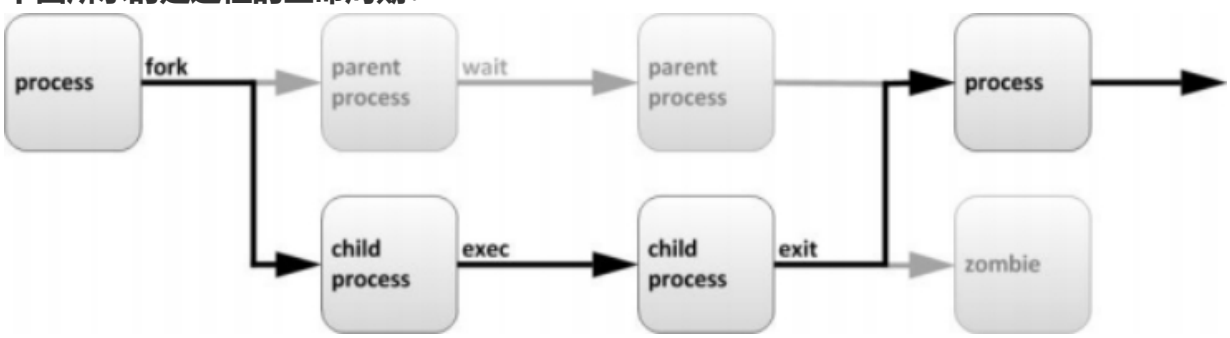
1.什么是进程?

程序一般是放置在物理磁盘中，然后通过用户的执行来触发。触发后会加载到内存中成为一个个体，那就是进程。

进程有以下特点：
1.1.一个应用程序对应一个或多个进程(多进程编程,并发方式),但是一个进程对应唯一的应用程序
1.2.进程是系统资源(内存,信号,锁,定时器)分配的单位.进程之间的空间完全独立，之间必须通过某个通信机制才可通信
1.3.进程通信机制:管道(命名，匿名),消息队列,共享存储,磁盘文件,网络套接字,信号量集(锁)
1.4.线程是轻量级的进程，一个进程可以产生若干线程，CPU调度单位为线程，这些线程共享进程空间， 竞争激烈必须同步机制

程序： 二进制文件，静态 /bin/date,/usr/sbin/sshd
进程： 是程序运行的过程， 动态，有生命周期及运行状态。

下图所示的是进程的生命周期：



描述如下：
父进程复制自己的地址空间（fork [fɔ:k] 分叉）创建一个新的（子）进程结构。每个新进程分配一个唯一的 进程 ID （PID），满足跟踪安全性之需。PID 和 父进程 ID （PPID）是子进程环境的元素，任何进程都可 以创建子进程，所有进程都是第一个系统进程的后代。

centos5 或 6PID 为 1 的进程是： init
centos7 PID 为 1 的进程是： systemd

2.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：状态分为运行 R、休眠 S、僵尸 Z
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）

3.使用 ps 查看进程工具

3.1：常用的选项组合是 ps -aux

常用的参数：
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
f：以树状形式显示

```
[root@exercise1 ~]# ps -aux | more
```

```
[root@home ~]# ps -aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.1 125220  3716 ?        Ss   09:10   0:02 /usr/lib/systemd/systemd --switched-root --system --deserialize 21
root          2  0.0  0.0      0     0 ?        S    09:10   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    09:10   0:00 [ksoftirqd/0]
root          5  0.0  0.0      0     0 ?        S<   09:10   0:00 [kworker/0:0H]
root          7  0.0  0.0      0     0 ?        S    09:10   0:00 [migration/0]
root          8  0.0  0.0      0     0 ?        S    09:10   0:00 [rcu_bh]
root          9  0.0  0.0      0     0 ?        R    09:10   0:00 [rcu_sched]
```

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 (可说是 idle 状态啦！)，但可被某些讯号(signal) 唤醒。
T ： 该程序目前正在挂起或者是停止了;
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+ ，如下图：

```
root         525  0.0  0.0  55452   892 ?        S<sl 09:10   0:00 /sbin/auditd
dbus         548  0.0  0.0  24548  1676 ?        Ss   09:10   0:00 /bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --
root         550  0.0  0.3 310720  6568 ?        Ssl  09:10   0:00 /usr/sbin/rsyslogd -n
polkitd      553  0.0  0.6 534144 13052 ?        Ssl  09:10   0:00 /usr/lib/polkit-1/polkitd --no-debug
root         554  0.0  0.0  24204  1676 ?        Ss   09:10   0:00 /usr/lib/systemd/systemd-logind
root         555  0.0  0.3  99608  6084 ?        Ss   09:10   0:00 /usr/bin/VGAuthService -s
root         556  0.0  0.3 305296  6296 ?        Ssl  09:10   0:01 /usr/bin/vmtoolsd
root         558  0.0  0.4 545976  9248 ?        Ssl  09:10   0:00 /usr/sbin/NetworkManager --no-daemon
root         561  0.0  0.0  25856   940 ?        Ss   09:10   0:00 /usr/sbin/atd -f
root         562  0.0  0.0 126232  1596 ?        Ss   09:10   0:00 /usr/sbin/crond -n
chrony       565  0.0  0.0 115640  1772 ?        S    09:10   0:00 /usr/sbin/chronyd
root         803  0.0  0.8 113372 15940 ?        S    09:10   0:00 /sbin/dhclient -d -q -sf /usr/libexec/nm-dhcp-helper -pf /var/run/dh
root         865  0.0  0.2 105996  4080 ?        Ss   09:10   0:00 /usr/sbin/sshd -D
root         868  0.0  0.9 562400 18620 ?        Ssl  09:10   0:00 /usr/bin/python -Es /usr/sbin/tuned -l -P
root         885  0.0  0.0 110056   840 tty1     Ss+  09:10   0:00 /sbin/agetty --noclear tty1 linux
root        1105  0.0  0.1  89544  2084 ?        Ss   09:10   0:00 /usr/libexec/postfix/master -w
```

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 进程(process) 被触发启动的时间;
TIME ： 该进程(process) 实际使用 CPU运作的时间。
COMMAND ： 该程序的实际指令

例 1： 查看进程状态

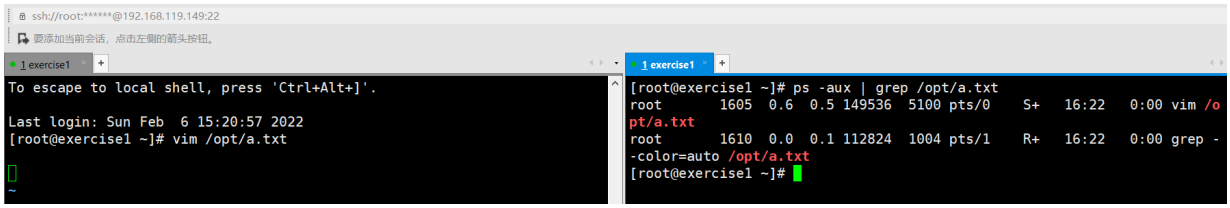
```
1 [root@exercise1 ~]# vim /opt/a.txt
```

