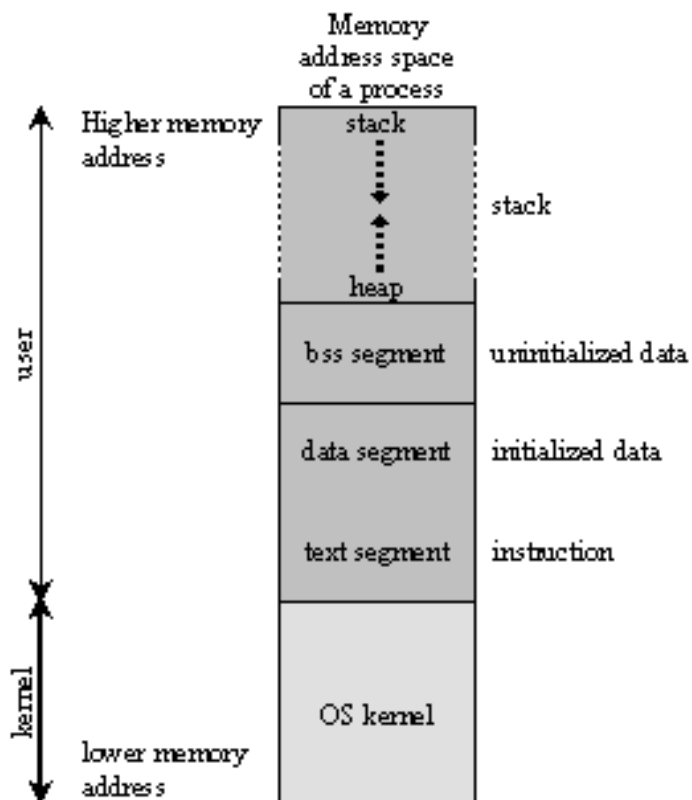
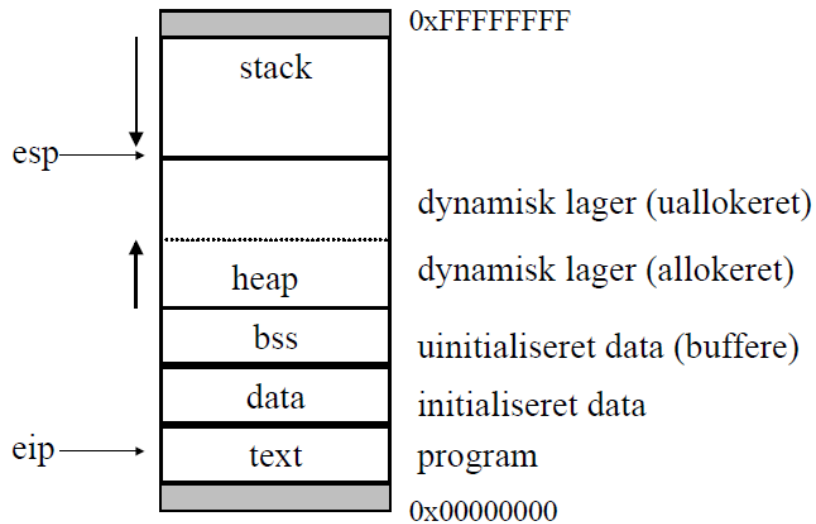
((소스 설명))

