CV RAMAN GLOBAL UNIVERSITY

BHUBANESWAR, ODISHA

CASE STUDY REPORT

COMPUTER ORGANIZATION (CSE21308)

TOPIC - PAGING IN OPERATING SYSTEM

SUBMITTED BY:-

SL. NO.	Name	Registration Number
1	Hitesh Kumar Sejpada	2201020950
2	Purnima Pattnaik	2201020960
3	Riya Mehta	2201020961
4	Kaibalya Prasad Satpathy	2201020962
5	Mohit Kumar Sahukar	2201020964
6	Anup Kumar Mahato	2201020915

GROUP – 4

Case Study Group- 3

UNDER THE GUIDANCE OF: Dr. Smita Rani Parija

C. V. Raman Global University, Odisha, Bhubaneswar
PIN -752054, India.
2023-2024

DECLARATION

This is to certify that the experiential learning report entitled "PAGING IN OPERATION SYSTEM " submitted in partial fulfillment of the requirement for the award of Bachelor of Technology in CSE of the C. V. Raman Global University, Odisha during the year 2023-2024, is a faithful record of the bonafide work carried out by **Hitesh Kumar Sejpada** (2201020950), **Purnima Pattnaik** (2201020960), **Riya Mehta** (2201020961), **Kaibalya Prasad Sathpathy** (220102962), **Mohit Kumar Sahukar** (2201020964), **Anup Kumar Mahato** (2201020915) under my guidance and supervision.

DR .SMITA RANI PARIJA
Department of ECE
C. V. Raman Global University, Odisha, Bhubaneswar, PIN- 752054

ACKNOWLEDGEMENT

It is a great privilege for us to express our special thanks of gratitudeto our respected teacher, MRS.SMITA RANI PARIJA Assistant Professor, Department of **Electronics And Communication Engineering**, C.V. Raman Global University, for their constant guidance, valuable suggestions, supervision and inspiration throughout the work without which, it would have been difficult to complete the work within the scheduled time.

We are also indebted to the Heat of the Department, Computer Science and Engineering, C.V. Raman Global University for permitting us to pursue the project. We would like to take this Opportunity to thank all the respected teachers of this Department for being a perennial source of Inspiration and showing the right path at the time of necessity.

Thanking You

GROUP 4
2ND YEAR
4TH SEMESTER

ABSTRACT

Paging is a fundamental memory management technique employed by modern operating systems to efficiently utilize memory resources and facilitate multitasking. This paper provides a comprehensive exploration of paging mechanisms, elucidating its principles, benefits, and implementation strategies.

The abstract begins by delineating the foundational concepts of memory management and the motivations behind the development of paging. It elucidates the shortcomings of contiguous memory allocation and introduces paging as a solution to mitigate fragmentation and enhance memory utilization.

Furthermore, the abstract delves into the core components of paging, including page tables, page table entries, and page replacement algorithms. It elucidates the role of the memory management unit (MMU)in translating virtual addresses to physical addresses and discusses the mechanisms of address translation through page tables.

Moreover, the abstract examines various paging strategies such as demand paging, pre-paging, and inverted page tables, highlighting their respective advantages and trade-offs. It also discusses common page replacement algorithms such as FIFO, LRU, and Clock, elucidating their impact on system performance and efficiency.

Additionally, the abstract discusses the challenges and optimizations associated with paging, including TLB (Translation Lookaside Buffer) management, page table organization, and memory access patterns. It explores techniques for minimizing page faults and optimizing memoryaccess times to enhance system responsiveness.

Lastly, the abstract discusses real-world implementations of paging in popular operating systems like Linux, Windows, and macOS, highlightingtheir unique approaches and optimizations.

