****

Linux系统及应用

实验报告

|  |  |
| --- | --- |
| 学生姓名： | 高林 |
| 指导教师： | 胡小龙 |
| 学 院： | 信息科学与工程学院 |
| 专业班级： | 计科1602 |

本科生院制

2018年12月

目录

[实验一 Linux的安装 4](#_Toc534372739)

[1、实验目的 4](#_Toc534372740)

[2、实验设备 4](#_Toc534372741)

[3、实验原理 4](#_Toc534372742)

[4、实验步骤 4](#_Toc534372743)

[实验2 Linux基本操作 8](#_Toc534372744)

[1、实验目的 8](#_Toc534372745)

[2、实验内容 8](#_Toc534372746)

[3、实验记录 8](#_Toc534372747)

[4、实验心得 19](#_Toc534372748)

[实验3 Linux系统管理 20](#_Toc534372749)

[1、实验目的 20](#_Toc534372750)

[2、实验内容 20](#_Toc534372751)

[3、实验记录 21](#_Toc534372752)

[4. 实验心得 24](#_Toc534372753)

[实验4 Linux Shell程序设计 25](#_Toc534372754)

[1、实验目的 25](#_Toc534372755)

[2、实验内容 25](#_Toc534372756)

[3、实验要求 27](#_Toc534372757)

[4. 实验心得 28](#_Toc534372758)

[实验5 Linux 高级程序设计 28](#_Toc534372759)

[1、实验目的 28](#_Toc534372760)

[2、实验内容 28](#_Toc534372761)

[3、实验要求 29](#_Toc534372762)

[4. 实验心得 32](#_Toc534372763)

[实验6 Linux内核 32](#_Toc534372764)

[1、实验目的 32](#_Toc534372765)

[2、实验内容 32](#_Toc534372766)

[3、实验要求 33](#_Toc534372767)

[4. 实验心得 44](#_Toc534372768)

# ****实验一 Linux的安装****

## 1、实验目的

* 1. 了解硬盘分区的概念和方法；
  2. 掌握硬盘的分区规划；
  3. 掌握Linux操作系统的安装和配置过程。

## 2、实验设备

一台pc机、RedHat Linux 7.2以上版本、**VMware Workstation v5.5**

## 3、实验原理

Linux可以以多种方式安装在PC机上： (1)独立分区安装、 (2)DOS分区安装和 (3)虚拟机VMWare下安装。鉴于VMware下安装对原来系统影响较小且不影响本实验目的，因此采用VMWare下安装方式。

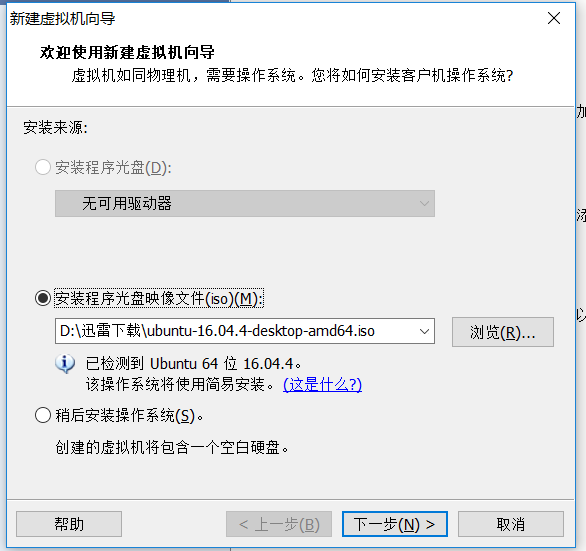
## 4、实验步骤

(1) 在Windows XP下安装VMware 5.5

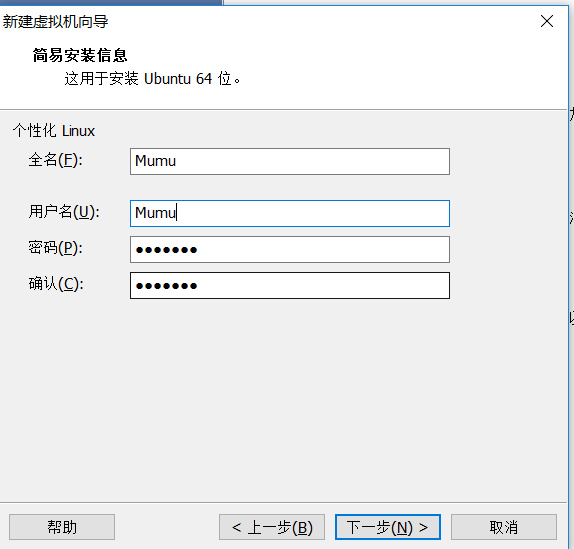
1. 配置虚拟机
2. 启动虚拟机
3. 启动Linux安装过程
4. 安装过程配置
5. 安装后配置
6. 第1次启动 VMWare下Linux操作系统

**5、实验记录**

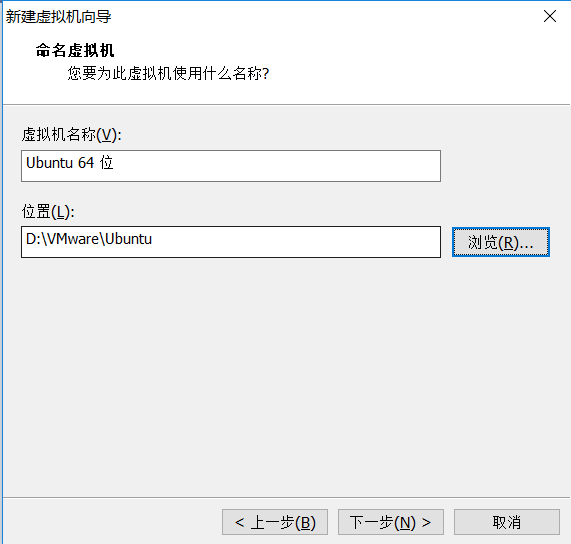
1. 安装好VM后点击新建虚拟机，使用下好的Ubuntu系统光盘映像文件，如图所示：



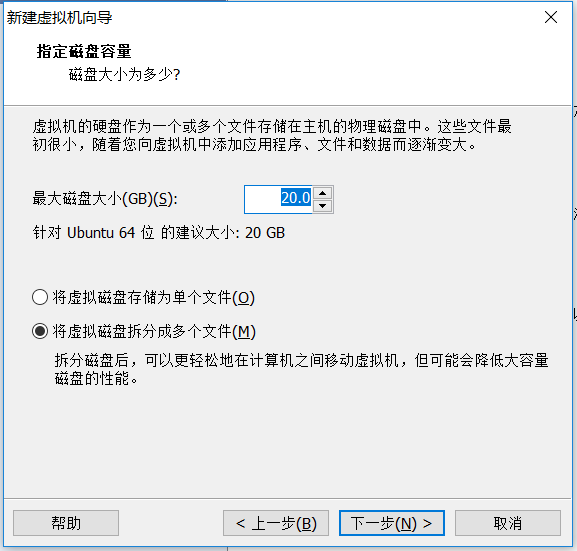
2. 点击下一步，填写虚拟机名称和登录虚拟机用户名和密码，如图所示：



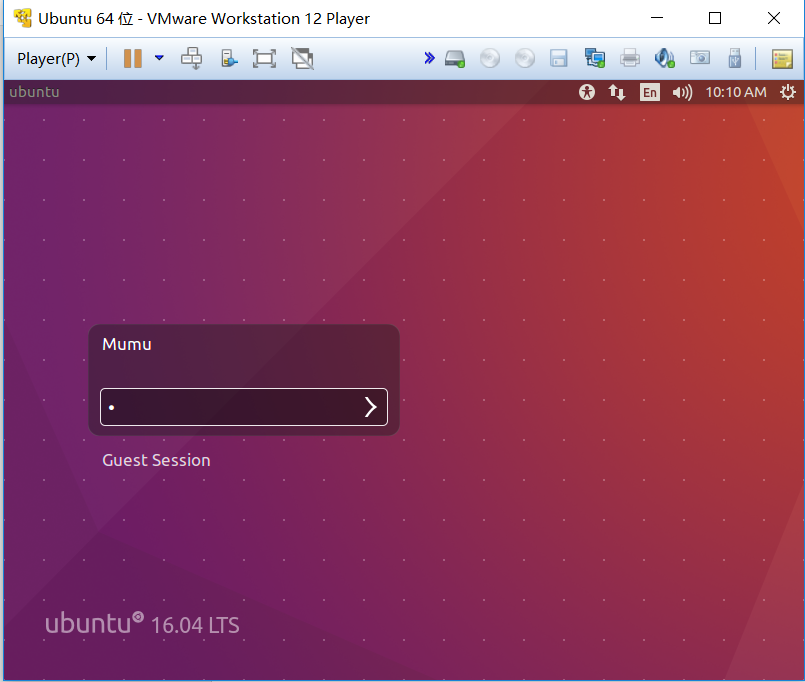
3. 选择虚拟机安装位置，如图所示：



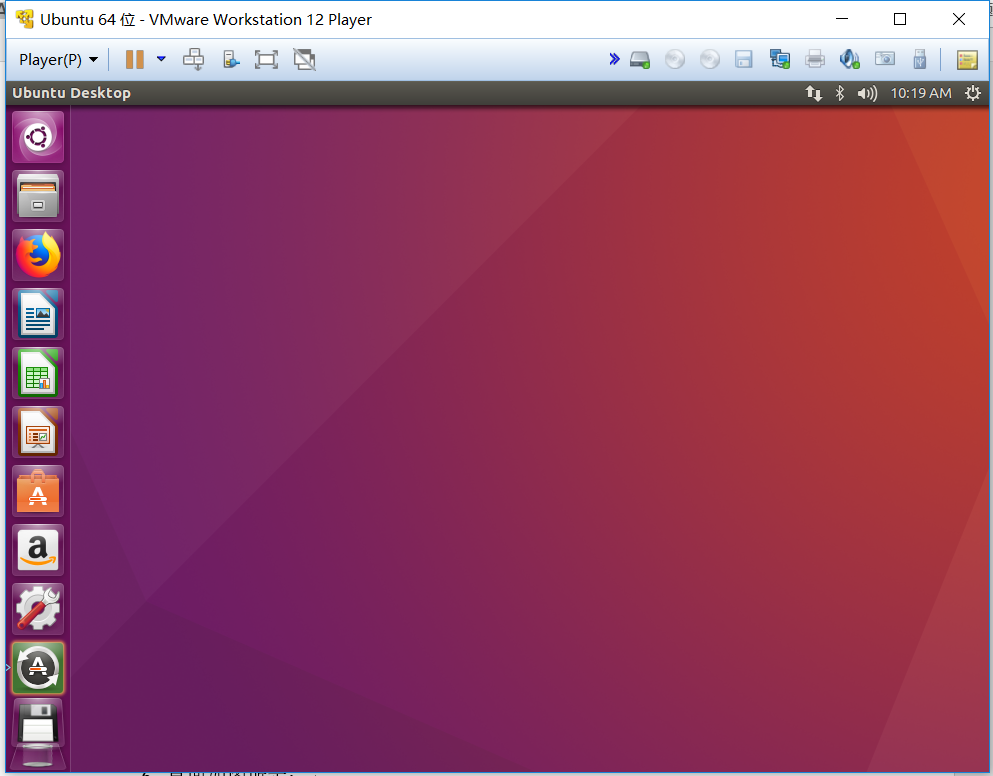
4. 为新建的虚拟机指定磁盘容量以及虚拟机文件存储方式，如图所示：



5. 安装完成后登陆界面如图所示：



6. 桌面如图所示：



# ****实验2 Linux基本操作****

## 1、实验目的

(1)复习Linux基本命令。

(2)掌握常用Linux命令

## ****2、实验内容****

(1)练习命令行模式下的常用命令：

man:命令帮助

ALT-Fx:虚终端切换

常用命令:cat、Ls、ps、chmod、kill、ln、cp、mv、rm、cd、pwd、mkdir、chown、who、w、wc、whoami、date、uname等

