Memory Management

Christopher Rodriguez

Colorado State University Global

22SC-CSC300-1: Operating Systems and Architecture

Dr. Jonathan Vanover

2022-06-19

Memory Management

It was the Atlas supervisor in 1959, and then more broadly IBM's OS 360 in 1965, that introduced the concept of “multitasking”—executing (or *seeming* to execute) many processes simultaneously on one machine—to the world of computer science (Denning, 2017, p. 32). With that introduction came a new question: What is the best way to divide up a system's memory? This paper looks over the basics of memory, memory management, and paging.

Physical, Logical, Virtual, and MMU

Memory is basically temporary storage for the computer, usually an assembly of storage cells each able to store the same number of bits. *RAM* is volatile (all stored data gets lost with a loss of power) memory that is both readable and writable. *Memory Locations* are the specific physical cells used to store the data, whereas *Memory Addresses* are the CPU's way of designating a location in memory (which is not (usually) the same as the location itself) (Crow, 1996, p. 3).

These last two terms are important: Because Addresses don't correspond to specific Locations when it comes to memory, an Address can point to whatever Location in memory it needs to in order to work. And, because of this, it is possible for a program to operate in a different location in memory than it believes it actually is (Crow, 1996, p. 12).

Physical memory is the hardware inside the computer that is used to provide the RAM for the system. *Logical* memory is the addresses provided by the CPU to the programs and data the system is using, regardless of the true memory locations they map to. And *Virtual* memory is one step further, where the addresses map to artificial locations that get translated to RAM or data on disk, enlarging the memory available to a system by allowing the CPU to use more memory than it actually has (Alverti et al., 2020, p. 516), thus potentially keeping more parts of that memory *contiguous*. One example of virtual memory is with *swap space* on GNU/Linux, where either an entire partition or a file on a specific partition can be used as a part of the virtual memory a system can utilize.

COMDEX, Lotus, and Intel eventually standardized a smaller section of memory that each of these addresses might refer to—the “page”, which was traditionally 4 KiB but is now much larger—with their “Expanded Memory Specification”, or “EMS 3.0” (Crow, 1996, p. 7). This process of translating between virtual addresses, pages, and physical memory is facilitated by a dedicated piece of hardware: the *Memory Management Unit*, or “MMU” (Oliveri & Balzarotti, 2022, 1:4). This combination of separate ways to refer to a machine's memory allows for interesting and useful methods for the allocation and management of the memory a system actually uses, and the MMU is the final step for enabling this to work.

Allocation with Respect to Contiguity

The translation between these virtual or logical addresses and the physical address of the actual hardware imposes a not-insignificant amount of overhead (and this is doubly true if the Operating System is a Virtualized guest running inside a host OS) (Alverti et al., 2020, p. 516). Therefore, it is worthwhile to try to keep all memory as *contiguous*—that is, allocated as one continuous block of memory—as possible during allocation. There have been many ways memory allocation has been attempted over the years, though the solutions can be divided into two rough groups: “eager” paging, and “ranger” paging.

With “eager” paging, more than a single page in memory is allocated in advance for a specific task. The original cap was around 2MB, however that limit has long been passed due to the sheer size of the data we process generally today. The downside to this choice is that it wastes resources at a *very* high rate—on x86_64 based systems, the initially-supported page sizes are 512GB and 1024GB (Alverti et al., 2020, p. 516). This exacerbates the downsides immensely: huge amounts of space is wasted and it becomes harder to find large enough free blocks the longer the system is running.

In contrast, in “ranger” paging—where applications are loaded without consideration to contiguity—uses “asynchronous defragmentation”: the scattered pages are migrated into a contiguous region of memory when not directly in use (Alverti et al., 2020, p. 515), during otherwise-idle

CPU time. Apart from the obvious downside of “double touching” the data being loaded into memory, this can also cause excessive address translation time due to the initially fragmented, non-contagious nature of the data being loaded.

A Brief Note on Paging

There are two main approaches to paging used in today's world, centered around different ways of storing the pages and their data for later use by the system: “radix trees”, and “inverted page tables” (Oliveri & Balzarotti, 2022, 1:4).

