TheBazaar . Engineering . **MemoryAccountingTools**

-- SteveLongerbeam - 05 Feb 2004

ConsumerEdition Memory Accounting Tools

Technical Specification for Displaying Memory Usage

Contents

Introduction

This specification describes various /proc filesystem entries that display total and per-process memory usage information in a Linux system.

Purpose of Feature

The /proc entries described in this specification can be used to analyze memory usage. Several /proc entries display global usage, i.e. whether memory is free or is being used for any purpose. The rest of the /proc entries show memory usage by specific processes.

Together, these /proc entries make it possible to determine if and how page frames are being used both globally and per process. If a page frame is being used by a process, it can also be determined whether that page is used only by that process (private) or shared with other processes.

Feature Requirements

- The /proc tools must make it possible to get an accurate accounting of all system memory pages required by a process, and which and how many are currently used by the process.
- The tools must make it possible to determine which, and how many, memory pages are program text, program data, and shared library text and data.

- The tools must make it possible to determine which, and how many, memory pages are shared with other processes, and which and how many are private.
- The tools must make it possible to determine how many pages are reserved from process usage, i.e. are reserved for kernel text and data.

High Level Design

Global Memory Usage

There are four /proc entries that provide global memory usage information.

/proc/meminfo

This is the standard global memory information entry, and provides total, used, and free numbers for memory and swap space, in units of bytes and kilobytes.

Refer to the proc(5) man page (`man 5 proc`) for more information about `/proc/meminfo`.

/proc/nodeinfo

This is a new entry that displays the memory nodes in the system. Systems with physically contiguous memory will have only one node, but NUMA machines and machines with discontiguous memory will have two or more nodes. An example output is:

```
node id    start      end    defalloc    name
-------  --------  --------  --------  --------
   0     10000000  11000000      1     SDRAM0
   1     11000000  11800000      0     SDRAM1
   2     11c00000  12000000      0     SDRAM2
   3     20000000  2002e000      0     SRAM
```

`Start` and `end` are the physical start and end addresses of the memory node. The `defalloc` column will appear if the kernel supports MemoryTypeBasedAllocation, but will be absent if not. The last column displays the name of the node if any.

/proc/memmap

This entry provides a global view of the free and allocated physical memory pages. It provides data in a bit map format, one bit represents a single page frame. A one means the page is allocated, a zero means the page is free.

The /proc entry that measures this memory walks through all nodes, and within each node, walks through all page descriptors. If the page is not reserved for kernel use, and has a page usage count of zero, it is a free page and the corresponding bit in the bitmap is cleared, otherwise it is in use and the bit is set.

This bit map is displayed as an array of 32-bit unsigned integers in hexadecimal format. Eight unsigned integers are displayed per line, so each line represents 8*32 = 256 pages.

The `/proc/memmap` entry will appear by enabling the kernel config option `CONFIG_MEMORY_ACCOUNTING`.

/proc/iomem

This entry lists the range of physical addresses reserved by the kernel for its code and data:

```
% cat /proc/iomem | grep Kernel
```

```
00100000-0022d98a : Kernel code
0022d98b-00298b7f : Kernel data
```

In the above example, 301 pages are reserved for kernel code, and 107 pages are reserved for kernel data.

Refer to the proc(5) man page (`man 5 proc`) for more information about `/proc/iomem`.

Per-Process Memory Usage

There are five /proc entries that provide process memory usage information.

