Paging

In computer operating systems, **paging** is a memory management scheme by which a computer stores and retrieves data from secondary storage[a] for use in main memory.[1] In this scheme, the operating system retrieves data from secondary storage in same-size blocks called *pages*. Paging is an important part of virtual memory implementations in modern operating systems, using secondary storage to let programs exceed the size of available physical memory.

For simplicity, main memory is called "RAM" (an acronym of "random-access memory") and secondary storage is called "disk" (a shorthand for "hard disk drive"), but the concepts do not depend on whether these terms apply literally to a specific computer system.

Contents

External links

History

Ferranti introduced paging on the Atlas, but the first mass-market memory pages were concepts in computer architecture, regardless of whether a page moved between RAM and disk.[2][3] For example, on the PDP-8, 7 of the instruction bits comprised a memory address that selected one of 128 (2^7) words. This zone of memory was called a *page*. This use of the term is now rare. In the 1960s, swapping was an early virtual memory technique. An entire program would be "swapped out" (or "rolled out") from RAM to disk, and another one would be *swapped in* (or *rolled in*).[4][5] A swapped-out program would be current but its execution would be suspended while its RAM was in use by another program.

A program might include multiple overlays that occupy the same memory at different times. Overlays are not a method of paging RAM to disk but merely of minimizing the program's RAM use. Subsequent architectures used memory segmentation, and individual program segments became the units exchanged between disk and RAM. A segment was the program's entire code segment or data segment, or sometimes other large data structures. These segments had to be contiguous when resident in RAM, requiring additional computation and movement to remedy fragmentation.[6]

The invention of the page table let the processor operate on arbitrary pages anywhere in RAM as a seemingly contiguous logical address space. These pages became the units exchanged between disk and RAM.

Page faults

When a process tries to reference a page not currently present in RAM, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system. The operating system must:

1. Determine the location of the data on disk.
2. Obtain an empty page frame in RAM to use as a container for the data.
3. Load the requested data into the available page frame.
4. Update the page table to refer to the new page frame.
5. Return control to the program, transparently retrying the instruction that caused the page fault.

When all page frames are in use, the operating system must select a page frame to reuse for the page the program now needs. If the evicted page frame was dynamically allocated by a program to hold data, or if a program modified it since it was read into RAM (in other words, if it has become "dirty"), it must be written out to disk before being freed. If a program later references the evicted page, another page fault occurs and the page must be read back into RAM.

The method the operating system uses to select the page frame to reuse, which is its page replacement algorithm, is important to efficiency. The operating system predicts the page frame least likely to be needed soon, often through the least recently used (LRU) algorithm or an algorithm based on the program's working set. To further increase responsiveness, paging systems may predict which pages will be needed soon, preemptively loading them into RAM before a program references them.

Page replacement techniques

Demand paging

When pure demand paging is used, pages are loaded only when they are referenced. A program from a memory mapped file begins execution with none of its pages in RAM. As the program commits page faults, the operating system copies the needed pages from a file, e.g., memory-mapped file, paging file, or a swap partition containing the page data into RAM.

Anticipatory paging

This technique, sometimes also called *swap prefetch,* predicts which pages will be referenced soon, to minimize future page faults. For example, after reading a page to service a page fault, the operating system may also read the next few pages even though they are not yet needed (a prediction using locality of reference). If a program ends, the operating system may delay freeing its pages, in case the user runs the same program again.

Free page queue, stealing, and reclamation

The free page queue is a list of page frames that are available for assignment. Preventing this queue from being empty minimizes the computing necessary to service a page fault. Some operating systems periodically look for pages that have not been recently referenced and then free the page frame and add it to the free page queue, a process known as "page stealing". Some operating systems[b] support *page reclamation*; if a program commits a page fault by referencing a page that was stolen, the operating system detects this and restores the page frame without having to read the contents back into RAM.

Pre-cleaning

The operating system may periodically pre-clean dirty pages: write modified pages back to disk even though they might be further modified. This minimizes the amount of cleaning needed to obtain new page frames at the moment a new program starts or a new data file is opened, and improves responsiveness. (Unix operating systems periodically use sync to pre-clean all dirty pages; Windows operating systems use "modified page writer" threads.)

Thrashing

After completing initialization, most programs operate on a small number of code and data pages compared to the total memory the program requires. The pages most frequently accessed are called the working set.

