《操作系统》实验指导书

光电信息与计算机工程学院计算机工程系

2020 年 9 月

目　　录

操作系统实验大纲

实验目的

实验是操作系统原理课程中不可缺少的重要教学环节，实验目的是使学生理论联系实际，使学生在实践探索中去发现问题，去解决问题，提高学生获取知识和应用技术的能力，培养学生分析问题和解决问题的能力。

操作系统原理课程要求理论与实践相结合，本门实验课程就是对理论知识的一个补充，与理论学习起着相辅相成的作用。通过本实验课程的学习，借助实例分析，使同学了解计算机系统的工作过程，资源管理策略以及并发活动的处理方法，以便为今后的课程学习和高级程序设计（如进程，线程，同步）打好基础。

实验教学的基本要求

1、进一步熟悉和掌握结构化程序设计语言编制方法。
2、进一步熟悉和掌握汇编语言程序设计方法。
3、掌握操作系统进程的调度算法。
4、掌握操作系统存储管理分区分配算法。
5、加深对 I/O 传输控制的理解。
6、学习 Linux 操作系统下的常用命令的操作。比如文件的基本操作，使用 vi 编辑器编辑文件，在 Linux 下进行 C 语言的编程和调试。
7、了解和熟悉 Linux 下的 Shell 编程。
8、学习 Linux 下进程管理命令，理解进程创建的基本原理。
9、理解 Linux 下如何编程利用信号量实现进程同步和进程通信的原理。
10、学习 Linux 内存管理命令，理解页式管理页面置换算法的程序。

实验报告要求

每个实验均应编写实验报告，学生实验后要写出严谨的，实事求是、文字通顺的实验报告。实验报告应包括以下内容：

1、实验题目

2、实验目的

3、实验内容（流程图、源程序清单并附上注释）

4、实验结果及分析

- 运行结果（必须是上面程序运行输出的结果）。
- 如果程序未能通过，则应分析其原因。
- 调试本程序的心得体会。

实验项目与学时分配

序号	实验项目	学时
1	进程调度	4
2	存储管理调度算法	3
3	资源分配算法	3
4	Linux 的安装与使用	2
5	Linux 常用命令操作	2
6	Linux 的 vi 操作	2
7	Linux 下的 C 语言编程	2
8	Linux 的 Shell 编程	2
9	Linux 的进程管理	2
10	Linux 的进程通信	2
11	Linux 的内存管理	2
合计		26

实验一　进程调度

一、 实验目的

在采用多道程序设计的系统中，有若干个进程同时处于就绪状态。当就绪进程个数大于处理机数时，就必须依照某种调度策略决定哪些进程可以分到处理机，本实验模拟在单处理机情况下的处理机调度。

二、实验内容

1、 设计一个进程控制块 PCB

包括：进程名、进程优先数、进程所占 CPU 时间、还需要 CPU 时间、进程状态、下一列指针

2、 建立进程就绪队列:要求按照不同调度算法建立就绪队列

3、 编制两个调度算法，进程数由用户从键盘输入

（1）时间片轮转法（时间片为 2）

（2）优先数算法

初始优先数 = 50 - 运行时间

每运行一次优先数减 3，时间片为 2。

4、运行结果

（提示：name 为进程名； cputime 为 CPU 已运行的时间单位数；

needtime 为进程还需要运行的时间单位数；

count 为已经运行的轮数；round 为被分配的时间片数量；

state 中 R 代表运行，W 代表等待。）

（1） **时间片轮转法**（带下划线表示输入）

输入：

input name and needtime:

a1　3

a2　2

a3　4

a4　2

a5　1

显示结果：

Name	cputime	needtime	count	round	state
a1	0	3	0	2	R
a2	0	2	0	2	W
a3	0	4	0	2	W
a4	0	2	0	2	W
a5	0	1	0	2	W

就绪队列：a2、a3、a4、a5

完成队列：

Name	cputime	needtime	count	round	state
a2	0	2	0	2	R
a3	0	4	0	2	W
a4	0	2	0	2	W
a5	0	1	0	2	W
a1	2	1	1	2	W

就绪队列：a3、a4、a5 、a1

完成队列：

Name	cputime	needtime	count	round	state
a3	0	3	0	2	R
a4	0	2	0	2	W
a5	0	4	0	2	W
a1	2	1	1	2	W
a2	2	0	1	2	F

就绪队列：a4、a5 、a1

完成队列：a2

Name	cputime	needtime	count	round	state
a1	3	0	2	2	F
a2	2	0	1	2	F
a3	4	0	2	2	F
a4	2	0	1	2	F
a5	1	0	1	2	F

就绪队列：

完成队列：a2、a4、a5、a1、a3

(2) **优先数算法**（带下划线表示输入）

输入：

input name and needtime:

a1 3

a2 2

a3 4

a4 2

a5 1

显示结果：

Name	cputime	needtime	count	pri	state
a5	0	1	0	49	R
a2	0	2	0	48	W
a4	0	2	0	48	W
a1	0	3	0	47	W
a3	0	4	0	46	W

就绪队列：a2、a4、a1、a3

完成队列：

Name	cputime	needtime	count	pri	state
a2	0	2	0	48	R
a4	0	2	0	48	W
a1	0	3	0	47	W
a3	0	4	0	46	W
a5	1	0	1	49	**F**

就绪队列：a4、a1、a3

完成队列：a5

Name	cputime	needtime	count	pri	state
a4	0	2	0	48	R
a1	0	3	0	47	W
a3	0	4	0	46	W
a2	2	0	1	48	F
a5	1	0	1	49	F

就绪队列：a1、a3

完成队列：a5、a2

Name	cputime	needtime	count	pri	state
a1	0	3	0	47	R
a3	0	4	0	46	W
a4	2	0	1	48	F
a2	2	0	1	48	F
a5	1	0	1	49	F

就绪队列：a3、a2 、a4

完成队列：a5、a2、a4

Name	cputime	needtime	count	pri	state
a5	1	0	1	49	F
a2	2	0	1	48	F
a4	2	0	1	48	F
a1	3	0	2	44	F
a3	4	0	2	43	F

就绪队列：

完成队列：a5、a2、a4、a1、a3

实验二　资源分配算法

一、 实验目的

模拟实现银行家算法，用银行家算法实现资源分配和安全性检查。通过本次实验，使学生加深对死锁概念的理解和掌握，并培养学生对操作系统课程的兴趣与高级语言设计的能力。

二、 实验内容

1、要求

设计五个进程{P0 ··P4}共享三类资源{A，B，C}的系统，每一种资源数量为 10，5，7。进程可动态的申请和释放资源，系统按各进程的申请动态地分配资源，在 T0 时刻的资源分配情况如下表：

	Max	Allocation	Need	Available
P0	7，5，3	0，1，0	7，4，3	3，3，2
P1	3，2，2	2，0，0	1，2，2	
P2	9，0，2	3，0，2	6，0，0	
P3	2，2，2	2，1，1	0，1，1	
P4	4，3，3	0，0，2	4，3，1	

(1) 银行家算法的数据结构

- Resource: 一个长度为 m 向量,表示系统拥有的资源数目。
- Available：一个长度为 m 向量，表示每类资源的可用数目。
- Max： 一个 m×n 矩阵，定义每个进程的最大资源需求数。
- Allocation：一个 m×n 矩阵，定义当前分配给每个进程每类资源的数目。
- Need：一个 m×n 矩阵，表示每个进程还需多少资源。

(2) 实现银行家算法

设：Requesti 是进程 Pi 的请求向量。

当 Pi 发出资源请求后，系统按如下步骤进行检查：

- 如果 Requesti ≤Needi 则 go to 2,否则认为出错。

- 如果 Requesti ≤ Available 则 go to 3，否则表示无足够资源， Pi 等待。
- 系统进行试探分配，并求该相应数据结构数据

  Available：= Available- Requesti

  Allocationi：= Allocationi+ Requesti

  Needi：= Needi-Requesti
- 系统执行安全性算法：安全，把资源分配给 Pi，否则， Pi 等待。

(3) 实现安全性算法

- 设 Work 和 Finish 是长度分别为 m,n 的向量

  初试值 Work：=Available ，Finishi：= False（所有）
- 从进程集合中找到一个能满足下列条件的进程

  a. Finishi= False

  b. Needi ≤ Work 如找到 go to 3 否则 go to 4
- 当进程 Pi 获得资源后，顺利执行，直至完成，并释放分配给它的资源。

  Work：= Work+ Allocationi

  Finishi：= True go to 2
- 如果所有进程的 Finishi= True 则表示系统安全，否则为不安全。

2、 运行结果

Input the type of resource and number of customer: 3 5

Input the amount of resource (maximum , allocated) of each customer: P0 7，5，3 0，1，0 P1 3，2，2 2，0，0 P2 9，0，2 3，0，2 P3 2，2，2 2，1，1 P4 4，3，3 0，0，2

```
Input  the amout of  resources(available):
3 ,3 ,2
```

可以进行菜单选择进入银行家算法和安全性检测

```
1、judge the system security
2、judge the request security
3、quit
```

输入 1

Name	Work			Need			Allocation			Work+ Allocation			Finish
P1	3	3	2	1	2	2	2	0	0	5	3	2	T
P3	5	3	2	0	1	1	2	1	1	7	4	3	T
P4	7	4	3	4	3	1	0	0	2	7	4	5	T
P2	7	4	5	6	0	0	3	0	2	10	4	7	T
P0	10	4	7	7	4	3	0	1	0	10	5	7	T

SYSTEM SECURITY!!!

输入 2

```
Please input the customer’s name and request:
p1  1  0  2
```

Name	Work			Need			Allocation			Work+ Allocation			Finish
P1	2	3	0	0	2	0	3	0	2	5	3	2	T
P3	5	3	2	0	1	1	2	1	1	7	4	3	T
P4	7	4	3	4	3	1	0	0	2	7	4	5	T
P0	7	4	5	7	4	3	0	1	0	7	5	5	T
P2	7	5	5	6	0	0	3	0	2	10	5	7	T

```
SYSTEM SECURITY!!!
CUSTOMER P1 CAN GET RESOURCES IMMEDIATELY
```

输入 2

```
Please input the customer’s name and request:
p4   3   3   0
```

```
RESOURCE INSECURITY!!!
CUSTOMER P4 CAN NOT   OBTAIN RESOURCES IMMEDIATELY
```

输入 2

```
Please input the customer’s name and request:
p0   4   4   3
```

```
SYSTEM   INSUFFICIENT!!!
CUSTOMER P0 CAN NOT   OBTAIN RESOURCES IMMEDIATELY.
```

实验三　存储管理分区分配算法

一、 实验目的

通过对操作系统内存管理算法的模拟实现，了解可变分区的动态分配和回收算法实现，掌握最佳适应法和首次适应法放置策略，加深对内存管理的理解。

二、实验内容

1、在可变分区管理模式下采用首次适应法和最佳适应法实现主存的分配和回收。

- 建立分区描述区（用空闲分区表或空闲分区链描述内存所有空闲区）
- 建立自由主存队列（针对不同的放置策略建立相应队列结构）
- 编写分区分配算法

2、具体步骤：

从未分配表中找到一个足以容纳该作业的可用空白区（未分配区）；如果这个空白区比所要求的大，则将它分成两部分：一部部分成为已经分配的分区，剩余部分仍为空白区；修改数据结构的有关信息，并回送一个所分配分区的序号或该分区的始址。

