
**C 프로그램 컴파일 (1/9)**

Compile & Makefile

- 대부분의 유닉스 유틸리티와 상용 프로그램들은 C(또는 C++)로 작성되어 있음 (UNIX 자체가 C언어로 작성되어 있음)
- 일부 UNIX 시스템에는 C 컴파일러가 기본적으로 내장되어 있으며, C 언어를 알지 못하고서는 UNIX 시스템을 깊이 있게 이해하기 어려움
  - 최근에는 C 컴파일러를 UNIX와 별도로 판매하는 것이 일반적임 (cc)
  - 공개된 C 컴파일러로 gcc (GNU cc) 컴파일러가 널리 사용됨
- C 컴파일러 사용법

```
$ cc [-options] C-files
$ gcc [-options] C-files
```

 강원대학교 컴퓨터과학과

Page 2

UNIX System Programming  
by Yang-Sae Moon



## C 프로그램 컴파일 (2/9)

Compile & Makefile

### 기본적인 사용법

```
$ cc hello.c
```

- hello.c를 컴파일하여 object file인 hello.o를 만듦
- 또한, 실행 가능한 파일(executable file)로서 a.out을 만듦

```
user1@infravalley:~  
NewWorld [ ysmoon {177} ~/unix ] ls  
hello.c      practice/  
NewWorld [ ysmoon {178} ~/unix ] cat hello.c  
#include <stdio.h>  
main()  
{  
    printf("Hello!\n");  
}  
NewWorld [ ysmoon {179} ~/unix ] gcc hello.c  
NewWorld [ ysmoon {180} ~/unix ] ls  
a.out*      hello.c      practice/  
NewWorld [ ysmoon {181} ~/unix ] a.out  
Hello!  
NewWorld [ ysmoon {182} ~/unix ]
```



Page 3

UNIX System Programming  
by Yang-Sae Moon



## C 프로그램 컴파일 (3/9)

Compile & Makefile

### 주요 옵션(-c)

```
$ cc -c hello.c
```

- Object code(기계어 코드)만 생성하며, 실행 파일은 만들지 않음
- 상기 예의 경우, hello.c를 컴파일하여 object code인 hello.o를 만듦

```
user1@infravalley:~  
NewWorld [ ysmoon {189} ~/unix ] ls  
hello.c      practice/  
NewWorld [ ysmoon {190} ~/unix ] cat hello.c  
#include <stdio.h>  
main()  
{  
    printf("Hello!\n");  
}  
NewWorld [ ysmoon {191} ~/unix ] gcc -c hello.c  
NewWorld [ ysmoon {192} ~/unix ] ls  
hello.c      hello.o      practice/  
NewWorld [ ysmoon {193} ~/unix ] file hello.o  
hello.c:      C 프로그램 텍스트  
hello.o:      ELF 32-비트 MSB 재배치 가능 SPARC 버전 1  
NewWorld [ ysmoon {194} ~/unix ]
```



Page 4

UNIX System Programming  
by Yang-Sae Moon



## C 프로그램 컴파일 (4/9)

Compile & Makefile

### 주요 옵션(-o)

```
$ cc -o hello hello.c ($ cc -o hello hello.o)
```

- C 컴파일러는 실행 파일의 default 이름으로 a.out을 생성하며, 상기 -o 옵션을 사용하여 원하는 파일명으로 실행파일을 바꿀 수 있음
- 여러 C 파일(혹은 object file)을 묶어서 하나의 실행 파일을 생성할 수도 있음

```
user1@infravalley:~
NewWorld [ ysmoon {200} ~/unix ] ls
hello.c  practice/
NewWorld [ ysmoon {201} ~/unix ] cat hello.c
#include <stdio.h>
main()
{
    printf("Hello!\n");
}
NewWorld [ ysmoon {202} ~/unix ] gcc -o hello hello.c
NewWorld [ ysmoon {203} ~/unix ] ls
hello*  hello.c  practice/
NewWorld [ ysmoon {204} ~/unix ] file hello*
hello:  ELF 32-비트 MSB 실행 가능 SPARC 버전 1, 동적으로 링크됨, 분리 안
됨
hello.c:  C 프로그램 텍스트
NewWorld [ ysmoon {205} ~/unix ]
```



