2009년 06월 24일

Makefile은 뭘 하는 녀석일까~?

Makefile이라는 Utility를 다루려면, Makefile이 뭐 하는 녀석인가를 파악해야 합니다. 개발과정에서 Source Code를 수정하고, 컴파일하고, 컴파일 된 결과를 확인하는 작업이라는 것은 Software 개발의 전부라고 할 수 있습니다만, 정말로 지루하고 시간이 많이 소요되는 작업이 아닐 수 없습니다. Software 요리 재료에서 Compile이란 Section에서 여러 가지 방법을 이용하여, Source Code를 compile하고 link하고 Library도 만들어보고 했지만, 만일 file도 많아지고, 여러 가지 compile option을 여러 가지 file에 대하여 다르게 적용한다던가, 하는 여러 가지 작업이 병행되어야 한다면, 정말 힘들고 지루하기 짝이 없을 것입니다. 지금은 Snare Girls라는 Cafe에 앉아서 이 글을 쓰고 있습니다만, 아주 작은 Project를 만들어서 Makefile의 원리와 편리성을 맛보시는 것이 가장 이해에 지름길이 되리라 믿으며, Example을 하나 만들어서 얘기를 진행해 보도록 하지요. 문제가 많은 코드이지만, 그냥 저냥 이해해 주시면 정말 감사하겠습니다. 자, 그럼 초심으로 돌아가서 위의 파일들을 compile 한 후, link하여 recipes.bin을 만들어 내려면 어떻게 하면 좋을 지 한번 고민해 봅시다.

spaghetti.h ----------

```
#define EQUAL =
#define  TRUE 1
typedef struct {
 char memberBool;
 int memberInt;
 char memberWord;
 } member_type;

extern int add(int a, int b);
```

spaghetti.c ----------

```
#include "spaghetti.h"
int zi EQUAL 0;
int rw EQUAL 3;
extern int relocate EQUAL 3;
member_type member =
{TRUE, 10, 2};
main()
{
 int stack;
 volatile int local,locall,localll;

 local EQUAL 3;
 locall EQUAL 4;

 localll EQUAL add (local, locall);
```

```
 stack EQUAL localll;
 return stack;

}
```

manupulation.c ----------

```
#include "spaghetti.h"
extern member_type member;
int add (int a, int b)
{
 return (a+b+member.memberWord);
}
```

앞서 나왔던 예제에서 한 두가지 차이점을 빼고는 거의 같습니다. 왜 차이점을 만들었냐 하면, 이렇게 해야 여러 개의 file을 한꺼번에 link하는 과정을 넣을 수 있으니까 수정을 했습니다만, 굳이 앞에서 그 예제를 찾아내어 틀린 점을 찾아낼 필요는 없으니까, 너무 걱정 마세요.

예제의 spaghetti.h, spaghetti.c, 그리고 manupulation.c 의 관계를 잘~ 염두해 두세요

(두개의 c file은 spaghetti.h를 공통으로 include하고 있어요)

어쨌거나, 초심으로 돌아가서 spaghetti.c하고, manupulation.c를 compile한 후 link을 하려면, 열심히 독수리 타법을 이용하여 console에 다음과 같이 입력하면 됩니다.

tcc -c spaghetti.c manupulation.c (c source file을 각각 link하지 말고, object file로 compile하라는 의미)
armlink -elf -o spaghetti.elf spaghetti.o manupulation.o (자, compile된 각각의 object file들을 모아서 elf type의 link된 executive file을 만듭시다.)

fromelf -o spaghetti.bin spaghetti.elf (from elf를 이용하여 spaghetti.elf형식의 executive file을 binary 형식으로 만듭시다.)

