

浙江大学实验报告

课程名称: Linux 程序设计 (课程综合实践 II) 实验类型: 设计型

实验项目名称: Shell 程序设计

学生姓名: 王子腾 专业: 软件工程 学号: 3180102173

电子邮件地址: 3180102173@zju.edu.cn 手机 (可选): 19975276477

实验日期: 2020 年 8 月 18 日

一、实验环境

计算机配置:

处理器	Intel Core i7-8550 CPU @ 1.80GHz 1.99Ghz
内存	8GB DDR4 2400MHz
硬盘	128GB PCIe 固态硬盘+1TB 硬盘
显卡	NVIDIA GeForce 150MX

操作系统环境:

Windows 10 家庭中文版

Linux 版本:

Ubuntu 18.04

二、实验内容和结果及分析

1. (10 分) 编写 shell 脚本, 统计指定目录下的普通文件、子目录及可执行文件的数目, 统计该目录下所有普通文件字节数总和, 目录的路径名字由参数传入。

```
1. #程序名: 1.sh
2. #作者: 王子腾 3180102173
3. #创建日期: 2020-07-19
4. #最后修改: 2020-07-20
5. #描述:
6. # 单路径目录参数:
7. # 统计指定目录下的普通文件、子目录及可执行文件的数目
8. # 统计该目录下所有普通文件字节数总和
9. # 多于一个参数: 打印错误信息
10. # 非路径目录参数: 打印错误信息
```

```

11.
12. #! /bin/bash
13. if test $# -ne 1 # 非单参数
14. then
15.     echo "请输入单路径参数" # 错误信息
16.     exit 1 # Error
17. fi
18. f_cnt=0 # 普通文件计数器
19. d_cnt=0 # 子目录计数器
20. x_cnt=0 # 可执行文件计数器
21. b_cnt=0 # 字节计数器
22. array=$(find $1 -maxdepth 1) # 找出参数路径内的所有文件
23. for i in $array # 遍历 $array
24. do
25.     if [ -f $i ] # 普通文件
26.     then
27.         ((f_cnt++))
28.         set `wc -c $i` # 获取字节数
29.         ((b_cnt+=${1}))
30.     fi
31.     if [ -d $i ] # 目录文件
32.     then
33.         ((d_cnt++))
34.     fi
35.     if [ -x $i ] # 可执行文件
36.     then
37.         ((x_cnt++))
38.     fi
39. done
40. echo "计数结果:" # 标题行
41. echo "-f -d -x bytes"
42. echo "$f_cnt $d_cnt $x_cnt $b_cnt" # 结果
43. exit 0

```

程序首先检测参数个数，若非单参数则报错并退出，之后设置各类型文件计数器，并开始循环遍历路径下所有文件，根据文件类型进行分别计数，并遍历结束后输出结果。

```

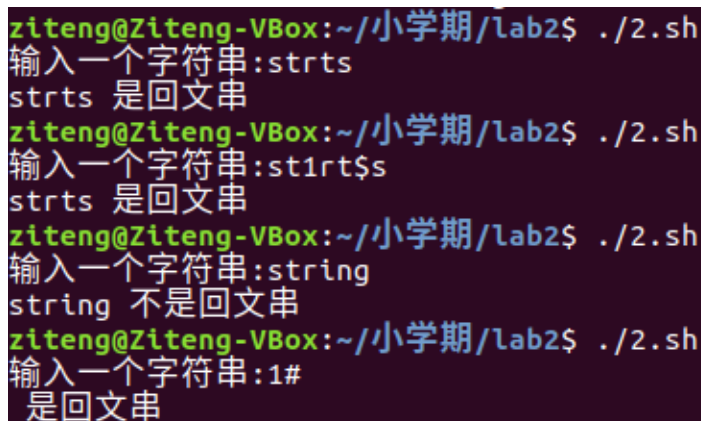
ziteng@Ziteng-VBox:~/小学期/lab2$ ./1.sh
请输入单路径参数
ziteng@Ziteng-VBox:~/小学期/lab2$ ./1.sh ~/exam
计数结果:
-f      -d      -x      bytes
8       1       5       16218
ziteng@Ziteng-VBox:~/小学期/lab2$ ls ~/exam
1.sh  2.sh  3.sh  a.out largeFile mediumFile smallFile test.c

```

2. （10 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。

```
1. #文件名: 2.sh
2. #作者: 王子腾 3180102173
3. #创建日期: 2020-07-20
4. #最后修改: 2020-07-21
5. #描述:
6. # 输入一个字符串参数:
7. # 忽略非字母字符, 检测是否回文
8.
9. #!/bin/bash
10. echo -n "输入 1 个字符串:"
11. read line # 读入一个参数, 存储在$line
12. line=`echo $line | tr -cd [:alpha:]\` # 删除非字母字符
13. rline=`echo $line | rev` # 反转字符串
14. if [[ $line == $rline ]] # 若正反序列相同
15. then
16.     echo "$line 是回文串" # 回文
17. else
18.     echo "$line 不是回文串" # 非回文
19. fi
20. exit 0
```

首先读入一个字符串并存储至\$line 变量，之后对字符串去除非字母字符，并将其反转的结果存在\$rline 中，将两者比较，若相等则为回文，否则非回文。



```
ziteng@Ziteng-VBox:~/小学期/lab2$ ./2.sh
输入一个字符串:strts
strts 是回文串
ziteng@Ziteng-VBox:~/小学期/lab2$ ./2.sh
输入一个字符串:st1rt$s
strts 是回文串
ziteng@Ziteng-VBox:~/小学期/lab2$ ./2.sh
输入一个字符串:string
string 不是回文串
ziteng@Ziteng-VBox:~/小学期/lab2$ ./2.sh
输入一个字符串:1#
是回文串
```

3. （15 分）编写一个实现文件备份和同步的 shell 脚本程序 dirsinc。程序的参数是两个需要备份同步的目录，如：

`dirsinc ~\dir1 ~\dir2` # ~\dir1 为源目录，~\dir2 为目标目录

dirsinc 程序实现两个目录内的所有文件和子目录（递归所有的子目录）内容保持一致。程序基本功能如下。

- 1) 备份功能：目标目录将使用来自源目录的最新文件，新文件和新子目录进行升级，源目录将保持不变。dircsync 程序能够实现增量备份。
- 2) 同步功能：两个方向上的旧文件都将被最新文件替换，新文件都将被双向复制。源目录被删除的文件和子目录，目标目录也要对应删除。
- 3) 其它功能自行添加设计。

```
1. 文件名: dircsync.sh
2. #作者: 王子腾 3180102173
3. #创建日期: 2020-07-19
4. #最后修改: 2020-07-20
5. #描述:
6. # 1: 备份 -- 更新且保持增量备份
7. # 2: 同步 -- 双向更新, 对应源目录删除, 保持双向最新
8.
9. #! /bin/bash
10. #备份
11. Sync1(){
12. #局部变量
13. local cur1 cur2 status1 list1 list2 i j
14. cur1=$1 # 源目录
15. cur2=$2 # 目标目录
16. status1=0 # 未找到
17. #若目标目录不存在, 则创建目录
18. if [ ! -d $cur2 ]
19. then
20. mkdir $cur2
21. fi
22. list1=$(ls $cur1) # $1 的全部文件
23. list2=$(ls $cur2) # $2 的全部文件
24. #遍历 list1, 即源目录所有文件
25. for i in $list1
26. do
27. #遍历 list2, 即目标目录所有文件
28. for j in $list2
29. do
30. if [[ $i != $j ]] # 不同文件
31. then
32. continue;
33. elif [ -d ${cur1}/${i} ] && [ -d ${cur2}/${j} ] # 同名且同为目录文件
34. then
```

```
35.     status1=1 # 已找到
36.     #echo -n "digi1:测试点"
37.     Sync1 ${cur1}/${i} ${cur2}/${j}
38.     elif [ -f ${cur1}/${i} ] && [ -f ${cur2}/${j} ] # 同名且同为普通
        文件
39.     then
40.         status1=1 # 已找到
41.         #echo "here1 测试点"
42.         if [[ `find ${cur1}/${i} -
            newer ${cur2}/${j}` == ${cur1}/${i} ]] # $i 更新
43.         then
44.             cp -rf ${cur1}/${i} ${cur2}/${j} # 更新$j 为$i
45.         fi
46.     fi
47. done
48.
49. #增量备份
50. if [[ $status1 == 0 ]] # 在$list2 中未找到$i 文件
51. then
52.     if [ ${cur1}/${i} == ${cur2} ]
53.     then
54.         echo "file exist" 1>/dev/null
55.     elif [ -d ${cur1}/${i} ] # 目录文件
56.     then
57.         cp -a ${cur1}/${i} $cur2
58.     else
59.         cp -a ${cur1}/${i} $cur2
60.     fi
61. fi
62.
63. status1=0
64. done
65.}
66.#同步
67.Sync2(){
68.    local cur1 cur2 status1 list1 list2 i j
69.    cur1=$1 # 源目录
70.    cur2=$2 # 目标目录
71.    status1=0 # 是否找到
72.    #若目标目录不存在,则创建目录
73.    if [ ! -d $cur2 ]
74.    then
75.        mkdir $cur2
76.    fi
```

```
77. list1=$(ls $cur1) # $1 的全部文件
78. list2=$(ls $cur2) # $2 的全部文件
79. #遍历list1,即源目录所有文件
80. for i in $list1
81. do
82. #遍历list2,即目标目录所有文件
83. for j in $list2
84. do
85. if [[ $i != $j ]] # 不同文件
86. then
87. continue;
88. elif [ -d ${cur1}/${i} ] && [ -d ${cur2}/${j} ] # 同名且同为目
    录文件
89. then
90. status1=1 # 已找到
91. Sync2 ${cur1}/${i} ${cur2}/${j}
92. elif [ -f ${cur1}/${i} ] && [ -f ${cur2}/${j} ] # 同名且同为普
    通文件
93. then
94. status1=1 # 已找到
95. if [[ `find ${cur1}/${i} -
    newer ${cur2}/${j}` == ${cur1}/${i} ]] # $i 更新
96. then
97. cp -rf ${cur1}/${i} ${cur2}/${j} # 更新$j 为$i
98. fi
99. fi
100. done
101. #首先按照list2的目录同步list1内文件
102. if [[ $status1 == 0 ]] # 在$list2中未找到$i文件
103. then
104. if [ ${cur1}/${i} == ${cur2} ]
105. then
106. echo "file exist" 1>/dev/null
107. elif [ -d ${cur1}/${i} ] # 目录文件
108. then
109. cp -a ${cur1}/${i} $cur2
110. else
111. cp -a ${cur1}/${i} $cur2
112. fi
113. fi
114. #外循环重置
115. status1=0
116. done
117. #全部重置
```

```
118. status1=0
119. #反向更新, 将目标目录内文件反向更新到源目录
120. for i in $list2
121. do
122. #源目录
123. for j in $list1
124. do
125. if [[ ${i} != ${j} ]] # 不同文件
126. then
127. continue;
128. #此处不进行增量备份
129. elif [ -f ${cur2}/${i} ] && [ -f ${cur1}/${j} ] # 同名且同为普
    通文件
130. status1=1 # 已找到
131. if [[ `find ${cur2}/${i} -
    newer ${cur1}/${j}` == ${cur2}/${i} ]] # $i 更新
132. then
133. cp -rf ${cur2}/${i} ${cur1}/${j} # 更新$j 为$i
134. fi
135. fi
136. done
137. #删除更新
138. if [[ $status1 == 0 ]] # 在源目录内未找到
139. then
140. if [ ${cur1} == ${cur2}/${i} ]
141. then
142. echo "file exist" 1>/dev/null
143. elif [ -d ${cur2}/${i} ] # 目录文件
144. then
145. rm -r ${cur2}/${i} # 删除目标目录内的, 但不在源目录内的文件
146. else
147. cp ${cur2}/${i} ${cur1}
148. fi
149. fi
150. status1=0;
151. done
152.}
153.#入口
154.#保存传参
155.src=$1
156.dest=$2
157.echo "选择一项功能:"
158.echo " 1:备份"
159.echo " 2:同步"
```

```
160.echo -n "你的选择: "  
161.read line  
162.#根据输入参数选择函数执行  
163.case $line in  
164. 1) Sync1 $src $dest  
165. ;;  
166. 2) Sync2 $src $dest  
167. ;;  
168.esac  
169.#输出标志  
170.echo "更新完成"  
171.exit 0
```

程序首先输出提示，并读取用户输入，通过 case 语句进行功能匹配，进而调用 Sync1 或 Sync2 函数进行备份或同步。

在备份功能中，Sync1 函数首先判断当前目标地址是否存在，若不存在则新建地址；之后通过嵌套循环，分别将 src 目录中的文件与 dest 目录的文件依次匹配：若未匹配上，则直接 continue；若匹配上且为子目录，则递归进入该子目录并记录找到，之后将 dest 中不存在但出现在 src 中的文件统一复制到 dest 中，实现增量复制；在同步功能中，Sync2 函数同样首先判断当前目标地址是否存在，若不存在则新建地址；之后通过两次嵌套循环，分别从正向和反向将 src 目录中的文件与 dest 目录的文件依次匹配：若未匹配上，则 continue；若匹配上且为子目录，则递归进入该子目录并记录找到；若匹配上且为普通文件，则记录找到并判断 src 中文件修改日期是否更近，日期更近则将该文件在 dest 中更新，若没找到，则直接将文件复制到 dest 中；之后反向更新，在 dest 中以同样的方式反向将修改日期更近的文件反向更新，并且将 dest 中删除的文件在 src 中删除，实现单向删除，双向更新，最终完成任务并结束函数。

备份功能测试：

①创建~/tmp1 目录，并使其为空，之后通过备份功能将~/lab2 中的内容备份至~/tmp1 中；

测试结果：运行正确


```

ziteng@Ziteng-VBox:~$ mkdir tmp1
ziteng@Ziteng-VBox:~$ dirsinc ./lab2 ./tmp1
选择一项功能:
    1:备份
    2:同步
你的选择: 1
更新完成
ziteng@Ziteng-VBox:~$ ls -l ./lab2
总用量 16
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:34 1.sh
-rwxr-xr-x 1 ziteng ziteng 1029 8月 18 21:34 2.sh
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:35 dirsinc.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:36 test5
ziteng@Ziteng-VBox:~$ ls -l ./tmp1
总用量 16
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:36 1.sh
-rwxr-xr-x 1 ziteng ziteng 1029 8月 18 21:36 2.sh
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:36 dirsinc.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:36 test5
ziteng@Ziteng-VBox:~$

```

②修改~/lab2 中某一文件 1.sh，并再次备份至~/tmp1，检测更新旧文件；

测试结果：运行正确

```

ziteng@Ziteng-VBox:~$ gedit ./lab2/1.sh
ziteng@Ziteng-VBox:~$ dirsinc ./lab2 ./tmp1
选择一项功能:
    1:备份
    2:同步
你的选择: 1
更新完成
ziteng@Ziteng-VBox:~$ ls -l ./lab2
总用量 16
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:39 1.sh
-rwxr-xr-x 1 ziteng ziteng 1029 8月 18 21:34 2.sh
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:35 dirsinc.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:36 test5
ziteng@Ziteng-VBox:~$ ls -l ./tmp1
总用量 16
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:39 1.sh
-rwxr-xr-x 1 ziteng ziteng 1029 8月 18 21:36 2.sh
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:36 dirsinc.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:36 test5
ziteng@Ziteng-VBox:~$