- 编写分区回收算法。（检查回收的分区是否与空白区相邻接，如有则加以合并，使之成为一个连续的空白区）

3、 运行结果(带下划线的表示输入)

(1) 最佳适应法

```
 Index    *        adr      *        end       *        size
   1                 0               32766                32767
input      the   way (best   or   first):best
Assign   or   Accept: as
input APPLACATION:   30000
SUCCESS!!!   ADDRESS=2767
```

```
Index    *    adr    *    end    *    size
  1            0          2766         2767
Assign  or  Accept: ac
Input  adr  and  size:3000  2767
```

```
Index    *    adr    *    end    *    size
  1            0          2766         2767
  2          3000         5766         2767
Assign  or  Accept : ac
Input  adr  and  size: 8000  4000
```

```
Index    *    adr    *    end    *    size
  1            0          2766         2767
  2          3000         5766         2767
  3          8000         11999        4000
Assign  or  Accept :ac
Input  adr  and  size: 2767  10
```

```
Index    *    adr    *    end    *    size
  1          3000         5766         2767
  2            0          2776         2777
  3          8000         11999        4000
Assign  or  Accept: as
input APPLACATION:  3000
SUCCESS!!!  ADDRESS=9000
```

```
Index      *      adr      *      end      *      size
  1              8000             8999             1000
  2              3000             5766             2767
  3                 0             2776             2777
Assign  or  Accept: as
input APPLACATION:  3000
Too  large  application!
```

(2) 首次适应法

```
Index      *      adr      *      end      *      size
  1                 0            32766            32767
input  the  way (best  or  first):first
Assign  or  Accept: as
input APPLACATION:  30000
SUCCESS!!!  ADDRESS=2767
```

```
Index      *      adr      *      end      *      size
  1                 0             2766             2767
Assign  or  Accept: ac
Input  adr  and  size:3000  2767
```

```
Index      *      adr      *      end      *      size
  1                 0             2766             2767
  2              3000             5766             2767
Assign  or  Accept : ac
Input  adr  and  size: 8000  4000
```

```
Index      *      adr      *      end      *      size
  1                 0                2766              2767
  2                3000              5766              2767
  3                8000             11999              4000
Assign  or  Accept :ac
Input  adr  and  size: 2767  10
```

```
Index      *      adr      *      end      *      size
  1                 0                2776              2777
  2                3000              5766              2767
  3                8000             11999              4000
Assign  or  Accept: as
input APPLACATION:  3000
SUCCESS!!!  ADDRESS=9000
```

```
Index      *      adr      *      end      *      size
  1                 0                2776              2777
  2                3000              5766              2767
  3                8000              8999              1000
Assign  or  Accept: as
input APPLACATION:  3000
Too  large  application!
```

实验四　Linux 的安装与使用

一、 实验目的

了解虚拟机软件的功能和特点，学习和动手安装linux 操作系统，了解 Linux 的目录结构。

二、预备知识

1、Linux 简介

Linux 操作系统是一个基于 POSIX 和 UNIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。它是一个类 UNIX 的操作系统，能免费使用并能自由传播。它诞生于 1991 年的 10 月 5 日，这一天 Linux0.01 版正式向外公布，后来借助于 Internet 的传播，并经过全世界各地众多编程高手的共同改进、扩充和完善，现已成为世界上使用最多的一种 UNIX 操作系统，并且使用人数还在不断增长。

Linux 内核并非是操作系统，它只是一个完整操作系统的组成部分，负责提供硬件抽象层、磁盘及文件系统控制、多任务等功能的系统软件。

Linux 内核分为稳定版和开发版。一些新特性、实验性改进等首先在开发版中进行，在开发版中测试以后，再在稳定版中进行相同的修改。一旦开发版经过足够的发展，开发版就会成为新的稳定版。两种版本相互关联，相互循环。

Linux 内核版本号格式如 major.minor.patchlevel，其中各部分说明如下。

（1）major：表示主版本号，有结构性变化时才变更。

（2）minor：表示次版本号，新增功能时才发生变化；一般奇数表示测试版，偶数表示稳定版。

（3）patch：表示对次版本的修订次数或补丁包数。

Linux 发行版指的是通常所说的“Linux 操作系统”，它是各个公司或组织推出的版本。一个典型的 Linux 发行版包括：Linux 内核，

一些 GNU 库和工具，命令行 shell，图形界面的 X 窗口系统和相应的桌面环境（如 KDE 或 GNOME），并包含数千种从办公包、编译器、文本编辑器到科学工具的应用软件。发行版一般为不同的目的而制作，包括对不同计算机结构的支持，对一个具体区域或语言的本地化，甚至针对一些实时应用和嵌入式系统。

Linux 主流的发行版本有 Ubuntu、Fedora、openSUSE、Linux Mint、CentOS、RedHat，本实验主要在 RedHat 上进行。

RedHat（www.redhat.com）是全球最大的开源技术厂家，专注于服务器版的开发，其产品 Red Hat Enterprise Linux 也是全世界应用最广泛的 Linux。RedHat 公司总部位于美国北卡罗来纳州，在全球拥有 22 个分部。RedHat Linux 因其易于安装而闻名，在很大程度上减轻了用户安装程序的负担，其中 RedHat 提供的图形界面安装方式非常类似 Windows 系统的软件安装。这对于某些 Windows 用户而言，几乎可以像安装 Windows 系统一样轻松安装 RedHat 发行套件。2004 年，Red Hat 公司正式停止对 Red Hat9.0 版本的支持，标志着 Red Hat Linux 的正式完结，而桌面版的 Red Hat Linux 发行包则与来自民间的 Fedora 计划合并，Red Hat 公司不再开发桌面版的 Linux 发行包，而将全部力量集中在服务器版的开发上，也就是 Red Hat Enterprise Linux 版。2011 年 11 月 10 日，RHEL6 正式版发布。

2、Linux 目录结构

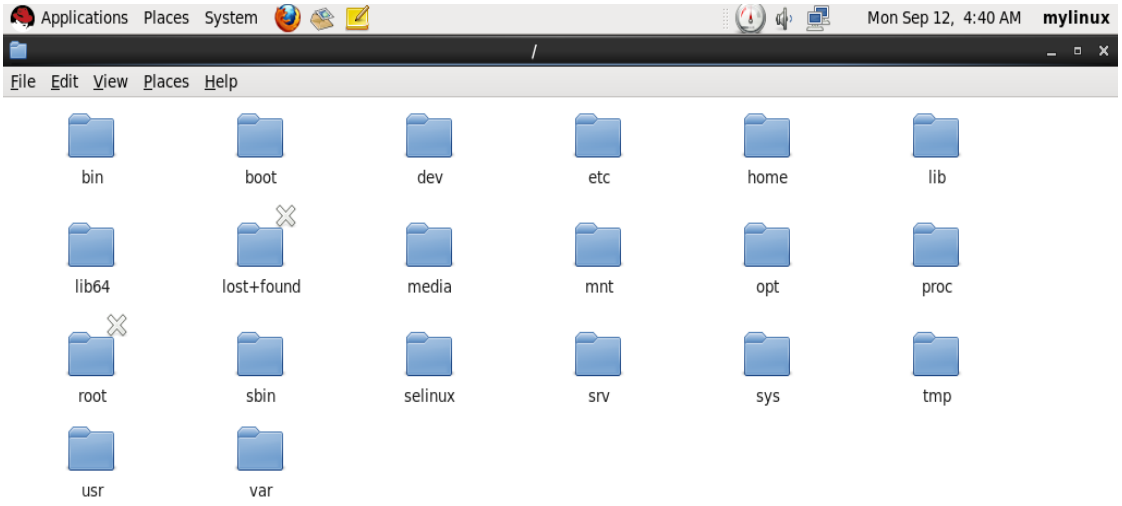


图 4-1 RHEL6 目录示意图

（1）/bin： 也称二进制（binary）目录，包含了那些供系统管理员和普通用户使用的重要的 Linux 命令的可执行文件。

（2）/sbin、/usr/sbin、/usr/root/sbin: 存放了系统管理的工具、应用软件和通用的 root 用户权限的命令，只有系统管理员可以使用这些目录下的程序。

（3）/boot: 存放系统启动时要用到的程序，包括 Linux 内核的二进制映像。

（4）/dev：设备（device）的英文缩写。在这个目录下存放了所有 Linux 系统中使用的外部设备，每个设备都是以文件的形式存在，包括键盘和鼠标，但是这里没有存放外部设备的驱动程序。

（5）/etc：Linux 系统中最重要的目录之一。这个目录下存放了系统管理时要用到的各种配置文件和子目录。

1）/etc/init.d: 用来存放系统或服务器以 System V 模式启动的脚本，还在以 System V 模式启动或初始化的系统中常见，如 Fedora/RedHat。

2）/etc/xinit.d:如果服务器是通过 xinetd 模式运行的，它的脚本要放在这个目录下。有些系统没有这个目录，如 Slackware，有些老的版本也没有。在 RedHat/Fedora 中比较新的版本中存在。

