第三十四章

1. 进程组，会话，作业控制。

进程组和会话是为了支持shell作业控制而定义的抽象概念。

1. 概述：

概述了进程组，会话，控制终端，前台进程等等概念。具体可以看书本。

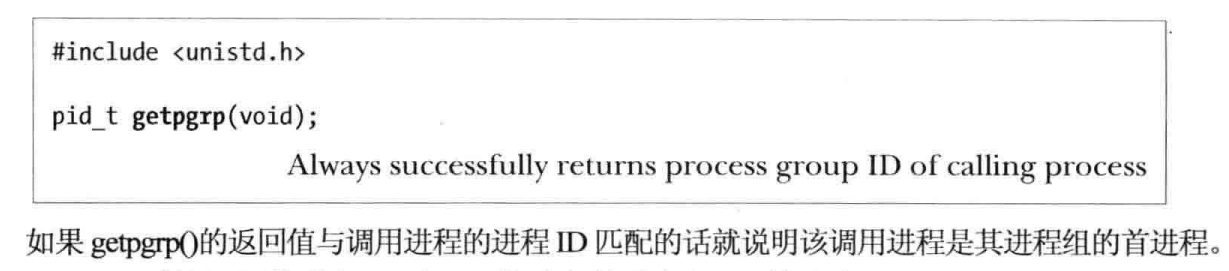
通过一张图可以看出它们的关系：

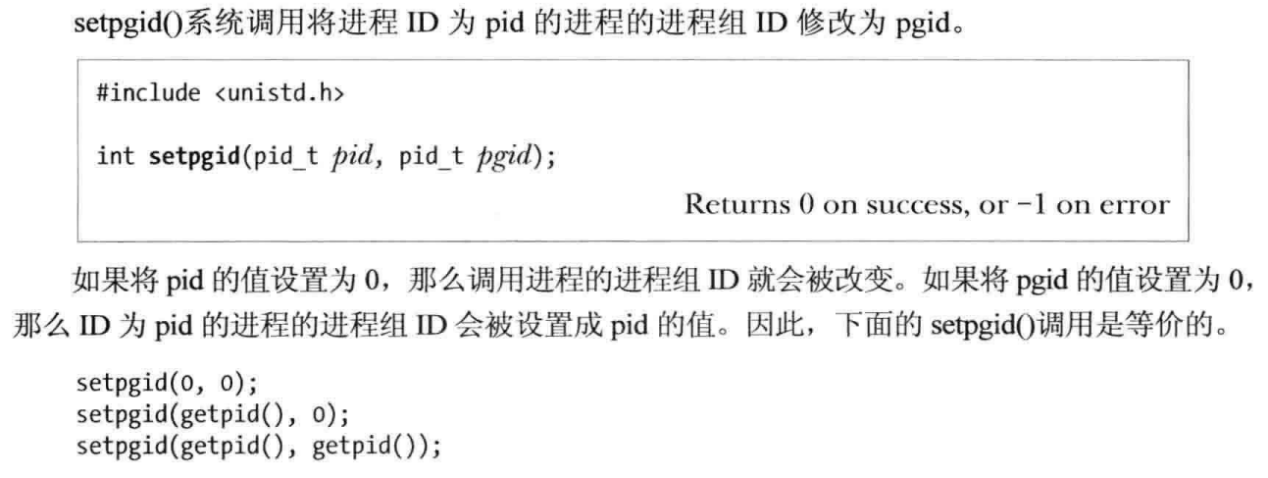


1. 进程组：

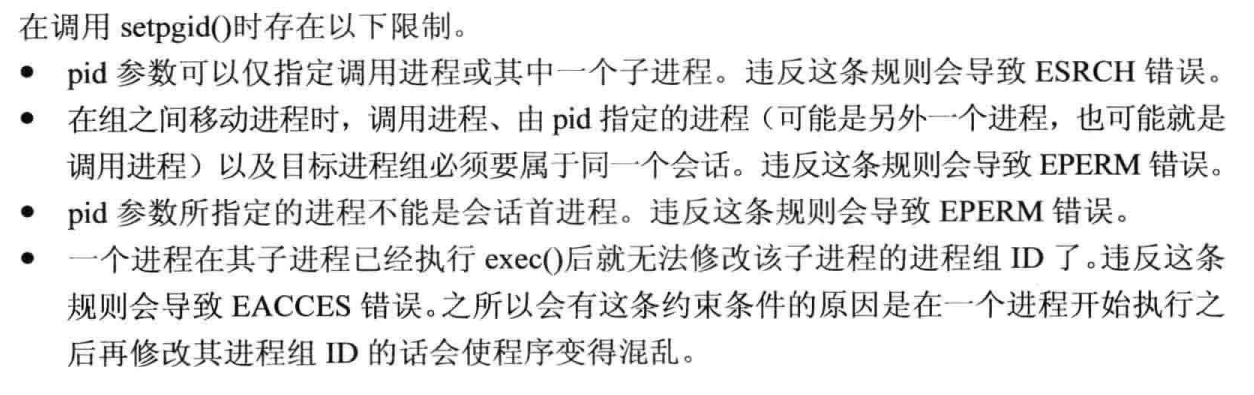
每个进程都拥有一个以数字表示的进程组ID，表示该进程所属的进程组。新进程会继承父进程的进程组ID。

