Linux系统命令与网络、磁盘参数和日志监控-课件

Linux系统命令与网络、磁盘参数和日志监控

授课老师： Darren老师 QQ326873713

班 主 任： 柚子老师 QQ2690491738

日　　期：2022年07月28日

0 课程重点

1. Linux基础命令和工具
2. **CPU性能监控**
3. 内存性能监控
4. **文件IO性能监控**
5. **网络IO性能监控**

强烈推荐：

Linux 大牛，Netflix 高级性能架构师 Brendan Gregg的博客http://www.brendangregg.com

主要分为：

- **CPU**
- 内存
- **磁盘**
- **网络**

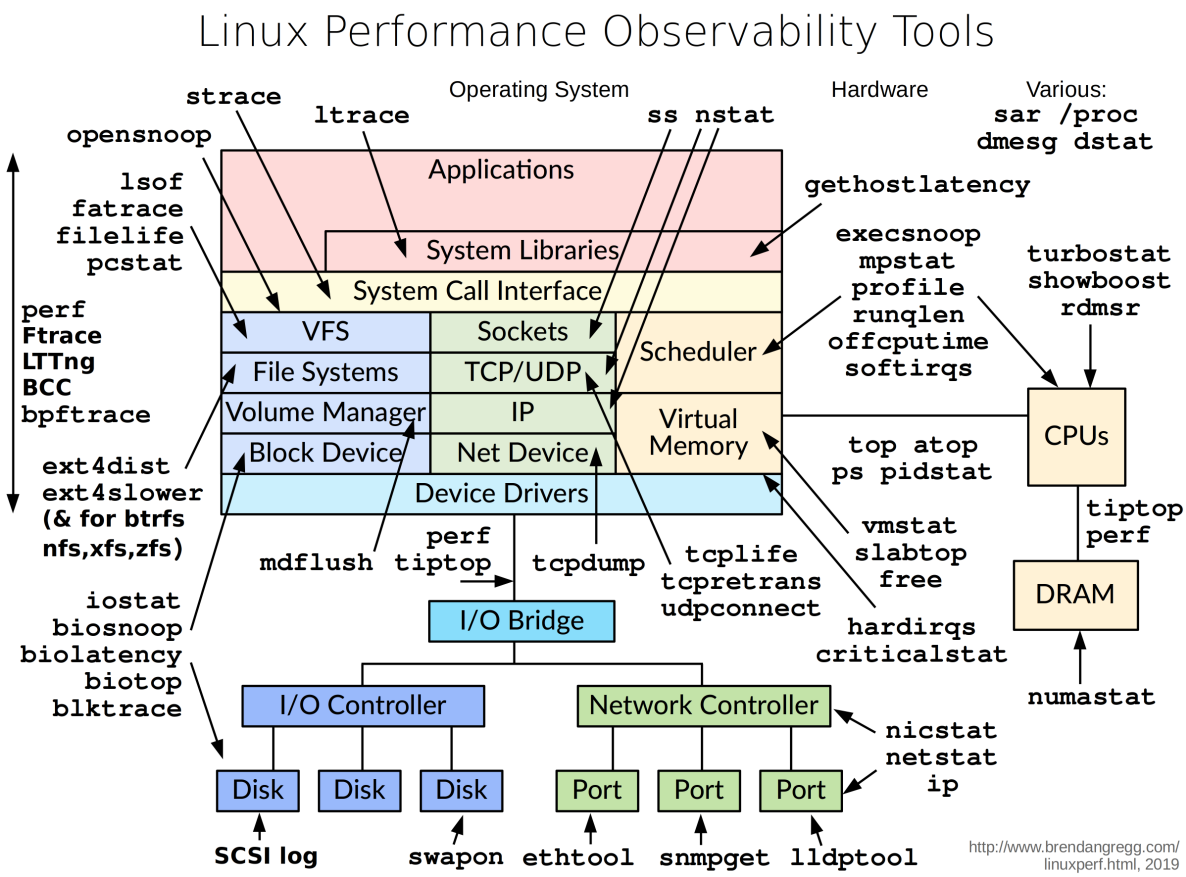
四大块，下面对应的命令大部分都不是专为某一个模块设计的。所以大家先把基本的命令都掌握，再去细分每个命令的侧重点。

0.1 监控

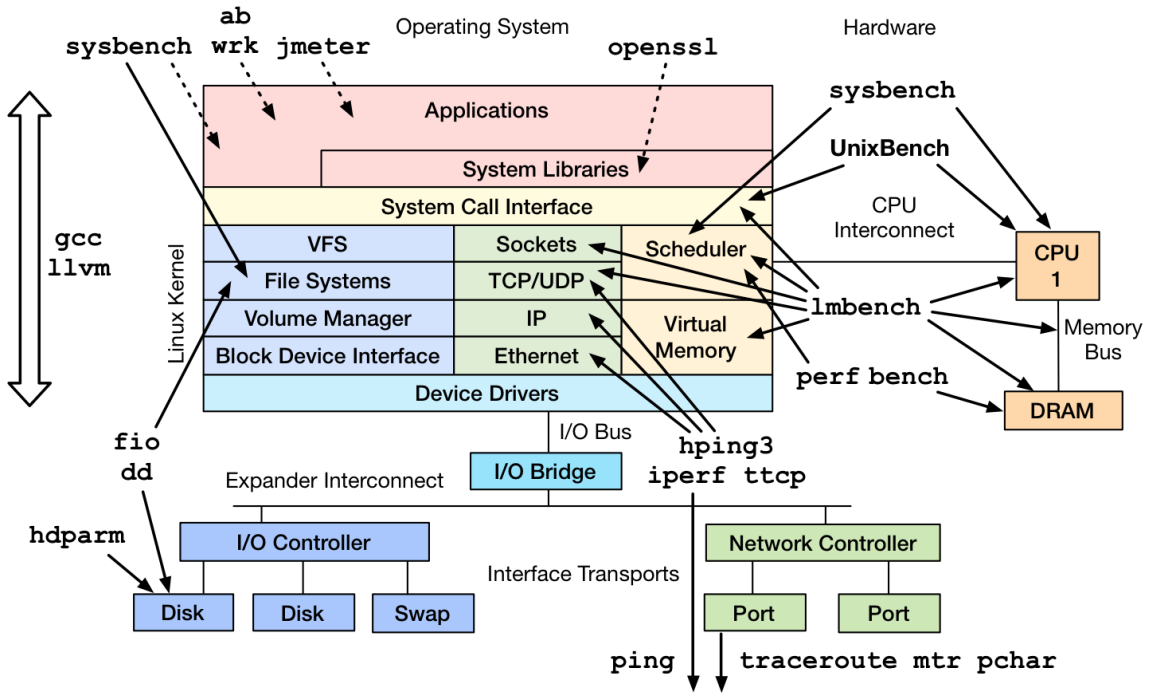
常用的命令：

- free
- ping
- vmstat(VirtualMeomoryStatistics,虚拟内存统计)
- **iostat** 用于报告中央处理器（CPU）统计信息和整个系统、适配器、tty 设备、磁盘和 CD-ROM 的输入/输出统计信息
- **dstat** 显示了cpu使用情况，磁盘io情况，网络发包情况和换页情况，输出是彩色的，可读性较强，相对于vmstat和iostat的输入更加详细且较为直观。
- **pidstat** 主要用于监控全部或指定进程占用系统资源的情况,如CPU,内存、设备IO、任务切换、线程等。
- **top** 命令的汇总区域显示了五个方面的系统性能信息：负载、进程状态、cpu使用率、内存使用、交换分区。

- **iotop LINUX进程实时监控工具，**界面风格类似top命令
- **htop** 是Linux系统中的一个互动的进程查看器,一个文本模式的应用程序(在控制台或者X终端中),需要ncurses。
- **mpstat** Report processors related statistics. 报告CPU的统计信息。
- **netstat** 用于显示与IP、TCP、UDP和ICMP协议相关的统计数据，一般用于检验本机各端口的网络连接情况。
- **ps** 显示当前进程的状态
- **strace** Trace system calls and signals。跟踪程序执行过程中产生的系统调用及接收到的信号，帮助分析程序或命令执行中遇到的异常情况。
- ltrace A library call tracer 跟踪进程调用库函数的情况
- **uptime** 能够打印系统总共运行了多长时间和系统的平均负载，uptime命令最后输出的三个数字的含义分别是1分钟，5分钟，15分钟内系统的平均负荷
- **lsof** (list open files)是一个列出当前系统打开文件的工具。
- **perf** 是Linux kernel自带的系统性能优化工具。优势在于与Linux Kernel的紧密结合，它可以最先应用到加入Kernel的new feature，用于查看热点函数，查看cashe miss的比率，从而帮助开发者来优化程序性能。
- tcpdump
- sar
- blktrace



0.2 测试



sysbench是一个模块化、跨平台、多线程基准测试工具，可用于以下性能测试：

1.CPU性能

2.磁盘IO性能

3.调度程序性能

4.内存分配及传输速度

5.POSIX线程性能

6.数据库性能（OLTP基准测试）

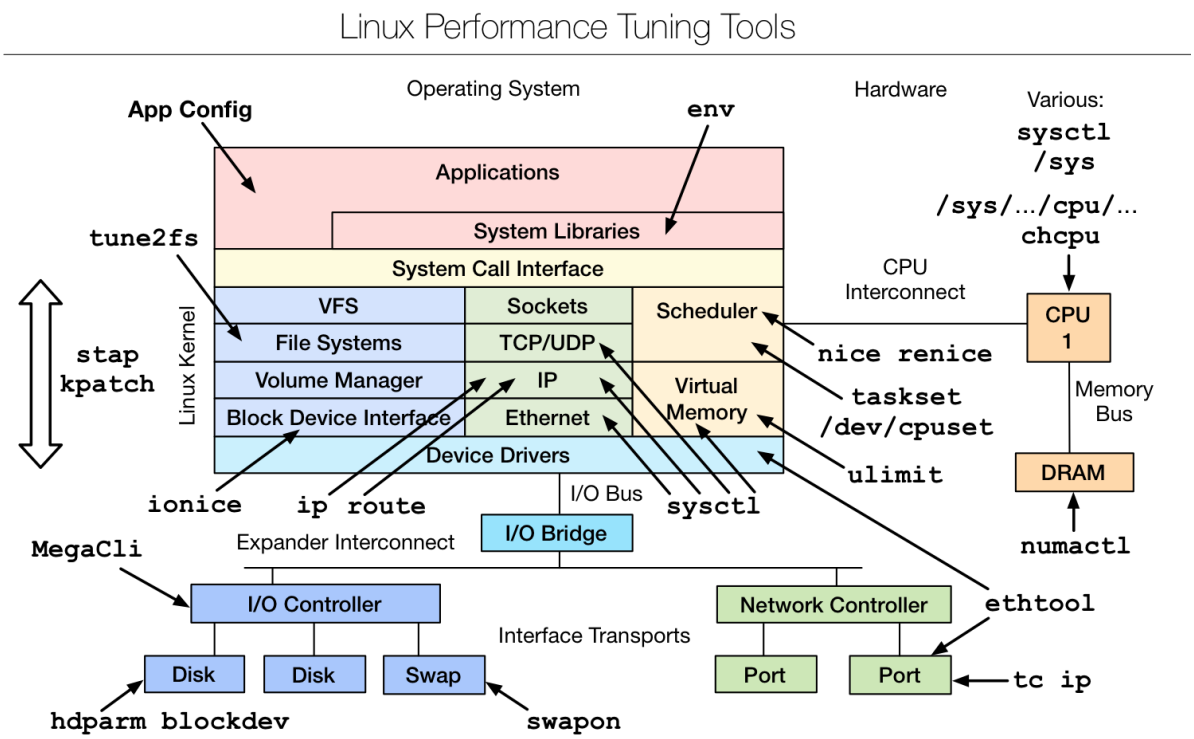
Linux CPU使用率主要是从以下几个维度进行统计：

- %usr：普通进程在用户模下下执行的时间；
- %sys：进程在内核模式下的执行时间；
- %nice：被提高优先级的进程在用户模式下的执行时间；
- %idle：空闲时间。
- %iowait：等待I/O完成的时间。
- %irp：处理硬中断请求花费的时间。
- %soft：处理软中断请求花费的时间。
- %steal：是衡量虚拟机CPU的指标，是指分配给本虚拟机的时间片被同一宿主机别的虚拟机占用，一般%steal值较高时，说明宿主机的资源使用已达到瓶颈；

一般情况下，CPU大部分的时间片都是消耗在用户态和内核态上。

sys和user间的比例是相互影响的，%sys比例高意味着被测服务频繁的进行用户态和系统态之间的切换，会带来一定的CPU开销，这样分配处理业务的时间片就会较少，造成系统性能的下降。对于IO密集型系统，无论是网络IO还是磁盘IO，一般都会产生大量的中断，从而导致%sys相对升高，其中磁盘IO密集型系统，对磁盘的读写需要占用大量的CPU，会导致%iowait的值一定比例的升高，所以当出现%iowait较高时，需排查是否存在大量的不合理的日志操作，或者频繁的数据载入等情况；

0.3 优化



1 Linux基础命令和工具

1.1 grep搜索字符

grep 命令用于在文件中执行关键词搜索，并显示匹配的效果。部分常用选项

参数	作用
-c	仅显示找到的行数
-i	忽略大小写
-n	显示行号
-v	反向选择 – 仅列出没有关键词的行。v 是 invert 的缩写。
-r	递归搜索文件目录
-C n	**打印匹配行的前后n行**

在指定文件查找，查找login关键字

```
grep  login  ImUser.cpp
```

多个文件中搜索

```
grep  login  ImUser.cpp MsgConn.cpp
```

在多个文件搜索的时候，可以使用通配符。在以 cpp结尾的文件中，搜索包含login 的行

```
 grep  login  *.cpp
```

递归搜索目录下所有文件, 搜索 msg_server目录下所有文件，打印出包含 login的行。

```
 grep login -r msg_server/
```

反向查找，查找文件中，不包含 CImUser 的行。

```
grep -v CImUser ImUser.cpp
```

找出文件中包含 login的行，并打印出行号

```
grep  -n login  ImUser.cpp
```

找出文件中包含 login的行，打印出行号，并显示前后3行

```
grep -C 3 -n login  ImUser.cpp
```

找出文件中包含 login的行，打印出行号，并显示前后3行，并忽略大小写

```
grep -C 3 -i -n login  ImUser.cpp
```

1.2 find查找文件

通过文件名查找文件的所在位置，文件名查找支持模糊匹配
find [指定查找目录] [查找规则] [查找完后执行的action]
常用的操作：

```
find . -name FILE_NAME
find . -iname FILE_NAME 忽略文件名称大小写
find /etc -maxdepth 1 -name passwd     ##查找/etc/下名称中带有passwd的文件，查找一层
find /mnt -size 20K       ##查找/mnt文件大小近似20k的文件
find /mnt -size +20K      ##查找/mnt文件大小大于20k的文件
find /mnt -size -20K      ##查找/mnt文件大小小于20k的文件
find /etc -maxdepth 2 -mindepth 2 -name .conf  ##查找/etc/下名称中带有.conf的文件，且只查找第二层
find /mnt -type d         ##按type查找/mnt中目录
find /mnt -type f         ##按type查找/mnt中文件
find /mnt -cmin 10        ##查找/mnt中十分钟左右修改的
find /mnt -cmin +10       ##查找/mnt中十分钟以上修改的
find /mnt -cmin -10       ##查找/mnt中十分钟以内修改的
find /mnt -ctime 10       ##查找/mnt中十天左右修改的
find /mnt -ctime +10      ##查找/mnt中十天以上修改的
find /mnt -ctime -10      ##查找/mnt中十天以内修改的
```

1.3 ls显示文件

-t 可以查看最新修改的时间
-l 每行显示一个条目
-h 可以结合显示文件的GB，MB等(human)；
-R 递归显示
-n 显示组id和gid

练习：
ls -lt 按最新修改的时间排序，新修改的在前面显示。
ls -ltR 按最新修改的时间排序，新修改的在前面显示，并显示子目录的文件信息
ls -lh 以单位显示文件大小

1.4 wc命令

wc命令用于计算字数。 利用wc指令我们可以计算文件的Byte数、字数、或是列数，若不指定文件名称、或是所给予的文件名为"-"，则wc指令会从标准输入设备读取数据。
语法

wc [-clw][--help][--version][文件...]

参数：

- -c或--bytes或--chars 只显示Bytes数。
- -l或--lines 只显示行数。
- -w或--words 只显示字数。
- --help 在线帮助。
- --version 显示版本信息。

练习：
wc testfile   # testfile文件的统计信息
7 92 607 testfile   # testfile文件的行数为7、单词数92、字节数607

wc -l testfile

1.5 ulimit用户资源

Linux系统对每个登录的用户都限制其最大进程数和打开的最大文件句柄数。为了提高性能，可以根据硬件资源的具体情况设置各个用户的最大进程数和打开的最大文件句柄数。可以用ulimit -a来显示当前的各种系统对用户使用资源的限制:

[root@ubuntu~]# ulimit -a
core file size   (blocks, -c) 0
data seg size   (kbytes, -d) unlimited
scheduling priority   (-e) 0
file size   (blocks, -f) unlimited
pending signals   (-i) 7269
max locked memory   (kbytes, -l) 64
max memory size   (kbytes, -m) unlimited
open files   (-n) 100001
pipe size   (512 bytes, -p) 8
POSIX message queues   (bytes, -q) 819200
real-time priority   (-r) 0
stack size   (kbytes, -s) 8192
cpu time   (seconds, -t) unlimited
max user processes   (-u) 65535
virtual memory   (kbytes, -v) unlimited
file locks   (-x) unlimited
设置用户的最大进程数：
ulimit -u 1024

设置用户可以打开的最大文件句柄数：
ulimit -n 65530

1.6 curl http

由于当前的线上服务较多地使用了RESTful风格的API，所以集成测试就需要进行HTTP调用，查看返回的结果是否符合预期，curl命令当然是首选的测试方法。
使用方式：

curl -i "http://www.sina.com" # 打印请求响应头信息
curl -I "http://www.sina.com" # 仅返回http头
curl -v "http://www.sina.com" # 打印更多的调试信息
curl -verbose "http://www.sina.com" # 打印更多的调试信息
curl -d 'abc=def' "http://www.sina.com" # 使用post方法提交http请求
curl -sw '%{http_code}' "http://www.sina.com" # 打印http响应码

1.7 scp远程拷贝

secure copy的缩写, scp是linux系统下基于ssh登陆进行安全的远程文件拷贝命令。
scp命令是Linux系统中功能强大的文件传输命令，可以实现从本地到远程，以及从远程到本地的双向文件传输，用起来非常方便，常用来在线上定位问题时将线卜的一些文件下载到本地进行详查，或者将本地的修改上传到服务器上。
使用方式:
sudo apt-get install openssh-server
scp liaoqingfu@122.152.222.180:/home/liaoqingfu/test.txt . //下载122.152.222.180的文件
scp mi9.aac liaoqingfu@122.152.222.180:/home/liaoqingfu/ //上传文件到122.152.222.180

scp -r liaoqingfu@122.152.222.180:/home/liaoqingfu/test . //下载test整个目录到本地

scp -r liaoqingfu@122.152.222.180:/home/liaoqingfu/test .

scp -r test liaoqingfu@122.152.222.180:/home/liaoqingfu/ //上传本地test整个目录到122.152.222.180

1.8 dos2unix和unix2dos

用于转换Windows和UNIX的换行符，通常在Windows系统h开发的脚本和配置，UNIX系统下都需要转换。
使用方式：

```
dos2unix test.txt
unix2dos test.txt
```

转换整个目录

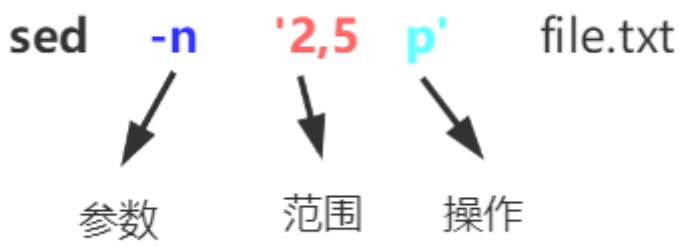
```
find . -type f -exec dos2unix {} \;
find ./ -type f                此命令是显示当前目录下所有的文件
```

