Управление памятью в Linux

В Linux применяется следующее основное правило: неиспользуемая страница оперативной памяти считается потерянной памятью. Оперативная память тратится не только для данных, используемых прикладными приложениями. В ней также хранятся данные для самого ядра и, самое главное, в эту память могут отображаться данные, хранящиеся на жестком диске, что используется для супер-быстрого к ним доступа — команда top указывает об этом в столбцах "buffers/cache" ("буферы / кэш"), "disk cache" ("дисковый кэш)" или "cached" ("кэшировано"). Кэшированная память по сути свободна, поскольку ее можно быстро освободить в случае, если работающей (или только что запущенной) программе потребуется память.

Сохранение кэша означает, что если кому-нибудь еще раз потребуются те же самые данные, то есть большая вероятность, что они все еще будут находиться в кэше в оперативной памяти.

Поэтому первое, чем можно воспользоваться в вашей системе, это команда `free`, которая предоставит вам первоначальную информацию о том, как используется ваша оперативная память.

```
xubuntu-home:~# free
     total       used       free     shared    buffers     cached
Mem: 1506       1373        133          0         40        359
-/+ buffers/cache:       972        534
Swap:                    486         24        462
```

В строке -/+ buffers/cache показывается, сколько памяти используется и сколько памяти свободно с точки зрения ее использования в приложениях. В этом примере приложениями уже используется 972 Мб памяти и еще 534 МБ памяти могут быть использованы.

Вообще говоря, если используется хотя бы немного памяти подкачки swap, то использование памяти вообще не повлияет на производительность системы.

Но если вы хотите получить более подробную информацию о вашей памяти, то вы должны проверить файл /proc/meminfo:

```
xubuntu-home:~# cat /proc/meminfo
MemTotal:        1543148 kB
MemFree:          152928 kB
Buffers:           41776 kB
Cached:           353612 kB
SwapCached:         8880 kB
Active:           629268 kB
Inactive:         665188 kB
Active(anon):     432424 kB
Inactive(anon):   474704 kB
Active(file):     196844 kB
Inactive(file):   190484 kB
Unevictable:         160 kB
Mlocked:             160 kB
HighTotal:        662920 kB
HighFree:          20476 kB
LowTotal:         880228 kB
LowFree:          132452 kB
SwapTotal:        498684 kB
SwapFree:         470020 kB
Dirty:                44 kB
Writeback:             0 kB
AnonPages:        891472 kB
Mapped:           122284 kB
Shmem:              8060 kB
Slab:              56416 kB
SReclaimable:      44068 kB
SUnreclaim:        12348 kB
KernelStack:        3208 kB
PageTables:        10380 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
```

```
WritebackTmp:          0 kB
CommitLimit:     1270256 kB
Committed_AS:    2903848 kB
VmallocTotal:     122880 kB
VmallocUsed:        8116 kB
VmallocChunk:     113344 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
DirectMap4k:       98296 kB
DirectMap4M:      811008 kB
```

Что означает MemTotal (Всего памяти) и MemFree (Свободная память), понятно для всех; остальные значения поясняются дальше:

Cached

Страничный кэш в системе Linux ("Cached:" в meminfo) является в большинстве систем самым крупным потребителем памяти. Каждый раз, когда вы выполняете операцию чтения read () из файла, расположенного на диске, данные считываются в память и помещаются в страничный кэш. После того, как операция read() завершается, ядро может просто выбросить страницу памяти, так как она не используется. Однако, если вы второй раз выполняете операцию чтения той же самой части файла, данные будут считываться непосредственно из памяти и обращения к диску не будет. Это невероятно ускоряет работу и, поэтому, в Linux так интенсивно используется кэширование страниц: ставка делается на то, что если вы обратились к некоторой странице дисковой памяти, то вскоре вы обратитесь к ней снова.