3）/etc/X11 是 X-Window 相关的配置文件存放地。

（6）/home:存放用户的主目录。

（7）/lib:用来存放系统动态连接共享库。

（8） /mnt：主要用来临时挂载文件系统的目录，系统管理员运行 mount 命令可以完成不同存储设备的挂载工作。

（9）/opt：用来安放临时测试的软件包。

（10）/proc：一个虚拟的目录，是系统内存的映射，可以通过直接访问这个目录来获取系统信息，即这个目录在内存里而不在硬盘上。

（11）/root：根用户的主目录。

（12）/tmp：用来存放不同程序执行时产生的临时文件。

（13）/usr：Linux 文件系统中最大的目录之一，系统中要用到的应用程序和文件几乎都在这里。

1）/usr/local：用来存放用户自编译安装软件的存放目录。

2）/usr/lib：是库文件的存储目录。

3）/usr/share：系统共用的文件存放目录。

4）/usr/src：内核源码存放的目录。

（14）/lost+found：一般是空的，当系统不正常关闭后，目录中就会出现没有关联的文件，这些文件可以用 Linux 工具 fsck 进行检查。

（15）/var：用来存放易变的数据，这些数据在系统运行过程中不断变化。

（16）/media：作为移动存储设备默认挂载点，如光盘。

（17）/sys：被建立在内存中的虚拟文件系统。

三、 实验内容

1、 安装 VMware 创建虚拟机

（1）运行 VMware workstation 12.0，选择左上角菜单文件-新建虚拟机，或者直接按 Ctrl+N 快捷键进入创建虚拟机向导。

（2）在弹出的窗口中选择自定义（高级）安装，点击下一步。

（3）默认，继续点击下一步，选择稍后安装操作系统，点击下一步。

（4）客户机操作系统选择 Linuux，版本选择 centos 64 位，点击下一步，命名名称和选择位置，点击下一步。

（5）默认，点击下一步。

（6）把虚拟机内存大小改为 2048MB，建议使用内存不超过最大建议使用内存，点击下一步。

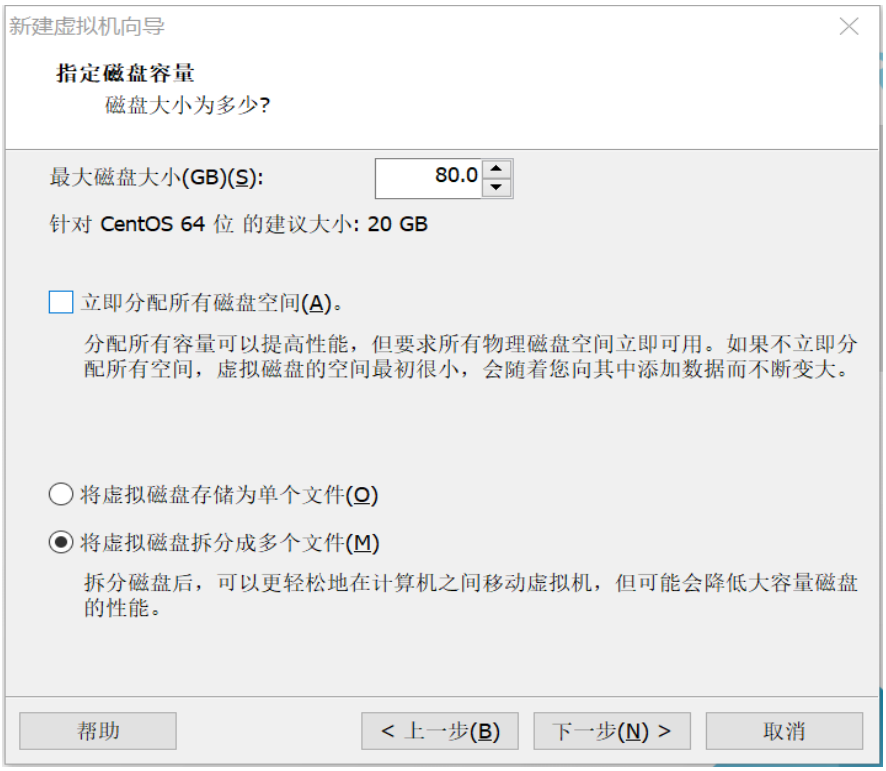
（7）选择“使用桥接网络”，点击下一步。

（8）后面两步默认选择，点击下一步。

（9）选择“创建新虚拟磁盘”，点击下一步。

（10）把最大磁盘大小改为 80GB（默认 20G 一般不够使用，建议设置略大一些），选择“将虚拟磁盘拆分为多个文件”，点击下一

步。

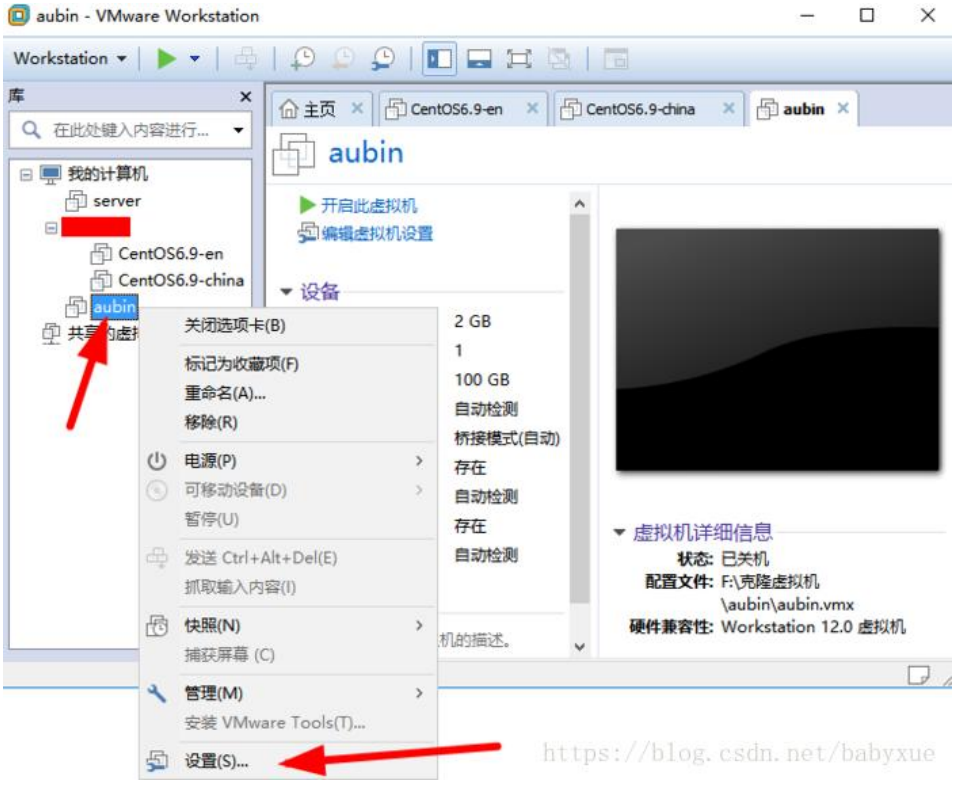


（11）默认，点击下一步

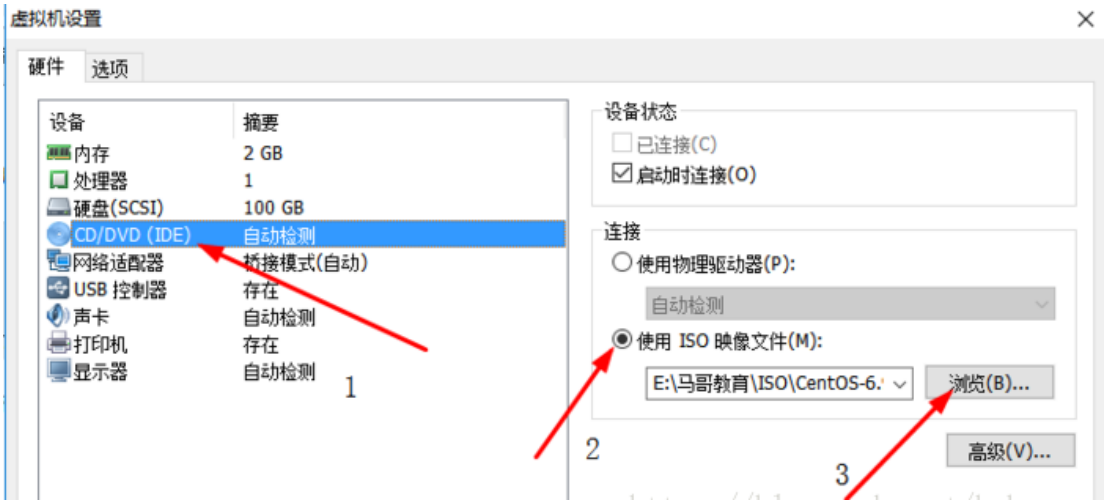
（12）默认，点击完成。

2、 安装 centos

（1） 打开步骤 1 中创建的虚拟机软件，在其上安装 centos。



右击刚创建的虚拟机，选择设置。先选择 CD/DVD，再选择使用 ISO 映像文件，最后选择下载好的镜像文件，点击确定。

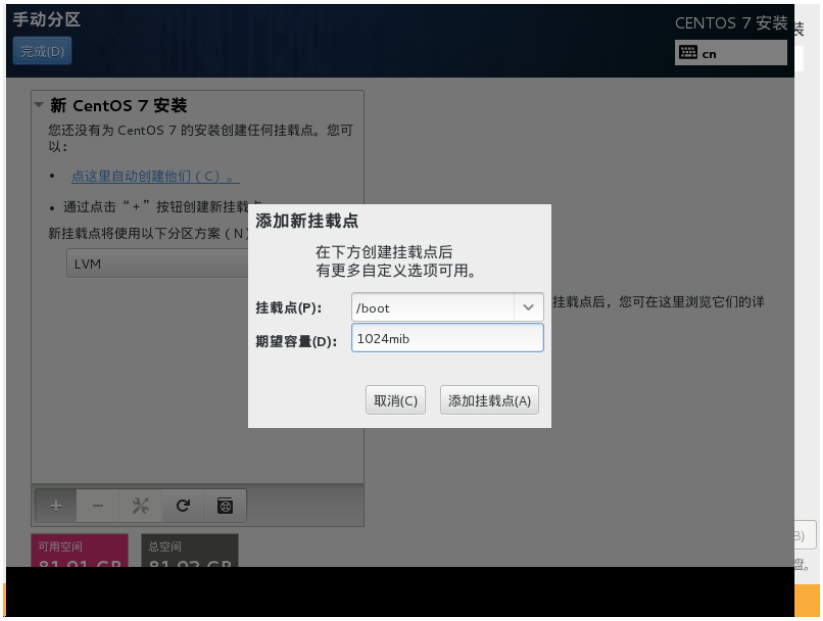


（2）开启虚拟机，选择第一项“Install CentOS 7 ”，安装直接 CentOS 7，回车，选择安装语言为简体中文，点击继续，

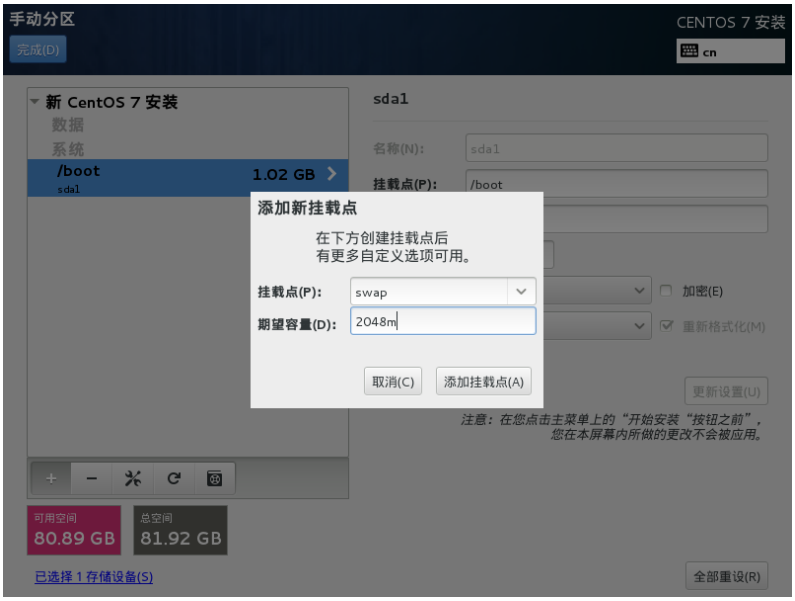
（3）设置时间，默认上海，点击“软件选择”，选择“Gnome 桌面”，点击完成

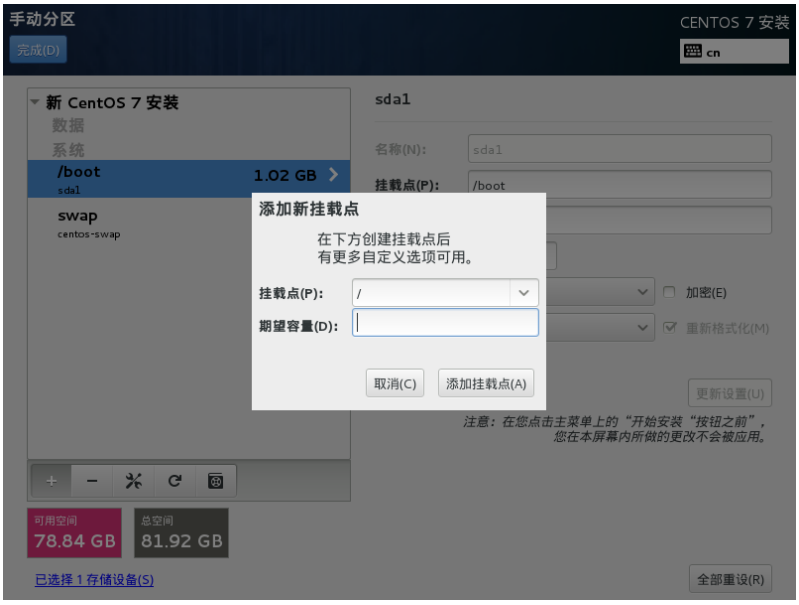
选择“安装位置”，选择“我要配置分区”，点击完成

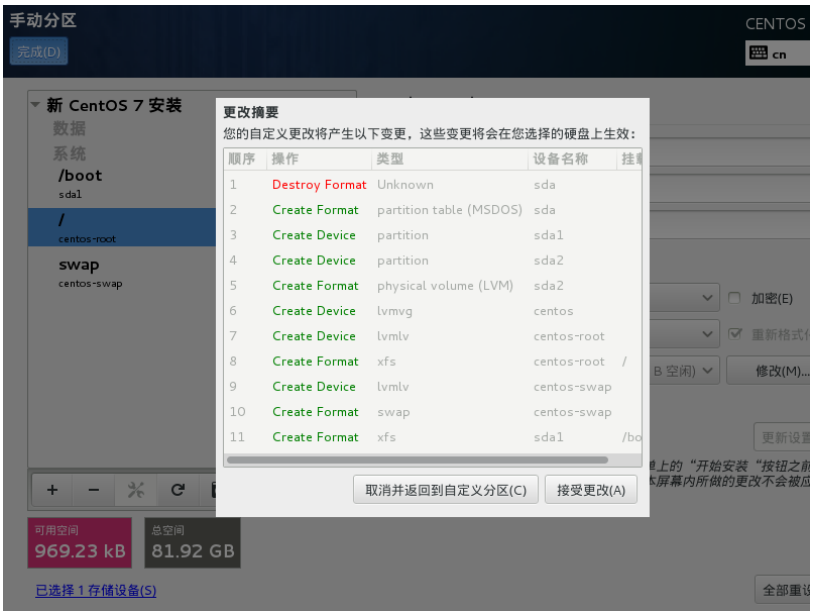
点击左下“+”号，选择/boot，给 boot 分区分 1024M。最后点击“添加挂载点”



然后以同样的办法给其他两个区分配好空间后点击完成，其中 swap 分 2048m，/分剩余所有空间（无需输入期望容量），之后再弹出窗口点击“接受更改”。


手动分区
完成(D)
CENTOS 7 安装
cn
新 CentOS 7 安装
数据
系统
/boot
sda1
1.02 GB
sda1
名称(N):
挂载点(P):
/boot
添加新挂载点
在下方创建挂载点后
有更多自定义选项可用。
挂载点(P):
swap
期望容量(D):
2048m
取消(C)
添加挂载点(A)
加密(E)
重新格式化(M)
更新设置(U)
注意：在您点击主菜单上的“开始安装“按钮之前”，
您在本屏幕内所做的更改不会被应用。
可用空间
80.89 GB
总空间
81.92 GB
已选择 1 存储设备(S)
全部重设(R)


手动分区
完成(D)
CENTOS 7 安装
cn
新 CentOS 7 安装
数据
系统
/boot
sda1
1.02 GB
swap
centos-swap
sda1
名称(N):
挂载点(P):
/boot
添加新挂载点
在下方创建挂载点后
有更多自定义选项可用。
挂载点(P):
/
期望容量(D):
取消(C)
添加挂载点(A)
加密(E)
重新格式化(M)
更新设置(U)
注意：在您点击主菜单上的“开始安装“按钮之前”，
您在本屏幕内所做的更改不会被应用。
可用空间
78.84 GB
总空间
81.92 GB
已选择 1 存储设备(S)
全部重设(R)


手动分区
完成(D)
CENTOS
cn
新 CentOS 7 安装
数据
系统
/boot
sda1
/
centos-root
swap
centos-swap
更改摘要
您的自定义更改将产生以下变更，这些变更将会在您选择的硬盘上生效：
顺序 操作 类型 设备名称
1 Destroy Format Unknown sda
2 Create Format partition table (MSDOS) sda
3 Create Device partition sda1
4 Create Device partition sda2
5 Create Format physical volume (LVM) sda2
6 Create Device lvmvg centos
7 Create Device lvmlv centos-root
8 Create Format xfs centos-root /
9 Create Device lvmlv centos-swap
10 Create Format swap centos-swap
11 Create Format xfs sda1 /bo
取消并返回到自定义分区(C)
接受更改(A)
加密(E)
修改(M)...
可用空间
969.23 kB
总空间
81.92 GB
已选择 1 存储设备(S)

（4）点击“网络和主机名”，可设置主机名，点击完成，然后点击“开始安装”。

（5）设置 root 密码，继续创建用户（如果密码过于简单，需要点击两次完成），等待系统安装完毕。

注：关于图形界面和命令行界面之间切换的问题如下：

