
교차 개발 환경

- Chapter 05 -

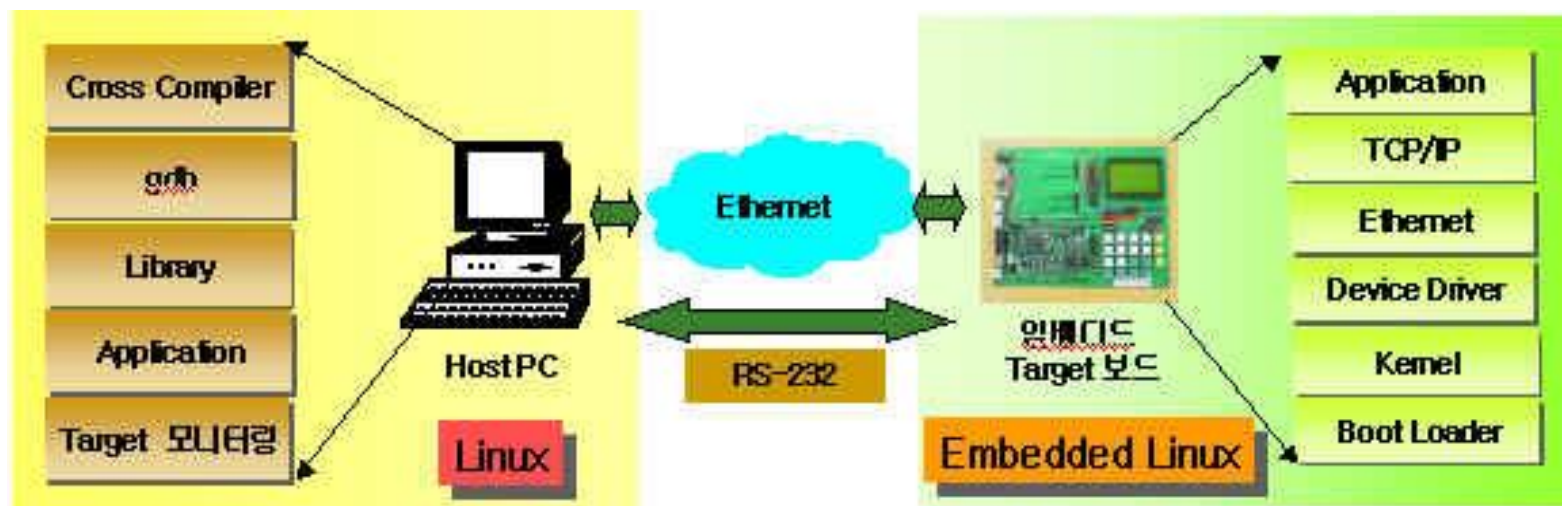
목차

- I. 교차 개발 환경
- II. 교차 개발 환경 도구
- III. 툴체인(Toolchain)
- IV. 포팅(Porting)
- V. Target Booting 절차

교차 개발 환경

▶ 교차개발환경(Cross-Development Environment)

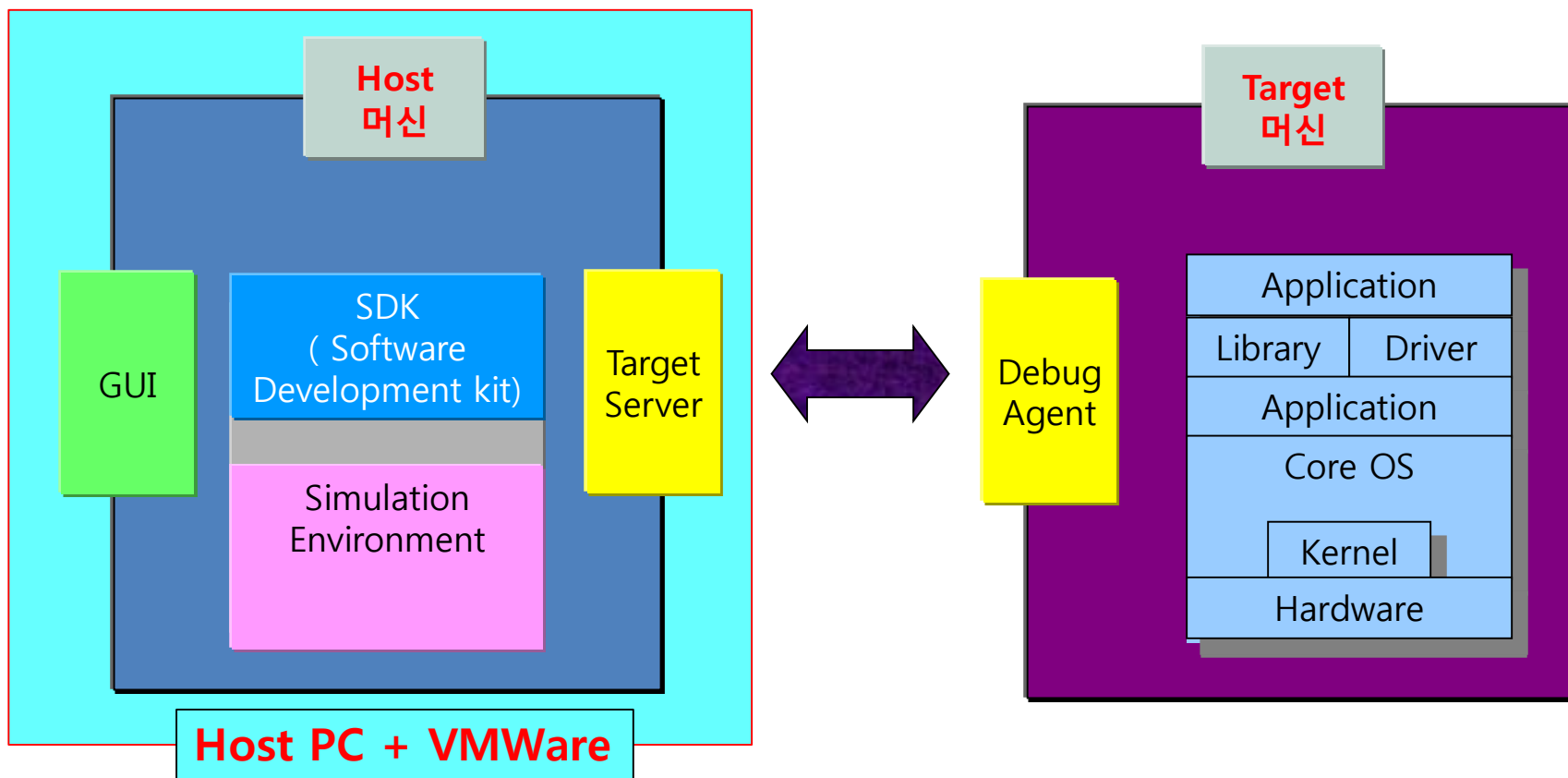
- ▶ 실제 S/W가 수행될 시스템과 개발하는 시스템이 다른 개발 환경
- ▶ 연산능력이 풍부한 호스트 컴퓨터에서 타겟에서 동작할 응용 프로그램을 개발(효율성이 높음)
 - ▶ 타겟은 메모리 용량이 적어서 타겟 내에 compiler를 탑재하기에 무리
- ▶ 호스트와 타겟에 사용되는 프로세서가 다른 경우, 실행은 호스트에서 되지만 만들어진 코드는 타겟 시스템에서 돌아갈 수 있는 컴파일러가 필요 → Cross Compiler라고도 함.
- ▶ 호스트와 타겟을 물리적으로 연결하는 다양한 방법 존재



