Memory Usage

Miguel de Icaza

miguel@gnu.org

Jacob Berkman

jberkman@andrew.cmu.edu

One factor taken into consideration when evaluating any piece of software is its memory usage. This is because of monetary cost (RAM, although inexpensive these days, is not free) and the performance hit that constant swapping yields.

It is very easy to look at information provided by programs such as the standard UNIX **top** command. The purpose of this document is to explain some terms relating to memory usage, and to demonstrate the proper way to determine a program's memory usage.

A Brief Example

The UNIX command "**top**" can be used to show information about running processes. Here is a small excerpt from a Linux computer running GNOME:

```
  PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COM-
MAND
 9410 jberkman   6   0 14384  13M  2908 S       0  0.0 14.9  11:12 xemacs
   94 root      16   0 14752  13M  1304 R       0  3.1 14.2 121:05 X
 8596 jberkman   0   0  4672 4672  3400 S       0  0.0  4.8   0:03 gnome-
help-
 9397 jberkman   1   0  4476 4476  2496 S       0  0.1  4.6   0:19 panel
25286 boc        0   0  4380 4380  3260 S       0  0.0  4.5   0:05 irssi
20751 boc        0   0  4004 4004  2920 S       0  0.0  4.1   0:03 panel
21012 jberkman   0   0  3312 3312  2652 S       0  0.0  3.4   0:00 session-
pro
```

There are four items that refer to memory in this listing. They will be explained in the next section

Memory Terms

When talking about memory usage and the reports you get from the operating system, you need to keep in mind the following terms and their meanings:

SIZE

This is the address space seen by the process. If the process maps 65 megs of memory, the address space of the process will be reported as 65 megs of memory, *even if only 100k of memory is actually used* .

This bit of information is not usually very useful. Unfortunately this is what most people believe is the actual memory usage of a program; it is not.

RSS

This is the "Resident Set Size". This is the amount of memory actually resident on memory. This is a better computation of what is actually loaded in memory.

This does not include any pieces of code that have been "swapped out".

So, for example, if you have a program that uses 100k of memory and the operating system discards or swaps 40k of memory, the RSS will be 60k.

The RSS includes both the memory which is unique to this process and the memory shared with other processes. In most modern Unix systems, a large part of this accounts for the memory used by shared libraries. So it usually includes the amount of memory that is used by other processes for the code that is shared: The libc is usually taken into account.

GNOME uses many shared libraries This causes the applications share more memory with each other, reducing the total amount of memory used.

SHARE

This is the amount of memory from the RSS that is shared with other applications.

%MEM

This is the percent of physical memory used by the process.

A Good Approximation

So, the actual memory used uniquely by an application is RSS - SHARE. If two programs use library "A", then the process would also be "responsible" for A/2 of the memory, but there are many variations (for example, which pieces of the library each program uses) that make this computation difficult.

A good approximation is thus "RSS - SHARE".

Library Usage

A large amount of functionality is availble to GNOME applications as shared libraries. Since GNOME applications share these libraries, the per-application memory cost is decreased.

The file manager in GNOME for example uses the following libraries:

CORBA Libraries

```
libgnorba.so.27 => /opt/gnome/lib/libgnorba.so.27 (0x40007000)
libORBitCosNaming.so.0 =>/opt/gnome/lib/libORBitCosNaming.so.0 (0x40011000)
libORBit.so.0 => /opt/gnome/lib/libORBit.so.0 (0x4001a000)
libIIOP.so.0 => /opt/gnome/lib/libIIOP.so.0 (0x4007c000)
libORBitutil.so.0 => /opt/gnome/lib/libORBitutil.so.0(0x40086000)
```

GNOME Graphical Libraries

```
libgnomeui.so.32 => /opt/gnome/lib/libgnomeui.so.32(0x40089000)
libart_lgpl.so.2 => /opt/gnome/lib/libart_lgpl.so.2(0x4013b000)
libgdk_imlib.so.1 => /opt/gnome/lib/libgdk_imlib.so.1(0x40149000)
```

Session Management Libraries

```
libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x4016a000)
libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x40173000)
```

GTK Libraries

```
libgtk-1.2.so.0 => /usr/local/lib/libgtk-1.2.so.0 (0x40188000)
libgdk-1.2.so.0 => /usr/local/lib/libgdk-1.2.so.0 (0x402a3000)
```

Glib Libraries

```
libgmodule-1.2.so.0 => /usr/local/lib/libgmodule-1.2.so.0(0x402d5000)
libglib-1.2.so.0 => /usr/local/lib/libglib-1.2.so.0(0x403c9000)
```

X11 Libraries

```
libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x402d9000)
libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x402e7000)
```

GNOME Libraries

```
libgnome.so.32 => /opt/gnome/lib/libgnome.so.32 (0x403a3000)
libgnomesupport.so.0 => /opt/gnome/lib/libgnomesupport.so.0(0x403b5000)
```

Standard UNIX Libraries

```
libz.so.1 => /usr/lib/libz.so.1 (0x404a4000)
libdb.so.2 => /lib/libdb.so.2 (0x403bb000)
libdl.so.2 => /lib/libdl.so.2 (0x403ea000)
libm.so.6 => /lib/libm.so.6 (0x4038a000)
libc.so.6 => /lib/libc.so.6 (0x403ff000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x2aaaa000)
```

File Manager Libraries

```
libext2fs.so.2 => /lib/libext2fs.so.2 (0x403ed000)
libcom_err.so.2 => /lib/libcom_err.so.2 (0x403fd000)
```

The "SIZE" reported for this application would reflect the maximum size the program would be using if it used all of those libraries.

The more libraries we use the bigger the SIZE goes, even if this does not reflect the actual memory usage.

Other common miss-understandings

Many systems use the free memory as cache in case the same application is restarted and this memory can be split between various parts of your kernel. Particularly using commands that only report the amount of free memory might led people to believe that there is a leak in the program. If you want to identify leaks correctly you should use the output from the `ps` program to track down memory usage.