

Linux 云计算集群架构师

学神 IT 教育：从零基础到实战，从入门到精通！

版权声明：

本系列文档为《学神 IT 教育》内部使用教材和教案，只允许 VIP 学员个人使用，禁止私自传播。否则将取消其 VIP 资格，追究其法律责任，请知晓！

免责声明：

本课程设计目的只用于教学，切勿使用课程中的技术进行违法活动，学员利用课程中的技术进行违法活动，造成的后果与讲师本人及讲师所属机构无关。倡导维护网络安全人人有责，共同维护网络文明和谐。

联系方式：

学神 IT 教育官方网站: <http://www.xuegod.cn>

Linux 云计算架构师进阶学习群 QQ 群: 1072932914



学习顾问：小语老师

学习顾问：边边老师

学神微信公众号

微信扫码添加学习顾问微信，同时扫码关注学神公众号了解最新动态，获取更多学习资料及答疑就业服务！

第十章 Centos8-系统进程管理

本节所讲内容:

- 10.1 进程概述和 ps 查看进程工具
- 10.2 uptime 查看系统负载-top 动态管理进程
- 10.3 前后台进程切换-nice 进程优先级-实战 screen 后台执行命令

10.1 进程概述和 ps 管理进程

10.1.1 什么是进程?

进程: 是程序运行的过程, 动态, 有生命周期及运行状态, 是已启动的可执行程序的运行实例。

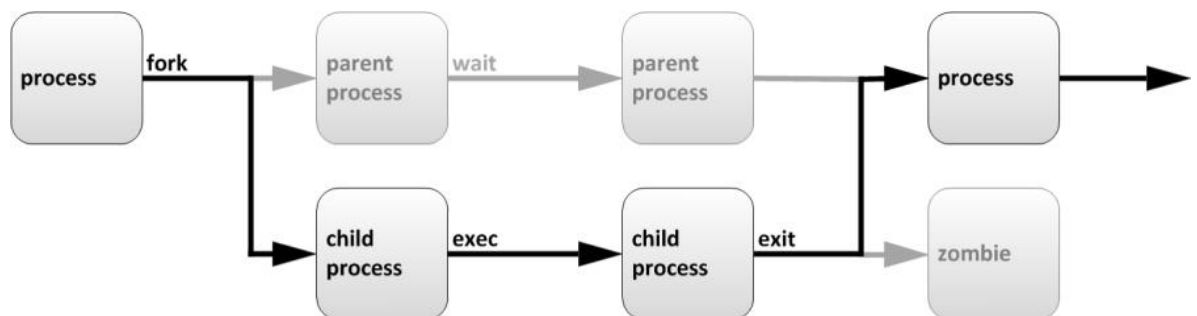
进程有以下组成部分:

- 已分配内存的地址空间;
- 安全属性, 包括所有权凭据和特权;
- 程序代码的一个或多个执行线程;
- 进程状态

线程: 进程和线程都是由操作系统所体现的程序运行的基本单元, 系统利用该基本单元实现系统对应用的并发性。进程和线程的区别在于: 简而言之, 一个程序至少有一个进程, 一个进程至少有一个线程。

程序: 二进制文件 (程序即二进制文件), 静态实体 /bin/date,/usr/sbin/sshd

下图所示的是进程的生命周期:



父进程复制自己的地址空间 (fork [fo: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

centos8 PID 为 1 的进程是: systemd

僵尸进程: 一个进程使用 fork 创建子进程, 如果子进程退出, 而父进程并没有调用 wait 或 waitpid 获取子进程的状态信息, 那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸进程。

父进程退出了, 子进程没有退出, 那么这些子进程就没有父进程来管理, 就变成僵尸进程。

10.1.2 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 (running)、休眠 S (sleep)、僵尸 Z (zombie)

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)