교차 개발 환경

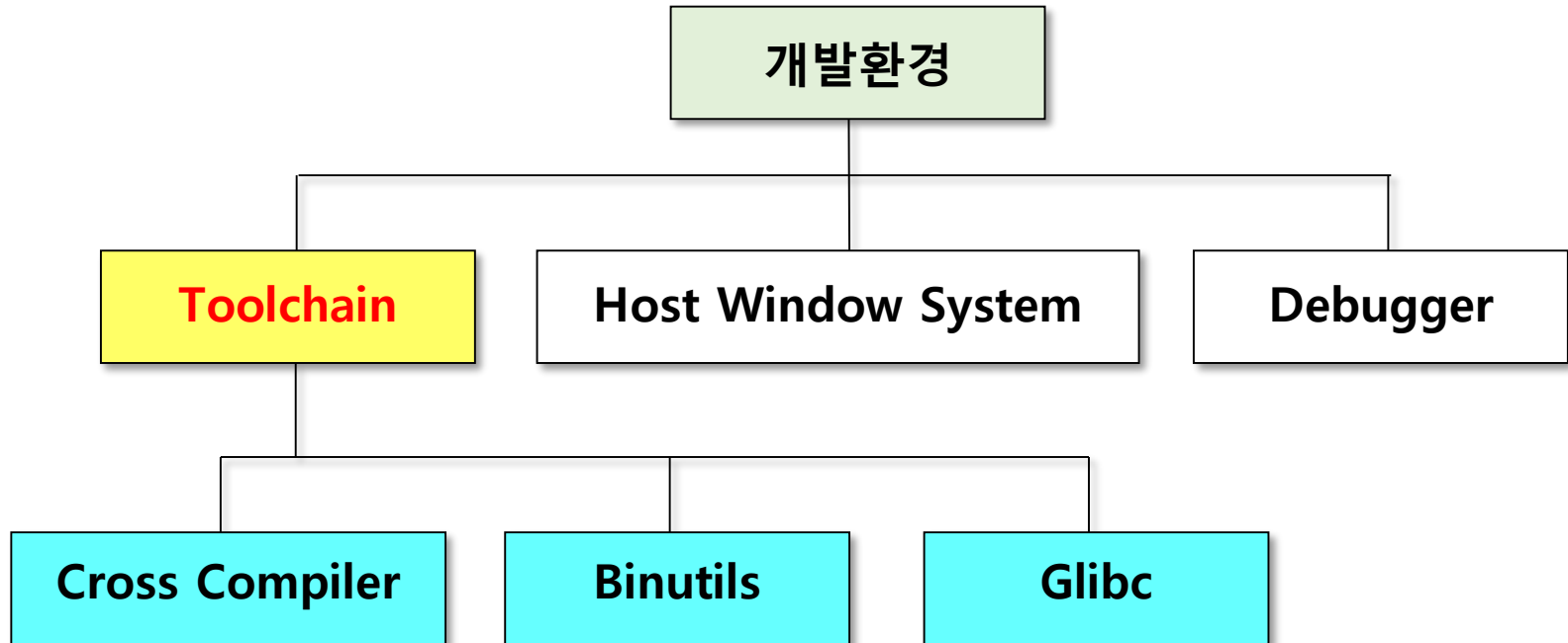
▶ 임베디드 리눅스 개발환경 개념도

- ▶ Host PC에서 VMWare를 이용해 리눅스 호스트 시스템(호스트 머신)을 설치
- ▶ 리눅스 호스트 시스템 상에서 컴파일/개발을 해서 타겟 보드(타겟 머신)로 다운로드, 설치, 실행



교차 개발 환경

▶ 교차 개발 도구 계층도



교차 개발 환경

▶ Target machine (타겟 머신=타겟 디바이스)

▶ 개발 보드, 포팅 대상인 보드나 H/W로 독립된 동작이 되지 않는 머신

- ▷ 독립된 동작을 위해서는 커널, 디바이스 드라이버, 응용 프로그램 등이 Host 머신을 통해서 개발된 뒤에 다운로드나 이식이 되어야 함

▶ Host machine (호스트 머신=호스트 디바이스)

▶ H/W 시스템을 제작한 후, 해당 시스템에서 동작할 S/W를 개발하는 시스템

▶ 개발된 H/W 시스템에 맞는 최적화된 시스템을 설치하여야 함

▷ 교차 컴파일러(Cross Compiler)

- Host 머신에서 컴파일하나 Target 머신 CPU 의존적인 코드 생성

▷ 교차 디버거(Cross Debugger)

- Target 머신에서 수행중인 프로그램을 Host 머신에서 관찰, 디버깅

▷ 교차 라이브러리(Glibc))

▷ 유틸리티(Binutils)

교차 개발 환경 도구

▶ 시리얼 케이블 (Serial Cable)

- ▶ Host 머신과 Target 머신의 정보 교환 및 상태 파악을 위한 연결에 이용
- ▶ Target 머신에 없는 콘솔을 Host 머신을 통해 사용할 수 있게 함
- ▶ Minicom 프로그램: 타겟 머신의 터미널로 사용
 - ▶ 우리 과정에서는 USB-Serial 케이블을 사용

▶ 이더넷 케이블 (Ethernet Cable)

- ▶ Host 머신과 Target 머신의 정보 교환 및 상태 파악을 위한 연결에 이용
- ▶ TFTP 등과 같은 프로토콜을 이용하여 고속의 데이터 통신이 가능하게 함
- ▶ NFS 프로토콜을 이용하여 타겟 머신에서 호스트 머신의 하드 디스크에 있는 파일을 공유할 때 사용
- ▶ Telnet 등과 같은 프로그램을 이용하여 원격에서 Target 머신을 부팅하거나 이용할 때 사용
- ▶ 최근에는 USB 케이블로 대체 되는 추세

교차 개발 환경 도구

▶ USB 케이블

- ▶ 대용량 데이터 전송을 위해 사용 (부트로더, 커널, 파일시스템 등)
- ▶ USB-JTAG 케이블

▶ USB-OTG 2.0 Cable

- ▶ Android 개발 시 Host 머신과 Target 머신의 정보 교환에 이용
 - ▶ Boot loader와 kernel image download 등에 사용
- ▶ 리눅스 커널을 마이크로 SD에 기록할 때도 사용

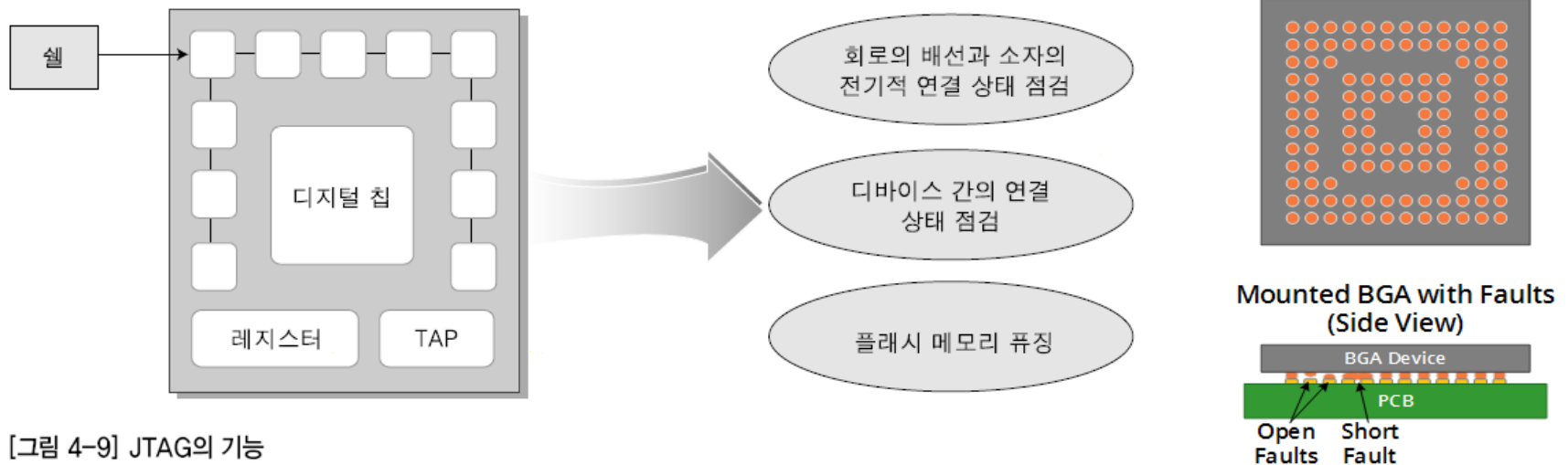
▶ micro SD 카드

- ▶ Boot loader, kernel image, root file system 저장용으로 사용
- ▶ 우리 개발환경에서는 리눅스용 시스템은 외부 Micro SD 카드에, 안드로이드용 시스템은 타겟 보드 내부의 eMMC에 설치

교차 개발 환경 도구

▶ JTAG(Joint Test Access Group) 케이블

- ▶ JTAG은 칩 내부에 Boundary-Scan Cell을 두어 외부 핀과 일대 일로 연결시켜 프로세서가 인위적인 동작을 수행할 수 있도록 하여 하드웨어(CPU 내부) 및 연결(납땜) 상태 등을 점검
 - ▷ Boundary-Scan이라고도 함
- ▶ JTAG의 칩 점검 방식은 1990년도에 IEEE에 표준화되어 IEEE 1149.1로 제정
- ▶ 부트로더를 타겟 시스템의 플래시 메모리에 최초로 퓨징 하려면 JTAG 유틸리티가 꼭 필요
 - ▷ 우리의 실습 과정은 특수하게 JTAG 방식 대신에 USB-Serial 케이블을 사용



[그림 4-9] JTAG의 기능

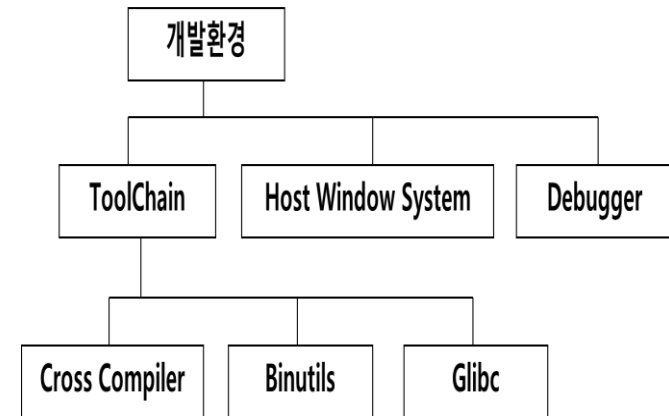
Toolchain

▶ 정의

- ▶ Target device의 Software의 개발을 진행하기 위해 필요한 host system의 cross compile 환경을 뜻함
- ▶ 소스를 컴파일하여 바이너리 실행 파일을 생성하기 위해 필요한 컴파일러 및 라이브러리, 바이너리 유틸리티 모음

▶ 구성요소

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



Toolchain

▶ Host Computer 환경 설정: TFTP

- ▶ TFTP(Trivial File Transfer Protocol)
- ▶ 시스템이 단순하고 간단함
- ▶ 어떤 형태의 전달 서비스 상에서도 동작 가능
- ▶ 시스템 가격이 저렴
- ▶ 정보 보호 기능이 없음
- ▶ UDP 프로토콜을 사용하기 때문에 데이터에 대한 보장성이 없음

