# Linux学习笔记

|  |
| --- |
|  |

## Linux基操

|  |
| --- |
|  |

### 1、目录相关操作

#### 1：查看文件ls

|  |  |  |
| --- | --- | --- |
| 操作 | 解释 | 备注 |
| ls -a | 列出目录下所有子目录和文件。包括以.开头的隐含文件 | 无简写 |
| ls -d [File] | 将目录像文件一样显示，而不显示其下的文件  只显示该文件（为了查看某个文件/文件夹是否存在？） | 无简写 |
| ls -i [File] | 输出文件的i节点的索引信息（包含inode号）  显示文件（文件前面有个数字，常用搭配：ls -ai，显示全部文件，每个文件前都有个数字）  ls -i [File]  ls -i  ls -ai | 无简写 |
| ls -k [File] | 以k字节的形式显示文件大小  ls -lk  ls -alk | 无简写 |
| ls -n | 与ls -l 效果相识，都是列出目录下文件的全部信息（不包括隐含文件）  ls -n使用数字来代替UID（用户ID名称）和GID（组ID名称） | 无简写 |
| ls -1(数字1) | 列出全部文件（只列出文件名），并且一行只显示一个文件名 | 无简写 |
| 排序：  ls -r  ls -t  ls -s | ls -r：对目录进行反向排序（只显示文件名）  ls -t：对目录文件按时间排序（最新创建的文件在第一个，只显示文件名）  ls -s：显示文件名和该文件的大小 | 无简写 |
| ls -l | 显示全部文件的详细信息（不显示隐含文件） | ll |
| ls -R | 列出所以子目录的文件（采用递归的方式） | 无简写 |
| ls -F | 列出所有文件，并在文件名后列出该文件的类型。  \* ：可执行的普通文件  / ：目录  @ ：符号链接  | ：FIFOs  = ：套接字（sockets） | 无简写 |

所有指令都可以相互搭配，并且没有先后顺序之分。

**（1）文件的****inode号包含的信息有：**

文件的字节数

文件拥有者的ID

文件的Group ID

文件的读、写和执行的权限

文件的时间戳（上一次inode号变动的时间、上一次文件内容变动的时间、文件上一次打开的时间）

链接数，即有多少文件名指向该inode

文件数据的block的位置

Linux文件系统的inode号和文件名（文件的目录项）是一一对应的，并且可以通过inode实行文件的逻辑结构和物理内存的转换。一个大文件在物理内存中并不一定占用连续的内存空间，此时就需要inode号来实现逻辑结构和物理内存的转换

#### 2：pwd

显示用户当前所在目录的绝对路径。

用法：pwd

绝对路径：路径表示以根目录为起始的路径。

相对路径：以当前所在目录作为起始的路径。

#### 3:文件夹创建和删除

mkdir：创建子文件夹

用法：

mkdir [FileNmae] ：在当前目录下创建一个子文件夹

mkdir [FileNmaeFather] [FileNmaeChild] 在名为FileNmaeFather的文件夹下创建名为FileNmaeChild的文件夹，如果FileNmaeFather文件夹不存在，则自动创建。

rmdir：删除一个文件夹（如果删除后父目录变为空目录，则把父目录也删除）

用法：rmdir[FileNmae]

rmdir只能删除空目录，如果一个文件夹中不空，需要使用rm -r

#### 4：文件创建和删除

touch：修改文件或目录的日期时间（包括存取时间和更改时间），或新建一个不存在的文件。

|  |  |  |
| --- | --- | --- |
| 操作 | 解释 | 备注 |
| touch [FileName] | 在当前目录下新建一个名为FileName的普通文件 |  |
| touch -a |  |  |
| touch -c |  |  |
| touch -d |  |  |
| touch -m |  |  |
| touch -r |  |  |
| touch -t |  |  |

rm：删除指定文件

用法：rm [FileName] :删除指定文件

|  |  |  |
| --- | --- | --- |
| 操作 | 解释 | 备注 |
| rm -f | 直接删除（可以删除只读文件） |  |
| rm -i | 删除前询问 |  |
| rm -r | 删除文件夹和该文件夹中全部的文件 |  |

rm -rf ./\*：删除当前目录的全部文件

rm -rf /\*：从家目录开始删除全部文件（不要才root下运行）

#### 5：文件移动mv

可以跨目录操作：移动一个文件，不一定要在该文件的目录下

语法：

mv [File ][Path ]：移动文件

mv [FileName ] [NewFileName ]：对文件重命名

特殊操作符：

mv helloworld.c ~ :直接将当前目录下的helloworld.c文件移动到家目录下(/home/zah)

mv /home/zah/zah\_3/helloworld.c . :将helloworld.c移动到当前目录下

mv /home/zah/zah\_3/helloworld.c .. :将helloworld.c移动到当前目录的上一级目录下

#### 6：文件复制cp

可以跨目录操作

|  |  |  |
| --- | --- | --- |
| 操作 | 解释 | 备注 |
| cp [File] [Path] | 将File（文件）拷贝到Path下 |  |
| cp -r [File] [Path] | 将File（文件夹）拷贝到Path下 |  |
| cp -r [File] [Path] [NewFile] | 将File（文件夹）拷贝到Path下，并重命名为NewFile |  |
| cp -force | 强行复制文件或目录，不管是否已经存在 | cp -f |
| cp -interactive | 覆盖文件前进行询问 | cp -i |
| cp -r | 递归拷贝，将文件夹下所有目录和文件全部拷贝 |  |
| cp -recusive | 将指定目录下的文件及子目录一并处理 | cp -R |

#### 7：cd

进入指定目录，改变当前指定路径。

用法：

cd [Address] ：进入Address路径下。（输入绝对路径）

cd ~ ：回到家目录（/home/zah）

cd .. ：返回上级目录（. ：当前目录。 .. ：上一级目录）

cd - ：访问最近访问的目录

### 2、文件相关操作

当文件不大时，可以使用cat

more：只能向后查看，不能向前查看

less：

#### 1：cat

|  |  |  |
| --- | --- | --- |
| 操作 | 解释 | 备注 |
| cat -b [File] | 查看文件时，对非空行进行编号 |  |
| cat -n [File] | 查看文件时，对所有行进行编号 |  |
| cat -s [File] | 查看文件时，不输出多行空行 |  |

#### 2：echo

功能：往文件里写内容

echo [”string”] > FlieName

每次使用echo，输入新的字符串后，老的字符串就被替换

#### 3：more

|  |  |  |
| --- | --- | --- |
| 操作 | 解释 | 备注 |
| more -n [File] | 查看文件时，对所有行进行编号（实验失败） |  |
| q | 退出 |  |

#### 4：less

查看文件时，不会加载全部内容

|  |  |  |
| --- | --- | --- |
| 操作 | 解释 | 备注 |
| less -i [File] | 忽略搜索时的大小写 | 参数 |
| less -N [File] | 显示每一行的行号 | 参数 |
| /“string” | 向下搜索字符串 | 阅读器操作 |
| ?“string” | 向上搜索字符串 | 阅读器操作 |
| n | 重复前一个搜索 | 阅读器操作 |
| N | 反向重复前一个搜索 | 阅读器操作 |
| q | 退出 | 阅读器操作 |

