Table of Contents

[1，内存的特点 0](#_Toc454526543)

[2，测试内存速度 0](#_Toc454526544)

[3，内存存储的数据有哪些 3](#_Toc454526545)

[4，我们可以优化哪部分程序 3](#_Toc454526546)

[5，查看系统内存信息 3](#_Toc454526547)

[6，cached实验 5](#_Toc454526548)

[7，buffers实验 6](#_Toc454526549)

[8，vmstat命令的使用 6](#_Toc454526550)

[9， buffer和cache的介绍 9](#_Toc454526551)

[10，释放buffer和cache 11](#_Toc454526552)

[11，OOM进程 12](#_Toc454526553)

[12，swap 14](#_Toc454526554)

[12.1，分区分多大 14](#_Toc454526555)

[12.2，什么样的数据才能往swap里面放？ 14](#_Toc454526556)

[13，使用内存文件系统 15](#_Toc454526557)

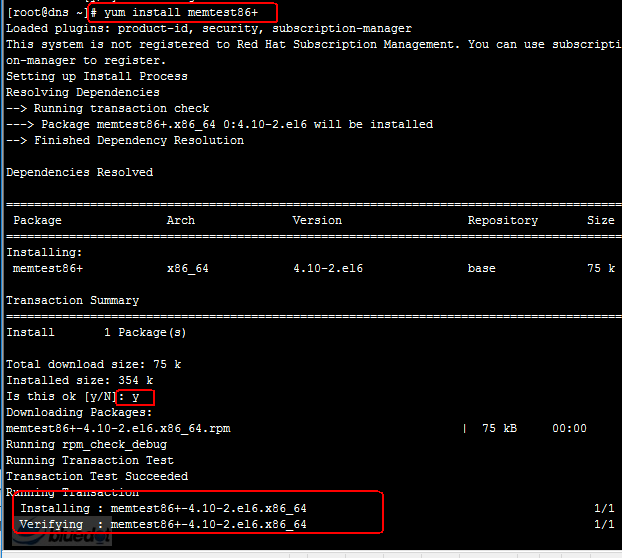
[13.1，测试将内存耗尽 16](#_Toc454526558)