#在另一个终端执行:

[root@exercise1 ~]# ps -aux | grep /opt/a.txt #查看状态 S 表示睡眠状态， + 表示前台

root 1605 0.6 0.5 149536 5100 pts/0 S+ 16:22 0:00 vim /opt/a.txt

root 1610 0.0 0.1 112824 1004 pts/1 R+ 16:22 0:00 grep --color=auto /opt/a.txt

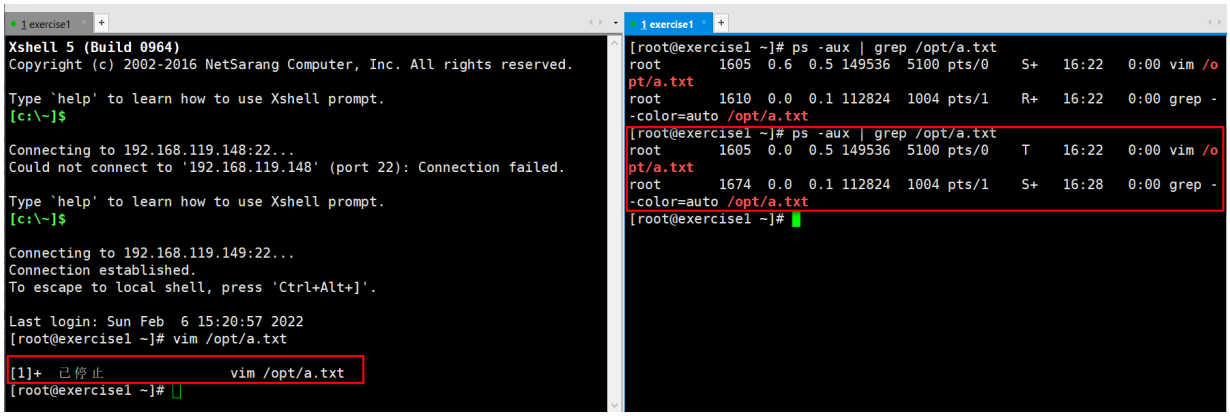


在vim /opt/a.txt 这个终端上 按下： ctrl+z

```
1 [1]+  已停止               vim /opt/a.txt
```

在另一个终端执行:

```
[root@exercise1 ~]# ps -aux | grep /opt/a.txt    #查看状态 T表示停止状态
root       1605  0.0  0.5 149536  5100 pts/0    T    16:22   0:00 vim /opt/a.txt
root       1674  0.0  0.1 112824  1004 pts/1    S+   16:28   0:00 grep --color=auto /opt/a.txt
```



注：

ctrl -c 是发送 SIGINT 信号，终止一个进程

ctrl -z 是发送 SIGSTOP 信号，挂起一个进程。将作业放置到后台(暂停)

例 2： D 不可中断状态

```
[root@exercise1 ~]# tar -zcvf /opt/usr-tar.gz /usr/
```

#然后在另一个终端不断查看状态，由S+， R+变为 D+

```
[root@exercise1 ~]# ps -aux | grep tar
root       1836  5.5  0.1 123672  1364 pts/0    S+   16:50   0:00 tar -z
cvf /opt/usr-tar.gz /usr/
root       1839  0.0  0.0 112824   988 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 13.0  0.1 123672  1444 pts/0    R+   16:50   0:00 tar -z
cvf /opt/usr-tar.gz /usr/
root       1842  0.0  0.0 112824   988 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 13.3  0.1 123672  1444 pts/0    R+   16:50   0:00 tar -z
cvf /opt/usr-tar.gz /usr/
root       1844  0.0  0.0 112824   988 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 14.1  0.1 123672  1444 pts/0    S+   16:50   0:00 tar -z
cvf /opt/usr-tar.gz /usr/
root       1846  0.0  0.0 112824   988 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 12.8  0.1 123672  1444 pts/0    S+   16:50   0:00 tar -z
cvf /opt/usr-tar.gz /usr/
root       1848  0.0  0.0 112824   984 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 13.0  0.1 123672  1444 pts/0    S+   16:50   0:00 tar -z
cvf /opt/usr-tar.gz /usr/
root       1850  0.0  0.0 112824   988 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 13.5  0.1 123672  1444 pts/0    S+   16:50   0:00 tar -z
cvf /opt/usr-tar.gz /usr/
root       1852  0.0  0.0 112824   988 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 12.5  0.1 123672  1444 pts/0    S+   16:50   0:01 tar -z
cvf /opt/usr-tar.gz /usr/
root       1854  0.0  0.0 112824   984 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 12.7  0.1 123672  1444 pts/0    D+   16:50   0:01 tar -z
cvf /opt/usr-tar.gz /usr/
root       1856  0.0  0.0 112824   988 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]# ps -aux | grep tar
root       1836 13.3  0.1 123672  1444 pts/0    R+   16:50   0:01 tar -z
cvf /opt/usr-tar.gz /usr/
[root@exercise1 ~]# ps -aux | grep tar
root       1836 13.6  0.1 123672  1444 pts/0    S+   16:50   0:01 tar -z
cvf /opt/usr-tar.gz /usr/
root       1860  0.0  0.0 112824   988 pts/1    R+   16:50   0:00 grep -
-color=auto tar
[root@exercise1 ~]#
```

例3：什么是僵尸进程，如何解决僵尸进程

僵尸进程：当你运行一个程序时，它会产生一个父进程以及很多子进程。 所有这些子进程都会消耗内核分配给它们的内存和 CPU 资源。这些子进程完成执行后会发送一个 Exit 信号然后死掉。这个 Exit 信号需要被父进程所读取。父进程需要随后调用 wait 命令来读取子进程的退出状态，并将子进程从进程表中移除。
若父进程正确读取了子进程的 Exit 信号，则子进程会从进程表中删掉。
但若父进程未能读取到子进程的 Exit 信号，则这个子进程虽然完成执行处于死亡的状态，但也不会从进程表中删掉。

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。
孤儿进程将被 init 进程（CentOS7 中是 systemd 进程，进程号为 1）所收养，并由 init （systemd） 进程对它们完成状态收集工作 。

僵尸进程的出现，追其**根本原因**，是其**父进程出现了问题，在子进程 exit() 后没有回收子进程的资源，而不是 Linux 系统的问题**；
此时运行的程序代码逻辑应该是有问题的，需要整改，如果出现僵尸进程，可以通过以下**两种方法**解决