## C 프로그램 컴파일 (5/9)

Compile & Makefile

### 주요 옵션(-g)

```
user1@infravalley:~
NewWorld [ ysmoon {213} ~/unix ] gcc -g -o hello hello.c
NewWorld [ ysmoon {214} ~/unix ] gdb hello
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "sparc-sun-solaris2.8"...
(gdb) break 3
Breakpoint 1 at 0x105d8: file hello.c, line 3.
(gdb) run
Starting program: /home/ysmoon/unix/hello

Breakpoint 1, main () at hello.c:3
3      {
(gdb) next
4      printf("Hello!\n");
(gdb) next
Hello!
5      }
(gdb) where
#0 main () at hello.c:5
(gdb) next 10
Single stepping until exit from function _start,
which has no line number information.

Program exited with code 01.
(gdb)
```

Debugger는 프로그  
래밍을 하는데 있어  
서 매우 유용한 tool  
이므로, 개인적으로  
반드시 숙지할 것



## C 프로그램 컴파일 (6/9)

Compile & Makefile

### 주요 옵션(-p)

```

user1@infravalley:~
IU-MPC1 [ mpc {111} ~/ ]
total 1
1 div_multi.c
IU-MPC1 [ mpc {112} ~/ ]
#include <stdio.h>

main()
{
    int i, j;
    float multi(int i, int j)
    {
        for(i=0; i < 10; i++)
            for(j=0; j < 10; j++)
                multi(i, j);
    }
}

Float multi(int i, int j)
{
    float sum = 0;
    for(k=0; k < 10; k++)
        sum += i * j;
    return sum;
}

IU-MPC1 [ mpc {113} ~/ ]

```

```

user1@infravalley:~
IU-MPC1 [ mpc {121} ~/unix ] cc -p -o div_multi div_multi.c
IU-MPC1 [ mpc {122} ~/unix ] ls
total 65
64 div_multi*
1 div_multi.c
IU-MPC1 [ mpc {123} ~/unix ] div_multi
IU-MPC1 [ mpc {124} ~/unix ] ls
total 77
64 div_multi*
1 div_multi.c
12 mon.out
IU-MPC1 [ mpc {125} ~/unix ] prof div_multi mon.out
Profile listing generated Sun Feb 13 01:43:20 2005 with:
prof div_multi mon.out
-----
* -p[rocedures] using pc-sampling;
* sorted in descending order by total time spent in each procedure;
* unexecuted procedures excluded
-----
Each sample covers 8.00 byte(s) for 3.8% of 0.0254 seconds

%time    seconds  cum %    cum sec  procedure (file)
57.7      0.0146    57.7     0.01     div (<div_multi>)
26.9      0.0068    84.6     0.02     main (<div_multi>)
15.4      0.0039    100.0    0.03     multi (<div_multi>)

IU-MPC1 [ mpc {126} ~/unix ]

```



## C 프로그램 컴파일 (7/9)

Compile & Makefile

### 주요 옵션(-O)

\$ cc -O -o hello hello.c

- 최적화(optimization) 컴파일을 수행함
- 컴파일러에 따라 -O이외에 -O3, -O4 등의 다양한 최적화 옵션을 제공함

```

user1@infravalley:~
NewWorld [ ysmoon {258} ~/unix ] ls
hello.c
NewWorld [ ysmoon {259} ~/unix ] gcc -o hello1 hello.c
NewWorld [ ysmoon {260} ~/unix ]
NewWorld [ ysmoon {260} ~/unix ] gcc -O -o hello2 hello.c
NewWorld [ ysmoon {261} ~/unix ]
NewWorld [ ysmoon {261} ~/unix ] ls -l hello?
-rwxr-xr-x 1 ysmoon other 6116 2월 13일 01:12 hello1*
-rwxr-xr-x 1 ysmoon other 6108 2월 13일 01:12 hello2*
NewWorld [ ysmoon {262} ~/unix ]
NewWorld [ ysmoon {262} ~/unix ] hello1
Hello!
NewWorld [ ysmoon {263} ~/unix ]
NewWorld [ ysmoon {263} ~/unix ] hello2
Hello!
NewWorld [ ysmoon {264} ~/unix ]
NewWorld [ ysmoon {264} ~/unix ]