获取进程的进程组ID：



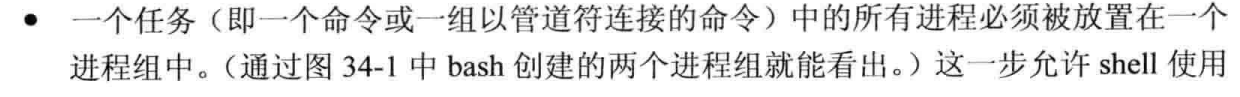


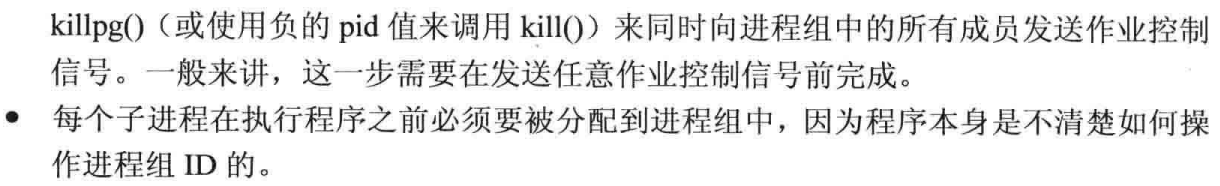
注意，调用setpgid的时候有限制：



我试了一下，进程组首进程可以改变自身进程组。

由于上述最后一个限制，shell在实现的时候需要注意许多事情。

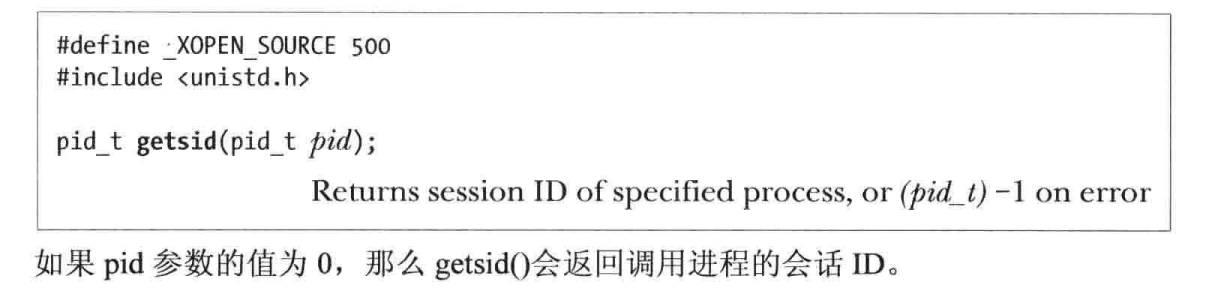


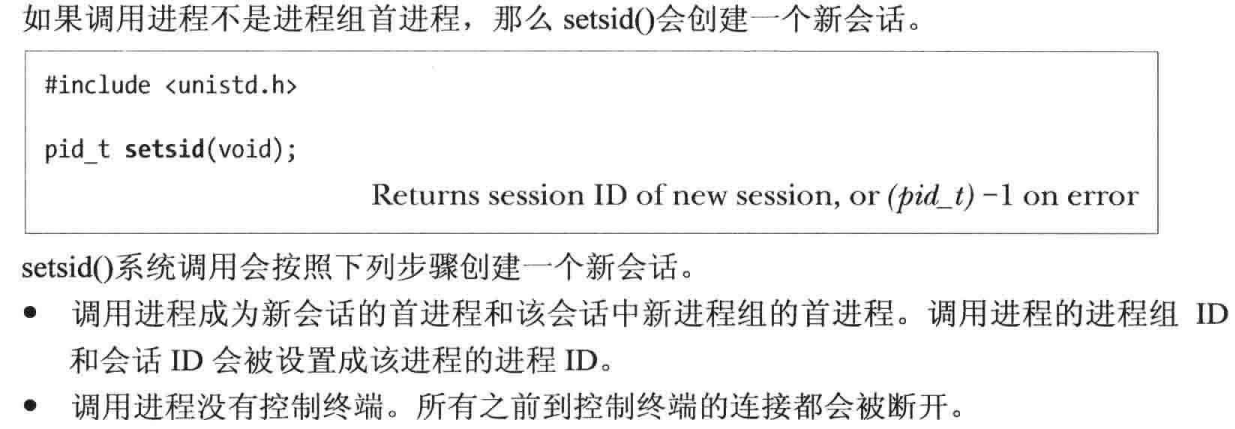


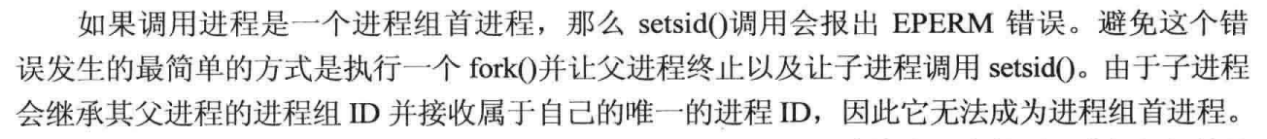
对于shell的实现来说，因为fork之后父子进程的调度顺序不确定，所以需要在父子进程调用setpgid同时改变子进程组ID。

1. 会话：

会话是一组进程组的集合。







约束进程组首进程对setsid的调用是很有必要的。因为这回破坏进程-进程组-会话的两级关系，因此，进程组的所有成员必须属于同一个会话。