1.9 sed 行处理

命令格式1：sed 's/原字符串/新字符串/' 文件
命令格式2：sed 's/原字符串/新字符串/g' 文件
这两种命令格式的区别在于是否有个“g”。没有“g”表示只替换第一个匹配到的字符串，有“g”表示替换**所有能匹配到的字符串**，“g”可以认为是“global”（全局的）的缩写，没有“全局的”结尾就不要替换全部。

sed命令是用来批量修改文本内容的，比如批量替换配置中的某个ip。
sed命令在处理时，会先读取一行，把当前处理的行存储在临时缓冲区中，处理完缓冲区中的内容后，打印到屏幕上。然后再读入下一行，执行下一个循环。不断的重复，直到文件末尾。

简单模式



一个简单的sed命令包含三个主要部分：`参数`、`范围`、`操作`。要操作的文件，可以直接挂在命令行的最后。

参数

-n 这个参数是`--quiet`或者`--silent`的意思。表明忽略执行过程的输出，只输出我们的结果即可。
还有另外一个参数：`-i`。**使用此参数后，所有改动将在原文件上执行。你的输出将覆盖原文件。非常危险**，一定要注意。

范围

2,5 表示找到文件中,2,3,4,5行的内容。
这个范围的指定很有灵性，请看以下示例（请自行替换图中的范围部分）。

> **5** 选择第5行。**2,5** 选择2到5行，共4行。**1~2** 选择奇数行。**2,5** 选择2到5行，共4行。**1~2** 选择奇数行。**2~2** 选择偶数行。

范围的选择还可以使用正则匹配。请看下面示例。

> **/**void**/,+3** 选择出现void字样的行，以及后面的三行。**2\^**void**/,/mem/** 选择以void开头的行，和出现mem字样行之间的数据。

为了直观，下面的命令一一对应上面的介绍，范围和操作之间是可以有空格的。

```
sed -n '5p' sed1.cpp

sed -n '2,5 p' sed1.cpp

sed -n '1~2 p' sed1.cpp

sed -n '2~2 p' sed1.cpp

sed -n '2,+3p' sed1.cpp

sed -n '2,$ p' sed1.cpp
```

```
sed -n '/void/,+3 p' sed1.cpp

sed -n '/^void/,/CLIENT_TYPE_FLAG_BOTH/p' sed1.cpp
sed -n '/^BroadcastPdu/,/CLIENT_TYPE_FLAG_BOTH/p' sed1.cpp
sed -n '/^void CImUserManager::BroadcastPdu/,/CLIENT_TYPE_FLAG_BOTH/p' sed1.cpp
```

操作

最常用的操作就是`p`，意思就是打印。比如，以下两个命令就是等同的：

```
cat file
sed -n 'p' file
```

除了打印，还有以下操作，我们来说常用的。

> **p** 对匹配内容进行打印。**d** 对匹配内容进行删除。这个时候就要去掉`-n`参数了，想想为什么。??????????????**w** 将匹配内容写入到其他地方。

`a`,`i`,`c`等操作虽基本但使用少，不做介绍。我们依然拿一些命令来说明。

```
sed -n '2,5 p' sed2.cpp
sed    '2,5 d' sed2.cpp
sed -n '2,5 w output.txt' sed2.cpp
```

替换模式

以上是`sed`命令的常用匹配模式，但它还有一个强大的替换模式，意思就是查找替换其中的某些值，并输出结果。使用替换模式很少使用`-n`参数。

替换模式的参数有点多，但第一部分和第五部分都是可以省略的。替换后会将**整个文本**输出出来。前半部分用来匹配一些范围，而后半部分执行替换的动作。

范围

这个范围和上面的范围语法类似。看下面的例子。

> **/sys/,+3** 选择出现sys字样的行，以及后面的三行。**/\^sys/,/mem/** 选择以sys开头的行，和出现mem字样行之间的数据。

具体命令为：

```
sed -n '/void/,+3 s/void/int/g' sed2.cpp
sed '/^void/,/CLIENT_TYPE_FLAG_BOTH/s/ImUser/User/g' sed2.cpp
```

命令

这里的命令是指s。也就是substitute的意思。

查找匹配

查找部分会找到要被替换的字符串。这部分可以接受纯粹的字符串，也可以接受正则表达式。看下面的例子。

> **a** 查找范围行中的字符串`a`
> 。**[a,b,c]** 从范围行里查找字符串a或者b或者c。

命令类似：

```
sed 's/a/b/g' file
sed 's/[a,b,c]/<&>/g' file
#这个命令我们下面解释
```

替换

是时候把找出的字符串给替换掉了。本部分的内容将替换查找匹配部分找到的内容。
可惜的是，这部分不能使用正则。常用的就是精确替换。比如把a替换成b。
但也有高级功能。和java或者python的正则api类似，sed的替换同样有`Matched Pattern`的含义，同样可以得到Group，不深究。常用的替位符，就是`&`。
`**&**`号，再重复一遍。当它用在替换字符串中的时候，代表的是原始的查找匹配数据。

> **[&]** 表明将查找到的数据使用[]包围起来。**“&”** 表明将查找的数据使用“”包围起来。

下面这条命令，将会把文件中的每一行，使用引号包围起来。

```
sed 's/.*/"&"/' file
```

flag 参数

这些参数可以单个使用，也可以使用多个，仅介绍最常用的。

> **g** 默认只匹配行中第一次出现的内容，加上g，就可以全文替换了。常用。**p** 当使用了`-n`参数，`p`
> 将仅输出匹配行内容。**w** 和上面的w模式类似，但是它仅仅输出有变换的行。**i** 这个参数比较重要，表示忽略大小写。**e** 表示将输出的每一行，执行一个命令。不建议使用，可以使用xargs配合完成这种功能。

看两个命令的语法：

```
sed -n 's/a/b/gipw output.txt' file
sed 's/^/ls -la/e' file
```

更进一步学习：https://github.com/Black-Gold/Learn/blob/1ee76ca2a9bbbbfe04850a1ccc9b9658e1eb39de/Linux_man_cn/sed.md

---

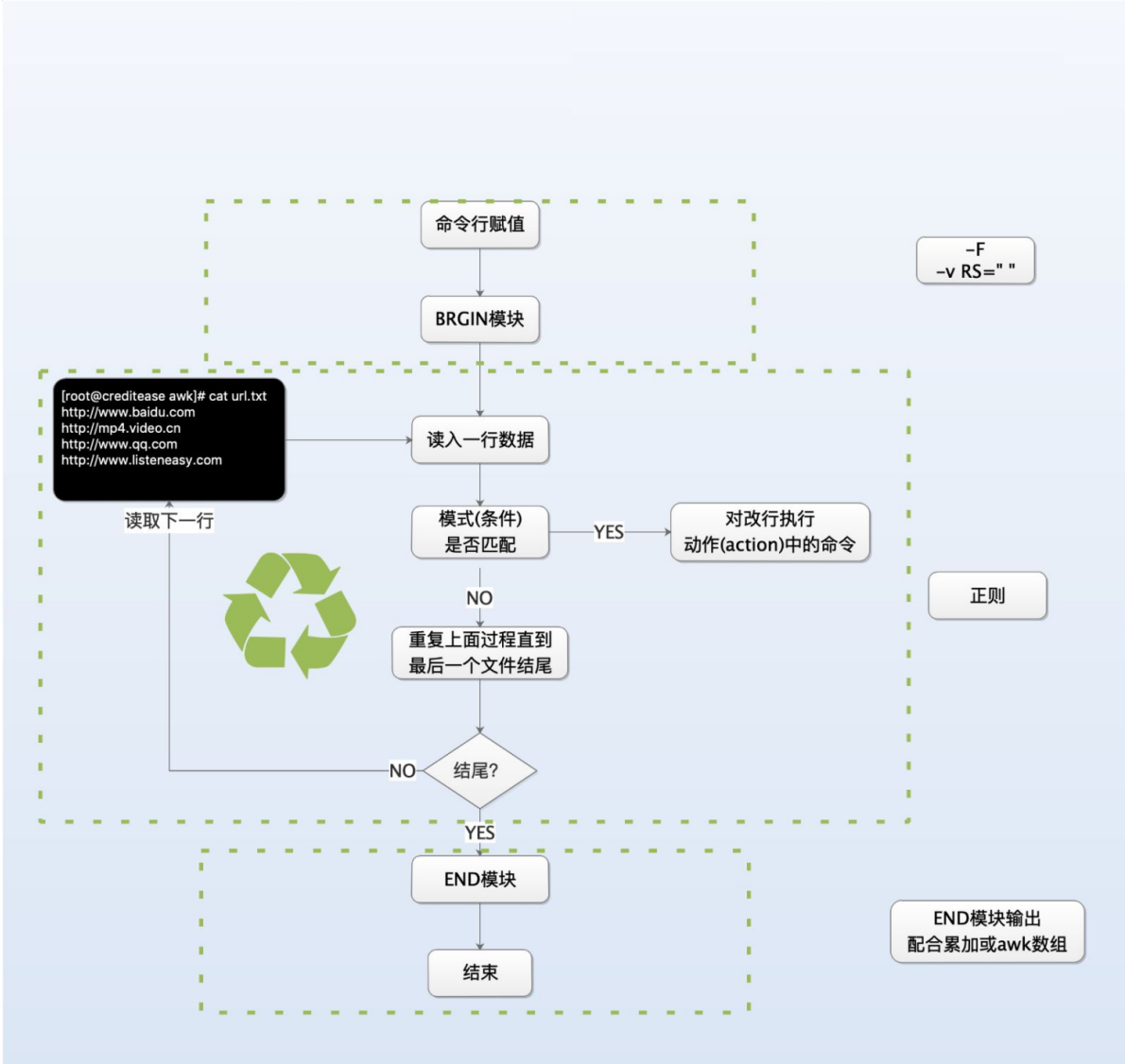
^M 就是\r

1.10 awk 列处理

`awk` 同 `sed` 命令类似，只不过 `sed` 擅长取行，`awk` 命令擅长取列。

原理：一般是遍历一个文件中的每一行，然后分别对文件的每一行进行处理
用法:

```
awk [可选的命令行选项] 'BEGIN{命令 } pattern{ 命令 } END{ 命令 }'  文件名
```



打印某几列

```
$ echo 'I love you' | awk '{print $3 $2 $1}'
youloveI
```

我们将字符串 `I love you` 通过管道传递给awk命令，相当于awk处理一个文件，该文件的内容就是`I love you`,默认通过空格作为分隔符(不管列之间有多少个空格都将当作一个空格处理)`I love you`就分割成三列了。
假如分割符号为`.`，可以这样用

```
$ echo '192.168.1.1' | awk -F "." '{print $2}'
168
```

条件过滤

我们知道awk的用法是这样的，那么pattern部分怎么用呢?

> awk [可选的命令行选项] 'BEGIN{命令 } pattern{ 命令 } END{ 命令 }' 文件名

```
$ cat score.txt
tom 60 60 60
kitty 90 95 87
jack 72 84 99
$ awk '$2>=90{print $0}' score.txt
kitty 90 95 87
```

$2>=90 表示如果当前行的第2列的值大于90则处理当前行，否则不处理。说白了pattern部分是用来从文件中筛选出需要处理的行进行处理的，这部分是空的代表全部处理。
pattern部分可以是任何条件表达式的判断结果，例如`>，<，==，>=，<=，!=`同时还可以使用`+，-，*，/`运算与条件表达式相结合的复合表达式，逻辑`&&，||，!`同样也可以使用进来。另外pattern部分还可以使用 /正则/ 选择需要处理的行。

判断语句

判断语句是写在`pattern{ 命令 }`命令中的，他具备条件过滤一样的作用，同时他也可以让输出更丰富

```
$ awk '{if($2>=90 )print $0}' score.txt
kitty 90 95 87
$ awk '{if($2>=90 )print $1,"优秀"; else print $1,"良好"}' score.txt
tom 良好
kitty 优秀
jack 良好
$ awk '{if($2>=90 )print $0,"优秀"; else print $1,"良好"}' score.txt
tom 良好
kitty 90 95 87 优秀
jack 良好
```

BEGIN 定义表头

```
awk [可选的命令行选项] 'BEGIN{命令 } pattern{ 命令 } END{ 命令 }'  文件名
```

使用方法如下：

```
$ awk 'BEGIN{print "姓名 语文 数学 英语"}{printf "%-8s%-5d%-5d%-5d\n",$1,$2,$3,$4}'
score.txt
姓名 语文数学英语
tom 60 60 60
kitty 90 95 87
jack 72 84 99
```

这里要注意，我为了输出格式好看，做了左对齐的操作(%-8s左对齐，宽8位)，`printf`用法和`c++`类似。
不仅可以用来定义表头，还可以做一些变量初始化的工作，例如

```
$ awk 'BEGIN{OFMT="%.2f";print 1.2567,12E-2}'
1.26 0.12
```

这里OFMT是个内置变量，初始化数字输出格式，保留小数点后两位。

END 添加结尾符

和BEGIN用法类似

```
$ echo ok | awk '{print $1}END{print "end"}'
ok
end
```

数据计算

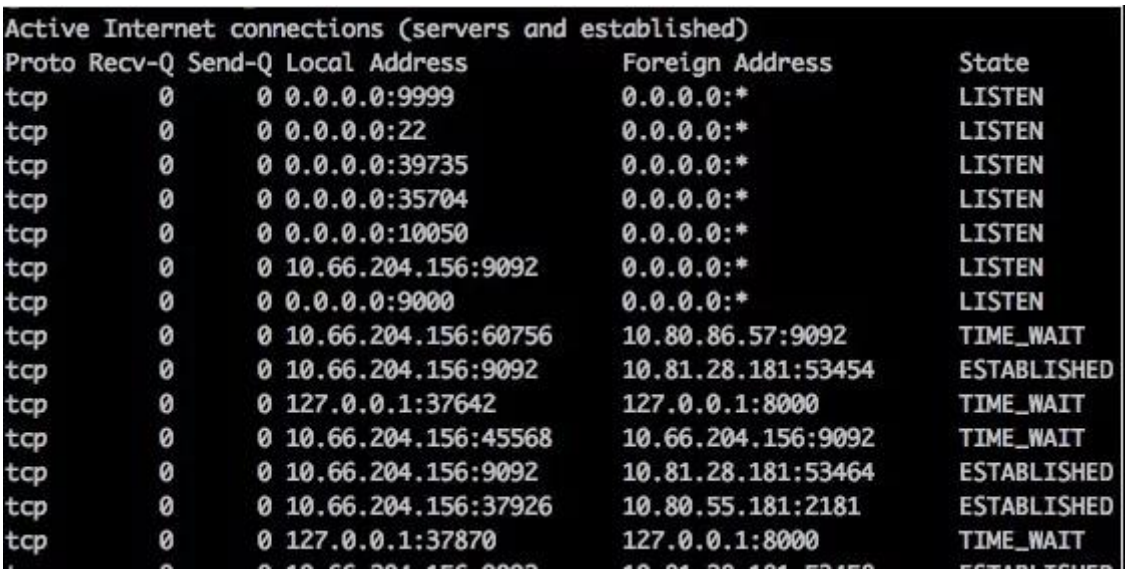
```
$ awk 'BEGIN{print "姓名 语文 数学 英语 总成绩"; \
sum1=0;sum2=0;sum3=0;sumall=0} \
{printf "%5s%5d%5d%5d%5d\n",$1,$2,$3,$4,$2+$3+$4;\
sum1+=$2;sum2+=$3;sum3+=$4;sumall+=$2+$3+$4}\
END{printf "%5s%5d%5d%5d%5d\n","总成绩",sum1,sum2,sum3,sumall}'\
 score.txt
姓名 语文 数学 英语 总成绩
  tom 60 60 60 180
kitty 90 95 87 272
 jack 72 84 99 255
总成绩 222 239 246 707
```

因为命令太长，末尾用`\`符号换行。

- BEGIN体里输出表头，并给四个变量初始化0
- pattern体里输出每一行，并累加运算
- END体里输出总统计结果
  当然了，一个正常人在用linux命令的时候是不会输入那么多格式化符号来对齐的，所以新命令又来了
  `column -t`（鬼知道我为什么会记得这么多乱七八糟的命令。）

范例：网络状态统计

本小节，采用awk统计netstat命令的一些网络状态，来看一下awk语言的基本要素。netstat的输出类似于：



```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:9999            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:39735           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:35704           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:10050           0.0.0.0:*               LISTEN
tcp        0      0 10.66.204.156:9092      0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:9000            0.0.0.0:*               LISTEN
tcp        0      0 10.66.204.156:60756     10.80.86.57:9092        TIME_WAIT
tcp        0      0 10.66.204.156:9092      10.81.28.181:53454      ESTABLISHED
tcp        0      0 127.0.0.1:37642         127.0.0.1:8000          TIME_WAIT
tcp        0      0 10.66.204.156:45568     10.66.204.156:9092      TIME_WAIT
tcp        0      0 10.66.204.156:9092      10.81.28.181:53464      ESTABLISHED
tcp        0      0 10.66.204.156:37926     10.80.55.181:2181       ESTABLISHED
tcp        0      0 127.0.0.1:37870         127.0.0.1:8000          TIME_WAIT
```

其中，第6列，标明了网络连接所处于的网络状态。我们先给出awk命令，看一下统计结果。

```
netstat  -ant |
awk ' \
    BEGIN{print  "State","Count" }  \
    /^tcp/ \
    { rt[$6]++ } \
    END{  for(i in rt){print i,rt[i]}  }'
netstat  -ant |
awk ' \
    BEGIN{print  "State","Count" }  \
    /^tcp/ \
    { if($4=="0.0.0.0:3306" ) rt[$6]++ } \
    END{  for(i in rt){print i,rt[i]}  }'
```

输出结果为：

```
State Count
LAST_ACK 1
LISTEN 64
CLOSE_WAIT 43
ESTABLISHED 719
SYN_SENT 5
TIME_WAIT 146
```

下面这张图会配合以上命令详细说明，希望你能了解awk的精髓。

乍一看，好吓人的命令，但是很简单。awk和我们通常的程序不太一样，它分为四个部分。
1、**BEGIN 开头部分**，可选的。用来设置一些参数，输出一些表头，定义一些变量等。上面的命令仅打印了一行信息而已。
2、**END 结尾部分**，可选的。用来计算一些汇总逻辑，或者输出这些内容。上面的命令，使用简单的for循环，输出了数组rt中的内容。
3、**Pattern 匹配部分**，依然可选。用来匹配一些需要处理的行。上面的命令，只匹配tcp开头的行，其他的不进入处理。
4、**Action 模块。**主要逻辑体，按行处理，统计打印，都可以。

> ### 注意点
>
> 1、awk的主程序部分使用单引号'包围，而不能是双引号
> 2、awk的列开始的index是0，而不是1

更进一步学习：https://github.com/Black-Gold/Learn/blob/1ee76ca2a9bbbbfe04850a1ccc9b9658e1eb39de/Linux_man_cn/awk.md

2 CPU性能监控

2.1 平均负载和CPU使用率

1 平均负载基础

平均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，它和 CPU 使用率并没有直接关系。

- 可运行状态的进程，是指正在使用 CPU 或者正在等待 CPU 的进程，也就是我们常用 ps 命令看到的，处于 R 状态（Running 或 Runnable）的进程。
- 不可中断状态的进程则是正处于内核态关键流程中的进程，并且这些流程是不可打断的，比如最常见的**是等待硬件设备的 I/O 响应**，也就是我们在 ps 命令中看到的 D 状态（Uninterruptible Sleep，也称为 Disk Sleep）的进程。

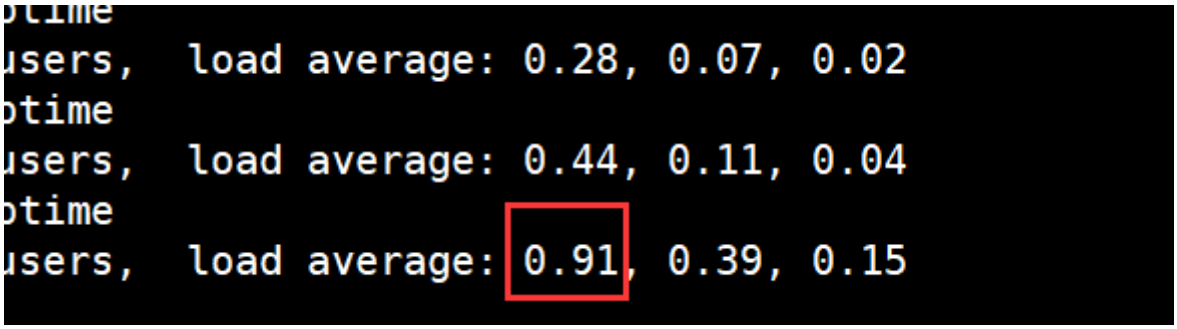
平均负载其实就是平均活跃进程数。平均活跃进程数，直观上的理解就是单位时间内的**活跃进程数。**

2 使用uptime命令分析平均负载

查看机器的启动时间、登录用户、平均负载等情况，通常用于在线上应急或者技术攻关中，确定操作系统的重启时间。

[root@ubuntusrc]# uptime
13:01:52 up 46 days, 22:03, 4 users, load average: 0.13, 0.08, 0.05
从上面的输出可以看到如下信息

- 当前时间： 13:01:52
- 系统已经运行的时间：43天22小时3分钟。
- 前在线用户：4个用户，是总连接数量，不是不同用户数量（**开一个终端连接就算一个用户**）。
- 系统平均负载:**0.13 , 0.08, 0.05**，为最近**1分钟、5分钟、15分钟的系统负载情况。**

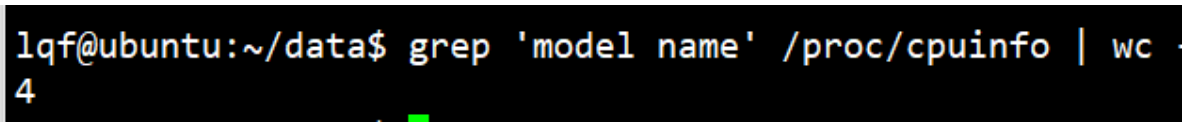


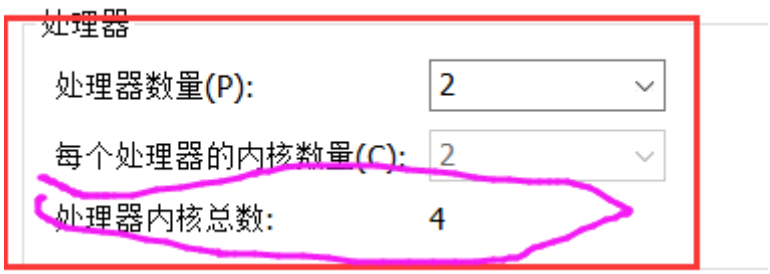
比如当**前平均负载为 2 时**，意味着什么呢?

