银河麒麟高级服务器操作系统健康检查手册

文档属性

客户名称		文件类别	技术文档
文件名	服务器操作系统健康检查手册	是否保密	否
编制者	陈思源	编制者职位	售后工程师
编制者邮箱	chensiyuan@kylinos.cn	编制日期	2021-8-11
版本修订记录			
版本号	修订时间	修订说明	
V1.0	2021-8-11	新建	

目录

一、文档说明

服务器的健康检查是服务器的一项重要工作，通过对设备的 CPU、内存、硬盘、网络、服务等使用情况的检查，能及时发现隐患，将故障处理在萌芽阶段，防止影响业务的重大故障发送，为使银河麒麟高级服务器操作系统健康检查做到规范化，明确化，使得未获人员检查时目标清晰，有迹可循，特制定此健康检查手册。

二、背景

本健康检查手册主要为银河麒麟操作系统而制定，由于银河麒麟操作系统搭建在不同硬件设备上，且版本众多，本手册涉及主要是银河麒麟服务器操作系统。

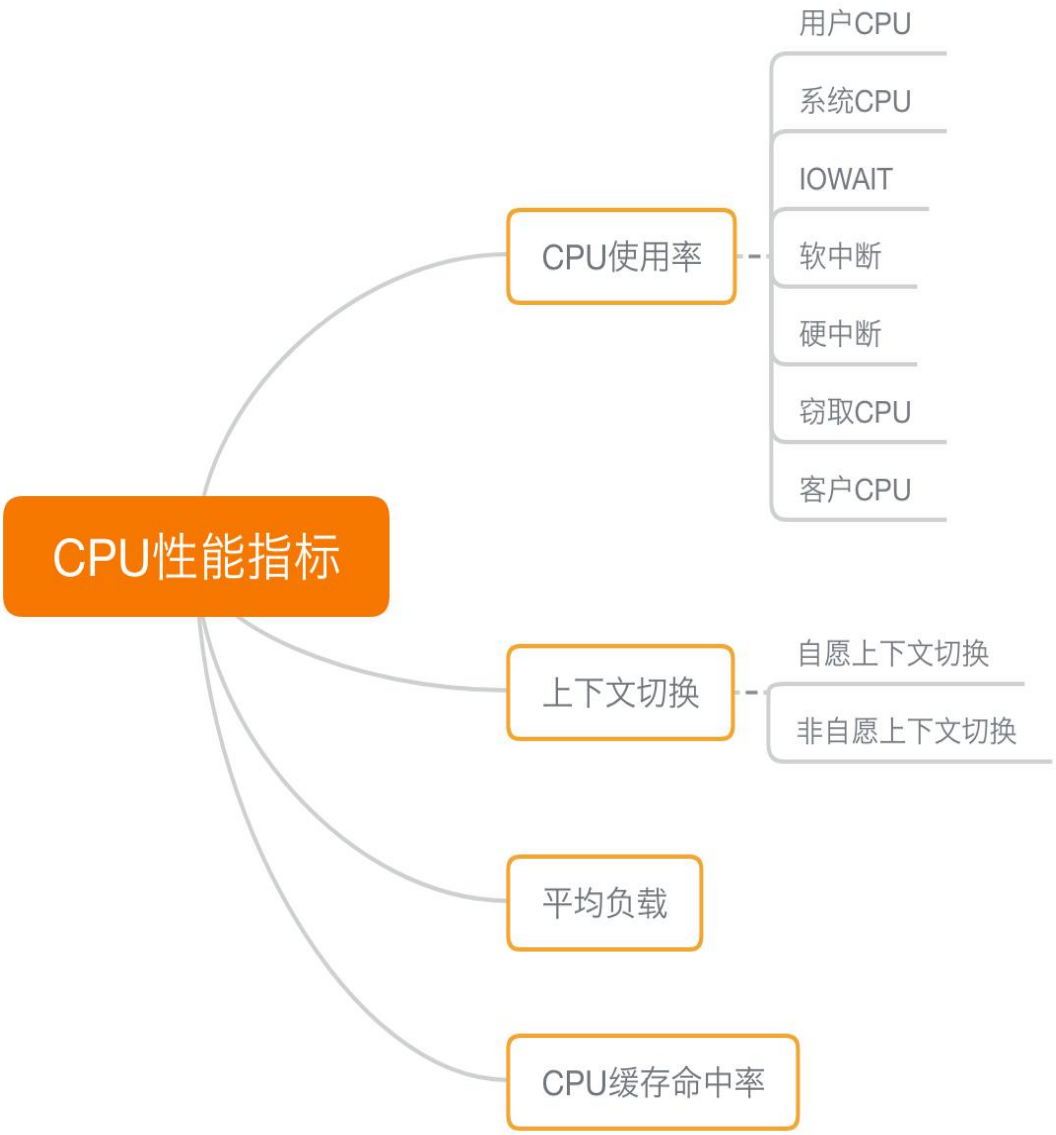
本手册适用于麒麟软件员工，以及各客户现场网管监控人员、现场故障处理人员而制定，使用本手册需要维护人员具有一定的操作系统基础，包括 shell 命令、网络、内存等系统有一定的了解。

三、方案

本手册将介绍系统健康状态的指标，使用银河麒麟操作系统自带的 top、vmstat、iostat、proc、sar 等系统健康状况检查工具的使用方法，给出参考的阈值。

四、CPU

银河麒麟高级服务器操作系统作为一个多任务操作系统,将每个 CPU 的时间划分为很短的时间片,再通过调度器轮流分配给各个任务使用,因此造成多任务同时运行的错觉，我们通常所说的 CPU 使用率,就是除了空闲时间外的其他时间占总 CPU 时间的百分比，还有一些其他的指标同样可以衡量 CPU 使用率，以下为 CPU 性能的相关指标：



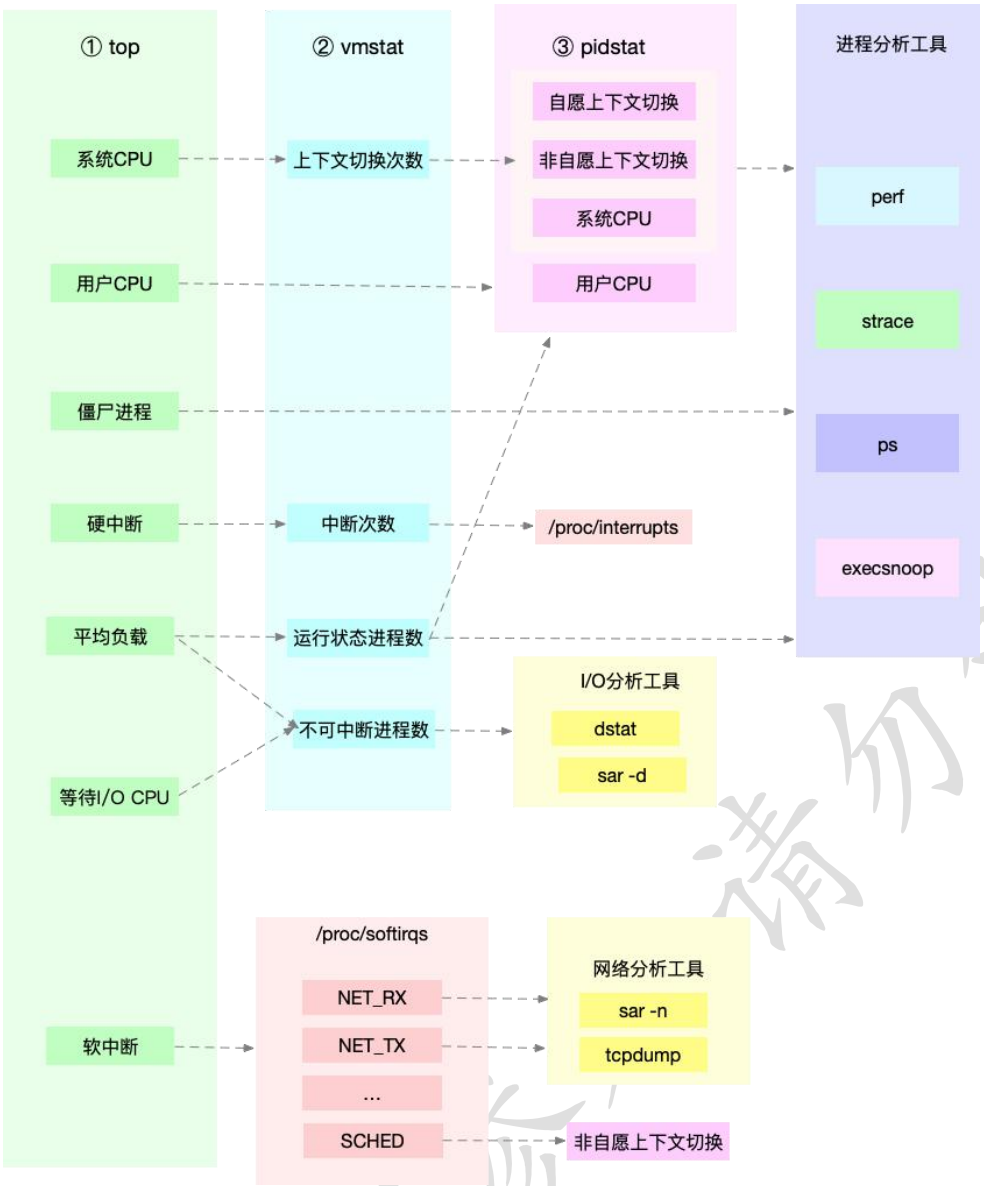
健康监控工具

根据指标找工具

根据指标查性能

根据指标找工具（CPU 性能）			
性能指标	工具	阈值	说明
平均负载	uptime top	波动不超过 200%	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	不超过 80%	top、vmstat、mpstat 只可以动态查看，而 sar 还可以记录历史数据
进程 CPU 使用率	top pidstat ps	不长时间占用	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
系统上下文切换	vmstat	波动不超过 50%	除了上下文切换次数，提供运行状态和不可中断状态进程数量
进程上下文切换	pidstat	波动不超过 50%	注意加上-w 选项
软中断	top /proc/softirqs mpstat	波动不超过 200%	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	波动不超过 200%	vmstat 提供总的中断 CPU 使用率，而 /proc/interrupts 提供各种中断在每个 CPU 上运行的累计次数
网络	dstat sar tcpdump	无丢包情况	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 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 的缓存及内核调用链，execsnoop 可以用来监控短时进程

CPU 健康检查思路



五、内存

在现代处理器中，与 CPU 执行代码或处理信息相比，向内存子系统保存信息或从中读取信息一般花费更长时间。在 CPU 执行指令或处理数据前，它会消耗相当多的空闲时间来等待从内存中取出指令和数据。处理器用不同层次的高速缓存（cache）来弥补这种缓存的内存性能。工具可以显示各种处理器告诉缓存确实发生的位置。

free

total：总内存

userd：已使用

free：未使用

shared：共享内存

buff/cache：缓存和缓冲区

available：新进程可用的

top (按 M 切换到内存排序)

