进程和计划任务

如何产生一个进程：

1）执行程序或者是命令

2）计划任务

如何终止一个进程：

1）程序或者是命令执行完毕，自动终止

2）强制终止进程

进程管理

一.程序和进程的关系

程序：通常为二进制文件，存放在硬盘介质中，是可执行的代码和数据

进程：程序被触发后，执行者的权限与属性、程序的程序代码与所需数据等都会被加载到内存中，操作系统并给予这个内存内的单元一个标识符（PID）,进程就是正在运行中的程序。

父子进程：进程创建的进程为子进程

fork and exec 进程呼叫的流程：

1、系统先以fork的方式复制一个与父进程相同的暂存进程，这个进程与父进程唯一的区别就是PID不同，并增加一个PPID（父进程ID）

2、暂存进程以exec的方式加载实际要执行的程序，变成新进程。

常驻在内存当中的进程通常都是负责一些系统所提供的功能以服务用户的各项任务，被称为服务（daemon）。大致分为两类：

- 系统本身所需要的服务：crond、atd、rsyslogd等
- 负责网络的服务：named、postfix等，会启动一个端口，供外部客户端连接

1.静态查看进程的统计信息；

（1）ps：

a:显示终端下所有进程信息，包括其他用户的进程

u:显示进程的拥有者

x:显示当前用户所在终端下的进程信息，和a一起用，显示所有进程

-e:显示系统内所有进程信息

-l:长格式显示

-f:完整的格式显示

ps -ef | grep httpd

UID PID PPID C STIME TTY TIME CMD

root 5088 1 2 14:35 ? 00:00:00 /usr/sbin/httpd

apache 5091 5088 0 14:35 ? 00:00:00 /usr/sbin/httpd

apache 5092 5088 0 14:35 ? 00:00:00 /usr/sbin/httpd

UID:进程的拥有者

PID:进程号

PPID:父进程号

C:cpu使用的资源百分比

STIME: 开始时间

TTY: 运行进程的终端名字

TIME:进程运行的时间

CMD:命令

ps -aux 显示结果

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.1 19348 1332 ? Ss Aug02 0:02 /sbin/init

root 2 0.0 0.0 0 0 ? S Aug02 0:00 [kthreadd]

root 3 0.0 0.0 0 0 ? S Aug02 0:02 [migration/0]

拥有者 进程号 占用cpu百分比 占用内存百分比 占用的虚拟内存的大小 驻留内存大小 终端 进程状态 开始时间 运行时间 命令

STAT：进程状态

R：该进程正在运行

S：休眠进程sleep,可以被唤醒

D：不可被唤醒的睡眠状态

Z：僵尸进程，实际上该进程已经终止，但是却无法移除到内存之外

T：该进程正在跟踪或者已经停止

（2）pstree:树形结构查看进程信息

-a:显示完整信息

-u:列出名字

-p:列出PID号

pstree

```
        init─┬─abrtd
             ├─acpidclear
             ├─atd
             ├─automount───4*[{automount}]
             ├─bonobo-activati───{bonobo-activat}
```

