[linux进程通信---几个发送信号的函数(kill,raise,alarm,pause)](http://blog.csdn.net/zzyoucan/article/details/9235685)

2013-07-03 20:42 5937人阅读 [评论](http://blog.csdn.net/zzyoucan/article/details/9235685#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/zzyoucan/article/details/9235685#report)

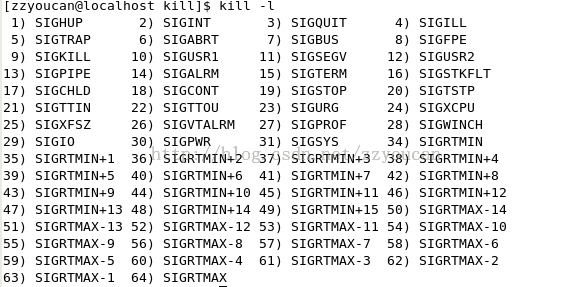
http://static.blog.csdn.net/images/category_icon.jpg 分类：

linux（22） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

信号：信号是unix中最古老的进程通信的一种方式，他是软件层次上对中断机制的模拟，是一种异步通信方式，信号可以实现用户空间进程和内核空间进程的交互，内核进程可以利用他通知用户空间进程发生了哪些系统事件，我们可以任何时候给进程发送信号而无需知道进程的状态，如果进程当前并未处于执行态，则信号则会由内核保存起来，如果进程是阻塞状态，那么信号传递会被延迟，直到阻塞结束时才会传递给进程。

看一下kill -l列出的各种命令：

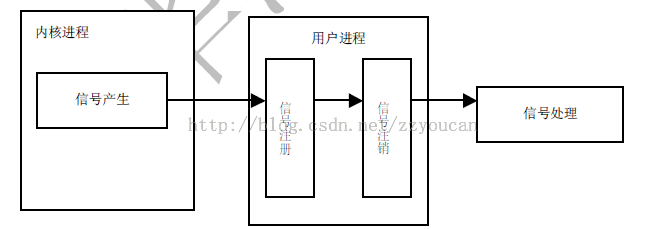


会发现一个规律，前32种信号会有各种不同的名称，后32种会以“SIGRTMIN”或者“SIGRTMAX”开头，前者是从unix继承下来的信号，称为不可靠信号(也称为非实时信号),后者为了解决“不可靠信号”的问题进行了更改和扩充的信号形成了可靠信号(也称为实时信号)

如果想要了解可靠与不可靠信号，需要了解信号的生命周期:

一个完整的信号周期可以分为三个重要阶段，三个重要阶段有四个重要事件刻画的：信号产生，信号在进程中注册，信号在进程中注销，执行信号处理函数

信号处理周期：



相邻的两个事件的时间间隔构成了生命周期的一个阶段，这里的信号处理有多种方式，一般由内核完成，也可以由用户进程完成

可靠信号与不可靠信号的区别：

不可靠信号如果发现信号已经在进程中注册，就会忽略该信号，因此若前一个信号还没有注销又产生了新的信号就是导致信号丢失

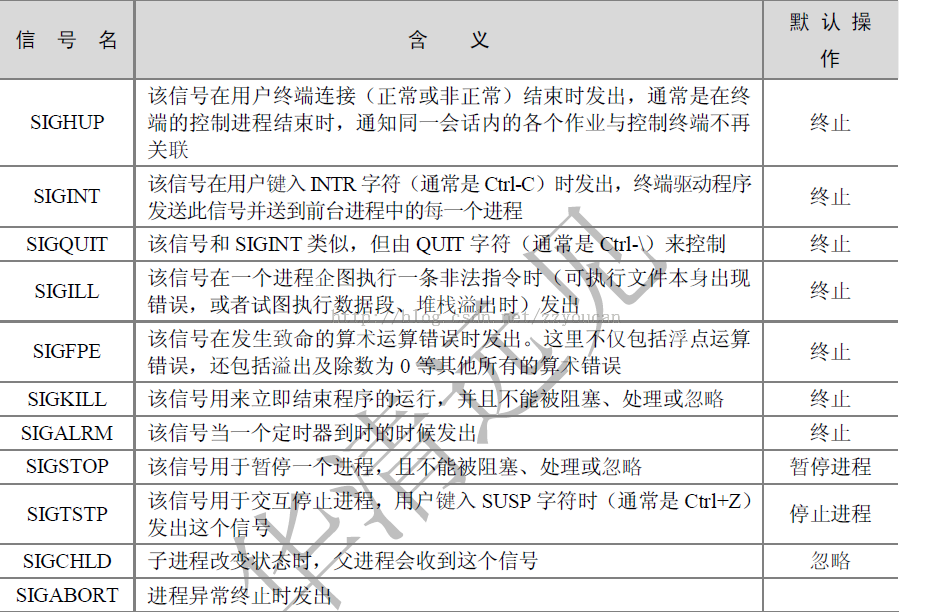
可靠信号发送给一个进程时，不管该信号是否已经在进程中注册，都会被再注册一次，因此信号不会丢失，所有可靠信号都支持排队，所有不可靠信号都不支持排队。

ps：这里信号的产生，注册，注销等是指信号的内部的实现机制，而不是调用信号的函数实现，所以信号注册与否，与本节后面讲到的发送信号函数(kill等)以及信号安装函数(signal()等)无关只与信号值有关

用户进程对信号的响应有三种方式：

1. 忽略信号，对信号不做任何处理，但是又两个信号是不能忽略的，即SIGKILL,SIGSTOP
2. 捕捉信号，定义信号处理函数，当信号发送时，执行相应的自定义处理函数
3. 执行缺省操作，linux对每种信号都规定了默认操作

linux中大多数信号是提供给内核的，下面列出了最为常见的信号的含义及其默认操作：



发送信号的函数主要有kill(),raise(),alarm(),pause()

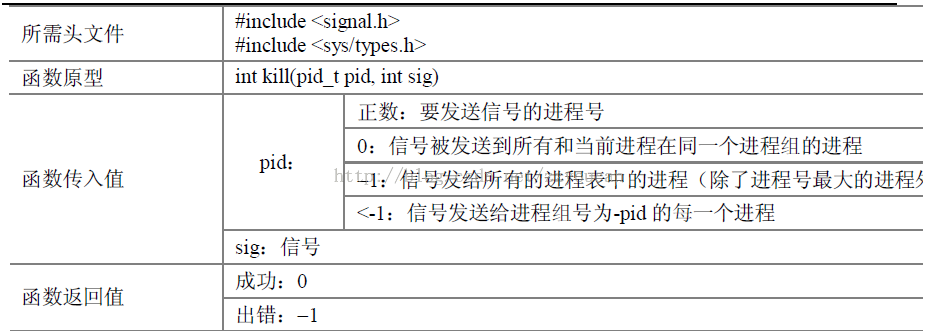
(1)kill()和raise()

kill()函数和熟知的kill系统命令一样，可以发送信号给信号和进程组（实际上kill系统命令只是kill函数的一个用户接口），需要注意的是他不仅可以终止进程(发送SIGKILL信号)，也可以向进程发送其他信号。

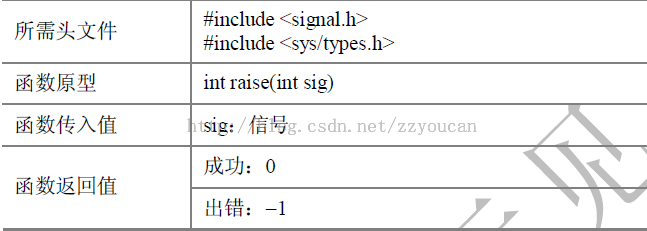
与kill函数不同的是raise()函数允许进程向自身发送信号。

(2)函数格式：

kill函数的语法格式：



raise()函数语法要点：



下面的例子使子进程不在父进程调用kill之前不退出，然后父进程调用kill使子进程退出：

**[cpp]** [view plain](http://blog.csdn.net/zzyoucan/article/details/9235685) [copy](http://blog.csdn.net/zzyoucan/article/details/9235685)