- 在只有 2 个 CPU 的系统上，意味着所有的 CPU 都刚好被完全占用。
- 在 4 个 CPU 的系统上，意味着 CPU 有 50% 的空闲。
- 而在只有 1 个 CPU 的系统中，则意味着有**一半的进程竞争不到 CPU**。

补充:

> 查看cpu信息：cat /proc/cpuinfo
> 直接获取cpu核数：grep 'model name' /proc/cpuinfo | wc -l





负载说明（现针对单核情况，不是单核时则乘以核数）：

- load<1：没有等待
- load==1：系统已无额外的资源跑更多的进程了
- load>1：进程都堵着等待资源

注意:

- load < 0.7时：系统很闲，要考虑多部署一些服务
- 0.7 < load < 1时：系统状态不错

- load == 1时：系统马上要处理不多来了，赶紧找一下原因
- load > 5时：系统已经非常繁忙了

不同load值说明的问题

1）1分钟 load >5，5分钟 load <3，15分钟 load <1
短期内繁忙，中长期空闲，初步判断是**一个抖动或者是拥塞前兆**

2）1分钟 load >5，5分钟 load >3，15分钟 load <1
短期内繁忙，中期内紧张，很可能是**一个拥塞的开始**

3）1分钟 load >5，5分钟 load >5，15分钟 load >5
短中长期都繁忙，**系统正在拥塞**

4）1分钟 load <1，5分钟Load>3，15分钟 load >5
短期内空闲，中长期繁忙，不用紧张，**系统拥塞正在好转**

举个例子，假设我们在一个单 CPU 系统上看到平均负载为 1.73（1分钟），0.60（5分钟），7.98（15分钟），那么说明在过去 1 分钟内，系统有 73% 的超载，而在 15 分钟内，有 698% 的超载，从整体趋势来看，系统的负载在降低。

3 平均负载与 CPU 使用率

平均负载是指单位时间内，处于可运行状态和不可中断状态的进程数。所以，它不仅包括了正在使用 CPU 的进程，还包括等待 CPU 和等待 I/O 的进程。

CPU 使用率，是单位时间内 CPU 繁忙情况的统计，跟平均负载并不一定完全对应。比如：

- CPU 密集型进程，使用大量 CPU 会导致平均负载升高，此时这两者是一致的；
- I/O 密集型进程，等待 I/O 也会导致平均**负载升高**，但 **CPU 使用率不一定很高**；
- 大量等待 CPU 的进程调度也会导致平均负载升高，此时的 CPU 使用率也会比较高。

4 CPU使用率监测命令

系统自带：ps、top

第三方安装：mpstat 、pidstat

安装对应的命令：**apt install stress sysstat**

其中sysstat 包括了mpstat 和pidstat 。

- mpstat 是一个常用的多核 CPU 性能分析工具，用来实时查看每个 CPU 的性能指标，以及所有 CPU 的平均指标。
- pidstat 是一个常用的进程性能分析工具，用来实时查看进程的 CPU、内存、I/O 以及上下文切换等性能指标。

压测命令：stress ，一个 Linux 系统压力测试工具，这里我们用作异常进程模拟平均负载升高的场景。

ps查找进程信息

ps用一于显示系统内的所有进程。

-l或l 采用详细的格式来显示进程状况。

查看帮助：ps --help all

常用方式：ps -elf 和ps -ef

输出:

```
root@ubuntu:/home/lqf# ps -ef
lqf       44540   3510  0 00:46 ?        00:00:03 /usr/bin/python3
/usr/bin/update-manager --no-update --no-focus-on-
lqf       56882   4182  0 01:02 pts/18   00:00:00 bash
lqf       57009  56882  0 01:06 pts/18   00:00:02 gdb ./ffmpeg_g
lqf       57119  57009  0 01:19 pts/18   00:00:00
/home/lqf/ffmpeg_sources/ffmpeg-4.2.1/ffmpeg_g -i source.200kbps.76
root      72736    909  0 15:22 ?        00:00:00 /sbin/dhclient -d -q -sf
/usr/lib/NetworkManager/nm-dhcp-helper -pf
root      73900   1134  0 15:51 ?        00:00:00 sshd: lqf [priv]
lqf       73957  73900  0 15:51 ?        00:00:00 sshd: lqf@pts/2
lqf       73958  73957  0 15:51 pts/2    00:00:00 -bash
root      73980   1134  0 15:51 ?        00:00:00 sshd: lqf [priv]
```

根据进程的名字或者其他信息，通过**grep命令找到目标进程**，也可以看到进程启动脚木的全路径。

更多参数说明:

```
###############################################################

常用参数：
-A     显示所有进程(等价于-e)(utility)
-a     显示一个终端的所有进程，除了会话引线
-N     忽略选择。
-d     显示所有进程，但省略所有的会话引线(utility)
-x     显示没有控制终端的进程，同时显示各个命令的具体路径。dx不可合用。(utility)
-p     pid 进程使用cpu的时间
-u     uid or username 选择有效的用户id或者是用户名
-g     gid or groupname 显示组的所有进程。
U      username 显示该用户下的所有进程，且显示各个命令的详细路径。如:ps U zhang;(utility)
-f     全部列出，通常和其他选项联用。如：ps -fa or ps -fx and so on.
-l     长格式(有F,wchan,C 等字段)
-j     作业格式
-o     用户自定义格式。
v      以虚拟存储器格式显示
s      以信号格式显示
-m     显示所有的线程
-H     显示进程的层次(和其它的命令合用，如：ps -Ha)(utility)
e      命令之后显示环境(如：ps -d e; ps -a e)(utility)
h      不显示第一行

==================================ps 的参数说明============================
```

l 长格式输出；
u 按用户名和启动时间的顺序来显示进程；
j 用任务格式来显示进程；
f 用树形格式来显示进程；
a 显示所有用户的所有进程(包括其它用户)。显示所有进程
-a 显示同一终端下的所有程序
x 显示无控制终端的进程；
r 显示运行中的进程；
ww 避免详细参数被截断；
-A 列出所有的进程
-w 显示加宽可以显示较多的资讯
-au 显示较详细的资讯
-aux 显示所有包含其他使用者的进程
-e 显示所有进程,环境变量
-f 全格式
-h 不显示标题
-l 长格式
-w 宽输出
a 显示终端上地所有进程,包括其他用户地进程
r 只显示正在运行地进程
x 显示没有控制终端地进程

常用的选项是组合是 aux 或 lax，还有参数 f 的应用。
pids 只列出进程标识符,之间运用逗号分隔，
该进程列表必须在命令行参数地最后一个选项后面紧接着给出，
中间不能插入空格。比如：ps -f1,4,5 显示的是进程ID为1,4,5的进程

下面介绍长命令行选项,这些选项都运用“--”开头：
--sort X[+|-] key [,[+|-] key [,…]] 从SORT KEYS段中选一个多字母键.“+”字符是可选地，
因为默认地方向就是按数字升序或者词典顺序，“-”字符是逆序排序(即降序).
比如： ps -jax -sort=uid,-ppid,+pid.
--help 显示帮助信息.
--version 显示该命令地版本信息.
在前面地选项说明中提到了排序键,接下来对排序键作进一步说明.
需要注意的是排序中运用的值是ps运用地内部值,并非仅用于某些输出格式地伪值.
排序键列表见下表.
============排序键列表==========================
c cmd 可执行地简单名称
C cmdline 完整命令行
f flags 长模式标志
g pgrp 进程地组ID
G tpgid 控制tty进程组ID
j cutime 累计用户时间
J cstime 累计系统时间
k utime 用户时间
K stime 系统时间
m min_flt 次要页错误地数量
M maj_flt 重点页错误地数量
n cmin_flt 累计次要页错误
N cmaj_flt 累计重点页错误
o session 对话ID
p pid 进程ID
P ppid 父进程ID
r rss 驻留大小
R resident 驻留页
s size 内存大小(千字节)
S share 共享页地数量
t tty tty次要设备号

```
T start_time 进程启动地时间
U uid   UID
u user   用户名
v vsize   总地虚拟内存数量(字节)
y priority 内核调度优先级
```

练习：

（1）检测是否有活动进程：

sudo ps -ef |grep "nginx: master process" |grep -v grep

（2）检测有几个同样的活动进程

sudo ps -ef |grep "nginx: master process" |grep -v grep |wc -l

top命令查询进程的cpu、内存信息

top命令用于查看活动进程的CPU和内存信息，能够实时显示系统中各个进程的资源占用情况，可以按照CPU、内存的使用情况和执行时间对进程进行排序。

使用方式：top

命令输出：

```
Tasks: 280 total,   1 running, 199 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.1 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  6143776 total,  1103656 free,  1211704 used,  3828416 buff/cache
KiB Swap:   998396 total,   998384 free,       12 used.  4461984 avail Mem

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

     1 root      20   0  185308   5828   3844 S   0.0  0.1   0:05.00 systemd

     2 root      20   0       0      0      0 S   0.0  0.0   0:00.04 kthreadd

     4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H

     ...
```

从输出可以看到整体的CPU占用率、CPU负载，以及进程占用CPU和内存等资源的情况。

我们可以用以下所示的top命令的快捷键对输出的显示信息进行转换。

- t：切换报示进程和CPU状态信息。
- n：切换显示内存信息。
- r：重新设置一个进程的优先级。系统提示用户输人需要改变的进程PID及需要设置的进程优先级，然后输入个正数值使优先级降低，反之则可以使该进程拥有更高的优先级，即是在原有基础上进行相加，默认优先级的值是100
- k：终止一个进程，系统将提示用户输入需要终止的进程PID o
- s：改变刷新的时间间隔。
- u：查看指定用户的进程。

练习：

top 命令查找cpu占用率最高的程序，找到对应的PID

top -Hp pid ，查看具体进程下的线程，比如

```
top - 18:01:11 up 1 day, 11:42,  4 users,  load average: 3.01, 1.46, 0.63
Threads:  12 total,   4 running,   8 sleeping,   0 stopped,   0 zombie
%Cpu(s): 35.1 us,  4.0 sy, 53.1 ni,  7.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0
MiB Mem :   2804.0 total,    125.1 free,    958.3 used,   1720.5 buff/cache
MiB Swap:   2048.0 total,   2047.2 free,      0.8 used.   1656.7 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
 30866 lqf       30  10 1242328 514848  31544 R  38.9  17.9   1:01.82 ffmpeg
 30867 lqf       30  10 1242328 514848  31544 R  37.9  17.9   1:03.77 ffmpeg
 30865 lqf       30  10 1242328 514848  31544 R  35.5  17.9   1:01.49 ffmpeg
 30784 lqf       20   0 1242328 514848  31544 R  30.9  17.9   1:11.59 ffmpeg
 30868 lqf       20   0 1242328 514848  31544 S  28.6  17.9   0:39.78 ffmpeg
```

mpstat

此命令用于实时监控系统CPU的一些统计信息，这些信息存放在/proc/stat文件中，在多核CPU系统里，不但能查看所有CPU的平均使用信息，还能查看某个特定CPU的信息。
使用方式：mpstat [-P {cpu|ALL}] [internal [count]]
当mpstat不带参数时，输出为从系统启动以来的平均值。

```
[root@ubuntu~]# mpstat -P ALL
```

Linux 3.10.0-957.5.1.el7.x86_64 (VM_0_ubuntu) 08/22/2019 *x86_64* (1 CPU)
05:00:44 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
05:00:44 PM all 1.67 0.00 1.65 4.21 0.00 0.02 0.00 0.00 0.00 92.45
05:00:44 PM 0 1.67 0.00 1.65 4.21 0.00 0.02 0.00 0.00 0.00 92.45
我们可以看到每个CPU核心的占用率、I/O等待、软中断、硬中断等。
输出各参数含义：

参数	含义
-P {cpu l ALL}	表示监控哪个CPU， cpu在[0,cpu个数-1]中取值
internal	相邻的两次采样的间隔时间
count	采样的次数，count只能和internal一起使用

使用mpstat -P ALL 5 2命令，表示每5秒产生一个报告，总共产生2个。

[root@ubuntu~]# mpstat -P ALL 5 2
Linux 3.10.0-957.5.1.el7.x86_64 (VM_0_ubuntu) 08/22/2019 *x86_64* (1 CPU)
05:04:39 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
05:04:44 PM all 1.41 0.00 1.21 3.42 0.00 0.00 0.00 0.00 0.00 93.96
05:04:44 PM 0 1.41 0.00 1.21 3.42 0.00 0.00 0.00 0.00 0.00 93.96
05:04:44 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
05:04:49 PM all 1.21 0.00 0.80 3.02 0.00 0.00 0.00 0.00 0.00 94.97
05:04:49 PM 0 1.21 0.00 0.80 3.02 0.00 0.00 0.00 0.00 0.00 94.97
Average: CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle

```
Average:    all   1.31   0.00   1.01   3.22   0.00   0.00   0.00   0.00   0.00  94.47
Average:      0   1.31   0.00   1.01   3.22   0.00   0.00   0.00   0.00   0.00  94.47
```

输出参数含义

当没有参数时，mpstat则显示系统启动以后所有信息的平均值。有interval时，第一行的信息自系统启动以来的平均信息。从第二行开始，输出为前一个interval时间段的平均信息。 输出各参数含义：

参数	释义	从/proc/stat获得数据
CPU	处理器ID	
%usr	在internal时间段里，用户态的CPU时间（%），不包含 nice值为负进程	usr/total*100
%nice	在internal时间段里，nice值为负进程的CPU时间（%）	nice/total*100
%sys	在internal时间段里，核心时间（%）	system/total*100
%iowait	在internal时间段里，硬盘IO等待时间（%）	iowait/total*100
%irq	在internal时间段里，硬中断时间（%）	irq/total*100
%soft	在internal时间段里，软中断时间（%）	softirq/total*100
%steal	显示虚拟机管理器在服务另一个虚拟处理器时虚拟CPU处在非自愿等待下花费时间的百分比	steal/total*100
%guest	显示运行虚拟处理器时CPU花费时间的百分比	guest/total*100
%gnice		gnice/total*100
%idle	在internal时间段里，CPU除去等待磁盘IO操作外的因为任何原因而空闲的时间闲置时间（%）	idle/total*100

pidstat

pidstat用于监控全部或指定的进程占用系统资源的情况，包括CPU、内存、磁盘I/O、程切换、线程数等数据。

- -u：表示查看cpu相关的性能指标
- -w：表示查看上下文切换情况，要想查看每个进程的详细情况，要加上-w
- -t：查看线程相关的信息，默认是进程的;常与-w结合使用(cpu的上下文切换包括进程的切换、线程的切换、中断的切换)
- -d：展示磁盘 I/O 统计数据
- -p：指明进程号

使用方式：pidstat [option] interval [count]

使用范例：pidstat -urd -p 进程ID

```
[root@VM_0_ubuntu ~]# pidstat -urd -p 24615
Linux 3.10.0-957.5.1.el7.x86_64 (VM_0_ubuntu)     08/22/2019  x86_64    (1 CPU)
```

输出CPU的使用信息 -u

```
03:48:12 PM   UID      PID    %usr %system  %guest    %CPU   CPU  Command
03:48:12 PM     0     24615    0.00    0.00    0.00    0.00     0  nginx
```

输出内存的使用信息 -r

```
03:48:12 PM   UID      PID  minflt/s  majflt/s    VSZ    RSS   %MEM  Command
03:48:12 PM     0     24615      0.00      0.00   58252  24940   1.32  nginx
```

输出磁盘I/O的使用信息 -d

```
03:48:12 PM   UID      PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
03:48:12 PM     0     24615      0.07      0.00      0.00  nginx
03:20:54 PM   UID      PID   cswch/s nvcswch/s  Command
03:20:54 PM     0     24615      0.03      0.00  nginx
```

CPU信息

- %usr　　#用户层任务正在使用的CPU百分比（with or without nice priority，NOT include time spent running a virtual processor）
- %system　#系统层正在执行的任务的CPU使用百分比
- %guest　　#运行虚拟机的CPU占用百分比
- %CPU　　#所有的使用的CPU的时间百分比
- CPU　　#处理器数量
- Command #命令

内存信息

- PID　　#进程号
- minflt/s　　#每秒次缺页错误次数(minor page faults)，次缺页错误次数意即虚拟内存地址映射成物理内存地址产生的page fault次数
- majflt/s　　#每秒主缺页错误次数(major page faults)，当虚拟内存地址映射成物理内存地址时，相应的page在swap中，这样的page fault为major page fault，一般在内存使用紧张时产生
- VSZ　　#该进程使用的虚拟内存(以kB为单位)
- RSS　　#该进程使用的物理内存(以kB为单位)
- %MEM　　#当前任务使用的有效内存的百分比
- Command　#任务的命令名

磁盘I/O

- PID　　#进程号
- kB_rd/s　#每秒此进程从磁盘读取的千字节数
- kB_wr/s　#此进程已经或者将要写入磁盘的每秒千字节数
- kB_ccwr/s #由任务取消的写入磁盘的千字节数
- Command #命令的名字

