**运维高频命令工具**

**一、网络管理工具**

**1.ip**

**1.1功能说明：**

ip命令是iproute软件包中的一个强大的网络配置工具，用于显示或管理Linux系统的路由，网络设备，策略路由和隧道

**1.2选项说明：**

| 参数选项 | 解释说明 |
| --- | --- |
| -s | 输出更详细的信息，为了显示更详细的信息，可重复使用此选项 |
| -r | 显示主机时，不使用IP地址，而是使用主机的域名 |
| 网络对象 | 指定要管理的网络对象（link：网络设备 address：IP地址 addrlabel：协议地址标签管理 route：路由表 rule：策略路由表 runnel：IP隧道 ） |
| help | ip help：查看ip命令的帮助 ip【object】help：查看指定的网络对象的帮助 |
| 操作命令 | 对指定的网络对象完成的具体操作。通常，每一个具体操作的命令后面又有一组相关的命令选项。不同的操作对象所支持的操作命令也不同。下面按照操作的网络对象给出所支持的常见操作命令 |
|  | link对象支持的操作命令：set（修改设备属性），show（显示设备属性） |
|  | address对象支持的操作命令：add（添加协议地址），del（删除协议地址），flush（清除协议地址），show（查看协议地址） |
|  | addrlabel对象支持的操作命令：add，del，list，flush |
|  | route对象支持的操作命令：add，change，replace，delete，show，flush，get |
|  | rule对象支持的操作命令：add，delete，flush，show |
|  | tunnel对象支持的操作命令：add，change，delete，prl，show |

**1.3使用说明：**

（1）显示网络设备属性

[root@KGC ~]# ip link show ens33

**显示网卡属性**

2: ens33: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP mode DEFAULT group default qlen 1000

link/ether 00:0c:29:b0:a0:0a brd ff:ff:ff:ff:ff:ff

[root@KGC ~]# ip -s link show ens33

**显示网卡详细信息**

2: ens33: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP mode DEFAULT group default qlen 1000

link/ether 00:0c:29:b0:a0:0a brd ff:ff:ff:ff:ff:ff

RX: bytes packets errors dropped overrun mcast

42003 429 0 0 0 0

TX: bytes packets errors dropped carrier collsns

37969 347 0 0 0 0

**额外显示了网卡的发送和接收数据包的数量**

[root@KGC ~]# ip -s -s link show ens33

**显示网卡更加详细的信息**

2: ens33: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP mode DEFAULT group default qlen 1000

link/ether 00:0c:29:b0:a0:0a brd ff:ff:ff:ff:ff:ff

RX: bytes packets errors dropped overrun mcast

47049 489 0 0 0 0

RX errors: length crc frame fifo missed

0 0 0 0 0

TX: bytes packets errors dropped carrier collsns

43239 403 0 0 0 0

TX errors: aborted fifo window heartbeat transns

0 0 0 0 2

（2）激活和关闭网卡

[root@KGC ~]# ip link set ens33 up

**激活ens33网卡**

[root@KGC ~]# ip link set ens33 down

**关闭ens33网卡**

1. 临时修改网卡MAC地址

[root@KGC ~]# ip link set ens33 address 0:0c:29:a8:ca:5f

**修改MAC地址**

（4）显示网卡信息

[root@KGC ~]# ip a

**效果与ip address一样，显示的结果包括激活和未激活的网卡**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: ens33: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 00:0c:29:b0:a0:0a brd ff:ff:ff:ff:ff:ff

inet 192.168.100.101/24 brd 192.168.100.255 scope global noprefixroute ens33

valid\_lft forever preferred\_lft forever

inet6 fe80::20c:29ff:feb0:a00a/64 scope link

valid\_lft forever preferred\_lft forever

3: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000

link/ether 52:54:00:e8:23:fd brd ff:ff:ff:ff:ff:ff

inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0

valid\_lft forever preferred\_lft forever

4: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc pfifo\_fast master virbr0 state DOWN group default qlen 1000

link/ether 52:54:00:e8:23:fd brd ff:ff:ff:ff:ff:ff

（5）添加删除网卡ip地址

[root@KGC ~]# ip a add 192.168.100.200/24 dev ens33

**添加ip地址**

[root@KGC ~]# ip a show ens33

2: ens33: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 00:0c:29:b0:a0:0a brd ff:ff:ff:ff:ff:ff

inet 192.168.100.101/24 brd 192.168.100.255 scope global noprefixroute ens33

valid\_lft forever preferred\_lft forever

**inet 192.168.100.200/24 scope global secondary ens33**

valid\_lft forever preferred\_lft forever

inet6 fe80::20c:29ff:feb0:a00a/64 scope link

valid\_lft forever preferred\_lft forever

**网卡ens33额外多出了一个ip地址**

[root@KGC ~]# ip a del 192.168.100.200/24 dev ens33

**删除ip地址**

[root@KGC ~]# ip a show ens33

2: ens33: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 00:0c:29:b0:a0:0a brd ff:ff:ff:ff:ff:ff

inet 192.168.100.101/24 brd 192.168.100.255 scope global noprefixroute ens33

valid\_lft forever preferred\_lft forever

inet6 fe80::20c:29ff:feb0:a00a/64 scope link

valid\_lft forever preferred\_lft forever

**注意：**

* **删除网卡的主IP地址，同时会删除该网卡的所有IP地址**
* **删除网卡的辅助IP地址，不会影响该网卡的其他IP地址**

1. 创建主网卡的别名ip地址

[root@KGC ~]# ip a add 192.168.100.200/24 dev ens33 label ens33:1

[root@KGC ~]# ip a show ens33

2: ens33: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 00:0c:29:b0:a0:0a brd ff:ff:ff:ff:ff:ff

inet 192.168.100.101/24 brd 192.168.100.255 scope global noprefixroute ens33

valid\_lft forever preferred\_lft forever

inet 192.168.100.200/24 scope global secondary ens33:1

valid\_lft forever preferred\_lft forever

inet6 fe80::20c:29ff:feb0:a00a/64 scope link

valid\_lft forever preferred\_lft forever

**二、进程管理工具**

**1.ps**

**1.1功能说明：**

用于列出执行ps命令的那个时刻的进程快照，就像用手机给进程照了一张照片。如果想要动态地显示进程，就需要使用top命令，该命令类似于把手机切换成录像模式。

**1.2选项说明：**

| **选项** | **解释说明** |
| --- | --- |
| -a | 显示所有终端下执行的进程 |
| -u | 显示指定用户相关的进程信息 |
| **-e** | **显示所有进程** |
| **-f** | **额外显示UID，PPID，C与STIME栏位** |
| -H | 显示进程树 |
| -l | 以详细的格式来显示进程的状况 |
| -o | 自定义输出指定的字段，以逗号分隔 |
| -sort key | key表示为指定字段排序，默认升序，+key升序，-key降序 |
| **a** | **显示与终端相关的所有进程，包含每个进程的完整路径** |
| **x** | **显示与终端无关的所有进程** |
| **u** | **显示进程的用户信息** |
| f | 显示进程树 |

