

第1章 进程管理

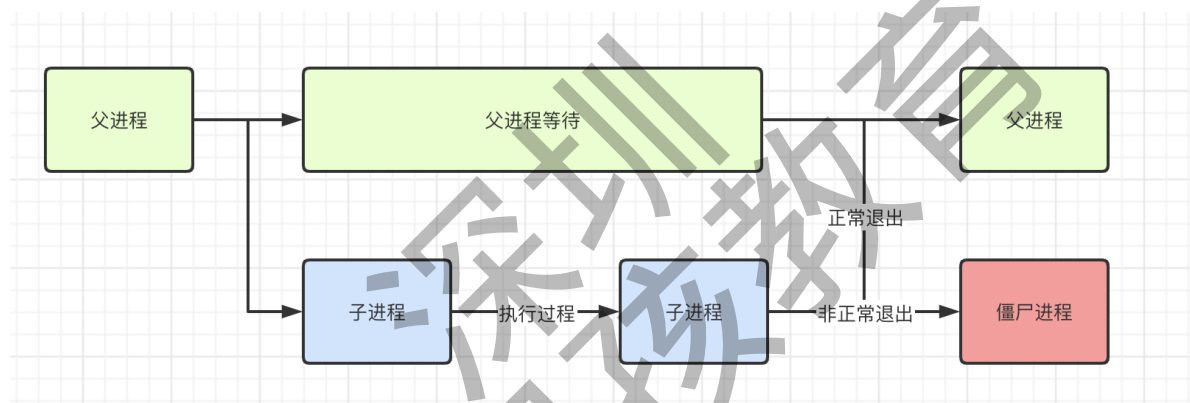
1.进程介绍

1.1 程序/进程的关系

1. 开发写好的代码，没有运行的时候，只是静态的文件，我们称之为程序，程序是数据和指令的集合。
2. 当我们把开发好的代码程序运行起来的时候，我们称之为进程。
3. 程序运行的时候系统会为进程分配PID，运行用户，分配内存等资源。

1.2 进程的生命周期

1. 当程序运行的时候会由父进程通过**fork**创建子进程来处理任务。
2. 子进程被创建后开始处理任务，当任务处理完毕后就会退出，然后子进程会通知父进程来回收资源。
3. 如果子进程处理任务期间，父进程意外终止了，那么这个子进程就变成了僵尸进程。



1.3 如何查看当前中端的进程号和父进程号

- 1 查看当前进程号：`echo $BASHPID`
- 2 查看父进程号：`echo $PPID`

2.进程监控命令

那么在程序运行后，我们如何了解进程运行的各种状态呢？这就需要使用各种查看进程状态的命令来查看了。

2.1 ps 查看当前的进程状态--最常用的

命令作用：

- 1 `ps process state` 进程状态
- 2 打印出进程当前运行的状态快照，并不是实时的监控，而是某一时刻系统进程的快照

常用参数：

- 1 `#UNIX风格`
- 2 `a` `#显示包含所有终端中的进程`

```

3 u          #此选项使ps列出您拥有的所有进程（与ps相同的EUID），或与a选项一起使用时列出
  所有进程。
4 x          #显示进程所有者的信息
5 f          #显示进程树
6 o          #属性，显示定制的信息
7 k          #属性，对属性进行排序，属性前加-表示倒叙
8 --sort    #同上
9
10 #使用标准语法查看系统上的每个进程
11 -e        #显示所有进程
12 -f        #显示完整格式程序信息
13 -p        #显示pid的进程
14 -C        #指定命令
15 -q        #指定id

```

运行结果：

```

1 #ps aux
2 [root@linux ~]# ps aux
3 USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
4 root           1  0.0  0.1 43524  3836 ?        Ss   04:20   0:02
  /usr/lib/systemd/
5 root           2  0.0  0.0      0      0 ?        S    04:20   0:00 [kthreadd]
6 root           3  0.0  0.0      0      0 ?        S    04:20   0:01
  [ksoftirqd/0]
7 root           5  0.0  0.0      0      0 ?        S<   04:20   0:00
  [kworker/0:0H]
8
9 #ps -ef
10 [root@linux ~]# ps -ef
11 UID          PID   PPID  C STIME TTY          TIME CMD
12 root         1     0  0 04:20 ?          00:00:02 /usr/lib/systemd/systemd
13 root         2     0  0 04:20 ?          00:00:00 [kthreadd]
14 root         3     2  0 04:20 ?          00:00:01 [ksoftirqd/0]
15 root         5     2  0 04:20 ?          00:00:00 [kworker/0:0H]
16 root         6     2  0 04:20 ?          00:00:01 [kworker/u256:0]

```

状态解释：

```

1 USER:      #启动进程的用户
2 PID:       #进程ID号
3 %CPU:      #进程占用的CPU百分比
4 %MEM:      #进程占用的内存百分比
5 VSZ:       #进程申请占用的虚拟内存大小，单位KB
6 RSS:       #进程实际占用的物理内存实际大小，单位KB
7 TTY:       #进程运行的终端，? 表示为内核程序，与终端无关
8 STAT:      #进程运行中个的状态，可以man ps关键词/STATE查询详细帮助手册
9
10 ###进程状态
11 D: 不可中断睡眠
12 R: 正在运行
13 S: 可中断睡眠
14 T: 进程被暂停
15 Z: 僵尸进程
16
17 ###进程字符
18 <: 高优先级进程  S<表示优先级较高的进程

