

---

# Building Embedded Linux System

Toolchains

Bootloader

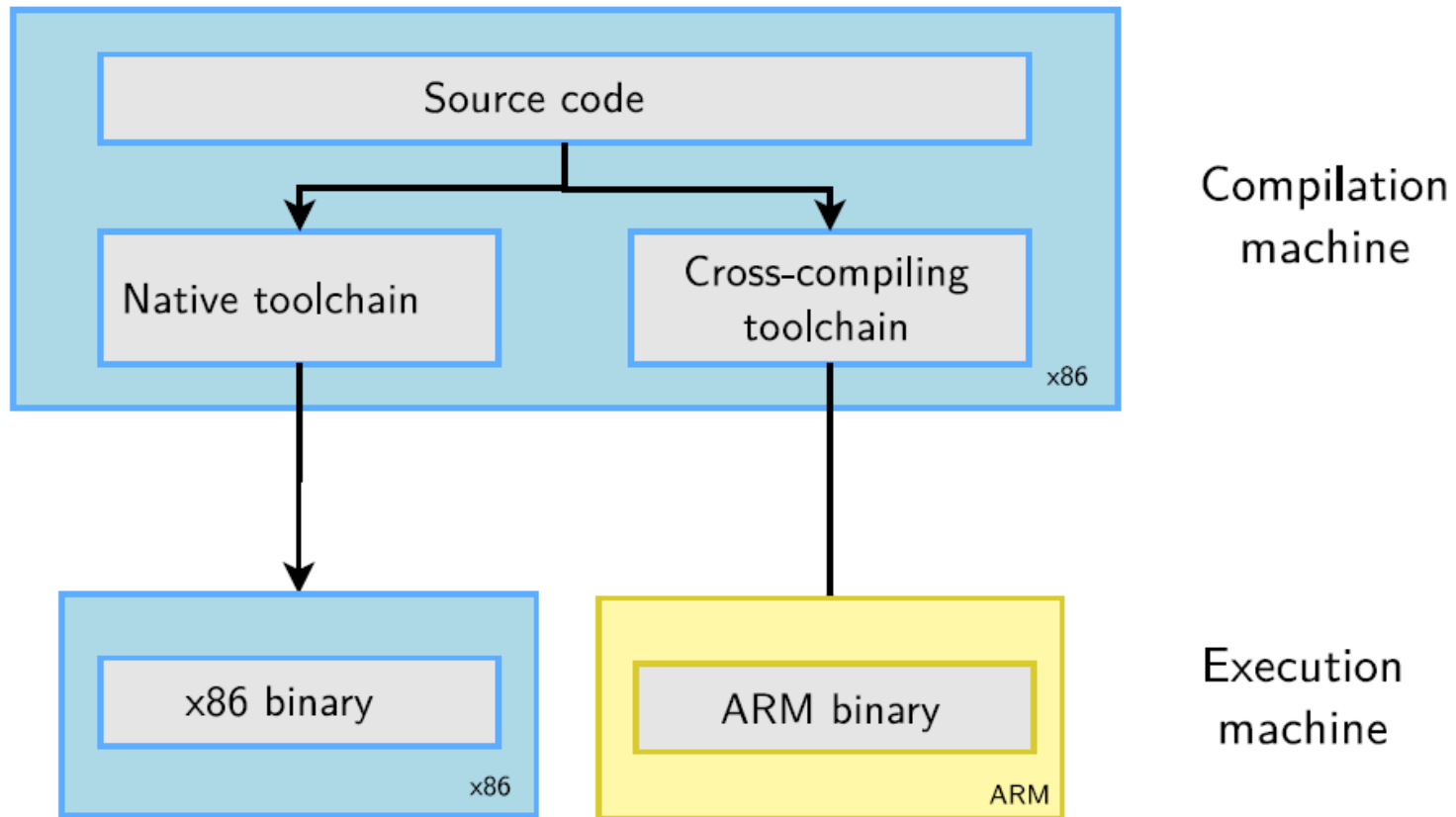
Kernel

Root File System

# Toolchain

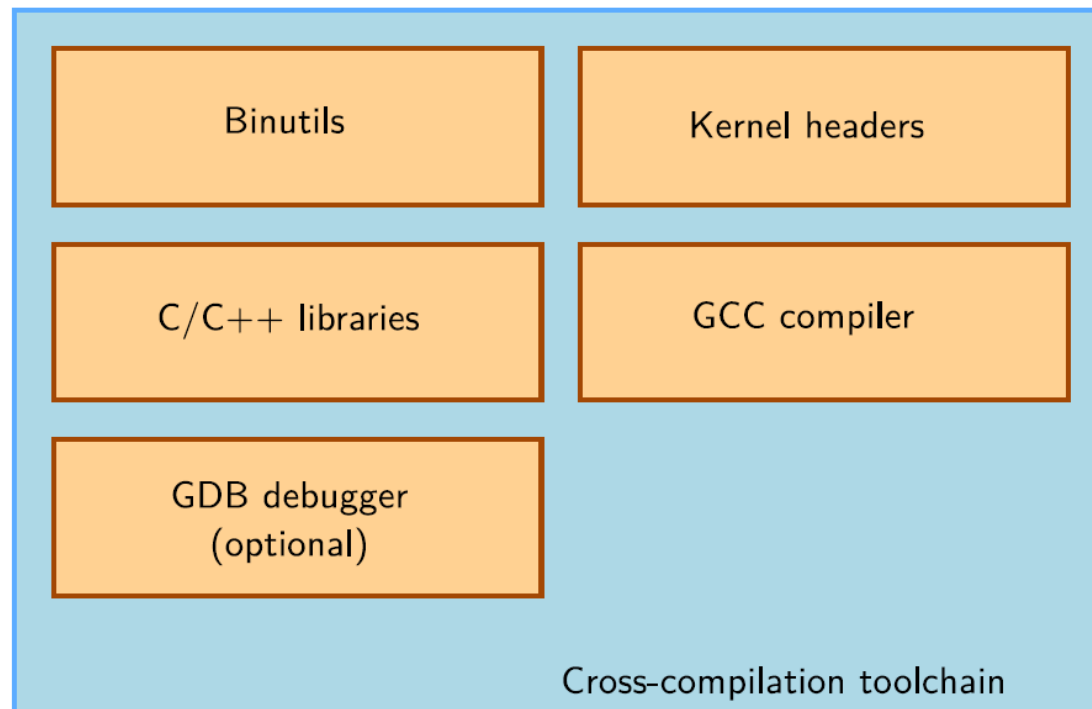
- 정의

- 소스를 컴파일하여 바이너리 실행 파일을 생성하기 위해 필요한 컴파일러 및 라이브러리, 바이너리 유틸리티 모음



# Toolchain

- 구성요소
  - GCC : 컴파일러
  - Binutils : 어셈블러 및 로더, 바이너리 파일 편집 유틸리티
  - Glibc : 크로스 컴파일을 위한 라이브러리 및 일반 라이브러리
  - Linux 커널 : 리눅스 커널 소스, Kernel headers



# Toolchain

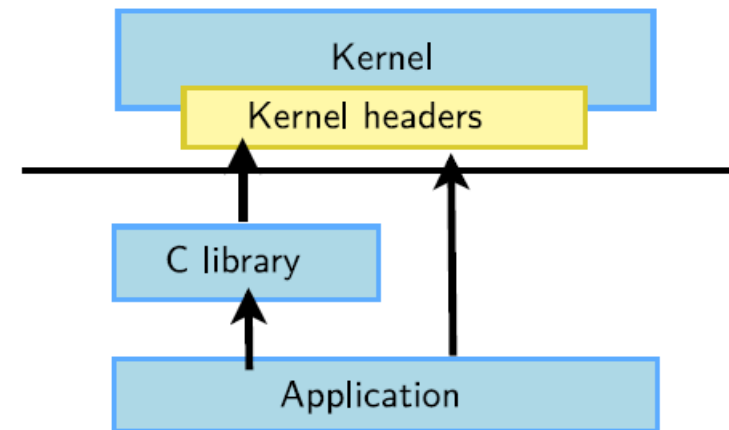
---

- ▶ **Binutils** is a set of tools to generate and manipulate binaries for a given CPU architecture
  - ▶ `as`, the assembler, that generates binary code from assembler source code
  - ▶ `ld`, the linker
  - ▶ `ar`, `ranlib`, to generate `.a` archives, used for libraries
  - ▶ `objdump`, `readelf`, `size`, `nm`, `strings`, to inspect binaries. Very useful analysis tools!
  - ▶ `strip`, to strip parts of binaries that are just needed for debugging (reducing their size).
- ▶ <http://www.gnu.org/software/binutils/>
- ▶ GPL license

# Toolchain

---

- ▶ The C library and compiled programs need to interact with the kernel
  - ▶ Available system calls and their numbers
  - ▶ Constant definitions
  - ▶ Data structures, etc.
- ▶ Therefore, compiling the C library requires kernel headers, and many applications also require them.
- ▶ Available in `<linux/...>` and `<asm/...>` and a few other directories corresponding to the ones visible in `include/` in the kernel sources



# Toolchain

---

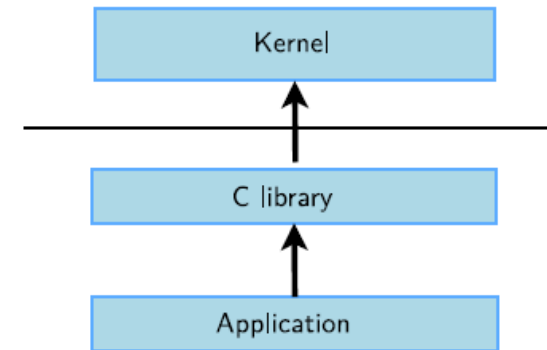
- ▶ GNU Compiler Collection, the famous free software compiler
- ▶ Can compile C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, and generate code for a large number of CPU architectures, including ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86\_64, IA64, Xtensa, etc.
- ▶ <http://gcc.gnu.org/>
- ▶ Available under the GPL license, libraries under the LGPL.