자자, 간단간단입니다만, 이렇게 작업하는 것은 상당히 귀찮은 일을 동반합니다. 어떤 일이냐 하면 예를 들어 spaghetti.c를 수정한 후, 다시 compile하여 link한 후, binary형식으로 만들려면,

```
tcc -c spaghetti.c
armlink -elf -o spaghetti.elf spaghetti.o manupulation.o
fromelf -o spaghetti.bin spaghetti.elf
```

와 같이 똑같은 작업을 다시 해줘야 합니다. 아 머리가 지끈지끈 거립니다. - 사실은 지금 Snare girl이라는 Cafe에 느긋하게 앉아 이 글을 쓰고 있습니다만, 옆 테이블의 말 많으신 여자분의 약 70dB정도의 소음으로 초당 600타 정도의 속도로 계속 떠들고 있어서 더 지끈 지끈 합니다. 어디 느긋하게 앉아서 두 눈을 풀고 멍한 상태로 커피 한잔 마시면서 시간을 보낼 수 있는 곳 아시면 지체 말고 제보 부탁합니다. 미리 감사 드립니다.-

여하튼 똑같은 작업을 다시 하자니, 벌써부터 손가락이 아파오기 시작합니다. 이런 의미에서 반복된 작업을 대신 해주는 뭔가가 있으면 좋을 것 같은데, 그 녀석이 바로 Makefile이라는 녀석 입니다요. DOS에서는 batch file이라고 해서 .bat 확장자를 갖는 반복된 작업을 대신 해 주는 script file이 있는 데 이것과 별반 다르지 않다고 생각한다면 무리가 없겠습니다..라고 생각 중입니다. 이런 의미 없는 작업을 반복하지 않기 위해서는 약간의 Makefile의 구조에 대해서 알 필요가 있습니다. 힘든 키보드 작업을 대신 할 만한 작업이라면 이 정도의 노력 정도는 값싸다고 생각해도 무리 없을 것 같습니다.

Makefile의 구조.

```
CC = tcc                                        매크로 정의
```

(만들고 싶은output) : 재료1 재료2 재료3 .....~ Make rule
	만들고 싶은 output을 만들기 위해 시키고 싶은 일 1 명령어
	만들고 싶은 output을 만들기 위해 시키고 싶은 일 2
<----> "이 사이는 꼭 tab으로 넣어야 합니다. 뭐 불편하지만, make rule이라니까 꼭! 지켜줘야 합니다.

그르니까 tab이 함부로 오면 안됩니다. tab을 함부로 넣으면 make는 뭔가의 명령어로 착각을 한다니까요! 암때나 tab넣으면 안돼요. 하지만 명령어를 넣을 때는 꼭!! tab을 넣어줘야 합니다!

오호, 상당히 간단하죠? 위의 Makefile의 구조를 해석하면 이렇습니다. 만들고 싶은 output을 만들기 위해서는 재료1, 재료2, 그리고 재료3이 필요하다는 의미이며, 만들고 싶은 output은 보통 .o file이나 .elf file등의 목적 file등이 되고, 재료들은 .o file이나 .c, .s, cpp등의 source file들이 됩니다. 만들고 싶은 output을 만들기 위해 시키고 싶은 일들은 재료들이 모두 만들어져 있을 때 만들고 싶은 output을 생성하기 위한 명령어들이 옵니다. 오, 만들고 싶은 output을 생성하기 위해서 할 수 있는 모든 일들은 이런 rule에 의해서 모두 만들어 질 수 있습니다.

맨 앞의 CC=tcc라고 선언한 부분은 Macro라고 하며, c의 #define과 마찬가지 입니다. Macro를 선언하는 경우는, Macro 선언 부분만 바꿔서도 유연하게 많은 일을 할 수 있도록 하려고 하는 것인데 이 경우에는 CC를 tcc로 선언해 놓고, 뭔가를 하려고 할 것 같은데 얕은 생각만으로도 CC는 c compiler이며, 이 부분만 수정하여, C compiler를 tcc 또는 gcc 또는 그 외의 c compiler로 쓸 수 있도록 하려고 선언한 것 같은 느낌입니다.

