一、操作系统历史

操作系统：统一管理软硬件资源，给应用层提供提供统一的接口。
计算机：美国军方发明，当时用语计算。
第一代：二十世纪 40~50 年代，电子管计算机，机器语言。
第二代：二十世纪 50~60 年代，贝尔实验室发明晶体管，机器语言+汇编语言。
第三代：二十世纪 60~70 年代，小型集成电路计算机，机器语言+汇编+B 语言。
第四代：二十世纪 70 年代至今，大型集成电路计算机，操作系统（ c、c++）。

UNIX：1967 年，美国贝尔实验室， 汇编 ->B 语言 ->C 语言。大型服务器。
DOS：集成在 windows 上的字符操作界面，
Linux：1991 年芬兰大学生 Linus。开源代码。 GNU 组织(反版权) GPL 协议
占据大部分除个人计算机以外的其他市场。
Windows：1975 年比尔盖茨创办微软公司(windows、 office) 90%个人 PC 市场。
Android、 Symbian：移动设备操作系统。
ios/os：苹果操作系统。
uCOS/ecos/wince：嵌入式操作系统。

二、 LINUX 简介

一、LINUX 介绍

1、 Linux：Linux 是由芬兰大学的 Linus Torvalds 李納斯发起创建的开源软件项目。
Linux 操作系统：用于嵌入式上，搭建开发环境，统一管理软硬件资源。
2、版本号 xx.yy.zz ：
①.xx 表为主版本号，yy 为次版本号，zz 为修订的版本号。
②.次版本号中，单数代表测试版，双数代表正式发行版
3、开源软件：
①源代码开放。
②GPL 协议：主要是以源代码形式发布，任何人都可以得到源代码，但是不提供任何的担保，不限制商业性质的发行和包装。
③LGPL 许可协议：允许在使用者自己的应用程序中使用程序库，即使不公开自己的源代码。
常见 Linux 操作系统：
A、红帽家族 Redhat：企业版(部分服务收费)
CentOS：完全免费的红帽
Fedora：最新功能的红帽
B、 Ubuntu ：界面非常漂亮。
C、 diban ：运用在服务器上

二、LINUX 目录

Linux 系统使用树形目录结构，所有文件都在根目录(/)下。

/bin	bin 是 binary 的缩写。这个目录沿袭了 UNIX 系统的结构，存放着使用者最经常使用的命令。例如 cp、ls、cat，等等。

/boot	这里存放的是启动Linux时使用的一些核心文件。
/dev	dev 是 device（设备）的缩写。这个目录下是所有Linux的外部设备，其功能类似 DOS 下的.sys 和 Win 下的.vxd。在Linux中设备和文件是用同种方法访问的。例如：/dev/hda 代表第一个物理 IDE 硬盘。
/etc	这个目录用来存放系统管理所需要的配置文件和子目录。
/etc/gateways	设定路由器
/etc/sysconfig/	设置 IP
network-scripts	
/ifcfg-eth0	
/etc/resolv.conf	设置 DNS
/etc/fstab	记录开机要 mount 的文件系统（做磁盘配额的时候用过这个文件）
/etc/inittab	设定系统启动时 init 进程把系统设置成什么样的 runlevel
/etc/group	设定用户的组名与相关信息
/etc/passwd	帐号信息
/etc/shadow	密码信息
/etc/exports	设定 NFS 系统用的
/home	用户的主目录，比如说有个用户叫 wang，那他的主目录就是/home/wang 也可以用~wang 表示。
/lib	这个目录里存放着系统最基本的动态链接共享库，其作用类似于 Windows 里的.dll 文件。几乎所有的应用程序都须要用到这些共享库。
/lost+found	这个目录平时是空的，当系统不正常关机后，这里就成了一些无家可归的文件的避难所。对了，有点类似于 DOS 下的.chk 文件。
/mnt	这个目录是空的，系统提供这个目录是让用户临时挂载别的文件系统。
/proc	这个目录是一个虚拟的目录，它是系统内存的映射，我们可以通过直接访问这个目录来获取系统信息。也就是说，这个目录的内容不在硬盘上而是在内存里。
/root	系统管理员（也叫超级用户）的主目录。作为系统的拥有者，总要有些特权啊！比如单独拥有一个目录。
/sbin	s 就是 Super User 的意思，也就是说这里存放的是系统管理员使用的管理程序。
/tmp	这个目录不用说，一定是用来存放一些临时文件的地方了。
/usr	这是最庞大的目录，我们要用到的应用程序和文件几乎都存放在这个目录下。其中包含以下子目录；

/usr/X11R6	存放 X-Window 的目录；
/usr/bin	存放着许多应用程序；
/usr/sbin	给超级用户使用的一些管理程序就放在这里；
/usr/doc	这是Linux文档的大本营；
/usr/include	Linux下开发和编译应用程序需要的头文件，在这里查找；
/usr/lib	存放一些常用的动态链接共享库和静态档案库；
/usr/local	这是提供给一般用户的/usr 目录，在这里安装软件最适合；
/usr/man	man 在Linux中是帮助的同义词，这里就是帮助文档的存放目录；
/usr/src	Linux开放的源代码就存在这个目录
/var	这个目录中存放着那些不断在扩充着的东西，为了保持/usr 的相对稳定，那些经常被修改的目录可以放在这个目录下，实际上许多系统管理员都是这样干的。顺带说一下系统的日志文件就在/var/log 目录中。

三、总结

1、用户应该将文件存在/home/user_login_name 目录下(及其子目录下)。
2、本地管理员大多数情况下将额外的软件安装在/usr/local 目录下并符号连接在 /usr/local/bin 下的主执行程序。
3、系统的所有设置在/etc 目录下。
4、不要修改根目录（“/”）或/usr 目录下的任何内容，除非真的清楚要做什么。 这些目录最好和 LINUX 发布时保持一致。
5、大多数工具和应用程序安装在目录：/bin, /usr/sbin, /sbin,/usr/x11/bin,/usr/local/bin。
6、所有的文件在单一的目录树下。没有所谓的“驱动符”。

三、常用命令及帐户管理

一、linux 命令格式

1、linux 命令的通用格式： 命令字 [命令选项] [命令参数]
注： 三者之间要用空格隔开，其中命令选项分短格式和长格式。
短格式使用 “-”符号， 例：-l;
长格式使用“--”符号， 例：--help;
还可以使用组合命令选项：例：-a -l 可以组合成 --al 或 --la。
2、命令提示符： 终端提示符[root@localhost ~]#
@之前：代表登录用户； @之后：是一个主机名；
主机名之后：当前终端的路径;
#:超级用户的提示符 $:普通用户提示符
3、回车的使用：我们输完一个命令后，则要以回车符作为所输入命令的结束
4、获得命令帮助
（1）help 命令 （对于内部命令） 例：help pwd
（2）--help 命令选项 （对于外部命令） 例：touch --help
（3）man 命令 （用于行册页）
（4）info 命令 （用于信息页）

二、常用命令

1、目录操作命令

（1）ls　　// 列举当前目录下的文件

(红色一般是压缩包，蓝色是文件夹，白色是普通文件，绿色代表可执行文件)

ls －l　　//列举文件的详细信息

ls －al　　//显示所有（包括隐藏的）文件和目录的列表

(Linux 下隐藏文件都是以 . 开头)

drwxr-xr-x　2 root root 4096　08-22 22:29　Desktop

第一个字母：文件属性

(d:目录文件， -：普通文件， s:套接字， l: 管道文件， p:链接文件)

三个字母一组分别表示权限 (r:可读， w:可写， x:可执行)

当前用户 － 当前用户组 － 其他用户组

（2）pwd　　//显示当前目录

（3）cd　　//目录更改命令

注：相对路径是以“.”或“. .”开始的路径；

绝对路径是以“/”开始的路径；

（4）mkdir　//新建目录命令

mkdir xxxx　　//创建指定的目录

mkdir xxx/xxx －p　　//没有上一级目录时,会创建上一级目录

（5）rmdir　//册除空目录命令

2、文件操作命令

（1）file　　//文件类型查看命令

（2）touch　//新建文件命令

（3）cp　　//复制文件或目录命令

cp －R　//复制非空目录

如果要从当前目录中复制到目标目录，目标目录写明详细目标绝对路径；如果要从其它目录中复制到当前目录，那么其它目录要为详细目标绝对路径

（4）rm　　//删除文件命令

rm －r　//强制删除目录但会出现提示

rm －rf　//强制删除目录且不出现提示

（5）mv　　//文件移动与文件重命名

（6）find　　//查找文件命令

find / -amin -10　　#查找在系统中最后 10 分钟访问的文件

find / -atime -2　　#查找在系统中最后 48 小时访问的文件

find / -empty　　#查找在系统中为空的文件或者文件夹

find / -group cat　#查找在系统中属于 groupcat 的文件

find / -mmin -5　　#查找在系统中最后 5 分钟里修改过的文件

find / -mtime -1　　#查找在系统中最后 24 小时里修改过的文件

find / -nouser　　#查找在系统中属于作废用户的文件

find / -user fred　　#查找在系统中属于 FRED 这个用户的文件

（7）whereis [应用程序名]　　//查找应用程序名的路径

3、文件查看命令

（1）cat　　//文本文件查看　（不能分屏显示）
（2）more　　//文本文件查看　（能分屏显示）
（3）less　　//文本文件查看　（能分屏显示，方便反复浏览）
（4）head [-数字]　　//显示文件首部 [指定行] 内容
（5）tail [-数字]　　//显示文件尾部 [指定行] 内容
（6）du　　//查看指定目录的大小

