설계과제 개요 : SoongSil Make

Linux System Programming by Jiman Hong (jiman@ssu.ac.kr),
Sping 2018, School of CSE, Soongsil University

1. 개요

make 유틸리티는 소프트웨어 개발을 위해 유닉스 개열 운영 체제에서 주로 사용되는 프로그램 빌드 도구이다. make 유틸리티는 여러 파일들끼리의 의존성과 각 파일에 필요한 명령을 정의함으로써 크기가 큰 프로그램을 컴파일 할 때 재컴파일이 필요한 소스코드만 컴파일이 되도록 한다.

2. 목표

make 유틸리티를 리눅스 시스템 함수를 사용하여 구현함으로써 make의 작동 원리 및 Makefile파일 작성 규칙에 대해 이해하고, 시스템 프로그램 및 설계 및 응용 능력을 향상시킨다.

3. 팀 구성

개인별 프로젝트

4. 개발환경

가. OS : **Ubuntu 16.04**
나. Tools : vi(m), gcc, gdb

5. 보고서 제출 방법

가. 제출할 파일
  가) 보고서와 소스파일을 함께 압축하여 제출
    (1) 보고서 파일 : 워드(hwp 또는 MS-Word)로 작성
    (2) 압축파일명 : #P1_학번_버전.zip (예. #P1_20180000_v1.0.zip)
나. 제출할 곳
  가) http://oslab.ssu.ac.kr/main 접속 [2018 LSP] ⇒ [Homework] ⇒ [과제제출]
  나) 게시물 제목은 파일이름과 동일함
  다) 압축된 파일을 첨부하여 과제 제출
다. 제출 기한
  가) 4월 1일(일) 오후 11시 59분 59초

6. 보고서 양식

보고서는 다음과 같은 양식으로 작성

1. 과제 개요
2. 설계
3. 구현
   - 각 함수별 기능
4. 테스트 및 결과
   - 테스트 프로그램의 실행 결과를 분석
5. 소스코드(주석 포함)

7. 설계 및 구현

가. ssu_make

1) ssu_make의 규칙은 GNU make의 규칙을 완전히 따르지 않음

2) ssu_make는 Makefile을 규칙에 따라 parsing 후 실행

나. 프로그램 명세

1) ssu_make [OPTION] [TARGET] [MACRO]

2) [OPTION]

가) -f filename : make가 사용할 파일을 Makefile에서 filename으로 변경

(1) -f 옵션이 없을 경우 기본 값으로 Makefile 사용

(2) Makefile, filename을 이름으로 하는 파일이 없을 경우 에러 출력

실행 예시 -f 옵션

```
kym@kym-ZBOX-ID91:~$ ./ssu_make
make: Makefile: No such file or directory
kym@kym-ZBOX-ID91:~$ ./ssu_make -f test1
make: test1: No such file or directory
kym@kym-ZBOX-ID91:~$ ./ssu_make -f ./assign1/test1
make: 'test_make1' is up to date.
```

나) -c directory: 작업 디렉터리를 현재 위치에서 directory로 변경

(1) -c 옵션이 없는 경우 기본 값으로 shell의 현재 작업 디렉터리를 사용

(2) Makefile의 include가 상대 경로를 사용할 경우 현재 작업 디렉터리가 기준임

실행 예시 -c 옵션

```
kym@kym-ZBOX-ID91:~$ ./ssu_make -ftest1
make: test1: No such file or directory
kym@kym-ZBOX-ID91:~$ ./ssu_make -c assign1 -f test1
make: 'test make1' is up to date.
```

다) -s : command를 출력하지 않음

실행 예시 -s 옵션

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test1
gcc -c test_code1.c
gcc -o test_make1 test_code1.o
kym@kym-ZBOX-ID91:~/assign1$ vim test_code1.c
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test1 -s
```

라) -h : 사용법 출력

실행 예시 -h 옵션

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -h
Usage : ssu_make [Target] [Option] [Macro]
Option:
 -f <file>      Use <file> as a makefile.
 -c <directory> Change to directory <directory> before reading the makefiles.
 -s             Do not print the commands as they are executedt
 -h             print usage
 -m             print macro list
 -t             print tree
```