**1.3使用说明：**

1. **ps 命令不接任何参数**

[root@KGC ~]# ps

PID TTY TIME CMD

10885 pts/0 00:00:00 bash

10930 pts/0 00:00:00 bash

10984 pts/0 00:00:00 bash

11164 pts/0 00:00:00 bash

11205 pts/0 00:00:00 ps

默认情况下，ps命令不接任何参数时，输出的是使用者当前所在终端（窗口）的进程，其输出结果中的各项说明如下。

* PID：是进程的标识号
* TTY：是进程所属的终端控制台
* TIME：列是进程所使用的总的CPU时间
* CMD：列是正在执行的命令行
* pts/0：虚拟终端（0代表打开的第一个）

1. **显示所有进程，格外显示UID，PPID，C与STIME栏位**

[root@KGC ~]# ps -ef | head -n 10

UID PID PPID C STIME TTY TIME CMD

root 2 0 0 11:04 ? 00:00:00 [kthreadd]

root 3 2 0 11:04 ? 00:00:00 [ksoftirqd/0]

root 5 2 0 11:04 ? 00:00:00 [kworker/0:0H]

root 6 2 0 11:04 ? 00:00:00 [kworker/u256:0]

root 7 2 0 11:04 ? 00:00:00 [migration/0]

root 8 2 0 11:04 ? 00:00:00 [rcu\_bh]

root 9 2 0 11:04 ? 00:00:00 [rcu\_sched]

root 10 2 0 11:04 ? 00:00:00 [lru-add-drain]

* UID：进程被该UID所拥有
* PID：进程的标识号
* PPID：进程的父进程的标识号
* C：CPU使用的资源百分比
* STIME：进程开始的时间
* TTY：该进程是在哪个终端机上面运作，若与终端机无关，则显示“？”，另外，tty1-tty6是本机上面的登入者进程，若为pts/0等，则表示为由网络连接进主机的进程。
* TIME：进程所使用的总的CPU时间
* CMD：正在执行的命令行

1. **显示cpu、内存、虚拟内存、实际内存的使用占比**

[root@KGC ~]# ps aux

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

root 1 0.0 0.2 125500 4032 ? Ss 11:04 0:01 /usr/lib/systemd/systemd --switched-root --system --des

root 2 0.0 0.0 0 0 ? S 11:04 0:00 [kthreadd]

root 3 0.0 0.0 0 0 ? S 11:04 0:00 [ksoftirqd/0]

root 5 0.0 0.0 0 0 ? S< 11:04 0:00 [kworker/0:0H]

root 6 0.0 0.0 0 0 ? S 11:04 0:00 [kworker/u256:0]

root 7 0.0 0.0 0 0 ? S 11:04 0:00 [migration/0]

root 8 0.0 0.0 0 0 ? S 11:04 0:00 [rcu\_bh]

root 9 0.0 0.0 0 0 ? R 11:04 0:00 [rcu\_sched]

root 10 0.0 0.0 0 0 ? S< 11:04 0:00 [lru-add-drain]

* USER:该进程属于的用户。
* PID：该进程的进程号。
* %CPU：该进程使用掉的CPU资源百分比。
* %MEM：该进程所占用的物理内存百分比。
* VSZ：该进程占用的虚拟内存量（单位Kbytes）
* RSS：该进程占用的实际内存量（单位Kbytes）
* TTY：该进程是在哪个终端机上面运作的，若与终端机无关，则显示“？”，另外，tty1-tty6是本机上面的登入者进程，若为pts/0等，则表示为由网络连接进主机的进程。
* **STAT**：该进程目前的状态，主要的状态包括如下几种。

R：正在运行，或者是可以运行。

S：正在终端睡眠中，可以由某些信号唤醒。

D：不可中断睡眠。

T：正在侦测或者是停止了。

Z：已经终止，但是其父进程无法正常终止它，从而变成zombie（僵尸）进程的状态

+：前台进程。

l：多线程进程。

N：低优先级进程。

<：高优先级进程。

s：进程领导者。

L：已将页面锁定到内存中。

* START：该进程被触发启动的时间
* TIME：该进程实际使用CPU运作的时间
* COMMAND：该进程的实际命令

1. **显示指定用户的进程信息**

[root@KGC ~]# ps -u root

PID TTY TIME CMD

1 ? 00:00:01 systemd

2 ? 00:00:00 kthreadd

3 ? 00:00:00 ksoftirqd/0

5 ? 00:00:00 kworker/0:0H

7 ? 00:00:00 migration/0

1. **以详细格式显示指定用户的进程状态**

[root@KGC ~]# ps -u root -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 0 1 0 0 80 0 - 31375 ep\_pol ? 00:00:01 systemd

1 S 0 2 0 0 80 0 - 0 kthrea ? 00:00:00 kthreadd

1 S 0 3 2 0 80 0 - 0 smpboo ? 00:00:00 ksoftirqd/0

1 S 0 5 2 0 60 -20 - 0 worker ? 00:00:00 kworker/0:0H

1 S 0 7 2 0 -40 - - 0 smpboo ? 00:00:00 migration/0

* F:代表这个进程的标志（flag），1 fork之后未执行的进程，4 使用超级用户权限

0代表全不是，5代表都是

* S：代表这个进程的状态（STAT），前面已经讲解过了
* UID:进程被该UID所拥有
* PID：进程的标识号
* PPID：父进程的ID
* C：CPU使用的资源百分比
* PRI：Priority（优先执行序）的缩写
* NI：Nice值
* ADDR:指出该进程在内存的哪个部分。如果是个running的进程，则一般是“-”
* SZ:使用掉的内存大小
* WCHAN:目前这个进程是否正在运作当中，若为“-”则表示正在运作
* TTY:该进程是在哪个终端机上面运作的，若与终端机无关，则显示“？”，另外，tty1-tty6是本机上面的登入者进程，若为pts/0等，则表示为由网络连接进主机的进程
* TIME：该进程实际使用CPU运作的时间
* CMD：该进程的实际命令