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <signal.h>
4. #include <sys/types.h>
5. #include <sys/wait.h>
6. **int** main()
7. {
8. pid\_t pid ;
9. **int** ret ;
10. **if** ((pid = fork()) < 0)
11. {
12. printf("fork error\n") ;
13. exit(1) ;
14. }
15. **else** **if** (pid == 0)
16. {
17. printf("child(pid:%d)id waiting for any signal\n", getpid()) ;
18. raise(SIGSTOP) ;//子进程暂停
19. exit(0) ;
20. }
21. **else**
22. {
23. //获取到pid子进程没有退出,指定WNOHANG不会阻塞，没有退出会返回0
24. **if** ((waitpid(pid, NULL, WNOHANG)) == 0)
25. {
26. **if** ((ret = kill(pid, SIGKILL)) == 0)//向子进程发出SIGKILL信号
27. {
28. printf("parent kill %d\n", pid) ;
29. }
30. }
31. waitpid(pid, NULL, 0) ;//等待子进程退出,是阻塞函数如同wait()
32. exit(0) ;
33. }
34. }

程序运行结果如下：

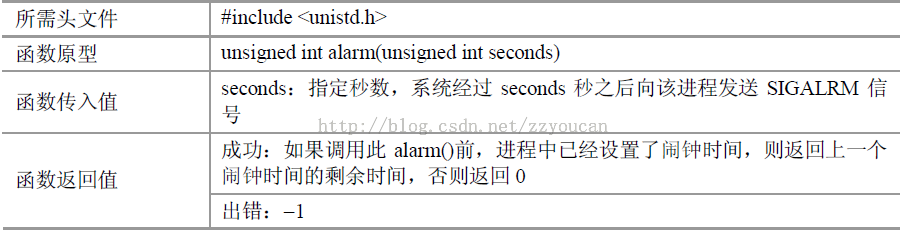
http://img.blog.csdn.net/20130703192003390?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvenp5b3VjYW4=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center看出父进程kill掉了子进程

(3)alarm()和pause()

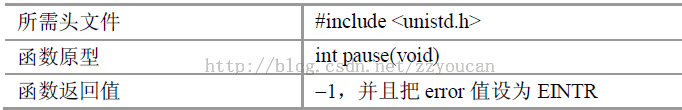
alarm()-----也称为闹钟函数，可以在进程中设置一个定时器，等到时间到达时，就会想进程发送SIGALARM信号，注意的是一个进程只能有一个闹钟时间，如果调用alarm()之前已经设置了闹钟时间，那么任何以前的闹钟时间都会被新值所代替

pause()----此函数用于将进程挂起直到捕捉到信号为止，这个函数很常用，通常用于判断信号是否已到

alarm()函数语法：



pause()函数语法如下：



下面的一个实例实现了sleep()函数的功能

**[cpp]** [view plain](http://blog.csdn.net/zzyoucan/article/details/9235685) [copy](http://blog.csdn.net/zzyoucan/article/details/9235685)

1. #include <unistd.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. **int** main()
5. {
6. **int** ret = alarm(5) ;//设置一个定时器
7. pause() ;//捕捉定时器信号
8. printf("I have been waken up\n", ret) ;
9. }
10. ~

执行程序会在5秒之后出现：

http://img.blog.csdn.net/20130703204151421?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvenp5b3VjYW4=/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

这个函数中的printf是不会执行的，因为定时器发送的SIGARAM的默认处理是终止程序，所以程序打印之前程序已经结束了，与sleep不同的是sleep是不会退出的。