VIRTUAL MEMORY

The University of Manchester Atlas Computer was the first computer to feature true virtual memory.

Vaida Diana-Laura

Technical University of Cluj-Napoca, 2023

VIRTUAL MEMORY SIMULATOR

Contents

1.Introduction

1.1.Context

In this project I want to create a simulator for the virtual memory. Virtual memory gives the impression there is more memory available than it actually is. It is a storage area that holds the files on your hard drive for retrieval when a computer runs out of RAM. Virtual memory provides virtual address mapping between applications and hardware memory, it frees up RAM by swapping data that has not been used recently over to a storage device, such as a hard drive or solid-state drive (SSD). It is important for improving system performance, multitasking and using large programs. Almost all computers nowadays use it because it is part of a computer's CPU and it is more cheaper than RAM. You have the ability to deactivate it in a computer but it isn't a really good idea.

1.2.Specifications

Virtual memory can use paging or segmentation. But my project will go down the paging approach.

The application is going to be implemented in Java using Intellij IDEA and Eclipse IDE(for the interface).

User interaction: Although virtual memory isn't a really known subject outside of computer science or once that is talked about a lot in everyday conversations, the project will have the goal to make it as user friendly as possible.

Project plan

Week 1: Research and document about the subject

Week 2: Start to create the interface and the classes needed

Week 3: Implement the operations of the project, the methods

Week 4: Modify or add methods to make it better

Week 5: Finish the documentation

Week 6: Check everything again

1.3.Objectives

- help users be more educated on the subject

-make an application that is as close as the real thing as possible

2.Bibliographic study

In computing, virtual memory, or virtual storage, is a memory management technique which creates the illusion to users of a very large (main) memory.

The computer's operating system, using a combination of hardware and software, maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory. Main storage, as seen by a process or task, appears as a contiguous address space or collection of contiguous segments. The operating system manages virtual address spaces and the assignment of real memory to virtual

memory. Address translation hardware in the CPU, often referred to as a memory management unit (MMU), automatically translates virtual addresses to physical addresses. Software within the operating system may extend these capabilities, utilizing, e.g., disk storage, to provide a virtual address space that can exceed the capacity of real memory and thus reference more memory than is physically present in the computer.

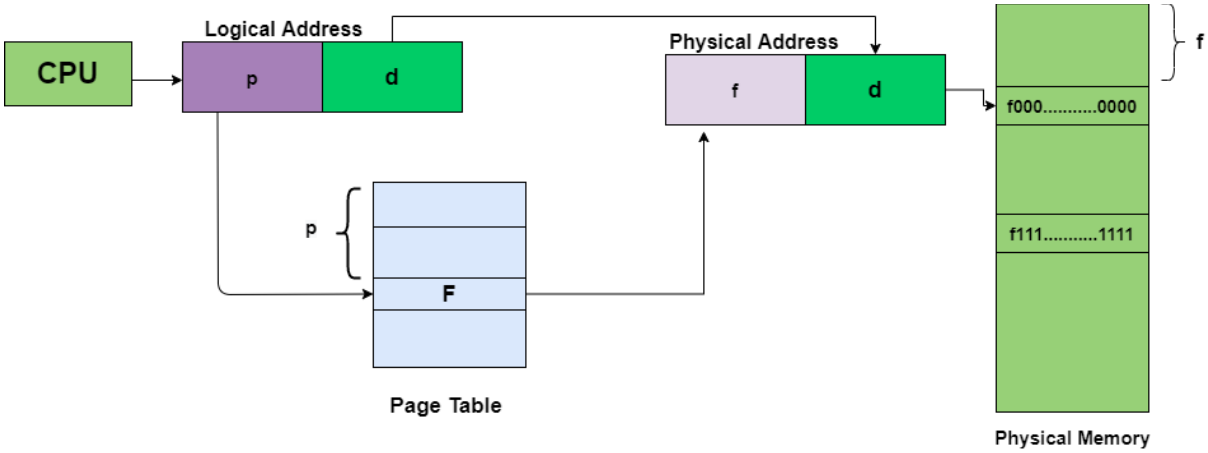
offset =input bits

logical address(page number + offset)= log2(input) bits

physical page rows(from the physical memory formed of physical addresses) = physical page size/ 2^ offset

page table(nr of rows) = virtual memory size / 2^offset

translation lookaside buffer(type of memory cache that stores recent translations of virtual memory to physical addresses to enable faster retrieval) = input bits



