**中国矿业大学计算机学院**

**2019级本科生课程作业**

|  |  |
| --- | --- |
| 课程名称 | Linux操作系统 |
| 作业次数 | 作业三（结课大作业） |
| 作业时间 | 2022年5月6日 |
| 姓名 | 胡钧耀 |
| 学号 | 06192081 |
| 专业 | 计算机科学与技术 |
| 任课教师 | 姜秀柱 |

目 录

[一、 基本命令（20'=2×10'） 1](#_Toc102770258)

[1.1 第一小问 1](#_Toc102770259)

[1.2 第二小问 5](#_Toc102770260)

[二、 shell编程（30'=10'+5×4'） 8](#_Toc102770261)

[2.1 问题分析与流程图绘制 8](#_Toc102770262)

[2.2 程序源代码 9](#_Toc102770263)

[2.3 运行截图 11](#_Toc102770264)

[三、 socket编程（30'=18'+12'） 15](#_Toc102770265)

[3.1 TCP运算服务 15](#_Toc102770266)

[3.2 UDP聊天室 18](#_Toc102770267)

[四、 进程（20'） 24](#_Toc102770268)

[4.1 问题分析 24](#_Toc102770269)

[4.2 程序源代码 24](#_Toc102770270)

[4.3 运行结果截图 27](#_Toc102770271)

[五、 附加（20'） 31](#_Toc102770272)

[5.1 问题分析 31](#_Toc102770273)

[5.2 程序源代码 31](#_Toc102770274)

[5.3 运行结果截图 32](#_Toc102770275)

[六、 感悟 33](#_Toc102770276)

[参考文献 33](#_Toc102770277)

# 基本命令（20'=2×10'）

题目：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。

（1）查看当前目录，在当前目录下创建一个新目录，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。

（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。

## 第一小问

该问题为基本命令的操作，根据问题描述，写出每一条要求对应命令的命令即可，分析问题可知，该小问的可以被拆解为如下步骤。

### 查看当前目录

使用pwd命令直接查询即可，本次作业的文件夹是Linux课程的结课实验的第一题，即*~/LinuxCourse/final\_lab/t1*，因此输入该命令后应该在命令行得到同样结果。

pwd

命令输入后结果如下，得到当前目录的路径*~/LinuxCourse/final\_lab/t1*。



图 1.1‑1 查看当前目录

### 在当前目录下创建一个新目录

使用mkdir命令可以创建新目录，因此创建名为*newfolder*的文件夹。为了验证*newfolder*是否创建成功。

mkdir newfolder

命令输入后结果如下，首先当前文件夹下为空，创建命令结束后，ls命令只列出了一个名为*newfolder*文件夹，说明创建成功。

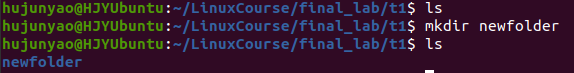


图 1.1‑2 创建目录

### 然后进入这个新目录

进入目录使用cd命令，直接进入*newfolder*文件夹即可。

cd newfolder

命令输入后，观察到命令行提示已经进入了*newfolder*文件夹内，即当前地址为*~/LinuxCourse/final\_lab/t1/newfolder*。或者使用pwd验证当前目录也可以。



图 1.1‑3 进入目录

### 在这个新目录下创建一个空文件

创建空文件使用touch命令，直接创建*newfile.txt*即可。可使用ls命令进行验证，查看当前文件夹内是否有名为*newfile.txt*的文件。

touch newfile.txt

命令输入后结果如下，首先当前文件夹下为空，创建命令结束后，ls命令只列出了一个名为*newfile.txt*的文件，说明创建成功。

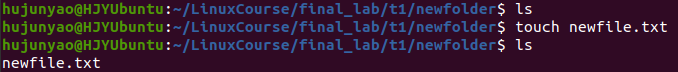


图 1.1‑4 创建空文件

### 查看该文件的简单列表

这一小问其实不是太懂具体问的什么，是指“该文件夹下的文件简单列表”还是说“该文件的简单信息列表”，但看命令的话，对于前者应该是使用ls命令，对于后者好像并没有简单信息，只有详细信息（下文讲提到）。

ls

命令输入后，观察到命令行提示当前文件夹只有*newfile.txt*一个文件。



图 1.1‑5 查看文件列表

### 查看该文件的文件类型

这里使用file命令即可查看该文件的类型。

file newfile.txt

命令输入后，提示*newfile.txt*是一个空文件，而不是文本文件。



图 1.1‑6 查看文件类型

### 查看该文件的详细属性

这里使用stat命令，其用于显示文件或文件系统的详细信息。

stat newfile.txt

命令输入后，*newfile.txt*文件的名称、文件大小、inode结点信息、链接信息、访问时间、更改时间等详细信息。

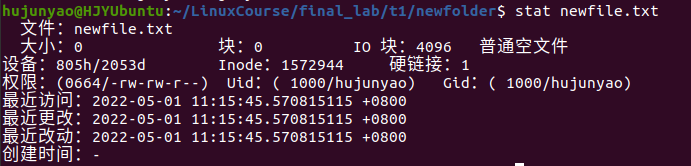


图 1.1‑7 查看文件详细属性

### 查看该文件的所占空间

这里使用du命令，其用于显示文件或文件系统的详细信息。

du newfile.txt

命令输入后，*newfile.txt*文件的大小将被显示出来，因为是空文件，只有0字节。



图 1.1‑8 查看文件详细属性

### 修改该文件的所有者为root

这里使用chown命令，修改*newfile.txt*的所有者为root，因为涉及到管理员，因此需要加上sudo前缀，扩大命令权限。

sudo chown root newfile.txt

命令输入前后，使用ls -l命令可以查看*newfile.txt*的所有者是否被更改，可知已经从hujunyao变成了root，说明所有者已经修改成功。

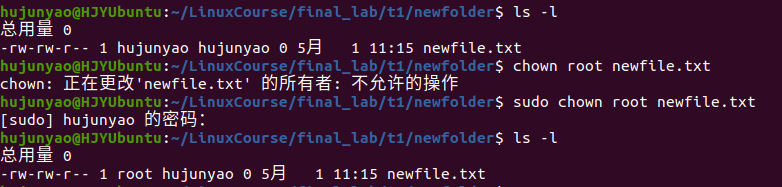


图 1.1‑9 修改文件所有者

### 修改该文件所有者读写执行完全权限

这里使用chmod命令，修改*newfile.txt*的所有者user的读写修改权限为完全权限，也就是rwx，同时需要加上sudo前缀，扩大命令权限。

sudo chmod u=rwx newfile.txt

命令输入前后，使用ls -l命令可以查看*newfile.txt*的所有者是否被更改，可知该文件的权限位已经从已经从-rw-rw-r--变成了-rwxrw-r--，说明已经修改成功。

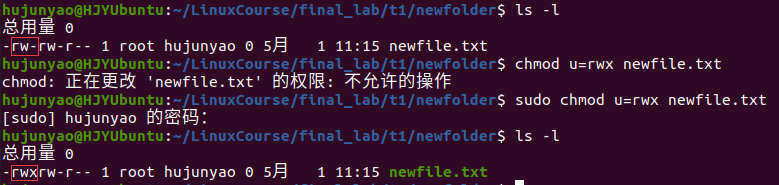


图 1.1‑10 修改文件读写执行权限

### 修改该文件的有效时间为2027年8月1日24时

这一小问也其实不是太懂具体问的什么，是指“修改用户密码的有效时间”，还是“修改该文件的一种时间（访问、更改、改动时间）”，个人理解是修改后者的更改时间，这个可以使用stat命令对文件时间进行修改，但是在题目中“24时”是不存在的，而且通常在生活中使用24小时制的时间也是从00:00-23:59，不存在“24时”的说法，就像进制一样，应该逢24进1，因此认为和题目对应的表达也就是“2027年8月2日0时”。使用该命令时，同时需要加上sudo前缀，扩大命令权限。m代表修改更改时间（modified time），t代表时间戳（timestamp），其语法是[[CC]YY]MMDDhhmm[.ss]。

sudo touch -mt 202708020000.00 newfile.txt

命令输入前后，使用touch命令可以查看*newfile.txt*的修改时间，用来验证文件的更改时间是否被修改。通过命令行的返回信息可知，修改时间已经从2022-05-01 11:15:45修改为了2027-08-02 00:00:00，说明修改成功。同时根据返回信息也可知“202708012400”这种日期是不符合语法的，hh的范围是00-23。



图 1.1‑11 修改文件日期

### 回到当前目录

返回到当前目录，也就是回到*~/LinuxCourse/final\_lab/t1*路径，这是当前文件夹的父级目录，使用..即可返回到指定的目录。

cd ..

