Name: Mayesha Bintha Mizan
ID-811281302

The goal of the project is to implement a virtual memory simulator in order to understand the behavior of a page table and the page replacement algorithms.

Part 1: Page Table

In modern operating systems, **virtual memory** is a fundamental feature that enables processes to execute efficiently, even when their memory requirements exceed the available physical memory. Virtual memory abstracts the physical memory, allowing each process to operate within its own virtual address space. Central to this mechanism is the **page table**, a critical data structure that maintains the mapping between a process's virtual addresses and the physical addresses in main memory. By managing these mappings, the page table ensures that only the necessary portions of a process are loaded into memory, optimizing memory usage and enabling multiple processes to run concurrently without interfering with each other's memory spaces.

The virtual memory simulator employs two essential data structures to effectively manage memory mappings: the **Page Table Entry (PTE)** and the **Inverted Page Table Entry (IPTE)**. Each **PTE** encapsulates critical information about a virtual page, including the **Physical Frame Number (PFN)**, which indicates the specific physical memory frame where the page resides. Additionally, each PTE contains a **Valid Bit** that signifies whether the page is currently loaded in physical memory and a **Dirty Bit** that denotes if the page has been modified since its last load. Complementing the PTE, the **IPTE** associates each physical frame with its corresponding process through the **Process ID (pid)** and the **Virtual Page Number (VPN)**, facilitating efficient reverse lookups. This dual structure ensures accurate and efficient mapping between virtual addresses and physical memory, enabling seamless memory management across multiple processes.

Initially, all physical frames are marked as free, indicating their availability for allocation. The **page allocation policy** implemented is straightforward: assign the first available free frame to a requesting virtual page. This sequential allocation continues until the free-frame list is exhausted. Once no free frames remain, the simulator invokes a **page replacement policy** to select a victim frame for replacement. The chosen page is then **swapped out** if it's marked as dirty, ensuring that any modifications are preserved. Subsequently, the new page is **loaded** into the freed frame from the swap disk. This policy ensures that memory is utilized efficiently while adhering to the constraints of limited physical resources, maintaining system stability and performance even under high memory demand.

Memory Management Unit (MMU)

The **Memory Management Unit (MMU)** serves as the cornerstone of the virtual memory simulator, responsible for translating virtual addresses generated by processes into corresponding physical addresses in memory. This translation is achieved by dividing each virtual address into two distinct components: the **Virtual Page Number (VPN)** and the **Offset**. The VPN identifies the specific page within the process's address space, while the offset pinpoints the exact byte within that page. Specifically, the simulator isolates the VPN by shifting the virtual address right by eight bits and extracts the offset by applying a bitwise AND operation with 0xFF. This segmentation allows the MMU to efficiently map virtual addresses to physical frames using the information stored in the **Page Table**, facilitating swift and accurate memory access.

Get Physical Frame Number (get_PFN)

The get_PFN function is responsible for verifying the presence and validity of a requested virtual page in the page table. It receives the Process ID (pid), the Virtual Page Number (VPN), and the Request Type (reqType) as inputs. The function accesses the page table entry corresponding to the given VPN and checks the Valid Bit to determine if the page is currently loaded in physical memory—a condition known as a page hit. If the page is valid and present in memory, the function handles write requests by setting the Dirty Bit, indicating that the page has been modified. Additionally, if the CLOCK-Pro replacement policy is active, the function sets the Reference Bit for the corresponding PFN to mark recent usage. Upon a successful page hit, the function returns the PFN, facilitating the translation to the physical address. Conversely, if the page is not valid or absent—a page miss—the function returns -1, signaling the need for further handling by the pagefault_handler.

Page Fault Handler (pagefault_handler)

The **pagefault_handler function** efficiently manages the fault and ensure the seamless loading of the requested page into physical memory. The function initiates by attempting to locate a free physical frame using the provided get_freeframe() method. If a free frame is available, it is directly assigned to the requesting virtual page, and the page table is promptly updated to reflect this new mapping, marking the page as valid and setting the dirty bit based on the type of request (read or write).

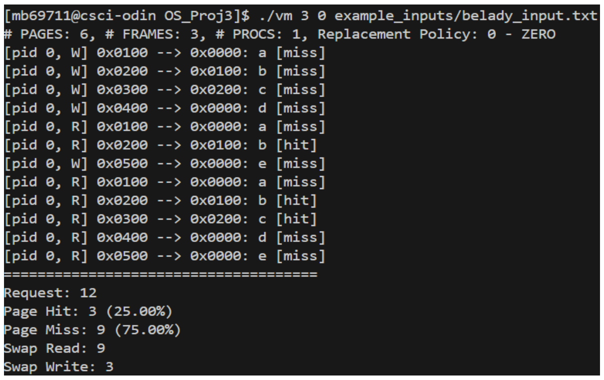
However, if no free frames are available, the function resorts to the active page replacement policy by calling **find_replacement()** to select a victim frame for eviction. Once a victim frame is identified, the corresponding Inverted Page Table Entry (IPTE) is retrieved to determine the owning process and the virtual page number of the page currently occupying that frame. If the victim page is marked as dirty indicating that it has been modified since being loaded into memory the swap_out() method is invoked to write the page back to the swap disk, thereby preserving data integrity. Subsequently, the requested page is loaded into the now-vacant frame using the swap_in() function. The page table and IPTE are updated to establish the new mapping between the Process ID (pid), Virtual Page Number (VPN), and Physical Frame Number (PFN).

Initial Page Replacement Policy: ZERO Replacement

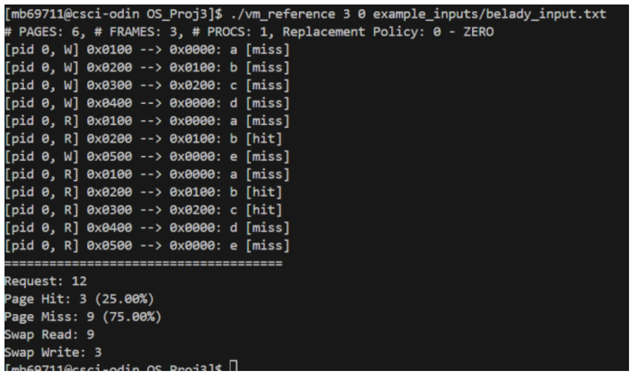
The simulator implements the **ZERO replacement algorithm**, a simplistic policy that always selects frame 0 for replacement when physical memory is full.

The ZERO replacement policy is straightforward: whenever a page needs to be replaced due to a page fault and no free frames are available, frame 0 is chosen as the victim. This approach simplifies initial testing and validation of the page table and fault handling mechanisms, ensuring that the core functionality operates correctly before introducing additional complexity with more advanced algorithms.

I run this program using belady_input, example.txt and also match the output with vm_reference.

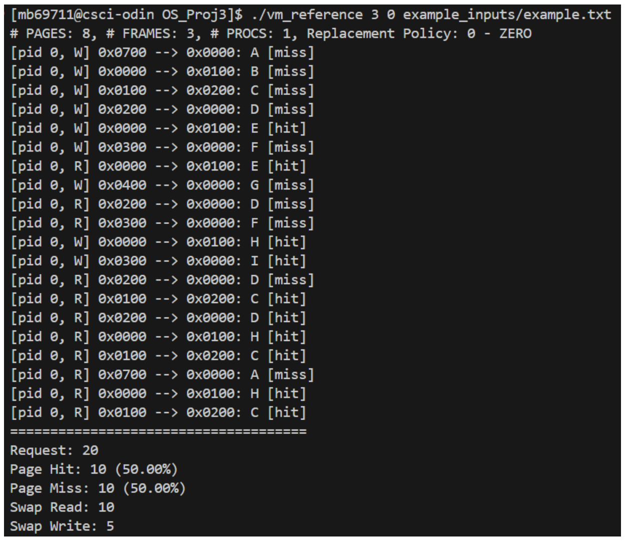


```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 0 example_inputs/belady_input.txt
# PAGES: 6, # FRAMES: 3, # PROCS: 1, Replacement Policy: 0 - ZERO
[pid 0, W] 0x0100 --> 0x0000: a [miss]
[pid 0, W] 0x0200 --> 0x0100: b [miss]
[pid 0, W] 0x0300 --> 0x0200: c [miss]
[pid 0, W] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0100 --> 0x0000: a [miss]
[pid 0, R] 0x0200 --> 0x0100: b [hit]
[pid 0, W] 0x0500 --> 0x0000: e [miss]
[pid 0, R] 0x0100 --> 0x0000: a [miss]
[pid 0, R] 0x0200 --> 0x0100: b [hit]
[pid 0, R] 0x0300 --> 0x0200: c [hit]
[pid 0, R] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0500 --> 0x0000: e [miss]
====================================
Request: 12
Page Hit: 3 (25.00%)
Page Miss: 9 (75.00%)
Swap Read: 9
Swap Write: 3
```