4、光盘的基本使用

（1）光盘驱动器设备文件　　/dev/cdrom
（2）挂载光盘使用 mount 命令　　# mount -t iso9660 /dev/cdrom /media/cdrom/

（3）使用命令访问光盘挂载点目录 # ls /media/cdrom/
（4）卸载光盘使用 umount 命令　　# umount /dev/cdrom

（5）使用 cp 命令制作光盘镜像文件　# cp /dev/cdrom mydatacd.iso
（6）使用 mount 命令挂载光盘镜像文件
　　# mount -o loop -t iso9660 mydatacd.iso /media/cdrom/
（7）通过挂载点目录访问 ISO 镜像文件的内容　# ls /media/cdrom/
（8）使用 umount 命令卸载光盘镜像文件　　# umount /media/cdrom/

5、U 盘的使用方法

（1）识别 USB 存储设备，包括 USB 硬盘、U 盘、MP3 播放器等
　/dev/sda /dev/sdb ……　　# fdisk －l
（2）使用 mount 命令挂载 U 盘　　# mount -t vfat /dev/sda1 /mnt/
（3）通过挂载点目录访问 U 盘的内容　　# ls /mnt
（4）使用 umount 命令卸载 U 盘　　# umount /mnt

三、用户管理命令

1、存放文件及意义

Linux 环境下的帐户系统文件主要有：
/etc/passwd、 /etc/shadow、 /etc/group、/etc/gshadow 四个文件。

（1）、/etc/passwd 每行定义一个用户帐户，此文件对所有用户可读。
一行又划分为多个字段定义用户帐号的不同属性，名字段间用“：”分隔。

表：/etc/passwd 文件中各字段的含义

字段	说明
用户名	用户登陆系统时使用的用户名，在系统中是唯一的
口令	存放加密的口令，口令是 x，这表明用户的口令是被/etc/shadow 文件保护的
用户标识号	系统内部用它来标识用户，每个用户的 UID 都是唯一的。root 用户的 UID 号是 0，普通用户从 500 开始，从 1 - 499 是系统的标准帐户
组标识号	系统内部用它来标识用户所属的组
注释性描述	例如存放用户全名等信息
宿主目录	用户登陆系统后所进入的目录
命令解释器	指示该用户使用的 Shell，Linux 默认的是 bash

（2）、/etc/passwd 文件对任何用户均可读，为了增加系统的安全性，
用户的口令通常用 shadow passwords 保护。/etc/shadow 只对 root 用户可读

表：/etc/shadow 文件中各字段的含义

字段	说明
用户名	用户的帐户名
口令	用户的口令，是加过密的
最后一次修改的时间	从 1970 年 1 月 1 日起 ,到用户最后一次更改口令的天数
最小时间间隔	从 1970 年 1 月 1 日起，到用户可以更改口令的天数
最大时间间隔	从 1970 年 1 月 1 日起，到必须更改口令的天数
警告时间	在口令过期之前多少天提醒用户更新
不活动时间	在用户口令过期之后到禁用帐户的天数
失效时间	从 1970 年 1 月 1 日起，到帐户被禁用的天数
标志	保留位

（3）、/etc/group 将用户进行分组是 Linux 对用户进行管理及控制访问权限的一种手段。
一个组中可以有多个用户，一个用户也可以属于多个组。该文件对所有用户可读。

表：/etc/group 文件中各字段的含义

栏位	说明
组名	组的名称
组口令	用户组的口令，用 x 表示
GID	组的识别号，
组成员	该组的成员

（4）、/etc/gshadow 该文件用于定义用户组口令、组管理员等信息，
该文件只有 root 用户可读。

表：/etc/gshadow 文件中各字段的含义

栏位	说明
组名	组的名称
组口令	用户组的口令，保存已加密的口令
组的管理员帐号	组的管理员帐号，管理员有权对该组添加、删除帐号
组成员	该组的成员，多个用户用‘,’分开

2、用户管理命令

adduser　　//添加用户账号
passwd　　//设置（更改）用户口令
userdel　　//删除用户账号
（只能删除/etc/passwd /etc/shadow /etc/group ,用户宿主目录下的信息不能删除）
userdel －r　　//删除用户账号所有信息，包括宿主目录下的配置文件。
usermod　　//设置属性
usermod －L (passwd -l) 用户名　　//禁用指定用户账号
usermod －U (passwd -u) 用户名　　//开启指定用户账号
usermod -g 组名 用户名　　//将指定用户加入某个组

3、用户组的管理命令及文件

groupadd　　//添加组账号
gpasswd　　//设置（更改）用户组口令

groupdel　　//删除组账号
chmod　　//修改文件权限
　　A、数字法：　r:4 w:2 x:1 //chmod 777 xxx　文件
　　B、加减法：　u:代表当前用户　g:当前用户所在组　o:其他组
　　+:增加权限　-:　减去权限
　　例：//chmod u+x xxx　文件
　　//chmod g+w xxx　文件

chown　　//更改文件的属主和属组
chown -R　用户名　文件名或目录名　　//更改文件或目录的属主
chgrp -R　工作组名　文件名或目录名　　//改变文件或目录工作组的属主
　　-R　表示递归修改子目录中文件

4、口令维护命令

passwd（用户帐户名）　　//设置用户口令
gpasswd　－a（用户帐户名）（组帐户名）　　//将用户添加到指定组
gpasswd　－d（用户帐户名）（组帐户名）　　//将用户从指定组中删除
gpasswd　－A（用户帐户名）（组帐户名）　　//将用户指定为组的管理员

5、用户和组状态命令

su（用户名）　　//切换用户帐户
id（用户名）　　//显示用户的 UID、GID
whoami　　//显示当前用户的名称
groups（用户名）　　//显示用户所属的组
newgrp（用户所属的组帐号）　//转换用户的当前组到制定的组

6、图形界面:　　命令行启动　system-config-users
　　菜单启动　　Applications-system settings-user and Groups

使用技巧:

1、有关删除文件和文件夹的技巧：在字符界面执行 rm 将会彻底删除，如果加- i ,则在要删除某目录或者文件时,可以提示用户是否确定要执行,从而防止误删除。如果在 windows 中可以先放回收站，如果以后需要时再恢复。

2、使用 rmdir 可以删除空目录，但是如果非空则报错，这时可用 rm -fr＜目录＞来删除目录。其中参数- r 为将整个目录全部删除，包括所有的子目录。-f 则是忽略不存在的文件，不给用户作任何提示。

3、先使用“gpasswd －a 用户名 组名”把用户加入某组才可以使用命令查看某组内所有的成员列表。

4、使用 groupdel 命令删除某组的时候不能删除用户的主组和有用户的组

5、使用“usermod －g 组名 用户名”可以将成员从一个组调到另一个组。

四、Linux 下的基本工具

一、Linux 系统中的编辑器

1、文本编辑器的分类:　根据编辑范围有：行编辑器、全屏幕编辑器
　　根据工作界面环境划分：字符界面编辑器、图形界面编辑器
　　全屏编辑器：vi、 vim(红帽家族独有的 vi 升级版)

图形编辑工具：gedit、 kedit //gedit xxx 文件:用 gedit 打开 xxx 文件;

2、 vim 的配置

Linux 通过用 vi 对系统配置文件的修改对 Linux 系统进行比较细致的管理工作

修改 vim 的配置文件:（当前用户的 home 目录下的.vimrc 文件）

Vim 每次启动时会读取.vimrc 文件的配置信息。

3、编码格式:（使用 gedit 编辑文件后，用 vim 打开文件检查一下语法是否高亮。 ）

4、vi(vim) 的模式 (vim xxx 文件 //默认进入命令行模式(选择模式))

命令模式：可在命令模式下面输入单字符或组合键可以实现相应的编辑命令操作。

输入模式：命令模式下按 A I O a i o 进入输入模式进行编辑。

a：插入到光标位置的下一个;

i：插入到光标位置的前一个;

o：插入到光标位置的下一行;