(2) 编辑器vi的使用：使用vi建立并修改一个文本文件

(3) 档案管理

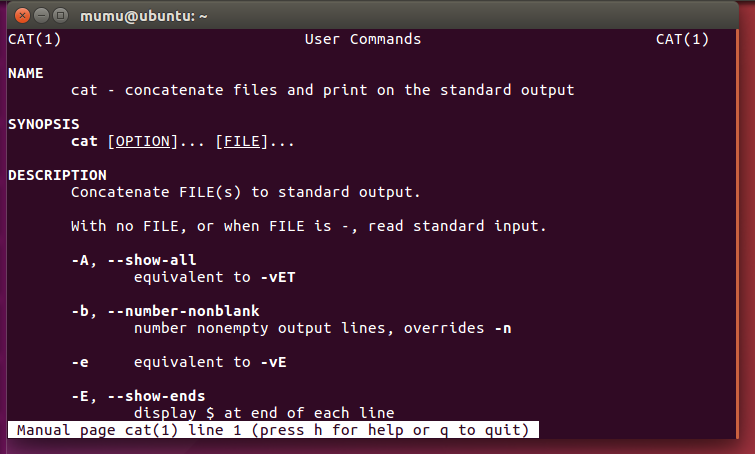
rpm、gzip、tar等命令

(4) 网络命令

telnet、ftp、talk

## 3、实验记录

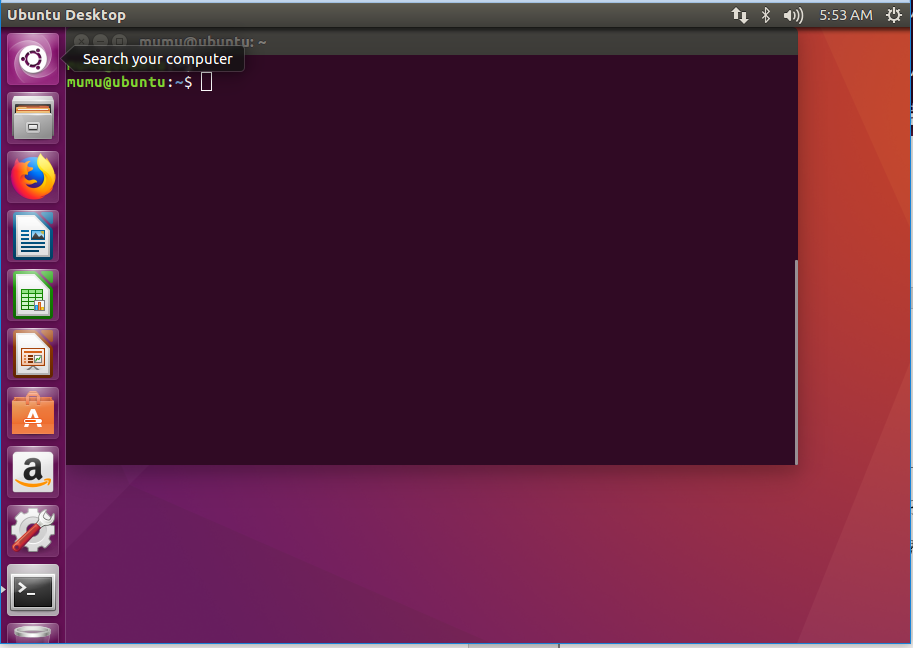
1. 输入man cat查询cat命令使用帮助，如图所示：



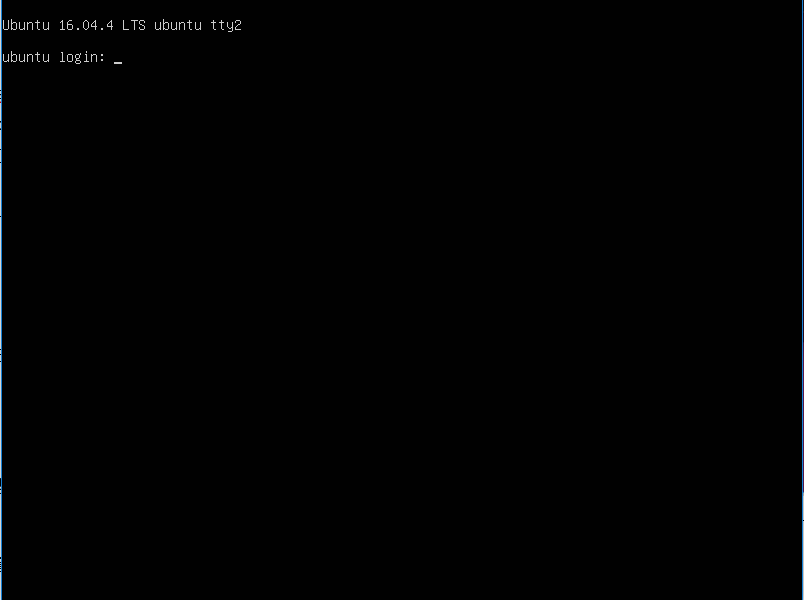
之后按q键退出查询模式。

2. 按ALT+Fx进行虚终端切换：

打开Linux虚拟机时是图形界面，如图所示：

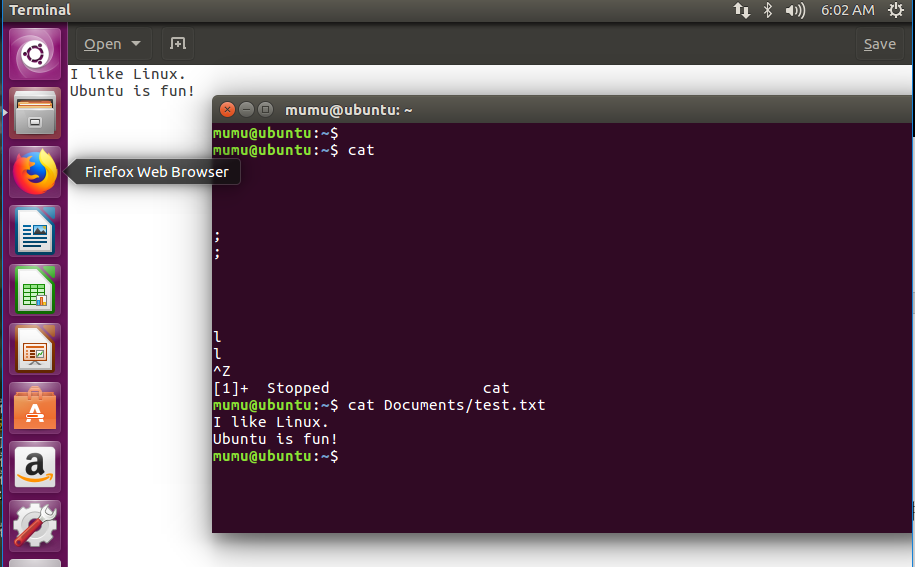


按下ALT+CTRL+F2后切换到第二个终端，如图所示：

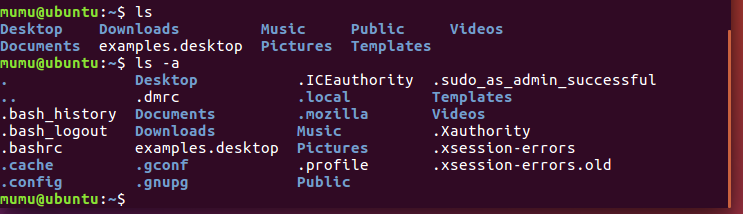


3. 常用命令的使用。

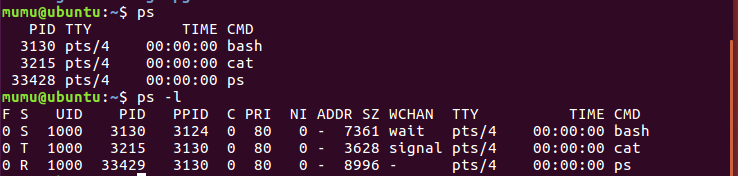
1）. Cat命令：输出文本文件内容，使用如图所示：



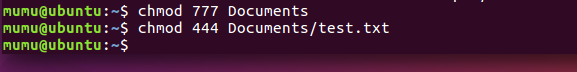
2）. Ls命令：列出目录下的文件，它后面还可以附加一些参数，如加-a是显示所有文件，包括隐藏文件。使用如图所示：



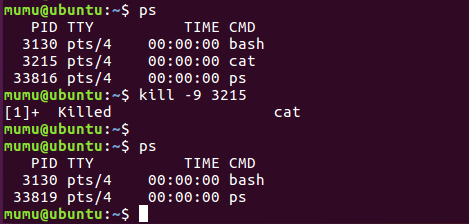
3）.ps命令：最基本进程查看命令。使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束、进程有没有僵尸、哪些进程占用了过多的资源等等.总之大部分信息都是可以通过执行该命令得到。ps是显示瞬间进程的状态，并不动态连续。使用如图所示：



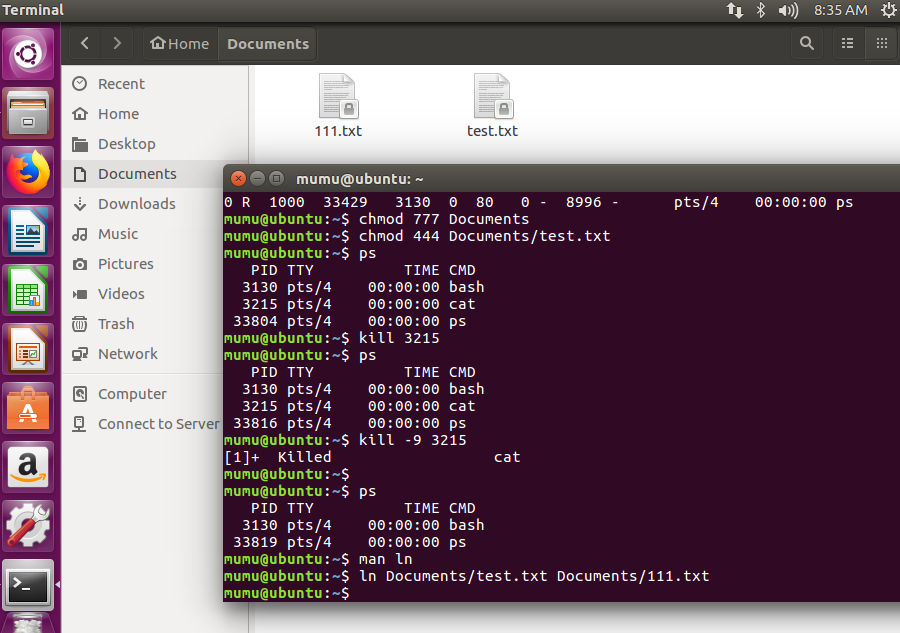
4）.chmod命令：改变一个或多个文件的存取模式。语法为：chmod abc file。其中a,b,c各为一个数字，分别表示User、Group、及Other的权限。r=4，w=2，x=1若要rwx属性则4+2+1=7；若要rw-属性则4+2=6；若要r-x属性则4+1=5。执行命令如图所示：