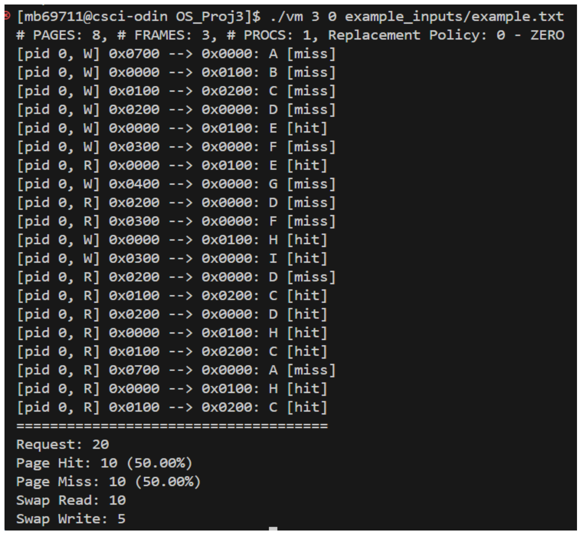


```
[mb69711@csci-odin OS_Proj3]$ ./vm_reference 3 0 example_inputs/belady_input.txt
# PAGES: 6, # FRAMES: 3, # PROCS: 1, Replacement Policy: 0 - ZERO
[pid 0, W] 0x0100 --> 0x0000: a [miss]
[pid 0, W] 0x0200 --> 0x0100: b [miss]
[pid 0, W] 0x0300 --> 0x0200: c [miss]
[pid 0, W] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0100 --> 0x0000: a [miss]
[pid 0, R] 0x0200 --> 0x0100: b [hit]
[pid 0, W] 0x0500 --> 0x0000: e [miss]
[pid 0, R] 0x0100 --> 0x0000: a [miss]
[pid 0, R] 0x0200 --> 0x0100: b [hit]
[pid 0, R] 0x0300 --> 0x0200: c [hit]
[pid 0, R] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0500 --> 0x0000: e [miss]
====================================
Request: 12
Page Hit: 3 (25.00%)
Page Miss: 9 (75.00%)
Swap Read: 9
Swap Write: 3
[mb69711@csci-odin OS_Proj3]$
```

Name: Mayesha Bintha Mizan
ID-811281302



```
[mb69711@csci-odin OS_Proj3]$ ./vm_reference 3 0 example_inputs/example.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 1, Replacement Policy: 0 - ZERO
[pid 0, W] 0x0700 --> 0x0000: A [miss]
[pid 0, W] 0x0000 --> 0x0100: B [miss]
[pid 0, W] 0x0100 --> 0x0200: C [miss]
[pid 0, W] 0x0200 --> 0x0000: D [miss]
[pid 0, W] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0300 --> 0x0000: F [miss]
[pid 0, R] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0400 --> 0x0000: G [miss]
[pid 0, R] 0x0200 --> 0x0000: D [miss]
[pid 0, R] 0x0300 --> 0x0000: F [miss]
[pid 0, W] 0x0000 --> 0x0100: H [hit]
[pid 0, W] 0x0300 --> 0x0000: I [hit]
[pid 0, R] 0x0200 --> 0x0000: D [miss]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
[pid 0, R] 0x0200 --> 0x0000: D [hit]
[pid 0, R] 0x0000 --> 0x0100: H [hit]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
[pid 0, R] 0x0700 --> 0x0000: A [miss]
[pid 0, R] 0x0000 --> 0x0100: H [hit]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
===================================
Request: 20
Page Hit: 10 (50.00%)
Page Miss: 10 (50.00%)
Swap Read: 10
Swap Write: 5
```



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 0 example_inputs/example.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 1, Replacement Policy: 0 - ZERO
[pid 0, W] 0x0700 --> 0x0000: A [miss]
[pid 0, W] 0x0000 --> 0x0100: B [miss]
[pid 0, W] 0x0100 --> 0x0200: C [miss]
[pid 0, W] 0x0200 --> 0x0000: D [miss]
[pid 0, W] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0300 --> 0x0000: F [miss]
[pid 0, R] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0400 --> 0x0000: G [miss]
[pid 0, R] 0x0200 --> 0x0000: D [miss]
[pid 0, R] 0x0300 --> 0x0000: F [miss]
[pid 0, W] 0x0000 --> 0x0100: H [hit]
[pid 0, W] 0x0300 --> 0x0000: I [hit]
[pid 0, R] 0x0200 --> 0x0000: D [miss]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
[pid 0, R] 0x0200 --> 0x0000: D [hit]
[pid 0, R] 0x0000 --> 0x0100: H [hit]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
[pid 0, R] 0x0700 --> 0x0000: A [miss]
[pid 0, R] 0x0000 --> 0x0100: H [hit]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
===================================
Request: 20
Page Hit: 10 (50.00%)
Page Miss: 10 (50.00%)
Swap Read: 10
Swap Write: 5
```

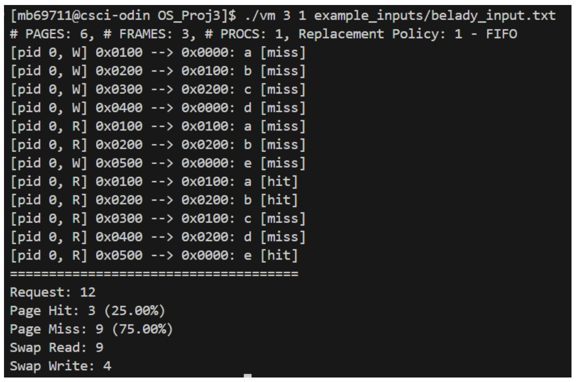
Part 2: Page Replacement Algorithm

In virtual memory systems, efficient management of physical memory is paramount to ensure optimal system performance. When physical memory frames are exhausted, the operating system must decide which existing pages to evict to accommodate new page requests. Page replacement algorithms are crucial in making these decisions, balancing factors such as access patterns, page usage frequency, and system overhead. This section explores the implementation of four-page replacement strategies within the virtual memory simulator: First-In-First-Out (FIFO), Least Recently Used (LRU), CLOCK, and the advanced CLOCK-Pro.

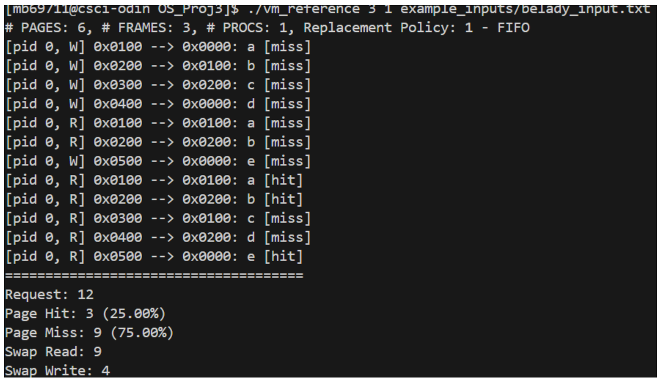
First-In-First-Out (FIFO)

The **First-In-First-Out (FIFO)** page replacement algorithm employs a straightforward mechanism to manage memory frames. It utilizes a counter, fifo_position, which keeps track of the next frame slated for replacement in a cyclical fashion. Whenever a page needs to be evicted from memory, the algorithm selects the frame currently pointed to by fifo_position. After selecting the frame, fifo_position is incremented by one and then taken modulo the total number of frames (MAX_PFN) to ensure it wraps around seamlessly. This approach ensures that frames are replaced in the exact order they were loaded into memory, maintaining a simple and predictable replacement pattern.

I tested FIFO with belady_input, example.txt, my generated input and also match the output with vm_reference.

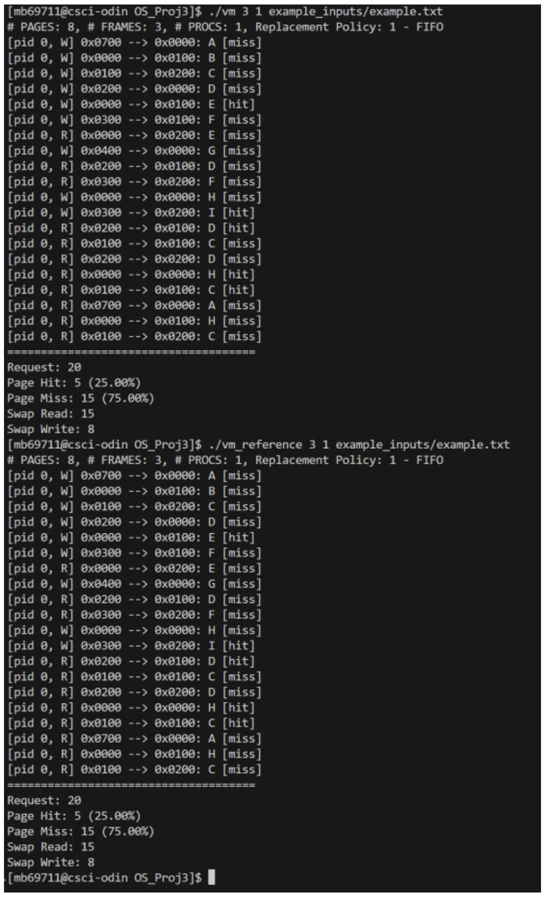


