**中国矿业大学计算机学院**

**2019 级本科生课程大作业**

课程名称 Linux操作系统

报告时间 2022-5-4

学生姓名 王杰永

学 号 03190886

专 业 计算机科学与技术

任课教师 姜秀柱

目录

[1 题目一 1](#_Toc101816425)

[1.1 题目要求 1](#_Toc101816426)

[1.2 题目作答 1](#_Toc101816427)

[2 题目二 9](#_Toc101816428)

[2.1 题目要求 9](#_Toc101816429)

[2.2 题目作答 9](#_Toc101816430)

[3 题目三 22](#_Toc101816431)

[3.1 题目要求 22](#_Toc101816432)

[3.2 题目作答 22](#_Toc101816433)

[4 题目四 38](#_Toc101816434)

[4.1 题目要求 38](#_Toc101816435)

[4.2 题目作答 38](#_Toc101816436)

[5 题目五（附加题） 48](#_Toc101816437)

[5.1 题目要求 48](#_Toc101816438)

[5.2 题目作答 48](#_Toc101816439)

* 1. 题目一
     1. 题目要求

给出完成以下功能的linux基本命令及每条命令的执行结果截图（20分，每小题10分）

子问题(1)：查看当前目录，在当前目录下创见一个新目录，然后进入这个新目录，在这个新目录下创建一个空文件，分别查看该文件的简单列表，文件类型和详细属性以及该文件所占空间，接下来将该文件的所有者改为root，赋给所有者读写执行完全权限，并将该文件的有效时间更新为2027年8月1日24时，最后回到当前目录。

子问题(2)：在当前目录下，用屏幕输出命令将“hello，world！“写入操作（1）建立的文件中，并用接受键盘输入在屏幕显示的命令向该文件添加三组姓名-学号对（姓名用拼音）的内容，然后计算该文件的单词数。接下来对该文件内容按行排序，输出最后一行，再将文件中的学号提取出来输出到一个新建文件中，比较这两个文件，比较结果输出到第三个文件中。最后将该目录下的三个文件拷贝到其父目录中，再将该目录删除。

* + 1. 题目作答
       1. 子问题(1)

**查看当前目录：**

命令用于显示用户当前所在的工作目录（以绝对路径显示）。

|  |
| --- |
| 命令1- pwd |
| pwd [-LP]  -L （默认值）打印环境变量"$PWD"的值，可能为符号链接。  -P 打印当前工作目录的物理位置。 |

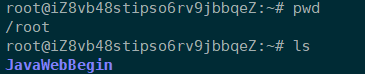
我们在服务器中执行命令，结果如下。

图1-1 运行结果

**创建新目录，并进入目录**

命令用于创建新目录。命令可以进入目标目录。

|  |
| --- |
| 命令1- mkdir |
| mkdir (选项)(参数)  -p 若所要建立目录的上层目录目前尚未建立，则会一并建立上层目录； |

|  |
| --- |
| 命令1- cd |
| cd [-L|[-P [-e]]] [dir]  -P 如果要切换到的目标目录是一个符号连接，那么切换到它指向的物理位置目录。 |

执行命令，并使用命令进入所创建的新目录中。结果如下：

图1-2 运行结果

**创建空文件，查看文件详细信息**

使用命令创建空文件，使用命令查看文件的详细信息。

|  |
| --- |
| 命令1- touch |
| touch(选项)(参数)  -d：<时间日期> 使用指定的日期时间，而非现在的时间；  -a：或--time=atime或--time=access或--time=use 只更改存取时间；  -c：或--no-create 不建立任何文件； |

|  |
| --- |
| 命令1- ls |
| ls [选项] [文件名...]  -R：递归列出遇到的子目录。  -a：列出所有文件，包括以 "." 开头的隐含文件。  -1：列出文件详细信息。 |

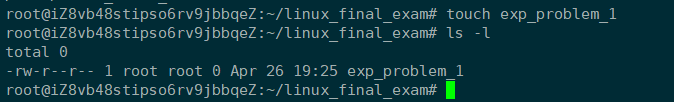
我们使用命令创建新文件，使用命令查看新文件的详细信息。结果如图：

图1-3 运行结果

在命令所列出的详细信息中，从左至右依次包括文件名、文件类型、权限、硬链接数、所有者名、组名、大小（byte）。

* -：表示该文件为普通文件
* rw-r--r--：表示该文件的读写可执行权限
* 1：对于普通文件，表示链接数
* root：用户名
* root：组名
* 0：文件大小
* Apr 26 19:25：最后修改时间
* exp\_problem\_1：文件名

因此，我们可以得出新建立的文件类型为普通文件，该文件所占空间为0（因为还没有向文件中写入数据）。

**修改文件所有者为root，赋给所有者读写执行完全权限**

文件的所有者一般是文件创建者（也就是root），可以通过**命令**修改文件所有者，通过命令修改文件或目录的权限。

|  |
| --- |
| 命令1- chown |
| chown(选项)(参数)  -R或——recursive：递归处理，将指定目录下的所有文件及子目录一并处理； |

|  |
| --- |
| 命令1- chmod |
| chmod [OPTION]... MODE[,MODE]... FILE...  chmod [OPTION]... OCTAL-MODE FILE... |

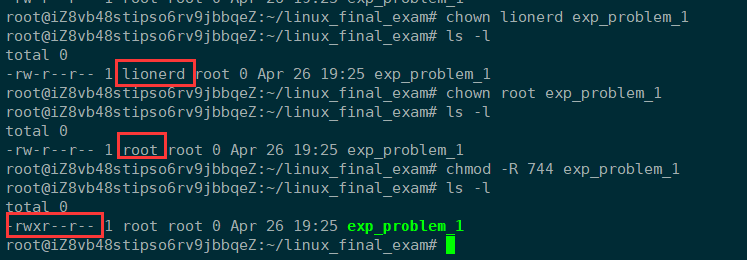
由于新创建的文件其所有者默认为root，因此这里我通过命令，将文件所有者更改为lionerd（仅为了展示效果），再通过命令将所有者修改回root，最后通过命令付给所有者读写执行完全权限。结果如图：

图1-4 运行结果

可以发现，在为文件添加可执行权限后，使用命令列出的文件名变成了绿色（表示可执行）。

**修改文件更新时间，返回当前目录**

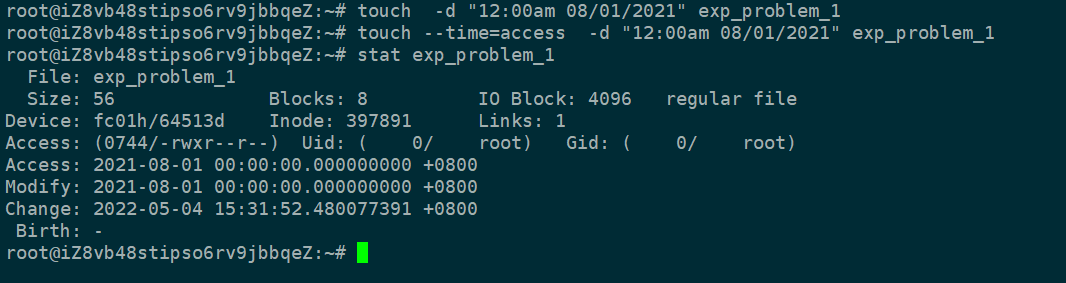
touch命令同样可以修改文件的文件访问时间、文件修改时间。结果如图：

图1-5 运行结果

可以看出，文件的时间成功被修改。最后通过cd ..可以返回父目录。

* + - 1. 子问题(2)

**向文件中写入“hello, world!”**

使用命令可以向标准输出打印信息，使用>对输出流重定向到文件。

|  |
| --- |
| 命令1- echo |
| echo(选项)(参数)  -e：激活转义字符。 |

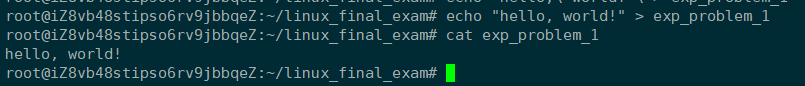
我们使用命令，将“hello，world！”写入文件。

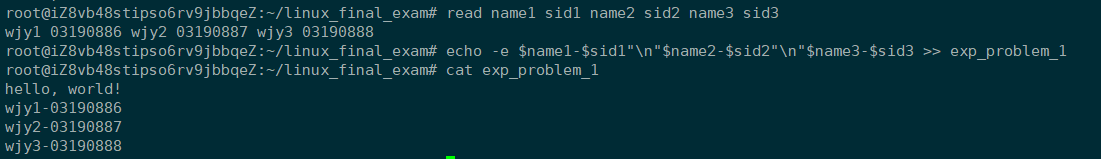
图1-6 运行结果

**向文件中追加三组姓名-学号对**

使用命令接受键盘输入，赋值到相应变量中。

|  |
| --- |
| 命令1- read |
| read(选项)(参数)  -p：指定读取值时的提示符；  -t：指定读取值时等待的时间（秒）。 |

使用命令，接受键盘输入赋值到指定变量中；使用命令，将变量值输出重定向到文件尾部。运行结果如下：

图1-7 运行结果

**计算文件单词数**

使用命令可以统计文件中字节数、字数、行数，并将统计结果显示输出。

|  |
| --- |
| 命令1- wc |
| wc(选项)(参数)  -c 统计字节数，或--bytes：显示Bytes数。  -l 统计行数，或--lines：显示列数。  -m 统计字符数，或--chars：显示字符数。  -w 统计字数，或--words：显示字数。一个字被定义为由空白、跳格或换行字符分隔的字符串。  -L 打印最长行的长度，或--max-line-length。 |

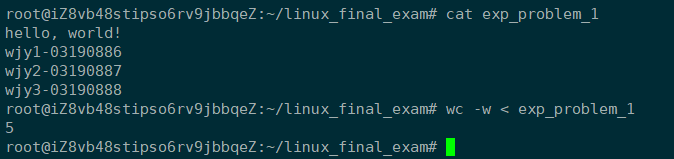
直接使用会打印单词数的同时打印文件名。我们可以使用输入重定向功能，获得目标文件中的单词数。即使用命令，获得文件中的单词数。结果如下：

图1-8 运行结果

**对文件内容按行排序，输出最后一行**

使用命令可以对文件内容排序；使用命令可以查看文件末尾若干行。

|  |
| --- |
| 命令1- sort |
| sort [OPTION]... [FILE]...  -r, --reverse 将结果倒序排列。 |

|  |
| --- |
| 命令1- tail |
| tail (选项) (参数)  -n, --line=NUM 输出文件的尾部NUM（NUM位数字）行内容。 |

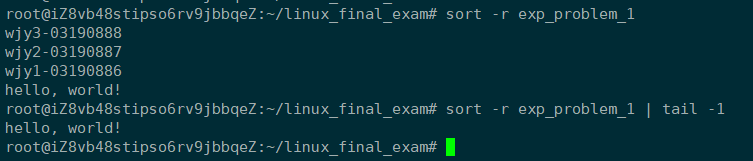
由于我们的文件内容恰好是按照ASCII码升序排序的，因此为了展示命令，我们通过参数对文件降序排序，即命令；为了展示排序后的最后一行内容，使用管道符将排序后的内容作为命令的输入，结果如下：

图1-9 运行结果

**提取文件中的学号**

命令是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。用于过滤/搜索的特定字符。可使用正则表达式能配合多种命令使用，使用上十分灵活。

|  |
| --- |
| 命令1- grep |
| grep(选项) (参数)  -E --extended-regexp 将范本样式为延伸的普通表示法来使用，意味着使用能使用扩展正则表达式。  -o 只输出文件中匹配到的部分。  -v --revert-match # 反转查找。 |

命令可以接受文件作为参数，意为按照指定模式匹配文件中内容；我们也可以通过管道符来将文件内容传给命令。

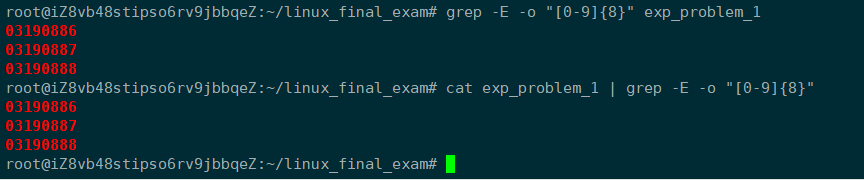
或命令都可以从文件中提取学号。其中，正则模式表示8位数字。匹配结果如下：

图1-10 运行结果

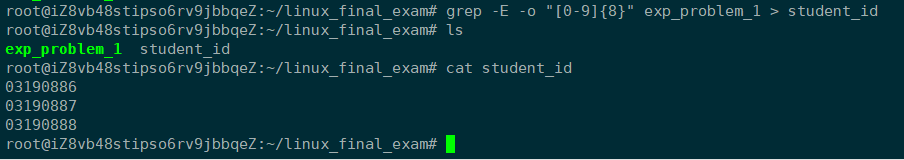
最后，通过输出重定向，即执行命令，我们将匹配内容保存到文件中。如下图：

图1-11 运行结果

**比较原文件与学号文件，并将比较结果保存至新文件**

命令为按行比较两个已排序的文件。

|  |
| --- |
| 命令1- comm |
| comm [OPTION]... FILE1 FILE2  -1 不输出第一列。  -2 不输出第二列。  -3 不输出第三列。 |