10.1.3 使用 ps 查看进程工具

1、ps 查看进程工具

例 1: 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

例 2: 常用的选项组合是 ps -aux

[root@xuegod63 ~]# ps -aux | more

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	0.1	191036	4028	?	Ss	20:21	0:03	/usr/lib/syste
root	2397	0.0	0.1	116692	3428	pts/0	Ss	20:25	0:00	bash
postfix	2954	0.0	0.1	91732	4024	?	S	20:27	0:00	pickup -l -t unix
root	2985	0.0	0.0	0	0	?	S	20:27	0:00	[kworker/3:0]
root	2994	1.2	0.0	0	0	?	S	20:28	0:15	[kworker/u128:0]
root	3451	0.0	0.0	0	0	?	S	20:33	0:00	[kworker/1:2]
root	3643	0.0	0.0	0	0	?	S	20:35	0:00	[kworker/u128:1]
root	3778	0.1	0.0	0	0	?	S	20:36	0:01	[kworker/0:1]

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 : 该程序目前正在运行, 或者是可被运行;

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 (僵尸) 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	4394	0.0	0.1	116692	3284	pts/1	Ss	20:52	0:00	bash
root	4435	0.0	0.2	151752	5292	pts/1	S+	20:52	0:00	vim a.txt
root	4437	0.0	0.1	116692	3280	pts/2	Ss+	20:52	0:00	bash
root	4542	0.0	0.0	0	0	?	S	20:56	0:00	[kworker/3:1]
root	4559	0.0	0.0	0	0	?	S	20:58	0:00	[kworker/0:0]
root	4567	0.0	0.0	107904	608	?	S	20:58	0:00	sleep 60
root	4568	0.0	0.0	151064	1828	pts/0	R+	20:59	0:00	ps - aux
root	4569	0.0	0.0	110436	964	pts/0	S+	20:59	0:00	more

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

例 1: 查看进程状态

```
[root@xuegod63 ~]# vim a.txt
```

在另一个终端执行:

```
[root@xuegod63 ~]# ps -aux | grep a.txt    #查看状态 S 表示睡眠状态, + 表示前台
root      4435  0.0  0.2 151752  5292 pts/1    S+   20:52   0:00 vim a.txt
root      4661  0.0  0.0 112676   996 pts/0    S+   21:05   0:00 grep --
```

color=auto a.txt

在 vim a.txt 这个终端上 按下: ctrl+z

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

在另一个终端执行:

```
[root@xuegod63 ~]# ps -aux | grep a.txt    #查看状态 T 表示停止状态
root      4435  0.0  0.2 151752  5292 pts/1    T    20:52   0:00 vim a.txt
root      4675  0.0  0.0 112676   996 pts/0    S+   21:05   0:00 grep --
```

color=auto 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) 前台进程收到这些信号, 就会采取相应动作。

ctrl-d 不是发送信号, 而是表示一个特殊的二进制值, 表示 EOF。代表输入完成或者注销在 shell 中, ctrl-d 表示退出当前 shell。

例 2: D 不可中断状态

```
[root@xuegod63 ~]# tar -zcvf 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@xuegod63 ~]# ps - axu | grep tar
root      1911  0.0  0.3 1279032 7184 ?        S<l  20: 23   0: 02 /usr/bin/pulseaudio
-- start
root      4746 14.0  0.0 123660  1480 pts/1    S+   21: 11   0: 01 tar - zcvf usr.tar.gz
/usr
root      4758  0.0  0.0 112680   984 pts/0    S+   21: 11   0: 00 grep -- color=auto tar
r
[ root@xuegod63 ~]# ps - axu | grep tar
root      1911  0.0  0.3 1279032 7184 ?        S<l  20: 23   0: 02 /usr/bin/pulseaudio
-- start
root      4746 14.8  0.0 123660  1480 pts/1    R+   21: 11   0: 02 tar - zcvf usr.tar.gz
/usr
root      4766  0.0  0.0 112680   984 pts/0    S+   21: 11   0: 00 grep -- color=auto tar
r
[ root@xuegod63 ~]# ps - axu | grep tar
root      1911  0.0  0.3 1279032 7184 ?        S<l  20: 23   0: 02 /usr/bin/pulseaudio
-- start
root      4746 15.7  0.0 123660  1480 pts/1    D+   21: 11   0: 02 tar - zcvf usr.tar.gz
/usr
root      4774  0.0  0.0 112680   984 pts/0    S+   21: 11   0: 00 grep -- color=auto tar
```