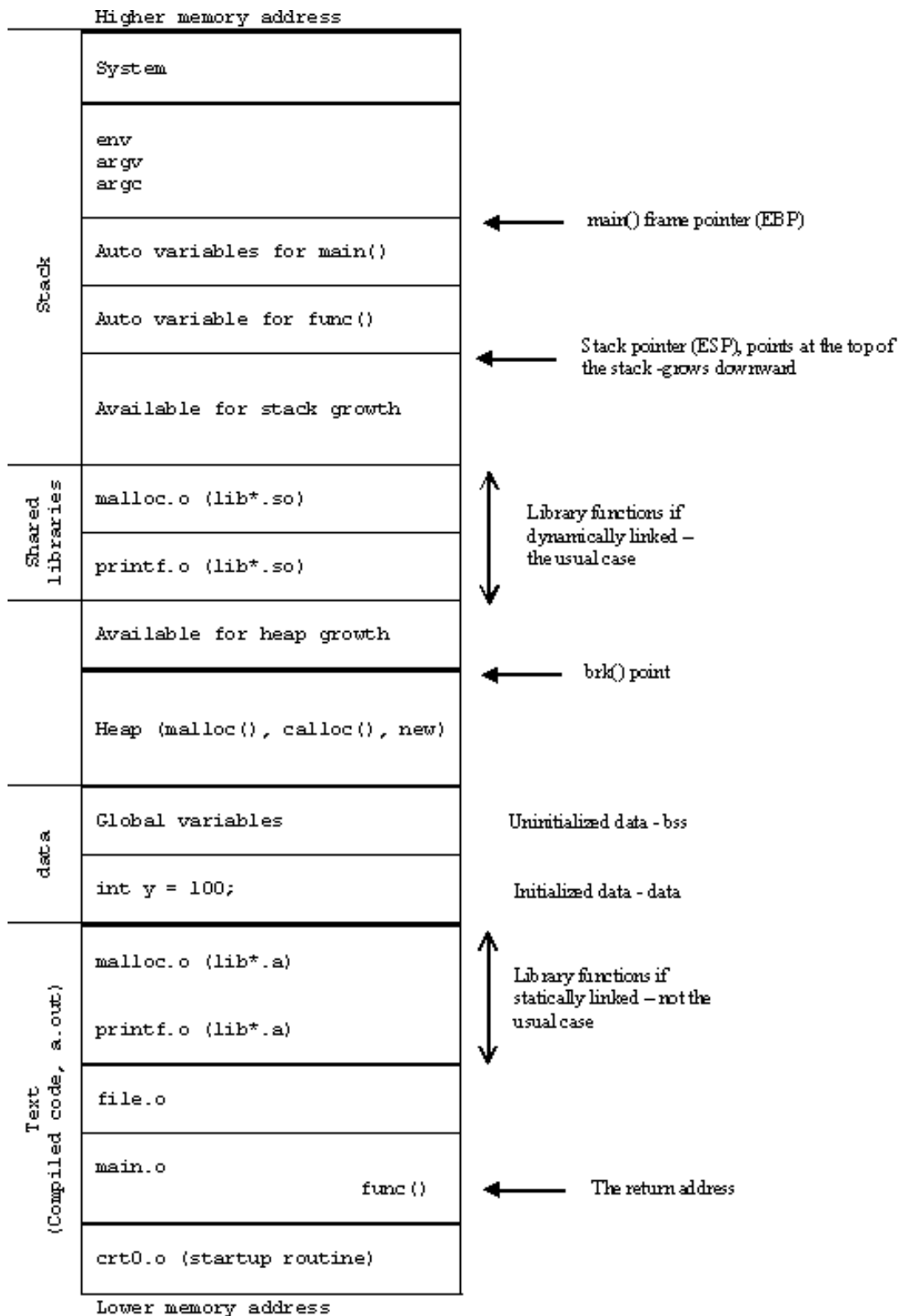
3.3 메모리 맵

Compiler, Loader



Process Memory Map





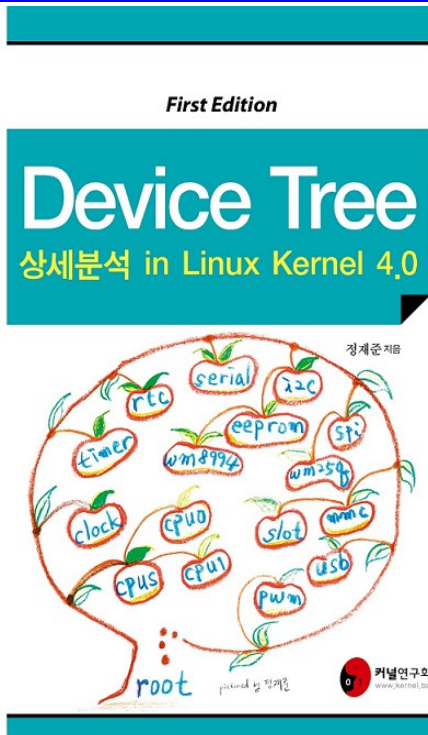
4. 리눅스 커널

이번장은 커널 소스에 관한 내용이므로 리눅스 커널에 대한 좀더 기술적인 배경지식이 필요하다. 리눅스 커널에 대한 좀더 자세한 내용은 필자가 집필한 아래 책을 참고하는 것도 좋을 듯하다.