dentry/inode caches

Каждый раз, когда вы в файловой системе выполняете операцию "ls'" (или любую другую операцию: open(), stat() и т.д.), ядру требуются данные, которые находятся на диске. Ядро анализирует эти данные, находящиеся на диске, и помещает его в некоторых структуры данных, независимые от файловой системы, с тем, чтобы они могли в различных файловых системах обрабатываться одним и тем же образом. Таким же самым образом, как кэширование страниц в приведенных выше примерах, ядро может после того, как будет завершена команда "ls", стереть эти структуры. Тем не менее, делается такое же предположение, как и раньше: если вы однажды считали эти данные, вы обязательно прочитаете их еще раз. Ядро хранит эту информацию в нескольких местах "кэша", которые называются кэш памятью dentry и inode. Кэш память dentries являются общей для всех файловых систем, но каждая файловая система имеет свой собственный кэш inodes.

Эта оперативная память является в meminfo составной частью "Slab:"

Вы можете просмотреть различную кэш память и узнать ее размеры с помощью следующей команды:

```
head -2 /proc/slabinfo; cat /proc/slabinfo  | egrep dentry\|inode
```

Buffer Cache

Кэш буфера ("Buffers:" в meminfo) является близким родственником кэш памяти dentry/inode. Данные dentries и inodes, размещаемые в памяти, представляют собой описание структур на диске, но располагаются они по-разному. Это, возможно, связано с тем, что у нас в копии, расположенной в памяти, используется такая структура, как указатель, но на диске ее нет. Может также случиться, что на диске байты будут располагаться не в том порядке, как это нужно процессору.

Отображение памяти в команде top: VIRT, RES и SHR

Если вы запускаете команду `top`, то три строки будут описывать к использованию памяти. Вы должны понимать их значение с тем, чтобы понять, сколько памяти требуется вашему серверу.

VIRT является сокращением от virtual size of a process (виртуальный размер процесса) и представляет собой общий объем используемой памяти: памяти, отображаемой самой в себя (например, памяти видеокарты для сервера X), файлов на диске, которые отображаются в память (особенно это касается разделяемых библиотек) и памяти, разделяемой совместно с другими процессами. Значение VIRT указывает, сколько памяти в настоящий момент доступно программе.

RES является сокращением от resident size (размер резидентной части) и является точным указателем того, сколько в действительности потребляется процессом реальной физической памяти. (Что также соответствует значению, находящемуся непосредственно в колонке %MEM). Это значение практически всегда меньше, чем размер VIRT, т.к. большинство программ зависит от библиотеки C.

SHR показывает, какая величина от значения VIRT является в действительности разделяемой (по памяти или за счет использования библиотек). В случае библиотек, это не обязательно означает, что вся библиотека находится в резидентной памяти. Например, если программа использует только несколько функций библиотеки, то при отображении в память будет использована вся библиотека, что будет учтено в значениях VIRT и SHR, но, на самом деле, будет загружена часть библиотеки, содержащая используемые функции, и это будет учтено в значении RES.

Подкачка памяти - swap

Прежде всего, ядро пытается не допустить, чтобы у вас значение свободной оперативной памяти приближалось к 0 байтов. Это связано с тем, что когда нужно освободить оперативную память, то обычно требуется выделить немного больше памяти. Это обусловлено тем, что нашему ядру требуется своего рода "рабочее пространство" для выполнения своих действий, и поэтому, если размер свободной оперативной памяти становится равным нулю, ядро ничего больше сделать не сможет.

На основании общего объема оперативной памяти и соотношения ее различных типов (память high/low), ядро эвристически определяет то количество памяти в качестве рабочего пространства, при котором оно чувствует себя комфортно. Когда эта величина достигается, ядро начинает возвращать память для других различных задач, описанных выше. Ядро может вернуть себе память из любой из этих задач.

Однако, есть другой потребитель памяти, о котором мы, возможно, уже забыли: данные пользовательских приложений.

