MANIPAL
ACADEMY *of* HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

OPERATING SYSTEMS LAB

MINI PROJECT—

VIRTUAL MEMORY MANAGER

Kiran Kishor (200905257)
Devansh Sood (200905177)
Ishaa Sahani (200905254)
Ramsai Rahul (200905255)

INTRODUCTION

Virtual Memory Manager (VMM)

Virtual memory management is a memory management strategy used in a computer's operating system that maps an extremely logical address space onto a smaller physical memory. It allows the execution of programs that are not entirely loaded in main memory, and thus the program size can be larger than the computer's physical memory.

Virtual memory uses both hardware and software to enable a computer to compensate for physical memory shortages, temporarily transferring data from random access memory (RAM) to disk storage.

Advantages of Virtual Memory

Extremely large processes, even larger than the physical memory, can be run without having to load the entire program into the main memory.

The degree of multiprogramming, along with CPU utilization and throughput, can be increased as user programs would occupy less space if they are only partially loaded into memory.

A programmer can have an extremely large virtual address space available to him, separated from the physical memory. This is helpful as the programmer doesn’t have to worry about the limitations of physical memory.

The efficiency of the operating system also increases as the time to swap programs in and out of memory is reduced.

CONCEPTS USED

<u>Demand Paging</u>

The virtual memory manager implements a demand paged-virtual memory, i.e., it manages memory in individual segments called *pages*, using a *pager*. When a process is swapped in, the pager guesses which pages of the process will be needed before the process is swapped out. If a process only uses the pages that have already been brought to memory (i.e., the pages are *memory resident*), the execution of the process continues normally. If a process wants to access a page that has not been brought to memory (an invalid page), it leads to a *page fault*. If a page fault occurs, we bring the required pages into memory for execution.

Hardware required for demand paging:

1. Page table with Translation Look-aside Buffer (TLB)
2. Secondary memory

<u>Page table with TLB:</u> The page table is used to define physical memory as it relates to logical memory, using the page's base address and offset. The TLB is a small and fast hardware cache, which stores the recent translations of virtual memory to physical memory, by using the LRU (least recently used) strategy. The TLB is used to reduce the time required to access a memory location.

<u>Secondary memory:</u> The secondary memory is a high-speed disk that stores the pages that are not present in main memory. Here, we implement secondary memory as a random-access file called **`BACKING_STORE.bin`**. It is a binary file of size 65,536 bytes, so we can randomly access any file position for reading.

Some additional terms:

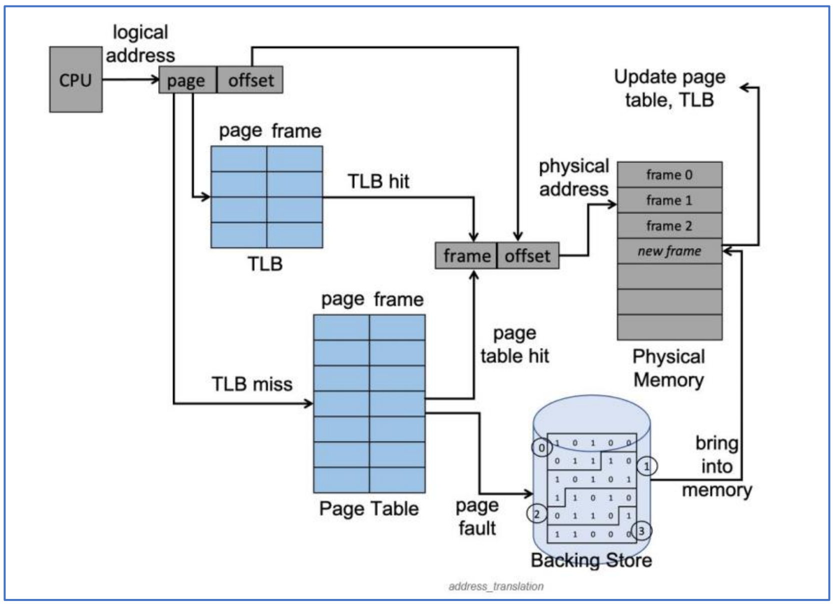
<u>Page fault rate:</u> The page fault rate refers to the percentage of address references that resulted in page faults.

<u>TLB hit:</u> TLB hit is a situation in which the page number that we are looking for is present in the TLB.

<u>TLB hit rate:</u> The TLB hit rate refers to the percentage of address references that were resolved in the TLB, that is the percentage of page numbers that were found in the TLB.