1. **查看某个进程在哪个CPU上运行**

[root@yuan ~]# ps -eo pid,pcpu,pmem,args

PID %CPU %MEM COMMAND

1 0.2 0.2 /usr/lib/systemd/systemd --switched-root --system --deserialize 22

2 0.0 0.0 [kthreadd]

3 0.0 0.0 [ksoftirqd/0]

4 0.0 0.0 [kworker/0:0]

5 0.0 0.0 [kworker/0:0H]

[root@yuan ~]# ps -eo pid,pcpu,pmem,args --sort=-pmem （+为正序可省略，-为倒序）

PID %CPU %MEM COMMAND

760 0.1 1.5 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid

1112 0.0 0.9 /usr/bin/python -Es /usr/sbin/tuned -l -P

1121 0.0 0.8 /usr/sbin/libvirtd

717 0.0 0.7 /usr/lib/polkit-1/polkitd --no-debug

774 0.0 0.5 /usr/sbin/NetworkManager --no-daemon

**ps命令支持排序，通过--sort，对应的查询的键名称对照如下：（通过man命令可查询）**

| **KEY** | **LONG** | **DESCRIPTION** |
| --- | --- | --- |
| **c** | **cmd** | **可执行文件的简单名称** |
| **C** | **pcpu** | **cpu利用率** |
| **g** | **pgrp** | **进程组ID** |
| **G** | **tpgid** | **控制tty进程组ID** |
| **j** | **cutime** | **累积用户时间** |
| **J** | **cstime** | **累计系统时间** |
| **k** | **utime** | **用户时间** |
| **o** | **session** | **会话ID** |
| **p** | **pid** | **进程ID** |
| **s** | **size** | **内存大小（字节为单位）** |
| **t** | **tty** | **控制tty的设备编号** |
| **U** | **uid** | **用户ID号** |
| **u** | **user** | **用户名称** |

**2.kill**

**2.1功能说明**

kill命令能够终止你希望停止的进程

**2.2选项说明**

| **选项** | **解释说明** |
| --- | --- |
| -l | 列出全部的信号名称 |
| -p | 指定kill命令只打印相关进程的进程号，而不发送任何信号 |
| **-s** | **指定要发送的信号** |

**2.3使用说明**

**（1）列出所有信号名称**

[root@KGC ~]# 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

[root@KGC ~]# kill -l kill

9

**可以使用-l参数对信号名和数字信号互换**

[root@KGC ~]# kill -l 9

KILL

**常用信号说明**

| **信号** | **说明** |
| --- | --- |
| HUP（1） | 挂起，通常因终端掉线或用户退出而引发 |
| INT（2） | 中断，通常是按下Ctrl+c组合键来发出这个信号 |
| QUIT（3） | 退出，通常是按下CTRL+\组合键来发出这个信号 |
| KILL（9） | 立即结束进程的运行 |
| TERM（15） | 终止，通常在系统关机时发送 |
| TSTP（20） | 暂停进程的运行，通常是按下Ctrl+z组合键来发出这个信号 |

1. **终止进程**

* kill指令默认使用的信号为15（TERM），用于结束进程。如果进程忽略此信号，则可以使用信号9强制终止进程。
* 一般是先通过ps等命令获取到要终止进程的进程号，然后直接使用“kill 进程号”就可以了。

[root@KGC ~]# kill xxxx（pid编号）

**kill 命令默认使用的信号为15，这种格式最常用**

[root@KGC ~]# kill -s 15 xxxx（pid编号）

**这种格式使用-s参数明确指定发送值为15的信号，效果和kill xxxx一样**

[root@KGC ~]# kill -9 xxxx（pid编号）

**信号9会强行终止进程，这会带来一些副作用，如数据丢失，或者终端无法恢复到正常状态等，因此应尽量避免使用，除非进程使用其他信号无法终止。**

1. **进程存活检测**

[root@KGC ~]# pgrep -l sshd

1112 sshd

10883 sshd

[root@KGC ~]# kill -s 0 1112

[root@KGC ~]# echo $?

0

[root@KGC ~]# kill -s 0 1111

bash: kill: (1111) - 没有那个进程

[root@KGC ~]# echo $?

1

**在kill的所有信号中，有一个十分特殊的信号值0，使用格式为kill -0 $pid。其中的-0表示不发送任何信号给$pid对应的进程，但是仍然会对$pid是否存在对应的进程进行检查**

**3.killall**

**3.1功能说明**

使用kill命令终止进程还需要先获取进程的pid进程号，这个过程有点繁琐，而使用killall命令就可以直接用“kill 进程名”这种形式终止进程。

**3.2选项说明**

| **选项** | **解释说明** |
| --- | --- |
| -I | 不区分大小写匹配 |
| -g | 终止属于该进程组的进程 |
| -i | 在终止进程之前询问是否确认 |
| -l | 列出所有已知的信号名 |
| -q | 如果没有进程终止则不提示 |
| -r | 使用正则表达式匹配要终止的进程名称 |
| -s | 用指定的信号代替默认信号 |
| **-u** | **终止指定用户的进程** |
| -v | 报告信号是否发送成功 |
| **-w** | **等待所有被终止的进程死去。killall每秒都会检查一次被终止的进程是否仍然存在，其仅在都死光后才返回。注意，如果信号被忽略，或者没有起作用，或者进程停留在僵尸状态，那么killall可能会永久等待** |

**3.3使用说明**

1. -w用法

[root@KGC ~]# killall -w crond

**由于使用-w选项，会等待几秒后命令执行结束**

[root@KGC ~]# killall -w crond

crond: no process found

（2）-u用法

[root@KGC ~]# ps -u username -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 1000 66080 66079 0 80 0 - 29054 do\_wai pts/1 00:00:00 bash

0 S 1000 66116 66080 0 80 0 - 37817 poll\_s pts/1 00:00:00 vim

[root@KGC ~]# killall -u username vim

[root@KGC ~]# ps -u username -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 1000 66080 66079 0 80 0 - 29074 n\_tty\_ pts/1 00:00:00 bash

**我们在日常工作中，有些服务是通过指定某一个特定的普通用户进行配置部署，由于企业针对系统账号都有自己的权限集权分配策略，有些服务是需要使用普通用户进行启动的，所以我们可以使用此命令来终止对应用户下启动的所有进程**

1. **pkill**

**4.1功能说明：**

pkill命令可通过进程名终止指定的进程。使用killall终止进程需要连续执行几次，而pkill可以杀死指定进程及其所有子进程。

**4.2 选项说明：**

| **选项** | **解释说明** |
| --- | --- |
| -t终端 | 杀死指定终端的进程 |
| -u用户 | 杀死指定用户的进程 |

**4.3使用说明：**

（1）通过进程名终止进程

[root@KGC ~]# /etc/init.d/crond status

**查看定时任务程序运行状态**

crond (pid 1274) is running...

