**实验三 shell编程和进程创建**

1. **实验要求：**
2. **shell编程**
3. 学习常见的shell命令；
4. shell编程基础与实践；
5. **进程创建**
6. 学习进程的基础知识，了解进程创建的基本原理；
7. 熟悉进程的创建、控制、执行和终止等系统调用函数；
8. 安装vim编辑器,使用vim编制程序。
9. **实验内容：**
10. 使用并说明常见的shell命令。
11. **目录操作命令：pwd、cd、ls、mkdir、rmdir**
12. **pwd：显示当前路径。**

**1685168001179**

1. **cd：更改当前目录，语法为cd [name] 。**

|  |  |
| --- | --- |
| 特殊符号 | 含义 |
| / | 根目录 |
| .. | 上级目录 |
| ~ | 当前用户的默认工作目录。 |

下面以跳转至根目录的etc文件目录为例：

1685168165229

1. **ls：列出文件或目录下的文件名，语法为ls [-atFlgR] [name]。**

|  |  |
| --- | --- |
| 常用选项 | 含义 |
| -a | 列出包括以 . 开始等的隐藏文件 |
| -t | 依照文件最后修改时间的顺序列出文件名 |
| -F | 列出当前目录下的文件名及其类型 |
| -l | -使用详细格式列表 |

下面以ls -t为例（仅列出部分结果）：

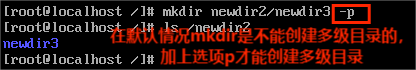


1. **mkdir：建立新目录，语法为mkdir [path]。**

下面分别在根目录和/home目录创建newdir目录与newdietest目录：



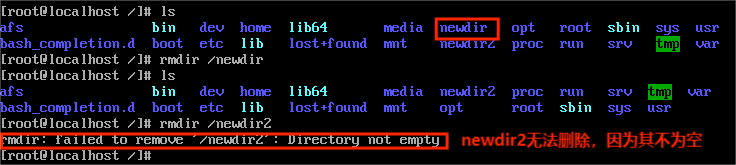
常用参数：**-p**，创建多级目录。



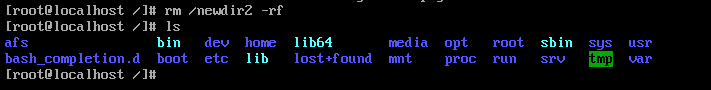
1. **rmdir：删除目录，但该目录必须为空，语法为rmdir [选项]。**

下面分别尝试删除刚刚创建的newdir目录与newdir2目录：

newdir成功删除；newdir2因为不为空，所以无法使用rmdir删除。



如果非要删除非空目录，需要使用 **rm - rf** 要删除的目录，这个指令也就是大家说的删库跑路指令。



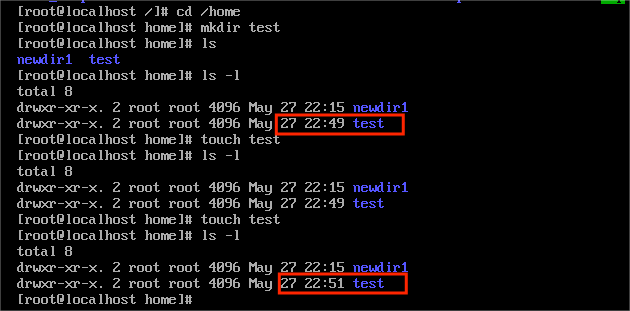
1. **文件操作命令：touch、file、cp、rm、mv、find**
2. **touch：修改文件或目录的时间属性，包括存取时间和更改时间。**

语法：touch [-acfm][-d<日期时间>][-r<参考文件或目录>] [-t<日期时间>][--help][--version][文件或目录…]

