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

TITLE: PAGED SEGMENTATION

Submitted by

Sravya D **1RV22CS202**
Varsha V P **1RV22CS225**

Computer Science and Engineering
2023-2024

CONTENTS

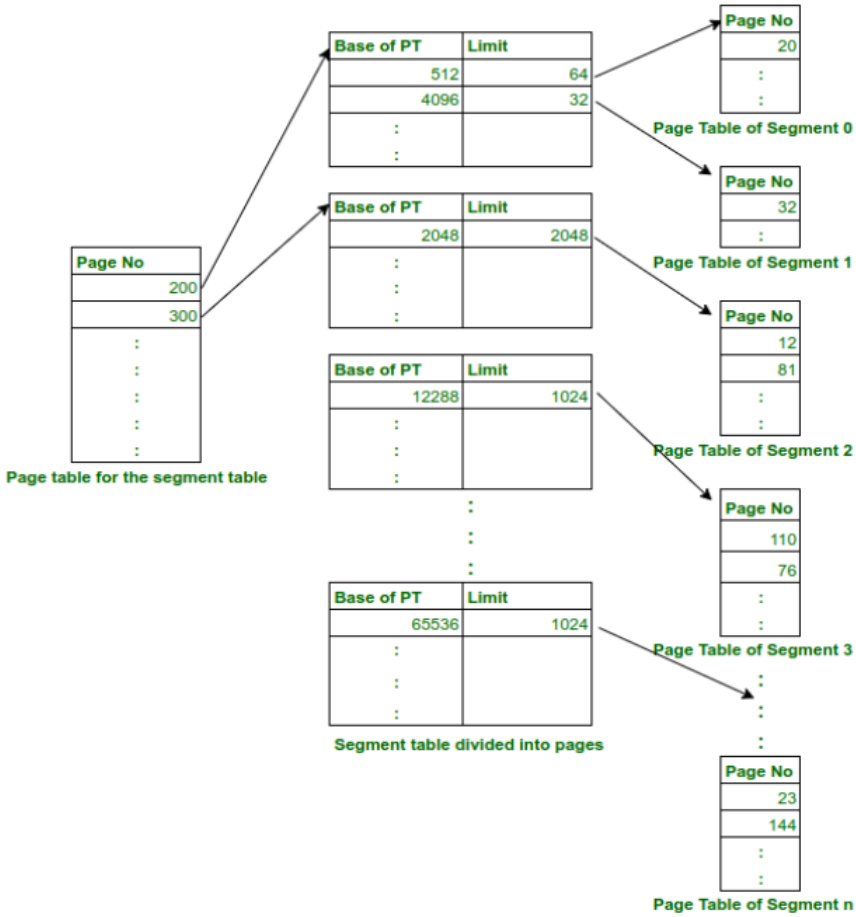
INTRODUCTION

Paged segmentation is a memory management technique employed by operating systems to organize and allocate memory in a structured manner. In paged segmentation, the main memory is divided into fixed-size units called pages, typically smaller than the size of the entire program or process. Similarly, the logical address space of a process is divided into fixed-size units known as logical pages. This segmentation allows for efficient allocation and management of memory resources.

Each process maintains a page table, which serves as a mapping between the logical pages of the process and the corresponding physical pages in the main memory. When a process requests memory, the operating system allocates memory pages to the process based on the available space and updates the page table accordingly. Paged segmentation enables processes to utilize memory dynamically, with the operating system handling the translation of logical addresses to physical addresses transparently to the process.

One of the key benefits of paged segmentation is its flexibility in managing memory. Since memory allocation is done in fixed-size units, the operating system can efficiently allocate and deallocate memory pages as needed, reducing fragmentation and improving overall system performance. Additionally, paged segmentation enables the implementation of virtual memory systems, allowing processes to access more memory than physically available by swapping pages between main memory and disk storage. Overall, paged segmentation plays a crucial role in optimizing memory utilization and enhancing the efficiency of modern operating systems.