TABLE OF CONTENTS

INTRODUCTION

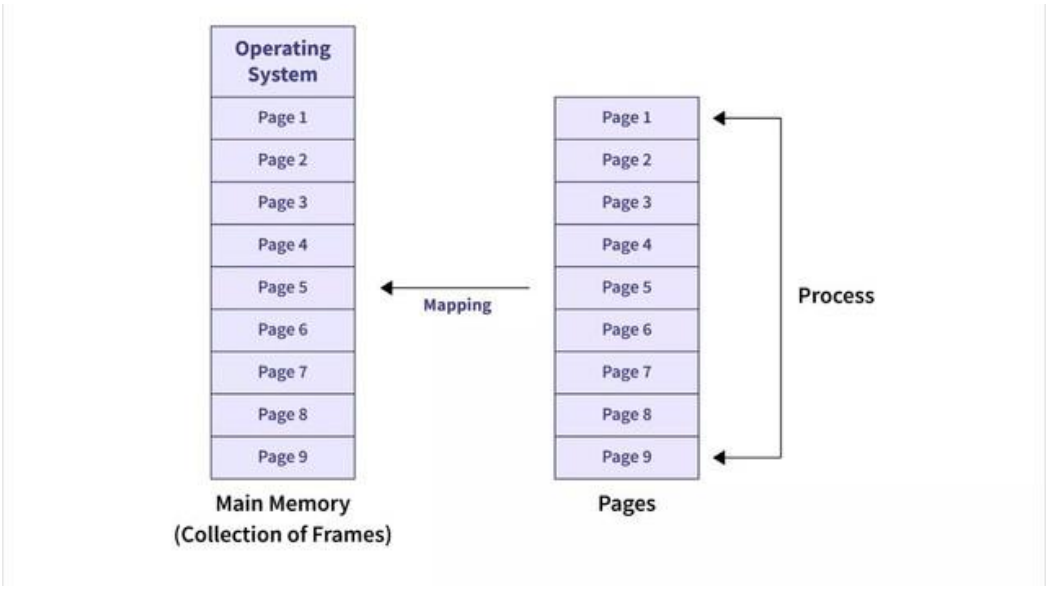
WHAT IS PAGING ?

In computer organization, paging refers to a memory management technique used by the CPU and the operating system to handle virtual memory. Virtual memory allows a computer to use more memory than is physically available by transferring data to and from the slower storage devices like hard drives or solid-state drives.

In this context, paging involves breaking up the computer's physical memory into fixed-size blocks called "frames" and breaking up the logical memory used by programs into similarly sized blocks called "pages." The operating system maintains a page table that maps the virtual pages used by a program to the physical frames in memory.

When a program accesses memory, the CPU generates a virtual address. The operating system translates this virtual address into a physical address using the page table. If the desired page is not currently in memory, a page fault occurs, and the operating system must retrieve the required page from secondary storage into a free frame in physical memory.

Paging allows for more efficient memory management because it enables processes to use memory that is not necessarily contiguous. It also facilitates memory protection and sharing among processes.



In computer organization courses, students often learn about the inner workings of paging systems, including page tables, page replacement algorithms, and the mechanisms for handling page faults. Understanding paging is crucial for designing efficient memory management systems and improving system performance.

IMPORTANCE OF PAGING

- ***Memory Management:*** Paging facilitates efficient memory management by dividing physical memory into fixed-size blocks called pages. This division enables the operating system to allocate and deallocate memory in smaller increments, reducing fragmentation and optimizing memory utilization. By dynamically mapping virtual memory addresses to physical addresses, paging allows processes to access memory in a seamless and organized manner.

- ***Virtual Memory:*** Paging enables the implementation of virtual memory, which provides each process with a contiguous and private address space, regardless of the physical memory's actual layout. Virtual memory allows processes to access more memory than is physically available, thereby enabling the execution of large programs and multitasking without requiring all programs to fit entirely in RAM.

- ***Protection and Isolation:*** Paging facilitates memory protection and process isolation by assigning each process its own virtual address space. This ensures that processes cannot interfere with each other's memory, enhancing system stability and security. Paging mechanisms, such as page tables and access permissions, enable the operating system to enforce memory protection policies and prevent unauthorized access to memory regions.

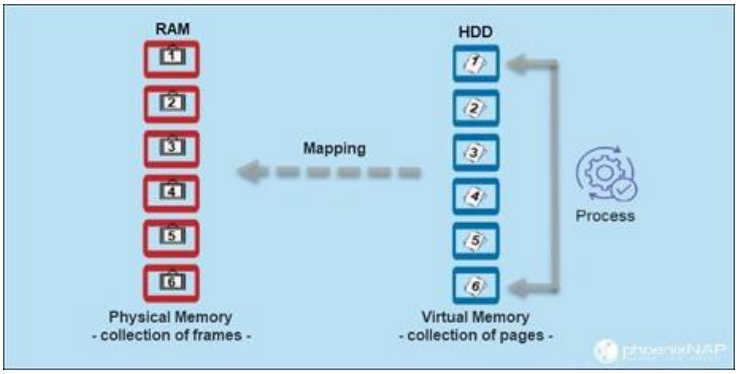
- ***Optimized Performance:*** Paging contributes to optimized system performance by enabling efficient memory access and minimizing the impact of memory-related bottlenecks. Through techniques like demand paging and page replacement algorithms, operating systems can prioritize memory access, prefetch frequently accessed pages, and manage memory resources effectively. This results in reduced disk I/O operations, minimizedpage faults, and improved overall system responsiveness.

