22.系统进程管理

先说说概念

1、程序

说起进程，就不得不先说下程序。先看定义：程序是指令和数据的有序集合，其本身没有任何运行的含义，是一个静态的概念。

2、进程（process）

狭义定义：进程就是一段程序的执行过程，进程则是在处理机上的一次执行过程，它是一个动态的概念。其实进程是包含程序的，进程的执行离不开程序，进程中的文本区域就是代码区，也就是程序。

广义定义：进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。它是操作系统动态执行的基本单元，在传统的操作系统中，进程既是基本的分配单元，也是基本的执行单元。

简单的来讲进程的概念主要有两点：

第一，进程是一个“执行中的程序”。程序是一个没有生命的实体，只有处理器赋予程序生命时，它才能成为一个活动的实体，我们称其为进程。

第二，进程是一个实体，拥有资源。每一个进程都有它自己的地址空间，一般情况下，包括文本区域（text region）、数据区域（data region）和堆栈（stack region）。文本区域存储处理器执行的代码；数据区域存储属性变量和进程执行期间使用的动态分配的进程的内存地址；堆栈区域存储着活动过程调用的指令和临时变量。

说到进程，提一下线程：线程是进程的一个执行流，是CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。一个进程由几个线程组成，线程与同属一个进程的其他的线程共享进程所拥有的全部资源。

总结：”进程——资源分配的最小单位，线程——程序执行的最小单位”

Linux中的进程状态：

R（TASK_RUNNING），可执行状态，只有在该状态的进程才可能在CPU上运行。

S（TASK_INTERRUPTIBLE），可中断的睡眠状态，处于这个状态的进程因为等待某某事件的发生（比如等待socket连接、等待信号量），而被挂起。

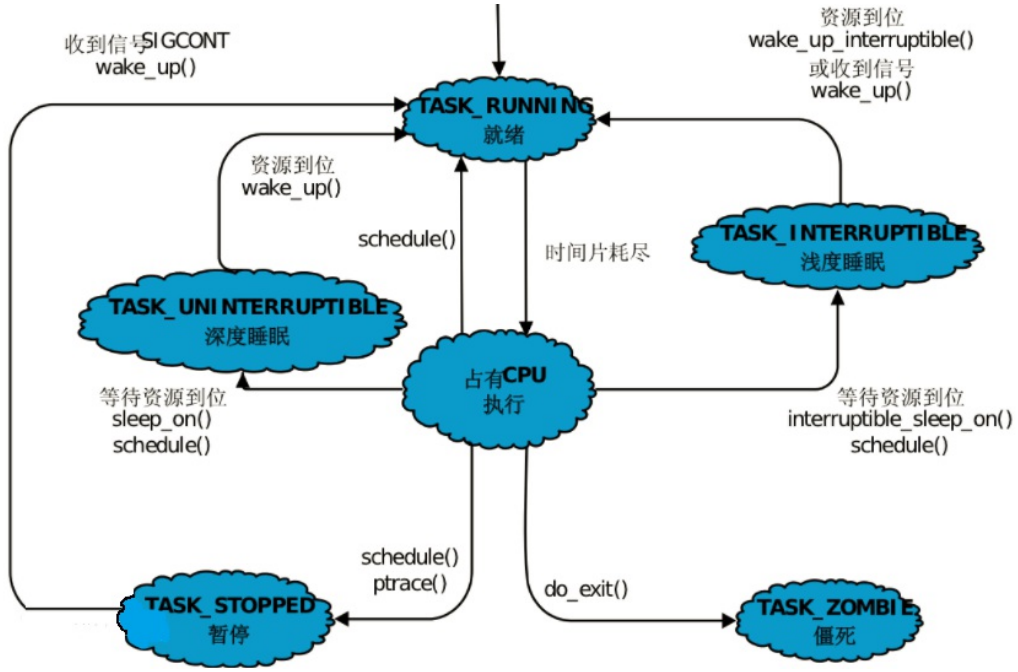
D（TASK_UNINTERRUPTIBLE），不可中断的睡眠状态，与TASK_INTERRUPTIBLE状态类似，进程处于睡眠状态，但是此刻进程是不可中断的，这种进程的特征就是kill -9命令杀不死，而TASK_UNINTERRUPTIBLE状态存在的意义就在于，内核的某些处理流程