若文件不存在，系统会建立一个新的文件。

|  |  |
| --- | --- |
| 参数 | 含义 |
| -a | 改变档案的读取时间记录 |
| -m | 改变档案的修改时间记录 |
| -c | 假如目的档案不存在，不会建立新的档 |
| -f | 不使用，是为了与其他 unix 系统的相容性而保留 |
| -r | 使用参考档的时间记录,与 --file 的效果一样 |
| -d | 设定时间与日期，可以使用各种不同的格式 |
| -t | 设定档案的时间记录，格式与 date 指令相同 |

下面在/home文件夹中创建test，并且使用touch命令改变文件修改的时间：



1. **file：辨识文件类型。**

语法：file [-bcLvz][-f <名称文件>][-m <魔法数字文件>...][文件或目录...]

|  |  |
| --- | --- |
| 参数 | 含义 |
| -b | 列出辨识结果时，不显示文件名称 |
| -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 |
| -f<文件名称> | 指定名称文件，其内容有一个或多个文件名称时，让file依序辨识这些文件 |
| -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 |
| [文件或目录...] | 要确定类型的文件列表，多个文件之间使用空格分开，可以使用shell通配符匹配多个文件。 |

下面用file命令查询刚刚创建的test的文件类型：

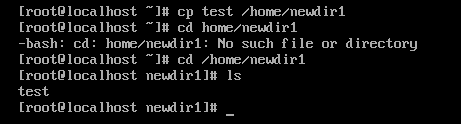


1. **cp：复制文件或目录**

语法：cp [options] source dest 参数说明：

|  |  |
| --- | --- |
| 参数 | 含义 |
| -a | 此选项通常在复制目录时使用，它保留链接、文件属性，并复制目录下的所有内容。其作用等于dpR参数组合 |
| -d | 复制时保留链接。这里所说的链接相当于 Windows 系统中的快捷方式 |
| -f | 覆盖已经存在的目标文件而不给出提示 |
| -i | 与 -f 选项相反，在覆盖目标文件之前给出提示，要求用户确认是否覆盖，回答 y 时目标文件将被覆盖 |
| -p | 除复制文件的内容外，还把修改时间和访问权限也复制到新文件中 |
| -r | 若给出的源文件是一个目录文件，此时将复制该目录下所有的子目录和文件 |
| -l | 不复制文件，只是生成链接文件 |

下面将根目录中的test文件拷贝到/home/newdir1下：

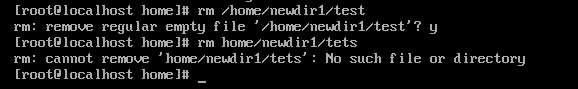


1. **rm：删除一个文件或目录。**

语法：rm [options] name...

|  |  |
| --- | --- |
| 参数 | 含义 |
| -i | 删除前逐一询问确认 |
| -f | 即使原档案属性设为唯读，亦直接删除，无需逐一确认 |
| -r | 将目录及以下之档案亦逐一删除 |

下面尝试将/home/newdir1/test文件删除，经过一次删除操作后，再次进行删除操作，可以发现该文件已被删除：

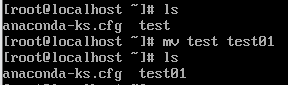


1. **mv：为文件或目录改名，或将文件或目录移入其它位置**

语法：mv [选项] 源文件或者目录 目标文件或者目录

|  |  |
| --- | --- |
| 参数 | 含义 |
| -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 |

下面尝试将根目录中的test文件更名为test01：

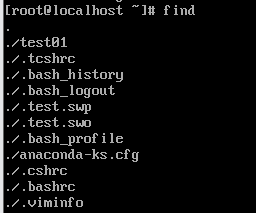


1. **find：在指定目录下查找文件。**

语法：find path -option [ -print ] [ -exec -ok command ] {} \

|  |  |
| --- | --- |
| 参数 | 含义 |
| -a | 必须满足两个条件才显示 |
| -o | 只要满足一个条件就显示 |
| -name | 按照文件名查找文件 |
| -iname | 按照文件名查找文件(忽略大小写) |
| -type | 根据文件类型进行搜索 |
| -perm | 按照文件权限来查找文件 |
| -user | 按照文件属主来查找文件 |
| -group | 按照文件所属的组来查找文件 |
| -fprint | 文件名：将匹配的文件输出到文件 |

