目录

[一、 进程 1](#_Toc160389955)

[1. 基本概念 1](#_Toc160389956)

[2. 结构特征 1](#_Toc160389957)

[3. 查看进程信息 1](#_Toc160389958)

[二、 创建进程 2](#_Toc160389959)

[1. 进程复制 2](#_Toc160389960)

[2. 子进程“偷梁换柱” 2](#_Toc160389961)

[三、 进程退出 4](#_Toc160389962)

[1. 函数 4](#_Toc160389963)

[2. exit()和\_exit()区别 4](#_Toc160389964)

[四、 等待子进程退出 5](#_Toc160389965)

[1. wait()函数 5](#_Toc160389966)

[2. waitpid函数 6](#_Toc160389967)

[五、 进程的生命周期 7](#_Toc160389968)

[1. 就绪态和执行态 7](#_Toc160389969)

[2. 僵死态 7](#_Toc160389970)

[3. 停止态 8](#_Toc160389971)

[4. 睡眠态 8](#_Toc160389972)

[5. 描述宏 8](#_Toc160389973)

[六、 进程组、会话、终端 9](#_Toc160389974)

[1. 进程组 9](#_Toc160389975)

[2. 会话 9](#_Toc160389976)

[3. 终端 10](#_Toc160389977)

# 进程

## 基本概念

程序：静态文件

进程：运行着的实体

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，包括文本区域（text region）、数据区域（data region）和堆栈（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。

进程是一个“执行中的程序”。程序是一个没有生命的实体，只有处理器赋予程序生命时（操作系统执行之），它才能成为一个活动的实体，我们称其为进程。

## 结构特征

进程由程序、数据和进程控制块（PID）三部分组成。

## 查看进程信息

进程关系：pstree（进程之间存在“父子关系”“兄弟关系”这样的亲缘关系）

进程的身份证PID：ps -ef

# 创建进程

## 进程复制

函数原型：pid\_t fork(void);

fork() 这个函数很特殊，成功创建子进程后居然有两个返回值，给父进程返回子进程pid，给子进程返回 0，如果创建失败那么就返回 -1。

fork函数执行完后会复制一份原来的进程（创建进程）。

fork函数后面的代码会执行两遍。fork之后父进程先执行还是子进程先执行不确定，取决于内核所使用的调度算法。

## 子进程“偷梁换柱”

当用fork函数创建新的子进程后，子进程往往要调用一种exec函数以执行另一个程序。当程序调用一种exec函数时，该进程执行的程序完全替换为新程序，而新程序则从其main函数开始执行。因为调用exec并不创建新进程，所以前后的进程ID并未改变。exec只是用磁盘上的一个新程序替换了当前进程的正文段、数据段、堆段和栈段。

有7种不同的exec函数可供使用，它们常常被统称为exec函数。

**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[]);

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

\*path：可执行文件的路径名。

\*arg：可执行程序所带的参数，第一个参数为可执行文件名字，没有带路经且arg必须以NULL结束。

\*file：如果参数file中包含/，则就将其视为路径名，否则就按PATH环境变量，在它所指定的各目录中搜寻可执行文件。

函数名中的字符可以帮助理解和分辨：

l（list）表示以参数列表的形式调用（需要指定绝对路径执行）。

v（vector）表示以参数数组的方式调用

e（environment）表示用户提供自定义环境变量，当调用execve函数时，操作系统首先根据filename指定的路径和名称找到对应的可执行文件。然后，操作系统创建一个新的进程，并将该可执行文件加载到新进程的内存空间中。接下来，操作系统将新进程的参数和环境变量设置为argv和envp指定的内容。最后，操作系统启动新进程的执行，从新程序的入口点开始执行代码。

p（path）表示PATH中搜索执行的文件，如果给出的不是绝对路径就会去PATH搜索相应名字的文件，如PATH没有设置， 则会默认在/bin,/usr/bin下搜索。

