计算机操作系统

实验三

姓名：楚逸飞

学号：2020302878

班级：14012003

# 实验问题

1.Shell编程

在图形化桌面出现之前，和Unix系统交互的唯一方式就是通过Shell提供的文本命令行界面。Shell是个很有用的脚本工具，它多半运行在比编译型语言还高的层次，它能够轻易处理文件与目录之类的对象。 Shell脚本最常用于系统管理工作，它可以把很多命令串在一起，放进一个独立的程序或者脚本中，这样只要执行这个脚本或者程序，便能完成工作。

Linux Shell的种类很多，目前流行的Shell包括ash、bash、ksh、csh、等， 我们重点关注bash Shell，bash Shell是一个增强的bourne Shell，这个是个标准的Unix的Shell，也是Linux上默认的Shell。不同的系统默认的shell是不同的，比如FreeBSD是以csh为其默认的Shell。在Linux系统中，默认的Shell是Bash。

本部分的主要内容：

(1)常见shell命令

(2)shell编程基础与实践

2.进程创建

进程是[操作系统](http://baike.baidu.com/view/880.htm" \t "http://baike.baidu.com/view/_blank)结构的基础；是一次程序的执行；是一个程序及其数据在处理机上顺序执行时所发生的活动。进程是60年代初首先由[麻省理工学院](http://baike.baidu.com/view/1935.htm" \t "http://baike.baidu.com/view/_blank)的[MULTICS系统](http://baike.baidu.com/view/8939006.htm" \t "http://baike.baidu.com/view/_blank)和IBM公司的[CTSS](http://baike.baidu.com/view/3394929.htm" \t "http://baike.baidu.com/view/_blank)/360系统引入的。进程是一个具有独立功能的程序关于某个数据集合的一次运行活动。它可以申请和拥有系统资源，是一个动态的概念，是一个活动的实体。它不只是程序的[代码](http://baike.baidu.com/view/41.htm" \t "http://baike.baidu.com/view/_blank)，还包括当前的活动，通过[程序计数器](http://baike.baidu.com/view/178145.htm" \t "http://baike.baidu.com/view/_blank)的值和处理[寄存器](http://baike.baidu.com/view/6159.htm" \t "http://baike.baidu.com/view/_blank)的内容来表示。

* 定义

狭义定义：进程就是一段程序的执行过程。

广义定义：进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。它是[操作系统](http://baike.baidu.com/subview/880/4940471.htm" \t "http://baike.baidu.com/view/_blank)动态执行的[基本单元](http://baike.baidu.com/view/693012.htm" \t "http://baike.baidu.com/view/_blank)，在传统的[操作系统](http://baike.baidu.com/view/880.htm" \t "http://baike.baidu.com/view/_blank)中，进程既是基本的[分配单元](http://baike.baidu.com/view/954991.htm" \t "http://baike.baidu.com/view/_blank)，也是基本的执行单元。

进程是一个实体。每一个进程都有它自己的地址空间，一般情况下，包括[文本](http://baike.baidu.com/view/300107.htm" \t "http://baike.baidu.com/view/_blank)区域（text region）、数据区域（data region）和[堆栈](http://baike.baidu.com/view/93201.htm" \t "http://baike.baidu.com/view/_blank)（stack region）。文本区域存储处理器执行的代码；数据区域存储变量和进程执行期间使用的动态分配的内存；堆栈区域存储着活动过程调用的指令和本地变量。第二，进程是一个“执行中的程序”。程序是一个没有生命的实体，只有[处理](http://baike.baidu.com/view/989420.htm" \t "http://baike.baidu.com/view/_blank)器赋予程序生命时，它才能成为一个活动的实体，我们称其为[进程](http://baike.baidu.com/view/19746.htm" \t "http://baike.baidu.com/view/_blank)。

进程是操作系统中最基本、重要的概念。是多道程序系统出现后，为了刻画系统内部出现的动态情况，描述系统内部各道程序的活动规律引进的一个概念,所有多道程序设计操作系统都建立在进程的基础上。

进程和线程的一个简单解释

<http://www.ruanyifeng.com/blog/2013/04/processes_and_threads.html>

下面的链接详细介绍了Linux中fork()函数

<http://blog.csdn.net/jason314/article/details/5640969>

gcc编译器的常用命令

<https://www.jb51.net/article/180825.htm>

关于VI/VIM使用的链接

<https://www.runoob.com/linux/linux-vim.html>

# 二、实验主要内容

1.shell：

（1） 查阅相关资料，使用常见的shell命令，包括但不限于：

目录操作命令 —— pwd、cd、ls、mkdir

文件操作命令 —— touch、file、cp、rm、mv、find

归档及压缩命令 —— tar

输入输出与文件管理 —— echo、read、cat、grep、|（管道）、输入输出重定向

请给出你对这些命令的理解以及使用过程(附截图)

1. 查阅shell编程相关资料，编写shell脚本并执行。请提交脚本代码以及代码解释，并附实验过程截图。

1）实现从键盘读取两个数，并比较两个数大小，并打印结果。

提示：echo，read，if

2）实现读取文件的每一行

提示：cat，while，read

3）把当前目录（包含子目录）下所有后缀为“.txt”的文件后缀变更为“.h”

提示：find，for循环

4）编写make文件实现编译c或c++代码。

提示：make,gcc

2.进程创建

1. 学习进程的基础知识，了解Linux系统中进程创建的基本原理。
2. 熟悉进程的创建、控制、执行和终止等系统调用函数。
3. 安装vim编辑器,使用vim编制一段程序，使用系统调用fork()创建两个子进程，在此程序运行时，系统中就有一个父进程和两个子进程在活动。让每一个进程在屏幕上显示一个字符：父进程显示字符a，两个子进程分别显示字符b和子符c。试观察、记录并分析屏幕上进程调度和并发执行的情况。若在程序中使用系统调用nice()来改变各进程的优先级，会出现什么现象？

4）提交源程序清单，并附加流程图与注释。

并回答以下问题：

1. 系统调用fork()是怎样创建进程的？

2）当首次调用新创建的子进程时，其入口在哪里？

3）分析进程调度和进程并发执行的关系？

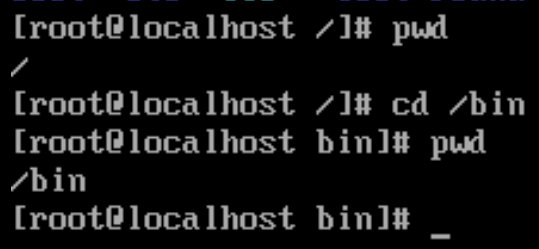
# 三、实验过程

**1.shell编程**

（1）查阅相关资料，使用常见的shell命令，请给出你对这些命令的理解以及使 用过程(附截图)

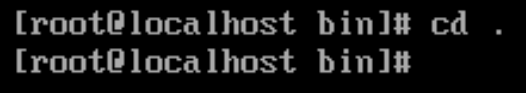
1）目录操作命令(pwd、cd、ls、mkdir)

