同济大学计算机系

计算机网络课程实验报告

学　　号 **1552239**

姓　　名 岳昊玮

专　　业 计算机科学与技术

授课老师 沈坚

§0.Linux 知识补充 - Makefile 文件的作用及编写方法

0 补充一下：什么是 Makefile

makefile 文件保存了编译器和连接器的参数选项,还表述了所有源文件之间的关系(源代码文件需要的特定的包含文件,可执行文件要求包含的目标文件模块及库等). 创建程序(make 程序)首先读取 makefile 文件,然后再激活编译器,汇编器,资源编译器和连接器以便产生最后的输出,最后输出并生成的通常是可执行文件.创建程序利用内置的推理规则来激活编译器,以便通过对特定 CPP 文件的编译来产生特定的 OBJ 文件.

1 Makefile 文件的作用

Makefile 关系到了整个工程的编译规则。一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，makefile 定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为 makefile 就像一个 Shell 脚本一样，其中也可以执行操作系统的命令。

Makefile 带来的好处就是——“自动化编译”，一旦写好，只需要一个 make 命令，整个工程完全自动编译，极大的提高了软件开发的效率。

Makefile 是和 make 命令一起配合使用的，make 是 一个命令工具，是一个解释 makefile 中指令的命令工具。

2 Makefile 文件的基本语法

Makefile 主要的 5 个部分 (显示规则, 隐晦规则, 变量定义, 文件指示, 注释)

a、显式规则。显式规则说明了，如何生成一个或多的目标文件。这是由 Makefile 的书写者明显指出，要生成的文件，文件的依赖文件，生成的命令。

b、隐晦规则。由于我们的 make 有自动推导的功能，所以隐晦的规则可以让我们比较粗糙地简略地书写 Makefile，这是由 make 所支持的。

c、变量的定义。在 Makefile 中我们要定义一系列的变量，变量一般都是字符串，这个有点你 C 语言中的宏，当 Makefile 被执行时，其中的变量都会被扩展到相应的引用位置上。

d、文件指示。其包括了三个部分，一个是在一个 Makefile 中引用另一个 Makefile，就像 C 语言中的 include 一样；另一个是指根据某些情况指定 Makefile 中的有效部分，就像 C 语言中的预编译#if 一样；还有就是定义一个多行的命令。

e、注释。Makefile 中只有行注释，和 UNIX 的 Shell 脚本一样，其注释是用“#”字符，这个就像 C/C++中的“//”一样。如果要在 Makefile 中使用“#”字符，可以用反斜框进行转义，如：“\#”。

Makefile 文件由一系列规则（rules）构成。每条规则的形式如下。

```
<target> : <prerequisites>
[tab]  <commands>
```

上面第一行冒号前面的部分，叫做"目标"（target），冒号后面的部分叫做"前置条件"（prerequisites）；第二行必须由一个 tab 键起首，后面跟着"命令"（commands）。

"目标"是必需的，不可省略；"前置条件"和"命令"都是可选的，但是两者之中必须至少存在一个。每条规则就明确两件事：构建目标的前置条件是什么，以及如何构建。

一个目标（target）就构成一条规则。目标通常是文件名，指明 Make 命令所要构建的对象，比如上文的 a.txt 。目标可以是一个文件名，也可以是多个文件名，之间用空格分隔。除了文件名，目标还可以是某个操作的名字，这称为"伪目标"（phony target）。

前置条件通常是一组文件名，之间用空格分隔。它指定了"目标"是否重新构建的判断标准：只要有一个前置文件不存在，或者有过更新（前置文件的 last-modification 时间戳比目标的时间戳新），"目标"就需要重新构建。

命令（commands）表示如何更新目标文件，由一行或多行的 Shell 命令组成。它是构建"目标"的具体指令，它的运行结果通常就是生成目标文件。行命令之前必须有一个 tab 键。如果想用其他键，可以用内置变量.RECIPEPREFIX 声明。.

