Tool-Chain

실습환경
Development environment

	호스트(Host) 환경	타겟(Target) 환경
Model	Legion Y540-15IRH	RaspberryPi4 B
CPU	Intel x86	Cortex-A72 (BCM2711)
OS	Ubuntu on VMware	Raspbian Buster

✓ 툴체인(Tool Chain)

호스트환경과는 다른 환경을 가진 시스템의 소프트웨어 제품을 만드는 데 사용되는 컴퓨터 프로그램 개발 도구들의 집합.

ex) 문서편집기, 컴파일러, 링커, 라이브러리 등

툴체인의 종류
Tool-chain

- ✓ **문서편집기**(Editor)
  단순한 문서파일을 편집하기 위해 쓰이는 소프트웨어
  2진 형식이 아닌 사람이 읽을 수 있는 자연어형식의 텍스트파일

- ✓ **링커**(Linker)
  컴파일러가 만들어낸 하나 이상의 목적파일을 가져와 이를 단일 실행 프로그램으로 병합하는 프로그램

- ✓ **컴파일러**(Compiler)
  프로그래밍 언어로 쓰여진 문서를 기계어로 번역시켜주는 프로그램
  컴파일러는 두 가지로 구분(네이티브 컴파일러, 크로스 컴파일러)

컴파일러의 구분
Compiler

- ✓ **네이티브 컴파일러(Native Compiler)**
  호스트머신에서 프로그램 소스코드를 컴파일하여 호스트머신의 바이너리 코드를 만들어내는 컴파일러

- ✓ **크로스 컴파일러(Cross Compiler)**
  호스트머신에서 프로그램 소스코드를 컴파일을 하지만 호스트머신의 바이너리 코드가 아닌 타겟머신의 바이너리 코드를 만들어내는 컴파일러

타겟머신의 툴체인
Tool-chain of Target-machine

✓ 실습에 활용 될 타겟머신 – 라즈베리파이(Raspberry Pi)

✓ 타겟머신에 필요한 툴체인

- Vim(Vi Improved)
  : 리눅스 환경에서 가장 많이 사용하는 문서편집기
- Arm-linux-gcc
  : Arm코어의 바이너리 코드를 생성시켜주는 크로스 컴파일러
- Make
  : 파일들 간의 종속관계를 파악하여 하나의 실행파일을 만들어주는 링커와 같은 역할