#### 5：head

head -n<num> [File] ：显示File的前num行（默认10行）

#### 6：tail

tail -n<num> [File] ：显示File的后num行（默认10行）

tail ：不指定文件，作为输入信息进行处理，用于查看日志文件。

tail -f [File] ：检测文件是否有变化（实时刷新，如果文件有变化，则输出变化的内容）

### 3、压缩和解压缩

#### 1：zip和unzip

语法：

zip [File].zip [Files]：将压缩成Files压缩成File.zip（具有更名效果），如果压缩一个文件夹，则该指令只压缩文件夹（文件夹内的文件和文件夹不跟着一起压缩）

zip -r [File].zip [Files]：采用递归的方式，将所有文件夹和文件都压缩

unzip [File].zip：将File.zip解压成Files（解压后的名字和zip文件名一样）

unzip [File].zip -d [Address]：将File.zip解压到指定路径

#### 2：tar

tar主要功能：打包、解包、不打开压缩包直接看内容

|  |  |  |
| --- | --- | --- |
| 参数 | 功能 | 备注 |
| -c | 建立一个压缩文件（create） |  |
| -x | 解开一个压缩文件 |  |
| -t | 查看tarfile里的文件 |  |
| -z | 使用gzip压缩 |  |
| -j | 使用bzip压缩 |  |
| -v | 压缩过程中显示文件（不建议用在背景执行过程中） |  |
| -f | f后加文档名，例如：  tar -cvf [File].zip [Files]  tar -zcvf [File].zip.gz [Files]  tar -jcvf [File].zip.bz2 [Files] | f后不能再有其它参数 |
| -C | 解压到指定目录 |  |

### 4、匹配查找

#### 1:find

Linux下，find在目录结构中搜索文件。

语法：

find [Path] [FileName]

find的一些参数：

find ./-name/-size/-type(文件类型)/-time

#### 2:grep

语法：grep [参数（指令）] [待查找内容] [查找路径]

参数：

-i：忽略大小写

-n：查找时输出行号

-v：反向查找（即查找不包含待查找内容的文本行）

-c：计算搜索到的次数

详细介绍：

<https://www.cnblogs.com/ggjucheng/archive/2013/01/13/2856896.html>

### 5、权限操作

#### 1：权限的表示

r：可读权限

w：可写权限

x：可执行权限（对于文件夹：具有可以进入该文件夹的权限）

-：不具有权限

#### 2：Linux下，使用ll输出信息的解释（使用ll可以查看权限）

r：没有可读权限，不能查看文件内容

w：没有可写权限，不能往文件中添加内容

x：没有可执行权限，不能使用 ./ 执行相应的文件

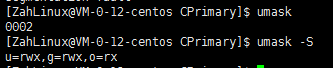
|  |  |
| --- | --- |
| 文件类型 | d：文件夹  -：普通文件  l：软连接（相当于windows下的快捷方式）  b：块设备（硬盘，光驱等）  p：管道文件  c：字符设备文件（屏幕接口）  s：套接口文件 |
| 文件所有者的权限（前3位）u | r--：只读(100)  --x：仅可执行(001)  -wx：可写可执行(011)  rwx：可读，可写，可执行(111)  -w-：可写(010)  rw-：可读可写(110)  r-x：可读和可执行(101)  ---：不具有任何权限(000) |
| 文件所属用户组的权限（中间3位）g |
| 其他用户的权限（末尾3位）o |
| 连接数 |  |
| 所属用户名 |  |
| 所属用户组名 |  |
| 文件大小 | 单位：字节 |
| 文件最新修改日期 |  |
| 文件名 |  |

#### 3：权限的更改

##### （1）默认权限设置umask

语法：umask [ugo]

ugo是默认权限的掩码（八进制）



##### （2）文件创建后权限的更改chmod

当前用户：u

所属组用户：g

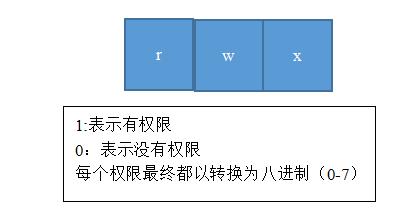
其他用户：o

语法：

**chmod [u/g/o] [+/-] [r/w/x] [FileName]**  对FileName的当前用户/所属组用户/其他用户 添加/去除 可读/可写/可执行操作

**chmod [+/-] [r/w/x] [FileName]** 对所有用户添加/去除 可读/可写/可执行操作

**chmod [uq gq oq] [FileName]** uq表示当前用户的权限码（八进制码），具体流程如下：



#### 4：沾滞位

沾滞位：对于其它用户来说，在修饰了沾滞位的目录中可以创建文件，但是只能删除自己的文件 chmod +t 文件（为该文件设置沾滞位）

### 6、其他操作

#### 1：系统日期date

**date的参数： (查看当前时间或日期)**

%H : 小时(00..23)

%M : 分钟(00..59)

%S : 秒(00..61)

%X : 相当于 %H:%M:%S

%d : 日 (01..31)

%m : 月份 (01..12)

%Y : 完整年份 (0000..9999)

%F : 相当于 %Y-%m-%d

例：

[zah@VM-0-12-centos zah\_2]$ date

Mon Mar 8 16:48:52 CST 2021

[zah@VM-0-12-centos zah\_2]$ date +%Y %m %d

date: extra operand ‘%m’

Try 'date --help' for more information.

[zah@VM-0-12-centos zah\_2]$ date +%Y:%m:%d

2021:03:08

[zah@VM-0-12-centos zah\_2]$ date +%Y%m%d

20210308

**修改时间（只有root用户才有此权限）**

date -s //设置当前时间，只有root权限才能设置，其他只能查看。

date -s 20080523 //设置成20080523，这样会把具体时间设置成空00:00:00

date -s 01:01:01 //设置具体时间，不会对日期做更改

date -s “01:01:01 2008-05-23″ //这样可以设置全部时间

date -s “01:01:01 20080523″ //这样可以设置全部时间

date -s “2008-05-23 01:01:01″ //这样可以设置全部时间

date -s “20080523 01:01:01″ //这样可以设置全部时间

#### 2：用户切换su

su - root：从普通用户切换到root用户

su root：从普通用户切换到root用户

exit :退出root用户，切换到之前的子用户

su 用户名称 ：从当前用户切换到指定用户（一般需要密码）

sudo：为当前用户提高权限

cat /etc/passwd：查看子用户

#### 3：查看目录结构tree

该命令需要安装：

（1）切换到root用户

（2）执行yum -y install tree

#### 4：查看历史执行过的命令history

快速执行：

！+ 之前执行过的命令的缩写

！+之前执行过的命令的缩写

#### 5：查看介绍man

man 命令名称 :例 man ls

man 1 命令名称：查看命令例man 1 ls

man 2 命令名称：查看系统函数

man 3 命令名称：查看库函数

### 7、快捷键

|  |  |  |
| --- | --- | --- |
| 操作 | 介绍 | 备注 |
| ctrl + c | 中断当前的命令输入。（不指向当前输入的命令，之前跳到新行）  中断正在执行的前台进程。 |  |
| tab | 快速补全命令 |  |
| ifconfig | 查看终端信息 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### 8、shell

shell 是命令行解释器的统称，并不指具体的解释器

[zah@VM-0-12-centos ~]$ echo $BASH

/bin/bash

具体的命令行解释器为：/bin/bash 称为：bash

命令行解释器：我们所有输入的命令都传递给命令行解释器，然后命令行解释器再将命令传递给操作系统，最后操作系统再通过命令行解释器将结果返回。

输入：env也可查看shell

### 9、Linux工具

#### 1：软件工具包安装yum

|  |  |  |
| --- | --- | --- |
| 操作 | 介绍 | 备注 |
| list | 查看应用商店的软件  [zah@VM-0-12-centos ~]$ yum list | grep tree | 指令前面没有- |
| install -y | 安装软件  -y的意思是：如果碰到依赖项，自动安装  例：# yum install -y lrzsz |  |
| remove | 软装卸载 |  |
| yum update | 更新所有包，但会保留老的版本 |  |
| yum upgrade | 更新所有包，但会删除老的版本 |  |
| yum clean all | 清除yum的缓存（之前下载的安装包啥的） |  |
| yum search 软件包 | 从应用商店寻找可用的软件包 |  |
| yum makecache | 将服务器中软件包的信息下载下来，方便后续的搜索  yum -C search 软件包名 (可用在不联网的情况下查看软件信息)  可用yum clean all清除，释放硬盘 |  |

