

“VIRTUAL MEMORY”

**Course name:** Computer Organization and Architecture

**Course ID**: CSE214

# Section: 02

**Course instructor:** Mohammad Noor Nabi

# Prepared by,

**Abu Musa Sakib - 1810617**

**Sumyia Afnan Mukta - 1810668**

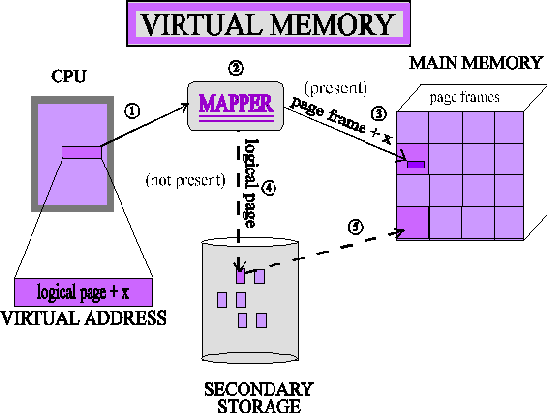
**Md. Ashikur Rahman - 1831110**​

**Fatematuz Jahura - 1830481**        ​

**Date of submission**: January 23, 2021

# What is Virtual Memory:

Virtual means something which appears to be present but is not in reality. Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. Allows user to use more memory for a program than the real memory of a computer.



One of the most important of all concepts related to Memory Management is Virtual Memory. Virtual Memory refers to the concept whereby a process with a larger size than available memory can be loaded and executed by loading the process in parts. The program memory is divided into pages and the available physical memory into frames. If a process

attempts to access a page that is not available in the main memory and the information of which does not exist in its page table, a page fault occurs. The Operating System now takes care of swapping this page into the main memory from the backing store. The Operating System follows its predefined algorithms for page replacement. Demand Paging refers to loading a page of program code from disk into memory as and when it is required by the program.

# Why Virtual Memory is needed:

Virtual memory is an imaginary memory which we are assuming. If we have a material that exceeds your memory at that time, we need to use the concept of virtual memory. Virtual memory is a temporary memory that is used along with the ram of the system.

# Since when did we start using virtual memory?

Everything changed with the release of **Windows 3.0 and the 8386 processor**. With these two together we could use virtual memory. With virtual memory we still use our physically installed RAM, but **we can also map RAM addresses to the hard drive.**

# Address space and Memory space:

Virtual memory is the address used by the programmer and the set of such addresses is called address space. An address in the main memory is called a physical address. The set of such locations in the main memory is called the memory space. Thus, the memory space consists of the actual main memory locations directly addressable for processing.

Diagram

Description automatically generated

Graphical user interface

Description automatically generated

# How Virtual Memory is used:

The programs sharing the memory change dynamically while the programs are running. The compiler sets each program into its own address space. Virtual Memory translates the program’s address space to physical addresses, enforcing protection of a program’s address space from other programs.VM allows a single user program to exceed the size of primary memory. A Virtual Memory block is called a page, and a virtual memory miss is called a page fault. Virtual Memory produces a virtual address, translated by a Software and Hardware combination to a physical address.

# Paging:

The logical address space of a process can be noncontiguous; the process is allocated physical memory whenever the latter is available. Divide physical memory into fixed- sized blocks called frames (size is the power of 2, between 512 bytes and 8,192 bytes) Divide logical memory into blocks of the small size called pages. Keep track of all free frames. To run a program of size n pages, need to find n free frames and load program. Set up a page table to translate logical to physical addresses.

Diagram

Description automatically generated

# Page fault:

A page fault occurs when a program accesses a page that has been mapped in address space but has not been loaded in the physical memory.

Though the term "page fault" sounds like an error, page faults are common and are part of the normal way computers handle virtual memory. In programming terms, a page fault generates an exception, which notifies the operating system that it must retrieve the memory blocks or "pages" from virtual memory for the program to continue.

Once the data is moved into physical memory, the program continues as normal. This process takes place in the background and usually goes unnoticed by the user.