```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 1 example_inputs/belady_input.txt
# PAGES: 6, # FRAMES: 3, # PROCS: 1, Replacement Policy: 1 - FIFO
[pid 0, W] 0x0100 --> 0x0000: a [miss]
[pid 0, W] 0x0200 --> 0x0100: b [miss]
[pid 0, W] 0x0300 --> 0x0200: c [miss]
[pid 0, W] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0100 --> 0x0100: a [miss]
[pid 0, R] 0x0200 --> 0x0200: b [miss]
[pid 0, W] 0x0500 --> 0x0000: e [miss]
[pid 0, R] 0x0100 --> 0x0100: a [hit]
[pid 0, R] 0x0200 --> 0x0200: b [hit]
[pid 0, R] 0x0300 --> 0x0100: c [miss]
[pid 0, R] 0x0400 --> 0x0200: d [miss]
[pid 0, R] 0x0500 --> 0x0000: e [hit]
===================================
Request: 12
Page Hit: 3 (25.00%)
Page Miss: 9 (75.00%)
Swap Read: 9
Swap Write: 4
```

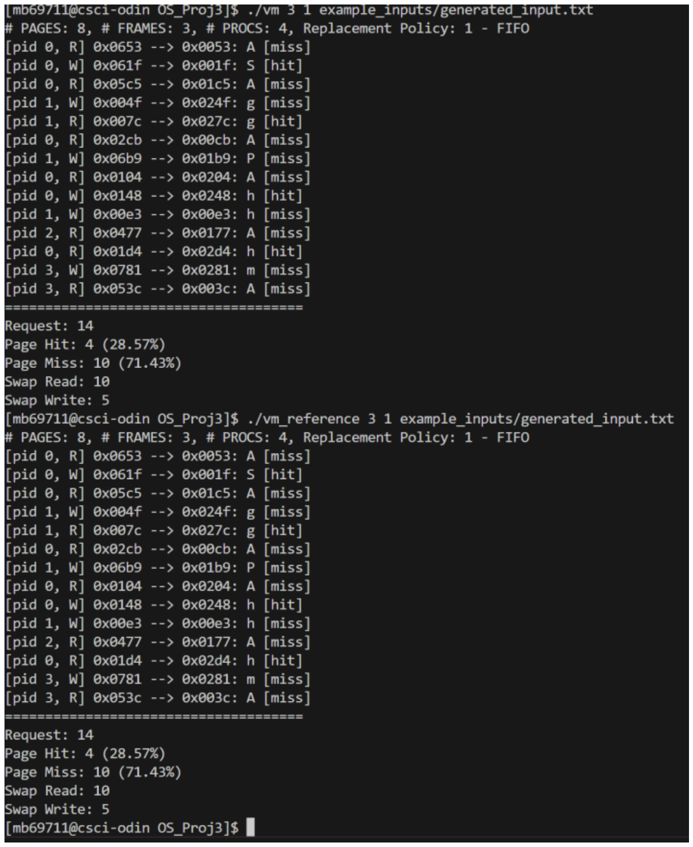


```
[mb69711@csci-odin OS_Proj3]$ ./vm_reference 3 1 example_inputs/belady_input.txt
# PAGES: 6, # FRAMES: 3, # PROCS: 1, Replacement Policy: 1 - FIFO
[pid 0, W] 0x0100 --> 0x0000: a [miss]
[pid 0, W] 0x0200 --> 0x0100: b [miss]
[pid 0, W] 0x0300 --> 0x0200: c [miss]
[pid 0, W] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0100 --> 0x0100: a [miss]
[pid 0, R] 0x0200 --> 0x0200: b [miss]
[pid 0, W] 0x0500 --> 0x0000: e [miss]
[pid 0, R] 0x0100 --> 0x0100: a [hit]
[pid 0, R] 0x0200 --> 0x0200: b [hit]
[pid 0, R] 0x0300 --> 0x0100: c [miss]
[pid 0, R] 0x0400 --> 0x0200: d [miss]
[pid 0, R] 0x0500 --> 0x0000: e [hit]
===================================
Request: 12
Page Hit: 3 (25.00%)
Page Miss: 9 (75.00%)
Swap Read: 9
Swap Write: 4
```

Name: Mayesha Bintha Mizan
ID-811281302



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 1 example_inputs/example.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 1, Replacement Policy: 1 - FIFO
[pid 0, W] 0x0700 --> 0x0000: A [miss]
[pid 0, W] 0x0000 --> 0x0100: B [miss]
[pid 0, W] 0x0100 --> 0x0200: C [miss]
[pid 0, W] 0x0200 --> 0x0000: D [miss]
[pid 0, W] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0300 --> 0x0100: F [miss]
[pid 0, R] 0x0000 --> 0x0200: E [miss]
[pid 0, W] 0x0400 --> 0x0000: G [miss]
[pid 0, R] 0x0200 --> 0x0100: D [miss]
[pid 0, R] 0x0300 --> 0x0200: F [miss]
[pid 0, W] 0x0000 --> 0x0000: H [miss]
[pid 0, W] 0x0300 --> 0x0200: I [hit]
[pid 0, R] 0x0200 --> 0x0100: D [hit]
[pid 0, R] 0x0100 --> 0x0100: C [miss]
[pid 0, R] 0x0200 --> 0x0200: D [miss]
[pid 0, R] 0x0000 --> 0x0000: H [hit]
[pid 0, R] 0x0100 --> 0x0100: C [hit]
[pid 0, R] 0x0700 --> 0x0000: A [miss]
[pid 0, R] 0x0000 --> 0x0100: H [miss]
[pid 0, R] 0x0100 --> 0x0200: C [miss]
=====================================
Request: 20
Page Hit: 5 (25.00%)
Page Miss: 15 (75.00%)
Swap Read: 15
Swap Write: 8
[mb69711@csci-odin OS_Proj3]$ ./vm_reference 3 1 example_inputs/example.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 1, Replacement Policy: 1 - FIFO
[pid 0, W] 0x0700 --> 0x0000: A [miss]
[pid 0, W] 0x0000 --> 0x0100: B [miss]
[pid 0, W] 0x0100 --> 0x0200: C [miss]
[pid 0, W] 0x0200 --> 0x0000: D [miss]
[pid 0, W] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0300 --> 0x0100: F [miss]
[pid 0, R] 0x0000 --> 0x0200: E [miss]
[pid 0, W] 0x0400 --> 0x0000: G [miss]
[pid 0, R] 0x0200 --> 0x0100: D [miss]
[pid 0, R] 0x0300 --> 0x0200: F [miss]
[pid 0, W] 0x0000 --> 0x0000: H [miss]
[pid 0, W] 0x0300 --> 0x0200: I [hit]
[pid 0, R] 0x0200 --> 0x0100: D [hit]
[pid 0, R] 0x0100 --> 0x0100: C [miss]
[pid 0, R] 0x0200 --> 0x0200: D [miss]
[pid 0, R] 0x0000 --> 0x0000: H [hit]
[pid 0, R] 0x0100 --> 0x0100: C [hit]
[pid 0, R] 0x0700 --> 0x0000: A [miss]
[pid 0, R] 0x0000 --> 0x0100: H [miss]
[pid 0, R] 0x0100 --> 0x0200: C [miss]
=====================================
Request: 20
Page Hit: 5 (25.00%)
Page Miss: 15 (75.00%)
Swap Read: 15
Swap Write: 8
[mb69711@csci-odin OS_Proj3]$
```