关于下面的题目中所用到的写法，我在 makefile 文件中针对每一条语句都进行了注释。

```
# 需要排除的目录
exclude_dirs := include bin

# 取得当前子目录深度为1的所有目录名称
dirs := $(shell find . -maxdepth 1 -type d)
echo:$(dirs)
	echo $(dirs)
# basename 命令用于去掉路径信息，返回纯粹的文件名，如果指定的文件有扩展名，则将扩展名也一并去掉。
dirs := $(basename $(patsubst ./%,%,$(dirs)))
# filter-out 反过滤函数 和"filter"函数实现的功能相反。过滤掉字串"TEXT"中所有符合模式
dirs := $(filter-out $(exclude_dirs),$(dirs))

SUBDIRS := $(dirs)
# addprefix 添加前缀_clean_ ，避免clean子目录操作同名，
clean_dirs := $(addprefix _clean_,$(SUBDIRS) )
.PHONY: subdirs $(SUBDIRS) clean
# 执行默认make target
$(SUBDIRS):
	$(MAKE) -C $@
subdirs: $(SUBDIRS)
# 执行clean
$(clean_dirs):
	$(MAKE) -C $(patsubst _clean_%,%,$@) clean
clean: $(clean_dirs)
```

/1552239-000102/makefile

```
        #使用的编译器
compiler = gcc
        #编译的target目标，也就是语句所要处理的对象，具体到这里就是我们所要编译的程序
prom = test
        #编译的源文件
src = $(shell find ./ -name "*.c")
        #shell函数主要用于执行shell命令，
        #具体到这里就是找出当前目录下所有的.c和.h文件
        #使用shell的好处就是当需要往工程中添加一个.c或.h，不必手动往obj常量再添加一个.o或.h文件
obj = $(src:%.c=%.o)
        #$(src:%.c=%.o)是一个字符替换函数，
        #它会将src所有的.c字串替换成.o，实际上就等于列出了所有.c文件要编译的结果
$(prom): $(obj)
        $(compiler) -o $(prom) $(obj) -Wall -g
        #语法规定Makefile中的任何命令之前都必须要有一个tab缩进，否则make就会报错。
        #%.o:%.c，这是一个模式规则，表示所有的.o目标都依赖于与它同名的.c文件（当然还有deps中列出的头文件）
        #命令部分的$<和$@，其中$<代表的是依赖关系表中的第一项（如果我们想引用的是整个关系表，那么就应该使用$^）
        #具体到我们这里就是%.c。而$@代表的是当前语句的目标，即%.o
%.o: %.c $(deps)
        $(compiler) -c $< -o $@
        #终端中执行make clean命令时，它就会去删除该工程生成的所有编译文件。
clean:
        -rm -rf $(obj) $(prom)
```

/1552239-000102/01/makefile

```
        #使用的编译器
compiler = gcc
        #编译的target目标，也就是语句所要处理的对象，具体到这里就是我们所要编译的程序
prom = test
        #编译的头文件
deps = $(shell find ./ -name "*.h")
        #编译的源文件
src = $(shell find ./ -name "*.c")
        #shell函数主要用于执行shell命令，
        #具体到这里就是找出当前目录下所有的.c和.h文件
        #使用shell的好处就是当需要往工程中添加一个.c或.h，不必手动往obj常量再添加一个.o或.h文件
obj = $(src:%.c=%.o)
        #$(src:%.c=%.o)是一个字符替换函数，
        #它会将src所有的.c字串替换成.o，实际上就等于列出了所有.c文件要编译的结果
$(prom): $(obj)
        $(compiler) -o $(prom) $(obj)
        #语法规定Makefile中的任何命令之前都必须要有一个tab缩进，否则make就会报错。
        #%.o:%.c，这是一个模式规则，表示所有的.o目标都依赖于与它同名的.c文件（当然还有deps中列出的头文件）
        #命令部分的$<和$@，其中$<代表的是依赖关系表中的第一项（如果我们想引用的是整个关系表，那么就应该使用$^）
        #具体到我们这里就是%.c。而$@代表的是当前语句的目标，即%.o
%.o: %.c $(deps)
        $(compiler) -c $< -o $@
        #终端中执行make clean命令时，它就会去删除该工程生成的所有编译文件。
clean:
        -rm -rf $(obj) $(prom)
```