# 1，内存的特点

内存的特点是速度快，所存数据不会保存，内存的最大小号来源于进程

# 2，测试内存速度

安装软件：memtest86+

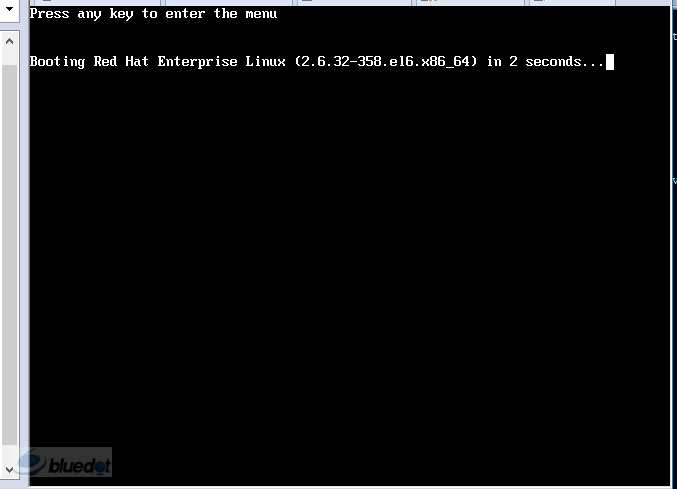


执行memtest-setup命令，多出一个操作系统

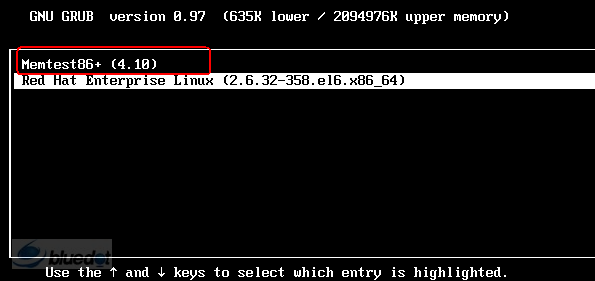


这个时候，执行reboot重启系统，然后在重启之后按回车选择系统，这个时候就可以看到memtest了，如下图所示

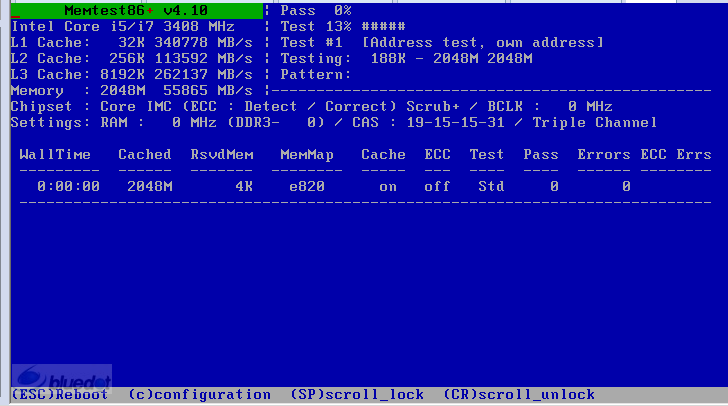
这里我们先按一下回车



然后就进入到这个界面了，这里选择Memtest86+



然后开始进行内存测试。 这里我们也能清晰的看到内存的各种速度。



# 3，内存存储的数据有哪些

程序代码,程序定义的变量(初始化和未初始化),继承父进程的环境变量,进程读取的文件,程序需要的库文件.还有程序本身动态申请的内存存放自己的数据

除了进程以外还有内核也要占用,还有buffer和cache,还有共享内存(如共享库)

我们使用管道符| 进程之间通讯也要使用到内存,socket文件

# 4，我们可以优化哪部分程序

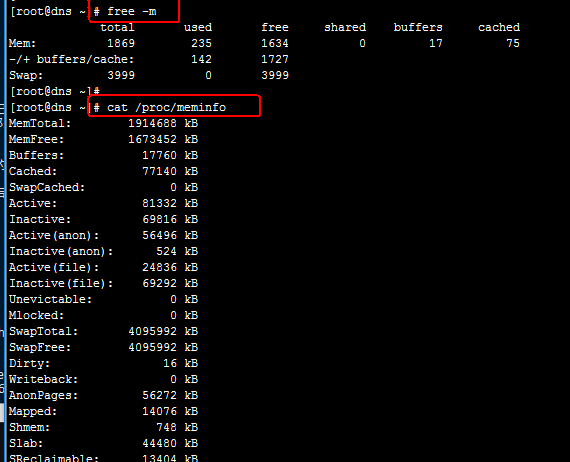
内核内存不能省。

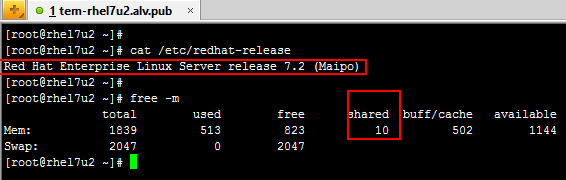
Buffer/cache 不能省

进程通讯不能省

# 5，查看系统内存信息

通过free命令可以查看内存，或者直接查看/proc/meminfo 文件的内容，free 命令显示出的内容实际上也是从/proc/meminfo里面抓出来的。





如上图所示，执行完free -m之后看到的界面，share里是0， 事实上，share在6之前包括6，这个地方永远都是0，已经被废弃掉了。Rhel7里面已经可以正常显示。而shered的值，就是cat /proc/meminfo|grep -I shm 中Shmem 的值。

[root@dns ~]# free -m

total used free shared buffers cached

Mem: 1869 275 1594 0 39 90

-/+ buffers/cache: 145 1723

Swap: 3999 0 3999

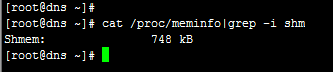
[root@dns ~]#

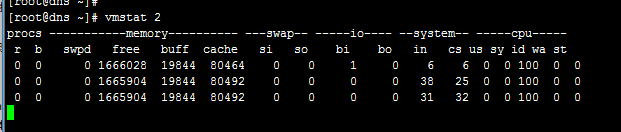
Free 和vmstat命令里看到的值都是从/proc/meminfo里面收集的信息。

Used 包括buffer和cached的值，是已使用的内存+buffers+cached，剩下的是free

-/+ buffers/cache: 145 1723

145是减去buffers和cached之后的物理内存和free之后的，1723就是free+buffers+cached的总量，这一部分内存也是可以用来使用的内存，可用内存，而145这部分就是不可用的了，因为它们已经在被使用了，free可以随时拿来用，buffers和cached虽然缓存着一些内容，但是当有程序需要来使用内存的时候，它们随时可以被拿来使用，所以它们也是可用内存。



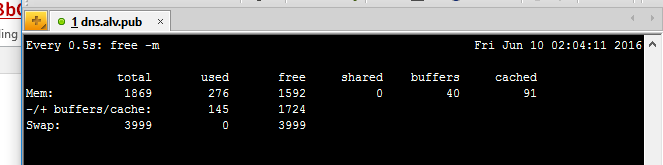


Buffers 索引缓存 存inode信息

Cached 页缓存 存block信息

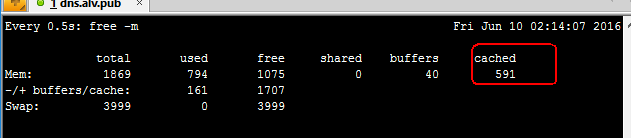
# 6，cached实验

咱们先开一个会话执行一个watch -n 0.5 free -m命令，每0.5秒刷新一次free -m 的数据。



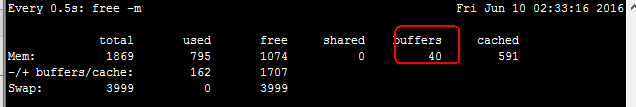
然后再开一个会话，执行dd if=/dev/zero of=1.txt bs=1M count=500