VIRT：虚拟内存

RES：常驻内存

SHR：共享内存

%MEM：使用的物理内存占总内存百分比

内存泄露

使用 vmstat，free 减少而 buff、cache 不变，可能泄露。

swap

内存资源紧张

1、杀死进程

2、内存回收

（一）文件页

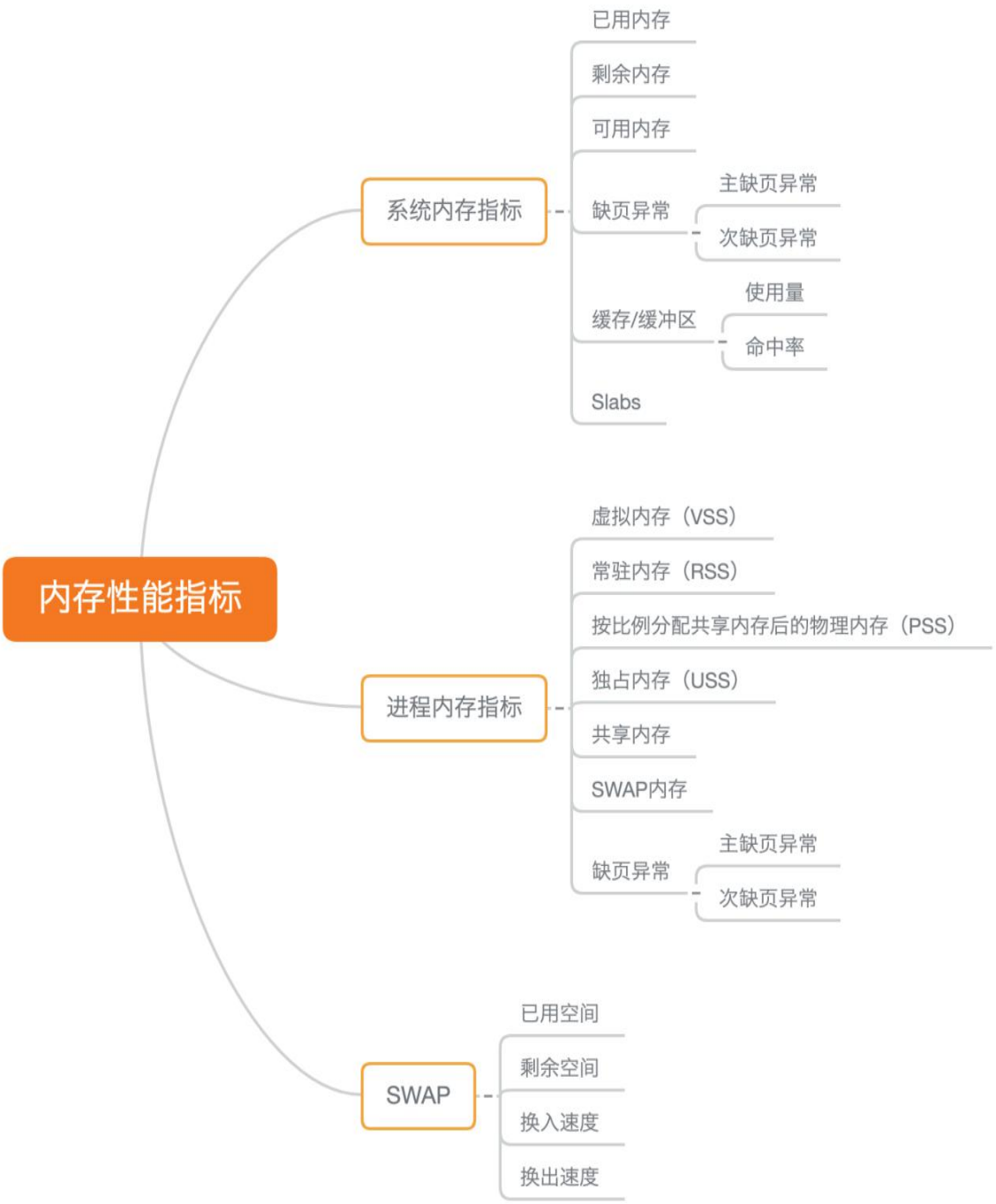
直接回收

脏页（被应用修改过）：写入磁盘后回收

（二）匿名页（堆内存）：使用 Swap 机制写入磁盘

作用：把一块磁盘空间或本地文件当做内存来使用

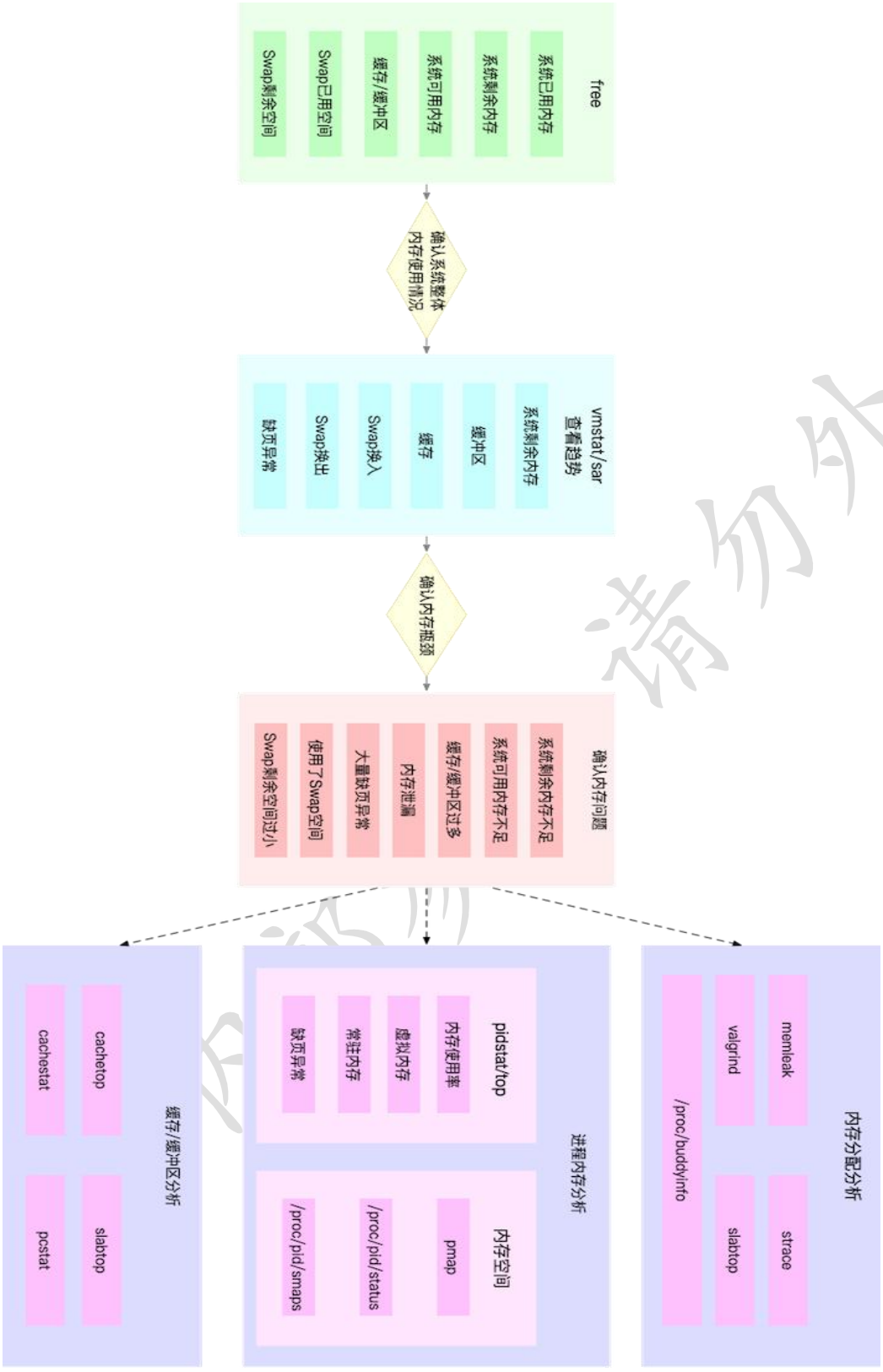
内存性能指标


内存性能指标
系统内存指标
已用内存
剩余内存
可用内存
缺页异常
主缺页异常
次缺页异常
缓存/缓冲区
使用量
命中率
Slabs
进程内存指标
虚拟内存（VSS）
常驻内存（RSS）
按比例分配共享内存后的物理内存（PSS）
独占内存（USS）
共享内存
SWAP内存
缺页异常
主缺页异常
次缺页异常
SWAP
已用空间
剩余空间
换入速度
换出速度

根据指标查找工具

根据指标查工具（内存性能）		
内存指标	阈值	性能工具
系统已用、可用、剩余内存	系统可用内存大于 20%	free vmstat sar /proc/meminfo
进程虚拟内存、常驻内存、共享内存	/	ps top
进程内存分布	/	pmap
进程 Swap 换出内存	换出后 10min 内换出	top vmstat /proc/pid/status
进程缺页异常	/	ps top
系统换页情况	不超过正常 50%	sar
缓存/缓冲区用量	不高过总内存 80%	free vmstat sar cachestat
缓存/缓冲命中率		cachetop
内存泄露检测	无内存泄漏	memleak valgrind

内存健康检查流程



六、文件系统与磁盘 I/O

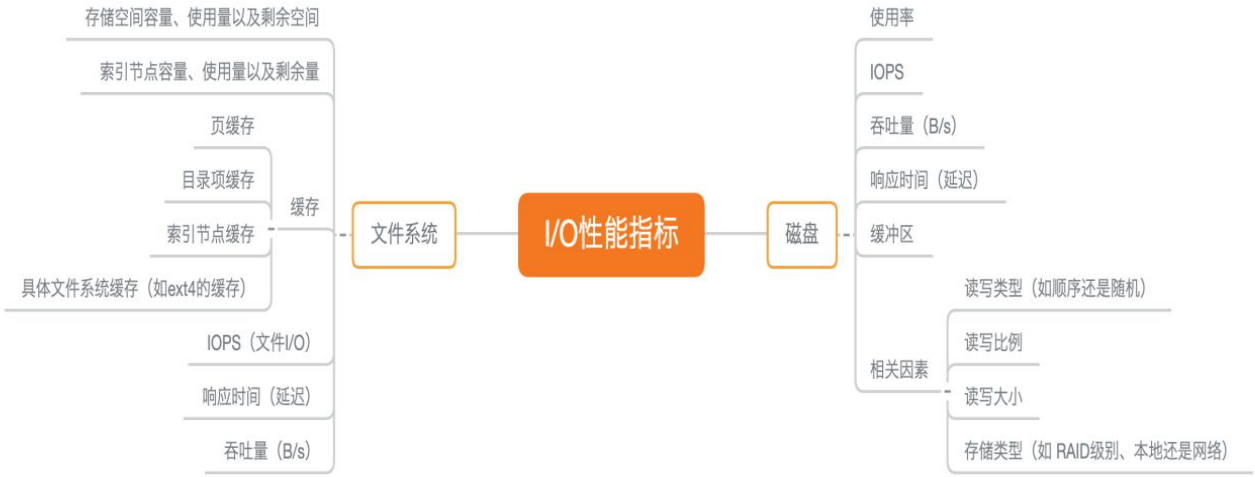
linux 中一切皆文件

每个文件中分配两种数据结构：

索引节点：唯一标识

目录项（缓存），维护文件系统的树状结构

I/O 性能指标

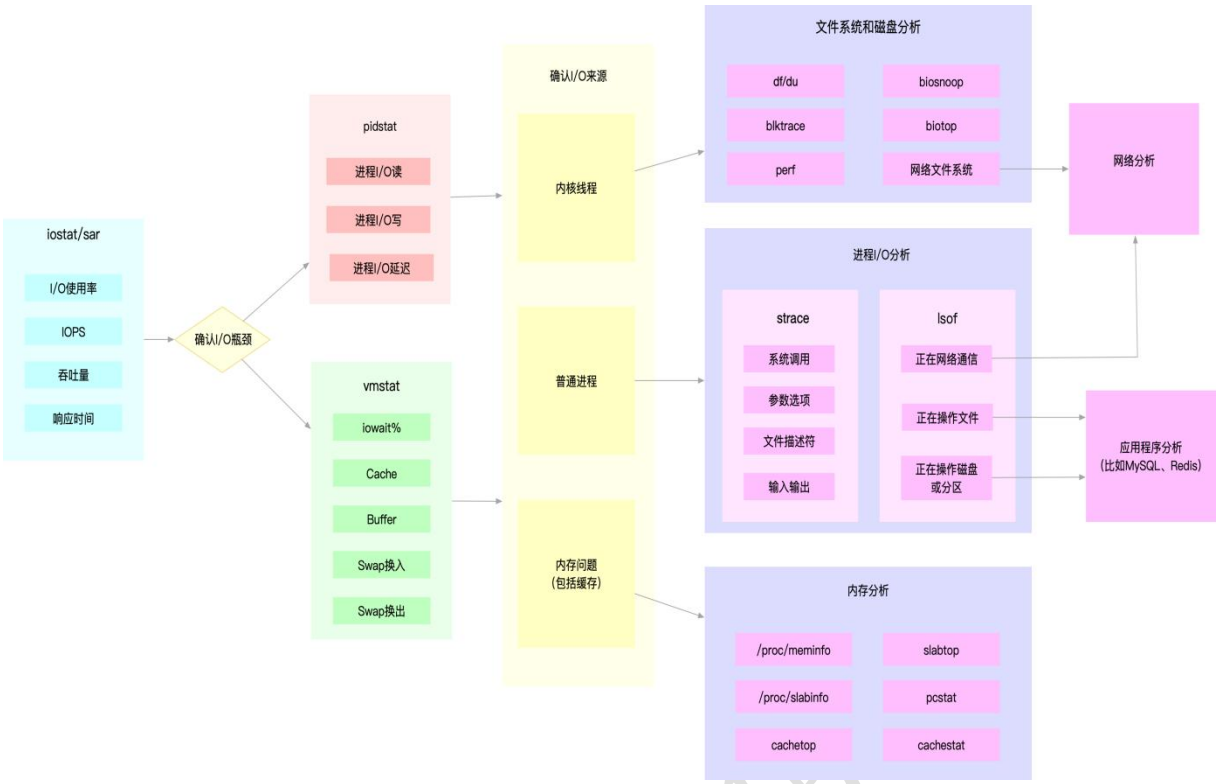


性能指标	含义	说明
r/s	每秒发送给磁盘的读请求数	合并后的请求数
w/s	每秒发送给磁盘的写请求数	合并后的请求数
rkB/s	每秒磁盘读取的数据量	kb
rrqm/s	每秒合并的读数据请求	%rrqm 表示合并读请求的百分比
wrqm/s	每秒合并的请求	%wrqm 表示合并请求的百分比
r_await	读请求处理完成等待时间	包括队列中的等待时间和设备实际处理的时间，单位为毫秒
w_await	写请求处理完成等待时间	包括队列中的等待时间和设备实际处理的时间，单位为毫秒
aqu-sz	平均请求队列长度	旧版中为 avgqu-sz
rareq-sz	平均请求大小	kb
wareq-sz	平均写请求大小	kb
svctm	处理 I/O 所需平均时间（不包括等待时间）	ms，这是推断的数据，并不能保证完全准确
%util	磁盘处理 I/O 的时间百分比	既使用率，由于可能存在并行 I/O，100%并不一定表面磁盘 I/O 饱和

健康检查工具

根据指标查找工具（文件系统和磁盘 I/O）			
性能指标	工具	阈值	说明
文件系统空间容量、使用量以及剩余空间	df	大于 15%	显示磁盘统计情况
索引节点容量、使用量以及剩余量	df	大于 15%	使用-i 选项
页缓存和可回收 slab 缓存	/proc/meminfo、sar、vmstat	/	sar-r
缓冲区	/proc/meminfo、sar、vmstat	/	sar-r
目录项、索引节点	/proc/slabinfo、slabtop	/	slabtop 更直观
磁盘 I/O 使用率、IOPS、吞吐量、响应时间、I/O 平均大小、等待队列长度	iostat sar dstat	/	iostat -d -x 或 sar -d
进程 I/O 大小以及 I/O 延迟	pidstat iotop	不大于 10s	pidstat -d
块设备 I/O 事件跟踪	blktrace	/	blktrace -d /dev/sda -o |blkparse -i
进程 I/O 系统调用跟踪	strace	/	通过系统调用跟踪进程的 I/O

I/O 健康检查思路



IO 异常定位步骤

top 发现 wa 即 iowait 高
iostat -d -x 1 发现磁盘读取瓶颈
pidstat -d 1 (-d 表示 io 情况) 发现大量读取的进程并符合 iostat 的结果，由此找到进程
用 lsof 和 strace 看下进程具体在读什么

七、网络

带宽 b/s
吞吐量 b/s
使用率 吞吐量/带宽
延时（区分场景）
FPS（包/S），评估转发能力
其他：可用性，并发连接数，丢包率，重传率

网络配置命令

ifconfig，ip -s addr
状态：RUNNING（ifconfig），LOWER_UP(ip)
MTU：最大传输单元，最大 IP 包大小，超过则分片

IP，MAC，子网
TX，RX 的 error

套接字状态

netstat -nlp，ss-nlp
I：只监听套接字，n：显示数字地址和端口，p：进程信息
套接字处于 establish
recv-q: 套接字缓冲未被应用程序取走的字节数
send-q: 未被远程主机确认的字节数
处于 listening
recv-q: sync backlog(半连接队列长度，即三次握手未完成)当前值
send-q:sync backlog 最大值

协议栈统计

sar -n DEV(网络接口) 1
rxpck/s 接收 pps
rxkB/s 接受吞吐量
rxcmp/s 接受压缩包数
%ifutil 网络接口使用率
网卡速度: ethtool eth0 grep Speed
延时: ping, icmp 协议

八、服务

Systemd 是 Linux 系统工具，用来启动守护进程，已成为大多数发行版的标准配置。

Systemd 并不是一个命令，而是一组命令，涉及到系统管理的方方面面。

服务配置信息

Systemd 默认从目录/etc/systemd/system/读取配置文件。但是，里面存放的大部分文件都是符号链接，指向目录/usr/lib/systemd/system/，真正的配置文件存放在那个目录。

如果配置文件里面设置了开机启动，systemctl enable 命令相当于激活开机启动。

与之对应的，systemctl disable 命令用于在两个目录之间，撤销符号链接关系，相当于撤销开机启动。

```
[Unit]
Description=vsftpd FTP server
After=network.target

[Service]
Type=simple
ExecStart=/usr/sbin/vsftpd /etc/vsftpd.conf
ExecReload=/bin/kill -HUP $MAINPID
ExecStartPre=-/bin/mkdir -p /var/run/vsftpd/empty

[Install]
WantedBy=multi-user.target
~
~
```

配置文件由区块组成，常用的区块有三个

Unit

[Unit]区块通常是配置文件的第一个区块，用来定义 Unit 的元数据，以及配置与其他 Unit 的关系。它的主要字段如下：

Description：简短描述

Documentation：文档地址

Requires：当前 Unit 依赖的其他 Unit，如果它们没有运行，当前 Unit 会启动失败

Wants：与当前 Unit 配合的其他 Unit，如果它们没有运行，当前 Unit 不会启动失败

BindsTo：与 Requires 类似，它指定的 Unit 如果退出，会导致当前 Unit 停止运行

Before：如果该字段指定的 Unit 也要启动，那么必须在当前 Unit 之后启动

After：如果该字段指定的 Unit 也要启动，那么必须在当前 Unit 之前启动

Conflicts：这里指定的 Unit 不能与当前 Unit 同时运行

Condition...：当前 Unit 运行必须满足的条件，否则不会运行

Assert...：当前 Unit 运行必须满足的条件，否则会报启动失败

INSTALL

[Install]通常是配置文件的最后一个区块，用来定义如何启动，以及是否开机启动。它的主要字段如下。

WantedBy：它的值是一个或多个 Target，当前 Unit 激活时（enable）符号链接会放入/etc/systemd/system 目录下面以 Target 名 + .wants 后缀构成的子目录中

