

浙江大学实验报告

课程名称: Linux 应用技术基础 实验类型: 综合型

实验项目名称: 实验三 程序设计

学生姓名: 应承峻 专业: 软件工程 学号: 3170103456

电子邮件地址: 3170103456@zju.edu.cn

实验日期: 2019 年 5 月 14 日

一、实验环境

计算机配置: 处理器 英特尔 Core i7-8750H @ 2.20GHz 六核
内 存 16 GB (三星 DDR4 2667MHz)
主硬盘 PeM280240GP4C15B (240 GB/固态硬盘)
显 卡 Nvidia GeForce GTX 1060 (6 GB)

操作系统环境: Windows 10 64 位 (DirectX 12)

Linux 版本: ubuntu-17.04

二、实验内容和结果及分析

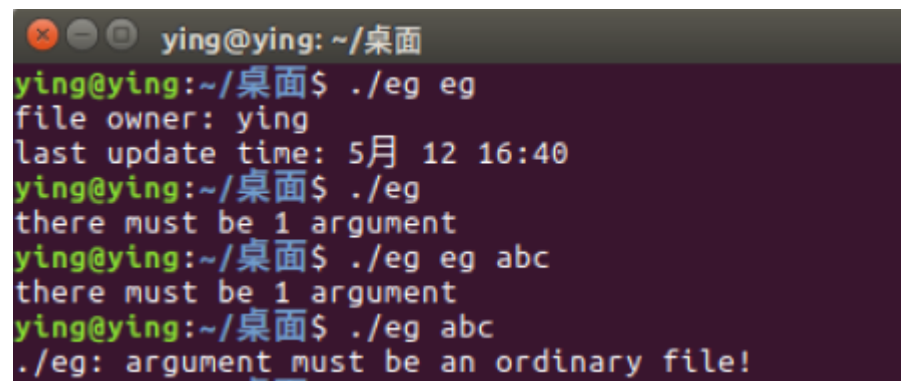
1. (15 分) 编写一个 shell 脚本程序, 它带一个命令行参数, 这个参数是一个文件。如果这个文件是一个普通文件, 则打印文件所有者的名字和最后的修改日期。如果程序带有多个参数, 则输出出错信息。

```
if test $# -ne 1 #不是一个参数
then
    echo "there must be 1 argument" #输出报错信息
    exit 1
fi
if test -f "$1" #文件类型是普通文件
then
    filename=$1 #文件类型是普通文件
    set -- $(ls -l $filename) #执行查询操作
    echo "file owner: $3" #输出文件所有者
```

```

    echo "last update time: $6 $7 $8" #输出月-日-时间
    exit 0
fi
echo "$1: argument must be an ordinary file!" #不是普通文件报错
exit 0

```



```

ying@ying: ~/桌面
ying@ying:~/桌面$ ./eg eg
file owner: ying
last update time: 5月 12 16:40
ying@ying:~/桌面$ ./eg
there must be 1 argument
ying@ying:~/桌面$ ./eg eg abc
there must be 1 argument
ying@ying:~/桌面$ ./eg abc
./eg: argument must be an ordinary file!