```

③删除~/lab3 中某一文件 dirsinc.sh，增加 1.txt，并再次备份至~/tmp1，检测增量备份；

测试结果：成功

```

ziteng@Ziteng-VBox:~$ rm lab2/2.sh
ziteng@Ziteng-VBox:~$ touch lab2/1.txt
ziteng@Ziteng-VBox:~$ dirsnc ./lab2 ./tmp1
选择一项功能:
    1:备份
    2:同步
你的选择: 1
更新完成
ziteng@Ziteng-VBox:~$ ls -l ./lab2
总用量 12
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:39 1.sh
-rw-r--r-- 1 ziteng ziteng  0 8月 18 21:41 1.txt
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:35 dirsnc.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:36 test5
ziteng@Ziteng-VBox:~$ ls -l ./tmp1
总用量 16
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:39 1.sh
-rw-r--r-- 1 ziteng ziteng  0 8月 18 21:41 1.txt
-rwxr-xr-x 1 ziteng ziteng 1029 8月 18 21:36 2.sh
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:36 dirsnc.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:36 test5
ziteng@Ziteng-VBox:~$

```

同步功能测试:

①首先在文件管理中删除原有主目录下的 tmp1 目录，并调用 dirsnc 命令同步功能将~/lab2 与~/tmp1 同步，检测程序在第二个参数目录不存在时能否如期创建新测试目录;

结果: 运行正确

②根据题目描述，期望运行结果为两目录内容完全一致并且皆为最新文件;

测试结果: 运行正确

```

ziteng@Ziteng-VBox:~$ dirsnc ./lab2 ./tmp1
选择一项功能:
    1:备份
    2:同步
你的选择: 2
更新完成
ziteng@Ziteng-VBox:~$ ls -l ./lab2
总用量 12
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:44 1.sh
-rw-r--r-- 1 ziteng ziteng  0 8月 18 21:44 1.txt
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:44 dirsnc.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:44 test5
ziteng@Ziteng-VBox:~$ ls -l ./tmp1
总用量 12
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:44 1.sh
-rw-r--r-- 1 ziteng ziteng  0 8月 18 21:44 1.txt
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:44 dirsnc.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:44 test5
ziteng@Ziteng-VBox:~$

```

③为检验递归子目录功能，我们预先在~/lab2/test5 子目录中加入了两个普通文件和一个目录文件，经检测，子文件递归同步功能正常。

测试结果：运行正确

```
ziteng@Ziteng-VBox:~$ ls -l ./lab2/test5
总用量 12
-rw-r--r-- 1 ziteng ziteng 1954 8月 18 21:44 mediumFile
-rw-r--r-- 1 ziteng ziteng 963 8月 18 21:44 smallFile
drwxr-xr-x 2 ziteng ziteng 4096 8月 18 21:44 test1
ziteng@Ziteng-VBox:~$ ls -l ./tmp1/test5
总用量 12
-rw-r--r-- 1 ziteng ziteng 1954 8月 18 21:44 mediumFile
-rw-r--r-- 1 ziteng ziteng 963 8月 18 21:44 smallFile
drwxr-xr-x 2 ziteng ziteng 4096 8月 18 21:44 test1
ziteng@Ziteng-VBox:~$
```

④分别修改~/tmp1/1.sh 和~/lab2/2.sh 文件，检测旧文件被新文件替换以及双向更新功能；

测试结果：运行正确

```
ziteng@Ziteng-VBox:~$ gedit lab2/1.txt
ziteng@Ziteng-VBox:~$ gedit tmp1/1.sh
ziteng@Ziteng-VBox:~$ dirsync ./lab2 ./tmp1
选择一项功能:
    1:备份
    2:同步
你的选择: 2
更新完成
ziteng@Ziteng-VBox:~$ ls -l ./lab2
总用量 12
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:48 1.sh
-rw-r--r-- 1 ziteng ziteng 0 8月 18 21:48 1.txt
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:44 dirsync.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:44 test5
ziteng@Ziteng-VBox:~$ ls -l ./tmp1
总用量 12
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:48 1.sh
-rw-r--r-- 1 ziteng ziteng 0 8月 18 21:48 1.txt
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:44 dirsync.sh
drwxr-xr-x 3 ziteng ziteng 4096 8月 18 21:44 test5
ziteng@Ziteng-VBox:~$
```

⑤删除~/lab3 中的 test5 文件夹，测试目标文件的相应删除状况；

测试结果：运行正确

```

ziteng@Ziteng-VBox:~$ rm -r lab2/test5
ziteng@Ziteng-VBox:~$ dirsync ./lab2 ./tmp1
选择一项功能:
  1:备份
  2:同步
你的选择: 2
更新完成
ziteng@Ziteng-VBox:~$ ls -l ./lab2
总用量 8
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:50 1.sh
-rw-r--r-- 1 ziteng ziteng  0 8月 18 21:48 1.txt
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:44 dirsnc.sh
ziteng@Ziteng-VBox:~$ ls -l ./tmp1
总用量 8
-rwxr-xr-x 1 ziteng ziteng 719 8月 18 21:50 1.sh
-rw-r--r-- 1 ziteng ziteng  0 8月 18 21:48 1.txt
-rwxr-xr-x 1 ziteng ziteng 3265 8月 18 21:44 dirsnc.sh
ziteng@Ziteng-VBox:~$

```

4. （25 分）用 bash 编写程序，实现一个简单的**作业管理系统**。可以使用图形和数据库软件包来实现，也可以用文件形式实现数据的存储。系统至少具备以下的基本功能：

系统中根据不同的权限分为三类用户：管理员、教师、学生，简要说明如下：

- 管理员：

创建、修改、删除、显示（list）教师帐号；教师帐户包括教师工号、教师姓名，教师用户以教师工号登录。

创建、修改、删除课程；绑定（包括添加、删除）课程与教师用户。课程名称以简单的中文或英文命名。

- 教师：

对某门课程，创建或导入、修改、删除学生帐户，根据学号查找学生帐号；学生帐号的基本信息包括学号和姓名，学生使用学号登录。

发布课程信息。包括新建、编辑、删除、显示（list）课程信息等功能。

布置作业或实验。包括新建、编辑、删除、显示（list）作业或实验等功能。

查找、打印所有学生的完成作业情况。

- 学生：

在教师添加学生账户后，学生就可以登录系统，并完成作业和实验。

基本功能：新建、编辑作业或实验功能；查询作业或实验完成情况。

需求描述

➤ 数据需求:

本系统对数据存储分为两种类型，一种是文件格式，一种是子目录格式，其中课程的对应关系通过子目录存储，内部可储存文件格式的作业/实验，对于用户信息（用户类型，账号）和密码，同样通过普通文件存储。

➤ 功能需求:

实现命令行 UI 界面，对每一步操作均给予菜单提示，根据用户输入进行层级切换以及操作实现。所有类型用户的登陆都需要进行账号与密码的匹配。

管理员: 管理员可对课程和教师的数据进行管理，包括对教师账户的增删改查，对课程数据的增删改查，以及将教师与课程进行绑定与解绑操作。

教师: 教师可对于某门课程进行选课学生、课程信息、作业实验管理，包括对学生账户的增删改查，对课程信息的增删改显，以及对课程作业要求的增删改显。如果添加的学生不在全部学生名单之中，则先将其添加入系统，再加入课程名单中，同时老师还可以检查学生的完成情况。

学生: 学生可以选择自己修读的课程，上传作业、修改作业，并查看自己作业完成情况。

设计文档

➤ 设计思想

本系统的设计基于功能需求进行，对于 shell 脚本语言，其主要特点为文件操作能力较强，逻辑能力较强，但数值操作能力较弱，因此本系统所有结构基于文件系统，均采用文件的形式操作，对于不同层级使用子目录或普通文件表示结构含义。

对于 shell 语言函数调用的特点，我们采用树状调用结构，自顶而下，通过 `entrance.sh` 作为系统唯一入口，并调用不同模块以及函数，通过逐级深入，层级交互进行功能实现，在层级间提供向上和向下的入

口，通过用户输入来进行功能调用。

同时，为方便用户直观化了解当前系统状态，除查询功能之外，本系统还提供了命令行菜单，并且提供返回通道，供用户在功能之间灵活切换。

➤ 功能模块

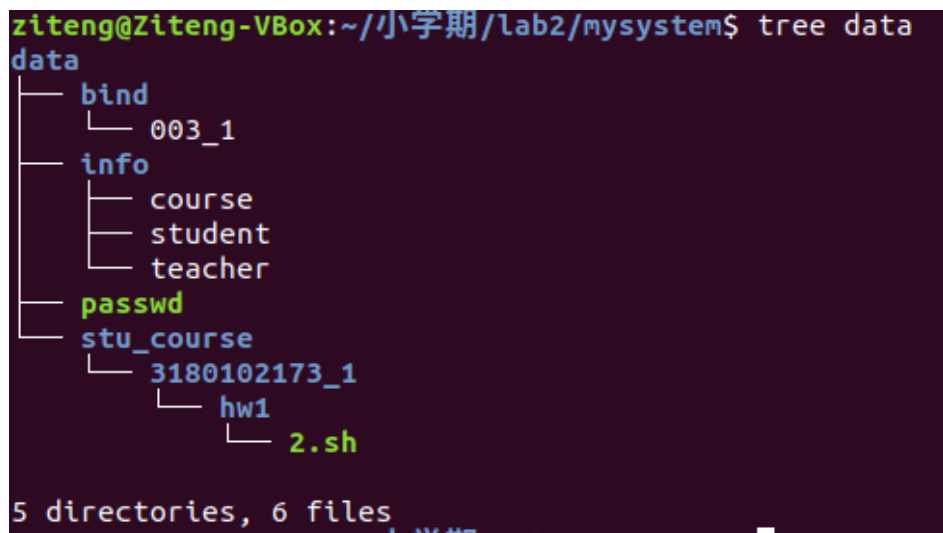
本系统基于文件和功能结构，可以分为四大功能模块：入口模块、管理员模块、教师模块、学生模块。

进入系统首先进入入口模块，在这一模块中，用户根据自己的身份选择子模块，进入不同功能模块，在这一过程中，入口模块将包含选中的子模块文件。

对于管理员、教师、学生三个子功能模块，每个模块在入口处均设立了身份验证功能，用户需要根据账号和密码，与储存在 `data/passwd` 中的数据进行比对，成功后方可进入系统。

➤ 数据结构

本系统基于文件系统，将所有数据均以文本、普通文件、子目录的形式存储，其主要结构如下：



```
ziteng@Ziteng-VBox:~/小学期/lab2/mysystem$ tree data
data
├── bind
│   └── 003_1
├── info
│   ├── course
│   ├── student
│   └── teacher
├── passwd
├── stu_course
│   └── 3180102173_1
│       └── hw1
│           └── 2.sh
└── 5 directories, 6 files
```

其中 `info` 目录存储了教师、学生、课程的基本信息，用于与之后的高级操作进行核实与比对；`passwd` 文件中存储了所有账户的账号和密码；`stu_course` 目录存储了学生和课程的对应信息，教师可直接在本目录内根据课程 `id` 为选课学生布置作业，学生在作业目录内可提交作业文件；`bind` 目录存储了教师和课程的对应信息，绑定功能在

此目录内实现。

➤ 算法

管理员对教师和课程的创建通过重定向写入对应文件的方式实现，其中文件分别为 `data/info/teacher` 和 `data/info/course` 两个文件，教师对学生信息的修改对应 `data/info/student` 文件，对于教师和学生的创建、修改、删除应伴随 `data/passwd` 文件的变化。

对于信息的写入，采用的是单行单信息组，通过 `echo` 的重定向实现。

课程信息和作业实验的发布通过对 `data/bind` 和 `data/stu_course` 目录内的子目录进行操作，通过增加、修改、删除普通文件来实现。

源代码

➤ `entrance.sh`

```
1. #文件名:entrance.sh
2. #作者:王子腾 3180102173
3. #创建日期:2020-07-27
4. #最后修改:2020-07-30
5.
6. #! /bin/bash
7.
8. #身份验证提示
9. echo "选择权限:"
10. echo "1 管理员"
11. echo "2 教师"
12. echo "3 学生"
13. echo "q 退出"
14.
15. #获取用户输入身份码
16. read type
17. #将相应文件包含在本段中,并启动入口文件
18. case $type in
19.     #管理员
20.     1)  . ./adm.sh
21.         admin;;
22.     #教师
23.     2)  . ./teach.sh;
24.         teacher;;
25.     #学生
26.     3)  . ./stu.sh
27.         student;;
28.     #正常退出
```

```
29.     q)  exit 0;;
30.     #其他输入, 非法
31.     *)  echo "错误输入!"
32.         exit 1;;
33.esac
34.# 错误退出
35.exit 1
```

➤ adm.sh

```
1. #文件名:adm.sh
2. #作者:王子腾 3180102173
3. #创建日期:2020-07-25
4. #最后修改:2020-07-28
5. #! /bin/bash
6.
7. #入口文件
8. admin()
9. {
10.     login_adm      #管理员登陆
11.     operation_adm  #操作界面
12.     exit 1         #错误返回
13.}
14.
15.#管理员登陆
16.login_adm()
17.{
18.    #获取密码
19.    passwd=$(cat data/passwd | grep "^a" | awk '{print $2}')
20.    #五次试错机会
21.    for((i=0;i<5;i++))
22.    do
23.        echo -n "管理员密码-> "
24.        read input
25.        #检测输入值与密码是否匹配
26.        if test $passwd = $input
27.        then
28.            echo "欢迎 admin 登入"
29.            echo "";
30.            #成功匹配
31.            return 0;
32.        else
33.            #回显错误次数
34.            j=$((4-$i));
35.            echo "密码错误! 您还有 $j 次机会"
36.        fi
```



```
37.     done
38.     #密码错误退出, 错误码 2
39.     exit 2
40.}

41.#管理员初始菜单
42.operation_adm()
43.{
44.    #操作提示
45.    echo "请选择操作:"
46.    echo "1 管理教师信息"
47.    echo "2 管理课程信息"
48.    echo "q 退出系统"
49.
50.    #获取输入序号
51.    read type
52.    case $type in
53.        1)  manage_teach;;
54.        2)  manage_course;;
55.        q)  exit 0;;
56.        #其他输入, 错误
57.        *)  echo "错误输入!"
58.            operation_adm;;
59.    esac
60.    #未知异常
61.    exit 1
62.}
63.
64.#管理教师信息
65.manage_teach()
66.{
67.    #操作提示
68.    echo ""
69.    echo "请选择具体操作:"
70.    echo "1 增加教师信息"
71.    echo "2 删除教师信息"
72.    echo "3 修改教师信息"
73.    echo "4 查询教师信息"
74.    echo "r 返回上一层"
75.    echo "q 退出系统"
76.
77.    #获取输入序号
78.    read op
79.    case $op in
80.        1)  add_teach;;
```

```
81.         2) del_teach;;
82.         3) mod_teach;;
83.         4) que_teach;;
84.         r) operation_adm;;
85.         q) exit 0;;
86.         #其他输入, 错误
87.         *) echo "错误输入!"
88.            manage_teach;;
89.     esac
90.     #未知异常
91.     exit 1
92.}
93.
94.#添加教师账户, 以工号为主键primary key
95.add_teach()
96.{
97.    #操作提示
98.    echo ""
99.    echo -n "请输入教师工号-> "
100.    #获取工号
101.    read id
102.    #检查已有账户, 工号是否重复
103.    if test 0 != $(cat data/info/teacher | awk '{print $1}' | gr
        ep -c "$id")
104.    then
105.        #重复, 返回上一界面
106.        echo "工号已存在"
107.        manage_teach
108.        exit 1
109.    fi
110.    #不重复, 进一步添加
111.    echo -n "请输入教师姓名-> "
112.    read name
113.    #将账户输入账户文件
114.    echo $id $name >> data/info/teacher
115.    echo -n "请输入新账户密码-> "
116.    read passwd
117.    #将账户密码输入密码文件
118.    echo "t"$id $passwd >> data/passwd
119.
120.    echo "添加成功! 新教师账户为$id $name"
121.    #返回上一级
122.    manage_teach
123.    #异常退出
```

```
124.     exit 1
125. }
126.
127. #删除教师账户
128. del_teach()
129. {
130.     #操作提示
131.     echo ""
132.     echo -n "请输入教师工号-> "
133.     read id
134.     #检查是否存在该工号
135.     if test 0 != $(cat data/info/teacher | awk '{print $1}' | gr
        ep -c "$id")
136.     then
137.         #存在, 删除账户信息和密码信息
138.         sed -i '/'"$id"'/d' data/info/teacher
139.         sed -i '/'"$id"'/d' data/passwd
140.         echo "删除成功!"
141.         #返回上一级
142.         manage_teach
143.         #未知错误, 退出错误码 1
144.         exit 1
145.     else
146.         echo "工号不存在!"
147.         #返回上一级
148.         manage_teach
149.         #未知错误, 退出错误码 1
150.         exit 1
151.     fi
152. }
153.
154. mod_teach()
155. {
156.     #操作提示
157.     echo ""
158.     echo -n "请输入教师工号-> "
159.     read id
160.     #检查是否存在该工号
161.     if test 0 != $(cat data/info/teacher | awk '{print $1}' | gr
        ep -c "$id")
162.     then
163.         #存在, 执行修改操作
164.         #获取修改后账户信息
165.         echo -n "请输入修改后教师工号-> "
```

```

166.      read newid
167.      echo -n "请输入修改后教师姓名-> "
168.      read name
169.      echo -n "请输入修改后账户密码-> "
170.      read newpswd
171.      #删除原账户信息和密码信息
172.      sed -i '/'"$id"/d' data/info/teacher
173.      sed -i '/'"$id"/d' data/passwd
174.      #添加新账户信息和密码信息
175.      echo $newid $name >> data/info/teacher
176.      echo "t"$newid $newpswd >> data/passwd
177.      echo "修改成功! 新账户为:$newid $name"
178.      #返回上一级
179.      manage_teach
180.      #未知错误, 退出错误码 1
181.      exit 1
182.  else
183.      echo "工号不存在!"
184.      #返回上一级
185.      manage_teach
186.      #未知错误, 退出错误码 1
187.      exit 1
188.  fi
189.}
190.
191.que_teach()
192.{
193.    #操作提示
194.    echo ""
195.    echo "请选择查询方式:"
196.    echo "1 按教师工号查询"
197.    echo "2 按教师姓名查询"
198.    echo "3 列举所有教师"
199.    echo "r 返回上一层"
200.    read type
201.    case $type in
202.        1) que_teach_id;;
203.        2) que_teach_name;;
204.        3) que_teach_list;;
205.        r) manage_teach;;
206.        #其他输入, 不合法
207.        *) echo "输入错误!"
208.           que_teach;;
209.    esac