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 1 example_inputs/generated_input.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 4, Replacement Policy: 1 - FIFO
[pid 0, R] 0x0653 --> 0x0053: A [miss]
[pid 0, W] 0x061f --> 0x001f: S [hit]
[pid 0, R] 0x05c5 --> 0x01c5: A [miss]
[pid 1, W] 0x004f --> 0x024f: g [miss]
[pid 1, R] 0x007c --> 0x027c: g [hit]
[pid 0, R] 0x02cb --> 0x00cb: A [miss]
[pid 1, W] 0x06b9 --> 0x01b9: P [miss]
[pid 0, R] 0x0104 --> 0x0204: A [miss]
[pid 0, W] 0x0148 --> 0x0248: h [hit]
[pid 1, W] 0x00e3 --> 0x00e3: h [miss]
[pid 2, R] 0x0477 --> 0x0177: A [miss]
[pid 0, R] 0x01d4 --> 0x02d4: h [hit]
[pid 3, W] 0x0781 --> 0x0281: m [miss]
[pid 3, R] 0x053c --> 0x003c: A [miss]
=====================================
Request: 14
Page Hit: 4 (28.57%)
Page Miss: 10 (71.43%)
Swap Read: 10
Swap Write: 5
[mb69711@csci-odin OS_Proj3]$ ./vm_reference 3 1 example_inputs/generated_input.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 4, Replacement Policy: 1 - FIFO
[pid 0, R] 0x0653 --> 0x0053: A [miss]
[pid 0, W] 0x061f --> 0x001f: S [hit]
[pid 0, R] 0x05c5 --> 0x01c5: A [miss]
[pid 1, W] 0x004f --> 0x024f: g [miss]
[pid 1, R] 0x007c --> 0x027c: g [hit]
[pid 0, R] 0x02cb --> 0x00cb: A [miss]
[pid 1, W] 0x06b9 --> 0x01b9: P [miss]
[pid 0, R] 0x0104 --> 0x0204: A [miss]
[pid 0, W] 0x0148 --> 0x0248: h [hit]
[pid 1, W] 0x00e3 --> 0x00e3: h [miss]
[pid 2, R] 0x0477 --> 0x0177: A [miss]
[pid 0, R] 0x01d4 --> 0x02d4: h [hit]
[pid 3, W] 0x0781 --> 0x0281: m [miss]
[pid 3, R] 0x053c --> 0x003c: A [miss]
=====================================
Request: 14
Page Hit: 4 (28.57%)
Page Miss: 10 (71.43%)
Swap Read: 10
Swap Write: 5
[mb69711@csci-odin OS_Proj3]$
```

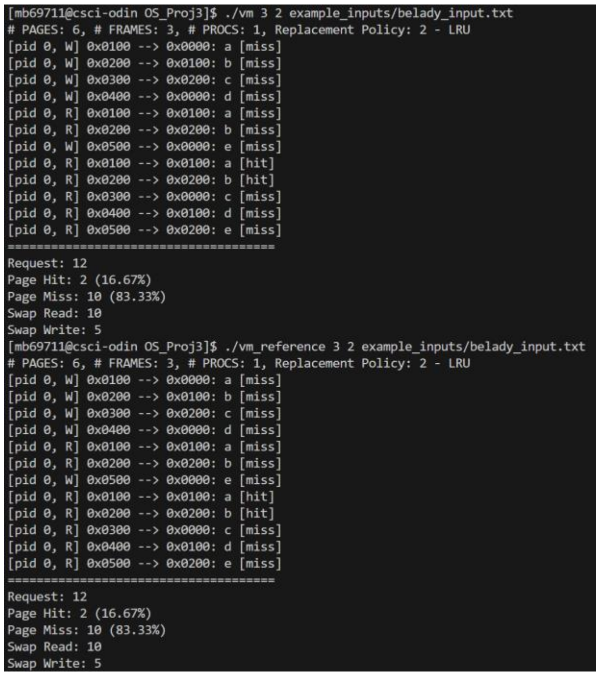
The table illustrates the **FIFO page replacement algorithm**'s performance as the number of physical frames increases from three to six using belady input.

Number of Frames	Page Hits	Hit Rate (%)	Page Misses	Miss Rate (%)	Swap Reads	Swap Writes
3	3	25.00%	9	75.00%	9	4
4	2	16.67%	10	83.33%	10	5
5	7	58.33%	5	41.67%	5	0
6	7	58.33%	5	41.67%	5	0

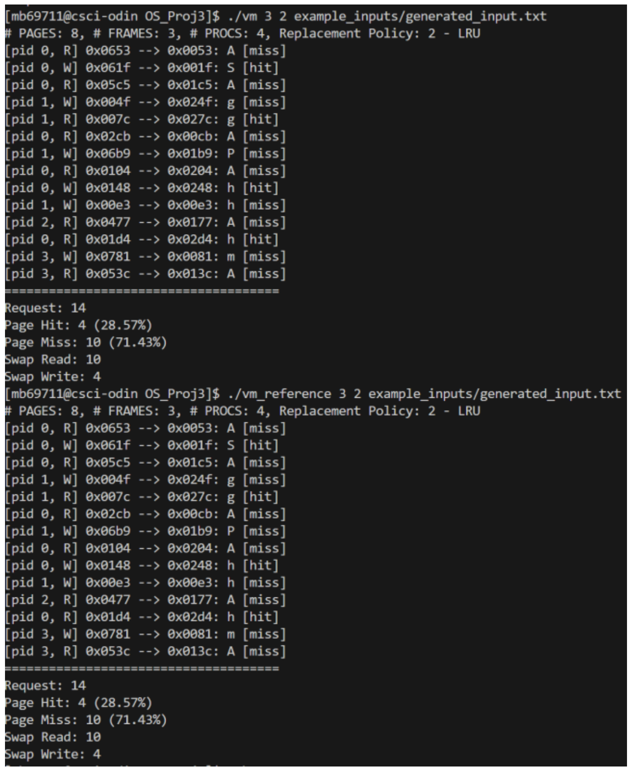
Least Recently Used (LRU) Page Replacement Algorithm

The **Least Recently Used (LRU)** page replacement algorithm in the simulator is meticulously implemented using a doubly linked list to accurately track the usage of each memory frame. The core function, lru(), identifies and removes the least recently used frame by targeting the tail of the list, thereby selecting it for eviction when a page fault occurs. When a new page is accessed or loaded into memory, the **add_to_lru(int PFN)** function is invoked to insert the corresponding frame at the head of the linked list, marking it as the most recently used. This dynamic insertion maintains the list's order, ensuring that the head always points to the most recently accessed frame. Additionally, whenever an existing page is accessed again, the **update_lru(int PFN)** function is called to move the accessed frame from its current position to the head of the list, effectively updating its status as the most recently used. This systematic updating ensures that frequently accessed pages remain near the front of the list, while less active pages naturally migrate towards the tail, making them prime candidates for replacement when necessary. Through this organized approach, the LRU algorithm optimizes memory usage by retaining pages that are actively in use and efficiently managing the eviction of those that are not, thereby enhancing the overall performance and responsiveness of the virtual memory system.

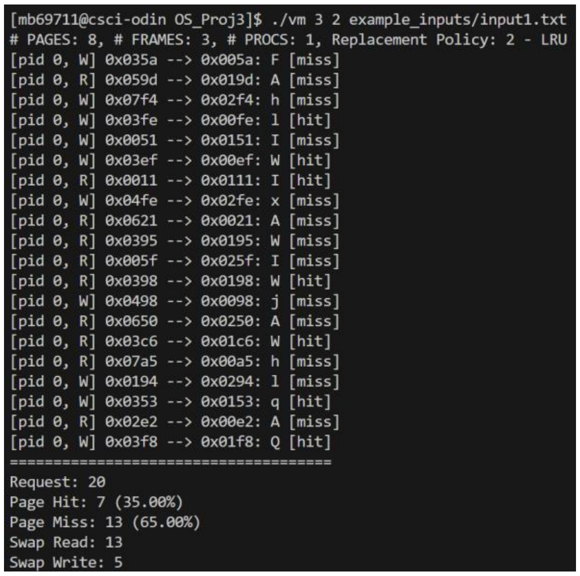
I tested LRU with belady_input, my generated input, input1 and also match the output with vm_reference.



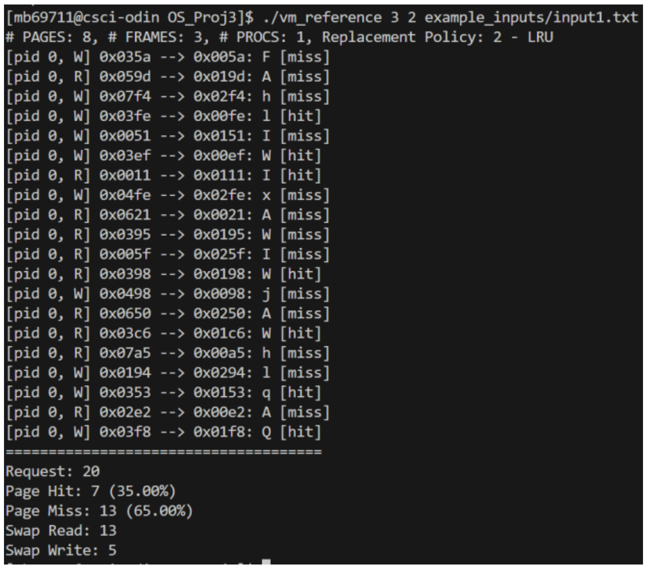
```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 2 example_inputs/belady_input.txt
# PAGES: 6, # FRAMES: 3, # PROCS: 1, Replacement Policy: 2 - LRU
[pid 0, W] 0x0100 --> 0x0000: a [miss]
[pid 0, W] 0x0200 --> 0x0100: b [miss]
[pid 0, W] 0x0300 --> 0x0200: c [miss]
[pid 0, W] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0100 --> 0x0100: a [miss]
[pid 0, R] 0x0200 --> 0x0200: b [miss]
[pid 0, W] 0x0500 --> 0x0000: e [miss]
[pid 0, R] 0x0100 --> 0x0100: a [hit]
[pid 0, R] 0x0200 --> 0x0200: b [hit]
[pid 0, R] 0x0300 --> 0x0000: c [miss]
[pid 0, R] 0x0400 --> 0x0100: d [miss]
[pid 0, R] 0x0500 --> 0x0200: e [miss]
=====================================
Request: 12
Page Hit: 2 (16.67%)
Page Miss: 10 (83.33%)
Swap Read: 10
Swap Write: 5
[mb69711@csci-odin OS_Proj3]$ ./vm_reference 3 2 example_inputs/belady_input.txt
# PAGES: 6, # FRAMES: 3, # PROCS: 1, Replacement Policy: 2 - LRU
[pid 0, W] 0x0100 --> 0x0000: a [miss]
[pid 0, W] 0x0200 --> 0x0100: b [miss]
[pid 0, W] 0x0300 --> 0x0200: c [miss]
[pid 0, W] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0100 --> 0x0100: a [miss]
[pid 0, R] 0x0200 --> 0x0200: b [miss]
[pid 0, W] 0x0500 --> 0x0000: e [miss]
[pid 0, R] 0x0100 --> 0x0100: a [hit]
[pid 0, R] 0x0200 --> 0x0200: b [hit]
[pid 0, R] 0x0300 --> 0x0000: c [miss]
[pid 0, R] 0x0400 --> 0x0100: d [miss]
[pid 0, R] 0x0500 --> 0x0200: e [miss]
=====================================
Request: 12
Page Hit: 2 (16.67%)
Page Miss: 10 (83.33%)
Swap Read: 10
Swap Write: 5
```



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 2 example_inputs/generated_input.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 4, Replacement Policy: 2 - LRU
[pid 0, R] 0x0653 --> 0x0053: A [miss]
[pid 0, W] 0x061f --> 0x001f: S [hit]
[pid 0, R] 0x05c5 --> 0x01c5: A [miss]
[pid 1, W] 0x004f --> 0x024f: g [miss]
[pid 1, R] 0x007c --> 0x027c: g [hit]
[pid 0, R] 0x02cb --> 0x00cb: A [miss]
[pid 1, W] 0x06b9 --> 0x01b9: P [miss]
[pid 0, R] 0x0104 --> 0x0204: A [miss]
[pid 0, W] 0x0148 --> 0x0248: h [hit]
[pid 1, W] 0x00e3 --> 0x00e3: h [miss]
[pid 2, R] 0x0477 --> 0x0177: A [miss]
[pid 0, R] 0x01d4 --> 0x02d4: h [hit]
[pid 3, W] 0x0781 --> 0x0081: m [miss]
[pid 3, R] 0x053c --> 0x013c: A [miss]
=====================================
Request: 14
Page Hit: 4 (28.57%)
Page Miss: 10 (71.43%)
Swap Read: 10
Swap Write: 4
[mb69711@csci-odin OS_Proj3]$ ./vm_reference 3 2 example_inputs/generated_input.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 4, Replacement Policy: 2 - LRU
[pid 0, R] 0x0653 --> 0x0053: A [miss]
[pid 0, W] 0x061f --> 0x001f: S [hit]
[pid 0, R] 0x05c5 --> 0x01c5: A [miss]
[pid 1, W] 0x004f --> 0x024f: g [miss]
[pid 1, R] 0x007c --> 0x027c: g [hit]
[pid 0, R] 0x02cb --> 0x00cb: A [miss]
[pid 1, W] 0x06b9 --> 0x01b9: P [miss]
[pid 0, R] 0x0104 --> 0x0204: A [miss]
[pid 0, W] 0x0148 --> 0x0248: h [hit]
[pid 1, W] 0x00e3 --> 0x00e3: h [miss]
[pid 2, R] 0x0477 --> 0x0177: A [miss]
[pid 0, R] 0x01d4 --> 0x02d4: h [hit]
[pid 3, W] 0x0781 --> 0x0081: m [miss]
[pid 3, R] 0x053c --> 0x013c: A [miss]
=====================================
Request: 14
Page Hit: 4 (28.57%)
Page Miss: 10 (71.43%)
Swap Read: 10
Swap Write: 4
```



