Linux - 请允许我静静地后台运行

前言

常在 linux 下玩耍的开发者肯定会经常遇到需要对进程调度的情况，在 windows 中点击 `最小化` 去干别的就 OK 了，那么在 linux 下怎么办呢。

可能有的小伙伴会说，再开一个终端窗口不就好了么。可是开很多窗口管理会很不方便，还有万一手贱点了x，或者长时间不操作，远程终端断开了连接，进程停止了，再次打开，又是一番折腾。

今天来介绍几个命令，帮大家系统地梳理一下 linux 的进程调度，并附上一些自己的使用心得和踩过的坑。

名词

在此之前，我们必须（当然也不是必须，但了解原理有利于理解和解决错误）先弄懂几个名词。

进程组

进程组是一个或多个进程的集合，进程组方便了对多个进程的控制，在进程数较多的情况下，向进程组发送信号就行了。

它的 ID 由它的组长进程的进程 ID 决定。组长进程创建了进程组，但它并不能决定进程组的存活时间，只要进程组内还有一个进程存在，进程就存在，与组长进程是否已终止无关。

会话

会话(session)是一个或多个进程组的集合，它开始于用户登陆终端，结束于用户退出登陆。其义如其名，就是指用户与系统的一次对话的全程。

会话包括控制进程（与终端建立连接的领头进程），一个前台进程组和任意后台进程组。一个会话只能有一个控制终端，通常是登录到其上的终端设备或伪终端设备，产生在控制终端上的输入和信号将发送给会话的前台进程组中的所有进程。

控制终端

每当我们使用终端工具打开一个本地或远程 shell，我们便打开了一个控制终端，通过 `ps` 命令可以查看到 command 为 `ttyn` 的就是它对应的进程了，同时它对应 linux `/dev/` 目录下的一个文件。

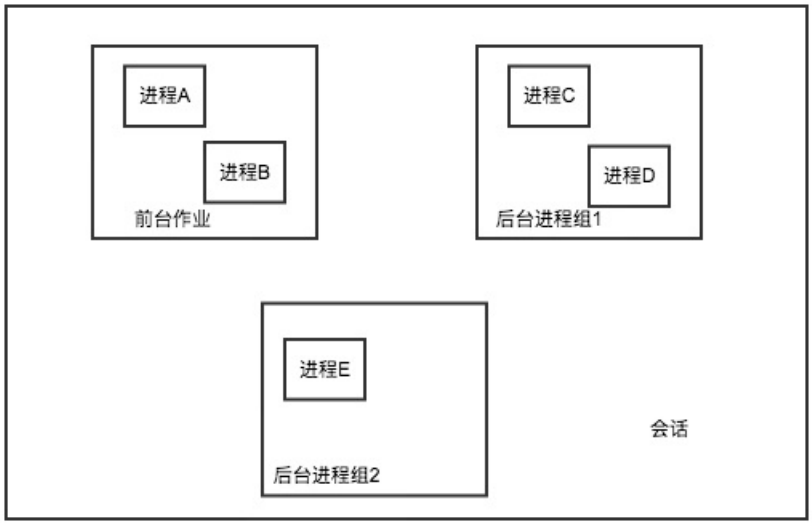
作业

作业的概念与进程组类似，同样由一个或多个进程组成，它分为前台作业和后台作业，一个会话会有一个前台作业和多个后台作业，与进程组不同的是，作业内的某个进程产生的子进程并不属于这个作业。

类比

以上几个概念可以类比为我们一次通过 QQ 聊天的全程，控制终端就是 QQ软件，关闭了此软件代表着聊天结束。聊天时发送的每一条信息都是一个进程，作业或进程组就是我们在聊的某一件事，它由很多条相互的信息构成。而会话则是我们指我们从开始聊天到结束聊天的全过程，可能会聊很多个事。

它们之间的相关图如下所示：



后台执行

我们每次在终端窗口执行命令的时候，进程总会一直占用着终端，走到进程结束，这段时间内，我们在终端的输入是没有用的。而且，当终端窗口关闭或网络连接失败后，再次打开终端，会发现进程已经中断了。这是因为用户注销或者网络断开时，SIGHUP信号会被发送到会话所属的子进程，而此 SIGHUP 的默认处理方式是终止收到该信号的进程。所以若程序中没有捕捉该信号，当终端关闭后，会话所属进程就会退出。

