CentOS 丛书目录 — 系统管理 — 网络服务 — 应用部署

进程管理和作业控制

内容提要

1. 掌握进程的相关概念
2. 熟悉Linux中进程的类型和启动方式
3. 学会查看和杀死进程
4. 理解何谓作业控制
5. 掌握实施作业控制的常用命令

进程概述

进程的概念

进程（Process）是一个程序在其自身的虚拟地址空间中的一次执行活动。之所以要创建进程，就是为了使多个程序可以并发的执行，从而提高系统的资源利用率和吞吐量。

进程和程序的概念不同，下面是对这两个概念的比较

- 程序只是一个静态的指令集合；而进程是一个程序的动态执行过程，它具有生命期，是动态的产生和消亡的。
- 进程是资源申请、调度和独立运行的单位，因此，它使用系统中的运行资源；而程序不能申请系统资源、不能被系统调度、也不能作为独立运行的单位，因此，它不占用系统的运行资源。
- 程序和进程无一一对应的关系。一方面一个程序可以由多个进程所共用，即一个程序在运行过程中可以产生多个进程；另一方面，一个进程在生命期内可以顺序的执行若干个程序。

Linux操作系统是多任务的，如果一个应用程序需要几个进程并发地协调运行来完成相关工作，系统会安排这些进程并发运行，同时完成对这些进程的调度和管理任务，包括CPU、内存、存储器等系统资源的分配。

Linux中的进程

在Linux系统中总是有很多进程同时在运行，每一个进程都有一个识别号，叫做PID（Process ID），用以区分不同的进程。系统启动后的第一个进程是init，它的PID是1。init是唯一一个由系统内核直接运行的进程。新的进程可以用系统调用fork来产生，就是从一个已经存在的旧进程中分出一个新进程来，旧的进程是新产生的进程的父进程，新进程是产生它的进程的子进程，除了init之外，每一个进程都有父进程。当系统启动以后，init进程会创建login进程等待用户登录系统，login进程是init进程的子进程。当用户登录系统后，login进程就会为用户启动shell进程，shell进程就是login进程的子进程，而此后用户运行的进程都是由shell衍生出来的。

除了进程识别号外，每个进程还有另外四个识别号。它们是实际用户识别号（real user ID）、实际组识别号以及有效用户识别号（effect user ID），和有效组识别号（effect group ID）。实际用户识别号和实际组识别号的作用是识别正在运行此进程的用户和组。一个进程的实际用户识别号和实际组识别号就是运行此进程的用户的识别号（UID）和组的识别号（GID）。有效用户识别号和有效组识别号的作用是确定一个进程对其访问的文件的权限和优先权。除了产生进程的程序被设置UID位和GID位之外，一般有效用户识别号和有效组识别号和实际用户识别号及实际组识别号相同。如果程序被设置了UID位或GID位，则此进程相应的有效用户识别号和有效组识别号，将和运行此进程的文件的所属用户的UID或所属组的GID相同。

例如，一个可执行文件/usr/bin/passwd，其所属用户是root（UID为0），此文件被设置了UID位。则当一个UID为500、GID为501的用户执行此命令时，产生的进程的实际用户识别号和实际组识别号分别是500和501，而其有效

用户识别号是0，有效组识别号是501。

所有这些设计都是为了在一个多用户、多任务的操作系统中，所有用户的工作都能够安全可靠地进行，这也是Linux操作系统的优秀性所在。

进程的类型

可以将运行在Linux系统中的进程分为三种不同的类型：

- 交互进程：由一个Shell启动的进程。交互进程既可以在前台运行，也可以在后台运行。
- 批处理进程：不与特定的终端相关联，提交到等待队列种顺序执行的进程。
- 守护进程：在Linux在启动时初始化，需要时运行于后台的进程。

以上三种进程各有各的特点、作用和不同的使用场合。

进程的启动方式

启动一个进程有两个主要途径：手工启动和调度启动。

1. 手工启动：由用户输入命令，直接启动一个进程便是手工启动进程。手工启动进程又可以分为前台启动和后台启动。
   I. 前台启动：是手工启动一个进程的最常用的方式。一般地，用户键入一个命令"ls -l"，这就已经启动了一个进程，而且是一个前台的进程。
   II. 后台启动：直接从后台手工启动一个进程用得比较少一些，除非是该进程甚为耗时，且用户也不急着需要结果的时候。假设用户要启动一个需要长时间运行的格式化文本文件的进程。为了不使整个shell在耗时进程的运行过程中都处于"瘫痪"状态，从后台启动这个进程是明智的选择。在后台启动一个进程，可以在命令行后使用&命令，例如：

```
# ls -R / >list &
```

调度启动方式是事先进行设置，根据用户要求让系统自行启动。见安排周期性任务

进程管理

查看系统中的进程

Linux是一个多用户多任务的操作系统，对多用户的状态查看可以使用who命令和w命令来进行。要对系统中的进程进行监测和控制，首先就要了解进程的当前运行情况，即查看进程。普通用户和管理员都可以查看系统中正在运行的进程，和这些进程的相关信息。在Linux中，使用ps命令对进程进行查看。ps是一个功能非常强大的进程查看命令。使用该命令使用户可以确定有哪些进程正在执行和执行的状态、进程是否结束、进程有没有僵死、哪些进程占用了过多的系统资源等等。总之，大部分信息都可以通过运行ps命令来获得。下面介绍ps命令的格式和常用选项。ps命令的格式如下：

```
ps [选项]
```