Most page faults are handled without any problems. However, an invalid page fault may cause a program to hang or crash. This type of page fault may occur when a program tries to access a memory address that does not exist. Some programs can handle these types of errors by finding a new memory address or relocating the data. However, if the program cannot handle the invalid page fault, it will get passed to the operating system, which may terminate the process.

This can cause the program to unexpectedly quit. While page faults are common when working with virtual memory, each page fault requires transferring data from secondary memory to primary memory. This process may only take a few milliseconds, but that can still be several thousand times slower than accessing data directly from memory.

Therefore, installing more system memory can increase your computer's performance, since it will need to access virtual memory less often.

Diagram

Description automatically generated

# Thrashing:

When paging and page stealing is used, a problem called "thrashing" can occur, in which the computer spends an unsuitably large amount of time transferring pages to and from a backing store, hence slowing down useful work. A task's working set is the minimum set of pages that should be in memory for it to make useful progress.

Thrashing occurs when there is insufficient memory available to store the working sets of all active programs. Adding real memory is the simplest response, but improving application design, scheduling, and memory usage can help. Another solution is to reduce the number of active tasks on the system. This reduces demand on real memory by swapping out the entire working set of one or more processes.

A picture containing diagram

Description automatically generated

# Translation Lookaside Buffer

Each virtual memory references can cause two physical memory accesses One to fetch the page table.

One to fetch the data.

To overcome this problem a high -speed cache is set up for page table entries called a Translation Lookaside Buffer. (TLB)

The idea used here is, place the page table entries in registers, for each request generated from CPU (virtual address), it will be matched to the appropriate page number of the page table, which will now tell where in the main memory that corresponding page resides. Everything seems right here, but the problem is register size is small (in practical, it can accommodate maximum of 0.5k to 1k page table entries) and process size may be big hence the required page table will also be big (let us say this page table contains 1M entries), so registers may not hold all the PTE’s of Page table. So, this is not a practical approach.

To overcome this size issue, the entire page table was kept in the main memory. but the problem here is two main memory references are required:

Diagram

Description automatically generated

1. To find the frame number
2. To go to the address specified by frame number.

To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is nothing, but a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved, and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.

# TLB function:

If a virtual address is presented to the memory management unit (MMU), the hardware checks TLB by comparing all entries simultaneously. If the match is valid, the frame number is taken from TLB without going through the page table. If a match is not found memory management unit (MMU) detects a miss and does a regular page table lookup It then evicts one old entry out of TLB and replaces it with the new one, so that next time the page is found in TLB.

Steps in TLB hit:

1. CPU generates a virtual address.
2. It is checked in TLB (present).
3. The corresponding frame number is retrieved, which now tells where in the main memory page lies.

Steps in Page miss:

1. CPU generates virtual address.
2. It is checked in TLB (not present).
3. Now the page number is matched to page table residing in main memory (assuming page table contains all PTE).
4. The corresponding frame number is retrieved, which now tells where in the main memory page lies.
5. The TLB is updated with new PTE (if space is not there, one of the replacement techniques comes into picture i.e., either FIFO, LRU or MFU, etc.).

Effective memory access time (EMAT): TLB is used to reduce effective memory access time as it is a high-speed associative cache.

EMAT = h\*(c+m) + (1-h) \*(c+2m)

where, h = hit ratio of TLB m = Memory access time c = TLB access time

# Swapping:

Swapping allows multiple programs to run parallelly in the operating system. The basic difference between paging and swapping is that paging avoids external fragmentation by allowing the physical address space of a process to be noncontiguous whereas, swapping allows multiprogramming.

It is an "expensive" process regarding its overall impact on the system performance since moving data to and from the disk has considerable overhead. The more applications requiring the system to do memory swapping, the slower the performance becomes due to the increased overhead. In this case, increasing the amount of physical RAM would be the best course of action rather than allowing the system to do constant data juggling between the disk and the memory.Diagram

Description automatically generated

# Segmentation:

In a system using segmentation, computer memory addresses consist of a segment id and an offset within the segment. A hardware [memory management unit](https://en.wikipedia.org/wiki/Memory_management_unit) (MMU) is responsible for translating the segment and offset into a [physical address](https://en.wikipedia.org/wiki/Physical_address), and for performing checks to make sure the translation can be done and that the reference to that segment and offset is permitted.

