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：黄炯睿　　专业：信息安全　　学号：3170103455

电子邮件地址：1172180735@qq.com

实验日期：2019 年 6 月 1 日

一、 实验环境

计算机配置： 处理器 intel® Core™ i7-7700HQ CPU @ 2.8GHz(8 CPUs)

内　存　　8G

显　卡　　Nvidia GeForce GTX 1050（6 GB）

操作系统环境：Windows 10 家庭中文版 64 位 版本： 17134.706

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。

```
#!/bin/bash
if test  $# -ne 1   #if the number of input parameter isn't equal to 1
then
    echo "please input exactly one parameter!"
    exit 1
fi

if test -f "$1"         #check the first parameter
then
    filename="$1"       #if the file is oridinary file
    set -- $(ls -l $filename --full-time)   #find the info about the file
    echo "the filename is $filename"    #print the filename
    echo "the owner is ${3}."       #print the file's owner
```

```
    echo "the last edit time is ${6} $7 ${8}." #print the time
    exit 0
fi
#error when the file is not oridinary.
echo "$filename isn't an ordinary file!"
exit 0
```

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./1.sh 1.txt
the filename is 1.txt
the owner is huangjiongrui.
the last edit time is 2019-06-02 16:57:45.486045790 +0800.
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./1.sh 1.sh
the filename is 1.sh
the owner is huangjiongrui.
the last edit time is 2019-06-02 17:43:57.250035217 +0800.
```

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 [ $# -ne 1 ] || !( test -e $1 ) || !( test -d $1 )
#check the number of parameter && if it's existing && if it's a directory
then
    echo "Please input exactly one existed directory name!"
    exit 1
fi

common=$(find $1 -type f|wc -l)             #查找输入文件夹中格式为普通文件 | 利用 wc 输出文件的个数
echo "common files: $common"
sub=$(find $1 -type d|wc -l)                #目录文件
echo "sub directory: $((sub-1))"
exe=$(find $1 -executable -type f|wc -l)    #可执行文件
echo "executable files: $exe"
```

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./2.sh 1.txt
Please input exactly one existed directory name!
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./2.sh ../class_of_linux/
common files: 6
sub directory: 0
executable files: 4
```

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 "Please input the string:"
read line #read a string
#使用 tr -c 选择非字母字符，使用-d 选项删除选中的字符。
str=$(echo $line| tr -c -d [:alpha:] )
#反转字符串
reverse=$(echo $line | rev)
```

```
if [[ $str == $reverse ]]  #如果字符串正着读反着读相等
then
    echo "$str is palindorme."
else
    echo "$str isn't palindorme."
fi
```

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。

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。

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ touch test.chuangjiongrui
@huangjiongrui-virtual-machine:~/class_of_linux$ vi test.c
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ gcc -o generate test.c
test.c:2:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main()
 ^
test.c: In function 'main':
test.c:6:5: warning: implicit declaration of function 'sleep' [-Wimplicit-function-decl
aration]
     sleep(10);
     ^
test.c:8:9: warning: implicit declaration of function 'system' [-Wimplicit-function-dec
laration]
         system("date");
         ^
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./generate >> dataFile &
[1] 17879
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ tail -f dataFile
2019年 06月 02日 星期日 20:30:39 CST
2019年 06月 02日 星期日 20:30:44 CST
2019年 06月 02日 星期日 20:30:49 CST
2019年 06月 02日 星期日 20:30:54 CST
2019年 06月 02日 星期日 20:30:59 CST
2019年 06月 02日 星期日 20:31:04 CST
2019年 06月 02日 星期日 20:31:14 CST
2019年 06月 02日 星期日 20:31:24 CST
2019年 06月 02日 星期日 20:31:34 CST
```

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ps
   PID TTY          TIME CMD
 16820 pts/0    00:00:00 bash
 17879 pts/0    00:00:00 generate
 17944 pts/0    00:00:00 ps
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ kill -9 17879
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ tail dataFile
2019年 06月 02日 星期日 20:31:54 CST
2019年 06月 02日 星期日 20:32:04 CST
2019年 06月 02日 星期日 20:32:14 CST
2019年 06月 02日 星期日 20:32:24 CST
2019年 06月 02日 星期日 20:32:34 CST
2019年 06月 02日 星期日 20:32:44 CST
2019年 06月 02日 星期日 20:32:54 CST
2019年 06月 02日 星期日 20:33:04 CST
2019年 06月 02日 星期日 20:33:14 CST
2019年 06月 02日 星期日 20:33:24 CST
[1]+  已杀死               ./generate >> dataFile
```

