RV Educational Institutions ®
RV College of Engineering ®

Autonomous Institution
Affiliated to Visvesvaraya
Technological University,
Belagavi

Approved by AICTE,
New Delhi

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**OPERATING SYSTEMS - CS235AI**

**REPORT**

# VIRTUAL MEMORY OPTIMIZATION TOOL

**Submitted by**

| | | |
|---|---|---|
| **OM GUPTA** | - | **1RV22CS133** |
| **RAHEEL JAWED** | - | **1RV22CS155** |
| **N. SASIDAR** | - | **1RV22CS123** |

**Computer Science and Engineering**
**2023-2024**

## CONTENTS

# INTRODUCTION

In the realm of modern computing, the efficient management of memory is paramount for smooth and responsive system operation. Virtual memory optimization stands at the forefront of this endeavour, offering a sophisticated approach to maximize resource utilization while minimizing overhead. This introduction delves into the fundamental principles and strategies employed in the optimization of virtual memory systems.

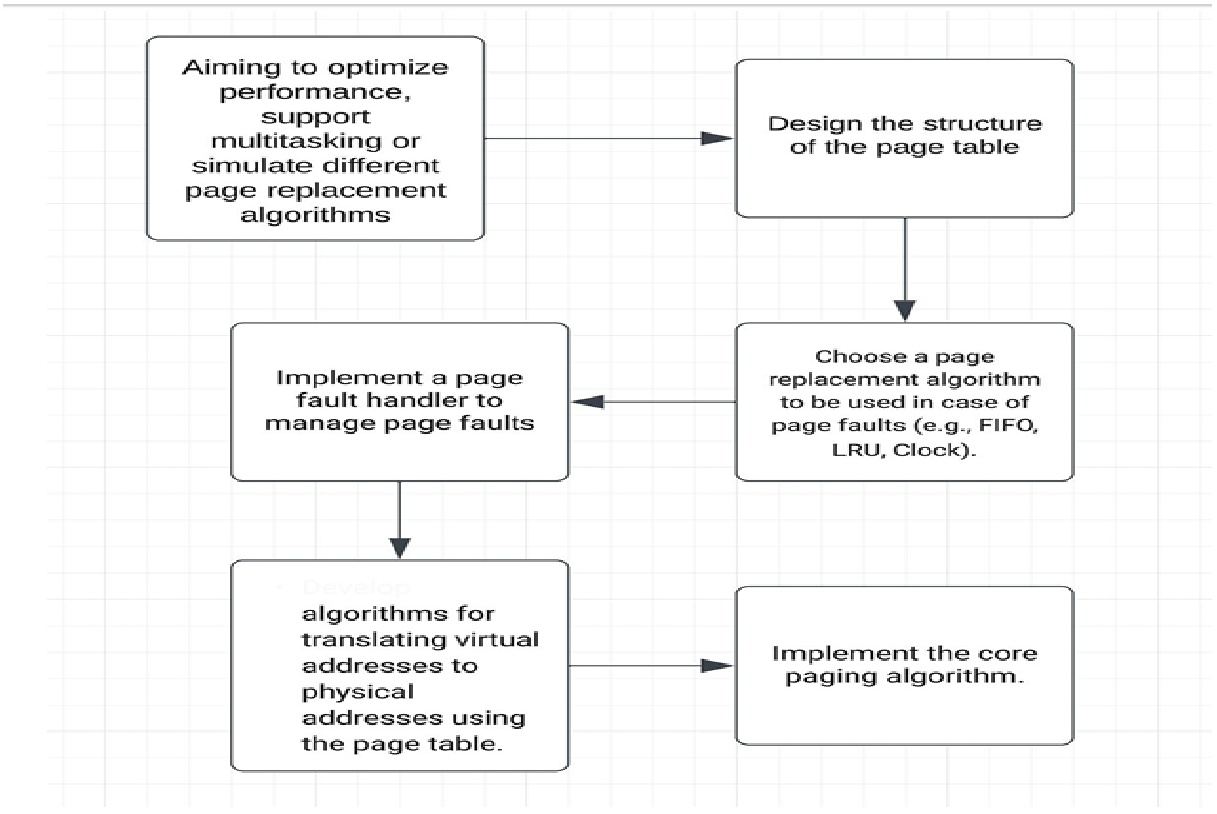
Virtual memory serves as a pivotal abstraction layer, enabling systems to transcend the constraints of physical RAM by utilizing disk space as an extension. This technique allows for the illusion of a vast address space, facilitating the execution of processes larger than the available physical memory. At its core, virtual memory optimization aims to strike a delicate balance between performance and resource utilization. By intelligently managing memory allocations, paging mechanisms, and cache hierarchies, optimization endeavours to enhance system responsiveness, throughput, and overall efficiency.