①pwd：查看当前所在目录

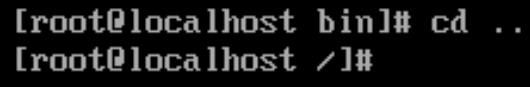


②cd：跳转到目标目录

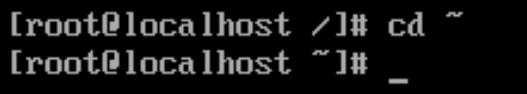
· user@computer: ~$ cd . # 到本目录



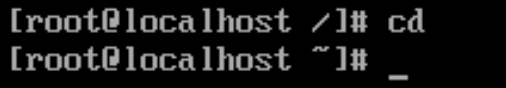
· user@computer: ~$ cd .. # 到上一级



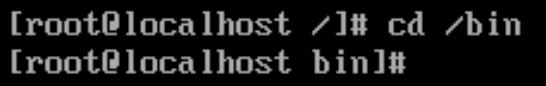
· user@computer: ~$ cd ~ # 到当前用户家目录



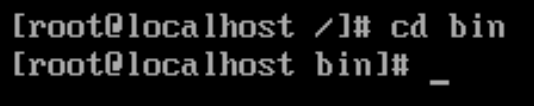
· user@computer: ~$ cd # 到当前用户家目录



· user@computer: ~$ cd /etc # 到绝对路径为/etc/的目录

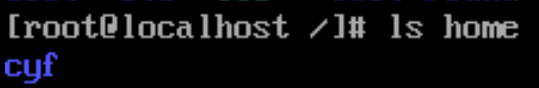


· user@computer: ~$ cd filename # 到当前目录中的filename目录

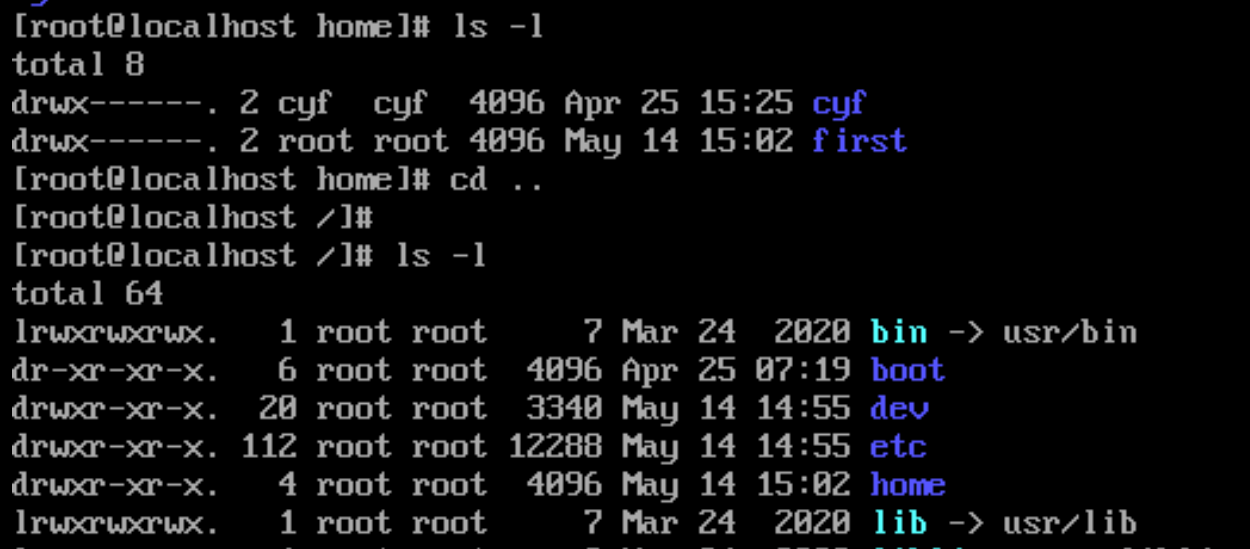


③ls：列出目录下的文件(可以指定目录，若不指定则列出当前目录下的文件)



(指定目录)

· -l ：以长格式显示

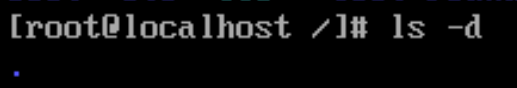


(文件较多，仅截图展示了一部分示意)

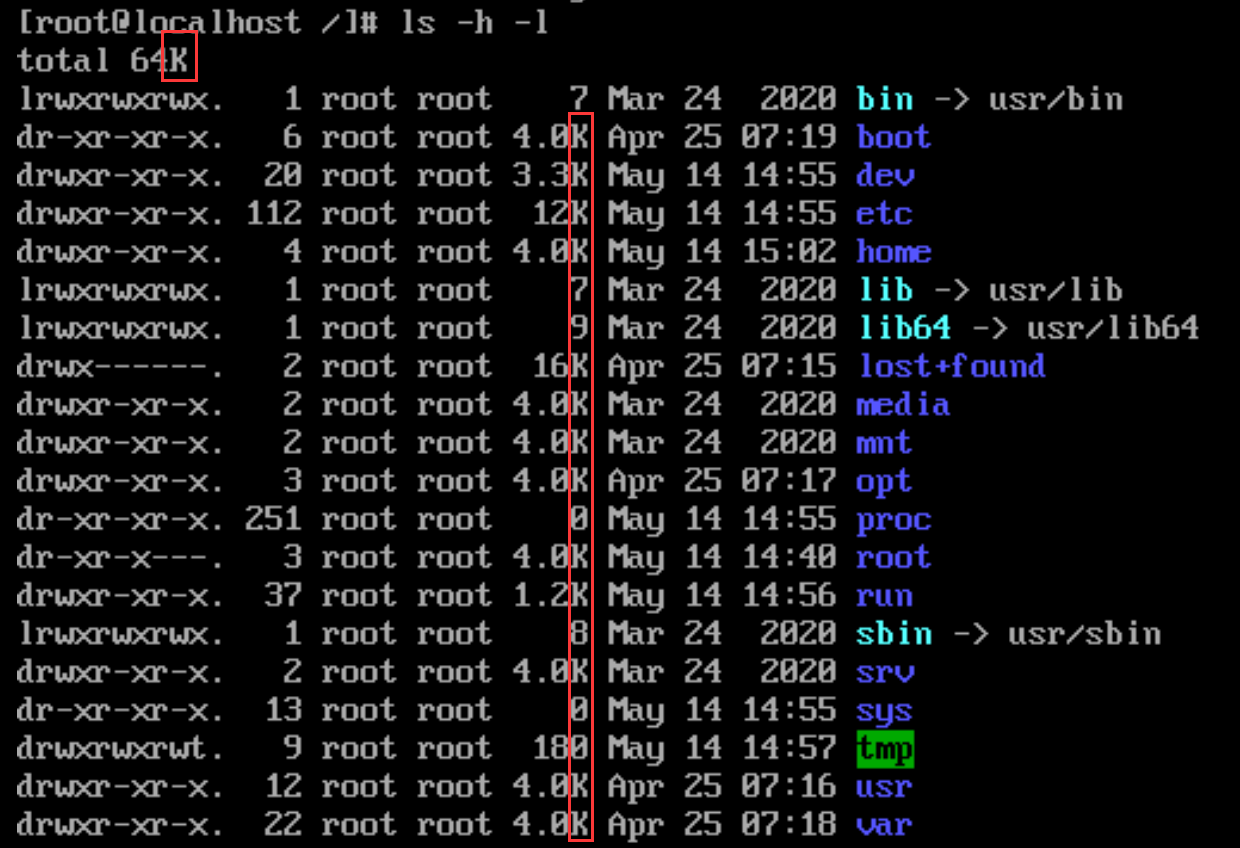
· -a：显示所有子目录和文件的信息，包括隐藏文件



· -d：显示目录本身的属性



· -h：大小以k为单位显示



④mkdir：创建文件夹

· mkdir [-p] [path]name

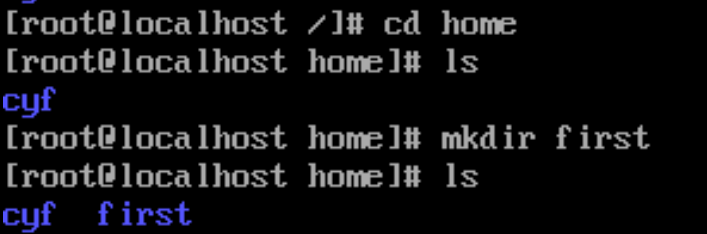
· -p 创建多级目录

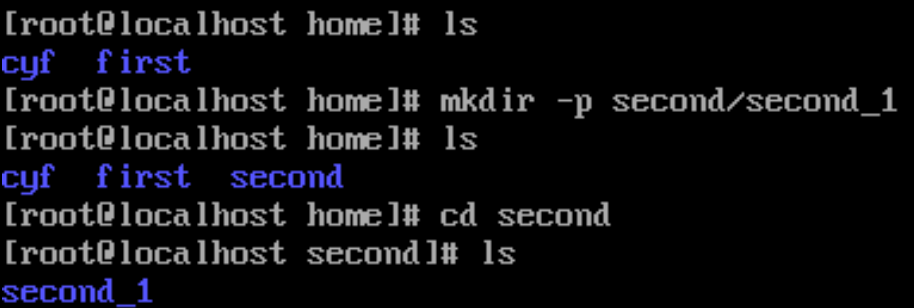
· user@computer: ~$ mkdir mydir

· user@computer: ~$ mkdir mydir0/mydir1 # 若mkdir0不存在，则 报错

· user@computer: ~$ mkdir -p mydir0/mydir1 # 若mkdir0不存在， 则先创建mydir0再创建mydir1

如以下两图所示：



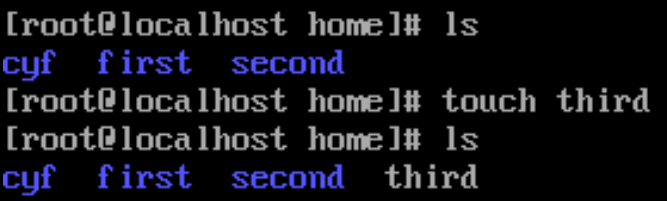


1. 文件操作命令(touch、file、cp、rm、mv、find)

