

目录

一：重要软件下载地址.....	2
二：gcc /g++命令重要参数	2
三：信号相关.....	2
四：linux top 命令查看内存及多核 CPU 的使用讲述	4
五：TCP/IP 协议的配置选项	11
六：TCP/IP 协议额外注意的一些算法、概念等	13

一：重要软件下载地址

链接：<https://pan.baidu.com/s/147TP-jTHad3-Trfx1wC-Rg>
提取码：46yn

二： gcc /g++命令重要参数

(1)-o: 指定编译链接后生成的可执行文件名，比如

gcc -o nginx nginx.c

(2)-c: 将.c 编译成.o 目标文件[仅执行编译操作，不进行链接操作]

gcc -c nginx.c

将生成 nginx.o 的目标文件

(3)-M: 显示一个源文件所依赖的各种文件

gcc -M nginx.c

(4)-MM: 显示一个源文件所依赖的各种文件，但不包括系统的一些头文件；

gcc -MM nginx.c

这种扫描是有用途的，尤其是在写 makefile 文件时，需要用到这些依赖关系，以做到比如当某个.h 头文件更改时，整个工程会自动重新编译的目的；

```
kuangxiang@bogon:/mnt/hgfs/linux/nginx$ ls
Makefile  misc  net  nginx  nginx.c  nginx.o  ngx_conf.c  ngx_conf.o  ngx_func.h  proc  signal
kuangxiang@bogon:/mnt/hgfs/linux/nginx$ gcc -MM nginx.c
nginx.o: nginx.c ngx_func.h
kuangxiang@bogon:/mnt/hgfs/linux/nginx$
```

(5) -g: 生成调试信息。GNU 调试器可利用该信息。

(6)-I: gcc 会先到你 用这个参数指定的目录去查找头文件，你在.c/.cpp 中可以

#include <abc.h> //这里用尖括号了

gcc -I /mnt/mydir

三：信号相关

kill 命令不同数字所能发出的不同信号

kill 的参数	该参数发出的信号	操作系统缺省动作
-1	SIGHUP（连接断开）	终止掉进程（进程没了）
-2	SIGINT（终端中断符，比如 ctrl+c）	终止掉进程（进程没了）
-3	SIGQUIT（终端退出符，比如 ctrl+\）	终止掉进程（进程没了）
-9	SIGKILL（终止）	终止掉进程（进程没了）
-18	SIGCONT（使暂停的进程继续）	忽略（进程依旧在运行不受影响）
-19	SIGSTOP（停止），可用 SIGCONT 继续，	停止进程（不是终止，进程还在）

	但任务被放到了后台	
-20	SIGTSTP（终端停止符，比如 ctrl+z），但任务被放到了后台，可用 SIGCONT 继续	停止进程（不是终止，进程还在）

进程状态：

状态	含义
D	不可中断的休眠状态(通常是 I/O 的进程)，可以处理信号，有 延迟
R	可执行状态&运行状态(在运行队列里的状态)
S	可中断的休眠状态之中（等待某事件完成），可以处理信号
T	停止或被追踪（被作业控制信号所停止）
Z	僵尸进程
X	死掉的进程
<	高优先级的进程
N	低优先级的进程
L	有些页被锁进内存
s	Session leader（进程的领导者），在它下面有子进程
t	追踪期间被调试器所停止
+	位于前台的进程组