Central to virtual memory optimization are algorithms that determine which pages to evict from physical memory when space becomes constrained. Popular algorithms such as Clock Algorithm, Least Recently Used (LRU), First-In-First-Out (FIFO), and Not Recently Used (NRU) play a crucial role in maintaining an optimal balance between page residency and access latency

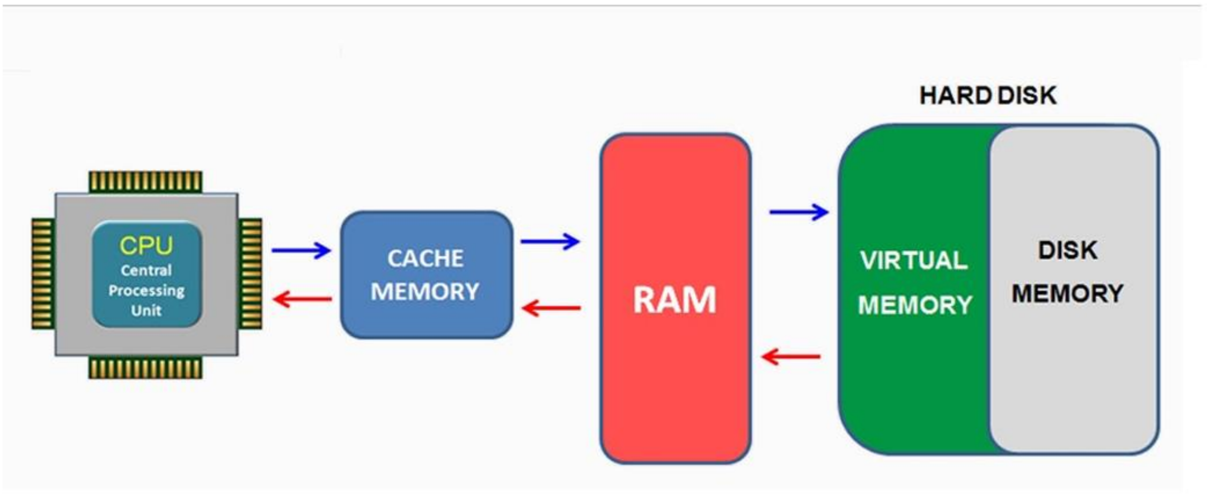
# SYSTEM ARCHITECTURE

- **Memory Management Unit (MMU):** The MMU is responsible for translating virtual addresses to physical addresses and managing memory access permissions.
- **Memory Manager:** The memory manager oversees the allocation and deallocation of physical memory pages. It interfaces with the MMU to manage page tables and control the mapping of virtual memory to physical memory. Memory manager also interacts with the operating system kernel to handle memory requests from processes and manage memory resources effectively.
- **Page Replacement Algorithms Module:** This module implements various page replacement algorithms such as LRU, FIFO, or NRU. It monitors page accesses and makes decisions on which pages to evict from physical memory when space becomes limited.
- **Working Set Analysis Module:** This module analyses the working set of processes to identify the set of pages actively used during execution. It provides insights into process memory behaviour and informs memory management decisions to prioritize critical pages.
- **Swap Space Manager:** The swap space manager is responsible for managing the disk space used for virtual memory paging. It allocates and deallocates swap space as needed, and may implement mechanisms for optimizing swap space utilization.