/1552239-000102/02/makefile

```
            # CFLAGS 表示用于 C 编译器的选项
CFLAGS = -g -Wall -Werror
            # -g 只是编译器，在编译时生成原生格式的调试符号信息
            # -wall 输出显示警告信息
            # -Werror 输出错误信息

            # LDFLAGS：gcc 等编译器会用到的一些优化参数，也可以在里面指定库文件的位置
LDFLAGS = -lpthread
CC = gcc
src = $(wildcard *.c)
            # patsubst  替换通配符  patsubst把$(src)中的变量符合后缀是.c的全部替换成.o，
target = $(patsubst %.c, %, ${src})

all: ${target}
            # phony 伪目标,为了避免在makefile中定义的只执行命令的目标和工作目录下的实际文件出现名字冲突
            # 比如在当前目录下面建一个名称为clean的文件，则make clean无法正常执行
.PHONY: all clean
            # $@  表示目标文件
            # %.o:%.c 用目录下的点c文件生成点o文件
%.o:%.c
            $(CC) ${CFLAGS} -c -o $@

%:%.o
            $(CC) ${LDFLAGS} -o $@

clean:
```

/1552239-000102/03/makefile

3、给出下列几种常用情况的 Makefile 文件的写法

3.2

将编写好的 makefile 和 3 个 test 文件放在 01 目录下面，进入 01 目录执行 make 命令，

```
[root@Anokoro ~]#  cd /home/1552239-岳昊玮/01
[root@Anokoro 01]# make
```

这里要注意的是，当因为 c++的代码中含有中文，因此在保存 c++代码时必须采用 gbk 的编码方式，否则会提示如下错误（未截全，但是都是由于编码方式而引起的错误）

```
[root@Anokoro 01]# make
g++    -c test1.cpp -o test1.o
g++    -c test2.cpp -o test2.o
test2.cpp:5:1: 错误：程序中有游离的'\342'
 printf(鈥滃渤鏄婂帇");//鎵撳嵃浣犵殑濮撳悕
 ^
test2.cpp:5:1: 错误：程序中有游离的'\200'
test2.cpp:5:1: 错误：程序中有游离的'\234'
test2.cpp:5:1: 错误：程序中有游离的'\345'
test2.cpp:5:1: 错误：程序中有游离的'\262'
test2.cpp:5:1: 错误：程序中有游离的'\263'
test2.cpp:5:1: 错误：程序中有游离的'\346'
```

Makefile 执行成功之后，运行./test，发现中文无法正确输出，必然也是编码的问题（这里我改成了.c 文件，上面那张图里面还是 cpp，是开始的时候没看清题目要求必须是.c，但是不管是 c 还是 cpp，上面那个问题都是一样的）

```
[root@Anokoro 01]# make
gcc -c test1.c -o test1.o
gcc -c test2.c -o test2.o
gcc -c test3.c -o test3.o
gcc -o test ./test1.o ./test2.o ./test3.o
[root@Anokoro 01]# ./test
1552239岳昊玮[root@Anokoro 01]#
```

如下图，还是直接在 linux 下编辑比较好，vim 编辑器

```
"test2.c" 7L, 90C 已写入
[root@Anokoro 01]# make clean
rm -rf ./test1.o ./test3.o ./test2.o  test
[root@Anokoro 01]# make
gcc -c test1.c -o test1.o
gcc -c test3.c -o test3.o
gcc -c test2.c -o test2.o
gcc -o test ./test1.o ./test3.o ./test2.o  -Wall -g
[root@Anokoro 01]# /home/1552239-岳昊玮/01/test
1552239岳昊玮
[root@Anokoro 01]# make
make: “test”是最新的。
```