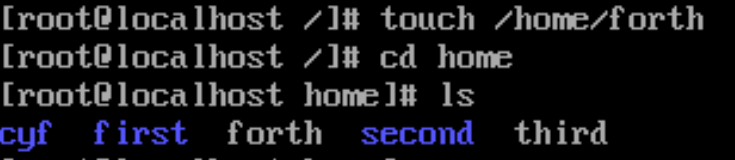
①touch：新建空文件，或更新文件时间标记

· touch path/filename

· user@computer: ~$ touch myfile # 在当前目录下新建myfile

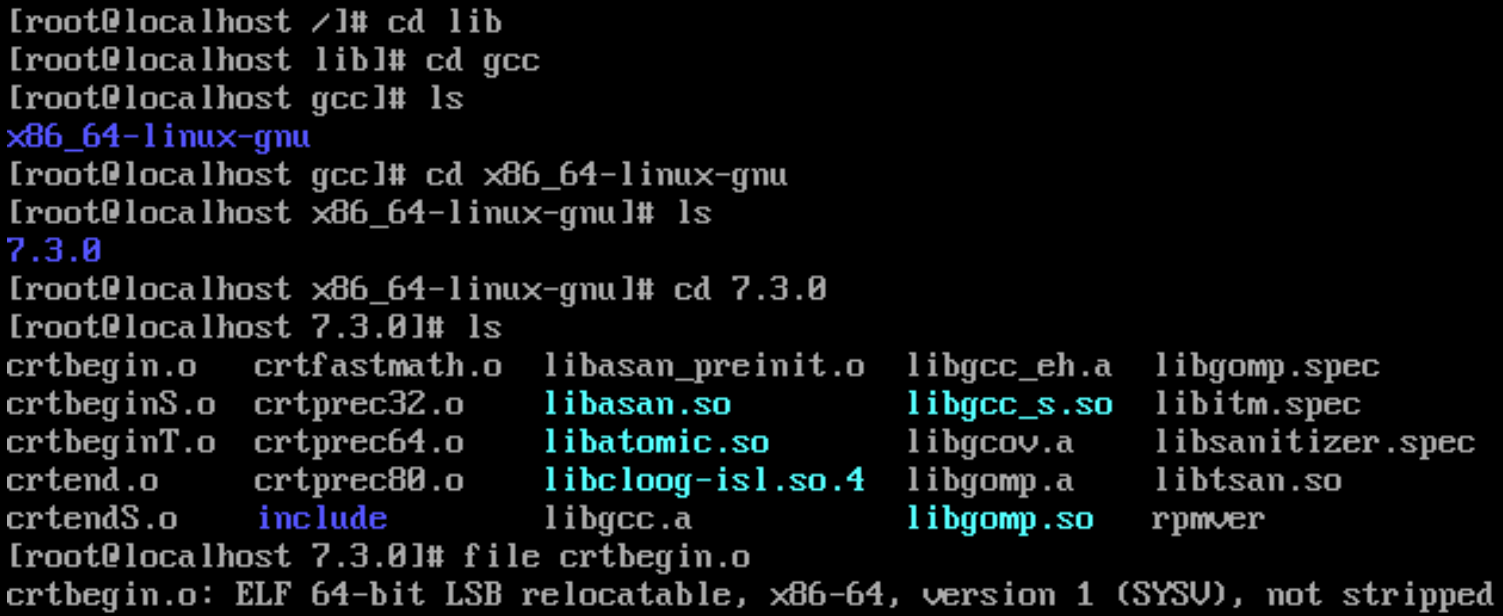


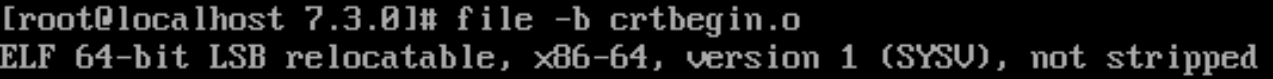
· user@computer: ~$ touch /user/local/txt/myfile # 在指定目录 下新建myfile



②file：辨识文件的类型

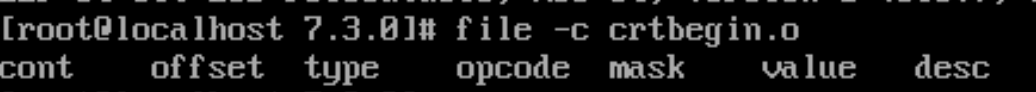
· -b 　列出辨识结果时，不显示文件名称。



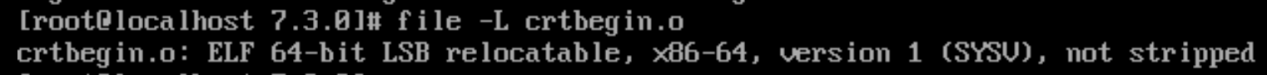


(对比)