 # 方法一，传递信号给其父进程，命令其回收子进程的资源
kill -SIGCHLD + 父进程号
#17) SIGCHLD 子进程结束时, 父进程会收到这个信号。

 # 方法二，直接 KILL 掉其父进程，将此进程变成孤儿进程，交给 init 进程管理，init 进程回收此进程的资源
kill -9 + 父进程号

 # 方法三，重启机器

3.2、ps 常用的参数： ps -ef

-e 显示所有进程
-f 显示完整格式输出
我们常用的组合： ps -ef

```
[root@panda ~]# ps -ef|head
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 Oct23 ?        00:01:47 /usr/lib/systemd/systemd --switc
hed-root --system --deserialize 21
root           2       0  0 Oct23 ?        00:00:00 [kthreadd]
root           3       2  0 Oct23 ?        00:00:06 [ksoftirqd/0]
root           5       2  0 Oct23 ?        00:00:00 [kworker/0:0H]
root           7       2  0 Oct23 ?        00:00:01 [migration/0]
root           8       2  0 Oct23 ?        00:00:00 [rcu_bh]
root           9       2  0 Oct23 ?        00:01:23 [rcu_sched]
root          10       2  0 Oct23 ?        00:00:05 [watchdog/0]
root          11       2  0 Oct23 ?        00:00:04 [watchdog/1]
```

包含的信息如下
UID: 启动这些进程的用户
PID: 子进程的ID
PPID: 父进程号
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

例 1： 测试 CPU 使用时间

```
[root@exercise1 ~]# dd if=/dev/zero of=/opt/a.txt count=10 bs=100M
记录了10+0 的读入
记录了10+0 的写出
1048576000字节(1.0 GB)已复制，4.15703 秒，252 MB/秒
```

```
#然后在另一个终端不断查看状态
[root@exercise1 ~]# ps -aux | grep dd
root           2  0.0  0.0      0     0 ?        S    22:21   0:00 [kthreadd]
root          41  0.0  0.0      0     0 ?        S<   22:21   0:00 [ipv6_addrconf]
dbus         522  0.0  0.1  32824  1864 ?        Ssl  22:21   0:00 /bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
root        1063 85.0 10.2 210508 102884 pts/0   R+   22:23   0:00 dd if=/dev/zero of=/opt/a.txt count=10 bs=100M
root        1065  0.0  0.0 112828   988 pts/1    R+   22:23   0:00 grep --color=auto dd
[root@exercise1 ~]#
```

注:
ps -aux 是用 BSD 的格式来显示进程。
ps -ef 是用标准的格式显示进程

4.uptime 查看 CPU 负载工具

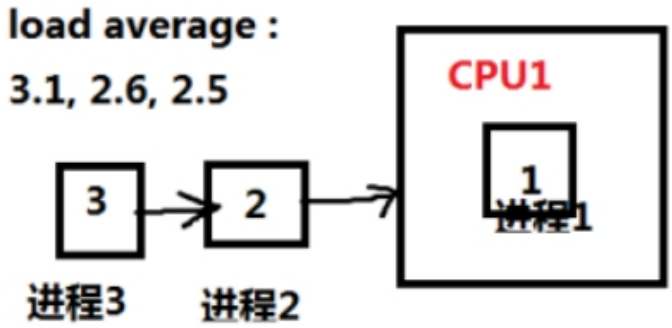
```
[root@localhost ~]# uptime
13:22:30 up 20days,  2 users,  load average: 0.06, 0.60, 0.48
```

弹出消息含意如下：

值	意义
13:22:30	当前时间
up 20days	系统运行时间，说明此服务器连续运行 20 天了
2 user	当前登陆用户数
load average: 0.06, 0.60, 0.48	系统负载，即任务队列的平均长度。 三个数值分别为 1 分钟、5 分钟、15 分钟前到现在的平均值。

任务队列的平均长度是什么?
大厅排除买票：

这时队列是 4:
cpu队列数为 3 时，如图：



互动：例 1：找出当前系统中，CPU 负载过高的服务器?

服务器 1： load average: 0.15， 0.08， 0.01 1 核

服务器 2： load average: 4.15， 6.08， 6.01 1 核

服务器 3： load average: 10.15， 10.08， 10.01 4 核

答案：服务器 2

如果服务器的 CPU 为 1 核心，则 load average 中的数字 >=3 负载过高，如果服务器的 CPU 为 4 核心，

则 load average 中的数字 >=12 负载过高。

经验：单核心，1 分钟的系统平均负载不要超过 3 ，就可以，这是个经验值。

如下图： 1 人只能买 1 张票，排第四的人可能会急。 所以我们认为超过 3 就升级CPU

5.top 命令

```
1 [root@exercise1 ~]# 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

进程和 CPU 的信息(第二、三行)

```
top - 14:42:12 up  5:18,  2 users,  load average: 0.00, 0.01, 0.05
Tasks:  89 total,   1 running,  88 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2031888 total,   405668 free,   130476 used,  1495744 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1678888 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
  973 root      20   0  145752   5280   3964 S  0.3  0.3   0:02.59 sshd
    1 root      20   0  128212   6824   4060 S  0.0  0.3   0:02.93 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.22 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
```

当有多个CPU时，这些内容可能会超过两行。内容如下：

值	意义
Tasks: 89 total	进程总数
1 running	正在运行的进程数
88 sleeping	睡眠的进程数
0 stopped	停止的进程数
0 zombie	僵尸进程数
Cpu(s): 0.0% us	系统用户进程使用CPU 百分比
0.0% 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
98.7% id	空闲 CPU 百分比
0.0% wa	cpu 等待 I/0 完成的时间总量。 测试：终端 1 ： 执行：top ;终端 2：dd if=/dev/zero of=/a.txt count=10 bs=100M ;终端 3：dd if=/dev/zero of=/a.txt count=10 bs=100M
0.0% hi （了解）硬中断消耗时间	硬中断，占的 CPU 百分比。1. 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。1. 通常，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丐可以稍后处理。 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

内存信息(第四五行)

```
KiB Mem :  2033552 total,  1376636 free,   340392 used,   316524 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1518140 avail Mem
```

内容如下：

Mem: 2033552k total　　物理内存总量
340392k used　　使用的物理内存总量
1376636k free　　空闲内存总量
316524k buff/cache　　用作内核缓存的内存量。 和free -k 一个意思

Swap: 2017948k total　　交换区总量
0k used　　使用的交换区总量
192772k free　　空闲交换区总量
1518148 avail Mem　　总的可利用内存是多少
注：
1.如果 swap 分区被使用，那么你的内存不够用了。
2.buff（buffers）：用于存放要输入到disk（块存储）的数据，； ==>可以理解成微博草稿(放在缓冲区中)
cache（cached）：存放从disk上读出的数据； ==>可以理解成正式发布微博

清理缓存：