命令输入前后，使用pwd命令可以查看当前路径地址，可知最后已经回到了*~/LinuxCourse/final\_lab/t1*，验证成功。

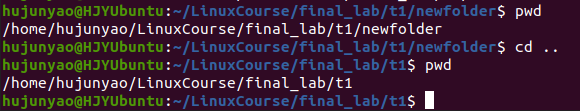


图 1.1‑12 回到当前目录

## 第二小问

该问题同样是为基本命令的操作，根据问题描述，其中有一些步骤具有前者的输出是后者的输入的特征，因此不单单是写出每一条要求对应命令的命令即可，最好要能使用管道来优化命令，简化输入，分析问题可知，该小问可以被拆解为如下步骤。

### 用屏幕输出命令将"hello, world!"写入操作（1）建立的文件中

屏幕输出命令就是echo，写入文件需要使用重定向，也就是将输出的文字使用覆盖写>输入到已经建立好的文件*newfolder/newfile.txt*中。

echo "hello, world!" > newfolder/newfile.txt

命令输入前后，使用cat命令可以查看*newfolder/newfile.txt*的内容是否经过了修改，未修改前，文件为空，输入命令后，输出了"hello, world!"，说明已经写入文件。

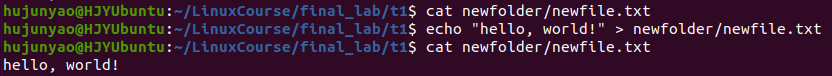


图 1.2‑1 写入文件

### 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，并计算该文件的单词数

既要写入文件，又要输出到屏幕上，此时光用echo不行，同时需要使用tee命令进行屏幕输出和管道输出，在添加姓名-学号对时。由于要输入三组，每组一行，为了能输入换行符\n，需要对echo命令加上-e参数。由于要进行追加写操作，对tee命令加上-a参数。最后要计算文件单词数，这里不能使用管道，而是用单独的命令，因为前一个tee命令输出了只是三组姓名-学号对，不包括文件中的第一行"hello, world!"，因此要用分号分隔开，单独使用wc命令，而不能使用 | wc，否则只统计三行数据。

echo -e "hujunyao-06192081\nzhangsan-11111111\nlisi-22222222" |

tee -a newfolder/newfile.txt; wc newfolder/newfile.txt

输入命令后，命令上输出了三行姓名-学号对，同时统计出了文件的字符统计数量，单词数是5个（"hello,", "world!"和三组姓名-学号对）。在该命令输入前后，使用cat命令可以查看*newfolder/newfile.txt*的内容是否经过了修改，未修改前，文件只有一行，输入命令后，文件为四行，说明已经写入文件。

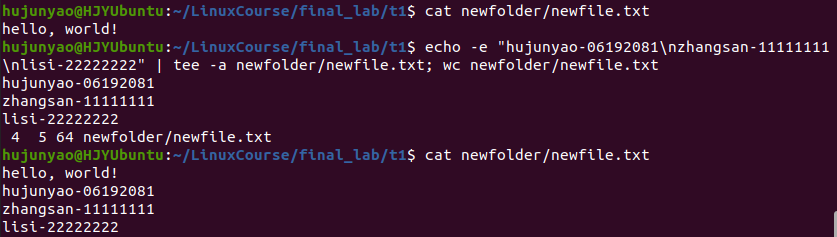


图 1.2‑2 键盘输入与单词统计

### 对文件内容按行排序，输出最后一行

按照行排序使用sort命令，输出最后一行使用tail命令，并加上-1参数设置只输出最后一行，使用管道后tail无需指定文件名，而是将sort的输出结果作为输入。

sort newfolder/newfile.txt | tail -1

这个文件后四行，开头分别为he，hu，z，l，排序结果是he，hu，l，z，所以输出结尾最后一行的结果应该是zhangsan-11111111，结果验证正确。

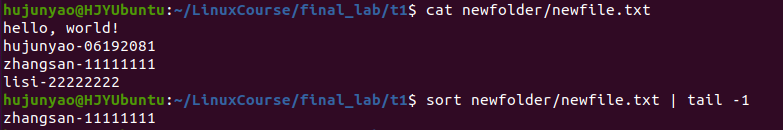


图 1.2‑3 排序与输出结尾

### 将文件中的学号提取出来输出到一个新建文件中

提取具有规则的信息可以使用正则表达式，也就是使用grep命令，学号在这里干扰项比较少，直接识别所有的连续数字即可，也就是[0-9]\*，同时使用了-o参数截取匹配的字符串，输出到新建文件可以使用管道，使用tee命令输出识别到的学号的同时也将其保存到文件*newfolder/stunum.txt*中。

grep -o "[0-9]\*" newfolder/newfile.txt | tee newfolder/stunum.txt

输入该命令前后可以使用cat命令来查看是否将搜索到的学号存入新文件中，验证结果说明三个学号都已经存入*newfolder/stunum.txt*，验证成功。

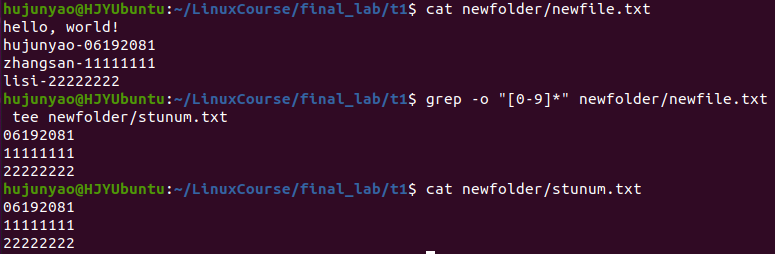


图 1.2‑4 提取信息并保存

### 比较这两个文件，比较结果输出到第三个文件中

比较文件使用diff命令，随后是两个需要比较的文件，也就是*newfolder/newfile.txt*和*newfolder/stunum.txt*进行比较，输出比较结果需要使用管道，使用tee命令输出识别到的学号的同时也将其保存到文件*newfolder/diff.txt*中。

diff newfolder/newfile.txt newfolder/stunum.txt |

tee newfolder/diff.txt

输入该命令后可以使用cat命令来查看是否将文件差异存入新文件中，验证成功。

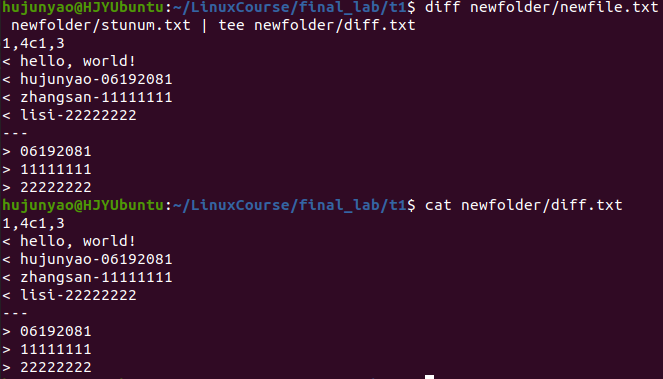


图 1.2‑5 比较文件并保存

### 将该目录下的三个文件拷贝到其父目录中

拷贝文件使用cp命令，这里拷贝文件夹下的所有文件，且没有子文件夹，所以直接使用星号选择文件即可。

cp newfolder/\* ..

输入该命令前后可以使用ll命令来查看是否将文件拷贝到父目录中，验证结果看到三个文件（*newfile.txt*，*stunum.txt*，*diff.txt*）都成功拷贝到*~/LinuxCourse/final\_lab*。

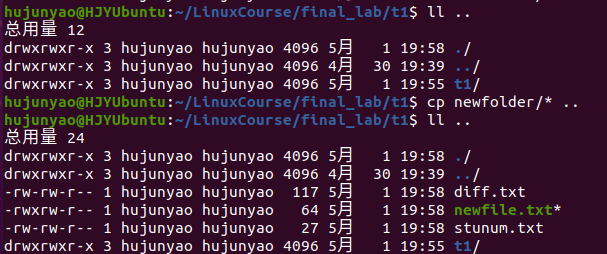


图 1.2‑6 拷贝文件

### 将该目录删除

该目录说的也就是*~/LinuxCourse/final\_lab/t1*目录，首先使用cd命令退回到父级目录，然后使用rm命令删除该文件夹，同时加上-r递归参数。

cd ..

rm -r t1

输入该命令前后可以使用ls命令来查看是否目录是否已被删除，验证结果正确。

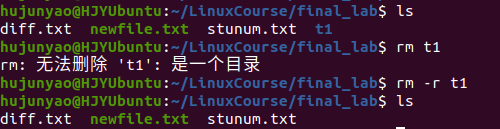


图 1.2‑7 删除目录

# shell编程（30'=10'+5×4'）

题目：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。

## 问题分析与流程图绘制

根据分析，分析后的程序主要流程图如下所示。由题目可知，需要编写的程序需要接受两种合法类型的参数数量，一种是没有参数，一种是有参数的，还有一种大于一个参数的非法情况。问题提到了首次登录的问题，认为应该把判断用户是否首次登陆放在运行该程序最前面，就如同网站用户登录一样，只有先修改不安全的密码才能进行以后的后续操作。对于没有参数的情况，是进行一系列用户的操作，如进入目录、创建文件等，相当于复写了Linux系统的文件基本操作，对于一个参数的情况，输入的参数是新建的用户名称，这里规定了只有root管理员才能进行新用户的创建，避免混乱，之后，还需要判断将要创建的用户是否存在，只有不存在的时候才能继续进行用户的创建。