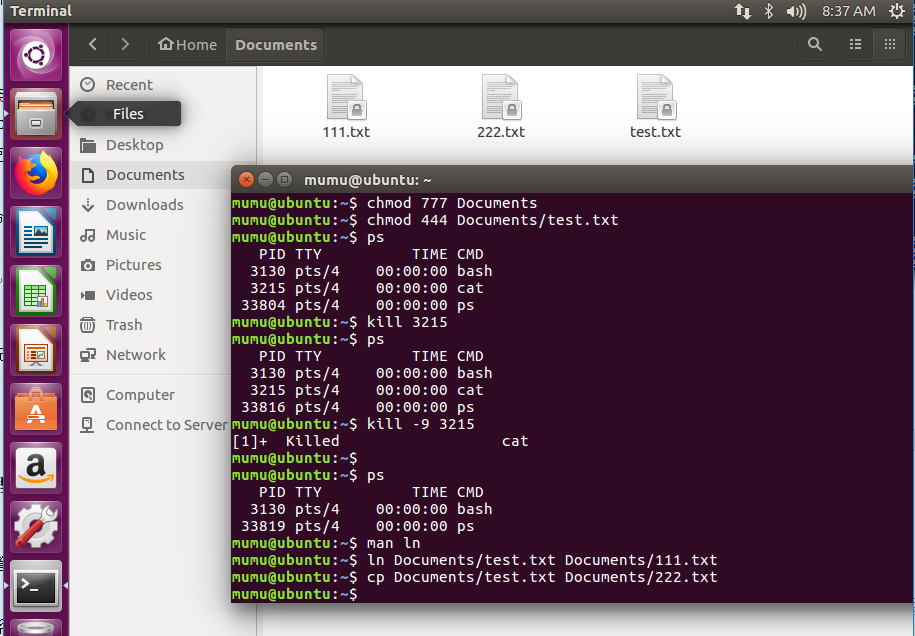
5）.kill命令：用于删除执行中的程序或工作。如图，一开始进程中有编号为3215的cat进程，之后执行命令kill -9 cat，再次查询进程后cat进程已被杀死。



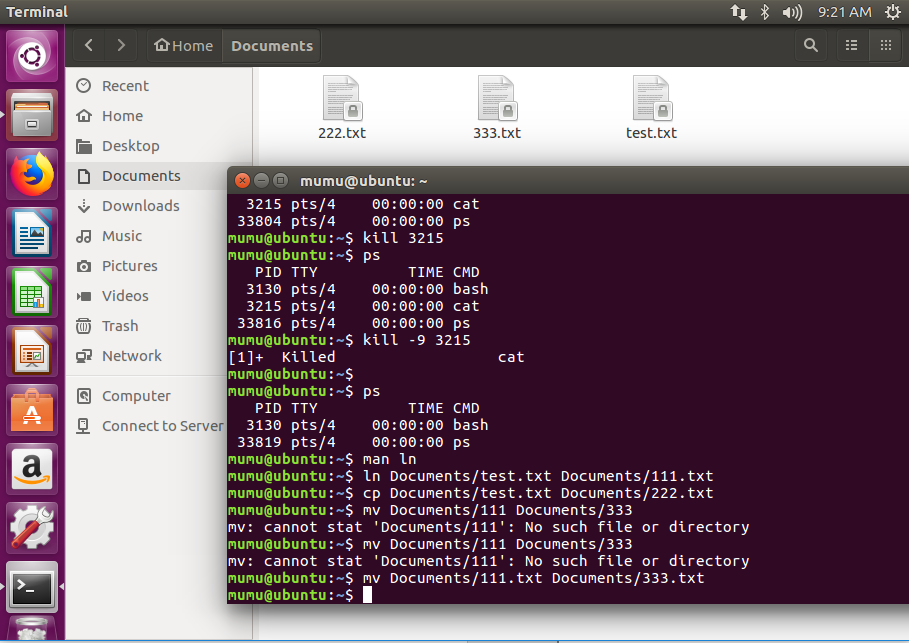
6）.ln命令：为某一个文件在另外一个位置建立一个同步的链接。之前在Documents文件夹下建立了一个test.txt文档，在执行ln Documents/test.txt 111.txt命令后，该文件夹下生成了链接文件。此时是硬连接，加上-s参数则是软连接，如图所示：



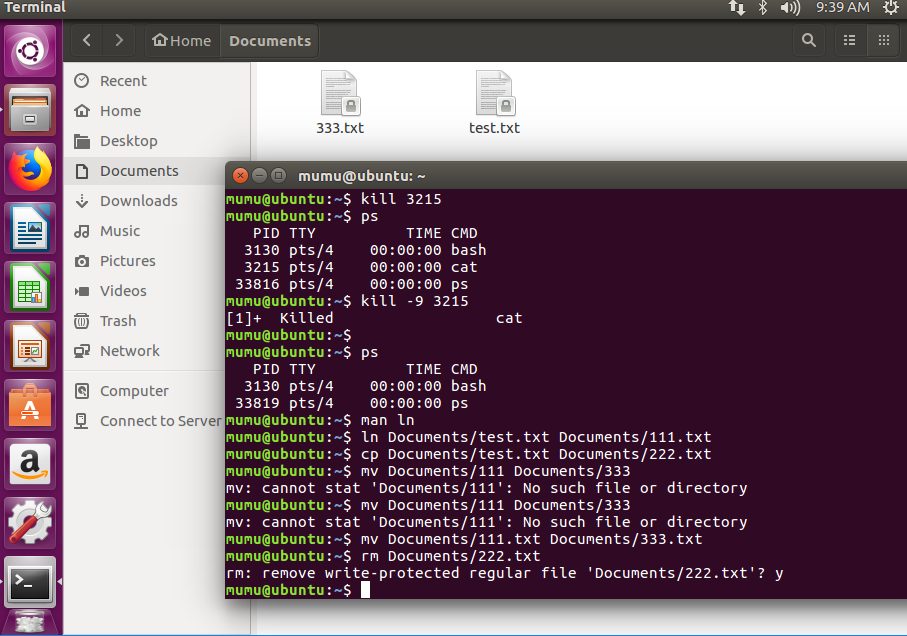
7）.cp命令：用于复制文件或目录。执行命令如图所示：



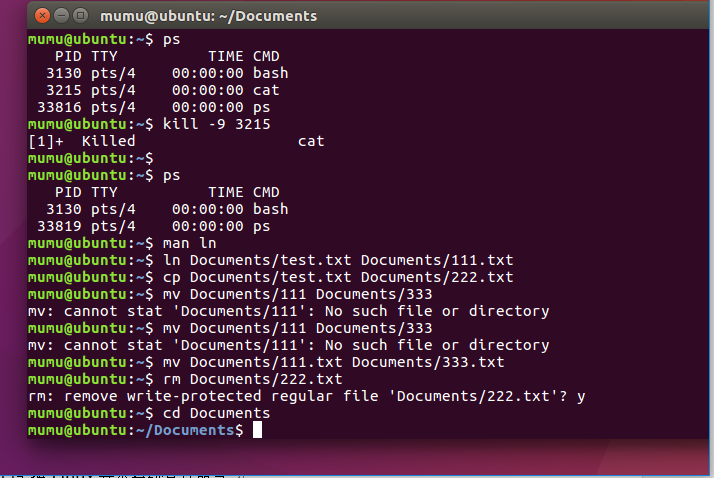
8）.mv命令：为文件或目录改名、或将文件或目录移入其它位置。更改文件操作如图所示，将111.txt改为333.txt：



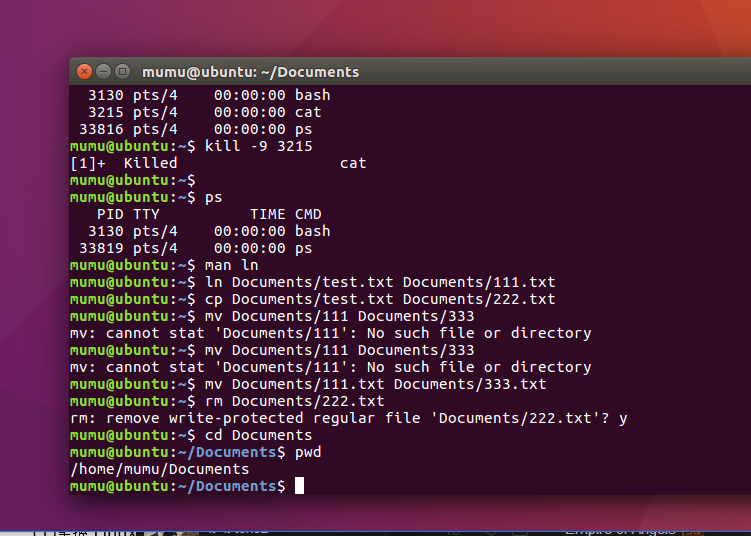
9）.rm命令：删除一个文件或者目录。执行rm Documents/222.txt命令后，会继续询问是否确定删除，输入y后删除文件，如图所示：



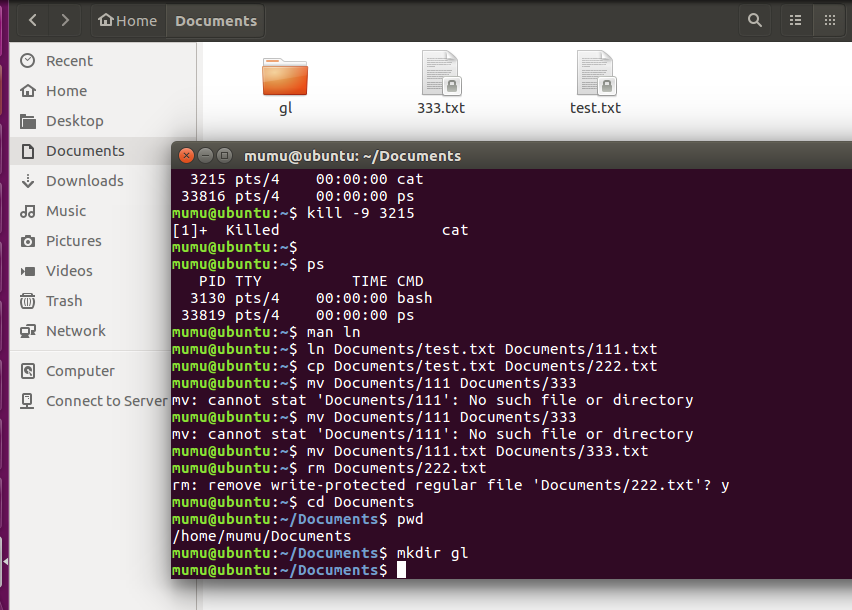
10）.cd命令：用于切换当前工作目录至 dirName(目录参数)，如图所示，进入Documents目录：