```

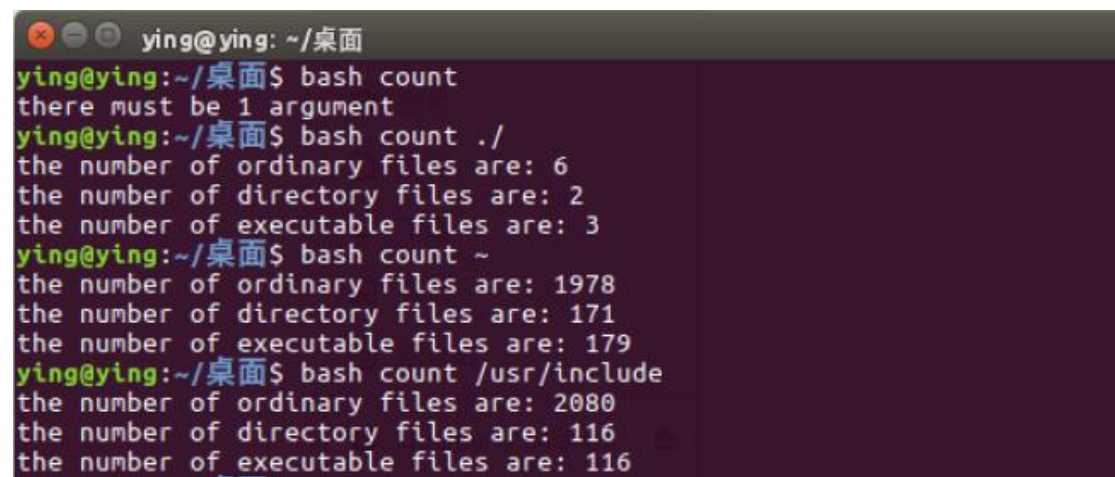
2. （15 分）编写 shell 程序，统计指定目录下的普通文件、子目录及可执行文件的数目，目录的路径名字由参数传入。

```

if test $# -ne 1 #不是一个参数，报错
then
    echo "there must be 1 argument"
    exit 1
fi
dcnt=0 #目录计数器
fcnt=0 #普通文件计数器
xcnt=0 #可执行文件计数器
array=$(find $1); #寻找当前目录下文件并存放到 array 数组中
for i in $array #遍历数组
do
    if [ -f $i ] #普通文件
    then
        ((fcnt++))
    fi
    if [ -d $i ] #目录文件
    then
        ((dcnt++))
    fi
    if [ -x $i ] #可执行文件
    then
        ((xcnt++))
    fi
done
echo "the number of ordinary files are: $fcnt"

```

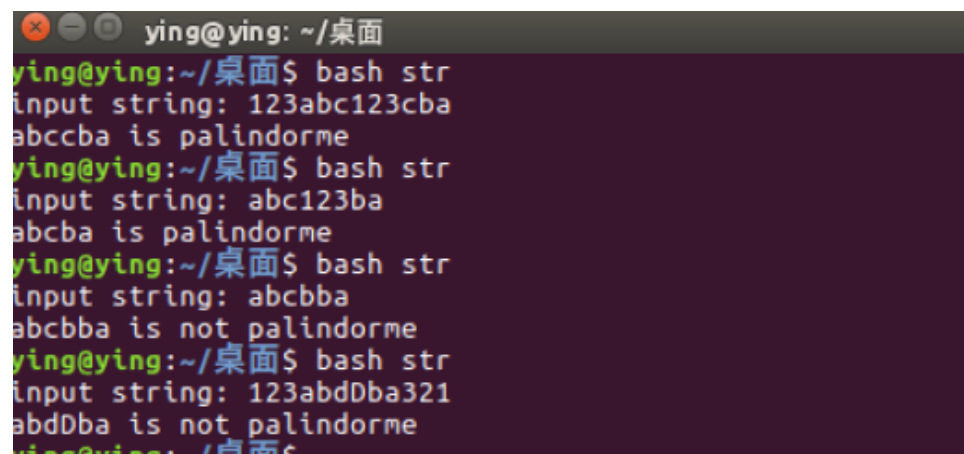
```
echo "the number of directory files are: $dcnt"
echo "the number of executable files are: $xcnt"
exit 0
```



```
ying@ying: ~/桌面
ying@ying:~/桌面$ bash count
there must be 1 argument
ying@ying:~/桌面$ bash count ./
the number of ordinary files are: 6
the number of directory files are: 2
the number of executable files are: 3
ying@ying:~/桌面$ bash count ~
the number of ordinary files are: 1978
the number of directory files are: 171
the number of executable files are: 179
ying@ying:~/桌面$ bash count /usr/include
the number of ordinary files are: 2080
the number of directory files are: 116
the number of executable files are: 116
```