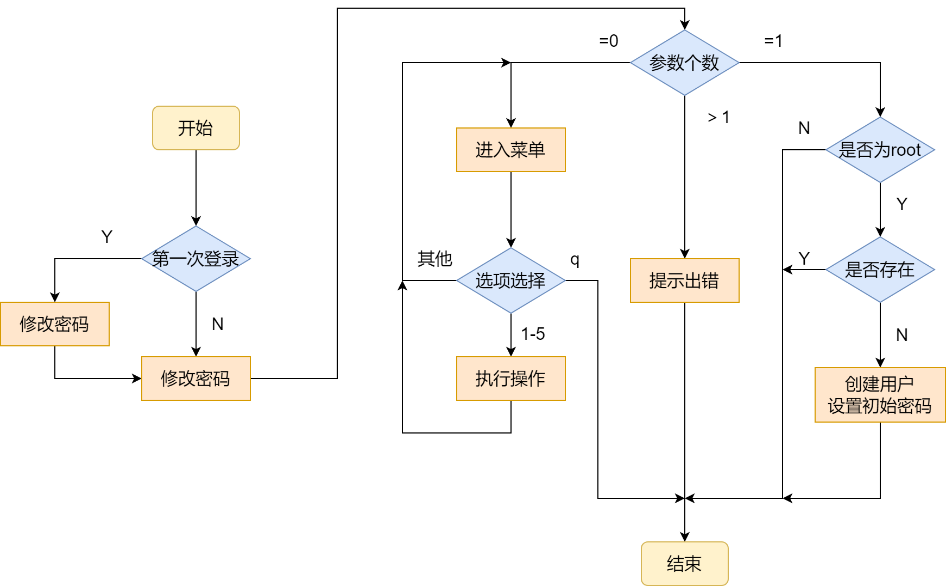


图 2.1‑1 Shell编程程序主要流程图

## 程序源代码

#!/bin/bash

# 检查用户是否首次登陆

if [ $(last | grep $(whoami) | wc -l) -le  1 ]

then

  echo "检测到首次登陆，您的密码过于简单，请重新设置"

  passwd $username

fi

# 判断输入参数个数

if [ $# -eq 0 ]

then

    echo "没有参数"

    echo ""

    while : true

    do

        echo ""

        echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*菜单\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

        echo "请根据下列提示选择操作: "

        echo "1. 显示用户选择的目录内容"

        echo "2. 按照用户输入的目录切换路径"

        echo "3. 按照用户输入的文件名在/home目录中创建文件"

        echo "4. 编辑用户输入的文件"

        echo "5. 删除用户选择的文件"

        echo "q. 退出"

        echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

        echo ""

        read -p "请选择 1-5: " option

        echo "你输入的是: " $option

        case $option in

        "1")

            echo "当前地址"

            pwd

        ;;

        "2")

            read -p "请输入需要切换的路径: " address;

            cd $address

            if [ $? -eq 0 ] #判断用户是否添加成功

            then

                    echo "已成功进入 $address"

            else

                    echo "未能进入 $address，请重试"

            fi

        ;;

        "3")

            read -p "请输入需要新建的文件: " filename;

            cd /home/$(whoami)

            touch $filename

            if [ $? -eq 0 ] #判断文件是否添加成功

            then

                    echo "已成功创建 $filename"

            else

                    echo "未能创建 $filename，请重试"

            fi

        ;;

        "4")

            read -p "请输入需要编辑的文件: " filename

            vim $filename

            if [ $? -eq 0 ] #判断文件是否编辑成功

            then

                    echo "已成功编辑 $filename"

            else

                    echo "未能编辑 $filename，请重试"

            fi

        ;;

        "5")

            read -p "请输入需要删除的文件: " filename

            rm $filename

            if [ $? -eq 0 ] #判断文件是否删除成功

            then

                    echo "已成功删除 $filename"

            else

                    echo "未能删除 $filename，请重试"

            fi

        ;;

        "q")

            echo "即将退出..."

            exit

        ;;

        \*)

            echo "参数错误，请重新输入..."

            continue

        ;;

        esac

    done

elif [ $# -eq 1 ]

then

    username=$1

    echo "输入的用户名是 $username"

    if [ $UID -ne 0 ]

    then

        echo "请切换到root用户进行新增用户！"

        exit 0

    fi

    if id -u $username >/dev/null 2>&1;

    then

            echo "用户 $username 存在，请重新输入"

            exit 0

    else

            echo "用户 $username 不存在，即将创建..."

            echo -e "123456\n123456\n\n\n\n\n\n\n" | adduser $username

            echo -e "\033[32m 默认密码为123456，请登陆后及时修改\033[0m"

            id $username

        if [ $? -eq 0 ] #判断用户是否添加成功

        then

            echo "已成功添加用户 $username "

        else

            echo "未能成功添加用户 $username "

        fi

    fi

else

    echo "参数太多了，有问题啊，重开吧"

    echo ""

fi

## 运行截图

### 参数数量测试

以hujunyao用户运行*t2.bash*程序，参数为0个，进入菜单界面；参数为1个，进入创建用户功能，但因非root管理员用户而退出程序；参数为1个以上，因参数过多而退出程序。三种情况的运行截图如下所示。



图 2.3‑1 参数数量测试

### 用户菜单操作测试

以hujunyao用户以0个参数运行*t2.bash*程序。进入菜单界面。

输入数字1，输出了当前所在的目录为*/home/hujunyao/LinuxCourse/final\_lab/t2*，然后再次循环运行到菜单界面。截图如下所示。



图 2.3‑2显示用户当前目录

输入数字2，程序要求输入需要跳转的路径，输入*/home/hujunyao*作为需要跳转的路径，然后程序将显示是否跳转成功，然后再次循环运行到菜单界面。截图如下所示。

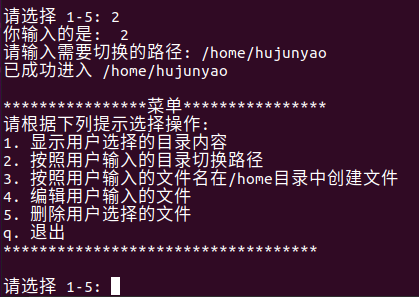


图 2.3‑3 切换用户当前路径

输入数字3，程序要求输入需要创建的文件名称，输入*test.txt*作为创建的文本文件，然后程序将显示是否创建成功，然后再次循环运行到菜单界面。可以使用ll和grep组合命令验证是否已经创建该文件，截图如下所示。

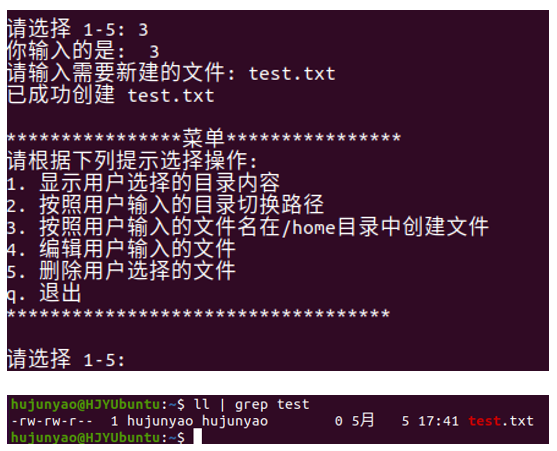


图 2.3‑4 创建文件

输入数字4，程序要求输入需要修改的文件名称，输入*test.txt*作为修改的文本文件，也就是之前创建的那一个文件，然后命令行将进入vim编辑器模式，可以进行文本的修改，随后将文件进行保存，然后再次循环运行到菜单界面。使用cat命令验证是否已经修改过该文件，截图如下所示。

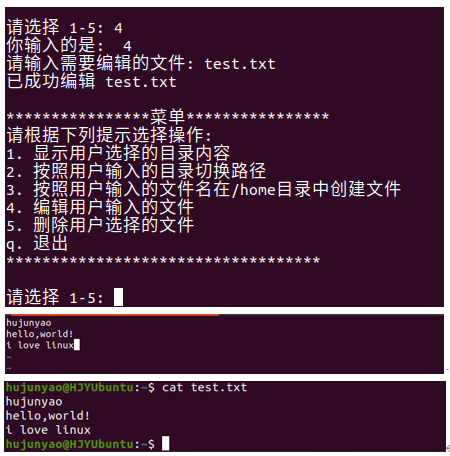


图 2.3‑5 修改文件

输入数字5，程序要求输入需要删除的文件名称，输入*test.txt*作为将要被删除的文本文件，也就是之前创建的那一个文件，然后再次循环运行到菜单界面。可以使用ll和grep组合命令验证是否已经删除该文件，截图如下所示。

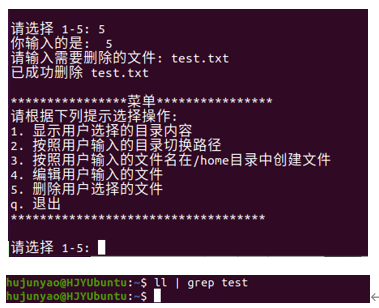


图 2.3‑6 删除文件

输入q，程序直接退出。输入其他非法选项，将提示重新输入，然后再次循环运行到菜单界面。截图如下所示。

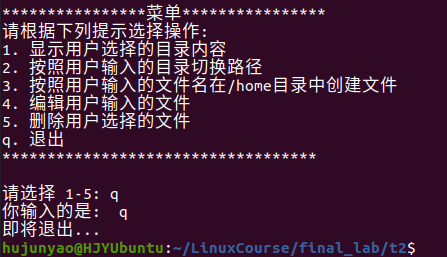
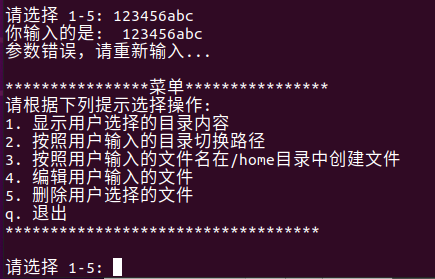
 

图 2.3‑7退出程序与非法输入

### 创建用户操作测试

以hujunyao用户以1个参数usertest运行*t2.bash*程序。由于hujunyao并不是root用户，会直接提示需要root权限并退出程序。因此需要使用su命令将当前用户设置为root管理员，再以1个参数usertest运行*t2.bash*程序。此时提示成功创建usertest，为了验证是否创建成功，同时测试程序是否能检测到参数代表的用户已经被创建，再在root管理员模式下以1个参数usertest运行*t2.bash*程序，此时程序将提示用户已存在，说明已经创建了usertest新用户，并且初始密码是123456。