下面在根目录下面执行find命令：



1. **归档及压缩命令：tar**

tar命令用于压缩文件，可以压缩，解压备份文件内的文件。

**压缩命令：**

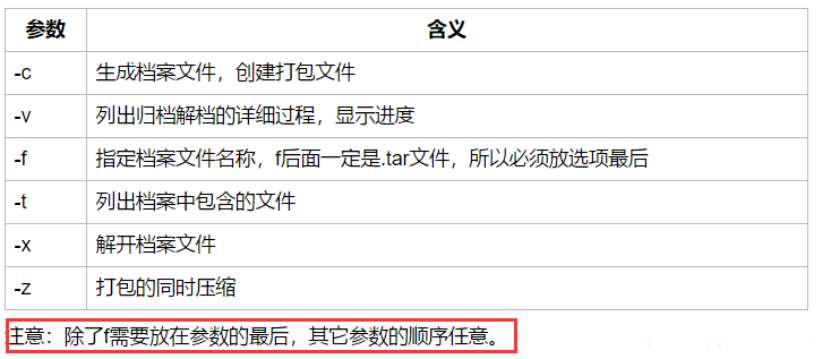
**tar -cvf 打包文件名 要打包的文件**

**解压命令：**

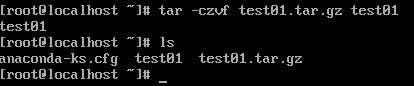
**tar -xvf 要解压的文件 解压到当前目录**

**tar -xvf 要解压的文件 -C /usr 把文件解压到/usr里面**

**常见参数及其含义：**



下面将根目录中的test01文件压缩：



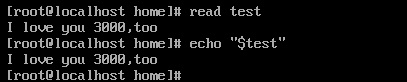
1. **输入输出与文件管理：echo、read、cat、grep、输入输出重定向**
2. **echo: 输出字符串**

语法：echo string

1685176124513

1. **read: 从标准输入读取数值**

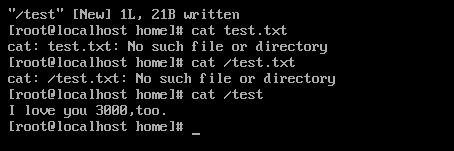
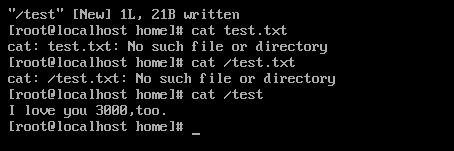
语法：read [选项] 变量名



1. **cat: 连接文件并打印到标准输出设备上**

语法：cat [-AbeEnstTuv] [--help] [--version] filename

在根目录下创建一个新的文件test，写入“I love you 3000,too.”，然后用cat命令输出:

****

1. **grep: 查找文件中符合条件的字符串**

语法格式：grep [-abcEFGhHilLnqrsvVwxy][-A<显示行数>][-B<显示列数>][-C<显示列数>][-d<进行动作>][-e<范本样式>][-f<范本文件>][--help][范本样式][文件或目录...]