命令会输出三列，第一列为FILE1独有的行，第二列为FILE2独有的行，第三列为FILE1，FILE2共有的行。

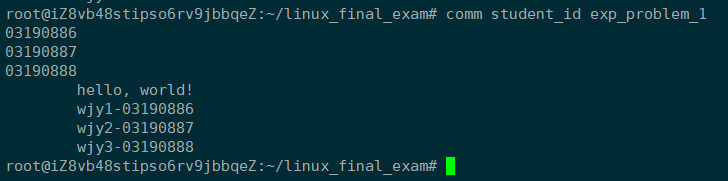
我们使用命令比较两文件的共同行（在创建学号文件时，仅仅提取了学号，因此两文件应该没有任何共同行）。结果如下：

图1-12 运行结果

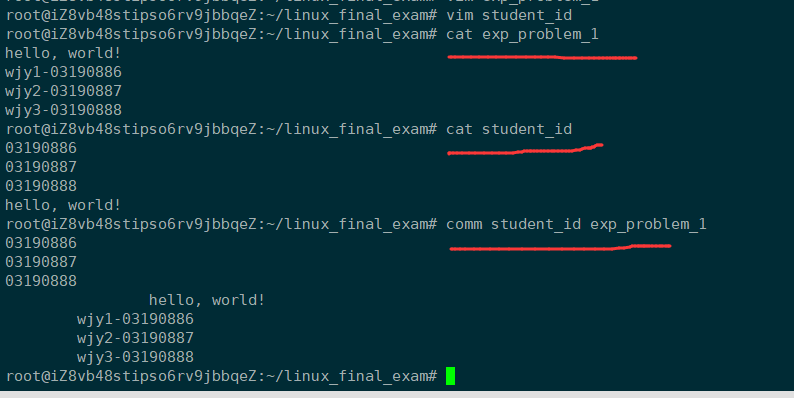
从结果中可以看出，命令的第三行输出为空，即两文件没有相同行。为了探究命令的效果，我们使用编辑器，在的尾部增加一行，再次使用命令，结果如下：

图1-13 运行结果

可以发现，命令输出的第三列中有了内容，内容正是两文件共有的行。

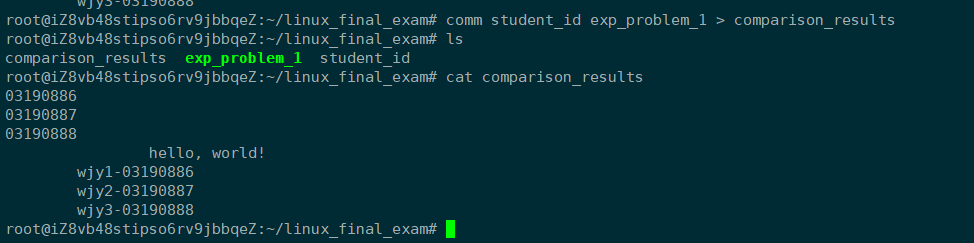
使用输出重定向，将比较结果重定向到文件中。即执行命令。结果如下：

图1-14 运行结果

**拷问文件至父目录，删除该目录**

命令可以实现文件拷贝；命令可以实现文件/文件夹删除。

|  |
| --- |
| 命令1- cp |
| cp(选项)(参数)  -f：强行复制文件或目录，不论目标文件或目录是否已存在；  -i：覆盖既有文件之前先询问用户； |

|  |
| --- |
| 命令1- rm |
| rm (选项)(参数)  -f：强制删除文件或目录；  -i：删除已有文件或目录之前先询问用户；  -r或-R：递归处理，将指定目录下的所有文件与子目录一并处理； |

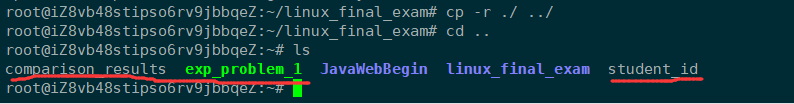
执行命令，将当前目录下的全部文件复制到父目录下。执行结果如图：

图1-15 运行结果

可以看出，新建的三个文件已经全部成功复制到了父目录中。

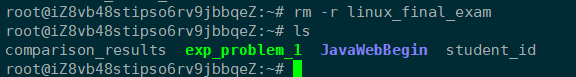
执行命令，删除题目一开始新建立的文件夹。执行结果如图：

图1-16 运行结果

* 1. 题目二
     1. 题目要求

编写一个shell程序，实现以下功能：该程序使用一个用户名为参数，如果指定参数的用户存在，就显示其存在，否则添加之，并设置初始密码为123456，显示添加的用户的id等信息，当用户第一次登录时，会提示用户立即修改密码。如果没有参数，则显示如下菜单：（1）显示用户选择的目录内容；（2）按照用户输入的目录切换路径；（3）按照用户输入的文件名在/home目录中创建文件；（4）编辑用户输入的文件；（5）删除用户选择的文件。要求程序执行过程中如发生语法错误或命令错误能够自动提示，同时停止执行，并给出相应提示信息。给出程序代码和全部执行结果截图（30分，10+4\*5）

* + 1. 题目作答

在完成此题目时，由于自己对shell脚本并不熟练，为了更好的沥青逻辑，采用面向过程式的编程思想，将全部功能组织为若干函数，且为每个子函数编写详细的多行注释，最终通过简单的程序入口函数完成了整个程序的调用和执行。

在实现判断用户是否存在以及判断已经存在的用户是否使用了简单密码123456的功能中，使用管道符配合grep命令，加以正则表达式约束，在/etc/passwd中查找用户信息，并给出了恰当的交互体验。

在具体到五种不同操作时，给出了合理美观的交互界面，并对每一种操作的非法输入加以提示并处理，因此程序有良好的鲁棒性。

程序的可执行脚本名称为，下面给出各功能的测试截图。

* + - 1. 子功能1

**使用一个用户名为参数，如果指定参数的用户存在，就显示其存在，否则添加之，并设置初始密码为123456，显示添加的用户的id等信息。**

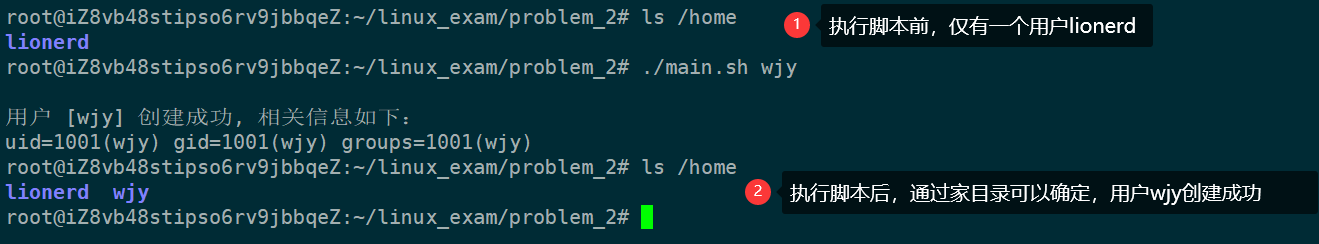
执行脚本，可以看出，新用户创建成功。

图2-1 运行结果

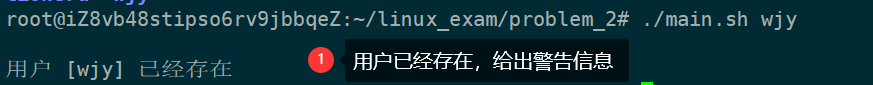
我们再次执行相同的命令，此时，由于用户已经存在，因此会给出相应的警告信息。

图2-2 运行结果

同时，创建用户应该为系统管理员的权限，当我们在非管理身份下执行该脚本时，脚本会给出相应的提示——权限不足，无法创建。

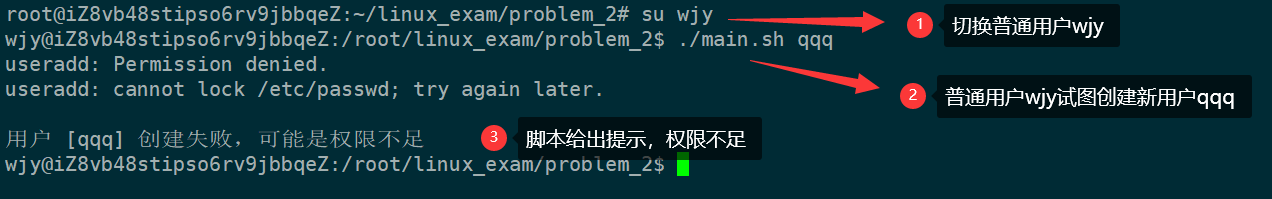
使用命令切换至用户，再次执行脚本，结果如下：

图2-3 运行结果

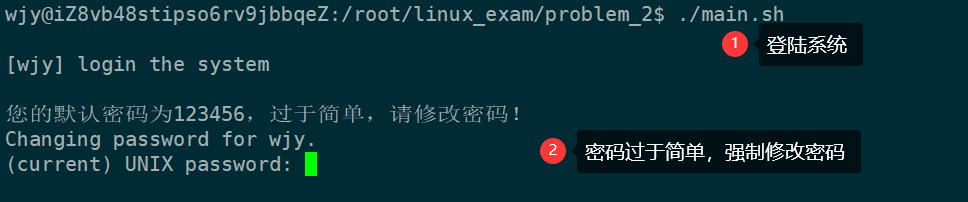
此时，我们以不带参数的形式执行该脚本，意为使用当前用户登录系统，即使用普通用户登录系统——由于当前用户的密码为默认值（123456），因此脚本给出提示，强制用户修改密码。

图2-4 运行结果

当然，在修改密码的时候，原密码输入错误、两次新密码输入不一致，均会警告并退出脚本。

图2-5 运行结果

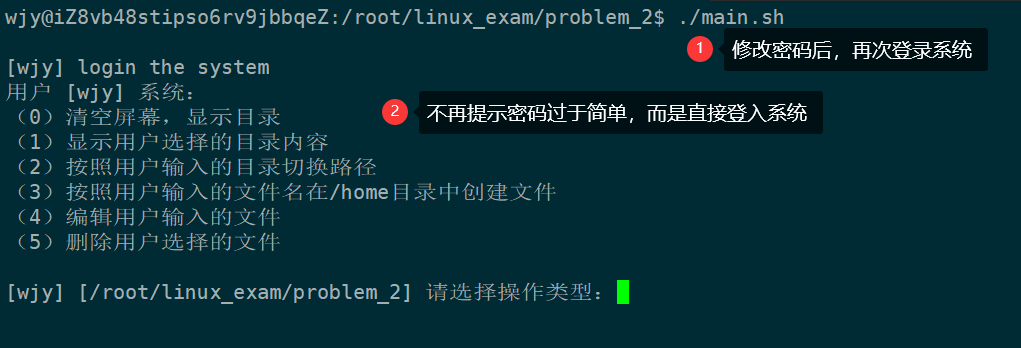
当用户修改密码成功后，再次登录系统，则不会提示“密码过于简单”等消息，直接进入系统。

图2-6 运行结果

* + - 1. 子功能2

**操作1 显示用户选择的目录内容**

进入系统，选择操作1，出现如下界面。

图2-7 运行结果

从上图可以看出，整个系统的人机交互提示栏中，显示了当前登陆系统的用户名以及用户当前所在的目录。

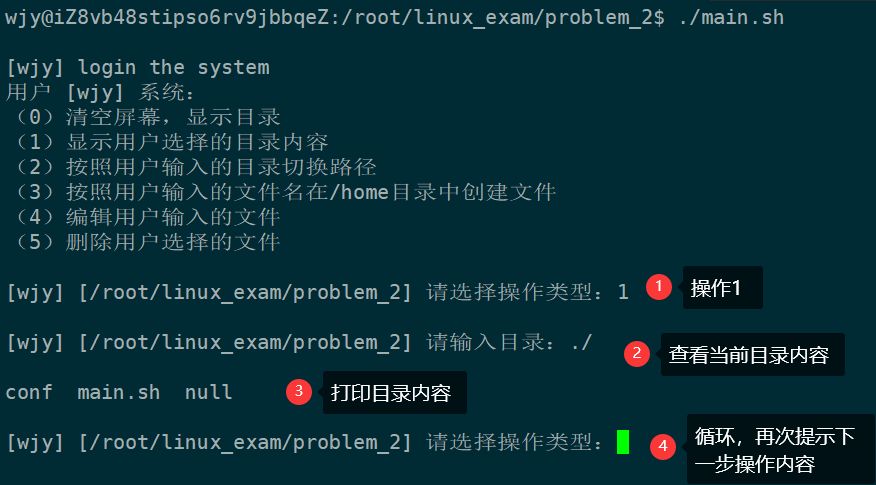
用户输入目录，会给出当前目录下的全部内容。同时，在操作完成后，程序循环重新给出下一步操作的提示。