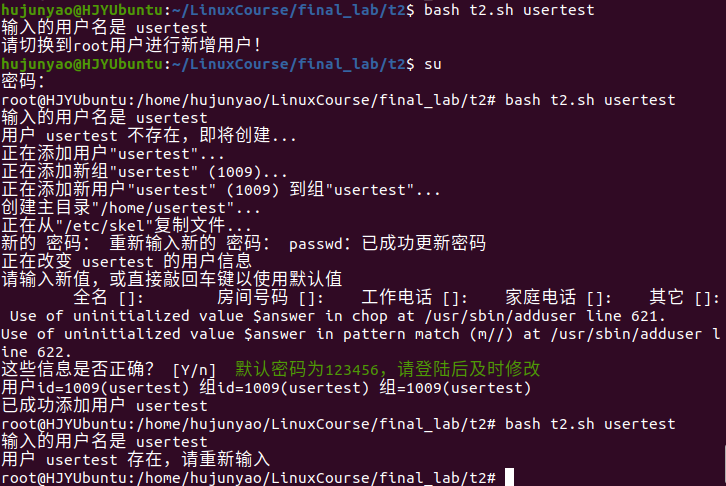


图 2.3‑8 创建用户

### 登录新用户提示修改密码测试

在root管理员模式下使用login命令登录新的用户usertest，使用初始密码123456可以成功登录，usertest用户以0个参数运行*t2.bash*程序，程序提示首次登录修改密码。成功修改密码后，程序继续运行，也就是进入菜单界面，作为usertest用户同样可以进行如下操作，但不能以1个参数运行该程序，因为usertest并没有root权限。

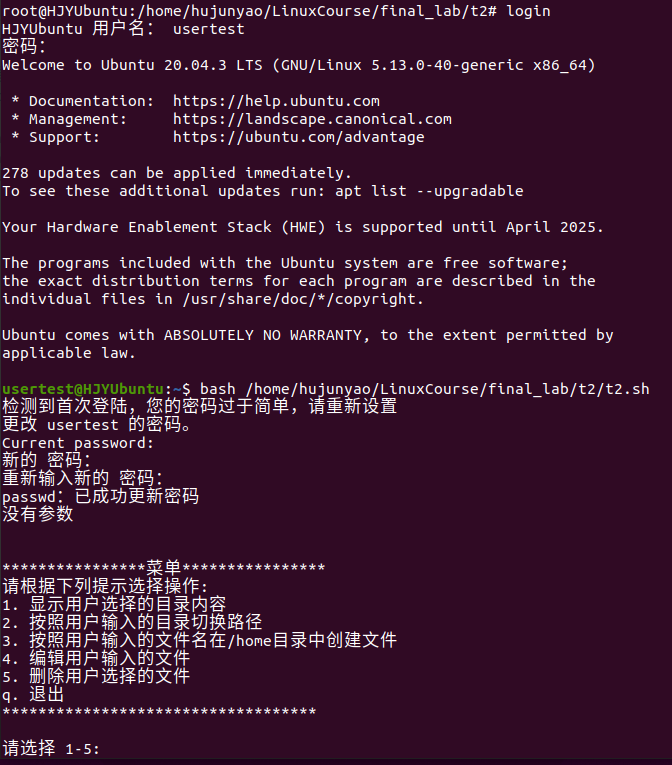


图 2.3‑9修改新用户首次登陆密码

# socket编程（30'=18'+12'）

题目：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。

## TCP运算服务

### 问题分析

本题选择实现加减乘除的运算服务，主要思路是利用TCP协议的相关方法，在客户端上输入需要计算的表达式，传给服务器，由服务器使用eval方法进行计算，如果能计算出结果，就将结果返回给客户端，如果计算出错，将出错结果返回给客户端，支持客户端连续发送不同的表达式进行持续计算。下图是TCP客户端和TCP服务器的运算服务流程图。

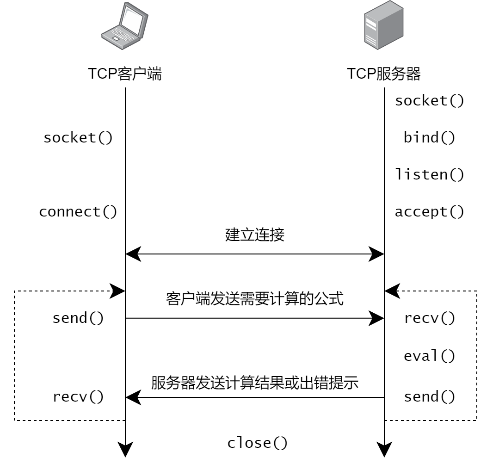


图 3.1‑1 TCP运算服务流程图

### 程序源代码

如下是客户端的源代码*client.py*。

#! /usr/bin/env python3

import socket

import time

print("客户端已开启")

sk=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

ip\_port=('192.168.175.128',5000)

while True:

    try:

        sk.connect(ip\_port)

        break

    except ConnectionRefusedError:

        print("服务器拒绝连接，继续尝试连接...")

        time.sleep(2)

print("socket已连接")

print(sk)

print("")

while True:

    message = input("请输入四则运算公式（输入exit退出）")

    sk.send(message.encode('utf-8'))

    if message==('exit'):

        print("客户端主动退出...")

        break

    print("[send] [" + time.ctime(time.time()) + "] " + message)

    print("服务器正在计算中...")

    data=sk.recv(1024)

    print ("[recv] [" + time.ctime(time.time()) + "] " + data.decode('utf-8'))

    print("计算结果为: " + data.decode('utf-8'))

    print("")

sk.close()

print ("退出")

如下是服务器的源代码*server.py*。

#! /usr/bin/env python3

import socket

import time

print("服务器已开启")

sk=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

ip\_port=('192.168.175.128',5000)

sk.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

sk.bind(ip\_port)

sk.listen(1)

print("socket已连接")

a,b=sk.accept()

print(a)

print("")

while True:

    data=a.recv(1024)

    recv\_msg = data.decode('utf-8')

    if recv\_msg == "exit":

        print("服务器被动退出...")

        break

    print ("[recv] [" + time.ctime(time.time()) + "] " + recv\_msg)

    print("服务器正在计算中...")

    try:

        result = str(eval(recv\_msg))

    except ZeroDivisionError:

        result = "【错误：除数为0】"

    except NameError:

        result = "【错误：暂不支持变量名输入】"

    except SyntaxError:

        result = "【错误：公式输入有误】"

    print("计算结果为: " + result)

    a.send(result.encode('utf-8'))

    print("[send] [" + time.ctime(time.time()) + "] " + result)

    print("")

sk.close()

print ("退出")

### 程序运行截图

首先配置好服务器的IP地址，然后开启两个命令行，先运行*server.py*，后运行*client.py*程序，将在建立连接后，双方提示已经建立连接，服务器的端口为5000，客户端的端口为50658，且客户端提示可以输入四则运算公式，输入公式后，服务器将接收到客户端发送的公式并进行计算，然后把计算结果返回客户端。同时为了测试程序的鲁棒性，同时测试了除数为零、公式语法有误、输入了字母变量的三种非法情况，均能返回相应错误提示，验证效果较好，最后退出客户端程序，服务器也收到消息并停止服务。执行过程截图如下所示。