- ***calability and Flexibility:*** Paging provides scalability and flexibility in managing memory resources, allowing operating systems to adapt to varying workload demands and resource constraints. The dynamic allocation and deallocation of memory pages enable efficient utilization of available resources, even in environments with limited physical memory. Additionally, paging facilitates the implementation of advanced memory management features, such as shared memory, memory-mapped files, and memory swapping, enhancing the versatility of operating systems.

- ***Support for Large Address Spaces:*** Paging allows operating systems to support large address spaces beyond the physical memory capacity. By using virtual memory addressing, processes can access a much larger address space than the available physical memory. This enables the execution of memory-intensive applications and facilitates the handling ofextensive data sets without requiring all data to be loaded into RAM simultaneously.

- ***Dynamic Memory Allocation:*** Paging facilitates dynamic memory allocation by allowing processes to request memory as needed and freeing up memory when it is no longer required. This dynamic allocation of memory pages enables efficient use of memory resources, minimizing wastage and fragmentation. Operating systems can allocate memory pageson-demand, ensuring that resources are allocated only when necessary.

- ***Improved Reliability and Fault Tolerance:*** Paging enhances system reliability and fault tolerance by isolating processes and protecting critical system memory. By assigning each process its own virtual address space, paging prevents processes from accessing unauthorized memory regions and minimizes the risk of system crashes due to memory corruption.
  Additionally, paging mechanisms can detect and handle memory access violations, preventing processes from accessing invalid memory addresses.



PAGE FAULT

A page fault occurs when a program attempts to access a portion of memory that is currently not loaded in physical RAM, necessitating the operating system to retrieve the required page from secondary storage (such as a hard disk) into RAM. This process is essential for virtual memory systems, allowing efficient memory management by swapping data between RAM and disk as needed.

Causes of Page Faults:

- ***Demand Paging:***

  Occurs when a program requests a page of memory that is not currently in RAM but exists in secondary storage.
  The operating system then retrieves the required page from secondary storage into RAM.

- ***Copy-on-Write:***

  Involves sharing memory pages between processes until one of them modifies the content.
  When a modification occurs, the operating system creates a separate copy of the page for the process, leading to a page fault when accessing the modified page.

- ***Stack Growth:***

  When a program's stack requires additional memory, but the current stack allocation is not sufficient, a page fault occurs.
  The operating system then allocates more memory for the stack, triggering the page fault.

- ***File Mapping:***

  Involves mapping a file's contents directly into memory.
  If a portion of the mapped file is accessed but not currently loaded into RAM, a page fault occurs, prompting the operating system to load the required page into memory.

- ***Page Replacement:***

  Occurs when the operating system needs to swap out a page from RAM to make space for a new page.
  If the swapped-out page is subsequently accessed, a page fault occurs, requiring the operating system to reload the page from secondary storage.

HOW TO HANDLE PAGE FAULTS ?

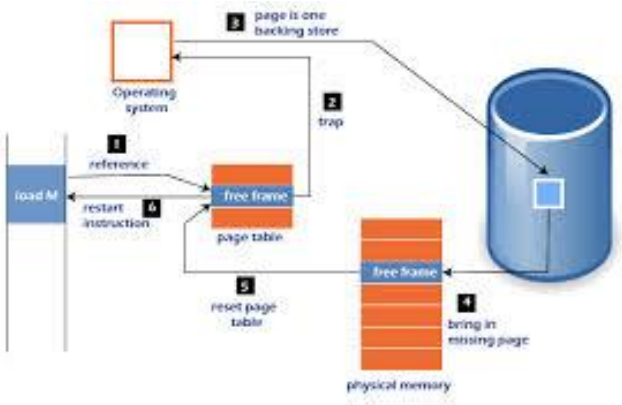
Page faults are a common occurrence in virtual memory systems, and handling them efficiently is crucial for maintaining system performance and responsiveness. When a page fault occurs, the operating system must manage the retrieval of the required page from secondary storage into RAM while minimizing the impact on overall system performance. Several strategies are employed to handle page faults effectively:

- ***Page Replacement Algorithms:***

Operating systems employ algorithms such as Least Recently Used (LRU), First-In-First-Out (FIFO), and Clock to determine which pages to evict from RAM when space is needed. LRU replaces the least recently used page, ensuring that frequently accessed pages remain in memory for faster access. FIFO, on the other hand, replaces the oldest page in memory, maintaining a simple and predictable eviction policy. The Clock algorithm maintains a circular list of pages and replaces the first page encountered that is not referenced, providing a balance between simplicity and efficiency.

- ***Prefetching:***

Predictive algorithms anticipate future memory accesses and preemptively load pages into RAM before they are requested. By analyzing past access patterns and predicting future ones, prefetching significantly reduces the latency of subsequent page faults, improving overall system performance and responsiveness.