#### 2、远程服务器和Windows交互

这两个功能是：lrzsz软件的功能，用于linux和Windows的文件互传

sz 要下载的文件名称

##### 1：上传文件rz

##### 2：下载文件sz

#### 3、编辑器vim

编辑器的演化：vi -> vim

##### （1）vim编辑器的三种模式

**插入模式INSERT：**可以详细的编辑代码

**正常模式NORMAL：**可以进行批量操作

**底行模式COMMAND：**输入指令，执行相应操作

正常模式

插入模式

底行模式

Esc

i

：

Esc

##### （2）正常模式下的操作

###### 1：移:光标的移动:

第一种方式：键盘中的上下左右键（INSERT模式下）

第二种方式：（必须在NORMAL模式下）

j：光标向下

k：光标向上

h：光标向左

l：光标向右

w：光标跳转到下一个词的词首

b：光标跳转到上一个词的词首

###### 2：删

Vim没有真正的删除，都是剪切，在下次剪切前不粘贴，就是删除

第一种方式：（NORMAL模式下进行）

dd：删除一行（剪切光标所在行）

p：将剪切板的内容粘贴到光标所在行的下一行

P：将剪切板的内容粘贴到光标所在行的上一行

num dd:删除光标所在行及下面的num-1行的内容

x：删除单个字符（删除光标所在位置的字符）

num x：删除多个字符（删除光标随着位置及后面num-1个字符）

X：删除光标所在的前一个字符

num X：删除光标所在的前num个字符

第二种方式：（INSERT模式下）

退格键删除：←

###### 3：复

yy：复制一行

num yy：复制多行（复制当前行及下面的num-1行）

###### 4：替

r：替换光标所在字符（先按r，再输入要替换的字符）

R：一次替换多个字符(直接进入INSERT模式，可以依次替换光标所在字符及后面的字符，完成后按Esc回到NORMAL模式)

替换字符串：（必须在命令行模式下进行）

:s/被替换的字符串/要替换的字符串/g 只替换光标所在行

:%s/被替换的字符串/要替换的字符串/g 替换全文

（如果不加g，则只替换第一个被替换的字符，加上g后，替换全部的被替换字符串）

###### 5：撤

撤销必须在NORMAL模式下才有用

u：撤销（Windows下的ctrl+z）

ctrl+r：反撤销(Windows下的ctrl+y)

###### 6：更

C：将光标后面的字符删除（作用域：所在行），并进入INSERT模式

cw：删除光标所在的单纯，并进入INSERT模式

###### 7：跳

gg:光标跳转到文件头部

G(shift+g):光标跳转到文件尾部

num G:光标跳到任意行(不能用数字键盘的数字，并且必须为G)

ctrl+o：光标跳转到上一次位置

gg=G：代码的缩进处理

##### （3）其它快捷方式

i：从NORMAL进入INSERT模式（并且插入字符时，插入在光标之前的位置）

I：光标定位到当前行的行首位置，并进入INSERT模式

o：进入INSERT模式，并将光标的下一行进行插入

O：进入INSERT模式，并将光标的上一行进行插入

A：光标直接定位到当前行的行尾模式，并进入INSERT模式

**gg=G：代码的缩进处理**

##### （4）注释

与C语言注释方法相同

多行注释：ctrl+v（进入可视模式），然后按k,j,选完要注释的行后，按I(shift i)进入IMSERT模式，插入 // ，最后按ESC即可。（l和h进行左右横选）

多行反注释：ctrl+v（进入可视模式），然后按k,j,选完要注释的行后，按x

##### （5）底行模式

底行模式操作：

wq:保存退出

q：不保存退出

q!：强制退出(不保存更改,当更改只读时，只有这种方式可以退出)

w:保存

: 进入命令行模式（底行模式）

：set nu 添加行号

: set nonu 去除行号

: vs同时打开多个文件，并分屏

同时打开多个文件时，跳转文件按ctrl+ww

按下/ 也会进入命令行，/ 要查找的字符串（光标会定位到第一次出现的位置，按n可以往下查找）

#### 4、编译器gcc/c++

##### 1、预处理

预处理过程：宏替换、头文件展开、去注释

-E：生成.i文件(只让编译器完成预处理过程后停下来)

gcc -E hello.c 只预处理，但不生成预处理文件（将生成结果显示出来）

gcc -E hello.c -o hello.i 预处理，并生成预处理文件，不显示预处理结果

##### 2、编译

编译过程：检测语法语义，并生成汇编代码

-S：生成.s文件（告诉编译器只完成编译构成就停下）

gcc -S hello.i -o hello.s //从.i文件进行编译

gcc -S hello.c -o hello.s //直接从.c文件开始预处理，并编译

##### 3、汇编

汇编过程：将汇编代码转化为机器语言

-c ：生成.o文件（相当于目标程序.obj文件）

gcc -c hello.c -o hello.o

##### 4、链接

动态链接：链接动态库，只是将动态库里的程序的信息表拿来（所以，程序运行时，必须加载动态库的文件。所以其加载较慢，并且需要动态库的存在。但其动态库可以共享，内存利用率较高）

静态链接：链接静态库，将静态库的文件拷贝下来。（所以，程序运行时，不需要加载库文件，所以速度快。但静态库很有可能会被多个程序共享，避免内存冗余，程序变得很大。）

链接过程：将多个目标程序链接为.exe

gcc hello.o -o hello (最后的hello就相当于,exe文件，./hello就可指向该应用程序)

./ 执行当前目录下的应用程序

例：

[zah@VM-0-12-centos ~]$ gcc hello.c -o hello //从.c直接到.exe文件

[zah@VM-0-12-centos ~]$ ls

hello hello.c hello.i hello.s hellos.zip install.sh zah\_2 zah\_3

[zah@VM-0-12-centos ~]$ ./hello //运行hello

i=55!!hellow!![zah@VM-0-12-centos ~]$

#### 5、调试器gdb

要想调试程序，需要在编译的时候将程序编译为Debug版本

linux系统默认生成的release版本，如果想要生成debug版本，需要加-g

gcc hello.c -o hellos -g debug版的程序更大

##### 1、调试可执行程序

gdb 可执行程序名称（只能调试debug版本的可执行程序）

l：查看源代码

b 源码文件中的行号：打断点

delete 断点的序号：去除断点

i b ：查看断点信息

watch:可以周期性的执行下个程序，并显示结果，可以检测一切命令的结果变化

disable 断点的序号：使断点失效（断点还存在，但是程序不会停留在失效的断点处）

r：运行程序（如果没有断点，直接全部运行）

start：开始调试

q：退出调试

s:逐语句指向（F11） //必须在运行的前提下

n:逐过程（F10）

c：继续执行，直到遇到下一个断点

按回车，可以继续上一次的指令（不需要重复输入指令）

p 变量名 ：输出此时变量的值

until num：直接运行到指定行

bt(backtrace)：查看函数调用栈信息，可以很快找到出错的程序行。（调用某个进程时，出错，该进程并没有出栈，所以出错点就在栈顶的位置）

p(print)：查看或设置一个变量的内容

##### 2、调试coredump文件

程序崩溃后的内存镜像文件

ulimit -a

ulimit -c

gdb bt :查看错误行

#### 6、make&makefile：项目自动化构建工具

|  |  |  |
| --- | --- | --- |
| 操作 | 解释 | 备注 |
| make | makefile的解释程序，逐行执行makefile中的项目规则 |  |
| makefile | 记录了项目构建流程的项目文件 |  |