SYSTEM ARCHITECTURE



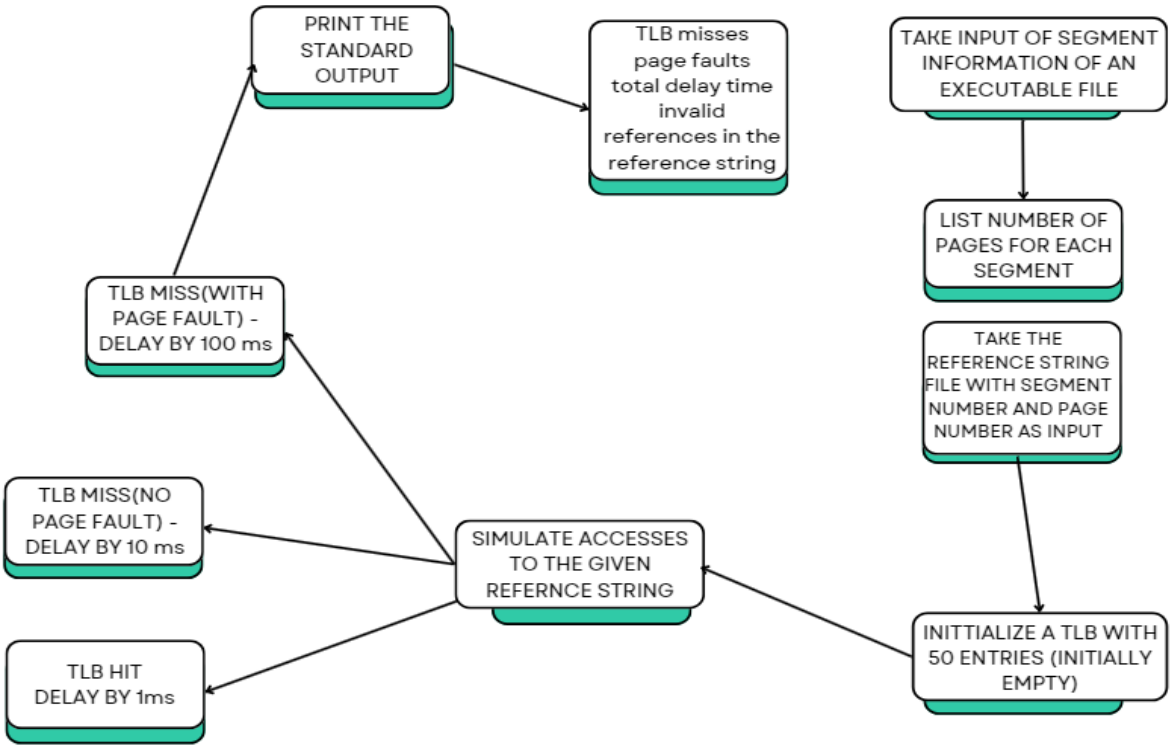
In a system architecture employing paged segmentation, the memory management unit (MMU) plays a central role in facilitating the translation of logical addresses generated by the CPU into physical addresses in the main memory. When a process running on the CPU generates a memory address, the MMU intercepts this address and consults the process's page table to determine the corresponding physical address. The page table, which is maintained by the operating system for each process, stores mappings between logical pages and physical pages, allowing for efficient address translation.

The page table typically resides in the main memory and is accessed by the MMU during address translation. Each entry in the page table contains information about the corresponding logical page, such as its location in the physical memory and its status (e.g.,

whether it is valid or not). When a process accesses a memory location, the MMU checks the page table to see if the corresponding page is present in the main memory. If the page is present, the MMU translates the logical address into the corresponding physical address, allowing the process to access the data stored in that page.

If the page referenced by the logical address is not currently in the main memory (a situation known as a page fault), the operating system intervenes to bring the required page into memory from secondary storage (e.g., disk). This process, known as paging, involves swapping out a page from main memory to make space for the new page. Once the required page is loaded into memory, the page table is updated accordingly, and the CPU can proceed with the memory access. Through this mechanism, paged segmentation enables efficient and dynamic memory management, allowing processes to utilize memory resources effectively while minimizing overhead.

METHODOLOGY



The report outlines a methodology for simulating Paging with Segmentation, a memory management technique used in operating systems. Initially, the program receives segment information from an executable file, detailing the sizes of various segments, each further divided into fixed-size pages. Following this, the program constructs a table listing the page counts for each segment based on the provided information.

Subsequently, the simulation involves accessing pages using a reference string file containing segment and page numbers. The program tracks TLB hits and misses, with a corresponding delay of 1 millisecond for hits and 10 milliseconds for misses without page faults. Page faults incur a 100-millisecond delay as required pages are fetched from virtual memory. Throughout the simulation, TLB and virtual memory management adhere to specified policies, with the TLB using FIFO and virtual memory employing the LRU algorithm.

The simulation provides insights into performance metrics such as TLB misses, page faults, and total delay time. Results are summarized at the end of the report, including the number of invalid references encountered during the simulation. This approach offers a comprehensive analysis of Paging with Segmentation, enhancing understanding of memory management strategies and their impact on system performance.

TOOLS AND API USED

I. System calls and POSIX functions

1. System:

   The system function is a standard C library function that allows a C program to execute shell commands. It takes a string as an argument, which represents the command to be executed, and then invokes the system's command interpreter (typically /bin/sh) to run the specified command.