常用信号列举：

信号名	信号含义
SIGHUP（连接断开）	是终端断开信号，如果终端接口检测到一个连接断开，发送此信号到该终端所在的会话首进程（前面讲过），缺省动作会导致所有相关的进程退出(上节课也重点讲了这个信号，xshell 断开就有这个信号送过来)； Kill -1 进程号也能发送此信号给进程；
SIGALRM（定时器超时）	一般调用系统函数 alarm 创建定时器，定时器超时了就会这个信号；
SIGINT（中断）	从键盘上输入 ctrl+C（中断键）【比如你进程正跑着循环干一个事】，这一 ctrl+C 就能打断你干的事，终止进程； 但 shell 会将后台进程对该信号的处理设置为忽略（也就是说该进程若在后台运行则不会收到该信号）；
SIGSEGV（无效内存）	内存访问异常，除数为 0 等，硬件会检测到并通知内核；其实这个 SEGV 代表段违例（segmentation violation），你有的时候运行一个你编译出来的可执行的 c 程序，如果内存有问题，执行的时候就会出现这个提示；
SIGIO（异步 I/O）	通用异步 I/O 信号，咱们以后学通讯的时候，如果通讯套接口上有数据到达，或发生一些异步错误，内核就会通知我们这个信号；
SIGCHLD（子进程改变）	一个进程终止或者停止时，这个信号会被发送给父进程；（我们想象下 nginx，worker 进程终止时 master 进程应该会收到内核发出的针对该信号的通知）；

SIGUSR1,SIGUSR2（都是用户定义信号）	用户定义的信号，可用于应用程序，用到再说；
SIGTERM（终止）	一般你通过在命令行上输入 <code>kill</code> 命令来杀一个进程的时候就会触发这个信号，收到这个信号后，你有机会退出前的处理，实现这种所谓 优雅退出 的效果；
SIGKILL（终止）	不能被忽略，这是杀死任意进程的可靠方法，不能被进程本身捕捉
SIGSTOP（停止）	不能被忽略，使进程停止运行，可以用 <code>SIGCONT</code> 继续运行，但进程被放入到了后台
SIGQUIT（终端退出符）	从键盘上按 <code>ctrl+\</code> 但 <code>shell</code> 会将后台进程对该信号的处理设置为忽略（也就是说该进程若在后台运行则不会收到该信号）；
SIGCONT（使暂停进程继续）	使暂停的进程继续运行
SIGTSTP（终端停止符）	从键盘上按 <code>ctrl+z</code> ，进程被停止，并被放入后台，可以用 <code>SIGCONT</code> 继续运行

四：linux top 命令查看内存及多核 CPU 的使用讲述

老师给大家推荐一篇网络上的文章，供大家参考，对 `top` 命令有一个进一步的了解：
<https://www.cnblogs.com/dragonsuc/p/5512797.html>

查看多核 CPU 命令

`mpstat -P ALL` 和 `sar -P ALL`

说明：`sar -P ALL > aaa.txt` 重定向输出内容到文件 `aaa.txt`

top 命令

经常用来监控 linux 的系统状况，比如 `cpu`、内存的使用，程序员基本都知道这个命令，但比较奇怪的是能用好它的人却很少，例如 `top` 监控视图中内存数值的含义就有不少的曲解。

本文通过一个运行中的 WEB 服务器的 `top` 监控截图，讲述 `top` 视图中的各种数据的含义，还包括视图中各进程（任务）的字段的排序。

top 进入视图

```
27 root      20   0   0   0   0 S   0.0   0.0   0:00.00 pm
[root@fronts-cn-xuchang-center ~]# top
top - 10:08:45 up 10 days, 3:05, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 135 total, 1 running, 134 sleeping, 0 stopped, 0 zombie
Cpu0  :  0.3%us,  0.0%sy,  0.0%ni, 99.3%id,  0.3%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.3%us,  0.3%sy,  0.0%ni, 97.7%id,  1.7%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.3%us,  0.0%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  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:   3808060k total, 3660048k used, 148012k free, 359760k buffers
Swap:  4184924k total,      0k used, 4184924k free, 2483956k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 27623 root      20   0 3473m 557m 16m S   1.0 15.0   61:59.91 java
 11746 root      20   0 15036 1248  936 R   0.3  0.0    0:00.52 top
      1 root      20   0 19232 1568 1280 S   0.0  0.0    0:03.27 init
      2 root      20   0      0      0      0 S   0.0  0.0    0:00.43 kthreadd
      3 root      RT    0      0      0      0 S   0.0  0.0    0:00.16 migration/0
      4 root      20   0      0      0      0 S   0.0  0.0    0:02.98 ksoftirqd/0
      5 root      RT    0      0      0      0 S   0.0  0.0    0:00.00 migration/0
      6 root      RT    0      0      0      0 S   0.0  0.0    0:01.32 watchdog/0
      7 root      RT    0      0      0      0 S   0.0  0.0    0:00.16 migration/1
      8 root      RT    0      0      0      0 S   0.0  0.0    0:00.00 migration/1
      9 root      20   0      0      0      0 S   0.0  0.0    0:02.75 ksoftirqd/1
     10 root      RT    0      0      0      0 S   0.0  0.0    0:01.19 watchdog/1
```