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, 程序被该 UID 所拥有

PID: 该进程的 ID

PPID: 该进程的父进程的 ID

C: 该进程生命周期中的 CPU 使用资源百分比

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 是本地上面的登录者程序, 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。

dd if=/dev/zero of=/zero.txt count=10 bs=100M

```
[root@localhost ~]# ps -axu | grep dd
```

注:

ps aux 是用 BSD 的格式来显示进程。

ps -ef 是用标准的 Unix 格式显示进程

10.2 uptime 查看系统负载-top 动态管理进程

10.2.1 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:

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	当前登录用户数
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?

大厅排队买票:

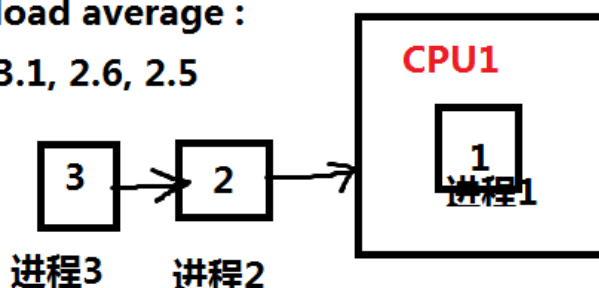


这时队列是 4:

cpu 队列数为 3 时, 如图:

load average :

3.1, 2.6, 2.5



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



10.2.2 top 命令

[root@xuegod63 ~]# 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)

```
Tasks: 481 total, 1 running, 480 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
```

当有多个 CPU 时, 这些内容可能会超过两行。内容如下:

Tasks: 481 total	进程总数
1 running	正在运行的进程数
480 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	<p>cpu 等待 I/O 完成的时间总量。</p> <p>测试:</p> <p>终端 1: 执行: top</p> <p>终端 2: dd if=/dev/zero of=/a.txt count=10 bs=100M</p> <p>终端 3: dd if=/dev/zero of=/a.txt count=10 bs=100M</p> <p>正常读写时, 如果 wa 占用较多 CPU, 那么就是磁盘性能问题, 建议更换磁盘。</p> <p>如下:</p> <pre> root@localhost ~ # top top - 10:33:34 up 9:05, 6 users, load average: 0.34, 0.16, 0.10 tasks: 174 total, 3 running, 171 sleeping, 0 stopped, 0 zombie Cpu(s): 0.7%us, 7.9%sy, 0.0%ni, 64.2%id, 26.5%wa, 0.0%hi, 0.7%si, 0.0%st Mem: 1164636k total, 1060824k used, 103812k free, 33536k buffers Swap: 1023992k total, 0k used, 1023992k free, 652672k cached </pre>
0.0% hi (了解) 硬中断消耗时间	<p>硬中断, 占用 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(Interrupt Request), 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)。# hi -> Hardware IRQ: The amount of time the CPU has been servicing hardware interrupts.</p>
0.0% si (了解) 软中断消耗时间	<p>软中断, 占用 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。# si -> Software Interrupts.: The amount of time the CPU has been servicing software interrupts.</p>

