Operating System

Me

February 8, 2021

Contents

1 Definitions

2 Memory Management

2.1 Hirachy

Der are different kinds of storage, namely Cache, Ram and SSD (orHDD). Cache ist the fastest becuase its most times built in the processor but usally has only a few megabytes becuase of its high pricepoint while SSD has higher capacities for a lower price but is also way slower. Ram has more storage than cache and is faster than an SSD But has less storage than an SSD and is slwoer than the cahce. Cache and Ram are dependet on electricty to saveda data so only an SSD (HDD) is suitbale for lon-time-storage of data. If one wants to implement memory management, one has to count in those factors so picutres aren't saved in the cache and numbers form calculations won't be saved on an hard drive.

2.2 Memory Organisation

RAM and and disc space need to be managed. There are two options avaiable:

2.2.1 Fixe-size pages

The name already implies that ever page (a page is basically a 'cell' in the ram/disc) has a fixed size. It's like having same sized boxes when one wants to organized their items. As long as there is an list that save win which page (box) data (items) are saved, data (items) is easily recoverable. If data is bigger than the page size, the data is just being spilt up and saved in multiple pages.If data needs to be split, the data isn't necessarily being saved in order, but 'wherever is space'. A problem that often occurs is when the size of a data set isn't a multiple of the page size. Let's say the page-size is 4 Byte and the processor needs 9 Byte for an operation. Now 3 pages a 8 Byte are being used, while 7 Bits are wasted. This is called 'internal fragmentation'. The solution isn't as simple as making the block-size[1] as tiny as possible, because then a file need to be cut in more pieces, and recovering it need more steps which impacts speed, but making the block size big, means a lot of space may be wasted.

2.2.2 Flexible sized chunks

The idea is, like a book shelf, data is stored in the first avaiable space that fits the size of data. The problem is when 4 bytes are removed from the page/block, and 5 bytes are now needed to be stored, the new data tis going to be stored at the end of the disk/block. To fil the hole a file with exactly 4 byte or 2 with 2 byte are needed. Those holes are a common occurances with flexible sized chunks.

[1] block when talking about disc, page when talking anbaout ram

2.2.3 Defragmetnation

To get rid of those blank spaces a process called defragmentation is being used. It basically moves all the data back to back until there is only one big chunk free space. It's rather complicated to do so because nearly every single data object need their addresses reallocated. This kind of defragmentation is possible with basically all types of memory. The more know kind of defragmetnation moving every parts of a file so that they are in order. It used with discs using fixed-sized blocks. This only usefull on spinning disc drives becuase now the disc head doesn't need to jump as jump around. There are also filesystems which to this job 'on-the-fly'.

2.2.4 Memory Allocation Algorithms with variable size allocations

There are several methods where to save data with variable size allocation:

First-Fit Find the first place which can fit the date in its entirety and place it there

Next-Fit Like first fir, but remembers where the last data was palced and searches for the next palced

Best-Fit First analyzes every free space and places the data where the least amoaunt of new free space is being created

Worst-Fit Like Best fit but searches a place which will leave the bigges remainder

uddy method The buddy method introduces Lvevel. Level 0 saves data at given size (block sze, 8 byte for example). The next Levels are always 2 Blocks combined from the Level before (Level 16 Byte, level 2 32 Byte). Levle 0 saves the smallest data (8 Byte or less). If the the data is bigger it's going to be save in another Level (15 Byte will be save in level 1). Also if an level has no space left, the data is being hand over to the next level. When a level has free space left, but not so much thath it can save data for what is supposed to, it will give it to the levle below (if Level one has 15 bytes free, it won't be able to safe something because it needs at least 16 Bytes, so the 15 Bytes are given to Levle 0).

Quick-Fit Like the buddy method, but level one can also have 3 level 0 blocks (buddy mehtod only allows pairs of 2).

2.3 Free Memory Management

There are 2 basic options to find out where and if a page is free:

Linekd List Having a list of all the free pages makes it easy to find a free page. The only thing oen has to do is take the first element of the list and redirect the start of the list to the next element. Adding a free page is also wasy. Just make the new page the start of the list and let the new page point to the old first page. The elements of the lsit need to contain start address of free block and end adrees(or size) of the

free block. If an address has 32 bit at least 64 bits are needed to save one page. But every element of the list also links to a next element, so 32 bits need to be added again. with 96 bits (12 bytes) for each element this would that on a 16GB PC with page size of 4KB 48 Mbytes are needed to be able to save every free page

Bitmap A bitmap is just a long sequence of Bits. If the first bit is set to 0 it means the page is free, if its oen its means the page isn't free (or vice versa). For the same 16GB pc with page sizes of 4KB the Bitmap only need 512 Kbyte to safe all pages and there state.