<http://www.aladin.co.kr/shop/wproduct.aspx?ItemId=27737654>



<http://www.aladin.co.kr/shop/wproduct.aspx?ItemId=65981744>




4.1 커널소스 다운로드


<http://www.kernel.org>

The Linux Kernel Archives

[About](#)
[Contact us](#)
[FAQ](#)
[Releases](#)
[Signatures](#)
[Site news](#)



Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Stable Kernel:

4.5

mainline:	4.6-rc2	2016-04-03	[tar.xz]	[pgp]	[patch]	[view diff]	[browse]
mainline:	4.5	2016-03-14	[tar.xz]	[pgp]	[patch]	[view diff]	[browse]
stable:	4.4.6	2016-03-16	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	4.1.21	2016-04-03	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	3.18.30	2016-04-03	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	3.14.65	2016-03-16	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	3.12.57	2016-03-18	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	3.10.101	2016-03-16	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	3.4.111	2016-03-21	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	3.2.79	2016-04-01	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
longterm:	2.6.32.71 [EOL]	2016-03-12	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff] [browse] [changelog]
linux-next:	next-20160408	2016-04-08					[browse]

라즈베리파이2 커널 소스 다운로드 및 빌드 방법은 아래 웹사이트에 자세히 설명되어 있다.

<https://www.raspberrypi.org/documentation/linux/kernel/building.md>

위에 기술한 내용을 참조하여 그대로 커널 소스를 다운로드 하여 빌드해 보자.

먼저 아래와 같이 git clone 명령으로 커널 소스를 다운로드 한다.

```
$ git clone --depth=1 https://github.com/raspberrypi/linux
```

다음과 같이 다운로드 된다. (약20분정도 소요됨)

```
Cloning into 'linux'...
remote: Counting objects: 270880, done.
remote: Compressing objects: 100% (165395/165395), done.
remote: Total 270880 (delta 184711), reused 161862 (delta 103163), pack-reused 0
Receiving objects: 100% (270880/270880), 303.30 MiB | 370 KiB/s, done.
Resolving deltas: 100% (184711/184711), done.
```

다운로드가 완료되면 리눅스 커널 소스 파일들중에서 vi 편집기로 Makefile을 열어 보면 아래와 같이 리눅스 커널 소스 버전이 4.1.10 임을 알 수가 있다. 현재 리눅스 커널이 4.2.3 버전이 배포된 시점이기 때문에 비교적 최신의 커널 버전이 라즈베리파이2에 포팅되었음을 알 수 있다.

```
VERSION = 4
PATCHLEVEL = 1
SUBLEVEL = 10
EXTRAVERSION =
NAME = Series 4800

# *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.

# o Do not use make's built-in rules and variables
#   (this increases performance and avoids hard-to-debug behaviour);
# o Look for make include files relative to root of kernel src
MAKEFLAGS += -rR --include-dir=$(CURDIR)

//이하 생략...
```

리눅스 커널 버전이 4.1.10 이므로 다음과 같이 소스 경로 이름을 rename 한다.

```
$ cd ..
$ ll
total 12
drwxr-xr-x  3 jamesjung jamesjung 4096 Oct 20 15:50 ./
drwxr-xr-x  9 jamesjung jamesjung 4096 Mar 21  2015 ../
drwxr-xr-x 24 jamesjung jamesjung 4096 Oct 20 16:42 linux/

$ mv linux linux-4.1.10-raspi2
$ ll
total 12
drwxr-xr-x  3 jamesjung jamesjung 4096 Oct 20 16:42 ./
drwxr-xr-x  9 jamesjung jamesjung 4096 Mar 21  2015 ../
drwxr-xr-x 24 jamesjung jamesjung 4096 Oct 20 16:42 linux-4.1.10-raspi2/
```

4.2 커널소스 빌드

커널 소스를 빌드하기 전에, 커널 빌드에 필요한 패키지들을 아래와 같이 설치한다.

```
# apt-get update

# apt-get install git-core gnupg flex bison gperf build-essential \
zip curl libc6-dev libncurses5-dev x11proto-core-dev \
libx11-dev libreadline6-dev libgl1-mesa-dev \
libgl1-mesa-dev g++-multilib openjdk-6-jdk tofrodos \
python-markdown libxml2-utils xsltproc zlib1g-dev u-boot-tools texinfo bc lrzsz

//설치 완료시까지 수분정도 소요됨.
```

라즈베리파이2 커널 소스를 빌드하는 방법은 아래와 같이 2가지 방식이 있다.