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 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 <pre>[root@localhost ~]# free -m</pre> <table><thead><tr><th></th><th>total</th><th>used</th><th>free</th><th>shared</th><th>buff/cache</th><th>available</th></tr></thead><tbody><tr><td>Mem:</td><td>3939</td><td>570</td><td>2656</td><td>9</td><td>711</td><td>3050</td></tr><tr><td>Swap: I</td><td>2047</td><td>0</td><td>2047</td><td></td><td></td><td></td></tr></tbody></table>		total	used	free	shared	buff/cache	available	Mem:	3939	570	2656	9	711	3050	Swap: I	2047	0	2047			
	total	used	free	shared	buff/cache	available																
Mem:	3939	570	2656	9	711	3050																
Swap: I	2047	0	2047																			
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: 如果 swap 分区, 被使用, 那么你的内存不够用了。

第 7 行进程信息

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
列名	含义										
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										
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)	<p>虚拟内存, 是进程正在使用的所有内存 (ps 中标为 VSZ)</p> <p>VIRT: virtual memory usage 虚拟内存</p>										

memory usage)	<p>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</p> <p>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</p>
RES (resident memory usage)	<p>是进程所使用的物理内存。实际实用内存 (ps 中标为 RSS)</p> <p>RES: resident memory usage 常驻内存</p> <p>1、进程当前使用的内存大小，但不包括 swap out</p> <p>2、包含其他进程的共享</p> <p>3、如果申请 100m 的内存，实际使用 10m，它只增长 10m，与 VIRT 相反</p> <p>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</p>
SHR	<p>共享内存大小，单位 kb</p> <p>SHR: shared memory 共享内存</p> <p>1、除了自身进程的共享内存，也包括其他进程的共享内存</p> <p>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</p> <p>3、计算某个进程所占的物理内存大小公式: $RES - SHR$</p> <p>4、swap out 后，它将会降下来</p>
S	<p>进程状态。</p> <p>D=不可中断的睡眠状态</p> <p>R=运行中或可运行</p> <p>S=睡眠中</p> <p>T=已跟踪/已停止</p> <p>Z=僵停</p>
%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

p: 进程 pid, 查看某个进程状态

数字键 1: 显示每个内核的 CPU 使用率，展示 cpu 数量，再按下，就收起来了

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

找到进程 PID

```
[root@localhost ~]# vim a.txt
```

```
[root@localhost ~]# ps axu | grep vim
```

Warning: bad syntax, perhaps a bogus '-'? See /usr/share/doc/procps-3.2.8/FAQ

```
root      9667  0.0  0.2 143620  3344 pts/1    S<+  19:15   0:00 vim a.txt
```

```
[root@localhost ~]# top -p 9667
```