파일 전송 프로토콜	FTP	TFTP
사용자 인증	필요	불필요
서버/클라이언트 통신	연결형 프로토콜인 TCP	비연결형 프로토콜인 UDP
파일 전송 디렉토리	임의 장소	특정 장소
사용 시스템	데스크탑, 워크스테이션 등	네트워크 컴퓨터 및 간단한 기기

Toolchain

▶ TFTP 설치 (tftpd)

▶ tftp download

```
# apt-get install xinetd tftpd tftp
```

▶ tftp 설정

```
# 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
}
```

Toolchain

▶ TFTP 설치 (tftpd)

▶ tftp 서비스 디렉터리 생성

```
# mkdir /tftpboot
```

▶ 서비스 재시작 (수퍼 데몬 실행)

```
# service xinetd restart
```

Toolchain

▶ TFTP 테스트

- ▶ tftp 디렉터리로 이동 후 파일 생성 및 홈 디렉터리로 이동

```
# cd /tftpboot
# cat > /tftpboot/test.txt
Hello World
(상기 내용을 입력한 후에 Ctrl + D를 눌러 입력 내용 저장)

# cd /root
```

- ▶ tftp 실행 후 get 명령을 통해 /tftpboot의 test.txt 파일을 다운로드 후 종료

```
# tftp localhost
tftp> get test.txt
tftp> quit
```

- ▶ 다운로드 받은 파일 내용 확인

```
# ls /root
# cat test.txt
Hello World
```

```
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
}
```

Toolchain

▶ 데몬(Daemon: 서버 프로세스)

- ▶ 프로그램을 실행할 때부터 사용자에게 의해 직접적으로 제어 받지 않는 특별한 프로세스
- ▶ 대부분의 데몬 프로세스는 마지막에 d라는 문자를 포함
- ▶ 일반적으로 백그라운드 모드로 수행하며 사건을 기다리거나 주기적으로 주어진 작업을 수행
- ▶ 리눅스는 다양한 기능과 서비스를 제공하기 위해 많은 데몬 프로세스를 실행
- ▶ pstree 명령을 실행하면 현재 수행 중인 데몬 파악 가능
- ▶ system-config-services나 ntsysv 명령을 사용해 서비스 데몬의 실행 조정
- ▶ 독립적 수행 여부에 따라 독립형 데몬과 종속형 데몬으로 분류

Toolchain

▶ 독립형 데몬

- ▶ 웹 서버(HTTPD), DB 서버(MYSQLD)와 같이 시스템에 독자적으로 프로세스가 구동되어 서비스를 제공
- ▶ 자주 호출되는 서비스의 경우 메모리에 상주시켜 독립형 데몬으로 사용하는 것이 적절
- ▶ 독립형 데몬의 실행 스크립트 파일은 대부분 /etc/init.d/ 디렉토리에 존재

```
service <서비스 이름> {start | restart | stop}
```

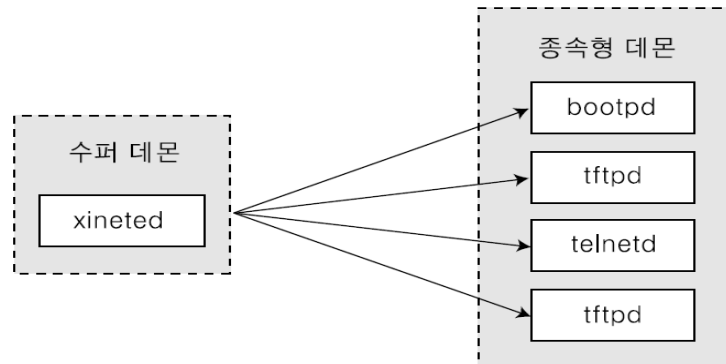
또는

```
/etc/init.d/<서비스 이름> {start | restart | stop}
```


Toolchain

▶ 종속형 데몬

- ▶ 작은 규모의 서비스는 종속형 데몬을 사용하는데 필요 시에만 수퍼 데몬이 이들을 작동시킴
 - ▷ 수퍼 데몬: 사용 빈도수가 적은 데몬 프로세스를 관리하는 특수한 데몬
- ▶ 수퍼 데몬 xinetd는 /etc/xinetd.d/<데몬 프로그램 이름>이라는 다수의 스크립트 파일 사용
 - ▷ 그림 설명: 평소에는 4개의 데몬 프로세스가 잠자고 있지만, 타겟 시스템이 호스트 시스템에서 서비스를 요청하면 수퍼 데몬을 통하여 서비스 제공
 - ▷ xinetd = eXtended Internet Daemon
- ▶ 종속형 데몬 관련 설정 파일을 수정하면 수퍼 데몬을 반드시 재실행

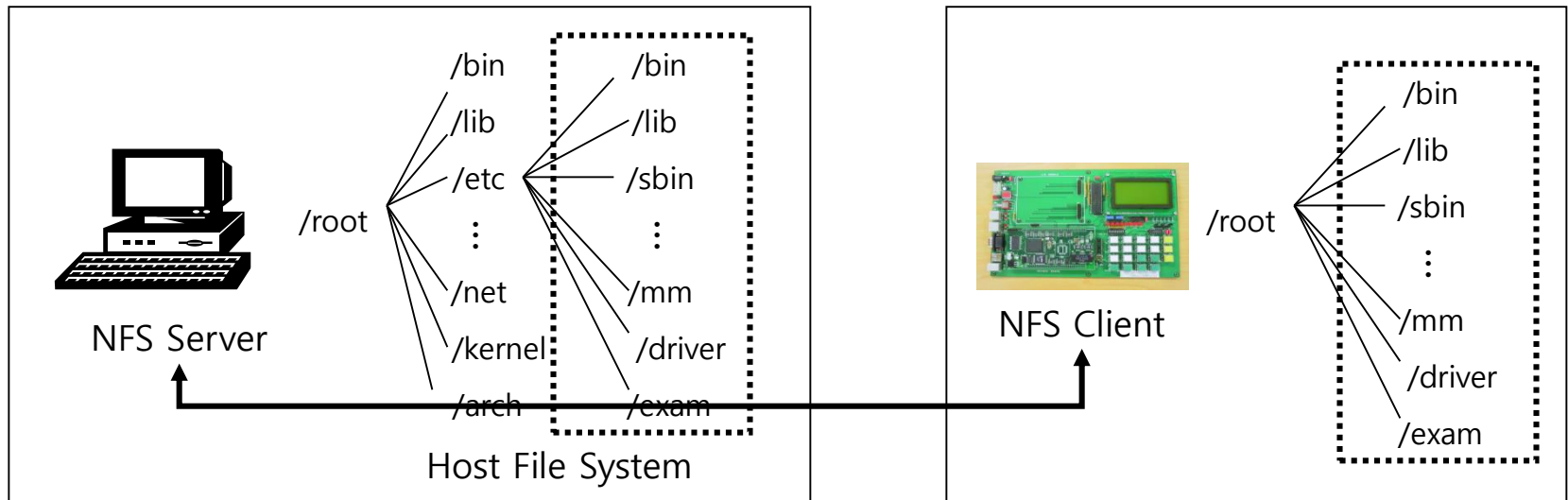


[그림 5-9] 수퍼 데몬과 종속형 데몬의 관계

Toolchain

▶ Host Computer 환경 설정: NFS(Network File System)

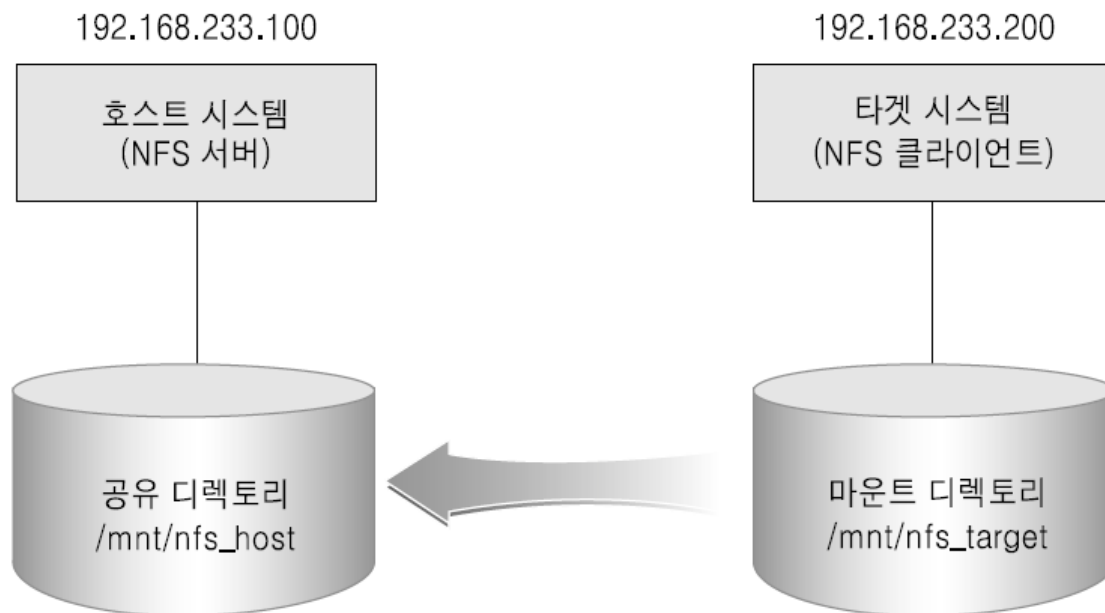
- ▶ 원격지의 컴퓨터에 있는 파일을 마치 자신의 컴퓨터에 있는 것처럼 이용
- ▶ SUN Microsystem사가 개발한 RPC(Remote Procedure Call) 기반 시스템
- ▶ 네트워크를 이용하는 Server/Client 기반 응용 프로그램
- ▶ File System이 존재하지 않는 Client 시스템에서 원격의 Host 시스템에서 설정된 일부 디렉터리를 이용 → 서버에 위치한 용량이 큰 자료도 마치 자신(클라이언트)의 자료처럼 사용
- ▶ 임베디드 시스템 개발 시 많이 이용됨