是不能被打断的。

T（TASK_STOPPED or TASK_TRACED），暂停状态或跟踪状态，向进程发送一个SIGSTOP信号，它就会因响应该信号而进入TASK_STOPPED状态，向进程发送一个SIGCONT信号，可以让其从TASK_STOPPED状态恢复到TASK_RUNNING状态，被跟踪的进程下一个断点，进程在断点处停下来的时候就处于TASK_TRACED状态。

Z（TASK_DEAD - EXIT_ZOMBIE），退出状态，进程成为僵尸进程，进程在退出的过程中，处于TASK_DEAD状态，在这个退出过程中，进程占有的所有资源将被回收，除了task_struct结构（以及少数资源）以外。于是进程就只剩下task_struct这么个空壳，故称为僵尸。

之所以保留task_struct，是因为task_struct里面保存了进程的退出码、以及一些统计信息。而其父进程很可能会关心这些信息。



3、父进程和子进程

子进程是父进程的复制品

父进程先执行fork()函数，执行的结果是系统中多出了一个跟父进程内容完全一样的进程，这个新进程被称为子进程，当然该进程属性中父进程指针是指向第一个进程的。

前后两个进程各自有自己的地址空间，形式上有点像把一个文件拷贝了一个副本。虽然资源也相互独立，但拷贝时父进程执行过程已生成的数据，子进程也拷了一份。说简单点像一个执行到半路的程序突然在系统中多出了一个孪生兄弟，什么都跟自己一样，但要管自己叫老爸。

当然这样的简单复制本身是没什么用处的。要让它发挥作用，还需要再执行exec(B)函数，这个函数可以让当前进程转而执行另一个可执行代码（一个新的程序）。简单的说

进程本来在执行A程序，一旦执行到这个函数，就转而开始执行B程序。

至此，父子两进程就变的不一样了，但不管它们各自执行的什么代码，其父子关系不会改变，在父进程中可以使用子进程的进程ID（在执行fork()时的返回值中得到）来终止子进程的执行。当然子进程也可以因为自己的执行程序结束而自然终止执行

父进程和子进程先后执行的问题，是这样的，在fork之后，是父进程先执行，然后一个时间片到达之后就是子进程再执行了。

每一个子进程都有一个父进程，当进程终止或者结束的时候，都会给父进程发送一个SIGCHLD信号，系统默认是父进程忽略这个信号，如果父进程希望被告知其子进程的这种状态改变，则应该捕获这个信号，捕捉函数一般是wait函数来取得子进程ID和子进程状态。CentOS6.x，系统启动的第一个进行是init，CentOS7.x，系统启动的第一个进行是systemd，其他的所有进行，都是第一个系统进程的后代。

wait函数是父进程等待子进程结束，也就是说当子进程结束的时候会发送给父进程一个信号SIGCHID，这时候父进程通过wait函数接收到这个信号，这时候父进程就知道子进程结束了。这个正好用在shell解析器的编写里面，shell解析器作为父进程，而命令行命令作为子进程，当子进程结束的时候就会告诉父进程，这时候父进程就可以提示输入下一个命令了。

4、进程的属性

进程ID（PID）：是唯一的数值，用来区分进程
父进程的ID（PPID）
启动进程的用户ID（UID）和所归属的组（GID）
进程状态：RSTZD
进程执行的优先级
进程所连接的终端名
进程资源占用：比如占用资源大小（内存、CPU占用量）

5、查看进程与管理

ps查看进程工具

举例1：

```
# ps -axu | more
```

参数：
a：显示跟当前终端关联的所有进程
u：基于用户的格式显示（U：显示某用户ID所有的进程）
x：显示所有进程，不以终端机来区分
注： 最后一列[xxxx] 使用方括号括起来的进程是内核态的进程。 没有括起来的是用

户态进程。

上面的参数输出每列含意：

USER: 启动这些进程的用户

PID: 进程的ID

%CPU 进程占用的CPU百分比；

%MEM 占用内存的百分比；

VSZ：进程占用的虚拟内存大小（单位：KB）

RSS：进程占用的物理内存大小（单位：KB）

STAT：该程序目前的状态，Linux进程有5种基本状态：

R ：该程序目前正在运作，或者是可被运作；

S ：该程序目前正在睡眠当中 ，但可被某些讯号(signal) 唤醒。

T ：该程序目前正在侦测或者是停止了；

Z ：该程序应该已经终止，但是其父程序却无法正常的终止他，造成 zombie （疆尸）程序的状态

D 不可中断状态.