3. （15 分）编写一个 shell 脚本，输入一个字符串，忽略（删除）非字母后，检测该字符串是否为回文(palindrome)。对于一个字符串，如果从前向后读和从后向前读都是同一个字符串，则称之为回文串。例如，单词“mom”，“dad”和“noon”都是回文串。

```
echo -n "input string: "
read line #读入一行数据
line=$(echo $line | tr -c -d [:alpha:]) #取出字符串的非字母字符，-c 表示反向选择，-d 表删除
reverse=$(echo $line | rev) #反转字符串
if [[ $line == $reverse ]] #判断两字符串是否相等
then
    echo "$line is palindorme" #相等
else
    echo "$line is not palindorme" #不相等
fi
exit 0
```



```
ying@ying: ~/桌面
ying@ying:~/桌面$ bash str
input string: 123abc123cba
abccba is palindorme
ying@ying:~/桌面$ bash str
input string: abc123ba
abcba is palindorme
ying@ying:~/桌面$ bash str
input string: abcbba
abcbba is not palindorme
ying@ying:~/桌面$ bash str
input string: 123abdDb321
abdDb321 is not palindorme
ying@ying:~/桌面$
```

4. （15 分）本实验目的观察使用带-f 选项的 tail 命令及学习如何使用 gcc 编译器，并观察进程运行。自己去查阅资料获取下面源程序中的函数（或系统调用）的作用。。创建一个文件名为 test.c 的 c 语言文件，内容如下：

```
#include <stdio.h>
main()
{
    int i;
    i = 0;
    sleep(10);
    while (i < 5) {
        system("date");
        sleep(5);
        i++;
    }
    while (1) {
        system("date");
        sleep(10);
    }
}
```

在 shell 提示符下，依次运行下列三个命令：

```
gcc -o generate test.c
./generate >> dataFile &
tail -f dataFile
```

- 第一个命令生成一个 c 语言的可执行文件，文件名为 generate；
- 第二个命令是每隔 5 秒和 10 秒把 date 命令的输出追加到 dataFile 文件中，这个命令为后台执行，注意后台执行的命令尾部加上&字符；
- 最后一个命令 tail -f dataFile，显示 dataFile 文件的当前内容和新追加的数据：

在输入 tail -f 命令 1 分钟左右后，按<Ctrl-C>终止 tail 程序。用 kill -9 pid 命令终止 generate 后台进程的执行。

最后用 tail dataFile 命令显示文件追加的内容。给出这些过程的你的会话。

-f 表示循环读取文件内容

```
ying@ying: ~/桌面
ying@ying:~/桌面$ gcc -o generate test.c
test.c: In function 'main':
test.c:5:5: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
    sleep(10);
    ^~~~~
test.c:7:2: warning: implicit declaration of function 'system' [-Wimplicit-function-declaration]
    system("date");
    ^~~~~
ying@ying:~/桌面$ ./generate >> dataFile &
[1] 2877

ying@ying:~/桌面$ tail -f dataFile
2019年 05月 12日 星期日 23:41:22 CST
2019年 05月 12日 星期日 23:41:27 CST
2019年 05月 12日 星期日 23:41:32 CST
2019年 05月 12日 星期日 23:41:37 CST
2019年 05月 12日 星期日 23:41:42 CST

ying@ying:~/桌面$ kill -9 2877
ying@ying:~/桌面$ job
未找到 'job' 命令, 您要输入的是否是:
命令 'jsb' 来自于包 'jsonbot' (universe)
命令 'jtb' 来自于包 'jtb' (universe)
命令 'jot' 来自于包 'athena-jot' (universe)
命令 'jdb' 来自于包 'openjdk-8-jdk-headless' (main)
命令 'jdb' 来自于包 'openjdk-9-jdk-headless' (universe)
命令 'bob' 来自于包 'python-sponge' (universe)
命令 'jo' 来自于包 'jo' (universe)
命令 'joe' 来自于包 'joe' (universe)
命令 'joe' 来自于包 'joe-jupp' (universe)
job: 未找到命令
[1]+ 已杀死                  ./generate >> dataFile
[1]+ 已杀死                  ./generate >> dataFile
```

注: pid 是执行 generate 程序的进程号; 使用 generate >> dataFile & 命令后, 屏幕打印后台进程作业号和进程号, 其中第一个字段方括号内的数字为作业号, 第二个数字为进程号; 也可以用 kill -9 %job 终止 generate 后台进程, job 为作业号。