11）.pwd命令：用于显示工作目录。如图所示：

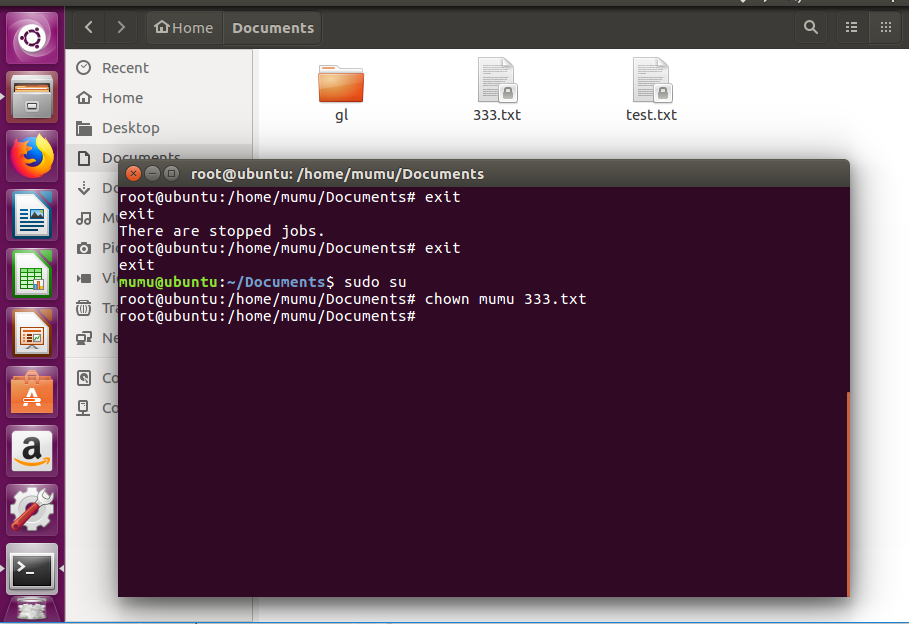


12）.mkdir命令：用于建立名称为 dirName 之子目录。如图所示，在Documents文件夹新建gl文件夹：

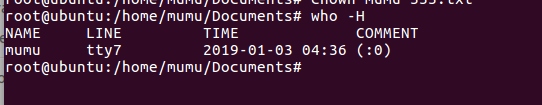


13）.chown命令：利用 chown 将指定文件的拥有者改为指定的用户或组，用户可以是用户名或者用户ID；组可以是组名或者组ID；文件是以空格分开的要改变权限的文件列表，支持通配符。

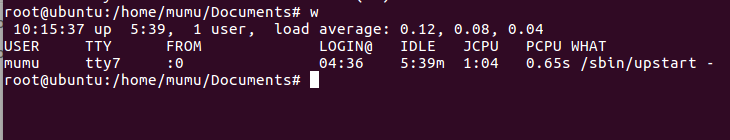
一般来说，这个指令只有是由系统管理者(root)所使用，一般使用者没有权限可以改变别人的文件拥有者，也没有权限可以自己的文件拥有者改设为别人。只有系统管理者(root)才有这样的权限。登入root账号后执行命令如图所示：



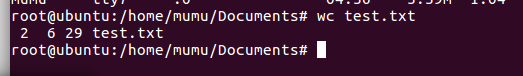
14）.who命令：用于显示系统中有哪些使用者正在上面，显示的资料包含了使用者 ID、使用的终端机、从哪边连上来的、上线时间、呆滞时间、CPU 使用量、动作等等。执行后如图所示：



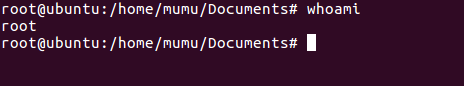
15）.w命令：用于显示目前登入系统的用户信息。执行后如图所示：



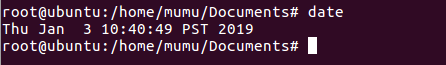
16）.wc命令：wc命令用于计算字数。计算test.txt文件字数如图所示，代表行数为2，单词数为6，字节数为29：



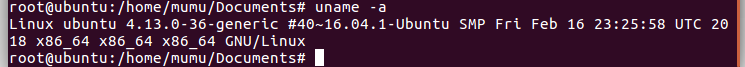
17）.whoami命令：用于显示自身用户名称。执行过程如图所示：



18）.date命令：用来显示当前系统时间。如图所示：

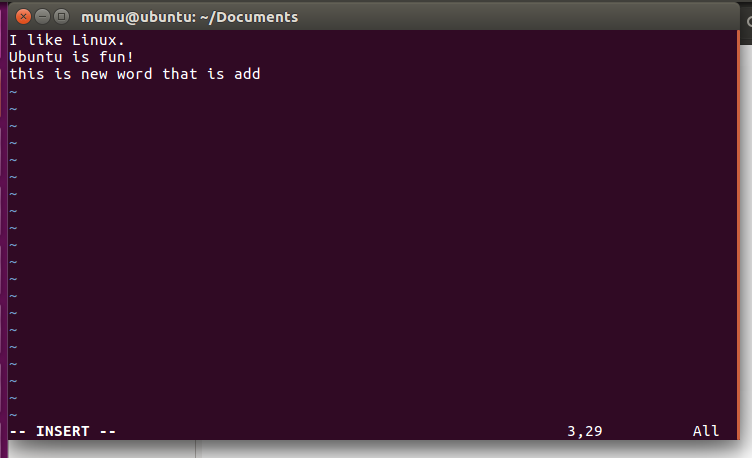


19）.uname命令：用于显示系统信息。如图所示：

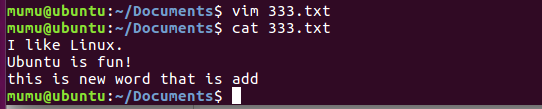


2. 编辑器vi的使用：使用vi建立并修改一个文本文件：

在终端中输入vi 333.txt，之后按i选择插入模式，并将文件更改如下：



之后按ESC键，再按：，输入wq，代表保存并退出。之后使用cat命令查看333.txt，可以看到更改成功，如图所示：



3. 档案管理：

1）.rpm命令：rpm 命令用于管理套件。

rpm(redhat package manager) 原本是 Red Hat Linux 发行版专门用来管理 Linux 各项套件的程序，由于它遵循 GPL 规则且功能强大方便，因而广受欢迎。逐渐受到其他发行版的采用。RPM 套件管理方式的出现，让 Linux 易于安装，升级，间接提升了 Linux 的适用度。

常用命令组合：

－ivh：安装显示安装进度--install--verbose--hash

－Uvh：升级软件包--Update；

－qpl： 列出RPM软件包内的文件信息[Query Package list]；

－qpi：列出RPM软件包的描述信息[Query Package install package(s)]；

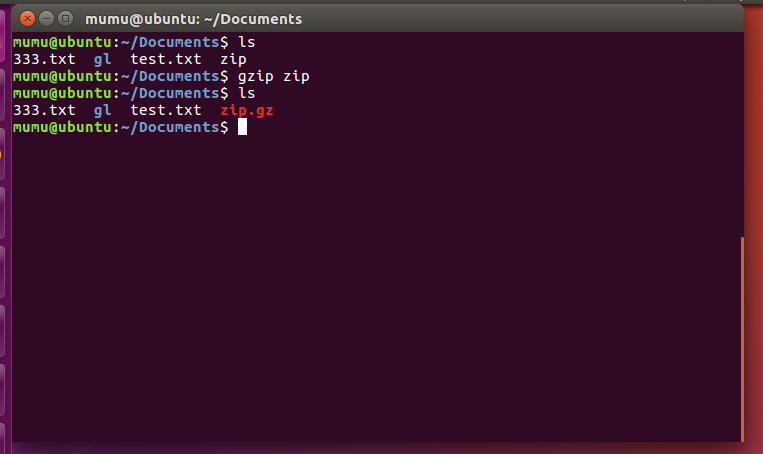
－qf：查找指定文件属于哪个RPM软件包[Query File]；

－Va：校验所有的 RPM软件包，查找丢失的文件[View Lost]；

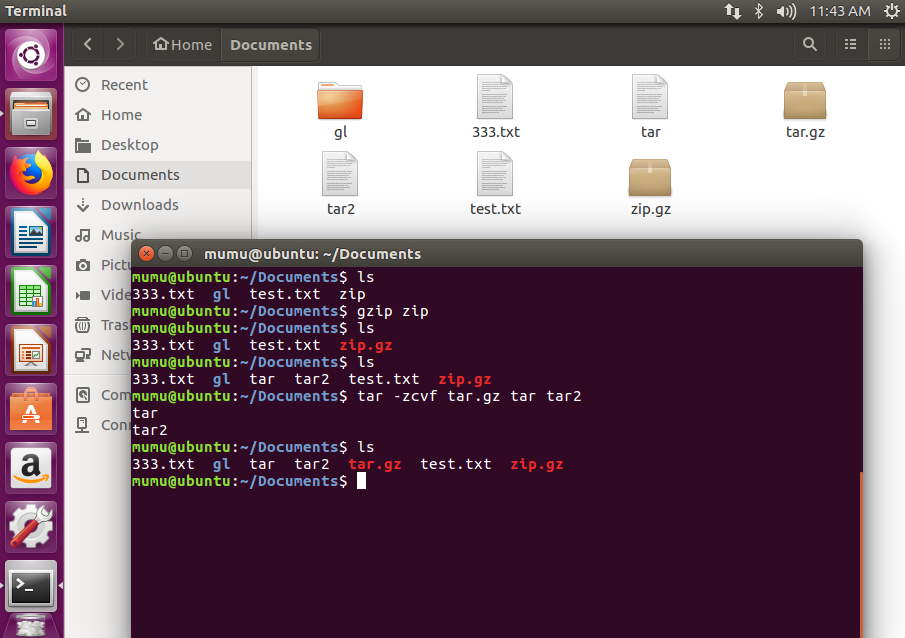
－e：删除包

Ubuntu的软件包格式是deb，如果要安装rpm的包，则要先用alien把rpm转换成deb。所以这里没有操作成功。

2）.gzip命令：用于压缩文件。首先用ls显示当前文件夹下所有的文件，然后使用gzip zip来压缩新建文件，之后再次查看文件，可以看到文件后面多了gz后缀，证明被压缩，如图所示：



