

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
1.1 题目要求.....	1
1.2 题目作答.....	1
2 题目二.....	9
2.1 题目要求.....	9
2.2 题目作答.....	9
3 题目三.....	22
3.1 题目要求.....	22
3.2 题目作答.....	22
4 题目四.....	38
4.1 题目要求.....	38
4.2 题目作答.....	38
5 题目五（附加题）.....	48
5.1 题目要求.....	48
5.2 题目作答.....	48

1 题目一

1.1 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。

1.2 题目作答

1.2.1 子问题(1)

查看当前目录:

`pwd`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 1-1 `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`pwd`，结果如下。

```
root@iZ8vb48stipso6rv9jbbqeZ:~# pwd
/root
root@iZ8vb48stipso6rv9jbbqeZ:~# ls
JavaWebBegin
```

图 1-1 运行结果

创建新目录，并进入目录

`mkdir`命令用于创建新目录。`cd`命令可以进入目标目录。

命令 1-2 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 1-3 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`mkdir linux_final_exam`，并使用`cd`命令进入所创建的新目录中。结果如下：

```
root@iZ8vb48stipso6rv9jbbqeZ:~# mkdir linux_final_exam
root@iZ8vb48stipso6rv9jbbqeZ:~# cd linux_final_exam/
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

图 1-2 运行结果

创建空文件，查看文件详细信息

使用`touch`命令创建空文件，使用`ls`命令查看文件的详细信息。

命令 1-4 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 1-5 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。

-l: 列出文件详细信息。

我们使用`touch exp_problem_1`命令创建新文件`exp_problem_1`，使用`ls -l`命令查看新文件的详细信息。结果如图：

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# touch exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# ls -l
total 0
-rw-r--r-- 1 root root 0 Apr 26 19:25 exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

图 1-3 运行结果

在命令`ls -l`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 `chown` 命令修改文件所有者，通过 `chmod` 命令修改文件或目录的权限。

命令 1-6 `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 1-7 `chmod`

`chmod [OPTION]... MODE[,MODE]... FILE...`

`chmod [OPTION]... OCTAL-MODE FILE...`

由于新创建的文件 `exp_problem_1` 其所有者默认为 `root`，因此这里我通过命令 `chown lionerd exp_problem_1`，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 `chmod -R 744 exp_problem_1` 付给所有者读写执行完全权限。结果如图：

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# chown lionerd exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# ls -l
total 0
-rw-r--r-- 1 lionerd root 0 Apr 26 19:25 exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# chown root exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# ls -l
total 0
-rw-r--r-- 1 root root 0 Apr 26 19:25 exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# chmod -R 744 exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# ls -l
total 0
-rwxr--r-- 1 root root 0 Apr 26 19:25 exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#

```

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`ls -l`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：

```
root@iZ8vb48stipso6rv9jbbqeZ:~# touch -d "12:00am 08/01/2021" exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~# touch --time=access -d "12:00am 08/01/2021" exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~# stat exp_problem_1
  File: exp_problem_1
  Size: 56          Blocks: 8          IO Block: 4096   regular file
Device: fc01h/64513d Inode: 397891       Links: 1
Access: (0744/-rwxr--r--)  Uid: (  0/   root)   Gid: (  0/   root)
Access: 2021-08-01 00:00:00.000000000 +0800
Modify: 2021-08-01 00:00:00.000000000 +0800
Change: 2022-05-04 15:31:52.480077391 +0800
 Birth: -
root@iZ8vb48stipso6rv9jbbqeZ:~#
```

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。

1.2.2 子问题(2)

向文件中写入 “hello, world!”

使用`echo`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 1-8 `echo`

`echo(选项)(参数)`

`-e`: 激活转义字符。

我们使用`echo hello, world! > exp_problem_1`命令，将 “hello, world!” 写入文件。

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# echo "hello, world!" > exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cat exp_problem_1
hello, world!
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

图 1-6 运行结果

向文件中追加三组姓名-学号对

使用`read`命令接受键盘输入，赋值到相应变量中。

命令 1-9 `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`read name1 sid1 name2 sid2 name3 sid3`命令，接受键盘输入赋值到指定变量中；使用 `echo -e $name1-$sid1"\n"$name2-$sid2"\n"$name3-$sid3 >> exp_problem_1`命令，将变量值输出重定向到文件尾部。运行结果如下：

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# read name1 sid1 name2 sid2 name3 sid3
wjy1 03190886 wjy2 03190887 wjy3 03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# echo -e $name1-$sid1"\n"$name2-$sid2"\n"$name3-$sid3 >> exp_problem_1
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cat exp_problem_1
hello, world!
wjy1-03190886
wjy2-03190887
wjy3-03190888

```

图 1-7 运行结果

计算文件单词数

使用`wc`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 1-10 `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`wc -w exp_problem_1`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`wc -w < exp_problem_1`，获得文件中的单词数。结果如下：

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cat exp_problem_1
hello, world!
wjy1-03190886
wjy2-03190887
wjy3-03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# wc -w < exp_problem_1
5
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#

```

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`sort`可以对文件内容排序；使用命令`tail`可以查看文件末尾若干行。

命令 1-11 `sort`

`sort` [OPTION]... [FILE]...