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。

除去主进程的话一共创建了 5 个进程。第一个进程返回的子进程又 fork 了一次，但父进程没有，所以一共有 6 个进程，减掉子进程后则为 5 个。

b. 创建了多少个单独线程？

三个单独的线程，其中一个是主线程，在 if 语句中有两个进程执行了 pthread_create 函数，所以这里创建了两个线程。

https://www.cnblogs.com/bastard/archive/2012/08/31/2664896.html

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。

（完成本题的有关知识请参考教材第8章）

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
int min=0x7fffffff,max=0x80000000;
double average=0;
int len;//输入参数的个数
void* get_average(void* arg){
   int i ;
   int *p = (int *)arg;//将 void*强制转化为 int* 在之后 i++中寻址更方便
   for(i=0;i<len;i++){
       average += p[i];
  }
   average /= 1.0*len;
}

void* get_min(void* arg){
   int i;
   int *p = (int *)arg;
   for(i=0;i<len;i++){
       if(min > p[i])
           min = p[i];
  }
}
```

```
void* get_max(void* arg){
   int i;
   int *p = (int *)arg;
   for(i=0;i<len;i++){
       if(max < p[i])
           max = p[i];
  }
}

int main (int argc, char* argv[]){
   pthread_t tid1,tid2,tid3;
   if (argc <= 1){
       printf("Please input some parameter!\n");
       return 1;
  }
   len = argc - 1; //除去命令行第一个参数（文件名参数）
   int *para = (int*)malloc(sizeof(int) * len);//申请恰好为 len 个的地址
   int i;
   for(i=0;i<len;i++){
       para[i] = atoi(argv[i+1]);//将输入参数的字符串类型转化为数字类型
  }
   /*创建线程*/
   pthread_create(&tid1,NULL,get_average,(void*)para);
   pthread_create(&tid2,NULL,get_min,(void*)para);
   pthread_create(&tid3,NULL,get_max,(void*)para);
   /*等待线程完成*/
   pthread_join(tid1,NULL);
   pthread_join(tid2,NULL);
   pthread_join(tid3,NULL);
   printf("Average:%f\nMin:%d\nMax:%d\n",average,min,max);
   return 0;
}
```

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./myprog
Please input some parameter!
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./myprog 123 212 3
Average:112.666667
Min:3
Max:212
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$
```

另外写了一个.sh 的脚本，完成了类似的功能：

```
read -p "Please input some numbers:" -a array
itemcount=${#array[@]}
max=$((array[0]))
min=$((array[0]))
sum=$((array[0]))
```

```
#-----------sum---------------
{
for (( i = 1; i < $itemcount; i++ )); do

    array[i]=${array[i]}
    sum=$(( sum+array[i] ))
done
echo "average is $((sum/itemcount))"
} &

#-----------min---------------
{
for (( i = 1; i < $itemcount; i++ )); do
    array[i]=${array[i]}
    if [[ array[i] -lt min ]]; then
        min=$((array[i]))
    fi
done
echo "min item is: $min"
} &

#-----------max---------------
{
for (( i = 1; i < $itemcount; i++ )); do
    array[i]=${array[i]}
    if [[ array[i] -gt max ]]; then
        max=$((array[i]))
    fi
done
echo "max item is: $max"
} &

wait
```

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。makfie 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 $(OPJECTS)
    $(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

a. 所有宏定义的名字

   CC OPTIONS OBJECTS SOURCES HEADERS

b. 所有目标文件的名字

   main.o stack.o misc.o

c. 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

d. 生成每个目标文件所需执行的命令

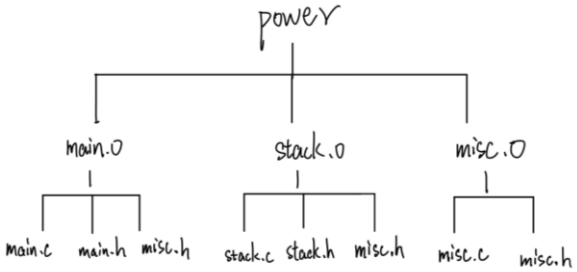
   stack.o: gcc -c stack.c -o stack.o

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

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

   power: gcc -O3 -o power main.o stack.o misc.o -lm

e. 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

   会执行以下命令：

   1. 预处理

      将.c 文件中所有的库文件和宏定义展开

   2. 编译

      将文件优化后生成汇编语言代码

3. 汇编
   将汇编语言代码转化为对应的可执行机器指令

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。

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux/8$ gcc -c main.c input
.c compute.c
main.c:6:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux/8$ gcc main.o input.o
compute.o -o power -lm
```

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux/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: 12
请输入y的值: 2
x的y次方是:144.000
```