1. 控制终端和控制进程：

什么是终端？我的理解是，现在的终端已经不是以前的终端，以前的终端是指多个设备连接到电脑共用电脑，这些设备就是终端。现在计算机已经普及了，终端，一般指你的键盘和字符界面，或者图像界面这种伪终端，而Linux里一切皆文件，终端在Linux的表示是/dev/tty，具体字符终端有/dev/tty1-6，而伪终端有无数个，/dev/pts/n。具体可以看

<http://zyan.cc/book/linux_c/html/ch34s01.html>

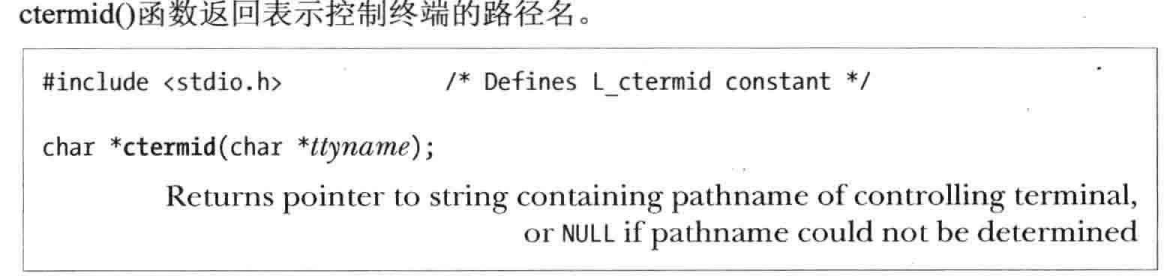
了解一下。而会话首进程打开一个没有成为某个会话的控制终端的终端时，会建立控制终端。一个终端只能成为一个会话的控制终端。因此，跟会话建立连接的时候才是控制终端。而书本文字也常常使用/dev/tty这些文件代指终端，你也可以认为/dev/tty也是终端，毕竟，它是终端在Linux里的表示。而会话首进程打开控制终端时，这个会话首进程也成为控制进程。