```

```
210.     #未知错误,退出错误码 1
211.     exit 1
212. }
213.
214. que_teach_id()
215. {
216.     #操作提示
217.     echo ""
218.     echo -n "请输入教师工号-> "
219.     read id
220.     #检查是否存在该工号
221.     if test 0 != $(cat data/info/teacher | awk '{print $1}' | gr
        ep -c "$id")
222.     then
223.         #查找到
224.         echo "查询结果为:"
225.         cat data/info/teacher | grep "^$id"
226.         #返回上一级
227.         que_teach
228.         #未知错误,退出错误码 1
229.         exit 1
230.     else
231.         echo "工号不存在!"
232.         #返回上一级
233.         que_teach
234.         #未知错误,退出错误码 1
235.         exit 1
236.     fi
237. }
238.
239. que_teach_name()
240. {
241.     #操作提示
242.     echo ""
243.     echo -n "请输入教师姓名-> "
244.     read name
245.     #检查是否存在该工号
246.     if test 0 != $(cat data/info/teacher | awk '{print $2}' | gr
        ep -c "$name")
247.     then
248.         echo "查询结果为:"
249.         #管道操作,输出目标行信息
250.         cat data/info/teacher | grep "$name"
251.         #返回上一级
```

```
252.         que_teach
253.         #未知错误, 退出错误码 1
254.         exit 1
255.     else
256.         echo "姓名不存在!"
257.         #返回上一级
258.         que_teach
259.         #未知错误, 退出错误码 1
260.         exit 1
261.     fi
262. }
263.
264. que_teach_list()
265. {
266.     #展示所有教师账户
267.     cat data/info/teacher | sort
268.     #返回上一级
269.     que_teach
270.     #未知错误, 退出错误码 1
271.     exit 1
272. }
273.
274. manage_course()
275. {
276.     #操作提示
277.     echo ""
278.     echo "请选择具体操作:"
279.     echo "1 增加课程信息"
280.     echo "2 删除课程信息"
281.     echo "3 修改课程信息"
282.     echo "4 绑定课程信息"
283.     echo "5 解除课程绑定"
284.     echo "r 返回上一层"
285.     echo "q 退出系统"
286.
287.     #获取操作码
288.     read type
289.     case $type in
290.         1) add_course;;
291.         2) del_course;;
292.         3) mod_course;;
293.         4) bind_course;;
294.         5) unbind_course;;
295.         r) operation_adm;;
```

```
296.         q)  exit 0;;
297.         #其他输入, 不合法
298.         *)  echo "输入错误!"
299.             manage_course;;
300.     esac
301.
302. #未知错误, 退出错误码 1
303.     exit 1
304. }
305.
306. add_course()
307. {
308.     #操作提示
309.     echo ""
310.     echo -n "请输入课程代号-> "
311.     read id
312.     #检查是否存在该课程号
313.     if test 0 != $(cat data/info/course | awk '{print $1}' | gre
        p -c "$id")
314.     then
315.         echo "课程代号已存在"
316.         #返回上一级
317.         manage_teach
318.         #未知错误, 退出错误码 1
319.         exit 1
320.     fi
321.
322.     echo -n "请输入课程名称-> "
323.     read name
324.     #输入课程信息
325.     echo $id $name >> data/info/course
326.
327.     echo "添加成功! 新课程为$id $name"
328.     #返回上一级
329.     manage_course
330.     #未知错误, 退出错误码 1
331.     exit 1
332. }
333.
334. del_course()
335. {
336.     #操作提示
337.     echo ""
338.     echo -n "请输入课程代号-> "
```

```
339.     read id
340.     #检查是否存在该课程号
341.     if test 0 != $(cat data/info/course | awk '{print $1}' | gre
        p -c "$id")
342.     then
343.         #删除该课程信息
344.         sed -i '/'"$id"/d' data/info/course
345.         echo "删除成功!"
346.         #返回上一级
347.         manage_course
348.         #未知错误,退出错误码 1
349.         exit 1
350.     else
351.         echo "课程不存在!"
352.         #返回上一级
353.         manage_course
354.         #未知错误,退出错误码 1
355.         exit 1
356.     fi
357. }
358.
359. mod_course()
360. {
361.     #操作提示
362.     echo ""
363.     echo -n "请输入课程代号-> "
364.     read id
365.     #检查是否存在该课程号
366.     if test 0 != $(cat data/info/course | awk '{print $1}' | gre
        p -c "$id")
367.     then
368.         echo -n "请输入修改后课程代号-> "
369.         read newid
370.         echo -n "请输入修改后课程名称-> "
371.         read name
372.         #删除原有课程信息
373.         sed -i '/'"$id"/d' data/info/course
374.         #输入新信息
375.         echo $newid $name >> data/info/course
376.         echo "修改成功! 新课程为:$newid $name"
377.         #返回上一级
378.         manage_course
379.         #未知错误,退出错误码 1
380.         exit 1
```



```
381.     else
382.         echo "课程号不存在!"
383.         #返回上一级
384.         manage_course
385.         #未知错误, 退出错误码 1
386.         exit 1
387.     fi
388. }
389.
390. bind_course()
391. {
392.     #操作提示
393.     echo ""
394.     echo -n "请输入课程代号-> "
395.     read cid
396.     #检查是否存在该课程号
397.     if test 0 = $(cat data/info/course | awk '{print $1}' | grep
        -c "$cid")
398.     then
399.         echo "课程号不存在"
400.         #返回上一级
401.         manage_course
402.         #未知错误, 退出错误码 1
403.         exit 1
404.     fi
405.
406.     echo -n "请输入教师工号-> "
407.     read tid
408.     #检查是否存在该工号
409.     if test 0 = $(cat data/info/teacher | awk '{print $1}' | gre
        p -c "$tid")
410.     then
411.         echo "工号不存在"
412.         #返回上一级
413.         manage_course
414.         #未知错误, 退出错误码 1
415.         exit 1
416.     fi
417.
418.     #检查是否已经绑定
419.     if test -f "data/bind/$tid_"$cid"
420.     then
421.         echo "该教师已绑定该课程"
422.         #返回上一级
```

```
423.         manage_course
424.         #未知错误,退出错误码 1
425.         exit 1
426.     fi
427.
428.     #建立新文件
429.     touch "data/bind/$tid_" "$cid"
430.     echo "绑定成功! $tid -- $cid"
431.     #返回上一级
432.     manage_course
433.     #未知错误,退出错误码 1
434.     exit 1
435.}
436.
437.unbind_course()
438.{
439.    #操作提示
440.    echo ""
441.    echo -n "请输入课程代号-> "
442.    read cid
443.    #检查是否存在该课程号
444.    if test 0 = $(cat data/info/course | awk '{print $1}' | grep
        -c "$cid")
445.    then
446.        echo "课程号不存在"
447.        #返回上一级
448.        manage_course
449.        #未知错误,退出错误码 1
450.        exit 1
451.    fi
452.
453.    echo -n "请输入教师工号-> "
454.    read tid
455.    #检查是否存在该工号
456.    if test 0 = $(cat data/info/teacher | awk '{print $1}' | gre
        p -c "$tid")
457.    then
458.        echo "工号不存在"
459.        #返回上一级
460.        manage_course
461.        #未知错误,退出错误码 1
462.        exit 1
463.    fi
464.
```

```

465.     #检查是否已绑定
466.     if test -f "data/bind/$tid_" "$cid"
467.     then
468.         rm "data/bind/$tid_" "$cid"
469.         echo "解除绑定成功! $tid != $cid"
470.         #返回上一级
471.         manage_course
472.         #未知错误, 退出错误码 1
473.         exit 1
474.     else
475.         echo "该教师未绑定该课程"
476.         #返回上一级
477.         manage_course
478.         #未知错误, 退出错误码 1
479.         exit 1
480.     fi
481. }

```

➤ teach.sh

```

1. #文件名: teach.sh
2. #作者: 王子腾 3180102173
3. #创建日期: 2020-07-26
4. #最后修改: 2020-07-28
5.
6. #!/bin/bash
7.
8. #教师功能入口
9. teacher()
10. {
11.     echo -n "请输入教师工号-> "
12.     read tid
13.     #检查该工号是否存在
14.     if test 0 = $(cat data/info/teacher | grep $tid | wc -l)
15.     then
16.         echo "用户不存在"
17.         teacher
18.         #函数错误返回, 退出错误码 1
19.         exit 1
20.     fi
21.
22.     #进入登陆模块
23.     login_teacher
24.     echo "欢迎老师登陆!"
25.
26.     #操作菜单

```

```
27. operation_teacher
28. #函数错误返回,退出错误码 1
29. exit 1
30.}
31.
32.#登录验证功能
33.login_teacher()
34.{
35.    #获取该账号密码
36.    passwd=$(cat data/passwd | grep "^t$tid" | awk '{print $2}')
37.    #提供5次试错机会
38.    for((i=0; i<5; i++))
39.    do
40.        echo -n "登录密码-> "
41.        read input
42.        #比较输入是否与密码匹配
43.        if test $passwd = $input
44.        then
45.            #验证成功,返回
46.            return 0
47.        else
48.            #计算剩余次数
49.            j=$((4-$i));
50.            echo "密码错误! 您还有 $j 次机会"
51.        fi
52.    done
53.    #密码验证失败,退出码 2
54.    exit 2
55.}
56.
57.#教师操作主界面
58.operation_teacher()
59.{
60.    #声明数学整数变量
61.    declare -i i=1
62.    declare -i option
63.    #操作提示
64.    echo ""
65.    echo "请选择课程:"
66.    #打印出所有课程信息,带序号
67.    for course in $(ls data/bind/$tid* | grep -
        o "[:alnum:]*$" | sort)
68.    do
```

```

69.         echo -
           e $i $course $(cat data/info/course | grep "^$course" | awk '{pri
nt $2}')
70.         i=i+1
71.     done
72.
73. #获取输入选择
74.     read option
75.     #输入是否合法(在给出序号范围内)
76.     if test $option -le $(ls data/bind/$tid* | grep -
       c "[[:alnum:]]*$")
77.     then
78.         #获取课程id
79.         cid=$(ls data/bind/$tid* | grep -
       o "[[:alnum:]]*$" | sort | sed -n $option"p")
80.         #获取课程名
81.         name=$(cat data/info/course | grep "^$cid" | awk '{print
       $2}')
82.         manage
83.         #函数错误返回,退出错误码 1
84.         exit 1
85.     else
86.         echo "输入错误!"
87.         #重新尝试
88.         operation_teacher
89.         #函数错误返回,退出错误码 1
90.         exit 1
91.     fi
92. }
93.
94. #管理该课程相关信息
95. manage()
96. {
97.     #操作提示信息
98.     echo ""
99.     echo "请选择进一步操作"
100.    echo "1 管理学生信息"
101.    echo "2 管理课程信息"
102.    echo "3 管理作业实验"
103.    echo "r 返回上一级"
104.    echo "q 退出"
105.    read op
106.
107.    #根据输入选择功能入口