```

18	N:	低优先级进程	SN表示优先级较低的进程
19	s:	子进程发起者	Ss表示父进程
20	l:	多线程进程	Sl表示进程以多线程运行
21	+	前台进程	R+表示该进程在前台运行，一旦终止，数据丢失
22	START:	#进程启动时间	
23	TIME:	#进程占用CPU时间	
24	COMMAND:	#程序运行的指令 []表示属于内核态的进程。没有[]表示用户态进程	

案例演示:

```

1  #1.查看进程的父子关系
2  ps auxf
3
4  #2.查看进程的特定属性
5  ps axo pid,cmd,%cpu,%mem
6
7  #3.按CPU利用率排序
8  ps axo pid,cmd,%cpu,%mem k -%cpu
9
10 #4.按内存使用倒序排序
11 ps axo pid,cmd,%cpu,%mem --sort %mem
12
13 #5.列出指定用户名和或用户ID的进程
14 ps -fu oldboy
15 ps -fu 1000
16
17 #6.查看指定进程ID对应的进程
18 ps -fp PID
19
20 #7.查找指定父进程ID下的所有的子进程
21 ps -f --ppid PID
22
23 #8.按照tty显示所属进程
24 ps -ft pts/1
25
26 #9.根据进程名查找所属PID
27 ps -C sshd -o pid=
28
29 #10.根据PID查找运行的命令
30 ps -p PID -o comm=

```

2.2 pstree 以树状图查看进程状态

命令作用:

```
1 | 以树状图显示父进程和子进程的关系
```

命令格式:

```
1 | pstree [选项] [pid|user]
```

常用参数:

```
1 -p 显示PID
2 -u 显示用户切换
3 -H pid 高亮显示指定进程
```

案例演示：

```
1 #1.查看指定pid的进程关系
2 pstree 1
3
4 #2.查看oldboy用户的进程关系
5 pstree oldboy
6
7 #3.显示pid
8 pstree -p
9 pstree -p 1
10 pstree -p oldboy
11
12 #4.显示运行用户
13 pstree -u
```

2.3 pidof 查看指定名称进程的进程号

命令作用：

```
1 显示指定程序的pid
```

举例：

```
1 pidof nginx
```

2.4 top 查看当前的进程状态

命令作用：

```
1 展示进程动态的实时数据
```

常用参数：

```
1 -d #指定刷新时间，默认3秒
2 -n #刷新多少次后退出
3 -p #指定pid
```

内置命令：

```
1 #帮助
2 h/? #查看帮助
3 q/esc #退出帮助
4
5 #排序：
6 P #按CPU使用百分比排序输出
7 M #按内存使用百分比排序输出
8
9 #显示
```

10	1	#数字1, 显示所有CPU核心的负载
11	m	#显示内存内存信息, 进度条形式
12	t	#显示CPU负载信息, 进度条形式
13		
14	#退出	
15	q	#退出top命令

运行结果:

```

1 [root@linux ~]# top
2 top - 16:42:49 up 12:22,  3 users,  load average: 0.00, 0.03, 0.05
3 Tasks: 106 total,  1 running, 105 sleeping,  0 stopped,  0 zombie
4 %Cpu(s):  0.0 us,  6.2 sy,  0.0 ni, 93.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
  st
5 KiB Mem : 2028088 total,  465376 free,  110864 used, 1451848 buff/cache
6 KiB Swap:      0 total,      0 free,      0 used. 1707948 avail Mem
7
8      PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM    TIME+  COMMAND
9          1 root        20   0  43524   3836   2580 S   0.0   0.2   0:02.61 systemd
10         2 root        20   0      0      0      0 S   0.0   0.0   0:00.00 kthreadd
11         3 root        20   0      0      0      0 S   0.0   0.0   0:01.45
  ksoftirqd/0
12         5 root        0 -20      0      0      0 S   0.0   0.0   0:00.00
  kworker/0:0H
13         6 root        20   0      0      0      0 S   0.0   0.0   0:01.24
  kworker/u256:0
14         7 root        rt   0      0      0      0 S   0.0   0.0   0:00.00
  migration/0
  
```

状态解释:

1	Tasks: 106 total,	#当前进程的总数
2	1 running,	#正在运行的进程数
3	105 sleeping,	#睡眠中的进程数
4	0 stopped,	#停止的进程数
5	0 zombie	#僵尸进程数
6		
7	%Cpu(s):	#平均CPU使用率, 按1查看每个cpu核的具体状态
8	0.0 us,	#用户进程占用CPU百分比
9	6.2 sy,	#系统进程占用CPU百分比
10	0.0 ni,	#优先级进程占用cpu的百分比
11	93.8 id,	#空闲cup
12	0.0 wa,	#CPU等待IO完成的时间, 大量的io等待, 会变高
13	0.0 hi,	#硬中断, 占的CPU百分比
14	0.0 si,	#软中断, 占的CPU百分比
15	0.0 st	#虚拟机占用物理CPU的时间

什么是I/O?

