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

TOPIC - VIRTUAL MEMORY MANAGEMENT

Submitted by

PAVITHRA N(1RV22CS137)
POSAM HARSHITHA (1RV22CS139)

Computer Science and Engineering 2023-2024

Submitted to

Dr.Jyoti Shetty
Assistant Professor
Department of Computer Science
R V College of Engineering

Table of Contents

INTRODUCTION

Virtual memory management is a cornerstone of modern operating systems, facilitating efficient memory utilization by abstracting physical memory through virtual addressing. This abstraction allows each process to have its own virtual address space, which appears contiguous and separate from other processes, enabling them to operate as though they have more memory than is physically available. Memory allocation techniques such as the Buddy algorithm, first fit, best fit, and worst fit dynamically assign memory blocks to processes, optimizing resource usage based on factors like fragmentation reduction and allocation speed. The Buddy algorithm divides memory into blocks of sizes that are powers of two, allowing for efficient allocation and deallocation of contiguous memory segments. First fit chooses the first available memory block that is large enough to satisfy a process's request, whereas best fit chooses the smallest available memory block, reducing wasted memory but potentially causing fragmentation. Worst fit selects the largest available memory block, which can reduce fragmentation but may result in less efficient memory usage.

Page allocation and deallocation mechanisms improve memory management by working at the granular level, managing memory in smaller units known as pages. Pages are fixed-size memory blocks, usually 4KB or 8KB in size, that serve as the unit of data transfer between disk and main memory. When a process accesses a page that is not currently in main memory, a page fault occurs, causing the operating system to load the needed page from disk into an available memory frame using demand paging. TLB (Translation Lookaside Buffer) caches recently used virtual-to-physical address translations, which accelerated address translation while decreasing the frequency of costly memory accesses to the page table. Modern operating systems can efficiently handle diverse workloads to ensure that applications run smoothly even in environments where physical memory resources are limited.

In addition to allocation, efficient page replacement algorithms are critical for optimizing memory utilization. Least Recently Used (LRU), First In, First Out (FIFO), and Optimal Paging algorithms are used to determine which pages to remove from main memory when new pages need to be loaded. LRU replaces the page that hasn't been accessed in the longest time, using temporal locality to predict future memory accesses. FIFO replaces the oldest page in memory, while Optimal Paging is theoretically optimal but impractical due to its requirement. of knowing future memory access patterns. Furthermore, displaying the page table with page numbers and frame numbers provides insight into the mapping of virtual addresses to physical memory locations, facilitating robust memory management and troubleshooting in complex operating system environments.

System Architecture:

System architecture in virtual memory management refers to the design and organization of hardware and software components that collectively manage virtual memory within an operating system. This architecture typically involves several key components:

- **Memory Management Unit (MMU):** The MMU is a hardware component responsible for translating virtual addresses used by the CPU into physical addresses in main memory. It utilizes techniques like page tables or segmentation to perform this translation efficiently.

- **Page Tables:**Page tables are data structures maintained by the operating system to map virtual addresses to physical addresses. They store information about which pages of virtual memory are currently mapped to which frames of physical memory.

- **Page Fault Handler:** The page fault handler is part of the operating system's kernel responsible for handling page faults, which occur when a process accesses a virtual memory page that is not currently in main memory. It initiates the loading of the required page from disk into memory.

- **Swap Space:** Swap space, also known as the paging file or swap partition, is a reserved area on disk used to store pages of memory that are not currently in use. When physical memory becomes full, pages may be swapped out to disk to free up space.