```

```
108.     case $op in
109.         1)  manage_student;;
110.         2)  manage_course;;
111.         3)  manage_homework;;
112.         r)  operation_teacher;;
113.         q)  exit 0;;
114.         #其他输入, 非法
115.         *)  echo "错误输入!"
116.             operation_teacher;;
117.     esac
118.     #函数错误返回, 退出错误码 1
119.     exit 1
120. }
121.
122. #管理学生账户
123. manage_student()
124. {
125.     #操作提示信息
126.     echo ""
127.     echo "请选择具体操作"
128.     echo "1 创建学生账户"
129.     echo "2 修改学生账户"
130.     echo "3 删除学生账户"
131.     echo "4 查找选课学生"
132.     echo "r 返回上一级"
133.     echo "q 退出"
134.     read op
135.     #根据输入选择功能入口
136.     case $op in
137.         1)  add_stu;;
138.         2)  mod_stu;;
139.         3)  del_stu;;
140.         4)  que_stu;;
141.         r)  operation_teacher;;
142.         q)  exit 0;;
143.         #其他输入, 非法
144.         *)  echo "输入错误!"
145.             exit 1;;
146.     esac
147.     #函数错误返回, 退出错误码 1
148.     exit 1
149. }
150.
151. #添加学生选课信息
```

```
152.add_stu()
153.{
154.    #操作提示信息
155.    echo ""
156.    echo -n "请输入新建学生学号-> "
157.    read sid
158.    #检查是否存在该学号
159.    if test 0 != $(cat data/info/student | awk '{print $1}' | gr
        ep -c "$sid")
160.    then
161.        #检测文件存在
162.        if test -d data/stu_course/$sid_"$cid
163.        then
164.            echo "该学生已选课!"
165.            manage_student
166.            #函数错误返回,退出错误码 1
167.            exit 1
168.        fi
169.        #建立子目录
170.        mkdir data/stu_course/$sid_"$cid
171.        echo "添加成功!"
172.        manage_student
173.        #函数错误返回,退出错误码 1
174.        exit 1
175.    fi
176.
177.    #不存在学号,需新建学生信息
178.    echo -n "请输入学生姓名-> "
179.    read sname
180.    echo -n "请输入账户密码-> "
181.    read spswd
182.    #将新信息输入
183.    echo $sid $sname >> data/info/student
184.    echo "s"$sid $spswd >> data/passwd
185.    #建立子目录
186.    mkdir data/stu_course/$sid_"$cid
187.    echo "添加成功!"
188.    manage_student
189.    #函数错误返回,退出错误码 1
190.    exit 1
191.}
192.
193.#删除选课信息
194.del_stu()
```

```
195.{
196.    #操作提示信息
197.    echo ""
198.    echo -n "请输入删除学生学号-> "
199.    read sid
200.    #检测是否存在选课信息
201.    if test -d data/stu_course/$sid_"$cid
202.    then
203.        #将选课目录删除
204.        rm -r data/stu_course/$sid_"$cid
205.        echo "删除成功!"
206.        manage_student
207.        #函数错误返回,退出错误码 1
208.        exit 1
209.    else
210.        echo "无该学生选课记录!"
211.        manage_student
212.        #函数错误返回,退出错误码 1
213.        exit 1
214.    fi
215.}
216.
217.#修改学生信息
218.mod_stu()
219.{
220.    #操作提示信息
221.    echo ""
222.    echo -n "请输入修改学生学号-> "
223.    read sid
224.    #检查是否存在该学号
225.    if test 0 != $(cat data/info/student | awk '{print $1}' | gr
        ep -c "$sid")
226.    then
227.        echo -n "请输入修改后学生学号-> "
228.        read newid
229.        echo -n "请输入修改后学生姓名-> "
230.        read name
231.        #删除原账户信息
232.        sed -i '/'"$tid"/d' data/info/student
233.        #输入新信息
234.        echo $newid $name >> data/info/student
235.        echo "修改成功! 新账户为:$newid $name"
236.        manage_student
237.        #函数错误返回,退出错误码 1
```



```
238.         exit 1
239.     else
240.         echo "学号不存在!"
241.         manage_student
242.         #函数错误返回,退出错误码 1
243.         exit 1
244.     fi
245. }
246.
247. #查询选课学生
248. que_stu()
249. {
250.     #操作提示信息
251.     echo ""
252.     echo -n "请输入查询学生学号-> "
253.     read sid
254.     #检查是否存在该学号
255.     if test 0 = $(cat data/info/student | grep -c "^$sid")
256.     then
257.         echo "该学生不存在!"
258.         manage_student
259.         #函数错误返回,退出错误码 1
260.         exit 1
261.     else
262.         #检查是否存在该学号的选课信息
263.         if test -d data/stu_course/$sid_"$cid
264.         then
265.             echo "学生信息为:"
266.             cat data/info/student | grep "^$sid"
267.             manage_student
268.             #函数错误返回,退出错误码 1
269.             exit 1
270.         else
271.             echo "该学生未选课!"
272.             manage_student
273.             #函数错误返回,退出错误码 1
274.             exit 1
275.         fi
276.     fi
277. }
278.
279. #管理课程信息
280. manage_course()
281. {
```

```
282.      #操作提示信息
283.      echo ""
284.      echo "请选择具体操作"
285.      echo "1 新建课程信息"
286.      echo "2 删除课程信息"
287.      echo "3 编辑课程信息"
288.      echo "4 显示课程信息"
289.      echo "r 返回上一级"
290.      echo "q 退出"
291.      read op
292.      #根据选择提供相应入口
293.      case $op in
294.          1)  add_info;;
295.          2)  del_info;;
296.          3)  mod_info;;
297.          4)  list_info;;
298.          r)  operation_teacher;;
299.          q)  exit 0;;
300.          #其他输入, 非法
301.          *)  echo "输入错误!"
302.              manage_course;;
303.      esac
304.      #函数错误返回, 退出错误码 1
305.      exit 1
306. }
307.
308. #添加课程信息
309. add_info()
310. {
311.     #操作提示信息
312.     echo ""
313.     echo -n "请输入课程信息-> "
314.     read line
315.     #加入课程信息
316.     echo $line >> data/bind/$tid_"$cid
317.     echo "信息添加成功!"
318.     manage_course
319.     #函数错误返回, 退出错误码 1
320.     exit 1
321. }
322.
323. #删除课程信息
324. del_info()
325. {
```

```
326.     #操作提示信息
327.     echo "文本内容为:"
328.     #展示已有课程信息
329.     cat data/bind/$tid_"$cid
330.     echo ""
331.     echo -n "请输入你要删除的行数-> "
332.     read num
333.     #删除该行信息
334.     sed -i "${num}d" data/bind/$tid_"$cid
335.     echo "删除成功!"
336.     manage_course
337.     #函数错误返回, 退出错误码 1
338.     exit 1
339. }
340.
341. #修改课程信息
342. mod_info()
343. {
344.     #操作提示信息
345.     echo "文本内容为:"
346.     #展示已有课程信息
347.     cat data/bind/$tid_"$cid
348.     echo ""
349.     echo -n "请输入你要修改的行数-> "
350.     read num
351.     echo -n "请输入你要修改的内容-> "
352.     read line
353.     #更新该行信息
354.     sed -i "${num}c${line}" data/bind/$tid_"$cid
355.     echo "修改成功!"
356.     manage_course
357.     #函数错误返回, 退出错误码 1
358.     exit 1
359. }
360.
361. #展示已有课程信息
362. list_info()
363. {
364.     echo ""
365.     #展示已有课程信息
366.     cat data/bind/$tid_"$cid
367.     manage_course
368.     #函数错误返回, 退出错误码 1
369.     exit 1
```

```
370.}
371.
372.#管理作业/实验
373.manage_homework()
374.{
375.    #操作提示信息
376.    echo ""
377.    echo "请选择具体操作"
378.    echo "1 新建作业信息"
379.    echo "2 删除作业信息"
380.    echo "3 编辑作业信息"
381.    echo "4 显示作业信息"
382.    echo "5 查看完成情况"
383.    echo "r 返回上一级"
384.    echo "q 退出"
385.    read op
386.    #根据输入提供入口
387.    case $op in
388.        1) add_homework;;
389.        2) del_homework;;
390.        3) mod_homework;;
391.        4) list_homework
392.           manage_homework;;
393.        5) check_homework;;
394.        r) operation_teacher;;
395.        q) exit 0;;
396.        #其他输入,非法
397.        *) echo "输入错误!"
398.           exit 1;;
399.    esac
400.    #函数错误返回,退出错误码 1
401.    exit 1
402.}
403.
404.#添加作业
405.add_homework()
406.{
407.    #操作提示信息
408.    echo ""
409.    echo -n "请输入作业名称-> "
410.    read wname
411.    #遍历已有选课学生目录
412.    for d in $(ls -d data/stu_course/*$cid)
413.    do
```

```
414.         #创建作业目录
415.         mkdir $d/$wname
416.     done
417.     echo "作业创建成功!"
418.     manage_homework
419.     #函数错误返回,退出错误码 1
420.     exit 1
421. }
422.
423. #删除作业
424. del_homework()
425. {
426.     #列举所有作业
427.     list_homework
428.     echo ""
429.     echo -n "请输入你要删除的序号-> "
430.     read num
431.     #删除目录信息
432.     del=$(ls $studir | head -n $num)
433.     #删除作业子目录
434.     rm -r data/stu_course/*$cid/$del
435.     echo "作业删除成功!"
436.     manage_homework
437.     #函数错误返回,退出错误码 1
438.     exit 1
439. }
440.
441. #修改作业名称
442. mod_homework()
443. {
444.     #操作提示信息
445.     list_homework
446.     echo -n "请输入你要修改的序号-> "
447.     read num
448.     #获取旧名称
449.     oldname=$(ls $studir | head -n $num)
450.     echo -n "请输入新名称-> "
451.     read newname
452.     #遍历该作业
453.     for d in $(ls -d data/stu_course/*$cid/$oldname)
454.     do
455.         #重命名
456.         mv $d $(echo $d | sed 's/'"$oldname"'/'"$newname"'/')
457.     done
```

```
458.     echo "作业修改成功!"
459.     manage_homework
460.     #函数错误返回, 退出错误码 1
461.     exit 1
462. }
463.
464. #列举所有作业
465. list_homework()
466. {
467.     #操作提示信息
468.     echo ""
469.     studir=$(ls -d data/stu_course/*$cid | head -1)
470.     echo "所有作业为:"
471.     declare -i i=1
472.     #遍历该作业
473.     for homework in $(ls $studir | sort)
474.     do
475.         echo $i $homework
476.         i=i+1
477.     done
478.     return 0
479. }
480.
481. #检查作业完成情况
482. check_homework()
483. {
484.     #列举作业信息
485.     list_homework
486.     echo ""
487.     echo -n "请输入你要检查的序号-> "
488.     read num
489.     #作业名
490.     wname=$(ls $studir | head -n $num)
491.     echo "完成情况如下:"
492.     #遍历选课学生
493.     for d in data/stu_course/*$cid/$wname
494.     do
495.         #切割学生学号
496.         sid=$(echo $d | cut -d '/' -f 3 | cut -d '_' -f 1)
497.         #检测是否提交
498.         if test 0 != $(ls $d | wc -l)
499.         then
500.             echo $sid "已提交"
501.         else
```

```
502.             echo $sid "未提交"
503.         fi
504.     done
505.     #返回
506.     manage_homework
507.     #函数错误返回,退出错误码 1
508.     exit 1
509. }
```

➤ stu.sh

```
1. #文件名:stu.sh
2. #作者:王子腾 3180102173
3. #创建日期:2020-07-28
4. #最后修改:2020-07-28
5.
6. #!/bin/bash
7.
8. #学生主入口
9. student()
10. {
11.     echo -n "请输入学号-> "
12.     read sid
13.     #检测是否存在该学号
14.     if test 0 = $(cat data/info/student | grep $sid | wc -l)
15.     then
16.         echo "用户不存在"
17.         exit 1
18.     fi
19.     #进入登陆功能
20.     login_student
21.     echo "欢迎同学登陆!"
22.     #进入主界面
23.     operation_student
24.     #函数因未知错误返回,退出码 1
25.     exit 1
26. }
27.
28. #学生登陆功能
29. login_student()
30. {
31.     #获取该学号密码
32.     passwd=$(cat data/passwd | grep "^s$sid" | awk '{print $2}')
33.     #5 次试错机会
34.     for((i=0; i<5; i++))
35.     do
```

```

36.      echo -n "登录密码-> "
37.      read input
38.      #检测输入是否正确
39.      if test $passwd = $input
40.      then
41.          #正确, 返回
42.          return 0
43.      else
44.          #错误提示
45.          j=$((4-$i));
46.          echo "密码错误! 您还有 $j 次机会"
47.      fi
48.  done
49.  #连续五次失败, 退出
50.  exit 2
51.}
52.
53.#学生操作主界面
54.operation_student()
55.{
56.    echo ""
57.    echo "请选择课程:"
58.    #声明整数变量
59.    declare -i i=1
60.
61.    #获取所有选课信息
62.    for c in $(ls -d data/stu_course/$sid* | cut -d "/" -
        f 3 | cut -d "_" -f 2 | sort)
63.    do
64.        #格式化输出
65.        echo $i $c $(cat data/info/course | grep "^$c" | awk '{pri
        nt $2}')
66.        i=i+1
67.    done
68.    read num
69.
70.    #检测输入 num 是否合法
71.    if test $num -le $(ls -d data/stu_course/$sid* | wc -l)
72.    then
73.        #获取课程号 课程名
74.        cid=$(ls -d data/stu_course/$sid* | cut -d "/" -
        f 3 | cut -d "_" -f 2 | sort | sed -n $num"p")
75.        cname=$(cat data/info/course | grep "^$cid" | awk '{print
        $2}')

```



```
76.         manage_course
77.         #函数因未知错误返回,退出码 1
78.         exit 1
79.     else
80.         echo "输入错误!"
81.         operation_student
82.         #函数因未知错误返回,退出码 1
83.         exit 1
84.     fi
85.}
86.
87.#管理选课信息
88.manage_course()
89.{
90.    #操作菜单提示
91.    echo ""
92.    echo "请选择操作:"
93.    echo "1 上传作业"
94.    echo "2 编辑作业"
95.    echo "3 查看完成情况"
96.    echo "r 返回上一层"
97.    echo "q 退出系统"
98.    read op
99.    case $op in
100.        1) add_hw;;
101.        2) mod_hw;;
102.        3) show_hw
103.            manage_course;;
104.        r) operation_student;;
105.        q) exit 0;;
106.        #其他输入, 错误
107.        *) echo "错误输入!"
108.            operation_student;;
109.    esac
110.    #函数因未知错误返回,退出码 1
111.    exit 1
112.}
113.
114.#提交作业文件
115.add_hw()
116.{
117.    #展示作业
118.    show_hw
119.    echo ""
```

```
120. echo -n "请选择要提交的作业-> "
121. #作业序号
122. read num
123. #检测输入是否合法
124. if test $num -le $(ls $d | wc -l)
125. then
126.     #作业名
127.     wname=$(ls $d | sort | sed -n $num"p")
128.     echo -n "请输入要提交文件的路径-> "
129.     read fname
130.     #拷贝该文件至提交作业文件夹
131.     cp $fname $d/$wname
132.     echo "提交成功!"
133.     manage_course
134.     #函数因未知错误返回,退出码 1
135.     exit 1
136. else
137.     echo "输入错误!"
138.     manage_course
139.     #函数因未知错误返回,退出码 1
140.     exit 1
141. fi
142.}
143.
144.#修改作业
145.mod_hw()
146.{
147.    #展示作业
148.    show_hw
149.    echo ""
150.    echo -n "请选择要修改的作业-> "
151.    read num
152.    #检测输入合法性
153.    if test $num -le $(ls $d | wc -l)
154.    then
155.        #获取作业名
156.        wname=$(ls $d | sort | sed -n $num"p")
157.        echo -n "请输入要提交的文件路径-> "
158.        read fname
159.        #删除旧作业
160.        rm $d/$wname/*
161.        #加入新作业
162.        cp $fname $d/$wname
163.        echo "提交成功!"
```

```
164.         manage_course
165.         #函数因未知错误返回,退出码 1
166.         exit 1
167.     else
168.         echo "输入错误!"
169.         manage_course
170.         #函数因未知错误返回,退出码 1
171.         exit 1
172.     fi
173. }
174.
175. #检查作业完成情况
176. show_hw()
177. {
178.     #获取所在目录
179.     d=$(ls -d data/stu_course/*$cid | head -1)
180.     echo ""
181.     echo "该课程任务为:"
182.     #声明整数变量
183.     declare -i i=1
184.     #列举作业
185.     for hw in $(ls $d | sort)
186.     do
187.         echo -n $i $hw " "
188.         #检测是否提交作业
189.         if test 0 = $(ls data/stu_course/$sid'_'$cid/$hw | wc -
190.         1)
191.         then
192.             echo "未提交"
193.         else
194.             echo "已提交"
195.         fi
196.         #步进
197.         i=i+1
198.     done
199. }
```

实验结果

管理员:

```

ziteng@Ziteng-VBox:~/小学期/lab2/mysystem$ bash entrance.sh
选择权限:
1 管理员
2 教师
3 学生
q 退出
1
管理员密码-> 123123
密码错误! 您还有 4 次机会
管理员密码-> 123123
密码错误! 您还有 3 次机会
管理员密码-> 000000
欢迎admin登入

请选择操作:
1 管理教师信息
2 管理课程信息
q 退出系统

```

测试登陆验证功能

```

请选择操作:
1 管理教师信息
2 管理课程信息
q 退出系统
1

请选择具体操作:
1 增加教师信息
2 删除教师信息
3 修改教师信息
4 查询教师信息
r 返回上一层
q 退出系统

```

教师管理界面

```

请选择具体操作:
1 增加教师信息
2 删除教师信息
3 修改教师信息
4 查询教师信息
r 返回上一层
q 退出系统
4

请选择查询方式:
1 按教师工号查询
2 按教师姓名查询
3 列举所有教师
r 返回上一层
3
001 Jack