Toolchain

▶ Host Computer 환경 설정: NFS

- ▶ 타겟 시스템에서 필요한 프로그램을 호스트 시스템에서 개발하고 NFS를 사용하면 전송의 번거로움이 없이 타겟 시스템에서 사용 가능



[그림 5-36] NFS 서버/클라이언트 구성

Toolchain

▶ Host Computer 환경 설정: NFS

▶ 장점

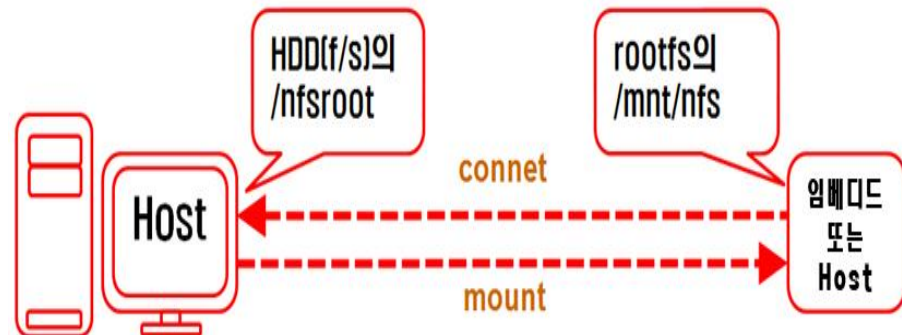
- ▶ 호스트 시스템에서 작업한 것을 NFS를 이용하면 다운로드 과정 없이 타겟 시스템에서 이용 가능
- ▶ 클라이언트 시스템의 리눅스 파일 시스템 위에서 호스트 시스템의 파일을 접근 및 실행이 가능
- ▶ 램디스크 상에 올리기에 너무 큰 파일도 NFS 상에서는 호스트의 기억 용량에 의존하기 때문에 쉽게 처리 가능

▶ 단점

- ▶ 네트워크의 과부하로 속도 저하 혹은 보안 허점 우려
- ▶ 장치 파일과 같은 특수 파일은 NFS에 연결된 디렉토리에 생성 불가
- ▶ 읽고 쓰는 속도가 빠른 파일로는 사용이 곤란

Toolchain

▶ NFS(Network File System) 설치



▶ NFS 다운로드

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

▶ nfs 서버 설정 파일 수정

```
# vim /etc/exports
```

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

- NFS 서버에서의 공유할 디렉토리 설정
- RW 가능
- 지속적으로 싱크를 맞춤
- 클라이언트도 서버와 동일한 루트 권한
- 속도향상을 위해 시스템 접근시마다 하위 디렉토리를 조사하는 기능(보안)을 생략

▶ nfs 디렉터리 생성

```
# mkdir /nfsroot
```

▶ nfs 서비스 재시작

```
# service nfs-kernel-server restart
```

Toolchain

▶ NFS 테스트

▶ NFS 서비스가 구동되면 localhost(자신의 IP)를 nfs로 연결해서 정상적으로 연결되는지 확인

▷ 연결할 디렉터리 생성

```
# mkdir /mnt/nfs
```

NFS 클라이언트의 디렉토리 설정

▷ 연결(마운트)

```
# mount -t nfs localhost:/nfsroot /mnt/nfs
```

NFS 서버의 nfsroot 디렉토리를 NFS 클라이언트의 디렉토리 /mnt/nfs로 연결

▷ nfs 서비스 디렉터리(/nfsroot)로 이동하여 파일을 생성

```
# cd /nfsroot  
# touch test
```

NFS 서버에서 test라는 파일 생성

▷ 연결된 /mnt/nfs에 동일한 파일이 출력되는지 확인

```
# ls /mnt/nfs
```

NFS 클라이언트에서 보이는지 확인

▷ 연결 해제

```
# umount /mnt/nfs
```

Toolchain

▶ Host Computer 환경 설정: Minicom

- ▶ Embedded 타겟 시스템은 대부분 출력을 위한 별도의 터미널을 가지고 있지 않음
- ▶ 따라서 일반적으로 serial port를 통한 터미널 프로그램을 이용
 - ▶ Linux에서는 일반적으로 minicom 이용
 - ▶ Windows 에서는 Hyper-Terminal 등을 이용
- ▶ Minicom이란?
 - ▶ 호스트와 타겟을 연결해주는 가상터미널 프로그램이다. 하이퍼터미널과 같은 개념이고, 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. ➔ 우리는 USB-serial 케이블을 쓰므로 설정이 일반적인 방법과 다소 다름
- ▶ Minicom 용도
 - ▶ 부트로더의 명령 프롬프트를 위한 콘솔
 - ▶ 임베디드 리눅스의 쉘 프롬프트를 위한 콘솔로 사용

Toolchain

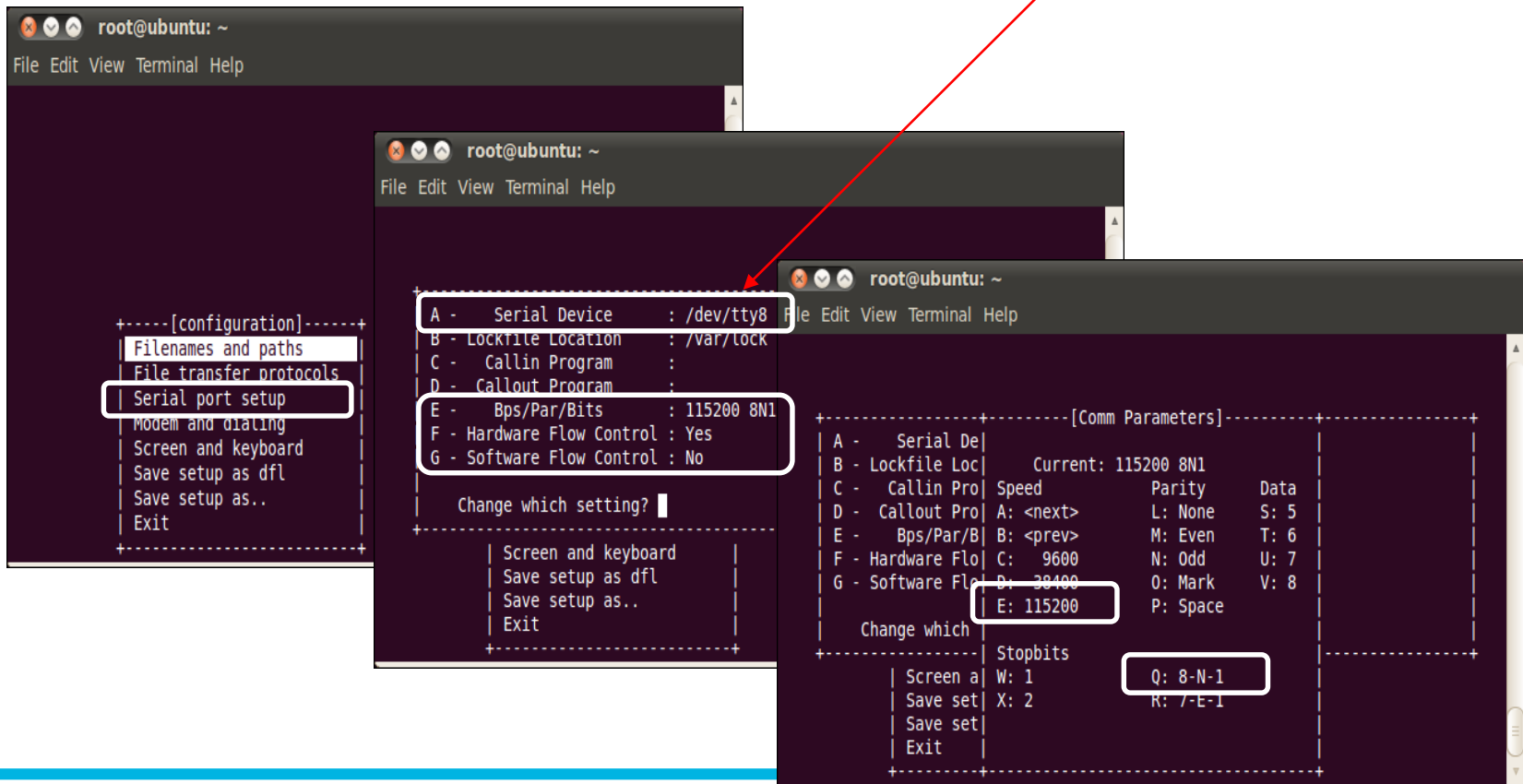
▶ Minicom 다운로드 및 설치

```
# apt-get install minicom
```

