# 6.1 程序和进程

程序：数据结构+算法

数据结构：组织和存储数据的方式，例如：数组、链表、栈、队列等等

算法：程序中用来处理这些数据的具体方法，例如：查找、排序、搜索、遍历等动作。

当程序运行起来后，就变成了进程。

**程序和进程的区别**

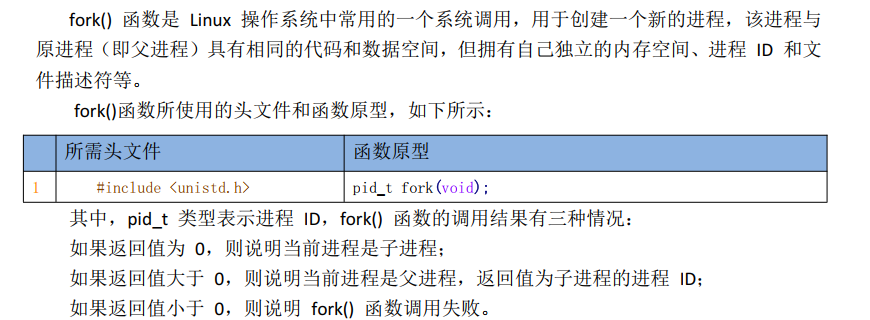


# 6.2 进程的创建

为什么要创建子进程？

* 并发处理，避免单个进程阻塞的问题
* 保护父进程，危险操作放在子进程
* 资源隔离
* 多任务处理，提高效率
* 进程间通信（管道，共享内存，信号量等）

如何创建？fork（）函数 / vfork（）



值得注意的一点：

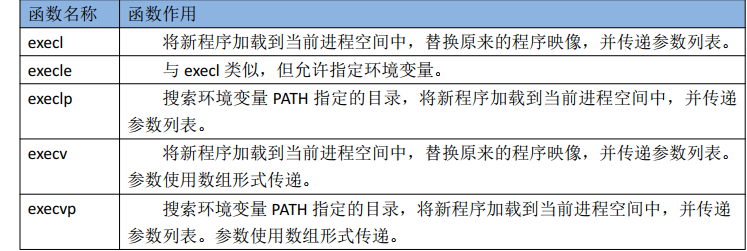
使用fork()函数时，会复制父进程的资源给子进程，这会花费一定的时间。为了减少延迟，故会先执行父程序，再执行子程序。

既然会复制，那肯定会浪费时间。当需要复制大量资源时，会浪费资源和性能，为此linux内核提供了写时复制。即子进程需要修改父进程地址空间的某些页时，再将这些页复制到子进程的地址空间里。

vfork()函数是一种比较特殊的进程创建函数，作用也是创建一个子进程，子进程与父进程共享同一个地址空间，包括程序代码、数据、堆栈等。只有在子进程调用 exec()函数族时，新的程序代码和数据才会被加载到子进程的地址空间中。 在 vfork()函数调用成功之后， 父进程和子进程共享同一个地址空间， 但是子进程会阻塞父进程， 直到它调用 exec()函数或 \_exit()函数为止。

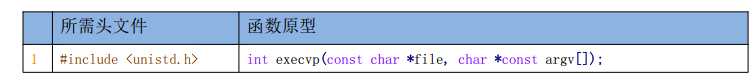
# 6.3 程序的执行

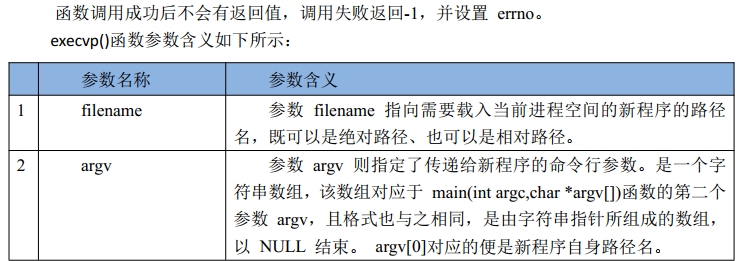
某个进程去运行其他程序，可以使用exec类函数族（这一族函数的共同点：取代当前进程的映像）



子进程和exec函数往往一起使用，可实现一个程序中创建多个进程，广泛运用于操作系统的进程管理和多进程编程中。

以execvp（）函数为例进行讲解

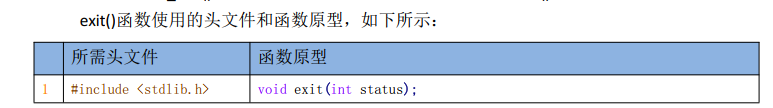




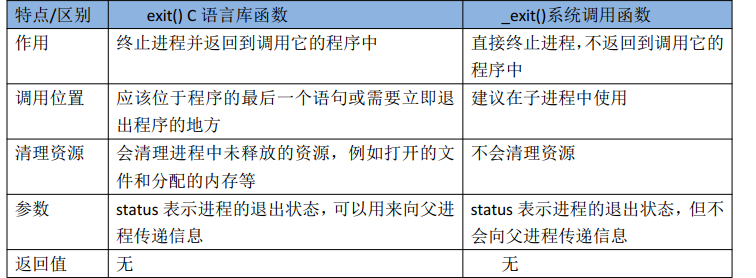
# 6.4 进程的退出 exit() \_exit()