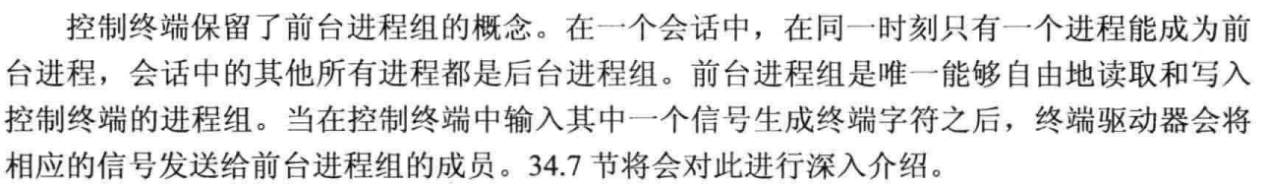
建立控制终端：





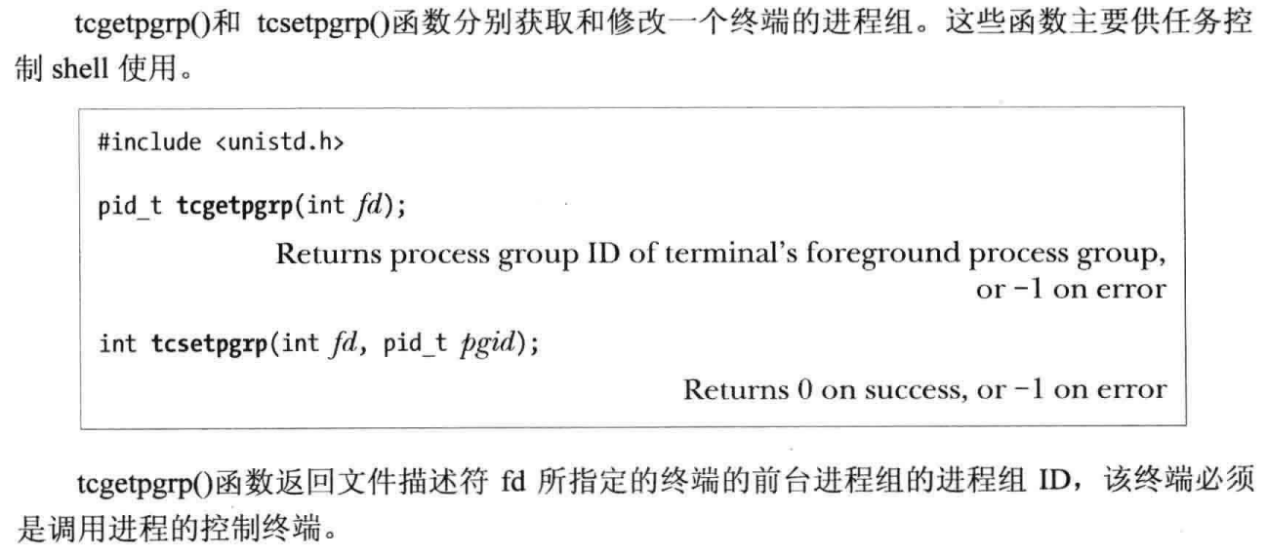
控制终端是针对会话而言的。

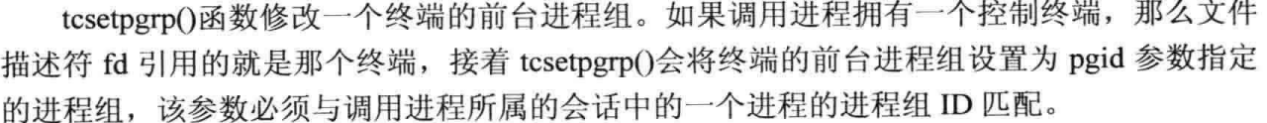
1. 前台和后台进程组：



注意，前后台进程组也是针对会话和控制终端而言的。

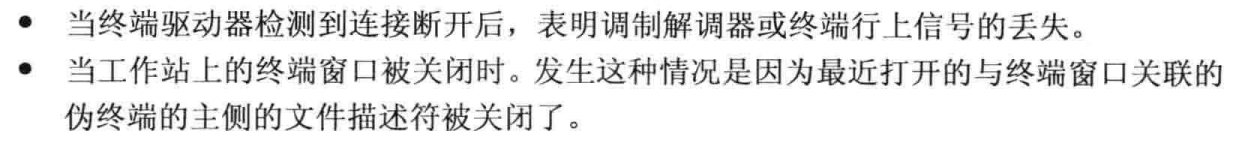
有可能出现一个会话没有前台进程组的情况。



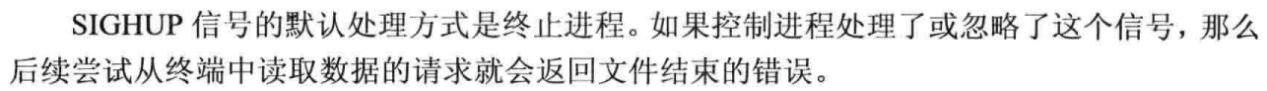


1. SIGHUP信号：

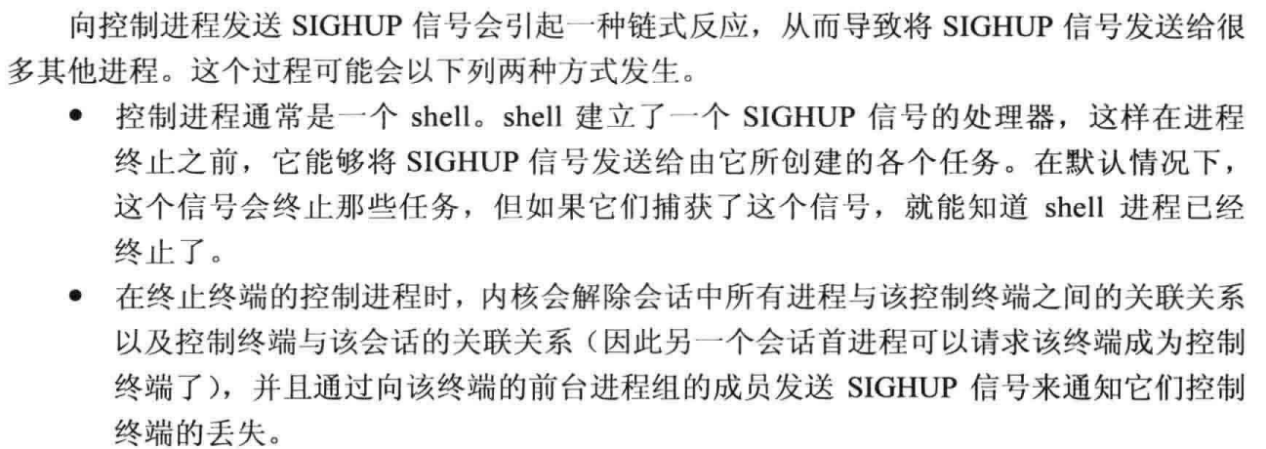