```



## C 프로그램 컴파일 (8/9)

Compile & Makefile

### 주요 옵션(-s)

```
$ cc -s hello.c
```

```
user1@infravalley:~
NewWorld [ ysmoon {289} ~/unix ] ls -l hello*
-rw-r--r-- 1 ysmoon other 76 2월 13일 01:12 hello.c
NewWorld [ ysmoon {290} ~/unix ] gcc -s hello.c
NewWorld [ ysmoon {291} ~/unix ] ls -l hello*
-rw-r--r-- 1 ysmoon other 76 2월 13일 01:12 hello.c
-rw-r--r-- 1 ysmoon other 520 2월 13일 01:16 hello.s
NewWorld [ ysmoon {292} ~/unix ] file hello.?
hello.c: C 프로그램 텍스트
hello.s: 어셈블러 프로그램 텍스트
NewWorld [ ysmoon {293} ~/unix ] head -10 hello.s
.file "hello.c"
gcc2 compiled.:
.section ".rodata"
.align 8
.LLC0:
.asciz "Hello!\n"
.section ".text"
.align 4
.global main
.type main,#function
NewWorld [ ysmoon {294} ~/unix ] gcc -o hello hello.s
NewWorld [ ysmoon {295} ~/unix ] ./hello
Hello!
NewWorld [ ysmoon {296} ~/unix ]
```



Page 9

UNIX System Programming  
by Yang-Sae Moon



## C 프로그램 컴파일 (9/9)

Compile & Makefile

### 주요 옵션(-l)

```
$ cc -o math_hello math_hello.c -lm
```

```
user1@infravalley:~
NewWorld [ ysmoon {315} ~/unix ] ls
math_hello.c practice/
NewWorld [ ysmoon {316} ~/unix ] cat math_hello.c
#include <stdio.h>
#include <math.h>

main()
{
    printf("Hello! sin(3.141592) = %.8f\n", sin(3.141592));
}
NewWorld [ ysmoon {317} ~/unix ] gcc -o math_hello math_hello.c
정의를 하지 않은 기호 첫번째 참조된 파일의
ld: 치명적: 기호 참조 오류. math_hello에 출력이 기록되지 않음
collect2: ld returned 1 exit status
NewWorld [ ysmoon {318} ~/unix ] ls /usr/lib/libm.a
/usr/lib/libm.a
NewWorld [ ysmoon {319} ~/unix ] gcc -o math_hello math_hello.c -lm
NewWorld [ ysmoon {320} ~/unix ] ./math_hello
Hello! sin(3.141592) = 0.00000065
NewWorld [ ysmoon {321} ~/unix ]
```

용합  
링크



Page 10

UNIX System Programming  
by Yang-Sae Moon



## 단일 모듈 프로그램 (1/2)

Compile & Makefile

☞ 하나의 C 프로그램 파일로 프로그래밍 된 경우

☞ 매우 간단한 프로그램인 경우에 단일 모듈 프로그램이 유리함

```
user1@infravalley:~  
NewWorld [ ysmoon {333} ~/unix ] cat main.c  
#include <stdio.h>  
  
int operation_plus(int, int), operation_minus(int, int);  
  
main()  
{  
    int i, j;  
  
    printf("input two numbers: "); scanf("%d%d", &i, &j);  
    printf("the result of plus operation is %d\n", operation_plus(i,j));  
    printf("the result of minus operation is %d\n", operation_minus(i,j));  
}  
  
int operation_plus(int a, int b)  
{  
    return (int)(a + b);  
}  
  
int operation_minus(int a, int b)  
{  
    return (int)(a - b);  
}  
NewWorld [ ysmoon {334} ~/unix ]
```