A：插入到光标位置的末尾;

I：插入到光标位置的开始位置;

O：插入到光标位置的下一行开始位置;

末行模式：在命令模式下按”:”进入末行模式;

在输入模式按”Esc”+”:”进入末行模式;

set nu:显示行号； ./字符串：搜索字符串

5、命令:

一般模式	编辑模式	指令模式
h 左	a,i,r,o,A,I,R,O 进编辑模式	:w 保存
j 下	dd 删除光标当前行	:w! 强制保存
k 上	ndd 删除 n 行	:wq! 保存后离开
l 右	yy 复制当前行	:e! 还原原始挡
0 ,^ 移动到行首	Nyy 复制 n 行	:w filename 另存为
$ 移动到行尾	P,p 粘贴	:set nu 设置行号
H 屏幕最上	u 撤消	:set nonu 取消行号
M 屏幕中央	Ctrl+r 重做上一个动作	ZZ 保存离开
L 屏幕最下	Ctrl+z 暂停退出	:set nohlsearch 永久的关闭高亮显示
G 档案最后一行	/word 向下搜索	:sp 同时打开两个文档
	? word 向上搜索	Ctrl+w 两个文档设换
	Gg 移动到档案第一行	:nohlsearc 暂时关闭高亮显示

二、gcc 编译器和 gdb 调试工具

gcc: Linux 下的正统编译器，部分兼容 C99 语法。

C 语言标准:

1985 年 --- 1989 年，美国标准化委员会起草 C 语言标准。 C89、 ANSI c。

1990 年， 国际化标准委员会(ISO) C90。 C89、 c90 --- 标准 C。
1999 年， ISO 增加新的关键字 (inline、 bool……) C99 标准。
2011 年， ISO C11 标准。

1、分步编译：

编辑文本文件 – 预处理 – 编译 – 汇编 – 链接库 – 生成可执行文件

```
gcc -E xxx.c -o xxx.i        //将.c 文件编译到预处理处停止
gcc -S xxx.i -o xxx.s        //将预处理后的文件编译到汇编处停止
as xxx.s -o xxx.o            //编译生成二进制文件
gcc xxx.o -o xxx             //生成 xxx 可执行文件
```

2、直接编译：

```
gcc xxx.c                    //默认编译生成 a.out 的可执行文件
gcc xxx.c -o xxx             //编译生成可执行文件，可更改名字
gcc -o xxx xxx.c             //编译生成可执行文件，可更改名字 -o 后必须接生成的名字;
```

3、执行可执行程序

```
./xxx 可执行文件             //执行当前文件夹的 xxx 文件
                             (加./是因为所在路径没有在环境变量里)
```

Linux 下的环境变量：程序运行所需要的参数。

```
export PATH=$PATH:/xxx 路径      //将 xxx 路径添加到环境变量中(重启后会失效)
                    //将设置环境变量的语句添加到开机自启动配置文件中。(永久生效)
```

4、gdb: 调试工具

```
a、 gcc xxx.c -g  //生成可调式文件
b、 gdb a.out     //调试可执行程序
c、 l             //显示所有代码;
d、 r             //运行程序;
e、 b n           //设置断点(n 代表行数);
f、 p num         //打印 num 的值;
g、 q             //退出;
```

三、库和工程管理工具

库：将函数源码打包 头文件：相关函数的声明

库的作用：封装、保护源码、收费

/lib --- 库文件 xxx.so (动态库) xxx.a(静态库)

1、制作静态库

```
gcc -c xxx.c                 //编译生成.o 文件
ar -rc xxx.a xxx.o           //将.o 文件打包生成静态库
gcc xxx.c xxx.a -o xxx       //利用静态库编译程序
```

2、制作动态库

```
gcc -c xxx.c                              //编译生成.o 文件
gcc -fpic -shared xxx.o -o xxx.so         //编译生成动态库
gcc xxx.c xxx.so -o xxx                   //利用动态库编译程序
```

3、区别：

静态库：编译的时候将库文件直接打包进可执行程序里面；

动态库：执行可执行程序时再去环境变量中链接库文件；

ldd 命令：查看可执行程序时需要的库文件；

注：直接将动态库拷贝到/lib 或/lib64 目录下即可执行通过。

4、MAKE 工程管理工具：

①有许多文件要编译怎么办?

②假如修改一个文件， 整个工程还需要重新编译吗?

Linux 下有 make 工程管理工具，管理当前目录下文件的编译工作。

依赖于 makefile/Makefile 文件， makefile 文件里面写当前目录下的文件的编译规则。

如何编写 makefile? (最好用 vim 编写)

目标：依赖

xxxxxxx 语法命令

make //执行 makefile

```
main:main.c libfun.a
    gcc main.c libfun.a -o main
libfun.a:fun.o
    ar -rc libfun.a fun.o
fun.o:fun.c
    gcc -c fun.c
```

make clean //还原

```
main:main.c libfun.a
    gcc main.c libfun.a -o main
libfun.a:fun.o
    ar -rc libfun.a fun.o
fun.o:fun.c
    gcc -c fun.c
clean:
    rm *o *a main -f
```

五、 SHELL 的使用

一：Shell 的环境

Shell 程序位于操作系统内核与用户之间，负责接收用户输入的命令，在对已输入的命令进行解释后，将需要执行的命令程序传递给操作系统内核执行，因此程序充当了一个“命令解释器”的角色。 如 OS 中的 command.exe 程序， windows 中的 cmd.exe 程序。

二、Bash 的主要功能

(1) Bash 功能为用户提供了方便的命令编辑环境。

(2) Bash 的命令和文件名补全功能为用户提供了快速输入命令和文件名的方式。

(3) Bash 的命令历史功能使用户可以重复执行已使用过的命令。

(4) Bash 的命令别名功能为用户提供了快速输入复杂命令的方法。

(5) Bash 支持对用户提交的作业进行控制，
提供查看作业信息、调整作业成绩的运行方式等功能。

(6) Bash 允许用户将常用的命令序列定义为功能键，实现一键操作的效果。

(7) Bash 提供了丰富的变量类命令与控制结构，增强了 Shell 脚本程序的灵活性。

三、Shell 变量

1、环境变量

（1）查看环境变量：set 命令　　例：set | more

（2）显示字符串或 Shell 变量的值：echo 命令　　例：echo $PATH

（3）常用环境变量介绍

USER 表示当前用户的登录名称

UID 表示当前用户的用户号

SHELL 表示当前用户的登录的 Shell

HOME 表示当前用户的登录的宿主目录

PWD 表示用户当前所在的目录

PATH 表示当前用户的命令搜索路径

PS1 表示当前用户的主提示符

PS2 表示当前用户的辅助提示符

（4）环境变量全局配置文件“profile”和“bashrc”

2、位置变量

3、预定义变量

4、用户自定义变量

（1）自定义变量的设置　　例：DAY=Sunday

（2）自定义变量的查看与引用　　例：echo $DAY

set | grep DAY

（3）export 命令用于输出变量为全局变量

例：export DAY=Sunday

则变量名 DAY 成为了全局变量，全局变量可以应用于所有的子 Shell

（4）自定义变量的清除　　例：unset DAY

四、Bash 常用功能

1、命令和文件名补全功能：按 Tab 键

2、历史命令：　history 命令

查看历史保存文件命令：~/.bash.history

历史清除命令：history -c

快速找到历史列表中的一个命令：$ history | grep cat

3、别名命令：　alias

别名的显示命令：alias

别名的定义命令：alias ss=‘ls －l’

别名取消命令：　alias=ss　或　Alias －a

五、管道与重定向

1、标准输入输出

2、重定向：

（1）输入重定向：<

（2）输出重定向：>,>>(追加)

将标准输出重定向到文件：$ ls /etc/ > etcdir

将标准输出重定向追加到文件：$ ls /etc/sysconfig/ >> etcdir

（3）错误重定向： 2>，2>>

将错误输出重定向到文件：$ nocmd 2> errfile

（4） 输出与错误重定向的组合使用:&>

将标准输出和错误输出重定向到文件：$ ls afile bfile &> errfile

3、管道

“|”符用于连接左右两个命令，将“|”左边的命令执行结果（输出）作为“|”右边命令的输入。 （相当于加工处理）

使用方法：命令 1|命令 2|命令 3……|命令 n

使用举例： $ ls －Rl /etc | more

$ cat /etc/passwd | wc

$ cat /etc/passwd | grep lrj

$ ps －aux |tail -2 | more

六、Shell 脚本

1、shell 脚本的特点：

（1）shell 脚本相当于 DOS 中的批处理文件，是多个命令的集合

（2）shell 脚本保存在文本文件中，我们可以对其进行阅读和编辑

（3）shell 脚本由 Shell 环境解释执行的，不需要在执行前进行编译

（4）shell 脚本执行 Shell 程序时，Shell 脚本文件需要具有可执行（X）的属性

2、基本脚本编程

（1）建立 Shell 文件 例：vi hello.sh

（2）脚本运行环境设置 例：#!/bin/bash （注：“#!”与路径名之间没有空格）

（3）注释行的使用：以“#”符开始，只是起解释说明的作用

例：# This is my first HelloWorld program

（4）脚本语句：脚本语句的内容就是我们根据需要实现某种功能而输入的一些命令集合

例：mkdir /root/aaa

touch /root/aaa/test

echo Hello!