When the working set is a small percentage of the system's total number of pages, virtual memory systems work most efficiently and an insignificant amount of computing is spent resolving page faults. As the working set grows, resolving page faults remains manageable until the growth reaches a critical point. Then faults go up dramatically and the time spent resolving them overwhelms time spent on the computing the program was written to do. This condition is referred to as thrashing. Thrashing occurs on a program that works with huge data structures, as its large working set causes continual page faults that drastically slow down the system. Satisfying page faults may require freeing pages that will soon have to be re-read from disk. "Thrashing" is also used in contexts other than virtual memory systems; for example, to describe cache issues in computing or silly window syndrome in networking.

A worst case might occur on VAX processors. A single MOVL crossing a page boundary could have a source operand using a displacement deferred addressing mode, where the longword containing the operand address crosses a page boundary, and a destination operand using a displacement deferred addressing mode, where the longword containing the operand address crosses a page boundary, and the source and destination could both cross page boundaries. This single instruction references ten pages; if not all are in RAM, each will cause a page fault. As each fault occurs the operating system needs to go through the extensive memory management routines perhaps causing multiple I/Os which might including writing other process pages to disk and reading pages of the active process from disk. If the operating system could not allocate ten pages to this program, then remedying the page fault would discard another page the instruction needs, and any restart of the instruction would fault again.

To decrease excessive paging and resolve thrashing problems, a user can increase the number of pages available per program, either by running fewer programs concurrently or increasing the amount of RAM in the computer.

Sharing

In multi-programming or in a multi-user environment, many users may execute the same program, written so that its code and data are in separate pages. To minimize RAM use, all users share a single copy of the program. Each process's page table is set up so that the pages that address code point to the single shared copy, while the pages that address data point to different physical pages for each process.

Different programs might also use the same libraries. To save space, only one copy of the shared library is loaded into physical memory. Programs which use the same library have virtual addresses that map to the same pages (which contain the library's code and data). When programs want to modify the library's code, they use copy-on-write, so memory is only allocated when needed.

Shared memory is an efficient way of communication between programs. Programs can share pages in memory, and then write and read to exchange data.

Implementations

Ferranti Atlas

The first computer to support paging was the supercomputer Atlas,[7][8][9] jointly developed by Ferranti, the University of Manchester and Plessey in 1963. The machine had an associative (content-addressable) memory with one entry for each 512 word page. The Supervisor[10] handled non-equivalence interruptions[c] and managed the transfer of pages between core and drum in order to provide a one-level store[11] to programs.

Microsoft Windows

Windows 3.x and Windows 9x

Paging has been a feature of Microsoft Windows since Windows 3.0 in 1990. Windows 3.x creates a hidden file named `386SPART.PAR` or `WIN386.SWP` for use as a swap file. It is generally found in the root directory, but it may appear elsewhere (typically in the WINDOWS directory). Its size depends on how much swap space the system has (a setting selected by the user under Control Panel → Enhanced under "Virtual Memory"). If the user moves or deletes this file, a blue screen will appear the next time Windows is started, with the error message "The permanent swap file is corrupt". The user will be prompted to choose whether or not to delete the file (whether or not it exists).

Windows 95, Windows 98 and Windows Me use a similar file, and the settings for it are located under Control Panel → System → Performance tab → Virtual Memory. Windows automatically sets the size of the page file to start at 1.5× the size of physical memory, and expand up to 3× physical memory if necessary. If a user runs memory-intensive applications on a system with low physical memory, it is preferable to manually set these sizes to a value higher than default.

Windows NT

The file used for paging in the Windows NT family is `pagefile.sys`. The default location of the page file is in the root directory of the partition where Windows is installed. Windows can be configured to use free space on any available drives for pagefiles. It is required, however, for the boot partition (i.e., the drive containing the Windows directory) to have a pagefile on it if the system is configured to write either kernel or full memory dumps after a Blue Screen of Death. Windows uses the paging file as temporary storage for the memory dump. When the system is rebooted, Windows copies the memory dump from the pagefile to a separate file and frees the space that was used in the pagefile.[12]

Fragmentation

In the default configuration of Windows, the pagefile is allowed to expand beyond its initial allocation when necessary. If this happens gradually, it can become heavily fragmented which can potentially cause performance problems.[13] The common advice given to avoid this is to set a single "locked" pagefile size so that Windows will not expand it. However, the pagefile only expands when it has been filled, which, in its default configuration, is 150% the total amount of physical memory. Thus the total demand for pagefile-backed virtual memory must exceed 250% of the computer's physical memory before the pagefile will expand.