   For example, system("ls -l") would execute the ls -l command in the shell, displaying a detailed listing of files in the current directory.

2. time_t:

   The time function is a POSIX function that returns the current calendar time as the number of seconds elapsed since the Unix epoch (00:00:00 UTC on January 1, 1970). It takes a single parameter of type time_t *, into which it stores the current time. This function is often used for timestamping, measuring time intervals, or scheduling tasks.

3. clock_t:

   clock_t is a data type defined in the C standard library header <time.h>. It represents clock ticks, which are units of CPU time consumed by a program or a specific section of code. The clock_t type is typically implemented as an arithmetic data type, such as long int or unsigned long int.

   It is primarily used for measuring the CPU time taken by a program or specific operations within a program. The clock function returns the current CPU time consumed by the program since the program started execution, measured in clock ticks.

4. rand and srand:

   The rand function is a standard C library function that generates pseudo-random integer numbers in the range of 0 to RAND_MAX. The srand function initializes the pseudo-random number generator with a seed value. If srand is not called before rand, the rand function is automatically seeded with a value of 1. By setting a specific seed with srand, you can produce a predictable sequence of random numbers, which is useful for testing and debugging.

5. floor:

   The floor function is a mathematical function defined in the math.h header file. It rounds a floating-point number down to the nearest integer that is less than or equal to the given value. It takes a single parameter of type double or float. The function returns a double value representing the largest integer less than or equal to the input value.

6. Ceil:

   The ceil function, also defined in the math.h header file, rounds a floating-point number up to the nearest integer that is greater than or equal to the given value. Similar to floor, it also takes a single parameter of type double or float. The function returns a double value representing the smallest integer greater than or equal to the input value.

II. File handling Functions:

1. fopen() and fclose():

   fopen function is used to open a file and obtain a file pointer for further operations. It takes two arguments: the filename and the mode in which the file should be opened (e.g., "r" for reading, "w" for writing, "a" for appending, etc.). Once the file operations are done, it's essential to close the file using the fclose function to release system resources and ensure data integrity.

   fopen() is used to open “segmentInfo.txt”.

2. fprintf() and fscanf():

   fprintf and fscanf are used for formatted input and output operations to files, similar to printf and scanf for standard input and output. fprintf writes formatted data to a file stream, while fscanf reads formatted data from a file stream. These functions are versatile and can handle various data types, including integers, floating-point numbers, characters, strings, etc. fprintf() is used to write generated reference data into ref.txt.

III. Standard C Libraries:

1. Stdio.h:

   This header file provides functions for standard input and output operations.

   Functions like printf, scanf, fopen, fclose, fread, fwrite, etc., are declared in stdio.h.

It is used for reading input from the console, writing output to the console, and performing file I/O operations.

2. Unistd.h:

   unistd.h provides access to various POSIX operating system API functions, including system calls, process control, and file operations. Functions like sleep, fork, exec, pipe, read, write, etc., are declared in unistd.h.

   It is used for tasks like pausing program execution (sleep), creating new processes (fork), executing programs (exec), and performing low-level I/O operations (read, write).

3. Stdlib.h

   stdlib.h contains functions for general-purpose utility functions, memory allocation, process control, and other miscellaneous tasks. Commonly used functions include malloc, free, atoi, rand, system, exit, etc. It is used for dynamic memory allocation (malloc, free), converting strings to numeric values (atoi, atof), generating random numbers (rand, srand), and terminating programs (exit).

4. String.h

   string.h provides functions for manipulating strings, such as copying, concatenating, comparing, and searching. Functions like strcpy, strcat, strlen, strcmp, strstr, etc., are declared in string.h. It is used for tasks like copying strings (strcpy), concatenating strings (strcat), finding the length of a string (strlen), and comparing strings (strcmp).

5. math.h

   math.h contains mathematical functions and constants for performing mathematical operations. Functions like sin, cos, sqrt, pow, ceil, floor, etc., are declared in math.h. It is used for tasks like trigonometric calculations (sin, cos), finding square roots (sqrt), exponentiation (pow), and rounding numbers (ceil, floor).

6. Time.h: time.h provides functions for working with date and time values, including retrieving the current time and formatting time values. Functions like time, ctime, strftime, difftime, etc., are declared in time.h. It is used for tasks like getting the current time (time), formatting time values as strings (ctime, strftime), and calculating time differences (difftime).

IV. Dynamic memory allocation:

1. Malloc():

   The malloc function in C is used to dynamically allocate memory during program execution.