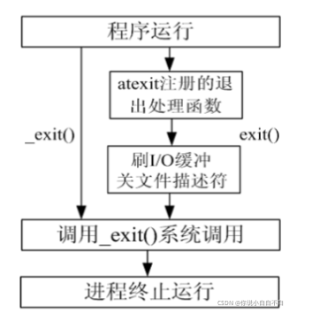
**进程执行完所有的任务后退出进程**

* 释放资源
* 防止僵尸进程（一种已经结束了执行，但是它的父进程没有调用wait()和waitpid（）函数回收它的状态信息的进程）
* 程序的正确性和稳定性



exit()函数和\_exit函数的区别：

**exit()函数会刷新缓冲区，\_exit() 不会刷新缓冲区**



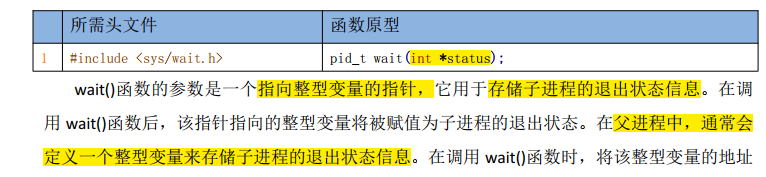
子进程建议使用\_exit()函数，不会像exit()函数做一些清理工作（如：文件描述符的关闭，导致父进程无法再使用这些文件描述符）， 而在父进程则建议使用exit() 函数，刷新缓冲区，执行清理工作，保证程序的正前性和稳定性。

# 6.5 等待子程序的中止 wait()

**（1）为什么要等？**

* 获取子程序的状态，判断子程序是否执行成功
* 回收子程序的资源，防止僵尸进程
* 实现进程的同步和通信。父进程可以等待子进程完成之前的任务后，分配新任务。

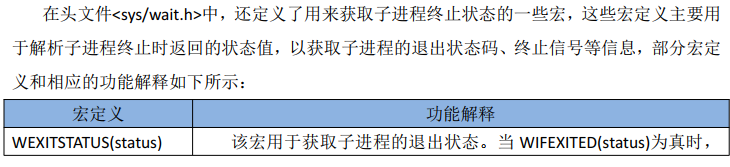
**（2）Wait函数的使用**

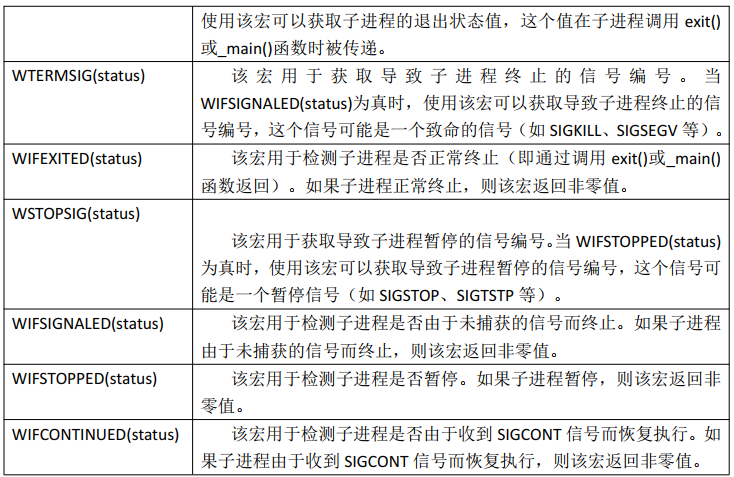


wait()函数会将退出状态信息存储在该整型变量中，父进程就可以通过该信息来判断子进程是否运行成功。指针为空指针时，wait函数则不会返回子进程的退出状态信息，wait()函数的返回值为子进程的进程ID，即pid。