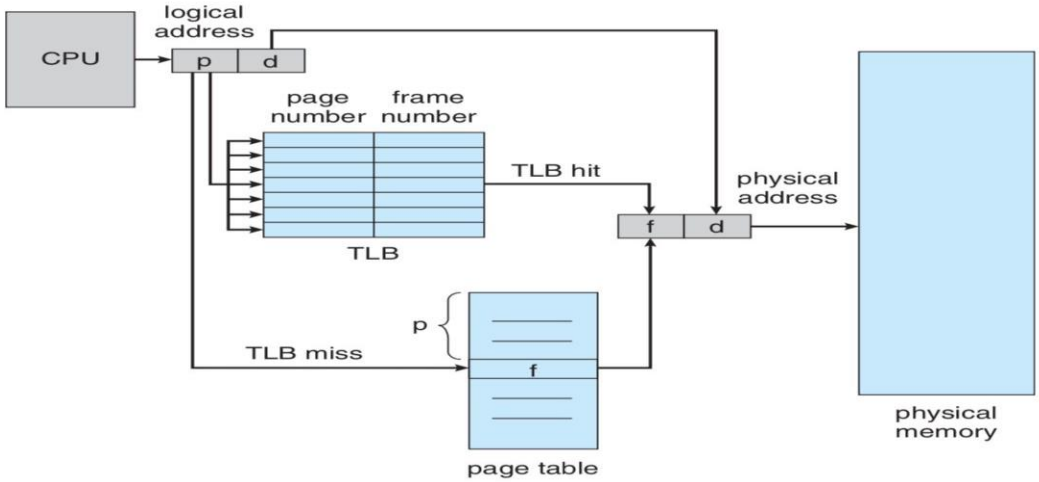
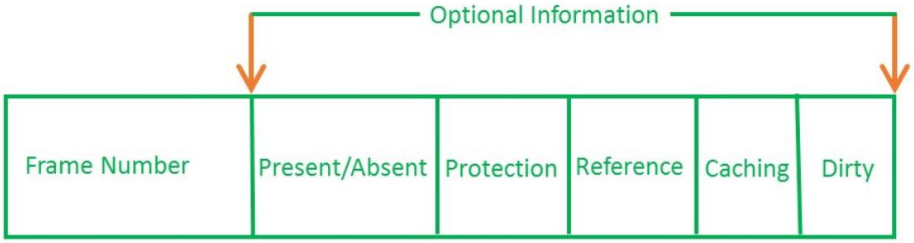


Figure 8.14 Paging hardware with TLB.



PAGE TABLE ENTRY

Information Stored in Page Table Entry

- **Frame Number –** It gives the frame number in which the current page you are looking for is present.
- **Present/Absent Bit:** Present or absent bit says whether a particular page you are looking for is present or absent. In case it is not present, that is called Page Fault. It is set to 0 if the corresponding page is not in memory.
- **Protection Bit:** The protection bit says what kind of protection you want on that page. So, these bits are for the protection of the page_frame (read, write, etc).
- **Referenced Bit:** Referenced bit will say whether this page has been referred to in the last clock cycle or not. It is set to 1 by hardware when the page is accessed.
- **Caching Enabled/Disabled:** Sometimes we need fresh data. Let us say the user is typing some information from the keyboard and your program should run according to the input given by the user. In that case, the information will come into the main memory.
- **Modified/Dirty Bit:** Modified bit says whether the page has been modified or not. Modified means sometimes you might try to write something onto the page. If a page is modified, then whenever you should replace that page with some other page, then the modified information should be kept on the hard disk or it has to be written back or it has to be saved back. It is set to 1 by hardware on the write-access to a page which is used to avoid writing when swapped out.

3.Analysis

3.1.Paging

In computer operating systems, memory paging is a memory management scheme by which a computer stores and retrieves data from secondary storage for use in main memory. In this scheme, the operating system retrieves data from secondary storage in same-size blocks called pages.