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 2 example_inputs/input1.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 1, Replacement Policy: 2 - LRU
[pid 0, W] 0x035a --> 0x005a: F [miss]
[pid 0, R] 0x059d --> 0x019d: A [miss]
[pid 0, W] 0x07f4 --> 0x02f4: h [miss]
[pid 0, W] 0x03fe --> 0x00fe: l [hit]
[pid 0, W] 0x0051 --> 0x0151: I [miss]
[pid 0, W] 0x03ef --> 0x00ef: W [hit]
[pid 0, R] 0x0011 --> 0x0111: I [hit]
[pid 0, W] 0x04fe --> 0x02fe: x [miss]
[pid 0, R] 0x0621 --> 0x0021: A [miss]
[pid 0, R] 0x0395 --> 0x0195: W [miss]
[pid 0, R] 0x005f --> 0x025f: I [miss]
[pid 0, R] 0x0398 --> 0x0198: W [hit]
[pid 0, W] 0x0498 --> 0x0098: j [miss]
[pid 0, R] 0x0650 --> 0x0250: A [miss]
[pid 0, R] 0x03c6 --> 0x01c6: W [hit]
[pid 0, R] 0x07a5 --> 0x00a5: h [miss]
[pid 0, W] 0x0194 --> 0x0294: l [miss]
[pid 0, W] 0x0353 --> 0x0153: q [hit]
[pid 0, R] 0x02e2 --> 0x00e2: A [miss]
[pid 0, W] 0x03f8 --> 0x01f8: Q [hit]
=====================================
Request: 20
Page Hit: 7 (35.00%)
Page Miss: 13 (65.00%)
Swap Read: 13
Swap Write: 5
```