The fragmentation of the pagefile that occurs when it expands is temporary. As soon as the expanded regions are no longer in use (at the next reboot, if not sooner) the additional disk space allocations are freed and the pagefile is back to its original state.

Locking a pagefile size can be problematic if a Windows application requests more memory than the total size of physical memory and the pagefile, leading to failed requests to allocate memory that may cause applications and system processes to fail. Also, the pagefile is rarely read or written in sequential order, so the performance advantage of having a completely sequential page file is minimal. However, a large pagefile generally allows use of memory-heavy applications, with no penalties beside using more disk space. While a fragmented pagefile may not be an issue by itself, fragmentation of a variable size page file will over time create a number of fragmented blocks on the drive, causing other files to become fragmented. For this reason, a fixed-size contiguous pagefile is better, providing that the size allocated is large enough to accommodate the needs of all applications.

The required disk space may be easily allocated on systems with more recent specifications (i.e. a system with 3 GB of memory having a 6 GB fixed-size pagefile on a 750 GB disk drive, or a system with 6 GB of memory and a 16 GB fixed-size pagefile and 2 TB of disk space). In both examples the system is using about 0.8% of the disk space with the pagefile pre-extended to its maximum.

Defragmenting the page file is also occasionally recommended to improve performance when a Windows system is chronically using much more memory than its total physical memory. This view ignores the fact that, aside from the temporary results of expansion, the pagefile does not become fragmented over time. In general, performance concerns related to pagefile access are much more effectively dealt with by adding more physical memory.

Unix and Unix-like systems

Unix systems, and other Unix-like operating systems, use the term "swap" to describe both the act of moving memory pages between RAM and disk, and the region of a disk the pages are stored on. In some of those systems, it is common to dedicate an entire partition of a hard disk to swapping. These partitions are called *swap partitions*. Many systems have an entire hard drive dedicated to swapping, separate from the data drive(s), containing only a swap partition. A hard drive dedicated to swapping is called a "swap drive" or a "scratch drive" or a "scratch disk". Some of those systems only support swapping to a swap partition; others also support swapping to files.

Linux

The Linux kernel supports a virtually unlimited number of swap backends (devices or files), and also supports assignment of backend priorities. When the kernel needs to swap pages out of physical memory, it uses the highest-priority backend with available free space. If multiple swap backends are assigned the same priority, they are used in a round-robin fashion (which is somewhat similar to RAID 0 storage layouts), providing improved performance as long as the underlying devices can be efficiently accessed in parallel.[14]

Swap files and partitions

From the end-user perspective, swap files in versions 2.6.x and later of the Linux kernel are virtually as fast as swap partitions; the limitation is that swap files should be contiguously allocated on their underlying file systems. To increase performance of swap files, the kernel keeps a map of where they are placed on underlying devices and accesses them directly, thus bypassing the cache and avoiding filesystem overhead.[15][16] Regardless, Red Hat recommends swap partitions to be used.[17] When residing on HDDs, which are rotational magnetic media devices, one benefit of using swap partitions is the ability to place them on contiguous HDD areas that provide higher data throughput or faster seek time. However, the administrative flexibility of swap files can outweigh certain advantages of swap partitions. For example, a swap file can be placed on any mounted file system, can be set to any desired size, and can be added or changed as needed. Swap partitions are not as flexible; they cannot be enlarged without using partitioning or volume management tools, which introduce various complexities and potential downtimes.

Swappiness

Swappiness is a Linux kernel parameter that controls the relative weight given to swapping out of runtime memory, as opposed to dropping pages from the system page cache, whenever a memory allocation request cannot be met from "free" memory. Swappiness can be set to values between 0 and 100 (inclusive). A low value causes the kernel to prefer to evict pages from the page cache while a higher value causes the kernel to prefer to swap out "cold" memory pages. The default value is `60`; setting it higher will increase overall throughput (particularly disk I/O) at the risk of occasional high latency if cold pages need to be swapped back in, while setting it lower (even 0) may provide more consistently low latency. Certainly the default values work well in most workloads, but desktops and interactive systems with more than adequate RAM for any expected task may want to lower the setting while batch processing and less interactive systems may want to increase it.[18]