# METHODOLOGY



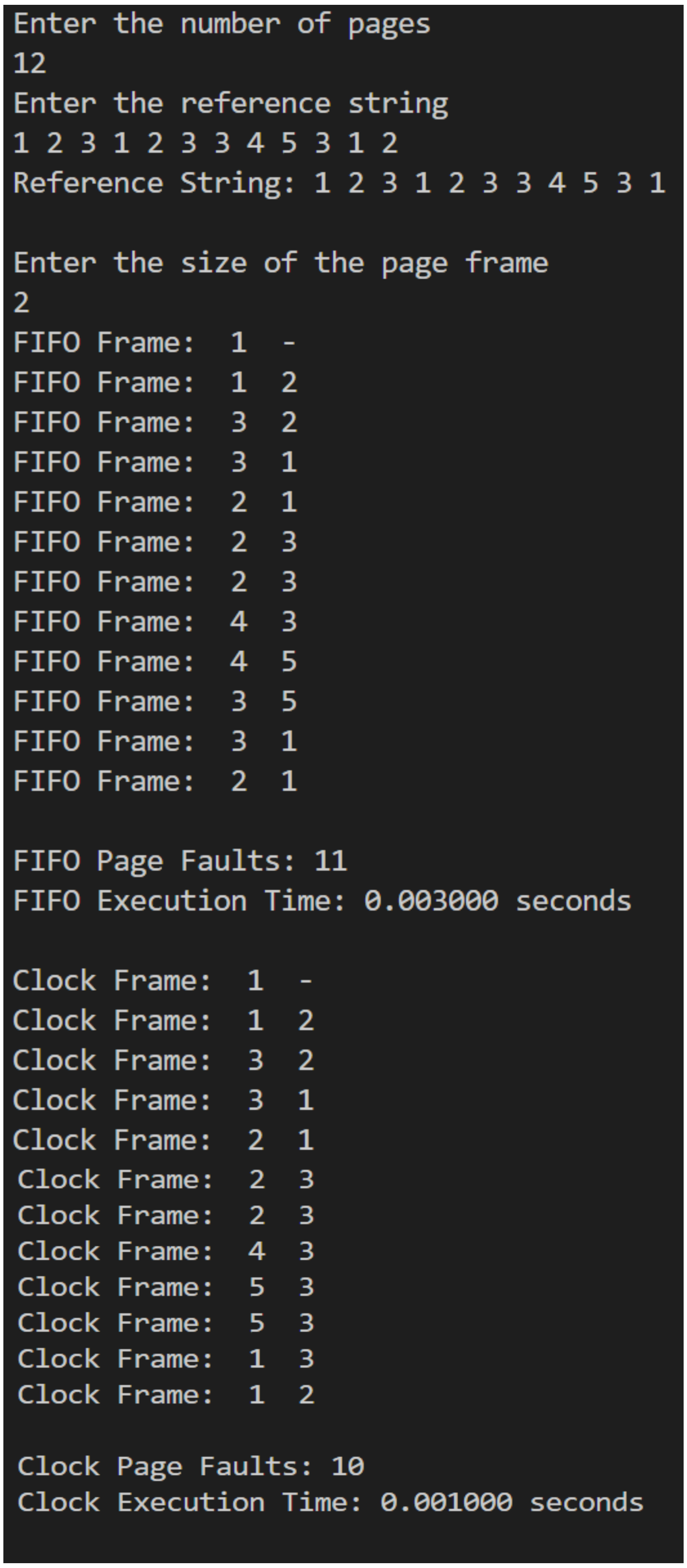
Aiming to optimize performance, support multitasking or simulate different page replacement algorithms
Design the structure of the page table
Choose a page replacement algorithm to be used in case of page faults (e.g., FIFO, LRU, Clock).
Implement a page fault handler to manage page faults
Develop
algorithms for translating virtual addresses to physical addresses using the page table.
Implement the core paging algorithm.