```

列举(list)查询功能

```
请选择查询方式:
1 按教师工号查询
2 按教师姓名查询
3 列举所有教师
r 返回上一层
1

请输入教师工号-> 003
工号不存在!

请选择查询方式:
1 按教师工号查询
2 按教师姓名查询
3 列举所有教师
r 返回上一层
1

请输入教师工号-> 001
查询结果为:
001 Jack
```

教师工号查询功能

```
请选择查询方式:
1 按教师工号查询
2 按教师姓名查询
3 列举所有教师
r 返回上一层
2

请输入教师姓名-> Linda
姓名不存在!

请选择查询方式:
1 按教师工号查询
2 按教师姓名查询
3 列举所有教师
r 返回上一层
2

请输入教师姓名-> Jack
查询结果为:
001 Jack
```

教师姓名查询功能

```
请选择具体操作:
1 增加教师信息
2 删除教师信息
3 修改教师信息
4 查询教师信息
r 返回上一层
q 退出系统
1

请输入教师工号-> 003
请输入教师姓名-> Linda
请输入新账户密码-> 003
添加成功! 新教师账户为003 Linda
```

增加教师信息功能

```
请选择具体操作:
1 增加教师信息
2 删除教师信息
3 修改教师信息
4 查询教师信息
r 返回上一层
q 退出系统
2

请输入教师工号-> 003
删除成功!
```

删除教师信息功能

```
请选择具体操作:
1 增加教师信息
2 删除教师信息
3 修改教师信息
4 查询教师信息
r 返回上一层
q 退出系统
3

请输入教师工号-> 001
请输入修改后教师工号-> 003
请输入修改后教师姓名-> Robert
修改成功! 新账户为:003 Robert
```

修改教师信息功能

```
请选择操作:
1 管理教师信息
2 管理课程信息
q 退出系统
2

请选择具体操作:
1 增加课程信息
2 删除课程信息
3 修改课程信息
4 绑定课程信息
5 解除课程绑定
r 返回上一层
q 退出系统
1
```

课程管理界面

```
请选择具体操作:
1 增加课程信息
2 删除课程信息
3 修改课程信息
4 绑定课程信息
5 解除课程绑定
r 返回上一层
q 退出系统
1

请输入课程代号-> 2
请输入课程名称-> Linux
添加成功! 新课程为2 Linux
```

增加课程信息功能

```
请选择具体操作:
1 增加课程信息
2 删除课程信息
3 修改课程信息
4 绑定课程信息
5 解除课程绑定
r 返回上一层
q 退出系统
3

请输入课程代号-> 2
请输入修改后课程代号-> 3
请输入修改后课程名称-> OS
修改成功! 新课程为:3 OS
```

修改课程信息功能

```
请选择具体操作:
1 增加课程信息
2 删除课程信息
3 修改课程信息
4 绑定课程信息
5 解除课程绑定
r 返回上一层
q 退出系统
2

请输入课程代号-> 3
删除成功!
```

删除课程信息功能

```
请输入课程代号-> 1
请输入课程名称-> Math
添加成功! 新课程为1 Math
```

```
请选择具体操作:
1 增加课程信息
2 删除课程信息
3 修改课程信息
4 绑定课程信息
5 解除课程绑定
r 返回上一层
q 退出系统
4

请输入课程代号-> 1
请输入教师工号-> 003
绑定成功! 1 -- 003
```

绑定课程功能

```
请选择具体操作:
1 增加课程信息
2 删除课程信息
3 修改课程信息
4 绑定课程信息
5 解除课程绑定
r 返回上一层
q 退出系统
5

请输入课程代号-> 1
请输入教师工号-> 003
解除绑定成功! 1 !- 003
```

解除绑定功能

教师:

```
ziteng@Ziteng-VBox:~/小学期/lab2/mysystem$ bash entrance.sh
选择权限:
1 管理员
2 教师
3 学生
q 退出
2
请输入教师工号-> 003
登录密码-> 123
密码错误! 您还有 4 次机会
登录密码-> 003
欢迎老师登陆!
```

教师登陆功能

```
请选择课程：
1 1 Math
1

请选择进一步操作
1 管理学生信息
2 管理课程信息
3 管理作业实验
r 返回上一级
q 退出
```

课程选择与操作界面

```
请选择具体操作
1 创建学生账户
2 修改学生账户
3 删除学生账户
4 查找选课学生
r 返回上一级
q 退出
1

请输入新建学生学号-> 3180102173
请输入学生姓名-> 王子腾
请输入账户密码-> 2173
```

创建选课学生（新账户）

```
请选择具体操作
1 创建学生账户
2 修改学生账户
3 删除学生账户
4 查找选课学生
r 返回上一级
q 退出
1

请输入新建学生学号-> 3180102173
添加成功！
```

创建选课学生（已有账户）

```
请选择具体操作
1 创建学生账户
2 修改学生账户
3 删除学生账户
4 查找选课学生
r 返回上一级
q 退出
4

请输入查询学生学号-> 31801021
该学生未选课！

请选择具体操作
1 创建学生账户
2 修改学生账户
3 删除学生账户
4 查找选课学生
r 返回上一级
q 退出
4

请输入查询学生学号-> 3180102173
学生信息为：
3180102173 王子腾
```

查找选课学生


```
请选择具体操作
1 创建学生账户
2 修改学生账户
3 删除学生账户
4 查找选课学生
r 返回上一级
q 退出
2

请输入修改学生学号-> 3180102173
请输入修改后学生学号-> 3180102173
请输入修改后学生姓名-> wzt
修改成功! 新账户为:3180102173 wzt
```

修改学生账户

```
请选择具体操作
1 创建学生账户
2 修改学生账户
3 删除学生账户
4 查找选课学生
r 返回上一级
q 退出
3

请输入删除学生学号-> 3180102173
删除成功!
```

删除学生账户

```
请选择进一步操作
1 管理学生信息
2 管理课程信息
3 管理作业实验
r 返回上一级
q 退出
2

请选择具体操作
1 新建课程信息
2 删除课程信息
3 编辑课程信息
4 显示课程信息
r 返回上一级
q 退出
1

请输入课程信息-> 本门课程考试时间定为2020.8.30, 望周知!!
信息添加成功!
```

新建课程信息

```
请选择具体操作
1 新建课程信息
2 删除课程信息
3 编辑课程信息
4 显示课程信息
r 返回上一级
q 退出
4

课程考试时间为2020.8.30, 望周知!!
大家复习
```

显示课程信息

```

请选择具体操作
1 新建课程信息
2 删除课程信息
3 编辑课程信息
4 显示课程信息
r 返回上一级
q 退出
3

课程考试时间为2020.8.30，望周知!!
大家复习
请输入你要修改的行数-> 2
请输入你要修改的内容-> 认真复习
修改成功!

请选择具体操作
1 新建课程信息
2 删除课程信息
3 编辑课程信息
4 显示课程信息
r 返回上一级
q 退出
4

课程考试时间为2020.8.30，望周知!!
认真复习

```

编辑课程信息

```

请选择具体操作
1 新建课程信息
2 删除课程信息
3 编辑课程信息
4 显示课程信息
r 返回上一级
q 退出
2
文本内容为:
课程考试时间为2020.8.30，望周知!!
认真复习

请输入你要删除的行数-> 2
删除成功!

请选择具体操作
1 新建课程信息
2 删除课程信息
3 编辑课程信息
4 显示课程信息
r 返回上一级
q 退出
4

课程考试时间为2020.8.30，望周知!!

```

删除课程信息

```

请选择进一步操作
1 管理学生信息
2 管理课程信息
3 管理作业实验
r 返回上一级
q 退出
3

请选择具体操作
1 新建作业信息
2 删除作业信息
3 编辑作业信息
4 显示作业信息
5 查看完成情况
r 返回上一级
q 退出

```

管理作业实验界面

```
请选择具体操作
1 新建作业信息
2 删除作业信息
3 编辑作业信息
4 显示作业信息
5 查看完成情况
r 返回上一级
q 退出
1

请输入作业名称 -> hw1
作业创建成功!
```

创建作业信息

```
请选择具体操作
1 新建作业信息
2 删除作业信息
3 编辑作业信息
4 显示作业信息
5 查看完成情况
r 返回上一级
q 退出
4

所有作业为:
1 hw1
```

显示作业信息

```
请选择具体操作
1 新建作业信息
2 删除作业信息
3 编辑作业信息
4 显示作业信息
5 查看完成情况
r 返回上一级
q 退出
3

所有作业为:
1 hw1
请输入你要修改的序号 -> 1
请输入新名称 -> HW1
作业修改成功!
```

修改作业信息

```
请选择具体操作
1 新建作业信息
2 删除作业信息
3 编辑作业信息
4 显示作业信息
5 查看完成情况
r 返回上一级
q 退出
2

所有作业为:
1 HW1

请输入你要删除的序号 -> 1
作业删除成功!
```

删除作业信息

```
请选择具体操作
1 新建作业信息
2 删除作业信息
3 编辑作业信息
4 显示作业信息
5 查看完成情况
r 返回上一级
q 退出
5
```

所有作业为:

1 hw1

请输入你要检查的序号-> 1

完成情况如下:

3180102173 未提交

查看学生完成情况

学生:

```
ziteng@Ziteng-VBox:~/小学期/lab2/mysystem$ ./entrance.sh
```

选择权限:

1 管理员

2 教师

3 学生

q 退出

3

请输入学号-> 3180102173

登录密码-> 000

密码错误!您还有 4 次机会

登录密码-> 2173

欢迎同学登陆!

登录验证功能

请选择课程:

1 1 Math

1

请选择操作:

1 上传作业

2 编辑作业

3 查看完成情况

r 返回上一层

q 退出系统

1

操作界面

请选择操作:

1 上传作业

2 编辑作业

3 查看完成情况

r 返回上一层

q 退出系统

1

该课程任务为:

1 hw1 未提交

请选择要提交的作业-> 1

请输入要提交文件的路径-> ../1.sh

提交成功!

上传作业

```
请选择操作:
1 上传作业
2 编辑作业
3 查看完成情况
r 返回上一层
q 退出系统
3
```

```
该课程任务为:
1 hw1 已提交
```

查看作业完成情况

```
请选择操作:
1 上传作业
2 编辑作业
3 查看完成情况
r 返回上一层
q 退出系统
2
```

```
该课程任务为:
1 hw1 已提交
```

```
请选择要修改的作业-> 1
请输入要提交的文件路径-> ../2.sh
提交成功!
```

编辑作业

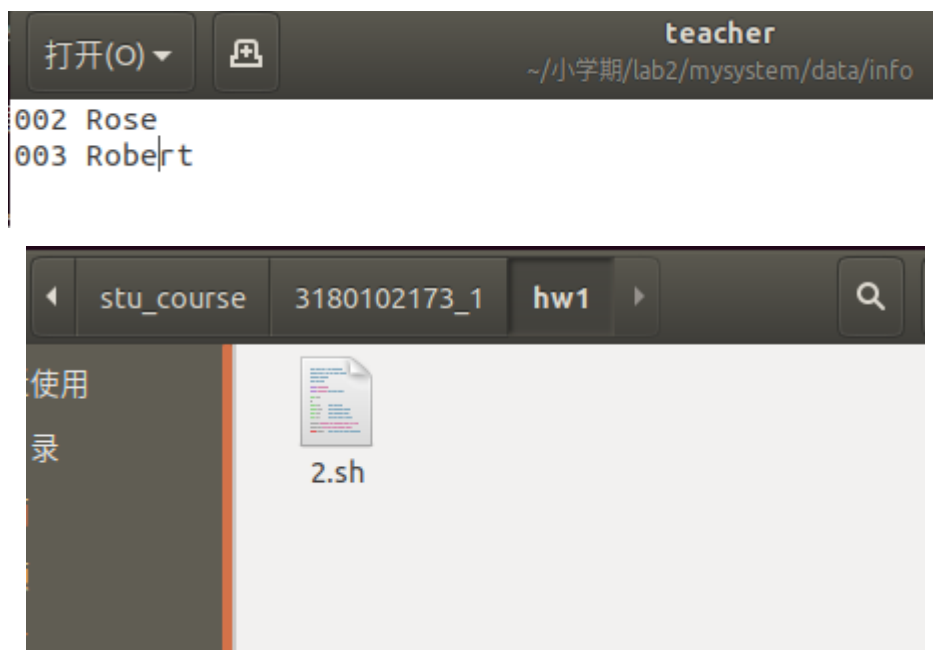
```
ziteng@Ziteng-VBox:~/小学期/lab2/mysystem$ tree data
data
├── bind
│   └── 003_1
├── info
│   ├── course
│   ├── student
│   └── teacher
├── passwd
└── stu_course
    └── 3180102173_1
        └── hw1
            └── 2.sh

5 directories, 6 files
```

最终数据结构

```
passwd
~/小学期/lab2/mysystem/data

a 000000
t003 003
t002 002
s3180102173 2173|
```



部分数据文件截图

5.1（30 分）使用任何一种程序设计语言实现一个 **shell** 程序的基本功能。

shell 或者命令行解释器是操作系统中最基本的用户接口。写一个简单的 **shell** 程序——**myshell**，它具有以下属性：

1. 这个 **shell** 程序必须支持以下内部命令：**bg**、**cd**、**clr**、**dir**、**echo**、**exec**、**exit**、**environ**、**fg**、**help**、**jobs**、**pwd**、**quit**、**set**、**shift**、**test**、**time**、**umask**、**unset**。

2. 其他的命令行输入被解释为程序调用，**shell** 创建并执行这个程序，并作为自己的子进程。程序的执行的环境变量包含一下条目：

parent=<pathname>/**myshell**。

3. **shell** 必须能够从文件中提取命令行输入，例如 **shell** 使用以下命令行被调用：

myshell batchfile

这个批处理文件应该包含一组命令集，当到达文件结尾时 **shell** 退出。很明显，如果 **shell** 被调用时没有使用参数，它会在屏幕上显示提示符请求用户输入。

4. **shell** 必须支持 I/O 重定向，**stdin** 和 **stdout**，或者其中之一，例如命令行为：

programname arg1 arg2 < inputfile > outputfile

使用 **arg1** 和 **arg2** 执行程序 **programname**，输入文件流被替换为 **inputfile**，输出文件流被替换为 **outputfile**。

stdout 重定向应该支持以下内部命令：**dir**、**environ**、**echo**、**help**。

使用输出重定向时，如果重定向字符是>，则创建输出文件，如果存在则覆盖之；如果重定向字符为>>，也会创建输出文件，如果存在则添加到文件尾。

5. shell 必须支持后台程序执行。如果在命令行后添加&字符，在加载完程序后需要立刻返回命令行提示符。
6. 必须支持管道（“|”）操作。
7. 命令行提示符必须包含当前路径。

设计文档

➤ 设计思想

myshell 系统采用 C++ 语言开发而成，采用了面向对象和结构化的思路，将进程等过程信息定义在 STL 内，通过对对象的操作来获取结构内信息。由于 C++ 为面向对象的高级语言，而 Shell 又是基于操作系统内核和交互层的功能设计，因此一些文件和定向类的变量、函数无法直接应用于底层交互，在系统设计中用到了一些 Linux 的系统调用，如 open 指令。

对于命令的调用，myshell 将内部和外部命令区分开，在 Command.cpp 文件中定义了 ls、pwd、cd、time 等内部命令的实现方法，对于外部命令，则通过提供解析环境和运行环境，并转化为内部命令处理。在命令运行过程中，通过创建子进程来执行单条命令，执行过程中主进程与子进程不保持同步，各自运行状态根据命令的前后台转换进行匹配：若命令为前台执行，则采用同步机制，即主进程暂停，子进程执行该命令并返回完成信号，主进程接收到信号后继续运行，此时子进程被关闭，在这个过程中，只有一个进程在实际运行，为同步机制；若命令为后台执行，则主进程与子进程同时运行，当子进程执行完毕并向主进程返回完成信号时，根据主进程状态执行相应的信号处理函数，并关闭子进程，此过程中两个进程同时执行，从而达到后台处理命令的效果，为异步机制。