3、脚本运行的方法（例：hello.sh 为脚本文件）

（1）bash hello.sh （不需要可执行属性）

（2）.hello.sh （不需要可执行属性）

（3）./hello.sh （相对路径，需要可执行属性）

（4）/root/hello.sh （绝对路径，需要可执行属性）

六、应用程序安装与管理

一、Linux 应用程序组成

1、普通执行程序文件，保存在“/usr/bin”目录中

2、服务器执行程序文件和管理程序文件，保存在“/usr/sbin”目录中

3、应用程序配置文件，保存在“/etc”目录下

4、应用程序文档文件，保存在“/usr/share/doc/”目录下

5、应用程序物册页文件，保存在“/usr/share/man”目录下

二、RPM（Redhat Package Manager）包管理

1、RPM 包的查询命令

rpm －qa　　//查询 Linux 系统中的所有软件包
rpm －q 包名称　　//查询指定名称软件包是否安装
rpm －qi 包名称　　//查询指定名称软件包的详细信息
rpm －ql 包名称　　//查询指定名称软件包包中所包括的文件列表
rpm －qf 包名称　　//查询指定文件所属的软件包
rpm －qpi 包名称　　//查询指定 RPM 包文件的详细信息
rpm －qpl 包名称　　//查询指定 RPM 包中包含的文件列表

2、使用 rpm 命令安装软件包

rpm －i 安装包名称 （这是基本安装）
rpm －ivh 安装包名称 （安装时会显示详细信息）

注：RPM 包的依赖关系，例：A 依赖于 B，则必须先安装 B 再安装 A。

3、使用 rpm 命令卸载软件包

rpm －e 软件包名称

注：RPM 包的依赖关系，如：A 依赖于 B，则必须先卸载 A 再卸载 B。

4、使用 rpm 命令升级软件包

rpm －U 软件包名称

注：如果该软件包没有安装就直接安装到当前系统。

三、应用程序的编译安装

1、确认当前系统中具备软件编译的环境：rpm －qa ， grep gcc
2、获得应用程序的源代码软件包的文件夹（挂载光盘文件）
3、释放源代码软件包

tar zxf 包名称 （该包格式的后缀名为 .tar.bz2）
tar jxf 包名称 （该包格式的后缀名为 .tar.gz）

4、设置安装路径：　./configure －prefix=程序安装目录的绝对路径
5、程序编译过程：　make
6、程序安装过程：　make install
7、清理多余文件：　make clean
8、卸载：　make uninstall

四、在图形界面系统工具完成 RPM 包安装

1、命令方式：system-config-packages
2、菜单项启动方式：Applications → System Settings → Add/Remove Applications

七、 Linux 系统管理

一、启动过程：

①开机流程简介：

1、加载 BIOS 的硬件信息，并取得第一个开机装置的代号；
2、读第一个开机装置的 MBR 的 boot Loader (亦即是 lilo, grub,spfdisk 等等)的开机信息；

3、加载 Kernel 操作系统核心信息，Kernel 开始解压缩，并且尝试驱动所有硬件装置；
4、Kernel 执行 init 程序并取得 run-level 信息；
5、init 执行 /etc/rc.d/rc.sysinit 档案；
6、启动核心的外挂模块 (/etc/modprobe.conf)；
7、init 执行 run-level 的各个批次档(Scripts)；
8、init 执行 /etc/rc.d/rc.local 档案；
9、执行 /bin/login 程序，并等待使用者登入；
10、登入之后开始以 Shell 控管主机。

②启动,关机,登入,登出相关命令：

<login>　　登录
<logout> 、<exit>　　登出
<shutdown>、<halt>　停止系统
<reboot>　　重启动
<poweroff>　切断电源
<sync>　　把内存里的内容写入磁盘
<lilo>、<grub>　安装 lilo 启动管理程序

二、运行级别：

0：关闭
1：单用户模式，用于管理员对系统进行维护。
2：多用户模式，在该模式下不能使用 NFS。
3：完全多用户模式：用于将主机作为服务器。
4：保留，未分配。
5：图形登录的多用户模式：图形界面登录，图形操作环境。
6：重新启动系统。

显示当前的运行级别：runlevel
更改当前的运行级别：init 1 2 3 5

三、系统服务的启动状态：

查看服务启动状态：chkconfig －list 服务名称 ；
设置独立服务的启动状态：chkconfig --level 运行级别表 服务名称 on | off | reset ；
设置非独立服务的启动状态：chkconfig 服务名称 on | off | reset ；

非独立服务的启动状态由 xinetd 服务在系统中指定运行级别的启动状态决定，xinetd 服务启动后才能启动非独立服务程序。当使用 chkconfig 对非独立服务程序的启动状态进行更改后，需要 service xinetd restart 重新启动 xinetd 服务。

INIT 的配置文件为：/etc/inittab
系统初始化脚本：系统启动过程中，执行/etc/rc.d/rc.sysinit 后，接着执行/etc/rc.local。

进程：是 Linux 系统中的基本运行单位，可对其进行查看、调整、启用和停止操作。进程是程序代码在处理器中的运行：操作系统在执行程序时，程序代码被读取到内存中，驻留在内存中的程序代码作为进程在处理器中被动态执行。Linux 是多进程操作系统，每个程序启动时都可以创建一个或几个进程，每个进程都是一个独立的任务。

查看系统内所有进程：ps aux
简单显示当前进程：ps
查看进程树：pstree，可显示进程与子进程的详细列表。

USER－用户　　PID—进程号　　CPU－CPU 占用率　MEM－内存占用率
VSZ－虚拟内存大小　RSS－占用内存　　TTY－运行终端　　STAT－当前状态
START－启动时间　　TIME－占用 CPU 时间　　COMMAND－程序名称

全屏显示进程信息：top

q 键退出 ，P 键－按 CPU 排序 ，N 键－按打开时间排序，A 键－按 PID 号排序
在后台启动进程：命令后加“ &”
将后台程序调入终端前台执行：fg 后台程序名
结束当前进程：Ctrl＋C
将当前终端中运行的程序调入后台并停止执行：Ctrl＋Z
查看后台进程：jobs
(强制)终止进程：kill (-9) 进程号

系统初始化时调用的脚本：位于/etc/r.d 内的 rc.sysinit 和 rc.local
格式：＊＊＊＊＊ 分 时 天 月 周
定时启动任务服务：cron 服务程序的软件包名称　　vixie-cron
查询服务状态：　service crond status
启动/重启服务：　service crond start | restart
查看 cron 任务：crontab -l
覆盖原有 cron 任务：crontab
删除现有用户的 cron 任务：crontab －r
调用文本编辑器：crontab －e
用户配置 cron 任务目录：/var/spool/cron/用户名
cron 启动脚本：　/etc/init.d/crond
系统预设的 cron 任务配置文件及目录：　/etc/crontab 文件，/etc/cron.d 目录。
日志：应用程序日志、系统日志。 存放于/var/log
启动日志：boot.log

四、磁盘空间配额：

软限制－警告值，硬限制－最大值。
可对用户和组的可用磁盘空间和可使用文件数量进行设置。
①修改/etc/fstab，在分区装载设置中添加 usrquota 和 grpquota。
②重启系统。
③运行 quotacheck －cmug /，建立文件系统配额文件 aquota.user 和 aquota.group。
④edquota -u 用户名 | -g 组名，编辑用户/组配额。设置磁盘配额宽限时间：edquota －t。

Filesystem	Blocks	soft	hard	indos	soft	hard
/dev/hda3	17636	0	0	0	0	0

⑤显示配额信息：quota -u 用户名 | -g 组名。
⑥启用配额－quotaon /，停用配额－quotaoff /。
※可以使用虚拟磁盘对所做的磁盘配额进行检查。
切换用户：su －u 用户名
检查：dd if=/dev/zero of=/tmp/aa bs=1M count=2