```
[mb69711@csci-odin OS_Proj3]$ ./vm_reference 3 2 example_inputs/input1.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 1, Replacement Policy: 2 - LRU
[pid 0, W] 0x035a --> 0x005a: F [miss]
[pid 0, R] 0x059d --> 0x019d: A [miss]
[pid 0, W] 0x07f4 --> 0x02f4: h [miss]
[pid 0, W] 0x03fe --> 0x00fe: l [hit]
[pid 0, W] 0x0051 --> 0x0151: I [miss]
[pid 0, W] 0x03ef --> 0x00ef: W [hit]
[pid 0, R] 0x0011 --> 0x0111: I [hit]
[pid 0, W] 0x04fe --> 0x02fe: x [miss]
[pid 0, R] 0x0621 --> 0x0021: A [miss]
[pid 0, R] 0x0395 --> 0x0195: W [miss]
[pid 0, R] 0x005f --> 0x025f: I [miss]
[pid 0, R] 0x0398 --> 0x0198: W [hit]
[pid 0, W] 0x0498 --> 0x0098: j [miss]
[pid 0, R] 0x0650 --> 0x0250: A [miss]
[pid 0, R] 0x03c6 --> 0x01c6: W [hit]
[pid 0, R] 0x07a5 --> 0x00a5: h [miss]
[pid 0, W] 0x0194 --> 0x0294: l [miss]
[pid 0, W] 0x0353 --> 0x0153: q [hit]
[pid 0, R] 0x02e2 --> 0x00e2: A [miss]
[pid 0, W] 0x03f8 --> 0x01f8: Q [hit]
=====================================
Request: 20
Page Hit: 7 (35.00%)
Page Miss: 13 (65.00%)
Swap Read: 13
Swap Write: 5
```

The table illustrates the **LRU page replacement algorithm**'s performance as the number of physical frames increases from three to six using belady input.

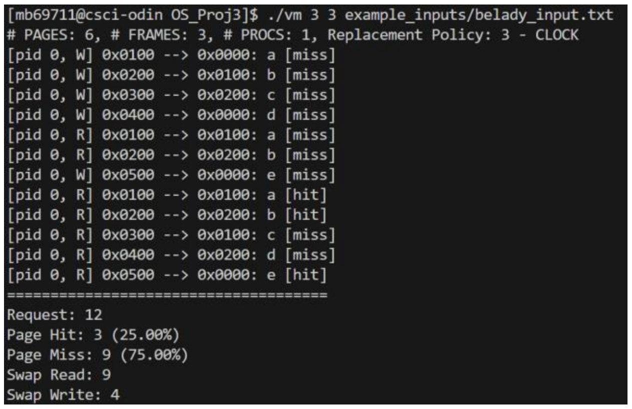
Name: Mayesha Bintha Mizan
ID-811281302

Number of Frames	Page Hits	Hit Rate (%)	Page Misses	Miss Rate(%)	Swap Reads	Swap Writes
3	2	16.67%	10	83.33%	10	5
4	4	33.33%	8	66.67%	8	4
5	7	58.33%	5	41.67%	5	0
6	7	58.33%	5	41.67%	5	0

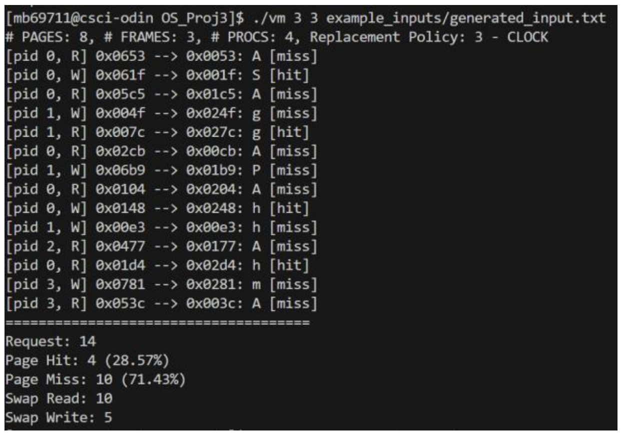
CLOCK Page Replacement Algorithm

The **CLOCK** page replacement algorithm in the simulator is implemented using **a circular buffer** and a reference bit array to efficiently manage memory frames. The core function, clock(), continuously scans the physical frames in a cyclic manner, starting from the current position of clockHand. For each frame, it checks the corresponding reference bit in the clockReferenceBits array. If the reference bit is 0, indicating that the page has not been recently accessed, the algorithm selects this frame for eviction by returning its **Physical Frame Number (PFN)**. Before moving to the next frame, clock() sets the reference bit of the selected frame to 1, preparing it for future reference. If the reference bit is 1, the algorithm resets it to 0 and advances the clockHand to the next frame, continuing the search for a suitable victim. This mechanism ensures that pages which have been recently accessed are given priority to remain in memory, while those that have not been accessed recently are more likely to be evicted. Additionally, the add_to_clock(int PFN) function is utilized to set the reference bit of a frame to 1 whenever a page is accessed or loaded into that frame, thereby marking it as recently used. This is an efficient way to give pages a "second chance" before eviction.

I tested Clock with belady_input, my generated input,example and input2 and I did the algorithm by hand and check that the result is correct.

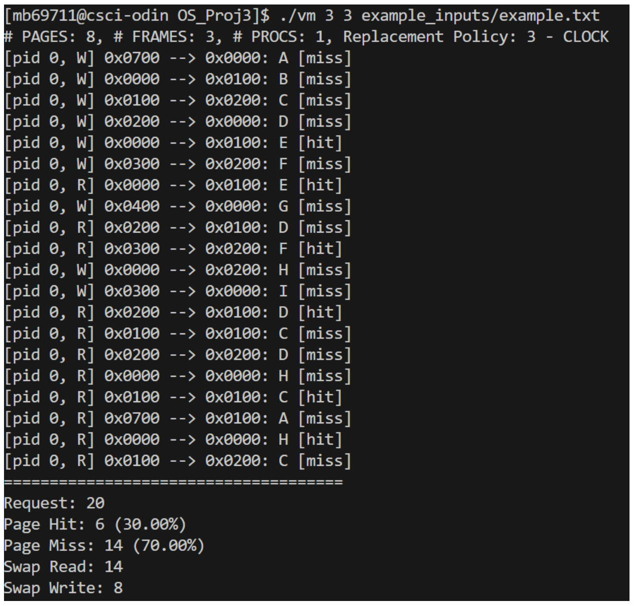


```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 3 example_inputs/belady_input.txt
# PAGES: 6, # FRAMES: 3, # PROCS: 1, Replacement Policy: 3 - CLOCK
[pid 0, W] 0x0100 --> 0x0000: a [miss]
[pid 0, W] 0x0200 --> 0x0100: b [miss]
[pid 0, W] 0x0300 --> 0x0200: c [miss]
[pid 0, W] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0100 --> 0x0100: a [miss]
[pid 0, R] 0x0200 --> 0x0200: b [miss]
[pid 0, W] 0x0500 --> 0x0000: e [miss]
[pid 0, R] 0x0100 --> 0x0100: a [hit]
[pid 0, R] 0x0200 --> 0x0200: b [hit]
[pid 0, R] 0x0300 --> 0x0100: c [miss]
[pid 0, R] 0x0400 --> 0x0200: d [miss]
[pid 0, R] 0x0500 --> 0x0000: e [hit]
=====================================
Request: 12
Page Hit: 3 (25.00%)
Page Miss: 9 (75.00%)
Swap Read: 9
Swap Write: 4
```

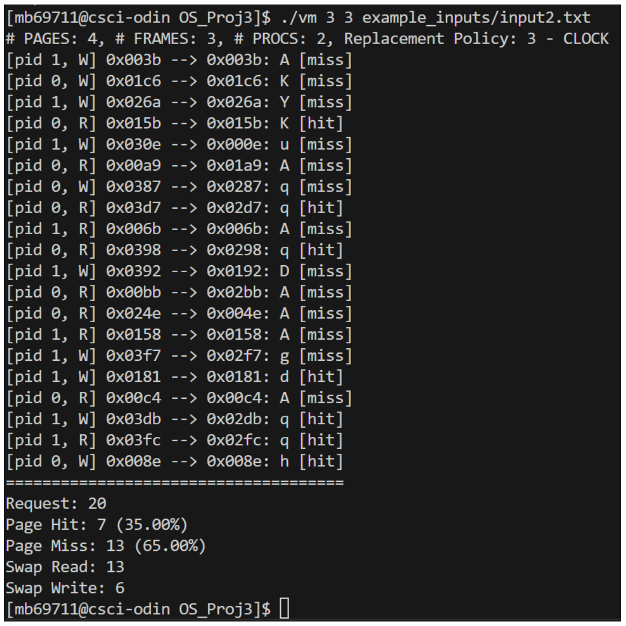