```
echo 1 > /proc/sys/vm/drop_caches
 echo 2 > /proc/sys/vm/drop_caches
 echo 3 > /proc/sys/vm/drop_caches
```

drop_caches的值可以是0-3之间的数字，代表不同的含义：
0：不释放（系统默认值）
1：释放页面缓存
2：释放目录项和inodes
3：释放所有缓存

第 7 行进程信息

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
```

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），用户不能调整

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虚拟内存，是进程正在使用的所有内存（ps 中标为 VSZ）
（virtual　memory usage）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是进程所使用的物理内存。实际实用内存（ps 中标为 RSS）
（resident　memory usage）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VIR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

参数：

```
-b :    以批量的方式执行top，通常搭配数据流重定向将结果输出为文件

-n :    与-b搭配，意义是需要执行几次top的输出结果

-p  :   进程 IP ，查看某个进程状态
```

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。

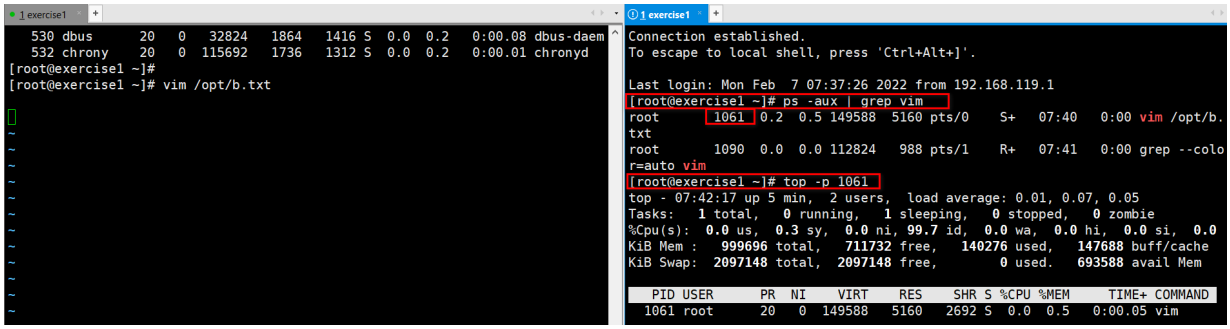
进入到top里面操作

```
q  :  退出

P  : 按 CPU 排序

M  : 按内存排序

T  :  按时间排序

k  : 给予PID一个信号

r  : 给予PID重新制订一个nice值

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
```

例 1： 运行 top ， 依次演示一下 top 的快捷键，让大家看一下效果

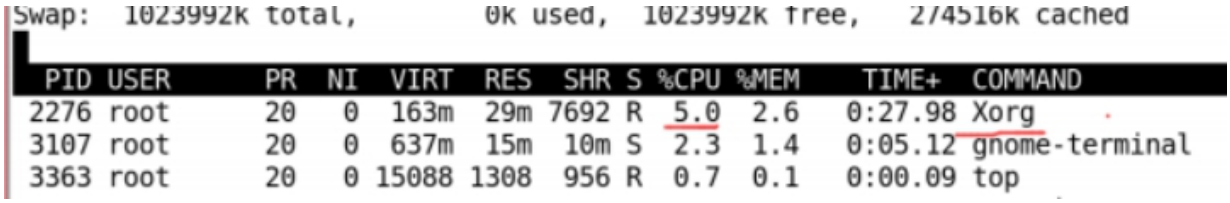
例 2： 使用TOP动态只查看某个或某些进程的信息

```
[root@exercise1 ~]# vim /opt/b.txt

在另外一个终端找到进程 ID
[root@exercise1 ~]# ps -aux | grep vim
root       1061  0.2  0.5 149588  5160 pts/0    S+   07:40   0:00 vim /opt/b.txt
root       1090  0.0  0.0 112824   988 pts/1    R+   07:41   0:00 grep --color=auto vim

[root@exercise1 ~]# top -p 1061
```



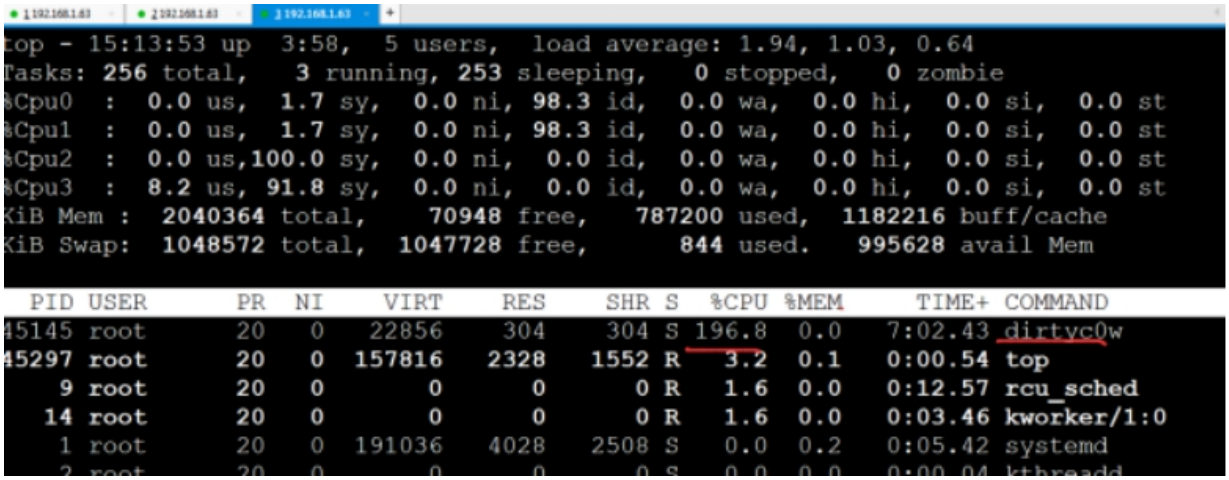
实战 1： 找出系统中使用CPU 最多的进程

解决方案：运行 top ， 找出使用 CPU 最多的进程 ，按大写的 P ，可以按 CPU 使用率来排序显示



互动：在 linux 系统中一个进程，最多可以使用 100%cpu 对吗?