- r, --reverse 将结果倒序排列。
-

命令 1-12 `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`exp_problem_1`内容恰好是按照 ASCII 码升序排序的，因此为了展示`sort`命令，我们通过`-r`参数对文件降序排序，即`sort -r exp_problem_1`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`tail`命令的输入，结果如下：

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# sort -r exp_problem_1
wjy3-03190888
wjy2-03190887
wjy1-03190886
hello, world!
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# sort -r exp_problem_1 | tail -1
hello, world!
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

图 1-9 运行结果

提取文件中的学号

`grep`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 1-13 `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 --invert-match # 反转查找。

`grep`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`grep`命令。

`grep -E -o "[0-9]{8}" exp_problem_1`或`cat exp_problem_1 | grep -E -o "[0-9]{8}"`命令都可以从文件中提取学号。其中，正则模式`"[0-9]{8}"`表示 8 位数字。匹配结果如下：

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# grep -E -o "[0-9]{8}" exp_problem_1
03190886
03190887
03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cat exp_problem_1 | grep -E -o "[0-9]{8}"
03190886
03190887
03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

图 1-10 运行结果

最后，通过输出重定向，即执行命令 `grep -E -o "[0-9]{8}" exp_problem_1 > student_id`，我们将匹配内容保存到文件 `student_id` 中。如下图：

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# grep -E -o "[0-9]{8}" exp_problem_1 > student_id
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# ls
exp_problem_1  student_id
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cat student_id
03190886
03190887
03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

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

`comm` 命令为按行比较两个已排序的文件。

命令 1-14 `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。
-

`comm` 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 `comm student_id exp_problem_1` 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：

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# comm student_id exp_problem_1
03190886
03190887
03190888
      hello, world!
      wjy1-03190886
      wjy2-03190887
      wjy3-03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

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`comm`命令的效果，我们使用`vim`编辑器，在`student_id`的尾部增加一行"`hello,world!`"，再次使用`comm`命令，结果如下：

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# vim student_id
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cat exp_problem_1
hello, world!
wjy1-03190886
wjy2-03190887
wjy3-03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cat student_id
03190886
03190887
03190888
hello, world!
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# comm student_id exp_problem_1
03190886
03190887
03190888
      hello, world!
      wjy1-03190886
      wjy2-03190887
      wjy3-03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

图 1-13 运行结果

可以发现，`comm`命令输出的第三列中有了内容，内容`hello,world!`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`comparison_results`中。即执行命令`comm student_id exp_problem_1 > comparison_results`。结果如下：

```
wjy3-03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# comm student_id exp_problem_1 > comparison_results
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# ls
comparison_results  exp_problem_1  student_id
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cat comparison_results
03190886
03190887
03190888
      hello, world!
      wjy1-03190886
      wjy2-03190887
      wjy3-03190888
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam#
```

图 1-14 运行结果

拷问文件至父目录，删除该目录

命令`cp`可以实现文件拷贝；命令`rm`可以实现文件/文件夹删除。

命令 1-15 `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 1-16 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 `cp -r ./ ../`, 将当前目录下的全部文件复制到父目录下。执行结果如图:

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cp -r ./ ../
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_final_exam# cd ..
root@iZ8vb48stipso6rv9jbbqeZ:~# ls
comparison_results  exp_problem_1  JavaWebBegin  linux_final_exam  student_id
root@iZ8vb48stipso6rv9jbbqeZ:~#
```

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 `rm -r linux_final_exam`, 删除题目一开始新建的文件夹。执行结果如图:

```
root@iZ8vb48stipso6rv9jbbqeZ:~# rm -r linux_final_exam
root@iZ8vb48stipso6rv9jbbqeZ:~# ls
comparison_results  exp_problem_1  JavaWebBegin  student_id
root@iZ8vb48stipso6rv9jbbqeZ:~#
```

图 1-16 运行结果

2 题目二

2.1 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 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)

2.2 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 `main.sh`，下面给出各功能的测试截图。

2.2.1 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 `./main.sh wjy`，可以看出，新用户 `wjy` 创建成功。

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_2# ls /home
lionerd
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_2# ./main.sh wjy

用户 [wjy] 创建成功，相关信息如下：
uid=1001(wjy) gid=1001(wjy) groups=1001(wjy)
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_2# ls /home
lionerd wjy
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_2#
  
```

1 执行脚本前，仅有一个用户lionerd

2 执行脚本后，通过家目录可以确定，用户wjy创建成功

图 2-1 运行结果

我们再次执行相同的命令 `./main.sh wjy`，此时，由于用户 `wjy` 已经存在，因此会给

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_2# ./main.sh wjy

用户 [wjy] 已经存在
  
```

1 用户已经存在，给出警告信息

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 `su` 命令切换至用户 `wjy`，再次执行脚本，结果如下：

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_2# su wjy
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ ./main.sh qq
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.

用户 [qq] 创建失败，可能是权限不足
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$
  
```

1 切换普通用户wjy

2 普通用户wjy试图创建新用户qqq

3 脚本给出提示，权限不足

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 `wjy` 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。

```
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ ./main.sh

[wjy] login the system

您的默认密码为123456，过于简单，请修改密码！
Changing password for wjy.
(current) UNIX password: 
```