5个基本状态后，还可以加一些字母，比如：Ss、R+

它们含意如下:：

<: 表示进程运行在高优先级上

N: 表示进程运行在低优先级上

L: 表示进程有页面锁定在内存中

s: 表示进程是控制进程

l: 表示进程是多线程的

+: 表示当前进程运行在前台

START：该 process 被触发启动的时间；

TIME ：该 process 实际使用 CPU 运作的时间。

COMMAND：该程序的实际指令

举例2：

ps -ef

参数：

-e 显示所有进程

-f 显示完整格式输出

包含的信息如下

UID: 启动这些进程的用户

PID: 进程的ID

PPID: 父进程的进程号

C：进程生命周期中的CPU利用率

STIME：进程启动时的系统时间

TTY：表明进程在哪个终端设备上运行。如果显示 ?表示与终端无关，这种进程一般是内核态进程。另外， tty1-tty6 是本机上面的登入者程序，若为 pts/0 等，则表示运行在虚拟终端上的进程。

TIME：运行进程一共累计占用的CPU时间

CMD：启动的程序名称

uptime查看系统负载和top动态管理进程

uptime查看CPU负载工具

uptime，这个命令我们在前面的课中，讲w命令的时候，有相同的信息的输入出

```
[root@CentOS7 ~]# w
 13:38:43 up  1:41,  1 user,  load average: 0.03, 0.04, 0.09
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    192.168.238.1    13:38    2.00s  0.09s  0.01s w
[root@CentOS7 ~]# uptime
 13:38:47 up  1:41,  1 user,  load average: 0.03, 0.04, 0.09
[root@CentOS7 ~]#
```

top命令

top　#top弹出的每行信息含意如下：

第一行内容和uptime弹出的信息一样

第二、三行是进程和CPU的信息，当有多个CPU时，这些内容可能会超过两行。

```
Tasks: 120 total,   1 running, 119 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

Tasks: 120 total：进程总数

1 running：正在运行的进程数

119 sleeping：睡眠的进程数

0 stopped：停止的进程数

0 zombie：僵尸进程数

Cpu(s): 0.2 us 系统用户进程使用CPU百分比。

0.2 sy 内核中的进程占用CPU百分比

0.0 ni 用户进程空间内改变过优先级的进程占用CPU百分比

99.7 id 空闲CPU百分比

0.0 wa cpu等待I/O完成的时间总量。

0.0 hi（了解）　硬中断消耗时间

硬中断，占的CPU百分比。 硬中断是由硬件产生的，比如，像磁盘，网卡，键盘，时钟等。每个设备或设备集都有它自己的IRQ（中断请求）。基于IRQ，CPU可以将相应的请求分发到对应的硬件驱动上（注：硬件驱动通常是内核中的一个子程序，而不是一个独立的进

程）。

0.0 si（了解）软中断消耗时间

软中断，占的CPU百分比。 软中断是一些对I/O的请求。这些请求会调用内核中可以调度I/O发生的程序。对于某些设备，I/O请求需要被立即处理，而磁盘I/O请求通常可以排队并且可以稍后处理。根据I/O模型的不同，进程或许会被挂起直到I/O完成，此时内核调度器就会选择另一个进程去运行。I/O可以在进程之间产生并且调度过程通常和磁盘I/O的方式是相同。

0.0 st （steal 偷） st：虚拟机偷取物理的时间。

比如：物理机已经运行了KVM，XEN虚拟机。KVM虚拟机占用物理机的cpu时间

第四、五行内存信息，free命令显示信息是一样的