마) -m : Makefile 및 ssu_make 실행 시 입력한 매크로를 출력

(1) -m 옵션이 있는 경우 command는 실행되지 않음

(2) Makefile에서 다른 파일을 include하였을 경우 해당 파일의 매크로도 출력

실행 예시 -m 옵션

```
CC = gcc
USER = OSLAB
WELCOME = "WELCOME MESSAGE"
OS ?= LINUX

test_make1 : test_code1.o
	$(CC) -o test_make1 test_code1.o

test_code1.o : test_code1.c
	gcc -c test_code1.c
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test1 -m
-----------------macro list---------------------
CC-> gcc
USER-> OSLAB
WELCOME-> "WELCOME MESSAGE"
OS-> LINUX
```

바) -t : 의존성 그래프 출력

실행 예시 -t 옵션

```
ssu_make : main.o data_structure
data_structure : graph.o list.o
main.o : main.c main.h
graph.o : graph.c graph.h
list.o : list.c list.h
a : b
b : a
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test7 -t
---------------------graph----------------------
root┬ssu_make┬main.o┬main.c
    │        │       └main.h
    │        └data_structure┬graph.o┬graph.c
    │                       │        └graph.h
    │                       └list.o┬list.c
    │                              └list.h
    ├data_structure┬graph.o┬graph.c
    │              │       └graph.h
    │              └list.o┬list.c
    │                     └list.h
    ├main.o┬main.c
    │      └main.h
    ├graph.o┬graph.c
    │       └graph.h
    ├list.o┬list.c
    │      └list.h
    ├a─b─a
    └b─a─b
```

3) [TARGET]

가) Makefile에서 실행 될 target을 지정

나) target이 없을 경우 기본 값으로 가장 위에 있는 target을 기본 값으로 사용

다) 최대 5개까지 지정 가능

4) [MACRO]

가) Makefile에서 사용할 매크로 지정

나) macro=value, macro="value"의 형태만 가능

다) 최대 5개까지 지정 가능

실행 예시 ssu_make 실행 시 macro 지정

```
CC = gcc
USER = OSLAB
WELCOME = "WELCOME MESSAGE"
OS ?= LINUX

test_make1 : test_code1.o
	$(CC) -o test_make1 test_code1.o

test_code1.o : test_code1.c
	gcc -c test_code1.c
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test1 -m OS=UNIX LANG=KO
-----------------macro list--------------------
CC-> gcc
USER-> OSLAB
WELCOME-> "WELCOME MESSAGE"
OS->UNIX
LANG->KO
```

5) Makefile 규칙

가) ssu_make의 Makefile은 아래의 형태만 가능함

```
1. Makefile의 의존성 및 실행 command 설정
target : <dependency1> <dependency2> ... <dependencyN>
<탭>command1
<탭>command2
<탭>commandN

2. macro를 value로 정의
macro = value

3. macro가 정의되어있지 않으면 macro를 value로 정의
macro ?= value

4. Makefile에 다른 파일 추가
include pathname1 pathname2 ... pathnameN

5. 주석
#
```

(1) command 라인은 탭으로 시작해야하고, 나머지는 공백으로 시작할 수 없음

(2) Makefile의 구성요소는 탭 또는 공백 문자로 구분함

(3) '=', '?=', ':' 앞뒤에는 탭 또는 공백 문자가 없을 수 있음

(4) Makefile이 규칙과 다른 경우 오류가 있는 라인 번호를 출력하고 종료

실행 예시 Makefile이 규칙과 다른 경우

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test5
test5:3: *** missing separator.  Stop.
```

나) 긴 명령어의 경우 \를 사용하여 여러 문장으로 표시할 수 있음

실행 예시 긴 문장 처리

```
PRINT = "HELLO  \
         WORLD"

test_make2 : test_source2.c \
             test_header2.h
        gcc -o test_make2 test_source2.c test_header2.h
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test2 -m
-----------------macro list--------------------
PRINT-> "HELLO           WORLD"
```

다) Makefile은 한 개 이상의 target이 있어야 함

실행 예시 Makefile에 target이 없는 경우

```
CC = gcc
USER = OSLAB
WELCOME = "WELCOME MESSAGE"
OS ?= LINUX
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make
make: *** No targets.  Stop.
```

라) dependency는 target에서 먼저 찾고 없을 경우 현재 디렉터리의 파일을 찾음, 현재 디렉터리에도 dependency가 없을 경우 에러

실행 예시 ssu_make 실행 시 지정한 target이 Makefile에 없는 경우

```
test_make3 : test_code3.o
        gcc -o $@ test_code3.o

test_code3.o : test_code3.c
        gcc -c $*.c
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test3 test_make3 test
make: 'test_make3' is up to date.
make: *** No rule to make target 'test'.  Stop.
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test3 test test_make3
make: *** No rule to make target 'test'.  Stop.
```

실행 예시 dependency가 없는 경우