1 登陆系统

2 密码过于简单，强制修改密码

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。

```
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ ./main.sh

[wjy] login the system

您的默认密码为123456，过于简单，请修改密码！
Changing password for wjy.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
Sorry, passwords do not match
passwd: Authentication token manipulation error
passwd: password unchanged

密码修改失败，请重新登录系统！
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ 
```

1 登陆系统

2 为默认密码，强制修改

3 两次密码不一致，脚本结束

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。

```
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ ./main.sh

[wjy] login the system
用户 [wjy] 系统:
(0) 清空屏幕，显示目录
(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

[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 
```

1 修改密码后，再次登录系统

2 不再提示密码过于简单，而是直接登入系统

图 2-6 运行结果

2.2.2 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。

```
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ ./main.sh

[wjy] login the system
用户 [wjy] 系统:
(0) 清空屏幕，显示目录
(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

[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 1
[wjy] [/root/linux_exam/problem_2] 请输入目录: 
```

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wjy以及用户当前所在的目录/root/linux_exam/problem_2。

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。

```
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ ./main.sh

[wjy] login the system
用户 [wjy] 系统:
(0) 清空屏幕，显示目录
(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

[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 1
[wjy] [/root/linux_exam/problem_2] 请输入目录: ./
conf main.sh null
[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 
```

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。

```
用户 [wjy] 系统:
(0) 清空屏幕，显示目录
(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

[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 1
[wjy] [/root/linux_exam/problem_2] 请输入目录: ...../
conf main.sh null
[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 1
[wjy] [/root/linux_exam/problem_2] 请输入目录: ...../
[wjy] [/root/linux_exam/problem_2] 目录 ...../ 不存在!
[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 
```

图 2-9 运行结果

2.2.3 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。

```
用户 [wjy] 系统：
(0) 清空屏幕，显示目录
(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

[wjy] [/root/linux_exam/problem_2] 请选择操作类型：1
[wjy] [/root/linux_exam/problem_2] 请输入目录： ./
conf main.sh null
[wjy] [/root/linux_exam/problem_2] 请选择操作类型：2
[wjy] [/root/linux_exam/problem_2] 请输入要切换的路径： /home/wjy
[wjy] [/home/wjy] 请选择操作类型：1
[wjy] [/home/wjy] 请输入目录： ./
[wjy] [/home/wjy] 请选择操作类型：
```

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。

```
[wjy] [/home/wjy] 请选择操作类型：2
[wjy] [/home/wjy] 请输入要切换的路径： .../
[wjy] [/home/wjy] 目录 .../ 不存在！
[wjy] [/home/wjy] 请选择操作类型：
```

图 2-11 运行结果

2.2.4 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`test.log`文件，之后使用操作 1 查看家目录下的文件信息，证明了`test.log`文件创建成功。

```
用户 [wjy] 系统：
(0) 清空屏幕，显示目录
(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

[wjy] [/home/wjy] 请选择操作类型： 3
[wjy] [/home/wjy] 请输入要创建的文件名： test.log
[wjy] [/home/wjy] 文件 test.log 创建成功！
[wjy] [/home/wjy] 请选择操作类型： 1
[wjy] [/home/wjy] 请输入目录： ./
test.log
[wjy] [/home/wjy] 请选择操作类型：
```

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。

```
用户 [wjy] 系统：
(0) 清空屏幕，显示目录
(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

[wjy] [/home/wjy] 请选择操作类型： 1
[wjy] [/home/wjy] 请输入目录： ./
test.log
[wjy] [/home/wjy] 请选择操作类型： 3
[wjy] [/home/wjy] 请输入要创建的文件名： test.log
[wjy] [/home/wjy] 文件 test.log 已经存在，无法创建！
[wjy] [/home/wjy] 请选择操作类型：
```

图 2-13 运行结果

2.2.5 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：

```
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ cat /home/wjy/hello.sh
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ ./main.sh

[wjy] login the system
用户 [wjy] 系统:
(0) 清空屏幕，显示目录
(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

[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 4

[wjy] [/root/linux_exam/problem_2] 请输入要编辑的文件路径: /home/wjy/hello.sh

[wjy] [/root/linux_exam/problem_2] 请选择操作类型: ^C
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ cat /home/wjy/hello.sh
echo "Hello world!"
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ chmod u+x /home/wjy/hello.sh
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$ /home/wjy/hello.sh
Hello world!
wjy@iZ8vb48stipso6rv9jbbqeZ:/root/linux_exam/problem_2$
```

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。

```
用户 [wjy] 系统:
(0) 清空屏幕，显示目录
(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

[wjy] [/root/linux_exam/problem_2] 请选择操作类型: 4

[wjy] [/root/linux_exam/problem_2] 请输入要编辑的文件路径: hhhhhh

[wjy] [/root/linux_exam/problem_2] 文件 hhhhhh 不存在，无法编辑

[wjy] [/root/linux_exam/problem_2] 请选择操作类型:
```