Finding information

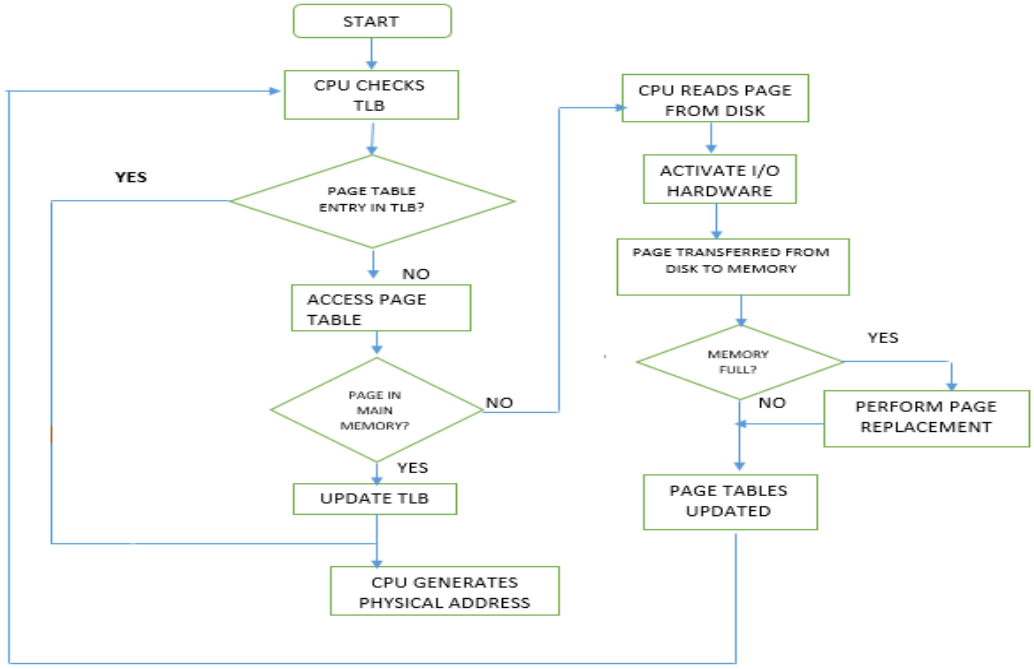
I can find information checking the physical memory and using a TLB and a page table with the following diagram.

The page table, generally stored in main memory, keeps track of where the virtual pages are stored in the physical memory. First, the page table is looked up for the frame number. Second, the frame number with the page offset gives the actual address. Thus, any straightforward virtual memory scheme would have the effect of doubling the memory access

time. Hence, the TLB is used to reduce the time taken to access the memory locations in the page-table method. The TLB is a cache of the page table, representing only a subset of the page-table contents.

Translation lookaside buffer case diagram

This diagram shows what happens



The flowchart provided explains the working of a TLB. If it is a TLB miss, then the CPU checks the page table for the page table entry. If the present bit is set, then the page is in main memory, and the processor can retrieve the frame number from the page-table entry to form the physical address.

Replacing a page

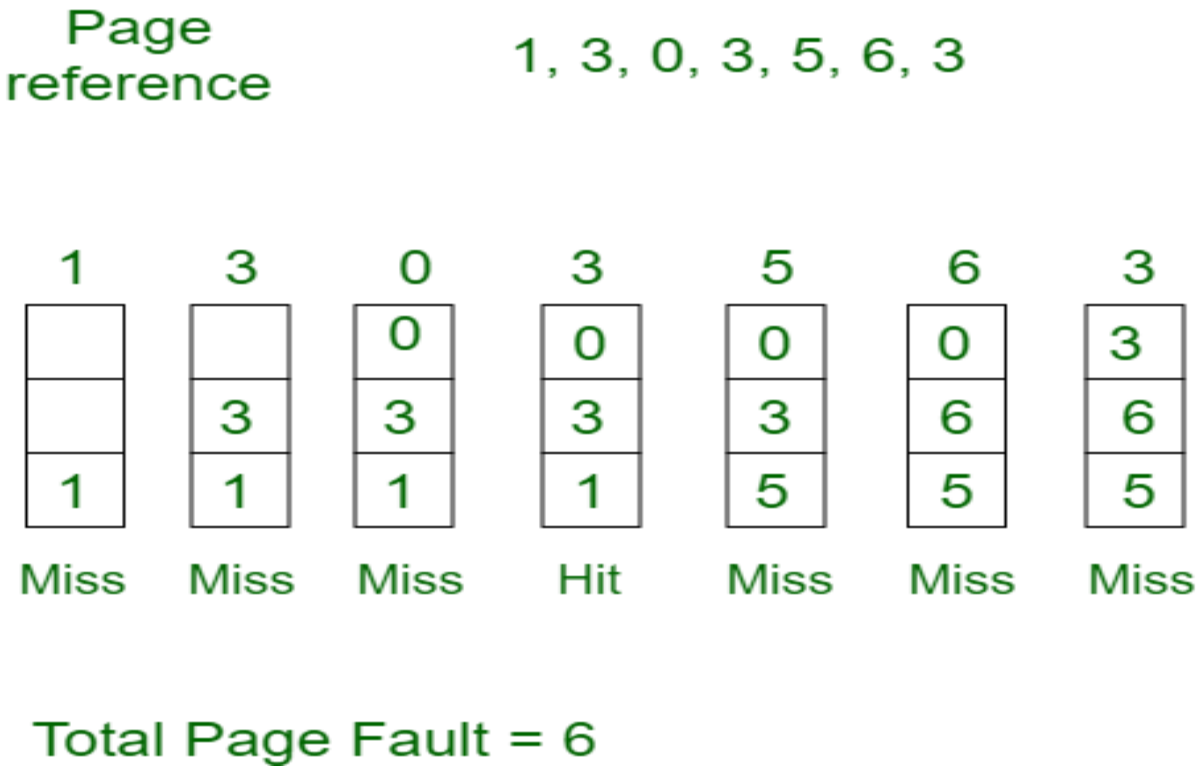
This technique is used for the translation lookaside buffer. It is helpful because TLB stores the most recently accessed pages, without going into the page table and doing a page walk. The page walk is time-consuming when compared to the processor speed, as it involves reading the contents of multiple memory locations and using them to compute the physical address. After the physical address is determined by the page walk, the virtual address to physical address mapping is entered into the TLB.