RequiredBy：它的值是一个或多个 Target，当前 Unit 激活时，符号链接会放入/etc/systemd/system 目录下面以 Target 名 + .required 后缀构成的子目录中

Alias：当前 Unit 可用于启动的别名

Also：当前 Unit 激活（enable）时，会被同时激活的其他 Unit

SERVICE

[Service]区块用来 Service 的配置，只有 Service 类型的 Unit 才有这个区块。它的主要字段如下。

Type：定义启动时的进程行为。它有以下几种值。

Type=simple：默认值，执行 ExecStart 指定的命令，启动主进程

Type=forking：以 fork 方式从父进程创建子进程，创建后父进程会立即退出

Type=oneshot：一次性进程，Systemd 会等当前服务退出，再继续往下执行

Type=dbus：当前服务通过 D-Bus 启动

Type=notify：当前服务启动完毕，会通知 Systemd，再继续往下执行

Type=idle：若有其他任务执行完毕，当前服务才会运行

ExecStart：启动当前服务的命令

ExecStartPre：启动当前服务之前执行的命令

ExecStartPost：启动当前服务之后执行的命令

ExecReload：重启当前服务时执行的命令

ExecStop：停止当前服务时执行的命令

ExecStopPost：停止当其服务之后执行的命令

RestartSec：自动重启当前服务间隔的秒数

Restart：定义何种情况 Systemd 会自动重启当前服务，可能的值包括 always（总是重启）、on-success、on-failure、on-abnormal、on-abort、on-watchdog

TimeoutSec：定义 Systemd 停止当前服务之前等待的秒数

Environment：指定环境变量

服务管理

```
# 立即启动一个服务
$ sudo systemctl start apache.service
# 立即停止一个服务
$ sudo systemctl stop apache.service
# 重启一个服务
$ sudo systemctl restart apache.service
# 杀死一个服务的所有子进程
$ sudo systemctl kill apache.service
# 重新加载一个服务的配置文件
$ sudo systemctl reload apache.service
# 重载所有修改过的配置文件
$ sudo systemctl daemon-reload
```

显示某个 Unit 的所有底层参数

$ systemctl show httpd.service

显示某个 Unit 的指定属性的值

$ systemctl show -p CPUShares httpd.service

设置某个 Unit 的指定属性

$ sudo systemctl set-property httpd.service CPUShares=500

systemctl status

systemctl status 命令用于查看系统状态和单个 Unit 的状态。# 重启系统

显示系统状态

$ systemctl status

显示单个 Unit 的状态

$ sysystemctl status bluetooth.service

显示远程主机的某个 Unit 的状态

$ systemctl -H root@rhel7.example.com status httpd.service

systemd-analyze

systemd-analyze 命令用于查看启动耗时。

查看启动耗时

$ systemd-analyze

查看每个服务的启动耗时

$ systemd-analyze blame

显示瀑布状的启动过程流

$ systemd-analyze critical-chain

显示指定服务的启动流

$ systemd-analyze critical-chain atd.service

除了 status 命令，systemctl 还提供了三个查询状态的简单方法，主要供脚本内部的判断语句使用。

显示某个 Unit 是否正在运行

$ systemctl is-active application.service

显示某个 Unit 是否处于启动失败状态

$ systemctl is-failed application.service

显示某个 Unit 服务是否建立了启动链接

$ systemctl is-enabled application.service

附录：健康检查常用工具

vmstat

vmstat 命令是最常见的 Linux/Unix 监控工具，属于 sysstat 包。可以展现给定时间间隔的服务器的状态值,包括服务器的 CPU 使用率，内存使用，虚拟内存交换情况,IO 读写情况。这个命令是我查看 Linux/Unix 最喜爱的命令，一个是 Linux/Unix 都支持，二是相比 top，可以看到整个机器的 CPU,内存,IO 的使用情况，而不是单单看到各个进程的 CPU 使用率和内存使用率(使用场景不一样)。

用法

vmstat [-n] [延时[次数]]

Linux 内存监控 vmstat 命令输出分成六个部分：

1、进程 procs

r：在运行队列中等待的进程数 ，当这个值超过了 CPU 数目，就会出现 CPU 瓶颈了。这个也和 top 的负载有关系，一般负载超过了 3 就比较高，超过了 5 就高，超过了 10 就不正常了，服务器的状态很危险。top 的负载类似每秒的运行队列。如果运行队列过大，表示你的 CPU 很繁忙，一般会造成 CPU 使用率很高。

b：在等待 io 的进程数 ，这个不多说，表示进程阻塞。

2、内存 memoy

swpd：正在使用虚拟的内存大小（单位 KB）。

free：空闲的内存（单位 KB）。

buff: 用作缓冲的内存大小（单位：KB）。

cache：用作缓存的内存大小（单位：KB）。

3、swap 交换页面

si: 每秒从交换区写入内存的大小，单位：KB/秒。

so: 每秒从内存写到交换区的大小，单位：KB/秒。

4、io 块设备 （block）

bi：每秒读取的块数（读磁盘），单位：块/秒。

bo：每秒写入的块数（写磁盘)，单位：块/秒。

5、system 系统

in: 每秒的中断数，包括时钟中断。（interrupt），值越大，会看到由内核消耗的 cpu 时间 sy 会越多秒上下文切换次数，例如我们调用系统函数，就要进行上下文切换，线 程的切换，也要进程上下文切换，这个值要越小越好，太大了，要考虑调低线程或者进 程的数目。

cs: 每秒的环境（上下文）转换次数（context switch），同上。

6、cpu 中央处理器

us：用户进程执行消耗 cpu 时间(user time)，us 的值比较高时，说明用户进程消耗 的 cpu 时间多，但是如果长期超过 50%的使用，那么我们就该考虑优化程序算法或其 他措施了。

sy：系统进程消耗 cpu 时间(system time)，sys 的值过高时，说明系统内核消耗的 cpu 资源多，这个不是良性的表现，我们应该检查原因。这里 us + sy 的参考值为 80%， 如果 us+sy 大于 80%说明可能存在 CPU 不足

Id：空闲时间(包括 IO 等待时间),一般来说 us+sy+id=100

wa：等待 IO 时间 wa 过高时，说明 io 等待比较严重，这可能是由于磁盘大量随机访 问造成的，也有可能是磁盘的带宽出现瓶颈。

常见问题及解决方法

如果 r 经常大于 4，且 id 经常少于 40，表示 cpu 的负荷很重。

如果 si，so 长期不等于 0，表示内存不足。

如果 disk 经常不等于 0，且在 b 中的队列大于 3，表示 io 性能不好。

1.)如果在 processes 中运行的序列(processr)是连续的大于在系统中的 CPU 的个数表示系统现在运行比较慢,有多数的进程等待 CPU。

2.)如果 r 的输出数大于系统中可用 CPU 个数的 4 倍的话,则系统面临着 CPU 短缺的问题,或者是 CPU 的速率过低,系统中有多数的进程在等待 CPU,造成系统中进程运行过慢。

3.)如果空闲时间(cpu id)持续为 0 并且系统时间(cpu sy)是用户时间的两倍(cpu us)系统则面临着 CPU 资源的短缺。

当发生以上问题的时候请先调整应用程序对 CPU 的占用情况.使得应用程序能够更有效的使用 CPU.同时可以考虑增加更多的 CPU. 关于 CPU 的使用情况还可以结合 mpstat, ps aux top prstat –a 等等一些相应的命令来综合考虑关于具体的 CPU 的使用情况,和那些进程在占用大量的 CPU 时间.一般情况下，应用程序的问题会比较大一些.比如一些 sql 语句不合理等等都会造成这样的现象.

内存问题现象:

内存的瓶颈是由 scan rate (sr)来决定的.scan rate 是通过每秒的始终算法来进行页扫描的.如果 scan rate(sr)连续的大于每秒 200 页则表示可能存在内存缺陷.同样的如果 page 项中的 pi 和 po 这两栏表示每秒页面的调入的页数和每秒调出的页数.如果该值经常为非零值,也有可能存在内存的瓶颈,当然,如果个别的时候不为 0 的话,属于正常的页面调度这个是虚拟内存的主要原理.

解决办法:

1.调节 applications & servers 使得对内存和 cache 的使用更加有效.

2.增加系统的内存.

3. Implement priority paging in s in pre solaris 8 versions by adding line "set priority paging=1" in /etc/system. Remove this line if upgrading from Solaris 7 to 8 & retaining old /etc/system file.

关于内存的使用情况还可以结 ps aux top prstat –a 等等一些相应的命令来综合考虑关于具体的内存的使用情况,和那些进程在占用大量的内存.一般情况下，如果内存的占用率比较高,但是,CPU 的占用很低的时候,可以考虑是有很多的应用

程序占用了内存没有释放,但是,并没有占用 CPU 时间,可以考虑应用程序,对于未占用 CPU 时间和一些后台的程序,释放内存的占用。

r 表示运行队列(就是说多少个进程真的分配到 CPU)，我测试的服务器目前 CPU 比较空闲，没什么程序在跑，当这个值超过了 CPU 数目，就会出现 CPU 瓶颈了。这个也和 top 的负载有关系，一般负载超过了 3 就比较高，超过了 5 就高，超过了 10 就不正常了，服务器的状态很危险。top 的负载类似每秒的运行队列。如果运行队列过大，表示你的 CPU 很繁忙，一般会造成 CPU 使用率很高。

5.常见性能问题分析

IO/CPU/men 连锁反应

1.free 急剧下降

2.buff 和 cache 被回收下降，但也无济于事

3.依旧需要使用大量 swap 交换分区 swpd

4.等待进程数，b 增多

5.读写 IO，bi bo 增多

6.si so 大于 0 开始从硬盘中读取

7.cpu 等待时间用于 IO 等待，wa 增加

内存不足

1.开始使用 swpd，swpd 不为 0

2.si so 大于 0 开始从硬盘中读取

io 瓶颈

1.读写 IO，bi bo 增多超过 2000

2.cpu 等待时间用于 IO 等待，wa 增加 超过 20

3.sy 系统调用时间长，IO 操作频繁会导致增加 >30%

4.wa io 等待时间长

iowait% <20%	良好
iowait% <35%	一般
iowait% >50%	

5.进一步使用 iostat 观察

CPU 瓶颈：load,vmstat 中 r 列

1.反应为 CPU 队列长度

2.一段时间内，CPU 正在处理和等待 CPU 处理的进程数之和，直接反应了 CPU 的使用和申请情况。

3.理想的 load average：核数*CPU 数*0.7

CPU 个数：grep 'physical id' /proc/cpuinfo | sort -u

核数：grep 'core id' /proc/cpuinfo | sort -u | wc -l

4.超过这个值就说明已经是 CPU 瓶颈了

CPU 瓶颈

1.us 用户 CPU 时间高超过 90%

涉及到 web 服务器，cs 每秒上下文切换次数

例如我们调用系统函数，就要进行上下文切换，线程的切换，也要进程上下文切换，这个值要越小越好，太大了，要考虑调低线程或者进程的数目,例如在 apache 和 nginx 这种 web 服务器中，我们一般做性能测试时会进行几千并发甚至几万并发的测试，选择 web 服务器的进程可以由进程或者线程的峰值一直下调，压测，直到 cs 到一个比较小的值，这个进程和线程数就是比较合适的值了。系统调用也是，每次调用系统函数，我们的代码就会进入内核空间，导致上下文切换，这个是很耗资源，也要尽量避免频繁调用系统函数。上下文切换次数过多表示你的 CPU 大部分浪费在上下文切换，导致 CPU 干正经事的时间少了，CPU 没有充分利用，是不可取的。

1.cs 可以对 apache 和 nginx 线程和进程数限制起到一定的参考作用

2.我们一般做性能测试时会进行几千并发甚至几万并发的测试，选择 web 服务器的进程可以由进程或者线程的峰值一直下调，压测，直到 cs 到一个比较小的值，这个进程和线程数就是比较合适的值了

top

top 命令经常用来监控 linux 的系统状况，是常用的性能分析工具，能够实时显示系统中各个进程的资源占用情况。

语法

top 的使用方式 top [-d number] | top [-bnp]

参数含义

-d：number 代表秒数，表示 top 命令显示的页面更新一次的间隔。默认是 5 秒。

-b：以批次的方式执行 top。

-n：与-b 配合使用，表示需要进行几次 top 命令的输出结果。

-p：指定特定的 pid 进程号进行观察。

在 top 命令显示的页面还可以输入以下按键执行相应的功能（注意大小写区分的）：

?：显示在 top 当中可以输入的命令

P：以 CPU 的使用资源排序显示

M：以内存的使用资源排序显示

N：以 pid 排序显示

T：由进程使用的时间累计排序显示

k：给某一个 pid 一个信号。可以用来杀死进程

r：给某个 pid 重新定制一个 nice 值（即优先级）

q：退出 top（用 ctrl+c 也可以退出 top）。

top 前 5 行信息统计

top - 05:43:27 up 4:52, 2 users, load average: 0.58, 0.41, 0.30

05:43:27　　表示当前时间

up 4:52 系统运行时间 格式为时：分

2 users 当前登录用户数

load average: 0.58, 0.41, 0.30 系统负载，即任务队列的平均长度。 三个数值分别为 1 分钟、5 分钟、15 分钟前到现在的平均值。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。

Tasks: 159 total, 1 running, 158 sleeping, 0 stopped, 0 zombie

%Cpu(s): 37.0 us, 3.7 sy, 0.0 ni, 59.3 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

159 total　　进程总数

1 running　　正在运行的进程数

158 sleeping 睡眠的进程数

0 stopped　　停止的进程数

0 zombie　　僵尸进程数

37.0 us 用户空间占用 CPU 百分比

3.7 sy　内核空间占用 CPU 百分比

0.0 ni　用户进程空间内改变过优先级的进程占用 CPU 百分比

59.3 id 空闲 CPU 百分比

0.0 wa 等待输入输出的 CPU 时间百分比

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

KiB Mem: 1530752 total, 1481968 used, 48784 free, 70988 buffers

KiB Swap: 3905532 total, 267544 used, 3637988 free. 617312 cached Mem

KiB Mem: 1530752 total 物理内存总量

1481968 used　使用的物理内存总量
48784 free　空闲内存总量
70988 buffers（buff/cache） 用作内核缓存的内存量
KiB Swap: 3905532 total 交换区总量
267544 used　使用的交换区总量
3637988 free　空闲交换区总量
617312 cached Mem 缓冲的交换区总量。
3156100 avail Mem　代表可用于进程下一次分配的物理内存数量
进程信息
PID 进程 id
PPID　父进程 id
RUSER Real user name
UID 进程所有者的用户 id
USER　进程所有者的用户名
GROUP 进程所有者的组名
TTY 启动进程的终端名。不是从终端启动的进程则显示为 ？
PR 优先级
NI　nice 值。负值表示高优先级，正值表示低优先级
P　最后使用的 CPU，仅在多 CPU 环境下有意义
%CPU　上次更新到现在的 CPU 时间占用百分比
TIME　进程使用的 CPU 时间总计，单位秒
TIME+　进程使用的 CPU 时间总计，单位 1/100 秒
%MEM 进程使用的物理内存百分比
VIRT　进程使用的虚拟内存总量，单位 kb。VIRT=SWAP+RES
SWAP　进程使用的虚拟内存中，被换出的大小，单位 kb
RES　进程使用的、未被换出的物理内存大小，单位 kb。RES=CODE+DATA
CODE　可执行代码占用的物理内存大小，单位 kb
DATA　可执行代码以外的部分(数据段+栈)占用的物理内存大小，单位 kb
SHR　共享内存大小，单位 kb
nFLT　页面错误次数
nDRT　最后一次写入到现在，被修改过的页面数。
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
COMMAND 命令名/命令行
WCHAN 若该进程在睡眠，则显示睡眠中的系统函数名
Flags　任务标志
示例
TOP 各进程是按照 CPU 的占用量进行排序的，按数字“1”可以监控每个逻辑 CPU 的状况

```
top - 14:10:30 up  5:31,  1 user,  load average: 0.24, 0.53, 0.65
任务: 308 total,   1 running, 305 sleeping,   0 stopped,   2 zombie
%Cpu0  :  2.7 us,  0.7 sy,  0.0 ni, 96.0 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu1  :  2.0 us,  0.3 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu2  :  2.7 us,  0.3 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  2.7 us,  0.0 sy,  0.0 ni, 97.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :  1.0 us,  0.7 sy,  0.0 ni, 98.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  0.7 us,  0.7 sy,  0.0 ni, 98.0 id,  0.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  2.3 us,  1.3 sy,  0.0 ni, 94.6 id,  1.7 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  :  0.7 us,  0.7 sy,  0.0 ni, 98.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  15527.0 total,   7898.5 free,   3642.2 used,   3986.3 buff/cache
MiB Swap:  18631.0 total,  18510.0 free,    121.0 used.  11033.6 avail Mem
```