툴체인 다운로드
Download Tool-chain

✓ 호스트머신(Ubuntu)에서 타겟머신(RaspberryPi)에 필요한 툴체인 다운로드

- Vim

  $ sudo apt install vim

- Arm-linux-gcc

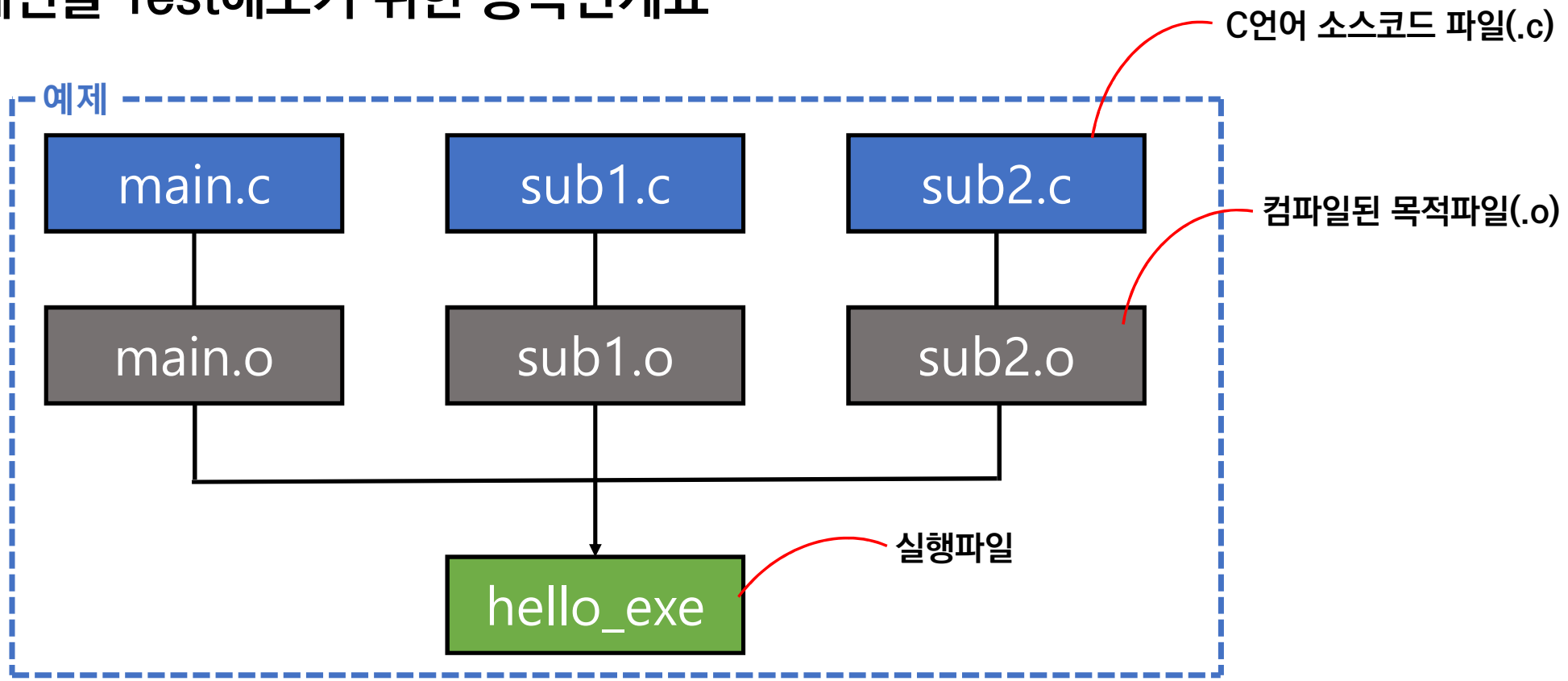
  $ sudo apt install gcc-arm-linux-gnueabihf (Cortex ARM : 라즈베리파이 2이상 버전)

  $ sudo apt install gcc-arm-linux-gnueabi (ARM : 라즈베리파이 B+이하 버전)

- Make

  $ sudo apt install make

✓ 툴체인을 Test해보기 위한 종속관계표



✓ 호스트머신에서 Vi Editor를 사용하여 3개의 간단한 소스코드를 작성

$ vi main.c

```
#include <stdio.h>

extern void sub1();
extern void sub2();

int main(){
        printf("Hello Main!!₩n");
        sub1();
        sub2();
        return 0;
}
```

$ vi sub1.c

```
#include <stdio.h>

void sub1(){
        printf("Hello Sub1!!₩n");
}
```

$ vi sub2.c

```
#include <stdio.h>

void sub2(){
        printf("Hello Sub2!!₩n");
}
```

✓ Make 유틸리티 사용해보기

: 종속성을가진 3개의 소스코드(main.c, sub1.c, sub2.c)들을 위한
Makefile을 작성을 해보기 전에 Make유틸리티를 간단히 알아보도록 하자

- Make란?

  - 전의 슬라이드에 설명했듯이 소스코드 파일들의 종속성을 파악하여 실행파일로 만들어주는 링커와 같은 역할을 하는 유틸리티이다.

  - Make유틸리티는 특정 디렉터리내의 Makefile(makefile도 가능)이라는 파일내의 명령어들을 순차적으로 실행한다.
    즉, make를 사용하기 위해서는 Makefile이라는 파일이 작업 디렉터리내에 존재하여야 한다.

  - Makefile내에 작성되어 실행되는 명령어들은 쉘 명령어들로 이루어져 있다.

  - 여러 종속성을가진 소스파일 중 특정 소스파일의 코드수정이 필요하여 다시 컴파일을 진행하여야 할 때 make를 사용하게 되면 수정된 소스파일만 컴파일을 수행하여 실행파일을 생성하게된다.
    즉, 기존의 수정이 되지 않은 소스파일들은 컴파일을 진행하지 않음으로 전체 컴파일속도가 향상하게 된다.