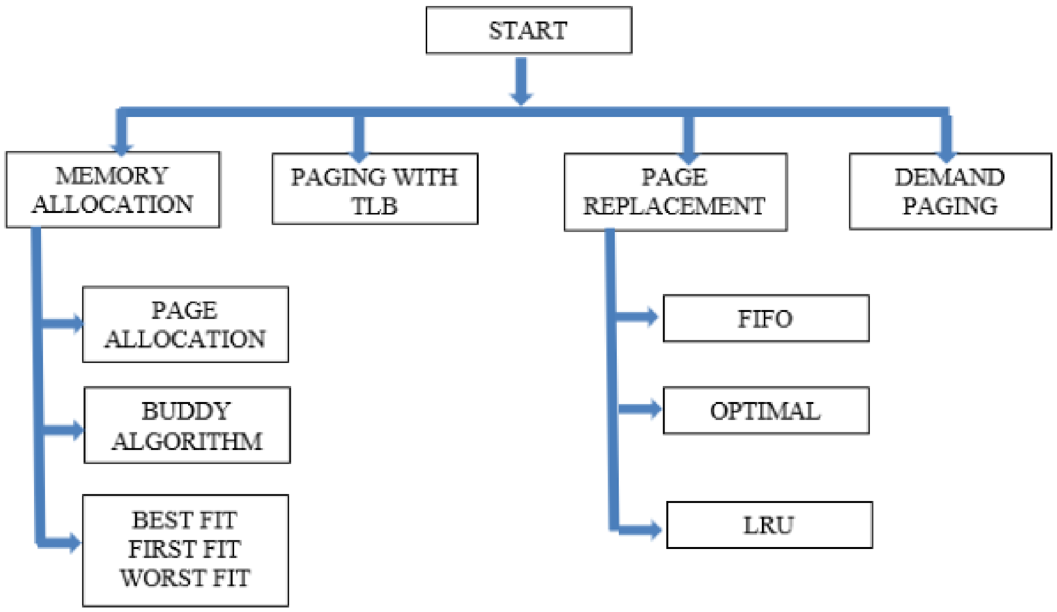
- **TLB (Translation Lookaside Buffer):** The TLB is a cache that stores recently used virtual-to-physical address translations, speeding up address translation and reducing the frequency of accesses to the page tables.

- **Memory Allocation Policies:** These policies determine how physical memory is allocated to processes and how pages are managed to optimize memory utilization and minimize fragmentation. Techniques like buddy allocation, first fit, best fit, and worst fit are commonly used.

- **Page Replacement Algorithms:** These algorithms determine which pages should be evicted from main memory when space is needed for loading new pages. Common algorithms include Least Recently Used (LRU), First In, First Out (FIFO), and Optimal Paging.

The system architecture for virtual memory management is designed to efficiently utilize physical memory resources, provide a uniform and contiguous address space for processes, and handle memory-related operations such as allocation, deallocation, and paging effectively.

Methodology:



System Calls Used:

Memory Management:

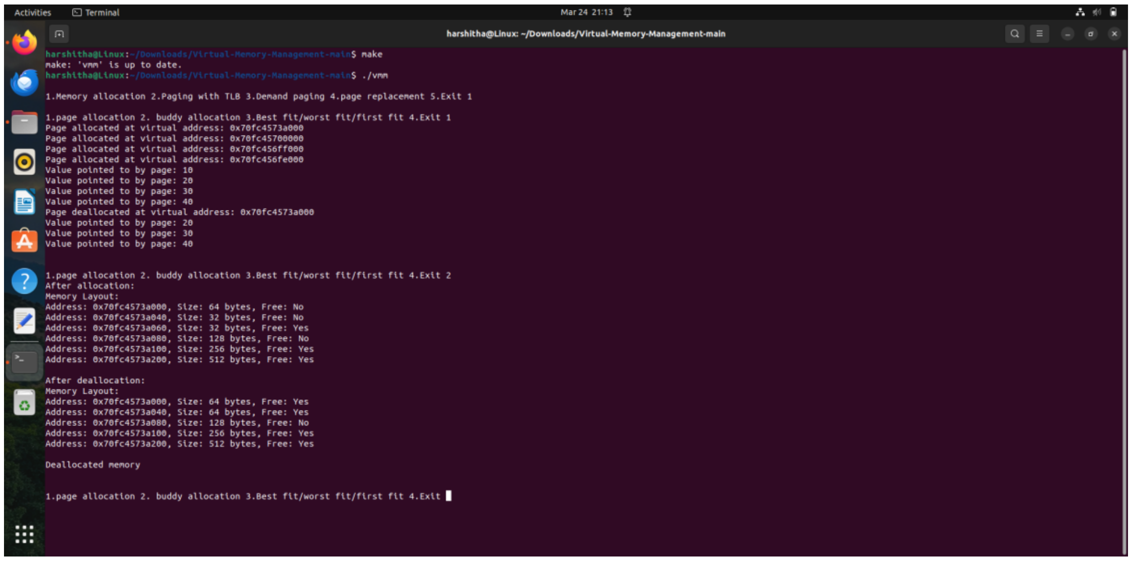
- **mmap()**: Memory Map - This system call is used to map files, devices, or memory segments into the calling process's address space. It allows for efficient access to files or devices as if they were part of the memory. mmap() returns a pointer to the mapped memory region upon success.
- **munmap():** Memory Unmap - This system call is used to remove mappings previously established by mmap(). It deallocates the memory region previously mapped by mmap().