Как только ядро принимает решение, что ему не требуется получать память из каких-либо других источников, которые мы описывали ранее, оно запускает память подкачки swap. В ходе этого процесса оно получает данные пользовательских приложений и записывает их в специальное место (или места) на диске. Обратите внимание, что это происходит не только тогда, когда оперативная память близка к заполнению, ядро может принять решение перенести в память swap также данные, находящиеся в оперативной памяти, если они некоторое время не использовались (смотрите раздел "Подкачка памяти").

По этой причине, даже система с огромным количеством оперативной памяти (даже если ее правильно настроить) может использовать память подкачки swap. Есть много страниц памяти, в которых находятся данные пользовательских приложений, но эти страницы используются редко. Все это является причиной, чтобы перенести их в раздел swap и использовать оперативную память для других целей.

Вы можете с помощью команды `free` проверить, используется ли память swap; для примера, который я уже использовал выше, в последней строке выдаваемых данных показывается информация о размере памяти swap:

```
xubuntu-home:~# free
             total       used       free     shared    buffers     cached
Mem:          1506       1373        133          0         40        359
-/+ buffers/cache:        972        534
Swap:          486         24        462
```

Мы видим, что на этом компьютере уже используется 24 мегабайта памяти swap и для использования доступно еще 462 Мб.

Таким образом, сам факт использования памяти swap не является доказательством того, что в системе при ее текущей рабочей нагрузке слишком мало оперативной памяти. Лучший способ это определить с помощью команды `vmstat` - если вы увидите, что много страниц памяти swap перемещаются на диск и обратно, то это означает, что память swap используется активно, что система "пробуксовывает" или что ей нужна новая оперативная память поскольку это ускорит подкачку данных приложений.

На моем ноутбуке Gentoo, когда он простаивает, это выглядит следующим образом:

```
~ # vmstat 5 5
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  0      0 2802448  25856 731076    0    0    99    14  365  478  7  3 88  3
 0  0      0 2820556  25868 713388    0    0     0     9  675  906  2  2 96  0
 0  0      0 2820736  25868 713388    0    0     0     0  675  925  3  1 96  0
 2  0      0 2820388  25868 713548    0    0     0     2  671  901  3  1 96  0
 0  0      0 2820668  25868 713320    0    0     0     0  681  920  2  1 96  0
```

Обратите внимание на то, что в выходных данных команды `free` у вас есть только 2 значения, относящихся к памяти swap: free (свободная память) и used (используемая память), но для памяти подкачки swap также есть еще одно важное значение: Swap cache (показатель кэширования памяти подкачки).

Кэширование памяти swap (Swap Cach)

Кеширование памяти swap по сути очень похоже на страничное кеширование. Страница данных пользовательского приложения, записываемая на диск, очень похожа на страницу данных файла, находящуюся на диске. Каждый раз, когда страница считывается из файла подкачки ("si" в vmstat), она помещается в кэш подкачки. Так же, как страничное кэширование, все это выполняется ядром. Ядро решает, нужно ли вернуть обратно на диск конкретную страницу. Если в этом возникнет необходимость, то можно проверить, есть ли копия этой страницы на диске и можно просто выбросить страницу из памяти. Это избавит нас от затрат на переписывание страницы на диск.

Кэширование памяти swap действительно полезно только когда мы читаем данные из памяти swap и никогда в нее не делаем записи. Если мы выполняем запись на страницу, то копия на диске не будет соответствовать копии, находящейся в памяти. Если это случится, то мы должны произвести запись страницы на диск точно также, как мы делали это первый раз. Несмотря на то, что затраты на сохранение всей страницы больше, чем затраты на запись небольшого измененного кусочка, система будет работать лучше.

Поэтому, чтобы узнать, что память swap действительно используется, мы должны из значения SwapUsed вычесть значение SwapCached, вы можете найти эту информацию в /proc/meminfo.