✓ Makefile의 작성규칙

- **빌드 규칙블록**(Rule Block)

```
〈목적파일〉 : 〈의존성파일〉
(tap) 〈명령어〉
```

➢ **목적파일** : 빌드 대상이 될 파일이다.

➢ **의존성파일** : 목적파일이 생성되기 위해 필요한 파일들을 나열하는 곳이다.
즉, 목적파일에 종속되는 파일들을 나열시키는 곳이다.

➢ **명령어** : 목적파일을 생성하기 위한 명령어이며 쉘 명령어들로 구성되어 있다.
명령어들의 앞에는 tab문자가 반드시 있어야한다.

✓ Makefile작성

- **빌드 규칙블록(Rule Block)을 따른 Makefile**

```
hello_exe : main.o sub1.o sub2.o
        arm-linux-gnueabi-gcc –o hello_exe main.o sub1.o sub2.o

main.o : main.c
        arm-linux-gnueabi-gcc –c main.c

sub1.o : sub1.c
        arm-linux-gnueabi-gcc –c sub1.c

sub2.o : sub2.c
        arm-linux-gnueabi-gcc –c sub2.c
```

목적파일 : **의존성파일**

(tab) **명령어**

- hello_exe라는 목적파일을 생성하기 위해서는 미리 종속관계표(7 Slide)에서 작성한 main.o sub1.o sub2.o라는 3개의 의존성파일들이 필요하고 첫 번째 규칙블록의 명령어부분을 실행하여 생성된다.
- 하지만 3개의 의존성파일들을 먼저 생성하여야 함으로 각 의존성파일들을 2, 3, 4번째 규칙블록들에서는 목적파일로 컴파일되는 것을 볼 수 있다.

✓ Make의 여러 기능

- **Make의 내장규칙**

```
hello_exe : main.o sub1.o sub2.o
        arm-linux-gnueabihf-gcc –o hello_exe main.o sub1.o
sub2.o

main.o : main.c

sub1.o : sub1.c

sub2.o : sub2.c
```

- 소스파일(*.c)을 오브젝트파일(*.o)로 컴파일하기 위해 수행되는 명령어들은 make유틸리티의 내장규칙으로 인해 자동으로 수행된다.
- 위의 Makefile과 같이 내장규칙으로 인해 전의 Makfile보다 더욱 단순히 구성된 모습을 볼 수 있다.
- Make에는 내장변수인 CC가 있는데 default값이 gcc로 되어있음으로 위의 Makefile은 에러가 발생한다.
  (이유 : 호스트의 컴파일러로 컴파일된 목적파일들을 타겟머신의 크로스 컴파일러로 실행파일을 생성시킬려고 하여 오류가 발생한다.)
- 내장변수의 내용은 다음 슬라이드를 참조하길 바란다.

✓ Make의 여러 기능

- **Make의 변수기능**

```
CC = arm-linux-gnueabihf-gcc
CFLAGS =
OBJS = main.o sub1.o sub2.o
TARGET = hello_exe

$(TARGET) : $(OBJS)
        $(CC) –o $@ $(OBJS)

main.o : main.c
sub1.o : sub1.c
sub2.o : sub2.c
```

CC : 컴파일러
CFLAGS : 컴파일 옵션
OBJS : 중간 산물 object파일 목록
TARGET : 목적파일

- Makefile내에는 변수를 사용할 수 있으며 =(할당연산자)를 통해 변수에 값을 할당할 수 있다.
  값이 할당 된 특정 변수를 사용하기 위해선 **$(변수)**와 같은 규칙을 따라야 한다.
- **CC**와 **CFLAGS**는 make의 내장변수이며 **$@**는 자동변수로 목적파일을 가르킨다.
- **CC**가 default값이 아니라 크로스 컴파일러인 arm-linux-gcc로 변경(할당)되었으니 규칙블록들은 전의 슬라이드와 다르게 오류가 발생하지 않고 arm-linux-gcc로 컴파일이 진행된다.