从 Linux 的命令界面切换到图形界面可以按 Ctrl+Alt+F1 进行切换；从图形界面切换到命令界面可以按 Ctrl+Alt+F2 进行切换。

3、安装 VMware Tools for Linux

（1）在 VMware 中启动 linux 虚拟机。找菜单虚拟机/设置/添加 CD 光驱，将 C:\programfiles(X86)\VMware\linux.iso（该文件在 VMware 的安装目录下查找）选中后确定。则在 linux 桌面上出现了一个装载了 VMware Tools 的虚拟光驱(事先用 root 登录)。

（2）选中该光盘，直接打开光盘文件，选中 VMwareTools-9.6.0-1294478.tar.gz 右键选择 Extract To..（提取到），在弹出的目录列表中选择 File System（文件系统）下面的 tmp，解压到 tmp 文件下。

（3）再输入命令 cd　/tmp/vmware-tools-distrib

（4）继续输入下面命令，按 Enter 键，选择默认值安装即可：

./vmware-install.pl，之后一路回车，完成后重启。（在终端用命令安装）

在实验室安装好的 linux 系统登录时，用户名：centos

密码：centos

实验五　Linux 的常用命令操作

一、 实验目的

了解 linux 命令格式，掌握文件管理的常用命令。

二、预备知识

Linux 命令简介

命令行界面始终是 Linux 与 UINX 强大的操控基础，这与 Windows OS 的操控原理有很大的不同。Windows 在 Windows 2000 以后就彻底成为一个完全图形化的操作系统。虽然在 Linux 文件中存在 GNOME 和 KDE 桌面环境，但它们只能算作是 Linux 上的一个应用，而不可能与 Linux 内核高度整合。因此，在 Linux 的学习中，Linux 命令的学习尤为关键。而命令的执行实际上是通过 Shell 来完成的，Shell 是用户和操作系统内核之间的接口，在图形化界面下 Shell 界面一般通过“终端”来体现。

Linux 命令行的语法结构如下所示。

$commad[[-]option(s)][option argument(s)][command argument(s)]

语法结构含义如下。

（1）$：Linux 系统普通用户的命令提示符，root 根用户的命令提示符为‘#’。

（2）Command：Linux 命令的名字。

（3）[[-]option(s)]：改变命令行为的一个或多个修饰符，即选项。

（4）[option argument(s)]：选项的参数。

（5）[command argument(s)]：命令的参数。

一般来说，Linux 命令主要包括以下 9 类。

（1）文件及目录管理命令，如 ls、cd、cat、grep、touch、pwd、od 等。

（2）磁盘及设备管理命令，如 mount、umount、df、du、fsck、dd 等。

（3）进程管理命令，如 ps、top、kill、bg、fg 等。

（4）用户管理及权限管理命令，如 chmod、su、useradd、userdel 等。

（5）系统管理及维护命令，如 date、cal、shutdown、clear、echo 等。

（6）网络通信命令，如 netstat、ifconfig、ping、telnet 等。

（7）数据备份命令，如 bzip2、bunzip2、gzip、gunzip、zip、unzip、tar 等。

（8）文档操作命令，如 csplit、sort、wc 等。

（9）文件传输打印命令：传输打印命令，如 lpr、bye、ftp 等。

Linux 命令十分丰富，只有在具体的操作和使用过程中才能体会命令的功能与强大。

二、 实验内容

1、帮助命令 man、help、info

方法 1：

（1）输入命令：man□ls，屏幕显示出手册页中 ls 命令相关帮助信息的第一页，介绍 ls 命令的含义。语法结构及-a、-A、-b 和-B 等选项的意义。

（2）使用 PageDown 键、PageUp 键及上、下方向键找到-s 选项的说明信息。

（3）由此可知，ls 命令的-s 选项等同于-size 选项，以文件块为单位显示文件和目录的大小。

（4）在屏幕上的：后输入 q，退出 ls 命令的手册页帮助信息。

方法 2：

输入命令：ls□--help，□代表空格。屏幕显示中文的帮助信息，由此可知 ls 命令的-s 选项等同于--size 选项，以文件块为单位列出所有文件的大小。

方法 3：

输入命令：info□ls，屏幕显示中文的帮助信息，由此可知 ls 命令的-s 选项等同于--size 选项，以文件块为单位列出所有文件的大小。

2、显示当前目录内的内容命令 ls、dir。

以长格式显示当前目录下的内容，包括隐藏文件。

输入命令：ls□-la，屏幕显示当前目录下的各种文件的详细信息。

3、显示路径命令 pwd、改变工作目录命令 cd。□代表空格，后同。

（1）输入命令：pwd 看当前目录的绝对路径。

（2）输入命令：cd□/usr/src/kernels，查看目录下的文件。

（3）再次输入命令：pwd 看当前目录位置，pwd 命令显示的是绝对路径。

（4）默认的用户目录为～，输入命令 cd□～，回到用户的工作目录下。回到上一层目录,输入命令“cd□..”。工作目录用“.”表示，其父目录用“..”表示。

（5）Linux 提供了命令帮助输入方式，只需输入文件或目录名的前几个字符，然后按 Tab 键，如无相重的，完整的文件名立即自动在命令行出现；如有相重的，再按一下 Tab 键，系统会列出当前目录下所有以这几个字符开头的文件名。

4、目录的创建与删除命令 mkdir、rmdir。

（1）使用 pwd 命令确认回到用户主目录，在主目录中同时创建子目录 newdirl，newdir2。依次输入命令 mkdir newdir1， mkdir newdir2，或者 mkdir newdir1|mkdir newdir2,再用命令 ls 查看新目录创建是否成功。

（2）在主目录中创建 dirl/dir2/dir3 分层的目录结构，输入命令 mkdir□-p□dir1/dir2/dir3，如果把参数-p 去了以后，尝试一下会出现什么情况。

（3）在主目录中将创建的目录 newdir1、newdir2 和 dir1/dir2/dir3 全部删除，使用命令：

rmdir□newdir1

rmdir□newdir2

通过帮助（rmdir□--help）查看如何利用 rmdir 命令一次性强制删除多层目录 dir1/dir2/dir3。

5、文件的复制，移动及删除命令 cp、mv 、rm。

（1）在当前用户的主目录下使用 vi 编辑器建立文件 filel.txt、file2.txt，或者进入桌面用户的主文件夹，单击右键，新建空文档，建立文件 file1.txt、file2.txt。或使用命令 cat□>file1.txt 和 catcat□>file2.txt 新建两个文件。