[root@KGC ~]# pkill crond

**终止定时任务进程**

[root@KGC ~]# /etc/init.d/crond status

crond dead but subsys locked

**进程被终止**

（2）通过终端名终止进程

[root@KGC ~]# w

**第二列TTY就是用户运行的终端**

15:57:09 up 1:05, 3 users, load average: 0.00, 0.00, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

yunjisua tty1 - 15:55 12.00s 0.02s 0.01s vim ttt

root pts/0 192.168.200.1 14:51 0.00s 0.07s 0.00s w

root pts/1 192.168.200.1 15:19 3:28 0.01s 0.00s bash

[root@KGC ~]# ps -u yunjisuan -l

**查看用户的进程**

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 500 1175 1174 0 80 0 - 27076 wait pts/1 00:00:00 bash

0 S 500 1210 1175 0 80 0 - 27076 n\_tty\_ pts/1 00:00:00 bash

4 S 500 1333 1322 0 80 0 - 27075 wait tty1 00:00:00 bash

0 S 500 1359 1333 0 80 0 - 35890 poll\_s tty1 00:00:00 vim

[root@KGC ~]# pkill -t tty1

**杀掉终端正在运行的进程**

[root@KGC ~]# ps -u yunjisuan -l

**vim进程没了**

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 500 1175 1174 0 80 0 - 27076 wait pts/1 00:00:00 bash

0 S 500 1210 1175 0 80 0 - 27076 n\_tty\_ pts/1 00:00:00 bash

4 S 500 1333 1322 0 80 0 - 27075 n\_tty\_ tty1 00:00:00 bash

[root@KGC ~]# pkill -9 -t tty1

**强行杀掉tty1终端（强踢登录用户）**

[root@KGC ~]# ps -u yunjisuan -l

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 500 1175 1174 0 80 0 - 27076 wait pts/1 00:00:00 bash

0 S 500 1210 1175 0 80 0 - 27076 n\_tty\_ pts/1 00:00:00 bash

[root@KGC ~]# w

**终端没了**

15:58:17 up 1:06, 2 users, load average: 0.00, 0.00, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

root pts/0 192.168.200.1 14:51 0.00s 0.08s 0.00s w

root pts/1 192.168.200.1 15:19 4:36 0.01s 0.00s bash

（3）通过用户名终止进程

[root@KGC ~]# w

16:08:24 up 1:16, 3 users, load average: 0.00, 0.00, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

yunjisua tty1 - 16:01 4.00s 0.03s 0.02s -bash

root pts/0 192.168.200.1 14:51 0.00s 0.09s 0.00s w

root pts/1 192.168.200.1 15:19 14:43 0.01s 0.00s bash

[root@KGC ~]# ps -u yunjisuan -l

**查看用户的进程信息**

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 500 1175 1174 0 80 0 - 27076 wait pts/1 00:00:00 bash

0 S 500 1210 1175 0 80 0 - 27076 n\_tty\_ pts/1 00:00:00 bash

4 S 500 1387 1366 0 80 0 - 27076 n\_tty\_ tty1 00:00:00 bash

0 T 500 1430 1387 0 80 0 - 35883 signal tty1 00:00:00 vim

[root@KGC ~]# pkill -u yunjisuan

**杀掉指定用户所有进程**

[root@KGC ~]# ps -u yunjisuan -l

**成功**

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

4 S 500 1175 1174 0 80 0 - 27076 wait pts/1 00:00:00 bash

0 S 500 1210 1175 0 80 0 - 27076 n\_tty\_ pts/1 00:00:00 bash

4 S 500 1387 1366 0 80 0 - 27076 n\_tty\_ tty1 00:00:00 bash

**5.top（实时显示）**

**5.1功能说明**

top命令用于实时地对系统处理器状态进行监控，它能够实时地显示系统中各个进程的资源占用状况。该命令可以按照CPU的使用，内存的使用和执行时间对系统任务进程进行排序显示，同时top命令还可以通过交互式命令进行设定显示。

**5.2选项说明**

| **选项** | **解释说明** |
| --- | --- |
| -a | 将进程按照使用内存排序 |
| -b | 以批处理的模式显示进程信息，输出结果可以传递给其他程序或写入到文件中。在这种模式下，top命令不会接受任何输入，一直运行直到达到-n选项设置的阈值，或者按Ctrl+C等组合键终止程序 |
| -c | 显示进程的整个命令路径，而不是只显示命令名称 |
| -d | 指定每两次屏幕信息刷新之间的时间间隔 |
| -H | 指定这个可以显示每个线程的情况，否则就是进程的总的状态 |
| -i | 不显示闲置或者僵死的进程信息 |
| -n | top输出信息更新的次数，完成后将退出top命令 |
| -p | 显示指定的进程信息 |

**5.3使用说明**

1. **不加选项**

[root@KGC ~]# top

top - 16:40:31 up 1:48, 3 users, load average: 0.00, 0.00, 0.00

Tasks: 77 total, 1 running, 76 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

Mem: 1004412k total, 152112k used, 852300k free, 11312k buffers

Swap: 2031608k total, 0k used, 2031608k free, 42304k cached

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

1 root 20 0 19232 1536 1256 S 0.0 0.2 0:00.76 init

2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd

3 root RT 0 0 0 0 S 0.0 0.0 0:00.00 migration/0

4 root 20 0 0 0 0 S 0.0 0.0 0:00.01 ksoftirqd/0

**显示说明：**

**top-**

* **16:40:31** 当前系统时间
* **up 1:48** 系统已经运行了1小时48分
* **3 users** 当前有3个用户登录系统
* **load average:0.00, 0.00, 0.00** load average后面三个数分别是1分钟、5分钟、15分钟的平均负载情况

**Tasks:**

* **77total,** 77个进程
* **1 running,** 一个进程处于运行状态
* **76 sleeping,** 76个进程在休眠中stoped
* **0 stopped,** 0个进程已经停止
* **0 zombie,** 0个僵尸进程