5. (15 分) 假设下面的代码段:

```
pid_t pid;
pid = fork();
if (pid==0) { /* child process */
    fork();
    pthread_create(...);
}
fork();
```

a. 创建了多少个单独进程? 不包括主进程。

创建了 4 个单独的进程:

pid_parent

```
fork(); +1
pid_child +1
fork(); +1
fork(); +1
```

b. 创建了多少个单独线程？

3 个单独的线程，进程本身有一个线程，然后子进程 `pthread_create` 又创建了一个线程，子进程的子进程也创建了一个线程。

(完成本题的有关资料请参考教材第 6、8 章)

6. (25 分) 编写一个多线程程序，计算一组数字的多种统计值。这个程序通过命令行传递一组数字，然后创建三个单独的工作线程，第一个线程求数字的平均值，第二个线程求最大值，第三个线程求最小值。例如，假设你的程序 `myprog` 被传递了如下整数（程序命令后的参数）：

`myprog 90 81 78 95 79 72 85`

程序将会输出：

The average value is 82

The mininum value is 72

The maxinum value is 95

表示平均值、最小值、最大值的变量将会作为全局变量。工作线程将会设置这些值；当工作线程退出时，父线程将输出这些值。

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#define MAXN 1000 //允许的最大元素个数
int maxn = 0x80000000, minn = 0x7FFFFFFF; //最大值和最小值
double aven = 0; //平均值
int len = 0; //数组元素个数
void* ave(void* arg) { //计算平均值
    int i;
    int* p = (int*) arg; //强制类型转换成整型指针
    for (i=0; i<len; i++) {
        aven += p[i];
    }
    aven = 1.0 * aven / len;
}

void* max(void* arg) { //计算最大值
    int i;
    int* p = (int*) arg; //强制类型转换成整型指针
    for (i=0; i<len; i++) {
```

```

        if (p[i] > maxn) maxn = p[i];
    }
}

void* min(void* arg) { //计算最小值
    int i;
    int* p = (int*) arg; //强制类型转换成整型指针
    for (i=0; i<len; i++) {
        if (p[i] < minn) minn = p[i];
    }
}

int main(int argc, char* argv[]) {
    int arr[MAXN];
    int i, err;
    len = argc - 1; //除去命令行第一个参数（文件名参数）
    for (i=1; i<argc; i++) {
        arr[i-1] = atoi(argv[i]); //字符串转换成整数
    }
    pthread_t t1, t2, t3;
    err = pthread_create(&t1, NULL, ave, arr); //创建线程 1
    if (err != 0) { //线程创建失败
        printf("thread create error!\n");
        exit(-1);
    }
    err = pthread_create(&t2, NULL, max, arr); //创建线程 2
    if (err != 0) { //线程创建失败
        printf("thread create error!\n");
        exit(-1);
    }
    err = pthread_create(&t3, NULL, min, arr); //创建线程 3
    if (err != 0) { //线程创建失败
        printf("thread create error!\n");
        exit(-1);
    }
    pthread_join(t1, NULL); //等待子线程结束
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    printf("%d %d %.2f\n", maxn, minn, aven);
    return 0;
}

```

```
ying@ying: ~/桌面/cpro
ying@ying:~/桌面/cpro$ gcc pthread.c -o out -lpthread
ying@ying:~/桌面/cpro$ ./out 25 12 34 38 -3 7
38 -3 18.83
```

(完成本题的有关知识请参考教材第8章)

下面题目为选做题，计算机学院各专业学生尽可能完成。

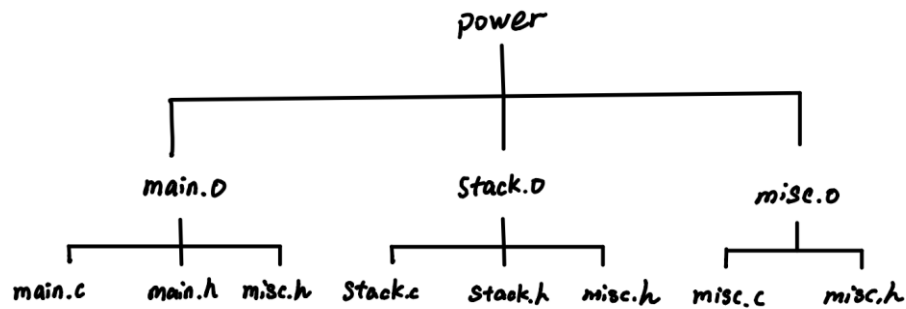
7. 在 linux 系统下的软件开发中，经常要使用 `make` 工具，要掌握 `make` 的规则。`makefile` 文件中的每一行是描述文件间依赖关系的 `make` 规则。本实验是关于 `makefile` 内容的，您不需要在计算机上进行编程运行，只要书面回答下面这些问题。

对于下面的 `makefile`:

```
CC = gcc
OPTIONS = -O3 -o
OBJECTS = main.o stack.o misc.o
SOURCES = main.c stack.c misc.c
HEADERS = main.h stack.h misc.h
power: main.c $(OBJECTS)
    $(CC) $(OPTIONS) power $(OBJECTS) -lm
main.o: main.c main.h misc.h
stack.o: stack.c stack.h misc.h
misc.o: misc.c misc.h
```

回答下列问题

- 所有宏定义的名字
CC OPTIONS OBJECTS SOURCES HEADERS
- 所有目标文件的名字
main.o stack.o misc.o
- 每个目标的依赖文件
main.o: main.c main.h misc.h
stack.o: stack.c stack.h misc.h
misc.o: misc.c misc.h
- 生成每个目标文件所需执行的命令
stack.o: gcc -c stack.c -o stack.o
misc.o: gcc -c misc.c -o misc.o
main.o: gcc -c main.c -o main.o
- 画出 `makefile` 对应的依赖关系树。



f. 生成 main.o stack.o 和 misc.o 时会执行哪些命令，为什么？

stack.o: gcc -c stack.c -o stack.o

misc.o: gcc -c misc.c -o misc.o

main.o: gcc -c main.c -o main.o

gcc main.o stack.o misc.o -o power -lm

因为要将 .c 文件转换成 .o 文件，并将 .o 文件链接成可执行文件

8. 用编辑器创建 main.c, compute.c, input.c, compute.h, input.h 和 main.h 文件。

下面是它们的内容。注意 compute.h 和 input.h 文件仅包含了 compute 和 input 函数的声明但没有定义。定义部分是在 compute.c 和 input.c 文件中。

main.c 包含的是两条显示给用户的提示信息。

```

$ cat compute.h
/* compute 函数的声明原形 */
double compute(double, double);
$ cat input.h
/* input 函数的声明原形 */
double input(char *);
$ cat main.h
/* 声明用户提示 */
#define PROMPT1 "请输入 x 的值: "
#define PROMPT2 "请输入 y 的值: "
$ cat compute.c
#include <math.h>
#include <stdio.h>
#include "compute.h"
double compute(double x, double y)
{
    return (pow ((double)x, (double)y));
}
$ cat input.c
#include <stdio.h>
#include "input.h"
double input(char *s)
{
    float x;

```

```

    printf("%s", s);
    scanf("%f", &x);
    return (x);
}
$ cat main.c
#include <stdio.h>
#include "main.h"
#include "compute.h"
#include "input.h"

main()
{
    double x, y;
    printf("本程序从标准输入获取 x 和 y 的值并显示 x 的 y 次方.\n");
    x = input(PROMPT1);
    y = input(PROMPT2);
    printf("x 的 y 次方是:%6.3f\n", compute(x, y));
}
$

```

为了得到可执行文件 `power`，我们必须首先从三个源文件编译得到目标文件，并把它们连接在一起。下面的命令将完成这一任务。注意，在生成可执行代码时不要忘了连接上数学库。

```

$ gcc -c main.c input.c compute.c
$ gcc main.o input.o compute.o -o power -lm
$

```

相应的 Makefile 文件是：