Подкачка памяти

Когда приложению нужна память, а вся оперативная память полностью занята, то в распоряжении ядра есть два способа освободить память: оно может либо уменьшить размер дискового кэша в оперативной памяти, убирая устаревшие данные, либо оно может сбросить на диск в swap раздел несколько достаточно редко используемых порций (страниц) программы. Трудно предсказать, какой из способов будет более эффективным. Ядро, исходя из недавней истории действий в системе, делает попытку приблизительно отгадать на данный момент эффективность каждого из этих двух методов.

До ядер версии 2.6 у пользователя не было возможности влиять на эти оценки, так что могла возникнуть ситуации, когда ядро часто делало неправильный выбор, что приводило к пробуксовыванию и низкой производительности. В версии 2.6 ситуация с подкачкой памяти была изменена.

Подкачке памяти назначается значение от 0 до 100, которое изменяет баланс между подкачкой памяти приложений и освобождением кэш памяти. При значении 100 ядро всегда предпочтет найти неактивные страницы и сбросить их на диск в раздел swap; в других случаях этот сброс будет осуществляться в зависимости от того, сколько памяти занимает приложение и насколько трудно выпонять кэширование при поиске и удалении неактивных элементов.

По умолчанию для этого устанавливается значение 60. Значение 0 дает нечто близкое к старому поведению, когда приложения, которым нужна память, заставляли немного уменьшить размер кэша оперативной памяти. Для ноутбуков, для которых предпочтительно иметь диски с меньшей скоростью вращения, рекомендуется использовать значение 20 или меньше.

Задание для лабораторной работы

1. Процессы и используемая память.

1.1 Выдать 10 процессов, потребляющих наибольшее количество памяти

```
# ps -auxf | sort -nr -k 4 | head -10
```

1.2 Выдать 10 процессов, потребляющих наибольший ресурс процессора

```
# ps -auxf | sort -nr -k 3 | head -10
```

2. free – использование памяти

Команда free показывает общее количество свободной и используемой системой физической памяти и памяти свопинга, а также размеры буферов, используемые ядром.

```
# free
```

Пример вывода данных:

```
total       used       free     shared    buffers     cached
Mem:      12302896    9739664    2563232          0     523124    5154740
-/+ buffers/cache:    4061800    8241096
Swap:      1052248          0    1052248
```

3. iostat – средняя загрузка процессора, активность дисков

Команда iostat выдает статистику использования процессора, а также статистику ввода/вывода для устройств, разделов и сетевых файловых систем (NFS).

```
# iostat
```

Пример вывода данных:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)      06/26/2009

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3.50    0.09    0.51    0.03    0.00   95.86

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda              22.04        31.88       512.03   16193351  260102868
sda1              0.00         0.00         0.00       2166        180
sda2             22.04        31.87       512.03   16189010  260102688
sda3              0.00         0.00         0.00       1615          0
```

4. sar – сбор и выдача данных о системной активности

Команда sar используется для сбора информации о системной активности и выдачи ее в виде отчета или ее сохранения. Чтобы увидеть значение считчика сетевой активности, введите:

```
# sar -n DEV | more
```

Для того, чтобы увидеть значения счетчиков сетевой активности, начиная с 24-го:

```
# sar -n DEV -f /var/log/sa/sa24 | more
```

С помощью команды sar Вы можете также выдавать данные в режиме реального времени:

```
# sar 4 5
```

Пример вывода данных:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)               06/26/2009

06:45:12 PM       CPU     %user     %nice   %system   %iowait    %steal     %idle
06:45:16 PM       all      2.00      0.00      0.22      0.00      0.00     97.78
06:45:20 PM       all      2.07      0.00      0.38      0.03      0.00     97.52
06:45:24 PM       all      0.94      0.00      0.28      0.00      0.00     98.78
06:45:28 PM       all      1.56      0.00      0.22      0.00      0.00     98.22
06:45:32 PM       all      3.53      0.00      0.25      0.03      0.00     96.19
Average:          all      2.02      0.00      0.27      0.01      0.00     97.70
```