There are some algorithms to replace a page, like LRU, optiomal and FIFO.

FIFO(page replacement algorithm)

There are many types of page replacement algorithms, but the one I want to use is FIFO.

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.



Placing a page in memory

Placing a page in memory, also known as "page-in" or "loading a page," is the process of bringing a page of data from secondary storage (e.g., a disk) into physical memory (RAM) so that it can be accessed by a running process. This operation is a fundamental aspect of virtual memory systems and is necessary to ensure that a process can access its required data efficiently.

4.Design

My project will contain an input number of processes, which will have randomly generated logical addreses. The other components will depend on the input from the user.

A physical memory will be present that is shared between all processes.

The user will have to specificy at first the:

-how many processes to be accessed

-physical page size

-offset bits for a process

-virtual memory size of a process

-the number of TLB(translation lookaside buffer) entries for a process

If any of these inputs isn't correct (like a negative value or a virtual memory size that isn't a power of 2, the program will generate an error.) Then the physical memory will be generated and the TLB and page table for every process.

Some of the operations include: finding information, placing a page in main memory, replacing a page, these instructions will be represented by buttons. The next instruction is an input from the user. Each process has it's own virtual memory.

Statics will be included as well like TLB and page table misses and hits.

I want to represent every important component from my project with classes.

The classes in my project are:

Processes – a class with a map and an index of the process, and a list of virtual addresses(which is another class)

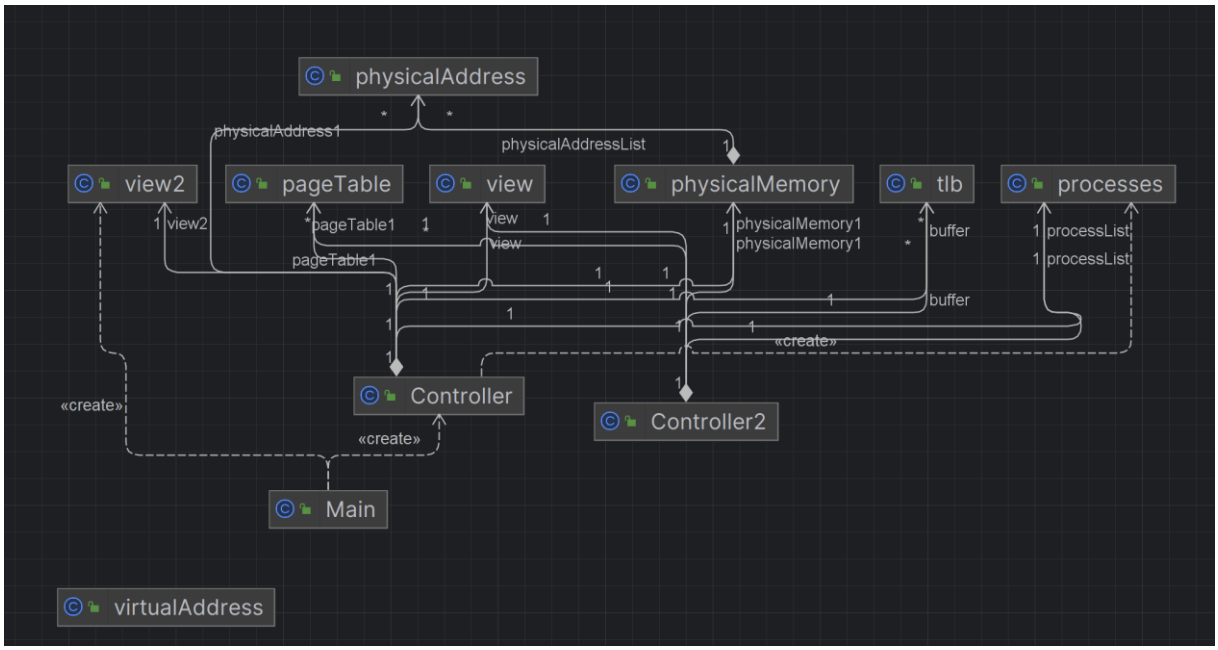
Virtual address – a class containing the page number and the offset

Translation lookaside buffer – a class containing the page number and the frame number, and the process

Page table – a class containing the index number(the processes it belongs to),the page,the frame,and the valid bit

The shared physical memory – a class containing a list of the physical addresses

Physical address – a class with the frame number and the content



5.Implementation

I have two classes,Controller and Controller2 that connect all my classes together.