第一行：

- 10:08:45 — 当前系统时间
- 10 days, 3:05 — 系统已经运行了 10 天 3 小时 5 分钟（在这期间没有重启过）
- 1 users — 当前有 1 个用户登录系统
- load average: 0.00, 0.00, 0.00 — load average 后面的三个数分别是 1 分钟、5 分钟、15 分钟的负载情况。

load average 数据是每隔 5 秒钟检查一次活跃的进程数，然后按特定算法计算出的数值。如果这个数除以逻辑 CPU 的数量，结果高于 5 的时候就表明系统在超负荷运转了。

第二行：

Tasks — 任务（进程），系统现在共有 135 个进程，其中处于运行中的有 1 个，134 个在休眠（sleep），stoped 状态的有 0 个，zombie 状态（僵尸）的有 0 个。

第三行：cpu 状态

- 0.3% 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
- 99.7% id — 空闲 CPU 百分比
- 0.0% wa — IO 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

在这里 CPU 的使用比率和 windows 概念不同，如果你不理解用户空间和内核空间，需要充充电了。

第四行：内存状态

- 3808060k total — 物理内存总量（4GB）
- 3660048k used — 使用中的内存总量（3.6GB）

148012k free — 空闲内存总量（148M）

359760k buffers — 缓存的内存量（359M）

第五行：swap 交换分区

4184924k total — 交换区总量（4G）

0k used — 使用的交换区总量（0M）

4184924k free — 空闲交换区总量（4G）

2483956k cached — 缓冲的交换区总量（2483M）

第四行中使用中的内存总量（used）指的是现在系统内核控制的内存数，空闲内存总量（free）是内核还未纳入其管控范围的数量。纳入内核管理的内存不见得都在使用中，还包括过去使用过的现在可以被重复利用的内存，内核并不把这些可被重新使用的内存交还到 free 中去，因此在 linux 上 free 内存会越来越少，但不用为此担心。

如果出于习惯去计算可用内存数，这里有个近似的计算公式：第四行的 free + 第四行的 buffers + 第五行的 cached，按这个公式此台服务器的可用内存：

148M+259M+2483M = 2990M。

对于内存监控，在 top 里我们要时刻监控第五行 swap 交换分区的 used，如果这个数值在不断的变化，说明内核在不断进行内存和 swap 的数据交换，这是真正的内存不够用了。

第六行是空行

第七行以下：各进程（任务）的状态监控

PID — 进程 id

USER — 进程所有者

PR — 进程优先级

NI — nice 值。负值表示高优先级，正值表示低优先级

VIRT — 进程使用的虚拟内存总量，单位 kb。VIRT=SWAP+RES

RES — 进程使用的、未被换出的物理内存大小，单位 kb。RES=CODE+DATA

SHR — 共享内存大小，单位 kb

S — 进程状态。D=不可中断的睡眠状态 R=运行 S=睡眠 T=跟踪/停止 Z=僵尸进程

%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 — 进程名称（命令名/命令行）

多 U 多核 CPU 监控

在 top 基本视图中，按键盘数字“1”，可监控每个逻辑 CPU 的状况：

```
top - 18:52:48 up 10 days, 3:29, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 135 total, 1 running, 134 sleeping, 0 stopped, 0 zombie
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.3%us,  0.0%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  0.0%sy,  0.0%ni, 98.7%id,  1.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3808060k total, 3625280k used, 182780k free, 360044k buffers
Swap:  4184924k total,      0k used, 4184924k free, 2484648k cached
```