由于ps命令的功能相当强大，所以该命令有大量的选项参数，这里只介绍几个最常用的选项，如下表。

选项	说明
a	显示所有进程
e	在命令后显示环境变量
u	显示用户名和启动时间等信息

x	显示没有控制终端的进程
f	显示进程树
w	宽行输出
-e	显示所有进程
-f	显示全部

下表列出了ps命令输出的重要信息的含义。

输出项	说明
PID	进程号
PPID	父进程的进程号
TTY	进程从那个终端启动
STAT	进程当前状态
START	进程开始执行的时间
VSZ	进程所占用的虚拟内存的空间，以kb为单位
RSS	进程所占用的内存的空间，以kb为单位
TIME	进程自从启动以来占用CPU的总时间
COMMAND/CMD	进程的命令名
USER	用户名
%CPU	占用CPU时间与总时间的百分比
%MEM	占用内存与系统内存总量的百分比
SIZE	进程代码大小+数据大小+栈空间大小（单位：KB）

其中，在进程状态（STAT）一栏里，表示状态的字符的意义见下表。

字符	说明
R	进程正在执行中（进程排在执行队列里，随时都会被执行）
S	进程处于睡眠状态（sleeping）
T	追踪或停止
Z	僵尸进程（zombie），进程已经被终止，但其父进程并不知道，没有妥善处理，导致其处于僵尸状态
W	进程没有固定的pages
<	高优先级的进程
N	低优先级的进程

系统管理员可以从上面的输出获得进程的运行信息。例如：当某个进程占用了过多的CPU和MEM资源，系统管理员就应该检查该进程是否为一个合法进程。例如：

```
#  ps                          # 显示出当前用户在shell下所运行的进程
#  ps -u osmond              # 只查看用户osmond的进程
#  ps -aux                     # 列出系统中正在运行的所有进程的详细信息
#  ps  - auxf                  # 显示系统进程树
```

1．如果想看清所运行的进程的完整命令行，可以使用w参数

2．ps命令经常同管道命令连用，如下面的形式：

```
# ps  - aux|more
# ps  - aux|grep httpd
```

杀死系统中的进程

kill 命令

通常情况下，可以通过停止一个程序运行的方法来结束程序产生的进程。但有时由于某些原因，程序停止响应，无法正常终止，这就需要用kill命令来杀死程序产生的进程，从而结束程序的运行。kill命令不但能杀死进程，同时也会杀死该进程的所有子进程。kill命令的格式是：

```
kill [-signal] PID
```

其中：PID是进程的识别号；signal是向进程发出的进程信号，下表列出了一些常用信号的说明。

信号	数值	用途
SIGHUP	1	从终端上发出的结束信号
SIGINT	2	从键盘上发出的中断信号（ctrl+c）
SIGQUT	3	从键盘上发出的退出信号（ctrl+\）
SIGFPE	8	浮点异常（如：被0除）
SIGKILL	9	结束接受信号的进程（强行杀死进程）
SIGTERM	15	kill命令默认的终止信号
SIGCHLD	17	子进程中止或结束的信号
SIGSTOP	19	从键盘来执行的信号（ctrl+d）

要终止一个进程首先要知道它的PID，这就需要用到上面介绍过的ps命令。例如，用户的xterm突然停止响应了，无法接受用户的输入，也无法关闭，可以进行如下操作。

```
#  （1）找到xterm对应的进程的PID
$ ps aux | grep xterm
osmond   1621  0.0  1.3  6980 1704 tty1   S    Aug01   0:01 [xterm]
osmond   1920  0.0  1.9  6772 2544 tty1   S    00:41   0:00 [xterm]
osmond   1921  0.0  0.5  3528  664 pts/1  R    00:41   0:00 grep xterm
#  （2）杀死进程
$ kill 1621
```

可以看到用户共启动了两个xterm，可以通过两个xterm启动的先后顺序来判断哪个进程对应的是要杀死的xterm，因为先启动的进程的PID总是要小于后启动的进程的PID。默认情况下，kill命令发送给进程的终止信号是15，有些进程会不理会这个信号，这时可以用信号9来强制杀死进程，信号9是不会被忽略的强制执行信号。例如，如果上面的命令没有能够杀死xterm，可以用信号9来结束它。

```
$ kill -9 1621
```

killall 命令

用户也可以用killall命令来杀死进程，和kill命令不同的是，在killall命令后面指定的是要杀死的进程的命令名称，而不是PID；和kill命令相同的是，用户也可以指定发送给进程的终止信号。

```
#  例如，要删除所有 apache 进程，可以用如下命令：
#  killall -9 apache
#  发送给进程的终止信号可以是信号的号码，也可以用信号的名称。
```