10.2.3 实战 1: 找出系统中使用 CPU 最多的进程

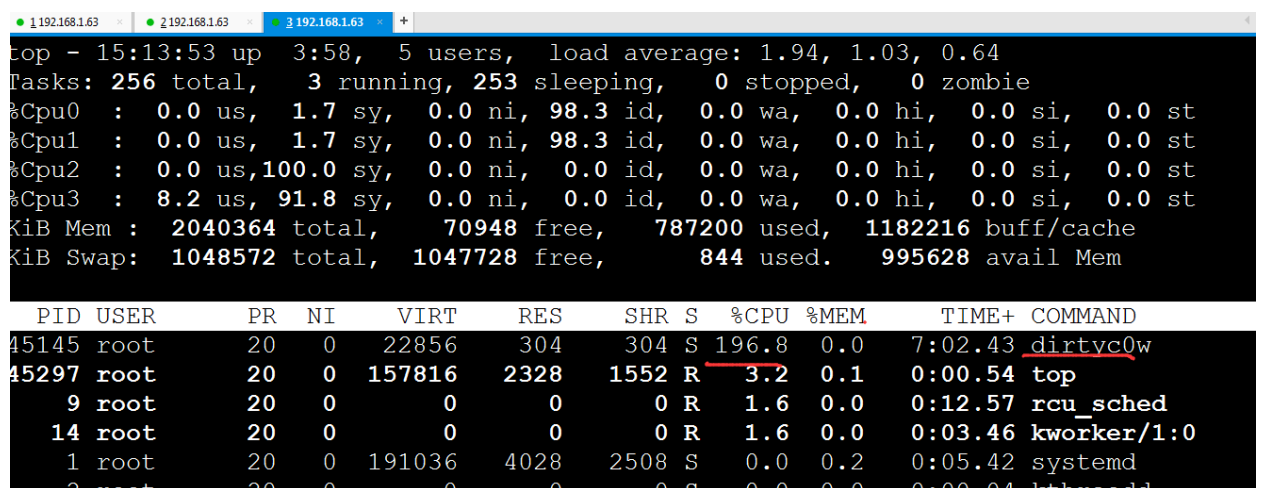
运行 top , 找出使用 CPU 最多的进程 , 按大写的 P, 可以按 CPU 使用率来排序显示

```
Swap:  1023992k total,        0k used,  1023992k free,   2/4516k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2276	root	20	0	163m	29m	7692	R	5.0	2.6	0:27.98	Xorg
3107	root	20	0	637m	15m	10m	S	2.3	1.4	0:05.12	gnome-terminal
3363	root	20	0	15088	1308	956	R	0.7	0.1	0:00.09	top

互动: 在 linux 系统中一个进程, 最多可以使用 100%cpu 对吗?

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



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
45145	root	20	0	22856	304	304	S	196.8	0.0	7:02.43	dirtycow
45297	root	20	0	157816	2328	1552	R	3.2	0.1	0:00.54	top
9	root	20	0	0	0	0	R	1.6	0.0	0:12.57	rcu_sched
14	root	20	0	0	0	0	R	1.6	0.0	0:03.46	kworker/1:0
1	root	20	0	191036	4028	2508	S	0.0	0.2	0:05.42	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.04	kthread

这是你第一次看见: 1

如果你的 4 核心的 cpu, 你可以运行 400%

```
Last login: Wed Jun  9 14:47:28 2021 from 172.19.250.56
(base) DingSX e[33;1m]16:16:43 ~
$stop
top - 16:16:56 up 5 days,  1:37, 15 users,  load average: 221.07, 399.37, 527.08
Tasks: 825 total,   8 running, 812 sleeping,   5 stopped,   0 zombie
%Cpu(s): 68.4 us, 31.5 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 21131493+total,  9159372 free, 56666412+used, 15373258+buff/cache
KiB Swap: 13841203+total, 13804825+free,  363776 used, 15445189+avail Mem
```

PID	USER	PR	NI	VR	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
113685	DingSX	20	0	108.6g	103.8g	11888	S	1024	5.2	10662:20	java
129885	DingSX	20	0	108.6g	103.8g	11896	S	978.8	5.2	9705:34	java
114302	DingSX	20	0	108.6g	103.8g	11888	S	846.9	5.2	10407:13	java
113537	DingSX	20	0	108.6g	103.8g	11884	S	805.9	5.2	10697:36	java
47602	ZhangYao	20	0	2418688	177980	2708	S	574.6	0.0	93:38.87	bowtie2-align-s
47754	ZhangYao	20	0	2419204	160552	2720	S	496.7	0.0	90:25.68	bowtie2-align-s
128720	xupeng2	20	0	72.5g	71.5g	1964	S	490.2	3.5	7252:10	hisat2-align-s
47935	ZhangYao	20	0	2418688	156116	2704	S	454.4	0.0	91:43.53	bowtie2-align-s
47295	ZhangYao	20	0	2419204	162036	2700	S	396.4	0.0	96:34.85	bowtie2-align-s
159041	WangWL	20	0	16.6g	3.5g	188272	S	116.9	0.2	1218:18	MATLAB
52829	WangWL	20	0	19.6g	1.1g	194072	S	66.8	0.1	0:44.32	MATLAB
46367	xupeng2	20	0	304276	289404	1500	R	23.1	0.0	6:37.27	bedtools
32992	xupeng2	20	0	223944	209072	1500	R	20.5	0.0	59:37.66	bedtools
54646	root	20	0	0	0	0	R	18.9	0.0	0:00.59	kworker/u580:0
31825	xupeng2	20	0	194100	179232	1504	R	18.6	0.0	65:41.14	bedtools
46332	xupeng2	20	0	304276	289404	1500	R	17.9	0.0	6:55.84	bedtools
112022	root	20	0	0	0	0	R	13.4	0.0	0:34.13	kworker/33:0
146120	HuangYX	20	0	67.4g	4.2g	8992	S	7.5	0.2	814:48.08	ipython
42665	root	20	0	0	0	0	S	4.6	0.0	4:28.76	kworker/2:1
49670	root	0	-20	0	0	0	S	3.3	0.0	3:50.22	kworker/2:0H
50412	ZhangWZ	20	0	31.9g	1.4g	222228	S	2.3	0.1	2:05.36	MATLAB
49418	root	20	0	0	0	0	S	1.3	0.0	0:05.76	kworker/u580:2

64 核心的 cpu, 可以运行到 6400%

10.2.4 lsof 命令 (netstat -Input 自行拓展)

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。(ipv4、ipv6、协议、:端口、@ip)

-p<进程号>: 列出指定进程号所打开的文件;

例:

```
[root@xuegod63 ~]# vim a.txt
```

```
[root@xuegod63 ~]# 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
```

```
[root@xuegod63 ~]# yum -y install lsof
```

```
[root@xuegod63 ~]# lsof -p 65641    #进程 pid 一般用于查看木马进程, 在读哪些文件
```

```
[root@xuegod63 ~]# lsof -i :22      #查看端口, 或查看黑客开启的后门端口是哪个进程在用
```

```
[root@xuegod63 ~]# lsof -c vim      #进程名, 显示 vim 进程现在打开的文件
```

```
[root@xuegod63 ~]# lsof /test/.abc.txt.swp    #显示占用文件.abc.txt.swp 的进程
```

```
[root@xuegod63 ~]# umount /test
umount: /test: 目标忙。
(有些情况下通过 lsof(8) 或 fuser(1) 可以
  找到有关使用该设备的进程的有用信息)
[root@xuegod63 ~]# lsof /test
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
bash      7298 root    cwd   DIR   8,17    26     64 /test
vim       7371 root    cwd   DIR   8,17    26     64 /test
vim       7371 root    3u    REG   8,17  12288    67 /test/.abc.txt.swp
```

在这个示例中, 用户 root 正在其/test 目录中进行一些操作。一个 bash 是实例正在运行, 并且它当前的目录为/test, 另一个则显示的是 vim 正在编辑/test 下的文件。要成功地卸载/test, 应该在通知用户以确保情况正常之后, 中止这些进程。这个示例说明了应用程序的当前工作目录非常重要, 因为它仍保持着文件资源, 并且可以防止文件系统被卸载。这就是为什么大部分守护进程(后台进程) 将它们的目录更改为根目录、或服务特定的目录的原因, 以避免该守护进程阻止卸载不相关的文件系统。

10.2.5 free 显示系统中可用内存和已用内存的数量

free 命令查看内存使用状态

子选项:

-b: 以字节为单位表示。

-k: 以 KB 为单位显示, 默认是以 KB 为单位显示。

-m: 以 MB 为单位显示。

-g: 以 GB 为单位显示。