**Cpu(s):**

* **0.0%us** 用户空间占用CPU的百分比
* **0.0%sy** 内核空间占用CPU的百分比
* **0.0%ni** 改变过优先级的进程占用CPU的百分比
* **100.0%id** 空闲CPU百分比
* **0.0%wa** I/O等待占用CPU的百分比
* **0.0%hi** 硬中断（Hardware IRQ）占用CPU的百分比
* **0.0%si** 软中断（Software Interrupts）占用CPU的百分比
* **0.0%st** 虚拟机占用CPU的百分比

**（2）进程按照使用内存排序**

[root@KGC ~]# top -a

top - 18:07:36 up 42 min, 1 user, load average: 0.00, 0.00, 0.00

Tasks: 164 total, 1 running, 163 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

Mem: 1004412k total, 153948k used, 850464k free, 10296k buffers

Swap: 2031608k total, 0k used, 2031608k free, 37868k cached

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

1236 root 20 0 98.0m 4036 3056 S 0.0 0.4 0:00.20 sshd

1209 postfix 20 0 81524 3424 2544 S 0.0 0.3 0:00.01 qmgr

1202 root 20 0 81272 3400 2496 S 0.0 0.3 0:00.03 master

1208 postfix 20 0 81352 3380 2504 S 0.0 0.3 0:00.03 pickup

**（3）以批处理模式显示进程信息**

[root@KGC ~]# top -b

......

1229 root 20 0 4064 576 496 S 0.0 0.1 0:00.00 mingetty

1231 root 20 0 4064 576 496 S 0.0 0.1 0:00.00 mingetty

1233 root 20 0 4064 576 496 S 0.0 0.1 0:00.00 mingetty

1235 root 20 0 4064 576 496 S 0.0 0.1 0:00.00 mingetty

1236 root 20 0 98.0m 4036 3056 S 0.0 0.4 0:00.22 sshd

1243 root 18 -2 12344 2580 516 S 0.0 0.3 0:00.00 udevd

1244 root 18 -2 12344 2584 516 S 0.0 0.3 0:00.00 udevd

1248 root 20 0 105m 1876 1520 S 0.0 0.2 0:00.04 bash

1301 root 20 0 17384 668 452 S 0.0 0.1 0:00.00 anacron

1318 root 20 0 15032 1340 984 R 0.0 0.1 0:00.01 top

**^C #退出使用快捷键Ctrl+C**

**（4）显示进程的完整路径**

[root@KGC ~]# top -c

top - 18:19:38 up 54 min, 1 user, load average: 0.00, 0.00, 0.00

Tasks: 164 total, 1 running, 163 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

Mem: 1004412k total, 154196k used, 850216k free, 10316k buffers

Swap: 2031608k total, 0k used, 2031608k free, 37904k cached

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

1 root 20 0 19232 1484 1220 S 0.0 0.1 0:00.98 /sbin/init

2 root 20 0 0 0 0 S 0.0 0.0 0:00.01 [kthreadd]

3 root RT 0 0 0 0 S 0.0 0.0 0:00.00 [migration/0]

**（5） 设置执行top命令后的信息刷新时间**

[root@KGC ~]# top -d 3

**果每隔3s刷新一次**

**（6）设置执行top命令后的信息刷新次数**

[root@KGC ~]# top -n 2

**新两次后终止退出，-n参数可以和-b参数配合使用**

**（7）将top输出结果的全部信息输出到文件中**

[root@KGC ~]# top -b -n1 > test

**以批处理方式，就刷新1次数据重定向到文件里**

[root@KGC ~]# cat test | wc -l

171

[root@KGC ~]# top -n1 > test

**如果不是批处理方式，数据量少**

[root@KGC ~]# cat test | wc -l

28

**特别提示：  
在工作中，如果没有必要，我们尽量不要在服务器上直接用top无任何参数的方式查看，因为这样会非常占用系统的资源。我们可以使用top -b -n1 > test的方式将数据重定向到文件里，再进行查看。**

**（8）显示指定的进程信息**

[root@KGC ~]# top -p 1126

top - 18:31:18 up 1:06, 1 user, load average: 0.00, 0.00, 0.00

Tasks: 1 total, 0 running, 1 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

Mem: 1004412k total, 154032k used, 850380k free, 10448k buffers

Swap: 2031608k total, 0k used, 2031608k free, 38060k cached

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

1126 root 20 0 66604 1184 468 S 0.0 0.1 0:00.00 sshd

**（9）显示指定用户的信息**

[root@KGC ~]# top -u yunjisuan

top - 18:33:05 up 1:08, 2 users, load average: 0.00, 0.00, 0.00

Tasks: 165 total, 1 running, 164 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

Mem: 1004412k total, 160388k used, 844024k free, 10796k buffers

Swap: 2031608k total, 0k used, 2031608k free, 41696k cached

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

1384 yunjisua 20 0 105m 1896 1528 S 0.0 0.2 0:00.01 bash

1403 yunjisua 20 0 140m 3968 2684 S 0.0 0.4 0:00.01 vim

**6.Nohup**

**6.1 功能说明**

nohup命令可以将程序以忽略挂起信号的方式运行起来，被运行程序的输出信息将不会显示到终端。无论是否将nohup命令的输出重定向到终端，输出都将写入到当前目录的nohup.out文件中。如果当前目录的nohup.out文件不可写，则输出重定向到$HOME/nohup.out文件中。

**6.2 使用说明**

正常情况下，如果用户退出登录或会话终止，则用户正在执行并可持续一段时间的命令（非守护进程）将自动终止。使用nohup命令可以实现在用户退出或当前会话终止后继续保持运行，具体的例子如下：

[root@KGC ~]# cd

[root@KGC ~]# nohup ping [www.baidu.com](http://www.baidu.com)

**让当前执行的进程始终运行，关闭界面也不消失**

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

^C[root@KGC ~]# ls

nohup.out test

[rroot@KGC ~]# cat nohup.out

**命令的执行记录会被记录在当前目录下的nohup.out中**

PING www.a.shifen.com (61.135.169.125) 56(84) bytes of data.