# Toolchain

---

- ▶ The C library is an essential component of a Linux system
  - ▶ Interface between the applications and the kernel
  - ▶ Provides the well-known standard C API to ease application development
- ▶ Several C libraries are available:  
*glibc, uClibc, musl, dietlibc, newlib, etc.*
- ▶ The choice of the C library must be made at the time of the cross-compiling toolchain generation, as the GCC compiler is compiled against a specific C library.



# Toolchain 설치

- /opt/toolchains 디렉토리를 만든다
  - # mkdir /opt/toolchains
- Toolchain 압축 파일의 압축을 푼다.
  - # tar jxvf arm-2014.05-29-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 -C /opt/toolchains/

```
control@lab-pc2: /opt/toolchains/arm-2014.05
control@lab-pc2:~$ cd /opt/toolchains
control@lab-pc2:/opt/toolchains$ ls
arm-2014.05
control@lab-pc2:/opt/toolchains$ cd arm-2014.05
control@lab-pc2:/opt/toolchains/arm-2014.05$ ls
arm-none-linux-gnueabi  bin  i686-pc-linux-gnu  lib  libexec  share
control@lab-pc2:/opt/toolchains/arm-2014.05$
```



# Toolchain 설치

- 컴파일에 사용되는 명령어들을 어느 디렉토리에서나 실행할 수 있기 위해서는 `.bashrc` 파일의 `PATH` 환경변수에 `/opt/toolchains/arm-2014.05/bin`을 추가해 주어야 한다.
- 크로스 컴파일러의 명령어 디렉토리(`/opt/toolchains/arm-2014.05/bin`)

```
control@lab-pc2: /opt/toolchains/arm-2014.05/bin
control@lab-pc2:~$ cd /opt/toolchains/arm-2014.05
control@lab-pc2:/opt/toolchains/arm-2014.05$ ls
arm-none-linux-gnueabi  bin  i686-pc-linux-gnu  lib  libexec  share
control@lab-pc2:/opt/toolchains/arm-2014.05$ cd bin
control@lab-pc2:/opt/toolchains/arm-2014.05/bin$ ls
arm-none-linux-gnueabi-addr2line  arm-none-linux-gnueabi-gcc-ranlib
arm-none-linux-gnueabi-ar          arm-none-linux-gnueabi-gcov
arm-none-linux-gnueabi-as          arm-none-linux-gnueabi-gdb
arm-none-linux-gnueabi-c++         arm-none-linux-gnueabi-gprof
arm-none-linux-gnueabi-c++filt     arm-none-linux-gnueabi-ld
arm-none-linux-gnueabi-cpp         arm-none-linux-gnueabi-nm
arm-none-linux-gnueabi-cs          arm-none-linux-gnueabi-objcopy
arm-none-linux-gnueabi-cs-daemon   arm-none-linux-gnueabi-objdump
arm-none-linux-gnueabi-elfedit     arm-none-linux-gnueabi-ranlib
arm-none-linux-gnueabi-g++         arm-none-linux-gnueabi-readelf
arm-none-linux-gnueabi-gcc         arm-none-linux-gnueabi-size
arm-none-linux-gnueabi-gcc-4.8.3  arm-none-linux-gnueabi-strings
arm-none-linux-gnueabi-gcc-ar      arm-none-linux-gnueabi-strip
arm-none-linux-gnueabi-gcc-nm     cache
control@lab-pc2:/opt/toolchains/arm-2014.05/bin$
```

# Toolchain 설치

- .bashrc 파일 수정
  - # vi .bashrc



```
control@lab-pc2: ~
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
  . /etc/bash_completion
fi
# Cross Compiler
export CROSS_COMPILE=arm-none-linux-gnueabi-
export PATH=/opt/toolchains/arm-2014.05/bin:$PATH
export ARCH=arm
```

- source 명령을 통해 변경된 .bashrc를 쉘에 적용
  - # source .bashrc

# Toolchain Test

---

- X86계열에서 사용하는 리눅스 gcc와 다른 점은 생성되는 코드가 ARM용으로 생성된다는 것이다. 간단한 프로그램을 작성하여 테스트 한다.
  - 테스트 프로그램 작성

A terminal window with a dark title bar containing the text 'control@lab-pc2: ~/work/hello'. The main area has a light yellow background and displays the following C code:

```
control@lab-pc2:~/work/hello$ cat hello.c
#include <stdio.h>

void main(void)
{
    printf("Hello\n");
}
control@lab-pc2:~/work/hello$
```

# Toolchain Test

- 컴파일
  - # gcc -o hello hello.c
  - # arm-none-linux-gnueabi-gcc -o hello-arm hello.c
- 컴파일 된 파일 정보 확인(file 명령 사용)

```
control@lab-pc2: ~/work/hello
control@lab-pc2:~/work/hello$
control@lab-pc2:~/work/hello$ ls
hello hello-arm hello.c Makefile Makefile~
control@lab-pc2:~/work/hello$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0x988f2bde34210d29f567f
aaf8d0b83c14ed27583, not stripped
control@lab-pc2:~/work/hello$ file hello-arm
hello-arm: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.16, not stripped
control@lab-pc2:~/work/hello$
```

# Minicom이란?

---

- TARGET은 출력을 위한 별도의 터미널을 가지고 있지 않음
- 일반적으로 serial port를 통한 터미널 프로그램 이용
  - Linux에서는 일반적으로 minicom 사용
  - Windows에서는 HyperTerminal 과 유사한 프로그램을 사용
- Minicom이란?
  - 호스트와 타겟을 연결해주는 가상터미널이다. 하이퍼터미널과 유사한 프로그램이고, 타겟의 화면을 호스트에서 볼 수 있게 해준다.
- Minicom 프로그램을 사용하기 위해 먼저 설정을 해준다.
  - minicom 프로그램은 시리얼에 연결되어 있기 때문에 타겟의 시리얼 설정에 맞는 호스트 설정이 필요하다.
- Minicom 용도
  - 부트로더의 명령 프롬프트
  - 임베디드 리눅스의 쉘 프롬프트를 위한 콘솔로 사용