```

$ cat Makefile
power: main.o input.o compute.o
    gcc main.o input.o compute.o -o power -lm

main.o: main.c main.h input.h compute.h
    gcc -c main.c

input.o: input.c input.h
    gcc -c input.c

compute.o: compute.c compute.h
    gcc -c compute.c
$

```

(1)、创建上述三个源文件和相应头文件，用 gcc 编译器，生成 power 可执行文件，并运行 power 程序。给出完成上述工作的步骤和程序运行结果。

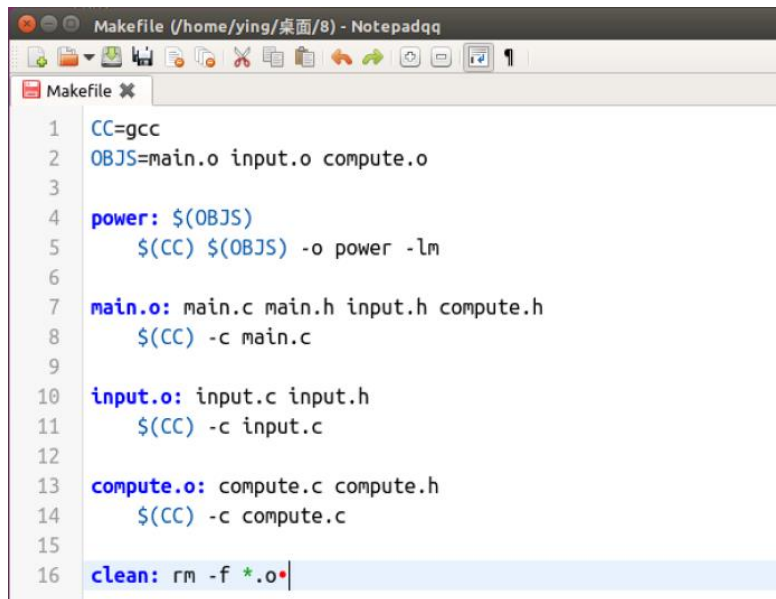
```
ying@ying:~/桌面/8$ gcc -c main.c input.c compute.c
ying@ying:~/桌面/8$ gcc main.o input.o compute.o -o power -lm
ying@ying:~/桌面/8$ ./power
本程序从标准输入获取x和y的值并显示x的y次方。
请输入x的值: 2
请输入y的值: 3
x的y次方是: 8.000
```

(2)、创建 Makefile 文件，使用 make 命令，生成 power 可执行文件，并运行 power 程序。给出完成上述工作的步骤和程序运行结果。

```
Makefile (U/home/ying/桌面/8) - Notepadqq
Makefile ❸
1  power: main.o input.o compute.o
2      gcc main.o input.o compute.o -o power -lm
3
4  main.o: main.c main.h input.h compute.h
5      gcc -c main.c
6
7  input.o: input.c input.h
8      gcc -c input.c
9
10 compute.o: compute.c compute.h
11      gcc -c compute.c
12
13 clean:
14      rm -f *.o
```

```
ying@ying: ~/桌面/8
ying@ying:~/桌面/8$ make
gcc -c input.c
gcc -c compute.c
gcc main.o input.o compute.o -o power -lm
ying@ying:~/桌面/8$ ./power
本程序从标准输入获取x和y的值并显示x的y次方。
请输入x的值: 2.6
请输入y的值: 0.3
x的y次方是: 1.332
```

也可以用变量（宏）定义的方式实现：