```
clear : clear_a
        clear
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test4
make: *** No rule to make target 'clear_a', needed by 'clear'.  Stop
```

마) 의존성 그래프에 순환이 생긴 경우 중첩된 target부터는 실행되지 않음

실행 예시 의존성 그래프에 순환이 있는 경우

```
a : b
        echo "aa"

b : c
        echo "bb"

c : a b
        echo "cc"

d : d
        echo "dd"
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test6 a
make: Circular c <- a dependency dropped.
make: Circular c <- b dependency dropped.
echo "cc"
cc
echo "bb"
bb
echo "aa"
aa
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test6 d
make: Circular d <- d dependency dropped.
echo "dd"
dd
```

바) target과 dependency의 이름을 갖는 파일이 현재 디렉터리에 존재할 경우 두 파일의 최근 수정 시간을 비교하여 target의 시간이 dependency의 시간보다 최신일 경우 command는 실행하지 않음

(1) target, dependency 중 파일이 아닌 것이 있을 경우 command는 실행됨

실행 예시 ssu_make

```
test_make3 : test_code3.o
        gcc -o $@ test_code3.o

test_code3.o : test_code3.c
        gcc -c $*.c
```

```
kym@kym-ZBOX-ID91:~/assign1$ vim test_code3.c
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test3
gcc -c test_code3.c
gcc -o test_make3 test_code3.o
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test3
make: 'test_make3' is up to date.
```

사) 내부 매크로

(1) ssu_make는 아래 매크로만을 내부 매크로로 사용함

(가) $* : 확장자가 없는 현재의 목표 파일

(나) $@ : 현재 목표 파일(target)

실행 예시 내부 매크로

```
test_make3 : test_code3.o
	gcc -o $@ test_code3.o

test_code3.o : test_code3.c
	gcc -c $*.c
```

```
kym@kym-ZBOX-ID91:~/assign1$ ./ssu_make -f test3
gcc -c test_code3.c
gcc -o test_make3 test_code3.o
```

다. 세부 기능 및 기능별 요구 조건

1) 입력 요구조건

가) ssu_make 실행 시 지정 가능한 매크로와 target의 수는 각각 5개를 넘을 수 없음

나) 인자들의 순서는 바뀔 수 있음

다) 모든 옵션은 동시에 쓸 수 있음

(1) -h 옵션도 동시에 쓸 수 있으나 사용법을 출력 후 ssu_make가 종료 됨

2) 출력 요구조건

가) 실행된 command는 표준 출력으로 출력

나) command는 system() 함수를 통해 실행

다) 의존성 그래프 출력의 경우 들여쓰기를 통해 부모와 자식 사이에 관계를 출력

(1) 같은 부모의 자식일 경우 시작 위치가 같으면 됨

과제 구현에 필요한 함수(필수 사용 X)

1. recomp(), regexec()

- 리눅스 시스템에서 정규 표현식을 처리하기 위해 사용하는 함수
- 정규 표현식을 regcomp()함수를 통해 컴파일 한 후 regexec()함수로 문장에 패턴에 있는지 확인

```
#include <regex.h>
int regexec(const regex_t *preg, const char *string,
            size_t nmatch, regmatch_t *pmatch, int eflags);
```

리턴값 : 성공 시 0, 오류 시 0이 아닌 값(regerror()함수로 사용)

```
#include <regex.h>
int regcomp(regex_t *preg, const char *pattern, int cflags);
```

리턴값 : 성공 시 0, 오류 시 에러 코드(regerror()함수로 사용)

regexec_example.c

```
#include <stdio.h>
#include <stdlib.h>
#include <regex.h>
int main(int argc, char *argv[])
{
        regex_t preg;
        char *pattern = argv[1];
        char *string = argv[2];
        int error_code;
        char buf[BUFSIZ];

        if((error_code = regcomp(&preg, pattern, REG_EXTENDED)) != 0){
                regerror(error_code, &preg, buf, BUFSIZ);
                fprintf(stderr, "regcomp error : %s\n", buf);
                exit(1);
        }
        if((error_code = regexec(&preg, string, 0, NULL, 0)) != 0){
                regerror(error_code, &preg, buf, BUFSIZ);
                fprintf(stderr, "regexec error : %s\n", buf);
                exit(1);
        }
        else{
                printf("match\n");
        }
}
```

실행결과