**makefile规则：**

目标对象：需要生成的可执行程序（目标程序）

依赖对象：生成目标对象时需要的文件。

**编译命令：**vim makefile

**编译格式：**

目标对象：依赖对象

编译命令

**例：**

hello:hello.o

gcc hello.o -o hello

hello.o:hello.s

gcc -c hello.s -o hello.o

hello.s:hello.i

gcc -S hello.i -o hello.s

hello.i:hello.c

gcc -E hello.c -o hello.i

**预定义变量：**

$@：

$^：

$<：

**例：**

a:a.c

gcc $^ -o $@

**make规则：**

找到makefile中第一个目标对象，生成它。为了生成它，会通过递归找到依赖对象的生成规则。

#### 7、GIT：项目的版本关联工具

git的目的：备份和回滚

##### 1、提交修改记录

操作：

git add --all ./ ：将当前目录下所有的更改都提交

##### 2、提交到本地仓库

操作：

git commit -m [remarks] ：添加注释

##### 3、同步到远端仓库

操作：

git push origin master：将本地仓库同步到远端仓库。

## Linux系统编程

|  |
| --- |
|  |

### 1、进程

#### 1、操作系统和冯诺依曼结构

##### （1）冯诺依曼结构

所有的硬件都是围绕内存工作的。

##### （2）操作系统

操作系统就是一个搞管理的软件。如何管理：先描述，再组织

**系统调用接口：**系统向上层提供的用于访问内核的接口（操作系统内核并不能直接访问）。**库函数：**系统调用接口的封装，库函数与系统调用接口的关系：上下级的调用关系

程序调用，运行在用户态。

程序运行（被调用的过程中），运行在内核。

|  |  |  |
| --- | --- | --- |
| 一些计算机概念简称 | | |
| 简写 | 介绍 | 备注 |
| TCB | 线程控制块 |  |
| MMU | 内存管理单元 | 负责处理CPU的内存访问请求，可以实现从虚拟地址内存到物理地址内存的转换、内存保护、CPU高速缓存控制 |
| CACHE | 高速缓存 |  |
| DMA | 直接内存存取 |  |
|  |  |  |

#### 2、进程概念

**进程概念：**运行中的程序，在操作系统的角度中是一个进程控制块（PCB），在Linux系统下进程是一个task\_struct结构体，是对程序动态运行的描述，通过描述实现操作操作系统对程序调度运行及管理。

系统感知进程的唯一实体是：进程控制块（PCB）

**进程描述的信息有：**

内存指针：进行运行到哪个位置

上下文数据（程序计数器）：程序切换时的状态信息，当程序回来时可以继续运行。

标识符：

状态：进程的状态

IO信息：

**进程的特点：**

进程并没有一个精确的定义。

（1）进程和程序的区别：

1.进程是操作系统最近本、最重要的概念。进程是动态的，具有生命周期。并且进程是可以并发执行的。而程序只是指令的合集，本身并没有所谓的生命周期，并且是静态的。进程是竞争CPU资源的基本单位，而程序没有竞争CPU资源的资格。一个进程可以执行一个或几个程序，一个程序也可能被一个或多个进程执行。（进程执行程序）

2.进程是运行中的程序，有前台和后台之分（前台进程：当前占用终端的进程，即正在与用户进行交换的进程），而程序只是一段静态指令集（也可以认为是没有状态），没有前后台之分。

（2）作业是用户需要计算机完成的某项任务，是要求计算机所做工作的集合，一个作业可由多个进程组成，且必须由一个进程组成。

（3）进程和线程

进程是资源管理的最小单位，而线程是程序执行的最小单位。Linux下，线程本质上是用进程实现的。

#### 4、进程状态

##### （1）就绪

就绪状态：进程已经准备好，随时可以运行，只是还没被分配CPU资源。

##### （2）运行

运行状态：正在调用CPU资源的进程。

##### （3）阻塞

阻塞状态：程序没有被进程调度。

##### （4）Linux下进程的状态

###### 1：运行

包含就绪状态和运行状态。（只要被分配CPU资源，立马就可以正常运行）

###### 2：可休眠中断

阻塞状态，在被操作系统打破休眠或满足自身唤醒条件后，立马变为运行状态。

###### 3：不可中断休眠

阻塞状态，操作系统不可打破该进程的休眠状态，只能等该进程自身满足唤醒条件后，才能变为运行状态。

###### 4：停止

停止的进程还会被操作系统调度，调度该进程时，CPU还是会正常消耗资源，但是并不进程处理，需要特殊的信号才能对其唤醒。

###### 5：僵尸进程

**僵尸态：**一个进程已经退出，但是其占用的资源并没有被完全释放。

**僵尸进程产生原因：**子进程退出时，会向父进程发送退出信息（这部分信息包含了子进程的退出原因），但是父进程并没有注意到子进程的退出信息，这就导致该子进程占用的资源并没有被完全释放，从而导致该子进程变为僵尸进程。

**僵尸进程的危害：**导致系统资源被占用而无法回收。并且每个用户能创建的子进程是有数目限制的，僵尸进程将导致用户可创建的子进程数目变少。

**僵尸进程的解决方法：**退出父进程、使用进程等待避免产生僵尸进程

###### 6：孤儿进程

**孤儿进程产生原因：**父进程先于子进程退出，导致该子进程变为孤儿进程。

**孤儿进程特点：**脱落终端，运行在后台。1号进程（int进程）变为孤儿进程的父进程。

孤儿进程不会造成系统资源的浪费。1号进程会自动完成孤儿进程的善后工作，所以孤儿进程不会有什么危害。

###### 7：守护进程或精灵进程

守护进程或精灵进程是一种特殊的孤儿进程。

###### 8：死亡进程

正常退出的进程，进程刚刚退出时，其状态就变为死亡进程，然后就被刷新掉。

#### 5、环境变量

环境变量作用：使程序和shell的运行环境的配置更加灵活、可以进行进程间的数据传递。

命令操作：env,set（查看所以变量（包括环境变量和普通变量（普通变量不可进行进程间的数据传递）））,echo,export,unset

典型的环境变量：PATH

获取环境变量的方式：

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

extern char \*\*environ

getenv()

**操作：**

echo：查看环境变量（echo $PATH）

export：设置或显示环境变量

unset：删除指定环境变量

#### 6、程序地址空间

**程序地址空间：**进程的虚拟地址空间，操作系统为每个进程所描述的虚拟的地址空间，在Linux下，是一个mm\_struct结构体。进程的运行是在物理内存空间上运行的。

**虚拟地址空间的意义：**为了给每个进程描述一个完整的连续的虚假地址空间，供进程进行地址分配，而实际存储经过页表映射后可以实现数据在物理内存上的离散式存储，提供内存利用效率，并且通过页表进行内存的安全访问（可以保证进程的独立性）。

**页表如何映射虚拟地址和物理内存地址：**

**分页式：**

将地址空间进行分页管理，可以提供内存利用率

虚拟地址组成：页号+页内偏移。 页表：页号+物理块起始地址

**分段式：例如代码段、堆、栈的分类**

便于编译器进行代码管理，但并不能解决内存利用率的问题

将虚拟地址组成分为：段号+段内偏移，段表：段号+物理段起始地址

**段页式：操作系统的使用方式（先分段，再在每个段内进行分页）**

**缺页中断：提供虚拟地址经过页表映射访问物理内存的时候发现数据并没有在物理内存中，就会触发缺页中断。（将内存不够用时，将内存中不活跃的数据，放到硬盘上（放到swap分区），从而腾出内存空间，这就是*内存置换*）。触发缺页中断后，操作系统又会把硬盘中的数据加载到内存中。**

内存置换算法：LRU（最久未使用算法）

写时拷贝就是缺页中断，操作系统为子进程创建的缺页中断。

**例题：**

**例1：**

|  |
| --- |
|  |

虚拟地址03FF F180 H中，高20位为虚拟页号，低12位为页内偏移。

虚拟页号03FF对应的实际物理内存地址为0153，该虚拟页号的页内偏移为F180H.