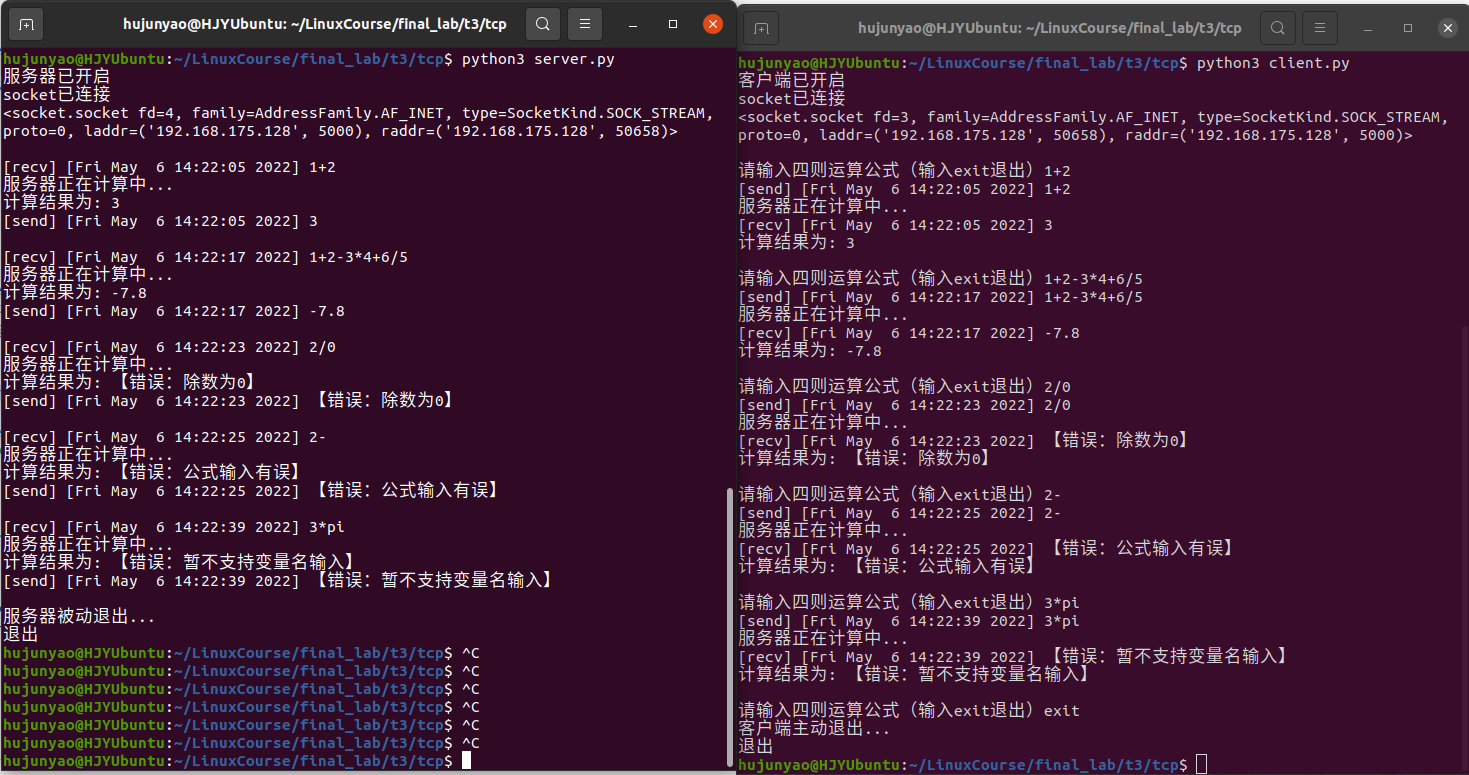


图 3.1‑2 TCP运算服务运行情况

若先运行*client.py*，再运行*server.py*程序，客户端将无法连接到服务器，将不断尝试进行连接，直到服务器开始服务。运行状态如下所示。



图 3.1‑3 客户端等待服务器开始服务

## UDP聊天室

### 问题分析

本题要求实现用户聊天室服务，主要思路是利用UDP协议的相关方法，在客户端上输入需要发送的消息，传给服务器，由服务器对其发送的信息进行转发给别的和服务器连接的客户端，每个客户端能从服务器接收到来自其他客户端的消息，并能在聊天室展示是否有新的客户端上线，或者老的客户端下线，同时，服务器只会把消息发送给正在上线的客户端，已经下线的或者没有上线的客户端将不进行发送。为了实现同一用户可以连续发送信息，并且接收方可以即时收到消息，构建了发送消息和接收消息两个线程，也就避免了必须双方各说一句才能继续进行发言的问题。下图是UDP客户端和UDP服务器的聊天服务流程图。

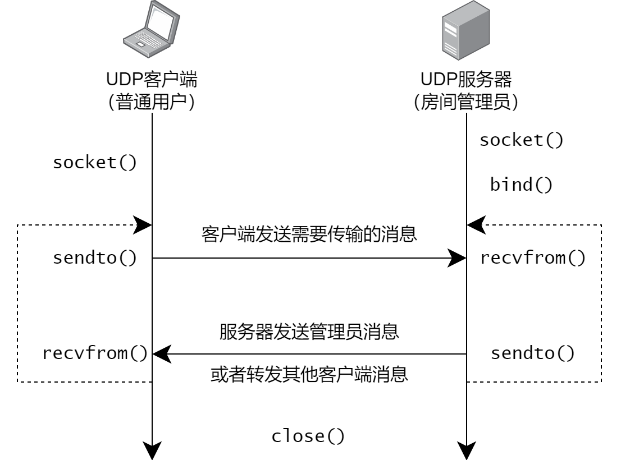


图 3.2‑1 UDP聊天服务流程图

聊天室支持多人进行聊天，而不仅仅是一个客户端和一个服务器进行信息互发，可以有多个客户端同时发送消息，其基本思想是将服务器作为信息收发中心，普通用户想要发送到聊天室的消息需要先发送到服务器，服务器再通过已经建立连接的客户端集合expected\_ip\_port依次进行转发，使当前除了发送信息的在线客户端用户自己和服务器本身，其他人都能收到消息。如果是管理员自行发布消息，将发送给所有在线的客户端用户。UDP聊天室多人转发功能演示拓扑图如下所示。

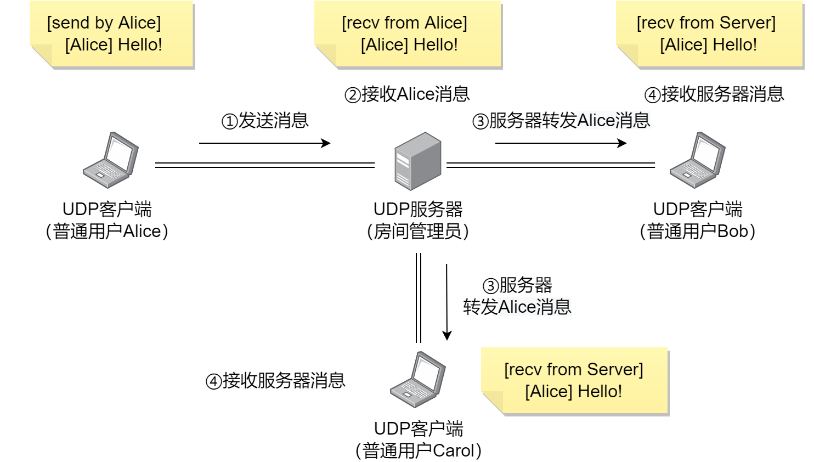


图 3.2‑2 UDP聊天室多人转发功能演示拓扑图

聊天室只会把消息传送给和服务器建立连接的在线用户，其实现思路主要是当某一客户端将要和服务器建立连接时，首先向服务器发送上线通知，这样服务器就能记录客户端的地址并进行保存，放进在线客户端集合expected\_ip\_port当中，后续如果有客户端发送的消息或者管理员的消息，可以通过集合记录来发送消息。同理，如果有客户端用户需要下线，客户端最后将给服务器发送下线通知，服务器识别到下线通知，将把对应的客户端用户从在线客户端集合expected\_ip\_port当中移除，后续如果有客户端发送的消息或者管理员的消息，就不能够发送给该客户端了。如果服务器管理员需要关闭聊天室，直接退出聊天室，这样其他的客户端用户也就不能通过服务器的转发功能进行聊天了，发送的信息无法被接收，也无法接收到其他用户发送的消息。UDP聊天室上线/下线功能与转发功能联动流程图如下所示。

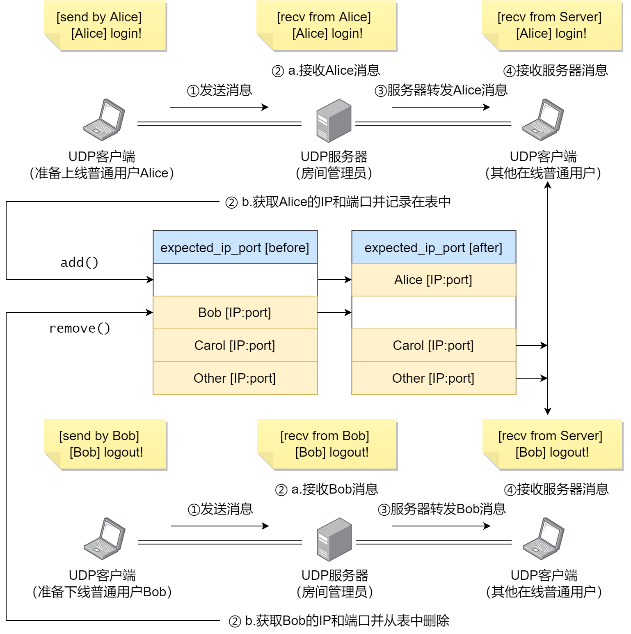


图 3.2‑3 UDP聊天室上线/下线功能与转发功能联动流程图

### 程序源代码

如下是客户端的源代码*client.py*。

#! /usr/bin/env python3

import socket

import threading

import time

import os

def sendThread(name):

    while True:

        message = input()

        if message == 'exit':

            sk.sendto(("[" + myname + "] 退出了聊天室").encode('utf-8'), ip\_port)

            print("退出聊天室")

            sk.close()

            os.\_exit(0)

        message = '[' + myname + '] ' + message

        sk.sendto(message.encode('utf-8'), ip\_port)

        print("[send] [" + time.ctime(time.time()) + "]")

        print(message)

        print("")

def recvThread(name):

    while True:

        data=sk.recvfrom(1024)#客户端发送的数据存储在recv里，1024指最大接受数据的量

        if data[0].decode('utf-8') == "exit":

            print("客户端被动关闭socket")

            break

        print ("[recv] [" + time.ctime(time.time()) + "]")

        print(data[0].decode('utf-8'))

        print("")

    print ("recv退出线程：" + name)

sk = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

ip = '192.168.175.128'

port = 10000

ip\_port=(ip, port)

print(sk)

print("")

myname = input("输入你的聊天昵称：")

sk.sendto(("[" + myname + "] 进入了聊天室").encode('utf-8'), ip\_port)

# 创建新线程

aRecvThread = threading.Thread(target=recvThread ,args=("recv",))

aSendThread = threading.Thread(target=sendThread ,args=("send",))

aRecvThread.start()

aSendThread.start()

print("")

aRecvThread.join()

aSendThread.join()