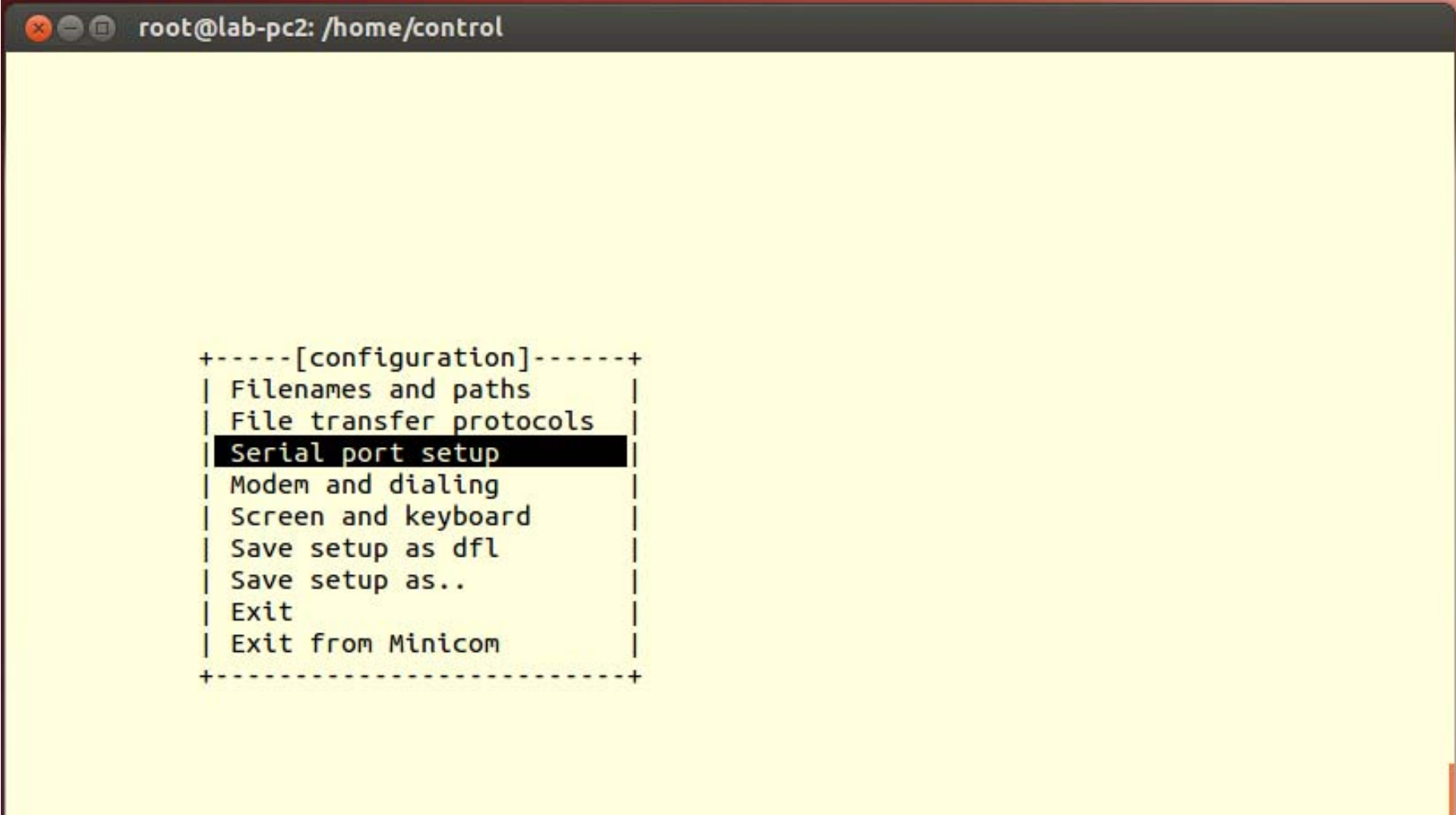
# Minicom 설정

- minicom 설치
  - # apt-get install minicom
- Serial port 확인
  - # dmesg

```
[ 290.719202] usb 2-2.2: new full-speed USB device number 5 using uhci_hcd
[ 290.934936] usb 2-2.2: New USB device found, idVendor=067b, idProduct=2303
[ 290.934946] usb 2-2.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 290.934951] usb 2-2.2: Product: USB-Serial Controller D
[ 290.934955] usb 2-2.2: Manufacturer: Prolific Technology Inc.
[ 291.091973] usbcore: registered new interface driver usbserial
[ 291.092295] usbcore: registered new interface driver usbserial_generic
[ 291.092606] usbserial: USB Serial support registered for generic
[ 291.095579] usbcore: registered new interface driver pl2303
[ 291.095970] usbserial: USB Serial support registered for pl2303
[ 291.096290] pl2303 2-2.2:1.0: pl2303 converter detected
[ 291.115651] usb 2-2.2: pl2303 converter now attached to ttyUSB0
root@lab-pc2:/home/control# ls /dev/ttyUSB0
/dev/ttyUSB0
root@lab-pc2:/home/control#
```

# Minicom 설정

- minicom 환경 설정 모드로 진입
  - # minicom -s



```
root@lab-pc2: /home/control

+-----[configuration]-----+
| Filenames and paths          |
| File transfer protocols      |
| Serial port setup          |
| Modem and dialing           |
| Screen and keyboard         |
| Save setup as dfl           |
| Save setup as..             |
| Exit                         |
| Exit from Minicom           |
+-----+-----+

```

# Minicom 설정

- COM Port 선택
  - Serial port setup 항목을 선택한다. A를 선택하여 Serial Device를 직렬 케이블이 연결된 직렬 포트에 설정한다.

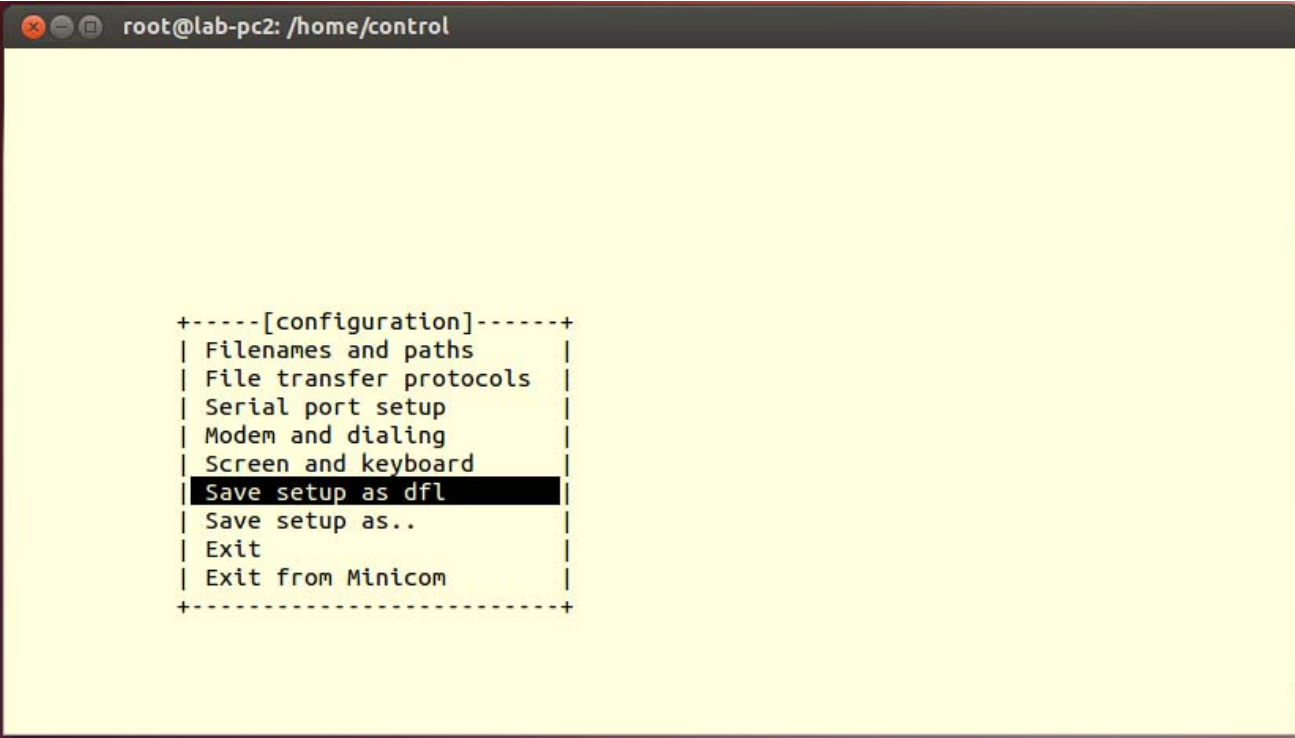
```
root@lab-pc2: /home/control

+-----+
| A -   Serial Device       : /dev/ttyUSB0 |
| B - Lockfile Location    : /var/lock     |
| C -   Callin Program     :               |
| D -   Callout Program    :               |
| E -   Bps/Par/Bits       : 115200 8N1   |
| F - Hardware Flow Control : Yes         |
| G - Software Flow Control : No         |
+-----+
|
| Change which setting? █
|
+-----+
| Screen and keyboard |
| Save setup as dfl   |
| Save setup as..    |
| Exit                |
| Exit from Minicom   |
+-----+
```



# Minicom 설정

- 설정 값 저장
  - Save setup as dfl를 선택한 후 Enter를 눌러 설정된 값을 저장한다.
- 저장 후 Exit를 눌러 설정 밖으로 나간다. (그 후 타켓 보드를 켜면 부팅되면서 그 화면이 minicom으로 보여진다.)



```
root@lab-pc2: /home/control

+-----[configuration]-----+
| Filenames and paths          |
| File transfer protocols      |
| Serial port setup           |
| Modem and dialing           |
| Screen and keyboard         |
| Save setup as dfl           |
| Save setup as..             |
| Exit                         |
| Exit from Minicom           |
+-----+-----+