在 top 的基本视图中，敲击“f”进入另一个视图，在这里可以编辑基本视图中的显示字段用上下键进行选择，空格键决

定是否在基本视图中显示这个选项。

```
Fields Management for window 1:Def, whose current sort field is %CPU
   Navigate with Up/Dn, Right selects for move then <Enter> or Left commits,
   'd' or <Space> toggles display, 's' sets sort.  Use 'q' or <Esc> to end!

* 进程号   = Process Id      PGRP    = Process Group   OOMs    = OOMEM Score c
* USER     = Effective Use   TTY     = Controlling T   ENVIRON = Environment v
* PR       = 优先级          TPGID   = Tty Process G   vMj     = Major Faults
* NI       = Nice Value      SID     = Session Id      vMn     = Minor Faults
* VIRT     = Virtual Image   nTH     = Number of Thr   USED    = Res+Swap Size
* RES      = Resident Size   P       = Last Used Cpu   nsIPC   = IPC namespace
* SHR      = Shared Memory   时间    = CPU Time        nsMNT   = MNT namespace
* 日       = Process Statu   SWAP    = Swapped Size    nsNET   = NET namespace
* %CPU     = CPU Usage       CODE    = Code Size (Ki   nsPID   = PID namespace
* %MEM     = Memory Usage    DATA    = Data+Stack (K   nsUSER  = USER namespac
* TIME+    = CPU Time, hun   nMaj    = Major Page Fa   nsUTS   = UTS namespace
* COMMAND  = Command Name/   nMin    = Minor Page Fa   LXC     = LXC container
  PPID     = Parent Proces   nDRT    = Dirty Pages C   RSan    = RES Anonymous
  UID      = Effective Use   WCHAN   = Sleeping in F   RSfd    = RES File-base
  RUID     = Real User Id    标志    = Task Flags <s   RSlk    = RES Locked (K
  RUSER    = Real User Nam   CGROUPS = Control Group   RSsh    = RES Shared (K
  SUID     = Saved User Id   SUPGIDS = Supp Groups I   CGNAME  = Control Group
  SUSER    = Saved User Na   SUPGRPS = Supp Groups N   NU      = Last Used NUM
  GID      = Group Id        TGID    = Thread Group
  GROUP    = Group Name      OOMa    = OOMEM Adjustm
```

top 是一个功能强大的工具，但它监控最小的单位是进程，如果想监控更小单位时，就需要用到 ps 或 netstate 命令来满足我们的要求

lscpu

lscpu 是用来显示 cpu 的相关信息 scpu 从 sysfs 和/proc/cpuinfo 收集 cpu 体系结构信息，命令的输出比较易读，命令输出的信息包含 cpu 数量，线程，核数，套接字和 Nom-Uniform Memeor Access(NUMA)，缓存等。

语法：

lscpu [-a|-b|-c] [-x] [-s directory] [-e [=list]|-p [=list]]

lscpu -h|-V

参数选项：

-a, –all: 包含上线和下线的 cpu 的数量，此选项只能与选项 e 或-p 一起指定

-b, –online: 只显示出上线的 cpu 数量，此选项只能与选项 e 或者-p 一起指定

-c, –offline: 只显示出离线的 cpu 数量，此选项只能与选项 e 或者-p 一起指定

-e, –extended [=list]: 以人性化的格式显示 cpu 信息，如果 list 参数省略，输出所有可用数据的列，在指定了 list 参数时，选项的字符串、等号(=)和列表必须不包含任何空格或其他空白。比如：’-e=cpu,node’ or ’–extended=cpu,node’

-h, –help:帮助

-p, –parse [=list]: 优化命令输出，便于分析.如果省略 list,则命令的输出与早期版本的 lscpu 兼容，兼容格式以两个逗号用于分隔 cpu 缓存列，如果没有发现 cpu 缓存，则省略缓存列，如果使用 list 参数，则缓存列以冒号（:）分隔。在指定了 list 参数时，选项的字符串、等号(=)和列表必须不包含空格或者其它空白。比如：’-e=cpu,node’ or ’–extended=cpu,node’

-s, –sysroot directory: 为一个 Linux 实例收集 CPU 数据，而不是发出 lscpu 命令的实例。指定的目录是要检查 Linux 实例的系统根

-x, –hex:使用十六进制来表示 cpu 集合，默认情况是打印列表格式的集合(例如：0，1)

示例

```
架构：                           x86_64
CPU 运行模式：                   32-bit, 64-bit
字节序：                         Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU:                             8
在线 CPU 列表：                  0-7
每个核的线程数：                 2
每个座的核数：                   4
座：                             1
NUMA 节点：                      1
厂商 ID:                         GenuineIntel
CPU 系列：                       6
型号：                           126
型号名称：                       Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
步进：                           5
CPU MHz:                         3510.743
CPU 最大 MHz:                    3900.0000
CPU 最小 MHz:                    400.0000
BogoMIPS:                        2995.20
虚拟化：                         VT-x
L1d 缓存：                       192 KiB
L1i 缓存：                       128 KiB
```

参数含义：

Architecture: 架构
CPU(s): 逻辑 cpu 颗数
Thread(s) per core: 每个核心线程
Core(s) per socket: 每个 cpu 插槽核数/每颗物理 cpu 核数
CPU socket(s): cpu 插槽数
Vendor ID: cpu 厂商 ID
CPU family: cpu 系列
Model: 型号
Stepping: 步进
CPU MHz: cpu 主频
Virtualization: cpu 支持的虚拟化技术
L1d cache: 一级缓存（google 了下，这具体表示表示 cpu 的 L1 数据缓存）
L1i cache: 一级缓存（具体为 L1 指令缓存）
L2 cache: 二级缓存

stress

-?
--help 显示帮助信息
--version 显示软件版本信息
-t secs:
--timeout secs 指定运行多少秒
--backoff usecs 等待 usecs 微秒后才开始运行
-c forks:
--cpu forks 产生多个处理 sqrt()函数的 CPU 进程
-m forks

--vm forks:产生多个处理 malloc()内存分配
-i forks
--io forks:产生多个处理 sync()函数的磁盘 I/O 进程
--vm-bytes bytes：指定内存的 byte 数，默认值是 1
--vm-hang:表示 malloc 分配的内存多少时间后在 free()释放掉
-d：
--hdd:写进程，写入固定大小，通过 mkstemp()函数写入当前目录
--hdd-bytes bytes:指定写的 byte 数，默认 1G
--hdd-noclean:不要将写入随机 ascii 数据的文件 unlink，则写入的文件不删除，会保留在硬盘空间。

stress-ng

stress-ng 的参数有几百项，可以模拟复杂的压力测试，但是兼容 stress 的参数。 主要使用参数：
-c N ：运行 N worker CPU 压力测试进程
--cpu-method all ：worker 从迭代使用 30 多种不同的压力算法，包括 pi, crc16, fft 等等
-tastset N：将压力加到指定核心上
-d N：运行 N worker HDD write/unlink 测试
-i N：运行 N worker IO 测试

mpstat

（sysstat 包）
用法
mpstat [选项] [<时间间隔> [<次数>]]
常用参数
-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 delay 一起使用
当 mpstat 不带参数时，输出为从系统启动以来的平均值。

监控所有 CPU 信息每 5 秒生成一个报告，重复 2 次
输出参数含义
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 值表示进程的优先级，数值越小优先级越高） 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（低表示 CPU 不足） idle/total*10

pidstat

（sysstat 包）
作用
监控全部或指定进程的 cpu、内存、线程、设备 IO 等系统资源的占用情况
用法
pidstat [选项] [<时间间隔>] [<次数>]
常用参数
-u：默认的参数，显示各个进程的 cpu 使用统计
-r：显示各个进程的内存使用统计
-d：显示各个进程的 IO 使用情况
-p：指定进程号
-w：显示每个进程的上下文切换情况
-t：显示选择任务的线程的统计信息外的额外信息
-T { TASK | CHILD | ALL }
这个选项指定了 pidstat 监控的。TASK 表示报告独立的 task，CHILD 关键字表示报告进程下所有线程统计信息。ALL 表示报告独立的 task 和 task 下面的所有线程。
注意：task 和子线程的全局的统计信息和 pidstat 选项无关。这些统计信息不会对应到当前的统计间隔，这些统计信息只有在子线程 kill 或者完成的时候才会被收集。
-h：在一行上显示了所有活动，这样其他程序可以容易解析。
-I：在 SMP 环境，表示任务的 CPU 使用率/内核数量
-l：显示命令名和所有参数
示例一：查看 PID 为 14561 的进程 CPU、内存、IO 使用情况

```
root@kylin-PC:/home/kylin# pidstat -u -r -d -p 14561
Linux 5.4.18-19-generic (kylin-PC)      2021年03月23日   _x86_64_        (8 CPU)

10时40分32秒    UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Comm
and
10时40分32秒   1000     14561    3.23    0.00    0.00    0.83    3.23     4  stre
ss

10时40分32秒    UID       PID  minflt/s  majflt/s     VSZ     RSS   %MEM  Command
10时40分32秒   1000     14561      0.00      0.00    3856     104   0.00  stress

10时40分32秒    UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s iodelay  Command
10时40分32秒   1000     14561      0.00      0.00      0.00       0  stress
```

参数含义
PID：进程 ID
%usr：进程在用户空间占用 cpu 的百分比
%system：进程在内核空间占用 cpu 的百分比
%guest：进程在虚拟机占用 cpu 的百分比
%CPU：进程占用 cpu 的百分比
CPU：处理进程的 cpu 编号
Command：当前进程对应的命令
Minflt/s:任务每秒发生的次要错误，不需要从磁盘中加载页
Majflt/s:任务每秒发生的主要错误，需要从磁盘中加载页
VSZ：虚拟地址大小，虚拟内存的使用 KB
RSS：常驻集合大小，非交换区五里内存使用 KB
kB_rd/s：每秒从磁盘读取的 KB
kB_wr/s：每秒写入磁盘 KB
kB_ccwr/s：任务取消的写入磁盘的 KB。当任务截断脏的 pagecache 的时候会发生。
示例二：显示 PID 为 14561 的进程的上下文切换情况

```
root@kylin-PC:/home/kylin# pidstat -w -p 14561
Linux 5.4.18-19-generic (kylin-PC)      2021年03月23日   _x86_64_        (8 CPU)

10时45分03秒    UID       PID   cswch/s nvcswch/s  Command
10时45分03秒   1000     14561      0.00      8.35  stress
```

参数含义
cswch/s:每秒主动任务上下文切换数量
nvcswch/s:每秒被动任务上下文切换数量
Command:命令名

cat /proc/interrupts

查看硬件中断/proc/interrupts 文件中列出当前系统使用的中断的情况，所以某个中断处理没有安装，是不会显示的。哪怕之前安装过，被卸载了。

示例：查看当前硬件中断

```
root@kylin-PC:/home/kylin# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
  1:          0          0          0      31753          0          0          0          0  IR-IO-APIC    1-edge      i8042
  8:          0          0          0          0          0          0          0          0  IR-IO-APIC    8-edge      rtc0
  9:          0       1072          0          0          0          0          0          0  IR-IO-APIC    9-fasteoi   acpi
 14:          0          0          0          0          0          0          0          0  IR-IO-APIC   14-fasteoi   INT3455:00
 16:          0          0          0          0          0          0          0          4  IR-IO-APIC   16-fasteoi   idma64.0, i2c_designware.0, i801_smbus
 20:          0          0          0          0          0          0          0          0  IR-IO-APIC   20-fasteoi   idma64.1
120:          0          0          0          0          0          0          0          0  DMAR-MSI    0-edge      dmar0
121:          0          0          0          0          0          0          0          0  DMAR-MSI    1-edge      dmar1
122:          0          0          0          0          0          0          0          0  IR-PCI-MSI 458752-edge      PCIe PME
123:          0          0          0          0          0          0          0          0  IR-PCI-MSI 466944-edge      PCIe PME
124:          0          0          0          0          0          0          0          0  IR-PCI-MSI 475136-edge      PCIe PME
125:          0          0          0          0          0          0          0          0  IR-PCI-MSI 479232-edge      PCIe PME
126:          0          0          0          0          0          0          0          0  IR-PCI-MSI 483328-edge      PCIe PME
127:          0          0          0          0          0          0          0          0  IR-PCI-MSI 487424-edge      PCIe PME
128:          0          0          0          0          0          0          0          0  IR-PCI-MSI 212992-edge      xhci_hcd
129:          0          0     809409          0          0          0          0          0  IR-PCI-MSI 327680-edge      xhci_hcd
130:          0          0          0          0          0          0     517105          0  IR-PCI-MSI 376832-edge      ahci[0000:00:17.0]
131:         19          0          0          0          0          0          0          0  IR-PCI-MSI 2097152-edge      rtsx_pci
132:          0          0          0          0         23          0          0          0  IR-PCI-MSI 2621440-edge      nvme0q0
133:          0          0          0          0          0         20          0          0  IR-PCI-MSI 3145728-edge      nvme1q0
134:          0          0      58790          0          0          0          0          0  IR-PCI-MSI 1572864-edge      enp3s0
135:          0          0    2069576          0          0          0          0          0  IR-PCI-MSI 32768-edge      i915
136:          0          0          0        128          0          0          0          0  IR-PCI-MSI 1048576-edge      amdgpu
137:         11          0          0          0          0          0          0          0  IR-PCI-MSI 2621441-edge      nvme0q1
138:          0          7          0          0          0          0          0          0  IR-PCI-MSI 2621442-edge      nvme0q2
139:          0          0          0          0          0          0          0          0  IR-PCI-MSI 2621443-edge      nvme0q3
```

从左到右分别是：中断的序号、在各自 CPU 上发送中断的次数，可编程中断控制器名，设备注册该

中断使用的名字，设备名称

cat /proc/softirqs

通过/proc/softirqs 文件内容的变化情况，你可以发现， TIMER（定时中断）、NET_RX（网络接收）、SCHED（内核调度）、RCU（RCU 锁）等这几个软中断都在不停变