上下文切换

- PID #PID号
- cswch/s #每秒自动上下文切换
- nvcswch/s #每秒非自愿的上下文切换
- Command #命令

```
# 每隔5秒输出1组数据
$ pidstat -w 5
Linux 4.15.0 (ubuntu)  09/23/18  _x86_64_  (2 CPU)

08:18:26      UID       PID   cswch/s nvcswch/s  Command
08:18:31        0         1      0.20      0.00  systemd
08:18:31        0         8      5.40      0.00  rcu_sched
...
```

这个结果中有两列内容是我们的重点关注对象。一个是 cswch ，表示每秒自愿上下文切换（voluntary context switches）的次数，另一个则是 nvcswch ，表示每秒非自愿上下文切换（non voluntary context switches）的次数。

这两个概念一定要牢牢记住，因为它们意味着不同的性能问题：

- 所谓自愿上下文切换，是指进程无法获取所需资源，导致的上下文切换。比如说， I/O、内存等系统资源不足时，就会发生自愿上下文切换。
- 而非自愿上下文切换，则是指进程由于时间片已到等原因，被系统强制调度，进而发生的上下文切换。比如说，大量进程都在争抢 CPU 时，就容易发生非自愿上下文切换。

场景一：CPU 密集型进程

1. 模拟一个 CPU 使用率 100% 的场景

```
stress --cpu 1 --timeout 600
```

2. 在第二个终端运行 uptime 查看平均负载的变化情况

```
lqf@ubuntu:~$ uptime
 16:01:48 up 1 day, 28 min,  4 users,  load average: 0.61, 0.19, 0.06
```

3. 第三个终端运行 mpstat 查看 CPU 使用率的变化情况

```
lqf@ubuntu:~$ mpstat -P ALL 5
Linux 4.15.0-142-generic (ubuntu)   11/02/2021  _x86_64_    (4 CPU)

04:02:14 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest
%gnice   %idle
04:02:19 PM  all   25.03    0.00    0.00    0.00    0.00    0.00    0.00    0.00
   0.00   74.97
04:02:19 PM    0  100.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
   0.00    0.00
04:02:19 PM    1    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
   0.00  100.00
04:02:19 PM    2    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
   0.00  100.00
04:02:19 PM    3    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
   0.00  100.00
```

正好有一个 CPU 的使用率为 100%，但它的 iowait 只有 0。这说明，平均负载的升高正是由于 CPU 使用率为 100% 。

4. 查看是哪个进程导致了cpu使用率为100%

```
lqf@ubuntu:~$ pidstat -u 5 1
Linux 4.15.0-142-generic (ubuntu)   11/02/2021  _x86_64_    (4 CPU)

04:05:44 PM   UID       PID    %usr %system  %guest    %CPU   CPU  Command
04:05:49 PM   122      1236    0.00    0.20    0.00    0.20     3  mysqld
04:05:49 PM   999      3098    0.20    0.00    0.00    0.20     1  redis-
server
04:05:49 PM  1000    103642  100.00    0.00    0.00  100.00     2  stress

Average:      UID       PID    %usr %system  %guest    %CPU   CPU  Command
Average:      122      1236    0.00    0.20    0.00    0.20     -  mysqld
Average:      999      3098    0.20    0.00    0.00    0.20     -  redis-
server
Average:     1000    103642  100.00    0.00    0.00  100.00     -  stress
```

场景二：I/O 密集型进程

1. 第一个终端运行 stress 命令模拟 I/O 压力

```
stress -i 1 --timeout 600
```

2. 第二个终端运行 uptime 查看平均负载的变化情况

```
uptime
```

3. 第三个终端运行 mpstat 查看 CPU 使用率的变化情况

```
mpstat -P ALL 5 1
```

4. 查看是哪个进程，导致 iowait较高

```
pidstat -u 5 1
```

场景三：大量进程的场景

1. 第一个终端模拟8个进程

```
stress -c 8 --timeout 600
```

2. 第二个终端uptime

```
uptime
```

3. 使用pidstat查看进程情况

```
lqf@ubuntu:~$ pidstat -u 5 1
Linux 4.15.0-142-generic (ubuntu)   11/02/2021  _x86_64_    (4 CPU)

04:13:57 PM   UID       PID    %usr %system  %guest    %CPU   CPU  Command
04:14:02 PM  1000      3748    0.00    0.20    0.00    0.20     3  unity-
panel-ser
04:14:02 PM  1000    103715   50.00    0.00    0.00   50.00     3  stress
04:14:02 PM  1000    103716   49.80    0.00    0.00   49.80     2  stress
04:14:02 PM  1000    103717   49.80    0.00    0.00   49.80     2  stress
04:14:02 PM  1000    103718   49.80    0.00    0.00   49.80     1  stress
04:14:02 PM  1000    103719   49.60    0.00    0.00   49.60     3  stress
04:14:02 PM  1000    103720   49.80    0.00    0.00   49.80     0  stress
04:14:02 PM  1000    103721   49.80    0.00    0.00   49.80     1  stress
04:14:02 PM  1000    103722   49.80    0.00    0.00   49.80     0  stress

Average:      UID       PID    %usr %system  %guest    %CPU   CPU  Command
Average:     1000      3748    0.00    0.20    0.00    0.20     -  unity-
panel-ser
Average:     1000    103715   50.00    0.00    0.00   50.00     -  stress
Average:     1000    103716   49.80    0.00    0.00   49.80     -  stress
Average:     1000    103717   49.80    0.00    0.00   49.80     -  stress
Average:     1000    103718   49.80    0.00    0.00   49.80     -  stress
Average:     1000    103719   49.60    0.00    0.00   49.60     -  stress
Average:     1000    103720   49.80    0.00    0.00   49.80     -  stress
Average:     1000    103721   49.80    0.00    0.00   49.80     -  stress
Average:     1000    103722   49.80    0.00    0.00   49.80     -  stress
```

可以看出，8 个进程在争抢 4 个 CPU，这些超出 CPU 计算能力的进程，最终导致 CPU 过载。

2.2 CPU上下文切换

2.2.1 什么是CPU上下文切换

所谓的上下文切换，就是把上一个任务的寄存器和计数器保存起来，然后加载新任务的寄存器和计数器，最后跳转到新任务的位置开始执行新任务。

根据任务的不同，CPU 的上下文切换就可以分为几个不同的场景，也就是进程上下文切换、线程上下文切换以及中断上下文切换。

2.2.2 有哪些上下文切换

1 系统调用上下文切换

linux 进程既可以在用户空间运行，又可以在内核空间中运行。
当它在用户空间运行时，被称为进程的用户态；当它进入进入内核空间的时候，被称为进程的内核态
从用户态到内核态的转变过程，需要通过系统调用来完成
CPU 寄存器里原来的指令位置是在用户态。但是为了执行内核态代码，需要先把用户态的位置保存起来，然后寄存器更新为内核态指令的新位置。最后跳转到内核态运行内核任务。

当系统调用结束后，CPU 寄存器需要恢复原来保存的用户态位置，然后再切换到用户空间，继续运行进程。一次系统调用发生了两次 CPU 上下文切换！

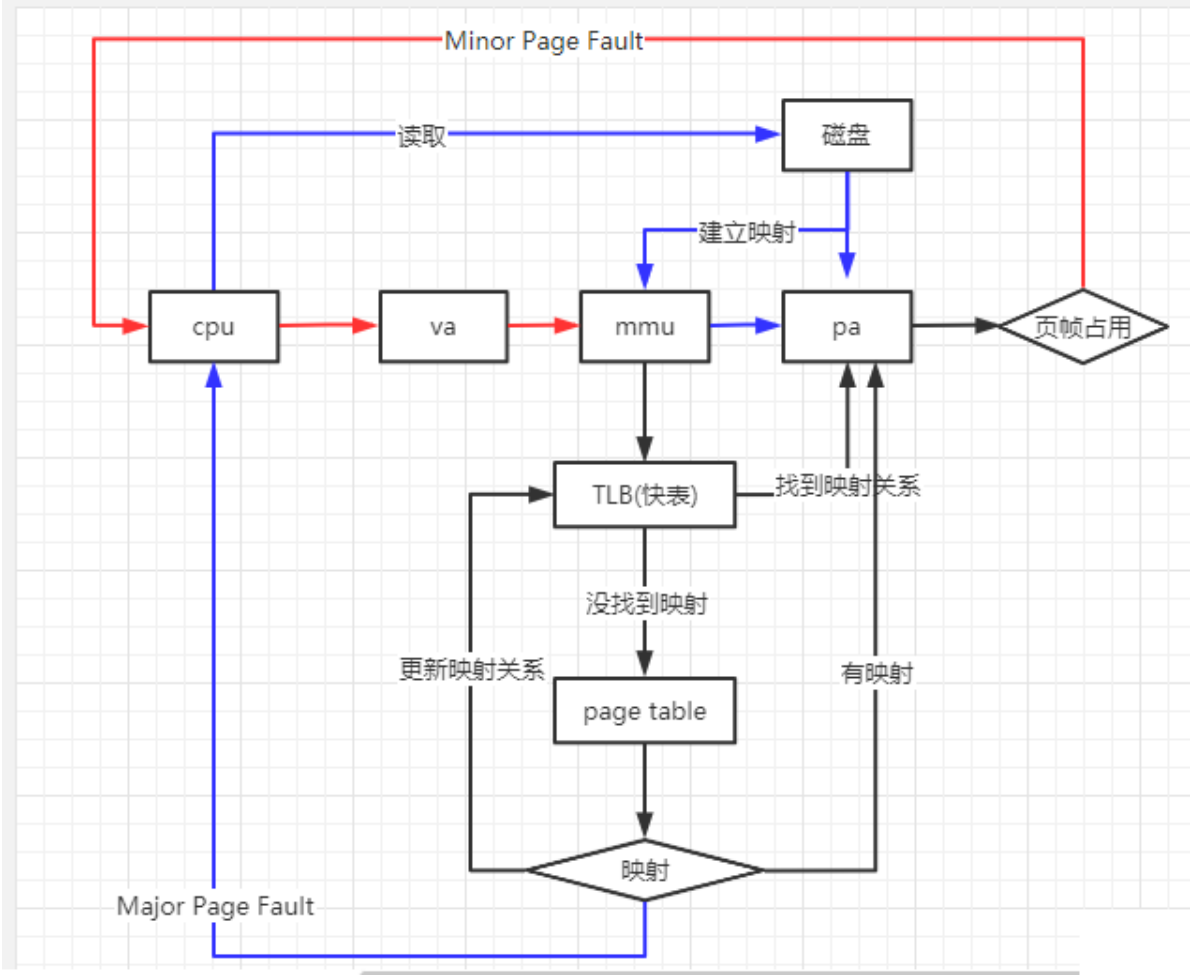
系统调用过程中对用户态的资源没有任何影响，也不会切换进程，所以也称为特权模式切换.

应用层（用户态）send，　真正发送数据 内核态　recv

2 进程上下文切换

进程是由内核来管理和调度的，所以进程的切换只发生在内核态。进程的上下文不仅包括了虚拟内存、栈、全局变量等用户空间的资源，还包括了内核堆栈、寄存器等内核空间的状态。

进程的上下文切换在保存当前进程的内核状态和 CPU 寄存器之前，需要先把该进程的虚拟内存、栈等保存下来；而加载了下一进程的内核态后再刷新进程的虚拟内存映射关系和用户栈，刷新虚拟内存映射就涉及到 TLB 快表 (虚拟地址缓存)，因此会影响内存的访问速度。



单次进程上下文切换的 CPU 时间在几十纳秒到数微秒之间。特别是在进程上下文切换次数较多的情况下，很容易导致 CPU 将大量时间耗费在寄存器、内核栈以及虚拟内存等资源的保存和恢复上，进而影响 cpu 的实际使用率。

进程上下文切换的原因
其一，为了保证所有进程可以得到公平调度，CPU 时间被划分为一段段的时间片，这些时间片再被轮流分配给各个进程。这样，当某个进程的时间片耗尽了，就会被系统挂起，切换到其它正在等待 CPU 的进程运行。（被动切换）
其二，进程在系统资源不足，这个时候进程也会被挂起，并由系统调度其他进程运行。（主动切换）
其三，当进程通过睡眠函数 sleep 这样的方法将自己主动挂起时，自然也会重新调度。
其四，当有优先级更高的进程运行时，为了保证高优先级进程的运行，当前进程会被挂起，由高优先级

进程来运行。
第五，发生硬件中断时，CPU 上的进程会被中断挂起，转而执行内核中的中断服务程序。

3 线程上下文切换

线程与进程的区别在于：线程是调度的基本单位，而进程是资源分配基本单位。内核中的任务调度，实际调度的是线程；而进程只是给线程提供了虚拟内存、全局变量等资源。
1：当进程只有一个线程时，可以认为进程就等于线程
2：当进程拥有多个线程时，共享虚拟内存和全局变量等资源。这些资源在上下文切换时不需要修改
3：线程也有自己的私有数据，比如栈和寄存器等，这些在上下文切换时需要保存

线程的上下文切换分为两种
1.前后两个线程属于不同进程。此时，因为资源不共享，**所以切换过程就跟进程上下文切换是一样。**
2.前后两个线程属于同一个进程。此时，因为虚拟内存是共享的，所以在切换时，虚拟内存这些资源就保持不动，只需要切换线程的私有数据、寄存器等不共享的数据

4 中断上下文切换

为了快速响应硬件的事件，中断处理会打断进程的正常调度和执行，转而调用中断处理程序，响应设备事件。而在打断其他进程时，就需要将进程当前的状态保存下来，这样在中断结束后，进程仍然可以从原来的状态恢复运行。

跟进程上下文不同，**中断上下文切换并不涉及到进程的用户态。**中断上下文只包括内核态中断服务程序执行所必需的状态，包括 CPU 寄存器、内核堆栈、硬件中断参数等

2.2.3 怎么查看上下文切换

过多的上下文切换，会把 CPU 时间消耗在寄存器、内核栈以及虚拟内存等数据的保存和恢复上，缩短进程真正运行的时间，成了系统性能大幅下降的一个元凶。

可以使用 **vmstat 这个工具**，来查询系统的上下文切换情况。此命令显示关于内核线程、虚拟内存、磁盘I/O 、陷阱和CPU占用率的统计信息。

vmstat

vmstat [options] [delay [count]]

```
# 每隔5秒输出1组数据
lqf@ubuntu:~$ vmstat 5
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache      si   so    bi    bo   in  cs   us  sy id
wa st
17  0     12 1179508 469892 3315788    0    0     6    20   49   74  0   0   99
0  0
16  0     12 1179376 469892 3315788    0    0     0     0 1048  678 100  0   0
0  0
16  0     12 1179376 469892 3315788    0    0     0     0 1045  675 100  0   0
0  0
16  0     12 1179376 469892 3315788    0    0     0     0 1047  685 100  0   0
0  0
```

需要注意如下内容。

- **r（Running or Runnable）**是就绪队列的长度，也就是正在运行和等待 CPU 的进程数。
- b（Blocked）则是处于不可中断睡眠状态的进程数。

- buff：是I/O系统存储的磁盘块文件的元数据的统计信息。
- cache：是操作系统用来缓存磁盘数据的缓冲区，操作系统会自动一调节这个参数，在内存紧张时操作系统会减少cache的占用空间来保证其他进程可用。
- si和so较大时，**说明系统频繁使用交换区**，应该查看操作系统的内存是否够用。
- bi和bo代表I/O活动，根据其大小可以知道磁盘I/O的负载情况。
- **in（interrupt）**则是每秒中断的次数。
- **cs**：参数表示线程环境的切换次数，此数据太大时表明线程的同步机制有问题。

2.2.4 案例分析

上下文切换频率是多少次才是正常的?

使用sysbench 来模拟系统多线程调度切换的情况。

有命令，都默认以 root 用户运行，运行 sudo su root 命令切换到 root 用户。

stress

1. 测试工具sysbench的安装：

```
apt install sysbench
```

2. 使用vmstat查看空闲系统的上下文切换次数

```
// 间隔1秒后输出1组数据
root@ubuntu:/home/lqf# vmstat 1 1
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-
----
 r  b   swpd   free   buff  cache      si   so    bi    bo   in   cs us sy
id wa st
 0  0     12 1173896 470200 3318136    0    0     6     20   49   74  0  0
99  0  0
```

3. 在另一个终端开始压测

```
// 以10个线程运行5分钟的基准测试，模拟多线程切换的问题
sysbench --num-threads=10 --max-time=300 --max-requests=10000000 --
test=threads run
```

4. 使用vmstat查看

```
# 每隔1秒输出1组数据（需要Ctrl+C才结束）
$ vmstat 1

procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-
----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id
wa st
 9  0      0 206356  67344 548616    0    0     0    14 1666 1596479  7 93
0  0  0
 7  0      0 206356  67344 548612    0    0     0     0 1680 1542861 10 90
0  0  0
 7  0      0 206356  67344 548612    0    0     0     0 1700 1550600 11 90
0  0  0
 7  0      0 206356  67344 548612    0    0     0     0 1704 1586618  9 91
0  0  0
 8  0      0 206356  67348 548612    0    0     0     8 1700 1493243  8 93
0  0  0
```

- cs 列：的上下文切换次数从之前的 35 骤然上升到了 139 万。
- r 列：**就绪队列的长度已经到了 8**，远远超过了系统 CPU 的个数 2，所以肯定会有大量的 CPU 竞争。
- us（user）和 sy（system）列：这两列的 CPU 使用率加起来上升到了 100%，其中系统 CPU 使用率，也就是 sy 列高达 84%，说明 CPU 主要是被内核占用了。
- in 列：**中断次数也上升到了 1 万左右**，说明中断处理也是个潜在的问题。

综合这几个指标，我们可以知道，系统的就绪队列过长，也就是正在运行和等待 CPU 的进程数过多，导致了大量的上下文切换，而上下文切换又导致了系统 CPU 的占用率升高。

5. **使用pidstat查看**

什么进程导致了这些问题呢?
在第三个终端再用 pidstat 来看一下， CPU 和进程上下文切换的情况：

```
# 每隔1秒输出1组数据（需要 Ctrl+C 才结束）
# -w参数表示输出进程切换指标，而-u参数则表示输出CPU使用指标
$ pidstat -w -u 1
```

从 pidstat 的输出你可以发现，CPU 使用率的升高果然是 sysbench 导致的，它的 CPU 使用率已经达到了 100%。但上下文切换则是来自其他进程，包括非自愿上下文切换频率最高的 pidstat ，以及自愿**上下文切换频率最高的内核线程 kworker 和 sshd**。

怪异的事儿：pidstat 输出的上下文切换次数，加起来也就几百，比 vmstat 的 139 万明显小了太多。这是怎么回事呢？难道是工具本身出了错吗？别着急，在怀疑工具之前，我们再来回想一下，前面讲到的几种上下文切换场景。其中有一点提到， Linux 调度的基本单位实际上是线程，而我们的场景 sysbench 模拟的也是线程的调度问题，那么，是不是 pidstat 忽略了线程的数据呢?