（2）进入“终端”，在主目录中建立子目录 path，将文件 filel.txt 复制到目录 path 下输入下述命令：

mkdir□path　　/* 创建子目录 path* /

cp□file1.txt　path　/*将文件 file1.txt 拷贝到目录 path */

（3）将主目录中的文件 file2.txt 移动到子目录 path 并更名为 file3.txt，输入下述命令

mv□file2. txt　path/file3. txt

（4）在主目录下交互式地删除子目录 path，包含其下的所有文件和子目录，输入下面的命令：

rm□path/file3.txt

rm□path/file1.txt

rmdir□path

6、显示文件内容命令 cat、more、head、less、tail。

（1）使用 cat 命令显示文件内容。

输入命令：cat□>f1，创建文件 f1。

输入命令：cat□f1，屏幕上输入点光标闪烁。

依次输入一些文字内容。使用 cat 命令进行输入时，只能用退格键（Back Space）来删除光标前一位置的字符。并且一旦按 Enter 键，该行输入的字符就不可修改。

文字内容输入后，按 Enter 键，让光标处于输入内容的下一行，按 Ctrl+D 键结束输入。

要查看文件是否生成，输入命令 ls 即可。

输入命令：cat□f1，查看 f1 文件的内容。

再输入命令：cat>>f1，屏幕上输入点光标闪烁。

输入“Hello□World!”后，按 Enter 键，让光标处于输入内

容的下一行，按 Ct1+D 键结束输入。

输入命令：cat☑fl，查看 fl 文件的内容，会发现 fl 文件中有了刚才输入的内容。

（2）分页显示/etc 目录中所有文件和子目录的信息。

输入命令：ls/etc | more，屏幕显示出 ls 命令输出结果的第 1 页，屏幕的最后一行上还出现-More-字样。按“空格”键可查看下一页信息，按 Enter 键可查看下一行信息。其中|是管道符，主要用来将左侧 ls 的显示结果重定向到右侧作为输入。

浏览过程中按 q 键，可结束分页显示。

（3）仅显示/etc 目录中前/后 15 个文件和子目录。

输入命令：ls□/etc | head –n 15，屏幕显示出 ls 命令输出结果的前面 15 行。

输入命令：ls□/etc | tail –n 15，屏幕显示出 ls 命令输出结果的后面 15 行。

（4）用上述命令查看/etc 下配置文件 services 的内容。

查看文件内容。输入命令：cat□/etc/services。

分屏查看该文件内容。输入命令：cat /etc/services | more。

只查看前面 20 行的内容。输入命令：head□–n 20 /etc/services。

只查看后面 20 行的内容。输入命令：tail□–n 20 /etc/services。

7、清除屏幕内容命令 clear。

输入命令 clear，则屏幕内容完全被清除。

8、通配符“*”、“?”的使用。

（1）显示/bin 目录中所有以 c 为首字母的文件和目录。

输入命令 ls□/bin/c*，屏幕将显示/bin 目录中以 c 开头的所有文件和目录。

（2）显示/bin 目录中所有以 c 为首字母，文件名只有三个字符的文件和目录。

输入命令 ls /bin/c??，按 Enter 键，屏幕显示/bin 目录中以 c 为首字母，文件名只有三个字符的文件和目录。

（3）显示/bin 目录中所有的首字母为 c 或 s 或 h 的文件和目录。

输入命令：ls /bin/[c,s,h]*，屏幕显示/bin 目录中首字母为 c 或 s 或 h 的文件和目录。

（4）显示/bin 目录中所有的首字母不是 a、b、c、d、e 的文件和目录。

输入命令：ls□/bin/[!a-e]*，屏幕显示/bin 目录中首字母不是 a、b、c、d、e 的文件和目录。

重复上一步操作，输入命令：!!，自动执行上一步操作中使用过的 ls /bin/[!a-e] *命令。

9、文件归档备份命令 tar

tar 是一个归档程序，就是说 tar 可以把许多文件打包成为一个归档文件或者把它们写入备份设备，如一个磁带驱动器。tar 是一个基于文件的命令，它本质上是连续地、首尾相连地堆放文件。使用 tar 可以打包整个目录树，这使得它特别适合用于备份，通常 Linux 下保存文件都是先用 tar 命令将目录或者文件打包成 tar 归档文件（也称为 tar 包），然后用 gzip 或 bzip2 压缩。正因为如此，Linux 下已压缩文件的常见扩展名为 tar.gz、tar.bz2，以及 tgz 和 tbz 等。此外，tar 是一个命令行的工具，没有图形界面。

作为系统的备份命令，对于管理员来讲，需要常对以下目录进行备份：

/etc 包含所有核心配置文件，其中包括网络配置、系统名称、防火墙规则、用户、组以及其他全局系统项。

var 包含系统守护进程（服务）所使用的信息，包括 DNS 配置、DHCP 租期、邮件缓冲文件、HTTP 服务器文件、dB2 实例配置等。

/home 包含所有默认用户的主目录，包括个人设置、已下载的文件和用户不希望失去的其他信息。

/root 根（root）用户的主目录。

/opt 是安装许多非系统文件的地方。Openoffice、JDK 和其他软件在默认情况下也安装在这里。

创建归档可以使用-cf 参数，如果需要显示日志，则可以使用-cvf 参数。查看归档可以使用-tf 参数。

（1）将整个/etc 录下的文件全部打包成为/tmp/etc.ta，输入以下命令。

tar□–cvf□/tmp/etc.tar□/etc /*仅对文件夹打包，不进行压缩*/

tar□–cvzf□/tmp/etc.tar.gz□/etc /*文件打包后，并且以 gzip 压缩*/

tar□–cvjf□/tmp/etc.tar.bz2□/etc /*文件打包后，并且以 bzip 压缩*/

需要注意，在参数 f 之后的文件名是自己取的，习惯上都用.tar 来作为辨识；参数 z 表示以 tar.gz 或.tgz 来代表 gzip 压缩过的 tar 文件；参数 j 表示以.tar.bz2 来作为 tar 文件名。

（2）查阅/tmp/etc.tar.gz 中的归档文件，使用参数 t。输入以下命令。

tar□-tvzf□/tmp/etc.tar.gz

（3）将/tmp/etc.tar.gz 文件解压缩在/usr/local/src 下，输入以下命令。

cd□/usr/local/src

tar□-zxvf□/tmp/etc.tar.gz

（4）在/home 当中，比 2016/09/01 新的文件才备份。输入以下命令。

tar□- N□"2016/09/01"□-zcvf□home. tar. gz□/home

（5）备份/home, /etc，但不要/home/dmtsai。输入以下命令。

tar□--exclude□/home/dmtsai□–zcvf□myfile.tar.gz□/home/*□/etc

10、文档压缩命令 gzip、bzip2

文档压缩有两个明显的好处，是可以减小存储空间，二是通过网络传输时，可以减少传输的时间。gzip，bzip 是在 Linux 系统中经常使用的一个对单个文件进行压缩和解压缩的命令、如果要对多个文件同时压缩，那么就要先用 tar 打包，然后再压缩。

（1）把用户目录下的每个文件压缩成.gz 文件。

回到用户主目录，输入命令 cd□~

输入命令：gzip□*

显示压缩文件结果，注意原文件不存在了。

（2）把上述每个压缩的文件解压，并列出详细的信息，使用解压参数 d。

输入命令：gzip□-dv□*

显示是否将原文件解压。

3）详细显示例 1 中每个压缩的文件的信息，并不解压。

输入命令：gzip　-l□*

实验六　Linux 的 Vi 操作

一、 实验目的

学习和掌握利用 vi 编辑器完成文件的输入和编辑。

二、预备知识

vi 编辑器的使用

vi 编辑器是 Linux 和 UNIX 上最基本的文本编辑器，工作在字符模式下。该编辑器最初由加州大学伯克利分校为 BSD UNIX 开发，后来作为标准包含在 Linux 所有版本中。由于不需要图形界面，使它成了效率很高的文本编辑器。

vi 编辑器是 Visual interface 的简称，通常称之为 vi。它在 Linux 上的地位就像 Edit 程序在 DOS 上一样。它可以执行输出、删除、查找、替换、块操作等众多文本操作，而且用户可以根据自己的需要对其进行定制，这是其他编辑程序所没有的。vi 编辑器并不是一个排版程序，它不像 Word 或 WPS 那样对字体格、格式、段落等其他属性进行编排，它只是一个文本编辑程序。

vim 是 vi 的加强版，比 vi 更容易使用。vi 的全部命令都可以在 vim 上使用。

要在 Linux 下编写脚本、系统配置文件或者 C、C++语言程序，尤其对于系统管理员来讲，vi 编辑器是最佳的选择。使用它的好处是几乎每一个版本的 Linux 都有它的存在，并且它在操作中不需要鼠标的支持，能够胜任在各种环境下使用。

vi 有三种基本工作模式：命令行模式、文本输入模式和末行模式。

命令行模式：当执行 vi 后，首先进入命令模式用于控制屏幕光标的移动，文本字符、字或行的删除、移动、复制某区段及进入插入模式或者进入末行模式。

插入模式：在命令行模式输入相应的命令进入该模式。只有在插入模式下，才可以进行文本输入，按 Esc 键可回到命令行模式。

末行模式：在命令模式下输入“：”，可进入末行命令模式。在该模式下可保存文件或退出 vi，也可以设置编辑环境，如寻找字符串、列出行号等。

1、进入、退出 vi 和切换命令

在系统提示符号输入 vi 及文件名称后，就进入 vi 全屏幕编辑界面： $ vi□文件名

$ i 进入插入模式，可以输入文字。按“Esc”键可退回到命令模式。

退出 vi 时要在命令行模式，按“:”冒号进入末行模式，输入下述命令即可退出。

：w 文件名（vi 将以指定的文件名保存）

：w （保存文件，但不退出 vi）

：wq （保存文件，退出 vi）

：q! （放弃编辑内容，退出 vi）

2、插入命令模式

按 i 键进入插入模式后，是从光标所在位置前面开始输入文字。

按 I 键进入插入模式后，是在光标所在行的第一个非空白字符前输入文字。

按 a 键进入插入模式后，是从目前光标所在位置的下一个位置开始输入文字。

按 A 键进入插入模式后，是从目前光标所在行尾输入文字。

按 o 键进入插入模式后，是在光标所在行的下一行插入新行，从行首输入文字。

按 O 键进入插入模式后，是在光标所在行的上一行插入新行，从行首输入文字。

按 Esc 键切换为命令模式。

3、末行模式命令

在命令行模式下，按:、/、?键进入末行模式。

输入：w>>文件名，是将内容写到文件原有内容的后面。

输入：x，是对修改后信息存盘，退出 vi。

输入：r 文件名，是将文件调入到编辑缓冲区。

输入：e!，是另行编辑文件，并放弃编辑缓冲区内容。

输入：s/old/new/g，是将当前行中首次出现的字符串 old 改为 new。

输入：%s/old/new/g，是将所有行出现的字符串 old 改为 new。

输入：set nu，是将编辑器显示符号。

输入：set nonu，是不让编辑器显示行号。

输入：set all，是显示环境变量。

输入/exp,是从光标往后查找字符串 exp，如果第一次找到的字符串不是想要的，可以一直按 n 键往后查找，直到找到想要的字符串为止。

输入/exp，是从光标往前查找字符串 exp，如果第一次找到的字符串不是想要的，可以一直按 n 键往前寻找，直到找到想要的字符串为止。

三、实验内容

1、vi 新建文件 f2，输入下面内容

How to Read Faster

When I was a schoolboy I must have read every comic book ever published, But as I got older, my eyeballs must have slowed down or something I mean, comic books started to pile up faster then I could read them!