```

# TFTP란?

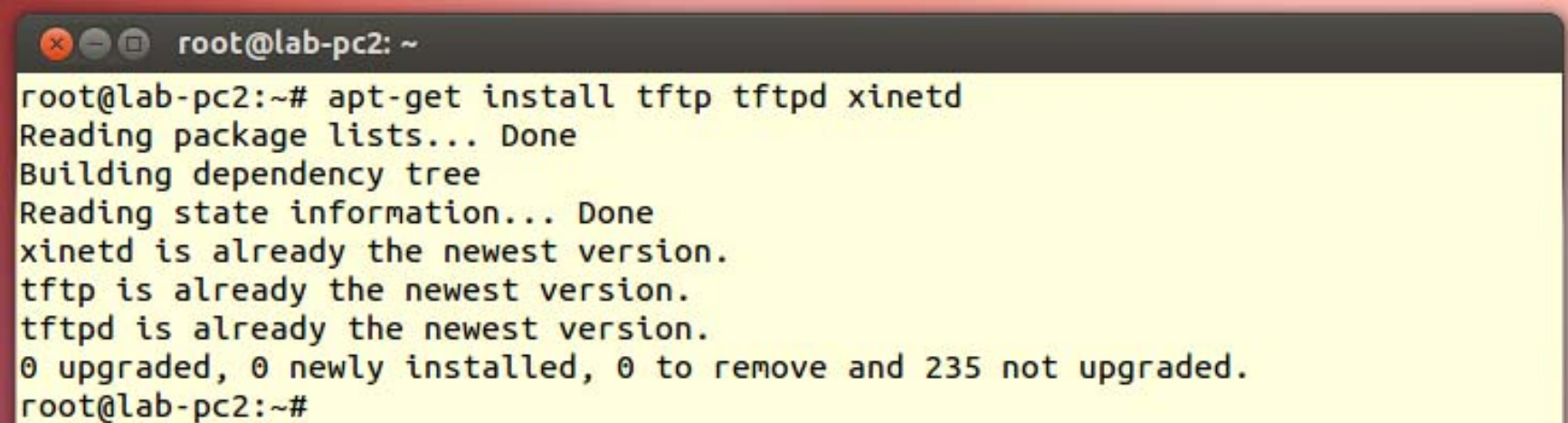
---

- TFTP(Trivial File Transfer Protocol)란?
  - FTP보다 간단하지만 기능이 조금 덜한 네트워크 어플리케이션이다. 이것은 사용자 인증이 불필요하고, 디렉토리를 보여주지 않아도 되는 곳에 사용된다.
  - 부트로더에서 **kernel** 이미지를 **Host**에서 **Target**으로 이더넷포트를 통하여 다운로드 하기 위해 사용한다.

# TFTP-Server 설치하기

---

- 프로그램을 설치
  - **# apt-get install tftp tftpd xinetd**
- 이미 설치되어 있을 경우의 화면



```
root@lab-pc2: ~  
root@lab-pc2:~# apt-get install tftp tftpd xinetd  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
xinetd is already the newest version.  
tftp is already the newest version.  
tftpd is already the newest version.  
0 upgraded, 0 newly installed, 0 to remove and 235 not upgraded.  
root@lab-pc2:~#
```

# TFTP-Server 환경 설정

---

- /etc/xinet.d/tftp 파일 작성 또는 수정
  - # vi /etc/xinetd.d/tftp

```
service tftp
{
protocol = udp
socket_type = dgram
wait = yes
user = root
server = /usr/sbin/in.tftpd
server_args = -s /tftpboot
disable = no
per_source = 11
cps = 100 2
flags = IPv4
}
```

# TFTP-Server 환경 설정

---

- tftpboot 디렉토리 생성
  - tftp가 연결된 디렉토리이다. /tftpboot라는 디렉토리가 없으면 mkdir로 만들어 준다.
  - # mkdir /tftpboot
- tftp가 실행되도록 다음 명령을 실행한다.
  - # /etc/init.d/xinetd restart
  - 또는
  - # service xinetd restart
  - # netstat -a | grep tftp

```
root@lab-pc2:~# netstat -a | grep tftp
udp          0          0 *:tftp          *:*
root@lab-pc2:~# █
```

- TFTP 시험

```
control@lab-pc2:~$ cd /tftpboot
control@lab-pc2:/tftpboot$ cat test.txt
test
control@lab-pc2:/tftpboot$ cd
control@lab-pc2:~$ tftp localhost
tftp> get test.txt
Received 6 bytes in 0.0 seconds
tftp> quit
control@lab-pc2:~$ cat test.txt
test
control@lab-pc2:~$ █
```

- NFS(Network File System)

- 컴퓨터 사용자가 원격지 컴퓨터에 있는 파일을 마치 자신의 컴퓨터에 있는 것처럼 검색하고, 저장하거나 수정하도록 해주는 클라이언트/서버 형태의 메커니즘

- NFS 서버 설치

- `# apt-get install nfs-kernel-server`

- NFS 서버 설정

- 공유디렉토리 생성
  - `# mkdir /nfsroot`
- `/etc/exports` 파일 수정
  - `# vi /etc/exports`

`/nfsroot *(rw,sync,no_root_squash,no_subtree_check)`

## ■ NFS 시작

### ■ NFS 데몬 재시작

- `# /etc/init.d/nfs-kernel-server restart`
- 또는
- `# service nfs-kernel-server restart`
- `# netstat -a | grep nfs`

## ■ NFS 테스트

```
control@lab-pc2:~$ mkdir tmp
control@lab-pc2:~$ sudo mount -t nfs localhost:/nfsroot ./tmp
control@lab-pc2:~$ cd tmp
control@lab-pc2:~/tmp$ ls
ButtonTest      fpga_fnd_driver.ko          fpga_test_push_switch
CanButtonTest   fpga_led_driver.ko          SocketCANexample
CanTest          fpga_push_switch_driver.ko  StopwatchTest
epit_driver.ko  fpga_test_fnd               Test
epit_test       fpga_test_led
control@lab-pc2:~/tmp$ cd ..
control@lab-pc2:~$ sudo umount localhost:/nfsroot
control@lab-pc2:~$ cd tmp
control@lab-pc2:~/tmp$ ls
control@lab-pc2:~/tmp$ cd ..
control@lab-pc2:~$ █
```



---

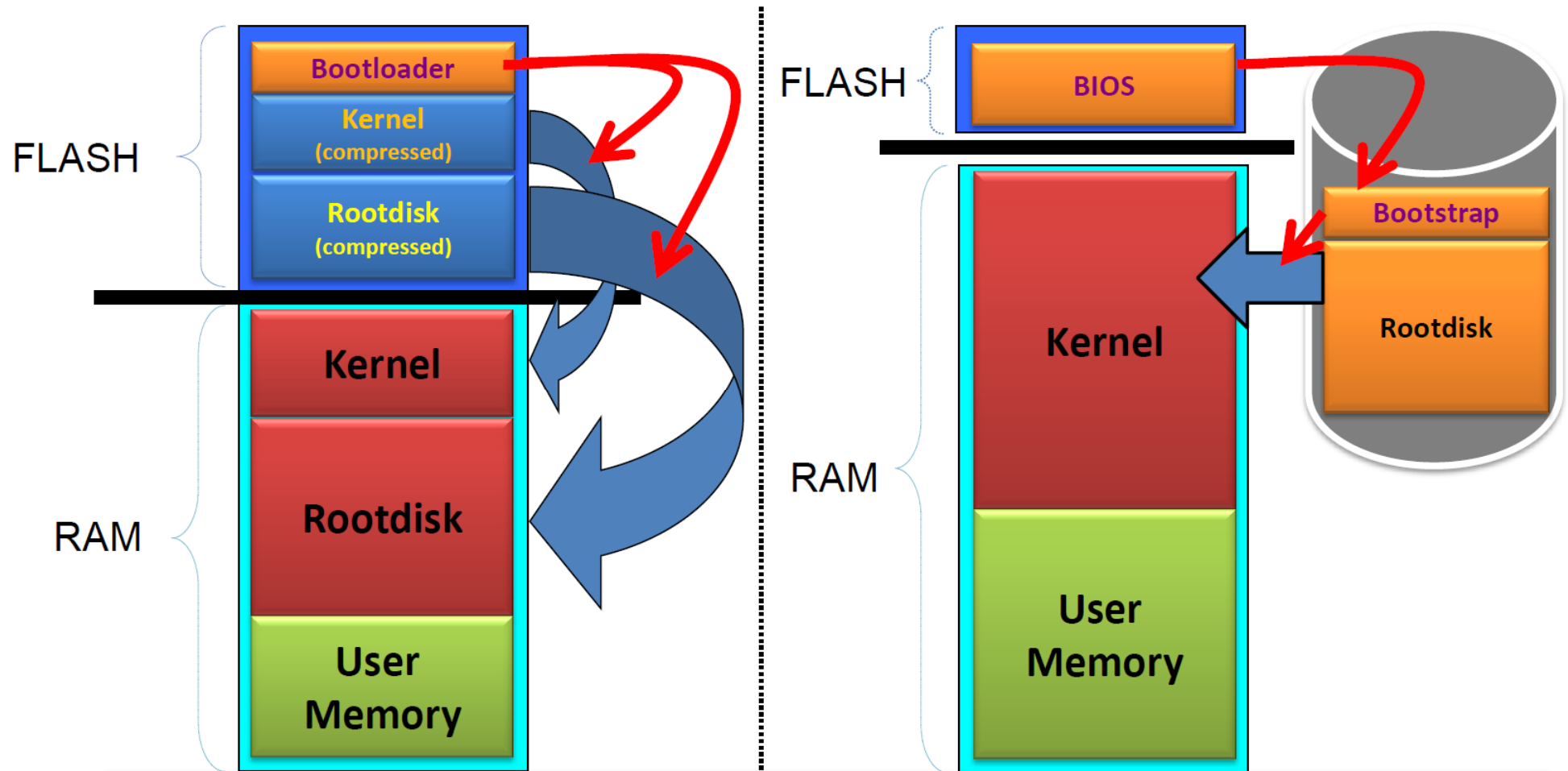
# Bootloader

# Bootloader Overview

---

- Target Board Initialization
  - Memory setting
  - CPU clock setting
  - GPIO setting
  - Serial port setting
  - MAC address and Ethernet port setting
- Kernel Booting
  - Image download from host by tftp
  - Copy image from Flash to ram or from ram to flash.
  - Jump to kernel
  - Command line interface