File Management:

- **fopen():** File Open - This system call is used to open a file, creating a new file descriptor if successful. It takes a filename and a mode (such as "r" for reading, "w" for writing, etc.) as arguments and returns a pointer to a FILE structure, which is used for subsequent file operations.
- **fclose():** File Close - This system call is used to close a file that was opened with fopen(). It flushes any buffered data associated with the file descriptor and releases any resources associated with it.
- **fseek():** File Seek - This system call is used to set the file position indicator for the specified file stream. It allows the program to move the position from which the next fread() or fwrite() operation will occur.
- **fread():** File Read - This system call is used to read data from a file into memory. It takes as arguments a pointer to a buffer where the data will be stored, the size of each element to be read, the number of elements to read, and a file stream. It returns the number of elements successfully read.

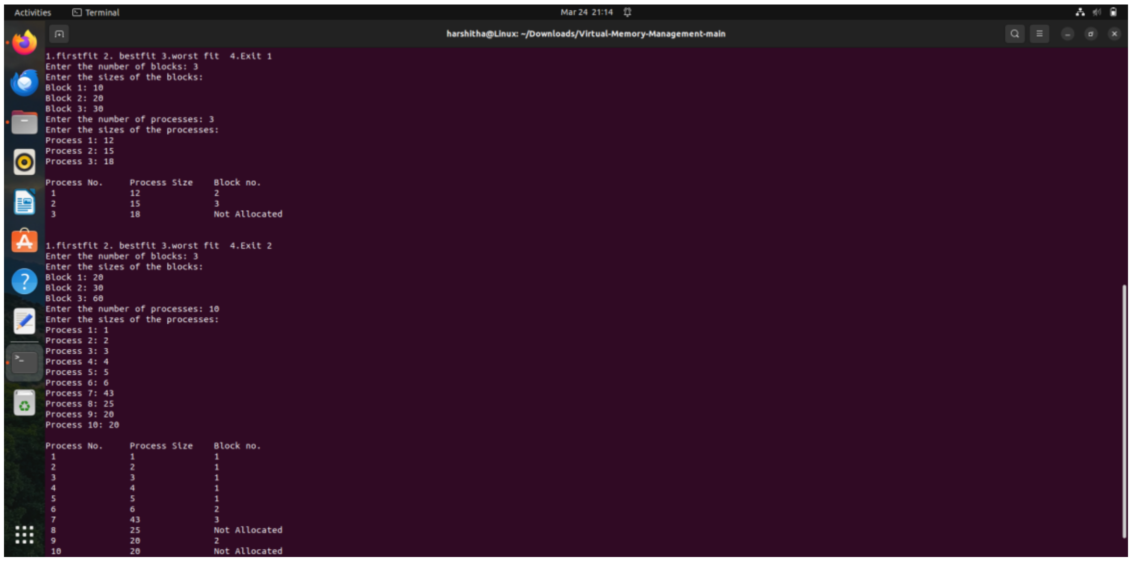
Output:

Memory Allocation and Buddy Algorithm



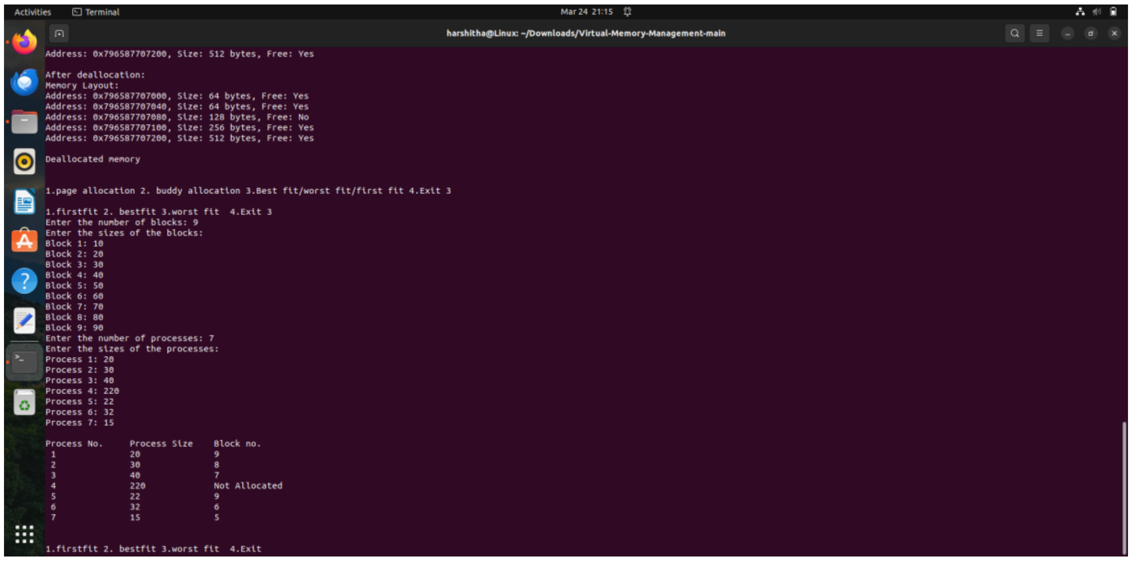
```
harshitha@Linux:~/Downloads/Virtual-Memory-Management-main$ make
make: 'vmm' is up to date.
harshitha@Linux:~/Downloads/Virtual-Memory-Management-main$ ./vmm

1.Memory allocation 2.Paging with TLB 3.Demand paging 4.page replacement 5.Exit 1

1.page allocation 2. buddy allocation 3.Best fit/worst fit/first fit 4.Exit 1
Page allocated at virtual address: 0x70fc4573a000
Page allocated at virtual address: 0x70fc45700000
Page allocated at virtual address: 0x70fc456ff000
Page allocated at virtual address: 0x70fc456fe000
Value pointed to by page: 10
Value pointed to by page: 20
Value pointed to by page: 30
Value pointed to by page: 40
Page deallocated at virtual address: 0x70fc4573a000
Value pointed to by page: 20
Value pointed to by page: 30
Value pointed to by page: 40


1.page allocation 2. buddy allocation 3.Best fit/worst fit/first fit 4.Exit 2
After allocation:
Memory Layout:
Address: 0x70fc4573a000, Size: 64 bytes, Free: No
Address: 0x70fc4573a040, Size: 32 bytes, Free: No
Address: 0x70fc4573a060, Size: 32 bytes, Free: Yes
Address: 0x70fc4573a080, Size: 128 bytes, Free: No
Address: 0x70fc4573a100, Size: 256 bytes, Free: Yes
Address: 0x70fc4573a200, Size: 512 bytes, Free: Yes

After deallocation:
Memory Layout:
Address: 0x70fc4573a000, Size: 64 bytes, Free: Yes
Address: 0x70fc4573a040, Size: 64 bytes, Free: Yes
Address: 0x70fc4573a080, Size: 128 bytes, Free: No
Address: 0x70fc4573a100, Size: 256 bytes, Free: Yes
Address: 0x70fc4573a200, Size: 512 bytes, Free: Yes

Deallocated memory


1.page allocation 2. buddy allocation 3.Best fit/worst fit/first fit 4.Exit
```

First-Fit and Best-Fit



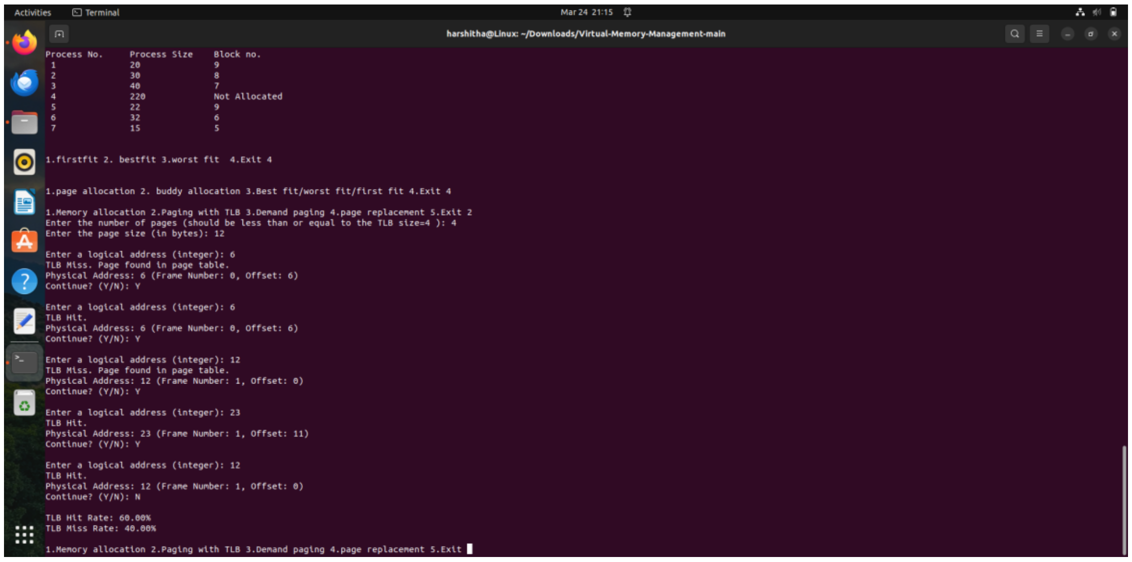
```
1.firstfit 2. bestfit 3.worst fit  4.Exit 1
Enter the number of blocks: 3
Enter the sizes of the blocks:
Block 1: 10
Block 2: 20
Block 3: 30
Enter the number of processes: 3
Enter the sizes of the processes:
Process 1: 12
Process 2: 15
Process 3: 18

Process No.     Process Size    Block no.
 1              12              2
 2              15              3
 3              18              Not Allocated


1.firstfit 2. bestfit 3.worst fit  4.Exit 2
Enter the number of blocks: 3
Enter the sizes of the blocks:
Block 1: 20
Block 2: 30
Block 3: 60
Enter the number of processes: 10
Enter the sizes of the processes:
Process 1: 1
Process 2: 2
Process 3: 3
Process 4: 4
Process 5: 5
Process 6: 6
Process 7: 43
Process 8: 25
Process 9: 20
Process 10: 20

Process No.     Process Size    Block no.
 1              1               1
 2              2               1
 3              3               1
 4              4               1
 5              5               1
 6              6               2
 7              43              3
 8              25              Not Allocated
 9              20              2
 10             20              Not Allocated
```