通过运行 man pidstat ，你会发现，pidstat 默认显示进程的指标数据，**加上 -t 参数后，才会输出线程的指标**。所以，我们可以在第三个终端里， Ctrl+C 停止刚才的 pidstat 命令，再加上 -t 参数，重试一下看看：

```
# 每隔1秒输出一组数据（需要 Ctrl+C 才结束）
# -wt 参数表示输出线程的上下文切换指标
$ pidstat -wt 1

08:14:05 UID TGID TID cswch/s nvcswch/s Command
...
08:14:05 0 10551 - 6.00 0.00 sysbench
08:14:05 0 - 10551 6.00 0.00 |__sysbench
08:14:05 0 - 10552 18911.00 103740.00 |__sysbench
08:14:05 0 - 10553 18915.00 100955.00 |__sysbench
08:14:05 0 - 10554 18827.00 103954.00 |__sysbench
```

现在你就能看到了，虽然 sysbench 进程（也就是主线程）的上下文切换次数看起来并不多，**但它的子线程的上下文切换次数却有很多。**看来，上下文切换罪魁祸首，还是过多的 sysbench 线程，这也解释了：**对于线程池而言，并不是开启的线程越多越好。**

每秒上下文切换多少次才算正常呢？这个数值其实取决于系统本身的 CPU 性能。 如果系统的上下文切换次数比较稳定，那么从数百到一万以内，都应该算是正常的。但当上下文切换次数超过一万次，或者切换次数出现数量级的增长时，就很可能已经出现了性能问题。

这时，你还需要根据上下文切换的类型，再做具体分析。比方说：

1. 自愿上下文切换变多了，说明进程都在等待资源，有可能发生了 I/O 等其他问题；
2. 非自愿上下文切换变多了，说明进程都在被强制调度，也就是都在争抢 CPU，说明 CPU 的确成了瓶颈；
3. 中断次数变多了，说明 CPU 被中断处理程序占用，还需要通过查看 /proc/interrupts 文件来分析具体的中断类型。

2.3 遇到CPU利用率高该如何排查

遇到CPU使用率高时，首先确认CPU是消耗在哪一块，如果是内核态占用CPU较高：

1. %iowait 高，这时要重点关注磁盘IO的相关操作，是否存在不合理的写日志操作，数据库操作等；
2. %soft或%cs 高，观察CPU负载是否较高、网卡流量是否较大，可不可以精简数据、代码在是否在多线程操作上存在不合适的中断操作等；
3. %steal 高，这种情况一般发生在虚拟机上，这时要查看宿主机是否资源超限；

如果是用户态较高，且没有达到预期的性能，说明应用程序需要优化。

2.3.1 根据指标查找工具

根据指标找工具（CPU性能）		
性能指标	**工具**	**说明**
平均负载	uptime top	uptime最简单； top提供了更全的指标
系统整体CPU使用率	vmstat mpstat top sar /proc/stat	top、vmstat、mpstat 只可以动态查看，而 sar 还可以记录历史数据 /proc/stat是其他性能工具的数据来源
进程CPU使用率	top pidstat ps htop atop	top和ps可以按CPU使用率给进程排序，而pidstat只显示实际用了CPU的进程 htop和atop以不同颜色显示更直观
系统上下文切换	vmstat	除了上下文切换次数， 还提供运行状态和不可中断状态进程的数量
进程上下文切换	pidstat	注意加上 -w 选项
软中断	top /proc/softirqs mpstat	top提供软中断CPU使用率， 而/proc/softirqs和mpstat提供了各种软中断在每个CPU上的运行次数
硬中断	vmstat /proc/interrupts	vmstat提供总的中断次数， 而/proc/interrupts提供各种中断在每个CPU上运行的累积次数
网络	dstat sar tcpdump	dstat和sar提供总的网络接收和发送情况，而tcpdump则是动态抓取正在进行的网络通讯
I/O	dstat sar	dstat和sar都提供了I/O的整体情况
CPU 个数	/proc/cpuinfo lscpu	lscpu更直观
事件剖析	perf execsnoop	perf可以用来分析CPU的缓存以及内核调用链，execsnoop用来监控短时进程

2.3.2 根据工具查指标

根据工具查指标（CPU性能）	
性能工具	**CPU性能指标**
uptime	平均负载
top	平均负载、运行队列、整体的CPU使用率以及每个进程的状态和CPU使用率
htop	top增强版，以不同颜色区分不同类型的进程，更直观
atop	CPU、内存、磁盘和网络等各种资源的全面监控
vmstat	系统整体的CPU使用率、上下文切换次数、中断次数， 还包括处于运行和不可中断状态的进程数量
mpstat	每个CPU的使用率和软中断次数
pidstat	进程和线程的CPU使用率、中断上下文切换次数
/proc/softirqs	软中断类型和在每个CPU上的累积中断次数
/proc/interrupts	硬中断类型和在每个CPU上的累积中断次数
ps	每个进程的状态和CPU使用率
pstree	进程的父子关系
dstat	系统整体的CPU使用率
sar	系统整体的CPU使用率，包括可配置的历史数据
strace	进程的系统调用
perf	CPU性能事件剖析，如调用链分析、CPU缓存、CPU调度等
execsnoop	监控短时进程

3 内存性能监控

3.1 内存是什么-虚拟内存和物理内存

该文原文链接参考：https://blog.csdn.net/lvyibin890/article/details/82217193

操作系统有**虚拟内存与物理内存**的概念。在很久以前，还没有虚拟内存概念的时候，程序寻址用的都是物理地址。程序能寻址的范围是有限的，这取决于CPU的地址线条数。比如在32位平台下，寻址的范围是2^32也就是4G。并且这是固定的，如果没有虚拟内存，且每次开启一个进程都给4G的物理内存，就可能会出现很多问题：

- 因为我的物理内存时有限的，当有多个进程要执行的时候，都要给4G内存，很显然你内存小一点，这很快就分配完了，于是没有得到分配资源的进程就只能等待。当一个进程执行完了以后，再将等待的进程装入内存。这种频繁的装入内存的操作是很没效率的

- 由于指令都是直接访问物理内存的，那么我这个进程就可以修改其他进程的数据，甚至会修改内核地址空间的数据，这是我们不想看到的
- 因为内存时随机分配的，所以程序运行的地址也是不正确的

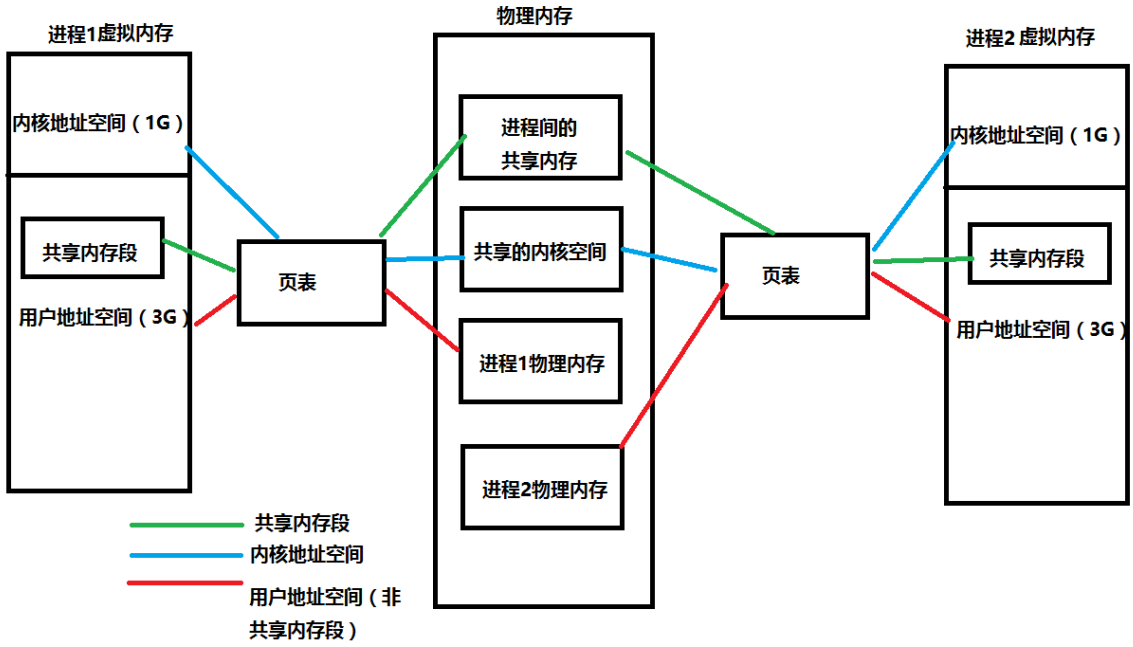
一个进程运行时都会得到4G的虚拟内存。这个虚拟内存你可以认为，每个进程都认为自己拥有4G的空间，这只是每个进程认为的，但是实际上，在虚拟内存对应的物理内存上，可能只对应的一点点的物理内存，实际用了多少内存，就会对应多少物理内存。

进程得到的这4G虚拟内存是一个连续的地址空间（这也只是进程认为），而实际上，它通常是被分隔成多个物理内存碎片，还有一部分存储在外部磁盘存储器上，在需要时进行数据交换。

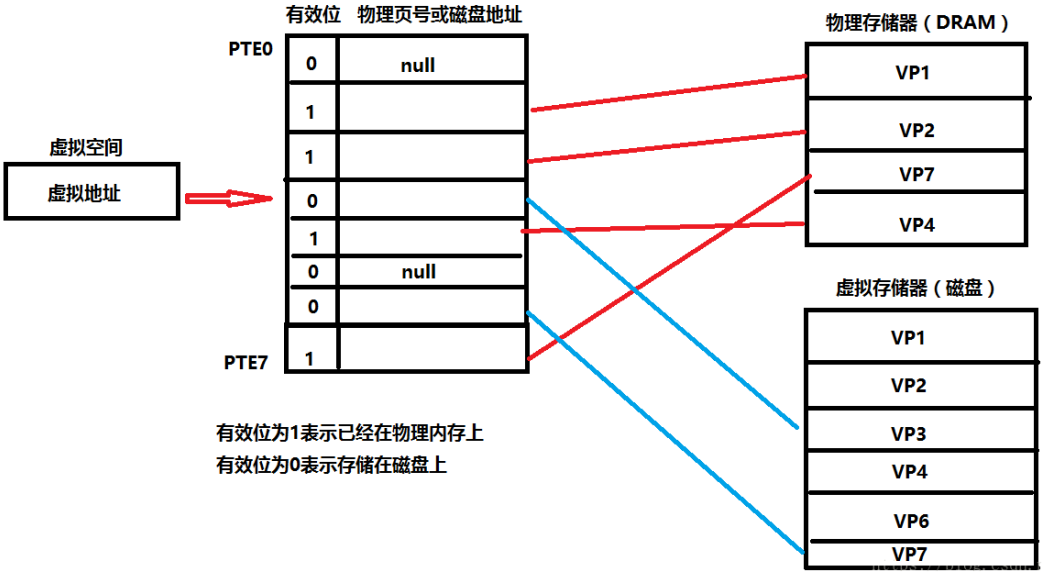
进程开始要访问一个地址，它可能会经历下面的过程

1. 每次我要访问地址空间上的某一个地址，都需要把地址翻译为实际物理内存地址
2. 所有进程共享这整一块物理内存，每个进程只把自己目前需要的虚拟地址空间映射到物理内存上
3. 进程需要知道哪些地址空间上的数据在物理内存上，哪些不在（可能这部分存储在磁盘上），还有在物理内存上的哪里，这就需要通过页表来记录
4. 页表的每一个表项分两部分，第一部分记录此页是否在物理内存上，第二部分记录物理内存页的地址（如果在的话）
5. 当进程访问某个虚拟地址的时候，就会先去看页表，如果发现对应的数据不在物理内存上，就会发生缺页异常
6. 缺页异常的处理过程，操作系统立即阻塞该进程，并将硬盘里对应的页换入内存，然后使该进程就绪，如果内存已经满了，没有空地方了，那就找一个页覆盖，至于具体覆盖的哪个页，就需要看操作系统的页面置换算法是怎么设计的了。

关于虚拟内存与物理内存的联系



页表的工作原理如下图



1. 我们的cpu想访问虚拟地址所在的虚拟页(VP3)，根据页表，找出页表中第三条的值.判断有效位。如果有效位为1，DRMA缓存命中，根据物理页号，找到物理页当中的内容，返回。
2. 若有效位为0，参数缺页异常，调用内核缺页异常处理程序。内核通过页面置换算法选择一个页面作为被覆盖的页面，将该页的内容刷新到磁盘空间当中。然后把VP3映射的磁盘文件缓存到该物理页上面。然后页表中第三条，有效位变成1，第二部分存储上了可以对应物理内存页的地址的内容。
3. 缺页异常处理完毕后，返回中断前的指令，重新执行，此时缓存命中，执行1。
4. 将找到的内容映射到告诉缓存当中，CPU从告诉缓存中获取该值，结束。

当每个进程创建的时候，内核会为进程分配4G的虚拟内存，当进程还没有开始运行时，这只是一个内存布局。实际上并不立即就把虚拟内存对应位置的程序数据和代码（比如.text .data段）拷贝到物理内存中，只是建立好虚拟内存和磁盘文件之间的映射就好（叫做存储器映射）。这个时候数据和代码还是在磁盘上的。当运行到对应的程序时，进程去寻找页表，发现页表中地址没有存放在物理内存上，而是在磁盘上，于是发生缺页异常，于是将磁盘上的数据拷贝到物理内存中。

另外在进程运行过程中，要通过malloc来动态分配内存时，也只是分配了虚拟内存，即为这块虚拟内存对应的页表项做相应设置，当进程真正访问到此数据时，才引发缺页异常。

可以认为虚拟空间都被映射到了磁盘空间中（事实上也是按需要映射到磁盘空间上，通过mmap，mmap是用来建立虚拟空间和磁盘空间的映射关系的）

3.2 内存中的buffer和cache

```
root@ubuntu:/home/lqf# free -h
              total        used        free      shared  buff/cache   available
Mem:           5.9G        1.2G        1.1G         36M        3.6G        4.3G
Swap:          974M         12K        974M
```

这个界面包含了物理内存 Mem 和交换分区 Swap 的具体使用情况，比如总内存、已用内存、缓存、可用内存等。其中缓存是 Buffer 和 Cache 两部分的总和。

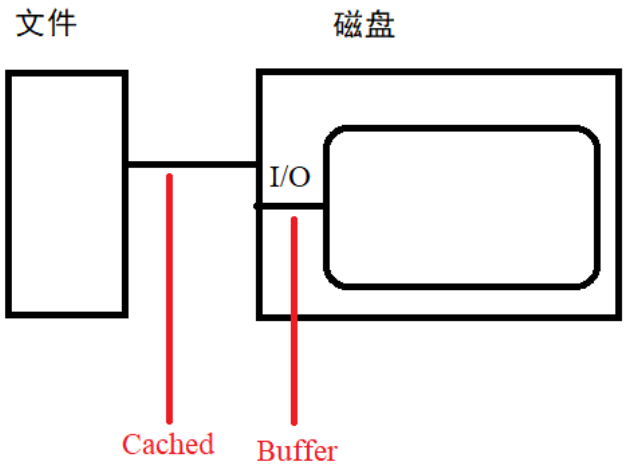
1. **Mem** 行(第二行)是内存的使用情况。
2. **Swap** 行(第三行)是交换空间的使用情况。

- **total** 列显示系统总的可用物理内存和交换空间大小。
- **used** 列显示已经被使用的物理内存和交换空间。
- **free** 列显示还有多少物理内存和交换空间可用使用。

- **shared** 列显示被共享使用的物理内存大小。
- **buff/cache** 列显示被 buffer 和 cache 使用的物理内存大小。
- **available** 列显示还可以被应用程序使用的物理内存大小。

对于buffer和cache的区别

1. Buffers 是对**原始磁盘块的临时存储**，也就是用来**缓存磁盘的数据**，通常不会特别大（20MB 左右）。这样，内核就可以把分散的写集中起来，统一优化磁盘的写入，比如可以把多次小的写合并成单次大的写等等。
2. Cached 是**从磁盘读取文件的页缓存**，也就是用来**缓存从文件读取的数据。**这样，下次访问这些文件数据时，就可以直接从内存中快速获取，而不需要再次访问缓慢的磁盘。



1. Buffer 既可以用作“将要写入磁盘数据的缓存”，也可以用作“从磁盘读取数据的缓存”。
2. Cache 既可以用作“从文件读取数据的页缓存”，也可以用作“写文件的页缓存”。

Buffer 是对磁盘数据的缓存，而 Cache 是文件数据的缓存，它们既会用在读请求中，也会用在写请求中。

在读写普通文件时，会经过文件系统，由文件系统负责与磁盘交互；而读写磁盘或者分区时，就会跳过文件系统，也就是所谓的“裸I/O“。这两种读写方式所使用的缓存是不同的，也就是文中所讲的 Cache 和 Buffer 区别。

3.3 内存优化监测

Buffer 和 Cache 分别缓存的是对磁盘和文件系统的读写数据。

- 从写的角度来说，不仅可以优化磁盘和文件的写入，对应用程序也有好处，应用程序可以在数据真正落盘前，就返回去做其他工作。
- 从读的角度来说，不仅可以提高那些频繁访问数据的读取速度，也降低了频繁 I/O 对磁盘的压力

cpu亲和性

批量数据写入和读取。

和实时性的问题。

讲日志

4 文件IO性能监控

写入速度

读取速度

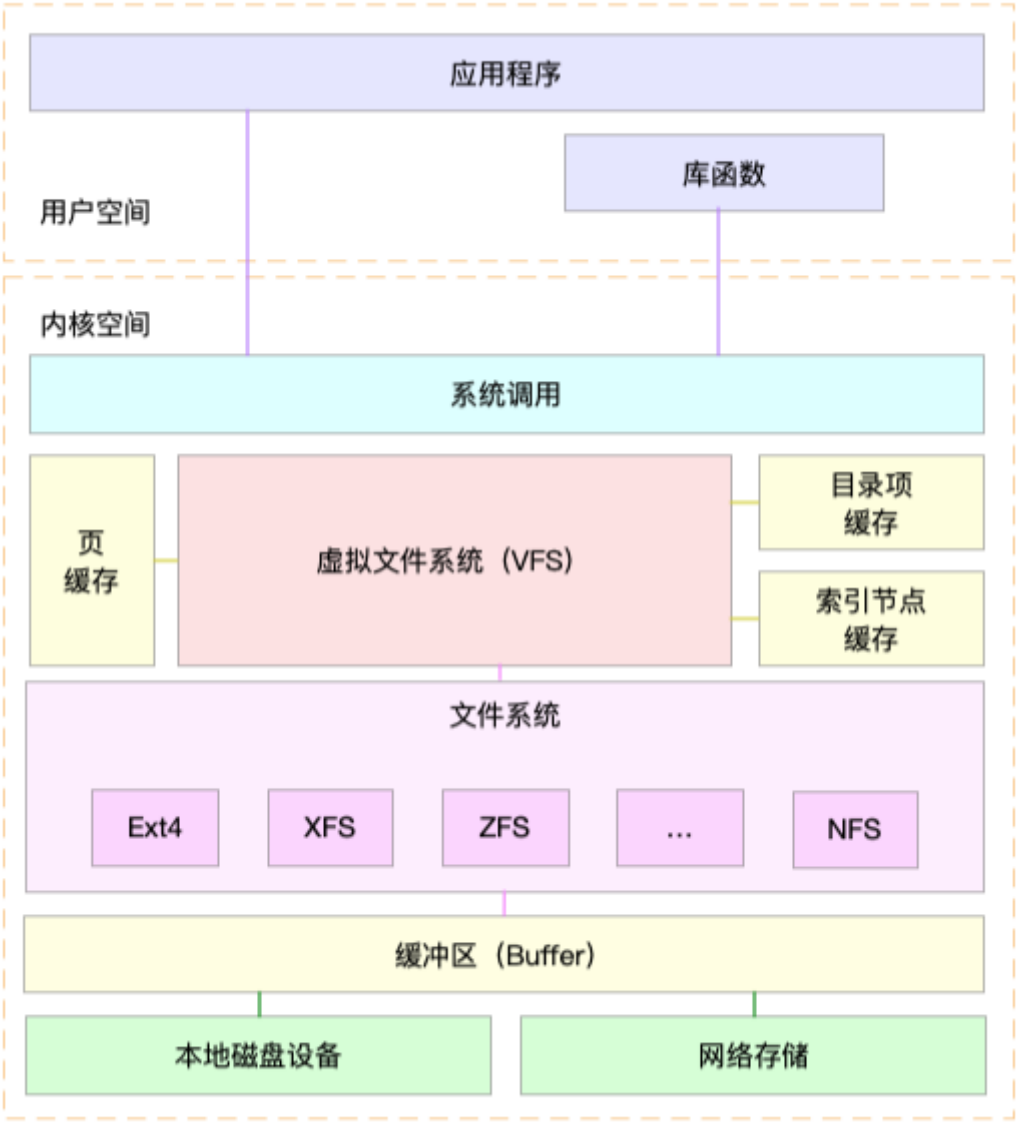
写入次数

读取次数
io等待时间 时间越大说明文件操作频繁

磁盘的统计参数

- tps：该设备每秒的传输次数（Indicate the number of transfers per second that were issued to the device.）。"一次传输"意思是"一次I/O请求"。
- 多个逻辑请求可能会被合并为"一次I/O请求"。"一次传输"请求的大小是未知的。
- kB_read/s：每秒从设备（drive expressed）读取的数据量；
- kB_wrtn/s：每秒向设备（drive expressed）写入的数据量；
- kB_read：读取的总数据量；
- kB_wrtn：写入的总数量数据量；这些单位都为Kilobytes。