图 2-15 运行结果

2.2.6 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。

```
用户 [wjy] 系统：
(0) 清空屏幕，显示目录
(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

[wjy] [/home/wjy] 请选择操作类型：1

[wjy] [/home/wjy] 请输入目录： ./ ① 当前目录下，存在hello.sh文件

hello.sh  test.log

[wjy] [/home/wjy] 请选择操作类型：5

[wjy] [/home/wjy] 请输入要删除的文件路径： hello.sh ② 使用操作5删除

[wjy] [/home/wjy] 文件 hello.sh 删除成功！

[wjy] [/home/wjy] 请选择操作类型：1

[wjy] [/home/wjy] 请输入目录： ./ ③ 再次查看，确定文件已经被删除

test.log

[wjy] [/home/wjy] 请选择操作类型： █
```

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。

```
用户 [wjy] 系统：
(0) 清空屏幕，显示目录
(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

[wjy] [/home/wjy] 请选择操作类型：5

[wjy] [/home/wjy] 请输入要删除的文件路径： aaaaaaaaaa ① 文件不存在，给出警告

[wjy] [/home/wjy] 文件 aaaaaaaaaa 不存在，无法删除

[wjy] [/home/wjy] 请选择操作类型： █
```

图 2-17 运行结果

2.2.7 代码

main.sh

```

1.  #!/bin/bash
2.
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. #####
21.
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. }
45.
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. }
69.
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 [$1] 系统: "
77.     echo " (0) 清空屏幕，显示目录"
78.     echo " (1) 显示用户选择的目录内容"
79.     echo " (2) 按照用户输入的目录切换路径"
80.     echo " (3) 按照用户输入的文件名在/home 目录中创建文件"
81.     echo " (4) 编辑用户输入的文件"
82.     echo " (5) 删除用户选择的文件"
83. }
84.
85. function operation_1() {
86.     CURRENT_PATH=$(pwd)
```

```
87.     echo
88.     read -p "[${1}] [${CURRENT_PATH}] 请输入目录: " DIR_PATH
89.
90.     if [ -d $DIR_PATH ]; then
91.         echo
92.         ls $DIR_PATH
93.     else
94.         echo
95.         echo "[${1}] [${CURRENT_PATH}] 目录 $DIR_PATH 不存在! "
96.     fi
97. }
98.
99. function operation_2(){
100.     CURRENT_PATH=$(pwd)
101.     echo
102.     read -p "[${1}] [${CURRENT_PATH}] 请输入要切换的路径: " DIR_PATH
103.
104.     if [ -d $DIR_PATH ]; then
105.         cd $DIR_PATH
106.     else
107.         echo
108.         echo "[${1}] [${CURRENT_PATH}] 目录 $DIR_PATH 不存在! "
109.     fi
110. }
111.
112. function operation_3(){
113.     # 注意, 这里还没有对文件名合法性做校验
114.     # 已知 bug: 文件名输入///, 显示创建成功, 但其实没有成功
115.     # !!!
116.     # !!!
117.     CURRENT_PATH=$(pwd)
118.     echo
119.     read -p "[${1}] [${CURRENT_PATH}] 请输入要创建的文件名: " FILE_NAME
120.
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. }
130.
```

```
131. function operation_4() {
132.     CURRENT_PATH=$(pwd)
133.     echo
134.     read -p "[${1}] [${CURRENT_PATH}] 请输入要编辑的文件路径: " FILE_PATH
135.
136.     if [ -f "${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 ${FILE_PATH} 不存在, 无法编辑"
143.     fi
144. }
145.
146. function operation_5() {
147.     CURRENT_PATH=$(pwd)
148.     echo
149.     read -p "[${1}] [${CURRENT_PATH}] 请输入要删除的文件路径: " FILE_PATH
150.
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. }
160.
161.
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"
170.
171.     change_password ${USER_NAME}
172.
173.     clear
174.     show_menu ${USER_NAME}
```

```
175.
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. }
207.
208.
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
```

3 题目三

3.1 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）

3.2 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：

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_4# tree
.
├── out
│   └── com
│       └── chen
│           ├── tcp
│           │   ├── calculator
│           │   │   └── CalculatorLib.class
│           │   ├── Client$1.class
│           │   ├── Client$2.class
│           │   ├── Client.class
│           │   ├── MessagePassing.class
│           │   ├── Server$1.class
│           │   ├── Server$2.class
│           │   └── Server.class
│           └── udp
│               ├── Client$1.class
│               ├── Client$2.class
│               ├── Client.class
│               ├── Server$1.class
│               └── Server.class
├── sh_compile.sh
├── sh_TCP_client_start.sh
├── sh_TCP_server_start.sh
├── sh_UDP_client_start.sh
├── sh_UDP_server_start.sh
└── src
    ├── com
    │   └── chen
    │       ├── tcp
    │       │   ├── calculator
    │       │   │   ├── CalculatorLib.java
    │       │   │   ├── libcal.dll
    │       │   │   ├── libcal.so
    │       │   │   └── Test.java
    │       │   ├── Client.java
    │       │   ├── MessagePassing.java
    │       │   └── Server.java
    │       └── udp
    │           ├── Client.java
    │           └── Server.java
    ├── Test.java
    └── TestServer.java