wait函数只能获取到某一个子进程的退出状态信息，若想获取多个子进程的退出状态信息，那就需要多次调用wait函数。

**(3) 宏来解析子进程的返回状态**





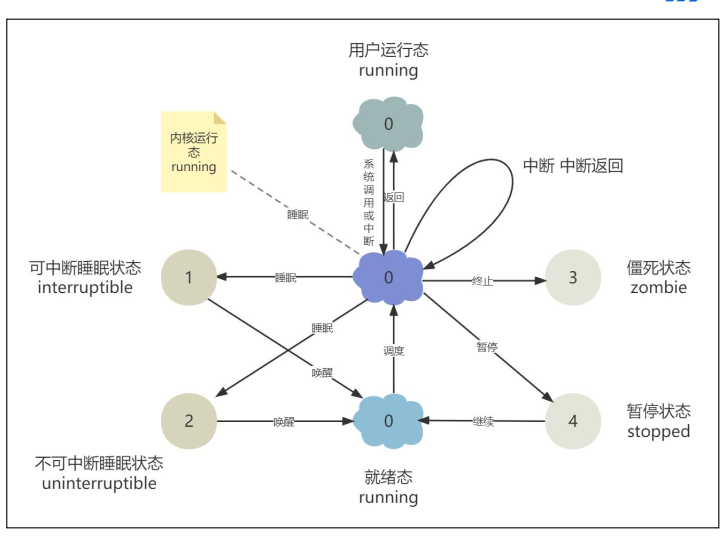
# 6.6 查看进程状态

**（1） 进程状态**

**五种状态：**

* **运行状态**
* **可中断睡眠状态（通常由于系统调用而进入的）**
* **不可中断睡眠状态（进程等待硬件设备或某些资源的释放进入的）**
* **暂停状态**
* **僵死状态**

**5个进程状态的相互转换关系**



**（2） ps命令和top命令**

最常用的“ps -aux” ， 表示显示当前系统中所有进程的详细信息， 包括进程的所有者、 CPU 占用率、 内存占用情况、 进程状态等

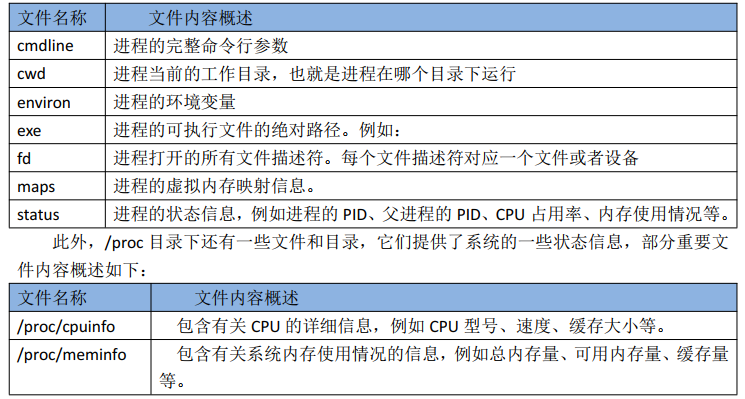
常用的 Linux 系统性能监控工具， 可以实时查看系统的 CPU、 内存、 I/O 等资源的使用情况， 同时还可以查看当前系统中运行的进程和它们的相关信息， 是系统管理员和性能优化工程师必不可少的工具之一。

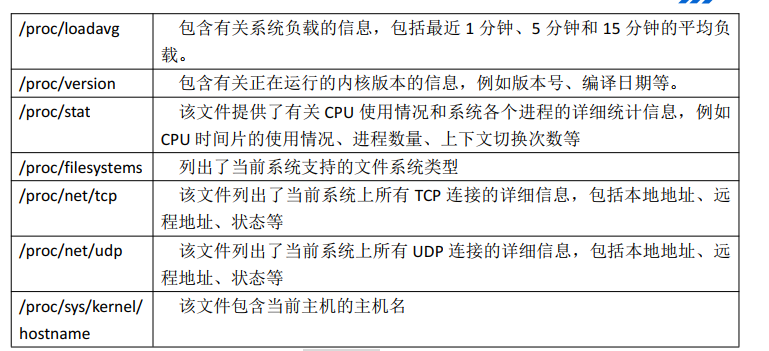
**(3) proc虚拟文件系统**

特殊的文件系统，伪文件系统，这个虚拟文件系统并不在磁盘上存储，而是在内存中动态生成的。

内容：内核中内核中运行进程和系统状态信息的快照

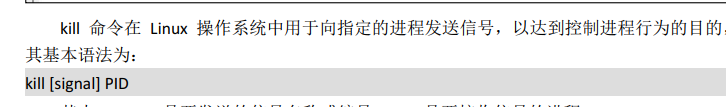
文件和目录：内核中进程和系统信息的数据结构在用户空间的映射

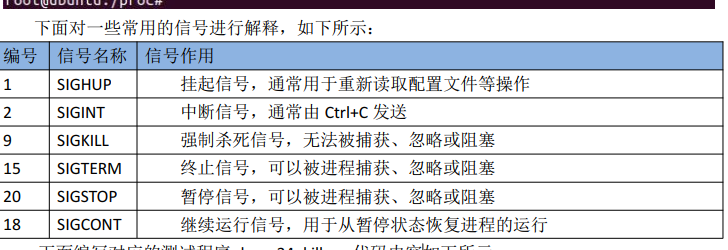




# 6.7 进程间的通信

(1 )kill 命令 用于向指定的进程发送信号， 以达到控制进程行为





**(2) siganl函数 主要用于捕捉和处理进程接收到的信号。 在任何需要处理信号的程序中， 都可以使用 signal 函数。**