系统调用、VFS、缓存、文件系统以及块存储之间的关系如下图所示：



Linux 内核在用户进程和文件系统的中间，又引入了一个抽象层，也就是虚拟文件系统 VFS（Virtual File System）。

I/O 指的是相对内存而言的 input 和 output。从文件、数据库、网络向内存中写入数据叫做 input；从内存向文件、数据库、网络中输出数据叫做 output。Linux 系统 I/O 分为内核准备数据和将数据从内核拷贝到用户空间两个阶段

4.1 I/O 的两种方式(缓存 I/O 和直接 I/O)

1 缓存 I/O

缓存 I/O 又被称作标准 I/O，大多数文件系统的默认 I/O 操作都是缓存 I/O。在 Linux 的缓存 I/O 机制中，数据先从磁盘复制到**内核空间的缓冲区**，然后从内核空间缓冲区复制到应用程序的地址空间(用户空间)。

- **读操作**：操作系统检查内核空间的缓冲区有没有需要的数据，如果已经缓存了，那么就直接从缓存中返回，也就是将数据复制到应用程序的用户空间；否则从磁盘中读取数据至内核空间的缓冲区，再将内核空间缓冲区的数据返回。
- **写操作**：将数据从用户空间复制到**内核空间的缓冲区**，这时对用户程序来说写操作就已经完成。至于什么时候将数据从内核空间写到磁盘中，这步由操作系统决定，除非显示地调用了 **sync 同步命令。**

缓存 I/O 的优点：

1. 在一定程度上分离了内核空间和用户空间，保护系统本身的运行安全；
2. 可以减少读盘的次数，从而提高性能。

缓存 I/O 的缺点：

1. 在缓存 I/O 机制中，DMA 方式可以将数据直接从磁盘读到内核空间的页缓存中，或者将数据从内核空间的页缓存中直接写回到磁盘上，而不能直接在应用程序地址空间(用户空间)和磁盘之间进行数据传输。这样，数据在传输过程中需要在应用程序地址空间(用户空间)和页缓存(内核空间)之间进行多次数据拷贝操作，这些数据拷贝操作所带来的 CPU 以及内存开销是比较大的。

2 直接 I/O

直接 IO 就是应用程序直接访问磁盘，**而不经过内核缓冲区**，这样做的目的是减少一次从内核缓冲区到用户程序地址空间的数据复制操作。

例如数据库管理系统这类应用，它们更倾向于选择自己的**缓存机制**，因为数据库管理系统往往比操作系统更了解数据库中存放的数据。数据库管理系统可以提供一种更加高效的缓存机制来提高数据库中存取数据的性能。

nginx
磁盘的文件 -》 sendfile -> 网络io

4.2 监控磁盘I/O的命令

1 iostat IO状态

该命令用于监控CPU占用率、平均负载值及I/O读写速度等。

参数介绍

iostat [-c] [-d] [-h] [-N] [-k | -m] [-t] [-V] [-x] [-z] [device [...] | ALL] [-p [device [,...] | ALL]] [interval [count]]

常用参数

1. `-c: 输出cpu统计信息`
2. `-d: 输出磁盘统计信息　　注：默认是两个都输出`
3. `-k|-m: 以kb/s|mb/s代替原来的块/s`
4. `-t: 输出时打印收集信息时刻的时间　注：时间的打印格式和系统变量S_TIME_FORMAT相关`
5. `-x: 输出详细的拓展统计数据，比如各种等待时间，队列，利用率等信息。`
6. `interval [count] :interval是统计的时间间隔单位是s，count则是统计次数`

await指的是平均等待时间，一般都在10ms左右。

[root@VM_0_ubuntu ~]# **iostat -x**

```
Device:  rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
loop0      0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
sda        0.00    80.00    5.00 1152.00    20.00 935160.00  1616.56     2.74    2.96    1.60    2.97   0.74  85.20

Device:  rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
loop0      0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
sda        0.00    80.00   15.00 1137.00   852.00 1049236.00  1823.07     2.30    2.59    1.07    2.61   0.72  82.80
```

```
Device:  rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz
await r_await w_await  svctm  %util
loop0     0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00
0.00    0.00    0.00   0.00   0.00
sda       0.00    80.00    5.00 1152.00    20.00 935160.00  1616.56    2.74
2.96    1.60    2.97   0.74  85.20

Device:  rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz
await r_await w_await  svctm  %util
loop0     0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00
0.00    0.00    0.00   0.00   0.00
sda       0.00    80.00   15.00 1137.00   852.00 1049236.00  1823.07   2.30
2.59    1.07    2.61   0.72  82.80
```

每个输出消息含义：

- rrqm/s: 每秒对该设备的读请求被合并次数，文件系统会对读取同块(block)的请求进行合并
- **wrqm**/s: 每秒对该设备的写请求被合并次数
- r/s: 每秒完成的读次数
- w/s: 每秒完成的写次数
- rkB/s: 每秒读数据量(kB为单位)
- wkB/s: 每秒写数据量(kB为单位)
- **avgrq-sz**:平均每次IO操作的数据量(扇区数为单位)，比如1616.56
- avgqu-sz: 平均等待处理的IO请求队列长度，比如2.74
- **await**: 平均每次IO请求**等待时间**(包括等待时间和处理时间，毫秒为单位)，比如w_await 2.97
- **svctm**: 平均每次IO请求的**处理时间**(毫秒为单位)，比如0.72
- %util: 采用周期内用于IO操作的时间比率，即IO队列非空的时间比率。表示该设备的繁忙程度，比如82.80。例如，如果统计间隔1秒，该设备有0.5秒在处理IO，而0.5秒闲置，则该设备的%util = 0.5/1 = 50%。一般地，如果该参数是100%表示设备已经接近满负荷运行。

关于 await和svctm

> 可以理解为IO请求的响应时间，包括队列等待时间和服务时间，我们分析IO问题时，如果await大于svctm，await-svctm差值越小，则说明队列时间越短, 反之差值越大，队列时间越长，说明磁盘io有性能问题。

cpu的统计信息，如果是多cpu系统，显示的所有cpu的平均统计信息。

- %user：用户进程消耗cpu的比例
- %nice：用户进程优先级调整消耗的cpu比例
- %sys：系统内核消耗的cpu比例
- %iowait：等待磁盘io所消耗的cpu比例
- %idle：闲置cpu的比例（不包括等待磁盘I/O）

常用用法

```
// kb/s显示磁盘信息，每2s刷新一次

iostat -d -k 2

// kb/s显示磁盘统计信息及扩展信息，每1s刷新 ，刷新10次结束

iostat -dkx 1 10
```

2 swapon查看分区使用情况

查看交互分区的使用情况

使用方法：swapon -s

[root@VM_0_ubuntu ~]# swapon -s

Filename Type Size Used Priority

/var/swap file 8191996 473856 -2

我们在安装系统的时候已经建立了 swap 分区。swap 分区通常被称为交换分区，这是一块特殊的硬盘空间，即当实际内存不够用的时候，操作系统会从内存中取出一部分暂时不用的数据，放在交换分区中，从而为当前运行的程序腾出足够的内存空间。 也就是说，当内存不够用时，我们使用 swap 分区来临时顶替。这种“拆东墙，补西墙”的方式应用于几乎所有的操作系统中

一般来讲，swap 分区容量应大于物理内存大小，建议是内存的两倍，但不超过 2GB。

3 df硬盘使用情况

该命令用于查看文件系统的硬盘挂载点和空间使用情况。

使用方式：df -h

[root@VM_0_ubuntu ~]# df -h

Filesystem Size Used Avail Use% Mounted on

/dev/vda1 50G 42G 5.6G 89% /

devtmpfs 909M 0 909M 0% /dev

tmpfs 920M 24K 920M 1% /dev/shm

tmpfs 920M 844K 919M 1% /run

tmpfs 920M 0 920M 0% /sys/fs/cgroup

tmpfs 184M 0 184M 0% /run/user/0

4 du目录文件大小

du常用的选项：

-h：以人类可读的方式显示，显示M或K
-a：显示目录占用的磁盘空间大小，还要显示其下目录和文件占用磁盘空间的大小**-s**：显示目录占用的磁盘空间大小，**不**显示其下子目录和文件占用的磁盘空间大小**-c**：显示几个目录或文件占用的磁盘空间大小，还要**统计**它们的总和

du -a 显示目录和目录下子目录和文件占用磁盘空间的大小。直接使用-a 以字节为单位，-ha 如下图以M或K为结果显示。

du -s 显示当前所在目录大小

du -s -h home 显示home目录大小

du -c 显示几个目录或文件占用的磁盘空间大小，还要**统计**它们的总和

du -lh --max-depth=1 : 查看当前目录下一级子文件和子目录占用的磁盘容量。

[root@VM_0_ubuntu 0voice_im]# du -lh **--max-depth=1**

6.1M ./php

11M ./ios

146M ./server

6.6M ./pb

723M ./auto_setup

29M ./android

133M ./win-client

18M ./mac

8.0K ./tutorial

28K ./doc

99M ./.git

1.2G .

4.3 文件IO写入频繁案例分析

1. 启动iostat -x命令监测

iostat -dkx 1 30

2. 使用sysbench模拟数据读写

sysbench的性能测试都需要做prepare,run,cleanup这三步，准备数据，跑测试，删除数据。在准备阶段创建测试所需数据，在清理阶段删除这些数据。

```
#cd要到你测试的磁盘目录下
cd /data/disktest
# 线程数=4 每隔4s输出一次结果 测试时间=60s
# 文件数=10 文件总大小=10G 文件操作模式=随机读写
sysbench --num-threads=4  --max-time=60 --test=fileio --file-num=10 --file-total-
size=10G --file-test-mode=rndrw prepare

sysbench --num-threads=4  --max-time=60 --test=fileio --file-num=10 --file-total-
size=10G --file-test-mode=rndrw run

sysbench --num-threads=4  --max-time=60 --test=fileio --file-num=10 --file-total-
size=10G --file-test-mode=rndrw cleanup
```

```
lqf@ubuntu:~$ iostat -dkx 1 30
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
loop0             0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
sda               0.00    80.00    5.00 1152.00    20.00 935160.00  1616.56     2.74    2.96    1.60    2.97   0.74  85.20

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
loop0             0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
sda               0.00    80.00   15.00 1137.00   852.00 1049236.00  1823.07     2.30    2.59    1.07    2.61   0.72  82.80

Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await r_await w_await  svctm  %util
loop0             0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
sda               0.00     0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00    0.00    0.00   0.00   0.00
```

5 网络IO性能监控

5.1 性能指标

通常用带宽、吞吐量、延时、PPS（Packet Per Second）等指标衡量网络的性能。

1. **带宽**，表示链路的最大传输速率，单位通常为 b/s （比特 / 秒）。
2. **吞吐量**，表示单位时间内成功传输的数据量，单位通常为 b/s（比特 / 秒）或者 B/s（字节 / 秒）。吞吐量受带宽限制，而吞吐量 / 带宽，也就是该网络的使用率。
3. **延时**，表示从网络请求发出后，一直到收到远端响应，所需要的时间延迟。在不同场景中，这一指标可能会有不同含义。比如，它可以表示，建立连接需要的时间（比如 TCP 握手延时），或一个数据包往返所需的时间（比如 RTT）。
4. **PPS**，是 Packet Per Second（包 / 秒）的缩写，表示以网络包为单位的传输速率。PPS 通常用来评估网络的转发能力，比如硬件交换机，通常可以达到线性转发（即 PPS 可以达到或者接近理论最大值）。而基于 Linux 服务器的转发，则容易受网络包大小的影响。

除了这些指标，**网络的可用性**（网络能否正常通信）、**并发连接数**（TCP 连接数量）、**丢包率**（丢包百分比）、**重传率**（重新传输的网络包比例）等也是常用的性能指标。

5.2 网络信息

5.2.1 网络配置

使用 ifconfig 或者 ip 命令，来查看网络的配置。

> ifconfig 和 ip 分别属于软件包 net-tools 和 iproute2，iproute2 是 net-tools 的下一代。通常情况下它们会在发行版中默认安装。
>
> 如果你找不到 ifconfig 或者 ip 命令，可以安装这两个软件包。

以网络接口 ens33为例

- 使用ifconfig

```
root@ubuntu:/home/lqf# ifconfig ens33
ens33     Link encap:Ethernet  HWaddr 00:0c:29:11:82:79
          inet addr:192.168.206.134  Bcast:192.168.206.255
Mask:255.255.255.0
          inet6 addr: fe80::ceb7:e01f:2d2b:c50e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:48485 errors:0 dropped:0 overruns:0 frame:0
          TX packets:124191 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:35666099 (35.6 MB)  TX bytes:8827549 (8.8 MB)
```

- 使用ip

```
root@ubuntu:/home/lqf# ip -s addr show dev ens33
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UP group default qlen 1000
    link/ether 00:0c:29:11:82:79 brd ff:ff:ff:ff:ff:ff
    inet 192.168.206.134/24 brd 192.168.206.255 scope global dynamic ens33
       valid_lft 1493sec preferred_lft 1493sec
    inet6 fe80::ceb7:e01f:2d2b:c50e/64 scope link
       valid_lft forever preferred_lft forever
    RX: bytes  packets  errors  dropped overrun mcast
    35667821   48505    0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    8831315    124236   0       0       0       0
```

以下和网络性能密切相关的指标需要掌握：

1. 网络接口的状态标志。ifconfig 输出中的 **RUNNING**，或 ip 输出中的 **LOWER_UP**，都表示物理网络是连通的，即网卡已经连接到了交换机或者路由器中。如果你看不到它们，通常表示网线被拔掉了。
2. MTU 的大小。MTU 默认大小是 1500，根据网络架构的不同（比如是否使用了 VXLAN 等叠加网络），你可能需要调大或者调小 MTU 的数值。
3. 网络接口的 IP 地址、子网以及 MAC 地址。这些都是保障网络功能正常工作所必需的，你需要确保配置正确。

4. 网络收发的字节数、包数、错误数以及丢包情况，特别是 TX 和 RX 部分的 errors、dropped、overruns、carrier 以及 collisions 等指标不为 0 时，通常表示出现了网络 I/O 问题。其中：
    1. errors 表示发生错误的数据包数，比如校验错误、帧同步错误等；
    2. **dropped** 表示丢弃的数据包数，即数据包已经收到了 Ring Buffer，但因为内存不足等原因丢包；
    3. **overruns** 表示超限数据包数，即网络 I/O 速度过快，导致 Ring Buffer 中的数据包来不及处理（队列满）而导致的丢包；
    4. carrier 表示发生 carrirer 错误的数据包数，比如双工模式不匹配、物理电缆出现问题等；
    5. collisions 表示碰撞数据包数。

5.2.2 套接字信息

使用**netstat** 或者 ss ，来查看套接字、网络栈、网络接口以及路由表的信息。

PS：查询网络的连接信息ss比netstat查询速度更快。

```
# head -n 3 表示只显示前面3行
-a (all)显示所有选项，默认不显示LISTEN相关
-t (tcp)仅显示tcp相关选项
-u (udp)仅显示udp相关选项
-n 拒绝显示别名，能显示数字的全部转化成数字。
-l 仅列出有在 Listen (监听) 的服务状态

-p 显示建立相关链接的程序名
-r 显示路由信息，路由表
-e 显示扩展信息，例如uid等
-s 按各个协议进行统计
-c 每隔一个固定时间，执行该netstat命令。

root@ubuntu:/home/lqf# netstat -nlp | head -n 3
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:6371            0.0.0.0:*               LISTEN
3098/redis-server *
```

```
# -l 表示只显示监听套接字
# -t 表示只显示 TCP 套接字
# -n 表示显示数字地址和端口(而不是名字)
# -p 表示显示进程信息
root@ubuntu:/home/lqf# ss -ltnp | head -n 3
State      Recv-Q Send-Q Local Address:Port               Peer Address:Port

LISTEN     0      128          *:6371                     *:*
users:(("redis-server",pid=3098,fd=6))
LISTEN     0      80     127.0.0.1:3306                     *:*
users:(("mysqld",pid=1236,fd=39))
```

netstat 和 ss 的输出也是类似的，都展示了套接字的状态、接收队列、发送队列、本地地址、远端地址、进程 PID 和进程名称等。其中，接收队列（Recv-Q）和发送队列（Send-Q）需要特别关注，它们通常应该是 0。当你发现它们不是 0 时，说明有网络包的堆积发生。当然还要注意，在不同套接字状态下，它们的含义不同。

当套接字处于连接状态（Established）时：

- **Recv-Q 表示套接字缓冲还没有被应用程序取走的字节数**（即接收队列长度）。
- **Send-Q 表示还没有被远端主机确认的字节数**（即发送队列长度）。

当套接字处于监听状态（Listening）时：

- Recv-Q 表示全连接**队列的长度。**
- Send-Q 表示全连接**队列的最大长度。**

所谓全连接，是指服务器收到了客户端的 ACK，完成了 TCP 三次握手，然后就会把这个连接挪到全连接队列中。这些全连接中的套接字，还需要被 accept() 系统调用取走，服务器才可以开始真正处理客户端的请求。

与全连接队列相对应的，还有一个半连接队列。所谓半连接是指还没有完成 TCP 三次握手的连接，连接只进行了一半。服务器收到了客户端的 SYN 包后，就会把这个连接放到半连接队列中，然后再向客户端发送 SYN+ACK 包。

5.2.3 协议栈统计信息-netstat命令

使用 netstat 或 ss 查看协议栈的信息

```
root@ubuntu:/home/lqf# netstat -s
Ip:
    351903 total packets received
    3 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    351838 incoming packets delivered
    664706 requests sent out
    4 outgoing packets dropped
    328 dropped because of missing route
....
```

```
root@ubuntu:/home/lqf# ss -s
Total: 1501 (kernel 8096)
TCP:   34 (estab 3, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total     IP        IPv6
*     8096      -         -
RAW   1         0         1
UDP   28        23        5
TCP   34        23        11
INET      63        46        17
FRAG      0         0         0
```

这些协议栈的统计信息都很直观。ss 只显示已经连接、关闭、孤儿套接字等简要统计，**而 netstat 则提供的是更详细的网络协议栈信息。**比如，上面 netstat 的输出示例，就展示了 TCP 协议的主动连接、被动连接、失败重试、发送和接收的分段数量等各种信息。

5.2.4 网络吞吐-sar命令

给 **sar 增加 -n 参数**就可以查看网络的统计信息，比如网络接口（DEV）、网络接口错误（EDEV）、TCP、UDP、ICMP 等等。执行下面的命令，你就可以得到网络接口统计信息：

```
root@iZbp1h2l856zgoegc8rvnhZ:~# sar -n DEV 1
Linux 4.4.0-93-generic (iZbp1h2l856zgoegc8rvnhZ)    11/02/2021  _x86_64_    (1
CPU)

06:38:15 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s
rxmcst/s   %ifutil
06:38:16 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00
     0.00      0.00
06:38:16 PM      eth0     11.46     10.42      6.35      2.47      0.00      0.00
     0.00      0.00
```

输出指标含义

- rxpck/s 和 txpck/s 分别是接收和发送的 PPS，单位为包 / 秒。
- rxkB/s 和 txkB/s 分别是接收和发送的吞吐量，单位是 KB/ 秒。
- rxcmp/s 和 txcmp/s 分别是接收和发送的压缩数据包数，单位是包 / 秒。
- %ifutil 是网络接口的使用率，即半双工模式下为 (rxkB/s+txkB/s)/Bandwidth，而全双工模式下为 max(rxkB/s, txkB/s)/Bandwidth。

[root@VM_0_ubuntu ~]# sar -n DEV 1 1

Linux 3.10.0-957.5.1.el7.x86_64 (VM_0_ubuntu) 08/23/2019 *x86_64* (1 CPU)