- 1 I/O称为硬盘的: 写入/读取也就是硬盘的读写。
- 2 简单来说: 系统发送读或写的指令, 硬盘收到指令后进行读或写数据, 这个过程就是一次IO。

什么是软中断和硬中断？

- 1 在Linux的实现中，有两种类型的中断。
- 2 硬中断是由请求响应的设备发出的（磁盘I/O中断、网络适配器中断、键盘中断、鼠标中断）
- 3 软中断被用于处理可以延迟的任务（TCP/IP操作，SCSI协议操作等等）
- 4
- 5 简单来说，就是中断就是告诉操作系统，停下你手里的工作，先处理我的任务。
- 6 硬中断是由硬件发出的，软中断是由正在运行的进程发出的。

案例演示：

- 1 #1. 查看指定pid的进程信息
- 2 top -H -p \$(pidof ping)

2.5 htop 更高级的top

命令作用：

- 1 界面更美观的增强版top

常用选项：

- 1 -d #指定刷新时间
- 2 -s #以指定列字段排序

常用指令：

- 1 t #显示进程树

运行结果：

CPU[0.0%] Tasks: 30, 14 thr; 1 running											
Mem[151M/1.93G] Load average: 0.09 0.05 0.05											
Swp[0K/0K] Uptime: 12:30:43											
PID	USER	PRI	NI	VRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1	root	20	0	43524	3836	2580	S	0.0	0.2	0:02.63	/usr/lib/systemd/syst
3115	root	20	0	39076	5004	4684	S	0.0	0.2	0:00.54	/usr/lib/systemd/syst
3136	root	20	0	45064	2444	1320	S	0.0	0.1	0:00.50	/usr/lib/systemd/syst
3138	root	20	0	121M	1340	988	S	0.0	0.1	0:00.00	/usr/sbin/lvmेटad -f
5519	root	16	-4	62044	1288	660	S	0.0	0.1	0:00.02	/sbin/auditd
5512	root	16	-4	62044	1288	660	S	0.0	0.1	0:00.16	/sbin/auditd
5923	polkitd	20	0	597M	10084	4680	S	0.0	0.5	0:00.00	/usr/lib/polkit-1/pol
5950	polkitd	20	0	597M	10084	4680	S	0.0	0.5	0:00.40	/usr/lib/polkit-1/pol
5960	polkitd	20	0	597M	10084	4680	S	0.0	0.5	0:00.00	/usr/lib/polkit-1/pol
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice -F8Nice +F9Kill F10Quit											

2.6 lsof 查看进程打开文件

命令作用：

- 1 查看当前系统文件的工具。
- 2 在Linux下，一切皆文件，包括硬件及网络协议等也都可以访问，系统会为每一个程序都分配一个文件描述符。

常用参数：

1	-c	#显示指定程序名所打开的文件
2	-a	
3	-d	#显示打开这个文件的进程名
4	-i	#显示符合条件的进程,IPv[46][proto][@host addr][:svc_list port_list]
5	-p	#显示指定进程pid所打开的文件
6	-u	#显示指定用户UID的进程
7	+d	#列出目录下被打开的文件
8	+D	#递归累出目录下被打开的文件

状态解释:

1	COMMAND	进程的名称
2	PID	进程标识符
3	USER	进程所有者
4	FD	文件描述符,应用程序通过文件描述符识别该文件。如cwd、txt等
5	TYPE	文件类型,如DIR、REG等
6	DEVICE	指定磁盘的名称
7	SIZE	文件的大小
8	NODE	索引节点(文件在磁盘上的标识)
9	NAME	打开文件的确切名称

案例演示:

```

1 #查看正在使用此文件的进程
2 lsof /var/log/nginx/access.log
3
4 #查看指定pid号的进程打开的文件
5 lsof -p $(cat /var/run/nginx.pid)
6
7 #查看指定程序打开的文件
8 lsof -c nginx
9
10 #查看指定用户打开的文件
11 lsof -u nginx
12
13 #查看指定目录下被打开的文件
14 lsof +d /var/log
15 lsof +D /var/log
16
17 #查看指定IP的连接
18 lsof -i@10.0.0.1
19
20 #查看指定进程打开的网络连接
21 lsof -i -a -n -p 8765
22
23 #查看指定TCP状态的连接
24 lsof -n -P -i TCP -s TCP:LISTEN

```

模拟误删除文件:

前提: 被误删的文件还有进程在使用, 并且这个进程不能重启或退出。

```

1 #1.新开一个终端使用tail查看nginx的访问日志,这里只是方便演示效果,不tail也可以恢复,只要nginx进程没有重启
2 [root@linux ~]# tail -f /var/log/nginx/access.log