- ***Optimized Page Placement:***

Operating systems prioritize placing frequently accessed pages in physical RAM to minimize page faults. Techniques like page coloring and hot/cold page separation are utilized to identify and segregate pages based on their access frequency and importance. By ensuring that frequently accessed pages are readily available in memory, optimized page placement reduces the likelihood of page faults and improves overall system efficiency.

- ***Page Clustering:***

Groups related pages together in memory to reduce the number of page faults. By clustering pages with high spatial locality, such as those belonging to the same process or data structure, page clustering enhances memory access patterns and reduces disk I/O overhead. This optimization strategy improves system performance by minimizing the need for costly disk accesses during page faults.

HOW PAGING IS DONE ?

Paging is a memory management scheme that eliminates the need for a contiguous allocation of physical memory. The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging. The basic purpose of paging is to separate each procedure into pages. Additionally, frames will be used to split the main memory. This scheme permits the physical address space of a process to be non – contiguous.

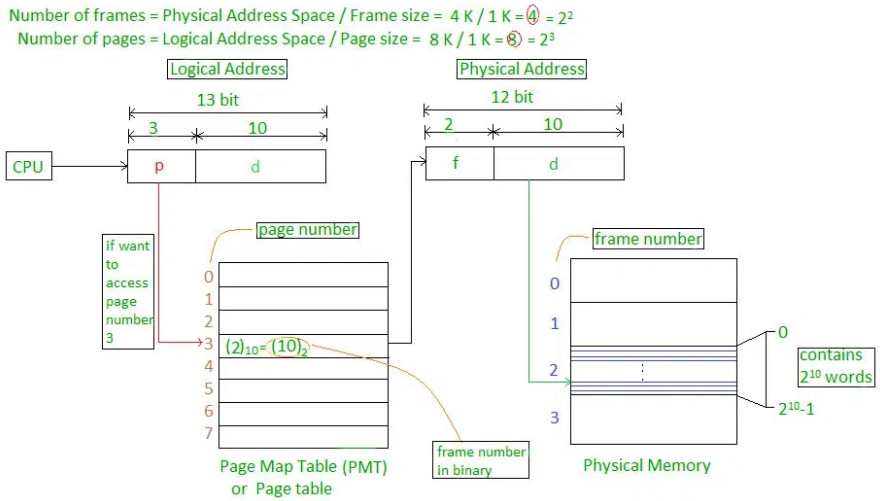
In paging, the physical memory is divided into fixed-size blocks called page frames, which are the same size as the pages used by the process. The process's logical address space is also divided into fixed-size blocks called pages, which are the same size as the page frames. When a process requests memory, the operating system allocates one or more page frames to the process and maps the process's logical pages to the physical page frames.

The mapping between logical pages and physical page frames is maintained by the page table, which is used by the memory management unit to translate logical addresses into physical addresses. The page table maps each logical page number to a physical page frame number.

In a paging scheme, the logical deal with the region is cut up into steady-duration pages, and every internet web page is mapped to a corresponding body within the physical deal with the vicinity. The going for walks tool keeps a web internet web page desk for every method, which maps the system's logical addresses to its corresponding bodily addresses. When a method accesses memory, the CPU generates a logical address, that is translated to a bodily address using the net page table. The reminiscence controller then uses the physical cope to get the right of entry to the reminiscence.
Let us consider an example:

EXAMPLE

Physical Address = 12 bits, then Physical Address Space = 4 K words
Logical Address = 13 bits, then Logical Address Space = 8 K words
Page size = frame size = 1 K words (assumption)



MEMORY PROTECTION AND SHARING

Paging in operating systems is a memory management scheme that divides the virtual memory used by processes into fixed-size blocks called pages. It enables efficient allocation of physical memory by mapping these pages to corresponding blocks of physical memory called frames. Paging allows for memory protection by assigning access permissions to each page and facilitates process isolation by providing each process with its own virtual address space..

- ***Memory Protection:*** One of the key features enabled by paging is memory protection, which ensures that processes cannot access memory regions that they are not authorized to access. When a program attempts to access a memory address, the CPU checks the corresponding page table entry to verify that the access is allowed based on the permissions set for that page. If the access violates the permissions, the CPU raises a memory access violation exception, halting the program and preventing unauthorized memory access. Memory protection is essential for ensuring the security and stability of the system.

- ***Memory Sharing:*** Paging also facilitates memory sharing among processes, allowing multiple processes to access the same physical memory pages. Memory sharing is advantageous for several reasons:

- ***Interprocess Communication (IPC):*** Shared memory can be used as a fast and efficient mechanism for communication between processes. Processes can write data to shared memory regions, and other processes can read or modify the data, enabling efficient data exchange without the need for complex communication mechanisms.