则，对应的实际物理内存地址为：实际页号地址+页内偏移=0153F180 H

**例2：一个分页存储管理系统中，地址长度为 32 位，其中页号占 8 位，则页表长度是**

页号占8位，则页表项的个数=28个

##### 1、进程切换

进程切换时需要保护现场。

保护现场：当出现中断时，把CPU现在的状态都保存在寄存器中，随后转向其它任务，当任务完成后，从寄存器中取出相关状态，继续原进程。这里需要保存的状态包括：CPU寄存器里所有的内容（CPU上正在处理的数据）、页表和程序计数器。而全局变量不需要保存（共有数据不需要保存）。

#### 7、进程创建

**pid\_t fork()：*代码共享，数据独有***。创建后，谁先运行看操作系统的调度（大部分是父进程先运行）

进程的创建流程：写时拷贝技术（不改变虚拟地址，只改变物理映射关系）

**pid\_t vfork()：**创建一个子进程并且阻塞父进程（先运行子进程，直到子进程退出或程序替换之后，父进程再运行）。vfork创建子进程后，父子进程共用一个虚拟地址空间（共用同一个栈空间，如果同时运行，父进程退出后，子进程也被强制出栈，因此阻塞父进程，让子进程先出栈，这样就不会因为父进程出栈而导致栈混乱）。 fork一开始没有写时拷贝技术，所以vfork不需要开辟新空间，所以其效率比fork高。但fork具有写时拷贝技术后，vfork不再主流。

**子进程从父进程那继承了整个进程的虚拟地址空间**，包括：

1. 进程上下文
2. 进程堆栈
3. 内存信息
4. 打开的文件描述符
5. 信号控制设置
6. 进程优先级、进程组号
7. 当前工作目录
8. 根目录
9. 资源限制
10. 控制终端等

##### fork()例题：

|  |  |
| --- | --- |
|  | fork()后，子进程与父进程的区别：  1、父进程设置的锁，子进程不继承  2、父进程和子进程的ID不同  3、子进程的未决警告被清除  4、子进程的未决信号集设置为空集。  答案：  AC |
|  | 除去0号进程，一共创建19个进程 |

#### 8、进程终止

方法1：main函数中调用return;

方法2：系统调用接口： void \_exit(int status) 可以在任何地方调用（谁调用谁退出，status：进程退出后的返回值）：退出进程时，不会刷新缓冲区

方法3：库函数接口 void exit(int staus)

return 和 exit在退出前，都会刷新系统缓冲区（printf 不加 \n 时，数据会被写入缓冲区，不会立即刷新显示文件，这是为了IO流更高效）

echo $? 打印上一次进程的返回值

进程退出：正常退出和异常退出

进程的返回值：用1个字节来保存。

接口：

perror(const char\* s)：打印上一次系统调用失败的信息。s是自己定义的信息标识符

strerror(errno)：errno是错误编号，一共256个错误原因。返回对应的错误原因。

#### 9、进程等待

**等待状态的意义：**父进程获取子进程退出的原因，从而释放子进程资源，避免僵尸进程。

**操作方法：**

**pid\_t wait(int \*status)：**等待任一子进程退出（任一子进程退出都会处理，并不等待特定的子进程）。status：用于获取子进程退出返回值。

wait接口是一个阻塞函数。子进程没有退出，则一直等待。

阻塞：为了完成一个功能，发起调用。若当前不具备完成条件，则一直等待。

**pid\_t waitpid(pid\_t pid,int \*status,int options)：**可以等待任意指定进程退出，也可以等待一个指定的进程。可以是默认阻塞接口，也可以设置为非阻塞。

非阻塞：为完成一个功能，发起调用，若当前不具备完成条件，则保存返回。（没有子进程退出：返回0）

**操作参数：**

**pid**

-1：表示等待任意子进程

>0：表示等待指定子进程

<-1：不关注

**\*status：**用于获取子进程退出返回值。（不仅有进程退出返回值，还有一些其他信息）

*只有进程正常退出时，返回值才有意义*

int \*status;//错误操作，野指针

status共4个字节：

高16位，没用

第9-16位：进程退出返回值

1-8位：

最高位：core dump，程序运行状态存储的文件大小，默认为0，不占空间。

1-7位：异常信号值（判断信号是否正常退出，如果出现错误，这7位保存异常信号的号码）

**程序运行数据（status）取出操作：**

status&0x7f：取最低的7位 等效WIFEXITED(status)

(status>>8)&0xff：取中间8位 等效WEXITSTATUS(status)

**options:**

0：默认阻塞等待

WNOHANG：设置为非阻塞（要放到while(1)里，否则只运行一遍）

**返回值：**

>0：退出子进程的pid

=0：没有子进程退出

-1：出错

#### 10、程序替换

子进程的意义：程序可能存在危险，这会导致一个进程崩溃，创建子进程就是让子进程去干危险的事情，即使崩溃，对父进程也是没有影响的。

程序替换：加载一个新的程序到内存中，更新当前进程的页表映射信息。

代码操作：exec函数族

#include <unistd.h>` //头文件

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 env[]);

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

int execvp(const char \*file,char \*const argv[]);

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

参数：

path：新的程序文件路径

arg：程序的运行参数

env：程序的环节变量

**参数：**

参数1：新的程序文件路径名

参数2：程序的运行参数

参数3：程序的环境变量

参数最终以NULL结尾（最后一个参数一定要设置为NULL）

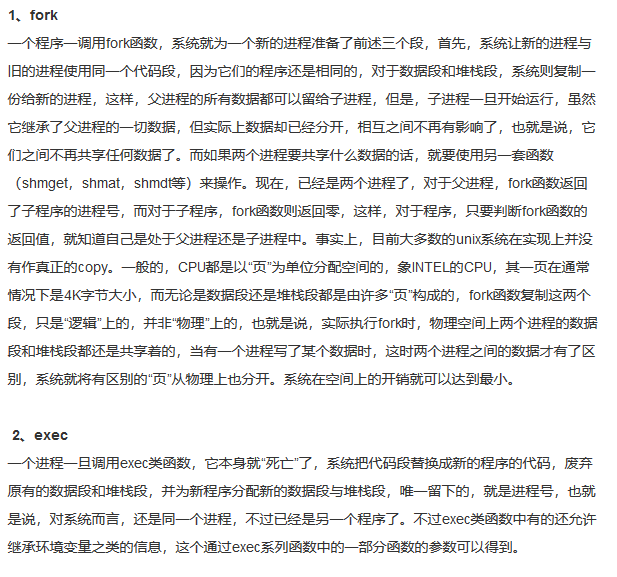
有p时，可以不带路径，指定必须在PATH环境变量指定的路径下

有e时，程序自己设定（覆盖式的）环境变量，没有e则使用默认已有的环境变量

l和v的区别：程序运行参数是不是不定参赋予还是字符数组赋予，赋值方式不同

##### 知识点

1、exec和fork都可以产生子进程，二者的区别是：



#### 11、其它

ps -ef：打印所有进程信息（获取所有进程的信息，然后输出到stdout）

### 2、基础IO（重操作）

输入输出：人机交互

基础IO：标准库IO接口，系统调用IO接口，文件描述符，IO流指针，动态库和静态库的生成和使用，文件系统初识

#### 1、标准库IO接口

##### fopen：打开文件

语法： FILE\* fopen(const char \*path,const char \*mode)