* 返回值：

如果执行成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno 中。

* 注意事项

l、v后缀必须任选其一使用，p、e任选其一。

有可能执行失败：文件路径错误；没有加NULL结尾；新程序没有执行权限

# 进程退出

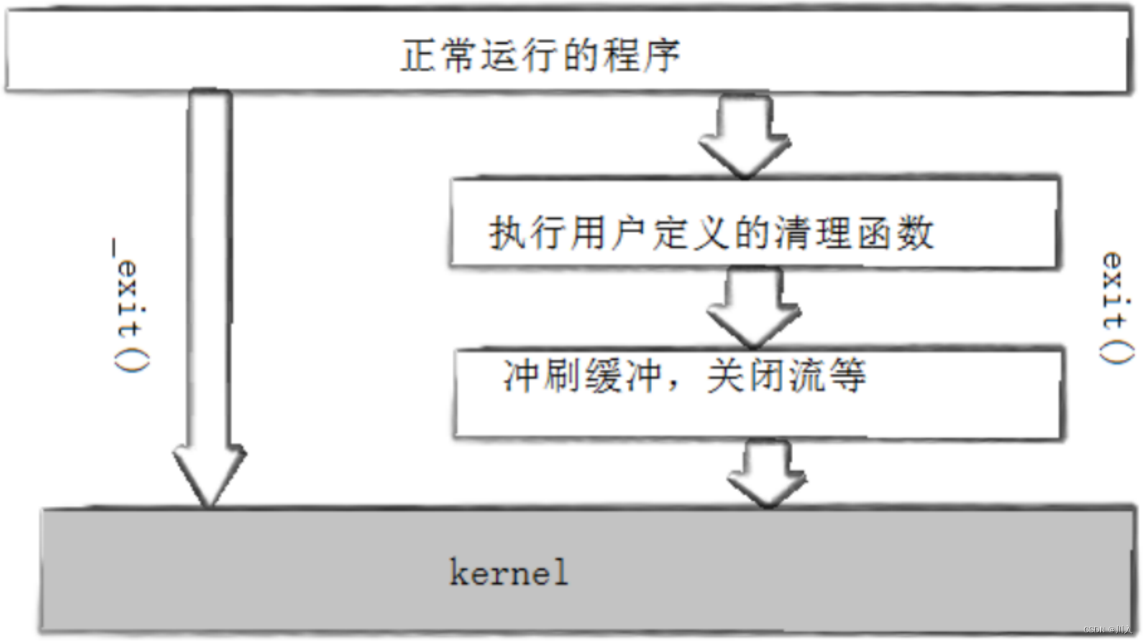
## 函数

exit()函数用于正常退出进程。当调用exit()函数时，它会执行一些清理操作（如关闭打开的文件描述符）并终止进程。它还通过返回一个退出状态码将控制权返回给操作系统。exit()函数可用于任意函数中，通过调用它可以使整个进程退出。

\_exit()系统调用是直接终止进程。当调用\_exit()函数时，进程立即终止，而不执行任何清理操作。与exit()不同，\_exit()函数不返回任何状态给操作系统，而是直接终止进程。通常，\_exit()函数用于异常情况或在需要立即终止进程而不进行清理操作的情况下。

return语句用于退出函数或方法，并将控制权返回到调用该函数或方法的位置。return语句只能用于函数或方法的内部，而不能使整个进程退出。当函数或方法中的所有代码都执行完毕或遇到return语句时，该函数或方法的执行将结束，控制权将返回给调用者。

## exit()和\_exit()区别



# 等待子进程退出

## wait()函数

pid\_t wait(int \*status);

功能：等待任一子进程终止，如果子进程终止了，此函数会回收子进程的资源。

调用wait函数的进程会被挂起(阻塞)，直到它的一个子进程退出或收到一个不能被忽视的信号时才被唤醒。若调用进程没有子进程或它的子进程已经结束，该函数立即返回。

参数：函数返回时，参数status中包含子进程退出时的状态信息。

子进程的退出信息在一个int中包含了多个字段，用宏定义可以取出其中的每个字段。

返回值：如果执行成功则返回子进程的进程号。出错返回-1，失败原因存在errno中。

**int** **main**()

{

pid\_t result;

**int** status;

result = fork();

**if**(result == -1){

**printf**("error!!!\r\n");

}

**if**(result == 0){

**printf**("son!!!\r\n");

**exit**(0);

}

**else**{

wait(&status);

**if**(WIFEXITED(status) == 1){

**printf**("exit value:%d\r\n",WEXITSTATUS(status));

**return** 0;

}

}

}

## waitpid函数

pid\_t waitpid(pid\_t pid, int \*status, int options);