如下是服务器的源代码*server.py*。

#! /usr/bin/env python3

import socket

import threading

import time

import os

def sendThread(name):

    while True:

        message = input()

        if message == 'exit':

            sk.sendto(("[" + myname + "] 关闭了聊天室").encode('utf-8'), ip\_port)

            print("关闭聊天室")

            sk.close()

            os.\_exit(0)

        message = "[管理员]" + message

        for i in expected\_ip\_port:

            sk.sendto(message.encode('utf-8'), i)

        print("[send] [" + time.ctime(time.time()) + "]")

        print(message)

        print("")

def recvThread(name):

    global expected\_ip\_port

    while True:

        recv\_data=sk.recvfrom(1024)#客户端发送的数据存储在recv里，1024指最大接受数据的量

        if recv\_data[1] not in expected\_ip\_port:

            for i in expected\_ip\_port:

                sk.sendto(recv\_data[0], i)

            expected\_ip\_port.add(recv\_data[1])

            print("[" + time.ctime(time.time()) + "]")

            print(recv\_data[0].decode('utf-8'))

            print("")

        elif "] 退出了聊天室" in recv\_data[0].decode('utf-8'):

            expected\_ip\_port.remove(recv\_data[1])

            for i in expected\_ip\_port:

                sk.sendto(recv\_data[0], i)

            print("[" + time.ctime(time.time()) + "]")

            print(recv\_data[0].decode('utf-8'))

            print("")

        else:

            for i in expected\_ip\_port:

                if i != recv\_data[1]:

                    sk.sendto(recv\_data[0], i)

            print("[recv] [" + time.ctime(time.time()) + "]")

            print(recv\_data[0].decode('utf-8'))

            print("")

    sk.close()

sk = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

ip = '192.168.175.128'

port = 10000

ip\_port=(ip, port)

expected\_ip\_port = set({})

sk.bind(ip\_port)

print(sk)

print("")

myname = '管理员'

print("聊天室已开启")

# 创建新线程

aRecvThread = threading.Thread(target=recvThread ,args=("recv",))

aSendThread = threading.Thread(target=sendThread ,args=("send",))

aRecvThread.start()

aSendThread.start()

print("")

aRecvThread.join()

aSendThread.join()

### 程序运行截图

为了验证程序的可行性，在系统中开启了一个服务器和三个客户端（Alice，Bob，Carol），先打开服务器，三个客户端分别在不同时刻进入聊天室，在不同时刻退出聊天室。主要验证的几种情况如下所述。

1. 服务器开启时，客户端能够正常进行上线、下线聊天室；
2. 服务器开启服务时，客户端发送信息，服务器和其他的在线的客户端都能收到消息，未上线和已经下线的客户端不会接收消息；
3. 服务器开启服务时，客户端可以连续发送信息和即时连续接收信息；
4. 服务器关闭服务后，上线的客户端既不能发送信息，也不能接收信息。

主要验证过程截图如下所示。

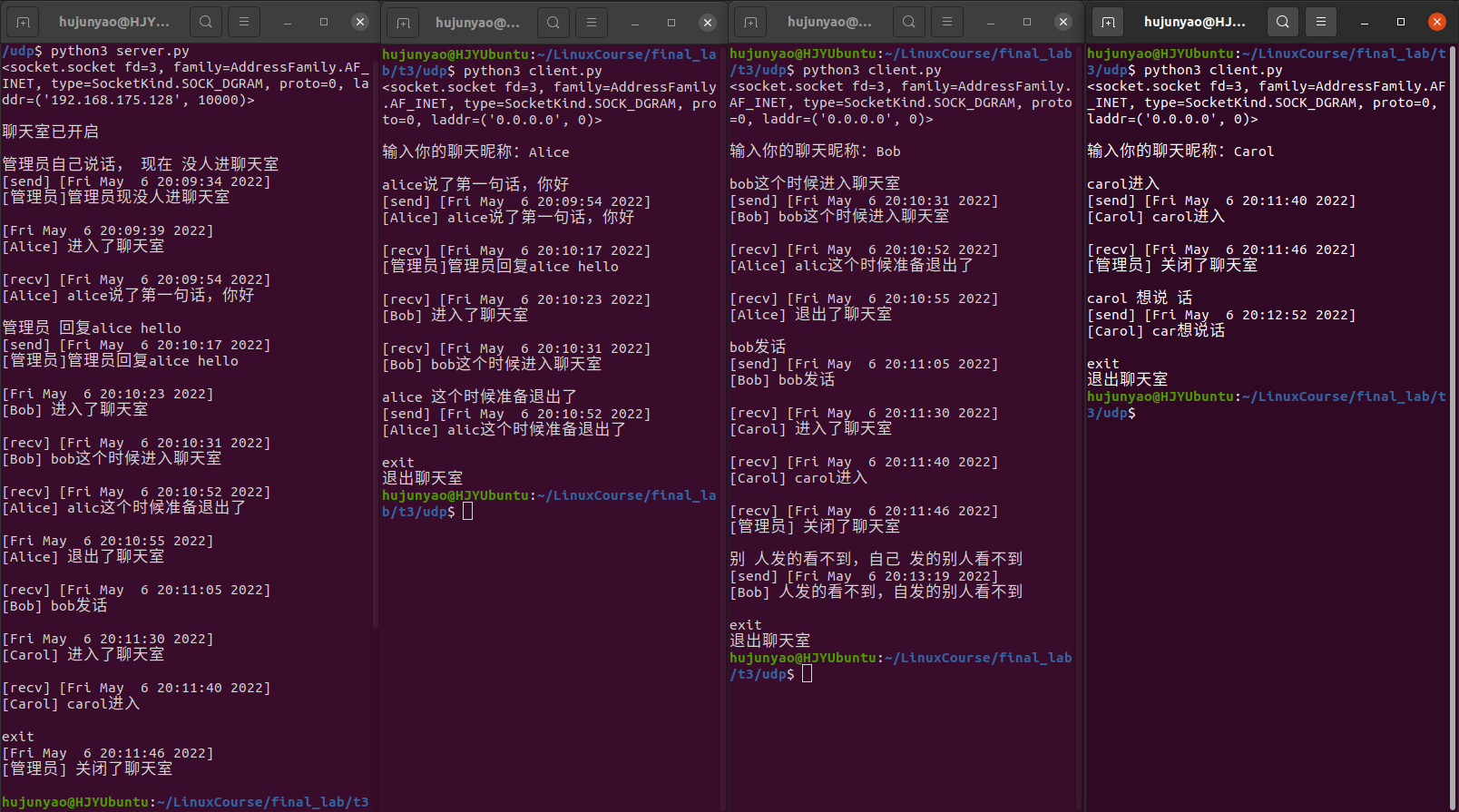


图 3.2‑4 UDP聊天室功能测试

# 进程（20'）

题目：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。

## 问题分析

## 程序源代码

如下是被调用的欢迎语句源代码*hello.py*。

#! /usr/bin/env python3

import sys

import os

def hello(name):

    print(f'\t\t\t\t{os.getppid()}:{os.getpid()} [调] Hello, {name}!')

hello(sys.argv[1])

如下是测试四个函数作用的主程序源代码*test.py*。

#! /usr/bin/env python3

import os

import time

def testForFork():

    def parent():

        for i in range(2):

            print(f'\t\tfork {i} ready , {os.getpid()}')

            newpid = os.fork()

            if newpid == 0:

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [子{i}] Hello')

            else:

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [父{i}] Hello')

            print(f'\t\tfork {i} finish , {os.getpid()}')

    print("\tparent() ready", os.getpid())

    parent()

    print("\tparent() finish", os.getpid())

def testForExec():

    def parent():

        for i in range(2):

            print(f'\t\tfork {i} ready , {os.getpid()}')

            pid = os.fork()

            if pid == 0:

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [子{i}] Hello ready')

                os.execl("hello.py", 'hello', 'hujunyao')

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [子{i}] Hello finish')

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [子{i}] Hello exit')

            else:

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [父{i}] Hello')

            print(f'\t\tfork {i} finish , {os.getpid()}')

    print("\tparent() ready", os.getpid())

    parent()

    print("\tparent() finish", os.getpid())

def testForExit():

    def parent():

        for i in range(2):

            print(f'\t\tfork {i} ready , {os.getpid()}')

            pid = os.fork()

            if pid == 0:

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [子{i}] Hello')

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [子{i}] exit ready')

                os.\_exit(0)

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [子{i}] exit finish')

            else:

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [父{i}] Hello')

            print(f'\t\tfork {i} finish , {os.getpid()}')

    print("\tparent() ready", os.getpid())

    parent()

    print("\tparent() finish", os.getpid())

def testForWait():

    def parent():

        for i in range(2):

            print(f'\t\tfork {i} ready , {os.getpid()}')

            newpid = os.fork()

            if newpid == 0:

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [子{i}] Hello')

            else:

                os.wait()

                print(f'\t\t\t{os.getppid()}:{os.getpid()} [父{i}] Hello')

            print(f'\t\tfork {i} finish , {os.getpid()}')

    print("\tparent() ready", os.getpid())

    parent()

    print("\tparent() finish", os.getpid())

print("\n\*\*\*\*\*\*\*\*\*菜单\*\*\*\*\*\*\*\*")

print("1. fork")

print("2. exec")

print("3. exit")

print("4. wait")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

option = eval(input("输入选项："))

if option == 1:

    print(f'testForFork() ready', os.getpid())

    testForFork()

    print(f'testForFork() finish', os.getpid())

    #time.sleep(5)

elif option == 2:

    print(f'testForExec() ready', os.getpid())

    testForExec()

    print(f'testForExec() finish', os.getpid())

    #time.sleep(5)

elif option == 3:

    print(f'testForExit() ready', os.getpid())

    testForExit()

    print(f'testForExit() finish', os.getpid())