그러면, 이제까지 하던 작업을 계속 해 본다면, spaghetti.bin을 만들어 내기 위해서 어떻게 Makefile을 만들어 내면 좋을까 고민해 봅시다.

짜잔~

spaghetti.mak ----------

```
spaghetti.bin : spaghetti.elf                                   1) 10)
 fromelf -bin -o spaghetti.bin spaghetti.elf                    11)

spaghetti.elf : spaghetti.o manupulation.o                      2)  5)  8)
 armlink -elf -o spaghetti.elf spaghetti.o manupulation.o       9)

spaghetti.o : spaghetti.c                                       3)
 tcc -c spaghetti.c                                             4)

manupulation.o : manupulation.c                                  6)
 tcc -c manupulation.c                                          7)
```

이렇게 spaghetti.mak를 만든 후, make -f spaghetti.mak를 실행하면, 다음과 같은 Message를 보여줍니다.

```
E:>make -f spaghetti.mak
tcc -c spaghetti.c
tcc -c manupulation.c
armlink -elf -o spaghetti.elf spaghetti.o manupulation.o
fromelf -bin -o spaghetti.bin spaghetti.elf
```

자, 자동으로 우리가 손으로 하던 작업을 해주지요? 정말 간단간단입니다. 이런 식이라면, compile에 관련 우리가 귀찮아 하던 작업을 대신 해 주겠지요? 그러면, 이중에 spaghetti.o를 하나 수정하고 다시 spaghetti.mak를 실행해 보겠습니다.

```
E:>make -f spaghetti.mak
```

tcc -c spaghetti.c
armlink -elf -o spaghetti.elf spaghetti.o manupulation.o
fromelf -bin -o spaghetti.bin spaghetti.elf

오, 이번에는 manupulation.c를 컴파일 하지 않고, 자동으로 spaghetti.c만 다시 컴파일 하여, - 다시 말하면, manupulation.c를 컴파일 하지 않고, 꼭 필요한 spaghetti.c만 다시 컴파일하여 - spaghetti.bin을 만들어줍니다요. 이런 기특한 녀석!하는 생각이 듭니다. 기특하니까, 조금 만 더 어떻게 동작하는지 안쪽을 들여다 봐도 그리 나쁘지 않지요. 자, 처음에 우리가 make file을 실행했던 그 순간을 다시 한번 음미해 볼까요~?

0) makefile 명령을 내리면, 제일 처음에 만나게 되는 만들고 싶은 output을 찾습니다.
우리 작업 중에는 spaghetti.bin이 해당되겠습니다.

1) spaghetti.bin을 만들기 위해서 그 재료를 봤더니, spaghetti.elf더라. 그러면, Makefile은 spaghetti.elf가 존재하는지 찾아봅니다.

2) 엇! spaghetti.elf가 없다고 판단하면, makefile은 spaghetti.elf를 만들어 내기 위하여, spaghetti.elf가 만들고 싶은 output인 곳을 찾습니다. 오 그렇군요. 그렇다면, Makefile은 여기에서 spaghetti.elf를 만들기 위한 재료들이 있는지 확인하게 되지요. 가만히 보니까, spaghetti.o이 없네요?

3) 음, 그러니까 재료들이 없단 말이지, 그러니까 먼저 spaghetti.o를 만들어 내야 되니까, spaghetti.o의 재료는 또 무엇인지 확인하게 됩니다. 오, 그 재료는 spaghetti.c입니다. spaghetti.c는 우리가 Coding한 source file입니다. 물론 존재하겠지요?

4) Makefile은 최종으로 존재하는 spaghetti.c를 확인했으니, spaghetti.o를 만들어 내기 위하여, rule을 이용하여 spaghetti.o를 compile하여 생성하게 됩니다.