· -c 　详细显示指令执行过程，便于排错或分析程序执行的情形。

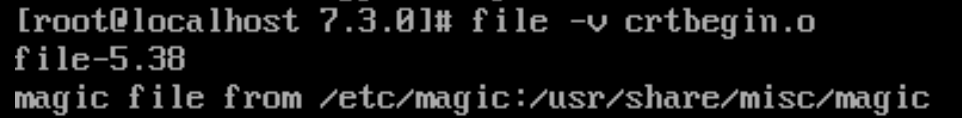


· -L 　直接显示符号连接所指向的文件的类别。



· -m<魔法数字文件> 　指定魔法数字文件。

· -v 　显示版本信息。



· -z 　尝试去解读压缩文件的内容

③cp：复制文件或目录

· cp [选项] source object

· 常用命令选项程度

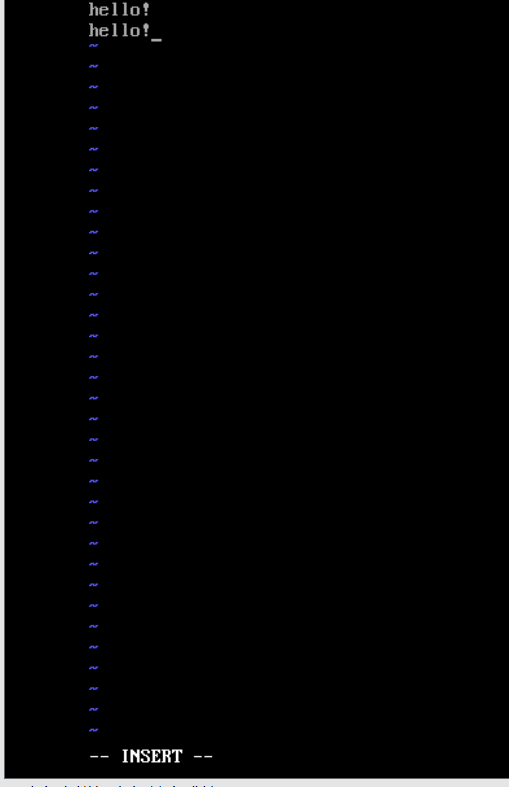
· -r：递归复制整个目录树

· -f: 强制覆盖同名文件

把home下的文件复制到newhome下

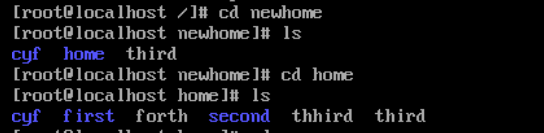
home下原本的文件和目录

Newhome下的third文件里原本的内容（左）以及home下third里原本的内容（右）





接下里来再查看newhome下的内容：

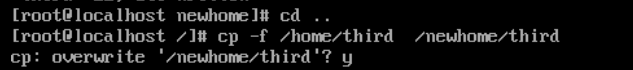


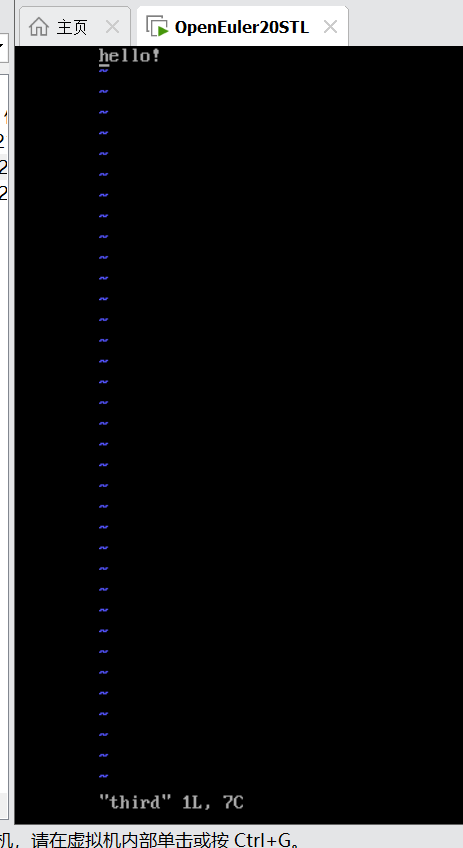
可以看到home已经被复制到了newhome里(复制文件夹的时候一定要用到-r)

接下来是-f的作用：



-f应该是覆盖内容而不给出提示，但是再openeuler系统中还是会有提示。输入y后即可成功覆盖



（覆盖后的结果）

④rm：删除文件或目录

· rm [选项] [file|dir]

· 常用命令选项

· -r：递归删除整个目录树

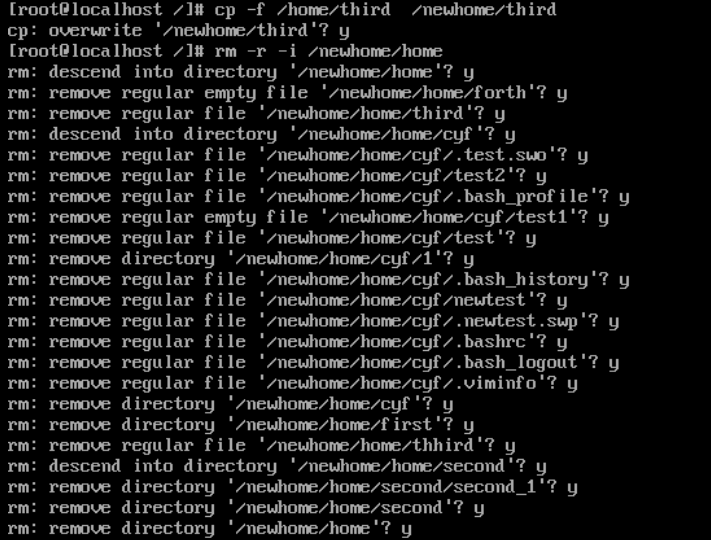
· -f: 即使文件的属性设置为只读，亦直接删除，无需逐一确认

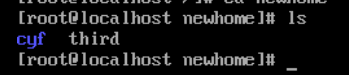
· -i 删除前逐一询问确认

· user@computer: ~$ rm ~/file

· user@computer: ~$ rm -r ~/directory # 删除目录时，应该加上 -r选项，否则会失败

删除在之前复制到newhome下的home目录：





可以看到删除前有逐一询问并且最终递归删除了整个目录

⑤mv：移动文件或目录; 若目标位置与源位置相同，则相当于改名

· mv [选项] 源文件或目录 目标文件或目录

· **-b**: 当目标文件或目录存在时，在执行覆盖前，会为其创建一个备份。

· **-i**: 如果指定移动的源目录或文件与目标的目录或文件同名，则会先询问是否覆 盖旧文件，输入 y 表示直接覆盖，输入 n 表示取消该操作。

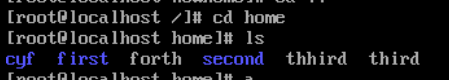
· **-f**: 如果指定移动的源目录或文件与目标的目录或文件同名，不会询问，直接 覆盖旧文件。

· **-n**: 不要覆盖任何已存在的文件或目录。

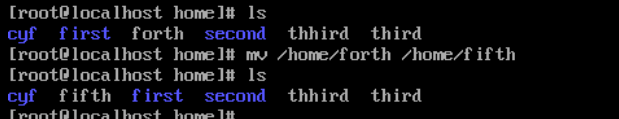
· **-u**：当源文件比目标文件新或者目标文件不存在时，才执行移动操作

· user@computer: ~$ mv /root/pic/\*.png /usr/local/share/pic # 将/root/pic目录下的所有后缀名为”\*.png”的文件移动到 /usr/local/share/pic目录下

· user@computer: ~$ mv /root/pic/kpic.png /root/pic/life.png # 把kpic.png文件改名为life.png

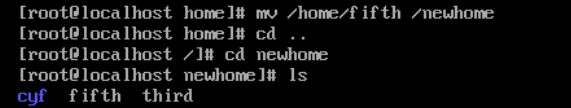


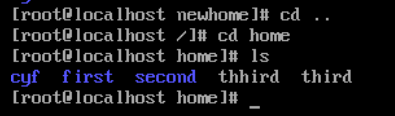
移动到相同的位置就相当于改名



将fifth文件从home移动到newhome





（可以看到原本home中的fifth没有了）

⑥find：Linux find 命令用来在指定目录下查找文件。任何位于参数之 前的字符串都将被视为欲查找的目录名。如果使用该命令时，不设置任 何参数，则 find 命令将在当前目录下查找子目录与文件。并且将查找 到的子目录和文件全部进行显示

· find path -option [ -print ] [ -exec -ok command ] {} \;

· 常用查找条件

· -name：按文件名称查找

· -user：按文件属主查找

· -type：按文件类型查找 取值：d/c/b/p/f/i/s

· -size: 按大小查找 取值：[+|-] n 单位为字节或块

· user@computer: ~$ find /etc -name “p\*” -type f # 查找etc下以 p开头的文件

· user@computer: ~$ find /etc -user root # 查找etc下属主为 root的文件或目录

· user@computer: ~$ find /etc -size +2048 -a -size -20480 # 在 /etc目录下查找大于1MB小于10MB的文件

· -exec: 对查找到的结果进行操作

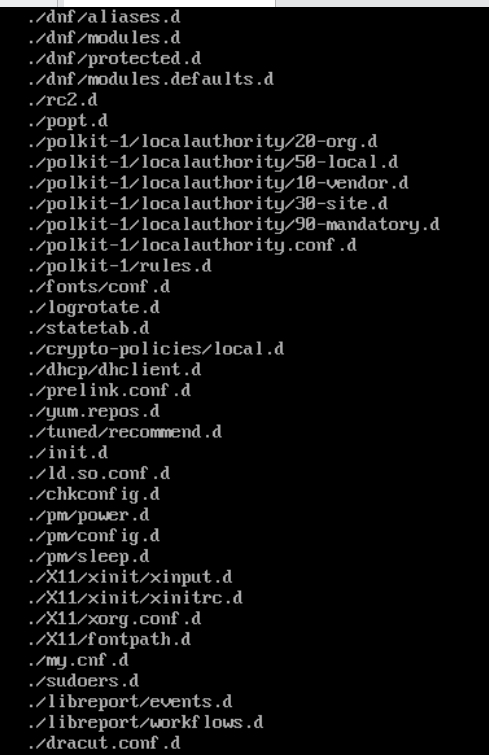
· find [条件] -exec shell命令 {} \; # {}表示找到的结果集

· user@computer: ~$ find / -type f –size 0 –exec ls –l {} \; # 查看文件长度为0的普通文件，并列出完整路径