答：如下图，可以看到dirtycow （脏牛漏洞，用于提权） 进程使用 196.8%



这是我第一次看见，因为这台虚拟机在一开始给配置的时候就是1核

如果你的 4 核心的cpu ， 你可以运行 400%

6.lsof 命令

lsof 命令用于查看你进程打开的文件，打开文件的进程，进程打开的端口(TCP、UDP)

-i<条件>： 列出符合条件的进程。 （4、6、协议、:端口、 @ip ）

-p<进程号>： 列出指定进程号所打开的文件;

```
#要先安装才可以使用
[root@exercise1 ~]# yum install -y lsof.x86_64
```

[root@exercise1 ~]# vim /opt/b.txt

在另外一个终端找到进程 ID

[root@exercise1 ~]# ps -aux | grep b.txt

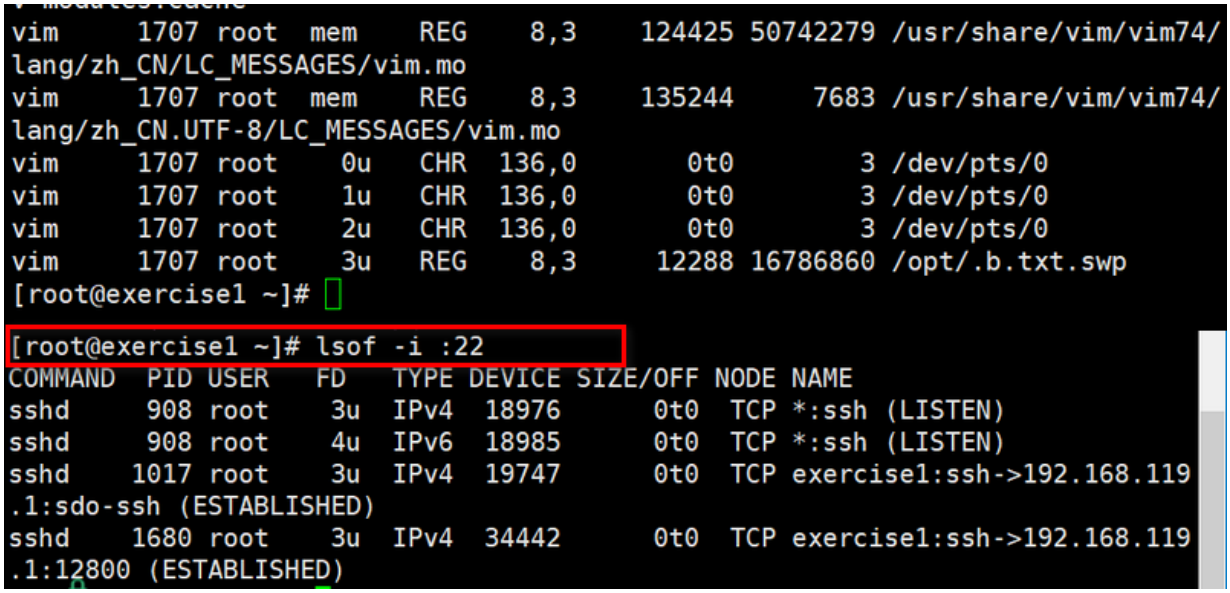
root 1707 0.0 0.5 149548 5120 pts/0 S+ 08:42 0:00 vim /opt/b.txt

root 1735 0.0 0.1 112824 1004 pts/1 S+ 08:45 0:00 grep --color=auto b.txt

[root@exercise1 ~]# lsof -p 1707 #一般用于查看木马进程，在读哪些文件

[root@exercise1 ~]# 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

```
[root@exercise1 ~]# ps -aux | grep b.txt
root       1707  0.0  0.5 149548  5120 pts/0    S+   08:42   0:00 vim /opt/b.
txt
root       1735  0.0  0.1 112824  1004 pts/1    S+   08:45   0:00 grep --colo
r=auto b.txt
[root@exercise1 ~]# lsof -p 1707
COMMAND  PID USER   FD   TYPE DEVICE  SIZE/OFF     NODE NAME
vim     1707 root  cwd    DIR    8,3       245 33574977 /root
vim     1707 root  rtd    DIR    8,3       224       64 /
vim     1707 root  txt    REG    8,3   2337208 50742303 /usr/bin/vim
vim     1707 root  mem    REG    8,3     61560     6113 /usr/lib64/libnss_fil
es-2.17.so
vim     1707 root  mem    REG    8,3 106172832 17194558 /usr/lib/locale/local
e-archive
vim     1707 root  mem    REG    8,3     11384    32386 /usr/lib64/libfreebl3
.so
vim     1707 root  mem    REG    8,3     14424     6129 /usr/lib64/libutil-2.
17.so
vim     1707 root  mem    REG    8,3     40600     6099 /usr/lib64/libcrypt-2
.17.so
vim     1707 root  mem    REG    8,3    115816     6105 /usr/lib64/libnsl-2.1
7.so
vim     1707 root  mem    REG    8,3    109976     6123 /usr/lib64/libresolv-
2.17.so
vim     1707 root  mem    REG    8,3     19888    71238 /usr/lib64/libattr.so
.1.1.0
vim     1707 root  mem    REG    8,3    402384    59971 /usr/lib64/libpcre.so
.1.2.0
vim     1707 root  mem    REG    8,3   2156592     6095 /usr/lib64/libc-2.17.
so
vim     1707 root  mem    REG    8,3    142144     6121 /usr/lib64/libpthread
-2.17.so
vim     1707 root  mem    REG    8,3   1647320 33849167 /usr/lib64/perl5/CORE
/libperl.so
vim     1707 root  mem    REG    8,3     19248     6101 /usr/lib64/libdl-2.17
.so
vim     1707 root  mem    REG    8,3     27752     7464 /usr/lib64/libgpm.so.
2.1.0
vim     1707 root  mem    REG    8,3     37056    71240 /usr/lib64/libacl.so.
1.1.0
vim     1707 root  mem    REG    8,3    174520    59910 /usr/lib64/libtinfo.s
o.5.9
vim     1707 root  mem    REG    8,3    155744     6137 /usr/lib64/libselinux
.so.1
vim     1707 root  mem    REG    8,3   1136944     6103 /usr/lib64/libm-2.17.
so
vim     1707 root  mem    REG    8,3    163312     6088 /usr/lib64/ld-2.17.so
vim     1707 root  mem    REG    8,3     26970 33661277 /usr/lib64/gconv/gcon
v-modules.cache
```



7.fuser：借由文件（或目录）找出正在使用该文件的进程

语法：fuser 参数 文件/目录

参数：

```
-u：除了进程的PID之外，还列出进程的拥有者

-m: 可以列出该文件的文件系统最顶层，一般用于umount

-v: 可以列出每个文件与进程和命令的完整相关性

-k:找出该文件或目录的PID，并KILL掉该PID

-i: 必须与-k配合，在删除前会询问
```

```
#要先安装才可以使用
[root@exercise1 ~]# yum -y install psmisc.x86_64
```

```
[root@exercise1 ~]# fuser -umv /proc
                     用户    进程号 权限   命令
/proc:               root     kernel mount (root)/proc
                     root       1 f.... (root)systemd
                     root     358 f.... (root)systemd-journal
```