Worst-Fit



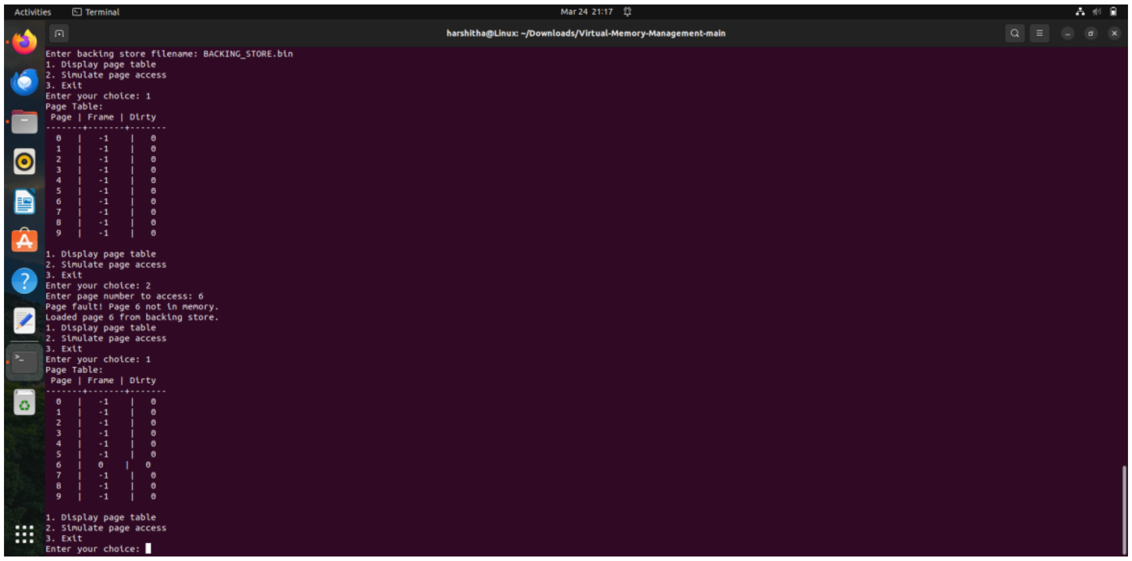
```
Address: 0x796587707200, Size: 512 bytes, Free: Yes

After deallocation:
Memory Layout:
Address: 0x796587707000, Size: 64 bytes, Free: Yes
Address: 0x796587707040, Size: 64 bytes, Free: Yes
Address: 0x796587707080, Size: 128 bytes, Free: No
Address: 0x796587707100, Size: 256 bytes, Free: Yes
Address: 0x796587707200, Size: 512 bytes, Free: Yes

Deallocated memory


1.page allocation 2. buddy allocation 3.Best fit/worst fit/first fit 4.Exit 3

1.firstfit 2. bestfit 3.worst fit  4.Exit 3
Enter the number of blocks: 9
Enter the sizes of the blocks:
Block 1: 10
Block 2: 20
Block 3: 30
Block 4: 40
Block 5: 50
Block 6: 60
Block 7: 70
Block 8: 80
Block 9: 90
Enter the number of processes: 7
Enter the sizes of the processes:
Process 1: 20
Process 2: 30
Process 3: 40
Process 4: 220
Process 5: 22
Process 6: 32
Process 7: 15

Process No.     Process Size    Block no.
 1              20              9
 2              30              8
 3              40              7
 4              220             Not Allocated
 5              22              9
 6              32              6
 7              15              5


1.firstfit 2. bestfit 3.worst fit  4.Exit
```