    #time.sleep(5)

elif option == 4:

    print(f'testForWait() ready', os.getpid())

    testForWait()

    print(f'testForWait() finish', os.getpid())

    #time.sleep(5)

else:

    print("没有该选项，请重试")

print("主程序 finish", os.getpid())

time.sleep(1)

## 运行结果截图

### fork()

每一次os.fork()会分出一个子进程。在子进程中返回0，在父进程中返回子进程的进程号。代码运行截图如下所示。

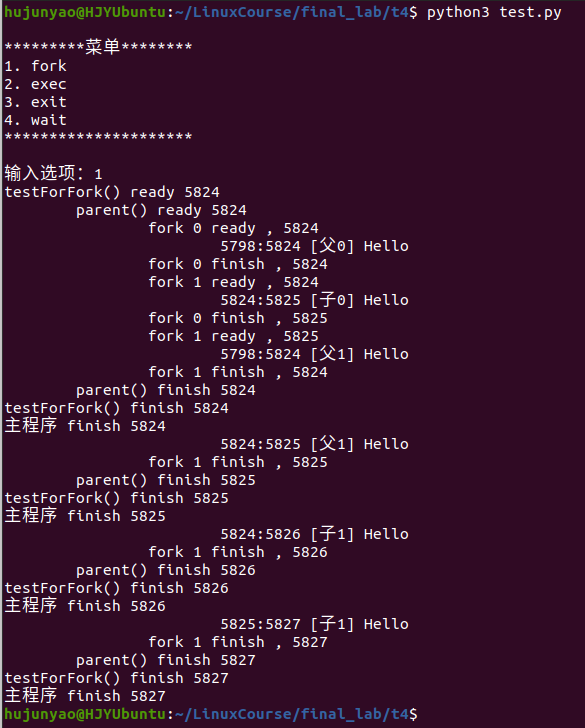


图 4.3‑1 os.fork()函数功能测试

通过绘制下图进行分析，这个测试文件进行了两次循环，第一次遇到分支函数，分成两个进程，两个进程分别进入第二次循环，遇到分支函数，一共得到四个进程，也就能看到主程序被结束了四次，第一次循环父子都输出了一次，而第二次循环，父子都输出了两次，这与分支函数的作用和预期一致，再观察进程和子进程的id号也同样能说明分支函数的作用。下图为os.fork()测试程序结果分析示意图。

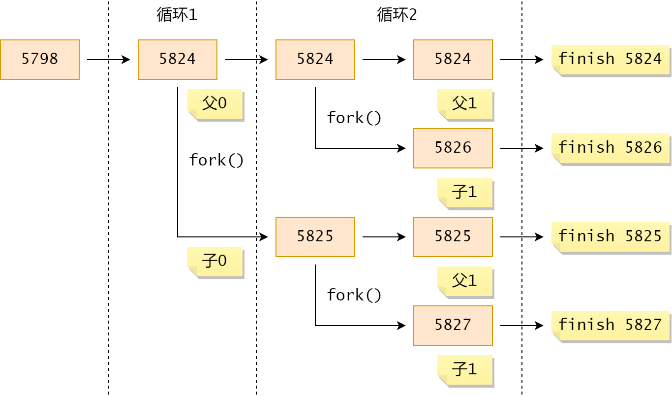


图 4.3‑2 os.fork()测试程序结果分析示意图

### exec()

事实上这里并没有一个函数叫做exec()，而是由一系列函数构成的函数组，例如os.execl、os.execle、os.execlp、os.execvpe等等。这些函数都将执行一个新程序，以替换当前进程。它们没有返回值。新程序会加载到当前进程中，且进程号与调用者相同。

exec\*函数的l和v变体不同在于命令行参数的传递方式。如果在编码时固定了参数数量，则l变体可能是最方便的，各参数作为execl\*()函数的附加参数传入即可。当参数数量可变时，v变体更方便，参数以列表或元组的形式作为args参数传递。结尾包含p的变体将使用环境变量来查找程序file，path必须包含正确的绝对或相对路径。

本实验选择了os.execl()进行测试，代码运行截图如下所示。

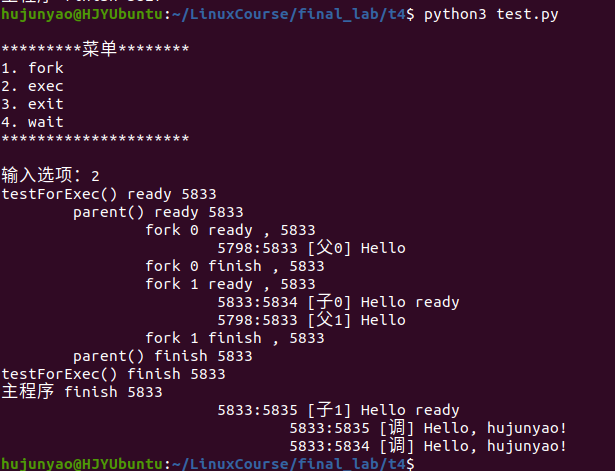


图 4.3‑3 os.execl()函数功能测试

通过绘制下图进行分析，这个测试文件进行了两次循环，第一次遇到分支函数，分成两个进程，父进程进入第二次循环，子进程遇到os.execl()，跳出循环，进行调用。父进程又遇到分支函数，分成两个进程，子进程遇到os.execl()，跳出循环，进行调用，所以，也就能看到主程序只被结束了一次，一共有三个线程。第一次循环父子都输出了一次，而第二次循环，父子都输出了两次，调用的两次也进行输出。这与os.fork()和os.execl()的作用和预期一致，再观察进程和子进程的id号也同样能说明os.execl()函数的作用。下图为os.execl()测试程序结果分析示意图。

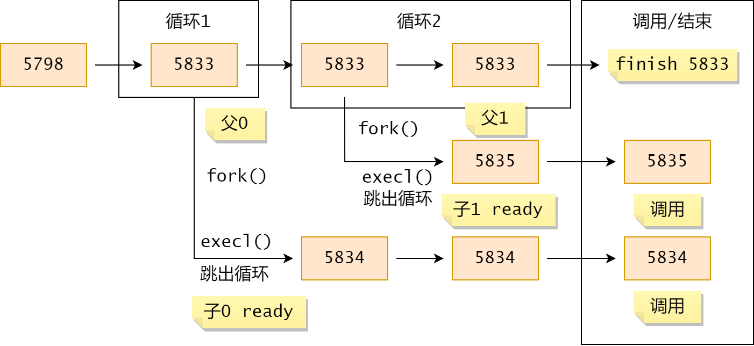


图 4.3‑4 os.execl()测试程序结果分析示意图

### exit()

os.\_exit(n)以状态码n退出进程，不会调用清理处理程序。退出的标准方法是使用sys.exit(n)。而\_exit()通常只应在fork()出的子进程中使用。代码运行截图如下所示。

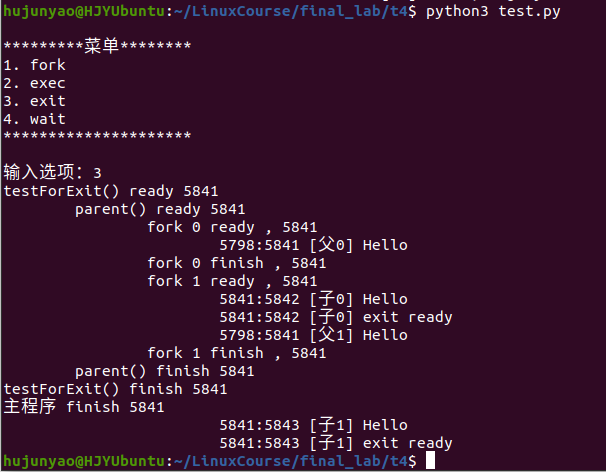


图 4.3‑5 os.\_exit()函数功能测试

通过绘制下图进行分析，这个测试文件进行了两次循环，第一次遇到分支函数，分成两个进程，两个进程继续执行，子进程遇到os.\_exit()，直接结束。父进程又遇到分支函数，分成两个进程，子进程遇到os.\_exit()，直接结束。所以，也就能看到主程序只被结束了一次，一共有三个线程。第一次循环父子都输出了一次，而第二次循环，父子都输出了两次，子进程退出的两次也进行输出，主线程退出输出一次。这与os.fork()和os.\_exit()的作用和预期一致，再观察进程和子进程的id号也同样能说明os.\_exit()函数的作用。下图为os.\_exit()测试程序结果分析示意图。