观察上图，服务器有 4 个逻辑 CPU，实际上是 1 个物理 CPU。

如果不按 1，则在 top 视图里面显示的是所有 cpu 的平均值。

进程字段排序

默认进入 top 时，各进程是按照 CPU 的占用量来排序的，在【top 视图 01】中进程 ID 为 14210 的 java 进程排在第一（cpu 占用 100%），进程 ID 为 14183 的 java 进程排在第二（cpu 占用 12%）。可通过键盘指令来改变排序字段，比如想监控哪个进程占用 MEM 最多，我一般的使用方法如下：

1. 敲击键盘“b”（打开/关闭加亮效果），top 的视图变化如下：

```
top - 10:41:15 up 10 days, 3:37, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 135 total, 1 running, 134 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.3%us, 0.3%sy, 0.0%ni, 99.4%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 3808060k total, 3610328k used, 197732k free, 360192k buffers
Swap: 4184924k total, 0k used, 4184924k free, 2484852k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  CODE  DATA  COMMAND
12363 root        20   0 15036 1248  936 R  1.2   0.0   0:00.36   56   496 top
27623 root        20   0 3469m 503m  11m S  1.2  13.5 62:22.47    4  3.2g java
   1 root        20   0 19232 1568 1280 S  0.0   0.0   0:03.27  140   288 init
   2 root        20   0     0     0     0 S  0.0   0.0   0:00.43    0     0 kthreadd
   3 root        RT    0     0     0     0 S  0.0   0.0   0:00.16    0     0 migration/0
   4 root        20   0     0     0     0 S  0.0   0.0   0:02.99    0     0 ksoftirqd/0
   5 root        RT    0     0     0     0 S  0.0   0.0   0:00.00    0     0 migration/0
   6 root        RT    0     0     0     0 S  0.0   0.0   0:01.32    0     0 watchdog/0
   7 root        RT    0     0     0     0 S  0.0   0.0   0:00.16    0     0 migration/1
   8 root        RT    0     0     0     0 S  0.0   0.0   0:00.00    0     0 migration/1
   9 root        20   0     0     0     0 S  0.0   0.0   0:02.76    0     0 ksoftirqd/1
  10 root        RT    0     0     0     0 S  0.0   0.0   0:01.20    0     0 watchdog/1
  11 root        RT    0     0     0     0 S  0.0   0.0   0:00.18    0     0 migration/2
  12 root        RT    0     0     0     0 S  0.0   0.0   0:00.00    0     0 migration/2
  13 root        20   0     0     0     0 S  0.0   0.0   0:02.09    0     0 ksoftirqd/2
  14 root        RT    0     0     0     0 S  0.0   0.0   0:01.20    0     0 watchdog/2
  15 root        RT    0     0     0     0 S  0.0   0.0   0:00.22    0     0 migration/3
  16 root        RT    0     0     0     0 S  0.0   0.0   0:00.00    0     0 migration/3
  17 root        20   0     0     0     0 S  0.0   0.0   0:01.40    0     0 ksoftirqd/3
  18 root        RT    0     0     0     0 S  0.0   0.0   0:01.18    0     0 watchdog/3
```

我们发现进程 id 为 12363 的“top”进程被加亮了，top 进程就是视图第二行显示的唯一的运行态（runing）的那个进程，可以通过敲击“y”键关闭或打开运行态进程的加亮效果。