查询etc下所有以后缀.d结尾的文件：



结果：



查找当前目录及其子目录下的所有普通文件:



结果：

（内容太多 不一一展示）

1. 归档以及压缩命令(tar) 是用来建立，还原备份文件的工具程序，它可以加入，解开备份文件内的文件

-c: 建立压缩档案

-x：解压

-t：查看内容

-r：向压缩归档文件末尾追加文件

-u：更新原压缩包中的文件

-f: 使用档案名字这个参数是最后一个参数，后面只能接档案名（-f是必须的）

-z：有gzip属性的

-j：有bz2属性的

-J：具有xz属性的

-Z：有compress属性的

-v：显示所有过程

-O：将文件解开到标准输出

(1) \*.tar 用 tar –xvf 解压

(2) \*.gz 用 gzip -d或者gunzip 解压

(3) \*.tar.gz和\*.tgz 用 tar –xzf 解压

(4) \*.bz2 用 bzip2 -d或者用bunzip2 解压

(5) \*.tar.bz2用tar –xjf 解压

(6) \*.Z 用 uncompress 解压

(7) \*.tar.Z 用tar –xZf 解压

(8) \*.rar 用 unrar e解压

(9) \*.zip 用 unzip 解压

(10) \*.xz 用 xz -d 解压

(11) \*.tar.xz 用 tar -zJf 解压

1. 输入输出以及文件管理(echo、read、cat、grep、|（管道）、输入输出重 定 向)

①echo：echo命令的功能是在显示器上显示一段文字，一般起到一个提示的 作用

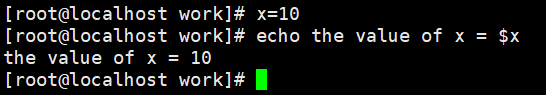
语法：echo [选项] [字符串]

‘-e‘扮演了转义字符反斜线的翻译器

输出输入的字符串：



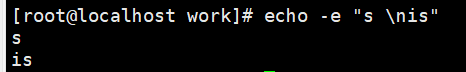
输出一个变量的值：



\b：会删除字符间所有空格



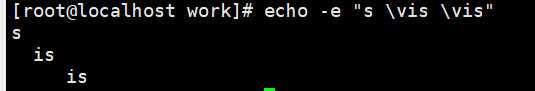
\n：换行

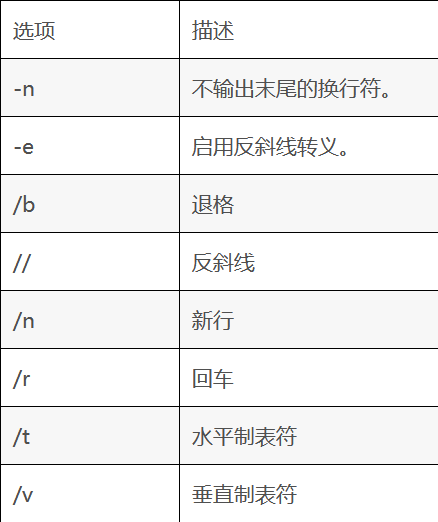


\t：空格之间加水平制表符



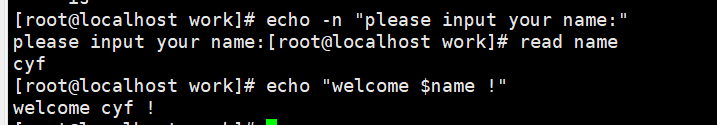
\v：加垂直制表符



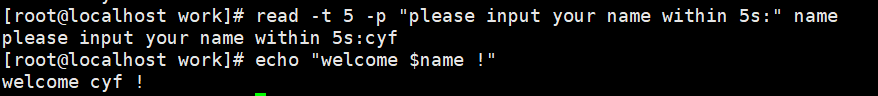


重新定向符来输出到一个新的文件而不是标准输出

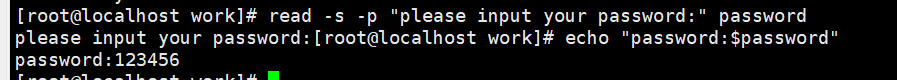
②read：read命令用来倾听标准输入或文件输入，把信息存放到变量中



设置等待时间：



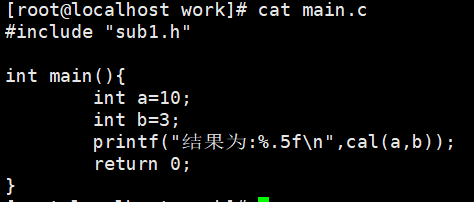
输入密码：（-s选项隐藏内容）



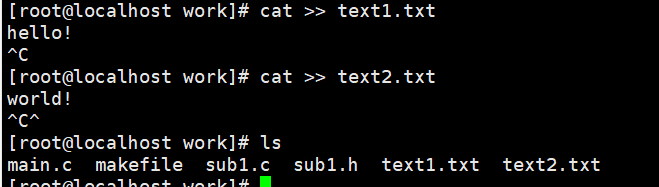
③cat：cat [选项] [文件]

主要有三个功能：

第一个是一次显示整个文件：

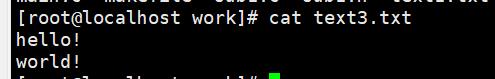


第二个是从键盘新创建一个文件：

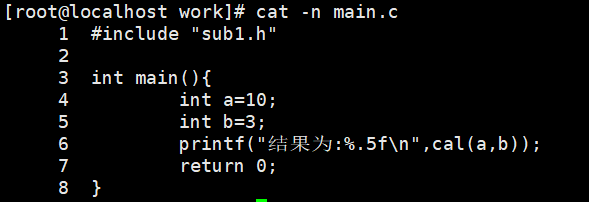


第三个是将几个文件合并为一个文件：

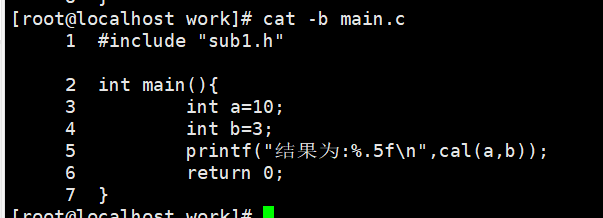


（查看内容是前两个文件内容的顺序拼接）

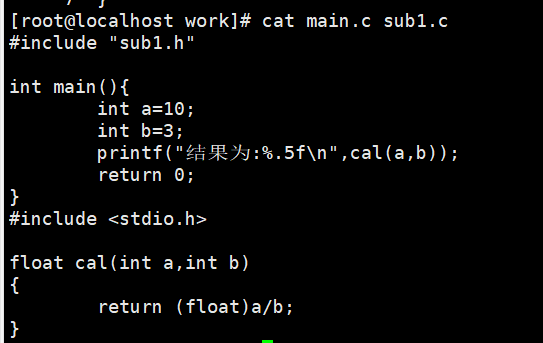
-n：查看文件内容并且从1开始输出行号



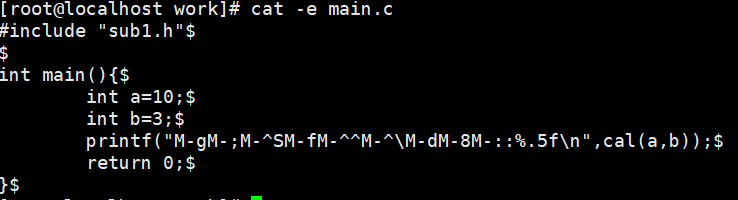
-b：与-n类似，但是 空白行不变号



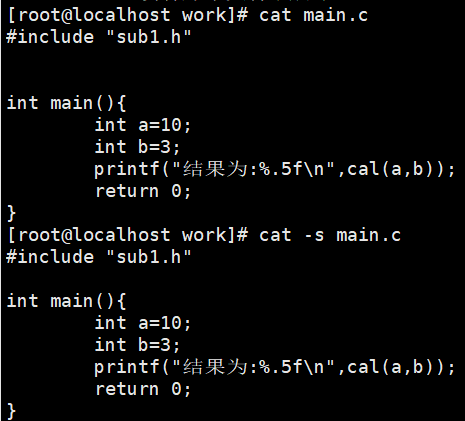
同时显示两个文件的内容：



-e：在输出内容每一行后面加一个符号



-s：当连续遇到两行以上的空白行就替换为一行空白行



④grep：grep命令是一种强大的文本搜索工具，它能使用[正则表达式](https://so.csdn.net/so/search?q=%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_40797605/article/details/_blank)搜索文本，并把匹配的行打印出来