5) Makefile은 다시 위로 돌아와 spaghetti.elf를 만들려고 시도합니다만, 어라? manupulation.o도 없네요. 뭐 있는 게 없습니다. Makefile은 다시 manupulation.o를 만들어 내기 위하여, 만들고 싶은 output이 manupulation.o인 경우를 찾아 내려갑니다. 머나먼 여행.

6) 오호라, spaghetti.o와 마찬가지로, manupulation.c만 있으면 manupulatoin.o를 만들어 낼 수 있다는 것을 알아내었습니다. 그렇다면 spaghetti.o를 만들어낼 때와 마찬가지로 7)번 rule을 따라 manupulation.o를 만들어 내지요. - 만약! spaghetti.o나 manupulation.o가 있었다면, 이 녀석들이 다시 만들 필요가 있는 녀석들인지를 Makefile은 검토하게 됩니다. .o와 .c를 비교하여 만들어진 시간이 같은지 확인해 주죠. 정말정말 기특한 녀석입니다.~ 만일 만들어진 시간이 다르면, 다시 만들어 주고, 같다면, 그냥 skip합니다.

8) 자, 이제 spaghetti.o와 manupulation.o가 만들어 졌으니, 처음에 하려던 spaghetti.elf를 만들어 낼 수 있습니다.

9) 이 rule을 이용하여 spaghetti.elf를 만들어 냅니다.

10), 11)을 이용하여 마지막 최종 목적인 spaghetti.bin을 만들어 냅니다. Spaghetti.bin이 만들어 지는 과정 재미있지요?

짜잔~ 편리하지요? 이렇게만 만들어 놓으면 별 탈없이, 귀찮은 일들을 Makefile이 대신 해줍니다.

 make에서는 선언하는 방법이 아~주 중요합니다. 이거 잘못하면 엉뚱한 방향으로 흐를 수도 있고, 무엇이 잘못되었는지 찾기 힘든 경우도 있어요.선언하는 방법에 따라 무엇이 틀린 지 정도를 알아보면,
CC := tcc → 사실은 이렇게 선언하는 것이 우리가 보통 생각하는 선언 입니다. 만일 이 선언 전에 TEMP := $(CC) temp 라고 선언했다면, CC는 아직 선언되기 전이기 때문에 TEMP 는 그냥 temp가 되고 이 선언 이후에 TEMP := $(CC) temp라고 선언했다면, TEMP는 tcc temp라고 표현됩니다. 조금 헷갈리시겠지만, makefile은 뭔가를 완성하기 위해서 계속 recursive하게 동작합니다. =과 :=의 차이를 보기 위해, 예를 든다면 CC = tcc라고 선언한 경우에는 TEMP = $(CC) temp라는 선언이 어디에 있더라도, 관계없이 TEMP = tcc temp가 됩니다. 어렵나요? CC ?=tcc → 이 경우에는 만일 CC가 선언되어 있지 않은 경우에만 CC가 tcc로 선언됩니다. #ifdef 과 똑같다고 보시면 됩니다. CC +=tcc → 이 경우에는 기존의 문자열에 공백을 두고 덧붙이는 경우 입니다. 예를 들어, CC=A, CC+=B, CC+C라고 해주면 결과는 CC = A(공백)B(공백)C가

되는 거죠. 무엇을 사용하느냐는 각각의 책임하에 사용하셔야 할 것입니다. 좀 무책임한 발언이겠지만. 꼬옥~ 알아두세요.

compile, option, link, 원리, make, makefile

Linked at at 2009/06/24 23:40

... nbsp; ⓟ Memory Map과 Linker의 만남 Locator ⓠ Makefile은 뭘하는 녀석일까~ ⓡ 컴파일을 더더더 쉽게. MACRO와 SUFFIX ⓢ 조금 더 Ma ... more

Commented by SMILE at 2009/06/27 15:24

주말에 소설 처럼 읽고 또 읽고 있습니다. 감사

Commented by 히언 at 2009/06/28 15:37

냐하하~ 완전 감사헤요~