## 단일 모듈 프로그램 (2/2)

Compile & Makefile

```
user1@infravalley:~  
NewWorld [ ysmoon {338} ~/unix ] ls  
main.c      practice/  
NewWorld [ ysmoon {339} ~/unix ] gcc -o main main.c  
NewWorld [ ysmoon {340} ~/unix ]  
NewWorld [ ysmoon {340} ~/unix ] ls  
main*      main.c      practice/  
NewWorld [ ysmoon {341} ~/unix ]  
NewWorld [ ysmoon {341} ~/unix ] main  
input two numbers: 300 100  
the result of plus operation is 400  
the result of minus operation is 200  
NewWorld [ ysmoon {342} ~/unix ]  
NewWorld [ ysmoon {342} ~/unix ] main  
input two numbers: 100 300  
the result of plus operation is 400  
the result of minus operation is -200  
NewWorld [ ysmoon {343} ~/unix ]
```



## 다중 모듈 프로그램 (1/8)

Compile & Makefile

- 여러 개의 C 프로그램 파일들로 프로그래밍 된 경우
- 복잡하며 대단위 프로그램인 경우에 다중 모듈 프로그램을 사용함  
(일반적으로, 모든 프로그램은 다중 모듈로 구성된다고 할 수 있음)
- 단일 모듈 프로그램의 문제점
  - 코드의 재사용(reuse)이 어렵고, 여러 사람이 참여하는 프로그래밍이 불가능함
  - 예를 들어, 앞서 "main.c" 프로그램을 작성하는데 있어서의 문제점은 다른 프로그램에서 "operation\_plus" 함수와 "operation\_minus" 함수를 사용할 수 없다는 점임
  - 즉, 다른 프로그램에서 operation\_plus 함수를 사용하고자 할 경우, 이 부분을 자르고 붙여서 원하는 프로그램에 삽입하여 사용하여야 하는 어려움이 있음



## 다중 모듈 프로그램 (2/8)

Compile & Makefile

- 함수의 재사용
  - 앞의 예에서 "operation\_plus" 함수를 공유하는 방법은 "main" 프로그램에서 해당 함수를 따로 분리하여 별도 파일로 작성한 후, 해당 파일을 컴파일한 후 원하는 프로그램에 링크하여 사용하는 것임
  - 이러한 기법은 동시에 많은 다른 프로그램들이 이 함수를 사용할 수 있게 하며 이러한 특성을 가진 함수를 재사용(reusable) 함수라 함



## 다중 모듈 프로그램 (3/8)

Compile & Makefile

### operation\_main.c의 소스 코드

```
user1@infravalley:~  
NewWorld [ ysmoon {352} ~/unix ]  
NewWorld [ ysmoon {352} ~/unix ] cat operation_main.c  
#include <stdio.h>  
#include "operation_plus.h"  
#include "operation_minus.h"  
  
main()  
{  
    int i, j;  
  
    printf("input two numbers: "); scanf("%d%d", &i, &j);  
    printf("the result of plus operation is %d\n", operation_plus(i,j));  
    printf("the result of minus operation is %d\n", operation_minus(i,j));  
}  
NewWorld [ ysmoon {353} ~/unix ]
```



## 다중 모듈 프로그램 (4/8)

Compile & Makefile

### 재사용 함수의 준비

- 재사용할 수 있는 함수를 준비하기 위해서는,
  - 함수의 소스 코드를 포함하는 소스 코드 파일과
  - 함수의 prototype을 포함하는 헤더 파일을 함께 작성해야 하며,  
헤더 파일은 접미어 ".h"를 갖도록 이름 지어야 한다.
- 그 후에 cc(gcc)의 -c 옵션을 사용하여 소스 코드 모듈을 컴파일하여 ".o" 형태의 object file을 생성한다.
- Object file은 실행 파일이 생성될 때 다른 object file과 결합이 가능하게 해주는 symbol table 정보와 함께 machine code를 포함한다.