对于管道和重定向的操作，myshell 通过在命令结构中定义 pipein、pipeout、indirect、outdirect 等变量判断当前命令的管道和重定向情况，并根据重定向文件进行文件描述符的操作，达到流转换的目的。

➤ 功能模块

本系统主要由 4 个模块组成，分别为主入口模块、命令执行模块、运行环境模块、命令解析模块组成。

Main.cpp 和 Main.h 构成主入口模块，其中规定了程序错误码、程序限定值、错误处理机制等必要函数实现和宏值定义，并提供了 myshell 的入口，为整个系统的 API 部分。

Command.cpp 和 Command.h 内定义了 myshell 允许的 19 条内部命令的实现，当子进程接受到解析模块传来的参数或调用时，将执行相关操作以实现命令的基本功能，本部分不涉及主进程和子进程的转换，仅为功能实现部分。

Environ.cpp 和 Environ.h 为系统运行环境模块，其中定义了进程的结构体、进程组 vector 结构、环境变量 envvar 的 map 映射结构等信息，并提供了进程初始化、进程查找与删除等功能，用以在 API 部分提供进程支持和环境搭建。

Parse.cpp 和 Parse.h 为命令解析模块，也是系统最复杂的模块，在这一部分中定义了命令结构体，包括进程、重定向、管道等信息，创立了命令组 vector 结构，并提供了命令行界面的主要功能和进程管理功能，包括命令行/文件初始化、信号初始化、命令解析、子进程控制、Ctrl-Z 处理等功能，通过与主入口模块和命令执行模块进行交互，为系统的中间 API 层。

➤ 数据结构

myshell 的数据信息采用结构体、枚举和 STL 实现，对于单个的进程、命令、参数，都采用结构体的方式定义；对解析状态、重定向方式、进程方式、进程状态等状态信息，则采用枚举的方式实现；对环境变量、进程列表、命令列表、参数列表等结构体的集合，采用 map 和 vector 等 STL 实现。

➤ 算法

内部命令的实现通过对环境模块和解析模块的状态量进行修改和输出，达到实现命令的目的，对于命令的解析，parse 模块通过正则表达式和字符串分割函数将当前命令的参数、管道、重定向等信息存储

在命令结构体内，并根据命令名调用命令执行模块。

对于进程操作，主进程在调用子进程时先向环境模块写入子进程信息，完成进程列表的更新，在子进程结束后根据返回信号执行进程列表删除更新，达到进程控制的目的。

为丰富显示效果，采用了宏定义的方式结合 `cout` 进行颜色切换，同时仿照 `ubuntu` 模式将目录文件、普通文件在显示时用不同的颜色区分。

源代码

➤ Main.h

```
1.  /**
2.   * 文件: Main.h
3.   * 作者: 王子腾 3180102173
4.   * 创建时间: 2020/7/15
5.   * 最后修改: 2020/8/5
6.   */
7.
8.  // 预编译保护
9.  #ifndef MAIN_H
10. #define MAIN_H
11.
12. // 系统调用库
13. #include<unistd.h>
14. #include<dirent.h>
15. #include<fcntl.h>
16. #include<sys/stat.h>
17. #include<sys/shm.h>
18. #include<termios.h>
19.
20. // C 标准库
21. #include<iostream>
22. #include<string>
23. #include<string.h>
24. #include<time.h>
25. #include<limits>
26. #include<signal.h>
27. #include<vector>
28. #include<map>
29.
30. using namespace std;
31.
```

```
32. // myshell 名称
33. #define MYSHELL "myshell"
34.
35. // 错误码
36. #define FILE_FAILED 200
37. #define DIR_FAILED 201
38. #define CD_FAILED 202
39. #define DUP_FAILED 203
40. #define FORK_FAILED 204
41.
42. // 程序限定值
43. // 环境变量
44. #define MAX_ENVNUM 64
45. #define MAX_ENVLEN 16
46. // 单条命令最大长度
47. #define MAX_COMLEN 256
48. // 最大进程数
49. #define MAX_JOBNUM 8
50. // 路径名最大长度
51. #define MAX_PATHLEN 256
52.
53. // 轮询时每次暂停时间
54. #define WAIT_TIME 500
55.
56. // Linux 控制台输出字体参数
57. #define NORMAL "\033[0m"
58. #define RED "\033[31m"
59. #define GREEN "\033[32m"
60. #define YELLOW "\033[33m"
61. #define BLUE "\033[34m"
62. #define PURPLE "\033[35m"
63.
64. // 显示错误信息并结束程序
65. void errorExit(string ErroInfo, int errorNo){
66.     cerr << RED << ErroInfo << NORMAL << endl;
67.     exit(errorNo);
68. }
69.
70. // 调试状态,用户不需用
71. void debug(int i){
72.     cerr << RED << "debug " << i << endl;
73. }
74.
75. #endif
```

➤ Main.cpp

```
1.  /**
2.   * 文件: Main.cpp
3.   * 作者: 王子腾 3180102173
4.   * 创建时间: 2020/8/5
5.   * 最后修改: 2020/8/7
6.   */
7.
8.  #include "Parse.h"
9.  #include "Environ.h"
10.
11. //主入口函数
12. int main(int argc, char* argv[]){
13.     //调用环境初始化函数
14.     envInit();
15.
16.     //根据参数个数选择执行模式
17.     if(argc==1)
18.         //不含参数,执行命令行模式
19.         init_commandline();
20.     else
21.         //含参数,解析文件命令
22.         init_file(argv[1]);
23. }
```

➤ Command.h

```
1.  /**
2.   * 文件: Command.h
3.   * 作者: 王子腾 3180102173
4.   * 创建时间: 2020/7/25
5.   * 最后修改: 2020/8/14
6.   */
7.
8. //预编译保护
9. #ifndef COMMAND_H
10. #define COMMAND_H
11.
12. #include "Parse.h"
13.
14. /**
15.  * 内部命令集
16.  * 1. 无参数命令
17.  * 2. 含参命令
18.  */
```

```
19. // 无参数命令
```

```
20. void bg();
```

```
21. void fg();
```

```
22. void clr();
```

```
23. void exit();
```

```
24. void jobs();
```

```
25. void env();
```

```
26. void time();
```

```
27. void pwd();
```

```
28. void quit();
```

```
29. // 含参数命令
```

```
30. void cd(cmdarg&);
```

```
31. void dir(cmdarg&);
```

```
32. void echo(cmdarg&);
```

```
33. void exec(cmdarg&);
```

```
34. void help(cmdarg&);
```

```
35. void set(cmdarg&);
```

```
36. void unset(cmdarg&);
```

```
37. void umask(cmdarg&);
```

```
38. void test(cmdarg&);
```

```
39. void shift(cmdarg&);
```

```
40.
```

```
41. #endif
```

➤ Command.cpp

```
1. /**
```

```
2.  * 文件: Command.cpp
```

```
3.  * 作者: 王子腾 3180102173
```

```
4.  * 创建时间: 2020/7/25
```

```
5.  * 最后修改: 2020/8/14
```

```
6. */
```

```
7.
```

```
8. #include "Command.h"
```

```
9. #include "Parse.h"
```

```
10. #include "Environ.h"
```

```
11.
```

```
12. //bg 命令的实现,将阻塞的进程转到后台执行
```

```
13. void bg(){
```

```
14.     // 搜索 1-Max_JOBNUM 的进程
```

```
15.     for(int i = 1; i < MAX_JOBNUM; i++){
```

```
16.         // 遍历进程表
```

```
17.         if(joblist.at(i).pid && joblist.at(i).type == BG && joblist.at(i).status == SUSPEND){
```

```
18.             // 将状态改为运行中
```

```
19.             joblist.at(i).status = RUN;
```

```
20.         // 继续执行
21.         kill(joblist.at(i).pid, SIGCONT);
22.         // 反馈
23.         cout << joblist.at(i).pid << "\t" << joblist.at(i).name << "\tRun
    " << endl;
24.     }
25. }
26.
27. //bg 命令的实现,将后台进程转到前台执行
28. void fg(){
29.     //检查第一个用户进程
30.     if(joblist.at(2).pid){
31.         // 反馈
32.         cout << joblist.at(2).pid << "\t" << joblist.at(2).name << "\tRun" <<
            endl;
33.         // 暂停
34.         while(true);
35.     }
36.     else
37.         // 进程不存在
38.         cout << "No Current Background Job" << endl;
39. }
40.
41.
42. //clr 命令的实现,即 clear
43. void clr(){
44.     // 屏幕操作符
45.     cout << "\033[2J\033[0;0H";
46. }
47.
48.
49. //exit 命令的实现,退出 myshell
50. void exit(){
51.     //设置当前 myshell 状态,用于 parse 模块解析
52.     envvar["status"] = "exit";
53. }
54.
55.
56. //jobs 命令的实现,打印进程表
57. void jobs(){
58.     //遍历进程表
59.     for (int i = 0; i < MAX_JOBNUM; i++)
60.     {
61.         //后台进程
```

```
62.         if(joblist.at(i).pid && joblist.at(i).type == BG)
63.             // RUN 运行
64.             if(joblist.at(i).status)
65.                 cout << joblist.at(i).pid << "\t" + joblist.at(i).name + "\tRun" << endl;
66.             // SUSPEND 挂起
67.             else
68.                 cout << joblist.at(i).pid << "\t" + joblist.at(i).name + "\tSuspend" << endl;
69.         }
70.
71. }
72.
73.
74. //environ 命令的实现,打印环境变量
75. void enviro(){
76.     //遍历环境向量 envvar
77.     for(auto i = envvar.begin(); i != envvar.end(); i++)
78.         cout << i->first << " = " << i->second << endl;
79. }
80.
81.
82. //time 命令的实现,打印时间
83. void time(){
84.     //获取系统时间
85.     time_t curTime = time(NULL);
86.     //打印
87.     cout << ctime(&curTime) << endl;
88. }
89.
90.
91. //pwd 命令的实现,打印当前目录
92. void pwd(){
93.     //打印 pwd 对应变量的值
94.     cout << envvar["pwd"] << endl;
95. }
96.
97.
98. //quit 命令的实现,同 exit
99. void quit(){
100.    //设置环境变量 status
101.    envvar["status"] = "exit";
102. }
103.
```

```

104.
105. //cd 命令的实现,切换当前目录
106. void cd(cmdarg& arg){
107.     //若参数不为空
108.     if(arg.cmdargu.size())
109.         //系统调用,切换目录
110.         if(chdir(arg.cmdargu.at(0).c_str()))
111.             //切换失败
112.             errorExit("Change directory", CD_FAILED);
113.         //切换成功
114.     else
115.         getcwd((char*)envvar["pwd"].c_str(), _POSIX_PATH_MAX);
116.
117.     //参数为空,调用 pwd
118.     pwd();
119. }
120.
121.
122. //dir 命令的实现,打印参数目录下的文件
123. void dir(cmdarg& arg){
124.     string dirName;
125.     //参数个数为零
126.     if(arg.cmdargu.empty())
127.         dirName = envvar["pwd"];
128.     //系统调用,开启目录
129.     DIR* dp = opendir(dirName.c_str());
130.     //调用失败
131.     if(!dp)
132.         errorExit("Directory Open", DIR_FAILED);
133.
134.     // 获取目录信息
135.     struct dirent * dirp;
136.     while((dirp = readdir(dp))){
137.         //仿 Ubuntu,忽略隐藏文件
138.         if(dirp->d_name[0]=='.')
139.             continue;
140.         //仿 Ubuntu 换色输出
141.         switch (dirp->d_type)
142.         {
143.             //普通文件
144.             case DT_REG:
145.                 cout << NORMAL << dirp->d_name << NORMAL << " ";
146.                 break;
147.             //目录文件

```

```

148.         case DT_DIR:
149.             cout << BLUE << dirp->d_name << NORMAL << " ";
150.             break;
151.             //链接文件
152.         case DT_LNK:
153.             cout << PURPLE << dirp->d_name << NORMAL << " ";
154.             break;
155.             //其他文件
156.         default:
157.             cout << dirp->d_name << " ";
158.             break;
159.     }
160.     cout << endl;
161. }
162. //关闭目录
163. closedir(dp);
164. //清理缓冲区占用
165. fflush(stdout);
166.}
167.
168.
169. //echo 命令的实现,输出字符串
170. void echo(cmdarg& arg){
171.     //遍历参数表
172.     for(auto i = arg.cmdargu.begin(); i != arg.cmdargu.end(); i++)
173.         cout << *i + " " ;//空格分割
174.     //换行
175.     cout << endl;
176.}
177.
178.
179. //exec 命令的实现,运行单条指令替代 myshell 进程
180. void exec(cmdarg& arg){
181.     //有参数
182.     if(arg.cmdargu.size())
183.     {
184.         //建立 exec 后的单条指令
185.         command newcmd;
186.         //指令参数
187.         newcmd.cmdname = arg.cmdargu[0];
188.         newcmd.indirect = NOREDIRECT;
189.         newcmd.outdirect = NOREDIRECT;
190.         newcmd.isbg = false;
191.         //配备详细参数表

```



```
192.         newcmd.arglist.cmdargu.assign(arg.cmdargu.begin()+1, arg.cmdargu.end
    ());
193.         newcmd.arglist.option.assign(arg.option.begin(), arg.option.end());

194.         newcmd.arglist.optargu.assign(arg.optargu.begin(), arg.optargu.end()
    );
195.
196.         //执行该指令
197.         excute(newcmd);
198.         //结束后退出 shell
199.         exit();
200.     }
201. }
202.
203.
204. //set 命令的实现,打印/修改环境变量
205. void set(cmdarg& arg){
206.     //无参数,打印环境变量
207.     if(arg.cmdargu.empty()){
208.         envir();
209.         return;
210.     }
211.     //修改/添加环境变量
212.     string name = arg.cmdargu[0];
213.     envvar[name] = arg.cmdargu[1];
214. }
215.
216.
217. //unset 命令的实现,消除环境变量
218. void unset(cmdarg& arg){
219.     //无参数
220.     if(arg.cmdargu.empty()){
221.         return;
222.     }
223.     //设置环境变量
224.     string name = arg.cmdargu[0];
225.     //pwd 和 status 作为必要环境变量,不可修改
226.     if(name == "pwd" || name == "status")
227.         return;
228.     //删除
229.     envvar.erase(name);
230. }
231.
232.
```

```

233. //umask 命令的实现,打印/设置 umask 值
234. void umask(cmdarg& arg){
235.     //无参数,打印当前 umask
236.     if(arg.cmdargu.empty())
237.     {
238.         cout << envvar["umask"];
239.         return;
240.     }
241.     //获取 umask 值
242.     string mask = arg.cmdargu[0];
243.     mask = (mode_t)strtol(mask.c_str(), NULL, 8);
244.     //设置 umask 值
245.     envvar["umask"]=mask;
246. }
247.
248.
249. //test 命令的实现,判断测试条件
250. void test(cmdarg& arg){
251.     //检查参数信息
252.     switch(arg.option[0])
253.     {
254.         //判断字符串是否存在字符
255.         case 'f':
256.             if(arg.cmdargu.size())
257.                 cout << "true" << endl;
258.             else
259.                 cout << "false" << endl;
260.         //是否为空字符串
261.         case 'n':
262.             if(!arg.cmdargu.size())
263.                 cout << "true" << endl;
264.             else
265.                 cout << "false" << endl;
266.         break;
267.     }
268. }
269.
270.
271. //shift 命令的实现,用于切换参数顺序(默认左移)
272. void shift(cmdarg& arg){
273.     //输入参数转换为数字
274.     int bit = atoi(arg.cmdargu.at(1).c_str());
275.     //读入字符串
276.     char line[MAX_COMLEN];