图2-8 运行结果

当用户输入的路径非法时，程序会给出相应警告信息。

图 2-9 运行结果

* + - 1. 子功能3

**操作2 按照用户输入的目录切换路径**

选择操作2，给出当前用户的家目录，从交互栏中的工作路径信息可以看出，当前工作路径已经改变。再次选择操作1，列出当前目录下的内容，证明确实改变了路径。

图2-10 运行结果

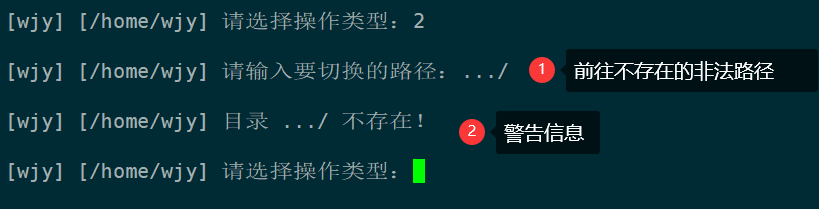
当选择的路径非法或者不存在时，脚本也能给出相应的提示信息。

图2-11 运行结果

* + - 1. 子功能4

**操作3 按照用户输入的文件名在/home目录中创建文件**

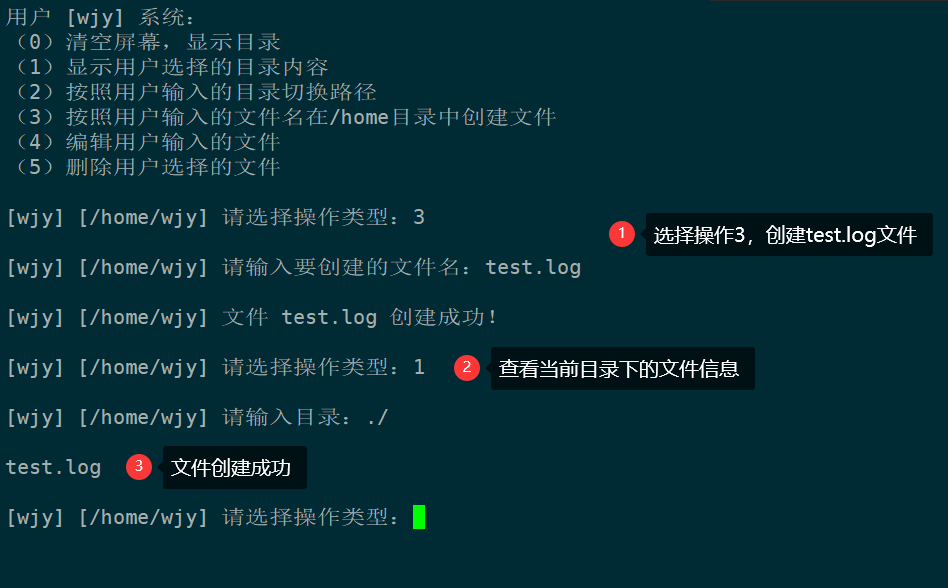
选择操作3，创建文件，之后使用操作1查看家目录下的文件信息，证明了文件创建成功。