3）.tar命令：用于备份文件。这里新建两个文件，tar和tar2，然后将两个文件打包成tar.gz，打包后以gzip压缩，如图所示：

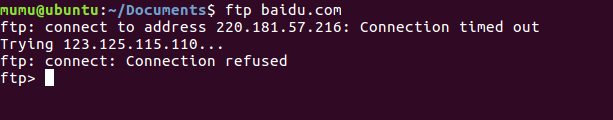


4.网络命令

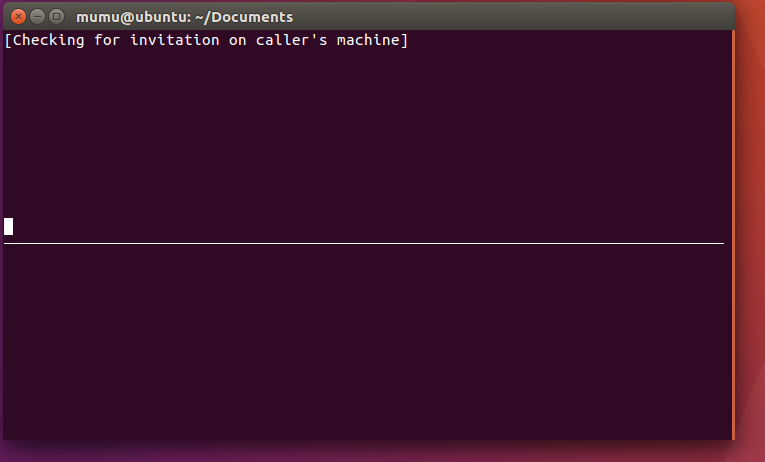
1）.telnet命令：用于远端登入。执行telnet指令开启终端机阶段作业，并登入远端主机。由于没有远程主机，所以无法连接。

2）.ftp命令：设置文件系统相关功能。FTP是ARPANet的标准文件传输协议，该网络就是现今Internet的前身。

向百度网站发起请求如图所示：



3）.talk命令：用于与其他使用者对谈。由于只有一个用户，在输入talk 用户名后界面如图所示：



## 4、实验心得

1. 在进行虚终端切换的时候，因为使用的是虚拟机以及Ubuntu系统，切换虚终端的操作有一些区别。在网络上搜索了很久之后才找到解决方案，即虚拟机是修改过热键的，正常情况下ALT+Fx的操作就要按ALT+CTRL+Fx才能实现。

2. 对于网络部分的指令理解不是很到位，因为很多都是需要进行配合，而没有相应的试验条件。

# 实验3 Linux系统管理

## ****1、实验目的****

(1)掌握Linux基本系统管理命令

(2)掌握基本管理方法

(3)初步了解Linux中与系统管理相关的有关配置文件

## 2、实验内容

(1) 用户管理

adduser、useradd、passwd等

观察：/etc/passwd文件

(2) 文件系统管理

df、du命令使用

mount、umount命令使用：安装usb盘或者光盘，查看分析/etc/fstab文件

(3) 存储器

free：了解存储器使用情况

(4) 关机

halt、shutdown –h、init 0

reboot、shutdown -r now

并比较它们之间的差别

(5) 网络配置

netconfig的使用

（6）启动配置

/etc/inittab：查看、修改并观察其响应

## 3、实验记录

1. 用户管理：

useradd命令：用于建立用户帐号。useradd可用来建立用户帐号。帐号建好之后，再用passwd设定帐号的密码．而可用userdel删除帐号。使用useradd指令所建立的帐号，实际上是保存在/etc/passwd文本文件中。

/etc/passwd是用来存储登陆用户信息的每一行代表一个用户的信息，一共包括7个字段的信息，每个字段的信息用冒号隔开。这7个字段分别代表：

1. 账号名称：即登陆时的用户名

2. 密码：早期UNIX系统的密码是放在这个文件中的，但因为这个文件的特性是所有程序都能够读取，所以，这样很容易造成数据被窃取，因此后来就将这个字段的密码数据改放到/etc/shadow中了

3. UID：用户ID，每个账号名称对应一个UID，通常UID=0表示root管理员

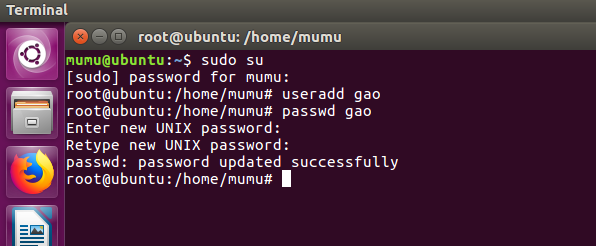
4. GID：组ID，与/etc/group有关，/etc/group与/etc/passwd差不多，是用来规范用户组信息的

5. 用户信息说明栏： 用来解释这个账号是干什么的

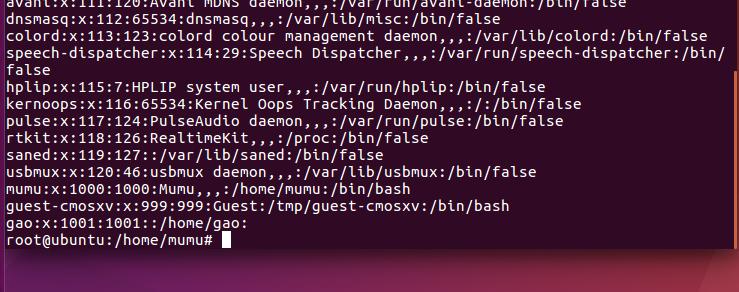
6. home目录：即用户登陆以后跳转到的目录，以root用户为例，/root是它的家目录，所以root用户登陆以后就跳转到/root目录这里

7. Shell：用户使用的shell，通常使用/bin/bash这个shell，这也就是为什么登陆Linux时默认的shell是bash的原因，就是在这里设置的，如果要想更改登陆后使用的shell，可以在这里修改。另外一个很重要的东西是有一个shell可以用来替代让账号无法登陆的命令，那就是/sbin/nologin。

执行过程如图所示：



之后查看etc/passwd文件，可以看到新建的用户的信息如图：



2.文件系统管理

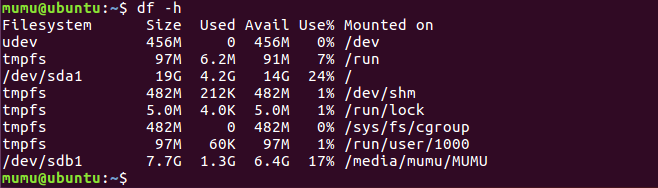
首先对U盘进行挂载：

在真机上打开服务管理，启动VMware USB Arbitration Service服务，然后在虚拟机内选择搭载U盘。这里还需要提一下/etc/fstab文件：

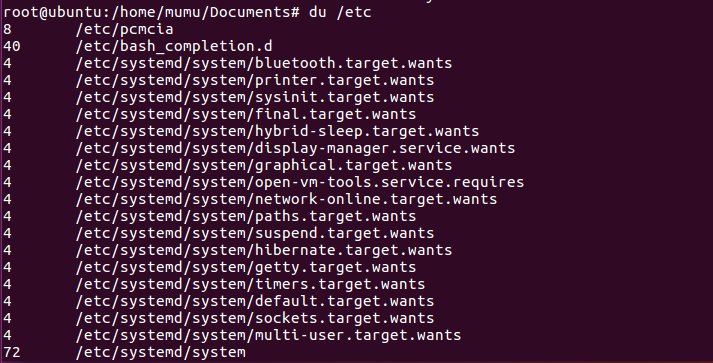
它的作用是磁盘被手动挂载之后都必须把挂载信息写入/etc/fstab这个文件中，否则下次开机启动时仍然需要重新挂载。

系统开机时会主动读取/etc/fstab这个文件中的内容，根据文件里面的配置挂载磁盘。这样我们只需要将磁盘的挂载信息写入这个文件中我们就不需要每次开机启动之后手动进行挂载了。

df命令：用于显示目前在Linux系统上的文件系统的磁盘使用情况统计。执行过程如图所示，/dev/sdb1就是我们的U盘：

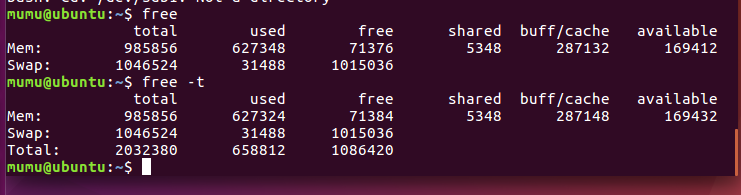


du命令：用于显示目录或文件的大小，会显示指定的目录或文件所占用的磁盘空间。执行过程如图所示：



3. 存储器：

Free命令：显示内存的使用情况，包括实体内存，虚拟的交换文件内存，共享内存区段，以及系统核心使用的缓冲区等，执行过程如图所示：



4. 关机

1）halt命令：若系统的 runlevel 为 0 或 6 ，则Linux halt命令关闭系统，否则以 shutdown 指令（加上 -h 参数）来取代。

2）shutdown -h命令：shutdown命令可以用来进行关机程序，并且在关机以前传送讯息给所有使用者正在执行的程序，shutdown 也可以用来重开机。

3）init 0命令：进行关机。

4）reboot命令：reboot命令用于用来重新启动计算机。若系统的 runlevel 为 0 或 6 ，则重新开机，否则以 shutdown 指令（加上 -r 参数）来取代。

基本上，在默认的情况下，这几个命令都会完成一样的工作（因为会先调用shutdown，而shutdown最后会调用halt）。不过，shutdown可以依据目前启动的服务来逐次关闭各服务后才关机；至于halt却能够在不理会目前系统状况下，进行硬件关机的特殊功能。

5. 网络配置

netconfig命令：用于设置网络环境。

6. 启用配置

/etc/inittab文件：init的进程号是1，从这一点就能看出，init进程是系统所有进程的起点，Linux在完成核内引导以后，就开始运行init程序，。

init程序需 要读取配置文件/etc/inittab。inittab是一个不可执行的文本文件，它有若干行指令所组成。

## 4. 实验心得

之前在查找Linux系统init流程的相关资料时总是能够看到inittab的身影，但是在我的Ubuntu上是没有这个文件的，到后来才知道采用 Upstart方式的Ubuntu上是没有inittab这个文件的。在旧式的System V initialization中，/etc/inittab可是个相当重要的文件。init进程启动后第一时间找的就是它！inittab负责初始化系统，设置系统runlevel及进 入各runlevel对应要执行的命令。假设当前inittab中设置的默认runlevle是5，则init会运行/etc/init.d/rc 5命令，该命令会依据系统服务的依赖关系遍历执行/etc/rc5.d中的脚本/程序。进入/etc/rc5.d目录可以发现里面的文件都是到/etc /init.d/下对应的脚本/程序的软链接。以S开头的为启动的意思，以K开头的为停止。并且S/K后面的两位数数字代表了服务的启动顺序（由服务依赖关系决定）。