- 라즈베리파이 보드에서 직접 빌드
- 리눅스 Host PC에서 크로스 컴파일

4.2.1 라즈베리파이 보드에서 직접 빌드

라즈베리파이1 환경설정

```
$ KERNEL=kernel  
$ make bcmrpi_defconfig
```

라즈베리파이2 환경설정

```
$ KERNEL=kernel7  
$ make bcm2709_defconfig
```

위와 같이 환경설정하면 .config 파일이 생성되고 아래와 같이 빌드를 실행한다.

커널 빌드

```
$ make zImage modules dtbs
```

라즈베리파이 보드에서 직접 커널 소스를 빌드하면, 완료하는데 약 2시간정도 소요된다. 처음에 빌드할때만 이정도 소요되고 이후로는 커널 소스 변경된 것만 컴파일하므로 수분내로 커널 소스 빌드 작업을 마무리할 수 있다.

4.2.2 리눅스 Host PC에서 크로스 컴파일

라즈베리파이1 환경설정

```
$ KERNEL=kernel  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- bcmrpi_defconfig
```

라즈베리파이2 환경설정

```
$ KERNEL=kernel7  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- bcm2709_defconfig
```


위와 같이 환경설정하면 .config 파일이 생성되고 아래와 같이 빌드를 실행한다.

커널 빌드

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- zImage modules dtbs
```

위의 커널 소스 빌드 작업은 PC의 동작 속도에 따라서 40분에서 1시간정도 소요된다.

4.3 커널 설치

위의 과정에서 빌드한 커널 이미지를 SD카드에 설치하기 전에 SD카드의 파티션 정보를 다음과 같이 확인한다.

라즈베리파이 보드에서 lsblk 명령 실행

```
# lsblk

NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0      179:0    0   7.4G  0 disk
mmcblk0p1    179:1    0    1G  0 part
mmcblk0p2    179:2    0    1K  0 part
mmcblk0p3    179:3    0   32M  0 part /media/pi/SETTINGS
mmcblk0p5    179:5    0   60M  0 part /boot
mmcblk0p6    179:6    0   6.3G  0 part /
```

fdisk 명령으로 위의 파티션 정보를 좀더 자세히 보면,

```
# fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.25.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): p
```

```

Disk /dev/mmcblk0: 7.4 GiB, 7948206080 bytes, 15523840 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00039e13

Device            Boot    Start        End    Sectors    Size Id Type
/dev/mmcblk0p1                8192    2121093    2112902      1G  e W95 FAT16 (LBA)
/dev/mmcblk0p2            2121728    15458303    13336576    6.4G  85 Linux extended
/dev/mmcblk0p3            15458304    15523839      65536    32M  83 Linux
/dev/mmcblk0p5            2129920    2252799     122880     60M  c W95 FAT32 (LBA)
/dev/mmcblk0p6            2260992    15458303    13197312    6.3G  83 Linux

Partition table entries are not in disk order.

```

mmcblk0p 1번 파티션과 5번 파티션은 FAT 파일시스템이고, mmcblk0p 2번, 3번, 6번은 리눅스 ext4 파일시스템이다. 이들 파티션들은 다음과 같은 용도로 사용된다.

1번 파티션: 커널 이미지 (recovery)

2번 파티션:

3번 파티션: /media/pi/SETTING

4번 파티션: /dev/root

5번 파티션: 커널 부트파일(/boot)

6번 파티션: 리눅스 루트파일(배포판, NOOBS)

SD카드를 다른 리눅스 머신에 장착하고 다음과 같이 파티션 정보를 확인하면,

```

# lsblk

NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
mmcblk0rpmb     179:48  0    128K  0 disk
mmcblk0boot0    179:16  0      2M  1 disk
mmcblk0boot1    179:32  0      2M  1 disk
mmcblk0         179:0   0    7.3G  0 disk
└─mmcblk0p1     179:1   0   153.1M  0 part /mnt/mmc1

```

```

├─mmcblk0p2 179:2 0 2.2G 0 part /
├─mmcblk0p3 179:3 0 4.3G 0 part /mnt/mmc3
└─mmcblk0p4 179:4 0 561.2M 0 part /mnt/mmc4
mmcblk1 179:64 0 7.4G 0 disk
├─mmcblk1p1 179:65 0 1G 0 part /mnt/mtd1
├─mmcblk1p2 179:66 0 1K 0 part
├─mmcblk1p3 179:67 0 32M 0 part /mnt/mtd3
├─mmcblk1p5 179:69 0 60M 0 part /mnt/mtd5
└─mmcblk1p6 179:70 0 6.3G 0 part /mnt/mtd6

```

위에서 커널 이미지는 /mnt/mtd5 파티션, 커널 모듈은 /mnt/mtd6 파티션에 설치하기 위해서 다음과 같이 마운트 한다.