图2-12 运行结果

当我们想要创建已经存在的文件时，脚本会给出警告“文件已经存在”，进而无法创建。

图2-13 运行结果

* + - 1. 子功能5

**操作4 编辑用户输入的文件**

我们使用操作4编辑hello.sh脚本内容。退出系统，为脚本增加可执行权限，执行脚本，结果如下：

图2-14 运行结果

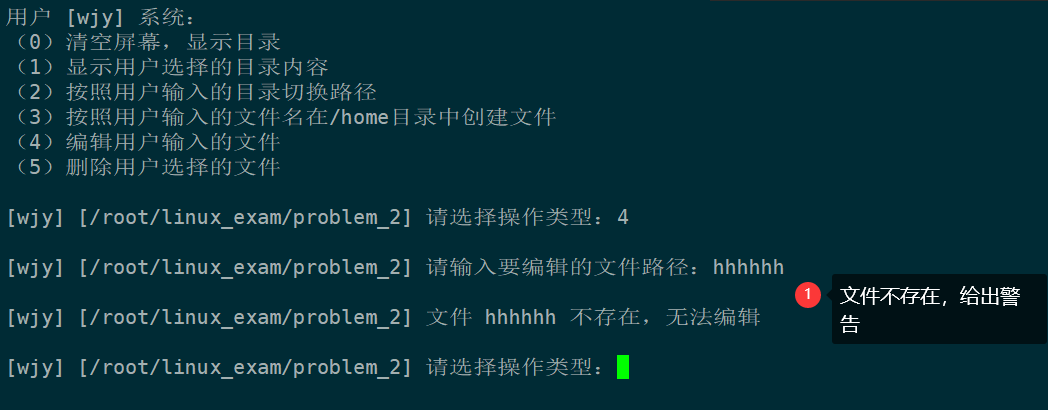
当然，如果操作4编辑的文件不存在，系统同样会给出相应的警告信息。

图2-15 运行结果

* + - 1. 子功能6

**操作5 删除用户选择的文件**

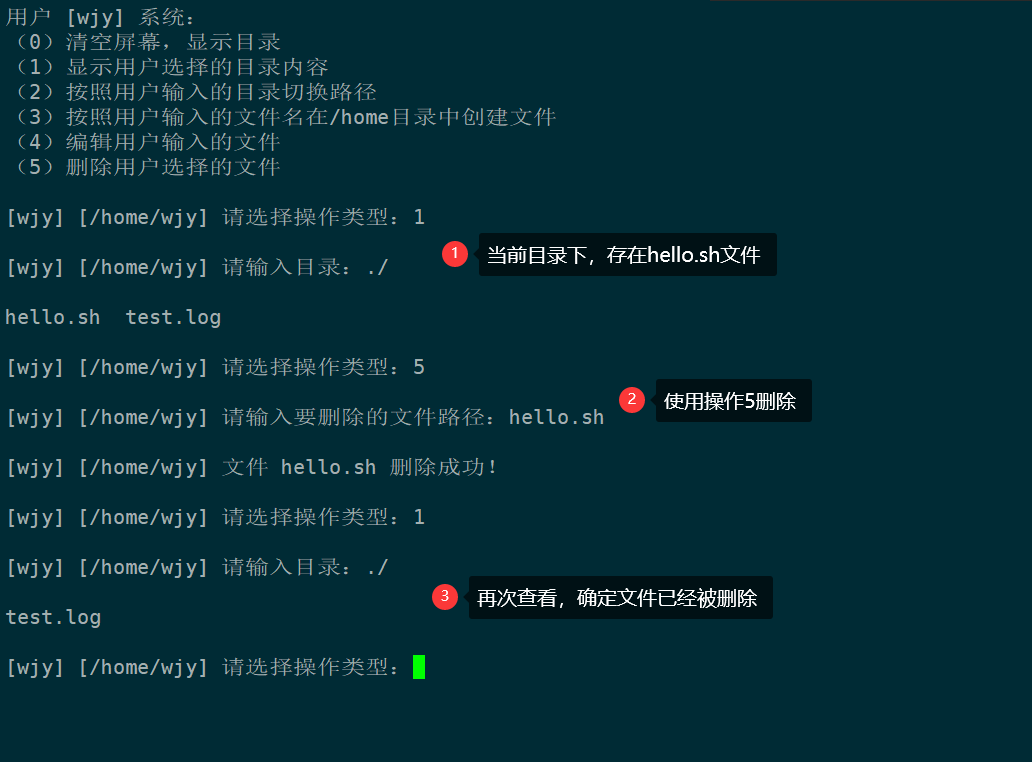
我们使用操作5，删除刚刚创建的hello.sh脚本文件。

图2-16 运行结果

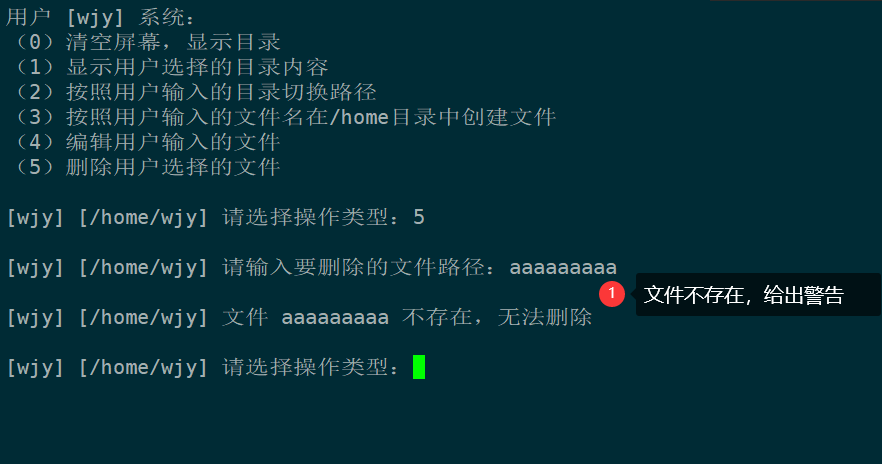
如果删除的文件不存在，同样给出警告信息。

图2-17 运行结果

* + - 1. 代码

|  |  |  |
| --- | --- | --- |
| main.sh |  |  |
| 1. *#!/bin/bash* 3. *######################################################* 4. *# author: 计科19-3班 王杰永* 5. *# 第一次写这样大型的shell脚本，为了更好的理清楚逻辑，* 6. *# 采用面向过程式编程思想，将全部功能组织了若干函数。（尽管* 7. *# 如此函数的数量多了看起来也很乱，因此做了一张函数名速查表* 8. *# ，方便查阅。* 9. *#* 10. *# 函数表：* 11. *# check\_user\_existence* 12. *# change\_password* 13. *# show\_menu* 14. *# operation\_1* 15. *# operation\_2* 16. *# operation\_3* 17. *# operation\_4* 18. *# operation\_5* 19. *# system\_main* 20. *######################################################* 22. *<<comment* 23. *$1：用户名* 24. *判断当前用户是否存在，不存在则创建$1用户，密码默认123456* 25. *comment* 26. **function** check\_user\_existence**(){** 27. query=$**(cat** **/**etc**/passwd|** **grep** ^$1:**)** 28. **if** **[[** -n $query **]]** ; **then** 29. **echo** 30. **echo** "用户 [$1] 已经存在" 31. **else** 32. useradd -d **/**home**/**$1 -s **/**bin**/bash** -m $1 33. **if** **[** $? -eq 0 **]**; **then** 34. **sed** -i "1a\**\$**1" conf 35. **echo** "$1:123456" **|** chpasswd 36. **echo** 37. **echo** "用户 [$1] 创建成功, 相关信息如下：" 38. **id** $1 39. **else** 40. **echo** 41. **echo** "用户 [$1] 创建失败，可能是权限不足" 42. **fi** 43. **fi** 44. **}** 46. *<<comment* 47. *$1 用户名* 48. *判断$1用户是否使用默认密码（123456）* 49. *是则修改密码* 50. *comment* 51. **function** change\_password**(){** 52. *# query=$(cat /etc/passwd| grep ^$1:| grep "initial password")* 53. query=$**(cat** conf**|** **grep** $1**)** 54. **if** **[[** -n $query **]]**; **then** 55. **echo** 56. **echo** "您的默认密码为123456，过于简单，请修改密码！" 57. **passwd** 58. **if** **[** $? -eq 0 **]**; **then** 59. **sed** -i "/$1/d" conf 60. **echo** 61. **echo** "密码修改成功" 62. **else** 63. **echo** 64. **echo** "密码修改失败，请重新登录系统！" 65. **exit** 1 66. **fi** 67. **fi** 68. **}** 70. *<<comment* 71. *$1 用户名* 72. *清空屏幕，显示菜单* 73. *comment* 74. **function** show\_menu**(){** 75. *# clear* 76. **echo** "用户 [$1] 系统：" 77. **echo** "（0）清空屏幕，显示目录" 78. **echo** "（1）显示用户选择的目录内容" 79. **echo** "（2）按照用户输入的目录切换路径" 80. **echo** "（3）按照用户输入的文件名在/home目录中创建文件" 81. **echo** "（4）编辑用户输入的文件" 82. **echo** "（5）删除用户选择的文件" 83. **}** 85. **function** operation\_1**(){** 86. CURRENT\_PATH=$**(pwd)** 87. **echo** 88. **read** -p "[$1] [$CURRENT\_PATH] 请输入目录：" DIR\_PATH 90. **if** **[** -d $DIR\_PATH **]**; **then** 91. **echo** 92. **ls** $DIR\_PATH 93. **else** 94. **echo** 95. **echo** "[$1] [$CURRENT\_PATH] 目录 $DIR\_PATH 不存在！" 96. **fi** 97. **}** 99. **function** operation\_2**(){** 100. CURRENT\_PATH=$**(pwd)** 101. **echo** 102. **read** -p "[$1] [$CURRENT\_PATH] 请输入要切换的路径：" DIR\_PATH 104. **if** **[** -d $DIR\_PATH **]**; **then** 105. **cd** $DIR\_PATH 106. **else** 107. **echo** 108. **echo** "[$1] [$CURRENT\_PATH] 目录 $DIR\_PATH 不存在！" 109. **fi** 110. **}** 112. **function** operation\_3**(){** 113. *# 注意，这里还没有对文件名合法性做校验* 114. *# 已知bug：文件名输入///，显示创建成功，但其实没有成功* 115. *# ！！！* 116. *# ！！！* 117. CURRENT\_PATH=$**(pwd)** 118. **echo** 119. **read** -p "[$1] [$CURRENT\_PATH] 请输入要创建的文件名：" FILE\_NAME 121. **if** **[** **!** -f "/home/$1/$FILE\_NAME" **]**; **then** 122. **touch** "/home/$1/$FILE\_NAME" 123. **echo** 124. **echo** "[$1] [$CURRENT\_PATH] 文件 $FILE\_NAME 创建成功！" 125. **else** 126. **echo** 127. **echo** "[$1] [$CURRENT\_PATH] 文件 $FILE\_NAME 已经存在，无法创建！" 128. **fi** 129. **}** 131. **function** operation\_4**(){** 132. CURRENT\_PATH=$**(pwd)** 133. **echo** 134. **read** -p "[$1] [$CURRENT\_PATH] 请输入要编辑的文件路径：" FILE\_PATH 136. **if** **[** -f "$FILE\_PATH" **]**; **then** 137. **vim** "$FILE\_PATH" 138. *# echo* 139. *# echo "[$1] [$CURRENT\_PATH] 文件 $FILE\_PATH 创建成功！"* 140. **else** 141. **echo** 142. **echo** "[$1] [$CURRENT\_PATH] 文件 $FILE\_PATH 不存在，无法编辑" 143. **fi** 144. **}** 146. **function** operation\_5**(){** 147. CURRENT\_PATH=$**(pwd)** 148. **echo** 149. **read** -p "[$1] [$CURRENT\_PATH] 请输入要删除的文件路径：" FILE\_PATH 151. **if** **[** -f "$FILE\_PATH" **]**; **then** 152. **rm** -r "$FILE\_PATH" 153. **echo** 154. **echo** "[$1] [$CURRENT\_PATH] 文件 $FILE\_PATH 删除成功！" 155. **else** 156. **echo** 157. **echo** "[$1] [$CURRENT\_PATH] 文件 $FILE\_PATH 不存在，无法删除" 158. **fi** 159. **}**  162. *<<comment* 163. *$1 当前登陆系统的用户* 164. *系统的主函数* 165. *comment* 166. **function** system\_main**(){** 167. USER\_NAME=$1 168. **echo** 169. **echo** "[$USER\_NAME] login the system" 171. change\_password $USER\_NAME 173. **clear** 174. show\_menu $USER\_NAME 176. **while** : 177. **do** 178. CURRENT\_PATH=$**(pwd)** 179. **echo** 180. **read** -p "[$USER\_NAME] [$CURRENT\_PATH] 请选择操作类型：" CHOICE 181. **case** "${CHOICE}" **in** 182. 0**)** 183. **clear** 184. show\_menu $USER\_NAME 185. **;;** 186. 1**)** 187. operation\_1 $USER\_NAME 188. **;;** 189. 2**)** 190. operation\_2 $USER\_NAME 191. **;;** 192. 3**)** 193. operation\_3 $USER\_NAME 194. **;;** 195. 4**)** 196. operation\_4 $USER\_NAME 197. **;;** 198. 5**)** 199. operation\_5 $USER\_NAME 200. **;;** 201. **\*)** 202. **echo** "default (none of above)" 203. **;;** 204. **esac** 205. **done** 206. **}**  209. *<<comment* 210. *程序入口* 211. *comment* 212. **if** **[** $# -eq 1 **]**; **then** 213. check\_user\_existence $1 214. **else** 215. system\_main $**(whoami)** 216. **fi** | | |
|  |
|  |
|  |
|  |
|  |

* 1. 题目三
     1. 题目要求

分别编写基于TCP和UDP的socket程序，（1）实现加减乘除和求幂（任选其一）的运算服务（TCP）；（2）实现即时聊天（UDP）。给出全部程序代码和执行结果截图（30分18+12）

* + 1. 题目作答

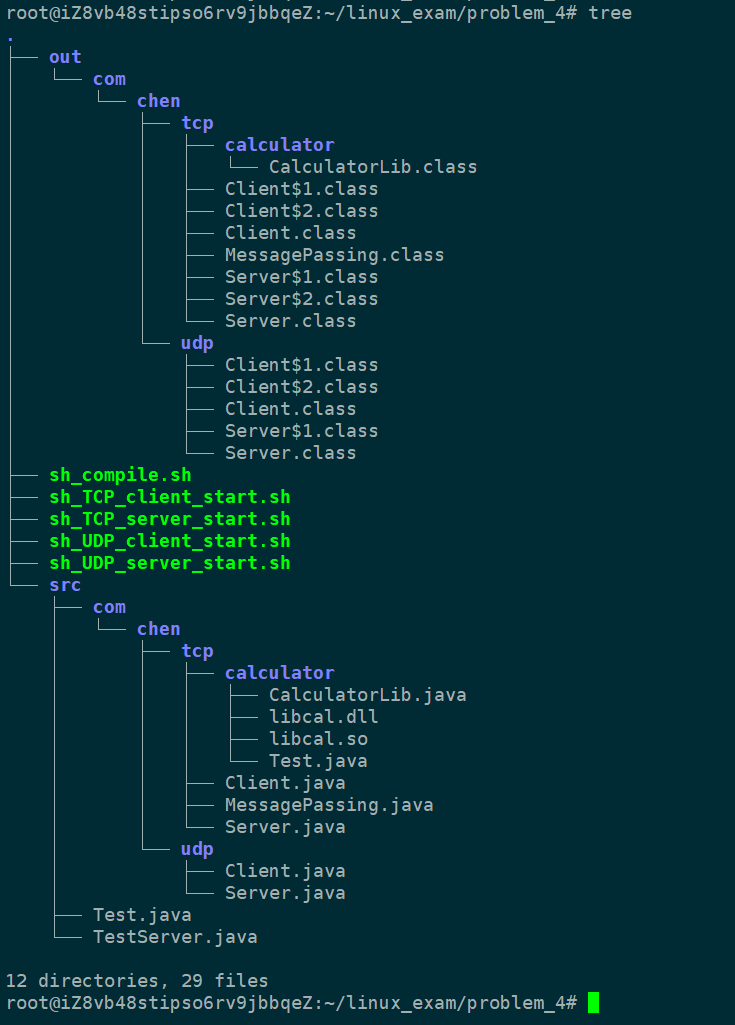
本题目使用Java语言，使用java.net.\*包中的相关方法。完整的java项目目录树如下图：

图3-1 项目的文件组织

src目录下存放全部源代码，其中，包com.chen.tcp下存放了TCP运算服务的全部源代码文件，包com.chen.udp下存放了UDP即时聊天的全部源代码文件；out目录下存放javac编译后的字节码文件；sh\_compile.sh脚本用来编译全部源代码文件；sh\_TCP\_client\_start.sh与sh\_TCP\_server\_start.sh是启动TCP运算的服务器与客户端脚本；sh\_UDP\_client\_start.sh与sh\_UDP\_server\_start.sh是启动UDP即时聊天的服务器与客户端脚本。（**这里再次体会到学习Linux Shell编程的用处之大！**）

* + - 1. 子问题(1)：基于TCP协议的运算服务

本项目的服务端部署到绑定了公网IP地址的阿里云服务器上，实现了任何机器运行客户端程序均可成功访问到服务端。项目允许多个待计算客户与服务器通信，获取计算结果；服务器对客户端发送的待计算表达式并没有采用传统的中缀表达式转后缀表达式、基于栈对后缀表达式计算的技术，而是从更偏向底层的编译原理角度完成表达式的计算——使用flex与bison完成了输入表达式的词法分析、语法分析，基于C语言，对表达式编译层面的语义动作信息进行解析，完成表达式计算。该部分代码在系统软件开发课程中实现。最后，对实现表达式计算的全部C文件打包成动态链接库文件，移植到Linux平台，使用Java调用动态链接库，完成了服务端表达式的计算。

服务端的高层逻辑采用多线程技术。由于TCP协议需要三次握手才可以进行通信，即先建立连接再通信，因此需要一个线程不断监听是否有客户端想要加入聊天室。同时，还需要一个线程不断接受全部客户端发来的消息并转发给其余客户端。