10:57:49 AM IFACE rxpck/s txpck/s rxkB/s txkB/s rxcmp/s txcmp/s rxmcst/s

10:57:50 AM eth0 9.09 13.13 0.56 1.01 0.00 0.00 0.00

10:57:50 AM lo 0.00 0.00 0.00 0.00 0.00 0.00 0.00

10:57:50 AM docker0 0.00 0.00 0.00 0.00 0.00 0.00 0.00

Average: IFACE rxpck/s txpck/s rxkB/s txkB/s rxcmp/s txcmp/s rxmcst/s

Average: eth0 9.09 13.13 0.56 1.01 0.00 0.00 0.00

Average: lo 0.00 0.00 0.00 0.00 0.00 0.00 0.00

Average: docker0 0.00 0.00 0.00 0.00 0.00 0.00 0.00

从输出中可以看到网卡的读写速度和流量，在应急过程中可以用来判断服务器是否上量。

此命令除了可以用于查看网卡的信息，还可以用来收集如下服务的状态信息。

- -A：所有报告的总和。
- -u：CPU利用率。
- -v：进程、I节点、文件和锁表状态。
- **-d：硬盘的使用报告。**
- -r：没有使用的内存页面和硬盘快。
- -g：串口I/O的情况。
- -b：缓冲区的使用情况。
- -a：文件的读写情况。
- -c：系统的调用情况。
- -R：进程的活动情况。
- -y：终端设备的活动情况。
- -w：系统的交换活动。

之前公司自己搞的CDN，并发稍微高点图片就拉取不下来。最后发现是云服务器做了限速。

5.2.5 连通性和延时

通常使用 ping ， 来测试远程主机的连通性和延时

ping

说明

ping命令是用于检测网络故障的常用命令，可以用来钡试一台主机到另外一台主机的网络是否连通。

语法

ping [-dfnqrRv][-c<完成次数>][-i<间隔秒数>][-I<网络界面>][-l<前置载入>][-p<范本样式>][-s<数据包大小>][-t<存活数值>][主机名称或IP地址]

- -d 使用Socket的SO_DEBUG功能。
- -c<完成次数> 设置完成要求回应的次数。
- -f 极限检测。
- -i<间隔秒数> 指定收发信息的间隔时间。
- -I<网络界面> 使用指定的网络接口送出数据包。
- -l<前置载入> 设置在送出要求信息之前，先行发出的数据包。
- -n 只输出数值。
- -p<范本样式> 设置填满数据包的范本样式。
- -q 不显示指令执行过程，开头和结尾的相关信息除外。
- -r 忽略普通的Routing Table，直接将数据包送到远端主机上。
- -R 记录路由过程。
- -s<数据包大小> 设置数据包的大小。
- -t<存活数值> 设置存活数值TTL的大小。
- -v 详细显示指令的执行过程。

检测和主机是否连通

ping baidu.com

ping指定次数

```
root@ubuntu:/home/lqf# ping -c 3 baidu.com
PING baidu.com (220.181.38.251) 56(84) bytes of data.
64 bytes from 220.181.38.251: icmp_seq=1 ttl=128 time=32.0 ms
64 bytes from 220.181.38.251: icmp_seq=2 ttl=128 time=31.7 ms
64 bytes from 220.181.38.251: icmp_seq=3 ttl=128 time=34.3 ms

--- baidu.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 31.740/32.717/34.324/1.154 ms
```

1. ping目标主机的域名和IP（ping会自动将域名转换为IP）
2. 不带包头的包大小和带包头的包大小（参考"-s"参数）
3. icmp_seq：ping序列，从1开始；如果数字不是按顺序递增也就意味着丢包了

   ttl：剩余的ttl；见TTL解释

   time: 响应时间,数值越小，连通速度越快；
4. 发出去的包数，返回的包数，丢包率，耗费时间；
5. 最小/最大/平均响应时间和本机硬件耗费时间；

TTL：每经过一个'路由点'，就把预先设定的这个TTL数值减1，直到最后TTL=1时报文就被扔掉，不向下转发。ping显示的TTL是指：被ping主机那里返回的报文，到了你这里，从它自己预设的TTL减小到还剩下多少。

范例

1. **指定ping的次数: -c 次数**

> ping -c 4 www.baidu.com

2. **只显示结果:-q**

> lqf@ubuntu:~$ ping -c 4 www.baidu.com -q

PING www.a.shifen.com (14.215.177.38) 56(84) bytes of data.

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

4 packets transmitted, 4 received, 0% packet loss, time 3008ms

rtt min/avg/max/**mdev** = 20.471/21.141/21.678/**0.444** ms

mdev 往返时延

mdev是 Mean Deviation 的缩写， 表示 ICMP包的RTT(Round-Trip Time,往返时延)偏离平均值的程度， 主要用来衡量网速的稳定性。 mdev 的值越大说明 网速越不稳定。 另外，不同的操作系统的mdev的名字也有所不同， 在mac下它叫作 stddev， 而在 Windows 下则根本没有这个统计指标。

RTT参考值

场景	参考值
本机	0.001ms
同机房	0.1ms
同城	1ms
不同城	20ms
中国北方到南方	50ms
国内到国外	200ms

3. **制定ping数据包的大小**

默认ping命令的数据包大小是64Bytes,通过-s选项可以制定数据包的大小

范围：1Byte-65507Byte

```
root@ubuntu:/home/lqf# ping  -s 254 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 254(282) bytes of data.
262 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.054 ms
262 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.056 ms
262 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.026 ms
262 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.042 ms
262 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.025 ms
```

补充：,如果以最大值去ping服务器，并且以多进程多终端的方式，可能会造成服务器响应迟缓，可以认为是一种网络攻击手段

4. **TTL 生成时间**

TTL（Time to Live），即生存时间，指的是数据包被路由器丢弃之前允许通过的路由器的跳数，为了防止数据包在路由器之间无限转发，必须设置一个TTL值，每次路由器转发后都会将这个值减 1，直到TTL的值为0，这个数据包的生命就被终结了.

命令 ： ping -t 255 www.baidu.com

若不指定ttl的值，系统会默认给定一个值，不同的操作系统有不同的默认TTL值：

操作系统 TCP传输 UDP传输

AIX 60 30

DEC Patchworks V5 30 30

FreeBSD 2.1 64 64

HP/UX 9.0x 30 30

HP/UX 10.01 64 64

Irix 5.3 60 60

Irix 6.x 60 60

UNIX 255 255

Linux 64 64

MacOS/MacTCP 2.0.x 60 60

OS/2 TCP/IP 3.0 64 64

OSF/1 V3.2A 60 30

Solaris 2.x 255 255

SunOS 4.1.3/4.1.4 60 60

Ultrix V4.1/V4.2A 60 30

VMS/Multinet 64 64

VMS/TCPware 60 64

VMS/Wollongong 1.1.1.1 128 30

VMS/UCX (latest rel.) 128 128

MS Windows 95/98/NT 3.51 32 32

Windows NT 4.0/2000/XP/2003 128 128

补充：

通过TTL设定值我们可以找到某主机到某主机的最小跳跃次数，即路由转发次数，如下，可以看到我的主机到百度的最小跳至少是11，因为ttl为10是显示跳跃超限，说明数据包在传给百度服务器的路上已经死了

```
root@ubuntu:/home/lqf# ping -t 11 www.baidu.com
PING www.a.shifen.com (14.215.177.38) 56(84) bytes of data.
^C  按Ctrl +C 退出ping
--- www.a.shifen.com ping statistics ---
79 packets transmitted, 0 received, 100% packet loss, time 79045ms
```

5.3 其他常用的网络相关命令

telnet

telnet是TCP/IP协议族的一员，是网络远程登录服务的标准协议，帮助用户在本地计算机上连接远程主机。

使用方式：telnet IP PORT

和ssh **的区别**

端口区别：telnet是23 ssh是22

本质：telnet是明码传输，**ssh**是加密传输

nc

验证服务器端口有没有开放

nc是NetCat的简称，在网络调试工具中享有“瑞士军刀”的美誉，此命令功能丰富、短小精悍、简单实用，被设计成一款易用的网络工具，可通过TCP/LJDP传输数据。

参数

- -l 用于指定nc将处于侦听模式。指定该参数，则意味着nc被当作server，侦听并接受连接，而非向其它地址发起连接。
- -p 暂未用到（老版本的nc可能需要在端口号前加-p参数，下面测试环境是centos6.6，nc版本是nc-1.84，未用到-p参数）
- -s 指定发送数据的源IP地址，适用于多网卡机
- -u 指定nc使用UDP协议，默认为TCP
- -v 输出交互或出错信息，新手调试时尤为有用
- -w 超时秒数，后面跟数字

常用示例

nc -l 9999 # 开启一个本地9999的TCP协议端口，由客户端主动发起连接，一旦连接必须由服务端发起关闭

nc -vw 2 114.215.169.66 9999 # 通过nc去访问129.204.197.215主机的11111端口，确认是否存活；可不加参数

nc -ul 9999 # 开启一个本地9999的UDP协议端口，客户端不需要由服务端主动发起关闭

nc 129.204.197.215 9999 < test # 通过129.204.197.215的9999TCP端口发送数据文件

nc -l 9999 > zabbix.file # 开启一个本地9999的TCP端口，用来接收文件内容

测试网速

A机器操作如下：

nc -l 9999 > /dev/null

B机器开启数据传输

nc 129.204.197.215 9999 </dev/zero

A机器进行网络监控

sar -n DEV 2 100000

在不熟悉公司的防火墙设置时，也可以通过监听端口来确定指定的端口是否被封。

mtr连通性测试

mtr命令是Linux系统中的网络连通性测试工具，也可以用来检测丢包率。

使用方式：mtr baidu.com

```
                                My traceroute  [v0.85]
VM_0_3_centos (0.0.0.0)                                  Fri Aug 23 10:46:00 2019
Resolver: Received error response 2. (server failure)n. of fields   quit
                                          Packets               Pings
 Host                                    Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. 100.119.162.129                       0.0%     9    5.2   8.1   1.6  17.5   4.3
 2. 100.119.170.88                        0.0%     9    0.7   0.9   0.7   2.2   0.4
 3. 10.200.135.229                        0.0%     9    0.6   0.7   0.6   0.7   0.0
 4. 10.196.93.226                         0.0%     9    1.2   1.3   1.2   1.6   0.0
 5. 120.241.50.1                         55.6%     9    4.7   5.7   2.7  12.3   4.5
 6. 221.183.59.153                        0.0%     9    4.6   4.7   4.6   4.9   0.0
 7. 221.183.37.137                       44.4%     9   38.0  38.2  38.0  38.6   0.0
 8. ???
 9. 39.156.27.5                           0.0%     9   37.6  37.5  37.4  37.6   0.0
10. ???
11. ???
12. ???
13. ???14. ???15. 39.156.69.79            0.0%     6   37.9  37.9  37.9  38.0   0.0
```

其中的第2列为丢包率，可以用来判断网络中两台机器的连通质量。

模拟丢包：sudo tc qdisc add dev eth0 root netem loss 10%

云服务器选点：声网做音视频通话 节点选取测试

nslookup

这是一款检测网络中DNS服务器能否正确解析域名的工具命令，并且可以输出。

使用方式：nslookup sina.com

[root@VM_0_ubuntu ~]# nslookup sina.com

Server: 183.60.83.19

Address: 183.60.83.19#53

Non-authoritative answer:

Name: sina.com

Address: 66.102.251.33

从输出中可以看到，sina.com域名被正确解析到IP地址66.102.251.33。

nslookup www.0voice.com

traceroute

traceroute可以提供从用户的主机到互联网另一端的主机的路径，虽然每次数据包由同一出发点到达同一目的地的路径可能会不一样，但通常来说大多数情况下路径是相同的。

使用方式：traceroute sina.com

[root@VM_0_ubuntu ~]# traceroute sina.com

traceroute to sina.com (66.102.251.33), 30 hops max, 60 byte packets

 1 100.119.162.130 (100.119.162.130) 1.462 ms 2.200 ms 3.004 ms

 2 100.119.170.58 (100.119.170.58) 1.060 ms 1.457 ms 1.852 ms

 3 10.200.135.213 (10.200.135.213) 0.596 ms 0.794 ms 10.200.135.197 (10.200.135.197) 0.678 ms

 4 14.18.199.78 (14.18.199.78) 1.008 ms 14.18.199.58 (14.18.199.58) 0.986 ms 0.951 ms

 5 * * *

 6 113.96.6.42 (113.96.6.42) 7.938 ms 121.14.50.25 (121.14.50.25) 2.913 ms 113.96.6.18 (113.96.6.18) 7.852 ms

 7 113.96.4.157 (113.96.4.157) 4.799 ms 113.96.0.114 (113.96.0.114) 9.306 ms 113.96.4.157 (113.96.4.157) 4.547 ms

8 202.97.67.46 (202.97.67.46) 1.633 ms 2.392 ms 202.97.94.138 (202.97.94.138) 24.012 ms

9 202.97.94.77 (202.97.94.77) 4.379 ms 202.97.94.102 (202.97.94.102) 11.375 ms 11.335 ms

10 202.97.51.134 (202.97.51.134) 160.060 ms 160.058 ms 160.048 ms

11 203.14.186.238 (203.14.186.238) 181.185 ms 181.146 ms 181.088 ms

12 218.30.41.245 (218.30.41.245) 157.970 ms 157.110 ms 157.156 ms

13 218.30.41.234 (218.30.41.234) 156.973 ms 159.204 ms 157.517 ms

14 66.102.251.33 (66.102.251.33) 157.370 ms * 168.266 ms

在输出中记录按序列号从1开始，每个记录代表网络一跳，每跳一次表示经过一个网关或者路由；我们看到每行有三个个时间，**单位是毫秒，指的是这一跳需要的时间。**

iptraf强大的网络监控

iptraf是一个实时监控网络流量的交互式的彩色文本屏幕界面。它监控的数据比较全面，可以输出TCP连接、网络接口、协议、端口、网络包大小等信息，但是耗费的系统资源比较多，且需要管理员权限。

使用方式：sudo iptraf

在进入主界面之前可以选择不同的选项，在不同的选项下可以查看不同维度的网络信息。

安装方法：

centos

yum -y install ncurses

yum -y install iptraf

ubuntu

sudo apt-get install iptraf

tcpdump- 抓包，然后用wireshark分析

tcpdump是网络状况分析和跟踪工具，是可以用来抓包的实用命令，使用前需要对TCP/IP有所熟悉，因为过滤使用的信息都是TCP/IP格式。

Ubuntu安装：sudo apt-get install tcpdump

捕获及停止条件

n -D 列举所有网卡设备

n -i 选择网卡设备

n -c 抓取多少条报文

n --time-stamp-precision 指定捕获时的时间精度，默认微妙micro，可选纳秒nano

n -s 指定每条报文的最大字节数，默认262144字节

练习：

tcpdump 抓取各种报文

tcpdump -D 显示网卡设备

tcpdump -i eth0 抓取eth0的报文

tcpdump -i lo 抓取回环报文

tcpdump -i eth0 -c 2 抓取2条报文后退出

tcpdump -i eth0 -c 2 --time-stamp-precision=nano 默认按纳秒

telnet 127.0.0.1 80 可以通过telnet 上去测试抓包，此时应用抓取lo

表达式

原语：名称或数字，以及描述它的多个限定词组成

限定词：

Type：设置数字或者名称所指示类型，例如host www.baidu.com
Dir：设置网络出入方向，例如dst port 80
Proto: 指定协议类型，例如udp

原语运算：

与：&&或者and
或：||或者or
非：!或者not

例如：src or dst portrange 6000-8000 && tcp

练习：

tcpdump -i eth0 -c 10 host www.0voice.com and port 80

同时在另一个终端curl www.0voice.com

限定词

Type：设置数字或者名称所指示类型

- host、port
- net, 设定子网， net 192.168.0.0 mask 等价于net 192.168.0.0/24
- portrange，设置端口范围，例如portrange 6000-8000

Dir：设置网络出入方向
src、dst、src or dst、src and dst
ra、ta、addr1、addr2、addr3、addr4（仅对IEEE 802.11 Wireless Lan有效）

Proto**：指定协议类型**

ethr、fddi、tr、wlan、ip、ip6、arp、tcp、udp等

文件操作

-w 输出结果至文件

-C 限制输入文件的大小，超出以后缀1等数字的方式递增。注意单位是1,000,000字节

-W 指定输出文件的最大数量，到达后重新复写第一个文件

-G 指定每个N秒就重新输出至新文件，注意-w参数应基于strftime参数指定文件名

-r 读取一个抓包文件

-V 将待读取的多个文件名写入一个文件中，通过读取该文件同时读取多个文件

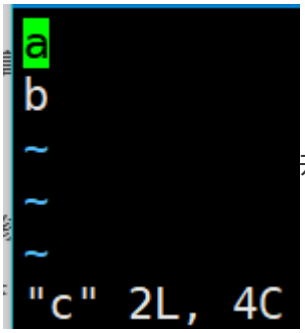
练习：

tcpdump -c 2 -w a 保存2个包到a文件

tcpdump -r a　读取a文件显示

tcpdump -c 2 -w b 保存2个包到b文件

vim c 文件，把逐行写入a、b，如下所示



并保存。

tcpdump -V c　读取多个文件

tcpdump -C 1 -W 3 abc 限制文件大小为1,000,000字节，输出文件最大数量为3

tcpdump -G 3 -w def%M-%S

strftime参考：

%a 星期几的简写

%A 星期几的全称

%b 月分的简写

%B 月份的全称

%c 标准的日期的时间串

%C 年份的后两位数字

%d 十进制表示的每月的第几天

%D 月/天/年

%e 在两字符域中，十进制表示的每月的第几天

%F 年-月-日

%g 年份的后两位数字，使用基于周的年

%G 年分，使用基于周的年

%h 简写的月份名

%H 24小时制的小时

%I 12小时制的小时

%j 十进制表示的每年的第几天

%m 十进制表示的月份

%M 十时制表示的分钟数

%n 新行符

%p 本地的AM或PM的等价显示

%r 12小时的时间

%R 显示小时和分钟：hh:mm

%S 十进制的秒数

%t 水平制表符

%T 显示时分秒：hh:mm:ss

%u 每周的第几天，星期一为第一天 （值从0到6，星期一为0）

%U 第年的第几周，把星期日做为第一天（值从0到53）

%V 每年的第几周，使用基于周的年

%w 十进制表示的星期几（值从0到6，星期天为0）

%W 每年的第几周，把星期一做为第一天（值从0到53）

%x 标准的日期串

%X 标准的时间串

%y 不带世纪的十进制年份（值从0到99）

%Y 带世纪部分的十进制年份

%z，%Z 时区名称，如果不能得到时区名称则返回空字符。

%% 百分号

输出时间格式

-t 不显示时间戳

-tt 自1970年1月1日0点至今的秒数

-ttt 显示邻近两行报文间经过的秒数

-tttt 带日期的完整时间

-ttttt 自第一个抓取的报文起经历的秒数

练习

sudo tcpdump -c 5 -t

sudo tcpdump -c 5 -tt

sudo tcpdump -c 5 -ttt

sudo tcpdump -c 5 -tttt

sudo tcpdump -c 5 -ttttt

分析信息详情

-e 显示数据链路层头部

-q 不显示传输层信息

-v 显示网络层头部更多的信息，如TTL、id等

-n 显示IP地址、数字端口代替hostname等

-S TCP信息以绝对序列号代替相对序列号

-A 以ASCII方式显示报文内容，适用HTTP分析

-x 以16进制方式显示报文内容，不显示数据链路层

-xx 以16进制方式显示报文内容，显示数据链路层

-X 同时以16进制及ASCII方式显示报文内容，不显示数据链路层

-XX 同时以16进制及ASCII方式显示报文内容，显示数据链路层

练习

sudo tcpdump -r a -e

sudo tcpdump -r a -q

sudo tcpdump -r a -v

sudo tcpdump -r a -n

sudo tcpdump -r a -S

sudo tcpdump -r a -A

sudo tcpdump -r a -x

sudo tcpdump -r a -xx

sudo tcpdump -r a **-X**

sudo tcpdump -r a **-XX**

其他练习：

显示来源IP或者目的IP为192.168.1.102的网络通信：

sudo tcpdump -i eth0 host 192.168.1.102

显示去往102.168.1.102的所有会话信息：

tcpdump -i eth1 'dst 192.168.1.102 and (port 21 or 20)'