```

# mkdir /mnt/mtd5
# mount -t vfat /dev/mmcblk1p5 /mnt/mtd5

# mkdir /mnt/mtd6
# mount -t ext4 /dev/mmcblk1p6 /mnt/mtd6

```

먼저 커널 모듈을 6번 파티션에 다음과 같이 설치한다.

```
# make INSTALL_MOD_PATH=/mnt/mtd6 modules_install
```

그런다음, 커널 이미지 파일들을 다음과 같이 복사한다.

```

# cp /mnt/mtd5/$KERNEL.img /mnt/mtd5/$KERNEL-backup.img
# scripts/mkkn1img arch/arm/boot/zImage /mnt/mtd5/$KERNEL.img
# cp arch/arm/boot/dts/*.dtb /mnt/mtd5/
# cp arch/arm/boot/dts/overlays/*.dtb* /mnt/mtd5/overlays/
# cp arch/arm/boot/dts/overlays/README /mnt/mtd5/overlays/

# umount mnt/mtd5
# umount mnt/mtd6

```

4.4 커널소스 디버깅(kgdb)

커널소스를 디버깅하기 위해서는 먼저 커널을 빌드할 때 아래와 같은 옵션들을 설정해야 한다.

커널 디버깅을 위한 빌드 옵션들

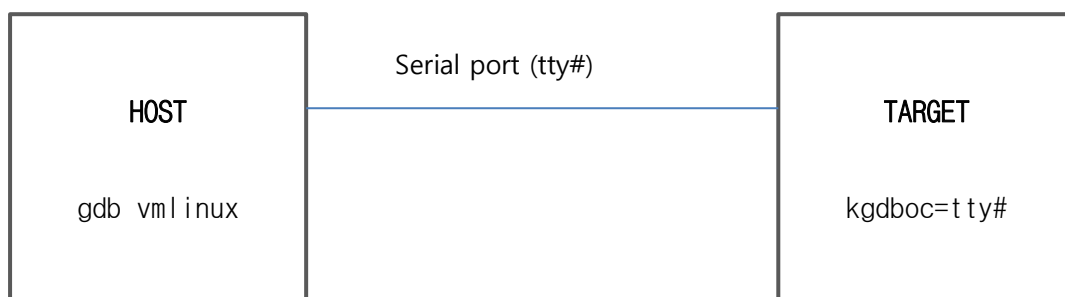
```
CONFIG_DEBUG_RODATA=n

CONFIG_DEBUG_INFO=y
CONFIG_GDB_SCRIPTS=y

CONFIG_FRAME_POINTER=y
CONFIG_KGDB=y

CONFIG_KGDB_SERIAL_CONSOLE=y
CONFIG_KGDB_KDB=y
CONFIG_KDB_KEYBOARD=y
```

kgdb로 커널 소스를 디버깅하기 위해서는 개발용 호스트 머신과 타겟보드가 다음과 같이 연결되어야 한다.



HOST 머신에서는 리눅스 커널을 빌드할 때 만들어진 vmlinux 파일을 gdb를 실행하여 로드하고, TARGET 머신에서는 커널 명령라인 옵션을 "kgdboc=tty#, 115200"을 등록한다. 위와 같이 HOST와 TARGET 머신을 시리얼 라인으로 연결하면 HOST에서 gdb를 사용하여 TARGET에 있는 리눅스 커널을 디버깅할 수 있다.

TARGET의 커널 명령라인에 kgdbwait 옵션을 추가하면 TARGET의 리눅스 커널이 부팅될 때

HOST에서 gdb로 연결하여 부팅시점에서부터 TARGET의 커널을 디버깅할 수 있다.

TARGET의 리눅스 커널이 부팅이 된 이후에는 다음과 같이 TARGET의 커널 동작을 수동 명령으로 멈춘후, HOST에서 gdb로 TARGET의 커널을 디버깅할 수 있다.

TARGET에서 실행

```
//시리얼 라인을 디버그 장치로 설정
echo tty# > /sys/module/kgdboc/parameters/kgdboc

//커널 동작 멈춤 → 디버깅 모드로 진입
echo g > /proc/sysrq-trigger
```

HOST에서 실행

```
//gdb 디버그 실행
gdb vmlinux

(gdb) set remotebaud 115200 //serial baud
(gdb) target remote /dev/ttyAMA0
(gdb) set debug remote 1

(gdb) set listsize 40
```

GDB 사용법

GDB가 정상 실행되면 터미널의 프롬프트가 (gdb)로 바뀌게 된다.

//종료

종료방법에는 크게 두가지가 있다.