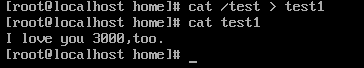
在刚刚创建的test文件中用grep命令查找字符“o”：

1685176577332

1. **标准输出重定向（>）：使用 > 符号将命令的输出保存到一个文件中**，如果该文件已存在，则会覆盖原有内容。

语法: command1 > file1

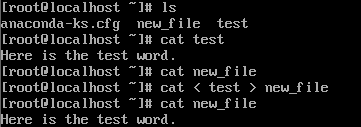
将cat /test的输出保存到文件test1中：



1. **标准输入重定向（<）：使用 < 符号将一个文件的内容作为命令的输入。**

语法：command1 < file1

在根目录下创建new\_file和test文件，将test的内容作为命令的输出，使用cat命令将其输出到new\_file文件中：



1. **管道操作**

在 Linux 中，管道操作（Pipeline）是一种通过将一个命令的输出连接到另一个命令的输入的方式，用于串联多个命令进行数据处理。通过管道操作，前一个命令的输出成为后一个命令的输入，依次类推，可以实现复杂的数据处理和转换操作。

语法：command1 | command2，管道操作会将 command1 的输出作为输入传递给 command2，command2 可以基于 command1 的输出进行处理。

使用管道操作，列出当前目录下的所有名为“test”的文件：



1. **编写shell脚本并执行**

先问了一下chatGPT如何在Linux上编写shell脚本并执行，下面是chatGPT给出的回答，非常清晰，比百度出来的实用多了：

Linux中编写和执行Shell脚本可以按照以下步骤进行：

1、打开文本编辑器：使用任何文本编辑器（如Vi、Nano、Gedit等），打开一个新的文件，用于编写Shell脚本的代码。

2、添加Shebang行：在脚本文件的第一行添加Shebang行，指定要使用的Shell解释器。例如，如果要使用Bash作为解释器，可以在第一行添加 #!/bin/bash。

3、编写Shell脚本代码：在文本编辑器中编写Shell脚本的代码。Shell脚本是一系列命令和控制结构的集合，用于自动化执行特定任务。

例如，以下是一个简单的Shell脚本示例，用于输出当前日期和时间：

#!/bin/bash

echo "Current date and time:"

date

4、保存文件：将脚本文件保存到你选择的目录中，确保文件名没有任何空格，并且以 .sh 作为文件扩展名（可选）。

5、赋予执行权限：在终端中，使用 chmod 命令为脚本文件赋予执行权限。例如，如果脚本文件名为 script.sh，可以运行以下命令：

chmod +x script.sh

这将使脚本文件具有可执行权限。

6、执行Shell脚本：通过在终端中输入脚本文件的路径来执行Shell脚本。例如，如果脚本文件位于当前目录下，可以运行以下命令：

./script.sh

脚本将开始执行，并在终端上显示输出。

请注意，Shell脚本的编写和执行需要一些基本的Linux命令和语法知识。你可以参考Shell脚本编程的教程和文档来进一步学习和掌握Shell脚本编程的技巧和技术。

1. **实现从键盘读取两个数，并比较两个数大小，并打印结果。**
2. 代码：

#!/bin/bash

read a

read b

if [ $a -eq $b ]; then

echo "$a = $b"

elif [ $a -gt $b ]; then

echo  "$a > $b"

else

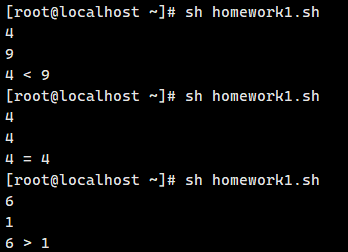
echo "$a < $b"

1. 代码解释：

首先读入用户输入的两个数a和b，

接下来进入if判断语句，如果a=b，输出“$a = $b”，如果a>b，输出“$a > $b”,如果a<b，输出“$a < $b”。

1. 执行结果



1. **实现读取文件的每一行**
2. 代码

#!/bin/bash

while read line

do

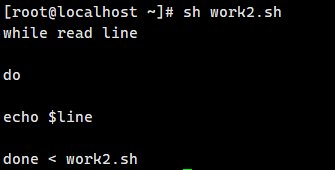
echo $line

done < filename

1. 代码解释

打开名为 filename 的文件，使用while循环逐行读取其中的内容，然后用echo命令将每一行输出到终端，直到line为空。

1. 执行结果（执行验证时输的是本题脚本文件的名字）



1. **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”**
2. 代码

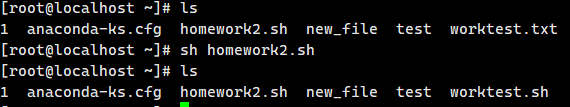
rename .txt .h $(find ./ -name "\*.txt")