我们要实现后台执行的目的，实际上是要完成如下两个目标：

- 使进程让出前台终端，让我们可以继续通过终端与系统进行交互。
- 使进程不再受终端关闭的影响，即系统在终端关闭后不再向进程发送 SIGHUP 信号或即使发送了信号程序也不会退出。

以下的命令就围绕着这两个目标来实现。

&

首先是我们最经常遇到的符号 &，将它附在命令后面可以使进程在后台执行，不会占用前台界面。它实际上是在会话中开启了一个后台作业，对作业的操作我们后面再说。

但我们会发现，如果此时终端被关闭后，进程还是会退出。这是因为，& 符号只有让进程让出前台终端的功能，无法让进程不受 SIGHUP 信号的影响。

nohup

nohup 应该是另外一个我们常用的命令了，它的作用如其字面意思，使进程不受 SIGHUP 信号的影响。但我们在使用 nohup php test.php 后会发现，进程还会一直占用前台终端，但即使终端被关闭或连接断开了，程序还是会执行，另外我们会发现在当前文件夹下多了个名为 nohup.out 的文件。

这是因为 nohup 的功能仅仅是让进程不受 SIGHUP 信号的影响，并不会让出前台终端，而且它还会在命令执行目录下建立 nohup.out 用以存储进程的输出。如果进程不需要输出，且不想让 nohup 创建文件，可以将标准输出和标准错误输出重定向。

我们常将 nohup 和 & 搭配到一块使用，执行命令如下 nohup command >/dev/null 2>&1 & 这样，就可以放心的等待进程运行结果了。

setsid

setsid 是另一个让进程在后台执行的命令，它的作用是让进程打开一个新的会话并运行进程，使用方式为 setsid command。

根据上面的概念我们得知终端关闭后进程退出是因为会话首进程向进程发送了 SIGHUP 信号，setsid 就厉害了，它直接打开一个新的会话来执行命令，那么原会话的终端的状态就再也不会影响到此进程了。

我们使用 pstree 来查看使用 setsid 和 nohup ... & 两种命令来运行进程时的进程树状态。

- nohup php test.php &

```
pstree -a |grep -C 6 test
  |-sshd
  |   `-sshd
  |       `-sshd
  |           `-bash
  |               `-sudo -s
  |                   `-bash
  |                       |-grep -C 6 test
  |                       |-php test.php
  |                       `-pstree -a
```

我是用 ssh 远程登陆的机器，所以 test.php 进程是挂在 sshd 进程下的。正常情况下，一旦 sshd 进程结束，则 test.php也无法幸免。

- setsid php test.php

```
pstree -a |grep -C 6 test
  |-{nscd}
  |-php test.php
  |-php-fpm
 --
  |-sshd
  |   `-sshd
```

使用了 setsid 后，test.php 进程已经与 sshd 进程同级，属于 init 进程的子进程了。

但是 setsid 并没有为进程分配一个输出终端，所以进程还是会输出到当前终端上。

setsid的坑

另外，setsid 有个略坑的地方： 在终端中直接使用 setsid command 运行进程时，终端前台并不会被影响，command 会在后台默默运行。而在 shell 脚本中，我们会发现运行 setsid 的进程会一直阻塞住，直到 command 进程执行结束。

这是因为，setsid 在其是进程组长时会 fork() 一个进程，但它不会 wait() 它的子进程，而是立刻退出，所以在终端内直接使用 setsid 时，setsid 作为进程组长不会占用终端界面。

而在 shell 脚本内，setsid 不是进程组长，它不会 fork() 子进程，而是由 bash 来fork() 一个子进程，而 bash 会 wait() 子进程，所以表现得像 setsid 在 wait() 子进程一样。

要解决这个问题，有两个办法：

- 使用上面介绍的 &符号，使 setsid 强行到后台执行。
- 使用 . 或 source 命令由终端执行 setsid；

其他

除了上面介绍的命令，还有 screen 和 tmux 等会话工具，他们都有自己的一套规范，也比较复杂，掌握本文的命令已经足够你驰骋 linux 进程控制了。当然有想了解新知识的可以查询学习一下，应该