```
1 CC=gcc
2 OBJS=main.o input.o compute.o
3
4 power: $(OBJS)
5     $(CC) $(OBJS) -o power -lm
6
7 main.o: main.c main.h input.h compute.h
8     $(CC) -c main.c
9
10 input.o: input.c input.h
11     $(CC) -c input.c
12
13 compute.o: compute.c compute.h
14     $(CC) -c compute.c
15
16 clean: rm -f *.o
```

9. 用 C 语言写一个名字为 myls 程序，实现类似 Linux 的 ls 命令，其中 myls 命令必须实现 -a、-l、-i 等选项的功能。要求 myls 程序使用系统调用函数编写，不能使用 exec 系统调用或 system() 函数等调用 ls 命令来实现。命令 man ls 可以得到更多 ls 选项的含义。

(完成本题的有关知识请参考教材第 5 章)

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <time.h>
#include <pwd.h>
#include <grp.h>
#define MAXN 100
void print(struct stat *st);

int main(int argc, char* argv) {
    DIR *dir;
    struct stat buf;
    struct dirent *dp;
    dir = opendir("./"); //打开当前目录并建立目录流
    while ((dp = readdir(dir)) != NULL) {
        printf("%6s ", dp->d_name);
        lstat(dp->d_name, &buf);
        print(&buf);
    }
}
```

```

    return 0;
}

void print(struct stat *st) {
    printf("%ld ", (long)st->st_ino); //inode

    switch (st->st_mode & S_IFMT) {
        case S_IFBLK: printf("b"); break; //块设备文件
        case S_IFCHR: printf("c"); break; //字符设备文件
        case S_IFDIR: printf("d"); break; //目录文件
        case S_IFIFO: printf("p"); break; //管道文件
        case S_IFLNK: printf("l"); break; //符号链接文件
        case S_IFSOCK: printf("s\n"); break; //socket
        default: printf("-"); break; //普通文件
    }

    if (st->st_mode & S_IRUSR) printf("r"); //文件所有者具可读取权限
    else printf("-");
    if (st->st_mode & S_IWUSR) printf("w"); //文件所有者具可写入权限
    else printf("-");
    if (st->st_mode & S_IXUSR) printf("x"); //文件所有者具可执行权限
    else printf("-");

    if (st->st_mode & S_IRGRP) printf("r"); //用户组具可读取权限
    else printf("-");
    if (st->st_mode & S_IWGRP) printf("w"); //用户组具可写入权限
    else printf("-");
    if (st->st_mode & S_IXGRP) printf("x"); //用户组具可执行权限
    else printf("-");

    if (st->st_mode & S_IROTH) printf("r"); //用户组具可读取权限
    else printf("-");
    if (st->st_mode & S_IWOTH) printf("w"); //用户组具可写入权限
    else printf("-");
    if (st->st_mode & S_IXOTH) printf("x"); //用户组具可执行权限
    else printf("-");

    printf(" %3ld ", (long)st->st_nlink); //链接数

    struct passwd *pwd = getpwuid(st->st_uid); //用户名
    printf(" %6s ", pwd->pw_name);

    struct group *grp = getgrgid(st->st_gid); //组名
    printf(" %6s ", grp->gr_name);
}

```

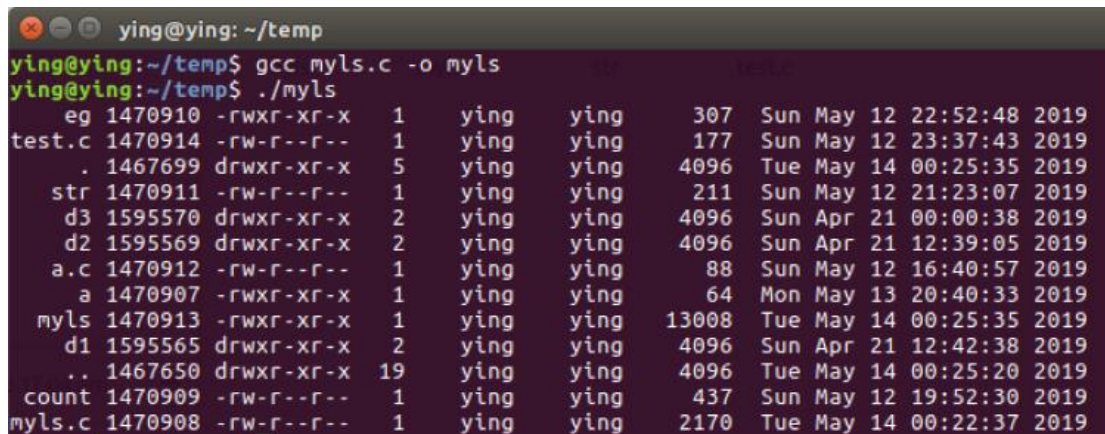
```

printf(" %6ld ", (long)st->st_size); //大小

printf(" %s", ctime(&st->st_mtime)); //最后修改时间

}