12 directories, 29 files
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_4#
```

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！）

3.2.1 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。

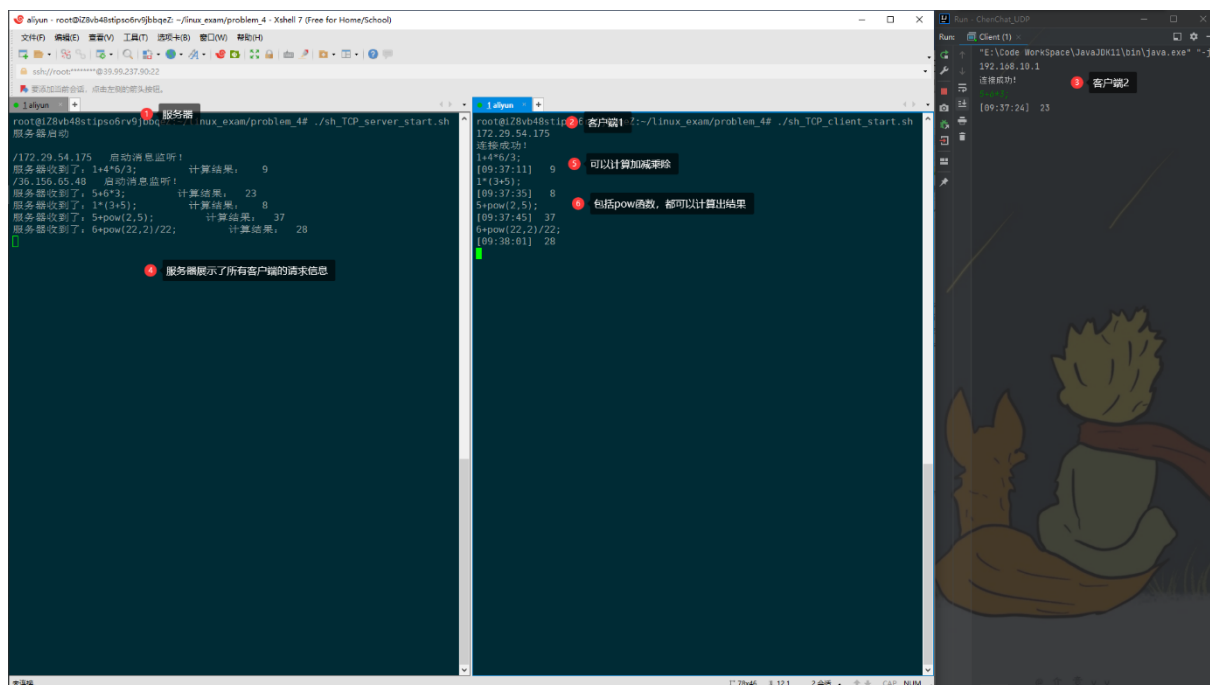


图 3-2 运行结果

全部代码包括启动的 shell 脚本如下：

Clinet.java

```
1. package com.chen.tcp;
2.
3. import java.io.IOException;
4. import java.net.InetAddress;
5. import java.net.Socket;
6. import java.util.Scanner;
7.
8. public class Client {
9.
10.     /**
11.      * 客户端套接字
12.      */
13.     private Socket socket;
14.
15.     /**
16.      * 与客户端绑定的 MP
17.      */
18.     MessagePassing mp;
19.
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(),
25.             socket.getOutputStream(), socket);
26.
27.         System.out.println("连接成功!");
28.     }
29.
30.     /**
31.      * 为当前客户端创建一个发送消息线程
32.      * @return 发送消息线程
33.      */
34.     private Thread threadSend() {
35.         Thread thread = new Thread(new Runnable() {
36.             @Override
37.             public void run() {
38.                 Scanner sc = new Scanner(System.in);
39.                 while(true) {
40.                     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. }
50.
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. }
72.
73. /**
74.  * 客户端启动
75.  */
76. public void begin() {
77.     threadSend().start();
78.     threadReceive().start();
79. }
80.
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.     }
87.
88. }
```

Server.java

```
1. package com.chen.tcp;
2.
3. import com.chen.tcp.calculator.CalculatorLib;
4.
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;
13.
14. public class Server {
15.
16.     /**
17.      * 服务器套接字
18.      */
19.     private ServerSocket serverSocket;
20.
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;
25.
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.     }
31.
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.     }
39.
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.     }
71.
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. }
113.
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.      }
127.
128.      public static void main(String[] args) throws IOException {
129.          Server server = new Server();
130.          server.begin();
131.      }
132.
133.  }
134.
```

MessagePassing.java