2. 敲击键盘“x”（打开/关闭排序列的加亮效果），top 的视图变化如下：

可以看到，top 默认的排序列是“%CPU”。

3. 通过“shift + >”或“shift + <”可以向右或左改变排序列，下图是按一次“shift + >”的效果图：

tasks: 135 total, 1 running, 134 sleeping, 0 stopped, 0 zombie											
Cpu0 : 0.3%us, 0.0%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
Cpu1 : 0.3%us, 0.0%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
Cpu2 : 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st											
Cpu3 : 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: 3808060k total, 3619796k used, 188264k free, 360156k buffers											
Swap: 4184924k total, 0k used, 4184924k free, 2484828k cached											
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
19	root	20	0	0	0	0	S	0.3	0.0	4:01.00	events/0
1727	root	20	0	30556	2816	2304	S	0.3	0.1	11:17.09	AliYunDunUpdate
2051	root	20	0	896m	9920	5852	S	0.3	0.3	25:07.89	AliHids
12333	root	20	0	15036	1244	936	R	0.3	0.0	0:00.73	top
1	root	20	0	19232	1568	1280	S	0.0	0.0	0:03.27	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.43	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.16	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:02.99	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	0:02.76	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:01.20	watchdog/1
11	root	RT	0	0	0	0	S	0.0	0.0	0:00.18	migration/2
12	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/2
13	root	20	0	0	0	0	S	0.0	0.0	0:02.09	ksoftirqd/2
14	root	RT	0	0	0	0	S	0.0	0.0	0:01.20	watchdog/2
15	root	RT	0	0	0	0	S	0.0	0.0	0:00.22	migration/3
16	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/3
17	root	20	0	0	0	0	S	0.0	0.0	0:01.40	ksoftirqd/3
18	root	RT	0	0	0	0	S	0.0	0.0	0:01.19	watchdog/3
19	root	20	0	0	0	0	S	0.0	0.0	4:00.99	events/0
20	root	20	0	0	0	0	S	0.0	0.0	0:35.45	events/1
21	root	20	0	0	0	0	S	0.0	0.0	0:34.14	events/2
22	root	20	0	0	0	0	S	0.0	0.0	4:09.11	events/3

视图现在已经按照%MEM来排序了。

改变进程显示字段

1. 敲击“f”键，top进入另一个视图，在这里可以编排基本视图中的显示字段：


```

top - 10:43:33 up 10 days, 3:42, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 135 total, 2 running, 133 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.4%us, 0.2%sy, 0.0%ni, 99.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 3808060k total, 3605128k used, 202932k free, 360216k buffers
Swap: 4184924k total, 0k used, 4184924k free, 2484864k cached

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
27623	root	20	0	3469m	497m	11m	S	1.7	13.4	62:27.23	java
2051	root	20	0	896m	9920	5852	S	0.3	0.3	25:08.63	AliHids
1767	root	20	0	305m	7720	5612	S	0.0	0.2	42:06.48	AliYunDun
11451	root	20	0	98368	3988	3040	S	0.0	0.1	0:00.68	sshd
1727	root	20	0	30556	2816	2304	S	0.0	0.1	11:17.51	AliYunDunUpdate
11531	root	20	0	171m	2988	2224	S	0.0	0.1	0:00.68	sudo
1671	root	20	0	16876	12m	2084	S	0.0	0.3	1:04.63	mdmon
1926	mysql	20	0	435m	22m	1792	S	0.0	0.6	5:26.55	mysqld
11484	longjian	20	0	105m	1764	1440	S	0.0	0.0	0:00.01	bash
11533	root	20	0	105m	1760	1432	S	0.0	0.0	0:00.04	bash
1	root	20	0	19232	1568	1280	S	0.0	0.0	0:03.27	init
11532	root	20	0	142m	1612	1232	S	0.0	0.0	0:00.00	su
1824	root	20	0	105m	1444	1212	S	0.0	0.0	0:00.00	mysqld_safe
1610	root	20	0	243m	5320	1028	S	0.0	0.1	0:31.11	rsyslogd
12501	root	20	0	15036	1236	936	R	0.7	0.0	0:00.12	top
11481	longjian	20	0	98368	1840	832	S	0.0	0.0	0:00.30	sshd
1981	root	20	0	114m	1268	660	S	0.0	0.0	0:04.91	crond
1697	root	20	0	4080	656	544	S	0.0	0.0	0:00.00	acpid

这里列出了所有可在 top 基本视图中显示的进程字段，有“*”并且标注为大写字母的字段是可显示的，没有“*”并且是小写字母的字段是不显示的。如果要在基本视图中显示

“CODE”和“DATA”两个字段，可以通过敲击“r”和“s”键：

- “回车”返回基本视图，可以看到多了“CODE”和“DATA”两个字段：