由于killall使用进程名称而不是PID，所以所有的同名进程都将被杀死。

作业控制

什么是作业控制

作业控制是指控制当前正在运行的进程的行为，也称为进程控制。作业控制是Shell的一个特性，使用户能在多个独立进程间进行切换。例如，用户可以挂起一个正在运行的进程，稍后再恢复它的运行。bash记录所有启动的进程并保持对所有已启动的进程的跟踪，在每一个正在运行的进程的生命期内的任何时候，用户可以任意地挂起进程或重新启动进程恢复运行。

例如，当用户使用Vi编辑一个文本文件，并需要中止编辑做其他事情时，利用作业控制，用户可以让编辑器暂时挂起，返回Shell提示符开始做其他的事情。其他事情做完以后，用户可以重新启动挂起的编辑器，返回到刚才中止的地方，就像用户从来没有离开编辑器一样。这只是一个例子，作业控制还有许多其他实际的用途。

实施作业控制的常用命令

下表列出了作业控制的常用命令或操作快捷键。

命令或快捷键	功能说明
cmd &	命令后的&符号表示将该命令放到后台运行，以免霸占终端
<Ctrl+d>	终止一个正在前台运行的进程（含有正常含义）
<Ctrl+c>	终止一个正在前台运行的进程（含有强行含义）
<Ctrl+z>	挂起一个正在前台运行的进程
jobs	显示后台作业和被挂起的进程
bg	重新启动一个挂起的作业，并且在后台运行
fg	把一个在后台运行的作业放到前台来运行

这些命令经常用于用户需要在后台运行，而却意外地把它放到了前台启动运行的时候。当一个命令在前台被启动运行时，它会禁止用户与Shell的交互，直到该命令结束。由于大多数命令的执行都能很快完成，所以一般情况下不会有什么问题。但是如果要运行的命令要花费很长时间的话，我们通常会把它放到后台，以便能在前台继续输入其他命令。此时，上面的命令就会派上用场了。

在进行作业控制时经常使用如下的作业标识符：

作业标识符	说明
%N	第N号作业
%S	以字符串S开头的被命令行调用的作业
%?S	包含字符串S的被命令行调用的作业
%+	默认作业(前台最后结束的作业，或后台最后启动的作业)，等同于 %%
%-	第二默认作业

作业控制举例

下面举一个简单的例子说明作业控制命令的使用。

```
#  列出所有正在运行的作业
$ jobs
#  在前台运行睡眠进程
$ sleep 100000
#  使用Ctrl+z挂起
```

```
[1]+  Stopped                 sleep 100000
#  在前台运行睡眠进程
$ sleep 200000
#  使用Ctrl+z挂起
[2]+  Stopped                 sleep 200000
#  在后台运行睡眠进程
$ sleep 300000 &
[3] 8941
#  运行cat命令
$ cat >example
This is a example.
#  使用Ctrl+z挂起
[4]+  Stopped                 cat >example
#  列出所有正在运行的作业
#  第1列是作业号，第2列中的+表示默认作业；-表示第二默认作业，第3列是作业状态
$ jobs
[1]   Stopped                 sleep 100000
[2]-  Stopped                 sleep 200000
[3]   Running                 sleep 300000 &
[4]+  Stopped                 cat >example
#  列出所有正在运行的作业，同时列出进程PID
$ jobs -l
[1]   8939 Stopped                 sleep 100000
[2]-  8940 Stopped                 sleep 200000
[3]   8941 Running                 sleep 300000 &
[4]+  8942 Stopped                 cat >example
#  将第二默认作业（以-标识）在后台继续运行
$ bg %-
[2]- sleep 200000 &
$ jobs -l
[1]-  8939 Stopped                 sleep 100000
[2]   8940 Running                 sleep 200000 &
[3]   8941 Running                 sleep 300000 &
[4]+  8942 Stopped                 cat >example
#  将1号作业在后台继续运行
$ bg %1
[1]- sleep 100000 &
$ jobs -l
[1]   8939 Running                 sleep 100000 &
[2]   8940 Running                 sleep 200000 &
[3]-  8941 Running                 sleep 300000 &
[4]+  8942 Stopped                 cat >example
#  将默认作业（以+标识）在前台继续运行
#  fg 等同于 fg %+；bg 等同于 bg %+
$ fg
cat >example
#  使用Ctrl+d结束进程
$ jobs -l
[1]   8939 Running                 sleep 100000 &
[2]-  8940 Running                 sleep 200000 &
[3]+  8941 Running                 sleep 300000 &
#  杀死 1号作业
$ kill %1
$ jobs -l
[1]   8939 Terminated              sleep 100000
[2]-  8940 Running                 sleep 200000 &
[3]+  8941 Running                 sleep 300000 &
#  杀死默认作业（以+标识）
$ kill %+
$ jobs -l
```

```
[2]-  8940 Running                 sleep 200000 &
[3]+  8941 Terminated              sleep 300000
```

- fg, bg和jobs命令只能接受作业号为参数
- kill 命令即可以接受作业号为参数也可以接受进程号为参数，例如上面的 kill %1 命令也可以使用 kill 8939

- 显示源文件
- 登录