这条命令代表从/dev/zero这个文件中读取数据，将数据写到当前目录下的1.txt，每一个块大小为1M，一个500个块，也就是一共500M的大小，然后我们可以看到，cached的值，增加了整整500M, 这五百M数据，是因为系统会先从/dev/zero 读取数据写入到 cached里面，然后从cached里面写入到磁盘。



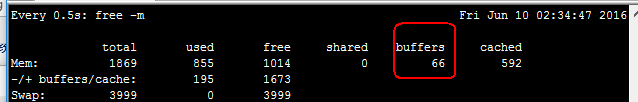
# 7，buffers实验

当前我们的buffers值是40.



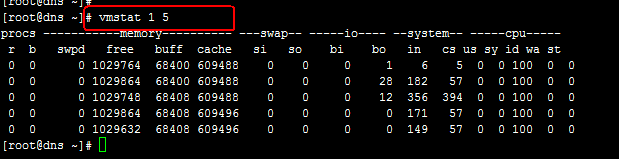
#find / 把/下所有文件都列出来 会读到directory block，通过inode找到文件名，有多少个文件就会读多少次目录块，所以我们现在的查找实际上是块操作，所以使用的是buffer

执行完 find / 之后，涨到了67



# 8，vmstat命令的使用

Vmstat 1 5, 代表执行vmstat 获取系统相关信息，每一秒钟更新一次数据，一共更新五次。如果后面不加那个5，只是执行vmstat 1，那么将会每秒钟更新一次，一直更新，直到我们手动结束该进程。



[root@dns ~]# vmstat 1 5

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 0 1029764 68400 609488 0 0 0 1 6 5 0 0 100 0 0

0 0 0 1029864 68400 609488 0 0 0 28 182 57 0 0 100 0 0

0 0 0 1029748 68408 609488 0 0 0 12 356 394 0 0 100 0 0

0 0 0 1029864 68408 609496 0 0 0 0 171 57 0 0 100 0 0

0 0 0 1029632 68408 609496 0 0 0 0 149 57 0 0 100 0 0

那么现在我们就来讲一下上面的这些值，都代表了什么。

首先是procs，procs下有 r 和b，

R：代表正在运行的进程数，如果长期大于cpu个数，说明cpu不足，需要增加cpu。

B：block，但这个是阻塞，表示在等待资源的进程数，比如正在等待I/O、或内存交换等，由于硬盘速度特别慢而导致内存同步的时候没有，那么现在告诉程序，说你不要产生数据，这就是阻塞，查看man手册可以看到的解释是The number of processes in uninterruptible sleep.。 就是出于不可中断的睡眠状态的进程。

B越大，证明磁盘压力很大。

——Memory ：

Swapd: 切换到内存交换区的内存数量(k表示)。如果swpd的值不为0，或者比较大，比如超过了100m，只要si、so的值长期为0，系统性能还是正常.

Free: 当前的空闲页面列表中内存数量（k表示）

Buff： 作为buffer cache的内存数量

Cache：作为page chache的内存数量，

——Swap

si: swap in ,代表把swap分区的数据放到内存

so：代表把内存数据放到磁盘上，也就是swap分区里。

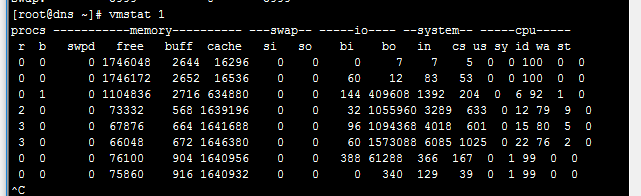
通过上面两个可以分析内存的使用情况，如果swap有数据是不是内存不够用了？不一定，因为系统会把一些用不到的进程放到swap里面，把腾出来的空间做缓存，如果发现si,so里面有数据，说明内存可能不够用了