Paging with TLB

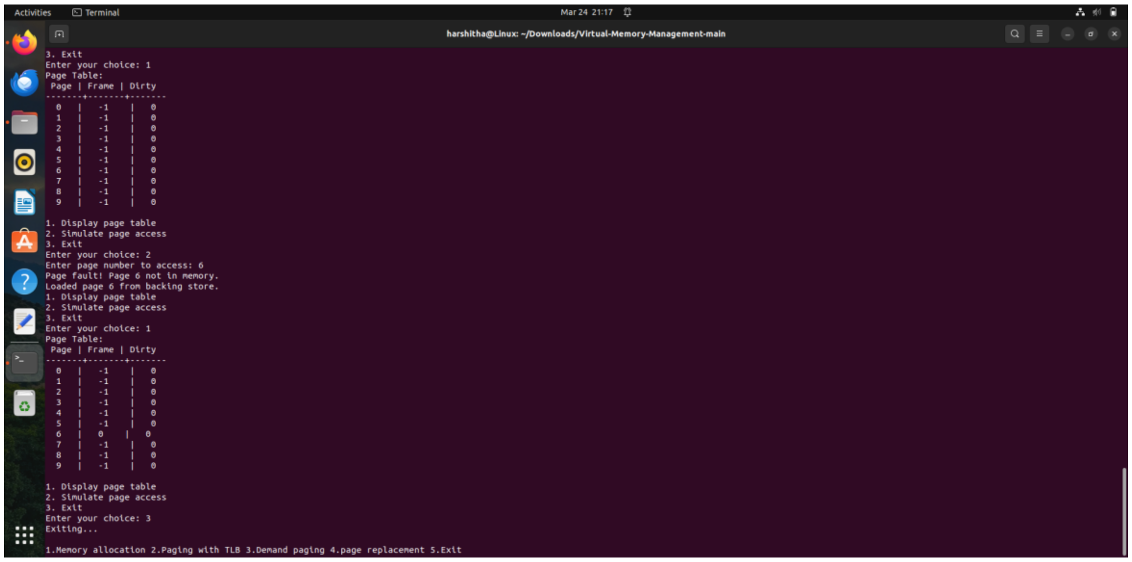


```
Process No.     Process Size    Block no.
 1              20              9
 2              30              8
 3              40              7
 4              220             Not Allocated
 5              22              9
 6              32              6
 7              15              5


1.firstfit 2. bestfit 3.worst fit  4.Exit 4


1.page allocation 2. buddy allocation 3.Best fit/worst fit/first fit 4.Exit 4

1.Memory allocation 2.Paging with TLB 3.Demand paging 4.page replacement 5.Exit 2
Enter the number of pages (should be less than or equal to the TLB size=4 ): 4
Enter the page size (in bytes): 12

Enter a logical address (integer): 6
TLB Miss. Page found in page table.
Physical Address: 6 (Frame Number: 0, Offset: 6)
Continue? (Y/N): Y

Enter a logical address (integer): 6
TLB Hit.
Physical Address: 6 (Frame Number: 0, Offset: 6)
Continue? (Y/N): Y

Enter a logical address (integer): 12
TLB Miss. Page found in page table.
Physical Address: 12 (Frame Number: 1, Offset: 0)
Continue? (Y/N): Y

Enter a logical address (integer): 23
TLB Hit.
Physical Address: 23 (Frame Number: 1, Offset: 11)
Continue? (Y/N): Y

Enter a logical address (integer): 12
TLB Hit.
Physical Address: 12 (Frame Number: 1, Offset: 0)
Continue? (Y/N): N

TLB Hit Rate: 60.00%
TLB Miss Rate: 40.00%

1.Memory allocation 2.Paging with TLB 3.Demand paging 4.page replacement 5.Exit
```