# Bootloader와 PC BIOS의 비교



# Bootloader의 종류

Bootloader	Video Support	Description	Architectures					
			x86	ARM	PowerPC	MIPS	SuperH	m68k
LILO	No	The main disk bootloader for Linux*	X					
GRUB	No	GNU's successor to LILO	X					
Loadlin	No	Loads Linux from DOS	X					
Etherboot	No	Loader to boot systems through Ethernet cards	X					
CoreBoot	No	Linux-based BIOS (LinuxBIOS) replacement	X					
blob	No	Loader from the LART hardware project		X				
PMON	Yes	Loader used in Agenda VR3				X		
sh-boot	No	Main loader of the LinuxSH project					X	
U-Boot	Yes	Universal loader based on PPCBoot and ARMBoot	X	X	X			
RedBoot	Yes	eCos-based loader	X	X	X	X	X	X

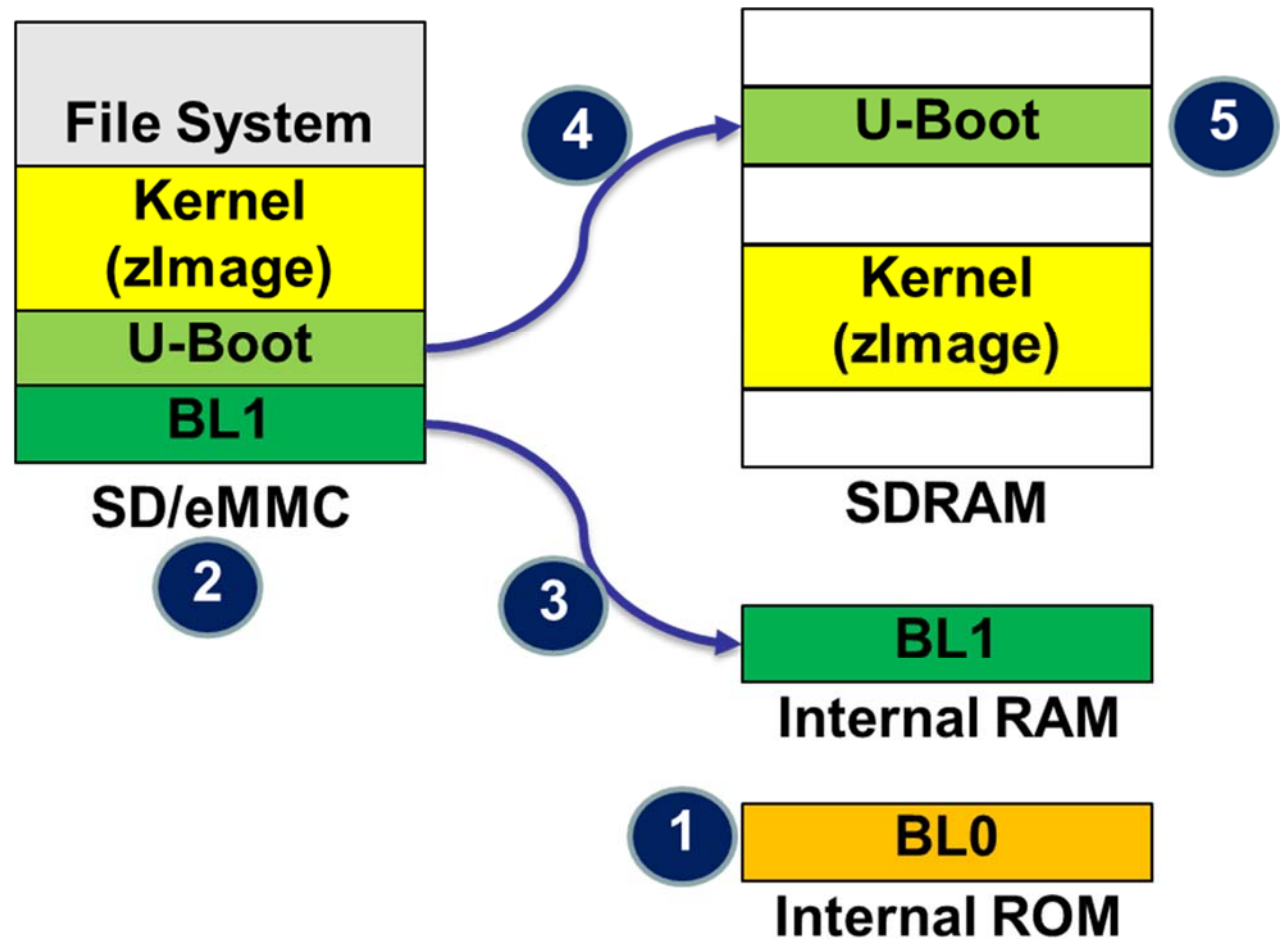
# Bootloader의 종류

---

- x86
  - X86에서는 아래의 2가지 타입이 주로 사용된다. : LILO and GRUB
  - LILO: <http://brun.dyndns.org/pub/linux/lilo/>
  - GRUB: <http://www.gnu.org/software/grub/>
  - 그 외의 bootloader: Rolo, EtherBoot, LinuxBIOS
- Arm architecture
  - U-Boot ARM bootloader의 표준이 되어가고 있다.
    - Armboot로 합쳐진 ppcboot는 u-boot로 진화
    - <http://armboot.sourceforge.net/>
  - BLOB
    - LART hardware project에서 소개된 bootloader
    - 다양한 ARM-based system에 포팅
    - <http://www.lart.tudelft.nl/lartware/blob>

# Booting Sequence

- ① Reset 후 내부 ROM 코드 실행
- ② Operating Mode 에 따라 부트 디바이스 선택
- ③ BL1과 일부 초기화 코드를 내부 RAM으로 복사
- ④ BL1에서 U-Boot 복사
- ⑤ U-Boot 실행 시작



---

**Kernel**

# Kernel Overview

---

- Kernel의 역사
  - UNIX: 1969 Thompson & Ritchie AT&T Bell Labs.
  - BSD: 1978 Berkeley Software Distribution.
  - Commercial Vendors: Sun, HP, IBM, SGI, DEC.
  - GNU: 1984 Richard Stallman, FSF.
  - POSIX: 1986 IEEE Portable Operating System unIX.
  - Minix: 1987 Andy Tannenbaum.
  - SVR4: 1989 AT&T and Sun.
  - Linux: 1991 Linus Torvalds Intel 386 (i386).
  - Open Source: GPL.



# Kernel Overview

---

- Linux의 특징
  - UNIX-like operating system.
  - 특징:
    - Preemptive multitasking.
    - Virtual memory (protected memory, paging).
    - Shared libraries.
    - Demand loading, dynamic kernel modules.
    - Shared copy-on-write executables.
    - TCP/IP networking.
    - SMP support.
    - Open source.