+本人学号姓名（另起一行）

注：在文件中需输入本人学号、姓名。

操作步骤：

（1）启动 OS 后，以普通用户身份登录字符界面。

（2）在 shell 命令提示符后面输入命令：vi，启动 vi 命令编辑器，进入命令模式。

（3）按 i 键，从命令模式转换为文本编辑模式，此时屏幕上的最底边出现--INSERT----字样.

（4）输入上述文本内容。如果输入出错，可使用退格键或 Delete 键删除错误的字符。

（5）按 Esc 键返回命令行模式。

（6）按：键进入最后行模式，输入 w f2 模式，就可以将正在编辑的内容保存为 f2 文件，屏幕底部显示“f2”[new] 3L，*C written 字样（其中*表示某整数），表示此文件有三行，*（*表示某整数）个字符。注意：vi 中行的概念与平时所说的行有所区别，再输入文字的过程中由于字符串长度超过屏幕宽度而发生的自动换行，vi 并不认为是一行。只有在 vi 中按一次 enter 键，另起一行的一行才是新的一行。

（7）按：键后输入 q，退出 vi。

2、打开文件 f2 并显示行号。

（1）输入命令：vi f2，启动 vi 文本编辑器并打开 f2 文件，

（2）按：键切换到最后行模式，输入命令：set nu，每一行前出现行号。

（3）vi 自动返回到命令模式连续两次输入 Z，就退出 vi。

3、在 f2 文件的第一行后插入如下一行内容：

With the development of society, the ability of reading becomes more and more important. 并在最后添加一行内容。

内容为：“We must know some methods to read-faster”。

（1）再次输入命令：vi□f2，启动 vi 文本编辑器并打开 f2 文件，显示符号。

（2）按：键进入末行模式输入数字 1，光标指示第一行。

（3） 按 a 或 i 键，进入文件编辑模式，屏幕上底部出现---INSERT---字样。利用方向键移动光标到第一行行尾后，按 Enter 键，另起一行。或者按 A 键进入文本编辑模式，光标直接定位到行尾，按 Enter 键，另起一行。或者按 O 键直接另起一行开始输入。输入 With the development of society, the ability of reading becomes more and more important.

（4）按：键进入末行模式，输入末行行尾数字，利用 Control+D 键翻屏，光标指示最后一行，将光标移动到最后一行的末尾，按 enter 键移动到最后一行末尾，按 Enter 键另起。输入 We must

know some methods to read faster.

4、将文本中所有的 eyeballs 用 eye-balls 替换。

按 Esc 键后输入“：”进入末行模式。因为当前 f2 文件中共有 5 行，所以输入命令：1,5 s/eyeballs/eye-balls/g，或者输入命令：%s/eyeballs/eye-balls/g，并按 Enter 键，

将文件中所有的 eyeballs 替换为 eye-balls。

5、把第二行移动到文件的最后，删除第一行和第二行并恢复删除，并不保存修改。

（1）按：键，再次进入末行模式，输入：2,2 m 5。

（2）按：键，输入 1,2 d，删除第一行和第二行。

（3）按 u 键恢复被删除的部分。

（4）按：键，进入末行模式，输入：q!，退出时不保存对文件的修改。

6、复制第二行，并添加到文件的最后，删除第二行，保存修改后退出 vi。

（1）再次输入命令：vi f2，启动 vi 文本编辑器并打开 f2 文件。

（2）按：键，进入末行模式，输入：2, 2 co 5，将第二行内容复制到第五行的后面。

（3）移动光标到第二行，输入：2d，原来的第二行消失。

（4）按：键，输入 wq，存盘并退出 vi。

实验七　Linux 下的 C 语言编程

一、 实验目的

学习和掌握在 linux 下用 vi 编辑器编写程序，利用 gcc 软件编译和执行 C 语言编写的程序。

二、预备知识

Linux 下 C 语言编译与调试

用户在 Linux 下编程，建议使用 C 和 C++语言。而要使用 C 和 C++语言编程，就必须要知道它们的编译和开发环境 GCC。GCC 是 GNU 推出的功能强大、性能优越的多平台编译器，是 GNU 的代表作品之一。GCC 不仅可以编译 C、C++程序，而且做了很多扩展，也可以处理 Fortran、Pascal、Objective-C、Java，以及 Ada 等语言程序。目前 GCC 的官方网址是 http:gcc.gnu.org/。

使用 GCC 由 C 语言源代码文件生成可执行文件的过程要经历四个相互关联的步骤：预处理（也称预编译，Preprocessing）、编译（Compilation）、汇编（Assembly）、和链接（Linking）。因此，其最终编译完成的目标代码的执行效率比一般的编译器相比平均要高 20%-30%。在编译的各个阶段，GCC 默认的文件格式是不同的。具体来讲，.c 为扩展的文件是 C 语言源代码文件；.a 为扩展的文件是由目标的文件构成的库文件；C、.cc 或.cxx 为扩展名的文件是 C++源代码文件；.m 为扩展的文件是 Object-C 源代码文件；.o 为扩展名的文件是是编译后的目标文件；.s 为扩展的文件是经过预编译的汇编语言源代码文件。

下面通过 HelloWord 程序开始，简单介绍 GCC 编译过程。

（1）编写源代码。

下面是 hello.c 的源代码，可以使用 vi 编辑器来编写源代码。

```
#include<stdio.h>
int main( )
```

```
{
   printf("Hello World!\n");
}
```

（2）分步编译源程序。

GCC 编译器是在命令行模式下工作，在桌面环境中，使用 GCC 编译器要在“终端”中使用。在使用 GCC 编译器的时候，用户必须给出一系列必要的调用参数和文件名称。

```
gcc -E hello.c -o hello.i      /*第一步，预编译过程：处理宏定义和
                                 include,并做语法检查*/
gcc -S hello.i -o hello.s      /*第二步，编译过程：编译过程：
                                 生成汇编代码*/
gcc -c hello.s -o hello.o      /*第三步，编译过程：编译过程：生成目
                                 标代码*/
gcc hello.o -o hello           /*第四步，连接过程：生成可执行代码。
                                 链接分为两种，一种是静态链接，另一
                                 种是动态链接。使用静态链接的好处
                                 是，依赖的动态链接库较少，对动态链
                                 接库的版本不会很敏感，具有较好的兼
                                 容性；缺点是生成的程序比较大。使用
                                 动态连接的好处是，生成的程序比较
                                 小，占用较少的内存*/
./hello                        /*执行 hello 文件，在执行 hello 文件时.
                                 在 hello 前添加“./”，shell 会在 PATH
                                 环境变量设置的目录中找可执行文件，
                                 这些目录中通常不包括当前目录，也无
                                 法找到 hello 文件*/
Hello.world!                   /*输出运行结果*/
```

其实，上述 GCC 四个编译过程，只需要一条命令就可以完成。之所以详细列出 GCC 在编译时候的每一个步骤，是希望能够清楚地理解 GCC 的编译过程。

gcc hello.c -o hello /*源代码直接变异成可执行文件*/

编写小程序的时候，都可以简单的编译源程序，以重建目标程序。但对于开发大型程序来说，由于某个程序的变动往往影响到其他程序的使用，并且程序间的依赖关系非常复杂，如果不能清晰的描述程序间的关系，则开发大程序是非常痛苦的事情，这就要使用到 make 工具，它会在必要时重新编译所有受改动影响的源文件。使用 make 工具时，程序员需要提供一个能够说明源文件之间的依赖关系和构建原则的文件，即 makefile 文件。make 命令会读取 makefile 文件的内容，它先确定要创建的目标文件，然后比较该目标所依赖的源文件的日期和时间，以决定采用哪条规则来创建目标。在 Linux 内核源代码中，就采用 makefile 文件对内核进行编译，每个目录中都存在一个 makefile 文件，任务是根据上级目录 makefile 的命令启动编译。

关于如何在 Linux 下进行软件开发已经超出了本实验的范围，感兴趣的读者可以查阅相关资料。一个强大的编译器更需要功能强大的调试器配合，才能算是完美。在 Unix 和 Linux 系统下一般使用 GDB 调试器。GDB 是 GNU 发布的一个强大的 Linux 下的程序调试工具。目前有一些针对 GDB 的“前端”程序，它们提供非常友好的用户界面，xxgdb、tgdb 和 ddd 都是这样的程序。虽然不如 Visual stdio、BCB 等可视化 IDE 的调试方便，但是 GDB 功能更加强大。

一般来说，GDB 主要调试的是 C/C++的程序。要调试 C/C++的程序，首先在编译时，要求必须把调试信息加到可执行文件中。使用编译器 GCC 的-g 参数可以做到这一点。例如

gcc -g hello.c -o hello

如果没有-g，用 GDB 调试时将看不见函数的函数名、变量名，所代替的全是运行时的内存地址。当用-g 把调试信息加入，并成功编译目标代码后，就可以使用 GDB 来调试。

启动 GDB 的方法有以下几种：

（1）gdb□program /*program 也就是执行文件*/

（2）gdb□program□core /*用 GDB 同时调试一个运行程序和 core 文件，core 是程序非法执行后，core dump 后产生的文件*/

（3）gdb□program□PID　/*如果程序是一个服务程序，那么可以指定这个服务程序运行时的进程 ID.GDB 会自动 attach 上去，并调试它。program 应该应该在 PATH 环境变量中搜索得到*/

GDB 的命令可以使用 help 命令来查看。启动 GDB 后，进入 GDB 的调试环境，可以使用 GDB 的命令调试程序。进入 GDB 后，若想退出可以在（gdb）后输入 quit。

注：若安装 VMware12.0 和 CentOS7，gcc 和 gdb 将自动安装。若安装的是 RedHat6.2，gcc 已自动安装，而 gdb 需自行安装。先下载安装 termcap，再安装 gdb。

三、 实验内容

实验基础：

先创建一个 hello.c 文件，然后编译并执行，查看结果。

```
#include<stdio.h>
main()
{
   printf(“Hello !”);
}
```

Linux 下 C 语言编程调试：

本实验的目的是调试 C 语言程序 greeting，该程序的功能是显示一个简单问候语，再将该字符串反序列出。例如，输入 Linux，反序为 xunil。

实验步骤如下。

1、用 vi 编辑器创建一个 greeting.c 文件。

```
/********greeting.c*******/
#include<stdio.h>
#include <string.h>
main()
{
```

```
    char my_string[]="hello there";
    my_print(my_string);
    my_print2(my_string);
}
void my_print(char *string)
{
    printf("the string is %s\n",string);
}
void my_print2(char *string)
{
    char *string2;
    int size,i;
    size=strlen(string);
    string2=(char *)malloc(size+1);
    for(i=0;i<size;i++)                    /*第 19 行* /
      string2[size-i]=string[i];
    string2[size+1]='\0';
    printf("The string printed backward is %s\n",string2);
}
```

2、gcc 命令编辑这个程序：gcc -o greeting greeting.c，执行程序（./greeting）。显示结果如下：

The string is hello there

The string printed backward is

输出的第一行是正确的，但是第二行并不是所期望的，所设想的输出应该是 The string printed backward is ereht olleh

显然，由于某种原因，my_print2 函数没有正常工作，可以用 GDB 对程序进行调试。

3、重新编译程序

为了能够使用 GDB 进行调试，需要在程序编译的时候加入调试参数-g，重新编译程序，输入命令：gcc -o greeting -g greeting.c

进入调试器，输入命令 gdb greeting 如果在输入命令时忘了把要调试的程序作为参数传给 gdb，则可以在 gdb 提示符下用 file 命令来载入它。

(gdb)file greeting /*该命令载入 greeting 可执行文件*/

(gdb) run /*该命令用来运行 greeting*/

查看在调试器中的输出和在 gdb 外面运行的结果是否一样。

4、查找问题

为了查找问题的所在，可以在 my_print2 函数的 for 语句后设置一个断点。具体做法是，在 gdb 提示符下输入 list 命令 2 次，列出源代码，查看行号。