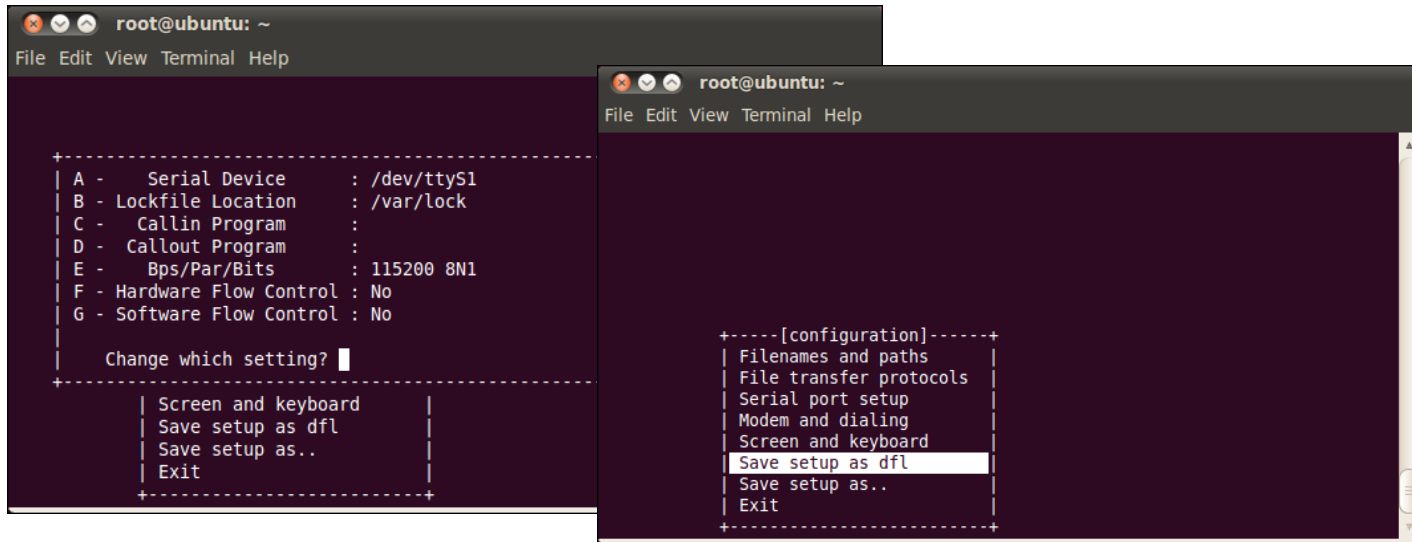
▶ minicom 설정

```
# minicom -s
```

우리는 시리얼 케이블을 사용하지 않고
USB-Serial 방식을 사용하기 때문에
/dev/ttyUSB0로 설정해야 함



Toolchain



```
root@ubuntu: ~
File Edit View Terminal Help

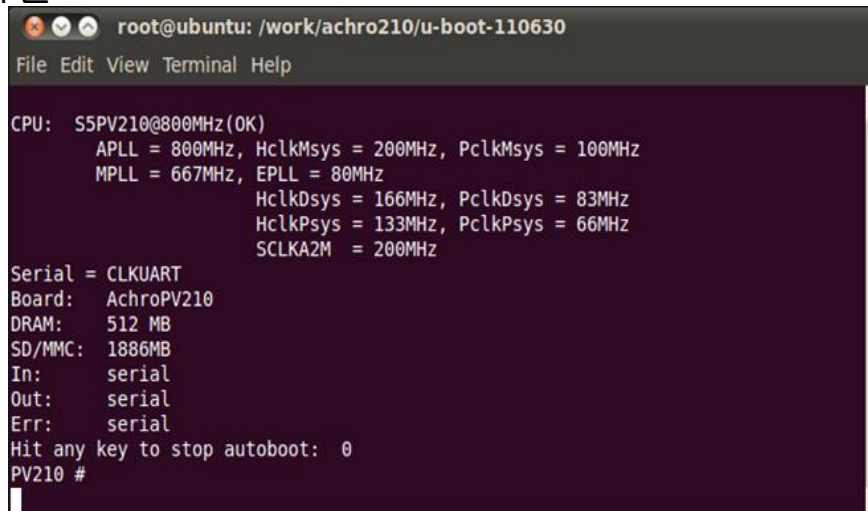
+-----+
| A -   Serial Device       : /dev/ttyS1 |
| B -   Lockfile Location   : /var/lock   |
| C -   Callin Program      :             |
| D -   Callout Program     :             |
| E -   Bps/Par/Bits        : 115200 8N1  |
| F -   Hardware Flow Control : No        |
| G -   Software Flow Control : No        |
|                                     |
| Change which setting? |             |
+-----+

| Screen and keyboard |
| Save setup as dfl   |
| Save setup as..    |
| Exit                |
+-----+
```

```
root@ubuntu: ~
File Edit View Terminal Help

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

▶ minicom 동작확인



```
root@ubuntu: /work/achro210/u-boot-110630
File Edit View Terminal Help

CPU: S5PV210@800MHz(OK)
      APLL = 800MHz, HclkMsys = 200MHz, PclkMsys = 100MHz
      MPLL = 667MHz, EPLL = 80MHz
           HclkDsys = 166MHz, PclkDsys = 83MHz
           HclkPsys = 133MHz, PclkPsys = 66MHz
           SCLKA2M  = 200MHz

Serial = CLKUART
Board: AchroPV210
DRAM: 512 MB
SD/MMC: 1886MB
In: serial
Out: serial
Err: serial
Hit any key to stop autoboot: 0
PV210 #
```

Toolchain

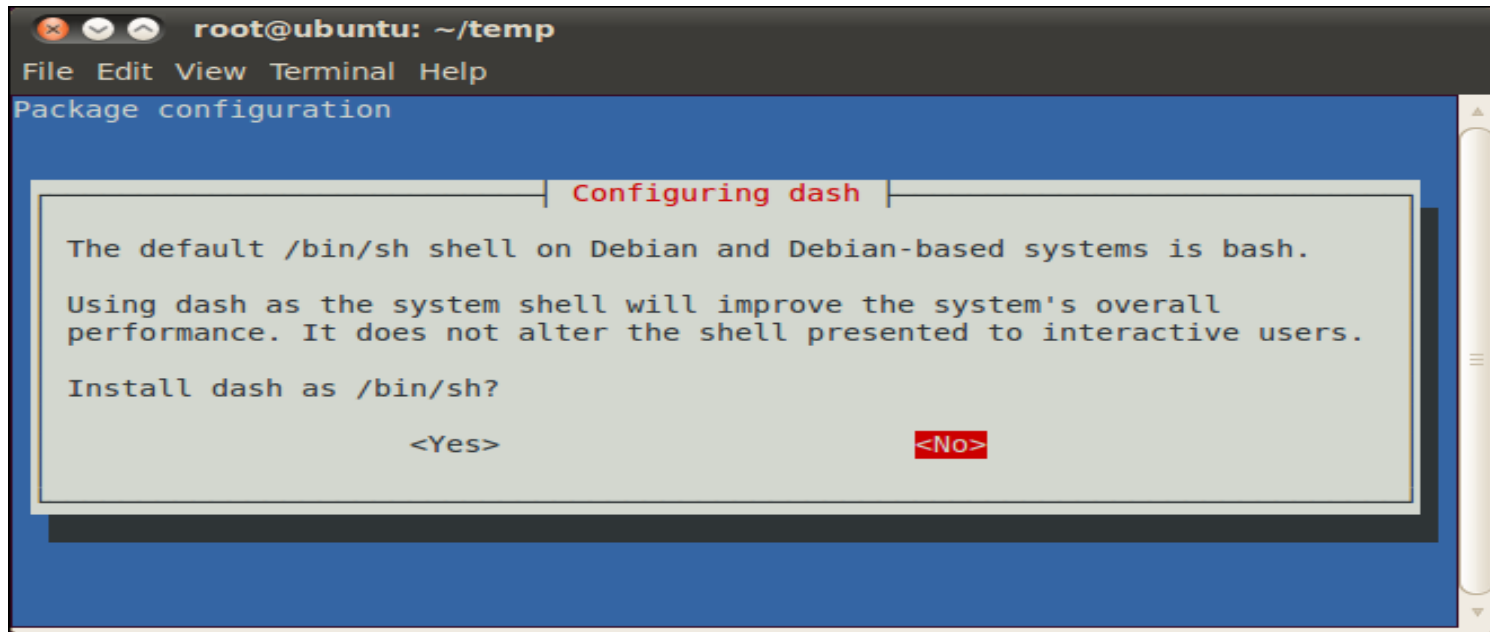
▶ Host Computer 환경 설정: 크로스 컴파일러

▶ 작업 디렉토리 생성 (-p : 상위 디렉토리까지 포함하여 디렉토리 생성)

```
# mkdir -p /work/achro5250
```

▶ 쉘 변경 : Configuring dash 에서 No를 선택

```
# dpkg-reconfigure dash
```



Toolchain

▶ 툴체인 설치

CD 내용을 PC 폴더로 공유했으면 CD 없이 /mnt/hgfs/~~ 디렉토리 밑에서 복사