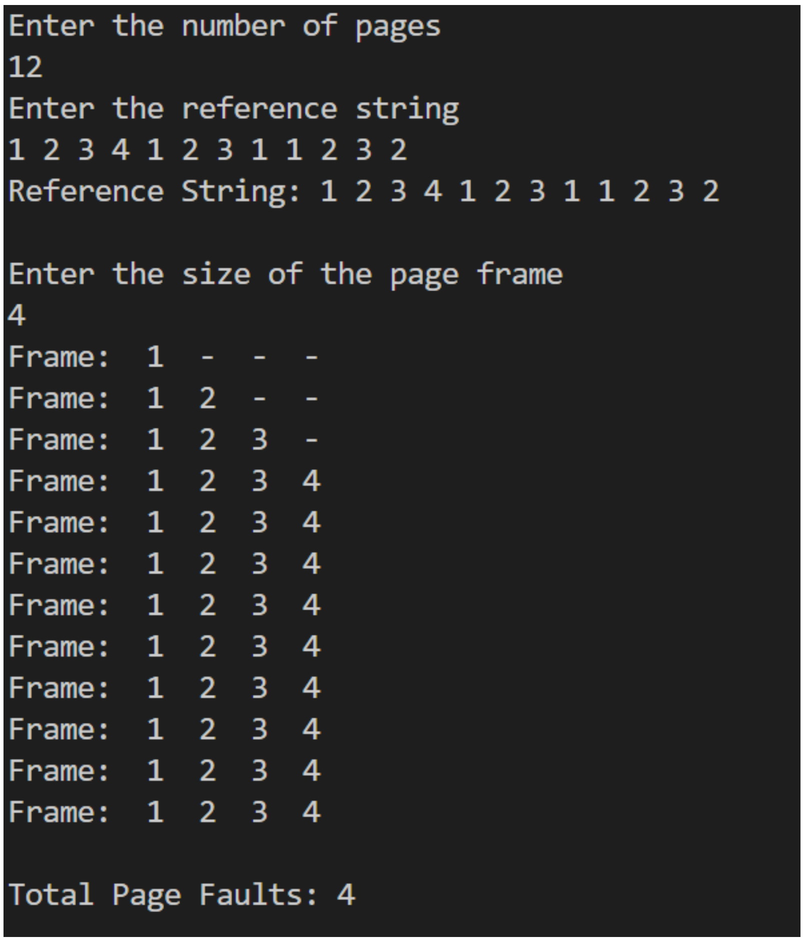
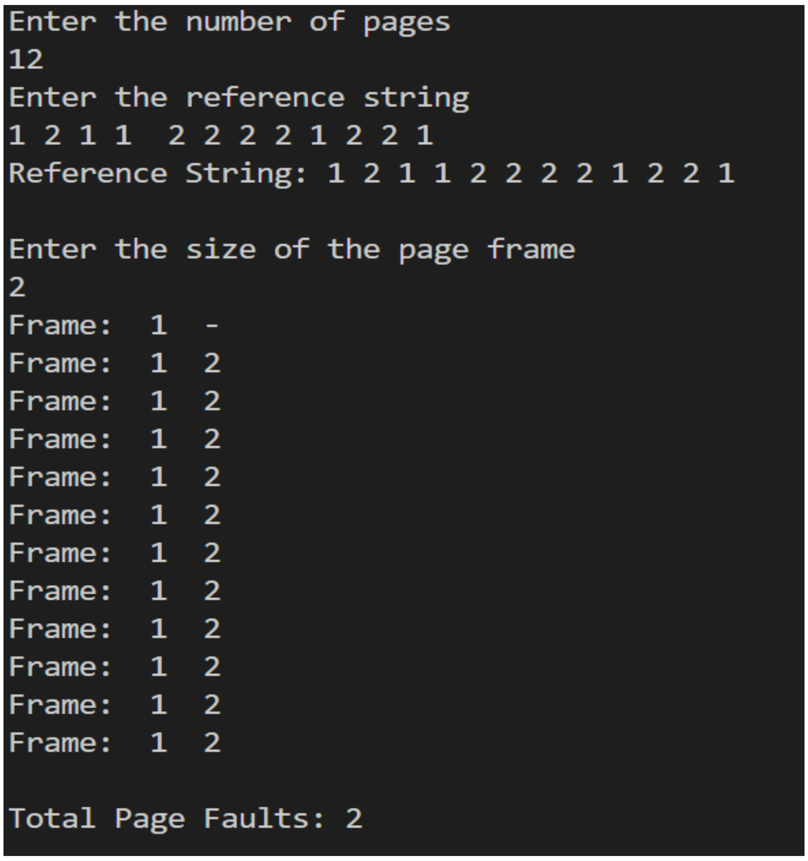
## SYSTEM CALLS