会比基础命令好用。

作业命令

使用上面的后台执行命令时可能还会遇到一些小状况：

- 被我们放在后台的进程执行时间过长，而我们又忘记使用 nohup 命令，那么终端一旦断开，进程又需要被重新执行。
- 我们直接开启了某个进程，又想在不中断进程的情况下让它让出前台终端；

这些都要牵涉到今天的第二个模块--作业；

我们在终端里运行的命令都可以理解为一个作业，有的占用前台终端，有的在后台默默执行，下面的命令就是为了调度这些作业。

jobs

jobs 是作业的基础命令，用它可以查看正在运行的作业的信息，其输出如下：

```
jobs
[1]-  Running                 php test.php &
[2]+  Stopped                 php test.php
```

前面[ ]内的数字是作业 ID，也是后面我们要操作作业的标识，然后是作业状态和命令。

ctrl+z

ctrl+z 严格来说并是作业命令，它只是向当前进程发送一个 SIGSTOP 信号，促使进程进入暂停(stopped)状态，此状态下，进程状态会被系统保存，此进程会被放置到作业队列中去，而让出进程终端。

使用它，我们可以暂停正在占用终端的进程而不停止它，从而让我们使用终端命令来操作此进程。

bg

bg是 backgroud 的缩写，顾名思义，bg %id 把作业放到后台进程中执行。

结合 ctrl+z 和 bg 命令，我们可以解决上面提出的第一个问题，不停止地将正在占用终端的进程放到后台执行。

fg

fg 与 bg 相对，使用它可以把作业放到前台来执行。

disown

disown 用来将作业从作业列表中移除，即使它 不属于 会话，这样终端关闭后不再向此作业发送 SIGHUP 信号，以阻止终端对进程的影响。

使用 disown 我们可以解决上面提出的第二个问题，不重新执行将一个没使用 nohup 命令的进程不受终端关闭影响。

守护进程

以上介绍的都是一些临时进程的处理，后台运行的进程的最终方法是将进程变成守护进程。

守护进程

“

守护进程(daemon)是生存期较长的一种进程，一般在系统启动时启动，系统关闭时停止，没有控制终端，也不会输出。如我们的服务器、fpm 等进程就是以守护进程的形式存在的。

创建过程

要创建一个守护进程，步骤为：

必选项

1. fork 子进程，退出父进程，子进程作为孤儿进程被 init 进程收养；
2. 使用 setsid, 打开新会话，进程成为会话组长，正式脱离终端控制；
3. 设置信号处理（特别是子进程退出处理）；

可选项：

4. 使用 chdir 改变进程工作目录，一般到根目录下，防止占用可卸载文件系统；
5. 用 umask 重设文件权限掩码，不再继承父进程的文件权限设置；
6. 关闭父进程打开的文件描述符；

代码

以下是 php 创建守护进程的伪代码，另外我的另一篇博客 初探PHP多进程 也稍微介绍了一些相关内容：

```
$pid = pcntl_fork();
if ($pid > 0) {
    exit; // 父进程直接退出
} elseif ($pid < 0) {
    throw_error(); // 进程创建失败
}

posix_setsid(); // setsid成为会话领导进程
chdir($dir); // 切换目录
umask(0); // 重置文件权限mask
close_fd(); // 关闭父进程的文件描述符
pcntl_signal($signal, $func); // 注册信号处理函数

while (true) {
    do_job(); // 处理进程任务
    pcntl_signal_dispatch(); // 分发信号处理
}
```

总结

linux 是开发者的基础技能，而进程的调度更是我们常用的功能，希望读完本文的同学们能有所收获。

又有大半个月没发博客了，最近鼓捣着重构代码，经常会在一个点上纠结半天，不知不觉就加了个班。而且这个是个没法精确度量工作量和目标的活儿，优化没有尽头嘛。不过由于要更多地考虑一下代码的抽象、效率和扩展，对自己也是个挑战，算是乐在其中吧~

最近可能会考虑写一个守护进程和 cron 进程调度器，嗯，希望给我算到工作量里，哈哈~想写的太多了，只怨自己还不够强大。。。

如果您觉得本文对您有帮助，可以点击下面的 **推荐** 支持一下我。博客一直在更新，欢迎 **关注** 。

参考：

setsid为什么会在脚本中阻塞-StackoOerflow

Linux 进程、进程组、会话周期、控制终端

分类: LINUX

标签: linux, 进程, 作业, 后台运行, 守护进程, Daemon