<u>Page Replacement</u>

If a page fault has occurred for a given process, the required page needed by the process must be brought in and assigned a frame. However, a situation may arise in which there are no free frames for the concerned page. With the page replacement technique, we search for a frame which is not being used by its process and free it. The frame's contents are swapped to the secondary memory, (here, **`BACKING_STORE.bin`**) and the page table is updated to indicate that the swapped-out page is no longer present in memory. The freed frame is used to bring in the required page to handle the page fault.



Paging hardware with TLB and backing store

CODE SNIPPET

The code segment that implements page replacement using the LRU algorithm is—

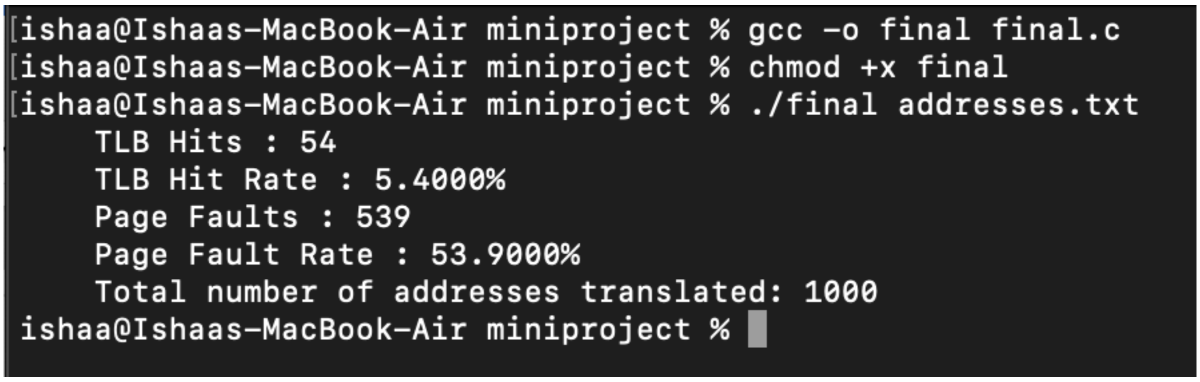
```
int LRU_index(int history[], int size, int min)
//determines which page has been used the least
{
    int index = 0;

    if (size == TLB_SIZE)               // for TLB
    {
        for(int i = 0; i < size; i++)
// search the history array to see if there's any lesser used element
        {
            if(history[i] < min)
            {
                min = history[i];
                index = i;
            }
        }
    }
    else                                // for page table
    {
        for(int i = 0; i < size; i++)
        {
// search the history array to see if there's any element less than min (lesser used)
            if(history[i] < min && history[i] != 0)
            {
                min = history[i];
                index = i;
            }
        }
    }

    return index;
// returns the index of the TLB/page table to be replaced
}
```