（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。

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux/8$ make
gcc -c main.c
main.c:6:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
gcc main.o input.o compute.o -o power -lm
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux/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: 23
请输入y的值: 10
x的y次方是:41426511213649.000
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux/8$
```

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)

```c
#include <sys/stat.h>
#include <sys/types.h>
#include <stdio.h>
#include <string.h>
#include <dirent.h>
#include <time.h>
#include <pwd.h>
#include <grp.h>
void print(struct stat *st){
    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"); //所有者所在的用户组具可读取权限
    else printf("-");
    if (st->st_mode & S_IWGRP) printf("w"); //所有者所在的用户组具可写入权限
    else printf("-");
```

```
    if (st->st_mode & S_IXGRP) printf("x"); //所有者所在的用户组具可执行权限
    else printf("-");

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

}
int main(int argc, char* argv[]){
    DIR *dir;
    struct dirent *item;
    struct stat buf;
    int i,count=0;
    dir = opendir("./");//打开当前目录并建立一个目录流
    if(strcmp(argv[1],"ls")!= 0) {//如果输入参数不是 ls
        printf("cannot recognize the command!\n");
        return 1;
    }
    //no option
    if(argc == 2){ // 如果输入参数仅有一个 ls
        item = readdir(dir);count++;
        while(item != NULL){
            //去除文件.和..
            if(strcmp(item->d_name,".")==0 || strcmp(item->d_name,"..")==0 ){
                item = readdir(dir);
                continue;
            }
            //左对齐占用 10 格的形式输出文件名
            printf("%-10s ",item->d_name);count++;
```

```
            if(count%5 == 1) printf("\n");//一行输出 5 个
            item = readdir(dir); //获取下一个文件
        }
        printf("\n");
    }
    // with option
    if(argc == 3){
    // -a 与无参数 ls 类似，但是要输出.和..文件
        if(strcmp(argv[2],"-a") == 0){
            item = readdir(dir);count++;
            while(item != NULL){
                printf("%-10s ",item->d_name);count++;
                if(count%5 == 1) printf("\n");
                item = readdir(dir); //获取下一个文件
            }
            printf("\n");
        }
    // -i
        if(strcmp(argv[2],"-i") == 0){
            item = readdir(dir);count++;
            while(item != NULL){
                if(strcmp(item->d_name,".")==0 ||
strcmp(item->d_name,"..")==0 ){
                    item = readdir(dir);
                    continue;
                }
                 //输出 inode 以及文件名
                printf("%ld %-10s ",item->d_ino,item->d_name);
                count++;
                if(count%4 == 1) printf("\n");
                item = readdir(dir); //获取下一个文件
            }
            printf("\n");
        }
    // -l
        if(strcmp(argv[2],"-l") == 0){
            item = readdir(dir);
            while(item != NULL){
                if(strcmp(item->d_name,".")==0 ||
strcmp(item->d_name,"..")==0 ){
                    item = readdir(dir);//读取下一个，不进行 print
                    continue;
                }
                lstat(item->d_name,&buf);//通过文件名查找文件的信息
```

```
                printf(" %-10s\n",item->d_name);//先打印文件名
                print(&buf);//调用自定义的打印函数，依次打印文件的信息
                item = readdir(dir);//获取下一个文件
            }
        }
    }
}
```

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./myls ls
myprog.c   8          myls.c     dataFile   myprog.sh
3.sh       2.sh       generate   1.sh       myls
myprog     test.c
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./myls ls -i
675315 myprog.c   678850 8          679188 myls.c     679158 dataFile
679162 myprog.sh  679079 3.sh       679157 2.sh       679152 generate
679141 1.sh       679168 myls       660075 myprog     679142 test.c

huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./myls ls -a
.          myprog.c   8          myls.c     ..
dataFile   myprog.sh  3.sh       2.sh       generate
1.sh       myls       myprog     test.c
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$
```