```
# cp -a /media/(Achro Disc)/toolchain/arm-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 /work/achro5250
# sync
# cd /work/achro5250
# mkdir /opt/toolchains
# tar jxvf arm-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 -C /opt/toolchains/
```

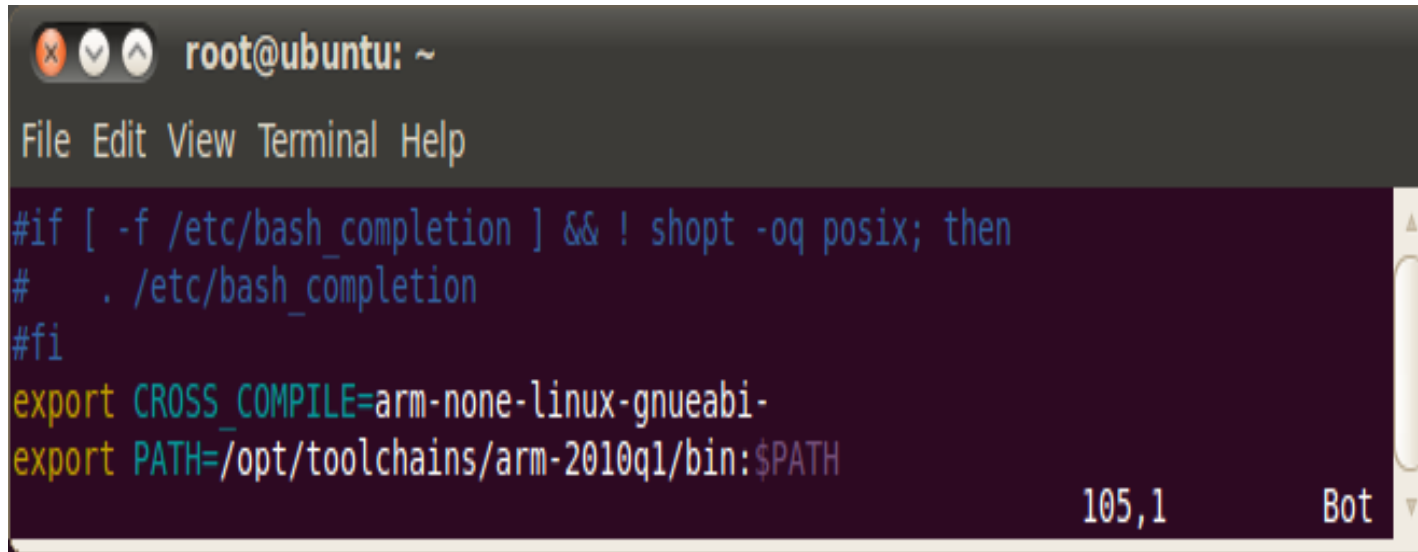
▶ 크로스 컴파일러 환경 설정 및 테스트

```
# vi /root/.bashrc
```

```
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.
if [ -f ~/.bash_aliases ]; then
. ~/.bash_aliases
fi
# Cross Compiler - achro5250 Android
export ARCH=arm
export CROSS_COMPILE=arm-none-linux-gnueabi-
export PATH=/opt/toolchains/arm-2010q1/bin:$PATH
```

끝 부분에 추가

Toolchain



```
root@ubuntu: ~  
File Edit View Terminal Help  
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then  
#   . /etc/bash_completion  
#fi  
export CROSS_COMPILE=arm-none-linux-gnueabi-  
export PATH=/opt/toolchains/arm-2010q1/bin:$PATH
```

105,1 Bot

▶ 크로스 컴파일러 패스 적용

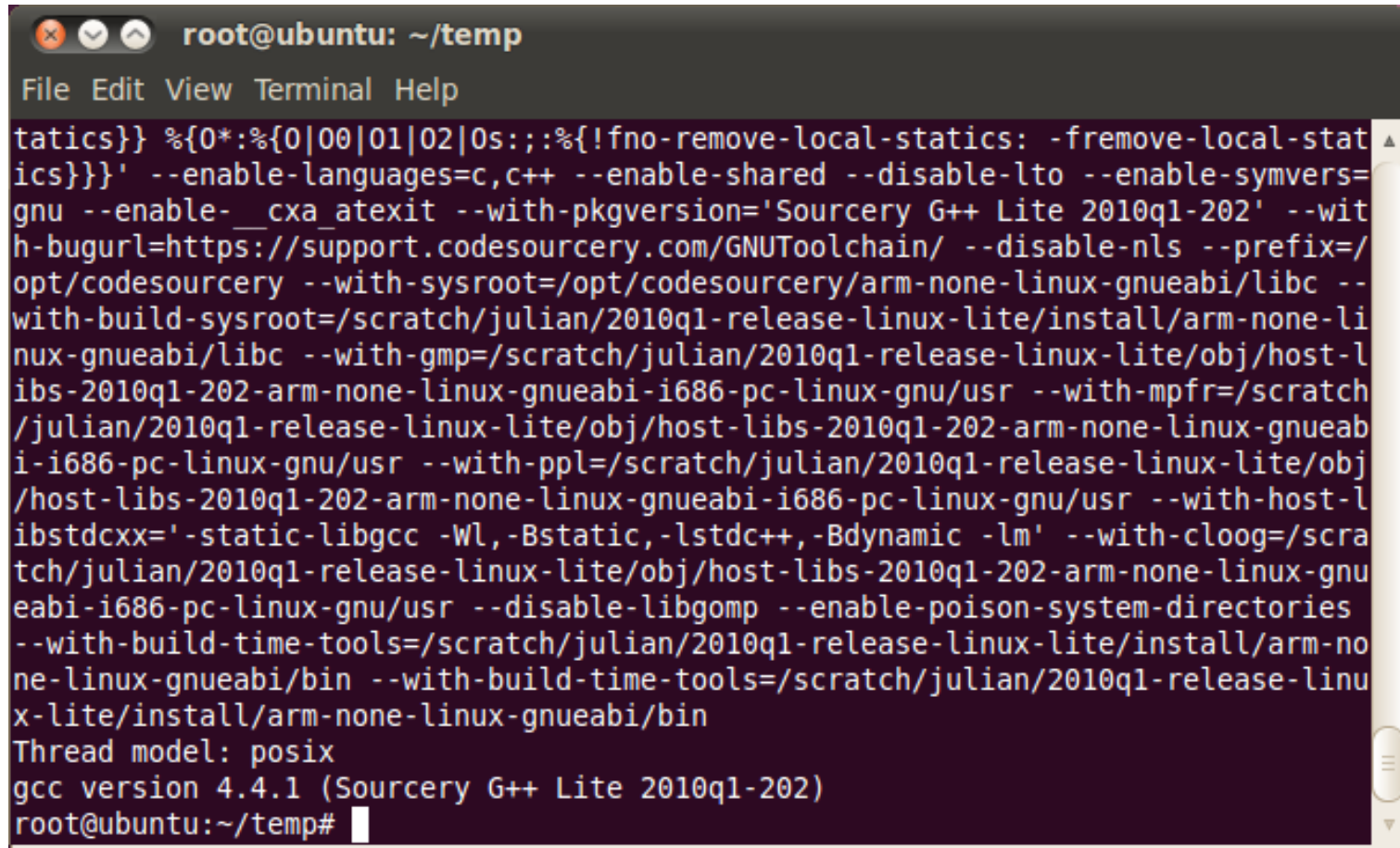
```
# source /root/.bashrc
```

▶ 크로스 컴파일러 버전 출력 (패스 적용 확인용)

```
# cd /root  
# arm-none-linux-gnueabi-gcc --v
```

Toolchain

▶ 실행 결과



```
root@ubuntu: ~/temp
File Edit View Terminal Help
tatics}} {%0*:%{0|00|01|02|0s::;%{!fno-remove-local-statics: -fremove-local-statics}}}' --enable-languages=c,c++ --enable-shared --disable-lto --enable-symvers=gnu --enable-__cxa_atexit --with-pkgversion='Sourcery G++ Lite 2010q1-202' --with-bugurl=https://support.codesourcery.com/GNUToolchain/ --disable-nls --prefix=/opt/codesourcery --with-sysroot=/opt/codesourcery/arm-none-linux-gnueabi/libc --with-build-sysroot=/scratch/julian/2010q1-release-linux-lite/install/arm-none-linux-gnueabi/libc --with-gmp=/scratch/julian/2010q1-release-linux-lite/obj/host-libs-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-mpfr=/scratch/julian/2010q1-release-linux-lite/obj/host-libs-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-ppl=/scratch/julian/2010q1-release-linux-lite/obj/host-libs-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-host-libstdcxx='-static-libgcc -Wl,-Bstatic,-lstdc++,-Bdynamic -lm' --with-cloog=/scratch/julian/2010q1-release-linux-lite/obj/host-libs-2010q1-202-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --disable-libgomp --enable-poison-system-directories --with-build-time-tools=/scratch/julian/2010q1-release-linux-lite/install/arm-none-linux-gnueabi/bin --with-build-time-tools=/scratch/julian/2010q1-release-linux-lite/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.4.1 (Sourcery G++ Lite 2010q1-202)
root@ubuntu:~/temp#
```

Toolchain

- ▶ 컴파일 테스트를 위한 테스트 코드 작성

```
# vim helloworld.c
```

- ▶ 소스코드 내용 입력

```
#include <stdio.h>
int main(int argc, char** argv)
{
    printf("Hello World!\n");
    return 0;
}
```

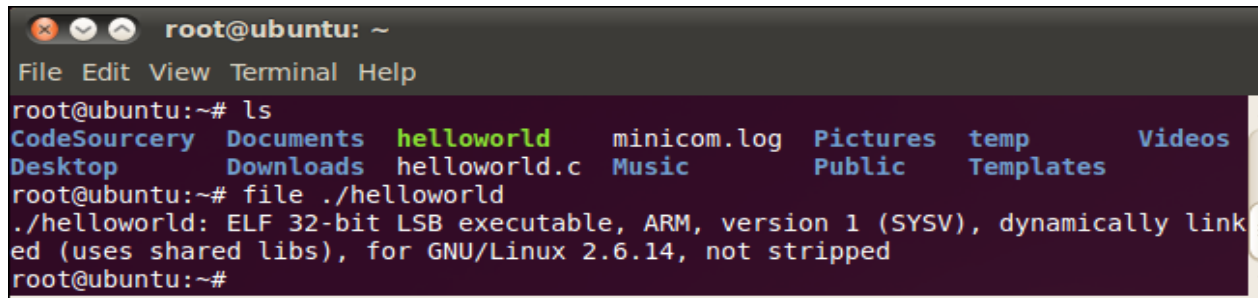
Toolchain

▶ 소스코드 컴파일