```



```

ying@ying: ~/temp
ying@ying:~/temp$ gcc myls.c -o myls
ying@ying:~/temp$ ./mysls
eg 1470910 -rwxr-xr-x 1 ying ying 307 Sun May 12 22:52:48 2019
test.c 1470914 -rw-r--r-- 1 ying ying 177 Sun May 12 23:37:43 2019
. 1467699 drwxr-xr-x 5 ying ying 4096 Tue May 14 00:25:35 2019
str 1470911 -rw-r--r-- 1 ying ying 211 Sun May 12 21:23:07 2019
d3 1595570 drwxr-xr-x 2 ying ying 4096 Sun Apr 21 00:00:38 2019
d2 1595569 drwxr-xr-x 2 ying ying 4096 Sun Apr 21 12:39:05 2019
a.c 1470912 -rw-r--r-- 1 ying ying 88 Sun May 12 16:40:57 2019
a 1470907 -rwxr-xr-x 1 ying ying 64 Mon May 13 20:40:33 2019
mysls 1470913 -rwxr-xr-x 1 ying ying 13008 Tue May 14 00:25:35 2019
d1 1595565 drwxr-xr-x 2 ying ying 4096 Sun Apr 21 12:42:38 2019
.. 1467650 drwxr-xr-x 19 ying ying 4096 Tue May 14 00:25:20 2019
count 1470909 -rw-r--r-- 1 ying ying 437 Sun May 12 19:52:30 2019
mysls.c 1470908 -rw-r--r-- 1 ying ying 2170 Tue May 14 00:22:37 2019

```

三、 讨论、心得（必填）（10 分）

① 在练习 2 中“编写 shell 程序，统计指定目录下的普通文件、子目录及可执行文件的数目，目录的路径名字由参数传入”，统计目录下的普通文件、子目录的方式还可以通过 find 命令来查找，具体命令如下：

```

echo `find $1 -type f | wc -l`
echo `find $1 -type d | wc -l`
echo `find $1 -type f -executable | wc -l`

```

② 很多时候会搞不清楚命令中需不需要空格，在对一些命令进行测试后，现在对空格的情况做一个总结：

- 在实验 2 的实验报告中，已经指出“重定向时，0 为 stdin，1 为 stdout，2 为错误信息，写重定向的命令时，一定要注意要写成 1> filename 而不是 1 > filename，也就是 1 与>之间不能够有空格否则会出错。”。在实验 3 中再次加以复习。
- declare 的赋值前后不能有空格。declare -x age=20。
- shell 编程中，赋值语句前后不能够有空格，如 a=3 不能写成 a = 3，也不能写成\$a=3。同时如果要将在命令的执行结果赋值给变量时，需要用以下两者之一的方式：res=\$(pwd)或 res=`pwd`。
- shell 编程中，使用[] 或 [[]] 包裹的表达式中，在操作数和操作符或者括号的前后都要至少留一个空格比如[\$# -ne 1]或[3 -eq 5]，而[3 -eq 5]则会报错找不到命令。[]与 test 的功能是相同的，双方括号[[expr]]命令中的表达式 expr 可以使用标准的字符串比较，也能够使用正则表达式。
- 双括号命令允许在比较过程中使用高级表达式，形式为((expr))，如((\$val1 ** 2 > 90))，但可以不加空格如((b =2**2))
- let 语句中，若表达式有空格，则要使用引号，如：
let "a = 8" "b = 13"
let c=a+b

- `expr` 命令运算符两边都需要保留空格,如 `expr 1 + 2` ,如果没保留空格如 `expr 1+2` 则会输出 `1+2`

④ `gcc -c test.c` 将生成 `test.o` 的目标文件, `gcc -o target test.c` 将生成可执行文件 `target`, `gcc -c a.c -o a.o` 与 `gcc -c a.c` 等价, 而 `gcc -o out a.o` 将直接生成可执行文件 `out`。 `gcc -c` 可以跟多个文件, 如 `gcc -c main.c input.c compute.c`
`gcc -o` 有两种写法:

- `gcc main.o input.o compute.o -o power -lm`
- `gcc -o power main.o input.o compute.o -lm`

`#include<pthread.h>`时需要加`-lpthread`

⑤心得: 在本次实验中, 由于上课讲得还是比较快, 很多细节还是没有把握, 通过做实验以及自己动手尝试, 能够补上很多的细节比如空格的问题已经整个程序从编译到链接的流程等等。在这里将做题时遇到的问题总结起来, 可以方便考试前的复习。