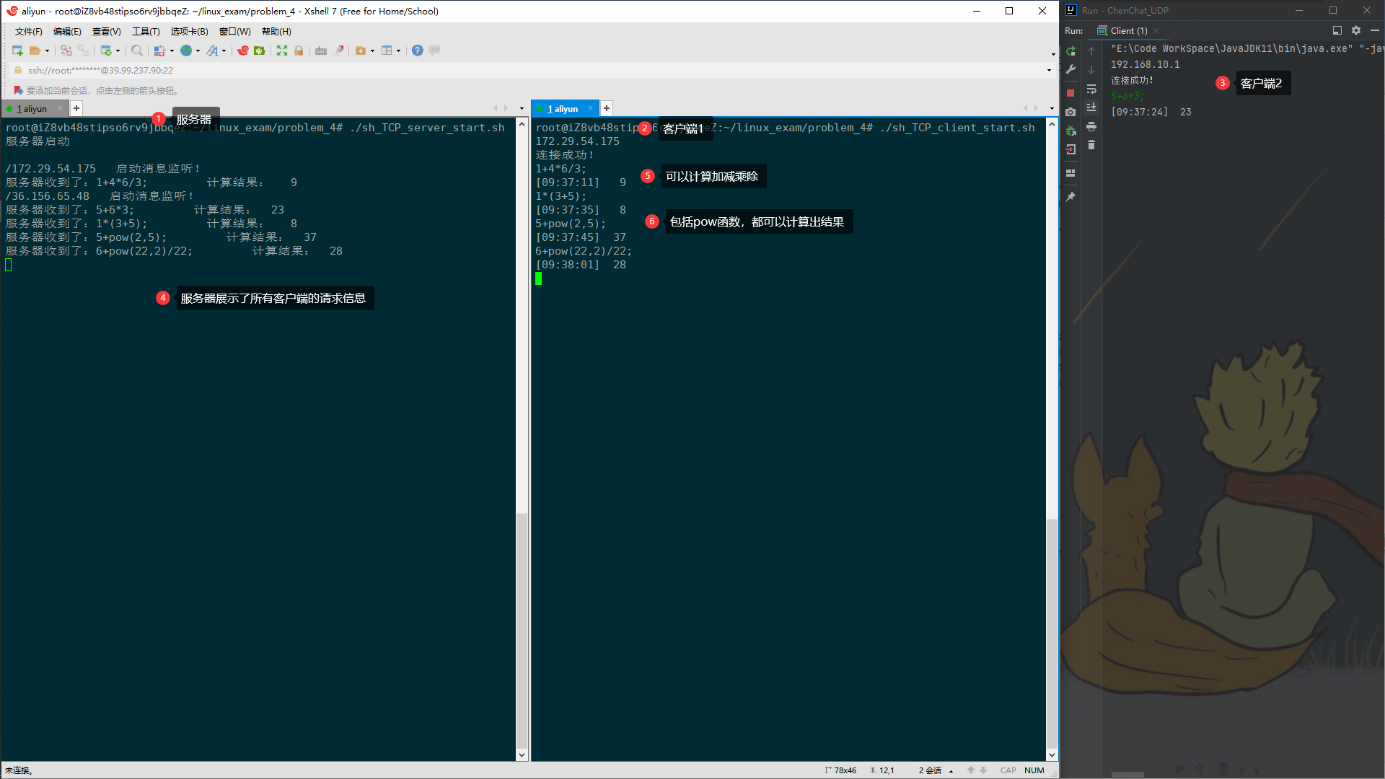
部分运行结果如下。可以计算简单加减乘除、括号以及pow函数等。

图3-2 运行结果

全部代码包括启动的shell脚本如下：

|  |  |  |
| --- | --- | --- |
| *Clinet.java* |  |  |
| 1. **package** com.chen.tcp; 3. **import** java.io.IOException; 4. **import** java.net.InetAddress; 5. **import** java.net.Socket; 6. **import** java.util.Scanner; 8. **public** **class** Client { 10. ***/\*\**** 11. ***\* 客户端套接字*** 12. ***\*/*** 13. **private** Socket socket; 15. ***/\*\**** 16. ***\* 与客户端绑定的MP*** 17. ***\*/*** 18. MessagePassing mp; 20. **public** Client(String ip, **int** port) **throws** IOException { 21. *// 套接字连接* 22. socket = **new** Socket(InetAddress.getByName(ip), port); 23. *// 消息处理绑定流* 24. mp = **new** MessagePassing(socket.getInputStream(), socket.getOutputStream(), socket); 26. System.out.println("连接成功！"); 27. } 29. ***/\*\**** 30. ***\* 为当前客户端创建一个发送消息线程*** 31. ***\* @return 发送消息线程*** 32. ***\*/*** 33. **private** Thread threadSend(){ 34. Thread thread = **new** Thread(**new** Runnable() { 35. @Override 36. **public** **void** run() { 37. Scanner sc = **new** Scanner(System.in); 38. **while**(**true**){ 39. String message = sc.next(); 40. **try** { 41. mp.send(message, **false**); 42. } **catch** (Exception e){ 43. e.printStackTrace(); 44. } 45. } 46. } 47. }); 48. **return** thread; 49. } 51. ***/\*\**** 52. ***\* 为当前客户端创建一个接收消息线程*** 53. ***\* @return 接收消息线程*** 54. ***\*/*** 55. **private** Thread threadReceive(){ 56. Thread thread = **new** Thread(**new** Runnable() { 57. @Override 58. **public** **void** run() { 59. **while**(**true**){ 60. String message = **null**; 61. **try** { 62. message = mp.receive(); 63. } **catch** (IOException e) { 64. e.printStackTrace(); 65. } 66. System.out.println(message); 67. } 68. } 69. }); 70. **return** thread; 71. } 73. ***/\*\**** 74. ***\* 客户端启动*** 75. ***\*/*** 76. **public** **void** begin(){ 77. threadSend().start(); 78. threadReceive().start(); 79. } 81. **public** **static** **void** main(String[] args) **throws** IOException { 82. String serverIp = "39.99.237.90"; 83. System.out.println(InetAddress.getLocalHost().getHostAddress()); 84. Client client = **new** Client(serverIp, 50001); 85. client.begin(); 86. } 88. } | | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| *Server.java* |  |  |
| 1. **package** com.chen.tcp; 3. **import** com.chen.tcp.calculator.CalculatorLib; 5. **import** java.io.IOException; 6. **import** java.io.InputStream; 7. **import** java.net.ServerSocket; 8. **import** java.net.Socket; 9. **import** java.util.HashMap; 10. **import** java.util.HashSet; 11. **import** java.util.Map; 12. **import** java.util.Set; 14. **public** **class** Server { 16. ***/\*\**** 17. ***\* 服务器套接字*** 18. ***\*/*** 19. **private** ServerSocket serverSocket; 21. ***/\*\**** 22. ***\* 存储连接到服务器的所有客户端的socket与相应的MessagePassing*** 23. ***\*/*** 24. Map<Socket, MessagePassing> clients; 26. **public** Server() **throws** IOException { 27. serverSocket = **new** ServerSocket(50001); 28. clients = **new** HashMap<>(); 29. System.out.println("服务器启动**\n**"); 30. } 32. ***/\*\**** 33. ***\* 启动服务器*** 34. ***\*/*** 35. **public** **void** begin(){ 36. *//开始监听客户端的连接请求* 37. threadAccept().start(); 38. } 40. ***/\*\**** 41. ***\* 服务器中用于不断监听是否有新的客户端请求连接的进程*** 42. ***\* @return 该进程*** 43. ***\*/*** 44. **private** Thread threadAccept(){ 45. Thread thread = **new** Thread(**new** Runnable(){ 46. @Override 47. **public** **void** run(){ 48. **while**(**true**) { 49. Socket oneClient = **null**; 50. **try** { 51. oneClient = serverSocket.accept(); 52. } **catch** (IOException e) { 53. e.printStackTrace(); 54. } 55. MessagePassing mp = **null**; 56. **try** { 57. mp = **new** MessagePassing(oneClient.getInputStream(), oneClient.getOutputStream(), oneClient); 58. } **catch** (IOException e) { 59. e.printStackTrace(); 60. } 61. *//加入服务器当前管理的客户端集合* 62. clients.put(oneClient, mp); 63. *//启动对客户端发来消息的监听* 64. threadReceive(oneClient, mp).start(); 65. System.out.println(oneClient.getLocalSocketAddress() + " 连接服务器"); 66. } 67. } 68. }); 69. **return** thread; 70. } 72. ***/\*\**** 73. ***\* 服务器为当前客户端创建一个接收消息线程，一旦接收到客户端的消息，则立即转发给其他客户。即接受->转发*** 74. ***\* @param currentClientSocket 当前客户端*** 75. ***\* @param currentMP 当前客户端的MessagePassing*** 76. ***\* @return 该线程*** 77. ***\*/*** 78. **private** Thread threadReceive(Socket currentClientSocket, MessagePassing currentMP){ 79. Thread thread = **new** Thread(**new** Runnable() { 80. @Override 81. **public** **void** run() { 82. System.out.println(currentClientSocket.getInetAddress() + " 启动消息监听！"); 83. **boolean** isClosed = **false**; 84. **while**(!isClosed){ 85. *// 接收消息* 86. String message = **null**; 87. **try** { 88. message = currentMP.receive(); 89. System.out.print("服务器收到了：" + message); 90. String ans = CalculatorLib.lib.do\_cal(message + "**\n**"); 91. System.out.println(" 计算结果：" + ans); 92. *// 转发消息* 93. sendToClient(ans, currentClientSocket); 94. } **catch** (IOException e) { 95. System.out.println(currentClientSocket.getLocalSocketAddress() + " 与服务器连接断开！"); 96. *//退出循环* 97. isClosed = **true**; 98. *//关闭资源* 99. **try** { 100. currentMP.resourceClose(); 101. } **catch** (IOException ex) { 102. System.out.println("资源关闭错误"); 103. ex.printStackTrace(); 104. } 105. *//删除该客户端* 106. clients.remove(currentClientSocket); 107. } 108. } 109. } 110. }); 111. **return** thread; 112. } 114. ***/\*\**** 115. ***\* 将服务器接收到的消息转发给其他所有客户端。*** 116. ***\* 这里将字节流解码为utf8，再编码成字节流，因此可以加以优化*** 117. ***\* @param message 接收到的消息*** 118. ***\* @param socket 发送message的客户端*** 119. ***\*/*** 120. **private** **void** sendToClient(String message, Socket socket){ 121. **try** { 122. clients.get(socket).send(message, **true**); 123. } **catch** (Exception e){ 124. System.out.println("计算错误！"); 125. } 126. } 128. **public** **static** **void** main(String[] args) **throws** IOException { 129. Server server = **new** Server(); 130. server.begin(); 131. } 133. } | | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| *MessagePassing.java* |  |  |
| 1. **package** com.chen.tcp; 3. **import** java.io.IOException; 4. **import** java.io.InputStream; 5. **import** java.io.OutputStream; 6. **import** java.net.Socket; 7. **import** java.nio.charset.StandardCharsets; 8. **import** java.text.SimpleDateFormat; 9. **import** java.util.Date; 11. **public** **class** MessagePassing { 13. **private** InputStream is; 14. **private** OutputStream os; 15. **private** Socket socket; 17. ***/\*\**** 18. ***\* 使用输入流、输出流构造MP*** 19. ***\* @param is 输入流，接受消息*** 20. ***\* @param os 输出流，发送消息*** 21. ***\*/*** 22. **public** MessagePassing(InputStream is, OutputStream os, Socket socket){ 23. **this**.is = is; 24. **this**.os = os; 25. **this**.socket = socket; 26. } 28. ***/\*\**** 29. ***\* 将message发送给当前TCP另一端*** 30. ***\* @param message 消息*** 31. ***\* @param needFormatting true则标识消息需要格式化，加上时间戳。*** 32. ***\* 否则不加入时间戳*** 33. ***\* @throws IOException 当连接断开后抛出异常。*** 34. ***\*/*** 35. **public** **void** send(String message, **boolean** needFormatting) **throws** IOException { 36. **if**(needFormatting) 37. message = messageFormatting(message); 38. **byte**[] buffer = message.getBytes(StandardCharsets.UTF\_8); 39. os.write(buffer); 40. *//socket.shutdownOutput();* 41. } 43. **public** String receive() **throws** IOException { 44. **byte**[] buffer = **new** **byte**[1024]; 45. **int** len = is.read(buffer); 46. String ans = **new** String(buffer, 0, len, StandardCharsets.UTF\_8); 47. **return** ans; 48. } 50. ***/\*\**** 51. ***\* 将消息格式化，加上时间戳。*** 52. ***\* @param message 消息*** 53. ***\* @return 格式化后的消息*** 54. ***\*/*** 55. **private** String messageFormatting(String message){ 56. Date dNow = **new** Date( ); 57. SimpleDateFormat ft = **new** SimpleDateFormat ("hh:mm:ss"); 58. **return** "[" + ft.format(dNow) + "]" + message; 59. } 61. ***/\*\**** 62. ***\* 关闭该socket及其输入输出流资源*** 63. ***\* @throws IOException 如果资源关闭失败，则抛出*** 64. ***\*/*** 65. **public** **void** resourceClose() **throws** IOException { 66. is.close(); 67. os.close(); 68. socket.close(); 69. } 70. } | | |
|  |
|  |
|  |
|  |
|  |

以下是编译与启动的shell脚本文件。

|  |  |  |
| --- | --- | --- |
| *compile.sh* |  |  |
| 1. BASE\_PATH="./src/com/chen/" 2. LIB\_PATH="/usr/share/maven/repo/net/java/dev/jna/jna/5.10.0/jna-5.10.0.jar" 3. OUT\_PATH="./out/" 5. FILE1="${BASE\_PATH}tcp/MessagePassing.java" 6. FILE2="${BASE\_PATH}tcp/Server.java" 7. FILE3="${BASE\_PATH}tcp/calculator/CalculatorLib.java" 8. FILE4="${BASE\_PATH}udp/Client.java" 9. FILE5="${BASE\_PATH}udp/Server.java" 11. **javac** -classpath $LIB\_PATH $FILE1 $FILE2 $FILE3 $FILE4 $FILE5 -d $OUT\_PATH 12. **echo** "编译成功！" | | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| *sh\_TCP\_client\_start.sh* |  |  |
| 1. OUT\_PATH="./out/" 2. LIB\_PATH="/usr/share/maven/repo/net/java/dev/jna/jna/5.10.0/jna-5.10.0.jar" 4. **cd** $OUT\_PATH 5. **java** -classpath .:$LIB\_PATH com.chen.tcp.Client | | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| *sh\_TCP\_server\_start.sh* |  |  |
| 1. OUT\_PATH="./out/" 2. LIB\_PATH="/usr/share/maven/repo/net/java/dev/jna/jna/5.10.0/jna-5.10.0.jar" 4. **cd** $OUT\_PATH 5. **java** -classpath .:$LIB\_PATH com.chen.tcp.Server | | |
|  |
|  |
|  |
|  |
|  |

* + - 1. 子问题(2)：基于UDP协议的即时聊天

本项目同样使用Java语言的Socket编程。为了实现即时的群聊聊天系统，同样需要使用多线程技术。UDP协议不需要建立连接便可以直接向目标套接字发送信息，因此相比于TCP协议而言，可以不需要使用额外线程来监听连接请求。