5. mpstat – использование мультипроцессора

Команда mpstat выводит данные об активности каждого имеющегося в наличие процессора, процессор 0 будет первым. Команда mpstat -P ALL выводит данные о среднем использовании ресурсов для каждого из процессоров:

```
# mpstat -P ALL
```

Пример вывода данных:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)               06/26/2009

06:48:11 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
06:48:11 PM  all    3.50    0.09    0.34    0.03    0.01    0.17    0.00   95.86   1218.04
06:48:11 PM    0    3.44    0.08    0.31    0.02    0.00    0.12    0.00   96.04   1000.31
06:48:11 PM    1    3.10    0.08    0.32    0.09    0.02    0.11    0.00   96.28     34.93
06:48:11 PM    2    4.16    0.11    0.36    0.02    0.00    0.11    0.00   95.25      0.00
06:48:11 PM    3    3.77    0.11    0.38    0.03    0.01    0.24    0.00   95.46     44.80
06:48:11 PM    4    2.96    0.07    0.29    0.04    0.02    0.10    0.00   96.52     25.91
06:48:11 PM    5    3.26    0.08    0.28    0.03    0.01    0.10    0.00   96.23     14.98
06:48:11 PM    6    4.00    0.10    0.34    0.01    0.00    0.13    0.00   95.42      3.75
06:48:11 PM    7    3.30    0.11    0.39    0.03    0.01    0.46    0.00   95.69     76.89
```

6. pmap – использование процессами оперативной памяти

Команда pmap выдает данные о распределении памяти между процессами. Использование этой команды позволит найти причину узких мест, связанных с использованием памяти.

```
# pmap -d PID
```

Для того, чтобы получить информацию об использовании памяти процессом с pid # 47394, введите:

```
# pmap -d 47394
```

Пример вывода данных:

```
47394:   /usr/bin/php-cgi
Address           Kbytes Mode  Offset           Device    Mapping
0000000000400000    2584 r-x-- 0000000000000000 008:00002 php-cgi
0000000000886000     140 rw--- 0000000000286000 008:00002 php-cgi
00000000008a9000      52 rw--- 00000000008a9000 000:00000   [ anon ]
0000000000aa8000      76 rw--- 00000000002a8000 008:00002 php-cgi
000000000f678000    1980 rw--- 000000000f678000 000:00000   [ anon ]
000000314a600000     112 r-x-- 0000000000000000 008:00002 ld-2.5.so
000000314a81b000       4 r---- 000000000001b000 008:00002 ld-2.5.so
000000314a81c000       4 rw--- 000000000001c000 008:00002 ld-2.5.so
000000314aa00000    1328 r-x-- 0000000000000000 008:00002 libc-2.5.so
000000314ab4c000    2048 ----- 000000000014c000 008:00002 libc-2.5.so
.....
.......
..
00002af8d48fd000       4 rw--- 0000000000006000 008:00002 xsl.so
00002af8d490c000      40 r-x-- 0000000000000000 008:00002 libnss_files-2.5.so
00002af8d4916000    2044 ----- 000000000000a000 008:00002 libnss_files-2.5.so
00002af8d4b15000       4 r---- 0000000000009000 008:00002 libnss_files-2.5.so
00002af8d4b16000       4 rw--- 000000000000a000 008:00002 libnss_files-2.5.so
00002af8d4b17000  768000 rw-s- 0000000000000000 000:00009 zero (deleted)
00007fffc95fe000      84 rw--- 00007ffffffea000 000:00000   [ stack ]
ffffffffff600000    8192 ----- 0000000000000000 000:00000   [ anon ]
mapped: 933712K    writeable/private: 4304K    shared: 768000K
```

Последняя строка очень важна:

- **mapped: 933712K** общее количество памяти, отведенного под файлы
- **writeable/private: 4304K** общее количество приватного адресного пространства
- **shared: 768000K** общее количество адресного пространства, которое данный процесс использует совместно другими процессами.