当一个控制进程失去其终端连接之后，内核会向它发送SIGHUP信号来通知它这一事实，还有可能发送SIGCONT信号。这一情况可能发生在下面两个情况：



第二个和伪终端有关。



比如一般的shell就针对SIGHUP信号做了某些处理。



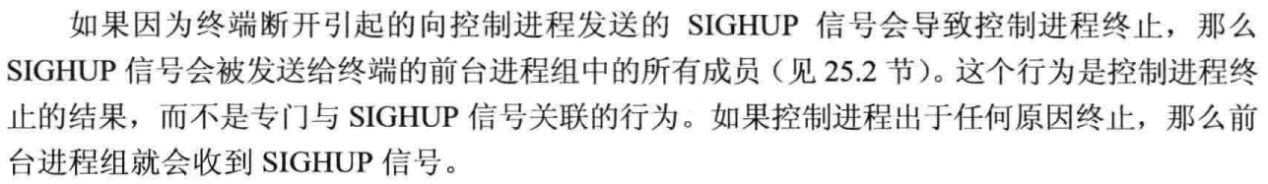
上述情况的前提是关闭终端，内核发送SIGHUP信号给控制进程的两种情况。

针对第一点，shell会发送SIGHUP信号给它创建的进程组的进程，注意，是它创建的进程组进程，然后再终止自己。

第二点，如果控制进程不是shell（比如下面例子使用exec执行别的程序，那么，仅仅发送SIGHUP信号给前台进程），此时不会出现第一点的情况，即不会再执行shell里面关于SIGHUP信号处理函数。