五、压缩命令：

*.Z compress 程序压缩的档案;
*.bz2 bzip2 程序压缩的档案;
*.gz gzip 程序压缩的档案;
*.tar tar 程序打包的数据，并没有压缩过;
*.tar.gz tar 程序打包的档案，其中并且经过 gzip 的压缩

compress filename 压缩文件 加[-d]解压
gzip filename 压缩 加[-d]解压
bzip2 -z filename 压缩 加[-d]解压
bzcat filename.bz2 查看压缩文件内容

tar 解/压缩 (x:解压， c:创建解压文件， v：打印提示信息， f:接文件)
tar -cvf /home/123.tar /etc 打包，不压缩
tar －cvf xxx.tar.gz xxx 文件 将 xxx 文件压缩为 xxx.tar.gz
tar -xvf 123.tar 解开包
tar －xvf xxx.tar.gz 解压
tar -zxvf /home/123.tar.gz 以 gzip 解压
tar -jxvf /home/123.tar.bz2 以 bzip2 解压
tar -ztvf /tmp/etc.tar.gz 查看 tar 内容

cpio -covB > [file|device] 备份
cpio -icduv < [file|device] 还原
文件归档：tar cf 归档文件名.tar 备份目录、文件
文件压缩归档：tar czf 归档文件名.tar.gz 备份目录、文件
查看归档文件：tar tf 归档文件名.tar
查看压缩归档文件：tar tzf 归档文件名.tar.gz
恢复归档文件：tar xf 归档文件名.tar －C 指定目录
恢复压缩归档文件：tar xzf 归档文件名.tar.gz －C 指定目录
解压 bz2 文件：tar jxf 文件名.bz2 -v 显示归档进度

八、 Linux 基本网络配置

关于网络的命令

Linux 下的网络分为三种: (桥接、 NAT、仅主机)
桥接：虚拟机、物理机、外网同一局域网里，可以之间进行访问
NAT：虚拟机、物理机、外网在同一局域网里，虚拟机可访问外网，外网不可以访问虚拟机
仅主机：虚拟机仅和主机进行通信

网络接口：eth0－系统网络接口；lo－环回网络接口 127.0.0.1。
查看网络接口信息：ifconfig 网络接口名称 -a 全部网络接口
测试与其它主机的网络连接：ping 目的主机地址 -c 指定数据包数量 Ctrl＋C 结束发送。
测试与其它主机的网络连接路径：traceroute 目的主机地址

查看当前主机名称：hostname
配置主机名称：hostname 主机名称，配置后需要重启计算机。
查询 DNS 服务器域名：
交互模式：nslookup，输入待解析域名↙，exit 退出。用于对 DNS 服务器进行测试。
命令模式：nslookup 待解析域名。用于查询域名对应的 IP 地址。
DHCP 网络设置：dhclient
临时配置网络：ifconfig 网络接口名称 ip 地址 netmask 子网掩码
手工配置网络：netconfig
禁用 #ifconfig eth0 down， ***启动*** #ifconfig eth0 up
重启网络配置：service network restart
查看(more、 less、 find)

添加 ip 地址：ifconfig 网络接口名称:1 ip 地址 netmask 子网掩码
添加默认网关路由：route add default gw 默认网关地址
删除默认网关路由：route del default gw 默认网关地址

注意： 添加默认网关前要确认系统路由表中的默认网关纪录不存在。
route－显示默认网关
显示域名解析：nslookup，输入 server。
设置新的解析地址：server 新解析地址。
网络服务启动脚本：/etc/init.d/network
网络接口配置文件：/etc/sysconfig/network-scripts/中，接口文件名 ifcfg-xxx。
启用网络接口：ifup 网络接口名称
停止网络接口：ifdown 网络接口名称 -a 全部网络接口
主机名称配置文件：/etc/sysconfig/network 重新配置后需要重启计算机。
本地主机名称解析文件：/etc/hosts
域名服务器配置文件：/etc/resolv.conf 可设置 3 条 nameserver 配置记录。

九、 NFS 文件系统

一、NFS 的概述和安装

1、NFS 的一般用法
在运行 NFS 服务器程序的主机中进行必要的配置，提供 NFS 共享目录的输出
在 NFS 客户机挂载 NFS 服务器输出的共享目录
2、NFS 服务器的安装
3、查看所需的软件包是否安装：#rpm －q nfs-utils portmap
4、NFS 的安装包文件：#cd /media/cdrom/RedHat/RPMS/
#ls nfs-utils* portmap*
5、安装 nfs-utils 和 portmap 两个软件包：
#rpm －ivh nfs-utils-1.0.6-4.i386.rpm portmap-4.0-6.3.i386.rpm

二、NFS 服务器的配置

1、NFS 服务器的配置文件：#cat /etc/exports
2、NFS 服务器的启动与停止

查询服务器的状态： Service 服务程序脚本名称 status

启动服务器： Service 服务程序脚本名称 start

停止服务器运行： Service 服务程序脚本名称 stop

3、设置服务器的开机启动状态： #chkconfig --list portmap

#chkconfig --list nfs

4、Showmount 命令：

用于查询显示 NFS 服务器的相关信息： #showmount –help

显示主机的 NFS 服务器信息： #showmount [NFS 服务器主机地址]

显示 NFS 服务器的输出目录列表： #showmount –e [NFS 服务器主机地址]

显示 NFS 服务器中被挂载的共享目录： #showmount –d [NFSf 服务器主机地址]

显示 NFS 服务器的客户机与被挂载的目录： #showmount –a [NFS 服务器主机地址]

5、Exportfs 命令：

当系统管理员对/etc/exports 文件进行设置修改后，并不会自动在 NFS 服务器中重新生效

输出共享目录： #exportfs -rv

停止输出所有目录： #exportfs -auv

输出(启动)所有目录： #exportfs –av

三、图形界面的 NFS 服务器配置工具：

NFS 客户端配置： 在 linux 中配置使用 NFS 客户端

显示 NFS 服务器的输出： #showmount –e

挂载 NFS 服务器中的共享目录： #mount –t nfs nfs 服务器地址:目录共享 本地挂载目录点

显示当前主机挂载的 NFS 共享目录： #mount | grep mnt

卸载系统中已挂载的 NFS 共享目录： #umount /mnt/

系统启动时自动挂载 NFS 文件： 需要将 NFS 的共享目录挂载信息写入/etc/fstab/文件，以实现对 NFS 共享目录的自动挂载：

#tail –l /etc/fstab

192.168.1.163:/home/pub /mnt nfs defaults 0 0

附录：关闭开机邮箱自启动命令：

```
关闭 sendmail 启动选项：
[root@localhost /]# chkconfig --list sendmail
sendmail      0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭
[root@localhost /]# chkconfig --level 2345 sendmail off
[root@localhost /]# chkconfig --list sendmail
sendmail      0:关闭 1:关闭 2:关闭 3:关闭 4:关闭 5:关闭 6:关闭

# service network restart   重启 net 网络
# service nfs reatart   nfs 服务开启
#service xinetd restart  启动 TFTP 服务
# service iptables stop    关闭防火墙
 # setsebool ftpd_disable_trans 1  关闭 SElinuxFTP 的拦截
```

十、文件操作

一、基于缓冲区的文件操作

1、缓冲区：标准 C 读写文件都要经过一个缓冲区，

满足一定条件(清理缓冲、遇’\n’)后再进行读写。

2、常用缓冲区：stdin(标准输入)、 stdout(标准输出)、 stderr(标准错误)

scanf、 printf： 基于缓冲区的文件操作。 scanf(不安全) --- fflush(stdin);

3、编译器如何描述文件：

使用 FILE 结构体，结构体定义在 stdio.h 里，包含文件所有信息。操作文件就应该操作 FILE 结构体里的元素，编译器已经封装一个所有操作文件的库，只需调用相应函数即可。

```
struct _iobuf {
        char *_ptr;
        int   _cnt;
        char *_base;
        int   _flag;
        int   _file;
        int   _charbuf;
        int   _bufsiz;
        char *_tmpfname;
        };
typedef struct _iobuf FILE;
```

4、操作文件：调取相应函数，进行再次封装即可。

A、打开函数：FILE *fopen(const char *path,char *mode);

以指定的方式打开指定路径下的文件，成功返回操作这个文件的指针，失败返回 NULL

参数 1：路径（绝对路径、相对路径）

参数 2：打开方式

文件使用方式	含义
“r/rb”(只读)	为输入打开一个文本/二进制文件
“w/wb”(只写)	为输出打开或建立一个文本/二进制文件
“a/ab”(追加)	向文本/二进制文件尾追加数据
“r+/rb+”(读写)	为读/写打开一个文本/二进制文件
“w+/wb+”(读写)	为读/写建立一个文本/二进制文件
“a+/ab+”(读写)	为读/写打开或建立一个文本/二进制文件

B、关闭函数：fclose(FILE *fp); //关闭 fp 执行的文件

C、单个字符写操作：int fputc(int ch,FILE *fp);

D、单个字符读操作：int fgetc(FILE *fp);

E、判断读写指针是否到达文件末尾：int feof(FILE *fp);

