Linux uses a virtual memory system (VM), as do all modern operating systems: the virtual memory is larger than the physical memory.

Each process has its own, protected address space. Addresses are virtual and must be translated to and from physical addresses by the kernel whenever a process needs to access memory.

The kernel itself also uses virtual addresses; however the translation can be as simple as an offset depending on the architecture and the type of memory being used.

The kernel allows fair shares of memory to be allocated to every running process, and coordinates when memory is shared among processes. In addition, mapping can be used to link a file directly to a process's virtual address space. Furthermore, certain areas of memory can be be protected against writing and/or code execution.

The **free** utility gives a very terse report on free and used memory in your system:

```
$ free -mt
          total     used     free    shared    buff/cache   available
Mem:      15893     3363      175       788         12354       11399
Swap:      8095        0     8095
Total:    23989     3363     8271
```

where the options cause the output to be expressed in MB's. See **man free** to see possible options.

This system has 16 GB of RAM and a 8 GB swap partition. At the moment, this snapshot was taken the system was pretty inactive and not doing all that much. Yet, the amount of memory being used is appreciable (if you include the memory assigned to the cache).

However, a lot of the memory being used is in the page cache, most of which is being used to cache the contents of files that have recently been accessed. If this cache is released, the memory usage will decrease significantly. This can be done by doing (as root user):

```
$ sudo su
# echo 3 > /proc/sys/vm/drop_caches
# exit
$ free -mt
          total      used      free     shared     buff/cache    available
Mem:      15893      3370     11103        788           1419        11419
Swap:      8095         0      8095
Total:    23989      3370     19199
```

If we had only wanted to drop the page cache, we would have echoed a 1, not a 3; we have also dropped the dentry and inode caches, which is why the freed memory is more than that released from the page cache.

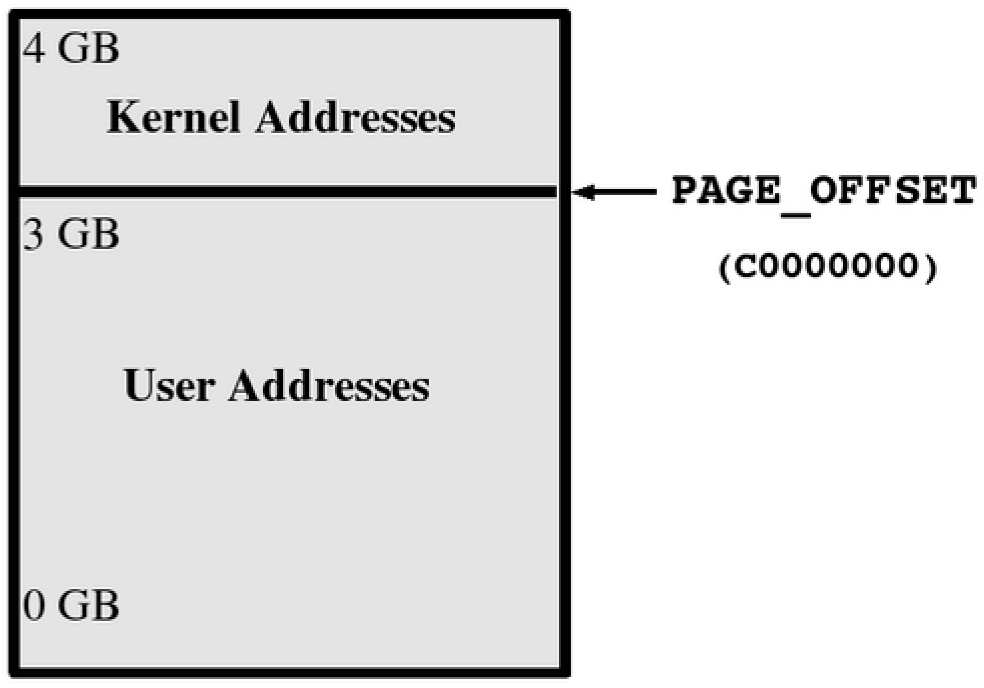
A more detailed look can be obtained by looking at **/proc/meminfo**:

```
$ cat /proc/meminfo

MemTotal:       16275064 kB
MemFree:        11059060 kB
MemAvailable:   11525932 kB
Buffers:           30416 kB
Cached:          1598188 kB
SwapCached:            0 kB
Active:          3880768 kB
Inactive:        1105144 kB
Active(anon):    3295948 kB
Inactive(anon):   994524 kB
Active(file):     584820 kB
Inactive(file):   110620 kB
Unevictable:        3596 kB
Mlocked:            3596 kB
SwapTotal:       8290300 kB
SwapFree:        8290300 kB
Dirty:               416 kB
Writeback:             0 kB
AnonPages:       3360760 kB
Mapped:           859028 kB
Shmem:           1007708 kB
Slab:              94048 kB
SReclaimable:      46268 kB
SUnreclaim:        47780 kB
KernelStack:       14272 kB
PageTables:        63688 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    16427832 kB
Committed_AS:   11194528 kB
VmallocTotal:   34359738367 kB
VmallocUsed:           0 kB
VmallocChunk:          0 kB
AnonHugePages:    980992 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
CmaTotal:              0 kB
CmaFree:               0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:      175384 kB
DirectMap2M:     8087552 kB
DirectMap1G:     9437184 kB
```

The output will depend somewhat on kernel version, and you should not write scripts that overly depend on certain fields being in this file.

In the following diagram (for 32-bit platforms), the first 3 GB of virtual addresses are used for user-space memory and the upper GB is used for kernel-space memory. Other architectures have the same setup, but differing values for **PAGE_OFFSET**; for 64-bit platforms, the value is in the stratosphere.



While Linux permits up to 64 GB of memory to be used on 32-bit systems, the limit per process is a little less than 3 GB. This is because there is only 4 GB of address space (i.e. it is 32-bit limited) and the topmost GB is reserved for kernel addresses. The little is somewhat less than 3 GB because of some address space being reserved for memory-mapped devices.

It is important to remember that applications do not write directly to storage media such as disks; they interface with the virtual memory system and data blocks written are generally first placed into cache or buffers, and then are flushed to disk when it is either convenient or necessary. Thus, in most systems, more memory is used in this buffering/caching layer than for direct use by applications for other purposes.

For a comprehensive review, see Ulrich Drepper's article "What Every Programmer Should Know About Memory". This covers many issues in depth, such as proper use of cache, alignment, NUMA, virtualization, etc.

Managing the memory on 32-bit machines with large amounts of memory (especially over 4 GB) is far more complex than it is in 64-bit systems.

It is hard to think of a good reason to be acquiring purely 32-bit hardware anymore for use as heavy iron; there is still plenty of use for 32-bit systems in the embedded world, etc., but there memory is not expected to be large enough to complicate memory management.

Of course, you may be running 32-bit applications on 64-bit hardware, and that may lead occasionally to questions which may be subtle.

However, to keep things simple, we will not focus on 32-bit systems with a lot of memory, as they are more and more becoming dinosaurs.

✓ Complete Go to next item