```

top - 10:57:37 up 10 days, 3:54, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 135 total, 1 running, 134 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.5%us, 0.2%sy, 0.0%ni, 99.3%id, 0.0%wa, 0.0%hi, 0.1%si, 0.0%st
Mem: 3808060k total, 3598396k used, 209664k free, 360492k buffers
Swap: 4184924k total, 0k used, 4184924k free, 2485016k cached

```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	UID	CODE	DATA	COMMAND
27623	root	20	0	3473m	495m	16m	S	2.3	13.3	62:33.27	0	4	3.2g	java
2051	root	20	0	896m	9920	5852	S	0.3	0.3	25:09.60	0	1112	740m	AliHids
1	root	20	0	19232	1568	1280	S	0.0	0.0	0:03.27	0	140	288	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.43	0	0	0	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.16	0	0	0	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:03.00	0	0	0	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	0	0	0	migration/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:01.32	0	0	0	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:00.16	0	0	0	migration/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	0	0	0	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	0:02.76	0	0	0	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:01.20	0	0	0	watchdog/1
11	root	RT	0	0	0	0	S	0.0	0.0	0:00.18	0	0	0	migration/2
12	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	0	0	0	migration/2
13	root	20	0	0	0	0	S	0.0	0.0	0:02.09	0	0	0	ksoftirqd/2
14	root	RT	0	0	0	0	S	0.0	0.0	0:01.20	0	0	0	watchdog/2
15	root	RT	0	0	0	0	S	0.0	0.0	0:00.22	0	0	0	migration/3
16	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	0	0	0	migration/3
17	root	20	0	0	0	0	S	0.0	0.0	0:01.41	0	0	0	ksoftirqd/3

top 命令的补充

top 命令是 Linux 上进行系统监控的首选命令，但有时候却达不到我们的要求，比如当前这台服务器，top 监控有很大的局限性。这台服务器运行着 websphere 集群，有两个节点服务，就是【top 视图 01】中的老大、老二两个 java 进程，top 命令的监控最小单位是进程，所以看不到我关心的 java 线程数和客户连接数，而这两个指标是 java 的 web 服务非常重要的指标，通常我用 ps 和 netstate 两个命令来补充 top 的不足。

监控 java 线程数：

```
ps -eLf | grep java | wc -l
```

监控网络客户连接数：

```
netstat -n | grep tcp | grep 侦听端口 | wc -l
```

上面两个命令，可改动 grep 的参数，来达到更细致的监控要求。

在 Linux 系统“一切都是文件”的思想贯彻指导下，所有进程的运行状态都可以用文件来获取。系统根目录/proc 中，每一个数字子目录的名字都是运行中的进程的 PID，进入任一个进程目录，可通过其中文件或目录来观察进程的各项运行指标，例如 task 目录就是用来描述进程中线程的，因此也可以通过下面的方法获取某进程中运行中的线程数量（PID 指的是进程 ID）：

```
ls /proc/PID/task | wc -l
```

在 linux 中还有一个命令 pmap，来输出进程内存的状况，可以用来分析线程堆栈：

```
pmap PID
```

大家都熟悉 Linux 下可以通过 top 命令来查看所有进程的内存，CPU 等信息。除此之外，还有其他一些命令，可以得到更详细的信息，例如进程相关

```
cat /proc/your_PID/status
```

通过 top 或 ps -ef | grep '进程名' 得到进程的 PID。该命令可以提供进程状态、文件句柄数、内存使用情况等信息。

内存相关

```
vmstat -s -S M
```

该可以查看包含内存每个项目的报告，通过-S M 或-S k 可以指定查看的单位，默认为 kb。结合 watch 命令就可以看到动态变化的报告了。

也可用 cat /proc/meminfo

要看 cpu 的配置信息可用

```
cat /proc/cpuinfo
```

它能显示诸如 CPU 核心数，时钟频率、CPU 型号等信息。

要查看 cpu 波动情况的，尤其是多核机器上，可使用

```
mpstat -P ALL 10
```

该命令可间隔 10 秒钟采样一次 CPU 的使用情况，每个核的情况都会显示出来，例如，每个核的 idle 情况等。

只需查看均值的，可用