——IO

Bi：block in,这里是块 进来，把块从硬盘搬进来，也就是说bi是读。

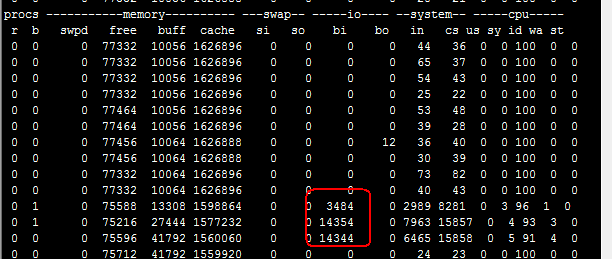
Bo：block out，把块从内存搬到硬盘，也就是说bo是写。

这里我们做一个测试，在一个会话中执行dd if=/dev/zero of=2.txt bs=1M count=4096，也就是创建一个大小为4G的文件，另一个会话中执行vmstat 1，观察数据，结果如下图所示，我们开始执行命令后bo那一行产生了大量的数据，也就是说，在写，写了4G数据之后，就结束了，



然后我们继续开着vmstat 1，然后在另一个会话执行一个find /， 执行的同时，观察vmstat 1这边的状态。

如下图所示，bi产生了数据，



——system

显示采集间隔内发生的中断数

In列表示在某一时间间隔中观测到的每秒设备中断数

Cs 列表示美妙产生的上下文切换次数。

——cpu

剩下的就是cpu的各种使用百分比

# 9， buffer和cache的介绍

根据时间和数据大小同步，主要用于写缓存

内核里面的一套系统：伙伴系统，负责吧内存里面的数据往硬盘上搬

rhel5：

kswapd pdflush

kswapd负责说什么时候搬数据

pdflush负责干活儿,他会一直开启着

rhel6:

kswapd负责说什么时候搬数据，但是干活儿的不是pdflush了

有需要搬的数据的时候，才产生一个进程---> flush 主设备号:从设备号 负责搬数据

已经同步到硬盘的数据就是干净数据

# cat /proc/sys/vm/dirty\_ 查看的是脏数据（缓存内还没来得急同步到硬盘的数据）

dirty\_background\_bytes dirty\_expire\_centisecs

dirty\_background\_ratio dirty\_ratio

dirty\_bytes dirty\_writeback\_centisecs

那么将数据同步到硬盘是什么意思呢？ 比如一个文件里面的内容，我们查看它的时候它应该是在存储在硬盘上的对吧？但是Linux内核为了达到最佳的磁盘操作效率，会把需要写入到磁盘的数据现在内存中缓存起来，在合适的时候才真正写入到磁盘中，这在绝大多数情况都是没有任何问题的，而且提高了系统的效率，但是如果系统当机、掉电，就会有些文件内容不会保存下来。在Linux系统关机或者重启时，会自动把缓冲区的内容自动同步到磁盘中。我们也可以手工去执行sync命令，强制将内存中的文件缓冲内容写到磁盘，这个命令是通过调用sync系统调用来实现的。但是，我认为一般情况下我们是没有必要去执行这个命令的，因为Linux内核会尽快让内存中的数据自动同步到磁盘上去，而且，谁又会知道什么时候会掉电或宕机呢？

[root@dns ~]# cat /proc/sys/vm/dirty\_expire\_centisecs

3000

想知道这里面是什么可以使用下面的man或者kernel-doc查看

单位百分之一秒，这里也就是30秒，30秒之后标记为脏数据，意味着用户写的数据在30秒之后才有可能被刷入磁盘，在这期间断电可能会丢数据

cat /proc/sys/vm/dirty\_writeback\_centisecs // 5秒钟往硬盘同步一次数据5秒同步一次脏数据（在缓存中的）

500

假如我内存1G

1秒 100M

2秒 300M

3秒 400M

4秒 400M

还没到5秒，但是内存使用已经超过1G了，这时候怎么办？下面的文件来解决

[root@localhost ~]# cat /proc/sys/vm/dirty\_ratio

40 //如果单个进程占用的buffer/cache达到内存总量的40%,立刻同步。

假如我内存1G，一个进程

1秒 1M

2秒 3M

3秒 4M

4秒 40M

那要是1000个进程呢？这时候怎么办？下面的文件来解决

[root@dns ~]# cat /proc/sys/vm/dirty\_background\_ratio

10 //所有进程占用的buffer/cache使得剩余内存低于内存总量的10%，立刻同步

# cat /proc/sys/vm/dirty\_background\_bytes //上面的ratio文件用百分比，这个用字节限制，但是百分比存在的时候，字节不生效

0

如果服务器是一个数据服务器，比如NAS，dirty\_writeback和dirty\_ratio里面的数值可以适当改大一点,存储需要频繁读数据的时候，可以直接从内存里面读，而且在同步数据的时候会使用更大的连续的块儿。

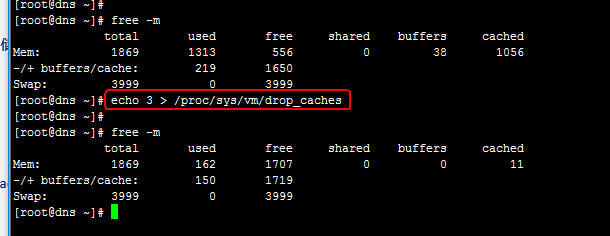
# 10，释放buffer和cache

释放buffer和cache的方式是修改这个文件/proc/sys/vm/drop\_caches 1 释放buffe 2 释放cache ，3 buffer/cache都释放