init进程是所有进程的父进程

二、动态查看进程

top：实时查看系统运行状态 h 帮助 M 按照内存使用排序 P 按照CPU排序 N 按照PID来排序 q 退出

top - 15:07:52 up 2 days, 7:10, 9 users, load average: 0.00, 0.01, 0.00

系统时间 运行时长 当前9个用户登录系统 系统在1分钟、5分钟、15分钟的平均负载情况

Tasks: 192 total, 1 running, 191 sleeping, 0 stopped, 0 zombie

总共192个进程 1个正在运行 191个休眠 0个停止的 0个僵尸进程

Cpu(s): 1.5%us, 0.2%sy, 0.0%ni, 98.1%id, 0.2%wa, 0.0%hi, 0.0%si, 0.0%st

us 用户空间占用CPU百分比

sy 内核空间占用CPU百分比

ni 用户进程空间内改变过优先级的进程占用CPU百分比

id 空闲CPU百分比

wa 等待输入输出的CPU时间百分比，要特别注意，代表的是I/O情况

hi 硬中断（Hardware IRQ）占用CPU的百分比

si 软中断（Software Interrupts）占用CPU的百分比

st (Steal time) 是当 hypervisor 服务另一个虚拟处理器的时候，虚拟 CPU 等待实际 CPU 的时间的百分比

多核CPU的话，按数字1，可以展开其他CPU情况

Mem: 1016516k total, 914012k used, 102504k free, 100136k buffers

total 物理内存总量

used 使用的物理内存总量

free 空闲内存总量

buffers 用作内核缓存的内存量

Swap: 2097144k total, 7220k used, 2089924k free, 374972k cached

total 交换区总量

used 使用的交换区总量

free 空闲交换区总量

cached 缓冲的交换区总量。

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
23539 root      20   0  149m  25m 7040 S  4.3  2.5   4:40.87 Xorg
24019 root      20   0  308m  16m 9556 S  2.7  1.7   2:54.35 gnome-terminal
 5521 root      20   0  336m  15m  11m S  0.7  1.6   0:01.83 gnome-panel
 1598 root      20   0  165m 2588 2132 S  0.3  0.3   3:43.59 vmtoolsd
 5636 root      20   0 15032 1284  932 R  0.3  0.1   0:00.11 top
23679 root      20   0  403m  11m 8376 S  0.3  1.1   0:12.13 metacity
23706 root      20   0  446m  21m  11m S  0.3  2.2   3:20.17 vmtoolsd
```

VIRT 进程使用的虚拟内存总量，单位kb。VIRT=SWAP+RES

SWAP 进程使用的虚拟内存中，被换出的大小，单位kb。

RES 进程使用的、未被换出的物理内存大小，单位kb。RES=CODE+DATA

CODE 可执行代码占用的物理内存大小，单位kb

DATA 可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位kb

SHR 共享内存大小，单位kb

按f键可以编辑显示的列

仅动态查看某个进程的状态

```
# top -p 23539     //其中23539是进程号
```

2.top动态查看进程：

-P:顺序查看

3.pgrep:查看pid

-l:显示进程名

-U:指定用户

-t:指定终端

1、前台运行：

直接在终端运行命令

```
# firefox 172.16.254.251
```

会发现该程序一直占用终端，其他命令不能够再在这个终端运行

2、后台运行

1）命令执行的时候，直接将其放置于后台运行

firefox 172.16.254.251 &

[1] 6222

后台运行不会占用执行命令的终端，用户仍然可以使用这个终端做操作

2）对于一个已经运行的命令，如何将其放置于后台

firefox 172.16.254.251

^Z[1] Done firefox 172.16.254.251 //按下ctrl+z