```
# arm-none-linux-gnueabi-gcc -o helloworld helloworld.c
```

▶ 컴파일 된 바이너리 확인

```
# file ./helloworld
```

A terminal window titled 'root@ubuntu: ~' with a menu bar (File, Edit, View, Terminal, Help). The terminal shows the following commands and output:

```
root@ubuntu:~# ls
CodeSourcery  Documents  helloworld  minicom.log  Pictures  temp  Videos
Desktop       Downloads  helloworld.c  Music        Public    Templates

root@ubuntu:~# file ./helloworld
./helloworld: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.14, not stripped

root@ubuntu:~#
```

Toolchain

▶ Host Computer 환경 설정: USB용 Download Tool 설치

▶ USB-Serial, USB-OTG 케이블을 사용을 위한 설정

▶ 호스트에서 개발을 usb 장치를 사용하기 위한 라이브러리 다운로드

```
# apt-get install libusb-dev
```

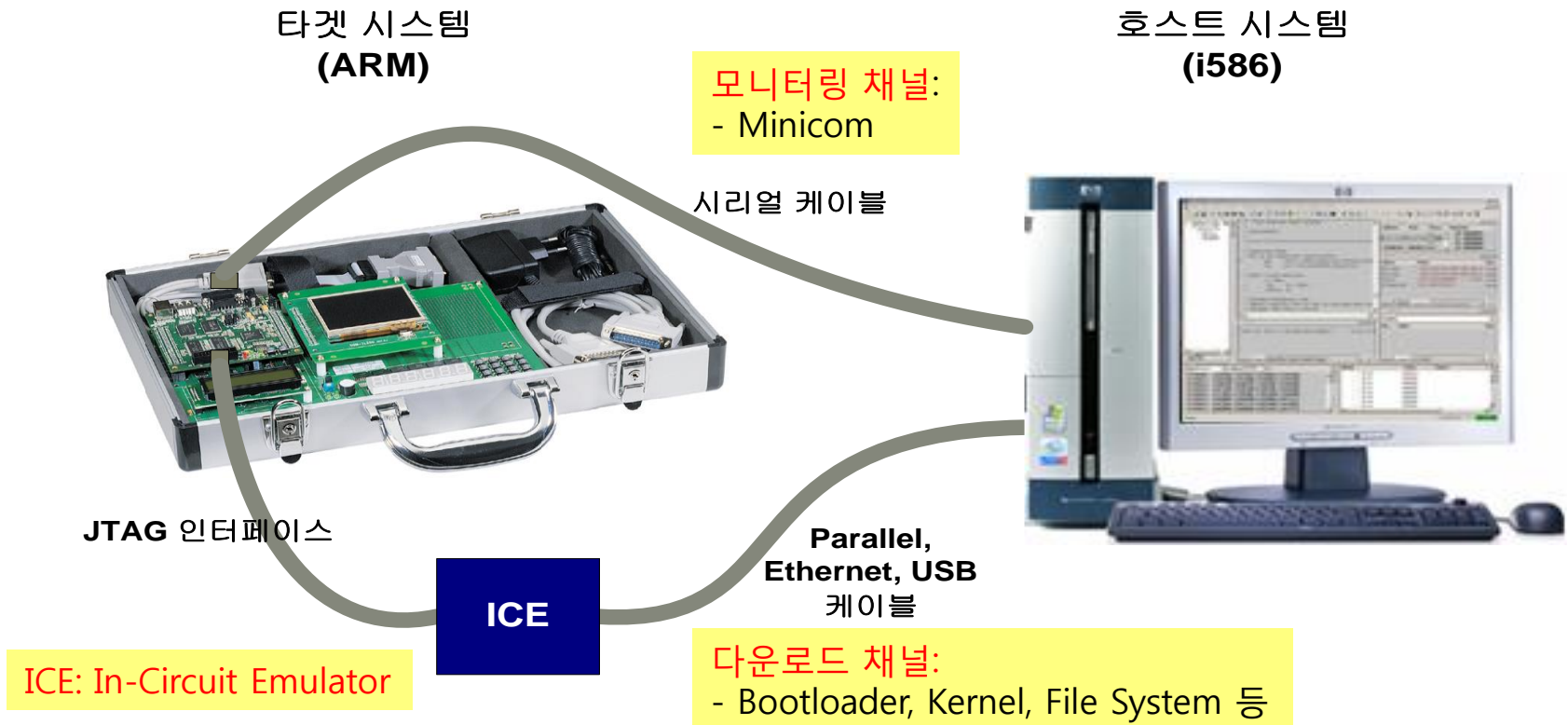
▶ smdk-usbdl 프로그램의 설치