**path：**要打开文件的路径名称（缺省时，表示当前文件夹）

**mode：**文件打开方式：

r：打开一个文件，用于读（文件不存在会报错，不会自动新建文件）

r+：打开一个文件，用于读和写

w：打开一个文件，用于写（若文件存在，清空原文件内容。如果文件不存在，则自动创建）

w+：打开一个文件，用于读和写（若文件存在，清空原文件内容。如果文件不存在，则自动创建）

//以上，光标起始都在文件初识位置

a：追加只写。（数据总是追加到文件末尾（不会清空源文件内容），如果文件不存在，则自动创建）

a+：读和追加写（如果文件不存在，则自动创建）

//a和a+ 写时起始光标在文件末尾，读时光标在文件起始位置

/////以上 都是进行文本操作

b：对文件进行原始的二进制操作 （推荐对文件进行操作的时候，进行二进制操作。这样不会出错。系统调用接口都是二进制操作）

**返回值：**FILE\* （文件流指针，文件的操作句柄）

操作句柄：遥控器相对于电视机。

如果文件操作失败返回NULL。

##### fwrite：文件写入操作

size\_t fwrite(const void \*ptr,size\_t size,size\_t numemb,FILE \*stream)

ptr：要写入的数据

size：块大小

numemb：块个数

写入数据总长度=size\*numemb

stream：fopen返回的操作句柄，返回要写入的文件句柄

返回值：成功返回实际写入文件的完整块个数（如果某一块数据没有完整写入时，该块的返回值设为0）。 如果写入失败：返回0

所以：返回0不代表一定写入失败，还有可能时数据没有完全写入。

##### fread：文件读取操作

size\_t fread(void \*ptr,size\_t size,size\_t numemb,FILE \*stream)

ptr：一块内存空间的首地址，用于存放读取的数据

size：块大小

numemb：块个数

写入数据总长度=size\*numemb

stream：fopen返回的操作句柄

返回值：成功返回读取到的完整块个数。读取失败返回0，读到文件末尾也返回0，

所以：返回0代表：没有完整读取、读取失败、读到文件末尾（此时已读取的大小小于块大小）

所以：使用fread时，块大小设置为1

fread和fwrite使用时，块大小尽量设定为1，这样可以避免歧义。

feof：检测是否读到文件末尾

ferror：检测读取文件是否出错

##### fseek：跳转到文件的指定位置

int fseek(FILE \*stream,long offset, int whence)

stream：fope返回的操作句柄

offset：偏移量

whence：相对偏移的起始位置（断点续传，将大文件分为很多小文件，一次只传一个小文件，可以提高传输效率同时保证传输的完整性。该参数就是已传递小文件的序号）

whence中包含：

SEEK\_SET:起始位置

SEEK\_CUR：当前位置

SEEK\_END：末尾位置

这三个参数加上offset就可以定位到文件的任意位置。

返回值：成功返回0 失败返回-1

##### fclose：关闭文件

文件打开必须关闭，否则会造成资源泄露

int fclose(FILE \*fp)

fp：fopen返回的操作句柄

返回值：成功返回 失败返回

#### 2、系统调用IO接口

系统调用接口：操作系统（OS）向外提供的完成指定内核功能的接口。

##### 1、open

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

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

**参数介绍：**

**pathname：**要打开的文件名称

**flags：**打开方式

必选项：（三选一）

O\_RDONLY：只读

O\_WRONLY：只写

O\_RDWR ：可读可写

可选项：

O\_CREAT：文件不存在则创建（创建普通文件，在创建管道文件时，不可使用该参数），存在则打开（不清空）

O\_EXCL：与CREAT一起使用表示：文件存在则报错，不存在则创建（防止一个文件被多次创建）

O\_TRUNC：打开文件的同时清空原有内容。

O\_APPEND：写入文件时，写到文件末尾（追加写）

例：

w+ 等效于 O\_RDWR | O\_TRUNC | O\_CREAT

**mode：**权限

若使用了O\_CREAT，则一定要加上该权限 0777。（创建文件必须给权限，否则用户无法操作）

**返回值：**

成功返回非负整数（系统调用的操作句柄，也叫文件描述符）

失败返回-1

mode\_t umask(mode\_t mode)：设置当前进程的文件创建掩码。实际权限=给定权限(0777) &(掩码取反) 当设置为umask(0);时，实际权限=给定权限

umask只能在文件创建是起作用，如果文件已存在，其不起作用。

**知识点：**

**1、Linux下两个进程同时打开一个文件，则：**

（1）两个进程中分别产生两个独立的fd

（2）两个进程可以任意对文件进行读写操作，操作系统并不保证写的原子性

（3）进程可以通过系统调用对文件加锁，从而实现对文件内容的保护

（4）任何一个进程删除该文件，并不会使另一进程产生读写失败，因为进程打开文件后，进程已经具备文件的描述信息

（5）两个进程可以分别读取文件的不同部分而不受影响

（6）一个进程对文件长度的内容的修改，另一个进程可以立即感知

##### 2、write

ssize\_t write(int fd,const void \*buf,size\_t count);

ssize\_t ：有符号整形

size\_t ：无符号整形

参数：

**fd：**open返回的操作句柄（文件描述符）

**buf：**要写入的数据

**count：**要写入的数据长度

**返回值：**

成功返回实际写入的数据长度。失败返回-1

##### 3、read

ssize\_t read(int fd,void \*buf,size\_t count);

参数：

**fd：**open返回的操作句柄（文件描述符）

**buf：**存放读取到数据的缓冲区地址

**count：**要读取的数据长度

**返回值：**

成功返回实际读取到的数据长度。失败返回-1

##### 4、lseek

int lseek(int fd,off\_t offset, int whence)