```
[root@localhost ~]# free -m
```

	total	used	free	shared	buff/cache	available
Mem: 972	972	603	69	24	299	123
Swap: 2047		69	1978			

其中:

第一行: total 是总内存量, used 是已经使用的内存量, free 是空闲的内存, shared 是多个进程共享的内存总数, buffers 是缓冲内存数, cache 是缓存内存数。默认单位是 KB。available 实际可用

(case 加速读, buffers 加速写。)

第二行开始: total 系统中有 972MB 的物理内存, used 是已经使用的内存数量。free 是空闲的内存数量。shared 是多个进程共享的内存数量。buff/cache 用来作为缓冲和缓存的空间, 内核会在内存将要耗尽时释放这部分内存给其他进程使用。available: 可使用空间, 评估有多少内存可用于启动新应用程序, 不包括 swap, 不同于 free 和 cache 字段。available 字段考虑了页缓存, 而不是所有可回收的内存。正因为这个原因所以通常 free+buff/cache 的数值要比 available 的数值大。

互动: 执行 free 命令查看系统状态, 这一瞬间, 当前系统, 真正, 还有多少 M 内存可以使用?

答案 1: free+ buff/cache=69 +299=368M

答案 2: available=123M

10.3 前后台进程切换- nice 进程优先级-实战 screen 后台执行命令

令

10.3.1 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,它不需要终端的交互;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。

10.3.2 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进程.它是一个作业控制命令
fg	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
(foreground process)	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

```
[root@xuegod63 ~]# vim a.txt      #打开后,然后执行 ctrl+z
```

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

```
[root@xuegod63 ~]# ps -axu | grep vim
```

```
root    43710  0.8  0.2 151744  5304 pts/3    T   18:26   0:00 vim a.txt
```

```
root    43720  0.0  0.0 112676   984 pts/3    S+  18:26   0:00 grep --color=auto
```

vim

```
[root@xuegod63 ~]# jobs          #查看当前有多少在后台运行的进程
```

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

```
[root@xuegod63 ~]# fg 1          #将后台挂起的进程恢复到前台运行
```

10.3.3 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 ctrl+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 ctrl+z |

例 1: kill 和 killall 终止进程

```
[root@xuegod63 ~]# yum -y install psmisc
[root@xuegod63 ~]# kill -9 pid
[root@xuegod63 ~]# killall vim
[root@xuegod63 ~]# pkill vim
```

10.3.4 进程的优先级管理

优先级取值范围为 (-20,19), 值越小优先级越高, 默认优先级是 0

优先级越高占用的 CPU 值就越高

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@xuegod63 ~]# nice -n 5 vim a.txt
```

输入内容, 然后 ctrl+z 挂起

```
[root@xuegod72 ~]# nice -n 5 vim a.txt
[1]+  Stopped                  nice -n 5 vim a.txt
```

通过 ps 查看这个文件的 PID 号

```
[root@xuegod63 ~]# ps -aux|grep vim
```

```
[root@xuegod72 ~]# ps -aux|grep vim
root      26154  0.0  0.1 151224  4952 pts/1    TN   18:28   0:00 vim a.txt
root      26270  0.0  0.0 112644   956 pts/1    S+   18:30   0:00 grep --color=auto vim
```

通过 top 命令查看优先级

[root@xuegod63 ~]# top -p 26154

```
[root@xuegod72 ~]# top -p 26154
top - 18:31:27 up 1 day, 5:21, 4 users, load average: 0.00, 0.01, 0.05
Tasks: 1 total, 0 running, 0 sleeping, 1 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.1 sy, 0.0 ni, 99.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 2999960 total, 573612 free, 957084 used, 1469264 buff/cache
KiB Swap: 3071996 total, 3071996 free, 0 used. 1763920 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
26154	root	25	5	151224	4952	2576	T	0.0	0.2	0:00.06	vim

改变正在运行的进程的优先级