[root@dns ~]# cat /proc/sys/vm/drop\_caches

3

清空缓存 echo 3 > /proc/sys/vm/drop\_caches



内存如果真耗尽了，后果无法预测。

# 11，OOM进程

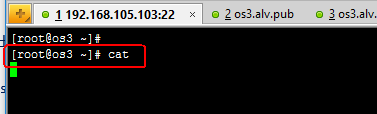
当内存耗尽的时候，系统会出现一个OOM killer进程在系统内随机杀进程。

每个运行的程序都会有一个score（分），这个是不良得分，所以谁分高，就杀谁。

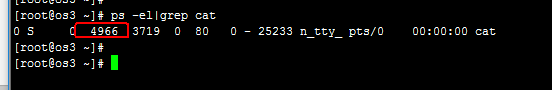
如果还不行的话，他会自杀，也就是杀kernel，就会出现内核恐慌（panic），所以会死机。

实验：

这里我们就在一个会话里执行一个cat



然后在另一个会话里，我们先查看一下它的进程。



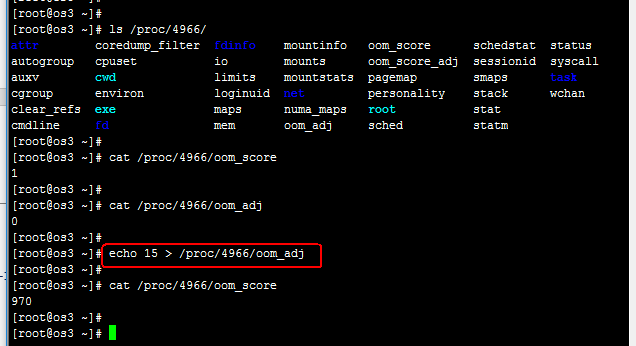
Ok，现在我们看到我们刚才直接的cat命令的进程是4966，那么接下来，我们就去查看关于这个进程的更多信息。

[root@os3 ~]# cat /proc/3831/oom\_adj

0

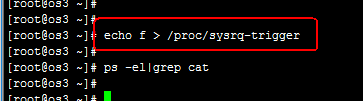
我们可以通过修改这个文件里面的内容来干预oom得分。 -17 15，-17是免杀，15是先杀掉。

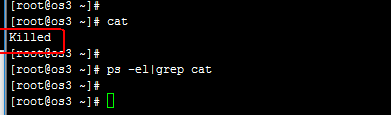
这里我们执行，echo 15 > /proc/3831/oom\_adj ，然后查看/proc/4966/oom\_score的值，可以发现，它变大了。



然后执行 echo f > /proc/sysrq-trigge，手动启动oom\_killer，必杀一个。因为上面已经把4966的adj改成了15，所以这次启动杀死了cat进程。

这个时候我们再去看执行cat的那个会话，发现这个命令已经被结束了，进程已经终止了。





# 12，swap

那么到底怎么解决内存耗尽的问题呢？那就是swap

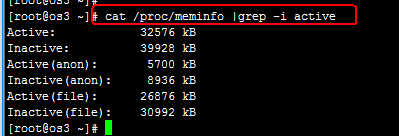
假如a,b,c已经把内存占满了，那么来了个d，内核先看看abc谁不忙，就把谁的数据先放到swap里面去，比如a不用，把a的数据放到swap里面去，释放出来的空间给d。

## 12.1，分区分多大

Swao分区分多大呢？一般情况下是内存的两倍，不过现在内存很大比如256G，那就没必要2倍了。

## 12.2，什么样的数据才能往swap里面放？

这里我们执行cat /proc/meminfo |grep -i active



Active活跃数据，inactive为非活跃数据，又分为匿名数据和文件数据。

匿名数据不能往swap里面放

文件形式的active不能往swap里面放，只有文件的inactive才能往swap放，所以并不是有了swap，内存就解决了。

什么时候放进去？ 根据swap\_tendency（swap 趋势）

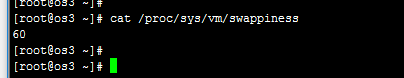
Swap\_tendency mapped+ratio/2 + distress + vm\_swappiness,

这就是swap趋势，如果这个值到达100，就往交换分区里面放，如果小于100，尽量不往里面放，但是就算到100，页只能说内核倾向于要往swap里面放，但也不一定放。