```

```

3
4 #2.然后在另一个终端重看日志的进程信息
5 [root@linux ~]# lsof |grep access.log
6 nginx      9067      root    5w      REG          253,0        0
67881734 /var/log/nginx/access.log
7 nginx      9068      nginx   5w      REG          253,0        0
67881734 /var/log/nginx/access.log
8 tail       9099      root    3r      REG          253,0        0
67881734 /var/log/nginx/access.log
9
10 #3.此时我们模拟误删除了日志文件
11 [root@linux ~]# rm -rf /var/log/nginx/access.log
12 [root@linux ~]# ll /var/log/nginx/access.log
13 ls: 无法访问/var/log/nginx/access.log: 没有那个文件或目录
14
15 #4.然后我们再次查看相关进程会发现日志文件后多了一个(deleted)字样, 等得出PID以及文件描述符
16 [root@linux ~]# lsof |grep access.log
17 nginx      9067      root    5w      REG          253,0        0
67881734 /var/log/nginx/access.log (deleted)
18 nginx      9068      nginx   5w      REG          253,0        0
67881734 /var/log/nginx/access.log (deleted)
19 tail       9099      root    3r      REG          253,0        0
67881734 /var/log/nginx/access.log (deleted)
20
21 #5.然后我们直接进入/proc/9067查看一下
22 [root@linux fd]# ll
23 总用量 0
24 lrwx----- 1 root root 64 4月  4 15:58 0 -> /dev/null
25 lrwx----- 1 root root 64 4月  4 15:58 1 -> /dev/null
26 l-wx----- 1 root root 64 4月  4 15:58 2 -> /var/log/nginx/error.log
27 lrwx----- 1 root root 64 4月  4 15:58 3 -> socket:[92536]
28 l-wx----- 1 root root 64 4月  4 15:58 4 -> /var/log/nginx/error.log
29 l-wx----- 1 root root 64 4月  4 15:58 5 -> /var/log/nginx/access.log
(deleted)
30 lrwx----- 1 root root 64 4月  4 15:58 6 -> socket:[92525]
31 lrwx----- 1 root root 64 4月  4 15:58 7 -> socket:[92537]
32
33 #6.通过查看可以发现, 名称为5的连接就是我们删除后的文件
34 [root@linux fd]# head -n 3 5
35 10.0.0.1 - - [04/Apr/2021:16:43:42 +0800] "GET / HTTP/1.1" 502 559 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/88.0.4324.192 Safari/537.36" "-"
36 10.0.0.1 - - [04/Apr/2021:16:43:42 +0800] "GET / HTTP/1.1" 502 559 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/88.0.4324.192 Safari/537.36" "-"
37 10.0.0.1 - - [04/Apr/2021:16:43:42 +0800] "GET / HTTP/1.1" 502 559 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/88.0.4324.192 Safari/537.36" "-"
38
39 #7.接着我们就可以把数据恢复出来了
40 [root@linux fd]# cat 5 > /var/log/nginx/access.log
41 [root@linux fd]# ll /var/log/nginx/access.log
42 -rw-r--r-- 1 root root 1194 4月  4 16:44 /var/log/nginx/access.log

```

3.进程管理命令

3.1 控制信号

什么是控制信号？

- 1 简单来说，就是向进程发送控制信号，一般用于停止或杀死进程。
- 2 每个信号对应1个数字，信号名称以SIG开头，可以省略SIG头。

Linux常见控制信号

1	信号名称	数字编号	信号含义
2	HUP	1	无需关闭进程并且让其重新加载配置文件
3	KILL	9	强制杀死正在运行的进程
4	TERM	15	终止正在运行的进程，不特别指定数字信号的话，默认就是15

控制进程的命令

- 1 kill #结束指定PID的进程
- 2 killall #根据进程名杀死进程
- 3 pkill #同样是根据进程名杀死进程

3.2 kill

命令作用：

- 1 根据进程的pid号杀死指定的进程

常用参数：

- 1 kill -1 pid #重新加载配置
- 2 kill -15 pid #优雅的停止进程
- 3 kill -9 pid #强制终止进程

案例演示：

```
1 #1.安装nginx
2 [root@linux ~]# yum install nginx -y
3
4 #2.创建2个新的默认网页
5 [root@linux ~]# echo v1 > /usr/share/nginx/html/index.html
6 [root@linux ~]# cat /usr/share/nginx/html/index.html
7 v1
8 [root@linux ~]# echo v2 > /usr/share/nginx/html/index2.html
9 [root@linux ~]# cat /usr/share/nginx/html/index2.html
10 v2
11
12 #3.启动nginx并查看网页信息
13 [root@linux ~]# systemctl start nginx
14 [root@linux ~]# curl 127.0.0.1
15 v1
16
17 #4.修改nginx配置文件，将默认首先更新为index2.html
18 [root@linux ~]# sed -i 's#index.html#index2.html#g'
    /etc/nginx/conf.d/default.conf
```

```

19 [root@linux ~]# grep "index2" /etc/nginx/conf.d/default.conf
20     index index2.html index.htm;
21
22 #5.这时在访问一次。会发现依然是v1版本，为什么？因为我们虽然修改了配置文件，但是并没有重
    启，所以配置文件没有被加载。
23 [root@linux ~]# curl 127.0.0.1
24 v1
25
26 #6.查看nginx自带的启动文件里是如何重新加载配置文件的
27 [root@linux ~]# rpm -ql nginx |grep nginx.service
28 /usr/lib/systemd/system/nginx.service
29 [root@linux ~]# grep -i "reload" /usr/lib/systemd/system/nginx.service
30 ExecReload=/bin/sh -c "/bin/kill -s HUP $(cat /var/run/nginx.pid)"
31 #可以发现nginx重新加载配置是分为了2步操作：
32 1.找出正在运行的nginx的pid号
33 2.向nginx进程发送HUP信号，也就是0型号以使其重新加载配置
34
35 #7.发送控制信号重新加载配置
36 [root@linux ~]# pidof nginx
37 20951 20950
38 [root@linux ~]# cat /var/run/nginx.pid
39 20950
40 [root@linux ~]# kill -1 20950
41
42 #8.重新查看网页可以发现已经切换到了v2版本，说明配置被重新加载了
43 [root@linux ~]# curl 127.0.0.1
44 v2
45
46 #9.使用kill杀死进程
47 [root@linux ~]# kill 20950
48 [root@linux ~]# ps -ef|grep nginx