```
1. package com.chen.tcp;
2.
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;
10.
11. public class MessagePassing {
12.
13.     private InputStream is;
14.     private OutputStream os;
15.     private Socket socket;
16.
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.     }
27.
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.     }
42.
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.     }
49.
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.     }
60.
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/"
4.
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"
10.
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"
3.
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"
3.
```

4. `cd $OUT_PATH`
5. `java -classpath .:$LIB_PATH com.chen.tcp.Server`

3.2.2 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：

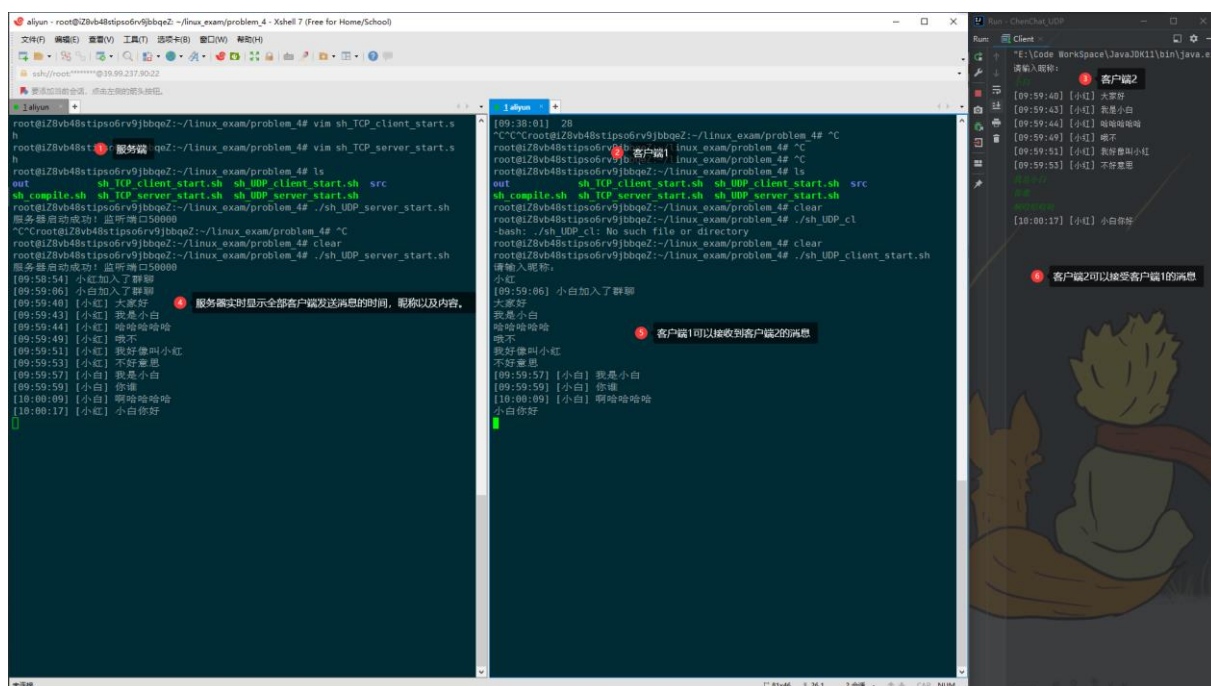


图 3-3 运行结果

项目的全部代码以及 shell 脚本如下：

Client.java

1. `package com.chen.udp;`
- 2.
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;`

```
11.
12. public class Client {
13.
14.     String serverIp = "39.99.237.90";
15.
16.     DatagramSocket clientSocket = new DatagramSocket();
17.
18.     String name;
19.
20.     Client() throws Exception {
21.         System.out.println("请输入昵称: ");
22.         name = new Scanner(System.in).next();
23.     }
24.
25.     public void begin() throws Exception{
26.         sendWelcomeMessage();
27.         threadRecieve().start();
28.         threadSend().start();
29.     }
30.
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.     }
43.
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. }
53.
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. }
64.
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. }
77.
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. }
91.
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.     }
97.
98.     public static void main(String[] args) throws Exception{
99.         Client client = new Client();
100.         client.begin();
101.     }
102.
103. }
```

Server.java

```
1. package com.chen.udp;
2.
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.*;
9.
10. public class Server {
11.
12.     /**
13.      * 服务器套接字
14.      */
15.     DatagramSocket serverSocket = new DatagramSocket(50000);
16.
17.     Vector<InetAddress> ips;
18.     Vector<Integer> ports;
19.
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.     }
25.
26.     public void begin() throws Exception{
27.         threadRecieve().start();
28.     }
29.
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,
37. bys.length);
38.                     String msg = recieveOneMessage(dp);
39.                     System.out.println(msg);
40.                     sendToAllClients(msg, dp);
41.                 }
42.             }
43.         });
44.         return thread;
45.     }
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(),
57. StandardCharsets.UTF_8);
58.     }
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.     }
72.
```

```
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.     }
82.
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.     }
90.
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"
3.
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"
3.
4. cd $OUT_PATH
```

```
5. java -classpath .:$LIB_PATH com.chen.udp.Client
```

4 题目四

4.1 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）

4.2 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。

4.2.1 `fork` 函数

fork 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.

fork 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 *fork* 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.

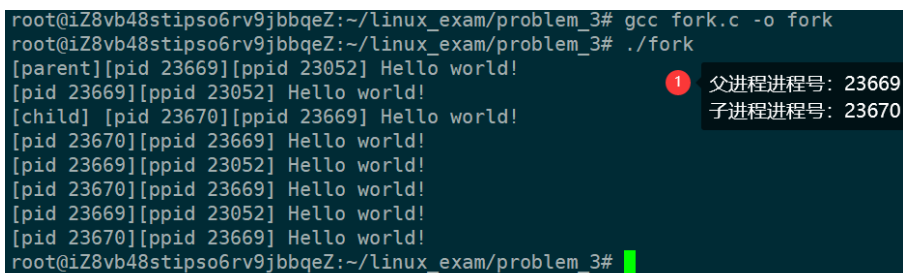
由此编写了如下的 *hello world* 欢迎语程序。

fork.c