1. 代码解释

rename .txt .h：这是rename命令的基本语法。它指示rename命令将所有符合条件的文件的扩展名 .txt 替换为 .h。

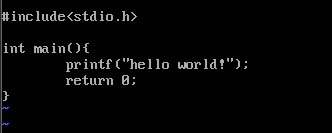
$(find ./ -name "\*.txt")：这部分是命令替换，使用find命令查找当前目录（./）及其子目录中所有扩展名为 .txt 的文件。$(...)将会被替换为find命令输出的结果。

1. 执行结果

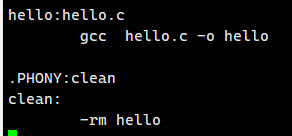


1. **编写make文件实现编译c或c++代码**
2. 代码

hello.c:



Makefile：

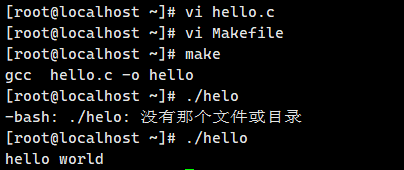


1. 代码解释

hello.c文件中是一个很简单的“hello world！输出程序”。

编写 makefile 文件可以帮助实现 C 或 C++ 代码的编译和构建。在makefile文件中hello为生成的文件名，hello.c为引入的.c源文件，下面一行为gcc操作。

1. 执行结果



1. **进程创建**
2. **学习进程的基础知识，了解Linux系统中进程创建的基本原理。**
3. **进程的基础知识：**

进程是运行在Linux中的程序的一个实例。当在Linux系统中执行一个程序时，系统会为这个程序创建特定的环境，这个环境包含系统运行这个程序所需的所有东西。

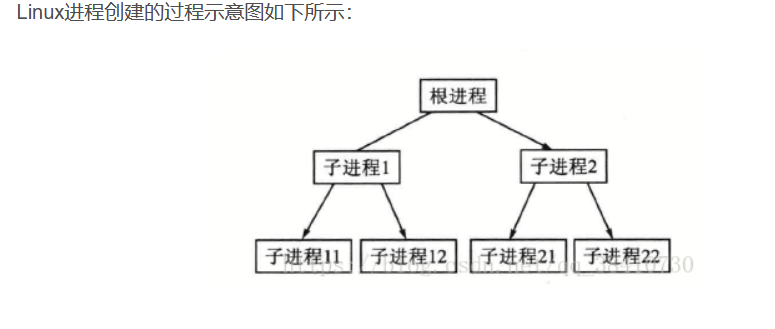
每当在Linux中执行一个命令，都会创建或启动一个新的进程。

每个终端窗口可能都运行了一个shell，每个运行的shell都分别是一个进程。当从shell调用一个命令时，对应的程序就会在一个新进程中执行，当这个程序的进程执行完成后，shell的进程将恢复运行。

操作系统通过被称为PID或者进程ID的数字编码来追踪进程，系统中的每一个进程都有一个唯一的PID。

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，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。