/proc/[pid]/maps

This is the standard process memory map entry. A single line is displayed for each of the processes virtual memory regions (VMA). A sample output is (in this case the map for a `bash` shell):

```
00008000-0008d000 r-xp 00000000 00:09 2524773    /bin/bash
00094000-0009a000 rw-p 00084000 00:09 2524773    /bin/bash
0009a000-000c7000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 00:09 2359683    /lib/ld-2.3.2.so
40016000-40017000 rw-p 00000000 00:00 0
4001d000-4001f000 rw-p 00015000 00:09 2359683    /lib/ld-2.3.2.so
4001f000-40054000 r-xp 00000000 00:09 2360362    /lib/libncurses.so.5.2
40057000-40064000 rw-p 00030000 00:09 2360362    /lib/libncurses.so.5.2
40064000-40067000 rw-p 00000000 00:00 0
40067000-40069000 r-xp 00000000 00:09 2360339    /lib/libdl-2.3.2.so
4006f000-40072000 rw-p 00000000 00:09 2360339    /lib/libdl-2.3.2.so
40072000-4018b000 r-xp 00000000 00:09 2360334    /lib/libc-2.3.2.so
40192000-40197000 rw-p 00118000 00:09 2360334    /lib/libc-2.3.2.so
40197000-40199000 rw-p 00000000 00:00 0
40199000-401a2000 r-xp 00000000 00:09 2360338    /lib/libnss_files-2.3.2.so
401a9000-401aa000 rw-p 00008000 00:09 2360338    /lib/libnss_files-2.3.2.so
bfffb000-c0000000 rwxp ffffc000 00:00 0
```

The VMA information displayed per line is:

- virtual start address in hexadecimal
- virtual end address in hexadecimal
- permissions
- file byte offset (if the VMA maps a file) in hexadecimal
- device major:minor numbers of the block device containing the file being mapped (if the VMA maps a file)
- file inode number (if the VMA maps a file)
- file name (if the VMA maps a file)

Refer to the proc(5) man page (`man 5 proc`) for more information about `/proc/[pid]/maps`.

/proc/[pid]/memmap

This entry displays detailed information about whether pages are resident (allocated and mapped) within each region of the process. Just as in `/proc/[pid]/maps` above, one line is displayed for each VMA of the process.

Each line displays two characters for each page-size chunk of the region. If a page is not yet resident for that chunk, a space and dash " -" is displayed, otherwise the usage counter of the resident page is displayed in decimal, as in " 2" or "10".

The /proc entry code accomplishes this by walking through each regions page table entries (pte) to determine if a physical page is resident for that area, determined by the "present" bit in the pte. If so, the pte is converted to a page descriptor and the page usage counter is printed, otherwise " -" is printed.

A sample output, corresponding to the `bash` shell's memory map displayed above, is:

```
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 - 3 3 3 3 ...
 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
1010101010101010111110101110101010111111010
 1
 1 1
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 - - - - - - - - - - - - - - - - - 3 - - - ...
 - - - - 1 1 1 1 1 1 1 1 1
 - - 1
 4 4
 - 1 1
11111111111111111111111111111111111111111111111 - - - - - - - - - 4 4 ...
 1 1 1 1 1
 1 1
 6 6 6 - - 3 - - 6
 1
 1 - 1 1 1
```

Notice the first line, which corresponds to the text of `bash`. Notice that most of this region has resident pages, and most of the resident pages have a usage count of 3. This means that there are currently 2 `bash` shells running. Two of the three usage counts represent the page being mapped in the two `bash` processes page tables, and the remaining count is due to the fact that the page has been inserted into the kernel's page cache, which counts once towards a usage.

Notice also that the 4th line shows that the entire text of ld.so has resident pages, and those pages have high (10 or 11) usage counts. This reflects the fact that ld.so is used by virtually all programs for mapping their shared libraries. The same goes for libc text (the 12th line).

(The "..." characters are artificially inserted here to signify that the line continues but has been omitted for brevity).

The `/proc/[pid]/memmap` entry will appear by enabling the kernel config option `CONFIG_MEMORY_ACCOUNTING`.

/proc/[pid]/nodemap

The format of this entry is identical to `/proc/[pid]/memmmap`, except that, instead of displaying the page usage count of resident pages, the entry displays the memory node ID in which the page resides. For contiguous memory systems with a single node, the node ID's of all resident pages will read 0.

This entry is mostly used to verify the proper behavior of MTA enabled systems (see MemoryTypeBasedAllocation).