```

3.3 pkill和killall

命令作用：

- 1 类似于kill，只不过kill需要先找出pid号，然后再杀掉进程
- 2 而pkill则可以将kill的两步操作合二为一，只需要根据进程名就可以杀死进程
- 3 不过要注意的是pkill的优势也有可能造成误杀，因为是根据进程名杀死进程，那么只要匹配上进程名的进程，哪怕只是名称里包含了匹配的，也会被杀掉。

案例演示：

```

1 #1.启动并查看nginx
2 [root@linux ~]# systemctl start nginx
3 [root@linux ~]# ps -ef|grep nginx|grep -v grep
4 root      21088      1  0 16:49 ?          00:00:00 nginx: master process
    /usr/sbin/nginx -c /etc/nginx/nginx.conf
5 nginx     21089    21088  0 16:49 ?          00:00:00 nginx: worker process
6
7 #2.使用pkill杀死进程
8 [root@linux ~]# pkill nginx
9 [root@linux ~]# ps -ef|grep nginx|grep -v grep
10
11 #3.重新启动nginx并查看进程
12 [root@linux ~]# systemctl start nginx

```

```

13 [root@linux ~]# ps -ef|grep nginx|grep -v grep
14 root      21141      1  0 16:51 ?        00:00:00 nginx: master process
   /usr/sbin/nginx -c /etc/nginx/nginx.conf
15 nginx     21142    21141  0 16:51 ?        00:00:00 nginx: worker process
16
17 #4.使用killall杀死nginx进程
18 [root@linux ~]# killall nginx
19 [root@linux ~]# ps -ef|grep nginx|grep -v grep

```

4.进程后台管理

前台运行与后台运行

1. 前台进程就是运行在当前的终端，并且运行中的信息都会输出到屏幕上，会一直占用终端的使用。如果当前终端关闭了，则进程就自动退出了。
2. 而后台进程则可以在终端的后台继续运行，但是并不会占用当前的终端使用。即使当前终端关闭了，进程也不会退出。

&,jobs,bg,fg -- 了解即可

命令作用：

```

1 &                #把未启动的进程放在后台执行
2 jobs            #查看后台进程
3 ctrl + z        #将运行中的进程放在后台
4 bg              #
5 fg              #

```

案例演示：

```

1 [root@linux ~]# sleep 3000 &
2 [1] 22125
3 [root@linux ~]# sleep 4000
4 ^Z
5 [2]+  已停止                  sleep 4000
6 [root@linux ~]# ps aux|grep sleep
7 root      22125  0.0  0.0 108052   612 pts/0    S   20:41    0:00 sleep 3000
8 root      22127  0.0  0.0 108052   616 pts/0    T   20:41    0:00 sleep 4000
9 [root@linux ~]# jobs
10 [1]-  运行中                  sleep 3000 &
11 [2]+  已停止                  sleep 4000
12
13 [root@linux ~]# bg %2
14 [2]+  sleep 4000 &
15 [root@linux ~]# jobs
16 [1]-  运行中                  sleep 3000 &
17 [2]+  运行中                  sleep 4000 &
18
19 [root@linux ~]# fg %1
20 sleep 3000
21
22 [root@linux ~]# kill %2
23 [root@linux ~]# jobs
24 [2]-  已终止                  sleep 4000
25 [3]+  运行中                  sleep 3000 &

```

screen -- 重点推荐

命令作用：

- 1 | 工作中非常推荐使用的后台进程管理工具，作用是可以将进程放在后台运行，并且可以随时切换到前台。

常用参数：

- 1 | `screen -S 终端名称` #新建一个指定名称的终端
- 2 | `Ctrl + a + d` #切换到前台，但是保持后台运行的进程
- 3 | `screen -ls` #查看已经放在后台的进程名称列表
- 4 | `screen -r 名称或ID号` #进入指定名称的后台程序

案例演示：

- 1 | #1.创建名为ping的终端
- 2 | [root@linux ~]# screen -S ping
- 3 |
- 4 | #2.切换到后台后执行前台命令
- 5 | [root@linux ~]# ping 127.0.0.1
- 6 | PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
- 7 | 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.012 ms
- 8 | 64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.038 ms
- 9 |
- 10 | #3.切出后台，切换到前台
- 11 | Ctrl + a + d
- 12 |
- 13 | #4.查看已经在运行的后台进程列表
- 14 | [root@linux ~]# screen -ls
- 15 | There is a screen on:
- 16 | 22237.ping (Detached)
- 17 | 1 Socket in /var/run/screen/S-root.
- 18 |
- 19 | #5.重新进入指定名称的后台进程
- 20 | [root@linux ~]# screen -r ping

nohub &

命令作用：

- 1 | 将进程放在后台执行，并且把运行过程输出到日志里。

常用参数：

- 1 | `nohup 命令 >> 日志 2>&1 &`

案例演示：

- 1 | `nohup /usr/sbin/nginx >> /var/log/nginx/access.log 2>&1 &`

第2章 系统平均负载监控

uptime 查看系统平均负载

什么是系统负载？

- 1 系统负载是指单位时间内，系统处于可运行状态和不可中断状态的平均进程数，也就是平均活跃进程数。
- 2 简单来说就是可以被分配给CPU处理的活动进程数。
- 3 注意，平均负载指的并不是系统CPU使用率。