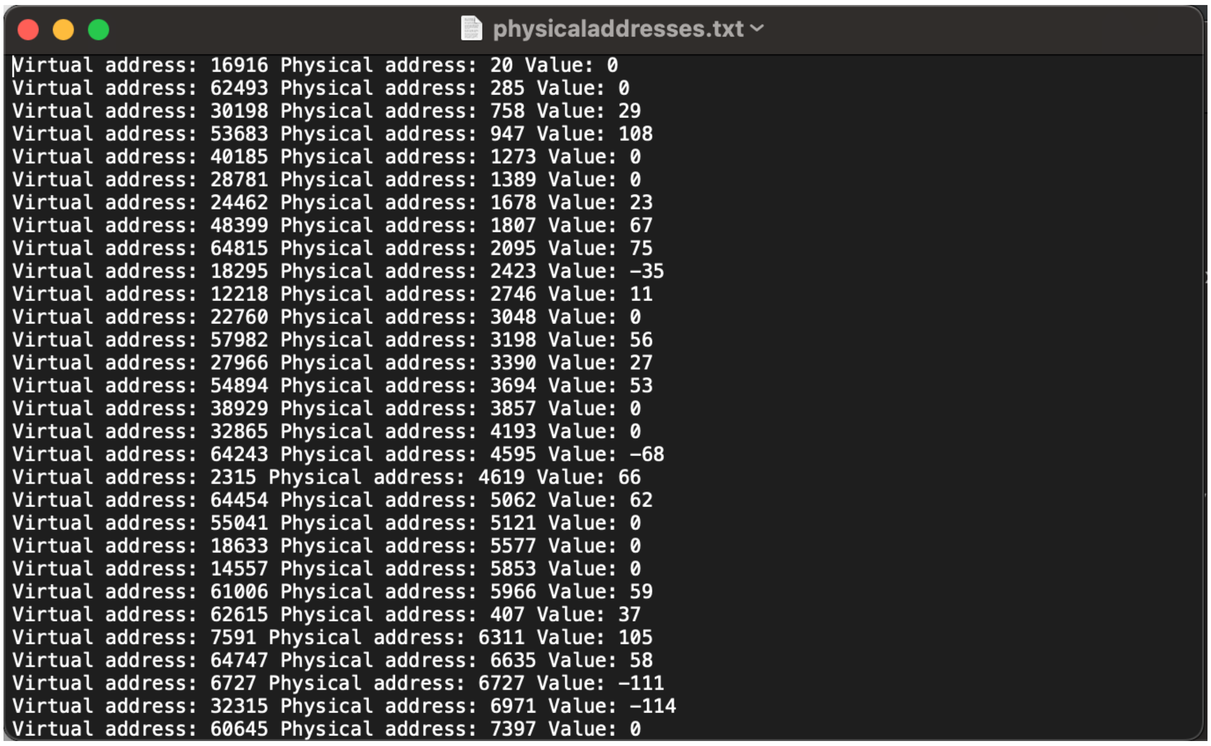
RESULTS

The output of the program when executed is—



```
[ishaa@Ishaas-MacBook-Air miniproject % gcc -o final final.c
[ishaa@Ishaas-MacBook-Air miniproject % chmod +x final
[ishaa@Ishaas-MacBook-Air miniproject % ./final addresses.txt
     TLB Hits : 54
     TLB Hit Rate : 5.4000%
     Page Faults : 539
     Page Fault Rate : 53.9000%
     Total number of addresses translated: 1000
 ishaa@Ishaas-MacBook-Air miniproject %
```

This is the output file—



physicaladdresses.txt

```
Virtual address: 16916 Physical address: 20 Value: 0
Virtual address: 62493 Physical address: 285 Value: 0
Virtual address: 30198 Physical address: 758 Value: 29
Virtual address: 53683 Physical address: 947 Value: 108
Virtual address: 40185 Physical address: 1273 Value: 0
Virtual address: 28781 Physical address: 1389 Value: 0
Virtual address: 24462 Physical address: 1678 Value: 23
Virtual address: 48399 Physical address: 1807 Value: 67
Virtual address: 64815 Physical address: 2095 Value: 75
Virtual address: 18295 Physical address: 2423 Value: -35
Virtual address: 12218 Physical address: 2746 Value: 11
Virtual address: 22760 Physical address: 3048 Value: 0
Virtual address: 57982 Physical address: 3198 Value: 56
Virtual address: 27966 Physical address: 3390 Value: 27
Virtual address: 54894 Physical address: 3694 Value: 53
Virtual address: 38929 Physical address: 3857 Value: 0
Virtual address: 32865 Physical address: 4193 Value: 0
Virtual address: 64243 Physical address: 4595 Value: -68
Virtual address: 2315 Physical address: 4619 Value: 66
Virtual address: 64454 Physical address: 5062 Value: 62
Virtual address: 55041 Physical address: 5121 Value: 0
Virtual address: 18633 Physical address: 5577 Value: 0
Virtual address: 14557 Physical address: 5853 Value: 0
Virtual address: 61006 Physical address: 5966 Value: 59
Virtual address: 62615 Physical address: 407 Value: 37
Virtual address: 7591 Physical address: 6311 Value: 105
Virtual address: 64747 Physical address: 6635 Value: 58
Virtual address: 6727 Physical address: 6727 Value: -111
Virtual address: 32315 Physical address: 6971 Value: -114
Virtual address: 60645 Physical address: 7397 Value: 0
```

CONCLUSION

We have successfully implemented a virtual memory manager using LRU demand paging. When the program is compiled and executed with proper input files, we get the number of TLB hits and hit ratio, page fault and fault rate and the total number of logical addresses translated into physical addresses by the program.

The LRU policy is a widely used page-replacement algorithm and is considered to be efficient.

BIBLIOGRAPHY

1. A. Silberschatz, P. B. Galvin and G. Gagne, *Operating System Concepts (9c)*, Wiley and Sons (Asia) Pte Ltd, 2013.

2. https://www.geeksforgeeks.org/translation-lookaside-buffer-tlb-in-paging/

3. https://www.geeksforgeeks.org/page-replacement-algorithms-in-operating-systems/

4. https://cse.unl.edu/~jbradley/courses/2020_spring/CSCE451-851-OS/PA5.html