```

```

277.     char* line_p = line;
278.     fgets(line, MAX_COMLEN, stdin);
279.     //左移
280.     for (int i = 0; i < bit; ++i)
281.     {
282.         //连续空格和 tab,跳过
283.         while (' ' == *line_p || '\t' == *line_p)
284.             ++line_p;
285.         //插入分割空格和 tab
286.         strsep(&line_p, " \t\n");
287.     }
288.     //跳过开头空白符
289.     while (' ' == *line_p || '\t' == *line_p)
290.         ++line_p;
291.     //输出
292.     cout << *line_p;
293. }

```

➤ Environ.h

```

1.  /**
2.   * 文件: Environ.h
3.   * 作者: 王子腾 3180102173
4.   * 创建时间: 2020/7/23
5.   * 最后修改: 2020/8/5
6.   */
7.
8.  //预编译保护
9.  #ifndef ENVIRON_H
10. #define ENVIRON_H
11.
12. #include "Main.h"
13.
14. //进程状态量:枚举
15. //进程执行类型
16. enum jobType { FG, BG };
17. //进程执行状态
18. enum jobStatus { RUN, SUSPEND };
19.
20. //进程结构体
21. typedef struct job
22. {
23.     pid_t pid;    //进程号
24.     string name;  //进程名
25.     jobType type; //进程类型
26.     jobStatus status; //进程状态

```

```

27. } job;
28.
29. //map 环境变量映射
30. map<string, string> envvar;
31.
32. //vector 进程表
33. vector<job> joblist;
34.
35. //环境初始化 API
36. void envInit();
37.
38. //STL 进程表初始化
39. void jobInit();
40.
41. //根据进程号查找相应进程在进程表中的位置
42. int jobFind(pid_t pid);
43.
44. //根据进程号删除进程表中相应进程
45. void jobdel(pid_t pid);
46.
47. //管道信息
48. int pipeFd[2];
49.
50. //mask 变量设置
51. int* mask;
52.
53. //初始化 mask 变量
54. void maskInit();
55.
56. #endif

```

➤ Environ.cpp

```

1. /**
2.  * 文件: Environ.cpp
3.  * 作者: 王子腾 3180102173
4.  * 创建时间: 2020/7/23
5.  * 最后修改: 2020/8/5
6.  */
7.
8. #include "Environ.h"
9.
10.
11. //环境初始化 API
12. void envInit(){
13.     string curDir;

```

```
14. //初始化 pwd 变量
15. getcwd((char*)curDir.c_str(), _POSIX_PATH_MAX);
16. envvar["pwd"] = curDir;
17. //初始化 shell 位置
18. envvar["shell"] = "/usr/local/myshell";
19.
20. //API 调用进程和 mask 初始函数
21. jobInit();
22. maskInit();
23. }
24.
25.
26. //STL 进程表初始化
27. void jobInit(){
28.     job newJob;
29.     //添加系统进程 myshell
30.     newJob.pid = 0;
31.     newJob.name = MYSHELL;
32.     newJob.type = FG;
33.     newJob.status = RUN;
34.     //加入 STL 进程表
35.     joblist.push_back(newJob);
36. }
37.
38.
39. //根据进程号查找相应进程在进程表中的位置
40. int jobFind(pid_t pid)
41. {
42.     int j = 0;
43.     //遍历进程表
44.     for(auto i = joblist.begin(); i != joblist.end(); i++, j++)
45.         if(i->pid == pid)//找到,返回位点
46.             return j;
47.     //未找到
48.     return -1;
49. }
50.
51.
52. //根据进程号删除进程表中相应进程
53. void jobdel(pid_t pid){
54.     //遍历进程表
55.     for(auto i = joblist.begin(); i != joblist.end(); i++)
56.         if(i->pid == pid)//找到,删除
57.             joblist.erase(i);
```

```

58. }
59.
60.
61. //初始化 mask 变量
62. void maskInit(){
63.     //八进制 0022
64.     *mask = 26;
65.     envvar["umask"]="0022";
66. }

```

➤ Parse.h

```

1. /**
2.  * 文件: Parse.h
3.  * 作者: 王子腾 3180102173
4.  * 创建时间: 2020/7/25
5.  * 最后修改: 2020/8/13
6.  */
7.
8. //预编译保护
9. #ifndef PARSE_H
10. #define PARSE_H
11.
12. #include "Main.h"
13.
14. // parse 的不同状态
15. enum parseStatus {
16.     NEWCMD, NEWCMD_PIPE, INREDIRECT, OUTREDIRECT, NEWCMDREADY
17. };
18.
19. // 枚举重定向方式
20. enum redirect { NOREDIRECT, COVER, APPEND };
21.
22. // 存储单条命令参数的结构体,包含 option 以及 option 和 command 的参数
23. typedef struct cmdarg
24. {
25.     vector<char> option;
26.     vector<string> optargu;
27.     vector<string> cmdargu;
28. } cmdarg;
29.
30. // 存储单条命令的结构体
31. typedef struct command
32. {
33.     // 命令名(小写)
34.     string cmdname;

```

```
35. // 参数列表
36. cmdarg arglist;
37. // 是否后台运行
38. bool isbg;
39. // 管道情况
40. bool pipein;
41. bool pipeout;
42. // 重定向情况
43. redirect indirect;
44. redirect outdirect;
45. // 重定向文件
46. string infile;
47. string outfile;
48. } command;
49.
50.
51. //无参数状态下初始化命令行模式
52. void init_commandline();
53.
54. //含参数状态下初始化文件
55. void init_file(string);
56.
57. //打印当前目录
58. void displayDir();
59.
60. //从输入流 is 获取命令
61. bool getCommand(istream &);
62.
63. //将读取的内容解析
64. bool parse(char*);
65.
66. //命令执行
67. void excute(command&);
68.
69. //子进程负责运行命令
70. void childProcess(command&);
71.
72. // 重定向初始化
73. void redirectInit(FILE* src, string fileName, redirect type);
74.
75. // 恢复重定向前的状态
76. void redirectEnd(FILE* dst, int oldFd);
77.
78. //初始化信息处理
```

```

79. void signalInit();
80.
81. //子进程结束函数
82. void childExit(int sig_no, siginfo_t* info, void* vcontext);
83.
84. //处理 ctrlz 信号,挂起当前进程
85. void ctrl_z_Handler(int sig_no);
86. #endif

```

➤ Parse.cpp

```

1. /**
2.  * 文件: Parse.cpp
3.  * 作者: 王子腾 3180102173
4.  * 创建时间: 2020/7/25
5.  * 最后修改: 2020/8/13
6.  */
7.
8. #include "Parse.h"
9. #include "Environ.h"
10. #include "Command.h"
11. #include <fstream>
12.
13. //处理子进程返回信号,并备份旧信道
14. struct sigaction sig_action;
15. struct sigaction old_sig_action;
16.
17. //无参数状态下初始化命令行模式
18. void init_commandline(){
19.     //初始化信号
20.     signalInit();
21.
22.     //判断当前 myshell 状态,当 run 时保持运行
23.     while(envvar["status"]=="run"){
24.         //当前目录
25.         displayDir();
26.         //接受命令
27.         getCommand(cin);
28.     }
29. }
30.
31. //含参数状态下初始化文件
32. void init_file(string fileName){
33.     //初始化信号
34.     signalInit();
35.     //打开文件

```



```

36.     ifstream fp(fileName);
37.     //文件打开失败
38.     if(!fp)
39.         errorExit("file read in", FILE_FAILED);
40.     //判断当前 myshell 状态,当 run 时保持运行
41.     while(envvar["status"]=="run")
42.         if(!getCommand(fp))
43.             break;
44. }
45.
46. //打印当前目录
47. void displayDir(){
48.     //获取 pwd 信息
49.     string currentdir = envvar["pwd"];
50.     //获取最后目录值
51.     int pos = currentdir.rfind('/');
52.     //输出
53.     cout << GREEN << "ziteng:" << currentdir.substr(pos)+"$ " << NORMAL ;
54.     //清空输出缓冲区
55.     fflush(stdout);
56. }
57.
58. //从输入流 is 获取命令
59. bool getCommand(istream & is){
60.     string com;
61.     //获取一行内容
62.     getline(is, com);
63.     //内容非空
64.     if(!com.empty())
65.     {
66.         if(com=="help")// help 实现
67.             com = "cat /usr/local/bin/readme_mysh | more\n";
68.         //入口,调用解析模块
69.         parse((char*)com.c_str());
70.         //解析成功
71.         return true;
72.     }
73.     //空内容
74.     else return false;
75. }
76.
77.
78. //将读取的内容解析
79. bool parse(char* com){

```

```
80.     //命令集合
81.     vector<command> cmdlist;
82.     //单条命令
83.     command Cmd;
84.     command* newCmd;
85.     //当前状态机
86.     parseStatus status = NEWCMD;
87.
88.     //控制选项
89.     bool option = false;
90.
91.     //存放分割后命令
92.     char* word;
93.     while((word = strsep(&com, " \t\n"))){
94.         //连续多个空白符
95.         if(!strlen(word))
96.             continue;
97.         //状态机解析
98.         switch (status)
99.         {
100.             //新命令
101.             case NEWCMD_PIPE:
102.             case NEWCMD:
103.                 //空白命令
104.                 if(word[0]==' ' || word[0] == '\t')
105.                     break;
106.                 //压入节点
107.                 cmdlist.push_back(Cmd);
108.                 newCmd = &cmdlist[cmdlist.size()-1];
109.                 newCmd->cmdname = word;
110.                 //检查管道状态
111.                 if(status = NEWCMD_PIPE)
112.                     newCmd->pipein = true;
113.                 //切换为读取中
114.                 status = NEWCMDREADY;
115.                 break;
116.                 //读取状态
117.                 case NEWCMDREADY:
118.                     //管道/重定向/后台符号
119.                     if(strlen(word)==1)
120.                     {
121.                         //后台执行
122.                         if(word[0] == '&'){
123.                             newCmd->isbg = true;
```

```

124.             status = NEWCMD;
125.             break;
126.         }
127.         //管道
128.         else if (word[0] == '|'){
129.             newCmd->pipeout = true;
130.             status = NEWCMD_PIPE;
131.             break;
132.         }
133.         //输入重定向
134.         else if (word[0] == '<'){
135.             newCmd->indirect = COVER;
136.             status = INREDIRECT;
137.             break;
138.         }
139.         //输出重定向
140.         else if (word[0] == '>'){
141.             newCmd->outdirect = COVER;
142.             status = OUTREDIRECT;
143.             break;
144.         }
145.     }
146.     //追加重定向
147.     else if (strlen(word) == 2){
148.         //重定向输出
149.         if(word[0] == '>' && word[1] == '>'){
150.             newCmd->indirect = APPEND;
151.             status = OUTREDIRECT;
152.             break;
153.         }
154.         //重定向输入
155.         else if(word[0] == '<' && word[1] == '<'){
156.             newCmd->outdirect = APPEND;
157.             status = INREDIRECT;
158.             break;
159.         }
160.     }
161.     //命令选项
162.     if('-' == word[0]){
163.         //多于一个选项
164.         if(strlen(word) != 2){
165.             cout << RED << "Illegal" << NORMAL << endl;
166.             return false;
167.         }

```

```

168.         newCmd->arglist.option.push_back(word[1]);
169.         option = true;
170.         break;
171.     }
172.     //是否有选项参数
173.     else if(option){
174.         newCmd->arglist.cmdargu[newCmd->arglist.cmdargu.size()-
175.         1] = word;
176.         option = false;
177.         break;
178.     }
179.     //没有选项参数
180.     else{
181.         newCmd->arglist.cmdargu.push_back(word);
182.         break;
183.     }
184.     //输入重定向
185.     case INREDIRECT:
186.         newCmd->infile = word;
187.         status = NEWCMDREADY;
188.         break;
189.     //输出重定向
190.     case OUTREDIRECT:
191.         newCmd->outfile = word;
192.         status = NEWCMDREADY;
193.         break;
194.     }
195. }
196. //初始化管道
197. pipe(pipeFd);
198. //遍历执行命令序列
199. for(auto i = cmdlist.begin(); i != cmdlist.end(); i++)
200.     excute(*i);
201. }
202.
203.
204. //命令执行
205. void excute(command& cmd){
206.     //开辟新进程
207.     pid_t pid = fork();
208.     switch (pid)
209.     {
210.         case -1: {

```

```

211.         errorExit("fork", FORK_FAILED);
212.     }
213.     //子进程
214.     case 0: {
215.         //轮询等待
216.         usleep(WAIT_TIME);
217.         //查找当前进程信息
218.         int pos = jobFind(pid);
219.         //主进程初始化
220.         while (joblist[pos].status == SUSPEND)
221.             usleep(WAIT_TIME);
222.         //子进程执行命令
223.         childProcess(cmd);
224.         //退出当前子进程
225.         exit(0);
226.     }
227.     //主进程
228.     default: {
229.         //建立新建进程信息
230.         job child;
231.         child.pid = pid;
232.         child.name = cmd.cmdname;
233.         child.status = SUSPEND;
234.         child.type = cmd.isbg ? BG : FG;
235.         joblist.push_back(child);
236.         //后台进程
237.         if (cmd.isbg)
238.             //提示信息
239.             cout << pid << '\t' << cmd.cmdname << endl;
240.         else { //前台进程
241.             sigaction(SIGCHLD, &old_sig_action, NULL);
242.             //主进程关闭管道
243.             if (cmd.pipein) {
244.                 close(pipeFd[0]);
245.                 close(pipeFd[1]);
246.             }
247.         }
248.         //子进程执行
249.         joblist.back().status = RUN;
250.
251.         //前台进程
252.         if (cmd.isbg) {
253.             //等待
254.             pause();