# 实验4 Linux Shell程序设计

## ****1、实验目的****

(1)掌握Linux shell程序运行方法

(2)掌握Linux Shell程序基本语法

(3)了解Linux环境变量

(3)编写简单Linux shell程序

## 2、实验内容

1. 查看/etc/.profile文件：相当于DOS下autoexe.bat
2. 通配符“\*”、“?”、“[]”的使用

$ls [a-c]\* 和 $ls [a,m,t]\*命令

(3)重定向和管道的使用

ls | more

cat > test.txt

(4) 变量

$lookup=/usr/mydir

$echo $lookup

$export lookup：让进程使用

(5) 编辑并运行以下程序（shell程序控制结构）

1 fortest

#!/bin/bash

for a in x y z

do

echo now a=$a

done

2 fortest1

#!/bin/bash

for a

do

echo now a=$a

done

3 functest

#!/bin/bash

setup()

{

echo setup…

}

do\_date()

{

date

}

chgdir()

{

cd $1

}

do\_date

setup

chgdir

4 paramtest

#!/bin/bash

echo filename:$0

echo arguments:$\*

echo number arg:$#

echo arg2:$2

shift

echo number arg:$#

echo arg2:$2

set hello,everone

echo args:$\*

echo arg2:$2

5 untiltest

#!/bin/bash

number=0

until (test $number –gt 5)

do

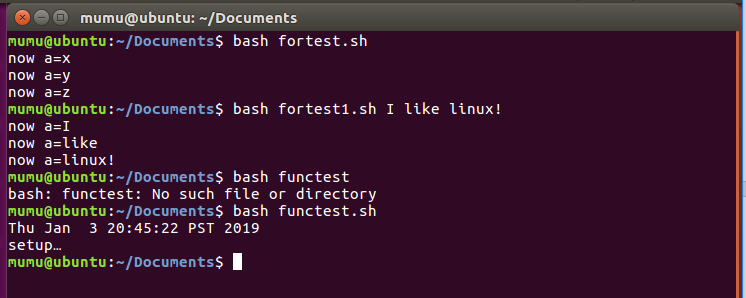
echo “ $number”

number=`expr $number + 1`

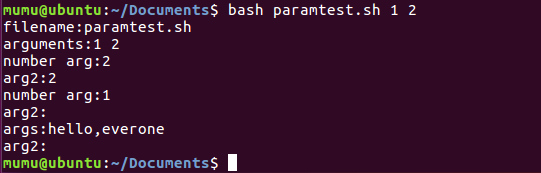
done

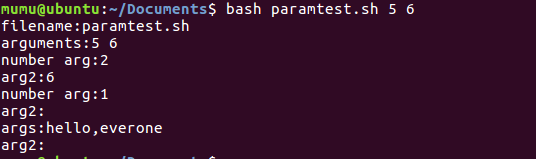
## 3、实验要求

1. 执行前三个程序截图过程如图：

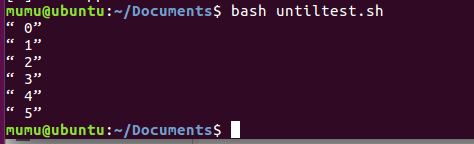


2. 执行第四个实验如图所示：



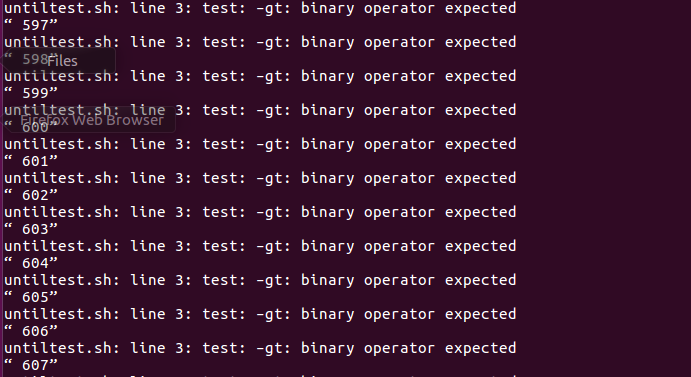


3.执行第五个实验如图所示：



## 4. 实验心得

在执行程序的过程中，出现过如下问题：



之后经过仔细排查，发现是在写入shell程序的时候，一个英文字符被打成了中文字符，将它改变之后程序就可以正常运行。

# 实验5 Linux 高级程序设计

## ****1、实验目的****

(1)了解Linux操作系统下应用程序开发流程

(2)掌握gun工具链的使用

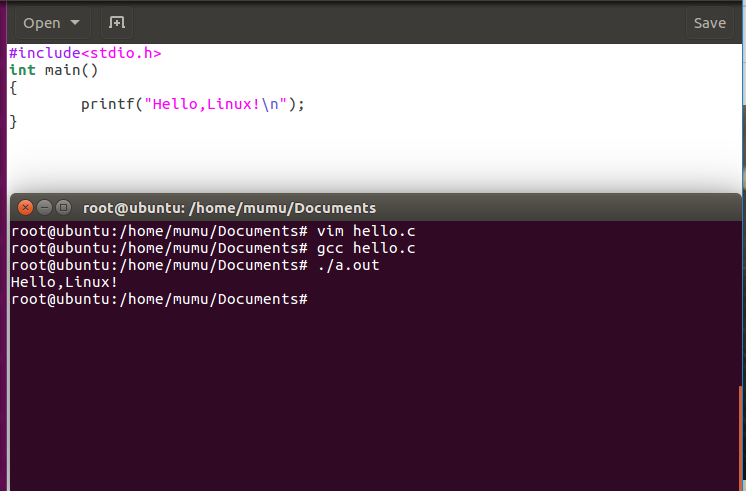
(3)了解Linux高级编程技巧（例如IPC机制、系统调用等）

## 2、实验内容

1. 编写一个简单的C语言程序，编写Makefile文件。了解编译过程，并用gdb进行调试。
2. 以下任选其一：
   1. 编写一个多进程通信程序，采用Message Queue或shared Memory或者Maped File机制进行通信
   2. 编写一个多线程程序(pthread)，实现2程同步互斥

## 3、实验要求

1. 编写简单的C语言程序并调试，程序和运行时情况如图所示：



2. 编写一个多线程程序(pthread)，实现2程同步互斥，源程序如下：

#include *<stdio.h>*

#include *<stdlib.h>*

#include *<unistd.h>*

#include *<pthread.h>*

#include *<errno.h>*

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int lock\_var = 0;

time\_t end\_time;

void pthread1(void \*arg);

void pthread2(void \*arg);

int main(int argc, char \*argv[])

{

pthread\_t id1,id2;

pthread\_t mon\_th\_id;

int ret;

end\_time = time(NULL)+10;

pthread\_mutex\_init(&mutex,NULL);

ret=pthread\_create(&id1,NULL,(void \*)pthread1, NULL);

**if**(ret!=0)

perror("pthread cread1");

ret=pthread\_create(&id2,NULL,(void \*)pthread2, NULL);

**if**(ret!=0)

perror("pthread cread2");

pthread\_join(id1,NULL);

pthread\_join(id2,NULL);

exit(0);

}

void pthread1(void \*arg)

{

int i;

**while**(time(NULL) < end\_time)

{

**if**(pthread\_mutex\_lock(&mutex)!=0)

{

perror("pthread\_mutex\_lock");

}

**else**

printf("pthread1:pthread1 lock the variable**\n**");

**for**(i=0;i<2;i++)

{

sleep(1);

lock\_var++;

}

**if**(pthread\_mutex\_unlock(&mutex)!=0)

{

perror("pthread\_mutex\_unlock");

}

**else**

printf("pthread1:pthread1 unlock the variable**\n**");

sleep(1);

}

}

void pthread2(void \*arg)

{

int nolock=0;

int ret;

**while**(time(NULL) < end\_time)

{

ret=pthread\_mutex\_trylock(&mutex);

**if**(ret==EBUSY)

printf("pthread2:the variable is locked by pthread1**\n**");

**else**

{

**if**(ret!=0)

{

perror("pthread\_mutex\_trylock");

exit(1);

}

**else**

printf("pthread2:pthread2 got lock.The variable is %d**\n**",lock\_var);

**if**(pthread\_mutex\_unlock(&mutex)!=0)

{

perror("pthread\_mutex\_unlock");

}

**else**

printf("pthread2:pthread2 unlock the variable**\n**");

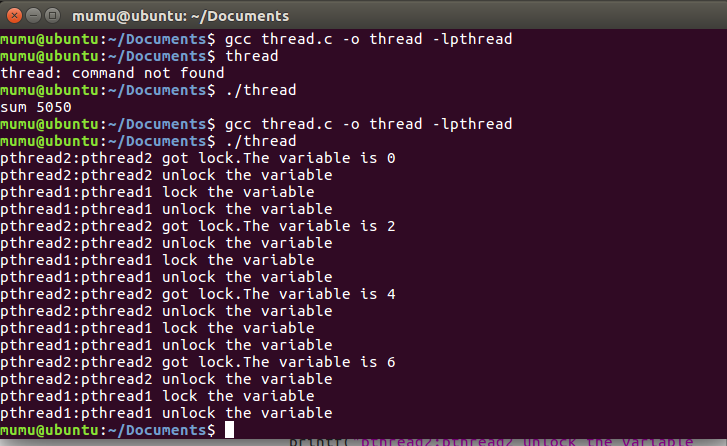
}

sleep(3);

}

}

编译执行过程如下：



## 4. 实验心得

1. 在编译完hello.c文件之后，直接输入a.out是无法执行的。上网查询之后将a.out命令换成./a.out命令，程序能够成功执行。

2. 在编译线程文件时，多次使用gcc命令编译不通过，上网查询之后发现原因是pthread 库不是 Linux 系统默认的库，连接时需要使用静态库 libpthread.a，所以在使用pthread\_create()创建线程，以及调用 pthread\_atfork()函数建立fork处理程序时，需要链接该库。

所以将gcc thread.c改为gcc thread.c -o thread -lpthread之后，程序编译成功。

# 实验6 Linux内核

## 1、实验目的

(1)了解Linux操作系内核

(2)掌握内核编译和内核升级方法

(3)了解有关内核编程

## 2、实验内容

1. 内核配置和编译（课后完成）
2. 察看/proc内存文件系统下的文件，写出每个目录内容

(3) 编写一个简单的字符虚拟设备程序（以模块方式编译、加载、使用）

## 3、实验要求

Linux 系统的开源性使其在嵌入式系统的开发中得到了越来越广泛的应用，但其本身并没有对种类繁多的硬件设备都提供现成的驱动程序，特别是由于工程应用中的灵活性，其驱动程序更是难以统一，这时就需开发一套适合于自己产品的设备驱动。对用户而言，设备驱动程序隐藏了设备的具体细节，对各种不同设备提供了一致的接口，一般来说是把设备映射为一个特殊的设备文件，用户程序可以像对其它文件一样对此设备文件进行操作。

现实世界中存在大量的设备，这些设备在电气特性和I/O方式上都各不相同。为了简化设备驱动程序员的工作，Linux系统从这些各异的设备中提取出了共性的特征，将其划分为三大类：字符设备、块设备和网络设备。内核针对每一类设备都提供了对应的驱动模型框架，包括基本的内核设施和文件系统接口。这样设备驱动程序员在写某类设备驱动程序时，就有一套完整的驱动模型框架可以使用，从而可以将大量的精力放在设备本身的操作上。下图展示了一个粗略的Linux设备驱动程序结构图：