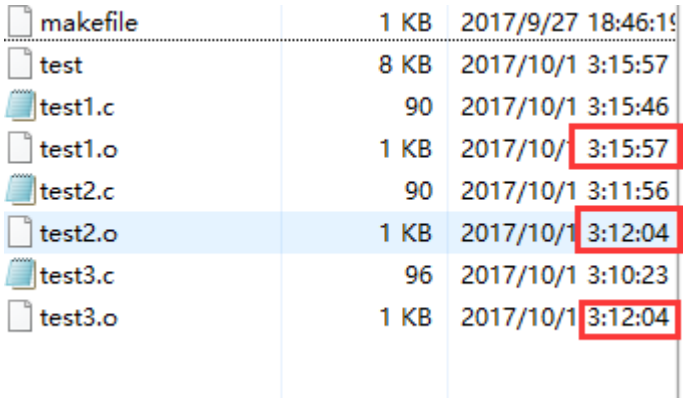
下面我在 test1.c 中的学号后面加个‘-’

```
//test1.c
#include <stdio.h>
int fun1()
{
        printf("1552239-");//打印你的学号
        return 0;
}
~
```

再次 make，效果如下所示，只重新编译了 test1.c

```
"test1.c" 7L, 90C 已写入
[root@Anokoro 01]# make
gcc -c test1.c -o test1.o
gcc -o test ./test3.o ./test2.o ./test1.o  -Wall -g
[root@Anokoro 01]# /home/1552239-岳昊玮/01/test
1552239-岳昊玮
[root@Anokoro 01]#
```

.o 文件的生成时间如下所示



3.3

```
[root@Anokoro 02]# make clean
rm -rf ./test3.o ./test1.o ./test2.o  test
[root@Anokoro 02]# make
gcc -c test3.c -o test3.o
gcc -c test1.c -o test1.o
gcc -c test2.c -o test2.o
gcc -o test ./test3.o ./test1.o ./test2.o
[root@Anokoro 02]# /home/1552239-岳昊玮/02/test
1552239-10岳昊玮
100
[root@Anokoro 02]#
```

下面把 a 改成 5 吧，

```
[root@Anokoro 02]# vim test.h
//test.h
#include <stdio.h>
int fun1();
int fun2();
#define a 5
```

```
[root@Anokoro 02]# make
gcc -c test3.c -o test3.o
gcc -c test1.c -o test1.o
gcc -c test2.c -o test2.o
gcc -o test ./test3.o ./test1.o ./test2.o
[root@Anokoro 02]# /home/1552239-岳昊玮/02/test
1552239-5岳昊玮
50
[root@Anokoro 02]#
```

可以看到3个.c都重新进行了编译

3.4

执行 make 后即可将 test1.c/test2.c/test3.c 分别编译为三个可执行文件 test1/test2/test3，效果如下

```
[root@Anokoro 03]# make
gcc -g -Wall -Werror  -lpthread  test2.c   -o test2
gcc -g -Wall -Werror  -lpthread  test3.c   -o test3
gcc -g -Wall -Werror  -lpthread  test1.c   -o test1
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test1
1552239
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test2
岳昊玮
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test3
1552239+岳昊玮[root@Anokoro 03]#
```

下面是 make 后面跟文件名进行单独的编译

```
[root@Anokoro 03]# make clean
rm -f  test2  test3  test1
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test1
-bash: /home/1552239-岳昊玮/03/test1: 没有那个文件或目录
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test2
-bash: /home/1552239-岳昊玮/03/test2: 没有那个文件或目录
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test3
-bash: /home/1552239-岳昊玮/03/test3: 没有那个文件或目录
[root@Anokoro 03]# make test1
gcc -g -Wall -Werror  -lpthread  test1.c   -o test1
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test1
1552239
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test2
-bash: /home/1552239-岳昊玮/03/test2: 没有那个文件或目录
[root@Anokoro 03]# make test2
gcc -g -Wall -Werror  -lpthread  test2.c   -o test2
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test2
岳昊玮
[root@Anokoro 03]# make test3
gcc -g -Wall -Werror  -lpthread  test3.c   -o test3
[root@Anokoro 03]# /home/1552239-岳昊玮/03/test3
1552239+岳昊玮
[root@Anokoro 03]#
```