示例：查看软中断信息

```
root@kylin-PC:/home/kylin# cat /proc/softirqs
                    CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
          HI:          33         28    1611602         26         10         12         57         15
       TIMER:     3126066    3072199    3314225    3080879    3065041    3055373    3116650    3107552
     NET_TX:           3        300       1716       1579          5          1       1573        650
     NET_RX:       13007      12673      71740      13550      14321      13672      13167      12682
      BLOCK:          30          0         10          0         31         31     519229          1
   IRQ_POLL:           0          0          0          0          0          0          0          0
    TASKLET:          20         17         45      20966         17         20         29       5142
      SCHED:     3745054    3351089    3463515    3184326    3183052    3149822    3205357    3189916
    HRTIMER:         392         95          3         11         22         33         10          0
        RCU:     1832707    1792421    1883165    1792997    1801585    1791343    1851504    1819366
```

sysbench

sysbench 是基于 LuaJIT 的可编写脚本的多线程基准测试工具。 它最常用于数据库基准测试，但也可用于创建不涉及数据库服务器的任意复杂的工作负载进行测试。

语法：

sysbench [options]... [testname] [command]

testname 代表 sysbench 执行的程序，如

fileio

cpu

memory

threads

mutex

或是一些代 lua 脚本，路径为/usr/sysbench/share/sysbench

command 是 sysbench 要执行的命令，包括如下几个命令

prepare 代表为测试提前准备数据，如插数或者建立文件等

run 代表真正执行测试

cleanup 代表清理测试过程中产生的数据，如删数或者删除文件等

help 显示测试程序或脚本的用法

options 代表测试的一些命令行选项列表

通用参数

--threads=N 使用的线程数量，默认值为 1。

--events=N 总请求数，与--time 选择一个设置即可，默认值为 0。

--time=N 总执行时间，与--events 选择一个设置即可，单位为 s，默认值为 10。

--forced-shutdown=STRING 超过--time 后强制中断，默认为 off。

--thread-stack-size=SIZE 每个线程的 stack 大小，默认为 64K。

--rte=Na 平均事务率(tps)，0 为不限速，默认值 0

--report-interval=N 表示 N 秒输出一次测试进度报告，0 表示关闭测试进度报告输出，仅输出最终的报告结

果，默认值为 0。
--report-checkpoints=[LIST,...]　　转储完整的统计数据，并在指定的时间点重置所有计数器。参数是一个用逗号分隔的值列表，表示从测试开始到必须执行报告检查点所经过的秒数，默认关闭
--validate[=on|off]　　在可能的情况下执行验证检查，默认为 off。
--config-file=FILENAME　　配置文件路径。
随机数生成参数
--rand-type=STRING 表示随机类型的模式，共有 4 种模式：uniform(固定)，gaussian(高斯)，special(特定)，pareto(帕雷特)，默认值为：special。
--rand-spec-iter=N　用于数字生成的迭代次数，默认值 12
--rand-spec-pct=N　对于’special’随机模式中待处理值的百分比，默认值 1
--rand-spec-res=N　对于’special’随机模式中指定值的比例，默认值为 75。
--rand-seed=N　　随机数生成依据，0 表示按当前时间为基础生成随机数，默认值 0
--rand-pareto-h=N　pareto 分布，默认值 0.2
日志参数
--verbosity=N　日志级别，5 为 debug 信息，0 位仅仅输出严重信息，默认值为 3。
--percentile=N　查询相应时间采样的百分比，0 禁用，默认值为 95%。
--histogram[=on|off] 在报告中打印延迟直方图，默认值关闭
CPU 参数
--cpu-max-prime=N　最大质数生成器的上限，默认值：10000。
fileio 参数
--file-num=N创建文件的数量，默认值：128。
--file-block-size=N　每次 IO 操作的 block 大小，默认值：16K。
--file-total-size=SIZE　　所有文件大小总和，默认值：2G。
--file-test-mode=STRING 测试模式：seqwr(顺序写), seqrewr(顺序读写), seqrd(顺序读), rndrd(随机读), rndwr(随机写), rndrw(随机读写)。
--file-io-mode=STRING　文件操作模式：sync(同步),async(异步),mmap(快速 map 映射)，默认值：sync。
--file-async-backlog=N　每个线程排队的异步操作数，默认值[128]。
--file-extra-flags=[LIST,...]使用额外的标志符来打开文件{sync,dsync,direct}。默认值空
--file-fsync-freq=N　在完成 N 次请求之后，执行 fsync()，0 表示不使用 fsync，默认值：100。
--file-fsync-all[=on|off]　每次写操作后执行 fsync()，默认值：off。
--file-fsync-end[=on|off]　　测试结束后执行 fsync()，默认值：on。
--file-fsync-mode=STRING　使用 fsync 或 fdatasync 方法进行同步，默认值：fsync。
--file-merged-requests=N 尽可能的合并 N 个 IO 请求数，0 表示不合并，默认值：0。
--file-rw-ratio=N　　测试时候的读写比例，默认值：1.5(即 3:2)。
threads 参数
--thread-yields=N　每个请求产生多少线程，默认值：1000。
--thread-locks=N　每个线程的锁的数量，默认值：8。
memory 参数
--memory-block-size=SIZE　测试时内存块大小，默认值：1K。
--memory-total-size=SIZE　传输数据可使用的最大内存大小，默认值：100G。
--memory-scope=STRING　　内存访问范围：global/local，默认值：global。
--memory-hugetlb[=on|off]　从 HugeTLB 池分配内存，默认值：off。
--memory-oper=STRING　　内存操作类型：read/ write/none，默认值：write。
--memory-access-mode=STRING　内存访问方式：seq(顺序)/rnd(随机)，默认值：seq。
mutex 参数
--mutex-num=N 数组互斥的总大小，默认值：4096。

--mutex-locks=N 每个线程互斥锁的数量，默认值：50000。
--mutex-loops=N 内部互斥锁的空循环数量，默认值：1000
数据库参数
--db-driver=STRING 连接数据库的驱动，默认 mysql
--db-ps-mode=STRING SQL 是否需要预编译，模式有：auto/disable，默认为 disable。
--db-debug[=on|off] 输出数据库层面的 debug 信息，默认为 off。

示例一：测试硬盘的 IOPS
sysbench 的性能测试需要做 prepare，run，cleanup 这三步，准备数据，跑数据，删除数据进入/home/kylin/公共的菜单进行测试
首先以随机读写测试模式准备 1 个 15GB 的执行时间 150s

```
root@kylin-PC:/home/kylin/公共的# sysbench fileio --file-num=1 --file-total-size
=15G --file-test-mode=rndrw --time=150 prepare
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

1 files, 15728640Kb each, 15360Mb total
Creating files for the test...
Extra file open flags: (none)
Creating file test_file.0
16106127360 bytes written in 702.42 seconds (21.87 MiB/sec).
```

然后以随机读写测试模式跑测试

```
Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Extra file open flags: (none)
1 files, 15GiB each
15GiB total file size
Block size 16KiB
Number of IO requests: 0
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Initializing worker threads...

Threads started!


File operations:
    reads/s:                      19.10
    writes/s:                     12.73
    fsyncs/s:                     0.32

Throughput:
    read, MiB/s:                  0.30
    written, MiB/s:               0.20

General statistics:
    total time:                          150.7687s
    total number of events:              4848

Latency (ms):
         min:                                    0.00
         avg:                                   31.06
         max:                                 1658.93
         95th percentile:                      150.29
         sum:                               150581.35

Threads fairness:
    events (avg/stddev):           4848.0000/0.00
    execution time (avg/stddev):   150.5813/0.00
```

最后删除数据
sysbench fileio --file-num=1 --file-total-size=15G --file-test-mode=rndrw --time=150 cleanup

```
root@kylin-PC:/home/kylin/公共的# sysbench fileio --file-num=1 --file-total-size=15G --file-test-mod
e=rndrw --time=300 cleanup
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Removing test files...
```

示例二：最大质数生成器运算 1000 次

```
kylin@kylin-PC:~$ sysbench cpu --cpu-max-prime=1000 run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Prime numbers limit: 1000

Initializing worker threads...

Threads started!

CPU speed:
    events per second: 80857.33

General statistics:
    total time:                          10.0001s
    total number of events:              808699

Latency (ms):
         min:                                    0.01
         avg:                                    0.01
         max:                                    0.15
         95th percentile:                        0.01
         sum:                                 9918.12

Threads fairness:
    events (avg/stddev):           808699.0000/0.00
    execution time (avg/stddev):   9.9181/0.00
```

kylin@kylin-PC:~$ sysbench cpu --cpu-max-prime=1000 run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1 #线程个数
Initializing random number generator from current time
Prime numbers limit: 1000#素数上限
Initializing worker threads...
Threads started!
CPU speed:
    events per second: 80857.33 #所有线程平均每秒完成 event 个数
General statistics:
    total time: 10.0001s #总共消耗时间
    total number of events: 808699 #所有线程完成 event 数
Latency (ms):
         min: 0.01
         avg: 0.01 #完成所有 event 平均时间
         max: 0.15 #完成 1 次 event 最多时间
         95th percentile: 0.01 #95%的 event 平均耗时
         sum: 9918.12 #所有线程总耗时
Threads fairness:
    events (avg/stddev): 808699.0000/0.00 #平均每个线程完成 envet 的次数，后一个值是标准差
    execution time (avg/stddev): 9.9181/0.00 #平均每个线程平均耗时，后一个值是标准差

示例三：以 2GB 的内存大小测试内存性能

```
Total operations: 2097152 (6945061.31 per second)

2048.00 MiB transferred (6782.29 MiB/sec)


General statistics:
    total time:                          0.3003s
    total number of events:              2097152

Latency (ms):
         min:                                    0.00
         avg:                                    0.00
         max:                                    0.05
         95th percentile:                        0.00
         sum:                                  133.75

Threads fairness:
    events (avg/stddev):           2097152.0000/0.00
    execution time (avg/stddev):   0.1338/0.00
```

示例四：每个请求产生两个线程，每个线程 4 个锁进行线程测试

```
root@kylin-PC:/home/kylin/公共的# sysbench threads --thread-yields=2 --thread-locks=4 run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Initializing worker threads...

Threads started!


General statistics:
    total time:                          10.0001s
    total number of events:              22390689

Latency (ms):
         min:                                    0.00
         avg:                                    0.00
         max:                                    0.18
         95th percentile:                        0.00
         sum:                                 7987.45

Threads fairness:
    events (avg/stddev):           22390689.0000/0.00
    execution time (avg/stddev):   7.9875/0.00
```

perf

-C 指定统计的 CPU 核心编号，不指定时统计全部核心（等价于-a）

-e 指定统计事件

-p 只统计特定 pid 指定的进程中产生的事件

-t 只统计特定 tid 指定的线程中产生的事件

-K 隐藏内核中的函数符号

-U 隐藏用户态的函数符号

-g perf record 工具专用的参数，记录函数的调用栈信息

perf 是 Linux 系统上最全面最方便的一个性能检测工具。由 Linux 内核携带并且同步更新。

安装

apt-get install linux-tools-5.4.18-19-generic

语法

perf [--version] [--help] COMMAND [ARGS]

perf 一共 22 种子命令，常用的是以下 5 种

perf list：查看当前软硬件环境支持的性能事件

perf stat：分析指定程序的性能概况

perf top：实时显示系统/进程的性能统计信息

perf record：记录一段时间内系统/进程的性能事件

perf report：读取 perf record 生成的

perf.data 文件，并显示分析数据

perf stat

参数

-e <event>：指定性能事件（可以是多个，用,分隔列表）

-p <pid>：指定待分析进程的 pid（可以是多个，用,分隔列表）

-t <tid>;：指定待分析线程的 tid（可以是多个，用,分隔列表）

-a：从所有 CPU 收集系统数据

-d：打印更详细的信息，可重复 3 次

-d：L1 和 LLC data cache

-d -d：dTLB 和 iTLB events

-d -d -d：增加 prefetch events

-r <n>;：重复运行命令 n 次，打印平均值。n 设为 0 时无限循环打印

-c <cpu-list>：只统计指定 CPU 列表的数据，如：0,1,3 或 1-2

-A：与-a 选项联用，不要将 CPU 计数聚合

-I <N msecs>：每隔 N 毫秒打印一次计数器的变化，N 最小值为 100 毫秒

参数含义

task-clock：任务真正占用的处理器时间，单位为 ms。CPUs utilized = task-clock / time elapsed，CPU 的占用率，值高，说明程序的多数时间花费在 CPU 计算上而非 IO。context-switches：上下文的切换次数。

CPU-migrations：处理器迁移次数。Linux 为了维持多个处理器的负载均衡，在特定条件下会将某个任务从一个 CPU 迁移到另一个 CPU。

page-faults：缺页异常的次数。当应用程序请求的页面尚未建立、请求的页面不在内存中，或者请求的页面虽然在内存中，但物理地址和虚拟地址的映射关系尚未建立时，都会触发一次缺页异常。另外 TLB 不命中，页面访问权限不匹配等情况也会触发缺页异常。

cycles：消耗的处理器周期数。

instructions：执行了多少条指令。IPC 为平均每个 cpu cycle 执行了多少条 branches：遇到的分支指令数。

branch-misses：预测错误的分支指令数。

perf record

收集一段时间内的性能事件到文件 perf.data，随后需要用 perf report 命令分析

-e <event>：指定性能事件（可以是多个，用,分隔列表）
-p <pid>：指定待分析进程的 pid（可以是多个，用,分隔列表）
-t <tid>：指定待分析线程的 tid（可以是多个，用,分隔列表）
-u <uid>：指定收集的用户数据，uid 为名称或数字
-a：从所有 CPU 收集系统数据
-g：开启 call-graph (stack chain/backtrace) 记录(显示调用关系)
-C <cpu-list>：只统计指定 CPU 列表的数据，如：0,1,3 或 1-2
-r <RT priority>：perf 程序以 SCHED_FIFO 实时优先级 RT priority 运行这里填入的数值越大，进程优先级越高（即 nice 值越小）
-c <count>： 事件每发生 count 次采一次样
-F <n>：每秒采样 n 次
-o <output.data>：指定输出文件 output.data，默认输出到 perf.data
利用 perf 性能调优时需要回答 3 个问题：

1、程序的哪一部分耗用了最多的时间？（Which parts of the program take the most execution time?）程序占用时间最多的部分称为“热点”。对热点进行优化分析是最有价值的，有可能很小的改动，得到意想不到的效果。

2、软/硬件性能事件的数量就代表需要实际修正的问题吗？（Do the number of software or hardware events indicate an actual performance issue to be fixed?）要回答这个问题，我们需要对程序中的算法和数据结构有一定的了解或直觉。了解程序的类型，是 cpu 密集性、memory 密集性、io 密集性，然后在针对热点位置具体分析。

3、如何修正这些性能问题？（How can we fix the performance issue?）我们需要研究程序的算法和数据结构，找到其中不合理的地方，修改代码，然后观察效果。性能调优是一项艰巨的实验工作，如果修改后的程序运行速度低于基准速度，请不要感到惊讶。如果一个假设失败，则尝试另一个。继续尝试！

示例:shiyong perf 工具迅速定位是哪个进程哪段代码引起的 CPU 使用率暴涨

先使用 stress -c 8 命令模拟出 CPU 利用率暴涨的现象

```
%Cpu0  : 98.7 us,  1.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 98.7 us,  1.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 98.3 us,  1.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 99.3 us,  0.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 98.7 us,  1.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 98.7 us,  1.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 98.7 us,  1.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

然后使用 perf 工具采样

```
root@kylin-PC:/home/kylin# perf record -a -e cycles -o cycle.perf -g sleep 10
[ perf record: Woken up 85 times to write data ]
[ perf record: Captured and wrote 24.902 MB cycle.perf (320016 samples) ]
```

把采样的数据生成报告