```
iostat -c
```

IO 相关

```
iostat -P ALL
```

该命令可查看所有设备使用率、读写字节数等信息。

另外，htop，有时间可以用一下。

Linux 查看物理 CPU 个数、核数、逻辑 CPU 个数

总核数 = 物理 CPU 个数 X 每颗物理 CPU 的核数

总逻辑 CPU 数 = 物理 CPU 个数 X 每颗物理 CPU 的核数 X 超线程数

查看物理 CPU 个数

```
cat /proc/cpuinfo | grep "physical id" | sort | uniq | wc -l
```

查看每个物理 CPU 中 core 的个数(即核数)

```
cat /proc/cpuinfo | grep "cpu cores" | uniq
```

查看逻辑 CPU 的个数

```
cat /proc/cpuinfo | grep "processor" | wc -l
```

查看 CPU 信息 (型号)

```
cat /proc/cpuinfo | grep name | cut -f2 -d: | uniq -c
```

五：TCP/IP 协议的配置选项

a) TCP_DEFER_ACCEPT 参数

用法范例：

```
setsockopt( listen_fd, IPPROTO_TCP, TCP_DEFER_ACCEPT, &timeout, sizeof(int) )
```

作用：一般三路握手后我们就可以用 `accept()` 函数把这个连接从 已完成连接 队列 中就拿出来来了，用了这个选项之后，只有客户端往这个连接上发送数据了，`accept()` 才会返回【而不再是三次握手就返回】，那是否有可能有用户连着你不发数据【恶意攻击】，那我这个时候我用这个选项不会触发 `accept()` 返回，因为唤醒 `accept()` 肯定会有一些系统上下文切换的，这也是代价；

b)tcp 参数，这些资料也是老师参考了一些网上对 nginx 的一些性能优化配置方法，大家可以参考借鉴，其实类似这样的参数非常多，大家完全可以自己百度慢慢认识；

在 `/etc/sysctl.conf` 文件中有一些配置项，可能有的影响客户端，有的影响服务器端；

`net.ipv4.tcp_syn_retries` 【客户端】：主动建立连接时，发送 `syn` 的重试次数；

`net.ipv4.ip_local_port_range` 【客户端】：主动建立时，本地端口范围，大家都知道，客户端端口一般系统分配；

net.ipv4.tcp_max_syn_backlog 【服务器】：处于 SYN_RECV 状态，未获得对方确认的连接请求数；就是那个**未完成连接队列**的大小。第五章第四节有讲过 listen 队列；

net.core.somaxconn 【服务器】：已完成连接队列【需要用 accept()取走】的大小受到该值的限制，此值是系统级别的最大的队列长度限制值；

net.ipv4.tcp_synack_retries 【服务器】：回应 SYN 包时会尝试多少次重新发送初始 SYN,ACK 封包后才决定放弃

net.core.netdev_max_backlog：在网卡接收数据包的速率比内核处理这些包的速率快时，允许送到待处理报文队列的数据包的最大数目。缺省值 1000，应对拼命发包攻击时可能有效；

net.ipv4.tcp_abort_on_overflow：超出处理能力时，对新来的 syn 连接请求直接回 rst 包；缺省是关闭的；

net.ipv4.tcp_syncookies：这个项也是防止一些 syn 攻击 用的，syn 队列满的时候，新的 syn 将不再进入未完成连接队列，而是用一种 syncookie 技术进行三路握手，也就是说服务会计算出 cookie 然后把这个 cookie 通过 syn/ack 标记的包返回给客户端，客户端发送数据时，服务根据数据包中带的 cookie 信息来 恢复连接。这个选项可能有些副作用，因为 syn/ack 包会返回一个序列号信息，现在返回的信息变成了 cookie，可能会使一些 tcp 协议的功能失效，大家用的时候要完全研究清楚这种参数；而且程序代码上是不是要做一些调整和配合都要搞清楚；因为老师没详细研究过这个参数，感觉这种方式似乎不需要 accept()就能把 用户接入进来，所以感觉代码上有可能要调整；