```
# cd ~/ACHRO-5250-1.5.0.2/utilities/linux_tools/smdk-usbdl  
# cp smdk-usbdl.tar.gz /work/achro5250  
# cd /work/achro5250  
# tar xvfz smdk-usbdl.tar.gz  
# cd dltool  
# cp -a smdk-usbdl /usr/bin
```


Porting

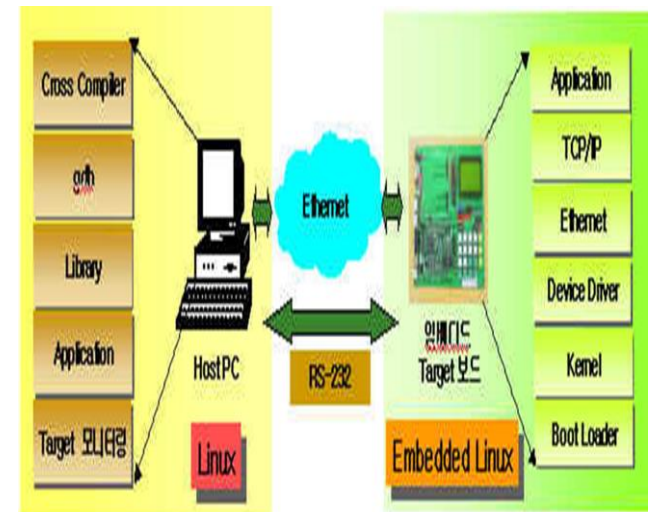
▶ 일반적인 포팅 방법의 구성

- ▶ JTAG 인터페이스를 이용해 바이너리 이미지를 타겟의 Flash 메모리에 기록
 - ▶ 바이너리 이미지: 부트로더, 커널, File System(root, user), Application S/W, Test S/W
- ▶ Serial 인터페이스(minicom 유틸리티)를 이용해 모니터링



Porting

- ▶ 우리 실습환경에서의 포팅 방법의 구성 (일반적이지 않음)
 - ▶ 리눅스 시스템의 전체가 Micro SD 카드에 기록됨
 - ▶ SD 카드의 초기화는 PC에서 USB-SDCARD Reader로 작업
 - ▶ 타겟에서의 작업은 시리얼 케이블과 Host에서의 Minicom 프로그램으로 작업을 원격 실행
 - ▷ USB-Serial 케이블은 모니터나 실행 제어용으로 항상 사용
 - ▶ 각종 이미지 파일 다운로드 방법
 - ▷ 부트로더
 - PC에서 USB-SDCARD Reader로 Micro SD 카드에 직접 fusing
 - ▷ Kernel
 - USB-OTG 케이블을 이용해 Micro SD 카드에 다운로드
 - ▷ File System
 - PC에서 USB-SDCARD Reader로 Micro SD 카드에 직접 fusing
 - ▷ Device Driver
 - 이더넷 + NFS를 이용해 다운로드
 - ▷ Device Driver 테스트/응용 프로그램
 - 이더넷 + NFS를 이용해 다운로드



Porting

▶ 필수 구성 요소

▶ Target System을 구동하기 위해서는 적어도 다음의 3가지 구성요소가 필요하다.

▷ Boot Loader

- 하드웨어를 초기화하고 커널 이미지를 SDRAM에 올려주어 수행을 넘겨주는 역할을 하는 프로그램

▷ OS Kernel

- OS 의 핵심 프로그램

▷ Root File System(RFS)

- Kernel에서 사용할 기본적인 File System

▶ 추가 구성 요소

▶ Target System을 보다 효율적으로 사용하기 위해 User File System을 추가 할 수 있다.

▷ User File System(UFS)

- Root File System에 포함되지 않은 util이나 data file등을 위해 추가적으로 사용할 File System

Porting

▶ Boot Loader(부트로더)

- ▶ 시스템이 부팅될 때 가장 먼저 실행되는 프로그램

- ▶ 주된 기능

 - ▷ Hardware Initialization

 - CPU clock, Memory timing, interrupt, UART, ETC.

 - ▷ Kernel Load

 - 운영 체제를 RAM에 올려주고 실행.

- ▶ 추가 기능

 - ▷ Target board에서 개발한 program을 전송하기 위한 기능들

 - ▷ Download Image to SDRAM

 - Serial - xmodem, uuencode

 - Ethernet - tftp, bootp

 - ▷ Fusing Flash Rom

 - Write, Read, Erase, Lock/Unlock

Porting

▶ Kernel(커널)

▶ OS의 핵심기능을 수행하는 프로그램

- ▶ Process management

- ▶ Memory management

- ▶ Device management

- ▶ Network management

- ▶ File System Management

- ▶ API (Application Programming Interface)

- ▶ ETC.

▶ Kernel Image(커널 이미지)

- ▶ Kernel 을 압축한 것으로 boot loader에 의해 압축이 해제되어 SDRAM으로 load 된다.

- ▶ 시스템의 저장 공간을 최소한으로 사용하기 위한 목적으로 압축 상태로 존재

Porting

▶ Root File System(루트 파일 시스템)

- ▶ “/” 디렉토리로 마운트 되는 파일 시스템으로 리눅스가 동작하기 위해 기본적으로 필요한 프로그램 및 설정 파일을 가지고 있다.
- ▶ Kernel 에서 기본적인 초기화 작업 후 Root File System을 마운트 한다.
- ▶ Embedded System은 대부분 크기가 작아야 하므로 hard disk를 가지고 있을 수 없기 때문에 주로 RAMDISK를 사용하여 Root File System을 구축한다.

Porting

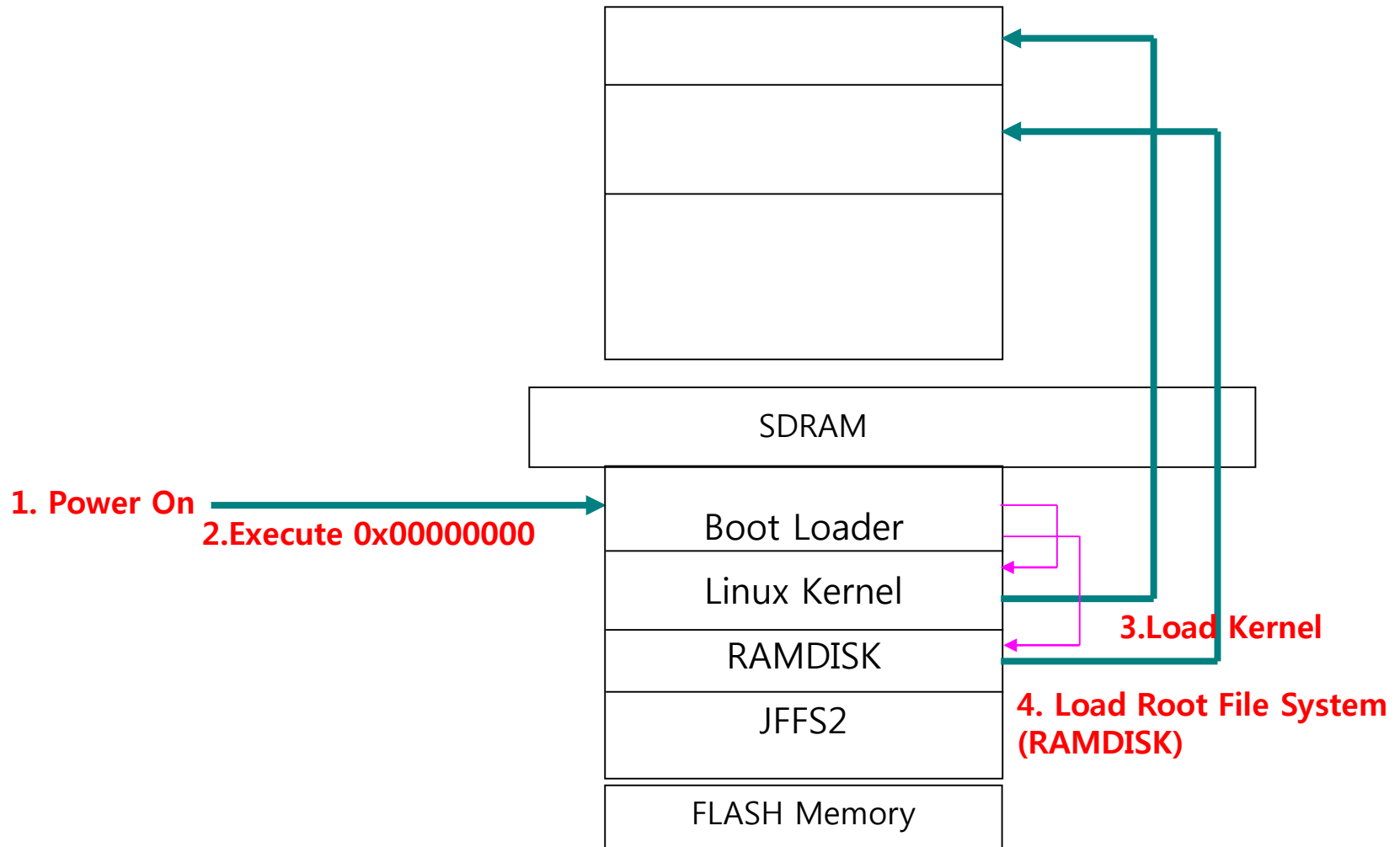
▶ User File System(사용자 파일 시스템)

- ▶ Root File System에 포함되지 않은 util이나 data file등을 위해 추가적으로 사용할 file system
- ▶ 필수 요소는 아니지만 target board를 보다 효율적으로 구성하기 위해서 필요하다.
- ▶ Flash File System으로 제작
 - ▷ RAMDISK와 달리 Flash Memory에 직접 file system을 구현한 것
 - ▷ Flash Memory에 바로 read/write hard disk와 동일하게 사용
 - ▷ 전원이 꺼진 후에도 작업 내용이 Flash Memory에 남아있다.
 - ▷ Flash memory는 기록하는 데에 한계가 있으므로 빈번하게 기록과 삭제가 일어나면 짧은 시간에 고장이 날 수 있다

Target Booting 절차

▶ Target Operating 순서 [1]

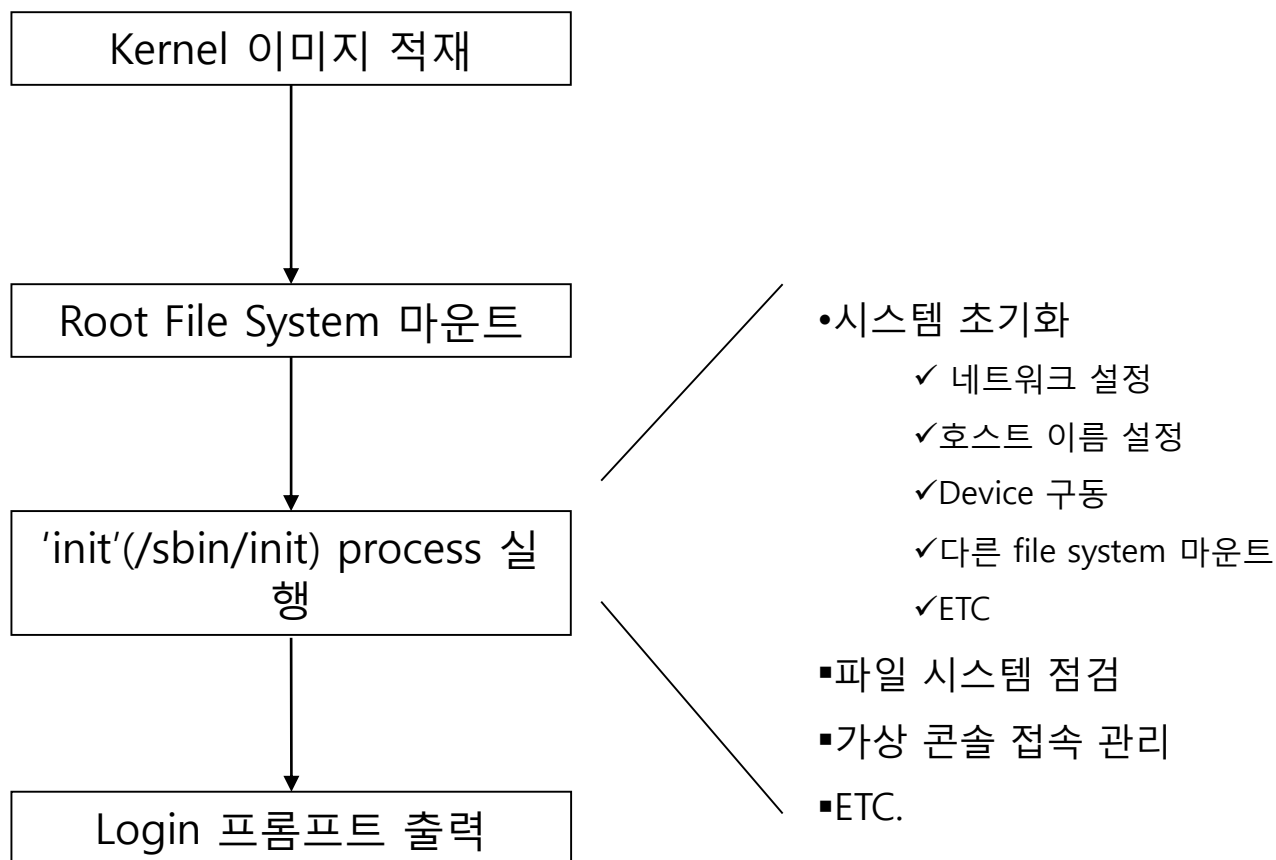
▶ Boot Loader



Target Booting 절차

▶ Target Operating 순서 [2]

▶ Kernel



Q & A