```
root@kylin-PC:/home/kylin# perf report -i cycle.perf | more
# To display the perf.data header info, please use --header/--header-only option
s.
#
#
# Total Lost Samples: 0
#
# Samples: 319K of event 'cycles'
# Event count (approx.): 167592197622
#
# Children      Self  Command          Shared Object                          S
ymbol



# ........  ........  ...............  ....................................  .
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................
#
    37.83%    37.70%  stress           libc-2.31.so                           [
.] random
            |
            ---random

    31.32%    31.21%  stress           libc-2.31.so                           [
.] random_r
            |
            ---random_r

     7.08%     7.05%  stress           libc-2.31.so                           [
.] rand
            |
            ---rand

     5.67%     0.00%  stress           stress                                 [
```

可以清楚的看到，CPU 利用率大部分来自 stress 的调用，perf 甚至精确的告诉我们是消耗在 libc-2.31so 这个 shared Object 上

pstree

dstat

dstat 命令是一个用来替换 vmstat、iostat、netstat、nfsstat 和 ifstat 这些命令的工具，是一个全能系统信息统计工具

语法 dstat [-afv] [options..] [delay [count]]

常用选项

-c：显示 CPU 系统占用，用户占用，空闲，等待，中断，软件中断等信息。

-C：当有多个 CPU 时候，此参数可按需分别显示 cpu 状态，例：-C 0,1 是显示 cpu0 和 cpu1 的信息。

-d：显示磁盘读写数据大小。

-D hda,total：include hda and total。

-n：显示网络状态。

-N eth1,total：有多块网卡时，指定要显示的网卡。

-l：显示系统负载情况。

-m：显示内存使用情况。

-g：显示页面使用情况。

-p：显示进程状态。

-s：显示交换分区使用情况。

-S：类似 D/N。

-r：I/O 请求情况。

-y：系统状态。

--ipc：显示 ipc 消息队列，信号等信息。

--socket：用来显示 tcp udp 端口状态。

-a：此为默认选项，等同于-cdngy。

-v：等同于 -pmgdsc -D total。

--output 文件：此选项也比较有用，可以把状态信息以 csv 的格式重定向到指定的文件中，以便日后查看。例：dstat --output /root/dstat.csv & 此时让程序默默的在后台运行并把结果输出到/root/dstat.csv 文件中。

命令参数

delay 两次输出之间的时间间隔，默认是 1s

count 报告输出的次数，默认是没有限制，一直输出知道 ctrl+c

命令插件

虽然 anyone 可以自由的为 dstat 编写插件，但 dstat 附带大量的插件已经大大扩展其功能，下面是 dstat 附带常用插件的概述

-disk-util 显示某一时间磁盘的忙碌状况

-freespace 显示当前磁盘空间使用率

-proc-count 显示正在运行的程序数量

-top-bio 显示块 I/O 最大的进程

-top-cpu 显示 CPU 占用最大的进程

-top-io 显示正常 I/O 最大的进程

-top-mem 显示占用最多内存的进程

示例：监察 CPU 和磁盘情况

```
root@kylin-PC:/home/kylin# dstat -c -d
--total-cpu-usage-- -dsk/total-
usr sys idl wai stl| read  writ
  6   1  91   2   0| 177k  329k
  1   1  99   0   0|   0     0
  1   0  99   0   0|   0     0
  1   1  98   1   0|   0    92k
  1   1  99   0   0|   0     0
  1   0  99   0   0|   0    40k
```

uptime

望名生义，uptime 命令告诉你系统启动 up 了（运行了）多长时间，uptime 会在一行中显示下列信息：当前时间、系统运行了多久时间、当前登录的用户有多少，以及前 1、5 和 15 分钟系统的平均负载。

语法：uptime [options]

示例：

```
kylin@kylin-PC:~$ uptime
 09:06:39 up 16 min,  1 user,  load average: 0.51, 0.58, 0.66
```

dstat

execsnoop

sar

sar（System ActivityReporter）是系统活动情况报告的缩写。sar 工具将对系统当前的状态进行取样，然后通过计算数据和比例来表达系统的当前运行状态。它的特点是可以连续对系统取样，获得大量的取样数据；取样数据和分析的结果都可以存入文件，所需的负载很小。 sar 是目前 Linux 上最为全面的系统性能分析工具之一，可以从多方面对系统的活动进行报告，包括：文件的读写情况、系统调用的使用情况、磁盘 I/O、CPU 效率、内存使用状况、进程活动及 IPC 有关的活动等。为了提供不同的信息，sar 提供了丰富的选项、因此使用较为复杂。

sar 可以用来查看系统的网络收发情况,还有一个好处是,不仅可以观察网络收发的吞吐量(BPS,每秒收发的字节数),还可以观察网络收发的 PPS,即每秒收发的网络帧数

语法

sar [选项] [<时间间隔> [<次数>]]

常用选项：sar --help 显示主要选项和报告

```
root@kylin-PC:/home/kylin# sar --help
用法: sar [ 选项 ] [ <时间间隔> [ <次数> ] ]
主要选项和报告（报告名以方括号分隔）:
        -B      分页状况 [A_PAGE]
        -b      I/O 和传输速率信息状况 [A_IO]
        -d      块设备状况 [A_DISK]
        -F [ MOUNT ]
                文件系统统计信息 [A_FS]
        -H      巨大页面利用率 [A_HUGE]
        -I { <中断列表> | SUM | ALL }
                中断信息状况 [A_IRQ]
        -m { <关键字> [,...] | ALL }
                电源管理统计信息 [A_PWR_...]
                关键字:
                CPU     CPU 瞬时时钟频率
                FAN     风扇速度
\t\tFREQ\tCPU 平均时钟频率
                IN      输入电压
                TEMP    设备温度
\t\tUSB\t连接的 USB 设备
```

常用命令

默认监控: sar 1 1 // CPU 和 IOWAIT 统计状态

(1) sar -b 1 1 // IO 传送速率

(2) sar -B 1 1 // 页交换速率

(3) sar -c 1 1 // 进程创建的速率

(4) sar -d 1 1 // 块设备的活跃信息

(5) sar -n DEV 1 1 // 网路设备的状态信息

(6) sar -n SOCK 1 1 // SOCK 的使用情况

(7) sar -n ALL 1 1 // 所有的网络状态信息

(8) sar -P ALL 1 1 // 每颗 CPU 的使用状态信息和 IOWAIT 统计状态

(9) sar -q 1 1 // 队列的长度（等待运行的进程数）和负载的状态

(10) sar -r 1 1 // 内存和 swap 空间使用情况

(11) sar -R 1 1 // 内存的统计信息（内存页的分配和释放、系统每秒作为 BUFFER 使用内存页、每秒被 cache 到的内存页）

(12) sar -u 1 1 // CPU 的使用情况和 IOWAIT 信息（同默认监控）

(13) sar -v 1 1 // inode, file and other kernel tablesd 的状态信息

(14) sar -w 1 1 // 每秒上下文交换的数目

(15) sar -W 1 1 // SWAP 交换的统计信息(监控状态同 iostat 的 si so)

(16) sar -x 2906 1 1 // 显示指定进程(2906)的统计信息，信息包括：进程造成的错误、用户级和系统级用户 CPU 的占用情况、运行在哪颗 CPU 上

(17) sar -y 1 1 // TTY 设备的活动状态

(18) 将输出到文件(-o)和读取记录信息(-f)

示例一：查看 CPU 使用情况并保存

```
root@kylin-PC:/home/kylin# sar 1 3 -o test
Linux 5.4.18-19-generic (kylin-PC)      2021年03月25日   _x86_64_        (8 CPU)

09时45分12秒      CPU      %user      %nice    %system    %iowait    %steal      %idle
09时45分13秒      all       1.00       0.00       0.75       9.15       0.00      89.10
09时45分14秒      all       0.51       0.00       0.13      30.63       0.00      68.73
09时45分15秒      all      12.14       0.00       2.35      30.73       0.00      54.77
平均时间：      all      4.59      0.00      1.09     23.51      0.00     70.81
```

查看 sar -f 文件名

```
root@kylin-PC:/home/kylin# sar -f test
Linux 5.4.18-19-generic (kylin-PC)      2021年03月25日   _x86_64_        (8 CPU)

09时44分50秒      CPU      %user      %nice    %system    %iowait    %steal      %idle
09时44分51秒      all       1.63       0.00       0.50      11.89       0.00      85.98
09时44分52秒      all       0.75       0.00       0.88      21.48       0.00      76.88
```

参数含义

%user 用户空间的 CPU 使用

%nice 改变过优先级的进程的 CPU 使用率

%system 内核空间的 CPU 使用率

%iowait CPU 等待 IO 的百分比

%steal 虚拟机的虚拟机 CPU 使用的 CPU

%idle 空闲的 CPU

在以上的显示当中，主要看%iowait 和%idle，%iowait 过高表示存在 I/O 瓶颈，即磁盘 IO 无法满足业务需求，如果%idle 过低表示 CPU 使用率比较严重，需要结合内存使用等情况判断 CPU 是否瓶颈。

实例二：查看内存使用情况

```
root@kylin-PC:/home/kylin# sar -r 1 3
Linux 5.4.18-19-generic (kylin-PC)      2021年03月25日   _x86_64_        (8 CPU)

09时47分51秒  kbmemfree   kbavail kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
09时47分52秒   10306736  12458112   2575644     16.20    105516   2556056   9661220     27.62   3441080   1448500     39620
09时47分53秒   10315240  12449044   2580428     16.23    105524   2542692   9661220     27.62   3418800   1456844     26040
09时47分54秒   10230584  12443776   2590096     16.29    106616   2613792   9658976     27.61   3487948   1474444    104612
平均时间：  10284187  12450311   2582056     16.24    105885   2570847   9660472     27.62   3449276   1459929     56757
```

参数含义

kbmemfree 空闲的物理内存大小

kbmemused 使用中的物理内存大小

%memused 物理内存使用率

kbbuffers　内核中作为缓冲区使用的物理内存大小，kbbuffers 和 kbcached:这两个值就是 free 命令中的 buffer 和 cache.
kbcached　缓存的文件大小
kbcommit　保证当前系统正常运行所需要的最小内存，即为了确保内存不溢出而需要的最少内存（物理内存+Swap 分区）
commit　这个值是 kbcommit 与内存总量（物理内存+swap 分区）的一个百分比的值
示例三：查看系统 Swap 分区、查看 IO 和传递速率

```
root@kylin-PC:/home/kylin# sar -b 1 3
Linux 5.4.18-19-generic (kylin-PC)      2021年03月25日   _x86_64_        (8 CPU)

09时54分53秒       tps      rtps      wtps      dtps   bread/s   bwrtn/s   bdscd/s
09时54分54秒      0.00      0.00      0.00      0.00      0.00      0.00      0.00
09时54分55秒      2.00      0.00      2.00      0.00      0.00    208.00      0.00
09时54分56秒      5.00      0.00      5.00      0.00      0.00    200.00      0.00
平均时间：      2.33      0.00      2.33      0.00      0.00    136.00      0.00


root@kylin-PC:/home/kylin# sar -W 1 3
Linux 5.4.18-19-generic (kylin-PC)      2021年03月25日   _x86_64_        (8 CPU)

09时54分34秒   pswpin/s pswpout/s
09时54分35秒      0.00      0.00
09时54分36秒      0.00      0.00
09时54分37秒      0.00      0.00
平均时间：      0.00      0.00
```

pswpin/s　每秒从交换分区到系统的交换页面（swap page）数量

pswpott/s　每秒从系统交换到 swap 的交换页面（swap page）的数量

tps　磁盘每秒钟的 IO 总数，等于 iostat 中的 tps
rtps　每秒钟从磁盘读取的 IO 总数
wtps　每秒钟从写入到磁盘的 IO 总数
bread/s　每秒钟从磁盘读取的块总数
bwrtn/s　每秒钟此写入到磁盘的块总数

iostat

iostat 用于输出 CPU 和磁盘 I/O 相关的统计信息.
语法：
iostat [选项] [<时间间隔> [<次数>]]
常见选项：
-c：　显示 CPU 使用情况
-d：　显示磁盘使用情况
-N：　显示磁盘阵列(LVM)　信息

-n： 显示 NFS 使用情况
-k： 以 KB 为单位显示
-m： 以 M 为单位显示
-t： 报告每秒向终端读取和写入的字符数和 CPU 的信息
-V： 显示版本信息
-x： 显示详细信息
-p：[磁盘] 显示磁盘和分区的情况
示例：详细统计 Io 信息

```
root@kylin-PC:/home/kylin# iostat -x -k -d 1 1
Linux 5.4.18-19-generic (kylin-PC)      2021年03月25日   _x86_64_        (8 CPU)

Device            r/s     rkB/s   rrqm/s  %rrqm r_await rareq-sz     w/s     wkB/s   wrqm/s
  %wrqm w_await wareq-sz     d/s     dkB/s   drqm/s  %drqm d_await dareq-sz  aqu-sz  %util
nvme0n1          0.01      0.05     0.00   0.00    0.39     6.07    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   0.00
nvme1n1          0.03      0.53     0.00   0.00    0.46    19.96    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   0.00
sda              7.52    416.98     3.19  29.83   24.36    55.48   14.67    335.67    15.49
  51.35   26.10    22.88     0.00      0.00     0.00   0.00    0.00     0.00    0.54   3.04
```

参数含义
rrqm/s 每秒对该设备的读请求被合并次数，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)
avgqu-sz 平均等待处理的 IO 请求队列长度
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)
svctm 平均每次 IO 请求的处理时间(毫秒为单位)
%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

strace

strace 是一个可用于诊断、调试和教学的 Linux 用户空间跟踪器。我们用它来监控用户空间进程和内核的交互，比如系统调用、信号传递、进程状态变更等。
语法：
strace [-dffhiqrtttTvxx] [-acolumn] [-eexpr]
常用选项
-tt 在每行输出的前面，显示毫秒级别的时间
-T 显示每次系统调用所花费的时间
-v 对于某些相关调用，把完整的环境变量，文件 stat 结构等打出来。
-f 跟踪目标进程，以及目标进程创建的所有子进程
-e 控制要跟踪的事件和跟踪行为,比如指定要跟踪的系统调用名称
-o 把 strace 的输出单独写到指定的文件
-s 当系统调用的某个参数是字符串时，最多输出指定长度的内容，默认是 32 个字节
-p 指定要跟踪的进程 pid，要同时跟踪多个 pid，重复多次-p 选项即可。

trace 有两种运行模式。

一种是通过它启动要跟踪的进程。用法很简单，在原本的命令前加上 strace 即可。比如我们要跟踪 "ls -lh /var/log/messages" 这个命令的执行，可以这样：

strace ls -lh /var/log/messages

另外一种运行模式，是跟踪已经在运行的进程，在不中断进程执行的情况下，理解它在干嘛。 这种情况，给 strace 传递个-p pid 选项即可。
追踪 PID 为 10385 的进程

```
root@kylin-PC:/home/kylin# strace -p 10385
strace: Process 10385 attached
restart_syscall(<... resuming interrupted read ...>) = 0
recvmsg(4, {msg_namelen=0}, 0)          = -1 EAGAIN (资源暂时不可用)
recvmsg(4, {msg_namelen=0}, 0)          = -1 EAGAIN (资源暂时不可用)
poll([{fd=4, events=POLLIN}, {fd=6, events=POLLIN}, {fd=7, events=POLLIN}], 3, 0) = 0 (Time
out)
```

示例一：追踪 10403 进程，看其启动时都有哪些子进程