At first, I generate a maximum of 10 random virtual addresses for every process. Every page for every process will be unique. I also have two main views. Every controller belongs to a view.

Processes- This is a class that contains the procceses, I store here a map with an index that represents the number of the process and a map with the page and frame number.

toString method – this method takes the list of virtual addresses of every process and prints it

addProcess method – puts a virtual address list at a certain process

findInformation method – this method checks if a page is already in the virtual space, and it helps the Controller2 to not add the page in that specific process if it already exists

Virtual addresses- This is contain the page number and the offset that makes up the logical address.

Generate method – this is the method I use at first to generate random logical addresses for every process based on calculations from the input, I make sure to not duplicate the same page twice using the isNumberPageInList method

Physical Addresses – this class contains the frame number and the content of that frame

Initialize method – here I add the physical addreses in a list,initializing every content with -1 for every frame, number of frames I calculated based on computations

addAddress method – using the page table I check which pages from the page table have the valid bit, and I put them in the RAM

hitMiss method – I check with the help of the translation lookaside buffer and page table if the page of a process from the input is mapped in memory,if it is is a hit if not is a miss

findInformation method – I use this method for the ''Find information'' button, I check if a page from a process exists mapped in the memory, first I check the translation lookaside buffer, then the page table, and if it doesn't exist, I send a message for every case

physicalMemory – this class contains a list with the physical addresses

toString method – i print the physical addresses that are contained in the list

tlb – here I have the process number,the frame and the page

initialize method – this method add a new entry to the list of translation lookaside buffer, I created a number of frames based on input with empty pages,that are going to be populated

toString method – prints the translation lookaside buffer

addTo method – this method uses a FIFO algorithm

I check if the tlb page at that certain process already exists in the tlb, if it does I leave it like that,generating a hit, if it's a different process I simply change the process.

If the page doesn't exist in the tlb, I add it at the next frame. If the tlb is full, I add it least recently accessed frame.

pageTable – class containing the process,frame,page and valid/invalid bit

initialize method – this method initializes the page table

toString method – prints a page tables of a certain process given as an input

addTo method – this method uses a FIFO algorithm

I check if the page at that certain process already exists in the page table, if it does I leave it like that,generating a hit, if it's a different process I simply change the process,I change the valid/invalid bit for the pages in the page table,but i leave the page in the RAM where it is.

If the page doesn't exist in the page table, I add it at the next frame. If the page table is full, I add it least recently accessed frame.

6.Testing and validation

Virtual memory size=2048

Physical page size=128

Offset bits for a process=2

Number of TLB=2

Nr of processes=1

Virtual addresses

Process 1

61 4

31 4

2 0

27 1

62 2

48 0

RAM

0 61

1 31

2 2

3 27

4 62

5 48

Page Table

2 2

27 3

31 1

58 5

61 0

62 4

TLB

0:62, 4

1:48, 5

Placing page 62 from process 1,generates a hit and the TLB remains the same as well as the RAM.

7.Conclusion

I didn’t manage to do everything I had planned,unfortunately. Next time I will have to do a project I will plan my time better.

I didn’t manage to implement the different kind of bits that are contained in a page table,excepting the valid/invalid bit. I also didn’t represent the virtual and physical addresses in hexadecimal. My project uses FIFO, not LRU.

8.Bibliography

1.Wikipedia page about virtual memory https://en.wikipedia.org/wiki/Virtual_memory

2. https://www.techtarget.com/searchstorage/definition/virtual-memory

3. https://www.javatpoint.com/os-virtual-memory

4. https://www.geeksforgeeks.org/virtual-memory-in-operating-system/

5. Lecture slides, prof. Gheorghe Sebestyen, Structures of computer systems
https://moodle.cs.utcluj.ro/mod/folder/view.php?id=46903

6.Lecture slides, prof. Adrian Coleșa, Operating systems
https://moodle.cs.utcluj.ro/mod/folder/view.php?id=41753

7. https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/9_VirtualMemory.html