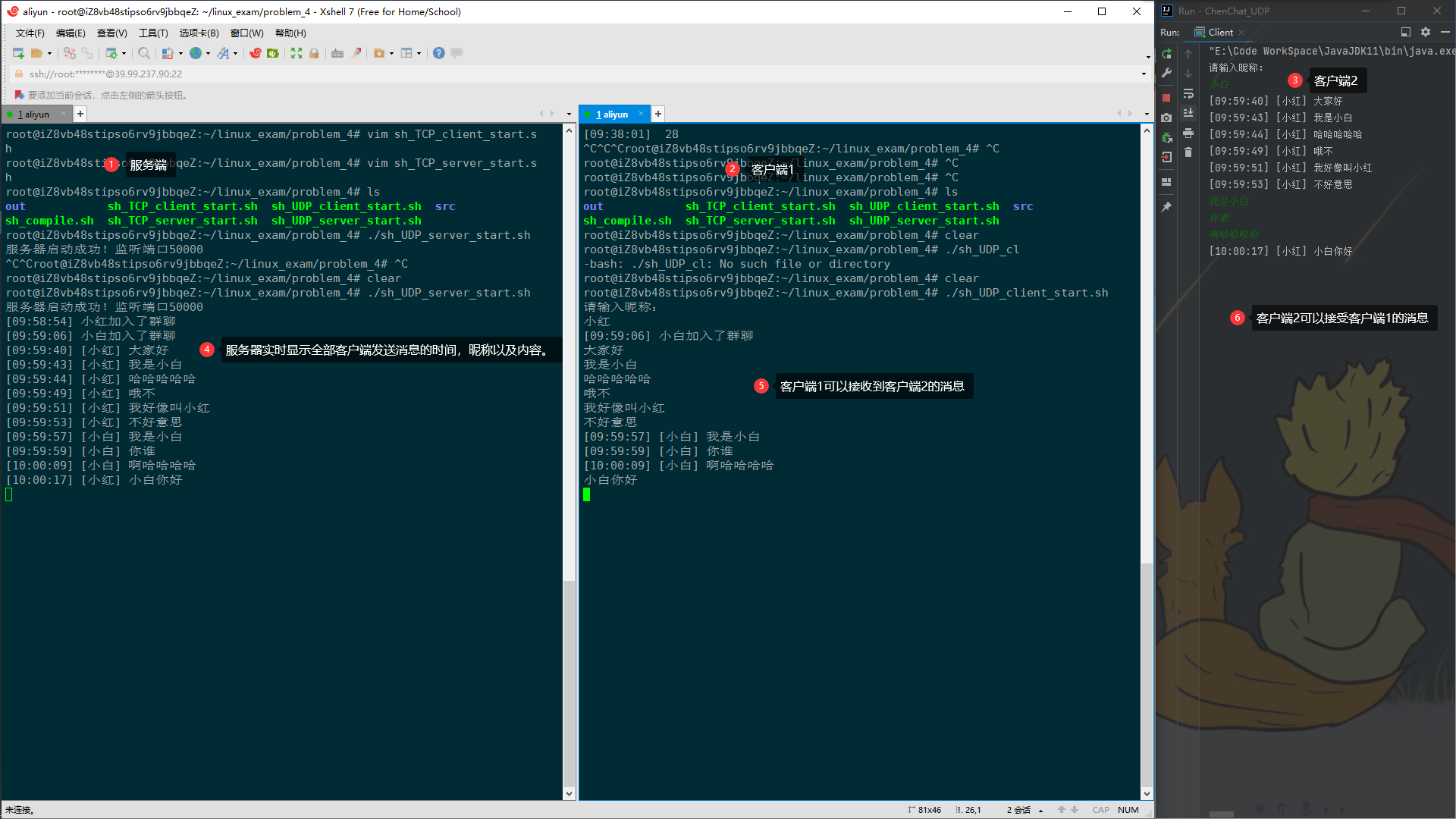
项目的运行截图如下：

图3-3 运行结果

项目的全部代码以及shell脚本如下：

|  |  |  |
| --- | --- | --- |
| *Client.java* |  |  |
| 窗体顶端   1. **package** com.chen.udp; 3. **import** java.io.IOException; 4. **import** java.net.DatagramPacket; 5. **import** java.net.DatagramSocket; 6. **import** java.net.InetAddress; 7. **import** java.nio.charset.StandardCharsets; 8. **import** java.text.SimpleDateFormat; 9. **import** java.util.Date; 10. **import** java.util.Scanner; 12. **public** **class** Client { 14. String serverIp = "39.99.237.90"; 16. DatagramSocket clientSocket = **new** DatagramSocket(); 18. String name; 20. Client() **throws** Exception { 21. System.out.println("请输入昵称："); 22. name = **new** Scanner(System.in).next(); 23. } 25. **public** **void** begin() **throws** Exception{ 26. sendWelcomeMessage(); 27. threadRecieve().start(); 28. threadSend().start(); 29. } 31. **private** **void** sendWelcomeMessage(){ 32. **try** { 33. Date dNow = **new** Date( ); 34. SimpleDateFormat ft = **new** SimpleDateFormat ("hh:mm:ss"); 35. String msg = "[" + ft.format(dNow) + "] " + name + "加入了群聊"; 36. **byte**[] bys = msg.getBytes(StandardCharsets.UTF\_8); 37. DatagramPacket dp = **new** DatagramPacket(bys, bys.length, InetAddress.getByName(serverIp), 50000); 38. clientSocket.send(dp); 39. } **catch** (Exception e){ 40. System.out.println("消息发送失败！"); 41. } 42. } 44. **private** **void** sendOneMessage(String msg){ 45. **try** { 46. **byte**[] bys = messageFormatting(msg).getBytes(StandardCharsets.UTF\_8); 47. DatagramPacket dp = **new** DatagramPacket(bys, bys.length, InetAddress.getByName(serverIp), 50000); 48. clientSocket.send(dp); 49. } **catch** (Exception e){ 50. System.out.println("消息发送失败！"); 51. } 52. } 54. **private** String recieveOneMessage(){ 55. **byte**[] bys = **new** **byte**[1024]; 56. DatagramPacket dp = **new** DatagramPacket(bys, bys.length); 57. **try** { 58. clientSocket.receive(dp); 59. } **catch** (IOException e) { 60. e.printStackTrace(); 61. } 62. **return** **new** String(dp.getData(), 0, dp.getLength(), StandardCharsets.UTF\_8); 63. } 65. **private** Thread threadRecieve(){ 66. Thread thread = **new** Thread(**new** Runnable() { 67. @Override 68. **public** **void** run() { 69. **while**(**true**) { 70. String msg = recieveOneMessage(); 71. System.out.println(msg); 72. } 73. } 74. }); 75. **return** thread; 76. } 78. **private** Thread threadSend(){ 79. Thread thread = **new** Thread(**new** Runnable() { 80. @Override 81. **public** **void** run() { 82. Scanner scanner = **new** Scanner(System.in); 83. **while**(**true**){ 84. String msg = scanner.nextLine(); 85. sendOneMessage(msg); 86. } 87. } 88. }); 89. **return** thread; 90. } 92. **private** String messageFormatting(String message){ 93. Date dNow = **new** Date( ); 94. SimpleDateFormat ft = **new** SimpleDateFormat ("hh:mm:ss"); 95. **return** "[" + ft.format(dNow) + "]" + " [" + name + "]" + " " + message; 96. } 98. **public** **static** **void** main(String[] args) **throws** Exception{ 99. Client client = **new** Client(); 100. client.begin(); 101. } 103. }窗体底端 | | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| *Server.java* |  |  |
| 1. **package** com.chen.udp; 3. **import** java.io.IOException; 4. **import** java.net.DatagramPacket; 5. **import** java.net.DatagramSocket; 6. **import** java.net.InetAddress; 7. **import** java.nio.charset.StandardCharsets; 8. **import** java.util.\*; 10. **public** **class** Server { 12. ***/\*\**** 13. ***\* 服务器套接字*** 14. ***\*/*** 15. DatagramSocket serverSocket = **new** DatagramSocket(50000); 17. Vector<InetAddress> ips; 18. Vector<Integer> ports; 20. Server() **throws** Exception { 21. ips = **new** Vector<>(); 22. ports = **new** Vector<>(); 23. System.out.println("服务器启动成功！监听端口50000"); 24. } 26. **public** **void** begin() **throws** Exception{ 27. threadRecieve().start(); 28. } 30. **private** Thread threadRecieve(){ 31. Thread thread = **new** Thread(**new** Runnable() { 32. @Override 33. **public** **void** run() { 34. **while**(**true**) { 35. **byte**[] bys = **new** **byte**[1024]; 36. DatagramPacket dp = **new** DatagramPacket(bys, bys.length); 37. String msg = recieveOneMessage(dp); 38. System.out.println(msg); 39. sendToAllClients(msg, dp); 40. } 41. } 42. }); 43. **return** thread; 44. } 46. **private** String recieveOneMessage(DatagramPacket dp){ 47. *//byte[] bys = new byte[1024];* 48. *//dp = new DatagramPacket(bys, bys.length);* 49. **try** { 50. serverSocket.receive(dp); 51. } **catch** (IOException e) { 52. e.printStackTrace(); 53. System.out.println("消息接受失败"); 54. } 55. addClient(dp.getAddress(), dp.getPort()); 56. **return** **new** String(dp.getData(), 0, dp.getLength(), StandardCharsets.UTF\_8); 57. } 59. **private** **void** addClient(InetAddress ip, **int** port){ 60. **boolean** flag = **true**; 61. **for**(**int** i = 0; i < ips.size(); i++){ 62. **if**(ips.get(i).equals(ip) && ports.get(i).equals(port)){ 63. flag = **false**; 64. **break**; 65. } 66. } 67. **if**(flag){ 68. ips.add(ip); 69. ports.add(port); 70. } 71. } 73. **private** **void** sendOneMessage(String msg, InetAddress ip, **int** port){ 74. **try** { 75. **byte**[] bys = msg.getBytes(StandardCharsets.UTF\_8); 76. DatagramPacket dp = **new** DatagramPacket(bys, bys.length, ip, port); 77. serverSocket.send(dp); 78. } **catch** (Exception e){ 79. System.out.println("消息发送失败"); 80. } 81. } 83. **private** **void** sendToAllClients(String msg, DatagramPacket dp){ 84. **for**(**int** i = 0; i < ips.size(); i++){ 85. **if**(ips.get(i).equals(dp.getAddress()) && ports.get(i).equals(dp.getPort())) 86. **continue**; 87. sendOneMessage(msg, ips.get(i), ports.get(i)); 88. } 89. } 91. **public** **static** **void** main(String[] args) **throws** Exception{ 92. Server server = **new** Server(); 93. server.begin(); 94. } 95. } | | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| *sh\_UDP\_server\_start.sh* |  |  |
| 1. OUT\_PATH="./out/" 2. LIB\_PATH="/usr/share/maven/repo/net/java/dev/jna/jna/5.10.0/jna-5.10.0.jar" 4. **cd** $OUT\_PATH 5. **java** -classpath .:$LIB\_PATH com.chen.udp.Server | | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| *sh\_UDP\_client\_start.sh* |  |  |
| 1. OUT\_PATH="./out/" 2. LIB\_PATH="/usr/share/maven/repo/net/java/dev/jna/jna/5.10.0/jna-5.10.0.jar" 4. **cd** $OUT\_PATH 5. **java** -classpath .:$LIB\_PATH com.chen.udp.Client | | |
|  |
|  |
|  |
|  |
|  |

* 1. 题目四
     1. 题目要求

编写一个显示“Hello，World！”的欢迎语程序，再编写一个分别采用fork（）、exec（）、exit（）、wait（）调用其执行和适时等待和退出的程序，体会四个函数对进程创建、调度（阻塞、关闭）释放资源的过程和作用。给出全部程序代码和执行结果截图（20分）

* + 1. 题目作答

本题目通过编写fork、execl、exit函数各自的测试程序，熟练掌握进程创建、退出的原理。最后，通过这些函数以及wait函数的组合使用，深入理解了进程执行、适时等待、退出以及调度释放资源的过程与作用。

* + - 1. fork函数

函数用于创建进程，以下是手册中对fork函数的描述：

|  |
| --- |
| *fork() creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.* |

函数原型如下：

|  |
| --- |
| #include <sys/types.h>  #include <unistd.h>    pid\_t fork(void); |
|  |

从手册中得知，子进程创建成功的话，函数存在两个返回值。在父进程中返回值大于0，返回子进程ID；而在子进程中返回值等于零。

使用函数创建的子进程与父进程运行在分开的内存地址空间中，对数据的读写不会相互影响。

*Memory writes, file mappings (mmap(2)), and unmappings (munmap(2)) performed by one of the processes do not affect the other.*

由此编写了如下的欢迎语程序。

|  |  |  |
| --- | --- | --- |
| fork.c |  |  |
| 1. #include <sys/types.h> 2. #include <unistd.h> 3. #include <stdio.h> 5. int main() 6. { 7. pid\_t pid = fork(); 8. if(pid > 0) { 9. printf("[parent][pid %d][ppid %d] Hello world!**\n**", getpid(), getppid()); 10. } else if(pid == 0) { 11. printf("[child] [pid %d][ppid %d] Hello world!**\n**", getpid(), getppid()); 12. } 14. for(int i = 0; i < 3; i++){ 15. printf("[pid %d][ppid %d] Hello world!**\n**", getpid(), getppid()); 16. sleep(1); 17. } 18. sleep(1); 19. return 0; 20. } | | |
|  |
|  |
|  |
|  |
|  |

调用函数，会有两个返回值。根据返回值的不同，父子进程分别打印自己的进程号pid以及各自的父进程号ppid。最后，两个进程均执行行14~17的for循环语句。程序的输出如下。

图4-1 运行结果

通过每一行输出语句前面的两个中括号，可以看出改行语句是子进程打印的还是父进程打印的。

* + - 1. exec函数（族）

exec()函数族会去在当前进程中调用另一个进程，执行新的进程的指令。其函数所需的头文件以及声明如下：

|  |
| --- |
| #include <unistd.h>  **extern** char \*\*environ;    int execl(const char \*path, const char \*arg, ...);  int execlp(const char \*file, const char \*arg, ...); |
|  |

根据手册，只有调用失败才会具有返回值-1.