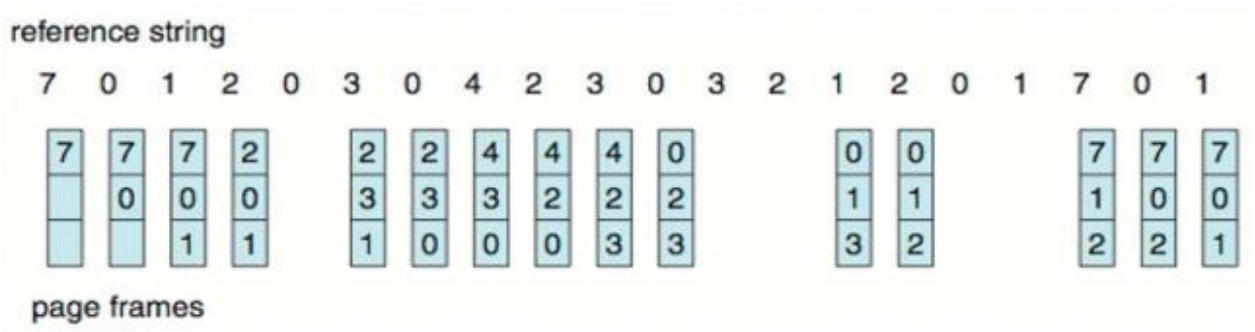
- ***Code Sharing:*** Shared libraries and code segments can be mapped into the address space of multiple processes, allowing them to share executable code. This reduces memory consumption and improves system performance by avoiding redundant copies of code in memory.

- ***Copy-on-Write (COW):*** Paging enables the implementation of copy-on-write mechanisms, where multiple processes initially share the same memory pages, but when one process attempts to modify a shared page, a copy of the page is created for that process. This optimization reduces memory overhead and improves performance by delaying the allocation of physical memory until it is actually needed.

PAGE REPLACEMENT ALGORITHMS

Page replacement algorithms are essential components of memory management in operating systems. Common algorithms include FIFO (First-In-First-Out), LRU (Least Recently Used), and Optimal.

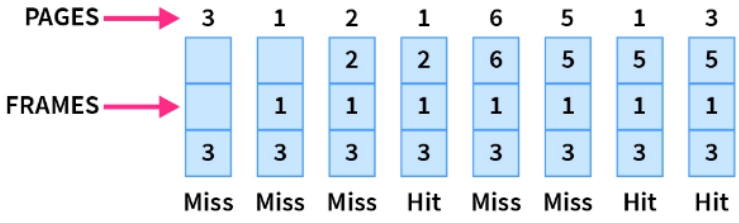
- ***FIFO (First-In-First-Out) Page Replacement Algorithm:***

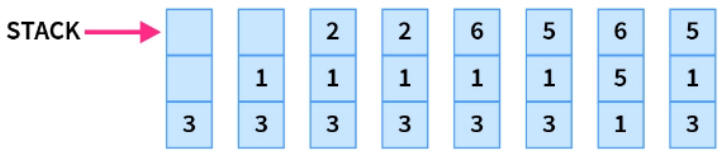


FIFO (First-In-First-Out)

Is a basic page replacement algorithm in memory management.
It replaces the oldest page in memory when a new page is required.
Upon system initialization, an empty queue is set up to track pages in memory.
When a page fault occurs, indicating the need for a new page, FIFO selects the oldest page.

If memory is full, FIFO removes the oldest page to make space for the new one.
The new page is then inserted into memory, and the page table is updated accordingly.
LIFO (Last-In-First-Out) Page Replacement Algorithm:

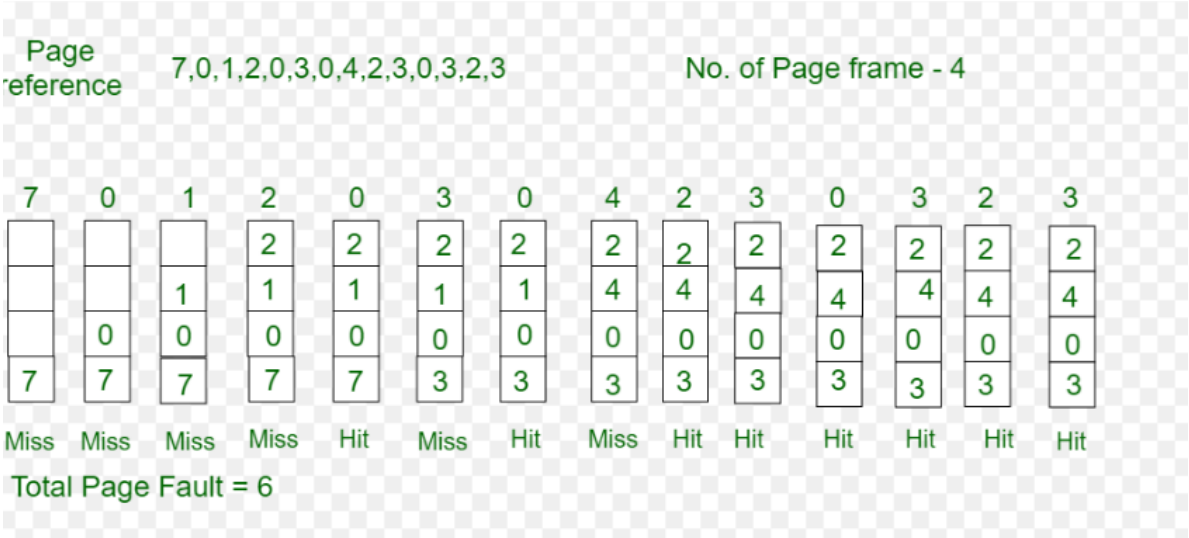




LIFO (Last-In-First-Out)

Is a page replacement algorithm that replaces the most recently brought-in page when a new page is needed.
It operates on the principle of retaining the latest pages in memory.
Upon system initialization, an empty stack is created to track pages.
When a page fault happens, indicating the need for a new page, LIFO selects the page most recently brought into memory.
If memory is full, LIFO removes the page that was most recently added to make space for the new page.
The new page is then inserted into memory, and the page table is updated accordingly.

LRU(Least Recently Used) Page Replacement Algorithm:



LRU (Least Recently Used) is a page replacement algorithm used in memory management systems. It operates on the principle that the page that has not been accessed for the longest time is the least likely to be accessed in the near future and thus can be chosen for replacement. Here's how LRU works:

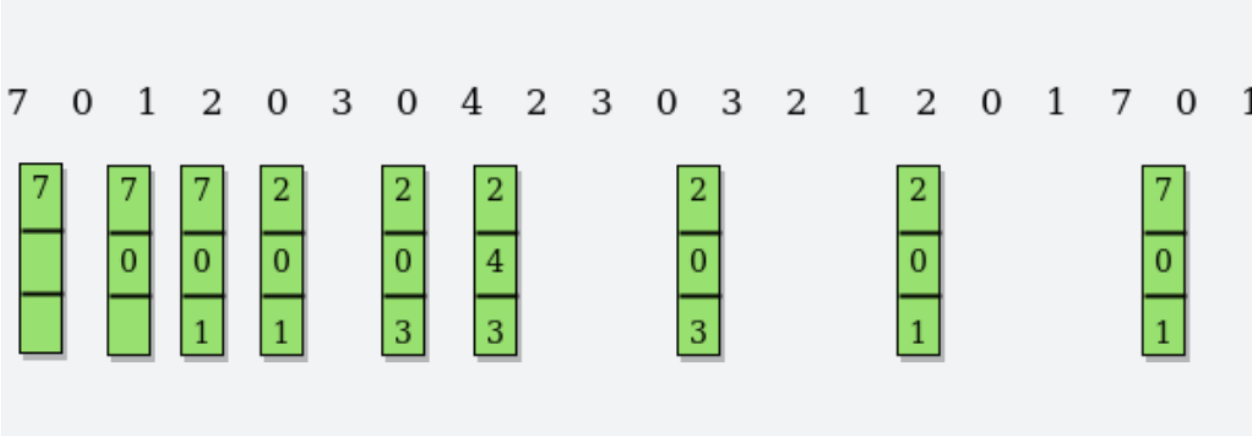
Initialization: When the system starts, it sets up a data structure, often a queue or a linked list, to keep track of the access history of pages.
Page Access: Each time a page is accessed, it is moved to the front of the data 8 that it is the most recently used page.
Page Replacement: When a page fault occurs and a new page needs to be loaded into memory, LRU selects the page that is at the end of the data structure, indicating that it has not been accessed for the longest time.

Updating Access History: After selecting the page for replacement, the new page is inserted into memory, and the access history data structure is updated accordingly

- LRU is favored for its ability to approximate the optimal page replacement strategy by prioritizing pages based on their recent access history. However, implementing LRU efficiently can be challenging due to the overhead of maintaining the access history data structure. Nonetheless, LRU is widely used in various memory management systems to improve overall performance by minimizing the occurrence of page faults.