64 bytes from 61.135.169.125: icmp\_seq=1 ttl=128 time=4.22 ms

64 bytes from 61.135.169.125: icmp\_seq=2 ttl=128 time=4.28 ms

64 bytes from 61.135.169.125: icmp\_seq=3 ttl=128 time=4.20 ms

64 bytes from 61.135.169.125: icmp\_seq=4 ttl=128 time=4.21 ms

64 bytes from 61.135.169.125: icmp\_seq=5 ttl=128 time=4.16 ms

64 bytes from 61.135.169.125: icmp\_seq=6 ttl=128 time=4.11 ms

64 bytes from 61.135.169.125: icmp\_seq=7 ttl=128 time=4.22 ms

64 bytes from 61.135.169.125: icmp\_seq=8 ttl=128 time=4.18 ms

64 bytes from 61.135.169.125: icmp\_seq=9 ttl=128 time=4.21 ms

--- www.a.shifen.com ping statistics ---

9 packets transmitted, 9 received, 0% packet loss, time 8508ms

rtt min/avg/max/mdev = 4.114/4.203/4.285/0.097 ms

**在工作中我们一般会配合&符号运行nohup命令，让程序直接在后台运行**

**7.strace:跟踪进程的系统调用**

**7.1 功能说明**

strace是Linux环境下的一款程序调试工具，用于检查一个应用程序所使用的系统调用以及它所接收的系统信息。strace会追踪程序运行时的整个生命周期，输出每一个系统调用的名字、参数、返回值和执行所消耗的时间等

**7.2 选项说明**

| **选项** | **解释说明** |
| --- | --- |
| -c | 统计每一个系统调用所执行的算时间、次数和出错的次数等 |
| -d | 输出strace关于标准错误的调试信息 |
| **-f** | **跟踪目标进程，以及目标进程创建的所有子进程** |
| -ff | 如果提供-o filename，则将所有进程的跟踪结果输出到相应的filename.pid中，pid是各进程的进程号 |
| -i | 输出系统调用的入口指针 |
| -q | 禁止输出关于脱离的消息 |
| -r | 输出每一个系统调用的相对时间 |
| -t | 在输出中的每一行前加上时间信息。例如：16:45:28 |
| **-tt** | **在输出中的每一行前加上时间信息，精确到微秒。例如11:18:59.759546** |
| -ttt | 在输出中的每一行前加上时间信息，精确到微秒，而且时间表示为UNIX时间戳。例如1486111461.650434 |
| -T | 显示每次系统调用所花费的时间 |
| -v | 对于某些相关调用，把完整的环境变量、文件stat结构等打印出来 |
| -x | 以十六进制形式输出非标准字符串 |
| -xx | 所有字符串以十六进制形式输出 |
| -o | 将strace的输出写入文件filename |
| -p | **指定要跟踪的进程pid，要同时跟踪多个pid，重复多次-p选项即可** |
| -s | 指定输出的字符串的最大长度，默认为32.并没有将文件名视为字符串，默认全部输出 |
| -u | 以username的UID和GID执行所跟踪的命令 |

**输出过滤器**

| **参数选项** | **解释说明** |
| --- | --- |
| -e expr | 输出过滤器，通过表达式，可以过滤掉你不想要的输出 |
| -e trace=open | 表示只跟踪open调用而-e trace！=open表示跟踪除open外所有 |
| -e trace=file | 只跟踪与文件操作有关的系统调用 |
| -e trace=process | 只跟踪与进程有关的系统调用 |
| -e trace=network | 只跟踪与网络有关的系统调用 |
| -e trace=signal | 只跟踪与系统信号有关的系统调用 |
| -e trace=desc | 只跟踪与文件描述符有关的系统调用 |
| -e trace=ipc | 只跟踪与进程通信有关的系统调用 |

**7.3使用说明**

（1）排查Nginx 403 forbidden错误

[root@localhost tmp]# strace -tt -f -o /tmp/test.txt /usr/local/nginx/sbin/nginx #f参数跟踪目标进程，以及目标进程创建的所有子进程，-tt参数在输出中的每一行前加上时间信息，-o将跟踪内容输出到文件里。

[root@localhost tmp]# cat test.txt

3824 05:37:14.300486 prctl(PR\_SET\_DUMPABLE, 1) = 0

3824 05:37:14.300498 rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

3824 05:37:14.300518 epoll\_create(512) = 8

3824 05:37:14.300535 eventfd2(0, 0) = 9

3824 05:37:14.300549 epoll\_ctl(8, EPOLL\_CTL\_ADD, 9, {EPOLLIN|EPOLLET, {u32=7095968, u64=7095968}}) = 0

3824 05:37:14.300569 mmap(NULL, 233472, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f73714f7000

3824 05:37:14.300607 brk(0x1fa5000) = 0x1fa5000

3824 05:37:14.300693 epoll\_ctl(8, EPOLL\_CTL\_ADD, 6, {EPOLLIN|0x2000, {u32=1901031440, u64=140133798998032}}) = 0

3824 05:37:14.300712 close(3) = 0

3824 05:37:14.300724 epoll\_ctl(8, EPOLL\_CTL\_ADD, 7, {EPOLLIN|0x2000, {u32=1901031664, u64=140133798998256}}) = 0

3824 05:37:14.300742 epoll\_wait(8, #epoll\_wait表示等待连接访问，因此后面的输出都是和前一次访问有关的，下面我们仔细看一下日志输出。

我们先将test.txt文件的内容清空，然后模拟去访问nginx

[root@localhost tmp]# cat test.txt

{{EPOLLIN, {u32=1901031440, u64=140133798998032}}}, 512, 4294967295) = 1

3824 05:57:59.271342 accept4(6, {sa\_family=AF\_INET, sin\_port=htons(64469), sin\_addr=inet\_addr("192.168.0.254")}, [16], SOCK\_NONBLOCK) = 3

3824 05:57:59.271383 epoll\_ctl(8, EPOLL\_CTL\_ADD, 3, {EPOLLIN|EPOLLET|0x2000, {u32=1901031888, u64=140133798998480}}) = 0

3824 05:57:59.271401 epoll\_wait(8, {{EPOLLIN, {u32=1901031888, u64=140133798998480}}}, 512, 60000) = 1

3824 05:57:59.271423 recvfrom(3, "GET / HTTP/1.1\r\nHost: 192.168.0."..., 1024, 0, NULL, NULL) = 308 #recvfrom接收到get请求

3824 05:57:59.271483 stat("/usr/local/nginx/html/www/index.html", 0x7fff69246350) = -1 ENOENT (No such file or directory) #查看index.html文件不存在

3824 05:57:59.271506 stat("/usr/local/nginx/html/www", {st\_mode=S\_IFDIR|0755, st\_size=4096, ...}) = 0