The `/proc/[pid]/nodemap` entry will appear by enabling the kernel config option `CONFIG_MEMORY_ACCOUNTING`.

/proc/[pid]/statrm

This entry summarizes the information provided by `/proc/[pid]/memmap`, and combines it with some of the information provided by `/proc/[pid]/maps`. As in `/proc/[pid]/maps`, it displays one line for each VMA of the process. On each line, the following fields are displayed:

- virtual start address in hexadecimal (same as `/proc/[pid]/maps`)
- total number of pages in the region in decimal
- of the total number of pages in the region, the number that are resident
- of the resident pages, the number which are shared with other processes, i.e. which have a usage count greater than one
- of the resident pages, the number which are dirty (modified)
- permissions (same as `/proc/[pid]/maps`)
- file byte offset in hex (same as `/proc/[pid]/maps`)
- file name (same as `/proc/[pid]/maps`)

A sample output, corresponding to the `bash` shell's `/proc/[pid]/maps` and `/proc/[pid]/memmap` displayed above, is:

```
00008000    133    115    115      0 r-xp 00000000 /bin/bash
00094000      6      6      0      6 rw-p 00084000 /bin/bash
0009a000     45     45      0     45 rwxp 00000000
40000000     22     22     22      0 r-xp 00000000 /lib/ld-2.3.2.so
40016000      1      1      0      1 rw-p 00000000
4001d000      2      2      0      2 rw-p 00015000 /lib/ld-2.3.2.so
4001f000     53     29     29      0 r-xp 00000000 /lib/libncurses.so.5.2
40057000     13      9      0      9 rw-p 00030000 /lib/libncurses.so.5.2
40064000      3      1      0      1 rw-p 00000000
40067000      2      2      2      0 r-xp 00000000 /lib/libdl-2.3.2.so
4006f000      3      2      0      2 rw-p 00000000 /lib/libdl-2.3.2.so
40072000    281    106    106      0 r-xp 00000000 /lib/libc-2.3.2.so
40192000      5      5      0      5 rw-p 00118000 /lib/libc-2.3.2.so
40197000      2      2      0      2 rw-p 00000000
40199000      9      5      5      0 r-xp 00000000 /lib/libnss_files-2.3.2.so
401a9000      1      1      0      1 rw-p 00008000 /lib/libnss_files-2.3.2.so
bfffb000      5      4      0      4 rwxp ffffc000
```

/proc/[pid]/statm

This is a standard entry that summarizes the information provided by `/proc/[pid]/statrm`. It displays only 7 decimal numbers:

- the total number of non-zero pte's in the process memory
- the total number of resident pages in the process memory
- of the resident pages, the number which are shared with other processes, i.e. which have a usage count greater than one
- of the resident pages, the number which are from text regions
- of the resident pages, the number which are from library regions
- of the resident pages, the number which are from data (including stack) regions
- of the resident pages, the number which are dirty (modified)

The second, third, and seventh numbers are a sum of the corresponding columns from `/proc/[pid]/statrm`.

A sample output, corresponding to the `bash` shell's `/proc/[pid]/statrm` displayed above, is:

```
357 357 279 121 0 236 78
```

Refer to the proc(5) man page (`man 5 proc`) for more information about `/proc/[pid]/statm`.

Acceptance Criteria

1. Verify that an accurate accounting can be made of the current and maximum memory usage of a process, using the above /proc tools.
2. Verify that the above tools can be used to accurately account for all memory pages currently used by all running processes, taking into account which pages are shared among processes and which are private.
3. Verify that the above tools can be used to accurately account for all memory pages currently reserved for kernel use.

Use Cases

In this example, we will calculate how much resident memory is being used by all currently running `bash` shells (this example was run on the OMAP1510 Innovator).