Swap death

When the system memory is highly insufficient for the current tasks and a large portion of memory activity goes through a slow swap, the system can become practically unable to execute any task, even if the CPU is idle. When every process is waiting on the swap, the system is considered to be in *swap death*.[19][20]

Swap death can happen due to incorrectly configured memory overcommitment.[21][22][23]

The original description of the "swapping to death" problem relates to the X server. If code or data used by the X server to respond to a keystroke is not in main memory, then if the user enters a keystroke, the server will take one or more page faults, requiring those pages to read from swap before the keystroke can be processed, slowing the response to it. If those pages don't remain in memory, they will have to be faulted in again to handle the next keystroke, making the system practically unresponsive even if it's actually executing other tasks normally.[24]

macOS

macOS uses multiple swap files. The default (and Apple-recommended) installation places them on the root partition, though it is possible to place them instead on a separate partition or device.[25]

AmigaOS 4

AmigaOS 4.0 introduced a new system for allocating RAM and defragmenting physical memory. It still uses flat shared address space that cannot be defragmented. It is based on slab allocation method and paging memory that allows swapping. Paging was implemented in AmigaOS 4.1 but may lock up system if all physical memory is used up.[26] Swap memory could be activated and deactivated any moment allowing the user to choose to use only physical RAM.

Performance

The backing store for a virtual memory operating system is typically many orders of magnitude slower than RAM. Additionally, using mechanical storage devices introduces delay, several milliseconds for a hard disk. Therefore, it is desirable to reduce or eliminate swapping, where practical. Some operating systems offer settings to influence the kernel's decisions.

- Linux offers the `/proc/sys/vm/swappiness` parameter, which changes the balance between swapping out runtime memory, as opposed to dropping pages from the system page cache.
- Windows 2000, XP, and Vista offer the `DisablePagingExecutive` registry setting, which controls whether kernel-mode code and data can be eligible for paging out.
- Mainframe computers frequently used head-per-track disk drives or drums for page and swap storage to eliminate seek time, and several technologies[27] to have multiple concurrent requests to the same device in order to reduce rotational latency.
- Flash memory has a finite number of erase-write cycles (see limitations of flash memory), and the smallest amount of data that can be erased at once might be very large (128 KiB for an Intel X25-M SSD [28]), seldom coinciding with pagesize. Therefore, flash memory may wear out quickly if used as swap space under tight memory conditions. On the attractive side, flash memory is practically delayless compared to hard disks, and not volatile as RAM chips. Schemes like ReadyBoost and Intel Turbo Memory are made to exploit these characteristics.

Many Unix-like operating systems (for example AIX, Linux and Solaris) allow using multiple storage devices for swap space in parallel, to increase performance.

Swap space size

In some older virtual memory operating systems, space in swap backing store is reserved when programs allocate memory for runtime data. Operating system vendors typically issue guidelines about how much swap space should be allocated.

Addressing limits on 32-bit hardware

Paging is one way of allowing the size of the addresses used by a process, which is the process's "virtual address space" or "logical address space", to be different from the amount of main memory actually installed on a particular computer, which is the physical address space.

Main memory smaller than virtual memory

In most systems, the size of a process's virtual address space is much larger than the available main memory.[29] For example:

- The address bus that connects the CPU to main memory may be limited. The i386SX CPU's 32-bit internal addresses can address 4 GB, but it has only 24 pins connected to the address bus, limiting installed physical memory to 16 MB. There may be other hardware restrictions on the maximum amount of RAM that can be installed.
- The maximum memory might not be installed because of cost, because the model's standard configuration omits it, or because the buyer did not believe it would be advantageous.
- Sometimes not all internal addresses can be used for memory anyway, because the hardware architecture may reserve large regions for I/O or other features.

Main memory the same size as virtual memory

A computer with true n-bit addressing may have 2^n addressable units of RAM installed. An example is a 32-bit x86 processor with 4 GB and without Physical Address Extension (PAE). In this case, the processor is able to address all the RAM installed and no more.

However, even in this case, paging can be used to create a virtual memory of over 4 GB. For instance, many programs may be running concurrently. Together, they may require more than 4 GB, but not all of it will have to be in RAM at once. A paging system makes efficient decisions on which memory to relegate to secondary storage, leading to the best use of the installed RAM.