```
root@kylin-PC:/home/kylin# strace -tt -T -f -e trace=file -o /home/kylin/strace.log -s 1024 -p 10403
strace: Process 10403 attached with 18 threads
strace: Process 16866 attached
strace: Process 16870 attached
strace: Process 16871 attached
strace: Process 16877 attached
```

free

free 指令会显示内存的使用情况，包括实体内存，虚拟的交换文件内存，共享内存区段，以及系统核心使用的缓冲区等。
语法
free [-bkmotV][-s <间隔秒数>]
常用选项
-b 以 Byte 为单位显示内存使用情况。
-k 以 KB 为单位显示内存使用情况。
-m 以 MB 为单位显示内存使用情况。
-h 以合适的单位显示内存使用情况，最大为三位数，自动计算对应的单位值。
-o 不显示缓冲区调节列。
-s<间隔秒数> 持续观察内存使用状况。
-t 显示内存总和列。
示例：每 3s 以总和的形式查询内存的使用信息

```
kylin@kylin-PC:~$ free -t -s 3
              总计         已用         空闲       共享     缓冲/缓存     可用
内存：      15899676     2412996    10898544      311472      2588136    12829268
交换：      19078140           0    19078140
总量：      34977816     2412996    29976684

              总计         已用         空闲       共享     缓冲/缓存     可用
内存：      15899676     2415996    10895000      312008      2588680    12825732
交换：      19078140           0    19078140
总量：      34977816     2415996    29973140
```

/proc/meminfo

/proc/meminfo 是了解 Linux 系统内存使用状况的主要接口，我们最常用的”free”、”vmstat”等命令就是通过它获取数据的 ，/proc/meminfo 所包含的信息比”free”等命令要丰富得多，具体参数及含义如下：

MemTotal: 29584 kB //物理内存
MemFree: 968 kB //剩余物理内存
Buffers: 28 kB //用来给文件做缓冲的大小
Cached: 4644 kB //被高速缓冲存储器（cache memory）用的内存的大小（等于 diskcache minus SwapCache ）.
SwapCached: 0 kB //缓存的大小，Android 很少使用 swap 的，经常为 0。被高速缓冲存储器（cache memory）用来交换空间的大小,用来在需要的时候很快的被替换而不需要再次打开 I/O 端口。
Active: 14860 kB //在活跃使用中的缓冲或高速缓冲存储器页面文件的大小，除非非常必要，否则不会被移作他用。
Inactive: 1908 kB //在不经常使用中的缓冲或高速缓冲存储器页面文件的大小，可能被用于其他途径。
HighTotal: 0 kB
HighFree: 0 kB //该区域不是直接映射到内核空间。内核必须使用不同的手法使用该段内存。
LowTotal: 29584 kB
LowFree: 968 kB
SwapTotal: 0 kB //交换空间的总大小
SwapFree: 0 kB /未被使用交换空间的大小
Dirty: 0 kB //等待被写回到磁盘的内存大小
Writeback: 0 kB //正在被写回到磁盘的内存大小。
Mapped: 12840 kB //设备和文件等映射的大小。
Slab: 2052 kB //内核数据结构缓存的大小，可以减少申请和释放内存带来的消耗。
CommitLimit: 29584 kB //当前系统可以申请的总内存
Committed_AS: 13148 kB //当前已经申请的内存，记住是申请
PageTables: 108 kB //管理内存分页的索引表的大小
VmallocTotal: 483328 kB //虚拟内存大小
VmallocUsed: 552 kB //已经被使用的虚拟内存大小
VmallocChunk: 482776 kB

ps

ps （英文全拼：process status）命令用于显示当前进程的状态，类似于 windows 的任务管理器。
语法
ps [options] [--help]
常用参数：
-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
示例：显示所有用户进程名中包含 su 的进程

```
kylin@kylin-PC:~$ ps -aux | grep su
root         871  0.7  0.0  14040  9300 ?        Ss   14:30   0:20 /sbin/wpa_sup
plicant -u -s -O /run/wpa_supplicant
root         886  0.0  0.0   2668   564 ?        Ss   14:30   0:00 /bin/sh -c /o
pt/kingsoft/wtool/wpsupdateserver
root         887  0.0  0.0  56340 11460 ?        S    14:30   0:00 /opt/kingsoft
/wtool/wpsupdateserver
root        5535  0.0  0.0  14568  5364 pts/0    S    14:36   0:00 su
kylin       6390  0.0  0.5 388620 91584 ?        Sl   14:40   0:00 /usr/lib/wech
at/微信 --type=gpu-process --no-sandbox --ignore-gpu-blacklist --supports-dual-g
pus=false --gpu-driver-bug-workarounds=1,7,24,63,76 --disable-gl-extensions=GL_K
HR_blend_equation_advanced GL_KHR_blend_equation_advanced_coherent --gpu-vendor-
id=0x8086 --gpu-device-id=0x8a52 --gpu-driver-vendor --gpu-driver-version --gpu-
driver-date --service-request-channel-token=94E5156256AEFAC6134A8107B6FC7DBB
root        9179  0.0  0.0  14568  5268 pts/1    S    14:59   0:00 su
kylin      10548  0.0  0.0   9524   724 pts/2    S+   15:14   0:00 grep --color=
auto su
```

参数含义：

USER: 行程拥有者

PID: pid

%CPU: 占用的 CPU 使用率

%MEM: 占用的记忆体使用率

VSZ: 占用的虚拟记忆体大小

RSS: 占用的记忆体大小

TTY: 终端的次要装置号码 (minor device number of tty)

STAT: 该行程的状态:

- D: 无法中断的休眠状态 (通常 IO 的进程)
- R: 正在执行中
- S: 静止状态
- T: 暂停执行
- Z: 不存在但暂时无法消除
- W: 没有足够的记忆体分页可分配
- <: 高优先序的行程
- N: 低优先序的行程
- L: 有记忆体分页分配并锁在记忆体内 (实时系统或 I/O)

START: 行程开始时间

TIME: 执行的时间

COMMAND:所执行的指令

cachestat

cachestat、cachetop、pcstat 等是 bcc 包中应用，使用它们前，我们首先要安装 bcc 软件包，安装命令如下

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 4052245BD4284CDD
echo "deb https://repo.iovisor.org/apt/xenial xenial main" | sudo tee /etc/apt/sources.list.d/iovisor.list
sudo apt-get update
sudo apt-get install -y bcc-tools libbcc-examples linux-headers-$(uname -r)
```

slabtop

该程序可以用来查看 slab 使用统计信息。该程序是从/proc/slabinfo 来获取该信息的，并且以更加简明的方式展现出来。它的输出风格和 top 命令很类似，只不过输出信息是一些 slab 使用的统计情况。

常用参数

-d　每 n 秒更新一次显示的信息，默认是每 3 秒

-s　指定排序标准进行排序

-o　显示一次后退出

-V　显示版本

-help　　显示帮助信息

示例：每 5s 刷新 slab 缓冲区信息

```
Active / Total Objects (% used)    : 1800129 / 1830299 (98.4%)
Active / Total Slabs (% used)      : 46117 / 46117 (100.0%)
Active / Total Caches (% used)     : 106 / 153 (69.3%)
Active / Total Size (% used)       : 447652.83K / 457247.98K (97.9%)
Minimum / Average / Maximum Object : 0.01K / 0.25K / 16.75K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
361530 359390  99%    0.10K   9270       39     37080K buffer_head
213720 203533  95%    0.20K   5480       39     43840K dentry
154840 154388  99%    0.07K   2765       56     11060K Acpi-Operand
122026 118659  97%    0.23K   3589       34     28712K vm_area_struct
111360 108231  97%    0.06K   1740       64      6960K anon_vma_chain
104190 104190 100%    0.13K   3473       30     13892K kernfs_node_cache
101808  99773  98%    1.13K   3636       28    116352K ext4_inode_cache
 62744  61418  97%    0.09K   1364       46      5456K anon_vma
 56320  56320 100%    0.03K    440      128      1760K kmalloc-32
 51712  51144  98%    0.06K    808       64      3232K kmalloc-rcl-64
 49024  46254  94%    0.25K   1532       32     12256K filp
 41922  41922 100%    0.04K    411      102      1644K ext4_extent_status
 40320  40213  99%    0.06K    630       64      2520K kmalloc-64
 38472  36782  95%    0.57K   1374       28     21984K radix_tree_node
 36500  36276  99%    0.62K   1460       25     23360K inode_cache
 23552  23552 100%    0.02K     92      256       368K kmalloc-16
```

valgrind

Valgrind 是一个 GPL 的软件，用于 Linux（For x86, amd64 and ppc32）程序的内存调试和代码剖析。你可以在它的环境中运行你的程序来监视内存的使用情况，比如 C 语言中的 malloc 和 free 或者 C++中的 new 和 delete。使用 Valgrind 的工具包，你可以自动的检测许多内存管理和线程的 bug，避免花费太多的时间在 bug 寻找上，使得你的程序更加稳固。

语法：

valgrind [options] prog-and-args

选择工具

tool-selection option, with default in [ ]:

工具选择选项，默认值在[]中：

–tool= use the Valgrind tool named [memcheck]

name 取值如下：

1、memcheck：检查程序中的内存问题，如泄漏、越界、非法指针等。

2、callgrind：检测程序代码覆盖，以及分析程序性能。

3、cachegrind：分析 CPU 的 cache 命中率、丢失率，用于进行代码优化。
4、helgrind：用于检查多线程程序的竞态条件。
5、massif：堆栈分析器，指示程序中使用了多少堆内存等信息。
6、lackey：Lackey 是小型工具，很少用到
7、nulgrind：Nulgrind 只是为开发者展示如何创建一个工具

df

df（disk free） 命令用于显示目前在 Linux 系统上的文件系统磁盘使用情况统计。
语法
df [options]
常用参数
-s：对每个 Names 参数只给出占用的数据块总数。
-a：递归地显示指定目录中各文件及子目录中各文件占用的数据块数。若既不指定-s，也不指定-a，则只显示 Names 中的每一个目录及其中的各子目录所占的磁盘块数。
-k：以 1024 字节为单位列出磁盘空间使用情况。
-x：跳过在不同文件系统上的目录不予统计。
-l：计算所有的文件大小，对硬链接文件则计算多次。
-i：显示 inode 信息而非块使用量。
-h：以容易理解的格式印出文件系统大小，例如 136KB、2 4MB、21GB。
-P：使用 POSIX 输出格式。
-T：显示文件系统类型。
示例：查看磁盘空间信息

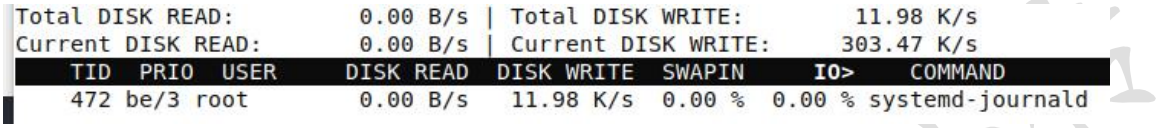
```
root@kylin-PC:/home/kylin# df -T
文件系统       类型          1K-块      已用      可用 已用% 挂载点
udev           devtmpfs    7914960         0   7914960    0% /dev
tmpfs          tmpfs       1589968      3084   1586884    1% /run
/dev/sda3      ext4      102687672  41577328  55851080   43% /
tmpfs          tmpfs       7949836    116720   7833116    2% /dev/shm
tmpfs          tmpfs          5120         4      5116    1% /run/lock
tmpfs          tmpfs       7949836         0   7949836    0% /sys/fs/cgroup
/dev/sda2      ext4         999320    111824    818684   13% /boot
/dev/sda5      ext4      733630648  70683760 625610776   11% /data
/dev/sda1      vfat         523248       280    522968    1% /boot/efi
tmpfs          tmpfs       1589964        48   1589916    1% /run/user/1000
```

iotop

iotop 命令是一个用来监视磁盘 I/O 使用状况的 top 类工具，iotop 具有与 top 相似的 UI，其中包括 pid、user、I/O、进程等相关信息等；
语法
iotop -[选项]
常用选项
--version：//显示程序的版本号并退出
-h, --help：//显示此帮助消息并退出
-o, --only：//仅显示实际执行 I / O 的进程或线程，只显示在划硬盘的程序

-b, --batch：//非交互模式，批量处理 用来记录日志的
-n NUM, --iter=NUM：//设定循环几次
-d SEC, --delay=SEC：//设定显示时间间隔[秒]
-p PID, --pid=PID：//要监控的进程/线程[全部]
-u USER, --user=USER：//用户监控[全部]
-P, --processes：//只显示进程，而不是所有线程
-a, --accumulated：//显示累积的 I / O 而不是带宽
-k, --kilobytes：//使用千字节而不是人性化的单位
-t, --time：//在每一行上添加一个时间戳（暗示--batch）
-q, --quiet：//抑制一些标题行（暗示--batch）

示例：查看 PID 为 472 的 IO 使用情况



参数含义
READ 读速率
WRITR 写速率
tid：线程 id，按 p 可转换进程 pid
PRIO：优先级
DISK READ：磁盘读取速率
DISK WRITE：磁盘写取速率
SWAPIN：swap 交换百分比
IO>：IO 等待所占用百分比
COMMAND：线程/进程详细信息

lsof

lsof 命令用于查看你进程开打的文件，打开文件的进程，进程打开的端口(TCP、UDP)。找回/恢复删除的文件。是十分方便的系统监视工具，因为 lsof 命令需要访问核心内存和各种文件，所以需要 root 用户执行。

语法
lsof【选项】
常用选项
-a：列出打开文件存在的进程；
-c<进程名>：列出指定进程所打开的文件；
-g：列出 GID 号进程详情；
-d<文件号>：列出占用该文件号的进程；
+d<目录>：列出目录下被打开的文件；
+D<目录>：递归列出目录下被打开的文件；
-n<目录>：列出使用 NFS 的文件；
-i<条件>：列出符合条件的进程。（4、6、协议、:端口、 @ip ）
-p<进程号>：列出指定进程号所打开的文件；
-u：列出 UID 号进程详情；

示例：查看 PID 为 6557 的进程打开的文件

```
root@kylin-PC:/home/kylin# lsof -p 6557
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND  PID  USER   FD      TYPE             DEVICE  SIZE/OFF    NODE NAME
qqmusic 6557 kylin  cwd       DIR               8,3      4096 4718594 /home/kylin
qqmusic 6557 kylin  rtd       DIR               8,3      4096       2 /
qqmusic 6557 kylin  txt       REG               8,3 116373960 3539241 /opt/QQmusic/qqmusic
qqmusic 6557 kylin  DEL       REG              0,25               190 /dev/shm/.org.chromium.Chromium.N4czV3
qqmusic 6557 kylin  DEL       REG              0,25               166 /dev/shm/.org.chromium.Chromium.uV9i52
qqmusic 6557 kylin  DEL       REG              0,25               231 /dev/shm/.org.chromium.Chromium.eCN5L3
```

参数详解

COMMAND：进程的名称

PID：进程标识符

USER：进程所有者

FD：文件描述符，应用程序通过文件描述符识别该文件。如 cwd、txt 等

TYPE：文件类型，如 DIR、REG 等

DEVICE：指定磁盘的名称

SIZE：文件的大小

NODE：索引节点（文件在磁盘上的标识）

NAME：打开文件的确切名称

blktrace

blktrace 能记录 I/O 所经历的各个步骤，从中可以分析是 IO Scheduler 慢还是硬件响应慢，以及各个时间段所用时间。

原理：

一个 I/O 请求进入 block layer 之后，可能会经历下面的过程：

Remap: 可能被 DM(Device Mapper)或 MD(Multiple Device, Software RAID) remap 到其它设备

Split: 可能会因为 I/O 请求与扇区边界未对齐、或者 size 太大而被分拆(split)成多个物理 I/O

Merge: 可能会因为与其它 I/O 请求的物理位置相邻而合并(merge)成一个 I/O

被 IO Scheduler 依照调度策略发送给 driver

被 driver 提交给硬件，经过 HBA、电缆（光纤、网线等）、交换机（SAN 或网络）、最后到达存储设备，设备完成 IO 请求之后再把结果发回。

语法：blktrace [参数]