```
(gdb) ctrl + d
(gdb) q
(gdb) quit
```

//소스보기

옵션에 따라 실행중인 프로그램의 소스를 다양한 방법으로 볼 수 있다.

```
l(list)
list 10
list [함수명]
list - //이전 10라인을 출력한다.
list [파일명]:[함수명]
list [파일명]:10
```

list 명령어를 사용하면 소스코드가 10줄 단위로 출력된다.

다음의 명령을 통해 출력단위를 변경할 수 있다.

```
set listsize 20
```

//세그멘테이션 폴트가 발생했을대

컴파일한 프로그램을 실행했을때 segmentation fault 가 발생하여

비정상 종료되었다면 다음의 명령어를 통해 오류 지점을 확인할 수 있다.

```
(gdb) r(run)
```

run 명령어는 GDB가 프로그램을 실행시켜 이상이 발생했을때의 파일과 지점을 출력해준다.

또한 관련 함수 또는 변수에 담긴 값을 출력하여 오류수정에 많은 도움을 준다.

오류 지점에 도달하기 전 과정을 확인하기 위해서는 다음 명령어를 이용하면 된다.

```
(gdb) bt
```

bt명령어는 백트레이스로 프로그램 스택을 역으로 탐색한다.

//브레이크 포인트

브레이크포인트는 다음의 방법들을 통해 설정 가능하다.

```
(gdb) b(break) [함수명]
(gdb) break 10
(gdb) break [파일명]:[함수명]
(gdb) break [파일명]:10
```

```
(gdb) break +2 //현재 행에서 2개 행 이후 브레이크포인트 설정
(gdb) break -2 //현재 행에서 2개 행 이전 브레이크포인트 설정
(gdb) break *0x8049000 //메모리주소에 설정(어셈블리로 디버깅시 이용)
(gdb) break 10 if var == 0 //var 변수의 값이 0일때 10번 행에 설정
```

브레이크포인트의 발동 조건은 다양하게 변경 가능하다.

```
(gdb) condition [N] var == 0 //var변수가 0일때 N번 브레이크포인트 동작
(gdb) condition [N] func(i) > 5
```

현재 설정된 브레이크포인트의 목록은 다음의 명령으로 확인 가능하다.

```
(gdb) info break
```

브레이크포인트는 GDB가 종료될때까지 유효하다.

따라서 필요없을때는 다음의 방법들을 통해 설정을 지운다.

```
(gdb) cl(clear) [함수명]
(gdb) clear 10
(gdb) clear [파일명]:[함수명]
(gdb) clear [파일명]:10
(gdb) d //모든 브레이크포인트 지움
(gdb) disable br //모든 브레이크포인트 비활성화
(gdb) disable br 1 3 //1번, 3번 브레이크포인트 비활성화
(gdb) enable br //모든 브레이크포인트 활성화
(gdb) enable br 1 3 //1번, 3번 브레이크포인트 활성화
```

//프로그램 실행

프로그램의 실행은 run 명령어를 이용한다.

만일 이미 실행중일때는 재실행한다.

```
(gdb) r(run)
```

프로그램 실행시 인자를 지정하기 위해서는 다음과 같이 이용한다.

```
(gdb) run arg1 arg2
```


실행중인 프로그램을 종료할 때는 kill 명령어를 이용한다.

```
(gdb) k(kill)
```

현재 실행중인 행의 수행을 멈추기 위해서는 step 명령어를 이용한다.

step 명령어는 한행씩 동작하도록 한다. next 명령어와는 함수 호출시 다른 결과를 보인다.

```
(gdb) s(step)
```

```
(gdb) step 6 //step을 6번 수행
```

현재 행의 실행이 멈춘상태에서 다음 행을 실행하기 위해서는

```
(gdb) n(next)
```

```
(gdb) next 6 //next를 6번 수행
```

만일 step명령을 이용중 루프에 빠져 나오지 못할경우에는 until 명령어를 이용한다.

```
(gdb) u(until)
```

한행씩이 아닌 다시 연결아서 실행하기 위해서는

```
(gdb) c(continue)
```

함수가 매우 길어 끝나는 지점으로 이동하기 위해서는 finish 명령어를 사용한다.