*The exec() functions return only if an error has occurred. The return value is -1, and errno is set to indicate the error.*

当函数调用成功时，会使用新进程的指令覆盖当前进程，故当前进程剩余代码不执行，因此也就不需要返回值了。

因为exec函数族的这一特点，一般是在子进程中使用exec函数族。先使用fork创建子进程，在子进程中使用exec()函数族函数，从而使得子进程执行我们想要执行的可执行文件。

excel()会执行path指定的可执行文件，但是excelp则会到环境变量下寻找file指定的可执行文件文件名。

编写的exec.c文件与exec.sh脚本如下：

|  |  |  |
| --- | --- | --- |
| *exec.sh* |  |  |
| 1. *#!/bin/bash* 3. *<<comment* 4. *获取当前进程pid* 5. *comment* 6. **function** getpid**(){** 7. **ps** -ef**|** **grep** -m 1 "exec.sh"**|** **grep** -v "grep"**|** **awk** '{print $2}' 8. **}** 10. *<<comment* 11. *获取当前进程的父进程ppid* 12. *comment* 13. **function** getppid**(){** 14. **ps** -ef**|** **grep** -m 1 "exec.sh"**|** **grep** -v "grep"**|** **awk** '{print $3}' 15. **}** 17. **for** i **in** $**(seq** 1 3**)**; **do** 18. pid=$**(**getpid**)** 19. ppid=$**(**getppid**)** 20. **echo** "I am in exec.sh, not a .c! [pid $pid][ppid $ppid] Hello world!" 21. **sleep** 1 22. **done** | | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |
| --- | --- | --- |
| *exec.c* |  |  |
| 1. #include <unistd.h> 2. #include <stdio.h> 3. #include <sys/types.h> 5. int main() 6. { 7. pid\_t pid = fork(); 8. if(pid > 0) { 9. printf("[parent][pid %d][ppid %d] Hello world!**\n**", getpid(), getppid()); 10. } else if(pid == 0) { 11. printf("[child] [pid %d][ppid %d] Hello world!**\n**", getpid(), getppid()); 12. execl("./exec.sh", "exec", NULL); 13. } 15. for(int i = 0; i < 3; i++){ 16. printf("I am in exec.c, not a shell! [pid %d][ppid %d] Hello world!**\n**", getpid(), getppid()); 17. sleep(1); 18. } 19. sleep(1); 20. return 0; 21. } | | |
|  |
|  |
|  |
|  |
|  |

在exec.c中，使用fork函数创建子进程，父子进程各自打印自己的进程号与父进程号。随后子进程调用execl函数，执行脚本exec.sh。

主进程进入exec.c中行15~18的for循环，打印三次Hello world；而子进程进入脚本exec.sh中，同样打印三次Hello world。但由于exec函数族的特性，子进程使用脚本中的语句覆盖当前的后文内容，因此子进程在执行完exec.sh后，并不会打印exec.c行15~18的内容了。执行结果如下图：

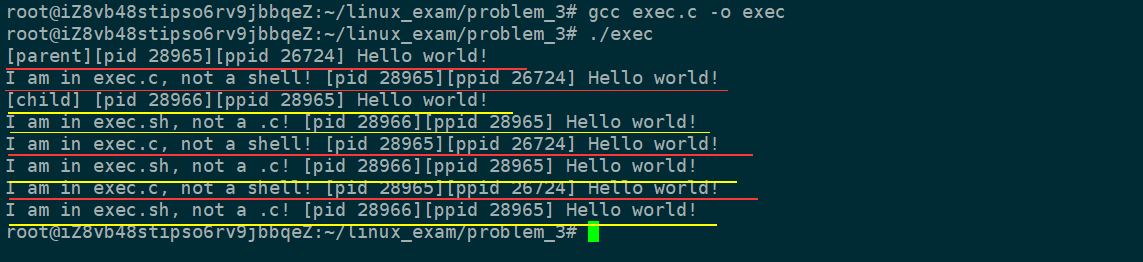


图4-2 运行结果

图4-2中，红线所标识的为父进程打印内容，黄线所标识的是子进程打印内容。可见，由于子进程打印了三条Hello world语句，由此可见子进程使用脚本exec.sh中的内容完全覆盖了exec.c文件的后文。

* + - 1. exit函数

退出进程有两个函数可以实现功能——exit与\_exit。其函数原型如下：

|  |
| --- |
| #include <stdlib.h>  void exit(int status);    #include <unistd.h>  void \_exit(int status); |
|  |

其中，exit是标准C库中的函数，而\_exit则是Linux的系统函数。参数status为退出进程时的状态，可以由父进程捕获从而回收子进程的资源。

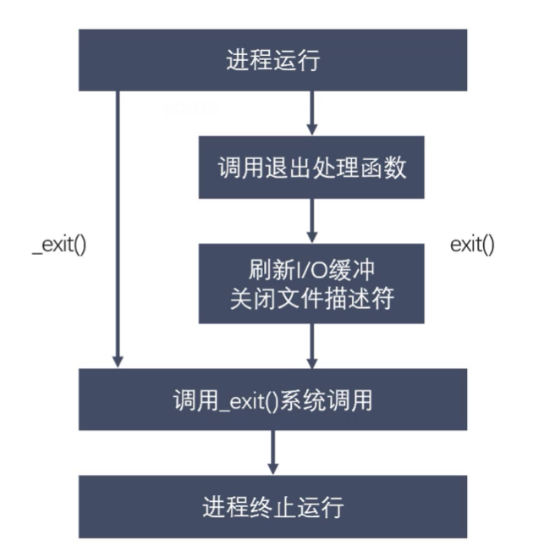
两者的区别在于，exit在退出进程时比\_exit多做了一些事情。\_exit仅仅是退出了进程，而exit会首先调用退出处理函数，接着刷新I/O缓冲并关闭各种文件描述符，最后调用\_exit实现退出。两者的区别如下图所示：

图4-3 两种exit函数的区别

为了测试exit与\_exit功能的不同，编写了如下exit.c文件。

|  |  |  |
| --- | --- | --- |
| *fork.c* |  |  |
| 1. #include <stdio.h> 2. #include <unistd.h> 3. #include <stdlib.h> 5. int main() 6. { 7. pid\_t pid = fork(); 8. if(pid > 0) { 9. printf("[parent][pid %d][ppid %d]**\n**", getpid(), getppid()); 10. printf("parent's hello world!"); 11. sleep(1); 12. \_exit(0); 13. } else if(pid == 0) { 14. printf("[child] [pid %d][ppid %d]**\n**", getpid(), getppid()); 15. printf("child's hello world!"); 16. sleep(1); 17. exit(0); 18. } 20. return 0; 21. } | | |
|  |
|  |
|  |
|  |
|  |

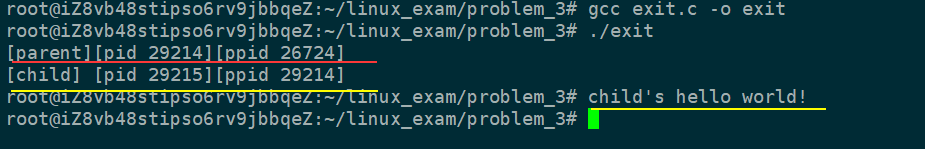
首先使用fork函数创建子进程，父子进程用于分别测试\_exit与exit函数。在父进程执行的函数体中，首先打印父进程的进程号等相关信息(注意，打印完后换行)，再打印Hello world语句（注意，打印完后不换行）；而在子进程执行的函数体中，首先打印子进程的进程号等相关信息（打印完后换行），再打印Hello world语句（打印完后不换行）。最终的结果如下：

图4-4 运行结果

同样，红色标识的信息是父进程打印的，而黄色标识的信息是子进程打印的。由此可以看出exit与\_exit底层机制的不同：打印完后换行（输出\n）会刷新I/O缓冲区，从而将输出缓冲区的所有内容打印到输出设备上，若没有刷新缓冲区，则输出设备上不会有任何打印的内容。父进程所打印的Hello world并没有刷新输出缓冲区，而调用\_exit函数仅仅是结束进程，因此父进程的Hello world语句不会显示；而子进程虽然没有刷新输出缓冲区，但其调用的exit函数会在退出前刷新缓冲区，因此我们可以看到子进程的Hello world语句。

* + - 1. wait与waitpid函数

wait函数用于父进程回收子进程占用的资源。其函数原型如下：

|  |
| --- |
| #include <sys/types.h>  #include <sys/wait.h>    pid\_t wait(int \*wstatus); |
|  |

使用wait函数回收资源时，父进程会被挂起（阻塞），直到它的一个子进程退出或者收到一个不能被忽略的信号时才被唤醒。手册对此的描述很清楚。

*If a child has already changed state, then these calls return immediately. Otherwise, they block until either a child changes state or a signal handler interrupts the call.*

如果子进程都已经结束，或者根本没有任何一个子进程，函数会立刻返回-1。函数的参数int\* wstatus会在调用后，保存被回收的子进程的状态信息。当函数调用成功时，会返回被回收的子进程的进程号，调用失败返回-1。

而waitpid函数则实现了非阻塞式的回收指定进程号的进程的资源。其函数原型如下：

|  |
| --- |
| #include <sys/types.h>  #include <sys/wait.h>    pid\_t waitpid(pid\_t pid, int \*wstatus, int options); |
|  |

waitpid函数可以通过参数1和参数3的不同组合产生不同的效果。

*The waitpid() system call suspends execution of the calling process until a child specified by pid argument has changed state. By default, waitpid() waits only for terminated children, but this behavior is modifiable via the options argument.*

从手册中可以简单的总结出如下用法：

参数pid可以指定要回收的子进程的进程号id：

* pid = -1: 表示要回收任意子进程
* pid = 0: 回收当前进程同一进程组中的任意一个进程
* pid > 0: 回收进程号为pid的子进程

参数options可以选择在回收时是否阻塞当前进程:

* 0: 阻塞（等价于wait）
* WNOHANG：不会阻塞，立即返回。

编写如下两个测试用例wait.c与waitpid.c来对函数效果测试。

|  |  |  |
| --- | --- | --- |
| *waitpid.c* |  |  |
| 1. #include <sys/types.h> 2. #include <sys/wait.h> 3. #include <unistd.h> 4. #include <stdio.h> 5. #include <stdlib.h> 6. int main() 7. { 8. pid\_t pid; 9. for(int i = 0; i < 5; i++){ 10. pid = fork(); 11. if(pid == 0){ 12. **break**; 13. } 14. sleep(1); 15. } 16. if(pid > 0){ 17. while(1){ 18. printf("父进程，pid:%d**\n**", getpid()); 19. int ret = waitpid(-1, NULL, WNOHANG); 20. if(ret > 0) 21. printf("回收了进程%d**\n**", ret); 22. sleep(1); 23. } 24. } 25. else{ 26. printf("子进程，pid:%d, ppid:%d**\n**", getpid(), getppid()); 27. sleep(getpid() % 10 + 3); 28. exit(0); 29. } 30. return 0; 31. } | | |
|  |
|  |
|  |
|  |
|  |

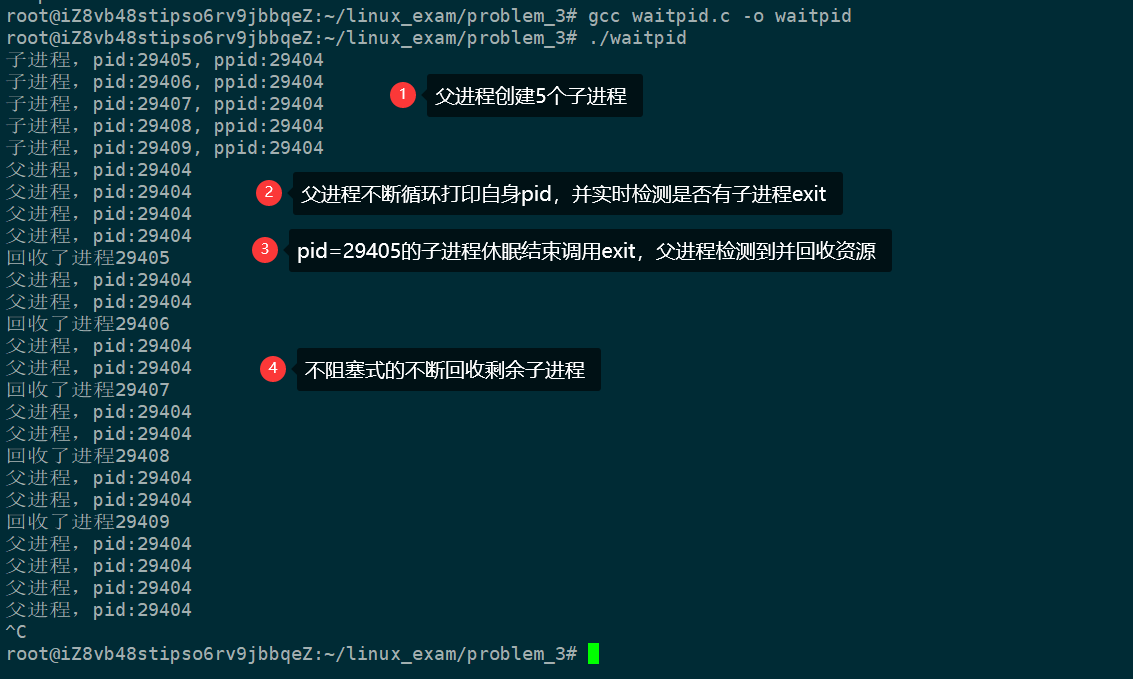
行8~15，当前进程以1秒为间隔共计创建5个子进程。随后，当前进程进入死循环，不断打印当前进程的pid，并实时调用waitpid不阻塞的回收任意子进程，成功回收则打印被回收的进程pid；而子进程则打印自身的pid以及ppid，随后休眠若干时间后，调用exit函数结束。程序执行结果如下：