Demand Paging

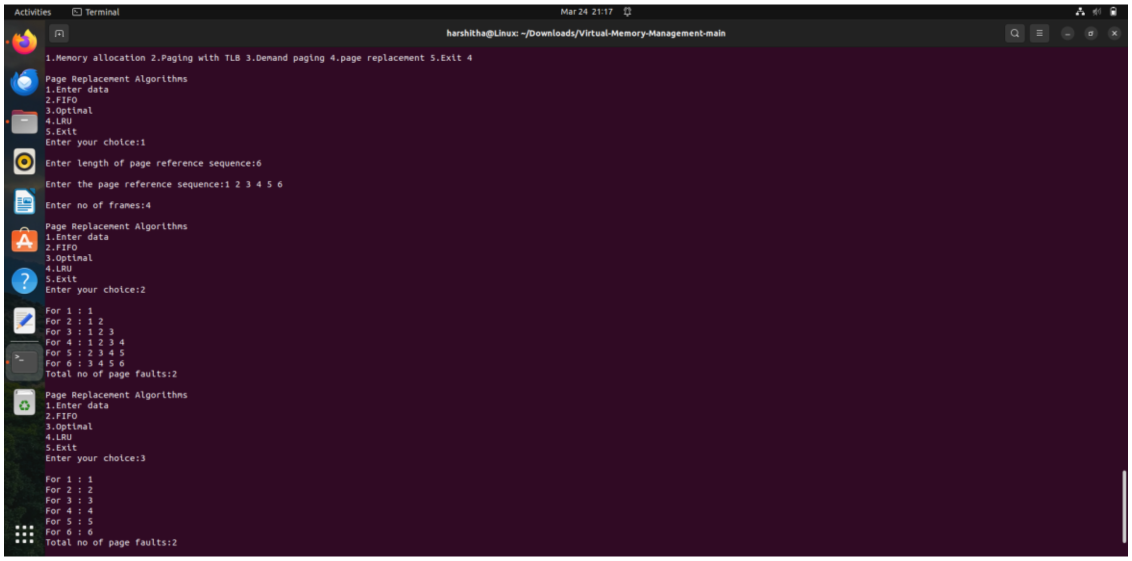


```
Enter backing store filename: BACKING_STORE.bin
1. Display page table
2. Simulate page access
3. Exit
Enter your choice: 1
Page Table:
 Page | Frame | Dirty
-------+-------+-------
  0   |   -1    |   0
  1   |   -1    |   0
  2   |   -1    |   0
  3   |   -1    |   0
  4   |   -1    |   0
  5   |   -1    |   0
  6   |   -1    |   0
  7   |   -1    |   0
  8   |   -1    |   0
  9   |   -1    |   0

1. Display page table
2. Simulate page access
3. Exit
Enter your choice: 2
Enter page number to access: 6
Page fault! Page 6 not in memory.
Loaded page 6 from backing store.
1. Display page table
2. Simulate page access
3. Exit
Enter your choice: 1
Page Table:
 Page | Frame | Dirty
-------+-------+-------
  0   |   -1    |   0
  1   |   -1    |   0
  2   |   -1    |   0
  3   |   -1    |   0
  4   |   -1    |   0
  5   |   -1    |   0
  6   |   0    |   0
  7   |   -1    |   0
  8   |   -1    |   0
  9   |   -1    |   0

1. Display page table
2. Simulate page access
3. Exit
Enter your choice: 
```