//到达末尾返回 1，没有到达返回 0。

F、文件指针定位：

①、 rewind(FILE *fp); //将文件指针定位到文件开始位置;

②、 int fseek(FILE *fp,int offset,int whence); //将文件指针定位到指定位置。

参数 2：偏移量； （正右负左）

参数 3：参考位置 （SEEK_SET ,SEEK_CUR ,SEEK_END）;

eg：fseek(fp,0,SEEK_SET) //定位到开始位置 --- 等同 rewind(fp);

fseek(fp,0,SEEK_END) //定位到文件末尾;

fseek(fp,-1,SEEK_END)　　//通常用来去除字符串的 ’\0’ ；

③、 int ftell(FILE *fp); //返回指针位置，通常用来确定文件大小；

G、字符串写操作：int fputs(const char *s,FILE *stream);

H、字符串读操作：char fgets(char *s,int n,FILE *stream);

I、块读操作：fread

J、块写操作：fwrite

K、格式化写操作：fscanf

L、格式化读操作：fprintf

5、函数的再次封装

二、基于文件描述符的文件操作

1、文件描述符：Linux 下的无缓冲区的文件操作。

每打开一个文件，内核都有一个结构体来描述文件信息，内核里面有一个大的结构体数组，对应的文件对应到结构体数组里面的第几个。其实就是一个非负整数(0-1023)。

2、打开关闭函数

man 命令：Linux 下的帮助命令，一共有 7 级

man 1　　系统命令

man 2　　系统调用（ Linux 下接函数）

man 3　　c 库函数

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int open(const char *pathname, int flags);

int open(const char *pathname, int flags, mode_t mode);

int creat(const char *pathname, mode_t mode);

参数： pathname 路径　　参数：flags 打开方式

只读： O_RDONLY、 只写： O_WRONLY、 读写： O_RDWR、

建立： O_CREAT、

O_EXCL (与 O_CREAT 连用， 如果有文件就返回错误)

O_TRUNC(将文件长度截取为 0)....

mode： 掩码。 权限。 0755 、 0655 (创建文件的时候要指定)

open("xxx",O_RDWR|O_CREAT,0755); == creat("xxxx",0755);

close(int fd);　　//关闭 fd 指向的文件

3、写操作：ssize_t write(int fd const void *buf,size_t count);

4、读操作：ssize_t read(int fd,void *buf,size_t count);

例：

```
write(fd,"helloworld",11);
lseek(fd,0,SEEK_SET);
read(fd,str,11);
printf("str = %s\n",str);
close(fd);
```

5、 文件指针定位：lseek(int fd,int offset,int whence);

6、 其他函数：

a、 原子操作函数： 对文件进行操作的时候不允许被打断

b、 复制文件描述符

c、 文件锁(防止多进程同时对文件进行操作)

7、 命令行传参： c99 规定 main 函数的书写规则：

a、 int main()

b、 int main(int argc,char *argv[]) //参数 1:传递参数的个数 参数 2：参数存放位置

例：

```
int main(int argc,char *argv[])
{
    int i;
    for(i=0;i<argc,i++)
        printf("%s\n",argv[i]);
    return 0;
}
```

十一、进程

一、进程初步认知

1、进程的定义

可执行程序： 1.exe、 a.out --- 磁盘上的二进制文件

进程： 一个运行着的程序，是一个动态的执行体，包含程序的调度到消亡的过程。

Windows、 Linux 是一个多进程、多用户的操作系统，各个进程相互独立运行。

2、进程查看

ps –aux //查看进程

USER:用户名 PID:进程号 STAT:状态(R:运行， S:休眠， Z:僵尸)

ps –ef //可以查看父进程

3、进程描述

Linux 下用一个非负整数描述一个进程，称为 PID，进程都有自己的唯一 PID。

getpid(); //得到自己的进程号;

getppid(); //得到父进程号;

4、进程资源（ 每一个进程都有自己的独立资源 size a.out 命令查看）

A、 代码段：存放二进制代码;

B、 堆：存放动态开辟内存;

C、栈：存放局部变量以及函数参数值;

D、 数据区：存放初始化或者未初始化的全局变量、静态变量;

E、 文字常量区： 常量;

5、创建子进程

#include <sys/types.h>

#include <unistd.h>

A、 fork(void);函数 //创建成功， 父进程返回子进程的 pid； 子进程返回 0

//失败， 父进程返回-1;

//成功创建子进程，子进程会复制父进程的所有资源，

//父、 子进程会在 fork 函数之后相互独立运行。

B、 vfork(void);函数 //返回值和 fork 相同，但不会复制父进程资源， 而是独占资源，

//创建成功后，父进程会等待子进程结束后再执行。

6、子进程的资源回收

A、父进程退出的时候，子进程已经消亡，父进程会回收子进程资源;

B、父进程主动调用相关函数;

7、特殊进程

A、祖先进程:进程号为 1 的进程,又叫做 init 进程,是除 0 号进程外的祖先进程。
操作系统启动时,首先启动引导进程(0 号进程),成功启动系统后,
引导进程会消亡, 1 号进程会启动其他进程,其他进程也会启动其他进程。

B、孤儿进程:父进程先退出,子进程没人管,由祖先进程接收。(占用 CPU 资源)

C、僵尸进程:子进程退出,父进程并没有清理子进程资源,在父进程退出之前的一段时间子进程称为僵尸进程;(占用 CPU 资源)

8、进程资源的回收

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

A、pid_t wait(int *status);函数 //保存进程退出时的状态(参数通常为 NULL),
//父进程会阻塞, 等待任一子进程的退出;

B、pid_t waitpid(pid_t pid, int *status, int options);

参数: pid>0:等待指定 pid 进程退出; pid=-1 等待所有子进程退出

status:保存退出时的状态;

options: 关于阻塞问题,通常为 0;

9、进程退出

```
A、 return          //返回、 退出;
B、 exit();         //会清理缓冲区的内容
C、 _exit();        //不会清理缓冲区的内容;
D、atexit();        //进程注册指定的函数,退出时执行。
E、 kill -9 进程号     //通过发送终止信号结束进程
F、 ctrl+c、ctrl+\     //发送信号终止进程
```

10、 exec 函数族

```
#include <unistd.h>
```

A、system 函数 //执行系统命令

例: system(“ls”)//显示文件,相当于直接执行 ls

B、exec 函数族() //通常与 vfork 函数联用, 最后一个参数以 NULL 结尾

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

例: execl("/bin/ls","ls","-l",NULL);

作用: a、调用其他应用程序 b、封装应用层

注:退出进程:释放内存;
资源回收:内核当中用来保存进程信息,由父进程回收;
多进程程序:函数 vfork()、exit()、wait() 最好同时使用;
短时间内进程就结束 --- vfork(); 长时间 --- fork();

二、进程间的通信机制

每一个进程都有自己的独立空间,相互独立运行,互不干扰。如果要共同完成一个整体

事件时需要多进程之间相互协调配合，所以进程间要进行通信。

通信的作用：传输数据、通知事件、共享数据、同步事件；

通信机制的分类：A、 信号：软中断的模拟，继承 UNIX 的古老的通信机制；

B、 管道：单向导通性， 一般读写同时进行；

C、 消息队列： 类似于链表，发送的一个消息体类似于一个链表的节点；

D、 共享内存： Linux 下最快的通信机制； (常用)

E、 信号量集：同步保护资源；

F、 套接字：网络之间的通信机制；

三、信号

是一种软中断模拟机制，可通过 kill -l、 man 7 signal 命令查看；

共 62 个信号，前 31 个继承 UNIX 的非实时信号(常用)，后 31 个是 Linux 的实时信号。

每一个信号都有自己独特的含义，大部分信号的含义几乎都是结束进程。

注： SIGKILL 和 SIGSTOP 信号不能被捕捉和被忽略，只能默认处理；

SIGHUP：从终端上发出的结束信号；

SIGINT：来自键盘的中断信号（Ctrl-C）；

SIGQUIT：来自键盘的退出信号（Ctrl-\）；

SIGFPE： 浮点异常信号（例如浮点运算溢出）；

SIGKILL：该信号结束接收信号的进程；

SIGALRM：进程的定时器到期时，发送该信号；

SIGTERM： kill 命令发出的信号；

SIGCHLD：标识子进程停止或结束的信号；

SIGSTOP：来自键盘（Ctrl-Z）或调试程序的停止执行信号

信号的产生：内核发送、键盘输入(ctrl+c 、 ctrl+\)、 发送函数；

1、信号的操作： (signal 函数)

A、 忽略：忽略信号原有的含义。

B、 默认处理：遵循信号原有的含义，相当于没有对信号处理，通常会导致进程的凋亡。

C、 捕捉：实现对信号的重定义(函数)。

#include <signal.h>

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);