✓ 호스트머신에서 Make를 사용하여 실행파일생성

$make

```
jihoon@machine:~/Hello$ ls
Makefile  main.c  sub1.c  sub2.c
jihoon@machine:~/Hello$ make
arm-linux-gnueabihf-gcc -c main.c
arm-linux-gnueabihf-gcc -c sub1.c
arm-linux-gnueabihf-gcc -c sub2.c
arm-linux-gnueabihf-gcc -o hello_exe main.o sub1.o sub2.o
rm *.o
jihoon@machine:~/Hello$ ls
Makefile  hello_exe  main.c  sub1.c  sub2.c
```

- 크로스 컴파일러인 arm-linux-gcc를 사용하여 목적파일(*.o)과 실행파일(hello_exe)이 컴파일되는 것을 확인가능하다.
- Makefile의 내용에 따라 명령어가 실행되는 것을 확인할 수 있다.

✓ 생성된 실행파일의 의존성 확인

$file hello_exe

```
jihoon@machine:~/Hello$ file hello_exe
hello_exe: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamical
ly linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, BuildID[s
ha1]=609fa8d70f6068b840c1c0635cfdc345e5892451, not stripped
```

- 타겟의 ARM코어에 의존성을 가진 바이너리 파일로 확인할 수 있다.
- 즉, ARM코어의 바이너리 코드로 짜여진 실행파일인 것이다.
- Host머신(x86) 에선 실행시키지 못하는 실행파일이다.

$./hello_exe로 호스트에서 타겟의 실행파일을 실행시킬 시 오류발생.

```
jihoon@machine:~/Hello$ ./hello_exe
bash: ./hello_exe: cannot execute binary file: Exec 형식 오류
```

툴체인 사용해보기 (실습)
Tool-Chain

✓ SSH 설치(타겟머신의 원격제어 및 업로드를 위함)

$ sudo apt install ssh

✓ 타겟머신으로 실행파일 전송(타겟머신의 ssh port가 열려있어야 함)

$ scp ./hello_exe pi@xxx.xxx.xxx.xxx:/home/pi

```
jihoon@machine:~/Hello$ scp ./hello_exe pi@10.255.255.40:/home/pi
pi@10.255.255.40's password:
hello_exe                                     100% 8340     1.1MB/s   00:00
```

- ssh에서 제공되는 scp명령어 사용
- 타겟의 password를 입력 후 전송가능
- 용도 : Target의 해당 디렉터리로 파일을 전송시키는 명령어
- 형식 : scp [보낼파일] [Target name]@[ip]:[Target Dir]

툴체인 사용해보기 (실습)
Tool-Chain

✓ Target으로 원격접속(xxx...란에 타겟 시스템의 IP주소를 입력)

$ ssh pi@xxx.xxx.xxx.xxx

```
jihoon@machine:~/Hello$ ssh pi@10.255.255.40
pi@10.255.255.40's password:
Linux raspberrypi 4.14.82-v7+ #1 SMP Thu Nov 22 20:57:19 KST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jan  4 18:39:47 2019 from 10.255.255.41
pi@raspberrypi ~ $
```

- ssh를 활용하여 Target으로 원격접속을 실행하게 되면 터미널창의 호스트명이 타겟머신의 호스트명으로 변경되는것을 확인가능 (jihoon -〉 pi)

툴체인 사용해보기
Tool-Chain

✓ Host에서 Target으로 전송된 실행파일 확인

```
pi@raspberrypi:~ $ ls
Desktop    Downloads  hello_exe  Music     Public     Videos
Documents  driver     MagPi      Pictures  Templates
pi@raspberrypi:~ $ file hello_exe
hello_exe: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamical
ly linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, BuildID[s
ha1]=609fa8d70f6068b840c1c0635cfdc345e5892451, not stripped
```

✓ Target에서 hello_exe파일 실행

```
pi@raspberrypi:~ $ ./hello_exe
Hello Main!!
Hello Sub1!!
Hello Sub2!!
```

E N D