(gdb) list /*该命令只能列出文件的部分，可以使用多次*/

(gdb) list

(gdb) list

假设断点的地方在第 20 行，

(gdb) break 20 /*输入如下命令作为断电*/

5、再次输入 run 命令，**程序挂起在输入断点的位置。**

6、设置一个观察 string2[size-i]变量的值的观察点来看错误是怎样产生的，输入命令:

(gdb) watch string2[size-i] /*该命令设置变量观察点*/

7、用 next 命令一步步执行 for 循环。

(gdb) next

记录每次循环后 string2[size-i]的值是多少，是否和期望值相同。确定程序出错位置，并分析其原因。如果 next 命令无法看出观察变量的变化，试着用 c 来查看。

8、修正程序错误并重新编译

找出了问题出在哪里后，退出 gdb，使用(gdb)quit 命令，修正程序错误并重新编译。下面是修改后的代码，仅供参考：

```
#include <stdio.h>
#include <string.h>
main()
```

```
{
        char my_string[]="hello there";
        my_print(my_string);
        my_print2(my_string);
}
void my_print(char *string)
{
         printf("the string is %s\n",string);
}
void my_print2(char *string)
{
         char *string2;
         int size,i;
         size=strlen(string);
         string2=(char*)malloc(size+1);
         for(i=0;i<size;i++)
                  string2[size-1-i]=string[i];
         string2[size]='\0';
         printf("The string printed backward is %s\n",string2);
}
```

实验八　Linux 的 Shell 编程

一、 实验目的

学会编写简单的 Shell 脚本程序，学会运行命令文件。

二、预备知识

Shell 本身是一个用 C 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言。作为命令语言，它交互式的解释和执行用户输入的命令；作为程序设计语言，它定义了各种变量和参数，并提供了许多在高级语言中才具有的控制结构，包括循环和分支。它虽然不是 Linux 系统核心的一部分，但它调用了系统核心的大部分功能来执行程序、建立文件并以并行的方式协调各个程序的运行。因此，对于用户来说，Shell 是最重要的实用程序，深入了解和熟练掌握 Shell 的特性及其使用方法，是用好 Linux 系统的关键。可以说，Shell 使用的熟练程度反映了用户对 Linux 使用的熟练程度。

bash。bash 是 Linux 系统默认使用的 Shell，它是 Brian Fox 和 Chet Ramey 共同完成，是 Bourne Again Shell 的缩写，内部命令一共有 40 个。Linux 使用它作为默认的 Shell 是因为它有以下的特色：I.可以使用类似 DOS 下面的 doskey 的功能，用方向键查阅和快速输入并修改命令；II.自动通过查找匹配的方式给出以某字符串开头的命令；III.包含了自身的帮助功能，只要在提示符下面输入 help 就可以得到相关的帮助。

Shell 功能与变量参数：

作为命令语言交互式地解释和执行用户输入的命令只是 Shell 功能的一个方面，Shell 还可以用来进行程序设计，它提供了定义变量和参数的手段及丰富的程序控制结构。使用 Shell 编程类似于 Windows 中的批处理文件，称为 Shell script，又叫 Shell 程序或 Shell 脚本文件。

1、Shell 基本功能

Shell 的基本功能是如何输入命令运行程序及如何在程序之间通过 Shell 的一些参数提供便利手段来进行通信。

（1）输入/输出重定向

在 Linux 中，每一个进程都有三个特殊的文件描述指针：标准输入（standard input，文件描述指针为 0）、标准输出（standard output，文件描述指针为 1）、标准错误输出（standard error，文件描述指针为 2）。这三个特殊的文件描述指针使进程在一般情况下接收标准输入终端的输入，同时由标准终端来显示输出，Linux 同时也向使用者提供可以使用普通的文件或管道来取代这些标准输入/输出设备。在 Shell 中，使用者可以利用>和<来进行输入/输出重定向。例如：

```
command >file        /*将命令的输入结果重定向到一个文件*/
command >&file       /*将命令的标准错误输出一起重定向到一个
                      文件*/
command >>file       /*将标准输出的结果追加到文件中*/
command>>&file       /*将标准输出和标准错误输出的结构都追加
                      到文件中*/
```

（2）Shell 管道

管道同样可以在标准输入/输出和标准错误输出间做代替工作。这样一来，可以将某一个程序的输出送到另一个程序的输入，其语法如下。

```
command1| command2 [| command3…]
```

也可以连同标准错误输出一起送入管道：

```
command1| &command2 [| &command3…]
```

（3）前台和后台

在 Shell 下面，一个新产生的进程可以通过用命令后面的符号；和&来分别以前台和后台的方式来执行，语法如下。

```
command;          /*产生一个前台的进程下一个命令须等该命令
                    运行结束后才能输入*/
```

command &　　/*产生一个后台的进程，此进程在后台运行的同时，可以输入其他的命令*/

2、Shell 程序的变量和参数

对 Shell 来讲，所有变量的取值都是一个字符串，Shell 程序采用$var 的形式来引用名为 var 的变量的值。

Shell 有以下几种基本类型的变量。

（1）Shell 定义的环境变量

Shell 在开始执行时就已经定义了一些和系统的工作环境有关的变量，这些变量用户还可以重新定义，常用的 Shell 环境变量如下。

HOME：用于保存注册目录的完全路径名。

PATH:用于保存用冒号分隔的目录路径名,Shell 将按 PATH 变量中给出的顺序搜索这些目录，找到的第一个与命令名称一致的可执行文件将被执行。

TERM：终端的类型。

UID：当前用户的标识符，取值是由数字构成的字符串。

PWD：当前工作目录的绝对路径名，该变量的取值随 cd 命令的使用而变化。

PS1：主提示符，在特权用户下，默认的主提示符是#；在普通用户下，默认的主提示符是$。

PS2：在 Shell 接收用户输入命令的过程中，如果用户在输入行的末尾输入“\”，然后按 Enter 键，或者当用户按 Enter 键时 Shell 判断出用户输入的命令没有结束，显示这个辅助提示符,提示用户继续输入命令的其余部分。默认的辅助提示符是>。

（2）用户定义的变量

用户可以按照下面的语法规则定义自己的变量。

变量名=变量值

要注意的一点是，在定义变量时，变量名前不应加符号$，在引用变量的内容时则应在变量名前加$；在给变量赋值时，等号两边一定不能留空格，若变量中本身就包含了空格，则整个

字符串都要用双引号括起来。

在编写 Shell 程序时，为了使变量名和命令名相区别，建议所有的变量名都用大写字母来表示。

在任何时候，建立的变量都只是当前 Shell 的局部变量，所以不能被 Shell 运行的其他命令或 Shell 程序所利用，export 命令可以将一局部变量提供给 Shell 执行的其他命令使用，其格式为

export 变量名

也可以在给变量赋值的同时使用 export 命令：

export 变量名=变量值

使用 export 说明的变量，在 Shell 以后运行的所有命令或程序中都可以访问到。

（3）位置参数

位置参数是一种在调用 Shell 程序的命令行中按照各自的位置决定的变量，是在程序名之后输入的参数。位置参数之间用空格分隔，Shell 取第一个位置参数替换程序文件中的$1，第二个替换$2，以此类推。$0 是一个特殊的变量，它的内容是当前这个 Shell 程序的文件名，所以$0 不是一个位置参数，在显示当前所有的位置参数时是不包括$0 的。

（4）内部命令

bash 命令解释程序包含了一些内部命令。内部命令在目录列表时是看不见的，它们由 Shell 本身提供。常用的内部命令有 echo、eval、exec、export、readonly、read、shift、wait、和点（.）。

echo。命令格式：echo arg；功能：在屏幕上打印出由 arg 指定的字符串。

eval。命令格式：eval args。功能：当 Shell 程序执行到 eval 语句时，Shell 读入参数 args，并将它们组合成一个新的命令，然后执行。

exec。命令格式：exec 命令 命令参数。功能：当 Shell 执

行到 exec 语句时，不会去创建新的子进程，而是转去执行指定的命令，当指定的命令执行完时，该进程，也就是最初的 Shell 就终止了，所以 Shell 程序中 exec 后面的语句将不再被执行。

export。命令格式：export 变量名 或 export 变量名=变量值。功能：Shell 可以用 export 把它的变量向下带入子 Shell 从而让子进程继承父进程中的环境变量；但子 Shell 不能用 export 把它的变量向上带入父 Shell。

注意：不带任何变量名的 export 语句将显示出当前所有的 export 变量。

readonly。命令格式：readonly 变量名。功能：将一个用户定义的 Shell 变量标识为不可变的；不带任何参数的 readonly 命令将显示出所有只读的 Shell 变量。

read。命令格式：read 变量名表。功能：从标准输入设备读入一行，分解成若干字，赋值给 Shell 程序内部定义的变量。

shift 语句。命令格式：Shift[n]功能：shift 语句按如下方式重新命名所有的位置参数变量：$2 成为$1，$3 成为$2…在程序中每使用一次 shift 语句，都使所有的位置参数依次向左移动一个位置，并使位置参数“$#”减一，直到减到 0。

wait。命令格式：wait[Process ID] 功能：Shell 等待在后台启动的所有子进程结束。wait 的返回值总是真。

exit。命令格式：exit[状态值] 功能：退出 Shell 程序。在 exit 之后可有选择地指定一个数字作为返回状态。

“.”（点）。命令格式：.Shell 程序文件名。功能：使 Shell 读入指定的 Shell 程序文件并依次执行文件中的所有语句。

（5）预定义变量

预定义变量和环境变量相类似，也是在 Shell 一开始时就定义了的变量。所不同的是，用户只能根据 Shell 的定义来使用这些变量，而不能重定义它。所有预定义变量都是由$符和另一个符号组成的，常用的 Shell 预定义变量如下。

$#：位置参数的数量。

$*：所有位置参数的内容。

$?：命令执行后返回的状态。

$$：当前进程的进程号。

$!：后台运行的最后一个进程号。

$0：当前执行的进程名。

其中，$?用于检查上一个命令执行是否正确（在 Linux 中，命令退出状态为 0 表示该命令正确执行，任何非 0 值表示命令出错）。

$$变量最常见的用途是用做临时文件的名字以保证临时文件不会重复。

（6）参数置换的变量

Shell 提供了参数置换能力以便用户可以根据不同的条件来给变量赋不同的值。参数置换的变量有四种，这些变量通常与某一个位置参数相联系，根据指定的位置参数是否已经设置来决定变量的取值，它们的语法和功能分别如下。

变量=${参数-word}：如果设置了参数，则用参数的值置换变量的值，否则用 word 置换。即这种变量的值等于某一个参数的值，如果该参数没有设置，则变量就等于 word 的值。

变量=${参数=word}：如果设置了参数，则用参数的值置换变量的值，否则把变量设置成 word 然后再用 word 替换参数的值。注意，位置参数不能用于这种方式，因为在 Shell 程序中不能为位置参数赋值。