## 다중 모듈 프로그램 (5/8)

Compile & Makefile

### operation\_plus.c와 operation\_minus.c 소스 코드

```
user1@infravalley:~
NewWorld [ ysmoon {361} ~/unix ]
NewWorld [ ysmoon {361} ~/unix ] cat operation_plus.c
#include <stdio.h>
#include "operation_plus.h"

int operation_plus(int a, int b)
{
    return (int)(a + b);
}
NewWorld [ ysmoon {362} ~/unix ]
NewWorld [ ysmoon {362} ~/unix ] cat operation_minus.c
#include <stdio.h>
#include "operation_minus.h"

int operation_minus(int a, int b)
{
    return (int)(a - b);
}
NewWorld [ ysmoon {363} ~/unix ]
```



Page 17

UNIX System Programming  
by Yang-Sae Moon



## 다중 모듈 프로그램 (6/8)

Compile & Makefile

### operation\_plus.h와 operation\_minus.h 소스 코드

```
user1@infravalley:~
NewWorld [ ysmoon {369} ~/unix ] cat operation_plus.h
/* function prototype of operation_plus() */
int operation_plus(int, int);
NewWorld [ ysmoon {370} ~/unix ]
NewWorld [ ysmoon {370} ~/unix ] cat operation_minus.h
/* function prototype of operation_minus() */
int operation_minus(int, int);
NewWorld [ ysmoon {371} ~/unix ]
```

### 헤더 파일에 포함되는 일반적인 내용

- 함수의 prototype (상기 파일이 예제에 해당함)
- 상수에 대한 정의 (#define MY\_PI 3.141592)
- 전역 변수에 대한 정의 (extern int common\_pi)



Page 18

UNIX System Programming  
by Yang-Sae Moon



## 다중 모듈 프로그램 (7/8)

Compile & Makefile

### 개별 파일을 각기 컴파일하고 링크하기

- 각 소스 코드 파일을 개별적으로 번역하기 위해서는 cc(gcc)의 -c 옵션을 사용
- 각 소스 코드 파일은 ".o"를 갖는 각기 다른 object file을 생성하게 됨
- 각각의 object file을 링크하여 실행 파일을 생성함

```
user1@infravalley:~$ ls
NewWorld [ ysmoon {378} ~/unix ] ls
operation_main.c  operation_minus.h  operation_plus.h
operation_minus.c  operation_plus.c  practice/
NewWorld [ ysmoon {379} ~/unix ] gcc -c operation_*.c
NewWorld [ ysmoon {380} ~/unix ] ls
operation_main.c  operation_minus.h  operation_plus.h
operation_main.o  operation_minus.o  operation_plus.o
operation_minus.c  operation_plus.c  practice/
NewWorld [ ysmoon {381} ~/unix ] gcc -o main operation_*.o
NewWorld [ ysmoon {382} ~/unix ] main
input two numbers: 100 50
the result of plus operation is 150
the result of minus operation is 50
NewWorld [ ysmoon {383} ~/unix ]
```



## 다중 모듈 프로그램 (8/8)

Compile & Makefile

### 다중 모듈 프로그램의 효과적인 컴파일 방법

- 변경된 파일에 대해서만 컴파일을 수행하고,
- 파일들의 연관 관계(dependency)에 따라서 필요한 파일만 다시 컴파일하며,
- 복잡한 형태의 대단위 프로그램의 컴파일을 용이하게 하기 위하여,

→ Makefile을 작성하고 make 명령어를 사용하여 컴파일을 수행함

```
user1@infravalley:~$ cat Makefile
NewWorld [ ysmoon {409} ~/unix ] make
Makefile      opegcc -c operation_main.c
operation_main.c  opegcc -c operation_plus.c
NewWorld [ ysmoon {410} ~/unix ] gcc -c operation_minus.c
main: operation_main.o gcc -o main operation_main.o operation_plus.o operation_minus.o
gcc -o $@ operation_main.o operation_plus.o operation_minus.o
NewWorld [ ysmoon {413} ~/unix ] main
input two numbers: 100 50
the result of plus operation is 150
the result of minus operation is 50
NewWorld [ ysmoon {414} ~/unix ]
NewWorld [ ysmoon {411} ~/unix ]
```