Each segment has a length and set of permissions (for example, *read*, *write*, *execute*) associated with it. A [process i](https://en.wikipedia.org/wiki/Process_(computing))s only allowed to make a reference into a segment if the type of reference is allowed by the permissions, and if the offset within the segment is within the range specified by the length of the segment. Otherwise, a [hardware](https://en.wikipedia.org/wiki/Hardware_exception) [exception](https://en.wikipedia.org/wiki/Hardware_exception) such as a [segmentation fault i](https://en.wikipedia.org/wiki/Segmentation_fault)s raised.

Segments may also be used to implement [virtual memory](https://en.wikipedia.org/wiki/Virtual_memory). In this case each segment has an associated flag indicating whether it is present in main memory or not. If a segment is accessed that is not present in main memory, an exception is raised, and the [operating system](https://en.wikipedia.org/wiki/Operating_system) will read the segment into memory from secondary storage.

Segmentation is one method of implementing memory protection.Chart, bar chart

Description automatically generated

## Segmentation Table maps 2-dimensional physical addresses; each table entry has:

Base: it contains the starting physical address where the segments reside in memory Limit specifies the length of the table

Segment-Table Base Register (STBR): it points to the segment table’s location is a memory.

Segment-Table Length Register (STLR): it indicates the number of segments used by a program.

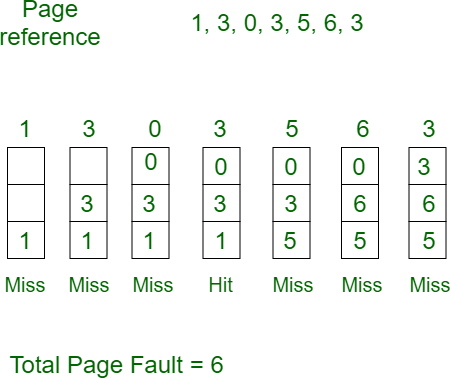
Segment number s is legal ifs < STLR.

# Page Replacement Algorithms in Operating Systems

**First in First Out (FIFO) –**

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

**Example-1** Consider page reference string 1, 3, 0, 3, 5, 6 with 3-page frames. Find number of page faults.



Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots

—> 3 Page Faults.

when 3 comes, it is already in memory so —> 0 Page Faults.

Then 5 comes, it is not available in memory, so it replaces the oldest page slot i.e., 1.

—>1 Page Fault.

6 comes, it is also not available in memory, so it replaces the oldest page slot i.e., 3 —>1 Page Fault.

Finally, when 3 come it is not available, so it replaces 0 1 page fault

Chart, line chart

Description automatically generated

# Advantages and Disadvantages:

The primary advantage or objective of Virtual Memory systems is the ability to load and execute a process that requires a larger amount of memory than what is available by loading the process in parts and then executing them. The disadvantage is that Virtual Memory systems tend to be slow and require additional support from the system's hardware for address translations. It can be said that the execution speed of a process in a Virtual Memory system can equal, but never exceed, the execution speed of the same process with Virtual Memory turned off. Hence, we do not have an advantage with respect to the execution speed of the process. The advantage lies in the ability of

the system to eliminate external fragmentation. The other disadvantage of Virtual Memory systems is the possibility of Thrashing due to excessive Paging and Page faults. In may be noted that Trash Point is a point after which the execution of a process comes to a halt; the system is busier paging pages in and out of the memory than executing them.

**Limitations**

The use of virtual memory has its tradeoffs, particularly with speed. It is generally better to have as much physical memory as possible, so programs work directly from RAM or physical memory. The use of virtual memory slows a computer because data must be mapped between virtual and physical memory, which requires extra hardware support for address translation.