什么是可运行状态和不可中断状态？

- 1 1.可运行状态进程，是指正在使用 CPU 或者正在等待 CPU 的进程，也就是我们ps命令看到处于R状态的进程。
- 2 2.不可中断进程，系统中最常见的是等待硬件设备的I/O响应，也就是我们ps命令中看到的D状态（也称为 Disk Sleep）的进程。
- 3 例如：当一个进程向磁盘读写数据时，为了保证数据的一致性，在得到磁盘回复前，它是不能被其他进程或者中断打断的，这个时候的进程就处于不可中断状态。如果此时的进程被打断了，就容易出现磁盘数据与进程数据不一致的问题。所以，不可中断状态实际上是系统对进程和硬件设备的一种保护机制。
- 4
- 5 在Linux的实现中，有两种类型的中断。
- 6 硬中断是由请求响应的设备发出的（磁盘I/O中断、网络适配器中断、键盘中断、鼠标中断）。
- 7 软中断被用于处理可以延迟的任务（TCP/IP操作，SCSI协议操作等等）

平均负载为多少合适？

- 1 最理想的状态是每个CPU核上都刚好运行着一个进程，这样每个CPU都得到了充分利用。
- 2 所以首先你需要知道你的服务器上有几个CPU，可以通过lscpu命令或top交互模式按1查看有几核。
- 3 假如有以下核数的CPU，负载为2时，说明了什么？
- 4
- 5

CPU核数	负载	说明
1	2	表示有一半的进程需要等待
2	2	表示刚好所有的CPU都被使用了
4	2	表示有一半的CPU处于空闲状态
- 6
- 7
- 8
- 9
- 10 uptime命令有三个数值，分别为1，5，15分钟的平均负载，那么该如何理解这三个数值呢？或者说更应该关注哪个值？
- 11 简单来说，三个数值都要看，但是一定要分析变化的趋势，虽然三个值是数字，但是他们表达的是1到15分钟CPU负载变化的趋势。
- 12
- 13 例如如下几种情况：
- 14 1.三个数值差不多一样，这表示了系统运行很稳定，15分钟内系统没有特别繁忙。
- 15 2.如果1分钟的值大于15分钟的值，那表示系统负载1分钟内有上升的趋势。但是并不能立即下结论，因为有可能只是临时的升高，所以要持续的观察一段时间，结合这段时间的趋势分析。
- 16 3.如果1分钟负载低于15分钟的值，那表示系统负载是在下降的趋势。

如何监控系统负载的趋势呢？

- 1 实际工作中我们不可能实时的使用uptime命令来监控系统负载，那么我们如何随时系统平均的负载呢？
- 2 1.自己编写脚本然后配合定时任务周期性的运行uptime命令并输出到文本里
- 3 2.推荐使用成熟的监控工具，比如zabbix,Prometheus等监控平台，因为他们具有丰富的图形报表功能，更容易观察变化趋势。

假如平均负载变高了，应该如何处理？

- 1 虽然理想情况是每个CPU核数都被使用，但是也不是说如果平均负载超过了CPU核数系统就不正常了。
- 2 如果只是短暂的超过了CPU核数，并且平均负载有下降的趋势，那可以理解为刚才系统只是暂时的繁忙，现在已经恢复正常。
- 3 如果持续的负载高，但是业务系统并没有变慢，系统也没有进一步变糟糕的趋势，那么也不用过于担心。
- 4
- 5 如果系统确实负载高了，那么可以从以下几个角度分析：
- 6 1. 运行任务是否CPU密集型
- 7 2. 运行任务是不是IO密集型

命令作用：

- 1 通过uptime命令我们可以了解到系统的CPU负载情况，当然top命令也可以。

运行结果：

- 1 [root@linux ~]# uptime
- 2 10:14:58 up 15:01, 1 user, load average: 0.01, 0.03, 0.05

状态解释：

- 1 10:14:58 #当前时间
- 2 up 15:01 #系统已启动时间
- 3 1 user #当前登陆的用户数
- 4 load average: 0.01, 0.03, 0.05 #系统 1分钟，5分钟，15分钟的平均负载