### Makefile의 필요성

- 다중 모듈 프로그램은 재사용성과 디스크 공간의 개념에서는 효과적이지만, 유지하는데 신중성이 필요
  - 예를 들어, "operation\_plus.c" 소스 코드를 수정한다면, 이 함수를 사용했던 모든 함수와 다시 링크 작업을 거쳐 실행파일을 생성해야 함
  - 비록 이러한 것이 큰 문제는 아닌 것처럼 보여도, 수천 개의 object file과 수백 개의 실행 파일을 갖는 시스템에서는 헤더, 소스 코드 파일, object file, 실행 파일의 모든 관계를 기억한다는 것은 매우 힘든 작업임
- Makefile과 make 유틸리티를 사용하여 효과적인 관리가 가능함



### "make" 유틸리티

- Make 유틸리티는 실행 파일에 대해 파일의 상호 의존 관계의 목록을 갖는 Makefile을 생성하도록 허용함
  - 일단 Makefile 파일이 생성되면, 실행 파일을 다시 만드는 것은 매우 쉬운 작업이 됨
- ```
$ make [-f makefile_name]
```
- make는 "Makefile"이라는 특수한 형식에 저장되어 있는 일련의 의존 규칙들에 근거하여 파일을 최신 버전으로 개정하는 유틸리티임
  - "-f" 옵션은 make의 대상(입력)이 되는 Makefile 이름을 명시할 수 있게 하며, 파일 이름이 명시하지 않으면 default로 "Makefile"을 입력으로 간주



## Makefile 구성

- “make”를 사용하여 실행 파일을 관리하기 위해서는 우선 Makefile을 만들어야 함
- 이 파일은 실행 파일을 만들기 위해서 사용되는 파일들 사이에 존재하는 상호 의존 관계의 목록을 포함해야 함
- Makefile은 어떠한 이름이라도 가질 수 있지만 (일반적으로) 실행 파일의 이름에 “.make”(혹은 “.mk”)라는 확장자를 붙여 다른 파일들과 구분함
- Makefile의 일반적인 구성형식은 다음과 같음

```
targetList: dependencyList
        commandList
```

- targetList은 object file(or 실행 파일)의 목록이고, dependencyList는 targetList에 있는 파일들이 의존하는 파일의 목록이며, commandList는 명령어의 목록으로 의존 파일로부터 object 파일을 재구성함
- 명령어 리스트 내에 있는 각각의 줄은 탭(tab) 문자에 의해서 시작되어야 함



## Makefile 구성 (계속)

- 예를 들어, 실행파일 “main”과 관련된 파일 상호 의존에 대하여 살펴보면 이 파일은 operation\_plus.o와 operation\_minus.o 와 operation\_main.o 등의 object file로 구성
- 만일 세 개의 파일 중 어느 하나의 파일이 변경되었다면, 컴파일러를 사용하여 이들 파일을 링크함으로써, main은 재구성될 수 있음
- 그러므로 “main.mk”내의 하나의 규칙은 다음과 같다.

```
main: operation_main.o operation_plus.o operation_minus.o
        gcc -o main operation_main.o operation_plus.o operation_minus.o
```

- 이제 세 개의 목적파일에 대해서도 동일한 과정을 전개하여야 함



### Makefile 구성 (계속)

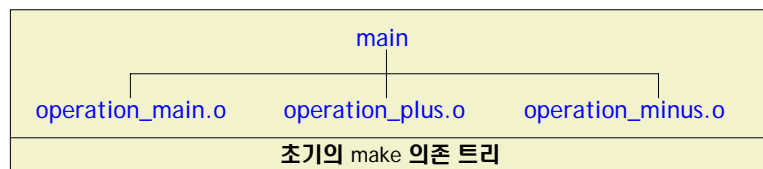
- 파일 operation\_main.o는 세 개의 파일 "operation\_main.c", "operation\_plus.h", "operation\_minus.h"에 관계됨
- 즉, 이 세 개의 파일 중 하나라도 변경되면 "operation\_main.c"를 컴파일하여 "operation\_main.o"가 재구성되어야 함
- 다음은 main.mk내의 나머지 규칙을 나타낸다.

```
operation_main.o: operation_main.c operation_plus.h operation_minus.h
    gcc -c operation_main.c
operation_plus.o: operation_plus.c operation_plus.h
    gcc -c operation_plus.c
operation_minus.o: operation_minus.c operation_minus.h
    gcc -c operation_minus.c
```