```
(gdb) finish
```

함수의 남은 부분을 수행하지 않고 빠져나오기 위해서는 return 명령어를 사용한다.

```
(gdb) return
```

return 명령어를 사용시 return 값을 임의로 지정하기 위해서는 다음과 같이 이용한다.

```
(gdb) return 1234
```

//와치포인트 설정

와치포인트는 변수값의 변화와 코드의 변화를 확인할때 편리하게 이용가능하다.

```
(gdb) watch [변수명] //변수에 값이 써질 때 브레이크
(gdb) rwatch [변수명] //변수의 값이 읽혀질 때 브레이크
(gdb) awatch [변수명] //변수에 읽기, 쓰기 경우에 브레이크
```

//변수와 레지스터 값 검사

현재 위치한 행에서 접근 가능한 지역변수들 목록 확인

```
(gdb) info locals
```

현재 위치한 행에서 접근 가능한 전역변수들 목록 확인

```
(gdb) info variables
```

확인하고싶은 변수의 값을 출력하기 위해서는 print 명령어를 사용한다.

```
(gdb) p(print) [변수명] //변수의 값
(gdb) print [함수명] //함수의 주소 값
```

포인터 변수의 경우 위의 방법으로 하면 주소값만이 출력된다.

포인터 변수의 값 또는 포인터 구조체 등의 값을 보기 위해서는 * 를 붙여준다.

```
(gdb) print *[변수명]
```

이중 포인터라면 ** 를 붙여준다.

GDB는 변수 뿐만 아니라 레지스터의 값도 확인할 수 있다.

```
(gdb) print $[레지스터명]
```

print 명령어는 지역변수를 우선하여 보여주기 때문에

지역변수와 전역변수에서 동일한 이름을 사용할때 전역변수를 확인하기 위해서는 :: 을 이용한다.

```
(gdb) print 'main.c'::[변수명]
```

파일명은 '따옴표' 으로 감싸야한다.

특정 함수에 있는 변수를 확인하기 위해서는

```
(gdb) print [함수명]::[변수명]
```

print 명령어로 변수 또는 레지스터를 확인할 때는 기본적으로 10진수로 출력한다.

이를 다른 형식으로 보고싶을 때는 다음과 같은 방법을 이용한다.

```
(gdb) print/t [변수명]    //2진수로
(gdb) print/o [변수명]    //8진수로
(gdb) print/d [변수명]    //10진수로 (int)
(gdb) print/u [변수명]    //부호없는 10진수로 (unsigned int)
(gdb) print/x [변수명]    //16진수로
(gdb) print/c [변수명]    //최초 1바이트 값을 문자형으로
(gdb) print/f [변수명]    //부동소수점값
(gdb) print/a [변수명]    //가장 가까운 심볼의 오프셋
```

print 명령어는 값을 보여줄뿐 아니라 값을 설정하는 것도 가능하다.

```
(gdb) print [변수명] = [값]
```

//화면에 변수의 값을 자동으로 디스플레이하기

display 명령어를 이용하면 매 단계가 진행될때마다 자동으로 변수의 값을 출력해준다.

```
(gdb) display [변수명]
```

display 변수를 해제하기 위해서는 undisplay 명령어를 이용한다.

```
(gdb) undisplay [N]
```

display 역시 x,c,o 등등을 이용해 다양한 형태로 출력 가능하다.

//커널 모듈 디버깅

Target:

```
modprobe kgdboc kgdboc=tty#,115200
```

Host:

```
(gdb) set solib-search-path 로컬모듈경로(/mnt/mmc4/modules/4.1.15-v7/kernel/drivers/)
```

```
(gdb) info sharedlibrary
```

gdb을 실행할때마다 환경설정 명령을 입력하는 것은 번거로운 작업이므로, gdb가 실행될 때 자동으로 실행되는 명령어를 /etc/gdb/gdbinit 파일에 다음과 같이 입력해 두면, 환경설정을 매번 하지 않아도 되므로 좀더 편리하다.

kgdb 자동설정(/etc/gdb/gdbinit)