```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 3 example_inputs/generated_input.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 4, Replacement Policy: 3 - CLOCK
[pid 0, R] 0x0653 --> 0x0053: A [miss]
[pid 0, W] 0x061f --> 0x001f: S [hit]
[pid 0, R] 0x05c5 --> 0x01c5: A [miss]
[pid 1, W] 0x004f --> 0x024f: g [miss]
[pid 1, R] 0x007c --> 0x027c: g [hit]
[pid 0, R] 0x02cb --> 0x00cb: A [miss]
[pid 1, W] 0x06b9 --> 0x01b9: P [miss]
[pid 0, R] 0x0104 --> 0x0204: A [miss]
[pid 0, W] 0x0148 --> 0x0248: h [hit]
[pid 1, W] 0x00e3 --> 0x00e3: h [miss]
[pid 2, R] 0x0477 --> 0x0177: A [miss]
[pid 0, R] 0x01d4 --> 0x02d4: h [hit]
[pid 3, W] 0x0781 --> 0x0281: m [miss]
[pid 3, R] 0x053c --> 0x003c: A [miss]
=====================================
Request: 14
Page Hit: 4 (28.57%)
Page Miss: 10 (71.43%)
Swap Read: 10
Swap Write: 5
```

Name: Mayesha Bintha Mizan
ID-811281302



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 3 example_inputs/example.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 1, Replacement Policy: 3 - CLOCK
[pid 0, W] 0x0700 --> 0x0000: A [miss]
[pid 0, W] 0x0000 --> 0x0100: B [miss]
[pid 0, W] 0x0100 --> 0x0200: C [miss]
[pid 0, W] 0x0200 --> 0x0000: D [miss]
[pid 0, W] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0300 --> 0x0200: F [miss]
[pid 0, R] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0400 --> 0x0000: G [miss]
[pid 0, R] 0x0200 --> 0x0100: D [miss]
[pid 0, R] 0x0300 --> 0x0200: F [hit]
[pid 0, W] 0x0000 --> 0x0200: H [miss]
[pid 0, W] 0x0300 --> 0x0000: I [miss]
[pid 0, R] 0x0200 --> 0x0100: D [hit]
[pid 0, R] 0x0100 --> 0x0100: C [miss]
[pid 0, R] 0x0200 --> 0x0200: D [miss]
[pid 0, R] 0x0000 --> 0x0000: H [miss]
[pid 0, R] 0x0100 --> 0x0100: C [hit]
[pid 0, R] 0x0700 --> 0x0100: A [miss]
[pid 0, R] 0x0000 --> 0x0000: H [hit]
[pid 0, R] 0x0100 --> 0x0200: C [miss]
====================================
Request: 20
Page Hit: 6 (30.00%)
Page Miss: 14 (70.00%)
Swap Read: 14
Swap Write: 8
```