```
KiB Mem :  2028116 total,  1681116 free,   103588 used,   243412 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1746976 avail Mem
```

第7行进程信息

PID	进程id
USER	进程所有者的用户名
PR	优先级（由内核动态调整），不是用户能影响到的
NI	进程优先级。 nice值。负值表示高优先级，正值表示低优先级，用户可以自己调整
VIRT（virtual memory usage）	虚拟内存，是进程正在使用的所有内存（ps中标为VSZ） VIRT：virtual memory usage 虚拟内存 1、进程“需要的”虚拟内存大小，包括进程使用的库、代码、数据等 2、假如进程申请100m的内存，但实际只使用了10m，那么它会增长100m，而不是实际的使用量
RES（resident memory usage）	是进程所使用的物理内存。实际实用内存（ps中标为RSS） RES：resident memory usage 常驻内存 1、进程当前使用的内存大小，但不包括swap out 2、包含其他进程的共享 3、如果申请100m的内存，实际使用10m，它只增长10m，与VIRT相反 4、关于库占用内存的情况，它只统计加载的库文件所占内存大小
SHR	共享内存大小，单位kb SHR：shared memory 共享内存 1、除M了自身进程的共享内存，也包括其他进程的共享内存 2、虽然进程只使用了几个共享库的函数，但它包含了整个共享库的大小 3、计算某个进程所占的物理内存大小公式：RES－SHR 4、swap out后，它将会降下来
S	进程状态。 D=不可中断的睡眠状态 R=运行中或可运行 S=睡眠中 T=已跟踪/已停止 Z=僵停
%CPU	上次更新到现在的CPU时间占用百分比
%MEM	进程使用的物理内存百分比
TIME+	进程使用的CPU时间总计，单位1/100秒
COMMAND	命令名/命令行

top快捷键：

默认3s刷新一次，按s修改刷新时间

按空格 ：立即刷新。

q退出

P：按CPU排序

M：按内存排序

T：按时间排序

数字键1：显示每个内核的CPU使用率

u/U：指定显示的用户

h:帮助

lsof命令

lsof命令用于查看运行进程读了那些文件，端口(TCP、UDP)被那些进程使用

-i :端口：列出符合条件的进程。

-p 进程号：列出指定进程号所打开的文件；

例：

```
# vim a.txt
# ps -axu | grep a.txt
root      43641  0.8  0.2 151744  5280 pts/3    S+   18:19   0:00 vim a.txt
root      43652  0.0  0.0 112676   996 pts/1    S+   18:19   0:00 grep --color=auto a.txt
# lsof -p  43641  #一般用于查看木马进程，在读哪些文件
# lsof -i :22    #用于查看端口，或查看黑客开启的后门端口是哪个进程在使用
```

pstree工具使用，# yum install -y psmisc

pstree：（display a tree of processes）以树状图显示进程，只显示进程的名字，且相同进程合并显示。

格式：pstree 或 pstree -p

以树状图显示进程，还显示进程PID。

```
# pstree -p
```

前后台进程

Linux后台进程与前台进程的区别

前台进程:是在终端中运行的命令，那么该终端就为进程的控制终端，一旦这个终端关闭，这个进程也随着消失

后台进程: 也叫守护进程（Daemon），是运行在后台的一种特殊进程，不受终端控制，它不需要与终端交互；

Linux的大多数服务器就是用守护进程实现的。比如，Web服务器httpd等。

进程的前台与后台运行

跟系统任务相关的几个命令（了解）

&	用在一个命令的最后，可以把这个命令放到后台执行.
ctrl + z	将一个正在前台执行的命令放到后台，并且暂停.
jobs	查看当前有多少在后台运行的命令
fg（foreground process）	将后台中的命令调至前台继续运行,如果后台中有多个命令，可以用 fg %jobnumber将选中的命令调出，%jobnumber是通过jobs命令查到的后台正在执行的命令的序号(不是pid)
bg(background process)	将一个在后台暂停的命令，变成继续执行;如果后台中有多个命令，可以用bg %jobnumber将选中的命令调出，%jobnumber是通过jobs命令查到的后台正在执行的命令的序号(不是pid)

实战恢复被挂起的进程（了解）

例： vim a.txt 按下： ctrl+z

```
#  vim a.txt      #打开后，然后执行 ctrl+z
# ps -axu | grep vim
# jobs       #查看当前有多少在后台运行的命令
   [1]+  已停止                 vim a.txt
# fg 1    #将后台挂起的进程恢复到前台运行
```

kill关闭进程

关闭进程有三个命令：kill killall pkill

kill 进程号 # 通过进程号关闭单个进程

killall和pkill # 通过进程名字杀死指的进程

通过信号的方式来控制进程的

kill -l =====> 列出所有支持的信号（了解） 用最多的是： 9 信号

```
信号编号   信号名
1）        SIGHUP      重新加载配置
2）        SIGINT       键盘中断   crtl+c
3）         SIGQUIT      退出
9）       SIGKILL       强制终止
15）      SIGTERM     终止（正常结束），缺省信号
18）        SIGCONT     继续
19）       SIGSTOP      停止
20）       SIGTSTP      暂停 crtl+z
```

只有第9种信号(SIGKILL)才可以无条件终止进程，其他信号进程都有权利忽略。

下面是常用的信号：