Optimal (Belady's) Page Replacement Algorithm:

The Optimal page replacement algorithm is theoretically the best approach, as it always selects the page that will result in the fewest number of page faults in the future.



Prediction: Optimal predicts which page in memory will not be accessed for the longest time in the future.

Future Access Analysis: It requires knowledge of future memory accesses, which makes it impractical for implementation in real systems.

Page Replacement: When a page fault occurs and a new page needs to be loaded into memory, Optimal selects the page that will not be accessed for the longest time in the future.

Perfect Replacement: Optimal achieves the lowest possible page fault rate among all page replacement algorithms because it always selects the optimal page for replacement.

Limitations: Despite its optimality, Optimal is not used in practice due to the impracticality of predicting future memory accesses in real-time operating systems.

Benchmark: Optimal is often used as a benchmark to evaluate the performance of other page replacement algorithms. Other algorithms are compared against Optimal to assess their effectiveness in minimizing page faults.
While Optimal provides insight into the best possible page replacement strategy, its reliance on future memory access patterns makes it unsuitable for practical implementation in real systems. Nonetheless, studying Optimal helps in understanding the theoretical limits of page replacement algorithms and aids in the design and evaluation of practical alternatives.

COMAPRISION OF ALL PAGE REPLACMENT ALORITHM

- ***Optimal Algorithm***
- ***Fifo Algorithm***
- ***Lru Algorithm***

Aspect	Optimal Algorithm	FIFO Algorithm	LRU Algorithm
Prediction of Future Access	Ideal but impractical	Doesn't predict future, replaces oldest page	Considers recent history, but not future
Complexity	Theoretical, impractical	Simple	Moderate
Practical Applicability	Not implementable	Widely used	Commonly used
Overhead	Not applicable	Low	Moderate
Performance	Optimal	May suffer from "Belady's Anomaly"	Balanced performance
Implementation	Theoretical concept	Easy to implement	Requires tracking recent accesses

ADVANCEMENTS IN PAGING TECHNOLOGY

Advancements in paging have been pivotal in enhancing the efficiency, scalability, and reliability of modern memory management systems within operating environments. These advancements encompass several key areas, each contributing to improved system performance and resource utilization:

- ***Page Size Optimization:***

Traditional paging systems typically employ fixed-size pages, but advancements have explored variable or dynamically sized pages.
Variable page sizes allow for more efficient memory utilization by aligning page sizes with the memory requirements of specific applications or data structures.

- ***Hierarchical Paging Structures:***

Hierarchical paging structures, such as multilevel page tables or inverted page tables, optimize memory access by organizing pages into hierarchical data structures.
This hierarchical organization reduces the memory overhead associated with large page tables, resulting in improved memory management efficiency.

- ***Transparent Huge Pages:***

Transparent Huge Pages (THP) provide a mechanism for automatically aggregating consecutive memory pages into larger "huge pages" transparently to applications.
THP reduces the overhead of managing a large number of small pages, improving memory access performance for memory-intensive applications.

- ***Page Coloring:***

Page coloring techniques aim to mitigate the negative effects of memory access contention by assigning pages of memory to specific processors or threads.
By reducing contention for memory access, page coloring enhances overall system performance and scalability in multicore or multiprocessor environments.

- ***Non-Uniform Memory Access (NUMA) Awareness:***

NUMA-aware paging algorithms optimize memory access patterns in NUMA architectures by considering the non-uniform latency associated with accessing memory across different nodes.
These algorithms strive to minimize memory access latency and maximize locality of reference in NUMA systems, improving overall system performance.

- ***Hardware Support:***

Advancements in hardware support, such as Memory Management Units (MMUs) and TLB (Translation Lookaside Buffer) optimizations, enhance the efficiency and scalability of paging operations

ADVANTAGES OF PAGING

- ***Efficient Memory Utilization:*** Paging divides physical memory into fixed-size blocks, enabling optimal allocation and utilization of memory resources, reducing wastage.

- ***Enhanced Multitasking:*** Each process is assigned its own virtual address space, allowing multiple processes to run concurrently without interference, thus improving system performance and responsiveness.

- ***Simplified Memory Management:*** Paging simplifies memory allocation by providing a uniform interface for memory access, eliminating the need for manual memory management tasks, such as memory fragmentation and relocation.

- ***Dynamic Resource Allocation:*** Paging supports dynamic allocation and deallocation of memory pages, allowing the system to efficiently manage memory resources based on the changing requirements of processes, maximizing system efficiency.

DISADVANTAGE OF PAGING

- ***Overhead:*** Paging introduces overhead due to the need for maintaining page tables, resulting in additional memory and processing requirements.