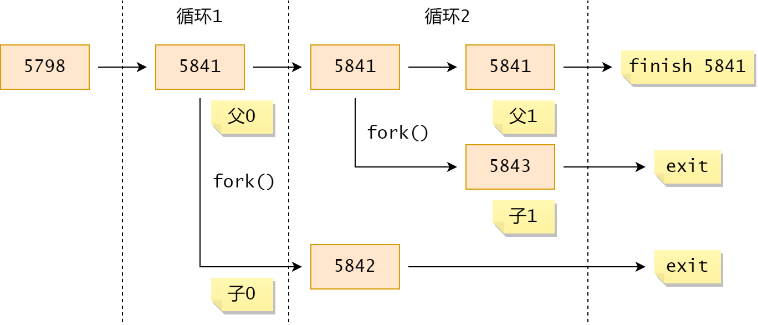


图 4.3‑6 os.\_exit()测试程序结果分析示意图

### wait()

os.wait()的作用是等待子进程执行完毕。代码运行截图如下所示。

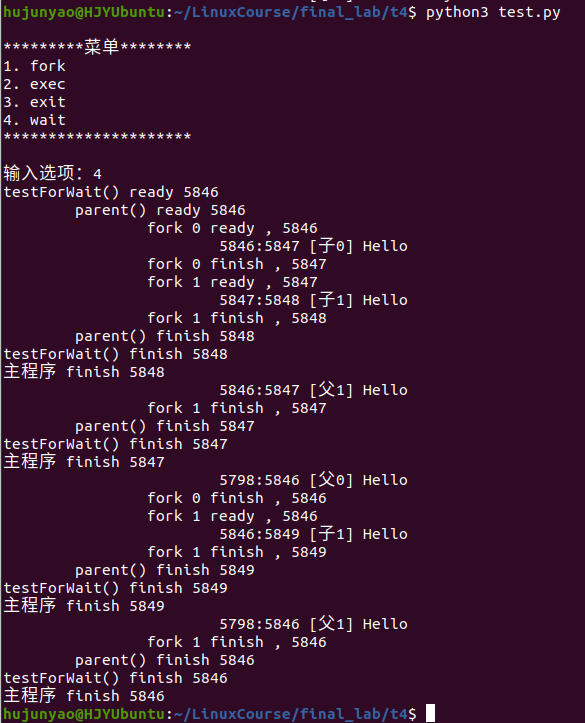


图 4.3‑7 os.wait()函数功能测试

通过绘制下图进行分析，这个测试文件进行了两次循环，第一次遇到分支函数，分成两个进程，两个进程继续执行，父进程遇到os.wait()，需要等待子进程结束再继续运行，不能直接结束。第一层父子进程又遇到分支函数，分成两个进程，父进程遇到os.wait()，需要等待子进程结束再继续运行，不能直接结束。所以，也就能看到主程序被结束了四次，一共有四个线程。第一次循环父子都输出了一次，而第二次循环，父子都输出了两次，一共四次，四个线程退出输出四次。线程结束的顺序是：层数高的先结束、同一层数子进程先结束，也就是主线程最后输出结束。这与os.fork()和os.wait()的作用和预期一致，再观察进程和子进程的id号也同样能说明os.wait()函数的作用。下图为os.wait()测试程序结果分析示意图。

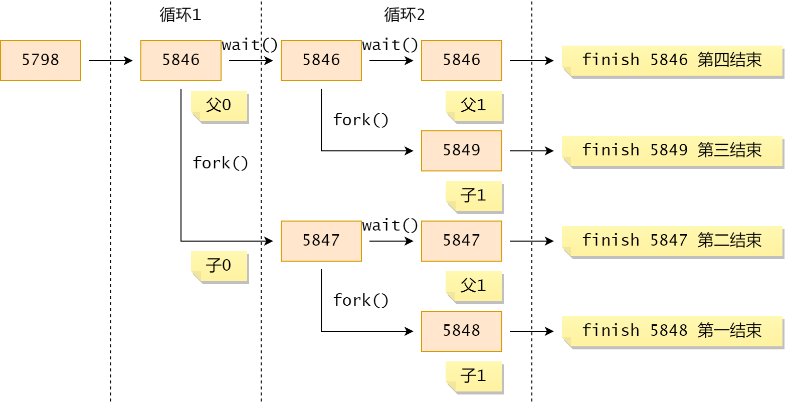


图 4.3‑8 os.wait()测试程序结果分析示意图

# 附加（20'）

题目：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（任选其一），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。

## 问题分析

本题选择监控CPU使用情况，获取CPU情况使用top命令，处理门槛设置为CPU占用大于90%，这里使用if条件语句判断，止损和保护措施是直接杀死当前占用CPU最高的程序，使用top、tail、head组合命令获取该程序的进程id，使用kill命令杀死进程，报警方式是给用户发送邮件进行提示，使用mutt命令发送给邮件，使用-a参数携带上日志附件。定时使用crontab -e命令，设置为每15分钟检测一次。

## 程序源代码

监控CPU代码*test.sh*如下。

#!/bin/bash

#提取当前IP

IP=$(ifconfig ens33 | awk 'NR==2{print $2}')

ip\_no\_sep=$(echo $IP | sed 's/\.//g')

#提取当前日期

today=$(date -d "0 day" +%Y年%m月%d日)

today\_no\_sep=$(date -d "0 day" +%Y%m%d)

filename="log\_${ip\_no\_sep}\_${today\_no\_sep}.txt"

echo "IP地址: $IP , 日期: $today " > $filename

echo "-----------------CPU----------------" >> $filename

cpu\_us=$(top -b -n 1 | grep Cpu | awk '{print $2}')

cpu\_sy=$(top -b -n 1 | grep Cpu | awk '{print $4}')

cpu\_id=$(top -b -n 1 | grep Cpu | awk '{print $8}')

cpu\_wa=$(top -b -n 1 | grep Cpu | awk '{print $10}')

cpu\_sum=$(echo 100 $cpu\_id | awk '{printf "%0.2f\n", $1 - $2}')

cpu\_sum\_int=$(echo $cpu\_sum | awk '{printf "%d\n", $1}')

cpu\_max\_pid=$(top -b -n 1 | head -8 | tail -1 | awk '{print $1}')

cpu\_max\_name=$(top -b -n 1 | head -8 | tail -1 | awk '{print $12}')

echo -e "CPU\_Sum: ${cpu\_sum}% (CPU\_Use: ${cpu\_us}%, CPU\_System: ${cpu\_sy}%)"

echo -e "CPU\_Idle: ${cpu\_id}%"

echo -e "CPU\_Wait: ${cpu\_wa}%"

echo -e "CPU\_max\_pid: ${cpu\_max\_pid}"

echo -e "CPU\_max\_name: ${cpu\_max\_name}"

echo -e "CPU\_Sum: ${cpu\_sum}% (CPU\_Use: ${cpu\_us}%, CPU\_System: ${cpu\_sy}%)" >> $filename

echo -e "CPU\_Idle: ${cpu\_id}%" >> $filename

echo -e "CPU\_Wait: ${cpu\_wa}%" >> $filename

echo -e "CPU\_max\_pid: ${cpu\_max\_pid}" >> $filename

echo -e "CPU\_max\_name: ${cpu\_max\_name}" >> $filename

if [ $cpu\_sum\_int -ge 90 ];

then

    kill -9 ${cpu\_max\_pid}

    echo "CPU 占用到达 $cpu\_sum\_int %了，请立刻检查！" | mutt "1078622540@qq.com" -s "[CPU监控]${cpu\_max\_name}对CPU占用过高" -a $filename

fi

定时执行命令如下。

crontab -e

0,15,30,45 \* \* \* \* bash /home/hujunyao/LinuxCourse/final\_lab/t5/test.sh

## 运行结果截图

在Ubuntu系统上打开火狐浏览器，连续打开多个页面，运行程序，提示当前CPU占用率达到97%，随后火狐浏览器自动关闭，QQ邮箱收到带有详细信息的报警邮件。

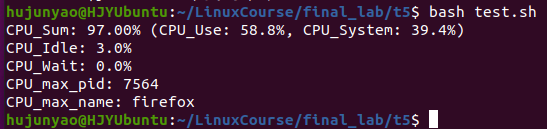
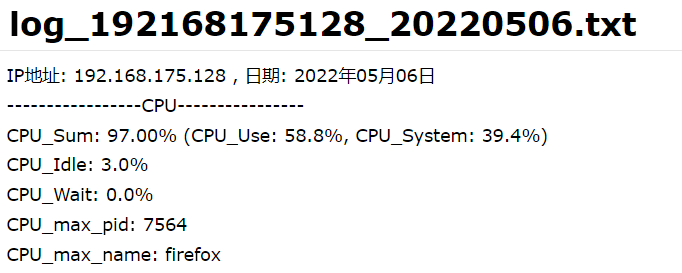
 

图 5.3‑1 日志文件输出测试

将程序设置为每15分钟执行一次，修改定时文件如下所示。

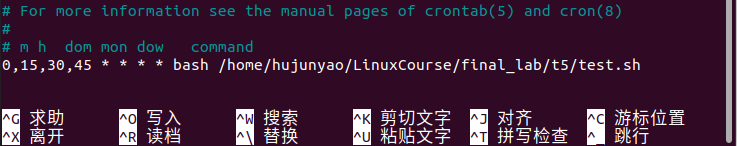


图 5.3‑2 定时执行程序设置

# 感悟

这是Linux操作系统的最后一次作业，也是结课大作业，总体来说，每个题对应了不同的章节，但是难度也是从简单到复杂，如果光听课可能还是不够，这门课程的实践性质很强，所以还是需要多写代码，才能掌握得很熟练，同时也需要自行在网上查询更多的资料去了解，多和优秀的同学交流经验，可能会有更多收获，比如添加用户的时候，adduser和useradd看起来差不多，但运行起来还是有一定差别的。

通过这次作业，我不仅巩固了对各种命令的操作和作用的理解，也加深了对操作系统、计算机组成原理等其他课程的一些认识。此外，在做题过程中，也使用了程序流程图、网络拓扑图等图形来描述程序设计的思路，也使用白盒测试的方法对做好的软件进行功能测试，将软件工程、软件测试等思想进行结合，体会到了计算机科学与技术作为一门学科的课程统一性。

# 参考文献

1. 鸟哥. 鸟哥的Linux私房菜（第四版）[M].人民邮电出版社,2019.
2. 刘遄. Linux 就该这么学[M]. 人民邮电出版社, 2017.
3. 菜鸟教程. Linux 教程[OL]. https://www.runoob.com/linux/linux-tutorial.html.