访问类型如下：
c 代表当前目录
e 将此文件作为程序的可执行对象使用
f 打开的文件。默认不显示。
F 打开的文件，用于写操作。默认不显示。
r 根目录。
m 映射文件或者共享库。
s 将此文件作为共享库（或其他可装载对象）使用

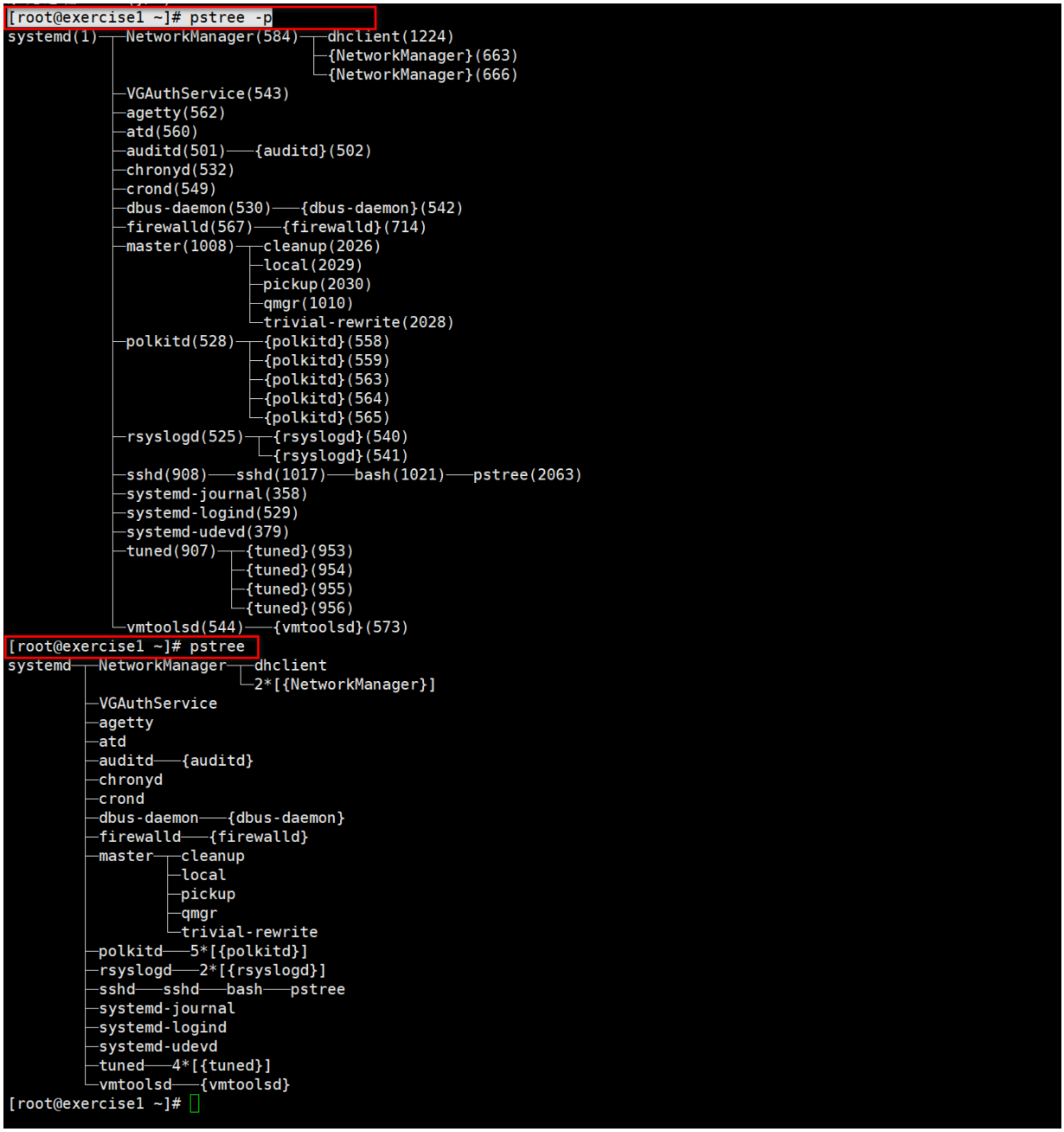
[root@exercise1 ~]# fuser -mki /home
/home: 1rce 2rc 3rc 5rc 6rc 7rc 8rc 9rc 10rc 13rc
14rc 15rc 16rc 17rc 18rc 19rc 25rc 26rc 27rc 28rc 36rc 38rc
39rc 41rc 60rc 92rc 228rc 229rc 231rc 241rc 242rc 243rc 247rc 248rc
250rc 252rc 274rc 275rc 276rc 277rc 278rc 279rc 280rc 281rc 282rc
283rc 284rc 358rce 379rce 399rc 419rc 421rc 423rc 424rc 426rc 428rc
429rc 431rc 432rc 501rce 525rce 528rce 529rce 530rce 532rce 543rce
544rce 549rce 560rce 562rce 567rce 584rce 907rce 908rce 1008rce
1010rce 1017rce 1021rce 1224rce 1677rc 1808rc 1964rc 2020rc 2026rce
2028rce 2029rce 2030rce
杀死进程 1？(y/N)

7.pstree 工具使用

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

```
[root@exercise1 ~]# pstree -p
```



8.前后台进程切换

Linux 后台进程不前台进程的区别

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，它不需要终端交互;
Linux 的大多数服务器就是用守护进程实现的。比如，sshd远程服务 等。

进程的前台与后台运行
跟系统任务相关的几个命令:

&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 .(相当于挂起)

jobs查看当前有多少在后台运行的进程 . 它是一个作业控制命令
参数:
-l 除了列出job number 与命令串之外，同时列出PID的号码
-r 仅列出正在后台run的任务
-s 仅列出正在后台stop的任务

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 %job-number 将选中的命令调出，%job-number 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 %job-number 将选中的命令调出，%job-number 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

例： vim /opt/b.txt按下： ctrl+z

```
[root@exercise1 ~]# tar -zcvf /opt/usr-tar.gz /usr/    #打开后，然后执行ctrl+z

[1]+  已停止               tar -zcvf /opt/usr-tar.gz /usr/
```

[root@exercise1 ~]# jobs #查看当前有多少在后台运行的进程
[1]+ 已停止 vim /opt/b.txt
[root@exercise1 ~]# bg 1 #继续执行

[root@exercise1 ~]# fg 1 #将后台挂起的进程恢复到前台运行

实战：脱机执行任务
nohup

语法:
nohup 命令 =>在终端前台执行任务
nohup 命令 & =>在终端后台执行任务

```
[root@exercise1 ~]# vim /opt/sleep.sh
#!/bin/bash
sleep 50s
echo "I have sleep 50 seconds"

[root@exercise1 ~]# chmod a+x /opt/sleep.sh    #添加权限
```

```
[root@exercise1 ~]# nohup /opt/sleep.sh &
[1] 2285
[root@exercise1 ~]# nohup: 忽略输入并把输出追加到"nohup.out"
```