```
[root@xuegod72 ~]# renice -10 26154
26154 (process ID) old priority 5, new priority -10
[root@xuegod72 ~]# top -p 26154
top - 18:34:52 up 1 day, 5:25, 4 users, load average: 0.00, 0.01, 0.05
Tasks: 1 total, 0 running, 0 sleeping, 1 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 : 2999960 total, 573736 free, 956960 used, 1469264 buff/cache
KiB Swap: 3071996 total, 3071996 free, 0 used. 1764044 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
26154	root	10	-10	151224	4952	2576	T	0.0	0.2	0:00.06	vim

```
7200 (进程 ID) 旧优先级为 5, 新优先级为 0
[root@xuegod63 ~]# ps aux | grep vim
root      7200  0.0  0.5 149688 5532 pts/0    S+   09:51   0:00 vim a.txt
root      7266  0.0  0.0 112724  988 pts/1    S+   10:04   0:00 grep --color=auto vim
[root@xuegod63 ~]# renice 1 7200
7200 (进程 ID) 旧优先级为 0, 新优先级为 1
[root@xuegod63 ~]# ps aux | grep vim
root      7200  0.0  0.5 149688 5532 pts/0    SN+  09:51   0:00 vim a.txt
root      7269  0.0  0.0 112724  988 pts/1    S+   10:04   0:00 grep --color=auto vim
[root@xuegod63 ~]# renice -1 7200
7200 (进程 ID) 旧优先级为 1, 新优先级为 -1
[root@xuegod63 ~]# ps aux | grep vim
root      7200  0.0  0.5 149688 5532 pts/0    S+   09:51   0:00 vim a.txt
root      7272  0.0  0.0 112724  988 pts/1    S+   10:04   0:00 grep --color=auto vim
```

10.3.5 实战: 使用 screen 后台实时执行备份命令

实战场景: 公司晚上需要备份 1T 数据, 我在 xshell 上直接执行备份脚本 back.sh 可以吗? 或直接运行 back.sh & 放到后台运行可以吗? 当关了 xshell 后, back.sh & 还在后台执行吗?

答: xshell 长时间连接, 如果本地网络偶尔断开或 xshell 不小心关闭, 都会让后台运行的备份命令停止运行的。正确做法使用: screen

10.3.6 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。

安装 screen 软件包

```
# rpm -ivh /mnt/Packages/screen-4.1.0-0.23.20120314git3c2946.el7_2.x86_64.rpm
```

或者

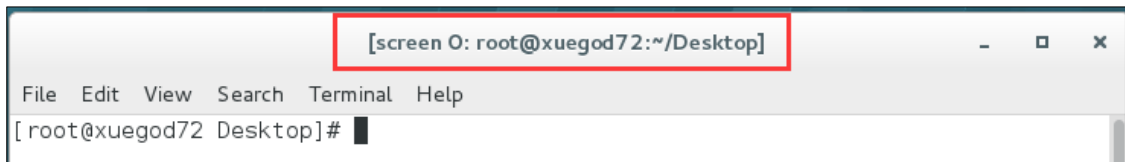
```
[root@xuegod63 ~]# yum -y install screen
```

10.3.7 screen 使用方法

直接在命令行键入 screen 命令回车, 如下图

```
[root@xuegod63 ~]# 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 窗口中那样



例如, 我们在做某个大型的操作但是突然之间断开:

实战: 使用 screen 后台实时执行命令备份命令

[root@xuegod63 ~]# screen #进入

[root@xuegod63 ~]# vim a.txt #执行命令, 或执行你自己需要运行的备份命令

此时想离开一段时间, 但还想让这个命令继续运行

[root@xuegod63 ~]# #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

detached [dɪˈtætʃt] 分离, 独立

半个小时之后回来了, 找到该 screen 会话:

[root@tivf06 ~]# screen -ls #查看已经建立的会话 ID

There is a screen on:

44074.pts-1.tivf06 (Detached)

1 Socket in /tmp/screens/S-root.

重新连接会话:

[root@xuegod63 ~]# screen -r 44074

root@xuegod63 ~]# exit #不想使用 screen 会话了, 执行: exit 退出。

附: 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

screen -S 会话 id -X quit -> 删除会话

screen -S 会话名 -X quit -> 删除会话

总结:

10.1 进程概述和 ps 查看进程工具

10.2 uptime 查看系统负载-top 动态管理进程

10.3 前后台进程切换- nice 进程优先级-实战 screen 后台执行命令