案例演示1：模拟CPU使用率100%的场景

这里我们使用三个工具来找出平均负载变高的原因

stress: 压测工具

mpstat: 多核CPU性能分析工具

pidstat: 实时查看CPU，内存，IO等指标

- 1 #1. 安装性能监控工具
- 2 [root@linux ~]# wget http://pagesperso-orange.fr/sebastien.godard/sysstat-11.7.3-1.x86_64.rpm
- 3 [root@linux ~]# rpm -ivh sysstat-11.7.3-1.x86_64.rpm
- 4
- 5 #2. 使用strss工具模拟CPU使用率100%
- 6 [root@linux ~]# stress --cpu 1 --timeout 600
- 7 stress: info: [18742] dispatching hogs: 1 cpu, 0 io, 0 vm, 0 hdd
- 8
- 9 #3. 在第2个终端运行uptime命令实时查看系统的负载
- 10 ##可以看到1分钟的负载正在升高
- 11 [root@linux ~]# watch -d uptime
- 12 16:06:47 up 20:53, 3 users, load average: 0.90, 0.67, 0.51
- 13
- 14 #4. 在第3个终端运行mpstat查看CPU使用率变化情况
- 15 ##可以看到CPU1使用率达到了100%，但是iowait只有0，这说明系统负载变高正是由于CPU使用率100%导致的
- 16 [root@linux ~]# mpstat -P ALL 5

```

17 Linux 3.10.0-957.el7.x86_64 (linux)      2021年04月05日  _x86_64_      (1
   CPU)
18
19 15时59分32秒 CPU    %usr  %nice    %sys %iowait    %irq  %soft  %steal
   %guest  %gnice   %idle
20 15时59分37秒 all  100.00    0.00    0.00    0.00    0.00    0.00    0.00
   0.00    0.00    0.00
21 15时59分37秒  0  100.00    0.00    0.00    0.00    0.00    0.00    0.00
   0.00    0.00    0.00
22
23 #5. 在第4个终端使用top命令查看CPU使用率
24 ##可以看到stress进程的CPU使用率达到了100%
25 [root@linux ~]# top
26 Tasks: 104 total,  2 running, 102 sleeping,   0 stopped,   0 zombie
27 %Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
   st
28 KiB Mem : 2028088 total, 1590216 free,   101924 used,   335948 buff/cache
29 KiB Swap:      0 total,      0 free,      0 used. 1735256 avail Mem
30
31      PID USER      PR  NI   VIRT   RES    SHR S  %CPU  %MEM    TIME+  COMMAND
32   18743 root      20   0   7312    100      0 R   99.7   0.0    6:55.68 stress
33   18712 root      20   0      0      0      0 S    0.3   0.0    0:00.06
   kworker/0:2
34      1 root      20   0 43528   3848   2580 S    0.0   0.2    0:03.66 systemd
35      2 root      20   0      0      0      0 S    0.0   0.0    0:00.01 kthreadd

```

第3章 内存监控

free 查看内存使用状态

命令作用：

1 查看Linux系统的内存使用情况

常用参数：

```

1 free [选项]
2
3 -b    以b为单位输出
4 -k    以k为单位输出
5 -m    以m为单位输出
6 -g    以g为单位输出
7 -h    以人类可读友好的单位输出
8 -s N  间隔N秒输出一次
9 -c N  输出N次后退出

```

运行结果：

```

1 #默认以b为单位输出
2 [root@linux ~]# free
3              total        used        free        shared  buff/cache
   available

```

4	Mem:	2028088	85188	1807080	9720	135820
	1783376					
5	Swap:	0	0	0		
6						
7	#以m为单位输出					
8	[root@linux ~]# free -m					
9		total	used	free	shared	buff/cache
	available					
10	Mem:	1980	82	1764	9	132
	1741					
11	Swap:	0	0	0		
12						
13	#以人类可读的单位输出					
14	[root@linux ~]# free -h					
15		total	used	free	shared	buff/cache
	available					
16	Mem:	1.9G	82M	1.7G	9.5M	132M
	1.7G					
17	Swap:	0B	0B	0B		

状态解释:

1	man free	#free命令帮助手册
2	total	#总计已安装的内存,统计信息来自/proc/meminfo的MemTotal和SwapTotal字段
3	used	#已使用内存,计算公式为: $used = total - free - buffers - cache$
4	free	#空闲的未使用内存,计算公式为: $total - used - buff - cache$
5	shared	#共享内存,主要由tmpfs使用
6		
7	buff/cache	#buffers和cache的总和
8	buffers	#缓冲区,写缓冲,目的是加快内存和硬盘之间的数据写入。存放内存需要写入到磁盘的数据。
9	cache	#缓存区,读缓存,目的是加快CPU和内存交换数据。存放的是内存已经读完的数据。
10		
11	available	#估计有多少内存可用于启动新应用程序而无需交换。与cache或free字段提供的数据不同,此字段考虑了页面缓存。
12	swap	#硬盘交换分区,目的是防止内存使用完了导致系统崩溃,临时将硬盘的一部分空间当作内存使用。

buff/cache解释:

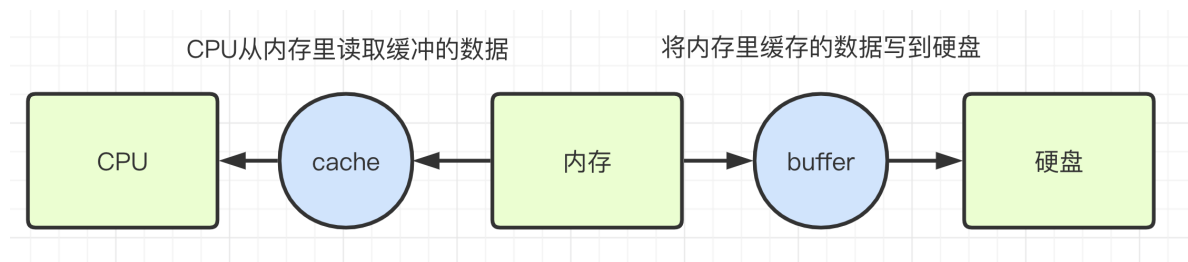
cache:

1	内存速度很快,但是CPU速度更快,所以为了提高CPU和内存之间交换数据的效率,设计了cache缓存这种技术。
2	CPU本身就有缓存,就是我们常听说的1级缓存2级缓存3级缓存,但是CPU内部的缓存成本太昂贵,所以设计的容量都非常小,那么就利用内存的部分空间来缓存CPU已经读过的数据,这样CPU下次访问只需要从cache里直接取就行了,不用再访问内存里的实际存储位置。

buffers:

- 1 | 内存速度很快，但是硬盘速度很慢，所以为了提高硬盘的写入效率，设计了**buffer**。
- 2 | **buffer**主要作用是将内存写完的数据缓存起来，然后通过系统调度策略在合适的时机再定期刷新到磁盘上。
- 3 | 这样可以减少磁盘的寻址次数，以提高写入性能。

示意图：



第4章 磁盘监控

iostat

1 |

iotop

命令作用：

- 1 | **iotop**用于监控磁盘的I/O使用情况，包含读和写的速率，IO百分比等信息，类似于**top**命令。

常用参数：

- 1 | **-o** #只显示正在进行io操作的进程，也可以先在**iotop**命令运行中按**o**
- 2 | **-b** #非交互模式，比较适合监控项取值或者脚本定时输出内容
- 3 | **-n N** #默认非交互模式会一直输出信息，**-n**选项可以控制输出次数，常结合**-b**参数一起使用
- 4 | **-d N** #设置每次数据更新的间隔，默认是1秒
- 5 | **-p PID** #监控指定PID的进程
- 6 | **-u USER** #监控指定用户运行程序产生的IO
- 7 | **-q** #禁止输出头几行，一般用于非交互模式
- 8 | **-qq** #不显示列名
- 9 | **-qqq** #不显示IO的总揽

交互式按键：

- 1 | 左/右键 #切换选中的列，并且按照选中的列进行排序
- 2 | **p** #默认显示的是线程TID，按**p**切换到进程PID
- 3 | **q** #退出

运行结果：

Total DISK READ :		0.00 B/s	Total DISK WRITE :		0.00 B/s		
Actual DISK READ:		0.00 B/s	Actual DISK WRITE:		0.00 B/s		
TID	PRIO	USER	DISK READ	DISK WRITE	SWAPIN	IO>	COMMAND
8912	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.03 %	[kworker/0:0]
1	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	systemd --s~serialize 22
2	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[kthreadd]
3	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[ksoftirqd/0]
5	be/0	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[kworker/0:0H]
6	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[kworker/u256:0]
7	rt/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[migration/0]
8	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[rcu_bh]
9	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[rcu_sched]
10	be/0	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[lru-add-drain]
11	rt/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[watchdog/0]
13	be/4	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[kdevtmpfs]
14	be/0	root	0.00 B/s	0.00 B/s	0.00 %	0.00 %	[netns]

状态解释：

1	#第一行 磁盘读写的速率总计
2	Total DISK READ :
3	Total DISK WRITE :
4	
5	#第二行 磁盘读写的实际速率
6	Actual DISK READ:
7	Actual DISK WRITE:
8	
9	#第三行 具体的进程速率信息
10	TID #线程ID, 按p切换为进程PID
11	PRIO #优先级
12	USER #运行用户
13	DISK READ #磁盘读的速率
14	DISK WRITE #磁盘写的速率
15	SWAPIN #swap交换分区百分比
16	IO #IO等待百分比
17	COMMAND #运行的进程

案例演示：

1	[root@linux opt]# iotop -b -n 1 -a -qqq head -n 5
2	1 be/4 root 0.00 B 0.00 B 0.00 % 0.00 % systemd --switched-root --system --deserialize 22
3	2 be/4 root 0.00 B 0.00 B 0.00 % 0.00 % [kthreadd]
4	3 be/4 root 0.00 B 0.00 B 0.00 % 0.00 % [ksoftirqd/0]
5	5 be/0 root 0.00 B 0.00 B 0.00 % 0.00 % [kworker/0:0H]
6	6 be/4 root 0.00 B 0.00 B 0.00 % 0.00 % [kworker/u256:0]

第5章 网络监控

iftop

命令作用：

1	实时显示网络流量，类似与top和iotop
---	-----------------------

常用参数：

```
1 | -h          #帮助说明
2 | -B          #以bytes为单位显示
3 | -i          #指定网卡
4 | -n          #不解析DNS
```

交互式按键：

```
1 | b          #显示进度条
2 | n          #打开或关闭DNS解析名称
3 | t          #切换流量显示模式
4 | q          #退出
```

运行结果：

Bars on	1.56KB	3.12KB	4.69KB	6.25KB	7.81KB
10.0.0.100	=> 10.0.0.1		340B	340B	340B
	<=		126B	64B	64B
10.0.0.100	=> 203.107.6.88		38B	38B	38B
	<=		38B	38B	38B
10.0.0.100	=> 10.0.0.2		0B	16B	16B
	<=		0B	50B	50B
TX:	cum: 3.07KB	peak: 536B	rates: 378B	393B	393B
RX:	1.19KB	289B	164B	152B	152B
TOTAL:	4.26KB	825B	542B	545B	545B

状态解释：

```
1 | TX          transport发送的数据包速率
2 | RX          receive接收的数据包速率
3 | TOTAL       接收和发送的数据包汇总
```

案例演示：

```
1 | [root@linux ~]# iftop -B -n -i eth0
```