With radix trees, data is stored hierarchically: Starting at the root, which is stored in a special register for both the operating system and the MMU, directory tables branch out to eventually point to the pages' physical location in the system, allowing the MMU to essentially do a search on this tree to translate a virtual address to a physical one. To contrast, with inverted page tables, the data structure is a hash table instead: The OS and MMU use the same hashing function to create indexes for this table and use these hashes as keys to access the data stored in a specific page (Oliveri & Balzarotti, 2022, 1:5).

It's important to mention that the above two groups are largely generalized, and not an indication that memory management is a “this or that” style problem. As mentioned by Oliveri and Balzarotti (2022), there is a large amount of complexity and variability in the way that the above are implemented across different architectures (p. 1:12). This paper is not meant to be comprehensive, but an overview of high-level concepts.

Conclusion

Notwithstanding the above disclaimer, the information presented in this paper is illustrative of three key concepts regarding memory management today: First, the memory management system is still largely controlled by hardware, as the MMU and OS need to agree on a scheme for storing the pages memory is split into. Second, each and every memory access involves many different parts of a computer, both hardware and software, that can vary greatly across architecture and operating system. And third, while the amount of memory available to a system has grown significantly in the last few decades, any method of improving the allocation, management, and effective use of memory in computing systems will have wide-reaching effects for the future of the field as a whole.

References

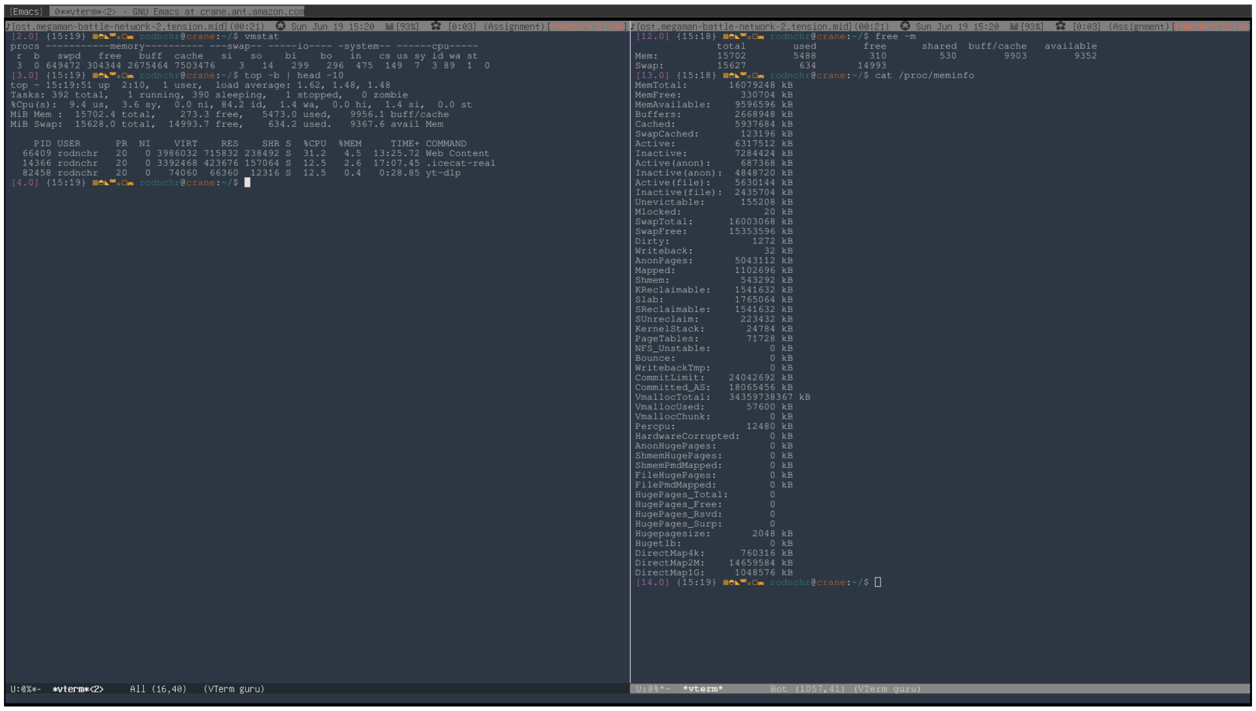
Alverti, C., Psomadakis, S., Karakostas, V., Gandhi, J., Nikas, K., Goumas, G., & Koziris, N. (2020, September 23). Enhancing and exploiting contiguity for fast memory virtualization. In C.-J. Wu, L. Eeckhout, A. Naithani, & K. Lakshminarasimhan (Eds.), *Isca '20: Proceedings of the ACM/IEEE 47th annual international symposium on computer architecture* (pp. 515–528). IEEE Press. https://doi.org/10.1109/ISCA45697.2020.00050

