Virtual Memory

Virtual Memory:

Virtual memory gives the illusion of large main memory. It is a technique that allows execution of processes that are not completely in main memory. Main advantage of this is that program can be larger than physical memory.

Whenever the process size is larger than the main memory then operating system helps in running that process and provide illusion that the main memory is available to load complete memory.

Without virtual memory the complete process load into memory but with the help of virtual memory we can load part of the of the process in the main memory and run it.

Virtual address space of a process refers to the logical or virtual view of how process is stored in memory.

Sparse address space: virtual address space that include holes are known as sparse address space.

Demand paging:

Load pages only as they are needed that is loading the page on demand during the program execution. Only part of the process present in the main memory. In demand paging do not load page until it required.

Now you have basic idea about virtual memory now explore more about virtual memory : https://www.geeksforgeeks.org/virtual-memory-operating-systems/

Valid/invalid bit shows the page which we are looking for is present or not in main memory.

Page fault: whenever the page which we are looking for is not present in the main memory then we are supposed to get it from secondary memory.

Page fault explanation with the help of diagram: https://www.geeksforgeeks.org/operating-system-page-fault-handling/

Dirty bit: whether the page after getting loaded in the main memory has been changed or not.

Write back: when we have various levels of memory whenever the modification bit is done at high level (main memory) so it must reflect at lower level (secondary memory). It is done when overwrite the page.

Reference bit: in the last work cycle which page is referred (1 = referred and 0= not referred). It is mainly used in page replacement algorithm.

Protection bit: it shows the rights to do action like read(R), write(W), execute(X).
Data section: RW
Code segment: RX
Stack segment: RW

Swapping and Thrashing concept : https://www.geeksforgeeks.org/virtual-memory-operating-systems/

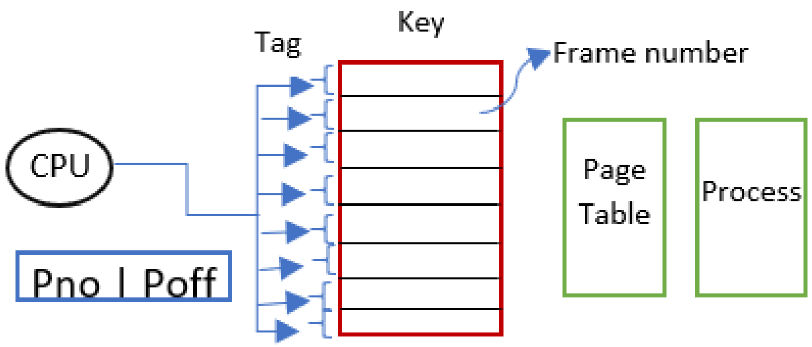
Techniques to handle Thrashing: https://www.geeksforgeeks.org/operating-system-techniques-handle-thrashing/

Questions of Virtual memory: https://www.geeksforgeeks.org/virtual-memory-questions/

Operating system based Virtualization: https://www.geeksforgeeks.org/operating-system-based-virtualization/

Swap Space: https://www.geeksforgeeks.org/operating-system-swap-space/

Translation look aside buffer (TLB):



In order to save the space, instead of loading the entire page table into the main memory they are using the concept of locality of references.

Locality of reference:
In order to run a process, we might need not run the entire process we just load the few pages and therefore we can even load only the page table entries that are concerning to those pages.

It uses cache memory which is faster than the main memory but cheaper than the registers. So, page table store in this cache.

Whenever we try to access any page table entry then try to look it present in cache or not.

A process will access only few pages for a long time so these few pages will keep on changing but at any instant of time a process will not access a CPU or process doesn't need all the pages to be available as it needs only few pages.
Whenever we need any page the page table entry will be referred for the first time. So, we load page table entry inside our TLB which is also referred as associate memory.

Associate memory: every element of this type of memory have tag. If we want to search a key, then compare the entire associative memory. If the key is present or not. In case the key is matched with any tag so, take out that entry corresponding to tag.

CPU before going to the page table, it will try to give this Pno|Poffset to all the tags present in the TLB. If this Pno matches with any of the tag, it will immediately give us the frame number. So, for this we need to go to the page table which is present in the main memory.

Since TLB is faster compare to main memory we can directly convert the virtual address to physical address then go to the main memory to get the word. Since TLB is smaller than the main memory so it not able to hold entire page table. It just holds frequently used page table entries.

TLB hit: The page which we are looking for is in TLB.
TLB miss: The page which we are looking for is not present in TLB.