```
oslab@localhost:~$ ./regexec_example Soo[a-z]+University SoongsilUniversity
match
oslab@localhost:~$ ./regexec_example Soo[a-z]+University SooUniversity
regexec error : No match
```

2. getopt()

- 리눅스 시스템에서 옵션을 처리하기 위해 사용하는 함수
- 옵션이 별도의 파라미터를 받는 경우 콜론으로 표시
- 전역변수 optarg : 옵션 뒤 별도의 파라미터가 오는 경우 optarg에 문자열로 저장됨
- 전역변수 optind : 다음번 처리될 옵션의 익덱스

```
#include <unistd.h>
int getopt(int argc, char * const argv[], const char *optstring);

                                                    리턴값 : 성공 시 옵션 문자, 오류 시 -1
```

getopt_example.c

```
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[]){
        int flag_a=0, flag_b=0, flag_f=0;
        char *fname;
        int c;
        while( (c = getopt(argc, argv, "abf:")) != -1){
                switch(c){
                        case 'a':
                                flag_a = 1;
                                break;
                        case 'b':
                                flag_b = 1;
                                break;
                        case 'f':
                                flag_f = 1;
                                fname = optarg;
                                break;
                        case '?':
                                printf("Uknows flag : %c\n", optopt);
                                break;
                }
        }
        if(flag_a)
                printf("flag a is ON\n");
        if(flag_b)
                printf("flag a is ON\n");
        if(flag_f)
                printf("flag f is ON & File name is %s\n", fname);
}
```

실행결과

```
oslab@localhost:~$ ./getopt_example -a -f test
flag a is ON
flag f is ON & File name is test
```

8. 설계 구성 요소

가. 목표 설정

1) 주어진 요구 조건을 이해하고 명확한 설계 목표를 설정한다.

2) 설계의 목표 및 요구조건을 문서화한다.

나. 분석

1) 목표 설정에서 명시한 요구 조건을 분석하고 해결을 위한 기본 전략을 수립한다.

2) 문제 해결을 위한 배경 지식을 이론 강의에서 듣고 개별적으로 학생들이 다양한 경로를 통해 자료를 찾아 분석한다.

다. 합성 (구조 설계)

1) 분석 결과를 토대로 적절한 구조를 도출한다.

2) 도출된 구조를 토대로 모듈을 작성한다.

라. 제작 (구현)

1) 각 모듈의 입출력을 명세한다.

2) 구조와 모듈을 C 언어로 구현하고 이를 컴파일하여 실행 파일을 만든다.

마. 시험 및 평가(성능 평가)

1) 모듈의 입출력 명세에 따라, 다양한 입력 데이타를 작성하고 모듈을 테스트한다.

2) 작성된 실행 파일이 안정적으로 수행되는지 다양하게 테스트하고 이를 평가한다.

바. 결과 도출

1) 안정적으로 수행되는 최종 결과(Output)를 캡쳐하여 최종 결과물은 보고서에 반영한다.

9. 평가 준거(방법)

가. 평가 도구 1)에 대한 평가 준거

1) 소스 코드 분석 및 새로운 모듈의 설계가 제대로 이루어졌는가?

가) 설계 요구 사항을 제대로 분석하였는가?

나) 설계의 제약 조건을 제대로 반영하였는가?

다) 설계 방법이 적절한가?

2) 문서화

가) 소스 코드에 주석을 제대로 달았는가?

나) 설계 보고서가 잘 조직화되고 잘 쓰여졌는가?

나. 평가 도구 2)에 대한 평가 준거

1) 요구조건에 따라 올바르게 수행되는가?

10. 기타

가. 보고서 제출 마감은 제출일 자정까지

나. 지연 제출 시 감점

1) 1일 지연 시 마다 30% 감점

2) 3일 지연 후부터는 미제출 처리

다. 압축 오류, 파일 누락

1) 50% 감점 처리 (추후 확인)

라. copy 발견 시

1) F 처리

11. 점수 배점

가. Makefile 규칙 검사 40
나. 실행 조건에 맞는 command만 실행 30
다. 매크로 사용(-m 옵션 포함) 8
라. 실행 시 입력 받은 인자 처리 8
마. -f 옵션 구현 2
바. -c 옵션 구현 2
사. -h 옵션 구현 2
아. -s 옵션 구현 2
자. -t 옵션 구현 6

※ 필수적으로 구현해야 할 기능 : 1, 2