**Designing a Virtual Memory Manager**

A Report on the package submitted by :

1. Shankar Narayanan D - 18PT33
2. Venkatesan S - 18PT40

**Subject** : **Operating Systems** – 18XT44



Department of Applied Mathematics and Computational Sciences

PSG College Of Technology

Coimbatore - 641004

**Designing a Virtual Memory Manager**

**TABLE OF CONTENTS :**

1. **Abstract of the Project**
   1. **Introduction**
   2. **Description of the Concept and Problem**
   3. **Library Calls**
   4. **Tools and Technology**
   5. **Workflow**
   6. **Results and Discussion**
   7. **Conclusion**
   8. **Bibliography**
2. **Abstract :**

**“I wanted to have virtual memory, at least as it's coupled with file systems” - Ken Thomson**

The Report will go through the **concept of Virtual Memory** from the perspective

of an Operating System and our Implementation of the same. It first presents an Introduction

to the concepts of virtual memory then it addresses handling of page faults.

This report further attends to the Idea, Scope and Workflow of our Implementation.

The concept of Demand Paging is used for handling page faults.

* 1. **Introduction**

The concept of virtual memory is the key to managing multiple processes efficiently

with the limits of the physical memory of a system. Virtual memory allows programs

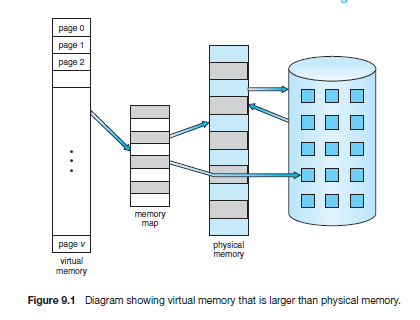
to execute with memory footprints that are larger than the available physical memory.

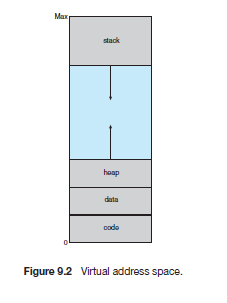
Paging is a memory management scheme that eliminates the need for contiguous allocation

of physical memory. This scheme permits the physical address space of a process to be non –

contiguous. Demand Paging suggests keeping all pages of the frames in the secondary

memory until they are required.





* 1. **Description of the Concept and Problem**

In Operating System, for each process page table will be created, which will

contain  Page Table Entry . This PTE will contain information like frame number, and

some other useful bits (e.g., valid/invalid bit). This page table entry will tell where in

the main memory the actual page is residing.A high-speed cache is set up for page table

entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is

nothing but a special cache used to keep track of recently used transactions. TLB contains

page table entries that have been most recently used.

* + 1. **Description of the problem**

This project consists of writing a program that translates logical to physical

addresses for a virtual address space of size 216 = 65,536 bytes.

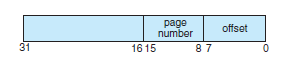
* + 1. **Specifics**

Your program will read addresses.txt file containing several 32-bit integer numbers

that represent logical addresses. However, you need only be concerned with 16-bit

addresses, so you must mask the rightmost 16 bits of each logical address.

These 16 bits are divided into (1) an 8-bit page number and (2) 8-bit page offset.

****

Other specifics include the following:

• 28 entries in the page table

• Page size of 28 bytes

• 16 entries in the TLB

• Frame size of 28 bytes

• 256 frames

• Physical memory of 65,536 bytes (256 frames × 256-byte frame size)

* + 1. **Address Translation**

Your program will translate logical to physical addresses using a TLB and page

table as outlined in Section 8.5. First, the page number is extracted from the

logical address, and the TLB is consulted. In the case of a TLB-hit, the frame

number is obtained from the TLB. In the case of a TLB-miss, the page table

must be consulted. In the latter case, either the frame number is obtained

from the page table or a page fault occurs.

* + 1. **Handling Page Faults**

Your program will implement demand paging. The

backing store is represented by the file BACKING STORE.bin, a binary file of size

65,536 bytes.Whena page fault occurs, you will read in a 256-byte page from the

file BACKING STORE and store it in an available page frame in physical memory.