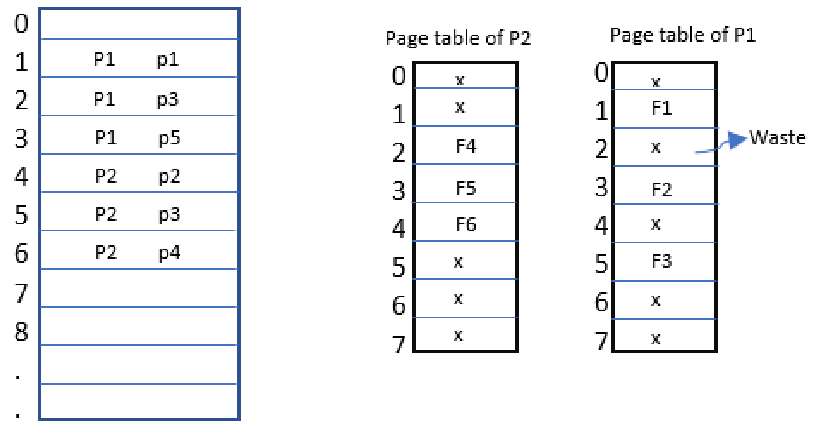
Effective memory access time (EMAT) = p (t + m) + (1-p) (t + km + m)

Where,
p = TLB hit

t = TLB access time
m = memory access time
k = memory level.

Inverted page table:

First explore this link: https://www.geeksforgeeks.org/operating-system-inverted-page-table/



Instead of having one-page table for every process. Inverted page table concept introduced that let us have only one-page table for all the processes and the entries in the page table will be equal to number of frames in the computer.
It maintains indexes as frame number the reason.

Page table load into memory it takes lots of space as the page table contains empty frame details also this leads to wastage of memory. So, for this we can only enter the details of frame of every process in one table.

This concept is not widely used as searching is time consuming.

Page replacement Algorithm:

First explore this link: https://www.geeksforgeeks.org/page-replacement-algorithms-in-operating-systems/

Local page replacement: the local algorithm works in such a way that whenever we have to replace any page then we are going to replace the page only from the frames which are allocated to that process. It will not disturb any other process.

Global page replacement: It consider everything for replacement from all the available frame it can replace anything. It can use many parameters like priorities.

Page replacements algorithms are:

1. **FIFO** (First in First Out): order of process coming.

2. **LRU** (Least recently used): replace the page which has not been referenced for a long time.
   Example: https://www.geeksforgeeks.org/program-page-replacement-algorithms-set-1-lru/

3. **Optimal**: replace the page which will not be referred longest. It gives least number of page fault. It is not practically implemented as future is consider).
   Example: https://www.geeksforgeeks.org/program-optimal-page-replacement-algorithm/

4. **Least frequently used:** https://www.geeksforgeeks.org/lfu-least-frequently-used-cache-implementation/

5. **Second chance page replacement**: https://www.geeksforgeeks.org/operating-system-second-chance-or-clock-page-replacement-policy/

Numerical: https://www.youtube.com/watch?v=Ub4VVDGLJx0

Example:

Reference string: 4, 7, 6,1,7,6,1,2,7,2
Frame = 3
H = hit

1. **Optimal page replacement algorithm**:

Page fault = 5

4	7	6	1	7	6	1	2	7	2
4	4	4	1	1	1	1	2	2	2
	7	7	7	7	7	7	7	7	7
		6	6	6	6	6	6	6	6
				H	H	H		H	H

2. **Least recently used (LRU) page replacement algorithm**

Page fault: 6

4	7	6	1	7	6	1	2	7	2
4	4	4	1	1	1	1	1	1	1
	7	7	7	7	7	7	2	2	2
		6	6	6	6	6	6	7	7
				H	H	H			H

3. **First in First Out (FIFO) page replacement algorithm**

Page fault: 6

4	7	6	1	7	6	1	2	7	2
4	4	4	1	1	1	1	1	1	1
	7	7	7	7	7	7	2	2	2
		6	6	6	6	6	6	7	7
				H	H	H			H

If we allocate a greater number of frames the page fault is decrease. As if we have more capacity then we will try to accommodate as many pages as possible so in future we didn't load them again.

If we increase the number of frames that allocated to a process, then page fault should reduce. So, this happens in optimal page replacement algorithm.

Exception:
Since by increasing the number of frames the number of page fault will reduce but in FIFO when we increase the number of frame then the page faults are increasing. This situation is not always true but there are chances of that.

Belady's Anomaly: If we increase the number of frames, then page fault will increase. This situation mainly occurs in FIFO.

Reason for belady's anomaly: https://www.geeksforgeeks.org/operating-system-beladys-anomaly/

Stack Algorithm:

Before starting, first explore about stack based algorithm: https://www.geeksforgeeks.org/operating-system-beladys-anomaly/

Reference string: 0 1 2 3 0 1 4 0 1 2 3 4

Optimal page replacement:

0	1	2	3	0	1	4	0	1	2	3	4
0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	2 0	3 3	3 3
	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1
		2 2	3 2	3 2	3 2	4 2	4 2	4 2	4 2	4 2	4 2
			3	3	3	4	4	4	4	4	4

This shows what is the state of memory when there are 3 frames or when there 4 frames.