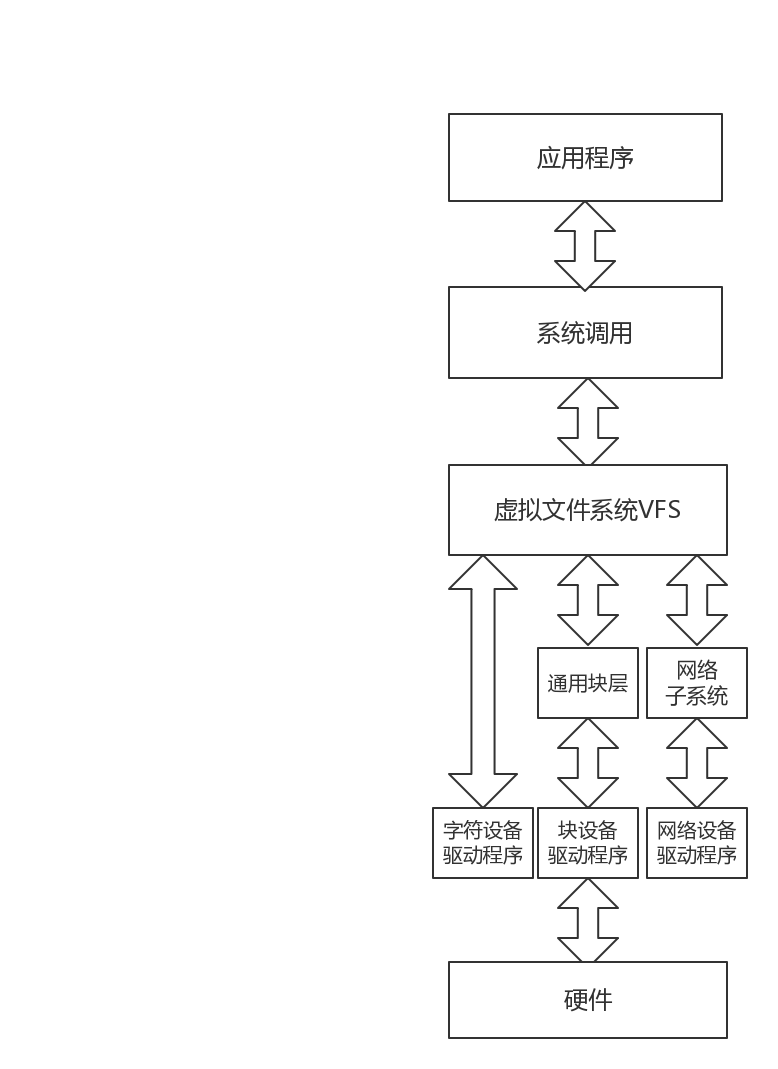


图1 Linux设备驱动程序结构图

设备驱动程序是操作系统内核和机器硬件之间的接口。设备驱动程序为应用程序屏蔽了硬件的细节，这样在应用程序看来，硬件设备只是一个设备文件，应用程序可以像操作普通文件一样对硬件设备进行操作。设备驱动程序是内核的一部分，它完成以下的功能:

1、对设备初始化和释放；

2、把数据从内核传送到硬件和从硬件读取数据；

3、读取应用程序传送给设备文件的数据和回送应用程序请求的数据；

4、检测和处理设备出现的错误。

对应三大类型的设备，也相应的有三种设备驱动程序：字符设备驱动程序、块设备驱动程序、网络设备驱动程序。其中，字符设备驱动程序是这三类设备驱动程序中最常见，也是相对比较容易理解的一种，现实中的大部分硬件都可由字符设备驱动程序来控制。这类硬件的特征是：在I/O过程中，只能一个字节一个字节进行读写操作的设备，不能随机读取设备中的某一数据、读取数据要按照先后数据。字符设备是面向流的设备，这种字符流的传输速率通常都比较缓慢（因为内核设施中不提供缓存机制），常见的字符设备有鼠标、键盘、串口、控制台和LED等。

考虑到我们现在对于Linux系统只是能够粗浅的使用，还没有进行系统的学习，所以团队讨论后决定设计相对简单的字符设备驱动程序，并在课程设计的过程中同时加强对Linux系统内核模块的理解。

字符设备提供给应用程序的是一个流控制接口，主要包括open、close（或release）、read、write、ioctl、poll和mmap等。在系统中添加一个字符设备驱动程序，实际上就是给上述操作添加对应的代码。对于字符设备和块设备,Linux内核对这些操作进行了统一的抽象，把它们定义在结构体file\_operations中。

在驱动程序中，当多个线程同时访问相同的资源时（驱动程序中的全局变量是一种典型的共享资源），可能会引发"竞态"，因此我们必须对共享资源进行并发控制。Linux内核中解决并发控制的最常用方法是自旋锁与信号量（绝大多数时候作为互斥锁使用）。

自旋锁与信号量"类似而不类"，类似说的是它们功能上的相似性，"不类"指代它们在本质和实现机理上完全不一样，不属于一类。

自旋锁不会引起调用者睡眠，如果自旋锁已经被别的执行单元保持，调用者就一直循环查看是否该自旋锁的保持者已经释放了锁，"自旋"就是"在原地打转"。而信号量则引起调用者睡眠，它把进程从运行队列上拖出去，除非获得锁。这就是它们的"不类"。

但是，无论是互斥信号量，还是自旋锁，在任何时刻，最多只能有一个保持者，即在任何时刻最多只能有一个执行单元获得锁。这就是它们的"类似"。

鉴于自旋锁与信号量的上述特点，一般而言，自旋锁适合于保持时间非常短的情况，它可以在任何上下文使用；信号量适合于保持时间较长的情况，只能在进程上下文使用。如果被保护的共享资源只在进程上下文访问，则可以以信号量来保护该共享资源，如果对共享资源的访问时间非常短，自旋锁也是好的选择。但是，如果被保护的共享资源需要在中断上下文访问（包括底半部即中断处理句柄和顶半部即软中断），就必须使用自旋锁。

阻塞操作是指，在执行设备操作时，若不能获得资源，则进程挂起直到满足可操作的条件再进行操作。非阻塞操作的进程在不能进行设备操作时，并不挂起。被挂起的进程进入sleep状态，被从调度器的运行队列移走，直到等待的条件被满足。

在Linux驱动程序中，我们可以使用等待队列（wait queue）来实现阻塞操作。wait queue很早就作为一个基本的功能单位出现在Linux内核里了，它以队列为基础数据结构，与进程调度机制紧密结合，能够用于实现核心的异步事件通知机制。等待队列可以用来同步对系统资源的访问，上节中所讲述Linux信号量在内核中也是由等待队列来实现的。

结合阻塞与非阻塞访问、poll函数可以较好地解决设备的读写，但是如果有了异步通知就更方便了。异步通知的意思是：一旦设备就绪，则主动通知应用程序，这样应用程序根本就不需要查询设备状态，这一点非常类似于硬件上"中断"地概念，比较准确的称谓是"信号驱动(SIGIO)的异步I/O"。

在本课程设计中主要实现一个简单的字符驱动设备，运用信号量和自旋锁进行相关的并发控制，并且完成对设备的打开、释放、读、写等基本操作。

整个字符设备驱动程序的模块设计如下图：

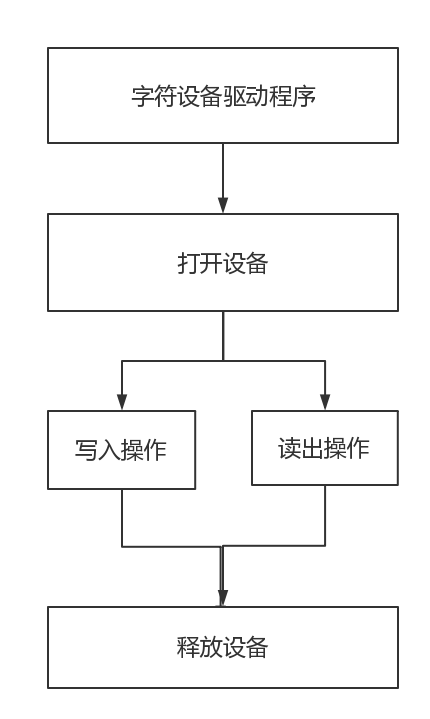


图2 字符设备驱动程序模块图

字符设备驱动内核框架的展开实际上是按照两条线进行的：一条是设备与系统的关系，一个字符设备对象首先要加入到系统中（由cdev\_map管理的哈希链表），此时设备号作为哈希索引值；另一条是设备与文件系统的关系，设备通过设备号以设备文件的形式向用户空间宣示其存在。这两条线间的联系通过文件系统接口去打开一个字符设备文件而建立：

Mknod命令将为字符设备创建一个设备节点，mknod的系统调用将会为此设备节点产生一个inode，mknod命令行中给出的设备号将被记录到inode->i\_rdev中，同时inode的i\_fop会将open成员指向chrdev\_open函数。

当用户空间open一个设备文件时，open函数通过系统进入内核空间。在内核空间，首先找到该设备节点所对应的inode，然后调用inode->i\_fop->open()，我们知道这将导致chrdev\_open函数被调用。同时，open的系统调用还将产生一个(fd,filp)二元组来标识本次的文件打开操作，这个二元组是一一对应的关系。

Chrdev\_open通过inode->i\_rdev在cdev\_map中查找inode对应的字符设备，cdev\_map中记录着所有通过cdev\_add加入系统的字符设备。

当在cdev\_map中成功查找到该字符设备时，chrdev\_open将inode->i\_cdev指向找到的字符设备对象，同时将cdev->ops赋值给filp->f\_op。

字符设备驱动程序负责实现struct file\_operations对象，在字符设备对象初始化时cdev\_init函数负责将字符设备对象cdev->ops指向该file——operations对象。

用户空间对字符设备的后续操作，比如read、write等，将通过open函数返回的fd找到对应的filp，然后调用filp->f\_op中实现的各类字符设备操作函数。

各个模块具体实现如下：

字符设备驱动主要应用了三种数据结构：①file\_operations结构，这是设备驱动程序所提供的一组用一个结构向系统进行说明的入口点；②file结构，主要用于与文件系统对应的设备驱动程序。代表一个打开的文件，它由内核在open时创建，并传递给在该文件上进行操作的所有函数，直到碰到最后的close函数。在文件的所有实例都被关闭之后，内核会释放这个数据结构；③ inode结构，提供了关于特殊设备文件/dev/mydev的信息。

各个结构的定义如下：

1. file\_operations结构

**static** **struct** file\_operations mydev\_fops = {

.read = mydev\_read,

.write = mydev\_write,

.open = mydev\_open,

.release = mydev\_release,};