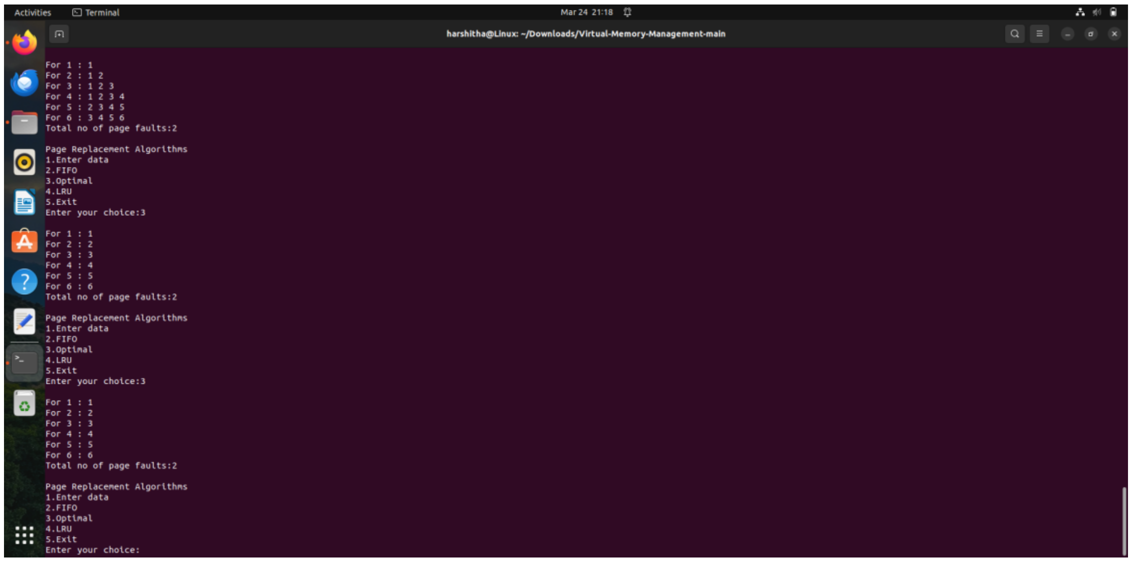
```
3. Exit
Enter your choice: 1
Page Table:
 Page | Frame | Dirty
-------+-------+-------
  0   |   -1    |   0
  1   |   -1    |   0
  2   |   -1    |   0
  3   |   -1    |   0
  4   |   -1    |   0
  5   |   -1    |   0
  6   |   -1    |   0
  7   |   -1    |   0
  8   |   -1    |   0
  9   |   -1    |   0

1. Display page table
2. Simulate page access
3. Exit
Enter your choice: 2
Enter page number to access: 6
Page fault! Page 6 not in memory.
Loaded page 6 from backing store.
1. Display page table
2. Simulate page access
3. Exit
Enter your choice: 1
Page Table:
 Page | Frame | Dirty
-------+-------+-------
  0   |   -1    |   0
  1   |   -1    |   0
  2   |   -1    |   0
  3   |   -1    |   0
  4   |   -1    |   0
  5   |   -1    |   0
  6   |   0    |   0
  7   |   -1    |   0
  8   |   -1    |   0
  9   |   -1    |   0

1. Display page table
2. Simulate page access
3. Exit
Enter your choice: 3
Exiting...

1.Memory allocation 2.Paging with TLB 3.Demand paging 4.page replacement 5.Exit
```

Page Replacement – FIFO



```
1.Memory allocation 2.Paging with TLB 3.Demand paging 4.page replacement 5.Exit 4

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:1

Enter length of page reference sequence:6

Enter the page reference sequence:1 2 3 4 5 6

Enter no of frames:4

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:2

For 1 : 1
For 2 : 1 2
For 3 : 1 2 3
For 4 : 1 2 3 4
For 5 : 2 3 4 5
For 6 : 3 4 5 6
Total no of page faults:2

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:3

For 1 : 1
For 2 : 2
For 3 : 3
For 4 : 4
For 5 : 5
For 6 : 6
Total no of page faults:2
```

Page Replacement - Optimal Paging and LRU



```
For 1 : 1
For 2 : 1 2
For 3 : 1 2 3
For 4 : 1 2 3 4
For 5 : 2 3 4 5
For 6 : 3 4 5 6
Total no of page faults:2

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:3

For 1 : 1
For 2 : 2
For 3 : 3
For 4 : 4
For 5 : 5
For 6 : 6
Total no of page faults:2

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:3

For 1 : 1
For 2 : 2
For 3 : 3
For 4 : 4
For 5 : 5
For 6 : 6
Total no of page faults:2

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.Exit
Enter your choice:
```

Conclusion:

In conclusion, our virtual memory management system has been successfully implemented, incorporating various memory allocation and deallocation strategies such as the Buddy algorithm, best-fit, worst-fit, and first-fit. These techniques enable efficient utilization of memory resources by dynamically assigning and reclaiming memory blocks based on specific criteria, thereby optimizing overall system performance. Additionally, we have integrated page replacement algorithms including FIFO (First In First Out), LRU (Least Recently Used), and Optimal Paging to enhance memory management further. These algorithms ensure optimal use of available memory by determining which pages to replace when new ones need to be loaded, thus minimizing unnecessary overhead and maximizing system efficiency. Moreover, our implementation includes functionality for demand paging and translation look-aside buffer (TLB), providing efficient handling of page faults and facilitating rapid address translation. Furthermore, our system offers insightful metrics such as hit and miss ratios, enabling comprehensive performance analysis and optimization. Looking ahead, future work could explore enhancements to these algorithms for even greater efficiency and scalability, as well as integration with advanced memory management techniques to meet the evolving demands of modern computing environments.

References

1. Christoph Scholl, Martin Böhnert, "A Dynamic Virtual Memory Management under Real-Time Constraints"]
2. How does OS manage memory and virtual memory? by Hridyesh singh bisht(Blog)
3. Operating Systems Concepts Book - 10th edition - Abraham Silberschatz, Peter Baer Galvin, Greg Gagne
4. Virtual Memory and its Concepts-Ms. Parveen Kaur*, in Journal of Advances and Scholarly Researches in Allied Education | Multidisciplinary Academic Research