First, list all the loaded processes with `ps ax`, greping for "bash":

```
root@10.0.1.95:~# ps ax | grep bash
  102 ttyS0    S      0:00 -bash
  113 pts/0    S      0:00 -bash
```

Now let’s list the resident page counts:

```
root@10.0.1.95:~# cat /proc/102/statrm
00008000    133    115    115      0 r-xp 00000000 /bin/bash
00094000      6      6      0      6 rw-p 00084000 /bin/bash
0009a000     45     45      0     45 rwxp 00000000
40000000     22     22     22      0 r-xp 00000000 /lib/ld-2.3.2.so
40016000      1      1      0      1 rw-p 00000000
4001d000      2      2      0      2 rw-p 00015000 /lib/ld-2.3.2.so
4001f000     53     29     29      0 r-xp 00000000 /lib/libncurses.so.5.2
40057000     13      9      0      9 rw-p 00030000 /lib/libncurses.so.5.2
40064000      3      1      0      1 rw-p 00000000
40067000      2      2      2      0 r-xp 00000000 /lib/libdl-2.3.2.so
4006f000      3      2      0      2 rw-p 00000000 /lib/libdl-2.3.2.so
40072000    281    106    106      0 r-xp 00000000 /lib/libc-2.3.2.so
40192000      5      5      0      5 rw-p 00118000 /lib/libc-2.3.2.so
40197000      2      2      0      2 rw-p 00000000
40199000      9      5      5      0 r-xp 00000000 /lib/libnss_files-2.3.2.so
401a9000      1      1      0      1 rw-p 00008000 /lib/libnss_files-2.3.2.so
bfffb000      5      4      0      4 rwxp ffffc000
root@10.0.1.95:~# cat /proc/113/statrm
00008000    133    111    111      0 r-xp 00000000 /bin/bash
00094000      6      6      0      6 rw-p 00084000 /bin/bash
0009a000     40     40      0     40 rwxp 00000000
40000000     22     22     22      0 r-xp 00000000 /lib/ld-2.3.2.so
40016000      1      1      0      1 rw-p 00000000
4001d000      2      2      0      2 rw-p 00015000 /lib/ld-2.3.2.so
4001f000     53     27     27      0 r-xp 00000000 /lib/libncurses.so.5.2
40057000     13      9      0      9 rw-p 00030000 /lib/libncurses.so.5.2
40064000      3      1      0      1 rw-p 00000000
40067000      2      2      2      0 r-xp 00000000 /lib/libdl-2.3.2.so
4006f000      3      2      0      2 rw-p 00000000 /lib/libdl-2.3.2.so
40072000    281    104    104      0 r-xp 00000000 /lib/libc-2.3.2.so
40192000      5      5      0      5 rw-p 00118000 /lib/libc-2.3.2.so
40197000      2      2      0      2 rw-p 00000000
40199000      9      5      5      0 r-xp 00000000 /lib/libnss_files-2.3.2.so
401a9000      1      1      0      1 rw-p 00008000 /lib/libnss_files-2.3.2.so
bfffb000      5      4      0      4 rwxp ffffc000
```

All the text regions are being shared between the two `bash` shells, which accounts for a maximum of 133+22+53+2+281+9 = 500 pages. To calculate how many text pages are currently resident, we take the maximum of the resident page count field from each shared text region. For example, we see that there are 115 resident pages for the first `bash` code, but only 111 for the second. So there are currently 115 resident pages for `bash` code. Following this procedure for all the shared text regions, we see there are 115+22+29+2+106+5 = 279 resident text pages (out of a total of 500).

The remaining regions contain only private pages, and must be counted seperately between the two `bash` shells. Out of a total of 6+45+1+2+13+3+3+5+2+1+5 = 86 pages for each shell, 6+45+1+2+9+1+2+5+2+1+4 = 78 pages are resident in the first shell, and 6+40+1+2+9+1+2+5+2+1+4 = 73 pages are resident in the second shell.

Putting it all together, out of a total of 500+86+86 = 672 pages that could be used by both `bash` shells, currently 279+78+73 = 430 pages are resident.

Additional Information

Known Problems

There are no known problems.

----- Revision r1.4 - 09 Feb 2004 - 18:28 GMT - SteveLongerbeam