3824 05:57:59.271525 stat("/usr/local/nginx/html/www/index.htm", 0x7fff69246350) = -1 ENOENT (No such file or directory) #查看index.htm文件不存在

#下面向用户返回403错误，并写入错误日志

3824 05:57:59.271546 write(5, "2018/01/14 05:57:59 [error] 3824"..., 200) = 200

3824 05:57:59.271587 writev(3, [{"HTTP/1.1 403 Forbidden\r\nServer: "..., 155}, {"<html>\r\n<head><title>403 Forbidd"..., 116}, {"<hr><center>nginx/1.10.2</center"..., 53}], 3) = 324

3824 05:57:59.271674 write(4, "192.168.0.254 - - [14/Jan/2018:0"..., 151) = 151

3824 05:57:59.271693 setsockopt(3, SOL\_TCP, TCP\_NODELAY, [1], 4) = 0

3824 05:57:59.271708 recvfrom(3, 0x1f3e190, 1024, 0, 0, 0) = -1 EAGAIN (Resource temporarily unavailable)

3824 05:57:59.271722 epoll\_wait(8,

#从上面的日志输出中，我们可以得知是因为2个文件不存在导致的403错误，因此我们检查配置文件就很容易发现问题。

server {

listen 80;

server\_name www.yunjisuan.com;

location / {

root html/www;

index index.html index.htm; #这里缺少了设置首页文件

}

}

（2）只跟踪与文件操作有关的系统调用

如果命令结果的输出实在太多了，很容易看花眼，因此可以使用过滤器，过滤掉无关的信息，比如只查看文件操作信息。

[root@localhost tmp]# strace -tt -f -e trace=file -o /tmp/test.txt /usr/local/nginx/sbin/nginx

[root@localhost tmp]# cat test.txt

3860 06:28:42.306924 stat("/usr/local/nginx/html/www/index.html", 0x7fff3ce39670) = -1 ENOENT (No such file or directory)

3860 06:28:42.306973 stat("/usr/local/nginx/html/www", {st\_mode=S\_IFDIR|0755, st\_size=4096, ...}) = 0

3860 06:28:42.306994 stat("/usr/local/nginx/html/www/index.htm", 0x7fff3ce39670) = -1 ENOENT (No such file or directory)

（3）通过pid跟踪进程

[root@localhost tmp]# /usr/local/nginx/sbin/nginx #启动nginx服务

[root@localhost tmp]# pgrep nginx -l

3873 nginx #nginx的master进程

3874 nginx #nginx的worker进程

[root@localhost tmp]# strace -tt -f -e trace=file -p 3874 #使用-p参数，只跟踪worker进程，结果更加精简

Process 3874 attached - interrupt to quit

06:49:38.629248 stat("/usr/local/nginx/html/www/index.html", 0x7fffbd4e7fc0) = -1 ENOENT (No such file or directory)

06:49:38.629329 stat("/usr/local/nginx/html/www", {st\_mode=S\_IFDIR|0755, st\_size=4096, ...}) = 0

06:49:38.629353 stat("/usr/local/nginx/html/www/index.htm", 0x7fffbd4e7fc0) = -1 ENOENT (No such file or directory)

（4）跟踪系统调用统计

strace不仅能够追踪系统调用，使用选项-c还能对进程所有的系统调用做一个统计分析

[root@localhost tmp]# strace -c /usr/local/nginx/sbin/nginx #使用-c参数为进程所有的系统调用做一个统计分析

% time seconds usecs/call calls errors syscall

------ ----------- ----------- --------- --------- ----------------

100.00 0.000047 47 1 clone

0.00 0.000000 0 29 read

0.00 0.000000 0 31 open

0.00 0.000000 0 33 close

0.00 0.000000 0 6 stat

0.00 0.000000 0 28 fstat

0.00 0.000000 0 1 lseek

0.00 0.000000 0 61 mmap

0.00 0.000000 0 36 mprotect

0.00 0.000000 0 7 munmap

0.00 0.000000 0 6 brk

0.00 0.000000 0 14 rt\_sigaction

0.00 0.000000 0 1 rt\_sigprocmask

0.00 0.000000 0 1 ioctl

0.00 0.000000 0 2 pread

0.00 0.000000 0 2 2 access

0.00 0.000000 0 5 socket

0.00 0.000000 0 4 4 connect

0.00 0.000000 0 1 bind

0.00 0.000000 0 2 listen

0.00 0.000000 0 1 setsockopt

0.00 0.000000 0 1 execve

0.00 0.000000 0 2 uname

0.00 0.000000 0 3 fcntl

0.00 0.000000 0 5 5 mkdir

0.00 0.000000 0 3 getrlimit

0.00 0.000000 0 1 geteuid

0.00 0.000000 0 1 statfs

0.00 0.000000 0 1 arch\_prctl

0.00 0.000000 0 2 1 futex

0.00 0.000000 0 1 epoll\_create

0.00 0.000000 0 1 set\_tid\_address

0.00 0.000000 0 1 set\_robust\_list

------ ----------- ----------- --------- --------- ----------------

100.00 0.000047 294 12 total

上面的结果将清楚地告诉我们调用了哪些系统函数，调用的次数是多少，消耗了多少时间等信息，这对我们分析程序来说是非常有用的。

（5）重定向输出

[root@localhost tmp]# strace -c -o /tmp/test.txt /usr/local/nginx/sbin/nginx #-o选项将strace的结果输出到文件中

[root@localhost tmp]# cat test.txt

% time seconds usecs/call calls errors syscall

------ ----------- ----------- --------- --------- ----------------

-nan 0.000000 0 29 read

-nan 0.000000 0 31 open

-nan 0.000000 0 33 close

-nan 0.000000 0 6 stat

（6）对系统调用进行计时

[root@localhost tmp]# /usr/local/nginx/sbin/nginx -s stop

[root@localhost tmp]# strace -T /usr/local/nginx/sbin/nginx #使用-T将每个系统调用所花费的时间打印出来，每个调用的时间花销在调用行最右边的尖括号里

execve("/usr/local/nginx/sbin/nginx", ["/usr/local/nginx/sbin/nginx"], [/\* 24 vars \*/]) = 0 <0.000075>

brk(0) = 0x1055000 <0.000003>

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fbbf84f6000 <0.000004>

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory) <0.000005>

open("/etc/ld.so.cache", O\_RDONLY) = 3 <0.000004>

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=15441, ...}) = 0 <0.000003>

mmap(NULL, 15441, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fbbf84f2000 <0.000003>

close(3) = 0 <0.000003>

open("/lib64/libdl.so.2", O\_RDONLY) = 3 <0.000005>

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340\r\0\0\0\0\0\0"..., 832) = 832 <0.000003>

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=19536, ...}) = 0 <0.000003>

mmap(NULL, 2109696, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fbbf80d4000 <0.000004>

mprotect(0x7fbbf80d6000, 2097152, PROT\_NONE) = 0 <0.000004>

......