[2]+ Stopped firefox 172.16.254.251

ctrl+z将前台进程放置于后台，但是该进程在后台是停止的状态

jobs //查看后台进程运行状态

[2]+ Stopped firefox 172.16.254.251

fg 2 //激活后台进程

[2]+ firefox 172.16.254.251 &

jobs

[2]+ Running firefox 172.16.254.251 &

bg 任务编号 ：指定任务在后台运行

fg 任务编号 ： 将后台运行的命令变成前台运行

fg 2

firefox 172.16.254.251

注意：不管你是前台运行，还是后台运行，只要终端一关闭，进程就停止了。

那么如何让一个命令或者程序脱离终端?

使用nohup命令。

nohup firefox 172.16.254.251 &

[1] 6537

nohup: ignoring input and appending output to `nohup.out' //按回车

IPC：Interconnect Process Communication 进程间通信

进程间通信的四种方式：

1、管道 pipe

2、信号 signal

3、消息 message

4、共享内存 shared memory

信号是类unix系统中的一种通信机制，它用来中断运行的进程执行某些操作

常用的信号：

查看信号的列表

kill -l

1) SIGHUP：重置进程的配置，即不停止服务的情况下，重新读取配置文件

kill -1 PID 或者kill -HUP PID

2) SIGINT：中断(interrupt)，相当于执行了ctrl+c

一个标签执行：

firefox 172.16.254.251

另一个标签执行

ps -ef | grep firefox

root 6855 6630 4 16:16 pts/0 00:00:01 /usr/lib64/firefox/firefox 172.16.254.251

root 6911 6898 0 16:17 pts/1 00:00:00 grep firefox

kill -2 6855 //发现第一个标签的命令被终止了

9) SIGKILL：强制杀死进程、无条件杀死

firefox 172.16.254.251

Killed

ps -ef | grep firefox

root 6982 6630 7 16:22 pts/0 00:00:01 /usr/lib64/firefox/firefox 172.16.254.251

root 7029 6898 0 16:23 pts/1 00:00:00 grep firefox

kill -9 6982

15) SIGTERM：终止进程，进程不一定会死

20) SIGTSTP：相当于按下ctrl+z的时候发送的信号

ps -ef | grep firefox

root 7104 6630 13 16:29 pts/0 00:00:01 /usr/lib64/firefox/firefox 172.16.254.251

root 7146 6898 0 16:29 pts/1 00:00:00 grep firefox

kill -20 7104

firefox 172.16.254.251

[1]+ Stopped firefox 172.16.254.251

pkill：按照进程的属性结束进程

按照进程名字杀死进程

pkill gnome-panel

按照用户名杀死进程

su - test

$ vim /etc/passwd

pkill -U test

killall：

killall 命令名字　//杀死全部的同名进程

xkill：杀死图形化资源

当执行了xkill之后，鼠标会变成“x”形，点到任何图形资源，就会终止该资源的运行

谦让值：nice

作用：指定或者调整用户进程的nice值

nice值越高，该进程抢占资源的能力越弱

nice值越低，该进程抢占资源的能力越强

范围：-20~19

1、相关命令

nice：运行时直接设置nice值

格式：nice -n 数字 command

renice：对于已经运行的，调整nice值

格式：renice 数字 PID

2、例子

cat loop.sh

#!/bin/bash

while true

do

echo hello > /dev/null

done

cp loop.sh loop1.sh

chmod +x loop*

1）在程序执行时，直接指定nice值

nice -n 7 ./loop.sh &

[1] 8890