系统就只开放第三个给用户设置

cat /proc/sys/vm/swappiness ## swap的喜好程度，范围0-100

60

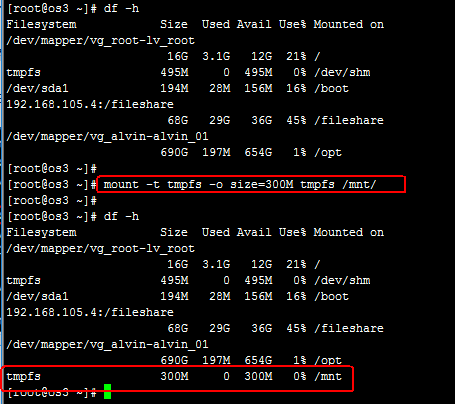


# 13，使用内存文件系统

df -h

tmpfs 495M 0 495M 0% /dev/shm

上面这个，就是共享内存，tmpfs。 内存里面的临时文件系统，系统会承诺拿出50%（这里是500M，因为在这个系统里内存是1G）的空间来做shm，只是承诺，实际用多少给多少。如果内存比较充足的情况下，我们可以拿内存当硬盘使用。



#mount -t tmpfs -o size=1000M tmpfs /mnt //挂内存

1572864000 bytes (1.6 GB) copied, 1.32034 s, 1.2 GB/s

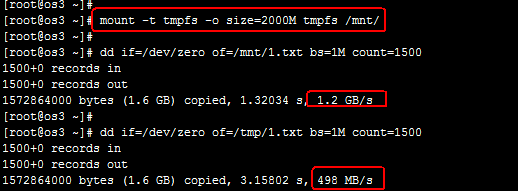
[root@os3 ~]#

[root@os3 ~]# dd if=/dev/zero of=/tmp/1.txt bs=1M count=1500

1500+0 records in

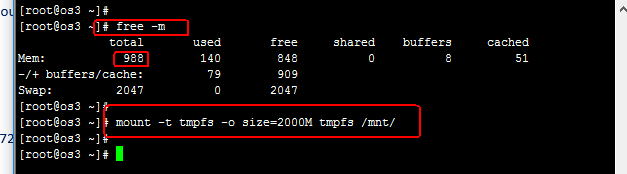
1500+0 records out

1572864000 bytes (1.6 GB) copied, 3.15802 s, 498 MB/s

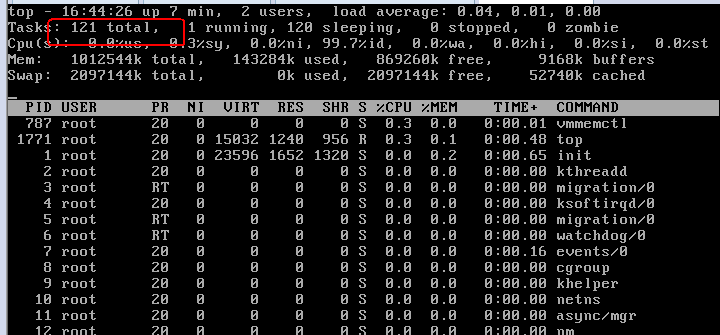


## 13.1，测试将内存耗尽

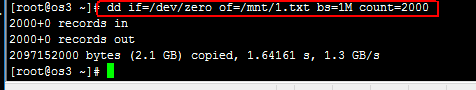
如下图所示，该服务器的内存是1G，这里我们创建一个承诺给2G空间的临时文件系统，挂载到/mnt目录下。



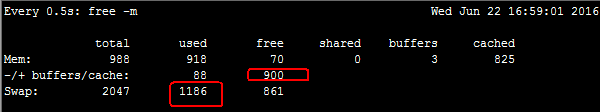
这个时候我们先运行一个top，可以看到当前的总进程数是121



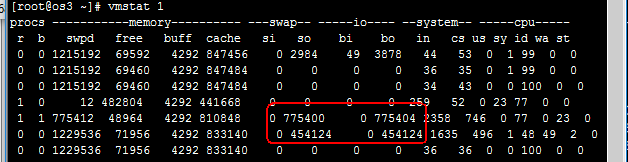
这个时候我们写入一个2G的文件，到/mnt里面，而那个是内存的临时文件系统，



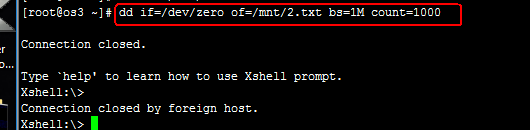
这个时候，我们执行free -m 可以看到swap使用了1186，加cached了825的数据量，计算一下，刚好过2000，