```
1. #include <sys/types.h>
```

```
2. #include <unistd.h>
3. #include <stdio.h>
4.
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.    }
13.
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`fork`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。



```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# gcc fork.c -o fork
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# ./fork
[parent][pid 23669][ppid 23052] Hello world!
[pid 23669][ppid 23052] Hello world!
[child] [pid 23670][ppid 23669] Hello world!
[pid 23670][ppid 23669] Hello world!
[pid 23669][ppid 23052] Hello world!
[pid 23670][ppid 23669] Hello world!
[pid 23669][ppid 23052] Hello world!
[pid 23670][ppid 23669] Hello world!
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3#
```

图 4-1 运行结果

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

4.2.2 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。

execel()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
2.
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. }
9.
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. }
16.
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>
4.
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.    }
14.
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：

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# gcc exec.c -o exec
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# ./exec
[parent][pid 28965][ppid 26724] Hello world!
I am in exec.c, not a shell! [pid 28965][ppid 26724] Hello world!
[child] [pid 28966][ppid 28965] Hello world!
I am in exec.sh, not a .c! [pid 28966][ppid 28965] Hello world!
I am in exec.c, not a shell! [pid 28965][ppid 26724] Hello world!
I am in exec.sh, not a .c! [pid 28966][ppid 28965] Hello world!
I am in exec.c, not a shell! [pid 28965][ppid 26724] Hello world!
I am in exec.sh, not a .c! [pid 28966][ppid 28965] Hello world!
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3#

```

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。

4.2.3 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：

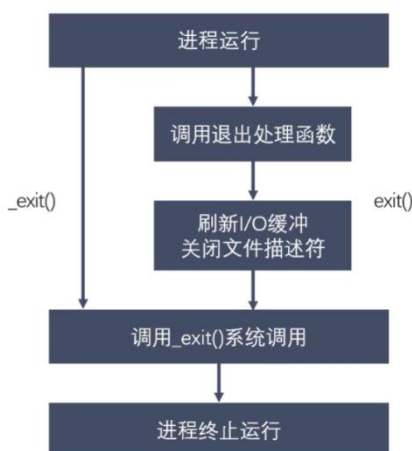


图 4-3 两种 exit 函数的区别

为了测试 exit 与 _exit 功能的不同，编写了如下 exit.c 文件。

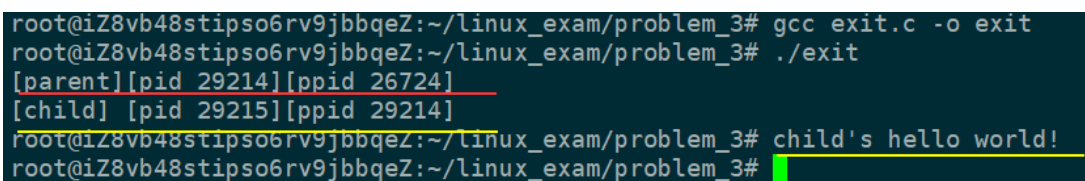
fork.c

```

1. #include <stdio.h>
2. #include <unistd.h>
3. #include <stdlib.h>
4.
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.    }
19.
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：



```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# gcc exit.c -o exit
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# ./exit
[parent][pid 29214][ppid 26724]
[child] [pid 29215][ppid 29214]
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# child's hello world!
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3#

```

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。

4.2.4 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：

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# gcc waitpid.c -o waitpid
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# ./waitpid
子进程, pid:29405, ppid:29404
子进程, pid:29406, ppid:29404
子进程, pid:29407, ppid:29404
子进程, pid:29408, ppid:29404
子进程, pid:29409, ppid:29404
父进程, pid:29404
父进程, pid:29404
父进程, pid:29404
父进程, pid:29404
回收了进程29405
父进程, pid:29404
父进程, pid:29404
回收了进程29406
父进程, pid:29404
父进程, pid:29404
回收了进程29407
父进程, pid:29404
父进程, pid:29404
回收了进程29408
父进程, pid:29404
父进程, pid:29404
回收了进程29409
父进程, pid:29404
父进程, pid:29404
父进程, pid:29404
父进程, pid:29404
^C
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3#
  
```

1 父进程创建5个子进程

2 父进程不断循环打印自身pid，并实时检测是否有子进程exit

3 pid=29405的子进程休眠结束调用exit，父进程检测到并回收资源

4 不阻塞式的不断回收剩余子进程

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：

```

root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# gcc wait.c -o wait
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3# ./wait
子进程, pid:29483, ppid:29482
子进程, pid:29484, ppid:29482
子进程, pid:29485, ppid:29482
子进程, pid:29486, ppid:29482
子进程, pid:29487, ppid:29482
父进程, pid:29482
回收了进程29483
父进程, pid:29482
回收了进程29484
父进程, pid:29482
回收了进程29485
父进程, pid:29482
回收了进程29486
父进程, pid:29482
回收了进程29487
父进程, pid:29482
父进程, pid:29482
父进程, pid:29482
父进程, pid:29482
父进程, pid:29482
^C
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_3#

```

1 父进程打印自身pid信息后，调用wait阻塞，无法继续打印pid信息。

2 直到有子进程休眠结束调用exit后，父进程回收资源，继续打印下一条pid信息。

3 再次被阻塞，等待下一个子进程休眠结束。

4 当全部进程资源回收完成，wait立即返回，不在阻塞，因此父进程循环打印自身pid信息。

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。

5 题目五（附加题）

5.1 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）

5.2 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：