9.kill 关闭进程

关闭进程 3 个命令：kill　killall　pkill

kill 关闭进程：kill 进程号　　# 关闭单个进程

killall 和 pkill 命令用于杀死指定名字的进程 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

```
[root@panda ~]# kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

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

例 1：　kill 和 killall 终止进程

```
[root@exercise1 ~]# kill -9 2342
[root@exercise1 ~]# killall sshd
[root@exercise1 ~]# pkill sshd
```

10.进程的优先级管理

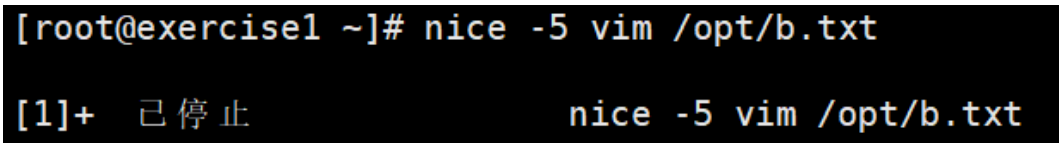
优先级取值范围为（-20,19），越小优先级越高, 默认优先级是 0

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

```
[root@exercise1 ~]# nice --5 vim /opt/b.txt          #如果指定优先级为11，写成-11表示
输入内容，然后ctrl+z 挂起
```

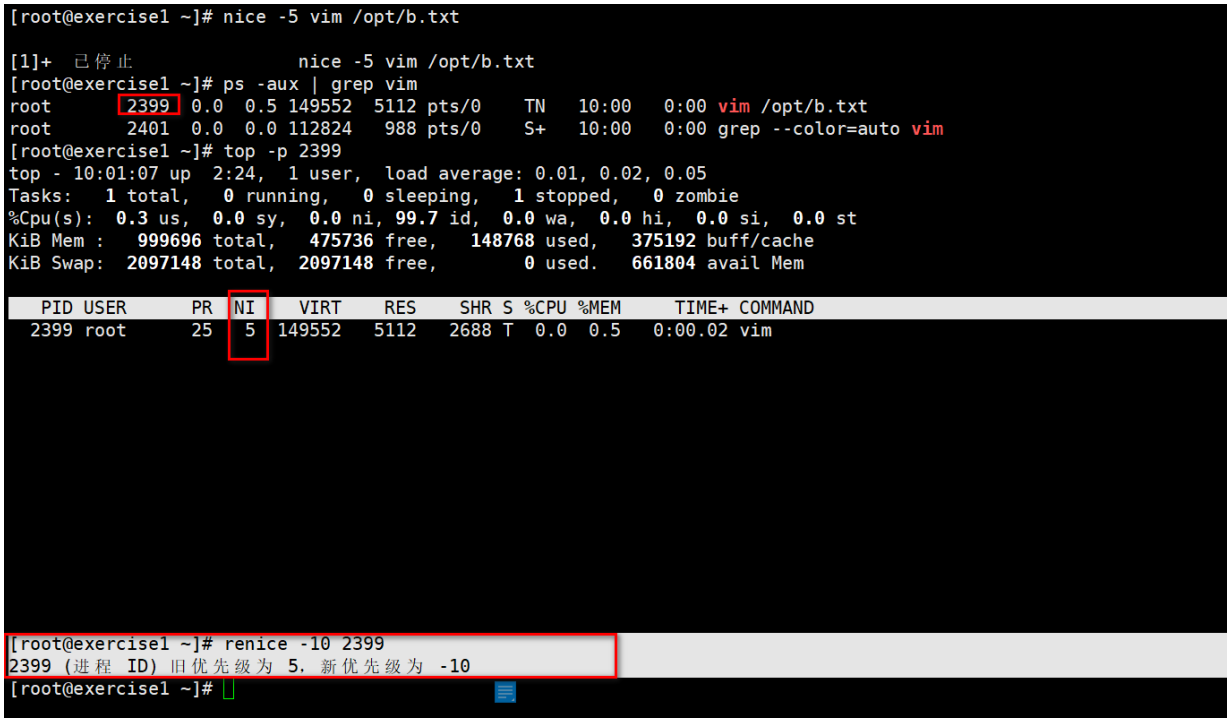



```
通过 ps 查看这个文件的 PID 号
[root@exercise1 ~]# ps -aux | grep vim
root       2399  0.0  0.5 149552  5112 pts/0    TN   10:00   0:00 vim /opt/b.txt
root       2401  0.0  0.0 112824   988 pts/0    S+   10:00   0:00 grep --color=auto vim
```

通过 top 命令查看优先级,并修改优先级
[root@exercise1 ~]# top -p 2399

[root@exercise1 ~]# renice -10 2399

2399 (进程 ID) 旧优先级为 5，新优先级为 -10



11.查看Linux 内核版本命令

1.uname:查看系统与内核相关信息
-a:列出所有系统相关的信息

cat /proc/version#查看内核版本

```
[root@exercise1 ~]# cat /proc/version
Linux version 3.10.0-693.el7.x86_64
(builder@kbuilder.dev.centos.org) (gcc version 4.8.5 20150623 (Red
Hat 4.8.5-16) (GCC) ) #1 SMP Tue Aug 22 21:09:27 UTC 2017
```

cat /etc/redhat-release#查看系统版本

```
[root@exercise1 ~]# cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
[root@exercise1 ~]#
```

12.dmesg:分析内核产生的信息 (了解)

dmesg（display message）命令用于显示开机信息
kernel 会将开机信息存储在 ring buffer 中。若是开机时来不及查看信息，可利用 dmesg 来查看。开机信息亦保存在 /var/log 目录中，名称为 dmesg 的文件里。

语法: dmesg [-cn][-s <缓冲区大小>]
参数

```
-c    显示信息后，清除 ring buffer 中的内容

-s<缓冲区大小>    预设置为 8196，刚好等于 ring buffer 的大小

-n    设置记录信息的层级
```

实例

```
[root@exercise1 ~]# dmesg    #查看开机信息
```

[root@exercise1 ~]# dmesg -c #清除开机信息，但/var/log/dmesg文件中仍然有这些信息

[root@exercise1 ~]# dmesg > boot.msg #将开机信息保存到 boot.msg文件中
[root@exercise1 ~]# ls #显示当前目录文件
boot.msg

#使用如'more'。 'tail', 'less '或者'grep'文字处理工具来处理'dmesg'命令的输出。由于dmesg日志的输出不适合
#在一页中完全显示，因此我们使用管道（pipe）将其输出送到more或者less命令单页显示

#列出加载到内核中的所有驱动
[root@exercise1 ~]# dmesg | more

或

[root@exercise1 ~]# dmesg | less

#列出所有被检测到的硬件
#要显示所有被内核检测到的硬盘设备，可以使用'grep'命令搜索'sda'关键词
[root@exercise1 ~]# dmesg | grep sda
注解 'sda'表示第一块 SATA硬盘，'sdb'表示第二块SATA硬盘。若想查看IDE硬盘搜索'hda'或'hdb'关键词

13.vmstat：检测系统资源变化

语法：vmstat [-aSdp]

参数