### Make 규칙의 순서

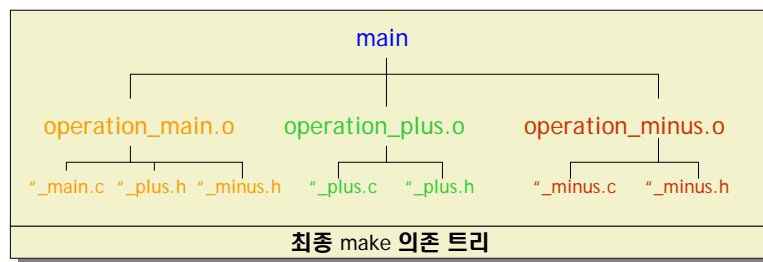
- make 유틸리티는 첫 번째 규칙을 초기에 조사함으로써, 상호 의존을 나타내는 트리를 생성한다.
- 첫 번째 규칙에 있는 각 object file은 의존 트리의 루트가 되고, 그것들의 의존 리스트에 있는 파일은 각 루트 노드의 리프 노드로서 추가 됨





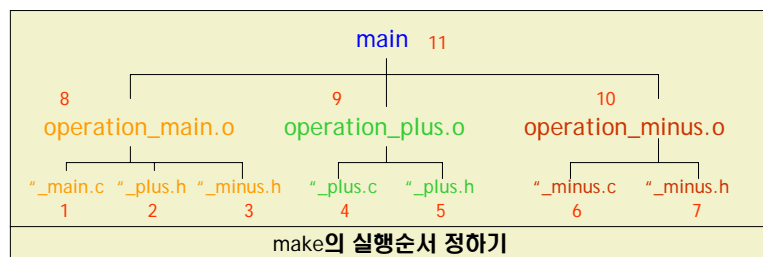
### Make 규칙의 순서 (계속)

- make 유틸리티는 의존 리스트 내의 각 파일과 연관된 각 규칙에 대해서도, 동일한 동작을 반복한다.
- 앞의 예에 대한 최종적인 트리는 다음과 같다.



### Make 규칙의 순서 (계속)

- make 유틸리티는 리프 노드로부터 루트 노드까지 진행(traverse)하며, 부모 노드의 마지막 수정 시간보다 자식 노드의 마지막 수정 시간이 더 최근인지를 살펴본다.
- 이와 같이 수정시간을 비교하면서 명령어(컴파일)를 수행하여 최근 버전을 생성한다.





## Makefile (9/11)

Compile & Makefile

### Makefile 예제: main.mk

```
user1@infravalley:~
NewWorld [ ysmoon {427} ~/unix ] cat main.mk
#
# simple makefile by ysmoon
#
CC      = gcc
PROGS   = main

all: $(PROGS)

main: operation_main.o operation_plus.o operation_minus.o
    $(CC) -o $@ operation_main.o operation_plus.o operation_minus.o

operation_main.o: operation_main.c operation_plus.h operation_minus.h
    $(CC) -c operation_main.c

operation_plus.o: operation_plus.c operation_plus.h
    $(CC) -c operation_plus.c

operation_minus.o: operation_minus.c operation_minus.h
    $(CC) -c operation_minus.c
NewWorld [ ysmoon {428} ~/unix ]
```