Typicall bitmaps are being used to see which page is free or not. Linked list uses less memory when less pages are free, but they only beat the constant size of a bitmap when memory usage is at abou 99%.

2.4 Without Memory Abstraction

No abstraction means that the evry RAM access is being directly written to the RAM. Depending on how this is implemented, different problmes may occur. Early mainframe laoded the OS in the lower address space of the RAM and had gave the application the following space. DOS used a similar approach, but reserved a part of the hugher address space to device driver. Those aproaces created the problem, that an application could write into the OS and inject malware in the system. PDAs saved the OS in a ROM so applications won't be able to write into them, but the OS still used RAM for its own operation so application still could access the data and modify it maliciously. Dos' approach made it also possible to mess with devices because the drivers are accessible.

2.4.1 Multitasking

This system ist suited very well for multitasking. One way to still be able to use multitasking is called 'multi programming'. If a process wan't been used order another process was started (or focused) the programms RAM data would be dumped to th edisc, if the programm was needed again, the other programm current program was dumped and the old one was loaded in again. Switching task would be time consuming because of the low speed of a disc. Also with those simpler OS's programmers often used static memory addresses. If now 2 programs are loaded in, because enough space was free, both programs could break, because they want to use the same memory space.

2.4.2 Static realoccation

If a programm has static memory addresses which it wants load into, but the OS can't give the addresses to the programm because tehy are already occupied, the os can just give a new static beginning address. So if the programm want to wirte into address 50 but 50 is already been used, the OS makes a new starting address, lets say 100. Now the programs address is being converted to 150 instead of 50.

This approach crates some complication. If an address should be saved in an register the code would look something like this:

$$MOV AX, 1000$$

Now if a programmer wants to save a value into a register the code woudl look like this:

$$MOV AX, 1000$$

The problem is, that an OS cant't tell if one value is an address or if it is just a constant, so just adding a specific value to every constant wont be possible.

2.4.3 Dynamic relocation

It uses the core concept of the static relcoation, but the base address is being saved in a register and added to every address call made by the programm. Antoher advatnatge is now a limit register address is being introduced. The limit reigster is the highest address a programm is allowed to use, os it can't conflict withother programms. Bot Conecpts are rather slow because every time a address is accessed, it first needs to be recalculated and then it can be accessed.

2.4.4 8086 addressing specials

8086 uses 16 bit long reigster but had 20 bit long addresses so two addresses are being needed to save an address. The reigster with containing the higher bits is called the segment, and the register with containing the lower bits is called the offset. THe real adress need to be calculated. First one hast to shift the bits of the segment four times to the left. To this numbes the offset is being added to create the actual adress. Strangley there are several combination of Segment:Offset for an physical adress (1234:0005, 1233:015 both create 12345).

2.5 Memory abstraction

Memory abstraction provides a process a set of pages. If a byte in a page needs to be adressed, the first thing one has to know is the page size. If the page size is 4KB 12 bits suffice to adress every Byte in one page. Those 12 bits are called the offset. A 32 bit adress which adresses 4KB pages is made outof a 20 bit page number which adresses a specific page and a 12 bit offset which adresses a specific byte in a page. The page number can be seen as a key, which return the value of a physical adress of a page, which an OS can use. With 32 bit system the page table would contain 2^20 entries. Assuming that the n-th row means the n-th page number (saving space by not implicitly saving the page number) each entry save a 32 bit long adress. The size of the table would be:

$$\frac{2^{20} \cdot 32}{8} \approx 4MB$$

On an 64 bit system the page tamble contains 2^{52} entries, each with 64 bit long adresses. This results in a page table with about 32TB. Adding layers (page:subpage:offset) can drastically lower the memory usage. Another option is to have larger page sizes, but this would most likely cause higher internal fragmentation.

2.6 Virtual Memory

There are different approaches to conserve memory.

Overlays With memory restrictions programmers couldn't load all of the programm at once, so they only loaded in overlays. An overlay contains a set if functionallity and would be laid over (replaced) if another overlay was needed.

Swapping[2] Swapping moves the entire memory of a process to the disc and loads in a other processes. Nowadays this isn't possible, because of the memory usage of modern programs (chrome tabs as an example).