```
# System-wide GDB initialization file.
```

```
set serial baud 115200
```

```
target remote /dev/ttyAMA0
```

```
# set debug remote 1 //remote debug message
```

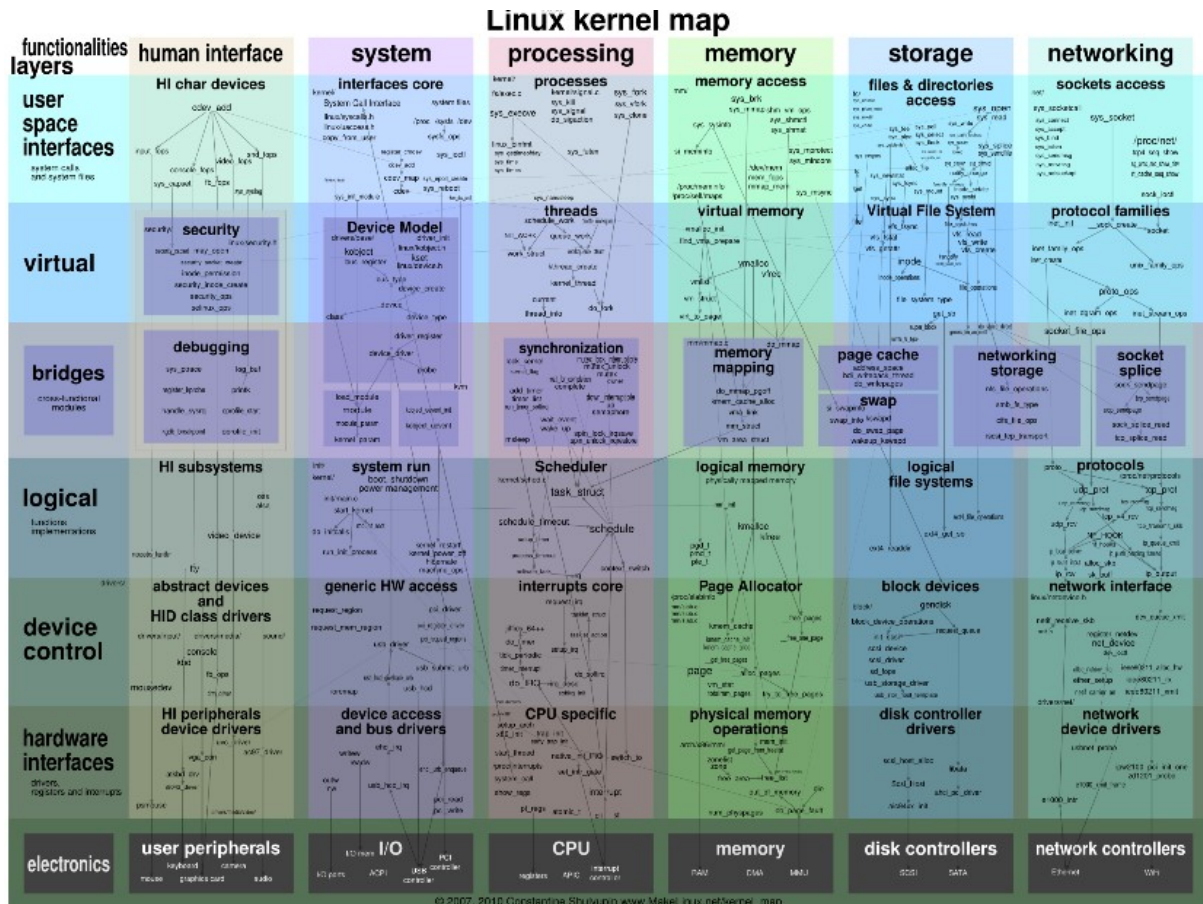
```
set listsize 40
```

기타 리눅스 커널 소스 디버깅에 대한 자세한 내용들은 아래 링크를 참조하기 바란다.

<http://landley.net/kdocs/Documentation/DocBook/xhtml-nochunks/kgdb.html>

4.5 디바이스드라이버

http://www.makelinux.net/kernel_map/



<http://lxr.free-electrons.com/ident>

Linux Cross Reference

Free Electrons
Embedded Linux Experts

• Source Navigation • Identifier Search • Freetext Search •

Version: 2.0.40 2.2.26 2.4.37 3.8 3.9 3.10 3.11 3.12 3.13 3.14 3.15 3.16 3.17 3.18 3.19 4.0 4.1 4.2 4.3 4.4 4.5

Identifier: Go get it

This page was automatically generated by LXR 0.3.1 (source). • Linux is a registered trademark of Linus Torvalds • [Contact us](#)

[HOME](#)
[DEVELOPMENT](#)
[SERVICES](#)
[TRAINING](#)
[DOCS](#)
[COMMUNITY](#)
[COMPANY](#)
[BLOG](#)