变量=${参数?word}：如果设置了参数，则用参数的值置换变量的值，否则就显示 word 并从 Shell 中退出，如果省略了 word，则显示标准信息。这种变量要求一定等于某一个参数的值，如果该参数没有设置，就显示一个信息，然后退出，因此这种方式常用于出错指示。

变量=${参数+word}：如果设置了参数，则用 word 置换变量，否则不进行置换。

所有这四种形式中的“参数”既可以是位置参数，也可以

是另一个变量，只是用位置参数的情况比较多。

Shell 程序的流程控制：

Shell 提供了用来控制程序执行流程的命令，包括条件、分支和循环条件，用户可以用这些命令建立非常复杂的程序。

与传统的语言不同的是，Shell 用于指定条件值的不是布尔表达式而是命令和字符串。

1、测试语句

测试语句 test 作为控制条件，通过测试 expression 来实现一个条件测试，测试语句的计算表达式的值返回真（0）或假（1）。test 在 Shell 脚本中常使用[expression]代替 test 命令，但是要注意在“[”之后和“]”之前保留空格。test 测试可以进行数值、字符和文件三个方面的测试，其测试符和相应的功能分别如下。

（1）整数测试

[n1-eq n2]表示若 n1 等于 n2，则测试条件为真。

[n1-ne n2]表示若 n1 不等于 n2，则测试条件为真。

[n1-gt n2]表示若 n1 大于 n2，则测试条件为真。

[n1-ge n2]表示若 n1 大于等于 n2，则测试条件为真。

[n1-lt n2]表示若 n1 小于 n2，则测试条件为真。

[n1-le n2]表示若 n1 小于等于 n2，则测试条件为真。

（2）字符串测试

[-z s]表示若字符串 s1 长度为 0，则测试条件为真。

[-n s]表示若字符串 s1 长度大于 0，则测试条件为真。

[s1]表示若字符串 s1 不为空，则测试条件为真。

[s1=s2]表示若两个字符串相等，则测试条件为真。

[s1!=s2]表示若两个字符串不相等，则测试条件为真。

（3）文件测试

[-e file]表示若文件存在，则测试条件为真。

[-r file]表示若文件存在且为用户可读，则测试条件为真。

[-w file]表示若文件存在且为用户可写，则测试条件为真。

[-x file]表示若文件存在且为用户可执行，则测试条件为真。

[-b file]表示若文件存在且为块设备，则测试条件为真。

[-c file]表示若文件存在且为字符设备，则测试条件为真。

[-d file]表示若文件存在且为目录文件，则测试条件为真。

[-f file]表示若文件存在且为普通文件，则测试条件为真。

[-p file]表示若文件存在且为 FIFO 文件，则测试条件为真。

[-s file]表示若文件存在且文件长度>0，则测试条件为真。

[-t file]表示若文件描述符与终端相关，则测试条件为真。

2、条件语句

Shell 程序中的条件判断是通过 if 条件语句来实现的，其一般格式为：

if 条件命令串 then 条件为真时的命令串 else 条件为假时的命令串 fi

3、循环语句

循环语句分三种：for、while 和 until。根据测试条件执行相应命令。

for 循环对一个变量的可能的值都执行一个命令序列。赋给变量的几个数值既可以在程序内以数值列表的形式提供，也可以在程序以外以位置参数的形式提供。for 循环的一般格式为：

for 变量名 [in 数值列表] do 若干个命令行 done

变量名可以使用户选择的任何字符串，如果变量名是 var，则在 in 之后给出的数值将顺序替换循环命令列表中的$var。如果省略了 in，则变量 var 的取值将是位置参数。对变量的每一个可能的赋值都将执行 do 和 done 之间的命令列表。

while 和 until 命令都是用命令的返回状态值来控制循环的。while 循环的一般格式为：

while 若干个命令行 1 do 若干个命令行 2 done

只要 while 的“若干个命令行 1”中最后一个命令的返回状态为真，while 循环就继续执行，do...done 之间的“若干个命令行 2”。

until 循环的格式为：

until 若干个命令行 1 do 若干个命令行 2 done

until 循环和 while 循环的区别在于：while 循环在条件为真时继续执行循环，为 until 是在条件为假时继续执行循环。

Shell 还提供了 true 和 false 两条命令用于建立无限循环结构的需要，它们的返回状态分别是总为 0 或总为非 0。

4、开关语句

case 条件选择为用户提供了根据字符串或变量的值从多个选项中选择一项的方法，其格式如下：

case 字符串 in

模式字符串 1)

若干个命令行 1

;;

模式字符串 2)

若干个命令行 2

;;

...

其他命令行

esac

Shell 通过计算字符串 string 的值，将其结果依次和各模式字符串匹配，直到找到一个匹配的表达式为止。如果找到了匹配项则执行它下面的命令，直到遇到一对分号（;;）为止。

在 case 表达式中也可以使用 Shell 的通配符（*、?、[]）。通常用*作为 case 命令的最后表达式，以便使在前面找不到任何相应的匹配项时执行“其他命令行”的命令。

5、退出控制语句

break 用于立即终止当前循环的执行，而 continue 用于不执行循环中后面的语句而立即开始下一个循环的执行。这两个语句只有放在 do 和 done 之间才有效。

Shell 程序的运行与调试：

1、Shell 程序的运行方法

用户可以用任何编辑程序来编写 Shell 程序。因为 Shell 程序是解

释执行的，所以不需要编译装配成目标程序。按照 Shell 编程的惯例，以 bash 为例，程序的第一行一般为#!/bin/bash，其中#表示该行是注释，叹号!告诉 Shell 运行叹号之后的命令并用文件的其余部分作为输入，也就是运行/bin/bash 并让/bin/bash 去执行 Shell 程序的内容。

（1）sh Shell 程序文件名

格式为　bash Shell 程序文件名

这实际上是调用一个新的 bash 命令解释程序，而把 Shell 程序文件名作为参数传递给它。新启动的 Shell 将去读指定的文件，执行文件中列出的命令，当所有的命令都执行完结束。该方法的优点是可以利用 Shell 调用功能。

（2）sh<Shell 程序文件名

格式为　bash<Shell 程序文件名

这种方式就是利用输入重定向，使 Shell 命令解释程序的输入取自指定的程序文件。

（3）用 chmod 命令使 Shell 程序成为可执行的

一个文件能否运行取决于该文件的内容本身可执行且该文件具有执行权。对于 Shell 程序，当用编辑器生成一个文件时，系统赋予的许可权限都是 644（rw-r-r--），因此，当用户需要运行这个文件时，只需要直接输入文件名即可。

在这三种运行 Shell 程序的方法中，最好按下面的方式选择：当刚建立一个 Shell 程序，对它的正确性还没有把握时，应当使用第一种方式进行调试。当一个 Shell 程序已经调试好时，应使用第三种方式把它固定下来，以后只要输入相应的文件名即可，并可被另一个程序所调用。

2、bash 程序的调试

在 Shell 编程过程中难免会出错，有的时候，调试程序比编写程序花费的时间还要多。

Shell 程序的调试主要是利用 bash 命令解释程序的选择项。调用 bash 的形式是

bash -选择项 Shell 程序文件名

几个常用的选择项如下。

-e：如果一个命令失败就立即退出。

-n：读入命令但是不执行它们。

-u：置换时把未设置的变量看做出错。

-v：当读入 Shell 输入行时把它们显示出来。

-x：执行命令时把命令和它们的参数显示出来。

上面的所有选项也可以在 Shell 程序内部用 set -选择项的形式引用，而 set +选择项则将禁止该选择项起作用。如果只想对程序的某一部分使用某些选择项时，则可以将该部分用上面两个语句包围起来。

（1）未置变量退出和立即退出

未置变量退出特性允许用户对所有变量进行检查，如果引用了一个未赋值的变量就终止 Shell 程序的执行。Shell 通常允许未置变量的使用，在这种情况下，变量的值为空。如果设置了未置变量退出选择项，则一旦使用了未置变量就显示错误信息，并终止程序的运行。未置变量退出选择项为-u。

当 Shell 运行时，若遇到不存在或不可执行的命令、重定向失败或命令非正常结束的等情况时，如果未经重新定向，该出错信息会打印在终端屏幕上，而 Shell 程序仍将继续执行。要想在错误发生时迫使 Shell 程序立即结束，可以使用-e 选项将 Shell 程序的执行立即终止。

（2）Shell 程序的跟踪调试 Shell 程序的主要方法是利用 Shell 命令解释程序的-v 或-x 选项来跟踪程序的执行

-v 选择项使 Shell 在执行程序的过程中，把它读入的每一个命令行都显示出来，而-x 选择项使 Shell 在执行程序的过程中把它执行的每一个命令在行首用一个“+”加上命令名显示出来。并把每一个变量和该变量所取的值也显示出来，因此，它们的主要区别在于：在执行命令行之前无-v 则打印出命令行的原始内容，而有-v 则打印出经过替换后的命令行的内容。

三、实验内容

本实验要求熟读预备知识中关于 Shell 的原理及使用方法，将可执行的命令在 Shell“终端”上进行测试执行，体会 Shell 的功能。然后将下述 Shell 脚本分别生成单独文件并执行，请描述各脚本的功能和输出结果。需要说明的是，在下述脚本中，会使用到 expr 的命令。

expr 命令一般用于整数值计算，但也可用于字符串操作。格式如下。

expr□argument□operator□argument

举例如下。

$expr□10□+□10

1、描述以下各脚本的功能和输出结果，写入实验报告。

（1）输出。

```
#!/bin/bash
echo□“Please type your number:”
read a
6
for ((i=0;i<=a;i++))
do
for ((p=1;p<=i;p++))
do
echo -n “$p”
done
echo
done
```

（2）计算器。

```
#!/bin/bash
s=0
while true
echo “.....+”
echo “.....-”
```

```
echo “.....*”
echo “...../”
echo “.....q”
echo “Please type your word:(e.g.1 + 2)”
read a b c
do
case $b in
+)
let s=a+c
echo “ $a + $c =” $s;;
-)
let s=a-c
echo “ $a - $c =” $s;;
*)
let s=a*c
echo “ $a * $c =” $s;;
/)
let s=a/c
echo “ $a / $c =” $s;;
esac
case $a in
q) break ;;
esac
done
```

（3）输出当前目录下所有文件，并输出文件总数和目录总数。

```
#!/bin/bash
ls -al
filenum=0
dirnum=0
for□q□in□`ls –a`        /*注意是斜撇*/
```

```
do
if [□-d□$q ]
then
dirnum=`expr $dirnum + 1`
fi
if [□-f□$9□]
then
filenum=`expr $filenum + 1`
fi
done
echo “The number of dirctory is $dirnum”
echo “The number of file is $filenum”
```

（4）带参数的输入。先将 shell 命令写入一个文件，然后这样执行：

```
chmod□777 □文件名
文件名 a b c d e f
#!/bin/bash
for (( b=$#; b>0;b-- ))
do
echo $*;
shift;
done
```

（5）用一个文件记录系统中所有 rpm 包名，如果文件不存在则创建，如果存在则显示文件内容。

```
#!/bin/bash
a=/root/file
if [ -f $a ]
then
cat $a
else
rpm -qa > $a
```

fi

2、编写一个 Shell 脚本，完成以下功能：创建四个文件 test1、test2、test3、test4，实现自动创建 dir1、dir2、dir3、dir4 四个目录，并将 test1、test2、test3、test4 四个文件复制到 dir1、dir2、dir3、dir4 相应的目录下。请将 Shell 脚本和实验结果截图写入实验报告。