Page 29

UNIX System Programming  
by Yang-Sae Moon



## Makefile (10/11)

Compile & Makefile


### Makefile의 실행 (\$ make [-f makefile])

```
user1@infravalley:~
NewWorld [ ysmoon {448} ~/unix ] ls
main.mk      operation_minus.c  operation_plus.c  practice/
operation_main.c  operation_minus.h  operation_plus.h
NewWorld [ ysmoon {449} ~/unix ] make -f main.mk
gcc -c operation_main.c
gcc -c operation_plus.c
gcc -c operation_minus.c
gcc -o main operation_main.o operation_plus.o operation_minus.o
NewWorld [ ysmoon {450} ~/unix ] main
input two numbers: 100 50
the result of plus operation is 150
the result of minus operation is 50
NewWorld [ ysmoon {451} ~/unix ] make -f main.mk
NewWorld [ ysmoon {452} ~/unix ] touch operation_plus.h
NewWorld [ ysmoon {453} ~/unix ] make -f main.mk
gcc -c operation_main.c
gcc -c operation_plus.c
gcc -o main operation_main.o operation_plus.o operation_minus.o
NewWorld [ ysmoon {454} ~/unix ]
```




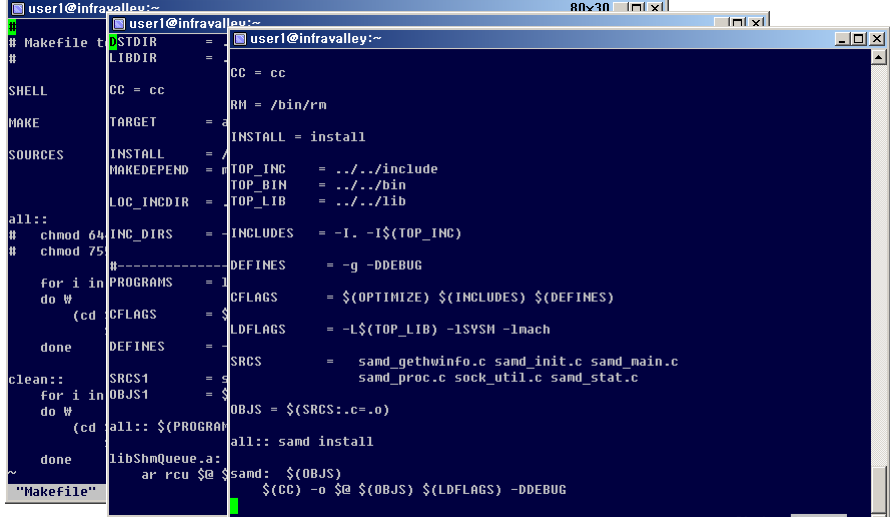
Page 30

UNIX System Programming  
by Yang-Sae Moon


**Makefile (11/11)**

Compile & Makefile


**복잡한 형태의 Makefile 예제**



```

user1@infravalley:~$ cat Makefile
# Makefile for sand
#
# Compiler and linker
CC = cc
RM = /bin/rm
INSTALL = install

# Source files and directories
SOURCES = sand_gethwinf.c sand_init.c sand_main.c
          sand_proc.c sock_util.c sand_stat.c
INC_DIRS = -I. -I$(TOP_INC)
LIBDIR =

# Flags
CFLAGS = $(OPTIMIZE) $(INCLUDES) $(DEFINES)
LDFLAGS = -L$(TOP_LIB) -lsysm -lnach


# Targets
all: sand install

# Rules
$(SOURCES):
$(CC) -o $@ $(OBJS) $(LDFLAGS) -DDEBUG


clean:
rm -f $(SOURCES) $(OBJS) $(EXECUTABLES)

.PHONY: all clean install


```


 Page 31


by Yang-Sae Moon


**기타 유용한 유틸리티**


Compile & Makefile


**ar: 파일 archive 유틸리티**

- 여러 개의 object file(.o)들을 하나로 묶어서 관리할 수 있도록 함
- 일반적으로 library(.a)를 만들 때 사용함


**SCCS: 소스 코드 관리 유틸리티**

- 대형 프로젝트를 진행하는 경우에 여러 버전의 소스를 관리해야 함
- 특히, 여러 사람이 관련된 프로그래밍을 수행할 경우에, 실수 방지 및 history 관리를 위하여 버전에 따른 소스 관리가 필수적인 요소임
- SCCS는 UNIX에서 제공하는 기본적인 소스 코드 관리 유틸리티임


 Page 32

UNIX System Programming  
by Yang-Sae Moon