Crow, J. (1996). The MS-DOS memory environment (K.-W. Wong, Ed.). *ACM SIGICE Bulletin*, *21*(3), 2–16. https://doi.org/10.1145/226036.226037

Denning, P. J. (2017). Multitasking without thrashing. *Communications of the ACM*, *60*(9), 32–34. https://doi.org/10.1145/3126494

Oliveri, A., & Balzarotti, D. (2022, March 30). *In the land of mmus: Multiarchitecture os-agnostic virtual memory forensics* (N. Li, Ed.) [AAM]. Association for Computing Machinery. https://doi.org/https://doi.org/10.1145/3528102

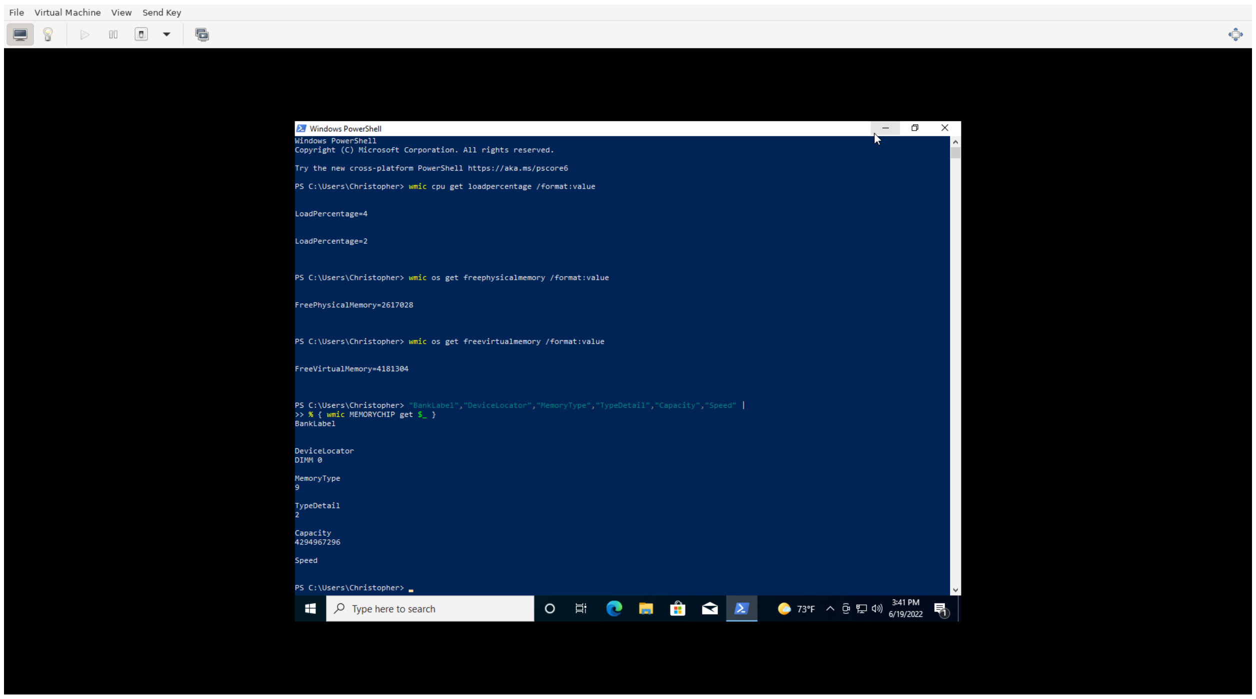
List of Figures



```
[Emacs]  0**vterm*<2> - GNU Emacs at crane.ant.amazon.com
♪[ost.megaman-battle-network-2.tension.mid](00:21)   Sun Jun 19 15:20   [93%]   [0:03]  (Assignment)[Pomodoro~21:46]
[2.0] {15:19} rodnchr@crane:~/$ vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 3  0 649472 304344 2675464 7503476    3   14   299   296  475  149  7  3 89  1  0
[3.0] {15:19} rodnchr@crane:~/$ top -b | head -10
top - 15:19:51 up  2:10,  1 user,  load average: 1.62, 1.48, 1.48
Tasks: 392 total,   1 running, 390 sleeping,   1 stopped,   0 zombie
%Cpu(s):  9.4 us,  3.6 sy,  0.0 ni, 84.2 id,  1.4 wa,  0.0 hi,  1.4 si,  0.0 st
MiB Mem :  15702.4 total,    273.3 free,   5473.0 used,   9956.1 buff/cache
MiB Swap:  15628.0 total,  14993.7 free,    634.2 used.   9367.6 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  66409 rodnchr   20   0 3986032 715832 238492 S  31.2   4.5  13:25.72 Web Content
  14366 rodnchr   20   0 3392468 423676 157064 S  12.5   2.6  17:07.45 .icecat-real
  82458 rodnchr   20   0   74060  66360  12316 S  12.5   0.4   0:28.85 yt-dlp
[4.0] {15:19} rodnchr@crane:~/$

U:@%*-  *vterm*<2>     All (16,40)    (VTerm guru)
```

```
♪[ost.megaman-battle-network-2.tension.mid](00:21)   Sun Jun 19 15:20   [93%]   [0:03]  (Assignment)[Pomodoro~21:46]
[12.0] {15:18} rodnchr@crane:~/$ free -m
              total        used        free      shared  buff/cache   available
Mem:          15702        5488         310         530        9903        9352