Although the processor in this example cannot address RAM beyond 4 GB, the operating system may provide services to programs that envision a larger memory, such as files that can grow beyond the limit of installed RAM. The operating system lets a program manipulate data in the file arbitrarily, using paging to bring parts of the file into RAM when necessary.

Main memory larger than virtual address space

A few computers have a main memory larger than the virtual address space of a process, such as the Magic-1,[29] some PDP-11 machines, and some systems using 32-bit x86 processors with Physical Address Extension. This nullifies a significant advantage of virtual memory, since a single process cannot use more main memory than the amount of its virtual address space. Such systems often use paging techniques to obtain secondary benefits:

- The "extra memory" can be used in the page cache to cache frequently used files and metadata, such as directory information, from secondary storage.
- If the processor and operating system support multiple virtual address spaces, the "extra memory" can be used to run more processes. Paging allows the cumulative total of virtual address spaces to exceed physical main memory.
- A process can store data in memory-mapped files on memory-backed file systems, such as the tmpfs file system or file systems on a RAM drive, and map files into and out of the address space as needed.
- A set of processes may still depend upon the enhanced security features page-based isolation may bring to a multitasking environment.

The size of the cumulative total of virtual address spaces is still limited by the amount of secondary storage available.

See also

- Bélády's anomaly
- Demand paging, a "lazy" paging scheme
- Expanded memory
- Memory management
- Memory segmentation
- Page (computer memory)
- Page cache, a disk cache that utilizes virtual memory mechanism
- Page replacement algorithm
- Page table
- Physical memory, a subject of paging
- Virtual memory, an abstraction that paging may create

Notes

a. Initially drums, and then hard disk drives and solid-state drives have been used for paging.
b. For example, MVS (Multiple Virtual Storage).
c. A non-equivalence interruption occurs when the high order bits of an address do not match any entry in the associative memory.

References

1. Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C. (2014), *Operating Systems: Three Easy Pieces (Chapter: Paging)* (http://pages.cs.wisc.edu/~remzi/OSTEP/vm-paging.pdf) (PDF), Arpaci-Dusseau Books, archived (https://web.archive.org/web/20140222024422/http://pages.cs.wisc.edu/~remzi/OSTEP/vm-paging.pdf) (PDF) from the original on 2014-02-22
2. Deitel, Harvey M. (1983). *An Introduction to Operating Systems*. Addison-Wesley. pp. 181, 187. ISBN 0-201-14473-5.
3. Belzer, Jack; Holzman, Albert G.; Kent, Allen, eds. (1981). "Operating Systems". *Encyclopedia of computer science and technology* (https://books.google.com/books?id=uTFirmDlSL8C&printsec=frontcover). **11**. CRC Press. p. 433. ISBN 0-8247-2261-2. Archived (https://web.archive.org/web/20170227051057/https://books.google.com/books?id=uTFirmDlSL8C&printsec=frontcover) from the original on 2017-02-27.
4. Belzer, Jack; Holzman, Albert G.; Kent, Allen, eds. (1981). "Operating systems". *Encyclopedia of computer science and technology* (https://books.google.com/books?id=uTFirmDlSL8C&printsec=frontcover). **11**. CRC Press. p. 442. ISBN 0-8247-2261-2. Archived (https://web.archive.org/web/20170227051057/https://books.google.com/books?id=uTFirmDlSL8C&printsec=frontcover) from the original on 2017-02-27.
5. Cragon, Harvey G. (1996). *Memory Systems and Pipelined Processors* (https://books.google.com/books?id=q2w3JSFD7I4C). Jones and Bartlett Publishers. p. 109. ISBN 0-86720-474-5. Archived (https://web.archive.org/web/20170227205647/https://books.google.com/books?id=q2w3JSFD7I4C) from the original on 2017-02-27.
6. Belzer, Jack; Holzman, Albert G.; Kent, Allen, eds. (1981). "Virtual memory systems". *Encyclopedia of computer science and technology* (https://books.google.com/books?id=KUgNGCJB4agC&printsec=frontcover). **14**. CRC Press. p. 32. ISBN 0-8247-2214-0. Archived (https://web.archive.org/web/20170227081754/https://books.google.com/books?id=KUgNGCJB4agC&printsec=frontcover) from the original on 2017-02-27.
7. Sumner, F. H.; Haley, G.; Chenh, E. C. Y. (1962). "The Central Control Unit of the 'Atlas' Computer". *Information Processing 1962*. IFIP Congress Proceedings. Proceedings of IFIP Congress 62. Spartan.
8. "The Atlas" (https://web.archive.org/web/20120728105352/http://www.computer50.org/kgill/atlas/atlas.html). University of Manchester: Department of Computer Science. Archived from the original (http://www.computer50.org/kgill/atlas/atlas.html) on 2012-07-28.
9. "Atlas Architecture" (http://www.chilton-computing.org.uk/acl/technology/atlas/p005.htm). *Atlas Computer*. Chilton: Atlas Computer Laboratory. Archived (https://web.archive.org/web/20121210142240/http://www.chilton-computing.org.uk/acl/technology/atlas/p005.htm) from the original on 2012-12-10.
10. Kilburn, T.; Payne, R. B.; Howarth, D. J. (December 1961). "The Atlas Supervisor" (http://www.chilton-computing.org.uk/acl/technology/atlas/p019.htm). *Computers - Key to Total Systems Control*. Conferences Proceedings. Volume 20, Proceedings of the Eastern Joint Computer Conference Washington, D.C. Macmillan. pp. 279–294. Archived (https://web.archive.org/web/20091231062425/http://www.chilton-computing.org.uk/acl/technology/atlas/p019.htm) from the original on 2009-12-31.
11. Kilburn, T.; Edwards, D. B. G.; Lanigan, M. J.; Sumner, F. H. (April 1962). "One-Level Storage System". *IRE Transactions on Electronic Computers*. Institute of Radio Engineers.