```
# nice -n 14 ./loop1.sh &
      [2] 8896
# top
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 8937 root      27   7  103m 1192 1028 R 94.4  0.1   0:43.74 loop2.sh
 8890 root      27   7  103m 1196 1028 R 81.4  0.1   2:58.99 loop.sh
 8896 root      34  14  103m 1196 1028 R 16.9  0.1   2:12.98 loop1.sh
```

从上实验看出，nice越大，获得的资源越少，反之则反。

2）使用renice调整运行中的进程的nice值

```
# renice 19 8890
  8890: old priority 7, new priority 19
# top
# ps -ef | grep loop
  root      8890  7656 80 09:37 pts/0    00:06:26 /bin/bash ./loop.sh
  root      8896  7656 49 09:38 pts/0    00:03:49 /bin/bash ./loop1.sh
  root      8937  7656 91 09:40 pts/0    00:05:09 /bin/bash ./loop2.sh
  root      9044  7656  0 09:45 pts/0    00:00:00 grep loop
# killall loop.sh
# killall loop1.sh
  [1]   Terminated              nice -n 7 ./loop.sh
# killall loop2.sh

  [2]-  Terminated              nice -n 14 ./loop1.sh
  [3]+  Terminated              nice -n 7 ./loop2.sh
```

进程的调度：

ctrl+Z：转入后台并且停止

jobs：查看处于后台的任务

fg:将后台的进程恢复前台运行，可指定任务号

ctrl+c：中断正在执行的命令

kill：终止PID -9 强制终止

pkill:根据 特定条件终止进程

-t:停止终端进程

-U:按用户名终止进程

服务控制：

1、开机是否启动

chkconfig --list httpd 查看httpd服务开启情况

chkconfig --level 3 httpd on(off)

2、立刻调好服务：

可是使用启动脚本绝对路径 /etc/rc.d/init.d/httpd start

使用service 命令 service iptables stop

三.计划任务：

1.at: 一次性计划任务

at [HH:MM] [yy-mm-dd]

>command

>(ctrl+d)

atq:查看任务

atrm:删除任务 后边指定要删除的任务号

时间格式也可以：now + count time-units （minutes，hours，days，weeks）

2.crontab 周期性计划任务

crontab命令----crond服务

全局设定的配置文件：/etc/crontab

*任意 ,多个不连续 - 连续 /间隔频率

系统默认的设置：/etc/cron.*

用户自定义的配置文件：/var/spool/cron/user_name

使用：

编辑任务：

crontab -e [-u user_name]

查看任务：

crontab -l [-u user_name]

删除任务：

crontab -r [-u user_name]

练习：1、每周一的早上 7:50 自动清空 FTP 服务器公共目录“/var/ftp/pub”中的数据

2、每天晚上的 10:30 自动执行任务，完成以下操作

- 显示当前的系统时间并查看已挂载磁盘分区的磁盘使用情况
- 将输出结果追加保存到文件/var/log/df.log中，以便持续观察硬盘空间的变化

补充资料：

PRI ： 进程优先权，代表这个进程可被执行的优先级，其值越小，优先级就越高，越早被执行

NI ： 进程Nice值，代表这个进程的优先值

%nice ： 改变过优先级的进程的占用CPU的百分比

PRI是比较好理解的，即进程的优先级，或者通俗点说就是程序被CPU执行的先后顺序，此值越小进程的优先级别越高。那NI呢？就是我们所要说的nice值了，其表示进程可被执行的优先级的修正数值。如前面所说，PRI值越小越快被执行，那么加入nice值后，将会使得PRI变为：PRI(new)=PRI(old)+nice。由此看出，PR是根据NICE排序的，规则是NICE越小PR越前（小，优先权更大），即其优先级会变高，则其越快被执行。如果NICE相同则进程uid是root的优先权更大。

在LINUX系统中，Nice值的范围从-20到+19（不同系统的值范围是不一样的），正值表示低优先级，负值表示高优先级，值为零则表示不会调整该进程的优先级。具有最高优先级的程序，其nice值最低，所以在LINUX系统中，值-20使得一项任务变得非常重要；与之相反，如果任务的nice为+19，则表示它是一个高尚的、无私的任务，允许所有其他任务比自己享有宝贵的CPU时间的更大使用份额，这也就是nice的名称的来意。

进程和线程区别：

首先来一句概括的总论：**进程和线程都是一个时间段的描述，是CPU工作时间段的描述。**

下面细说背景：

CPU+RAM+各种资源（比如显卡，光驱，键盘，GPS, 等等外设）构成我们的电脑，但是电脑的运行，实际就是CPU和相关寄存器以及RAM之间的事情。

一个最最基础的事实：CPU太快，太快，太快了，寄存器仅仅能够追的上他的脚步，RAM和别的挂在各总线上的设备完全是望其项背。那当多个任务要执行的时候怎么办呢？轮流着来?或者谁优先级高谁来？不管怎么样的策略，一句话就是在CPU看来就是轮流着来。

一个必须知道的事实：执行一段程序代码，实现一个功能的过程介绍 ，当得到CPU的时候，相关的资源必须也已经就位，就是显卡啊，GPS啊什么的必须就位，然后CPU开始执行。这里除了CPU以外所有的就构成了这个程序的执行环境，也就是我们所定义的程序上下文。当这个程序执行完了，或者分配给他的CPU执行时间用完了，那它就要被切换出去，等待下一次CPU的临幸。在被切换出去的最后一步工作就是保存程序上下文，因为这个是下次他被CPU临幸的运行环境，必须保存。

在CPU看来所有的任务都是一个一个的轮流执行的，具体的轮流方法就是：**先加载程序A的上下文，然后开始执行A，保存程序A的上下文，调入下一个要执行的程序B的程序上下**

文，然后开始执行B,保存程序B的上下文。。。。

进程和线程就是这样的背景出来的，**两个名词不过是对应的CPU时间段的描述，名词就是这样的功能。**

- **进程就是包换上下文切换的程序执行时间总和 = CPU加载上下文+CPU执行+CPU保存上下文**

线程是什么呢?

进程的颗粒度太大，每次都要有上下的调入，保存，调出。如果我们把进程比喻为一个运行在电脑上的软件，那么一个软件的执行不可能是一条逻辑执行的，必定有多个分支和多个程序段，就好比要实现程序A，实际分成 a，b，c等多个块组合而成。那么这里具体的执行就可能变成：

程序A得到CPU =》CPU加载上下文，开始执行程序A的a小段，然后执行A的b小段，然后再执行A的c小段，最后CPU保存A的上下文。

这里a，b，c的执行是共享了A的上下文，CPU在执行的时候没有进行上下文切换的。这**里的a，b，c就是线程，也就是说线程是共享了进程的上下文环境，的更为细小的CPU时间段。**

总结：

进程和线程都是一个时间段的描述，是CPU工作时间段的描述，不过是颗粒大小不同。