- **mmap() and munmap():** mmap() is used to map files or devices into memory, allowing processes to access them as if they were part of the process's address space. munmap() is used to unmap memory regions previously mapped with mmap(), releasing memory resources.
- **madvise():** The madvise() system call is used to provide advice or hints to the kernel about the intended usage of memory regions.
- **mlock() and munlock():** mlock() is used to lock memory pages into physical memory, preventing them from being swapped out to disk. munlock() is used to unlock previously locked memory pages, allowing them to be swapped out if necessary.
- **mincore():** The mincore() system call allows a process to query the status of memory pages to determine whether they are resident in physical memory, cached in memory, or swapped out to disk.
- **mremap():** mremap() allows a process to remap a memory region to a different address or resize the region.

## OUTPUT



```
Enter the number of pages
12
Enter the reference string
1 2 3 1 2 3 3 4 5 3 1 2
Reference String: 1 2 3 1 2 3 3 4 5 3 1

Enter the size of the page frame
2
FIFO Frame:  1  -
FIFO Frame:  1  2
FIFO Frame:  3  2
FIFO Frame:  3  1
FIFO Frame:  2  1
FIFO Frame:  2  3
FIFO Frame:  2  3
FIFO Frame:  4  3
FIFO Frame:  4  5
FIFO Frame:  3  5
FIFO Frame:  3  1
FIFO Frame:  2  1

FIFO Page Faults: 11
FIFO Execution Time: 0.003000 seconds

Clock Frame:  1  -
Clock Frame:  1  2
Clock Frame:  3  2
Clock Frame:  3  1
Clock Frame:  2  1
Clock Frame:  2  3
Clock Frame:  2  3
Clock Frame:  4  3
Clock Frame:  5  3
Clock Frame:  5  3
Clock Frame:  1  3
Clock Frame:  1  2

Clock Page Faults: 10
Clock Execution Time: 0.001000 seconds
```


```
Enter the number of pages
12
Enter the reference string
1 2 3 4 1 2 3 1 1 2 3 2
Reference String: 1 2 3 4 1 2 3 1 1 2 3 2

Enter the size of the page frame
4
Frame:  1  -  -  -
Frame:  1  2  -  -
Frame:  1  2  3  -
Frame:  1  2  3  4
Frame:  1  2  3  4
Frame:  1  2  3  4
Frame:  1  2  3  4
Frame:  1  2  3  4
Frame:  1  2  3  4
Frame:  1  2  3  4
Frame:  1  2  3  4
Frame:  1  2  3  4

Total Page Faults: 4
```

```
Enter the number of pages
12
Enter the reference string
1 2 1 1  2 2 2 2 1 2 2 1
Reference String: 1 2 1 1 2 2 2 2 1 2 2 1

Enter the size of the page frame
2
Frame:  1  -
Frame:  1  2
Frame:  1  2
Frame:  1  2
Frame:  1  2
Frame:  1  2
Frame:  1  2
Frame:  1  2
Frame:  1  2
Frame:  1  2
Frame:  1  2
Frame:  1  2

Total Page Faults: 2
```

# CONCLUSION

In conclusion, the realm of virtual memory optimization within operating systems represents a critical aspect of modern computing infrastructure. Through a cohesive system architecture and the utilization of specialized system calls, virtual memory optimization tools can effectively manage memory resources, enhance system performance, and ensure the efficient operation of computing environments.

By integrating components such as memory management units, page replacement algorithms, working set analysis modules, and swap space managers, these tools can intelligently allocate and deallocate memory, minimize fragmentation, and prioritize critical memory pages.

Additionally, configuration interfaces and performance evaluation modules enable administrators to fine-tune optimization strategies and parameters, ensuring optimal system performance under varying workloads.