```
-a ： 显示cpu/内存等信息

-S ： 单位：让显示的数据有单位

-d ： 显示磁盘的读写总量

-p ：分区 ：  可显示该分区的读写总量
```

例：统计目前主机状态，每秒1次，共5次

```
[root@exercise1 ~]# vmstat 1 5
procs -----------memory---------- ---swap-- -----io----   -system--
------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs
us sy id wa st
 2  0      0 469840   2100 379340    0    0    22    15   74   87
0  0 100  0  0
 0  0      0 469840   2100 379372    0    0     0     0   77   76
0  0 100  0  0
 0  0      0 469840   2100 379372    0    0     0     0   85   81
0  1 99  0  0
 0  0      0 469840   2100 379372    0    0     0     0   75   72
0  0 100  0  0
 0  0      0 469840   2100 379372    0    0     0     0   83   84
0  0 100  0  0
[root@exercise1 ~]#
```

r等待运行中的进程数量
b不可被唤醒的进程数量
这两项数量大，说明cpu使用高

si磁盘中将进程取出的容量
so由于内存不足而将没用到的进程写入到磁盘的虚拟分区的容量
这两项数值大，代表物理内存不足，需要频繁用到虚拟内存，导致系统性能差

bi磁盘读入的区块数量
bo写入磁盘的区块数量
这两项数值大，代表磁盘读写频繁（下载，跑数据库）

in每秒被中断的进程次数
cs每秒被执行的事件切换次数
这两项数值大，代表服务端口，网卡被频繁访问（服务被频繁使用）

14.iostat:磁盘性能查询命令

语法：iostat 参数 磁盘

参数

```
-c：仅显示CPU使用情况

-d：仅显示设备利用率

-k：显示状态以kb每秒为单位，而不使用块每秒

-m：显示状态以mb每秒为单位

-p：仅显示块设备和所有被使用的其他分区的状态

-t：显示每个报告产生时的时间

-x：显示扩展状态
```

例：统计目前磁盘状态，每秒1次，共3次

```
[root@exercise1 ~]# yum -y install sysstat   #需要安装使用

[root@exercise1 ~]# iostat 1 3
Linux 3.10.0-693.el7.x86_64 (exercise1)     2022年02月07日
_x86_64_    (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
               0.09    0.00    0.21    0.17    0.00   99.53
```

```
Device:            tps   kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               1.31      21.37      14.95     234932     164351
scd0              0.00       0.10       0.00       1046          0
```

```
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00  100.00

```

```
Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.00         0.00         0.00          0          0
scd0              0.00         0.00         0.00          0          0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00  100.00

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.00         0.00         0.00          0          0
scd0              0.00         0.00         0.00          0          0

[root@exercise1 ~]#
```

[root@exercise1 ~]# iostat -kx /dev/sda
Linux 3.10.0-693.el7.x86_64 (exercise1) 2022年02月07日 *x86_64* (1 CPU)

avg-cpu: %user   %nice %system %iowait %steal   %idle
          0.09   0.00   0.21   0.17   0.00   99.53

Device: rrqm/s   wrqm/s   r/s   w/s   rkB/s   wkB/s avgrq-sz avgqu-sz   await r_await w_await svctm %util
sda   0.00   0.13   0.62   0.69   21.30   14.93   55.54   0.01   8.88   13.01   5.18   2.47   0.32

[root@exercise1 ~]#

%user应用程序使用CPU的时间占比
%nice拥有高优先级的应用程序使用CPU的时间占比
%system内核程序使用CPU的时间占比
%iowait表示等待进行 I/O 所使用 CPU 的时间百分比
%steal显示虚拟机管理器在服务另一个虚拟处理器时虚拟CPU处在非自愿等待下花费时间的百分比
%idle 显示 CPU 的空闲时间
Device监测设备名称
rrqm/s每秒需要读取需求的数量

wrqm/s每秒需要写入需求的数量
r/s每秒实际读取需求的数量
w/s每秒实际写入需求的数量
rkB/s每秒实际读取的大小，单位为KB
wkB/s每秒实际写入的大小，单位为KB
avgrq-sz平均每次设备I/O操作的数据大小
avgqu-sz平均I/O队列长度
await平均每次设备I/O操作的等待时间
svctm平均每次设备I/O操作的服务时间
%util被I/O操作消耗的CPU百分比

拓展

ps -ef 是用标准的格式显示进程的、其格式如下

```
[root@5201351 ~]# ps -ef
UID         PID  PPID  C STIME TTY          TIME CMD
root          1     0  0 06:50 ?        00:00:02 /sbin/init
root          2     0  0 06:50 ?        00:00:00 [kthreadd]
root          3     2  0 06:50 ?        00:00:00 [migration/0]
root          4     2  0 06:50 ?        00:00:00 [ksoftirqd/0]
```

ps aux 是用BSD的格式来显示、其格式如下

```
[root@5201351 ~]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.1  19224  1540 ?        Ss   06:50   0:02 /sbin/init
root          2  0.0  0.0      0     0 ?        S    06:50   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    06:50   0:00 [migration/0]
root          4  0.0  0.0      0     0 ?        S    06:50   0:00 [ksoftirqd/0]
```

中断定义：指当出现需要时，CPU暂时停止当前程序的执行转而执行处理新情况的程序和执行过

硬件中断是由与系统相连的外设(比如网卡 硬盘 键盘等)自动产生的. 每个设备或设备集都有他自己的IRQ(中断请求), 基于IRQ, CPU可以将相应的请求分发到相应的硬件驱动上(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).处理中断的驱动是需要运行在CPU上的, 因此, 当中断产生时, CPU会暂时停止当前程序的程序转而执行中断请求
软中断不会直接中断CPU, 也只有当前正在运行的代码(或进程)才会产生软中断. 软中断是一种需要内核为正在运行的进程去做一些事情(通常为I/O)的请求

硬件中断和软中断的区别

- **硬件中断是由外设引发的, 软中断是执行中断指令产生的**
- **硬件中断的中断号是由中断控制器提供的, 软中断的中断号由指令直接指出, 无需使用中断控制器**
- **硬件中断是可屏蔽的, 软中断不可屏蔽**

- **硬件中断处理程序要确保它能快速地完成任务, 这样程序执行时才不会等待较长时间, 称为上半部**
- **软中断处理硬中断未完成的工作, 是一种推后执行的机制, 属于下半部**