参数： signum:要操作的信号；

参数： handler:怎么进行操作

(SIG_IGN(忽略信号)、 SIG_DFL(默认处理)、相应的处理函数)

2、闹钟函数 (alarm 函数)

#include <unistd.h>

unsigned int alarm(unsigned int seconds);

参数 seconds：正整数，代表时间(单位：秒)

计时 seconds 秒后会给当前进程发送 SIGALRM 信号，代表计**时结束。**

例：

```
void fun()
{
    printf("time out\n");
}
int main()
{
    signal(SIGALRM,fun);
    alarm(10);
    while(1);
    return 0;
}
```

3、信号的发送（发送信号函数：raise、kill）

```
#include <signal.h>
int raise(int sig);                    //向自身发送指定信号， sig：要发送的信号；
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);     //向指定进程发送指定信号， pid：进程号， sig：  发送的信号
```

四、管道

Linux 下的管道通信机制具有单向导通性，分为无名管道和有名管道。

无名管道只用于父子之间通信，有很大局限性，无实质性文件；

有名管道可以在任意两个进程之间进行通信，并且有实质的介质管道文件；

1、无名管道：（ pipe 函数）

```
#include <unistd.h>
int pipe(int pipefd[2]); //创建一个无名管道，返回管道两端读写操作的文件描述符
                         //存放在形参中， pipefd[0]:读； pipefd[1]:写
                         //无名管道是一个虚拟的文件，无法查看；
                         //如果没有数据写入，读数据会阻塞。
```

2、有名管道：

A、 创建管道(mkfifo 函数)

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *pathname, mode_t mode);
```

参数 *pathname：创建有名管道位置； 参数 mode： 指定管道文件掩码

B、删除文件(unlink 函数)（ 不仅可以删除管道，还可以删除其他文件）

```
#include <unistd.h>
int unlink(const char *pathname);
```

C、 特点： 读写两端都要同时进行操作， 写入的时候没有读， 写数据就会阻塞。

五、共享内存

```
ipcs -m          //查看共享内存；
```

是 Linux 下最快、最常用的通信机制。

1、键值的创建

进程间通信机制：信号(内核规定好的)、管道(有名管道产生介质文件) ---- 不需要键值

共享内存、消息队列、信号量集：需要一个键值，都在内核层产生一个文件，应用层看不到，但可以通过键值保证应用层访问共享内存、消息队列、信号量集是同一个。键值相同，则代表访问的共享内存、消息队列、信号量集是同一块。

创建键值函数：参数相同，创建的键值就相同。

```
# include <sys/types.h>
# include <sys/ipc.h>
key_t ftok(const char *pathname, int proj_id);
```

参数 *pathname：路径 参数 proj_id：正整数 返回值：键值(整型数)

2、共享内存的相关函数

A、定义：是内核层的一块特殊的内存，进程可以利用它进行通信。

B、使用流程：操作时先创建或访问共享内存，再将内存映射到本进程空间，直接进行读写操作，不用的时候分离以及删除。

C、创建或访问一块共享内存 (shmget 函数)

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
```

参数 key：键值(①指定为 IPC_PRIVATE：父子间通信、②ftok 创建)

参数 size：共享内存大小，单位 字节；

参数 shmflg： IPC_CREAT(没有就创建、有就访问)

IPC_EXCL(和 IPC_CREAT 结合，如果有就返回错误)

返回值：返回共享内存的操作符，供映射、分离、删除使用。

D、映射共享内存到进程空间、分离共享内存 (shmat 函数)

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);  //映射;
int shmdt(const void *shmaddr);                  //分离
```

参数 shmid：shmget 创建共享内存成功后的返回值;

参数 shmaddr：指定映射到进程空间的地址，通常设为 NULL 表示系统自动分配;

参数 shmflg：通常为 0;

返回值:返回映射成功后共享内存在进程空间的地址;

E、 从进程空间中将共享内存分离(shmdt 函数)

```
int shmdt(const void *shmaddr);        //将 shmaddr 指向的共享内存从进程空间中分离
```

F、控制共享内存（删除、设置属性、获取属性）

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

参数 shmid：shmget 函数创建成功的返回值，要操作的共享内存;

参数 cmd：通常设为 IPC_RMID，表示删除共享内存，此时第三个参数为 NULL;

还可以为 IPC_SET、IPC_STAT，表示设置或获取共享内存信息，

此时需要第三个参数

参数 buf：描述共享内存信息的结构体(详见 man 手册)

G、内存操作函数 (men 开头的 C 库函数)

```
void memset(void *s, int c, size_t n);  //将 s 指向的内存块的前 n 个字节全部设置为 c
```

例： char str[100]; memset(str,0,100);

例： char *p = (char *)malloc(100); memset(p,0,100);

```
void memcpy(void *dest,const void *src,size_t n);
                                    //将源存储块前 n 个字节拷贝到目的存储块
```

例： memcpy(p,"helloworld",11);

六、消息队列

类似一个链表，发送接收一个节点信息。

ipcs -s //查看消息队列;

1、创建或访问一个消息队列 (msgget 函数)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

参数 key：键值(IPC_PRIVATE：父子间通信、ftok 创建)

参数 msgflg： IPC_CREAT(没有就创建、有就访问)

IPC_EXCL(和 IPC_CREAT 结合，如果有就返回错误)

返回值：返回消息队列的操作符，供映射、分离、删除使用。

2、收发消息

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);        //发送;
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);        //接收;
```

参数 msqid：msgget 创建消息对列成功的返回值;

参数 msgp：发送或接收消息的存储位置，发送的消息是自定义的结构体，

并且结构体第一个元素必须是长整型，指定消息类型(1~5)

```
struct msgbuf {
        long mtype;          /* message type, must be > 0 */
        char mtext[1];       /* message data */
    };
```

参数 msgsz：消息体的大小，不大于 4K 字节;

参数 msgtyp：要接收的消息类型(和发送消息体里的 mtype 相同);

参数 msgflg：通常设置为 0;

3、控制消息队列

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

参数 msqid：msgget 创建成功的返回值;

参数 cmd：通常设为 IPC_RMID 表示删除消息队列;

还可以为 IPC_SET、IPC_STAT，表示设置或获取消息队列信息;

参数 buf：删除的时候设为 NULL，

如果设置或获取消息队列属性的时候要定义结构体变量;

七、信号量集

```
ipcs -s        //查看信号量集;
```

信号量集是信号量的集合，单个信号量是用来保护资源或者临界区代码。信号量为正可以进行操作并将信号量减一，当信号量为零时，再有进程请求信号量则需要等待其他进程释放信号量。

1、创建或访问一个信号量集

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

#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
参数 key：键值；
参数 nsems：创建的信号量集里有几个信号量；
参数 semflg：IPC_CREAT(没有就创建、有就访问)
IPC_EXCL(和 IPC_CREAT 结合，如果有就返回错误)
返回值： 创建成功后返回一个正整数供 semop、 semctl 函数使用；

2、设置信号量集

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, ...);
参数 semid：semget 创建成功的返回值；
参数 semnum：要设置的信号量在集合中的位置，从 0 开始；
参数 cmd： IPC_RMID:删除信号量，不需要第四个参数；
SETVAL：设置信号量值，此时第四个参数为设定的值；
GETVAL：获取相应的信号量的值，不需要第四个参数；
例：semctl(semnum,0,IPC_RMID); //删除信号量集里面的第一个信号
semctl(semnum,0,IPC_SETVAL,1);//设置信号量集里面第一个信号量的值为 1
semctl(semnum,0,IPC_GETVAL) //获取信号量集里面第一个信号量的值并返回；

3、消耗或释放信号量(p 操作、v 操作)

P 操作：将信号量减一； V 操作：将信号量加一；

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops);
参数 semid：semget 函数的返回值；
参数 *sops：结构体指针，类型不用再定义了；
struct sembuf
{
unsigned short sem_num; /* semaphore number */ 要操作信号量集里第几个信号量
short sem_op; /* semaphore operation */ p 操作 -1 v 操作 +1
short sem_flg; /* operation flags */ 通常设置为 0；
}
参数 nsops：由*sops 参数决定，*sops 可以为结构体数组，就是结构体体数组的大小；

4、 生成者消费者模型

生产者： 生产 v 操作
消费者： 消费 p 操作

十二、线程

一、线程定义

是一个轻量级的进程，所有的线程都运行在一个进程空间里面，共享进程的资源。
进程是操作系统调度的基本单位,有自己的独立空间,进程间通信有自己的特殊的机制，

涉及到内核操作，会占用 CPU 的大量空间，所以就有了线程。

（之前写的 main 函数就是一个主线程程序。）

进程：占用 CPU 多，进程间通信复杂，方便调度和移植。

线程：占用资源少，通信方便，不方便调度和移植。

注：编译多线程程序时，　　gcc XXX.c －lphtread　　　　//链接线程库；

二、线程的基本函数

（严禁在线程中书写进程相关函数）

1、创建线程

```
#include <pthread.h>
int pthread_create(pthread_t *restrict thread,const pthread_attr_t *restrict attr,
                   void *(*start_routine)(void*), void *restrict arg);
```