```
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_5# ./start.sh
[2022-05-04 14:49:38] CPU使用率:5% 空闲内存: 1819M[173.25M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:49:48] CPU使用率:5% 空闲内存: 1819M[172.605M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:49:58] CPU使用率:97% 空闲内存: 1819M[172.68M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:49:58] 您的服务器CPU使用率过高, 达到97%, 请尽快处理!
成功向邮箱 jieyongwang@cumt.edu.cn 发送 [CPU使用率]监控报告 邮件
[2022-05-04 14:50:10] CPU使用率:5% 空闲内存: 1819M[172.402M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:50:20] CPU使用率:5% 空闲内存: 1819M[172.637M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:50:30] CPU使用率:5% 空闲内存: 1820M[171.984M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:50:40] CPU使用率:5% 空闲内存: 1820M[172.211M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:50:50] CPU使用率:5% 空闲内存: 1820M[172.285M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:51:00] CPU使用率:5% 空闲内存: 196M[1796M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:51:00] 您的服务器可用内存空间仅有196M, 请尽快处理!
成功向邮箱 jieyongwang@cumt.edu.cn 发送 [内存使用空间]监控报告 邮件
[2022-05-04 14:51:12] CPU使用率:5% 空闲内存: 1819M[172.617M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:51:22] CPU使用率:5% 空闲内存: 1819M[172.691M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:51:32] CPU使用率:5% 空闲内存: 1819M[172.797M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:51:43] CPU使用率:5% 空闲内存: 1818M[173.844M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:51:53] CPU使用率:5% 空闲内存: 1818M[173.918M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:52:03] CPU使用率:5% 空闲内存: 1818M[174.023M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:52:13] CPU使用率:5% 空闲内存: 1818M[173.984M|1992.77M] 空闲磁盘: 29G[11G|40G]
[2022-05-04 14:52:23] CPU使用率:5% 空闲内存: 1819M[173.484M|1992.77M] 空闲磁盘: 29G[11G|40G]
^C
root@iZ8vb48stipso6rv9jbbqeZ:~/linux_exam/problem_5#
```

图 5-1 运行结果

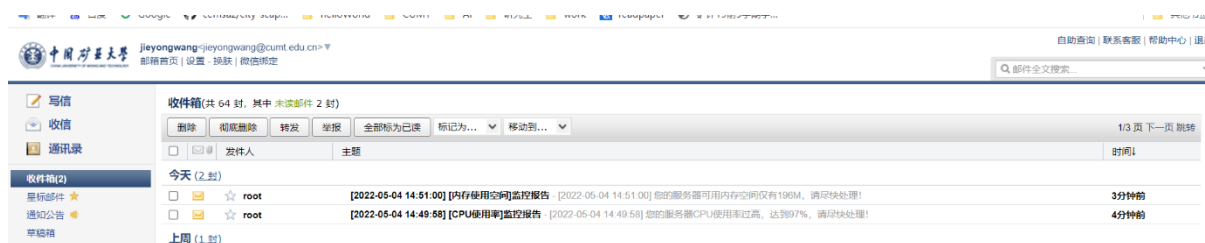


图 5-2 成功接收报警邮件

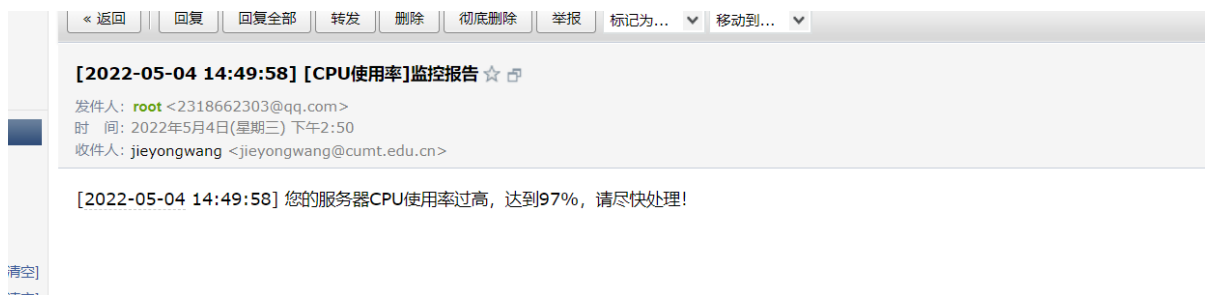


图 5-3 报警邮件内容 (1)



图 5-4 报警邮件内容 (2)

脚本的源代码如下：

start.sh

```

1. #!/bin/bash
2.
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
15.
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-$CPU_MSG" | bc)
22.
23.     # 全部内存减去已使用的内存，单位MB
24.     MEM_TOTAL=$(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 $1}')
27.
28.     DISK_TOTAL=$(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 $1}')
31.
32.     echo -e "$(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} ]"
33.
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
58.
59.     sleep $TIME
60. done
```