1. file结构

**static** ssize\_t mydev\_read ( **struct** file \*file, char \_\_user \*buf, size\_t count, loff\_t \*pos ) { */\*从设备读取count个数据到用户数据区buf中\*/*

int size = count < MAXBUF ? **count** : MAXBUF; */\*检测读取的数据大小count是否比设备数据缓冲区大,如大则截取MAXBUF的大小\*/*

printk ( "mydev: This is my device!**\n**" );

*/\*把设备内存mydev\_buf中的数据拷贝到用户空间buf中，数量为size\*/*

**if** ( copy\_to\_user ( buf, mydev\_buf, size ) ) { *//完成用户空间到内核空间的复制，如果数据拷贝成功，则返回零；否则，返回没有拷贝成功的数据字节数。*

up ( &sem );

**return** -ENOMEM; */\*内存不足错误\*/*

}

up ( &sem );

**return** size;

}

**static** ssize\_t mydev\_write ( **struct** file \*filp, **const** char \_\_user \*buf, size\_t count, loff\_t \*pos ) { */\*把buf中count个数据写入设备内存空间中\*/*

int size = count < MAXBUF ? **count** : MAXBUF; */\*检测写入的数据大小count是否比设备数据缓冲区大\*/*

*//获得信号量*

**if** ( down\_interruptible ( &sem ) ) {

**return** - ERESTARTSYS;

}

printk ( "mydev:This is my device!**\n**" ); *//内核下的printf，按优先级输出，数字越小越高，默认为4*

memset ( mydev\_buf, 0, **sizeof** ( mydev\_buf ) ); */\*将设备内存清空\*/*

*/\*把buf中的用户数据写入到设备内存mydev\_buf中，数量为size\*/*

**if** ( copy\_from\_user ( mydev\_buf, buf, size ) ) { *//完成内核空间到用户空间的复制*

up ( &sem );

**return** -ENOMEM;

}

up ( &sem ); *//Down和up对应p、v原语*

**return** size;

}

1. inode结构

**static** int mydev\_open ( **struct** inode \*inode, **struct** file \*file ) {

*//获得自选锁*

spin\_lock ( &spin ); *//用于获得自旋锁lock，如果能够立即获得锁，它就马上返回，否则，它将自旋在那里，直到该自旋锁的保持者释放，这时，它获得锁并返回。总之，只有它获得锁才返回。*

*//临界资源访问*

**if** ( globalvar\_count ) {

spin\_unlock ( &spin );

**return** -EBUSY; *//标准错误值，设备或资源忙*

}

globalvar\_count++;

*//释放自选锁*

spin\_unlock ( &spin );

**return** 0;

}

**static** int mydev\_release ( **struct** inode \*inode, **struct** file \*file ) {

globalvar\_count--;

printk ( "设备资源释放!**\n**" );

**return** 0;

}

1. 信号量定义

**static** **struct** semaphore sem; *// 定义互斥信号量*

sema\_init ( &sem, 1 ); *//初始信号量定义为1*

down\_interruptible ( &sem ) *//P原语*

up ( &sem );*//V原语*

1. 自旋锁定义

**static** DEFINE\_SPINLOCK ( spin ); */\* 定义自旋锁 \*/*

spin\_lock ( &spin ); *//用于获得自旋锁，如果能够立即获得锁，它就马上返回，否则，它将自旋在那里，直到该自旋锁的保持者释放，这时，它获得锁并返回。总之，只有它获得锁才返回。*

spin\_unlock ( &spin ); *//释放自旋锁*

1. 用户空间与内核空间的交互

copy\_from\_user(); *//完成内核空间到用户空间的复制*

copy\_to\_user(); *//完成用户空间到内核空间的复制，如果数据拷贝成功，则返回零；否则，返回没有拷贝成功的数据字节数。*

1. 模块的初始化和退出

**static** int \_\_init mydev\_init ( void ) { */\*模块初始化\*/*

dev\_t dev; */\*设备号\*/*

int error;

error = alloc\_chrdev\_region ( &dev, 0, 2, DEVICE\_NAME ); */\*动态分配一个设备号\*/*

**if** ( error ) { */\*返回值不为0表示分配失败\*/*

printk ( "动态分配设备号失败！**\n**" );

**return** error;

}

mydev\_cdev = cdev\_alloc(); */\*新分配一个字符设备对象\*/*

**if** ( mydev\_cdev == NULL ) {

printk ( "动态分配字符设备对象失败！**\n**" );

unregister\_chrdev\_region ( dev, 2 ); */\*注销一个分配的设备号区域\*/*

**return** -ENOMEM;

}

mydev\_cdev->ops = &mydev\_fops; */\*设定字符设备操作函数指针\*/*

mydev\_cdev->owner = THIS\_MODULE; */\*设备的属主\*/*

error = cdev\_add ( mydev\_cdev, dev, 1 ); */\*将设备添加到内核中去\*/*

**if** ( error ) {

printk ( "设备添加失败！**\n**" );

unregister\_chrdev\_region ( dev, 2 ); */\*注销一个分配的设备号区域\*/*

cdev\_del ( mydev\_cdev ); */\*删除字符设备对象\*/*

**return** error;

}

memset ( mydev\_buf, 0, **sizeof** ( mydev\_buf ) ); */\*清空设备缓冲区数据\*/*

memcpy ( mydev\_buf, DEFAULT\_MSG, **sizeof** ( DEFAULT\_MSG ) ); */\*设定设备缓冲区默认数据\*/*

printk ( "设备添加成功,设备缓冲区默认数据: Hello,Welcome to OS course design!**\n**" );

sema\_init ( &sem, 1 );

**return** 0;

}

**static** void \_\_exit mydev\_exit ( void ) { */\*模块卸载\*/*

unregister\_chrdev\_region ( mydev\_cdev->dev, 2 );

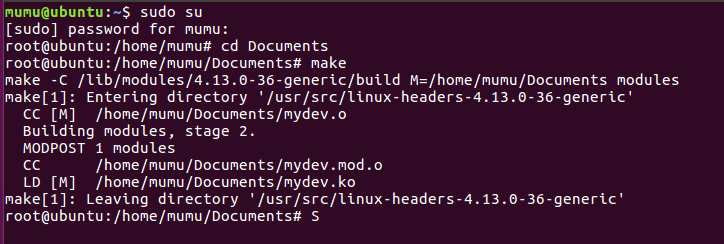
cdev\_del ( mydev\_cdev );

printk ( "设备删除成功！**\n**" );

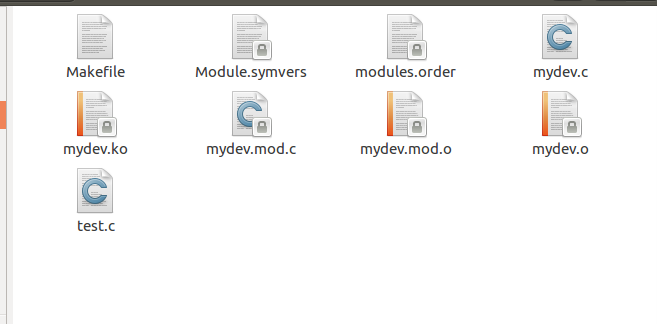
}

程序运行结果：

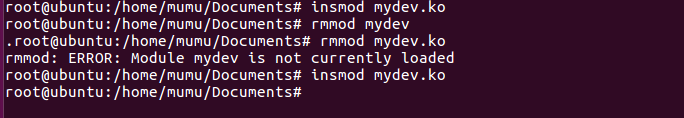
1. 用root权限进入对应文件夹对编写的字符设备驱动程序进行编译



对应文件夹生成文件



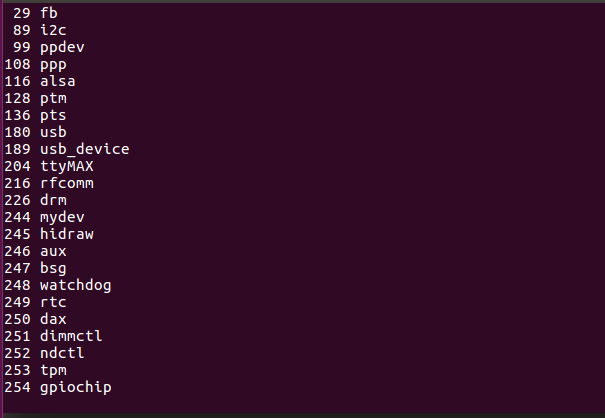
1. 将生成的驱动程序加载系统中



在后台内核输出监控中显示如下信息

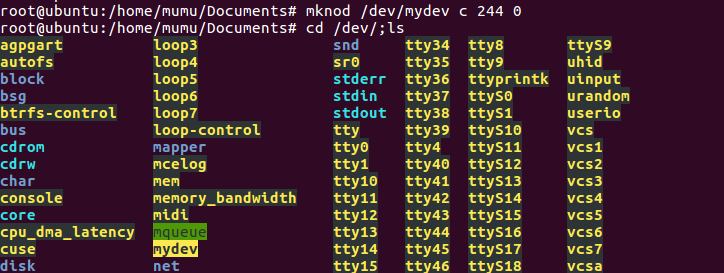


1. 显示主设，获取字符驱动设备的编号



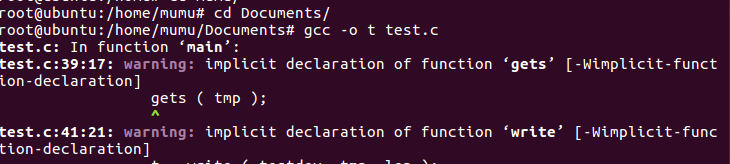
由图可知编号为244.

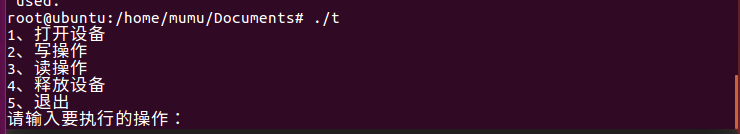
1. 由上面获得的编号为设备建立节点



可以看到已经建立好了mydev节点。

1. 使用测试文件对设备进行测试



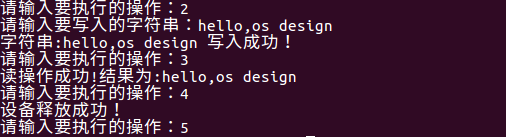


1. 对设备进行操作，检查正确性

同一时刻只能打开设备一次，在两个终端内尝试同时打开设备失败。



其他操作正常。



## 4. 实验心得

这次的课程设计提高了自己的自我学习能力和交流能力，Linux系统之前只是粗浅掌握，驱动程序的编写更是没有接触过。由于一些函数和接口已经过时，程序编译时会出现错误，需要按照最新的标准调整。

另外对程序的层次划分也不够明确。没有将不同的功能封装成一个个独立的子函数，没有实现模块化编程。这样可能会导致其他人在阅读我们的源代码时，觉得一头雾水，无从下手。以后一定会尽力完善这一点。

虽然整个过程出现了不少问题，但是这次实验还是让我收获良多。实验的目的，本就是对我们从课堂学习到的知识进行运用，并且锻炼我们的编程能力，以及要求我们进行一定量的课外自学。