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 3 example_inputs/input2.txt
# PAGES: 4, # FRAMES: 3, # PROCS: 2, Replacement Policy: 3 - CLOCK
[pid 1, W] 0x003b --> 0x003b: A [miss]
[pid 0, W] 0x01c6 --> 0x01c6: K [miss]
[pid 1, W] 0x026a --> 0x026a: Y [miss]
[pid 0, R] 0x015b --> 0x015b: K [hit]
[pid 1, W] 0x030e --> 0x000e: u [miss]
[pid 0, R] 0x00a9 --> 0x01a9: A [miss]
[pid 0, W] 0x0387 --> 0x0287: q [miss]
[pid 0, R] 0x03d7 --> 0x02d7: q [hit]
[pid 1, R] 0x006b --> 0x006b: A [miss]
[pid 0, R] 0x0398 --> 0x0298: q [hit]
[pid 1, W] 0x0392 --> 0x0192: D [miss]
[pid 0, R] 0x00bb --> 0x02bb: A [miss]
[pid 0, R] 0x024e --> 0x004e: A [miss]
[pid 1, R] 0x0158 --> 0x0158: A [miss]
[pid 1, W] 0x03f7 --> 0x02f7: g [miss]
[pid 1, W] 0x0181 --> 0x0181: d [hit]
[pid 0, R] 0x00c4 --> 0x00c4: A [miss]
[pid 1, W] 0x03db --> 0x02db: q [hit]
[pid 1, R] 0x03fc --> 0x02fc: q [hit]
[pid 0, W] 0x008e --> 0x008e: h [hit]
====================================
Request: 20
Page Hit: 7 (35.00%)
Page Miss: 13 (65.00%)
Swap Read: 13
Swap Write: 6
[mb69711@csci-odin OS_Proj3]$ 
```

The table illustrates the **Clock page replacement algorithm**'s performance as the number of physical frames increases from three to six using belady input.

Number of Frames	Page Hits	Hit Rate (%)	Page Misses	Miss Rate (%)	Swap Reads	Swap Writes
3	3	25.00%	9	75.00%	9	4
4	2	16.67%	10	83.33%	10	5
5	5	41.67%	7	58.33%	7	0
6	7	58.33%	5	41.67%	5	0

CLOCK-Pro Page Replacement Algorithm

The **CLOCK-Pro** page replacement algorithm in the simulator enhances the traditional CLOCK strategy by incorporating distinctions between **hot** and **cold** pages, thereby optimizing memory management through more nuanced tracking of page usage. The algorithm employs separate hands **HAND_COLD**, **HAND_HOT**, and **HAND_EVICTION** to manage different aspects of page replacement. The handle_cold_hand() function focuses on **cold** pages, identifying candidates for eviction by checking if a cold page has not been recently referenced (referenceBits [HAND_COLD] == 0) or if the number of hot pages exceeds a predefined **HOT_RATIO**. If a suitable cold page is found, it is marked for eviction by updating the clockProEvicted array and advancing the HAND_COLD pointer in a cyclical manner. Conversely, if a cold page is referenced, it is promoted to the **hot** category using the promote_to_hot(int PFN) function, which updates the relevant tracking arrays and increments the hot_pages_count.

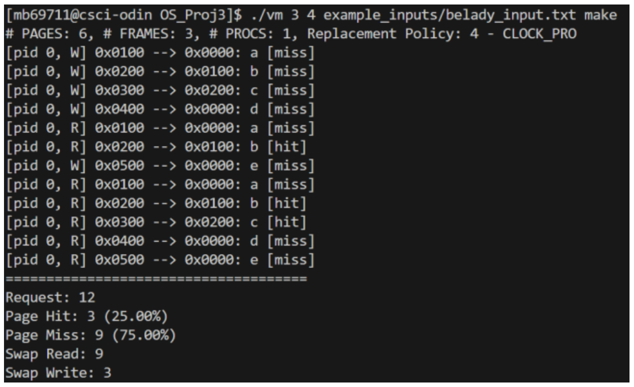
The **handle_hot_hand() function** manages **hot** pages by monitoring their reference bits. If a hot page is accessed (referenceBits[HAND_HOT] == 1), its reference bit is cleared to indicate recent usage. Should a hot page's reference bit remain unset and the number of hot pages exceeds the **HOT_RATIO**, the page is demoted to cold using the demote_from_hot(int PFN) function, thereby reducing the hot_pages_count and allowing the algorithm to maintain an optimal balance between hot and cold pages.

Name: Mayesha Bintha Mizan
ID-811281302

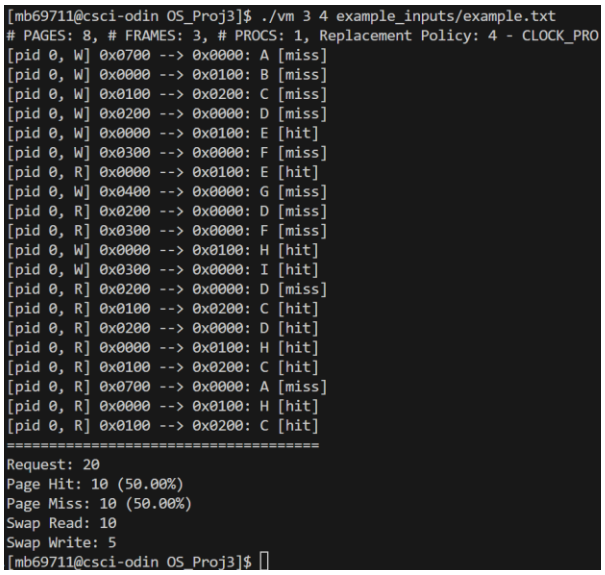
Additionally, **the handle_eviction_hand() function** oversees the **eviction** process by checking if an evicted page has been accessed again (referenceBits[HAND_EVICTION] == 1). If so, the page is re-promoted to cold, ensuring that frequently accessed pages are retained in memory. If not, the evicted page is permanently removed from the system, maintaining data integrity and freeing up resources.

The primary function, clock_pro(), maintain the overall replacement process by first attempting to evict non-referenced cold pages and, failing that, resetting reference bits and selecting frames based on the current state of hot pages. This dual-hand approach allows CLOCK-Pro to effectively prioritize pages that are both frequently and recently accessed, minimizing page faults and enhancing memory utilization. By dynamically promoting and demoting pages between hot and cold categories and efficiently handling evictions, CLOCK-Pro achieves a good balance between performance and resource management, making it a robust choice for optimizing virtual memory systems in environments with varying access patterns and memory constraints.

I tested Clock with belady_input, example and I did the algorithm by hand and check that the result is correct.



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 4 example_inputs/belady_input.txt make
# PAGES: 6, # FRAMES: 3, # PROCS: 1, Replacement Policy: 4 - CLOCK_PRO
[pid 0, W] 0x0100 --> 0x0000: a [miss]
[pid 0, W] 0x0200 --> 0x0100: b [miss]
[pid 0, W] 0x0300 --> 0x0200: c [miss]
[pid 0, W] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0100 --> 0x0000: a [miss]
[pid 0, R] 0x0200 --> 0x0100: b [hit]
[pid 0, W] 0x0500 --> 0x0000: e [miss]
[pid 0, R] 0x0100 --> 0x0000: a [miss]
[pid 0, R] 0x0200 --> 0x0100: b [hit]
[pid 0, R] 0x0300 --> 0x0200: c [hit]
[pid 0, R] 0x0400 --> 0x0000: d [miss]
[pid 0, R] 0x0500 --> 0x0000: e [miss]
====================================
Request: 12
Page Hit: 3 (25.00%)
Page Miss: 9 (75.00%)
Swap Read: 9
Swap Write: 3
```



```
[mb69711@csci-odin OS_Proj3]$ ./vm 3 4 example_inputs/example.txt
# PAGES: 8, # FRAMES: 3, # PROCS: 1, Replacement Policy: 4 - CLOCK_PRO
[pid 0, W] 0x0700 --> 0x0000: A [miss]
[pid 0, W] 0x0000 --> 0x0100: B [miss]
[pid 0, W] 0x0100 --> 0x0200: C [miss]
[pid 0, W] 0x0200 --> 0x0000: D [miss]
[pid 0, W] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0300 --> 0x0000: F [miss]
[pid 0, R] 0x0000 --> 0x0100: E [hit]
[pid 0, W] 0x0400 --> 0x0000: G [miss]
[pid 0, R] 0x0200 --> 0x0000: D [miss]
[pid 0, R] 0x0300 --> 0x0000: F [miss]
[pid 0, W] 0x0000 --> 0x0100: H [hit]
[pid 0, W] 0x0300 --> 0x0000: I [hit]
[pid 0, R] 0x0200 --> 0x0000: D [miss]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
[pid 0, R] 0x0200 --> 0x0000: D [hit]
[pid 0, R] 0x0000 --> 0x0100: H [hit]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
[pid 0, R] 0x0700 --> 0x0000: A [miss]
[pid 0, R] 0x0000 --> 0x0100: H [hit]
[pid 0, R] 0x0100 --> 0x0200: C [hit]
====================================
Request: 20
Page Hit: 10 (50.00%)
Page Miss: 10 (50.00%)
Swap Read: 10
Swap Write: 5
[mb69711@csci-odin OS_Proj3]$ 
```

The table illustrates the **Clock Pro page replacement algorithm**'s performance as the number of physical frames increases from three to six using belady input.

Number of Frames	Page Hits	Hit Rate (%)	Page Misses	Miss Rate (%)	Swap Reads	Swap Writes
3	3	25.00%	9	75.00%	9	3
4	5	41.67%	7	58.33%	7	2
5	7	58.33%	7	41.67%	7	0
6	7	58.33%	5	41.67%	5	0

The implementation and evaluation of the **First-In-First-Out (FIFO)**, **Least Recently Used (LRU)**, **CLOCK**, and **CLOCK-Pro** page replacement algorithms reveal distinct performance characteristics influenced by the number of physical frames allocated.

FIFO demonstrates a simplistic approach by evicting the oldest loaded pages without considering their access frequency or recency. For instance, with **3 frames**, FIFO incurs a **75.00% miss rate**, leading to **9 swap reads** and **4 swap writes**, indicating inefficiency in managing active memory demands.

Name: Mayesha Bintha Mizan
ID-811281302

LRU, on the other hand, intelligently tracks page usage patterns by retaining the most recently accessed pages. This strategy significantly reduces miss rates compared to FIFO. With **4 frames**, LRU achieves a **33.33% hit rate**, outperforming FIFO by effectively minimizing unnecessary page evictions. As the number of frames increases to **5 and 6**, LRU maintains a **58.33% hit rate**, aligning with FIFO's performance but with inherently better adaptability to dynamic access patterns.

The **CLOCK** algorithm, an approximation of LRU, utilizes a circular buffer and reference bits to manage page replacements. While CLOCK shows comparable performance to FIFO at lower frame counts, with a **25.00% hit rate** at **3 frames**, it fails to surpass FIFO's efficiency. However, as the frame count increases to **5 and 6**, CLOCK aligns its performance with FIFO and LRU, achieving a **58.33% hit rate** by better approximating LRU's behavior without the full overhead.

CLOCK-Pro emerges enhancement over traditional CLOCK by distinguishing between **hot** and **cold** pages. This dual categorization allows CLOCK-Pro to prioritize frequently accessed pages, thereby optimizing memory utilization more effectively than FIFO and CLOCK. With **4 frames**, CLOCK-Pro achieves a **41.67% hit rate**, outperforming both FIFO and CLOCK by reducing miss rates and swap operations. At **5 and 6 frames**, CLOCK-Pro matches the optimal performance of FIFO and LRU, maintaining a **58.33% hit rate** without incurring any swap writes. This efficiency underscores CLOCK-Pro's ability to balance complexity and performance, making it a robust choice for environments requiring nuanced memory management.

Among the evaluated page replacement algorithms, **CLOCK-Pro** stands out for its enhanced performance, especially in scenarios with limited physical frames. By intelligently distinguishing between hot and cold pages, CLOCK-Pro optimizes memory usage and minimizes page faults more effectively than FIFO, LRU, and CLOCK. While **FIFO** remains the simplest to implement, its inefficiency in dynamic environments limits its practicality. **LRU** offers a significant improvement by leveraging page access history, making it suitable for systems where access patterns are predictable. **CLOCK** provides a reasonable approximation of LRU with lower overhead but falls short of matching LRU's performance. Consequently, **CLOCK-Pro** is recommended for applications requiring robust and efficient memory management, as it delivers the best balance between complexity and operational efficiency.