* + 1. **How to Run Your Program**

Run the command:

**./a.out addresses.txt**

**1.3. Library Calls**

|  |  |
| --- | --- |
| Library Calls | Function it is used in |
| fopen() | int main(int argc,char \*\*argv)  fopen("BACKING\_STORE.bin"."rb"); |
| fseek() | void read\_from\_store(int page\_number)  fseek(output\_file,page\_number \* 256,SEEK\_SET) |
| fread() | void read\_from\_store(int page\_number)  fread(buffer,sizeof(signed char),256,backing\_store) |
| fclose() | int main(int argc,char \*\*argv)  fclose(address\_file);  fclose(backing\_store); |

**1.4 Tools and Technologies used**

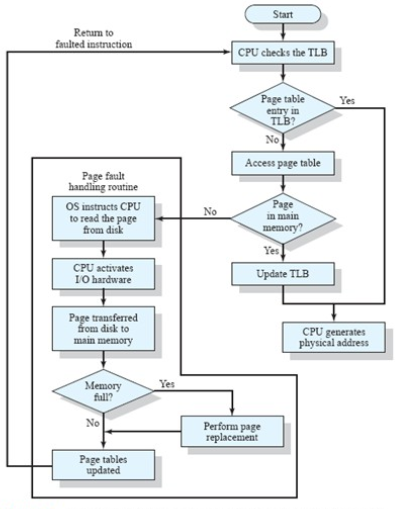
We have made use of the following tools for implementing our project :

* **Platform** : Linux Workstation - Ubuntu Desktop
* **Operating System** : Ubuntu
* **Compiler** : GCC
* **Programming Language** : C

We have used the following tools for the video presentation :

* **Screen Record** : Camtasia
* **Presentation** : Google Slides

**1.5 Workflow**

****

**TLB Hit :**

First the CPU checks the TLB for the frame number of the respective page number.If the page number is present in the TLB, it is a TLB hit.Then the CPU fetches the frame number of that page number. It then generates the physical address with the help of displacement from the logical address and frame number from the TLB.

**TLB Miss :**

In the event of not getting the page number in the TLB, the CPU searches for the page number in the page table. If the page number is found it fetches it’s frame number. It generates the physical address.

**Page Fault :**

If the page numberis not found in both TLB and page table then a fault occurs. If the page to be accessed is not in the physical storage then it is a page fault. A concept of Demand Paging is used. You will read in a 256-byte page from the file BACKING STORE and store it in an available page frame in physical memory. For example, if a logical address with page number 15 resulted in a page fault, your program would read in page 15 from BACKING STORE (remember that pages begin at 0 and are 256 bytes in size) and store it in a page frame in physical memory. Once this frame is stored (and the page table and TLB are updated), subsequent accesses to page 15 will be resolved by either the TLB or the page table.

**1.6 Results and Discussions**

It translate each logical address to a physical address and determine the contents of the signed byte stored at the correct physical address.

Program is to output the following values:

**1.** The logical address being translated (the integer value being read from

addresses.txt).

**2.** The corresponding physical address (what your program translates the

logical address to).

**3.** The signed byte value stored at the translated physical address

**1.6.1 Statistics**

After completion, your program is to report the following statistics:

1. Page-fault rate—The percentage of address references that resulted in

page faults.

1. TLB hit rate—The percentage of address references that were resolved in

the TLB.

Since the logical addresses in addresses.txt were generated randomly and do not reflect any

memory access locality, do not expect to have a high TLB hit rate.

**1.7. Conclusion**

Since the logical addresses in file addresses.txt were randomly generated, the TLB hit ratio for our program is too less. To get a high TLB hit ratio we should use some good replacement algorithm like Least Recently Used (LRU) or the given logical address should reflect some real time program’s one.

**1.8. Bibliography**

Book :

[1] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System

Concepts, Wiley publication, 9 th ed.