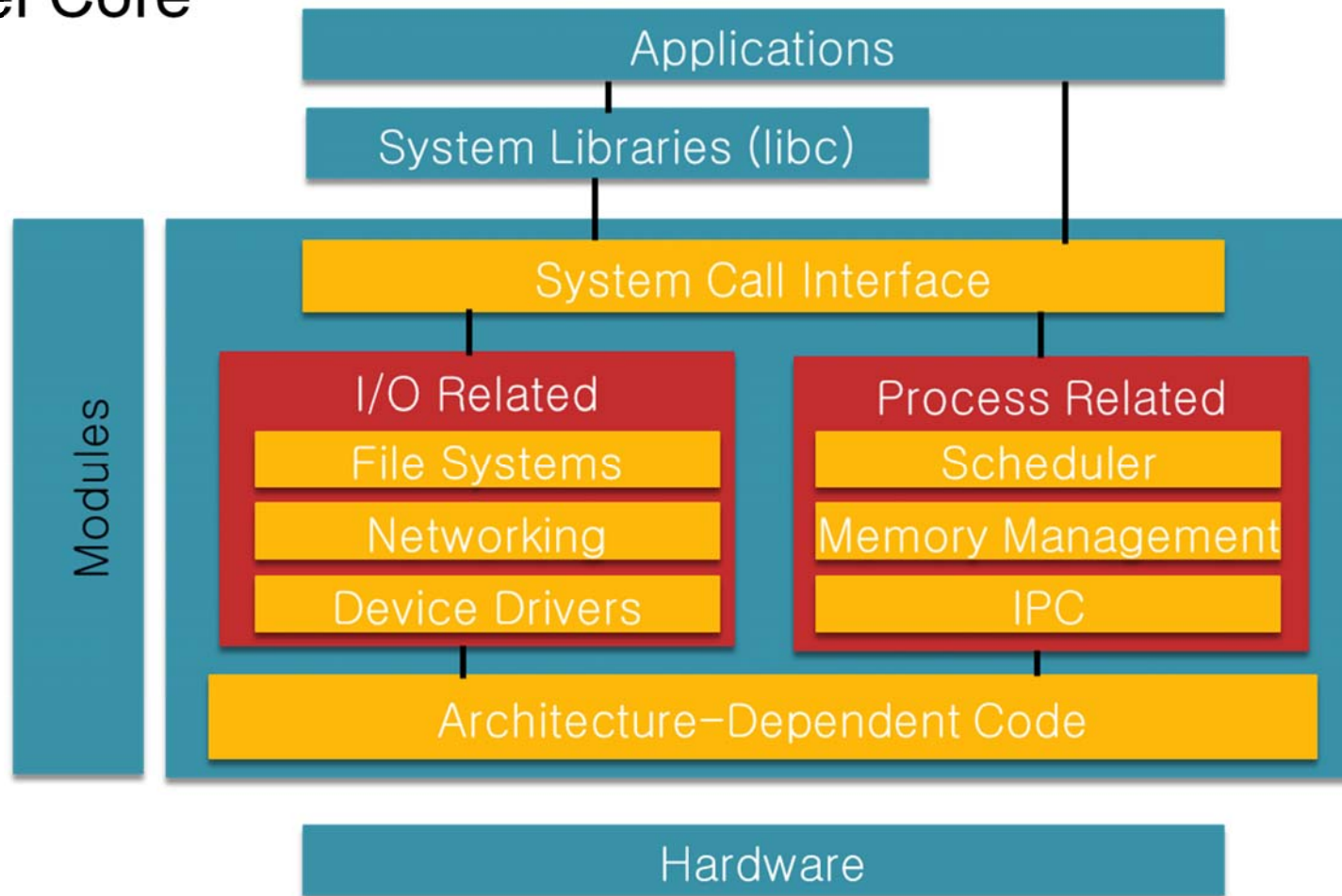
# Kernel Overview

---

- kernel이란?
  - AKA: executive, system monitor.
  - hardware에 접근하는 것을 제어하고 중재한다.
  - 핵심적인 개념을 구현하고 지원한다:
    - Processes, files, devices etc.
  - Schedules / allocates system resources:
    - Memory, CPU, disk, descriptors, etc.
  - Security와 protection 수행.
  - service (system calls)를 위한 사용자 요청에 대한 응답.

# Kernel Overview

- Kernel Core



# Kernel Overview

---

## ■ Linux Kernel

- 운영체제에서 가장 기초적이고 핵심적인 기능을 담당.
- 프로세스의 관리, CPU 스케줄링, 입출력제어
- 메모리, 파일, 주변장치와 같은 시스템 관리
- Monolithic Kernel이지만 Micro Kernel의 특징인 모듈을 이용
  - Monolithic kernel: 하나의 Kernel에 필요한 모든 기능이 통합. 같은 메모리 공간에 필요한 기능이 존재, 함수 호출 방식으로 Kernel에서 제공하는 기능에 접근. 구현이 간편하고 효율적, 포팅과 확장이 어렵다.
- Micro kernel
  - 필요한 기능들은 작은 서버 모듈로 나뉘어 설계, 서버를 관리할 수 있는 최소의 크기, 하드웨어 환경에 따라 기능의 확장과 기능 재구성이 용이. 단점은 서비스를 사용하는 과정에서 여러 번의 메시지 전송과 Context Switching이 발생. 높은 자원 사용.

# Linux Kernel

---

## ■ Process Management

- 하나의 프로그램은 하나 이상의 프로세스를 가질 수 있다.
- **Kernel**은 해당 프로세스가 시스템 자원을 분배하고, 다른 자원을 침해할 수 없도록 관리
- 프로세스의 생성과 소멸에 대한 전반적인 내용을 관리한다.

## ■ Memory Management

- 시스템에서 구동되는 프로세스는 프로세스만의 독립적인 메모리를 갖는다.
- **Linux**는 물리메모리에 직접적으로 데이터를 기록하는 것을 허용하지 않는다
- 반환된 메모리의 관리 등과 같은 기능을 수행

# Linux Kernel

---

## ■ File System Management

- Linux 는 범용운영체제로 다양한 파일시스템을 지원
- Linux 에서 모든 장치는 파일로 표현된다.
- CDFS, FAT, JFFS, YAFFS, FAT16/32, EXT3/4, NTFS등을 지원

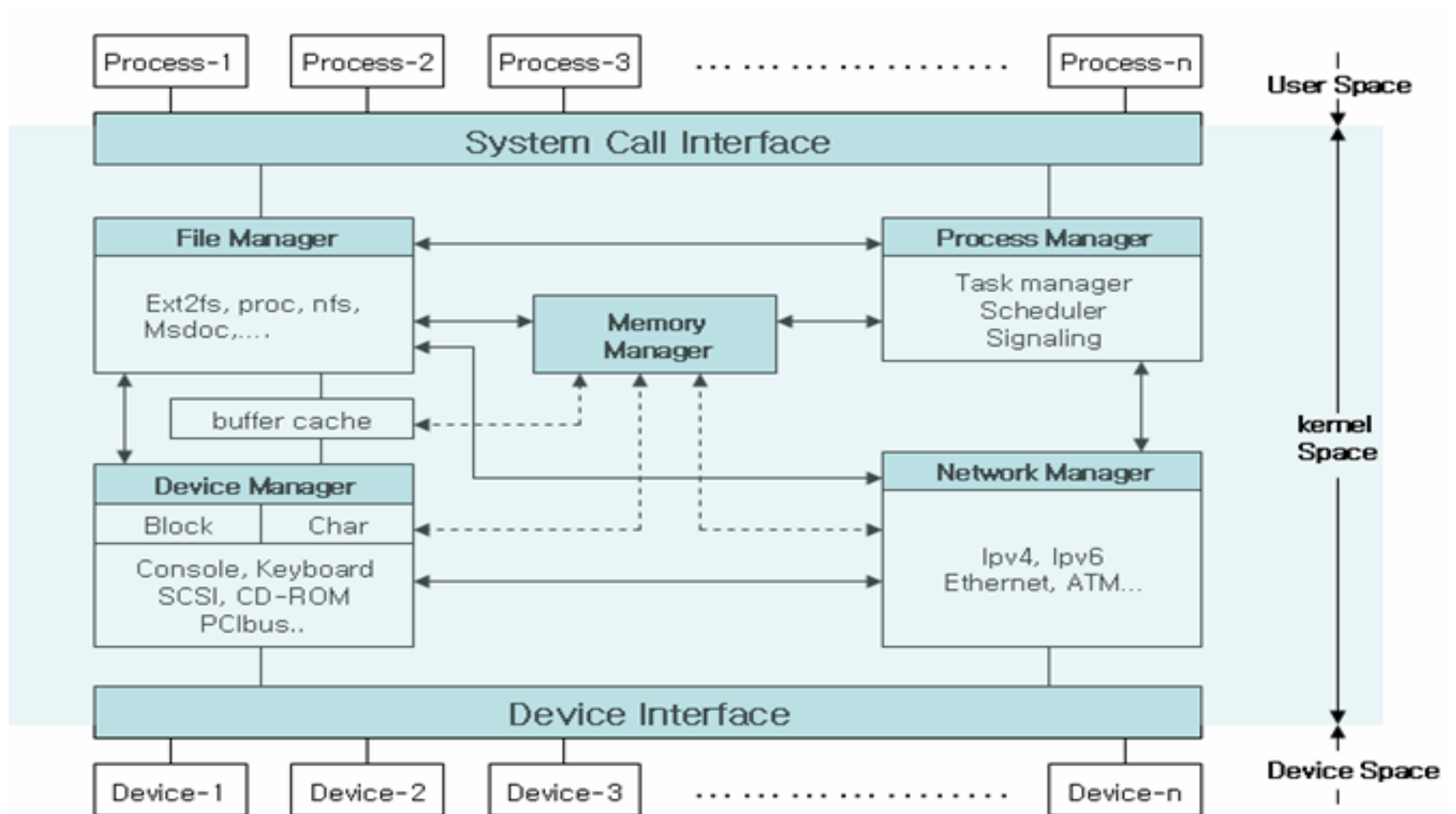
## ■ Device Management

- Bootloader에서 초기화된 이후 모든 디바이스는 Kernel 의 관리를 받는다.
- 응용 프로그램에서 디바이스를 이용해야 하는 경우 Kernel 을 통해서만 제어가 가능하다.
- 사용하고자 하는 장치 드라이버에 따라서 문자, 블록 방식으로 나뉘게 된다.