显示去往102.168.1.102的所有会话信息：

tcpdump -i eth0 'dst 192.168.1.102 and **tcp** and port 8080'

nmap

扫描某一主机打开的端口及端口提供的服务信息，通常用于查看本机有哪些端口对外提供服务，或者服务器有哪些端口对外开放。

使用方式：nmap -v -A localhost

比如我云服务器检测的结果：

```
Initiating SYN Stealth Scan at 16:12
Scanning localhost (127.0.0.1) [1000 ports]
Discovered open port 22/tcp on 127.0.0.1
Discovered open port 80/tcp on 127.0.0.1
Discovered open port 3306/tcp on 127.0.0.1
Discovered open port 554/tcp on 127.0.0.1
Discovered open port 10009/tcp on 127.0.0.1
Discovered open port 10000/tcp on 127.0.0.1
Discovered open port 10004/tcp on 127.0.0.1
Discovered open port 1935/tcp on 127.0.0.1
Discovered open port 9000/tcp on 127.0.0.1
Discovered open port 10003/tcp on 127.0.0.1
Discovered open port 10001/tcp on 127.0.0.1
```

但实际能否进行连通还要看防火墙的设置

安装

```
# ubuntu安装
sudo apt install nmap

#centos安装
yum install nmap
```

nmap localhost #查看主机当前开放的端口

nmap -p 1024-65535 localhost #查看主机端口（1024-65535）中开放的端口

nmap -PS 192.168.56.101 #探测目标主机开放的端口

nmap -PS 22,80,3306 192.168.56.101 #探测所列出的目标主机端口

nmap -O 192.168.56.101 #探测目标主机操作系统类型

nmap -A 192.168.56.101 #探测目标主机操作系统类型

lsof

lsof是系统管理/安全的尤伯工具。将这个工具称之为lsof真实名副其实，因为它是指"列出打开文件（lists openfiles）"。而有一点要切记，在Unix中一切（包括网络套接口）都是文件。

查看帮助文档： lsof -h

你可以使用它来获得你系统上设备的信息，你能通过它了解到指定的用户在指定的地点正在碰什么东西，或者甚至是一个进程正在使用什么文件或网络连接。具体可以使用man lsof查看帮助文档。

关键选项

默认 : 没有选项，lsof列出活跃进程的所有打开文件

组合 : 可以将选项组合到一起，如-abc，但要当心哪些选项需要参数

- -a : 结果进行"与"运算（而不是"或"）
- -l : 在输出显示用户ID而不是用户名
- -h : 获得帮助
- -t : 仅获取进程ID
- -U : 获取UNIX套接口地址
- -F : 格式化输出结果，用于其它命令。可以通过多种方式格式化，如-F pcfn（用于进程id、命令名、文件描述符、文件名，并以空终止）

lsof -i:9999

ethtool

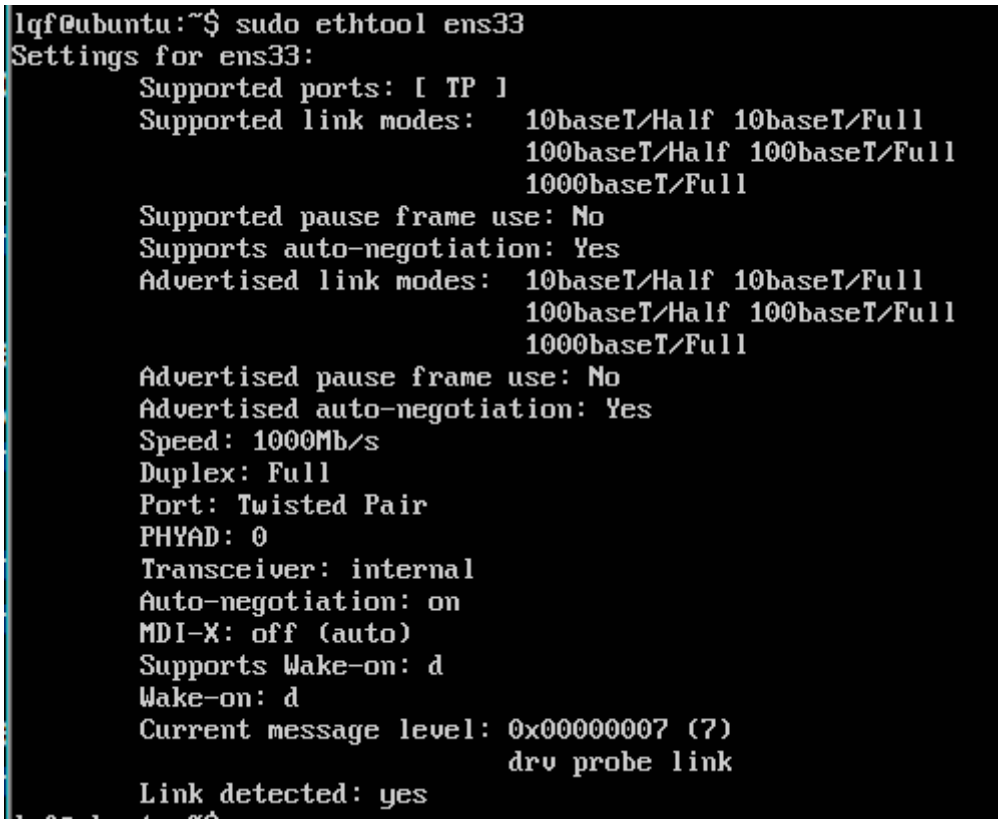
虚拟机不要设置成静态的ip

ethtool用于查看网卡的配置情况。

sudo apt install ethtool

命令使用格式：ethtool [option] interface

查看网卡：ethtool ens33 # PS在云主机检测不到网卡信息



范例

(1)查看网卡的接口信息

ethtool eth1 #查看网络接口eth1的信息

(2)关闭网卡eth1的自动协商

ethtool -s eth1 autoneg off

(3)修改网卡速率为 100Mb/s

ethtool -s eth4 speed 100

(4)查看网卡驱动信息

ethtool -i eth0

(5)查看网卡的一些工作统计信息

ethtool –S eth0

(6)停止和查看网卡的发送模块TX的状态

ethtool -A tx off eth0 #修改tx的状态

ethtool -a eth0 #查看tx的状态

(7)关闭网卡对收到的数据包的校验功能

ethtool -K rx off eth0 #关闭校验

ethtool –k eth0 #查看校验启动状态

获取网络信息

显示端口被某个程序占用

lsof -i:port

```
[root@VM_0_ubuntu msg_server]# lsof -i:80

COMMAND  PID USER  FD  TYPE  DEVICE SIZE/OFF NODE NAME

nginx  24614 root  10u IPv4 67976659    0t0 TCP *:http (LISTEN)

nginx  24615 root  10u IPv4 67976659    0t0 TCP *:http (LISTEN)



[root@VM_0_ubuntu msg_server]# lsof -i:8000

COMMAND   PID USER  FD  TYPE  DEVICE SIZE/OFF NODE NAME

msg_serve 17967 root   5u IPv4 18955252    0t0 TCP *:irdmi (LISTEN)
```

lsof -p 12 看进程号为12的进程打开了哪些文件

lsof -c abc 显示abc进程现在打开的文件，可以使用less进行分页，b 向后翻一页，d 向后翻半页

[root@VM_0_ubuntu msg_server]# lsof -c msg_server

显示的部分内容

msg_serve 17967 root 3w REG 253,1 7005733 525212 /root/im/0voice_im/auto_setup/im_server/im-server-1/msg_server/log/default.log

msg_serve 17967 root 4w REG 253,1 0 564603 /root/im/0voice_im/auto_setup/im_server/im-server-1/msg_server/log/TEST.log

msg_serve 17967 root 5u IPv4 18955252 0t0 TCP *:irdmi (LISTEN)

lsof abc.txt 显示打开文件abc.txt的进程

[root@VM_0_ubuntu msg_server]# lsof log/default.log

COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME

msg_serve 17967 root 3w REG 253,1 7016015 525212 log/default.log

tail 21816 root 3r REG 253,1 7016015 525212 log/default.log

5.4 netstat的其他用途

此命令显示网络连接、端口信息等。

1. **根据进程查找端口**

（1）根据进程名查找进程ID

ps -elf | grep 进程

输出

[root@VM_0_ubuntu ~]# ps -elf | grep nginx

0 S root 6875 2674 0 80 0 - 28178 pipe_w 11:02 pts/3 00:00:00 grep --color=auto nginx

1 S root 24614 1 0 80 0 - 8524 sigsus Jul27 ? 00:00:00 nginx: master process /usr/local/nginx/sbin/nginx

5 S root 24615 24614 0 80 0 - 14563 ep_pol Jul27 ? 00:00:13 nginx: worker process

获得进程号为 24614 （master process）

（2）根据进程ID查找进程开启的端口

netstat -nap | grep 24614

ubuntu sudo 权限

[root@VM_0_ubuntu ~]# netstat -nap | grep 24614

tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN 24614/nginx: master

tcp 0 0 0.0.0.0:10000 0.0.0.0:* LISTEN 24614/nginx: master

tcp 0 0 0.0.0.0:8088 0.0.0.0:* LISTEN 24614/nginx: master

tcp 0 0 0.0.0.0:9080 0.0.0.0:* LISTEN 24614/nginx: master

tcp 0 0 0.0.0.0:444 0.0.0.0:* LISTEN 24614/nginx: master

tcp 0 0 0.0.0.0:10080 0.0.0.0:* LISTEN 24614/nginx: master

unix 3 [ ] STREAM CONNECTED 67976664 24614/nginx: master

unix 3 [ ] STREAM CONNECTED 67976665 24614/nginx: master

2. **根据端口查找进程**

（1）根据使用端口

netstat -nap | grep 8080

[root@VM_0_ubuntu ~]# netstat -nap | grep 8080

tcp 0 0 0.0.0.0:8080 0.0.0.0:* LISTEN 25901/./login_serve

获得进程ID为25901

（2）根据进程ID查找进程的详细信息

[root@VM_0_ubuntu ~]# ps -elf | grep 25901

0 R root 7646 2674 0 80 0 - 28178 - 11:06 pts/3 00:00:00 grep --color=auto 25901

0 S root 25901 1 0 80 0 - 42506 ep_pol Jul20 ? 00:10:33 ./login_server

6 其他工具

nmon性能监控

centos安装

wget http://sourceforge.net/projects/nmon/files/nmon16e_mpginc.tar.gz

mkdir nmon16e_mpginc

tar -xvfz nmon16e_mpginc.tar.gz -C nmon16e_mpginc

cd nmon16e_mpginc

授权运行权限

chmod +x nmon_x86_64_centos7

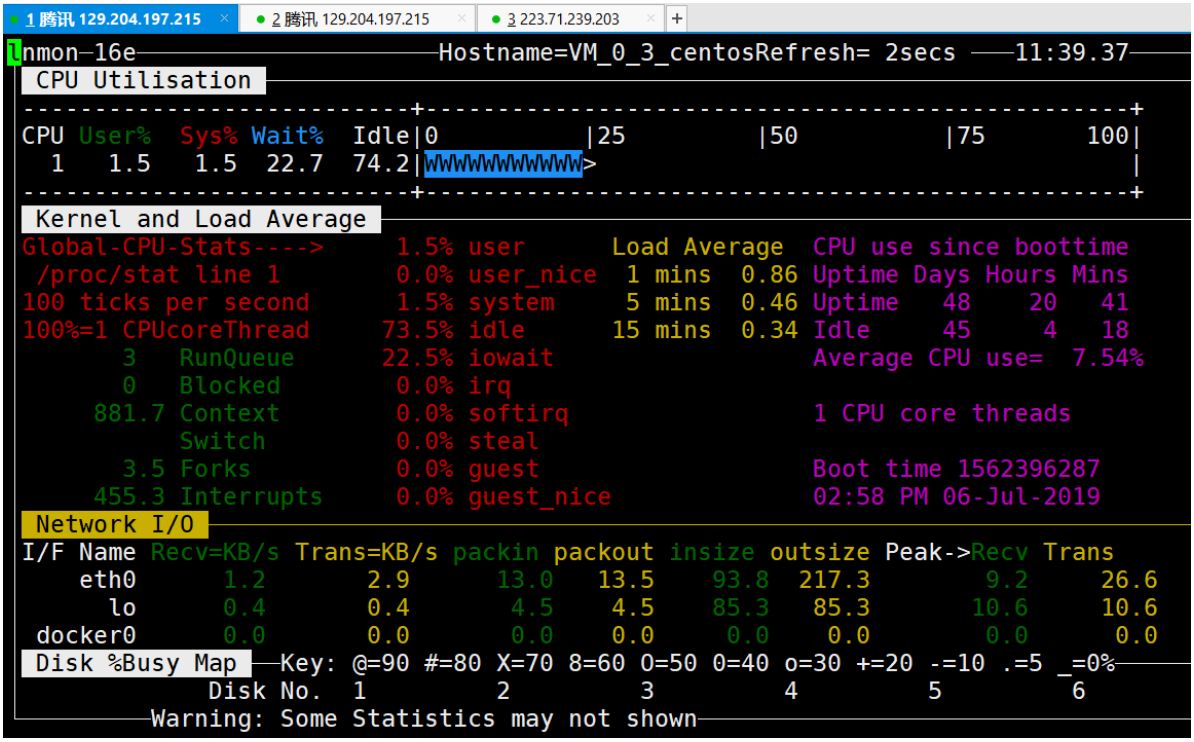
使nmon在任何地方都能运行

mv nmon_x86_64_centos7 /usr/bin/nmon

ubuntu安装

$ sudo apt-get install nmon

$ nmon



后台监控

为了配合性能测试，我们往往需要将一个时间段内系统资源消耗情况记录下来，这时可以使用命令在远程窗口执行命令：

./nmon/ nmon_x86_rhel5 -f -N -m /nmon/log -s 30 -c 120

其中各参数表示：

- -f 按标准格式输出文件：_YYYYMMDD_HHMM.nmon
- -N include NFS sections
- -m 切换到路径去保存日志文件
- -s 每隔n秒抽样一次，这里为30
- -c 取出多少个抽样数量，这里为120，即监控=120*(30/60/60)=1小时

根据小时计算这个数字的公式为：c=h3600/s，比如要监控10小时，每隔30秒采样一次，则c=103600/30=1200

该命令启动后，会在nmon所在目录下生成监控文件，并持续写入资源数据，直至360个监控点收集完成——即监控1小时，这些操作均自动完成，无需手工干 预，测试人员可以继续完成其他操作。如果想停止该监控，需要通过“#ps –ef|grep nmon”查询进程号，然后杀掉该进程以停止监控。

定时任务

除配合性能测试的短期监控，我们也可以实现对系统的定期监控，作为运营维护阶段的参考。定期监控实现如下：

（1）执行命令：#crontab –e

（2）在最后一行添加如下命令：

0 8 * * 1,2,3,4,5 /nmon/nmon_x86_rhel5 -f -N -m /nmon/log -s 30 -c 1200

表示：

周一到周五，从早上08点开始，监控10个小时（到18:00整为止），输出到/nmon/log

测试指标可视化

nmon命令 生成的nmon可以通过工具进行可视化展示，一般可以使用nmonchart、nmon_analyser

nmonchart

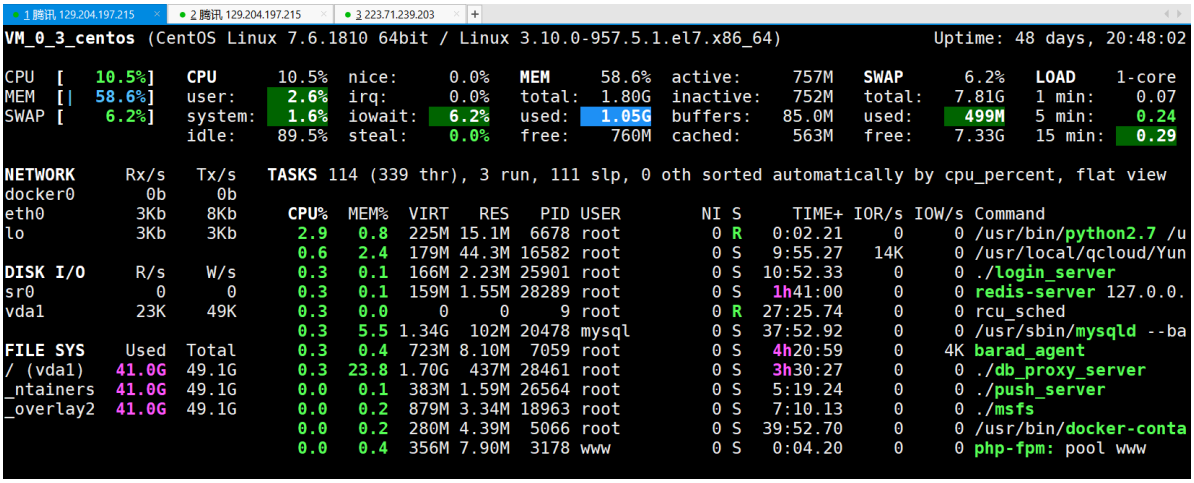
nmonechart 使用Google charts 生成html报告

glances系统监控

安装

```
#centos
yum install -y glances
#ubuntu
$ sudo apt-add-repository ppa:arnaud-hartmann/glances-stable
$ sudo apt-get update
$ sudo apt-get install glances
```

执行：glances



除了很多命令行选项之外，Glances 还提供了更多的可在其运行时开关输出信息选项的**快捷键**，下面是一些例子：

- a – 对进程自动排序
- c – 按 CPU 百分比对进程排序
- m – 按内存百分比对进程排序
- p – 按进程名字母顺序对进程排序
- i – 按读写频率（I/O）对进程排序
- d – 显示/隐藏磁盘 I/O 统计信息
- f – 显示/隐藏文件系统统计信息
- n – 显示/隐藏网络接口统计信息
- s – 显示/隐藏传感器统计信息
- y – 显示/隐藏硬盘温度统计信息
- l – 显示/隐藏日志（log）
- b – 切换网络 I/O 单位（Bytes/bits）
- w – 删除警告日志
- x – 删除警告和严重日志
- 1 – 切换全局 CPU 使用情况和每个 CPU 的使用情况
- h – 显示/隐藏这个帮助画面
- t – 以组合形式浏览网络 I/O
- u – 以累计形式浏览网络 I/O
- q – 退出（'ESC' 和 'Ctrl&C' 也可以

Glances 对于大多用户而言是个在系统资源上提供过多信息的工具。但是如果你是一个想要仅从命令行就能快速获取系统整体状况的系统管理员，那这个工具绝对是你的必备利器。

w

显示谁登录了系统并执行了哪些程序。

[root@VM_0_ubuntu ~]# w -h

root pts/0 113.246.155.156 10:58 2.00s 0.23s 0.23s -bash

root pts/2 113.246.155.156 11:30 26.00s 0.06s 0.00s tail -f /data/rtc/room_server/log/roll.log

root pts/3 113.246.155.156 11:53 1:30 0.03s 0.03s -bash

日志监控工具

tail

tail -h

在实时日志上打印颜色，给每个状态给上不同的颜色，INF绿色、WARN黄色、ERROR红色

tail -f /data/rtc/room_server/log/roll.log | perl -pe 's/(INF)/\e[0;32m$1\e[0m/g,s/(WARN)/\e[0;33m$1\e[0m/g,s/(ERROR)/\e[1;31m$1\e[0m/g'

multitail

可同时开启多视窗看log，适合用在看部署在很多机器上的项目的log -cS [color_scheme] : 可以选择输出的log的颜色，推荐使用goldengate，也可自定义(修改/etc/multitail.conf)

参考

Linux性能优化实战

linux性能之巅

如何使用netstat排查网络问题 - 知乎 (zhihu.com)