fd：open返回的操作句柄

offset：偏移量

whence：相对偏移的起始位置（断点续传，将大文件分为很多小文件，一次只传一个小文件，可以提高传输效率同时保证传输的完整性。该参数就是已传递小文件的序号）

返回值：

成功返回从文件起始位置到偏移位置的偏移量。失败返回-1

##### 5、close

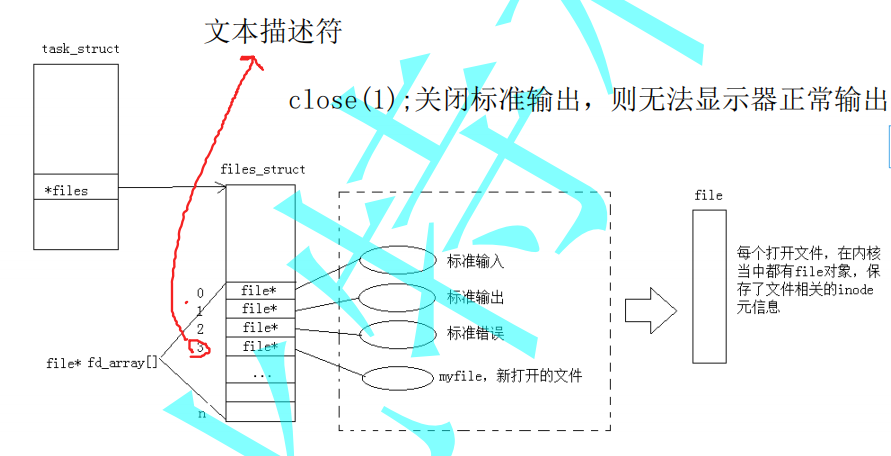
int close(int fd)：关闭文件

#### 3、文件描述符（重定向）

**文件描述符：**非负整数，系统调用IO接口操作句柄。

其本质上：文件描述符实际上是内核中进程打开的文件对应的描述信息的下标。进行IO操作时，通过下标就能找到对应的文件内容信息。

**文件描述符作用流程：**



**文件描述符分配规则：**

从小往大进行分配（最小未使用规则）。（Linux进程默认情况下会有3个缺省（默认）打开的文件描述符，分别是标准输入0（关闭后，无法输入）， 标准输出1（关闭后，无法输出）， 标准错误 2（关闭后，无法输出错误信息）。所以，自己的文件最小的文件描述符为：3）

close(1); 关闭标准输出，则无法通过任何输出文件进行输出。并且此时1号描述符被标记为未使用状态，下次再打开文件，其文件描述符就是1。

如果关闭标准文件描述符后：

关闭标准输出后，1号描述符指向新的文件。如果此时有打印输出请求，则数据被写入1号描述符所指向的文件。（不会立即刷新，因为\n刷新缓冲区是针对标准输出文件的。普通文件需要fflash）

##### 1、重定向（改变本来要写入标准输出的数据流向）

标准输出重定向就是将原本要写入标准输出文件的数据，不再写入标准输出，而是写入到指定的文件中。

重定向后，运行下列代码，可将ls输出的信息写入指定文件。

ls >> test.txt：追加输出

ls > test.txt：清空原内容，再输出

重定向能够作用于其它进程（子进程）的原理：

进程PCB中的file\_struct不会随着程序替换的发生而发生改变。所以，创建子进程后，立马进行重定向，然后再在子进程中进行程序替换，这样重定向就可以作用于子进程。

#### 4、IO流指针

文件流指针(库函数操作句柄，封装了文件描述符)：库函数IO接口的操作句柄FILE \*fp

文件描述符：系统调用IO接口的操作句柄int fd

库函数和系统调用接口的关系：库函数封装了系统调用接口。

文件流指针是一个结构体，因此可以多个文件流指针封装同一个文件描述符。

缓冲区和系统调用没关系。

#### 5、动态库和静态库的生成和使用

动态库于静态库的生成与使用：

**动态链接：**链接动态库，将库中所使用的函数符号信息表记录在可执行程序中（生成可执行程序中并没有把使用的库函数的实现拿过来）

**静态链接：**链接静态库，将库中所适用的函数实现信息直接写入可执行程序，运行时不需要依赖库的存在，有可能会造成代码冗余。

##### 1、动态库生成

首先将源码经过编译汇编后生成机器指令，然后生成动态库

**相关命令：**

gcc -fPIC-c testlib.c -o testlib.o //fPIC作用是产生位置无关代码

gcc --shared testlib1.o testlib2.o testlib3.o -o libtestlib.so //多个.o文件以空格为间隔以此排列，最终生成一个动态库

##### 2、静态库生成

ar -cr libtestlib.a child1.o child1.o child1.o child1.o //libtestlib.a就是静态库

##### 3、链接

gcc main.c -o main -libtestlib.o

gcc main.c -o main -libtestlib.so

gcc main.c -o main -libtestlib 等效于gcc main.c -o main -libtestlib.so //gcc优先链接动态库

***库函数不能有主函数。***

##### 4、库的使用

ldd main 查看当前程序依赖的动态库

###### （1）生成可执行程序时使用

使用gcc -[指定要链接的库（标准C库是自动链接的）] //写入头文件

为了gcc能在指定路径下找到头文件：

（1）将头文件放到指定路径下 /lib64

（2）设置环境变量 $LIBRARY\_PATH //库文件默认查找位置

export LIBRARY\_PATH=$LIBRARY\_PATH:./ 编译时将当前目录添加到环境变量中

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:./ 运行时将当前目录添加到环境变量中

（3）gcc main.c -o main -L./ -L可以指定当前目录为库路径（只能在编译是可行，运行时，无法指定库路径。-L选项通常用于链接静态库，因为静态库没有运行依赖）。-l命令链接指定的库文件。

###### （2）运行可执行程序时使用（仅针对使用动态库）

#### 6、文件系统初识

### 3、进程间通信（IPC）

操作系统为用户提供的用于实现进程间通信的方式。

**原因：**

进程间是无法直接通信的。因为每个进程都有独立的虚拟地址空间，访问的都是自己的虚拟地址，所以进程间具有独立性。

**解决方案：**

操作系统提供的几种进程间通信方式：**管道**、**共享内存（**system V**）**、消息队列（system V）、信号量（标准：system V和POSIX）

#### 1、管道

管道作用：传输数据资源

管道特点：半双工通信（可以选择方向的单工通信）

管道本质：在内核中开辟的一块缓冲区（内核空间中的一块内存，只能通过系统调用进行，这也是为了保证操作系统的安全）。不同进程都有自己独立的虚拟地址空间（虚拟用户内存空间），但所有的进程，都具有相同的内核空间（内核空间是共享的），所以，可以通过内核空间来实现通信。

##### 1：管道的分类

###### （1）命名管道

具有标识符的内核缓冲区，可以被其它进程找到，所以可用于同一主机上任意的进程间的通信。

命名1的标识符就是一个可见于文件系统的管道类型文件。

多个进程通过打开同一个管道文件，访问同一块内核中的缓冲区实现通信。

###### （2）匿名管道

没有标识符的内核缓冲区（没有标识符意味着其它进程无法通过标识找到该内核缓冲区），匿名管道只能用于具有亲缘关系的进程间的通信。

##### 2：管道的实现

Linux下管道也是当做文件进程操作的，所以通过IO操作就可以实现管道。

指令：

（1）int pipe(int pipefd[2]) //针对匿名管道

参数：  
pipefd[0]：用于读（从管道读取数据）

pipefd[1]：用于写（往管道输入数据）

返回值：成功返回0，失败返回-1

（2）mkfifo [filename] 创建一个命名管道文件（创建后并不一定会在内核空间中开辟缓冲区，直到该文件被访问或写入数据时，才会开辟缓冲区）

**头文件：<sys/stat.h>**

函数：int mkfifo (const char \*pathname,mode\_t mode)

pathname：文件名称

mode：权限

返回值：成功返回0，失败返回-1

**命名管道和命名管道文件的区别：**

命名管道文件是通过普通文件实现的，其本质是缓冲区标识。而命名管道本质是内核中的一块缓冲区。并且进程无论向管道中写入多少数据，管道文件的大小都不会变好，因为其本质只是标识符。