```
HUP     1     终端断线
INT      2     中断（同 Ctrl + C）
```

QUIT 3 退出（同 Ctrl + \）

TERM 15 终止

KILL 9 强制终止

CONT 18 继续（与STOP相反， fg/bg命令）

STOP 19 暂停（同 Ctrl + Z）

1) SIGHUP

本信号在用户终端连接(正常或非正常)结束时发出, 通常是在终端的控制进程结束时, 通知同一session内的各个作业, 这时它们与控制终端不再关联。

登录Linux时，系统会分配给登录用户一个终端(Session)。在这个终端运行的所有程序，包括前台进程组和后台进程组，一般都属于这个 Session。当用户退出Linux登录时，前台进程组和后台有对终端输出的进程将会收到SIGHUP信号。这个信号的默认操作为终止进程，因此前台进 程组和后台有终端输出的进程就会中止。不过可以捕获这个信号，比如wget能捕获SIGHUP信号，并忽略它，这样就算退出了Linux登录，wget也 能继续下载。

此外，对于与终端脱离关系的守护进程，这个信号用于通知它重新读取配置文件。

2) SIGINT

程序终止(interrupt)信号, 在用户键入INTR字符(通常是Ctrl-C)时发出，用于通知前台进程组终止进程。

3) SIGQUIT

和SIGINT类似, 但由QUIT字符(通常是Ctrl-)来控制. 进程在因收到SIGQUIT退出时会产生core文件, 在这个意义上类似于一个程序错误信号。

4) SIGILL

执行了非法指令. 通常是因为可执行文件本身出现错误, 或者试图执行数据段. 堆栈溢出时也有可能产生这个信号。

5) SIGTRAP

由断点指令或其它trap指令产生. 由debugger使用。

6) SIGABRT

调用abort函数生成的信号。

7) SIGBUS

非法地址, 包括内存地址对齐(alignment)出错。比如访问一个四个字长的整数, 但其地址不是4的倍数。它与SIGSEGV的区别在于后者是由于对合法存储地址的非法访问触发的(如访问不属于自己存储空间或只读存储空间)。

8) SIGFPE

在发生致命的算术运算错误时发出. 不仅包括浮点运算错误, 还包括溢出及除数为0等其它所有的算术的错误。

9) SIGKILL

用来立即结束程序的运行. 本信号不能被阻塞、处理和忽略。如果管理员发现某个进程终止不了，可尝试发送这个信号。

10) SIGUSR1

留给用户使用

11) SIGSEGV

试图访问未分配给自己的内存, 或试图往没有写权限的内存地址写数据.

12) SIGUSR2

留给用户使用

13) SIGPIPE

管道破裂。这个信号通常在进程间通信产生，比如采用FIFO(管道)通信的两个进程，读管道没打开或者意外终止就往管道写，写进程会收到SIGPIPE信号。此外用Socket通信的两个进程，写进程在写Socket的时候，读进程已经终止。

14) SIGALRM

时钟定时信号, 计算的是实际的时间或时钟时间. alarm函数使用该信号.

15) SIGTERM

程序结束(terminate)信号, 与SIGKILL不同的是该信号可以被阻塞和处理。通常用来要求程序自己正常退出，shell

命令kill缺省产生这个信号。如果进程终止不了，我们才会尝试SIGKILL。

17) SIGCHLD

子进程结束时，父进程会收到这个信号。

如果父进程没有处理这个信号，也没有等待(wait)子进程，子进程虽然终止，但是还会在内核进程表中占有表项，这时的子进程称为僵尸进程。这种情 况我们应该避免(父进程或者忽略SIGCHILD信号，或者捕捉它，或者wait它派生的子进程，或者父进程先终止，这时子进程的终止自动由init进程来接管)。

18) SIGCONT

让一个停止(stopped)的进程继续执行. 本信号不能被阻塞. 可以用一个handler来让程序在由stopped状态变为继续执行时完成特定的工作. 例如, 重新显示提示符...

19) SIGSTOP

停止(stopped)进程的执行. 注意它和terminate以及interrupt的区别:该进程还未结束, 只是暂停执行. 本信号不能被阻塞, 处理或忽略.

20) SIGTSTP

停止进程的运行, 但该信号可以被处理和忽略. 用户键入SUSP字符时(通常是Ctrl-Z)发出这个信号

21) SIGTTIN

当后台作业要从用户终端读数据时, 该作业中的所有进程会收到SIGTTIN信号. 缺省时这些进程会停止执行.

22) SIGTTOU

类似于SIGTTIN, 但在写终端(或修改终端模式)时收到.

23) SIGURG

有"紧急"数据或out-of-band数据到达socket时产生.

24) SIGXCPU

超过CPU时间资源限制. 这个限制可以由getrlimit/setrlimit来读取/改变。

25) SIGXFSZ

当进程企图扩大文件以至于超过文件大小资源限制。

26) SIGVTALRM

虚拟时钟信号. 类似于SIGALRM, 但是计算的是该进程占用的CPU时间.

27) SIGPROF

类似于SIGALRM/SIGVTALRM, 但包括该进程用的CPU时间以及系统调用的时间.

28) SIGWINCH

窗口大小改变时发出.

29) SIGIO

文件描述符准备就绪, 可以开始进行输入/输出操作.

30) SIGPWR

Power failure

31) SIGSYS

非法的系统调用。

在以上列出的信号中，程序不可捕获、阻塞或忽略的信号有：SIGKILL,SIGSTOP

不能恢复至默认动作的信号有：SIGILL,SIGTRAP

默认会导致进程流产的信号有：

SIGABRT,SIGBUS,SIGFPE,SIGILL,SIGIOT,SIGQUIT,SIGSEGV,SIGTRAP,SIGXCPU,SIGXFSZ

默认会导致进程退出的信号有：

SIGALRM,SIGHUP,SIGINT,SIGKILL,SIGPIPE,SIGPOLL,SIGPROF,SIGSYS,SIGTERM,SIGUSR1,SIGUSR2,SIGVTALRM

默认会导致进程停止的信号有：SIGSTOP,SIGTSTP,SIGTTIN,SIGTTOU

默认进程忽略的信号有：SIGCHLD,SIGPWR,SIGURG,SIGWINCH

例1： kill和killall终止进程

 # kill -9 2342