- ***Fragmentation:*** Paging can lead to fragmentation of physical memory, as pages are allocated and deallocated dynamically, potentially reducing memory efficiency over time.

- ***Page Faults:*** Excessive page faults can occur if the system's page replacement algorithm is inefficient, leading to decreased performance and increased response times.

- ***Complexity:*** Paging adds complexity to memory management, especially in systems with multiple levels of paging or advanced page replacement algorithms, which can be challenging to implement and optimize.

APPLICATIONS OF PAGING

Paging, as a fundamental memory management technique, finds application in various computing environments, ranging from desktop operating systems to large-scale server systems and embedded devices. Here are some key applications of paging along with examples:

- ***Desktop Operating Systems:***

Paging is extensively used in desktop operating systems like Windows, macOS, and Linux to manage virtual memory efficiently.
For example, when multiple applications are running simultaneously on a computer, paging allows the operating system to allocate and manage memory resources dynamically, ensuring smooth multitasking without running out of physical memory.

- ***Server Systems:***

Paging plays a crucial role in server systems that handle heavy workloads and large datasets.
In web servers, for instance, paging enables efficient handling of multiple client requests by dynamically allocating memory to different server processes as needed. This ensures optimal performance and responsiveness under varying load conditions.

- ***Database Management Systems (DBMS):***

DBMSs utilize paging to manage large databases that cannot fit entirely into main memory.
Paging allows the DBMS to retrieve and store data from secondary storage (e.g., disk) in smaller, manageable chunks known as pages. This enables efficient data access and manipulation, even for datasets that exceed the available physical memory.

- ***Embedded Systems:***

Paging is also employed in embedded systems with limited memory resources, such as smartphones, IoT devices, and embedded controllers.
In smartphones, for example, paging enables the efficient management of applications and system resources, ensuring optimal performance within the constraints of the device's memory capacity.

- ***Virtualization Technologies:***

Virtualization platforms like VMware, Hyper-V, and KVM utilize paging to provide virtual machines (VMs) with isolated and efficient memory management.
Paging allows virtualization platforms to allocate and manage memory resources for multiple VMs concurrently, ensuring efficient resource utilization and isolation between VMs.

After Deep Down Research And Learning Of Our Topic

PAGING IN OPERATING SYSTEM

We gain some knowledge about :

- ***Page Size Impact:*** Smaller page sizes increase overhead, while larger sizes lead to more internal fragmentation.
- ***Algorithm Effectiveness:*** LRU outperforms FIFO in reducing page faults.
- ***Workload Influence:*** Workloads with high locality of reference experience fewer page faults.
- ***Scalability Challenges:*** Higher process and memory demands result in increased thrashing and degraded performance.
- ***Memory Utilization Trade-offs:*** Paging balances efficient memory use with overhead considerations.
- ***Hardware Architecture Influence***: Differences in hardware configurations affect paging algorithm performance.
- ***Optimization Opportunities:*** Implementing memory-mapped files and shared memory regions can enhance system performance.TLB Size Impact: Smaller TLBs result in more frequent TLB misses and increased overhead.
- ***Page Fault Handling Overhead:*** Handling page faults incurs resource overhead, especially under heavy loads.
- ***Locality Influence:*** Workloads with high temporal and spatial locality experience fewer page faults.
- ***Swap Space Configuration:*** Inadequate swap space can lead to disk thrashing and performance degradation.
- ***Real-time Systems:*** Paging overhead is critical in real-time systems, impacting system predictability and responsiveness

Conclusion

In computer organization, paging stands out as a cornerstone technique for managing memory, particularly when dealing with large address spaces. It functions by dividing both primary memory (RAM) and secondary storage (like a hard disk) into fixed-size blocks called frames and pages, respectively. This approach enables efficient memory utilization by loading only the relevant pages of a process into main memory at any given time. This avoids loading the entire process, freeing up memory for other tasks.

Paging offers several advantages. It simplifies memory allocation by eliminating external fragmentation, a common issue where scattered free memory blocks become unusable due to their small sizes. Additionally, paging enhances protection between processes by isolating their address spaces. Changes made by one process are confined to its allocated pages, preventing unintended modifications to other processes' data, thus promoting system stability.

Reference

https://en.wikipedia.org/wiki/Memory_paging

https://www.geeksforgeeks.org/paging-in-operating-system/

https://www.slideshare.net/infomerlin/pagingppt-259902929

https://chat.openai.com/c/5cd2270e-d023-4d5a-b178-c8445405527d

https://gemini.google.com/app/

Operating System Concepts by Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne

Computer Organization and Design by Patterson and Hennessy

THANK YOU