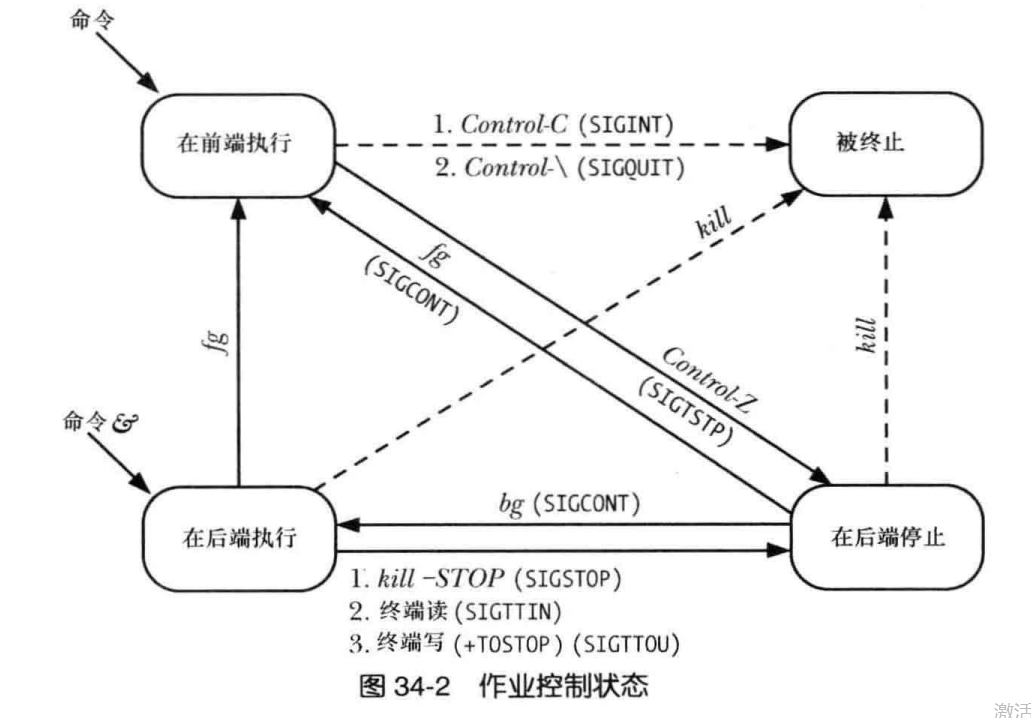
书本下面分别实现了两个例子，分别验证了上述文字。

注意下图这段话：发送给前台进程组所有成员是控制进程终止的结果，不是shell进程的SIGHUP信号处理函数的工作。

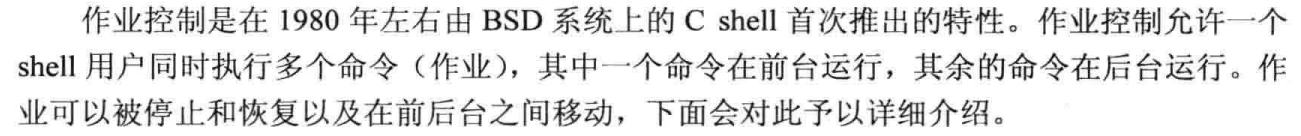


1. 在shell中使用作业控制：

当前作业是在是前台进程最新被挂起的作业。如果没有，则是后台最新作业。Jobs命令有+号的是当前作业，-号是上一个作业。



什么是作业控制：



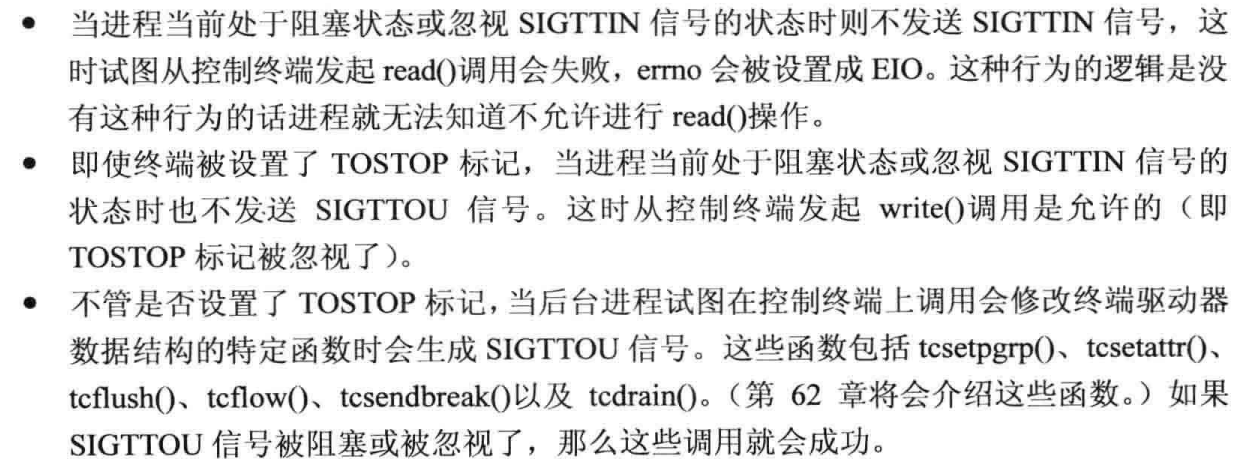
1. 实现作业控制：

实现作业控制需要的支持：

1. 提供特定的作业控制信号。
2. 终端驱动器必须记录控制进程和前台进程组ID。