12. Tsigkogiannis, Ilias (December 11, 2006). "Crash Dump Analysis" (http://blogs.msdn.com/iliast/archive/2006/12/11/crash-dump-analysis.aspx). *driver writing != bus driving*. Microsoft. Archived (https://web.archive.org/web/20081007215138/http://blogs.msdn.com/iliast/archive/2006/12/11/crash-dump-analysis.aspx) from the original on October 7, 2008. Retrieved 2008-07-22.
13. "Windows Sysinternals PageDefrag" (https://technet.microsoft.com/en-us/sysinternals/bb897426). *Sysinternals*. Microsoft. November 1, 2006. Archived (https://web.archive.org/web/20101225112753/http://technet.microsoft.com/en-us/sysinternals/bb897426) from the original on December 25, 2010. Retrieved 2010-12-20.
14. "swapon(2) – Linux man page" (http://linux.die.net/man/2/swapon). *Linux.Die.net*. Archived (https://web.archive.org/web/20140228161303/http://linux.die.net/man/2/swapon) from the original on 2014-02-28. Retrieved 2014-09-08.
15. " "Jesper Juhl": Re: How to send a break? - dump from frozen 64bit linux" (https://lkml.org/lkml/2006/5/29/3). LKML. 2006-05-29. Archived (https://web.archive.org/web/20101124235412/http://lkml.org/lkml/2006/5/29/3) from the original on 2010-11-24. Retrieved 2010-10-28.
16. "Andrew Morton: Re: Swap partition vs swap file" (https://lkml.org/lkml/2005/7/7/326). LKML. Archived (https://web.archive.org/web/20101124235448/http://lkml.org/lkml/2005/7/7/326) from the original on 2010-11-24. Retrieved 2010-10-28.
17. Chapter 7. Swap Space - Red Hat Customer Portal (https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/ch-swapspace.html) "Swap space can be a dedicated swap partition (recommended), a swap file, or a combination of swap partitions and swap files."
18. Andrews, Jeremy (2004-04-29). "Linux: Tuning Swappiness" (https://web.archive.org/web/20130524085654/http://kerneltrap.org/node/3000). *kerneltrap.org*. Archived from the original (http://kerneltrap.org/node/3000) on 2013-05-24. Retrieved 2018-01-03.
19. Rik van Riel (1998-05-20). "swap death (as in 2.1.91) and page tables" (http://lkml.iu.edu/hypermail//linux/kernel/9805.2/0707.html). Archived (https://web.archive.org/web/20171229195527/http://lkml.iu.edu/hypermail//linux/kernel/9805.2/0707.html) from the original on 2017-12-29.
20. Kyle Rankin (2012). *DevOps Troubleshooting: Linux Server Best Practices* (https://books.google.com/books?id=icPyQDU3xD4C&pg=PT159). Addison-Wesley. p. 159. ISBN 978-0-13-303550-6. Archived (https://web.archive.org/web/20171229195527/https://books.google.com/books?id=icPyQDU3xD4C&pg=PT159) from the original on 2017-12-29.
21. Andries Brouwer. "The Linux kernel: Memory" (https://www.win.tue.nl/%7Eaeb/linux/lk/lk-9.html). Archived (https://web.archive.org/web/20170813052950/http://www.win.tue.nl/~aeb/linux/lk/lk-9.html) from the original on 2017-08-13.
22. Red Hat. "Capacity Tuning" (https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-memory-captun.html). Archived (https://web.archive.org/web/20170723214620/https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Performance_Tuning_Guide/s-memory-captun.html) from the original on 2017-07-23.
23. "Memory overcommit settings" (https://iainvlinux.wordpress.com/2014/02/16/memory-overcommit-settings/). Archived (https://web.archive.org/web/20170531182419/https://iainvlinux.wordpress.com/2014/02/16/memory-overcommit-settings/) from the original on 2017-05-31.
24. Peter MacDonald (1993-02-10). "swapping to death" (http://tech-insider.org/linux/research/1993/0210.html). Archived (https://web.archive.org/web/20170328080649/http://tech-insider.org/linux/research/1993/0210.html) from the original on 2017-03-28.
25. John Siracusa (October 15, 2001). "Mac OS X 10.1" (https://arstechnica.com/reviews/os/macosx-10-1.ars/7). Ars Technica. Archived (https://web.archive.org/web/20080905095622/http://arstechnica.com/reviews/os/macosx-10-1.ars/7) from the original on September 5, 2008. Retrieved 2008-07-23.