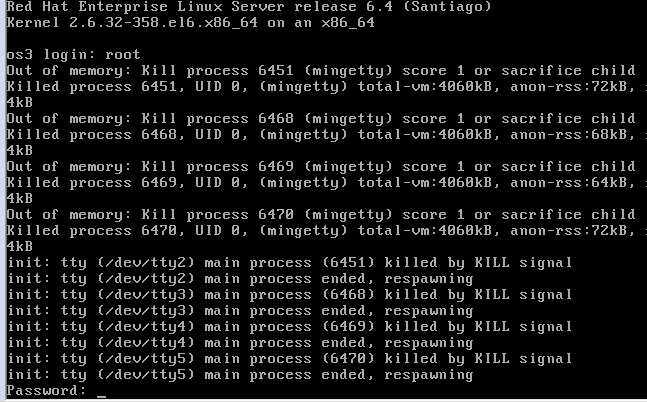
同时在执行dd命令之前我们就在另一个会话中开启了vmstat 1，这个时候我们就捕捉到了so 和bo产生的数据，so是代表什么呢？swap out，就是将数据从内存里搬到swap里，swap分区也硬盘，所以bo里面会有数据产生，



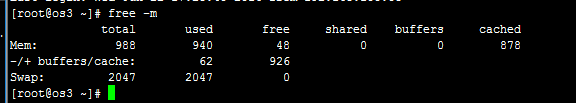
当我们再写入一个1G的文件之后呢，如下图所示，我们发现，会话断掉了，甚至别的会话也都断掉了，



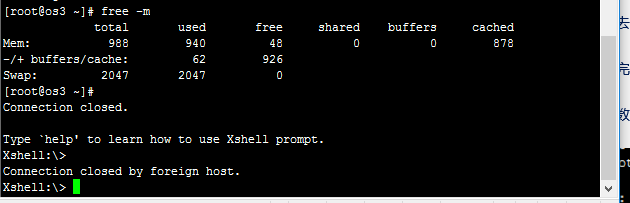
甚至系统console 那边，也给出了如下图所示的提示，进程都被杀掉了，



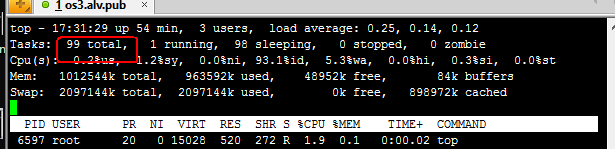
尝试去登录，也有时候会成功，有时候会失败，终于登录进去后，我们执行free -m，可以看到，swap空间已经全部被用完了，内存的free也只剩48，大量的内存被用于cached了，而cached里的数据量，也就是我们写到/mnt里面的数据量



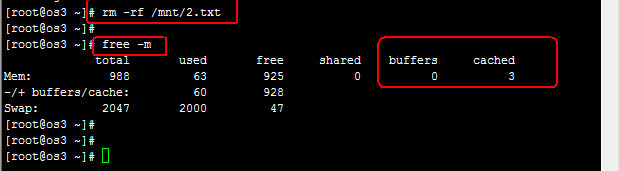
然后，ssh的会话也又自动断掉了。



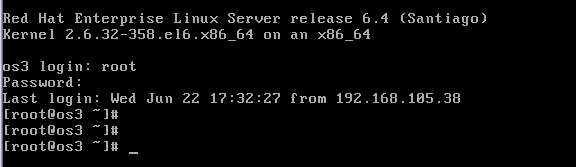
我们又好不容易登录进去后，执行了top，如下图所示，进程数量也少了很多，因为，被oom给杀掉了。



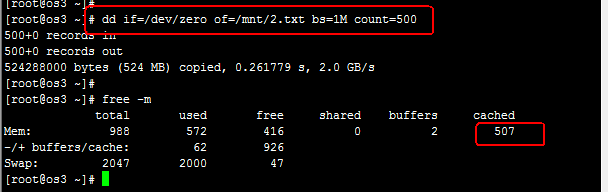
又一次登录进去之后，我们赶紧执行了rm -rf /mnt/2.txt，删掉了那个1G的文件。然后，一切就开始正常了。



之前在tty那边登录不进去的，现在也能登陆进去了。

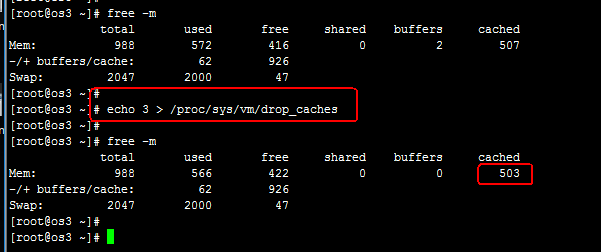


然后我们再次写一个500M的文件到/mnt里面，现在我们能看到cached里面又增加了500M的数据。



那么，之前我们说的清理缓存的方式，可以清理掉它么？执行echo 3 > /proc/sys/vm/drop\_caches,

结果如下图所示，清理不掉，



14，虚拟内存和物理内存

虚拟内存申请：

32位CPU,2^32也就是4G

64位cpu,2^64

每个程序都最多能申请4G的虚拟内存，但是现在这4G内存还和物理内存没关系呢，a说我先用100M，然后内核就会把100M映射给物理内存

VIRT就是程序运行的时候说申请的虚拟内存，RES就是映射的内存

为什么要有虚拟内存？

跟开发有关系，内存是有地址空间的，开发者在调用内存的时候如果直接调用物理内存，开发者不知道哪块儿地址被占用了，所以在中间内核站出来给开发者分配，开发者只需要提出需要多大内存，由内核来解决你的内存就可以了