##### 3：注意点

1、若管道中没有数据，则read就会阻塞。若管道中数据满了，write就会阻塞。

2、若管道所有的读端被关闭，则再调用write则会触发异常

3、若管道写端都被关闭，则继续read则会返回0，表示管道中不再有数据。

4、管道提供字节流传输服务：有序的（先进先出）、可靠的（写入和读出肯定成功）、基于连接的（有读必有写）流式传输（读完，数据立马从缓冲区清除）。

5、管道的阻塞：当只进行读或写操作时，操作系统不会创建缓冲区，这就会造成阻塞（程序一直等待写入的数据被读出）

6、自带同步与互斥：

互斥（概念）：通过同一时间进程对临界资源（公共资源，大家都可以访问的资源）的唯一访问实现访问操作安全。

临界区：对临界资源访问的这段代码。

同步（概念）：通过一些条件判断让进程对临界资源的访问更加合理有序。

互斥（体现）：对管道进行写入操作大小不超过PIPE\_BUF（4096字节），则保证操作的原子性（不可分割，即操作不可打断，是安全的）。

同步（体现）：若管道没有数据，则read阻塞。若管道满了，则write阻塞。

7、**管道的生命周期随进程**（不人为干预的情况下），即即使一个进程删除了管道文件，但依然可以通信。

8、管道的通信方式：半双工通信

##### 4、管道符

连接两个命令，将前面命令的打印结构传输给后面的命令进行处理。

管道符的实现就是通过匿名管道的方式实现的。

#### 2、共享内存

**共享内存：**共享区（用户内存中一块）的一块物理内存，用于实现进程间的数据共享（数据共享不是数据传输）。共享内存是进程间最快的通信方式，因为共享内存直接通过虚拟地址空间直接访问物理内存实现数据共享，而其它方式需要进行数据拷贝和粘贴。

**原理：**开辟一块物理内存，多个进程将映射了同一块到自己的虚拟地址空间，然后就可以通过各自的虚拟地址空间访问共享内存来实现数据共享。

**操作流程：**先创建或打开共享内存。然后将共享内存映射到自己进程的虚拟地址空间中。最后通过映射的虚拟地址进行各自内存操作。（凡是对物理内存操作的函数，都可以操作共享内存）。当不再共享数据时，各自进程可以解除映射关系。当所有的进程都不再映射共享内存时（共享内存中存在计数器，只有共享内存中映射连接数为0时，才可以删除。如果共享内存映射连接数不为0，进行删除时，共享内存不会直接删除，会变为即将删除状态，该状态下，任何进程都不能再映射该共享内存,当映射数为0时，自动删除），需要手动删除共享内存（共享内存的生命周期随内核，除非操作系统重启，否则其一直存在）

ipcs -m ：查看共享内存

ipcs -s ：查看信号量

ipcs -q ：查看队列信息

ipcs -t：输出信息的详细变化时间

ipcs -a：默认的输出信息，打印出当前系统中所有的进程间通信方式的信息

ipcs -p：输出IPC方式的进程ID

ipcs -c：输出IPC方式的创建者/拥有者

ipcs -u：输出当前系统下IPC各种方式的状态信息（输出共享内存、消息队列和信号量）

ipcrm：在ROOT用户下运行，移除一个消息对象（共享内存或信号集），同时会将与IPC对象相关连的数据一起删除。

ipcrm -m： 共享内存ID：将共享内存标记为被销毁状态

ipcrm -s：移除标识的信号

ipcrm -q：移除标识的消息队列

##### 1、操作接口

###### （1）创建共享内存

int shmget(key\_t key,size\_t size,int shmflg)

key：标识符（多个进程通过相同的标识符打开同一块共享内存），可以ftok生成标识符

size：共享内存创建时所开辟的空间大小（以内存页为单位：4096字节）

shmflg：打开方式（IPC\_CREAT|IPC\_EXCL）+创建权限（0664）

返回值：成功返回非负整数（共享内存的操作句柄），失败返回-1

###### （2）映射

void \*shmat(int shmid,const void \*shmaddr,int shmflg)

shmid：shmget返回的操作句柄

shmaddr：映射地址（通常置为NULL，让操作系统为我们选择共享内存地址）

shmflg：映射成功后的访问方式（SHM\_RDONLY:只读 0:读写）

返回值：成功返回映射后的首地址（共享内存首地址），失败返回-1（void \*-1）

###### （3）解除映射

int shmdt(const void \*shmaddr)

shmaddr:共享内存首地址

返回值：成功返回0，失败返回-1

###### （4）删除共享内存

shmctl(int shmid,int cmd,struct shmid\_ds \*buf)

shmid：shmget返回的操作句柄

cmd：设置为IPC\_RMID（标记共享内存被销毁（禁止映射））

buf：用于设置或获取共享内存信息，不用置为NULL

返回值：成功返回0，失败返回-1

###### （5）知识点

1：共享内存是 **非进程安全**的

2：共享内存的生命周期随进程，即使没有映射，只要不删除，一直都存在。

#### 3.消息队列

**原理**：内核中的一个优先级队列，多个进程通过访问同一个队列，向队列中添加或获取节点进行而实现进程间数据块传输。

特性：自动同步和互斥，生命周期随内核

操作：msgget

#### 4、信号量

本质：内核中的一个计数器

作用：用于实现进程间的同步和互斥（协调进程对临界资源的访问）

p操作：计数器-1（判断：若计数器<0，则阻塞进程）

v操作：计数器+1（唤醒一个进程的阻塞）

通过自身的计数器对资源进行计数，通过计数判断进程对资源的获取是否合理，不合理则阻塞。等待产生一个资源之后，唤醒阻塞的进程。

**同步的实现：**

通过计数器对资源进行计数，在获取资源前进行p操作。

产生资源后，进行v操作。

互斥的实现：

计数器为1，表示资源只有一个，进程访问资源之前进行p操作，访问完毕之后进行v操作。

## Linux网络编程

|  |
| --- |
|  |

底行模式：vnew

## 项目实例



### 1、初阶项目

#### 1、实现迷你shell

自主实现一个minishell：迷你shell（命令行解释器）

功能：将函数及参数传递给内核。首先捕捉键盘输入，然后解析输入（获得命令名称+运行参数），然后创建子进程，然后在子进程中进行程序替换（将子进程运行的程序替换为对应命令名称的命令程序），父进程等待子进程退出（否则会出现僵尸进程）

chdir：改变工作路径

cd是shell内建命令，需要自己实现

##### 1、流程

###### 1、捕捉键盘输入

**scanf：**以空格为结束符（不常用）

**gets()：**从标准输入文件（键盘也是一个文件）中读取一行数据

语法： char \*gets(char \*s)

gets()存在越界访问的危险，不建议用

**char \*fgets(char \*s, int size, FILE \*stream)** 可以通过size限制传入数据的大小，避免了越界访问

（1）输入字符串的读取、解析（剔除空格等无效字符）

isspace() :检测空白字符（空格、换行、回车、制表符） 这里不用

###### 2、子进程创建并进行程序替换

注意：子进程替换失败，如果不手动让其退出，那么子进程就变为了父进程。

###### 3、父进程等待子进程退出

##### 2、项目源码

##### 3、备注

**[用户名@主机名 当前所在目录]$**

fflush(stdout) //刷新标准输出文件，在没有换行的情况下显示输出（手动刷新标准输出文件的缓冲区）

fgets(buf,1023,stdin); 从标准输入文件中，读取1024字节的内容，放到buf中

buf[strlen(buf)-1]=’\0’;将字符串最后一个字符的回车替换为\0

printf() 会把内容先放到缓冲区

标准输出文件比较特殊：具有行缓存特性（所以，加\n可以实现立即打印。如果不加，只能等到进程结束时，缓冲区自动刷新）