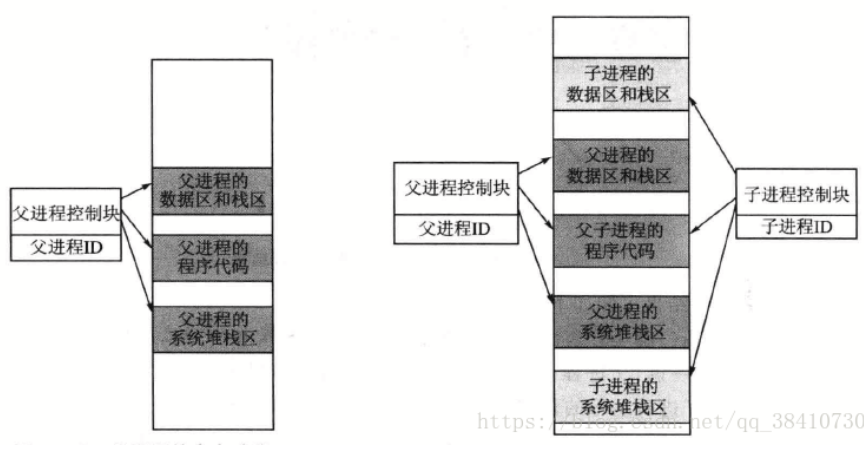
1. **创建的基本原理：**



在Linux中，父进程通过分裂（复制）的方式来创建子进程，创建一个子进程的系统调用叫做fork()。

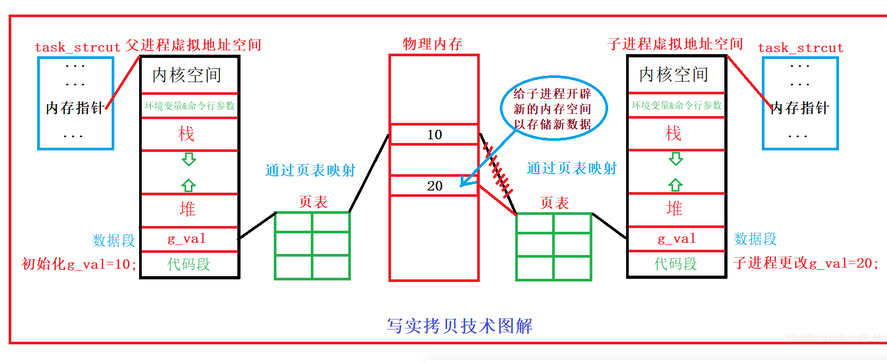
在系统中表示一个进程的实体就是进程控制块（PCB），创建新进程的主要工作就是创建一个新控制块，而创建一个新控制块的最简单的方法就是复制。复制并不是完全复制，因为父进程控制块中某些项的内容必须按照子进程的特性来修改，例如进程的标识、状态等。另外，子进程控制块还必须要有表示自己父进程的域和私有空间，例如数据空间、用户堆栈等。

1. **写时拷贝技术**：



子进程创建出来之后，与父进程映射访问同一块物理内存，但当父子进程当中有任意一个进程更改了内存中的数据时，会给子进程重新在内存中开辟一块空间，并将数据拷贝过去。这样避免了直接给子进程重新开辟内存空间，造成数据冗余。换句话说：如果父子进程都不更改内存中的值，那他们二者各自的进程虚拟地址空间通过页表映射，始终指向同一块物理内存。

正是通过这样的写时拷贝技术，才保证了父子进程代码共享但数据独有这一特性。



1. **熟悉进程的创建、控制、执行和终止等系统调用函数。**
2. **进程创建：fork();**

**返回值：**-1创建失败；0返回的是子进程；>0返回的是父进程，返回值是子进程ID。

**函数特性：**调用一次会有两个返回值；先返回哪个值不确定，一般先返回父进程；用户可以通过延时函数，决定进程的执行先后顺序；创建后，子进程复制父进程空间，这个空间子进程所独有的只有它的进程号、计数器和资源使用，其他的和父进程空间一样，两个空间相互独立，互不干涉，即使是全局变量，在子进程中不会随父进程中变量的值的改变而改变。