图4-5 运行结果

可以看出，父进程首先创建了5个不同的子进程。后进入死循环不断打印自身的pid信息。同时实时检测是否有子进程调用exit结束。当某一子进程sleep结束后，调用exit的动作会被父进程察觉，从而完成对子进程资源的回收。

下面是wait.c的测试代码。

|  |  |  |
| --- | --- | --- |
| *wait.c* |  |  |
| 1. #include <sys/types.h> 2. #include <sys/wait.h> 3. #include <unistd.h> 4. #include <stdio.h> 5. #include <stdlib.h> 6. int main() 7. { 8. pid\_t pid; 9. for(int i = 0; i < 5; i++){ 10. pid = fork(); 11. if(pid == 0){ 12. **break**; 13. } 14. sleep(1); 15. } 16. if(pid > 0){ 17. while(1){ 18. printf("父进程，pid:%d**\n**", getpid()); 19. int ret = wait(NULL); 20. if(ret > 0) 21. printf("回收了进程%d**\n**", ret); 22. sleep(1); 23. } 24. } 25. else{ 26. printf("子进程，pid:%d, ppid:%d**\n**", getpid(), getppid()); 27. sleep(getpid() % 10 + 3); 28. exit(0); 29. } 30. return 0; 31. } | | |
|  |
|  |
|  |
|  |
|  |

代码逻辑和watipid.c类似，区别在于使用wait函数阻塞式回收进程资源。执行结果如下：

图4-6 运行结果

父进程打印自身的pid信息后，由于调用了wait函数，自身被阻塞，因此不能继续

打印pid信息。直到某一子进程休眠结束调用exit后，父进程回收资源，才得以打印下一条pid信息，随后再次被阻塞，等待新的子进程休眠结束。最终，当全部进程资源回收完成后，wait函数立即返回，不再阻塞父进程。因此父进程得以循环打印自身pid信息。

* 1. 题目五（附加题）
     1. 题目要求

附加题，编写一个定时监控某日志文件、监控内存使用、监控CPU使用或监控硬盘空间使用的shell运维程序，要求设定处理门槛，实现及时止损和保护，同时立刻报警。给出全部程序代码（20分）

* + 1. 题目作答

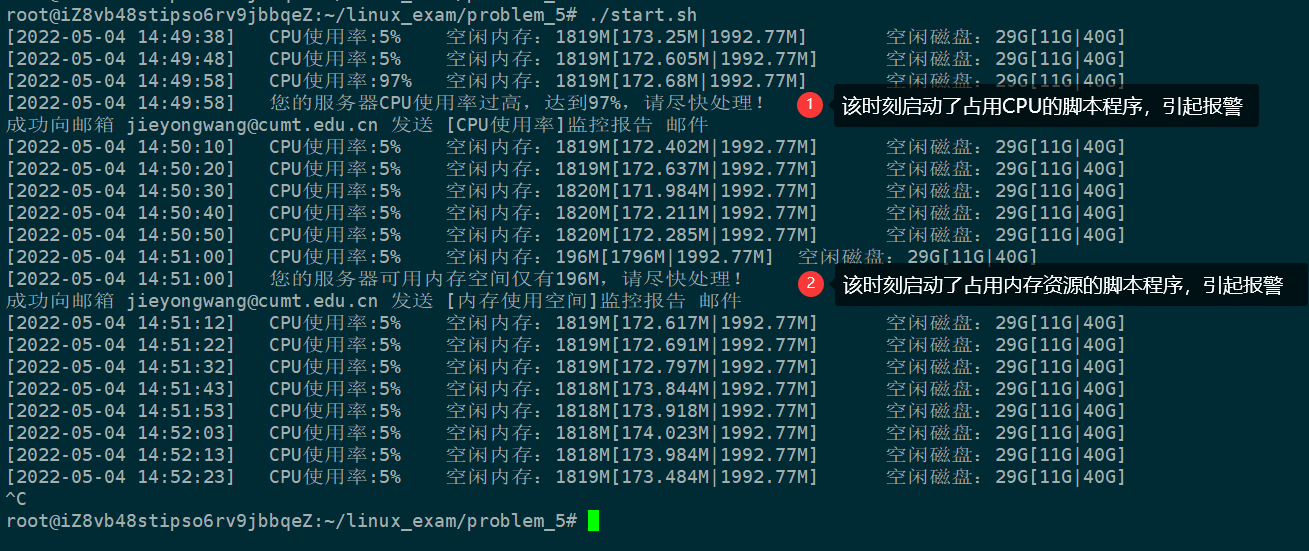
本项目实现了一个定时监控系统内存使用情况、磁盘空间使用情况、CPU使用率的shell运维程序。编码风格规范，全部超参数全部位于脚本开头设置，可以设置的超参数如下：

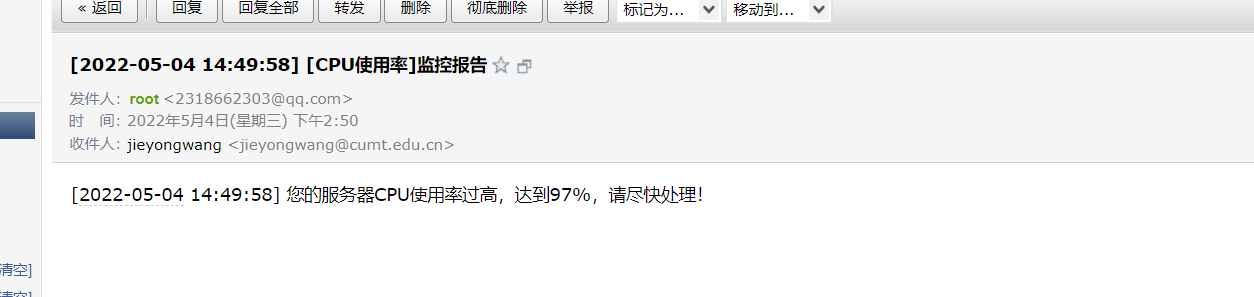
表5-1 超参数及其作用

|  |  |
| --- | --- |
| **参数** | **作用** |
| CPU\_WARNING\_THRESHOLD | CPU使用率报警阈值。当CPU使用率大于该超参数时，发送邮件报警，并记录在日志中 |
| MEM\_WARNING\_THRESHOLD | 可用内存大小报警阈值。当可用内存空间小于该超参数时，发送邮件报警，并记录在日志中 |
| DISK\_WARNING\_THRESHOLD | 磁盘剩余空间报警阈值。当磁盘空用空间小于该超参数时，发送邮件报警，并记录在日志中 |
| EMAIL\_ADDR | 接收报警邮件的邮箱地址 |
| TIME | 脚本检测间隔。每隔TIME秒，检测一次系统 |
| DEBUG | DEBUG模式，设置为1则不发送报警邮件，设置为0则发送报警邮件 |

通过vmstat、free、df命令，获取了CPU、内存、磁盘的使用情况；通过sed与awk命令，从系统资源使用情况中提取出有效信息；通过ssmtp、sendmail的配置，允许系统使用邮箱发送邮件。

部分运行结果如下图：

图5-1 运行结果

图5-2 成功接收报警邮件

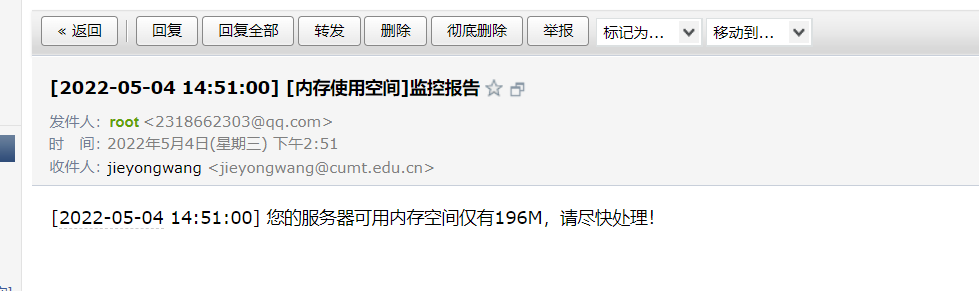
图5-3 报警邮件内容（1）

图5-4 报警邮件内容（2）

脚本的源代码如下：

|  |  |  |
| --- | --- | --- |
| *start.sh* |  |  |
| 1. *#!/bin/bash* 3. *# CPU使用率超过该值，报警* 4. CPU\_WARNING\_THRESHOLD=80 5. *# 空闲内存小于该值，报警* 6. MEM\_WARNING\_THRESHOLD=500 7. *# 空闲磁盘小于该值，报警* 8. DISK\_WARNING\_THRESHOLD=10 9. *# 接受报警的邮箱* 10. EMAIL\_ADDR="jieyongwang@cumt.edu.cn" 11. *# 检测的间隔时间* 12. TIME=10 13. *# DEBUG模式，为1则不发送邮件，为0则发送邮件* 14. DEBUG=0 16. **while** **true** 17. **do** 18. *# 100%减去CPU闲置率，得到使用率%* 19. CPU\_MSG=$**(vmstat|** **sed** -n '3,$p'**)** *# | awk '{x = x + $15} END {print x/5}' | awk -F. '{print $1}')* 20. CPU\_MSG=$**(echo** $CPU\_MSG**|** **awk** '{print $15}'**|** **awk** -F. '{print $1}'**)** 21. CPU\_MSG=$**(echo** "100-$CPU\_MSG" **|** **bc)** 23. *# 全部内存减去已使用的内存，单位MB* 24. MEM\_TOTAL=$**(free** **|** **grep** Mem**|** **awk** '{x = $2 / 1024} END {print x}'**)** 25. MEM\_USED=$**(free** **|** **grep** Mem**|** **awk** '{x = $3 / 1024} END {print x}'**)** 26. MEM\_FREE=$**(free** **|** **grep** Mem**|** **awk** '{x = ($2 - $3) / 1024} END {print x}'**|** **awk** -F. '{print $1}'**)** 28. DISK\_TOTAL=$**(df** -h**|** **grep** -e "^/"**|** **awk** '{x = $2} END {print x}'**)** 29. DISK\_USED=$**(df** -h**|** **grep** -e "^/"**|** **awk** '{x = $3} END {print x}'**)** 30. DISK\_FREE=$**(df** -h**|** **grep** -e "^/"**|** **awk** '{x = $2 - $3} END {print x}'**|** **awk** -F. '{print $1}'**)** 32. **echo** -e "$(date +[%Y-%m-%d\ %H:%M:%S])**\t**CPU使用率:${CPU\_MSG}%**\t**空闲内存：${MEM\_FREE}M[${MEM\_USED}M|${MEM\_TOTAL}M]**\t**空闲磁盘：${DISK\_FREE}G[${DISK\_USED}|${DISK\_TOTAL}]" 34. **if** **[** $CPU\_MSG -ge $CPU\_WARNING\_THRESHOLD **]**; **then** 35. MSG="$(date +[%Y-%m-%d\ %H:%M:%S])**\t**您的服务器CPU使用率过高，达到${CPU\_MSG}%，请尽快处理！" 36. **echo** -e $MSG 37. **if** **[** $DEBUG -eq 0 **]**; **then** 38. **echo** -e $MSG**|** mail -s "$(date +[%Y-%m-%d\ %H:%M:%S]) [CPU使用率]监控报告" $EMAIL\_ADDR 39. **echo** -e "成功向邮箱 $EMAIL\_ADDR 发送 [CPU使用率]监控报告 邮件" 40. **fi** 41. **fi** 42. **if** **[** $MEM\_FREE -le $MEM\_WARNING\_THRESHOLD **]**; **then** 43. MSG="$(date +[%Y-%m-%d\ %H:%M:%S])**\t**您的服务器可用内存空间仅有${MEM\_FREE}M，请尽快处理！" 44. **echo** -e $MSG 45. **if** **[** $DEBUG -eq 0 **]**; **then** 46. **echo** -e $MSG**|** mail -s "$(date +[%Y-%m-%d\ %H:%M:%S]) [内存使用空间]监控报告" $EMAIL\_ADDR 47. **echo** -e "成功向邮箱 $EMAIL\_ADDR 发送 [内存使用空间]监控报告 邮件" 48. **fi** 49. **fi** 50. **if** **[** $DISK\_FREE -le $DISK\_WARNING\_THRESHOLD **]**; **then** 51. MSG="$(date +[%Y-%m-%d\ %H:%M:%S])**\t**您的服务器磁盘剩余空间仅有${DISK\_FREE}G，请尽快处理！" 52. **echo** -e $MSG 53. **if** **[** $DEBUG -eq 0 **]**; **then** 54. **echo** -e $MSG**|** mail -s "$(date +[%Y-%m-%d\ %H:%M:%S]) [磁盘使用空间]监控报告" $EMAIL\_ADDR 55. **echo** -e "成功向邮箱 $EMAIL\_ADDR 发送 [磁盘使用空间]监控报告 邮件" 56. **fi** 57. **fi** 59. **sleep** $TIME 60. **done** | | |
|  |
|  |
|  |
|  |
|  |