Observation:
The number of pages which were present when number of frames is 3 are also present in when number of frames is 4.

Therefore,
Number of pages which were present with 3 frames subset of number of pages which were present with 4 frames.

FIFO:
Reference string: 0 1 2 3 0 1 4 0 1 2 3 4

0	1	2	3	0	1	4	0	1	2	3	4
0 0	0 0	0 0	3 0	3 0	3 0	4 4	4 4	4 4	4 4	4 3	4 3
	1 1	1 1	1 1	0 1	0 1	0 1	0 0	0 0	2 0	2 0	2 4
		2 2	2 2	2 2	1 2	1 2	1 2	1 1	1 1	3 1	3 1
			3	3	3	3	3	3	2	2	2
						X	X			X	

Whenever we increase the number of frames, the number of ages which are present with 3 frames should always be the subset of number of pages which are present with 4 frames but there are some case where stack property doesn't follow.
So, FIFO in some case doesn't follow stack property that is why belady's anomaly present in FIFO.

Reason for Belady's Anomaly is stack property.

*Stack Algorithm: LRU, Optimal.
*Non-Stack Algorithm: FIFO

Segmentation:

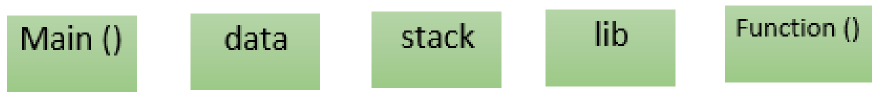
The problem with the paging is when we divide the process into pages sometimes two related parts of a program might fall into two different pages. So, this type of division leads to some useful items might falls in different pages which are supposed to be together.

Segmentation supports the user view of memory (as user view is different from the OS view). OS can directly see the process has pages and it can load it, but a user will always have different kind of view so for the user- entire process is divide into segments.

Logical address space is a collection of segments. Each segment has name and a length. The address specifies both the segment name and the offset within the segment.

Segmentation implementation with segmentation advantages and disadvantages:
https://www.geeksforgeeks.org/operating-systems-segmentation/

User's view of a program or segments be like:

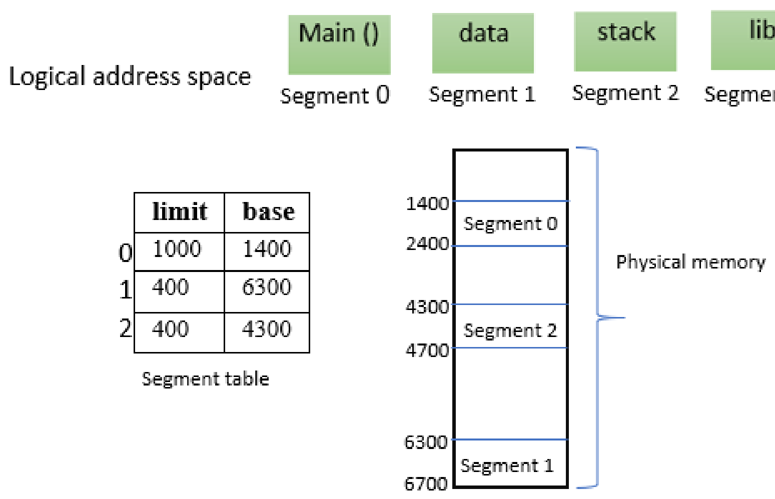


When we load a program, the entire segment loaded in memory.

Segment table:

Base: starting physical address.
Limit: size (length of the segment) or till what part it is running.

It contains one entry for each segment. Segment table also loaded in one page.



	limit	base
0	1000	1400
1	400	6300
2	400	4300

Segment table

Segmented paging:

Process is divided into segments and here segments is divided into pages. Main memory is divided into frames. Segments are closer to the user point of view and paging is closer to the implementation point of view. Therefore, user can look the program as collection of segments and OS will look as pages and frames.

Allocating kernel memory:

Buddy System:

The buddy system allocates memory from a fixed size segment consisting of physically contiguous pages. Memory is allocated from this segment using a power of 2 allocator, which satisfies request in unit sized as power of 2 (4KB, 8KB, 16KB, 32KB and so on).
The advantage of buddy system is how quickly adjacent buddies can be combined to form larger segments using a technique known as coalescing.

Please explore the give links in sequence to learn about buddy system:

a. Buddy system Memory allocation technique: https://www.geeksforgeeks.org/operating-system-buddy-system-memory-allocation-technique/
b. Buddy system and slab system: https://www.geeksforgeeks.org/operating-system-allocating-kernel-memory-buddy-system-slab-system/