1. **进程控制：**

**getpid()**：获取当前进程ID。

**getppid()**：获取当前京城父进程的ID。

**wait**：回收子进程，查看子进程的死亡原因。

**nice**：改变进程调度优先级wait(), sleep();

1. **进程执行：exec函数族**

**int execl(const char \*path, const char \*arg, ...)**

功能：execl()用于执行参数path字符串代表的文件路径，接下来参数代表执行文件时传递arg，最后一个参数必须以空指针结束。

**int execv(const char \*path, char \*const argv[])**

功能：执行参数path字符代表的文件路径，第二参数以数组指针来传递给执行文件。

**int execle(const char \*path, const char \*arg, ..., char \*const envp[])**

功能：执行那个参数path字符代表的文件路径，接下来参数代表执行文件时传递的参数arg，最后一个参数必须指向一个新的环境变量数组，成为新执行程序的环境变量。

**int execve(const char \*path, char \*const argv[], char \*const envp[])**

功能：执行path字符串代表的文件路径，中间参数利用数组指针来传递给执行文件，最后一个参数为传递给执行文件的新环境变量数组。

**int execlp(const char \*file, const char \*arg, ...)**

功能：从path环境变量所指目录中查找符合参数file的文件名，找到后执行该文件，接下来参数代表执行文件时传递的arg，最后一个参数必须以空指针NULL。

功能：从path环境变量所指定的目录中查找符合参数file的文件名，找到后执行此文件，第二个参数argv传递给要执行的文件

1. **进程终止：exit();**

系统调用函数，主动退出进程。

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？**
2. **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？**

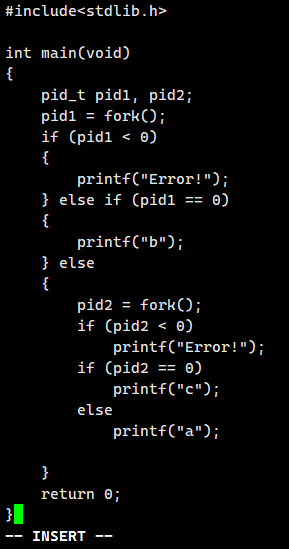
**3）分析进程调度和进程并发执行的关系？**

1. **安装vim编辑器**

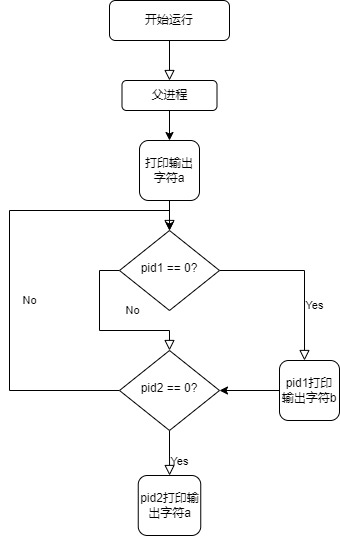
使用命令sudo yum install vim即可安装vim编辑器。

1. **编制程序**

**代码：**



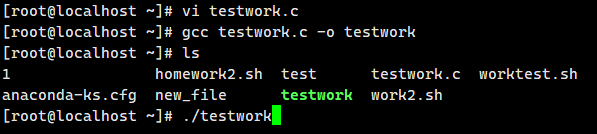
**流程图****:**

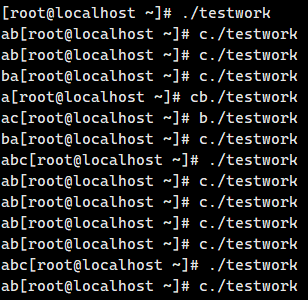


**代码解释：**

上面的代码中，首先通过fork()函数创建pid1，pid2两个子进程；并且通过条件语句确保进程正常运行时打印输出字符，否则打印输出“error！”字样。

**编译运行:**

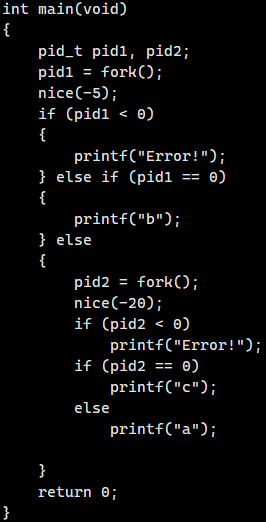




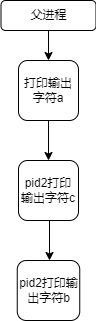
进程调度和并发执行的情况如上图所示，可见字符显示的顺序会发生变化，这与系统的调度有关。多次执行时，a/b/c 的显示顺序不一定，取决于进程的调度时机。