Virtual ram Instead of moving entire processes to the disc, only pages which are rarely or not at all used are moved to the disc. This is poosible because most processes aren't using all of their data/code simultaneously

2.6.1 MMU

For the MMU (Memory Management Unit) to support swapping singel pages, the page table needs to provide additional information.

present/absent Is the page currently in RAM or swapped out?

modified Does the version in RAM differ from the one stored on disk - has it been modified since being loaded into RAM?

recently Has it been accessed in the last "cycle", a time frame given by the implementation of the OS and / or the MMU.

caching on/off Is swapping allowed for this page? (Swapping the code, that takes care of loading a page from disk into memory would be a bad idea for example)

where the page has been swapped to

There is more information saced which is being accounted when deciding, which pages to swap out:

Page fault If an absent page is being accesed, a page fault exception is thrown which the OS neds to handle. The Os need to decide where to place the page in the memory and possibly swapping out a page currently loaded in memory

Page fault rate The pauge fault rate is ratio between all page accesses and page faults. The lower the number the better the swapping algorithm (or the more RAM is avaiable)

Thrashing If the page fault rate approaches 1 (nearly every page access first need to swap out a page from the RAM) the memory is basically overlaoded and either a process has a memory leak or a programm needs to be terminated for the PC to b eresponsive again. Possibly a RAM upgrade is needed.

2.7 Introduction to Swapping

Ideally only those pages which aren't beeing used will be swapped while important ones stay in the memory. One way to predict what ressources need to be loaded into memory is the principal of locallity. It's basically divided into two parts:

locality of code In program code loops play a mayor role. Loops repeat basically the same or at least similiar instructions. So it makes sense to load code chunks in loops into the memory.

loclaity of data Similar principal to the locality of code. For example implenting soting algortihms to coompare their time. In this case you'd want to use the same numbers. Every algorithm would make a copy of the number array so it wouldn't make sense to swap the main number array every time its copied.

2.8 Swapping Algorithms

Different algorithms (or at least conecpts) were made to for swapping pages from memory to the disc.

2.8.1 Recently used Algorithms

LRU Least Recently used takes a the page, which wans't used for the longest time (pages are timestamped) to swap it out of memory. Access time is rather bad ($\mathcal{O}(n)\}$) for an an List and $\mathcal{O}(\log n)$ for an tree (but a tree has also $\mathcal{O}(\log n)$ for adding an item).

NFU Not frequentliy used counts how often a page is being accesed in an given cycle. If a page is accesed in cycle the Recently-Bit is set to one and after the cycle an counter for the page is being incremented. There is a loss of precision because multiple accesses in on cycle only counts as one but it's normally 'good enough'. A bigger problem is, when a page that wasn't accessed a long time is being accesed. its counter is compared to the others rather low so it's going to be swapped out even if its needed in the next cycle.
Aging is a process which tries to fix this problem. All the bit (Recently-Bit + counter bits) are rotated to the right every cycle. So a page which was used a lot in the first part of an process 'ages' so it looses its priortiy over time.

NRU Not Recently Used saves to values, the Recently bit and the dirty bit. Those values are reset every cycle while Recently is set to one, if the page was accesed in the cycle and the dirty bit means that a page has been modified and need to be swapped (if a user did an input or something like that).

2.8.2 First-In-First-Out

Easily enough this syle of swapping pages just swaps the first page that was loaded in the memory. The probelm is important pages need to stay in the memory (Kernel ressources etc). An improvement is called 'second-chance'. If a page is being accesed a recently bit is being set and if the 'first' page is to be swapped, the page is just being placed to the end of the list and the recently bit is being set to 0. A further improvement to second chance is to link the 'last' page to the 'first' page so a a ring structure is being created. This is called a clock becuase the imagining this structure as a clock, the pointer would mve through it like a clock hand.

2.9 Working Set Algorithms

Working set loads pages only when pages are needed. A lot of loading is required. One als can't just say that working set shoudl remember specific pages shich need to be loaded when starting an process because a computer can't know what one wants to do so it either laods nothing (lots of extra loading required) or it loads everything it can (working set advatnatges are gone). To makes this work working set could only remember n cycles and ditch older ones (again a clock structure like in FIFO, called workin set clock). Another approach is by using the locality of code rule (load in code which is likely soon to be executed), so ie. browsers prelaod contents of a link so one can get faster access.

2.10 Implementation issues

2.10.1 Strategies

With a so called global strategie the pages of all processes are beeing considered to be swapped while with a local strategie processes with higher numbers of pages loaded into memory are being considered to be swapped. Local strategy needs to monitor the page fault rate for every process to determine which process to swap out the pages from.
If a processes page fault rate grows above the overall average, the process needs to load in more pages to memory (GIMP loading in a picture), while a lower page fault rates means thath the process freed up memory (GIMP closing the loaded in picture).

2.10.2 Page Size Optimisation

Page size from the memory should match the block size from the disc. Currently 4KB are standard for both block and page size on most OS'.

2.10.3 seperating data and instructions