Swap:         15627         634       14993
[13.0] {15:18} rodnchr@crane:~/$ cat /proc/meminfo
MemTotal:       16079248 kB
MemFree:          330704 kB
MemAvailable:    9596596 kB
Buffers:         2668948 kB
Cached:          5937684 kB
SwapCached:       123196 kB
Active:          6317512 kB
Inactive:        7284424 kB
Active(anon):     687368 kB
Inactive(anon):  4848720 kB
Active(file):    5630144 kB
Inactive(file):  2435704 kB
Unevictable:      155208 kB
Mlocked:              20 kB
SwapTotal:      16003068 kB
SwapFree:       15353596 kB
Dirty:              1272 kB
Writeback:            32 kB
AnonPages:       5043112 kB
Mapped:          1102696 kB
Shmem:            543292 kB
KReclaimable:    1541632 kB
Slab:            1765064 kB
SReclaimable:    1541632 kB
SUnreclaim:       223432 kB
KernelStack:       24784 kB
PageTables:        71728 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    24042692 kB
Committed_AS:   18065456 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       57600 kB
VmallocChunk:          0 kB
Percpu:            12480 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:         0 kB
FilePmdMapped:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      760316 kB
DirectMap2M:    14659584 kB
DirectMap1G:     1048576 kB
[14.0] {15:19} rodnchr@crane:~/$

U:@%*-  *vterm*        Bot (1057,41) (VTerm guru)
```

Figure 1: Checking Memory Usage on GNU/Linux



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Christopher> wmic cpu get loadpercentage /format:value


LoadPercentage=4


LoadPercentage=2



PS C:\Users\Christopher> wmic os get freephysicalmemory /format:value


FreePhysicalMemory=2617028



PS C:\Users\Christopher> wmic os get freevirtualmemory /format:value


FreeVirtualMemory=4181304



PS C:\Users\Christopher> "BankLabel","DeviceLocator","MemoryType","TypeDetail","Capacity","Speed" |
>> % { wmic MEMORYCHIP get $_ }
BankLabel


DeviceLocator
DIMM 0

MemoryType
9

TypeDetail
2

Capacity
4294967296

Speed


PS C:\Users\Christopher> _
```

Figure 2: Checking Memory Usage on Windows 10