功能：等待指定子进程终止，如果子进程终止了，此函数会回收子进程的资源。

返回值：如果执行成功则返回子进程ID。出错返回-1，失败原因存于errno中。

---- 从本质上讲，系统调用waitpid和wait的作用是完全相同的，但waitpid多出了两个可由用户控制的参数pid和options，从而为我们编程提供了另一种更灵活的方式。

pid：当参数pid取不同的值时，有不同的意义：

1> pid>0时，只等待进程ID等于pid的子进程，不管其它已经有多少子进程运行结束退出了，只要指定的子进程还没有结束，waitpid就会一直等下去。

2> pid=-1时，等待任何一个子进程退出，没有任何限制，此时waitpid和wait的作用一模一样。

3> pid=0时，等待和该进程在同一个进程组中的任何子进程，如果某个子进程已经加入了别的进程组，waitpid不会对它做任何理睬。

options：目前在Linux中只支持WNOHANG和WUNTRACED两个选项，可以用”|“运算符把它们连接起来使用，如：

ret = waitpid(-1,NULL,WNOHANG|WUNTRACED);

WNOHANG，表示即使没有子进程退出，它也会立即返回，不会像wait那样永远等下去。

WUNTRACED，与跟踪调试有关，极少用到。

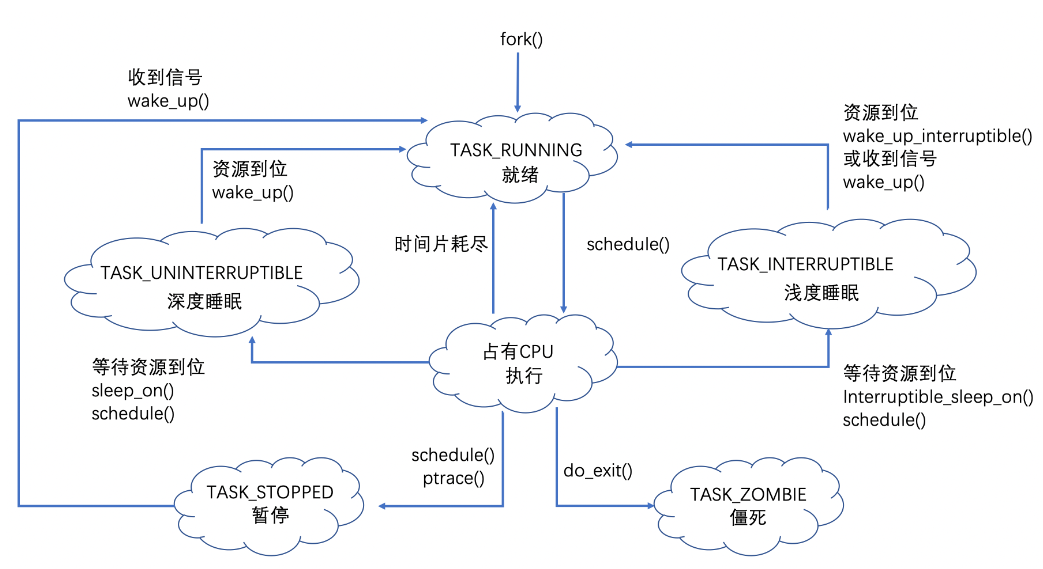
waitpid的返回值比wait稍微复杂一些，一共有3种情况：

>> 正常返回的时候，waitpid返回收集到的子进程的进程ID；

>> 如果设置了选项WNOHANG，而调用中waitpid发现没有已退出的子进程可收集，则返回0；

>> 调用中出错，则返回-1，这时errno会被设置成相应的值以指示错误所在。例如：当pid所指示的子进程不存在，或此进程存在，但不是调用进程的子进程，waitpid就会出错返回，这时errno被设置为ECHILD。

# 进程的生命周期



一个进程被fork出来后，进入就绪态；当被调度到获得CPU执行时，进入执行态；如果时间片用完或被强占时，进入就绪态；资源得不到满足时，进入睡眠态(深度睡眠或浅度睡眠)，比如一个网络程序，在等对方发包，此时不能占着CPU，进入睡眠态，当包发过来时，进程被唤醒，进入就绪态；如果被暂停，进入停止态；执行完成后，资源释放，此时父进程wait4还未收到它的信号，进入僵死态。