主要参数：

-c：只输出匹配行的计数。

-i：不区分大小写。

-h：查询多文件时不显示文件名。

-l：查询多文件时只输出包含匹配字符的文件名。

-n：显示匹配行及 行号。

-s：不显示不存在或无匹配文本的错误信息。

-v：显示不包含匹配文本的所有行。

--color=auto ：可以将找到的关键词部分加上颜色的显示。

pattern正则表达式主要参数:

\： 忽略正则表达式中特殊字符的原有含义。

^：匹配正则表达式的开始行。

$: 匹配正则表达式的结束行。

\<：从匹配正则表达 式的行开始。

\>：到匹配正则表达式的行结束。

[ ]：单个字符，如[A]即A符合要求 。

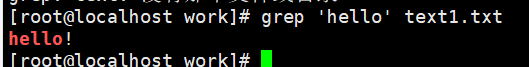
[ - ]：范围，如[A-Z]，即A、B、C一直到Z都符合要求 。

.：所有的单个字符。

\*：所有字符，长度可以为0。

简单的使用例子：

显示所有以test1.txt文件中包含hello的行











⑤|：命令格式

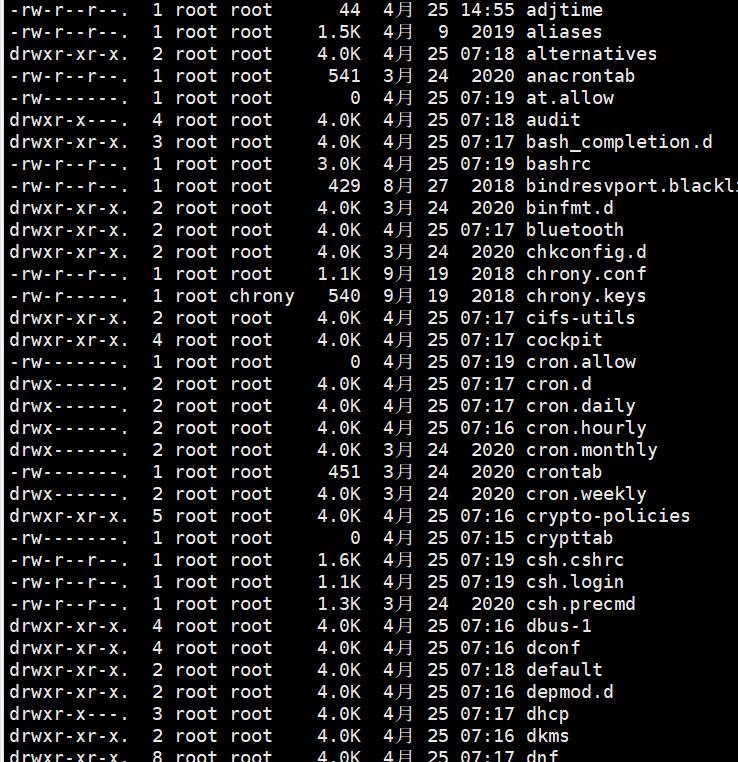
命令1 | 命令2

（将命令1正确输出的结果作为命令2的操作对象）

例如：

用翻页方式显示etc下的文件





⑥输入输出重定向：重定向就是把内容直接放到文件里而不是显示在控制台上

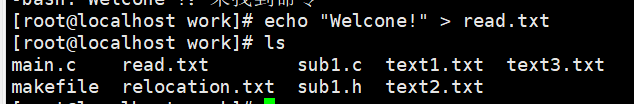
因此重定向的完整写法是：

1> 文件不存在则创建文件，文件存在则清空后将标准信息定向到该文件

1>> 文件不存在则创建文件，文件存在则将标准信息追加到该文件

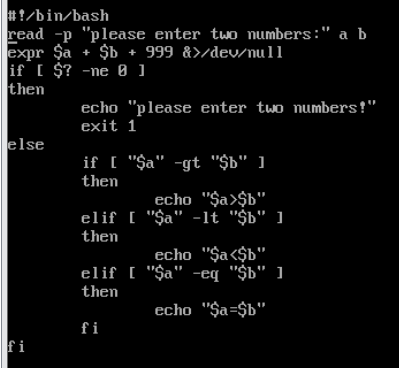
2> 文件不存在则创建文件文件存在则清空后将错误信息定向到该文件

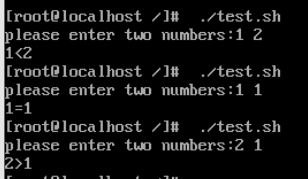
2>> 文件不存在则创建文件，文件存在则将错误信息追加到该文件



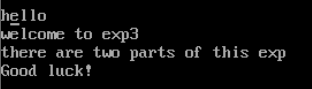
（2）查阅shell编程相关资料，编写shell脚本并执行。请提交脚本代码以及代 码解释，并附实验过程截图

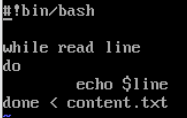
1）实现从键盘读取两个数，并比较两个数大小，并打印结果

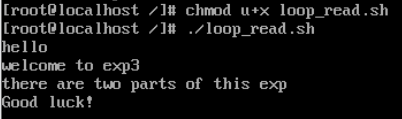
(代码展示，line2-line3判断是不是数字)

(运行结果展示)

2）实现读取文件的每一行

要读取的文件内容：  


(代码展示)

(结果展示)

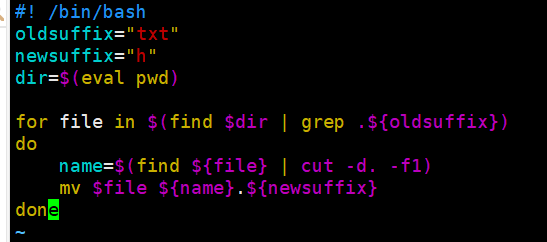
3）把当前目录（包含子目录）下所有后缀为“.txt”的文件后缀变更为“.h”





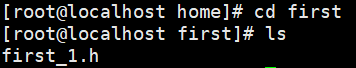
(两个需要被更改的txt文件，home下的fifth.txt 以及 home下first下的first\_1.txt)

(增加权限)

（Xshell上）

更改后的结果：

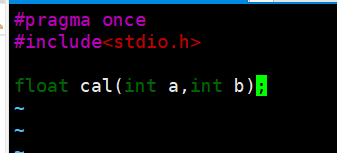


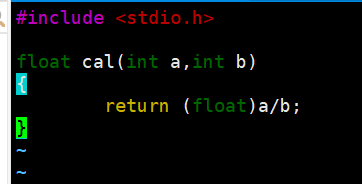


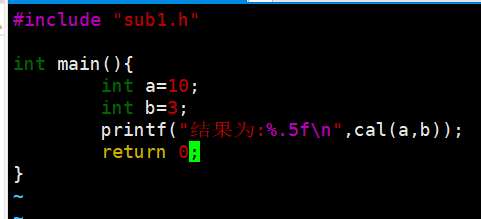
4）编写make文件实现编译c或c++代码

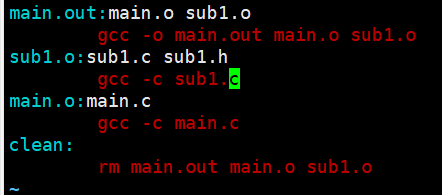
共编写以下四个文件：



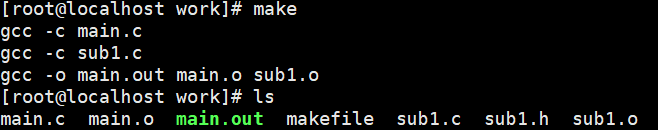
(sub1.h)

(sub1.c)

(main.c)

(makefile)

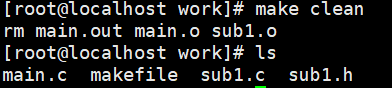
直接输入make即可执行：



执行main.out



删除文件：



**2.进程创建**

1）学习进程的基础知识，了解Linux系统中进程创建的基本原理。

①进程的概念:

进程（Process）是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。在早期面向进程设计的计算机结构中，进程是程序的基本执行实体；在当代面向线程设计的计算机结构中，进程是线程的容器。程序是指令、数据及其组织形式的描述，进程是程序的实体。

②进程的创建和构造：

进程创建是操作系统执行程序的需要或者用户或进程要求创建一个新的进程。进程创建首先是在进程表中为进程建立一个进程控制块PCB，采用fork()系统调用将复制执行进程的PCB块，U区和内存图像到新的进程。 主要内容包括：进程创建原语、fork()系统调用

在Linux系统中，除了系统启动之后的第一个进程由系统来创建，其余的进程都必须由已存在的进程来创建，新创建的进程叫做子进程，而创建子进程的进程叫做父进程。那个在系统启动及完成初始化之后，Linux自动创建的进程叫做根进程。根进程是Linux中所有进程的祖宗，其余进程都是根进程的子孙。具有同一个父进程的进程叫做兄弟进程。

子进程的创建

在Linux中，父进程以分裂的方式来创建子进程，创建一个子进程的系统调用叫做fork()。系统调用fork()为了在一个进程中分裂出子进程，Linux提供了一个系统调用fork()。这里所说的分裂，实际上是一种复制。因为在系统中表示一个进程的实体是进程控制块，创建新进程的主要工作就是要创建一个新控制块，而创建一个新控制块最简单的方法就是复制。

2）熟悉进程的创建、控制、执行和终止等系统调用函数。

在Linux中，我们通常使用fork函数来为一个已经存在的进程创建一个新进程。而这个新创建出来的进程被称为原进程的子进程，原进程被称为该进程的父进程。

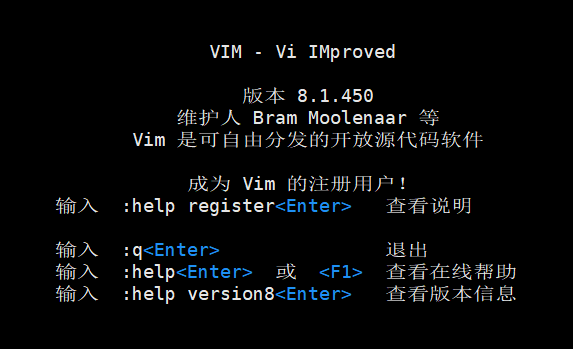
不止可以通过fork来创建子进程，vfork也同样是用来创建子进程的系统调用函数。

exit函数：stauts定义了进程的终止状态，由用户自己传递，父进程可以通过wait来获取该值。

\_exit函数：exit是库函数，\_exit是系统调用函数，而库函数内部封装了系统调用。 也就是说，调用exit函数最终也会调用\_exit来使进程退出，只不过在其调用\_exit之前，还会做一些其他的事情。

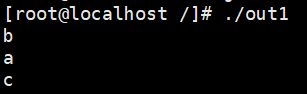
wait、exec函数簇。

1. 安装vim编辑器,使用vim编制一段程序，使用系统调用fork()创建两个子进程，在此程序运行时，系统中就有一个父进程和两个子进程在活动。让每一个进程在屏幕上显示一个字符：父进程显示字符a，两个子进程分别显示字符b和子符c。试观察、记录并分析屏幕上进程调度和并发执行的情况。若在程序中使用系统调用nice()来改变各进程的优先级，会出现什么现象？





结果：

（bac）

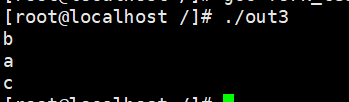
使用nice：



结果：

（abc）



（bac）

1. 提交源程序清单，并附加流程图与注释。

源程序：

①

#include<unistd.h>

#include<sys/types.h>

#include<sys/resource.h>

#include<sys/wait.h>

#include<stdlib.h>

int main()

{

pid\_t pid1,pid2;//定义两个进程变量

pid1=fork();//pid1开辟进程

if(pid1 < 0)

{//pid1小于零就报错

printf("error!\n");

}

else if(pid1 == 0)

{//pid1是0就输出b

printf("b\n");

exit(1);

}

else

{//pid1大于零就为pid2开辟进程

pid2=fork();

if(pid2 < 0)

{//pid2小于零就报错

printf("error\n");

}

else if(pid2 == 0)

{//pid2为零就输出c

printf("c\n");

exit(1);

}

else

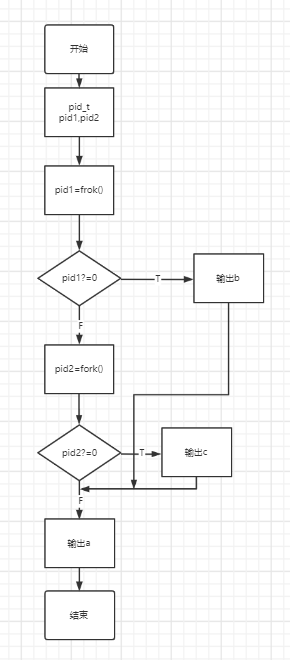
{//pid2大于零就输出a

printf("a\n");

}

}

}

(流程图中只能判断是否等于的关，所以就默认忽略了pid<0的情况，直接分为pid>0以及pid=0的情况)

②

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/resource.h>

#include<sys/wait.h>

#include<stdlib.h>

int main()

{

pid\_t pid1,pid2;//申请两个进程变量

int oldpri1,newpri1;//因为要改变b的优先级，故申请两个整型数表示新旧优先级

pid1=fork();//

if(pid1<0)

{//pid1小于零就报错

printf("error!\n");

}

else if(pid1 == 0)

{//pid1为零就执行nice，并且获取新旧优先级的值，仅用于查看，并且输出b

oldpri1 = getpriority(PRIO\_PROCESS,getpid());

//printf("old priority is:%d\n",oldpri1);

newpri1 = nice(2);

//printf("new priority is:%d\n",newpri1);

printf("b");

exit(1);

}

else

{//pid1大于零就为pid2开辟进程

pid2=fork();

if(pid2<0)

{//pid2小于零就报错

printf("error\n");

}

else if(pid2==0)

{//pid2等于零就输出c

printf("c");