```
# killall sshd
# pkill sshd
```

进程的优先级管理

为什么要有进程优先级？自从多任务操作系统诞生以来，进程执行占用cpu的能力就是一个必须要可以人为控制的事情。因为有的进程相对重要，而有的进程则没那么重要。

进程优先级起作用的方式从发明以来基本没有什么变化，无论是只有一个cpu的时代，还是多核cpu时代，都是通过控制进程占用cpu时间的长短来实现的。就是说在同一个调度周期中，优先级高的进程占用的时间长些，而优先级低的进程占用的短些。

优先级取值范围为（-20,19），越小优先级越高，默认优先级是0

命令1：nice 指定程序的运行优先级

格式：nice -n command

命令2：renice 改变程序的运行优先级

格式：renice -n pid

例1：指定运行vim的优先级为5，然后 查看改变的结果，然后在改变优先级

```
# nice -n 5 vim a.txt
```

输入内容，然后ctrl+z 挂起

通过ps查看这个文件的PID号

```
# ps -aux|grep vim
```

通过top命令查看优先级

```
# top -p 26154
```

改变正在运行的进程的优先级

screen命令

首先看一个问题：需要备份1T数据，我在CRT上直接执行备份脚本bk.sh 或让bk.sh & 放到后台运行， 当关了CRT后，bk.sh & 还在后台执行吗？

答：不能，CRT长时间连接的过程中，如果本地网络偶尔断开或CRT关闭，都会让后台运行的备份命令停止运行的。这个时候是需要： srceen命令的

screen概述和安装

Screen中有会话的概念,，用户可以在一个screen会话中创建多个screen窗口，在每一个screen窗口中就像操作一个真实的telnet/SSH连接窗口那样。

用域名安装screen：# yum -y install screen

creen使用方法

直接在命令行键入screen命令回车：# screen

Screen将创建一个执行shell的全屏窗口。你可以执行任意shell程序，就像在ssh窗口CRT中那样

常用screen参数

```
screen -S test  ->
```

新建一个叫test的会话

screen -ls　　->　　列出当前所有的会话

screen -r test　->　　回到test会话

exit　->　　退出test会话

例子：

screen　-S test　　#进入

vim a.txt　#执行命令

#在screen当前窗口键入快捷键Ctrl+a+d

[detached from 44074.pts-3.xuegod63]　　#分离出来独立的一个会话离开了，但这个程序不让它停止，过断时间之后回来了

screen -ls　#查看已经建立的会话ID

重新连接会话：# screen -r 会话ID

exit　#不想使用screen 会话了，执行：exit退出。