1. **使用系统调用nice()来改变各进程的优先级。**

**代码：**



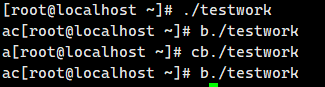
**流程图：**



**代码解释：**

加入了nice(-5)改变pid1的优先级。

**编译运行：**



字符的输出顺序由不确定变为确定。

1. **回答以下问题：**

**1）系统调用fork()是怎样创建进程的？**

当调用fork()时，操作系统会创建一个新的进程，该进程与调用fork()的父进程几乎完全相同。新进程会复制父进程的代码、数据和堆栈，并分配一个新的进程标识符。父进程和子进程之间的区别在于fork()函数的返回值。对于父进程，fork()返回新创建的子进程的进程标识符，而对于子进程，fork()返回0。这样，通过检查fork()的返回值，可以确定当前进程是父进程还是子进程。

**2）当首次调用新创建的子进程时，其入口在哪里？**

当首次调用新创建的子进程时，其入口在fork()调用之后的下一条指令处。也就是说，子进程会从fork()函数返回的地方开始执行代码。这意味着父进程和子进程会同时执行fork()之后的指令，但是它们在各自的进程空间中独立执行，彼此不会相互干扰。

**3）分析进程调度和进程并发执行的关系？**

首先区别并发和并行，并行是指两个或多个事件在同一时刻发生，而并发是指两个或多个事件在同一时间间隔内发生，在宏观上看并发仿佛多个进程同时在进行，但微观上其实进程并不是一起进行的，是由于进程调度，让每个进程使用一段时间的cpu，这个时间极短，看上去像是同时进行的。

进程调度和进程并发执行密切相关。进程调度是操作系统决定哪个进程获得CPU执行时间的过程，而进程并发执行指的是多个进程在同一时间段内执行。操作系统使用调度算法来决定进程的执行顺序和时间片分配，以实现公平性、优先级和资源利用效率等方面的考虑。

1. **实验分析与总结**

本次实验让我学到了很多关于shell编程和Linux进程的知识。一开始，我在编写shell脚本的过程中遇到了一些困难，卡了很长时间，查阅资料后发现，原来是很低级的错误，在执行脚本前没有使用chmod +x script.sh使脚本文件具有可执行权限。但通过查阅相关资料和不断的尝试，我逐渐掌握了常见的目录操作命令、文件操作命令以及归档和压缩命令等。

在实验的第二部分中，我安装了vim编辑器，并使用它编写了一个程序，通过系统调用fork()创建了两个子进程。在程序执行过程中，我观察到父进程和两个子进程分别在屏幕上显示了字符a、b和c，这验证了进程的并发执行和调度机制。并尝试使用系统调用nice()来改变进程的优先级。通过调整优先级，观察到进程的执行顺序发生了变化。优先级较高的进程会更频繁地获得CPU的执行时间片，从而提高了其执行速度。

在实验过程中，我也遇到了一些问题，例如对于makefile文件的编写和子进程的具体代码。但通过查阅资料和与同学的讨论，我逐渐克服了这些问题，并最终成功完成了实验任务。

通过这次实验，我对shell编程和Linux进程有了更深入的理解。我意识到Shell是一个强大的工具，可以用于自动化任务和系统管理。同时，深入了解进程的创建、控制、执行和终止等方面，让我对操作系统的工作原理有了更清晰的认识。在未来，我希望能够继续学习和探索Shell编程和Linux进程的更多知识，以提升自己在系统管理和软件开发方面的能力。