参数 *restrict thread：线程 ID，类型 pthread_t == unsigned long　打印 %lu ；

参数 *restrict attr：线程属性，通常指定为 NULL；

参数 *(*start_routine)(void*)：创建的线程的入口函数；

参数 *restrict arg：创建线程的时候传递给线程的参数；

2、线程退出

```
#include <pthread.h>
void pthread_exit(void *value_ptr);          //相当于进程中的 exit 函数；
```

参数为退出线程的状态，通常为 NULL；

3、线程等待

```
#include <pthread.h>
int pthread_join(pthread_t thread, void * *value_ptr); //等待指定的线程退出；
```

参数 thread：等待的线程 ID；

参数 *value_ptr：通常为 NULL；

4、线程清理　（两个函数成对出现，注册满足栈操作，先注册的最后决定是否执行）

```
#include <pthread.h>
void pthread_cleanup_push(void (*routine)(void*), void *arg);
```

参数 (*routine)(void*)：注册的清理函数；

参数　*arg：传递的清理函数的参数；

```
void pthread_cleanup_pop(int execute);
```

//决定注册的函数是否得到执行，参数为真表示执行，为假表示不执行；

5、获取线程 ID

```
#include <pthread.h>
pthread_t pthread_self(void);
```

6、perror 函数

```
#include <stdio.h>
#include <errno.h>
void perror(const char *s);  //将错误信息输出到终端 ，参数为自定义的提示语；
```

三、线程间通信

1、利用信号进行通信

线程间也可以利用信号来进行通信，一般用于捕捉信号，SIGUSR1、SIGUSR2

用 signal 函数进行注册，

用 pthread_kill 函数发送；

#include <signal.h>

int pthread_kill(pthread_t thread, int sig);　　//向指定的 pthread 线程发送指定信号 sig ；

2、利用进程空间资源进行通信

定义全局变量 ；

缺点：线程调度是通过延时去实现，有一定的出错几率；同时在一个进程空间中对资源进行操作容易造成数据紊乱。所以在编写多线程程序时，要记得数据的同步与互斥。

同步：数据的更新；

互斥：独占资源；

四、互斥

1、互斥锁：锁机制：①加锁状态，②解锁状态；

锁机制：对临界区代码进行保护的一种手段，一个线程对一段代码进行加锁，另外一个线程操作的时候也去加锁就会处于一种等待的状态，直到其解锁为止。

锁分为三类：

①快速互斥锁：最常用，对已加锁的临界区代码加锁就会进入等待状态，直到锁被打开；

②检错互斥锁：是快速互斥锁的非阻塞版本；

③递归互斥锁：允许多次加锁；

2、静态创建锁：pthread_mutex mutex = PTHREAD_MUTEX_INITIALIZER; 利用宏定义

//创建快速互斥锁

3、动态创建锁：

int pthread_mutex_init(pthread_mutex_t *mutex, constpthread_mutexattr_t *mutexattr)

参数 1：　描述锁的变量，提前定义即可，定义 pthread_mutex_t 类型的锁

参数 2：　指定锁的类型，　假如指定为 NULL 默认就是快速互斥锁

PTHREAD_MUTEX_INITIALIZER：　创建快速互斥锁

PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP：　创建递归互斥锁

PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP：　创建检错互斥锁

4、加锁：int pthread_mutex_lock(pthread_mutex_t *mutex);

5、解锁：int pthread_mutex_unlock(pthread_mutex_t *mutex);

6、摧毁锁：int pthread_mutex_destroy(pthread_mutex_t *mutex);

7、使用流程：　①确定临界区代码(保护对象)；

②对所有线程都遵循锁机制；

③不用的时候摧毁锁；

五、条件变量：

通常和锁结合在一起，类似于一种通知机制，一个线程由于没有满足一定条件而阻塞休眠，假如满足其他线程条件可将其唤醒。

1、　创建条件变量

a、动态创建：

int pthread_cond_init(pthread_cond_t *restrict cond,const pthread_condattr_t*restrict attr);

参数 1：　pthread_cond_t cond，　描述条件变量

参数 2：　条件变量属性，　设为 NULL

b、静态创建：　pthread_cond_t cond =PTHREAD_COND_INITIALIZER;

2、 由于没有满足一定条件阻塞

int pthread_cond_wait(pthread_cond_t *restrict cond,pthread_mutex_t *restrictmutex);

3、 唤醒条件变量： int pthread_cond_signal(pthread_cond_t *cond);

4、 摧毁条件变量： int pthread_cond_destroy(pthread_cond_t *cond);

5、 使用流程

a、 创建条件变量和互斥锁

b、 线程没有满足条件进程阻塞等待(等待要加锁)

c、 当条件满足， 一个线程发送解除阻塞信号

d、 不用的时候删除条件变量

六、信号灯

等同进程中的信号量操作。

若减一的时候信号量为 0，则程序就会阻塞，等待信号量为正才可以执行；

1、 创建信号灯/信号量

int sem_init(sem_t *sem, int pshared, unsigned int value);

参数 1： 描述信号量的类型的变量， 提前定义即可。

参数 2： 0 用于线程中， 1 用于进程中

参数 3： 信号量的初始值

通过信号量可以对共享资源进行保护。把信号量的值初始化成共享资源的数量，当要使用共享资源时，可以使用 wait 操作进行申请，只有当有剩余资源的时候才能够申请成功，否则要进行等待。而当使用完资源时，需要使用 post 操作进行释放，使资源的计数值加一，如果释放时有人在等待资源，则将其唤醒。

2、 PV 操作

int sem_wait(sem_t *sem); //-1

int sem_post(sem_t *sem); //+1

十三、网络编程

一、 IP 地址的分类

IP 地址是所有的计算机设备连接网络上分配的唯一的地址。所有的通信都依赖于 IP 地址。

IPV4： 是 32 位的数， 点为 4 端， 每段最大为 255。

IPV6： 是 128 位的数， 解决 IPV4 数量不够用的问题。

IP 地址分为：公有地址和私有地址(A、 B、 C、 D 类)

这 4 类的区分是字段表示的含义不同：

A 类：在 IP 地址的四段号码中，第一段为网络号码， 剩下的三段为本地计算机的号码

0.0.0.0-127.255.255.255；

B 类：在 IP 地址的四段号码中，前两段为网络号码，剩下的两段为本地计算机的号码

128.0.0.0-191.255.255.255；

C 类：在 IP 地址的四段号码中，前三段为网络号码，剩下的一段为本地计算机的号码

192.0.0.0-223.255.255.255

D 类： 类 IP 地址在历史上被叫做多播地址(multicast address)， 即组播地址。

224.0.0.0-255.255.255.255

二、 端口号

用以区分同一个 IP 地址提供的不同的网络服务。 "IP 地址+端口号" -- 区分服务；

自定义的 TCP、 UDP 传输的端口号一般大于 1024。

三、 套接字编程

套接字：网络编程传输需要的接口(网络协议、 IP 地址、 端口号)。

分为 3 类：流式套接字(tcp)、 数据报(udp)、 原始套接字；

注：我们主要关心 ISO、 tcp/ip 协议层的传输层： TCP、 UDP

1、 建立套接字：int socket(int domain, int type, int protocol);

参数 domain： 指定网络协议。 (IPV4、 IPV6)；

参数 type： 指定套接字类型；

参数 protocol： 指定为 0；

成功返回操作套接字的描述符；

2、 绑定端口：int bind(int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);

参数 sockfd： 创建套接字成功后的返回值；

参数 my_addr： struct sockaddr 结构描述套接字，

通常定义 struct sockaddr_in，方便我们填充端口号以及 IP 地址。

参数 addrlen： 参数 2 的大小；

3、 设置监听: 设置最大连接数。: int listen(int sockfd, int backlog);

//参数 sockfd ：要监听的套接字 参数 backlog: 请求最大连接数，通常为 5。

4、 等待接收：int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

5、 客户端去请求连接 ：

int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);

参数 sockfd ： 指定的套接字 ；

参数 serv_addr： 接收保存客户端的信息 ；

参数 addrlen：信息长度；

服务器端 ---- 客户端：设置静态 IP：

Linux 系统下：系统 → 管理 → 网络 →设置静态 IP

例：192.168.0.101 ； 255.255.255.0 ； 192.168.0.1；

设置完后： 激活网络、 #service network restart

使用 TCP 编程：

建立套接字 →绑定端口 → 设置监听→ 等待接收 →相互收发