## 就绪态和执行态

就绪态：进程准备就绪，等待被CPU执行时的状态。即进程已经具备运行条件，但是CPU还没有分配过来，需等待被CPU调度到，进入执行态。

执行态：占用CPU，在CPU上执行。

## 僵死态

僵死态：进程结束时，其他资源都释放，只留下了task\_struct，等待父进程wait4函数处理时的状态。

一个进程如果进入僵死态时，它占用的系统资源都已释放了，只是保留了task\_struct等待父进程处理。

如果一个进程一直是僵死态，通过kill是无法杀掉的，除非将它的父进程杀掉，它才会消失。

系统中有僵尸态的进程对系统资源来说没影响，可能是你写的程序有问题，未正常退出，使得父进程无法处理。

## 停止态

人为地暂停进程时的状态。

在linux中，按ctrl+z，当前终端下运行的进程就会进入停止态。按fg或bg恢复该进程的运行。fg与bg的区别是：fg是前台运行，bg是后台运行。

有个工具叫cpulimit，它限制进程的原理就是不断地停止进程，恢复进程，最终达到限制进程资源的效果

## 睡眠态

睡眠态分浅度睡眠和深度睡眠，区别在于：

* 浅度睡眠：可以被资源和信号唤醒
* 深度睡眠：只能被资源唤醒，比如你挂载一个NFS，当NFS服务器挂了时，你对这个挂载做不了任何操作，比如用kill命令发送任何信号都无效，处理的方法是：1.等待NFS服务器恢复；2.重启你的服务器。

## 描述宏

TASK\_RUNNING:就绪/运行状态 TASK\_INTERRUPTIBLE:可中断睡眠状态

TASK\_UNINTERRUPTIBLE:不可中断睡眠状态

TASK\_TRACED:调试态 TASK\_STOPPED:暂停状态

EXIT\_ZOMBIE:僵死状态 EXIT\_DEAD:死亡态

# 进程组、会话、终端

## 进程组

* 作用：

对相同类型的进程进行管理。

* 进程组的诞生

在shell里面直接执行一个应用程序，对于大部分进程来说,自己就是进程组的首进程，进程组只有一个进程；

如果进程调用了fork函数。那么当子进程同属一个进程组,父进程为首进程；

在shell中通过管道执行连接起来的应用程序,两个程序同属一个进程组。第一个程序为进程组的首进程；

* 进程组id

pgid,由首进程pid决定。

## 会话

* 作用

管理进程组

* 会话的诞生

调用setsid函数，新建一个会话，应用程序作为会话的第一个进程，称为会话首进程

用户在终端正确登录之后，启动shell时linux系统会创建一个新的会话，shell进程作为会话首进程

* 会话id

SID,会话首进程id

* 前台进程组

前台进程是与用户直接交互的进程。在任何时刻，只有一个进程组可以在前台运行。这个进程组可以从终端接收输入，向终端发送输出。如果一个前台进程正在运行，终端会被阻塞，即用户无法在终端进行其他操作，直到这个前台进程完成。

* 后台进程组

后台进程是在后台运行，不占用用户终端的进程。它们不会阻塞用户终端，用户可以在同一终端启动新的前台或后台进程。后台进程可以向终端发送输出，但通常不能从终端接收输入。

jobs：查看有哪些后台进程组。

* 前台进程与后台之间如何切换

使用 ctrl+z 可以将一个正在前台执行的进程挂起，并把它转到后台。

使用命令 bg 可以将一个被挂起的进程转到后台并继续执行。

使用命令 fg+job,job由jobs命令查看可以将一个在后台运行或被挂起的进程转到前台。

当启动一个进程时，在命令行的最后加上 & 符号，便可以使进程在后台运行。

## 终端

终端就是处理计算机主机输入输出的一套设备，它用来显示主机运算的输出，并且接受主机要求的输入。

* 物理终端（依赖物理设备）

直接连接在主机上的显示器、键盘鼠标统称。

串口终端

LCD终端

* 伪终端（不依赖物理设备）

ssh远程终端

桌面系统启动的终端

* 虚拟终端

附加在物理终端之上，用软件方式虚拟实现。

Linux自带的。