}

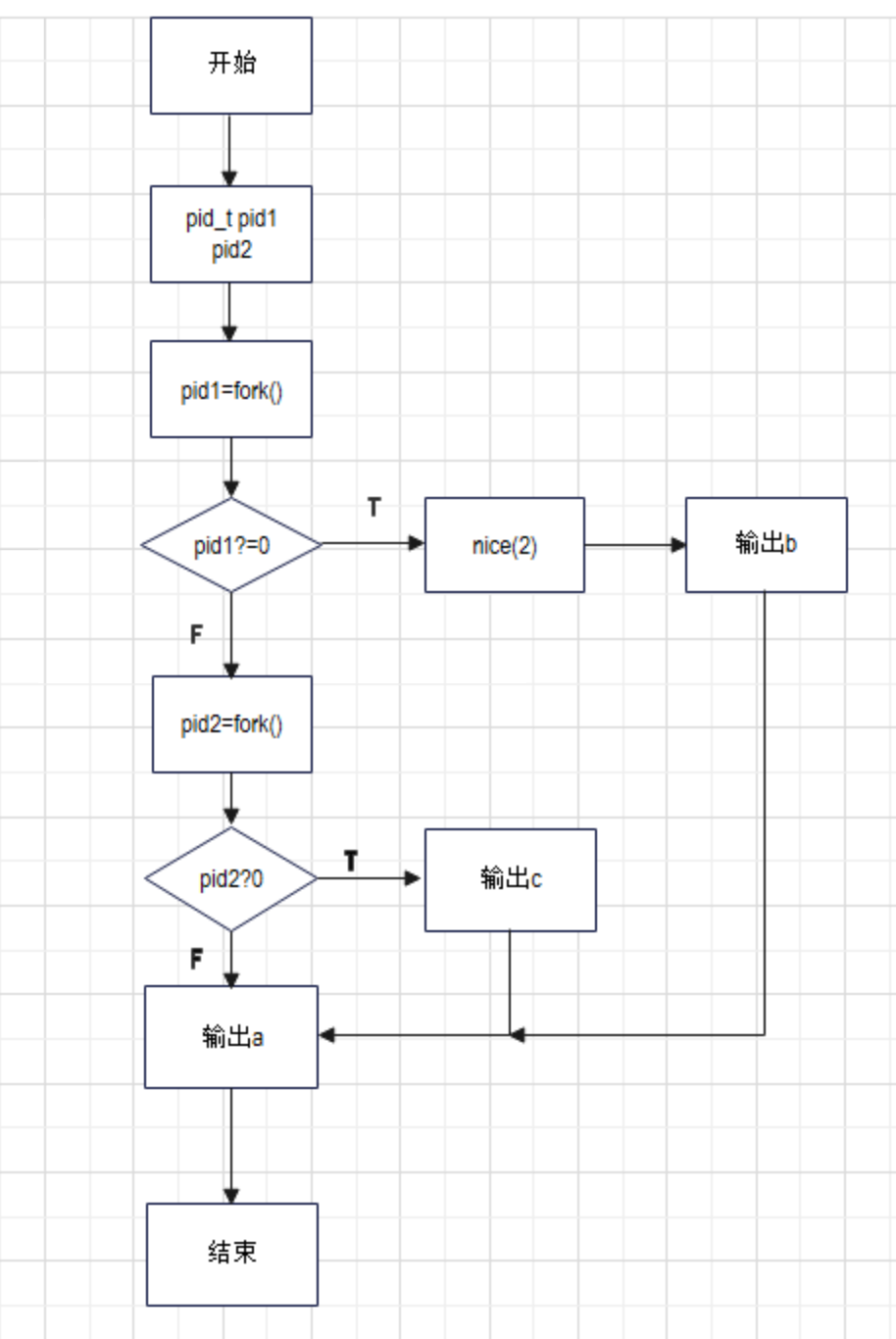
else

//pid2大于零就输出a

printf("a");

}

}

(流程图中只能判断是否等于的关，所以就默认忽略了pid<0的情况，直接分为pid>0以及pid=0的情况)

③

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/resource.h>

#include<sys/wait.h>

#include<stdlib.h>

int main()

{

pid\_t pid1,pid2;

int oldpri1,oldpri2,newpri1,newpri2;

pid1=fork();

if(pid1<0)

{

printf("error!\n");

}

else if(pid1==0)

{

oldpri1 = getpriority(PRIO\_PROCESS,getpid());

//printf("b old pri is :%d\n",oldpri1);

newpri1=nice(2);

//printf("bj new pri is :%d\n",newpri1);

printf("b\n");

exit(1);

}

else

{

pid2=fork();

if(pid2<0)

{

printf("error\n");

}

else if(pid2 == 0)

{

oldpri2 = getpriority(PRIO\_PROCESS,getpid());

//printf("c old pri is :%d\n",oldpri2);

newpri2=nice(1);

//printf("c new pri is :%d\n",newpri2);

printf("c\n");

exit(1);

}

else

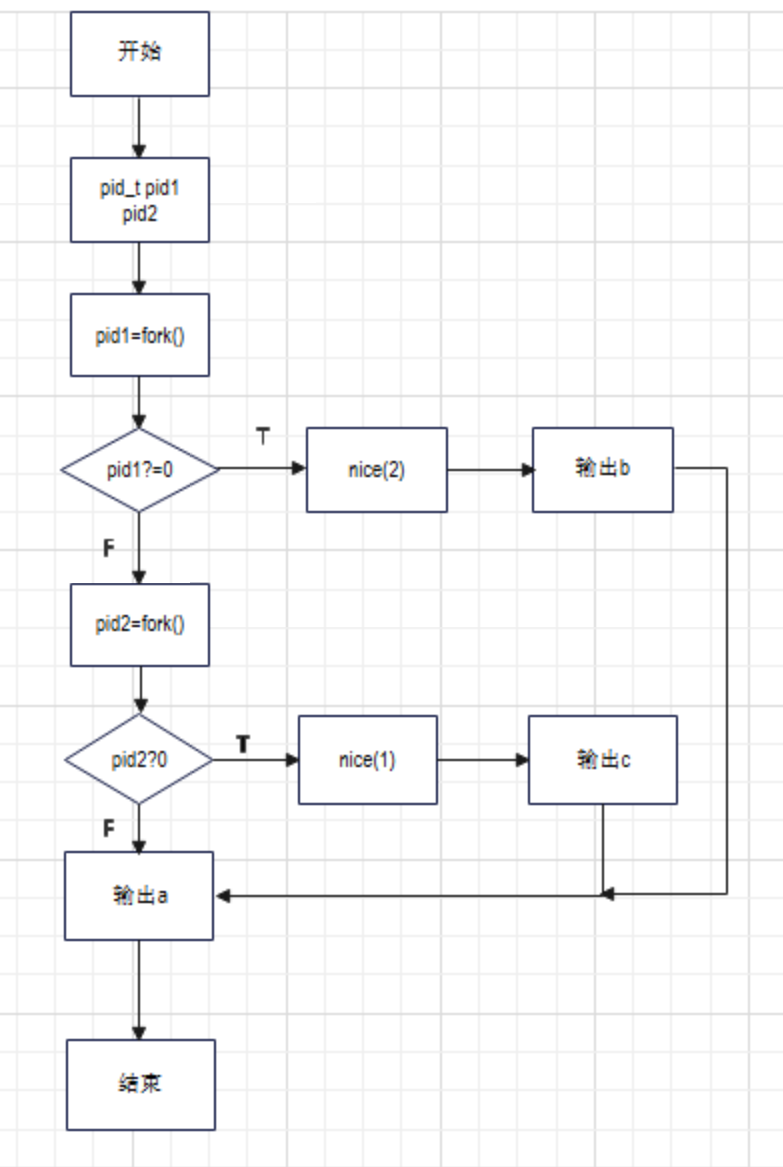
{

printf("a\n");

}

}

}

 (流程图中只能判断是否等于的关，所以就默认忽略了pid<0的情况，直接分为pid>0以及pid=0的情况)

并回答以下问题：

1. 系统调用fork()是怎样创建进程的？

fork（）函数通过系统调用创建一个与原来进程几乎完全相同的进程，也就 是两个进程可以做完全相同的事，但如果初始参数或者传入的变量不同，两 个进程也可以做不同的事。 一个进程调用fork（）函数后，系统先给新的 进程分配资源，例如存储数据和代码的空间。 然后把原来的进程的所有值 都复制到新的新进程中，只有少数值与原来的进程的值不同。 相当于克隆 了一个自己。

1.申请空白PCB（进程控制块）；

2.为新进程分配资源；

3.初始化PCB；

4.就新进程插入就绪队列；

1. 当首次调用新创建的子进程时，其入口在哪里？

fork()函数被调用一次，但返回两次；两次返回区别在于：子程序返回值是 0，而父进程返回值是子进程的ID。子进程和父进程运行相同的代码，但是 有自己的数据空间

见代码及注释：

pid\_t new\_pid;

new\_pid = fork();

//父进程在这里继续运行，子进程也从这里开始

switch(new\_pid){

case -1 : /\* error \*/

break;

case 0 : /\* this is child \*/

/\* 子进程会运行这里的代码 \*/

break;

default: /\* this is parent \*/

break;

}

//如果这后面还有代码，父进程和子进程均会运行

3）分析进程调度和进程并发执行的关系？

并发当有多个线程在操作时,如果系统只有一个CPU,则它根本不可能真正同时进行一个以上的线程，它只能把CPU运行时间划分成若干个时间段,再将时间 段分配给各个线程执行，在一个时间段的线程代码运行时，其它线程处于挂起状。.这种方式称之为并发

# 四、分析结果

本次实验主要研究了Shell编程以及进程创建的问题，由于对linux（openeuler）的基础不够扎实，所以在实验过程中也遇到了很多问题。尤其是编写shell程序以及使用vim编写进程调度的程序的时候遇到了很多困难。但是也通过查阅相关的资料以及向同学请教获得了及时解决，本次实验让我对各种常用的操作指令有了更多的了解，对shell编程等也有了一定的了解与练习。同时也让我对进程调度有了更深的体会。