myls ls -l 和 ls -l 的对比图：

```
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ./myls ls -l
 myprog.c
-rw-r--r--   1  huangjiongrui  huangjiongrui    1200  Tue Jun  4 16:11:28 2019
 8
drwxrwxr-x   2  huangjiongrui  huangjiongrui    4096  Wed Jun  5 00:58:27 2019
 myls.c
-rw-r--r--   1  huangjiongrui  huangjiongrui    3302  Wed Jun  5 10:19:07 2019
 dataFile
-rw-r--r--   1  huangjiongrui  huangjiongrui     860  Sun Jun  2 20:33:24 2019
 myprog.sh
-rwxr-xr-x   1  huangjiongrui  huangjiongrui     718  Mon Jun  3 14:31:50 2019
 3.sh
-rwxr-xr-x   1  huangjiongrui  huangjiongrui     228  Sun Jun  2 20:27:03 2019
 2.sh
-rwxr-xr-x   1  huangjiongrui  huangjiongrui     377  Sun Jun  2 19:47:20 2019
 generate
-rwxr-xr-x   1  huangjiongrui  huangjiongrui    8512  Sun Jun  2 20:30:13 2019
 1.sh
-rwxr-xr-x   1  huangjiongrui  huangjiongrui     576  Sun Jun  2 17:43:57 2019
 myls
-rwxr-xr-x   1  huangjiongrui  huangjiongrui   13176  Wed Jun  5 10:19:12 2019
 myprog
-rwxr-xr-x   1  huangjiongrui  huangjiongrui    9000  Tue Jun  4 16:11:31 2019
 test.c
-rw-r--r--   1  huangjiongrui  huangjiongrui     214  Sun Jun  2 20:28:51 2019
huangjiongrui@huangjiongrui-virtual-machine:~/class_of_linux$ ls -l
总用量 76
-rwxr-xr-x 1 huangjiongrui huangjiongrui   576 6月    2 17:43 1.sh
-rwxr-xr-x 1 huangjiongrui huangjiongrui   377 6月    2 19:47 2.sh
-rwxr-xr-x 1 huangjiongrui huangjiongrui   228 6月    2 20:27 3.sh
drwxrwxr-x 2 huangjiongrui huangjiongrui  4096 6月    5 00:58 8
-rw-r--r-- 1 huangjiongrui huangjiongrui   860 6月    2 20:33 dataFile
-rwxr-xr-x 1 huangjiongrui huangjiongrui  8512 6月    2 20:30 generate
-rwxr-xr-x 1 huangjiongrui huangjiongrui 13176 6月    5 10:19 myls
-rw-r--r-- 1 huangjiongrui huangjiongrui  3302 6月    5 10:19 myls.c
-rwxr-xr-x 1 huangjiongrui huangjiongrui  9000 6月    4 16:11 myprog
-rw-r--r-- 1 huangjiongrui huangjiongrui  1200 6月    4 16:11 myprog.c
-rwxr-xr-x 1 huangjiongrui huangjiongrui   718 6月    3 14:31 myprog.sh
-rw-r--r-- 1 huangjiongrui huangjiongrui   214 6月    2 20:28 test.c
```

三、　讨论、心得（必填）（10 分）

写代码毕竟还是你们实验型的学科，在课上听老师讲课觉得这些知识点与注意事项不过如此，实际上写的时候疯狂查资料，翻 ppt。这一份作业很好地帮助我巩固了许多命令的用法以及 shell 编程和线程控制等知识点。

在实验过程中遇到的最大问题是 shell 编程中的空格和括号问题，在这里做一个总结：

1. 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`
2. shell 编程中，使用[]或者[[]]包裹的表达式中，在括号的旁边需要留出空格，像表达式[3 -eq 5]会出现无法解析的错误。[[是 bash 程序语言的关键字。并不是一个命令，[[ ]] 结构比[ ]结构更加通用。在[[和]]之间所有的字符都不会发生文件名扩展或者单词分割，但是会发生参数扩展和命令替换。
3. 更为详细的总结通过在网上查找资料，找到了一篇优质的博客 https://blog.csdn.net/taiyang1987912/article/details/39551385

在使用 Makefile 编译的题目上，以前只知道这么打命令就可以编译链接，没有仔细地去看过它具体的实现过程。这次作业让我对于整个过程有了更好的理解，同时对于以前写程序时出现的动态库丢失错误有了更好的理解。

以下是查询的资料 https://blog.csdn.net/baohuan_love/article/details/14497477