## ■ Network Management

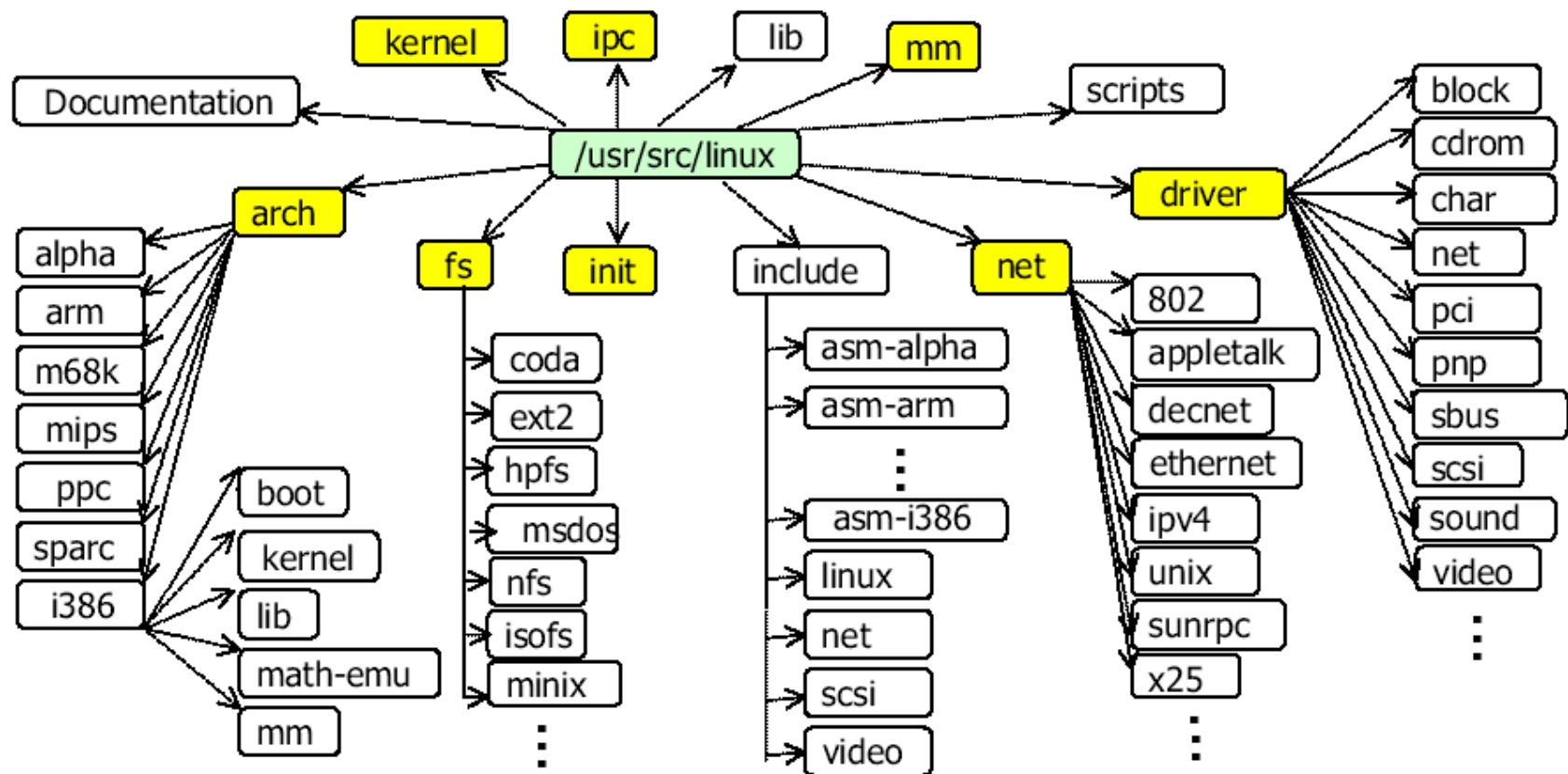
- Kernel 은 기기와 외부시스템이 정상적인 네트워크를 통해서 통신할 수 있도록 한다.
- IPv4, IPv6, Ethernet, ATM, CDMA등이 지원된다.

# Linux Kernel



# Kernel Source Tree

- Kernel source
  - [www.kernel.org](http://www.kernel.org)
  - Kernel의 대부분은 C로 작성되어 있다.
- Kernel Directories





# Kernel Source Tree

---

- **arch/**
  - CPU dependent (arch/i386, arch/alpha, arch/arm... )
  - arch/arm/boot/
    - Boot Strap
  - arch/arm/kernel/
    - Hardware dependent kernel management routines
    - Trap, Interrupt processing routines
    - Context Switching routines
    - Device configuration, initialization
  - arch/arm/mm/
    - Hardware dependent memory management routines
- **init/**
  - Hardware independent kernel initialization (start\_kernel)
    - Task 0 (init\_task or task[0]) creation
    - Demon process creation - Task 1, 2, 3 and so on

# Kernel Source Tree

---

- **kernel/**
  - Central section of the kernel
  - Hardware independent kernel management routines
    - fork, exit
    - Scheduler
    - Signal handling
    - Time management
- **mm/**
  - Hardware independent memory management routines
    - Virtual Memory Management
    - Paging / Swapping

# Kernel Source Tree

---

- **fs/**
  - Virtual file system Management
  - open, read – file system related system call routines
  - Subdirectories for special file systems (ext2, fat, ... )
  
- **ipc/**
  - IPC between processes
    - Semaphores
    - Shared memory
    - Message queues

# Kernel Source Tree

---

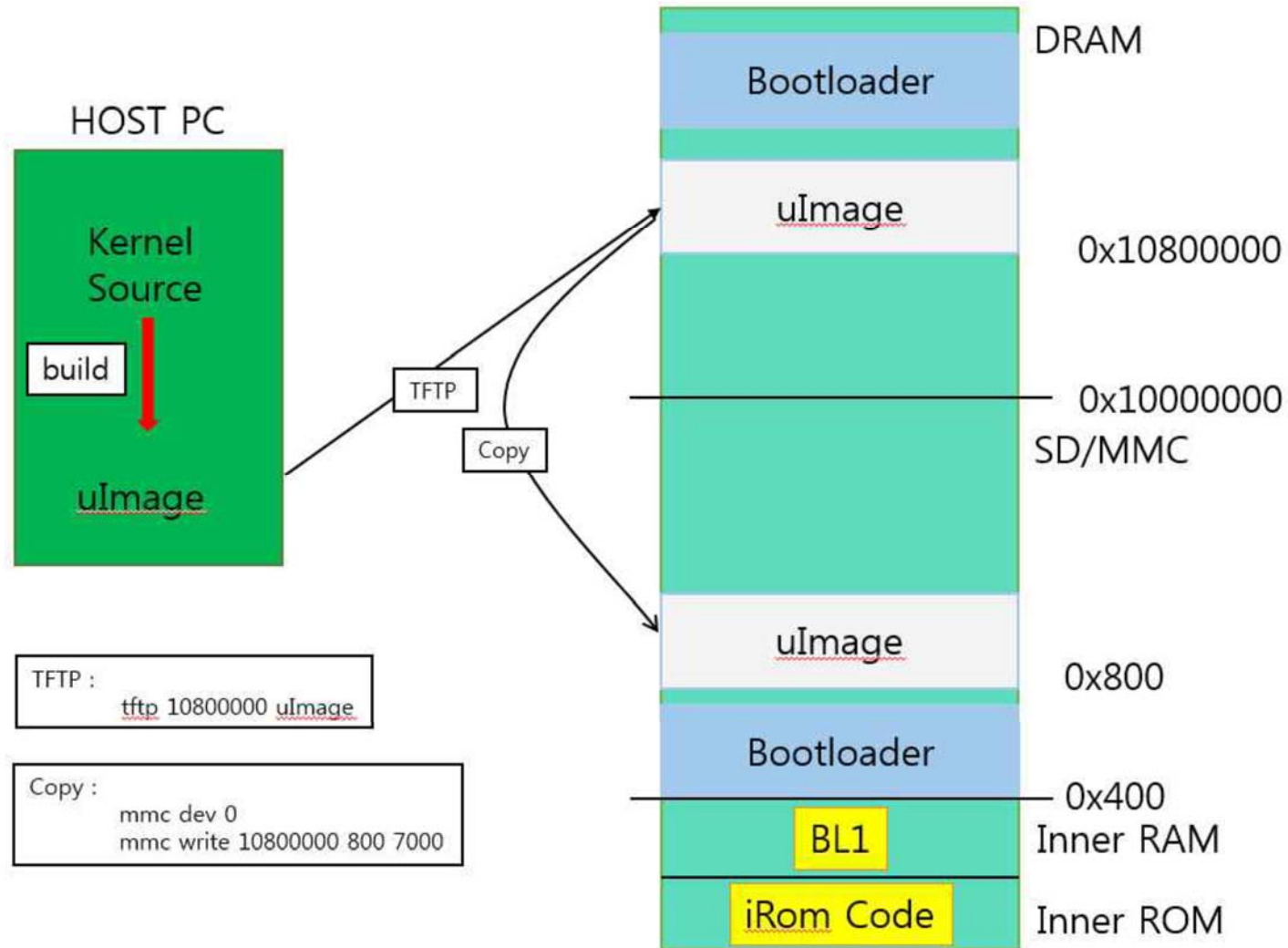
- **drivers/**
  - Device driver routines
  - drivers/block/ : Block device driver
  - drivers/char/ : Character device driver
  - drivers/net : Network device driver
  - drivers/pci/ : PCI bus control
  - drivers/sound/ : sound card device driver
  - drivers/cdrom/ : CD-ROM device driver
  - drivers/scsi/ : SCSI device driver
- **net/**
  - Network protocols : TCP/IP, ARP and so on
  - Socket interface