------------

程序1 程序2

4G 4G

-----------

kernel

-----------

物理内存

-----------

以上3层，第一层就是程序可以使用的虚拟内存，程序可以跟内核申请需要多少内存，内核就分配相应大小的物理内存给程序就可以了

========================

映射表：

概念：

内存是分页的，1个page是4k(默认值),在硬盘上分块，硬盘数据和内存数据是一一对应

问题：

条目非常多，查询特别慢

解决：

固有方法：

硬件TLB ，在cpu里面，用来解决查询映射表慢的问题，第一次查询过之后把结果缓存到TLB里面，以后再查的时候就可以直接从TLB里面提取

# yum install x86info

# x86info -a 可以查询TLB信息

自定义方法：

如果page变大，条目就会变少，这样就会提高查询速度

大于4k的分页称为hugepage 巨页 ，但是这个需要程序支持

那我们现在的操作系统是否支持巨页

# cat /proc/meminfo | grep -i hugepage

AnonHugePages: 26624 kB

HugePages\_Total: 0 我现在没有巨页

HugePages\_Free: 0

HugePages\_Rsvd: 0

HugePages\_Surp: 0

Hugepagesize: 2048 kB 说明现在我的系统支持2M的巨页

假如一个程序需要200M的巨页，那么就要把total改成100

#echo 100 > /proc/sys/vm/nr\_hugepages //修改巨页total数目

#mkdir dir1

#mount -t hugetlbfs none /dir1 那么现在程序使用/dir1就可以了

外翻：

TLB(Translation Lookaside Buffer)传输后备缓冲器是一个内存管理单元用于改进虚拟地址到物理地址转换速度的缓存。TLB是一个小的，虚拟寻址的缓存，其中每一行都保存着一个由单个PTE组成的块。如果没有TLB，则每次取数据都需要两次访问内存，即查页表获得物理地址和取数据

========================

进程间通信(IPC)

种类：

进程间通信的方式有5种:

1.管道(pipe) 本地进程间通信，用来连接不同进程之间的数据流

2.socket 网络进程间通信，套接字(Socket)是由Berkeley在BSD系统中引入的一种基于连接的IPC，是对网络接口(硬件)和网络协议(软件)的抽象。它既解决了无名管道只能在相关进程间单向通信的问题，又解决了网络上不同主机之间无法通信的问题。

以上两种unix遗留下来的

3.消息队列(Message Queues) 消息队列保存在内核中，是一个由消息组成的链表。

4.共享内存段(Shared Memory) 共享内存允许两个或多个进程共享一定的存储区，因为不需要拷贝数据，所以这是最快的一种IPC。

5.信号量集(Semaphore Arrays) System V的信号量集表示的是一个或多个信号量的集合。

查看：

ipc 进程间通信

#ipcs //这条命令可以看到后3种,前两种可以通过文件类型查看

含义：

管道

a | b

在内存打开一个缓冲区，a把结果存到缓冲区，b去缓冲区里面拿数据

管道通信的时候 独木桥：特点-->只能一个人 单向 先进先出

3个人过桥，一个一个的过，那如果100个人过，速度会很慢，所以管道传输的数据有限

socket

IE浏览器 访问网站 通过端口 端口在系统内实际不存在是个伪概念，只是一个标识，

a会打开一个buffer b会打开一个buffer ，这两个buffer用来接受数据包，并且重组，交给apache的socket，apache就会去socket接受数据

消息队列

跟管道基本一样 也是独木桥，也是单向，先进先出，但是他会对消息进程排队，谁着急谁先走，那么过河的人多了之后，同样也是数据传输较慢

共享内存段

开辟一块内存，a把数据全都丢到共享内存里面，b去共享内存拿数据，而且b可以按需选择拿哪些数据

共享内存段在oracle里面肯定要使用

信号量

在a和b之间传递信号，a把一个文件锁住给b发一个信号，说这个文件我正在使用

信号所携带的数据量非常有限，只能指定信号是干什么用的

==============================================

查看内存使用情况

[root@localhost ~]# sar -r 1 1

01时31分38秒 kbmemfree kbmemused %memused kbbuffers kbcached kbcommit %commit

01时31分39秒 6045368 1917916 24.08 67236 649020 2435764 17.73

kbcommit：保证当前系统所需要的内存,即为了确保不溢出而需要的内存(RAM+swap).

%commit：这个值是kbcommit与内存总量(包括swap)的一个百分比.