```

```
255.          //进程表中删除前台进程
256.          if (joblist.back().status == RUN)
257.              jobdel(pid);
258.          //监听 SIGCHLD
259.          sigaction(SIGCHLD, &sig_action, NULL);
260.          //更新 mask
261.          if (*mask < 26)
262.              umask(*mask);
263.      }
264.  }
265.  }
266.}
267.
268.//子进程负责运行命令
269.void childProcess(command& cmd){
270.    //备份旧的 stdin/stdout 文件描述符
271.    int oldInFd = dup(fileno(stdin));
272.    int oldOutFd = dup(fileno(stdout));
273.    //管道重定向
274.    if (cmd.pipein)
275.    {
276.        //从管道接收
277.        dup2(pipeFd[0], fileno(stdin));
278.        close(pipeFd[0]);
279.        close(pipeFd[1]);
280.    }
281.    //管道重定向
282.    if (cmd.pipeout)
283.    {
284.        //向管道输出
285.        dup2(pipeFd[1], fileno(stdout));
286.        close(pipeFd[0]);
287.        close(pipeFd[1]);
288.    }
289.    //文件重定向
290.    switch (cmd.indirect)//输入重定向
291.    {
292.        case NOREDIRECT:
293.            break;
294.        case COVER:
295.            redirectInit(stdin, cmd.infile, COVER);
296.            break;
297.        case APPEND:
298.            redirectInit(stdin, cmd.infile, APPEND);
```

```
299.         break;
300.     }
301.     switch (thisCmd->outdirect)
302.     {
303.         case NOREDIRECT:
304.             break;
305.         case COVER:
306.             {
307.                 redirectInit(stdout, thisCmd->outFile, COVER);
308.                 break;
309.             }
310.         case APPEND:
311.             {
312.                 redirectInit(stdout, thisCmd->outFile, APPEND);
313.                 break;
314.             }
315.     }
316.
317.     //根据命令名解析命令,并调用执行层
318.     if(cmd.cmdname=="pwd")
319.         pwd();
320.     else if(cmd.cmdname=="exit")
321.         exit();
322.     else if(cmd.cmdname=="clr")
323.         clr();
324.     else if(cmd.cmdname=="time")
325.         time();
326.     else if(cmd.cmdname=="echo")
327.         echo(cmd.arglist);
328.     else if(cmd.cmdname=="cd")
329.         cd(cmd.arglist);
330.     else if(cmd.cmdname=="exec")
331.         exec(cmd.arglist);
332.     else if(cmd.cmdname=="environ")
333.         enviro();
334.     else if(cmd.cmdname=="set")
335.         set(cmd.arglist);
336.     else if(cmd.cmdname=="unset")
337.         unset(cmd.arglist);
338.     else if(cmd.cmdname=="umask")
339.         umask(cmd.arglist);
340.     else if(cmd.cmdname=="test")
341.         test(cmd.arglist);
342.     else if(cmd.cmdname=="jobs")
```

```

343.     jobs();
344.     else if(cmd.cmdname=="shift")
345.         shift(cmd.arglist);
346.     else if(cmd.cmdname=="fg")
347.         fg();
348.     else if(cmd.cmdname=="bg")
349.         bg();
350.     else{//外部命令
351.         //参数转存
352.         vector<string> args;
353.         char *argp[MAX_COMLEN];
354.         //单类参数个数,计数器
355.         int i = 1, cnt = 0;
356.
357.         //调用文件名
358.         args.push_back(cmd.cmdname);
359.
360.         //存储命令选项
361.         while(i != cmd.arglist.option.size())
362.             if(cmd.arglist.option[i-1] != 0)
363.                 args.push_back("-"+cmd.arglist.option[i-1]);
364.         cnt += i;
365.         i = 1;
366.         //存储选项参数
367.         while(i != cmd.arglist.optargu.size())
368.             if(cmd.arglist.optargu[i-1] != "")
369.                 args.push_back(cmd.arglist.optargu[i-1]);
370.         cnt += i;
371.         i = 1;
372.         //存储命令参数
373.         while(i != cmd.arglist.cmdargu.size())
374.             if(cmd.arglist.cmdargu[i-1] != "")
375.                 args.push_back(cmd.arglist.cmdargu[i-1]);
376.         cnt += i;
377.         i = 0;
378.
379.         //结束
380.         argp[args.size()] = NULL;
381.         //赋回结果
382.         while(i++ < args.size())
383.             argp[i] = (char*)args[i].c_str();
384.         //调用 exec 调用
385.         execvp((char*)cmd.cmdname.c_str(), argp);
386.     }

```



```
387.
388.     //恢复文件重定向
389.     if (cmd.indirect != NOREDIRECT)
390.     {
391.         redirectEnd(stdin, oldInFd);
392.     }
393.     if (cmd.indirect != NOREDIRECT)
394.     {
395.         redirectEnd(stdout, oldOutFd);
396.     }
397.     //恢复管道重定向
398.     if (cmd.pipein)
399.     {
400.         dup2(oldInFd, fileno(stdin));
401.     }
402.     if (cmd.pipeout)
403.     {
404.         dup2(oldOutFd, fileno(stdout));
405.     }
406. }
407.
408. //重定向初始化
409. void redirectInit(FILE* src, string fileName, redirect type){
410.     //重定向模式
411.     int fd = 0;
412.     int fs = -1;
413.     switch (type)
414.     {
415.         case COVER:
416.             fd = open((char*)fileName.c_str(), O_RDWR | O_CREAT, 0644);
417.             break;
418.         case APPEND:
419.             fd = open((char*)fileName.c_str(), O_RDWR | O_APPEND | O_CREAT,
420.                 0644);
421.             break;
422.         default:
423.             break;
424.     }
425.     //复制文件描述符
426.     if (-1 == dup2(fd, fileno(src)))
427.     {
428.         errorExit("dup", DUP_FAILED);
429.     }
430.     //关闭文件描述符
```

```
430.     close(fd);
431. }
432.
433. // 恢复重定向前的状态
434. void redirectEnd(FILE* dst, int oldFd)
435. {
436.     //赋值旧文件描述符
437.     if (-1 == dup2(oldFd, fileno(dst)))
438.     {
439.         errorExit("dup", DUP_FAILED);
440.     }
441.     //关闭旧文件描述符
442.     close(oldFd);
443. }
444.
445. //初始化信息处理
446. void signalInit(){
447.     //监听 ctrlz
448.     signal(SIGSTP, ctrl_z_Handler);
449.     signal(SIGSTOP, ctrl_z_Handler);
450.
451.     //清零信号结构
452.     memset(&sig_action, 0, sizeof(sig_action));
453.
454.     //信号处理函数
455.     sig_action.sa_sigaction = childExit;
456.     sig_action.sa_flags = SA_RESTART | SA_SIGINFO;
457.     sigemptyset(&sig_action.sa_mask);
458.     //SIGCHLD 信号处理
459.     sigaction(SIGCHLD, &sig_action, &old_sig_action);
460. }
461.
462. //子进程结束函数
463. void childExit(int sig_no, siginfo_t* info, void* vcontext)
464. {
465.     //赋值信号源进程号
466.     pid_t pid = info->si_pid;
467.     //获取进程信息
468.     int pos = jobFind(pid);
469.
470.     //后台运行
471.     if(joblist[0].type)
472.         cout << NORMAL << joblist[pos].pid << "\t" << joblist[pos].name << "
\t" << "end" << endl;
```

```

473.
474.     //在进程表中删除进程
475.     jobdel(pid);
476.
477.     //清除输入缓冲区
478.     tcflush(fileno(stdin), TCIFLUSH);
479.
480.     //打印路径
481.     displayDir();
482. }
483.
484. //处理 ctrlz 信号,挂起当前进程
485. void ctrl_z_Handler(int sig_no){
486.     //进程表非空
487.     if(joblist.size()>1)
488.     {
489.         //阻塞信号
490.         kill(joblist.at(1).pid, SIGSTOP);
491.
492.         //更新进程状态
493.         joblist.at(1).status = SUSPEND;
494.         joblist.at(1).type = BG;
495.
496.         //反馈
497.         cout << NORMAL << joblist.at(1).pid << "\t" << joblist.at(1).name <<
            "\t" << "end" << endl;
498.     }
499. }

```

用户手册

1. *****
2. 欢迎使用 myshell
3. 作者：王子腾 3180102173
4. 最后修改日期：2020/8/18
5. 以下为用户手册，可以帮助您快速入门本系统
6. *****
- 7.
8. ***设计思想***
9. 开发语言：C++
10. 运行环境：Linux
11. 本系统基于 C++面向对象开发，是一款介于操作系统内核和上层用户层的命令解释器
12. myshell 支持内部命令和外部命令，支持管道、重定向，实现部分由四个模块组成，分别为主入口模块、命令执行模块、运行环境模块和命令解析模块。

- 13.
- 14.
15. ***名词解释***
16. *重定向*
17. myshell 默认的输入和输出默认基于 cin 和 cout，即标准输入输出，通过键盘和屏幕进行操作。
18. 我们可以将输入或输出通过'<'、'>'、'>>'符号进行重定向，用其他文件替代标准 I/O 源文件，这个过程叫做重定向。
19. --'<' 输入重定向
20. --'>' 输出重定向
21. --'>>' 输出重定向(追加)
- 22.
23. *管道*
24. 通过管道操作符'|'，用户可以将多个命令配合使用，如"command1 | command2"命令组合了两条命令，并将 command1 的标准输出传送至 command2 作为输入。
25. 管道连接的命令被称为过滤器，可以逐层将前方处理后返回的结果进行过滤，并最终得到目标解，管道常用于文本操作。
26. --'|' 管道操作符
- 27.
28. *程序环境*
29. myshell 的编写语言为 C++高级语言，内部使用了底层系统调用和部分 C 语言的信号处理函数，myshell 系统可在 Linux 环境下运行。
30. 运行环境模块定义了进程、环境变量、系统宏值等基本信息，并提供了进程支持和环境搭建，相关指令有：
31. --set
32. --unset
33. --environ
34. --umask
- 35.
36. *后台执行*
37. myshell 支持后台执行和进程的前后台切换，提供 Run 和 Suspend 两种运行状态，FG 和 BG 两种类型。
38. 若命令为后台执行，则在执行命令的同时，可以继续在前台进程中执行其他命令而无需等待该命令执行完毕，后台执行标识符为'&'
39. 相关指令有：
40. --fg
41. --bg
42. --jobs
- 43.
- 44.
45. ***支持指令***
46. 当前支持指令：
47. bg、cd、clr、dir、echo、exec、exit、environ、fg、help、jobs、pwd、quit、set、shift、test、time、umask、unset
- 48.

- 49.
50. ***致语***
51. 祝您使用愉快!!

运行截图

```
ziteng:mysHELL$ help
*****
欢迎使用mysHELL
作者：王子腾 3180102173
最后修改日期：2020/8/18
以下为用户手册，可以帮助您快速入门本系统
*****

***设计思想***
开发语言：C++
运行环境：Linux
本系统基于C++面向对象开发，是一款介于操作系统内核和上层用户层的命令解释器
mysHELL支持内部命令和外部命令，支持管道、重定向，实现部分由四个模块组成，分别为
主入口模块、命令执行模块、运行环境模块和命令解析模块。

***名词解释***
*重定向*
mysHELL默认的输入和输出默认基于cin和cout，即标准输入输出，通过键盘和屏幕进行操作。
```

help 命令

```
ziteng:mysHELL$ cd..
ziteng:sf_share$ pwd
/media/sf_share
ziteng:sf_share$ echo this is mysHELL
this is mysHELL
ziteng:sf_share$ time
Mon Aug 17 20:32:45 2020
ziteng:sf_share$ dir ..
sf_share ziteng
ziteng:sf_share$ environ
status=run
shell=/usr/local/mysHELL
pwd=/media/sf_share
ziteng:sf_share$ exit
ziteng@Ziteng-VBox: /media/sf_share/mysHELL$
```

cd、pwd、echo、time、dir、environ、exit 命令

```
ziteng@Ziteng-VBox: /media/sf_share/myshell
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
ziteng:myshell$
```

```
ziteng:myshell$ clr

ziteng:myshell$
```

clr 命令

```
ziteng:myshell$ sleep 7 &
2107  sleep
ziteng:myshell$ jobs
2107  sleep  Run
ziteng:myshell$
2107  sleep  end
ziteng:myshell$ sleep 5 &
2108  sleep
ziteng:myshell$ fg
2108  sleep  Run
ziteng:myshell$
```

后台 (&) 执行, jobs、fg 命令


```
ziteng:myshell$ sleep 5
^Z
2290  sleep  Suspend
ziteng:myshell$ jobs
2290  sleep  Suspend
ziteng:myshell$ bg
2290  sleep  Run
ziteng:myshell$
2290  sleep  end
ziteng:myshell$
```

bg 命令

```

ziteng:myshell$ set linux linus
ziteng:myshell$ set
status=run
shell=/usr/local/myshell
pwd=/media/sf_share/myshell
linux=linus
ziteng:myshell$ unset linux
ziteng:myshell$ set
status=run
shell=/usr/local/myshell
pwd=/media/sf_share/myshell
ziteng:myshell$ umask
0022
ziteng:myshell$ shift 2 < sample >> sample1
ziteng:myshell$ quit
ziteng@Ziteng-VBox:/media/sf_share/myshell$


```

打开(O) ▾  **sample**
sf_share /media/sf_share/myshell

```

123
abc
!@#
one two three

```

打开(O) ▾  **sample1**
sf_share /media/sf_share/myshell

```

!@#
one two three

```

set、unset、umask、shift、quit 命令，重定向

```

ziteng:myshell$ myshell
请输入运行参数
ziteng:myshell$ myshell test.sh a b c d
第一个参数为: a 参数个数为: 4
第一个参数为: b 参数个数为: 3
第一个参数为: c 参数个数为: 2
第一个参数为: d 参数个数为: 1
ziteng:myshell$ time | shift 4
2020
ziteng:myshell$ exec test -f sample
true
ziteng@Ziteng-VBox:/media/sf_share/myshell$

```

```
test.sh
sf_share /media/sf_share/myshell

#!/bin/bash
while [ $# != 0 ]
do
echo "第一个参数为: $1 参数个数为: $#"
```

子进程运行 shell 文件，管道，exec、test 命令

三、讨论、心得

本次实验让我熟悉了 shell 编程的逻辑与用法，以下为完成过程中遇到的问题解决方案与相关心得：

①set 后的参数问题：在最开始使用 set 指令解析数据域时，我按照 ppt 里的格式写好的代码，总是会遇到系统提示 set 附近参数有误，通过上网查找相关资料才明白需要添加 “--” 指令来解析参数，改为 set -- \$(ls -l \$filename)后顺利解决；

②改进：经过后期的学习与巩固，作业 1.sh 中的循环体部分可以进一步精简为：

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

通过将查找结果用管道传递给 wc 指令进而直接将结果输出，就不需要再经过循环和判断语句计数了，这样可以提高代码运行效率；

③递归异常的问题：作业 3 困扰我最大的点就在于的递归部分，在最开始编写程序时，我参照了 c、c++ 等高级语言的递归思路，忽略了变量的作用域问题，使用普通变量用以保存当前目录与循环中间值，以为这些变量会作为临时变量保存在递归栈中，递归返回时自动恢复。但测试程序时发现循环异常退出，观察到留下的 echo 标记并没有被执行，程序的循环与递归部分发生了问题，几次修改不成后，通过上网搜索 shell 的变量定义属性，了解到函数内直接实用的变量会被自动定义为全局变量，并在触发递归后被传递到递归函数中，在函数内被修改并返回当前栈后，由于是全局变量，因此并不会恢复之前

的值。了解问题原因后，我将函数体内的所有变量均声明为 `local` 作用域，递归异常的问题从而也就迎刃而解了。

④在作业管理系统的开发中，由于之前在数据库课程中开发过类似的管理系统，因此一上来就带入了高级语言的开发思路，却发现一时没有了入手点，随着复习课件中的编程思路，慢慢有了头绪，也渐渐明白了 `shell` 脚本语言与其他高级语言的区别，对于字符串的处理，`shell` 强在可以通过管道和重定向直接操作文件内容，通过 `sed`、`cut` 等指令从复杂信息中解析出关键字，并根据逐级菜单和调用间的变量持续性维持系统的流畅运转，实现了自顶向下的层次化程序设计思路。

⑤开发 `myshell` 的周期拖了很长，主要困难在于对信号的处理部分和对进程的控制，最早我使用的是《Linux 程序设计》作为工具书，阅读了信号传送部分，对 `SIGCHLD` 等基本信号和信号的 `Handle` 方式有了一定的了解，经过多次尝试，才成功通过 `signal.h` 中的 `signal` 等函数设置好了监听和处理机制，并成功通过子进程运行了指令。对于进程的控制位点，我同样参考了《Linux 程序设计》，在完成对信号部分的控制之后，我选择在所有指令的执行起步阶段通过 `fork` 指令开启子进程，并通过子进程来运行指令，在这个过程中通过不断判断当前 `myshell` 状态来设置阻塞或保持主进程运行，以此达到前后台切换的效果。对于环境的设置，我采用了 `map` 作为映射关系，通过 `string` 之间的匹配完成环境变量的对应，并通过 `vector` 容器实现了进程表和命令参数的灵活变更，这也是 `C++` 作为高级编程语言的优势，对 `STL` 的灵活运用减少了不少的工作量。对于命令的解析，由于题目对字符串的要求较为开放，我默认操作符间只有一个空格，采用了最直接的字符串查找和步进的方式检索关键符号，并分割指令名称，通过 `if-else` 匹配相应内部、外部指令，并通过调用 `Command.cpp` 中实现好的内部指令或 `execvp` 调用来实现，最终搭建成功内部、外部指令都可运行的环境。