常用选项

-A hex-mask 设置过滤信息 mask 成十六进制 mask

-a mask 添加 mask 到当前的过滤器

-b size　指定缓存大小 for 提取的结果，默认为 512KB

-d dev　添加一个设备追踪

-k　杀掉正在运行的追踪

-n num-sub 指定缓冲池大小，默认为 4 个子缓冲区

-o file　指定输出文件的名字

-r rel-path　指定的 debugfs 挂载点

-w seconds　设置运行的时间

示例一：使用 blktrace 跟踪计算机上的 I/O

1、使用 blktrace 跟踪计算机上的

```
blktrace -d /dev/sda -o -|blkparse -i -
```

2、同时，在另一个控制台上，启动以下命令以生成一些 I/O 以进行测试

```
dd if=/dev/zero of=/mnt/test1 bs=1M count=1
```

从 blktrace 控制台获得输出，如图所示：

```
Total (8,0):
 Reads Queued:           0,        0KiB  Writes Queued:         121,     2936KiB
 Read Dispatches:       24,        0KiB  Write Dispatches:       43,     2936KiB
 Reads Requeued:         0               Writes Requeued:         0
 Reads Completed:       24,        0KiB  Writes Completed:       55,     2936KiB
 Read Merges:            0,        0KiB  Write Merges:           80,      324KiB
 IO unplugs:            26               Timer unplugs:           0

Throughput (R/W): 0KiB/s / 277KiB/s
Events (8,0): 1355 entries
Skips: 0 forward (0 -   0.0%)
```

示例二：监控 30s 内的 IO

blktrace -w 30 -d /dev/sda -o io-debugging

另一个控制台上启动

dd if=/dev/sda of=/dev/null bs=1M count=10 iflag=direct

```
root@kylin-PC:/home/kylin# blktrace -w 30 -d /dev/sda -o io-debugging
=== sda ===
  CPU  0:                 2441 events,      115 KiB data
  CPU  1:                 1042 events,       49 KiB data
  CPU  2:                 6426 events,      302 KiB data
  CPU  3:                  106 events,        5 KiB data
  CPU  4:                 1102 events,       52 KiB data
  CPU  5:                 4983 events,      234 KiB data
  CPU  6:                 3858 events,      181 KiB data
  CPU  7:                 7340 events,      345 KiB data
  Total:                 27298 events (dropped 0),     1280 KiB data
```

示例三：blkparse 命令的摘录

```
  8,0    7        0    38.585766485     0  m   N bfq rq_pos_tree_lookup 154896800: returning 0
  8,0    7     1592    38.585766623 76102  D   W 118247120 + 8 [kworker/u16:2]
  8,0    7        0    38.585767208     0  m   N bfq dispatch requests: 1 busy queues
  8,0    7        0    38.585767357     0  m   N bfq286A select_queue: already in-service queue
  8,0    7        0    38.585767579     0  m   N bfq286A may_budget_timeout: wait_request 0 left 1 timeout 0
  8,0    7        0    38.585767745     0  m   N bfq286A select_queue: returned this queue
  8,0    7        0    38.585767926     0  m   N bfq286A bfqq_served 24 secs
  8,0    7     1593    38.585768215 76102  D   W 154896800 + 8 [kworker/u16:2]
  8,0    7        0    38.585768869     0  m   N bfq dispatch requests: 1 busy queues
```

第 1 列显示设备主，次元组

第 2 列提供有关 CPU 的信息，并继续显示发出 IO 进程的进程的顺序，时间戳和 PID。

第 6 列显示事件类型，例如，Q 表示由请求队列代码处理的 IO。

第 7 列是 R（代表读取），W（代表写入），D（代表区块），B（代表屏障操作），

最后一列是区块编号，后跟 + 号是请求的区块编号。[] 括号之间的最后一个字段是发出请求的流程的流程名称。

ifconfig

ifconfig 工具不仅可以被用来简单地获取网络接口配置信息，还可以修改这些配置。用 ifconfig 命令配置的网卡信息，在网卡重启后机器重启后，配置就不存在。要想将上述的配置信息永远的存的电脑里，那就要修改网卡的配置文件了。

语法：

ifconfig [-a] [-v] [-s] <interface> [[<AF>] <address>]

常用选项

-a 显示全部接口信息。

-s 显示摘要信息（类似于 netstat -i）。

<interface> address 为网卡设置 IPv4 地址。

<interface> add <address>给指定网卡配置 IPv6 地址。

<interface> del <address>删除指定网卡的 IPv6 地址。

<interface> netmask <address>设置网卡的子网掩码。掩码可以是有前缀 0x 的 32 位十六进制数，也可以是用点分开的 4 个十进制数。如果不打算将网络分成子网，可以不管这一选项；如果要使用子网，那么请记住，网络中每一个系统必须有相同子网掩码。

<interface> dstaddr <address>设定一个远端地址，建立点对点通信。

<interface> tunnel <address>建立隧道。

<interface> hw <address>设置硬件地址。

<interface> mtu <NN>设置最大传输单元。

<interface> [-]arp 设置指定网卡是否支持 ARP 协议。-表示不支持 arp。

<interface> multicast 为网卡设置组播标志。

<interface> [-]promisc 设置是否支持网卡的 promiscuous 模式，如果选择此参数，网卡将接收网络中发给它所有的数据包。-表示关闭混杂模式。

<interface> txqueuelen <NN>为网卡设置传输列队的长度。

<interface> up 启动指定网卡。

<interface> down 关闭指定网卡。该参数可以有效地阻止通过指定接口的 IP 信息流，如果想永久地关闭一个接口，我们还需要从核心路由表中将该接口的路由信息全部删除。

网卡字段说明

```
root@kylin-PC:/home/kylin# ifconfig enp3s0
enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.30.110.171  netmask 255.255.255.0  broadcast 172.30.110.255
        inet6 fe80::f815:d9af:f000:3734  prefixlen 64  scopeid 0x20<link>
        ether b4:a9:fc:da:55:42  txqueuelen 1000  (以太网)
        RX packets 1969295  bytes 2271028452 (2.2 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1259481  bytes 148360589 (148.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

第一行中

P：表示“接口已启用”。

BROADCAST ：表示“主机支持广播”。

RUNNING：表示“接口在工作中”。

MULTICAST：表示“主机支持多播”。

MTU:1500（最大传输单元）：1500 字节

第二行中

inet ：网卡的 IP 地址。

netmask ：网络掩码。

broadcast ：广播地址

第三行为网卡的 IPV6 地址

第四行 连接类型：Ethernet（以太网）HWaddr（硬件 mac 地址） txqueuelen（网卡设置的传输队列长度）

其他

RX packets 接收时，正确的数据包数。

RX bytes 接收的数据量。
RX errors 接收时，产生错误的数据包数。
RX dropped 接收时，丢弃的数据包数。
RX overruns 接收时，由于速度过快而丢失的数据包数。
RX frame 接收时，发生 frame 错误而丢失的数据包数。
TX packets 发送时，正确的数据包数。
TX bytes 发送的数据量。
TX errors 发送时，产生错误的数据包数。
TX dropped 发送时，丢弃的数据包数。
TX overruns 发送时，由于速度过快而丢失的数据包数。
TX carrier 发送时，发生 carrier 错误而丢失的数据包数。
collisions 冲突信息包的数目。

示例一：显示简要的网卡信息

```
root@kylin-PC:/home/kylin# ifconfig -s
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
enp3s0    1500  1970910      0      0 0       1260843      0      0      0 BMRU
lo       65536    83693      0      0 0         83693      0      0      0 LRU
vmnet1    1500        0      0      0 0           265      0      0      0 BMRU
vmnet8    1500        0      0      0 0           233      0      0      0 BMRU
wlp0s20f  1500   556671      0      7 0        722920      0      0      0 BMRU
```

ethtool

ethtool 是用于查询及设置网卡参数的命令。

语法：

ethtool [-a | -c | -g | -i | -d | -k | -r | -S |] ethX

常见选项

-a 查看网卡中接收模块 RX、发送模块 TX 和 Autonegotiate 模块的状态：启动 on 或 停用 off。

-A 修改网卡中 接收模块 RX、发送模块 TX 和 Autonegotiate 模块的状态：启动 on 或 停用 off。

-c display the Coalesce(聚合、联合) information of the specified ethernet card.聚合网口信息，使看起来更有规律。

-C Change the Coalesce setting of the specified ethernet card.修改网卡聚合信息。

-gDisplay the rx/tx ring parameter information of the specified ethernet card. 显示网卡的接收/发送环形参数。

-G Change the rx/tx ring setting of the specified ethernet card. 修改网卡的接收/发送环形参数。

-i 显示网卡驱动的信息，如驱动的名称、版本等。

-d 显示 register dump 信息，部分网卡驱动不支持该选项。

-e 显示 EEPROM dump 信息，部分网卡驱动不支持该选项。

-E 修改网卡 EEPROM byte.

-k 显示网卡 Offload 参数的状态：on 或 off，包括 rx-checksumming、tx-checksumming 等。

-K 修改网卡 Offload 参数的状态

-p 用于区别不同 ethX 对应网卡的物理位置，常用的方法是使网卡 port 上的 led 不断的闪；N 指示了网卡闪的持续时间，以秒为单位。

-r 如果 auto-negotiation 模块的状态为 on，则 restarts auto-negotiation.

-s 修改网卡的部分配置，包括网卡速度、单工/全双工模式、mac 地址等。加上-s 选项修改的内容才会生效

-S 显示 NIC- and driver-specific 的统计参数，如网卡接收/发送的字节数、接收/发送的广播包个数等。

-t 让网卡执行自我检测，有两种模式：offline or online.

示例一：让 eth2 标识的网卡的灯点亮 20s

ethtool --identify eth2 20

ping

在网络中 ping 是一个十分强大的 TCP/IP 工具。它的作用主要为：

（1）用来检测网络的连通情况和分析网络速度；

（2）根据域名得到服务器 IP；

（3）根据 ping 返回的 TTL 值来判断对方所使用的操作系统及数据包经过路由器数量。

我们通常会用它来直接 ping IP 地址，来测试网络的连通情况。

语法

ping [参数] [主机名或 IP 地址]

常用参数

-d 使用 Socket 的 SO_DEBUG 功能。

-f 极限检测。大量且快速地送网络封包给一台机器，看它的回应。

-n 只输出数值。

-q 不显示任何传送封包的信息，只显示最后的结果。

-r 忽略普通的 Routing Table，直接将数据包送到远端主机上。通常是查看本机的网络接口是否有问题。

-R 记录路由过程。

-v 详细显示指令的执行过程。

<ip>-c 数目：在发送指定数目的包后停止。

-i 秒数：设定间隔几秒送一个网络封包给一台机器，预设值是一秒送一次。

-I 网络界面：使用指定的网络界面送出数据包。

-l 前置载入：设置在送出要求信息之前，先行发出的数据包。

-p 范本样式：设置填满数据包的范本样式。

-s 字节数：指定发送的数据字节数，预设值是 56，加上 8 字节的 ICMP 头，一共是 64ICMP 数据字节。

-t 存活数值：设置存活数值 TTL 的大小。

示例一：向 www.qq.com 发送 10 个数据包

```
root@kylin-PC:/home/kylin# ping www.qq.com -c 10
PING ins-r23tsuuf.ias.tencent-cloud.net (221.198.70.47) 56(84) bytes of data.
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=1 ttl=51 time=3.96 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=2 ttl=51 time=4.02 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=3 ttl=51 time=3.74 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=4 ttl=51 time=16.0 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=5 ttl=51 time=3.89 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=6 ttl=51 time=3.77 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=7 ttl=51 time=4.44 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=8 ttl=51 time=12.9 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=9 ttl=51 time=4.97 ms
64 bytes from www47.asd.tj.cn (221.198.70.47): icmp_seq=10 ttl=51 time=3.77 ms

--- ins-r23tsuuf.ias.tencent-cloud.net ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 3.735/6.144/15.952/4.221 ms
```

结果显示：发送 10 次 56bytes 大小的包，10 次都成功接收，10 次数据返回的总时间为 9013ms，最快 3.735ms，平均 6.144ms 最慢 15.952ms 平均偏差 4.221ms

示例二：判断局域网某主机是否在线

```
root@kylin-PC:/home/kylin# arping  172.16.42.130 -c 4
ARPING 172.16.42.130 from 172.16.42.1 vmnet8
Sent 4 probes (4 broadcast(s))
Received 0 response(s)
```

该主机在线，则返回该主机的 Mac 地址（上图主机不在线），此命令也可以用来盘对 IP 是否被占用。

示例三：测试目标主机承压能力

以最快的速度，使用最大的包 ping

```
root@kylin-PC:/home/kylin# ping 172.16.130 -f -s 65507
PING 172.16.130 (172.16.0.130) 65507(65535) bytes of data.
...................^C
--- 172.16.130 ping statistics ---
18 packets transmitted, 0 received, 100% packet loss, time 271ms
```

注意：此用法非常危险，65535（包头+内容）*100 个包每秒=6.25MB，每秒发送 6.25MB 的数据，相当于 50Mbps 的带宽，完全可能导致目标主机拒绝服务。请勿用于非法用途，造成不良后果自负。

tcpdump

默认启动

tcpdump

普通情况下，直接启动 tcpdump 将监视第一个网络接口上所有流过的数据包。

监视指定网络接口的数据包

tcpdump -i eth1

如果不指定网卡，默认 tcpdump 只会监视第一个网络接口，一般是 eth0，下面的例子没有指定网

络接口。

监视指定主机的数据包

打印所有进入或离开 sundown 的数据包.

tcpdump host sundown

也可以指定 ip,例如截获所有 210.27.48.1 的主机收到的和发出的所有的数据包

tcpdump host 210.27.48.1

打印 helios 与 hot 或者与 ace 之间通信的数据包

tcpdump host helios and \(hot or ace \)

截获主机 210.27.48.1 和主机 210.27.48.2 或 210.27.48.3 的通信

tcpdump host 210.27.48.1 and \ (210.27.48.2 or 210.27.48.3 \)

打印 ace 与任何其他主机之间通信的 IP 数据包，但不包括与 helios 之间的数据包.

tcpdump ip host ace and not helios

如果想要获取主机 210.27.48.1 除了和主机 210.27.48.2 之外所有主机通信的 ip 包，使用命令：

tcpdump ip host 210.27.48.1 and ! 210.27.48.2

截获主机 hostname 发送的所有数据

tcpdump -i eth0 src host hostname

监视所有送到主机 hostname 的数据包

tcpdump -i eth0 dst host hostname

tcpdump 还可以配合系统的 wireshark 一起使用可以更直观的看出包的类型

首先在终端使用 tcpdump 并加-w 参数达到将数据包写到.cap 格式的文件中

```
root@kylin-PC: /home/kylin/桌面
文件(F)  编辑(E)  视图(V)  搜索(S)  终端(T)  帮助(H)
root@kylin-PC:/home/kylin/桌面# tcp
tcpdump          tcpretrans-perf
root@kylin-PC:/home/kylin/桌面# tcpdump -w file.cap
tcpdump: listening on enp3s0, link-type EN10MB (Ethernet), capture size 262144
ytes
^C19 packets captured
22 packets received by filter
0 packets dropped by kernel
```

然后用 wireshark 打开该文件就可以清晰的看到包的目的端口和源端口等信息