这个题目有一个需要注意的是 3 个 test 文件的主函数名必须是 main，因为我是直接从前面 cp 过来的，所以刚开始编译的时候会有问题。

3.5

```
[root@Anokoro home]# cd 1552239-000102
[root@Anokoro 1552239-000102]# make
make -C 01
make[1]: 进入目录“/home/1552239-000102/01”
gcc -c test3.c -o test3.o
gcc -c test2.c -o test2.o
gcc -c test1.c -o test1.o
gcc -o test ./test3.o ./test2.o ./test1.o  -Wall -g
make[1]: 离开目录“/home/1552239-000102/01”
make -C 02
make[1]: 进入目录“/home/1552239-000102/02”
gcc -c test3.c -o test3.o
gcc -c test1.c -o test1.o
gcc -c test2.c -o test2.o
gcc -o test ./test3.o ./test1.o ./test2.o
make[1]: 离开目录“/home/1552239-000102/02”
make -C 03
make[1]: 进入目录“/home/1552239-000102/03”
gcc -g -Wall -Werror  -lpthread  test2.c   -o test2
gcc -g -Wall -Werror  -lpthread  test3.c   -o test3
gcc -g -Wall -Werror  -lpthread  test1.c   -o test1
make[1]: 离开目录“/home/1552239-000102/03”
echo 01 02 03
01 02 03
[root@Anokoro 1552239-000102]#
```

```
[root@Anokoro 1552239-000102]# make
make -C 01
make[1]: 进入目录“/home/1552239-000102/01”
make[1]: “test”是最新的。
make[1]: 离开目录“/home/1552239-000102/01”
make -C 02
make[1]: 进入目录“/home/1552239-000102/02”
make[1]: “test”是最新的。
make[1]: 离开目录“/home/1552239-000102/02”
make -C 03
make[1]: 进入目录“/home/1552239-000102/03”
```

如果本小题的 makefile 被改名为 mymake，那在使用 make 命令的时候就要进行指定：

make -f mymake 或者 make --file mymake

```
[root@Anokoro 01]# /home/1552239-000102/01/test
1552239-岳昊玮
[root@Anokoro 01]# /home/1552239-000102/02/test
1552239-5岳昊玮
50
[root@Anokoro 01]# /home/1552239-000102/03/test
-bash: /home/1552239-000102/03/test: 没有那个文件或目录
[root@Anokoro 01]# /home/1552239-000102/03/test1
1552239
[root@Anokoro 01]# /home/1552239-000102/03/test2
岳昊玮
[root@Anokoro 01]# /home/1552239-000102/03/test3
1552239+岳昊玮
```

```
[root@Anokoro 1552239-000102]# make clean
make -C 01 clean
make[1]: 进入目录“/home/1552239-000102/01”
rm -rf ./test3.o ./test2.o ./test1.o  test
make[1]: 离开目录“/home/1552239-000102/01”
make -C 02 clean
make[1]: 进入目录“/home/1552239-000102/02”
rm -rf ./test3.o ./test1.o ./test2.o  test
make[1]: 离开目录“/home/1552239-000102/02”
make -C 03 clean
make[1]: 进入目录“/home/1552239-000102/03”
rm -f  test2  test3  test1
make[1]: 离开目录“/home/1552239-000102/03”
[root@Anokoro 1552239-000102]#
```

3.6

压缩 tar.bz2

```
[root@Anokoro 1552239-000102]# tar -cjvf /home/linux-makefile.tar.bz2 /1552239-000102
tar: 从成员名中删除开头的“/”
/1552239-000102/
/1552239-000102/01/
/1552239-000102/01/makefile
/1552239-000102/01/test1.c
/1552239-000102/01/test2.c
/1552239-000102/01/test3.c
/1552239-000102/02/
/1552239-000102/02/makefile
/1552239-000102/02/test.h
/1552239-000102/02/test1.c
/1552239-000102/02/test2.c
/1552239-000102/02/test3.c
/1552239-000102/03/
/1552239-000102/03/makefile
/1552239-000102/03/test1.c
/1552239-000102/03/test2.c
/1552239-000102/03/test3.c
/1552239-000102/makefile
[root@Anokoro 1552239-000102]#
```