26. AmigaOS Core Developer (2011-01-08). "Re: Swap issue also on Update 4 ?" (http://forum.hyperion-entertainment.biz/viewtopic.php?f=14&t=755#p9346). Hyperion Entertainment. Archived (https://web.archive.org/web/20130412080355/http://forum.hyperion-entertainment.biz/viewtopic.php?f=14&t=755#p9346) from the original on 2013-04-12. Retrieved 2011-01-08.
27. E.g., Rotational Position Sensing on a Block Multiplexor channel
28. "Aligning filesystems to an SSD's erase block size | Thoughts by Ted" (http://thunk.org/tytso/blog/2009/02/20/aligning-filesystems-to-an-ssds-erase-block-size). Thunk.org. 2009-02-20. Archived (https://web.archive.org/web/20101113065550/http://thunk.org/tytso/blog/2009/02/20/aligning-filesystems-to-an-ssds-erase-block-size) from the original on 2010-11-13. Retrieved 2010-10-28.
29. Bill Buzbee. "Magic-1 Minix Demand Paging Design" (http://www.homebrewcpu.com/demand_paging.htm). Archived (https://web.archive.org/web/20130605134128/http://www.homebrewcpu.com/demand_paging.htm) from the original on June 5, 2013. Retrieved December 9, 2013.

External links

- Swap Facts and quetions (https://help.ubuntu.com/community/SwapFaq) by Ubuntu Documentation
- Windows Server - Moving Pagefile to another partition or disk (http://it.toolbox.com/blogs/microsoft-infrastructure/moving-the-pagefilesys-to-another-partition-or-disk-35772) by David Nudelman
- How Virtual Memory Works (http://computer.howstuffworks.com/virtual-memory.htm) from HowStuffWorks.com (in fact explains only swapping concept, and not virtual memory concept)
- Linux swap space management (http://www.faqs.org/docs/linux_admin/x1762.html) (outdated, as the author admits)
- Guide On Optimizing Virtual Memory Speed (http://www.techarp.com/showarticle.aspx?artno=143) (outdated, and contradicts section 1.4 of this wiki page, and (at least) references 8, 9, and 11.)
- Virtual Memory Page Replacement Algorithms (https://web.archive.org/web/20110720003417/http://people.msoe.edu/~durant/courses/cs384/papers0405/mccrawt.pdf)
- Windows XP: How to manually change the size of the virtual memory paging file (http://support.microsoft.com/kb/308417/)
- Windows XP: Factors that may deplete the supply of paged pool memory (http://support.microsoft.com/?id=312362)
- SwapFs (http://www.acc.umu.se/~bosse/) driver that can be used to save the paging file of Windows on a swap partition of Linux