net.ipv4.tcp_fastopen 【客户端/服务器】：用于优化三次握手，默认不启用；

通过前面的学习，大家都知道了 TCP 的三次握手，其中的第三次握手是个 ack 包，大家抓一下包就会发现，这个包里一般是不携带额外数据的，但是这个包里是可以携带额外的数据的，怎么能携带上额外数据，有兴趣的同学可以测试，老师给大家提供一篇参照文章，“TCP 连接建立的三次握手过程可以携带数据吗？”，因为这个涉及到 tcp 协议内的细节内容，老师不深入谈了：参考：<http://0xfffff.org/2015/04/15/36-The-TCP-three-way-handshake-with-data/> 三次握手大家都熟悉，syn,syn/ack,ack，下次重新连接还要进行三次握手，这很耗费性能，那如果 syn/ack 的时候给客户端回一个 cookie，那么下次客户端重新连接到服务器的时候不用进行三次握手，而是 可以直接发送 syn 包，里边夹带 cookie 并夹带要发送的数据即可，那这样是不是省了好几步数据传输；不过要求 c/s 都要支持这种特性才能做到，因为客户端要发送一个带 fast open cookie request 请求的包给服务器的，而且你这种 fastopen 特性如果开启的话，可能还涉及到 fastopen 队列，这个队列的最大长度你可以也要考虑设置一下，这种探索或者说是代码怎么书写，如果大家有兴趣，可以自行探索，老师这里不带着大家一起探索；

net.ipv4.tcp_rmem：收数据缓存的最小值，默认值，最大值（单位：字节）

net.ipv4.tcp_wmem：发数据缓存的最小值，默认值，最大值（单位：字节）

tcp_adv_win_scale：这东西会把上边这个缓存拿出来一部分作为额外开销，这个数字用于确定拿出来多少作为额外开销；

其实**还有很多网络选项**，老师就不在这里一一列举，大家可以通过各种搜索引擎来继续深入学习；

六：TCP/IP 协议额外注意的一些算法、概念等

a)滑动窗口的概念：

tcp 协议引入滑动窗口主要是为了解决高速传输和流量控制问题【限速问题】；老师希望大家对滑动窗口要有概念，这个概念和实现一般都会在操作系统内核里边干，但是老师仍旧希望大家能够了解相关的概念；

b)Nagle 算法的概念：

这个算法是避免发送很小的报文，大家都知道，报文再小他也要有包头，那么我们把几个小报文组合成一个大一点的报文再发送，那至少我们能够少发好几个包头，节省了流量和网络带宽；

c)Cork 算法：

比 Nagle 更硬气，完全禁止发送小报文，除非累积到一定数量的数据或者是超过某一个时间才会发送这种报文；

d) Keep-Alive 机制：

用于关闭已经断开的 tcp 连接，这个咱们以往也提及过，那作为 TCP/IP 协议中的一个概念，这里也再次把他提及一下；

net.ipv4.tcp_keepalive_time：探测包发送周期单位是秒，默认是 7200（2 小时）；如果 2 小时内没有任何报文在这个连接上发送，那么开始启动 keep-alive 机制，发送 keepalive 包。

net.ipv4.tcp_keepalive_intvl：缺省值 75（单位秒）如果发送 keepalive 包没应答，则过 75 秒再次发送 keepalive 包；

net.ipv4.tcp_keepalive_probes：缺省值 9，如果发送 keepalive 包对方一直不应答，发送 9 次后，如果仍然没有回应，则这个连接就被关闭掉；

e) SO_LINGER 选项：

延迟关闭选项，一般调用 setsockopt(SO_LINGER)，这个选项设置 在连接关闭的时候，是否进行正常的四次挥手有关；因为缺省的 tcp/ip 协议可能会导致某一通讯方给对方发送 rst 包以结束连接从而导致对方收到 rst 包而丢弃收到的数据，那么这个延迟选项可以**避免给对方发送 rst**包导致对方丢弃收到的数据；

说的再白一点：这个选项用来设置延迟关闭的时间，等待套接字发送缓冲区中的数据发送完成；比较抽象，具体的大家可以百度一下；

延迟关闭并不是一个好事，所以大家要研究明白才能决定是否用它，咱们这个项目中，感觉没必要用；