# Kernel Source Tree

---

- **include/**
  - Header files for the kernel
    - Hardware independent : include/linux/
    - Hardware dependent : include/asm-\*/
- **modules/**
  - Kernel module routines
  - insmod(modprobe), rmmod : dynamic load and removal
- **lib/**
  - Kernel library routines
- **Doc/**
  - Kernel document

# Kernel Image Writing



---

# Root File System

for Embedded Linux

# Minimum Components

---

- **init:** The program that starts everything off, usually by running a series of scripts.
- **shell:** Needed to give you a command prompt but, more importantly, to run the shell scripts called by init and other programs.
- **daemons:** Various server programs, started by init.
- **libraries:** Usually, the programs mentioned so far are linked with shared libraries which must be present in the root file system.
- **Configuration files:** The configuration for init and other daemons is stored in a series of ASCII text files, usually in the /etc directory.



# Minimum Components

---

- Device nodes: The special files that give access to various device drivers.
- /proc and /sys: Two pseudo file systems that represent kernel data structures as a hierarchy of directories and files. Many programs and library functions read these files.
- kernel modules: If you have configured some parts of your kernel to be modules, they will be here, usually in /lib/modules/[kernel version].

# Directory Layout

---

- /bin: programs essential for all users
- /dev: device nodes and other special files
- /etc: system configuration
- /lib: essential shared libraries, for example, those that make up the C library
- /proc: the proc files system
- /sbin: programs essential to the system administrator
- /sys: the sysfs file system
- /tmp: a place to put temporary or volatile files
- /usr: as a minimum, this should contain the directories /usr/bin, /usr/lib and /usr/sbin, which contain additional programs, libraries, and system administrator utilities
- /var: a hierarchy of files and directories that may be modified at runtime, for example, log messages, some of which must be retained after boot

# Programs for the root file system

---

- The init program
  - You have seen in the previous chapter that init is the first program to be run and so has PID 1. It runs as the root user and so has maximum access to system resources. Usually, it runs shell scripts which start daemons: a daemon is a program that runs in the background with no connection to a terminal, in other places it would be called a server program.
- Shell
  - We need a shell to run scripts and to give us a command-line prompt so that we can interact with the system. An interactive shell is probably not necessary in a production device, but it is useful for development, debugging, and maintenance.

# Programs for the root file system

---

- Utilities
  - The shell is just a way of launching other programs and a shell script is little more than a list of programs to run, with some flow control and a means of passing information between programs. To make a shell useful, you need the utility programs that the Unix command-line is based on. Even for a basic root filesystem, there are approximately 50 utilities, which presents two problems. Firstly, tracking down the source code for each and cross compiling it would be quite a big job. Secondly, the resulting collection of programs would take up several tens of megabytes, which was a real problem in the early days of embedded Linux when a few megabytes was all you had. To solve this problem, BusyBox was born.
- BusyBox

# Files for the root file system

---

- Libraries for the root file system
- Device nodes

```
$ sudo mknod -m 666 dev/null c 1 3
```

```
$ sudo mknod -m 600 dev/console c 5 1
```

```
$ ls -l dev
```

```
total 0
```

```
crw----- 1 root root 5, 1 Oct 28 11:37 console
```

```
crw-rw-rw- 1 root root 1, 3 Oct 28 11:37 null
```

# Build Systems

---

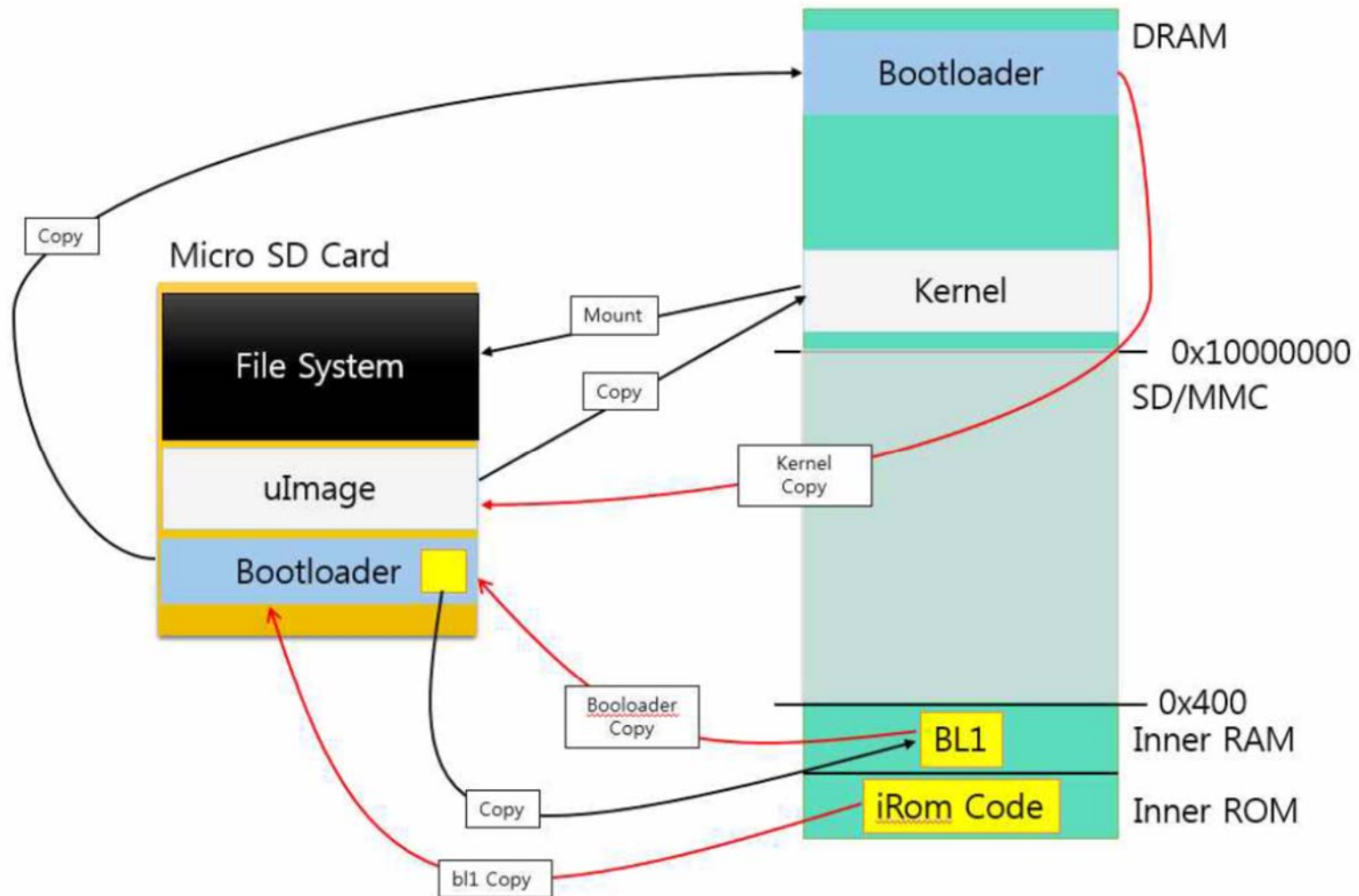
- The idea of a build system is to automate all the steps. A build system should be able to build, from upstream source code, some or all of the following:
  - The toolchain
  - The bootloader
  - The kernel
  - The root file system

# Build Systems

---

- **Buildroot: An easy-to-use system using GNU make and Kconfig (<http://buildroot.org>)**
- EmbToolkit: A simple system for generating root filesystems; the only one at the time of writing that supports LLVM/Clang out of the box (<https://www.embtoolkit.org>)
- OpenEmbedded: A powerful system which is also a core component of the Yocto Project and others (<http://openembedded.org>)
- OpenWrt: A build tool oriented towards building firmware for wireless routers (<https://openwrt.org>)
- PTXdist: An open source build system sponsored by Pengutronix ([http://www.pengutronix.de/software/ptxdist/index\\_en.html](http://www.pengutronix.de/software/ptxdist/index_en.html))
- Tizen: A comprehensive system, with emphasis on mobile, media, and in-vehicle devices (<https://www.tizen.org>)
- **The Yocto Project: This extends the OpenEmbedded core with configuration, layers, tools, and documentation: probably the most popular system (<http://www.yoctoproject.org>)**

# SD Booting





# NFS Booting