3. Shell要支持作业控制。

注意：内核允许一个进程向同一会话中的任意进程发送SIGCONT信号。

对后台进程的SIGTTIN和SIGTTOU信号的特殊规定：

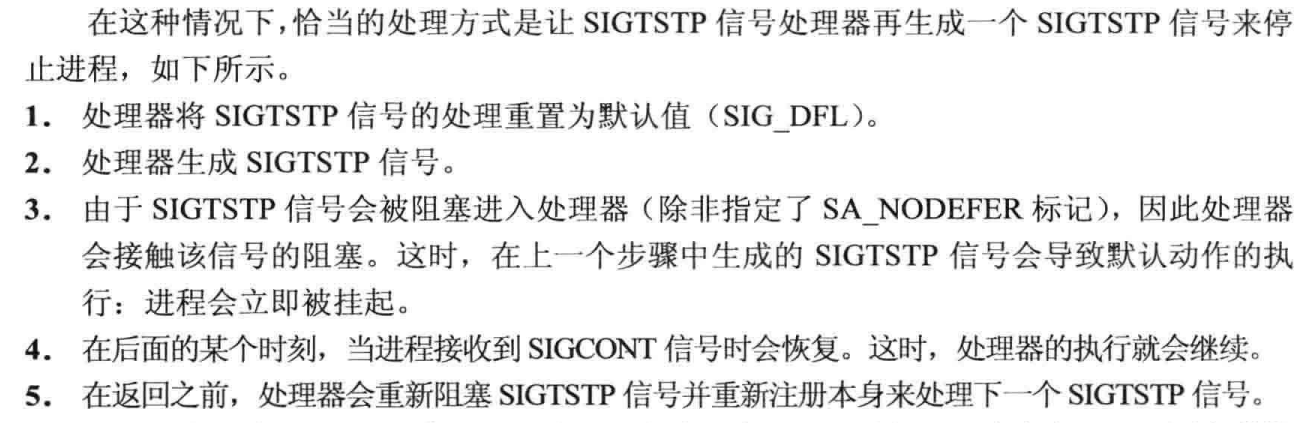


问题：处于阻塞状态如何发起read调用？

1. 处理作业控制信号：

屏幕处理程序需要对作业控制信号采取特殊动作，比如：vi。

屏幕处理程序需要处理终端停止信号：SIGTSTP，例如恢复屏幕设置。但是，如果捕获了这个程序，就无法停止，应该如何解决？



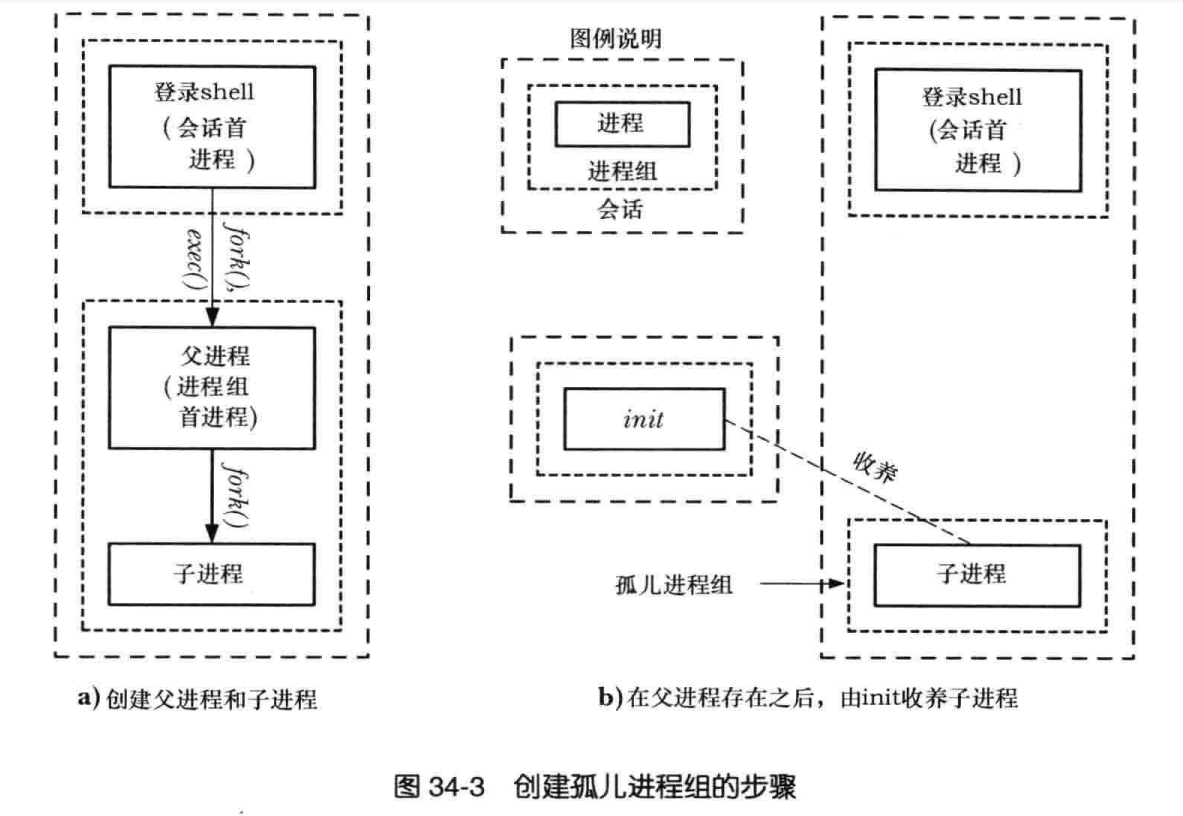
处理被忽略的任务控制和终端生成的信号：

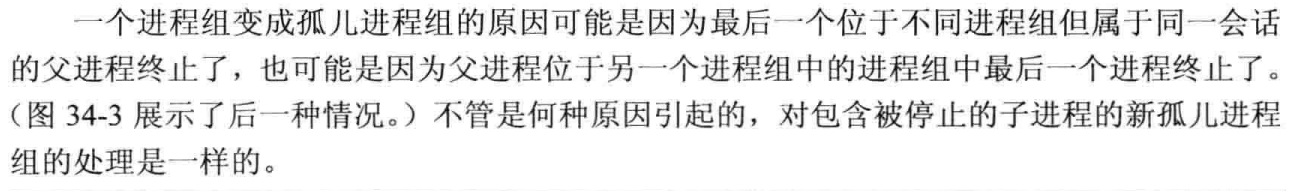
一个常规规则是：应用程序应该在终端和作业控制生成信号不被忽略的情况下才处理这些信号。

1. 孤儿进程组：

定义：当一个进程组满足每个成员的父进程本身是组的一个成员或不是组会话的一个成员的时候，就是孤儿进程组。

由于shell会从作业控制列表删除孤儿进程组，从而导致被停止的子进程永远的残留在系统中，所以，如果一个进程组变成了孤儿进程组，并且拥有了停止执行的成员，那么，系统会向进程组所有成员发送SIGHUP信号和SIGCONT信号。如果孤儿进程组不包含任何停止的进程，则不会发送，如果进程组变成孤儿进程组，里面没有停止进程，之后才发送信号使其停止，则不会产生SIGHUP信号和SIGCONT信号。





对于孤儿进程组，如果SIGTSTP,，SIGTTIN，SIGTTOU发送到孤儿进程组里的成员会被丢弃，如果建立了信号处理函数，则会被处理。而孤儿进程组里的发生read的时候会发生错误。