**strace命令很适合处理程序僵尸、命令执行报错等问题。如果从程序日志和系统日志中看不出问题出现的原因，则可以strace一下，也许会有答案，不过也需要使用者有足够的耐心去查看输出**

**8. ltrace**

**8.1功能说明：**

ltrace能够跟踪进程的库函数调用，它会显现出调用了哪个库函数，而strace则是跟踪进程的每个系统调用

**8.2选项说明：**

| 参数选项 | 解释说明 |
| --- | --- |
| -c | 统计库函数每次调用的时间，最后程序退出时打印摘要 |
| -C | 解码低级别名称（内核级）为用户级名称 |
| -d | 打印调试信息 |
| **-e expr** | **输出过滤器，通过表达式，可以过滤掉你不想要的输出** |
| -e printf | 表示只查看printf函数调用 |
| -f | 跟踪子进程 |
| -o filename | 将ltrace的输出写入文件filename |
| -p pid | 指定要跟踪的进程pid@ |
| -r | 输出每一个调用的相对时间 |
| -S | 显示系统调用 |
| -t | 在输出中每一行前加上时间信息。例如 16:45:28 |
| -tt | 在输出中每一行前加上时间信息。例如 11:18:59.759546 |
| -ttt | 在输出中每一行前加上时间信息，精确到微妙，而且时间表示为UNIX时间戳。 |
| -T | 显示每次调用所花费时间 |
| -u username | 以username的UID和GID执行所跟踪的命令 |

**8.3使用说明**

（1）ltrace使用

ltrace的用法与strace非常相似，选项功能也是类似，下面简单看一下ltrace命令的执行结果

[root@localhost tmp]# ltrace /usr/local/nginx/sbin/nginx

**后面直接接上要检测的命令语句**

memcpy(0x1d5b7b0, "www.yunjisuan.com", 17) = 0x1d5b7b0

memcpy(0x1d5b7c1, "/usr/local/nginx/", 17) = 0x1d5b7c1

malloc(2048) = 0x1d55530

memset(0x1d55530, '\000', 68) = 0x1d55530

memset(0x1d55530, '\000', 70) = 0x1d55530

memset(0x1d55530, '\000', 72) = 0x1d55530

memset(0x1d55530, '\000', 74) = 0x1d55530

memset(0x1d55530, '\000', 76) = 0x1d55530

memset(0x1d55530, '\000', 78) = 0x1d55530

memset(0x1d55530, '\000', 80) = 0x1d55530

memset(0x1d55530, '\000', 82) = 0x1d55530

memset(0x1d55530, '\000', 84) = 0x1d55530

memset(0x1d55530, '\000', 86) = 0x1d55530

......

sigemptyset(0x7fffb75519f8) = 0

sigaction(17, 0x7fffb75519f0, NULL) = 0

sigemptyset(0x7fffb75519f8) = 0

sigaction(31, 0x7fffb75519f0, NULL) = 0

sigemptyset(0x7fffb75519f8) = 0

sigaction(13, 0x7fffb75519f0, NULL) = 0

fork() = 3962

[pid 3961] exit(0 <unfinished ...>

[pid 3961] +++ exited (status 0) +++

（2）通过pid跟踪进程调用库函数

[root@localhost tmp]# pgrep -l nginx

3967 nginx

3968 nginx

**nginx的worker进程的pid号**

[root@localhost tmp]# ltrace -p 3968

**使用-p指定进程号**

\_\_errno\_location() = 0x7f0a392e1768

gettimeofday(0x7fff8f684cd0, NULL) = 0

memcpy(0x6b281f, "14", 2) = 0x6b281f

memcpy(0x6b2826, "2018", 4) = 0x6b2826

memcpy(0x6b282b, "12", 2) = 0x6b282b

memcpy(0x6b282e, "28", 2) = 0x6b282e

memcpy(0x6b2831, "30", 2) = 0x6b2831

localtime\_r(0x7fff8f684c88, 0x7fff8f684d20, 0x48400a, 12339, 0x7fff8f684b52) = 0x7fff8f684d20

memcpy(0x6b22fc, "2018", 4) = 0x6b22fc

memcpy(0x6b2302, "1", 1) = 0x6b2302

memcpy(0x6b2304, "14", 2) = 0x6b2304

memcpy(0x6b2308, "7", 1) = 0x6b2308

memcpy(0x6b230a, "28", 2) = 0x6b230a

memcpy(0x6b230d, "30", 2) = 0x6b230d

memcpy(0x6b2f91, "14", 2) = 0x6b2f91

memcpy(0x6b2f98, "2018", 4) = 0x6b2f98

memcpy(0x6b2f9e, "7", 1) = 0x6b2f9e

memcpy(0x6b2fa0, "28", 2) = 0x6b2fa0

memcpy(0x6b2fa3, "30", 2) = 0x6b2fa3

memcpy(0x6b2fa8, "5", 1) = 0x6b2fa8

memcpy(0x6b2faa, "0", 1) = 0x6b2faa

memcpy(0x6b364e, "2018", 4) = 0x6b364e

memcpy(0x6b3654, "1", 1) = 0x6b3654

memcpy(0x6b3656, "14", 2) = 0x6b3656

memcpy(0x6b365a, "7", 1) = 0x6b365a

memcpy(0x6b365c, "28", 2) = 0x6b365c

memcpy(0x6b365f, "30", 2) = 0x6b365f

memcpy(0x6b3663, "5", 1) = 0x6b3663

memcpy(0x6b3666, "0", 1) = 0x6b3666

memcpy(0x6b3cb4, "14", 2) = 0x6b3cb4

memcpy(0x6b3cb8, "7", 1) = 0x6b3cb8

memcpy(0x6b3cba, "28", 2) = 0x6b3cba

memcpy(0x6b3cbd, "30", 2) = 0x6b3cbd

epoll\_wait(8, 0x2639990, 512, 0xffffffff, 0x7fff8f684b52