   Syntax: void *malloc(size_t size);

   It takes the number of bytes to allocate as an argument (size) and returns a pointer to the allocated memory block. If the allocation is successful, malloc returns a pointer to the beginning of the allocated memory block. If it fails, it returns NULL. The allocated memory block is uninitialized, meaning its contents are indeterminate.

2. Free():

   The free function in C is used to deallocate memory that was previously allocated using malloc, calloc, or realloc.

   Syntax: void free(void *ptr);

   It takes a pointer to the memory block that needs to be deallocated as an argument (ptr). After calling free, the memory block pointed to by ptr is released back to the system, making it available for future allocations. It is essential to pass a valid pointer to free. Passing an invalid pointer or a pointer that has already been freed results in undefined behavior. Calling free does not change the value of the pointer itself; it merely deallocates the memory block it points to. Therefore, it is a good practice to set the pointer to NULL after calling free to avoid accessing deallocated memory accidentally.

V. Sleep Functionality

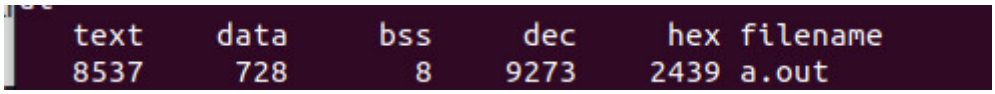
The delay function implements a delay mechanism, causing the program to pause execution for a specified duration. This is achieved by looping until the desired time has elapsed, simulating delays that may occur during real-world operations.

VI. Data Structures

The program employs various data structures such as arrays and structs to organize and manage data effectively. Arrays are utilized to represent Translation Lookaside Buffers (TLB), virtual memory, physical memory, and reference data. Meanwhile, structs encapsulate related data elements within a single unit, enhancing code readability and maintainability.

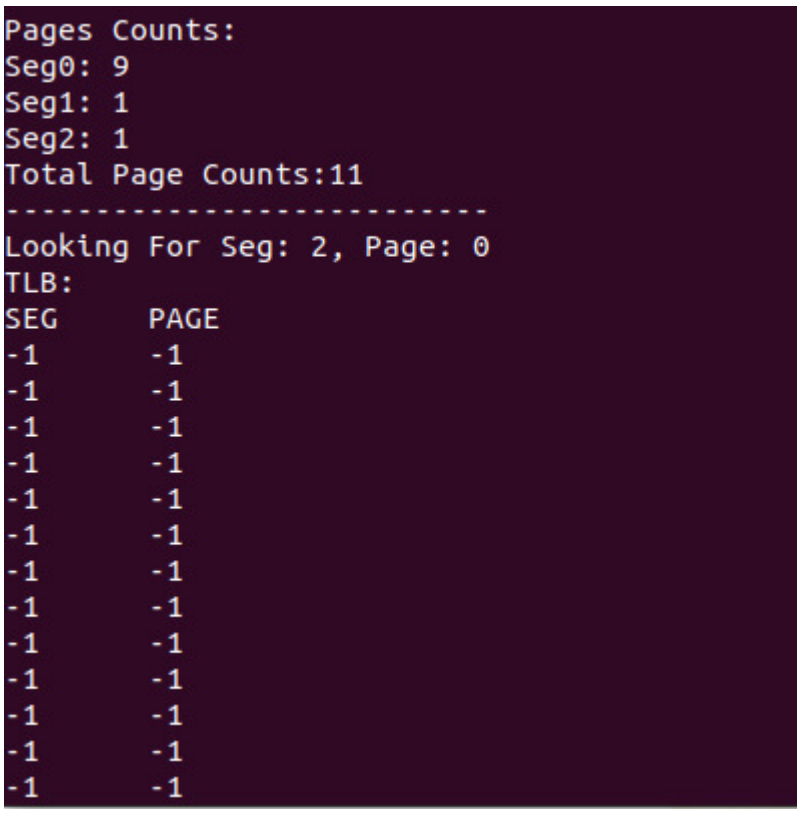
OUTPUTS AND RESULTS

1. The number and ratio of TLB misses.
2. The number and ratio of page faults.
3. The total delay time.
4. There may be invalid references in the reference string (accessing a non-existing segment or a non-existing page of an existing segment). Treat such references as if they don't exist but print the number of invalid references at the end.
5. The first entry of the segment table is in the physical memory initially, keeping all the other entries in the virtual memory. Whenever a particular page is accessed, it is brought into the physical memory. This ensures efficient memory usage.
6. Integrating paging and Segmentaion helps in eliminating any kind of fragmentation and hence memory wastage.



```
   text    data     bss     dec     hex filename
   8537     728       8    9273    2439 a.out
```

Parameters of the executable file



```
Pages Counts:
Seg0: 9
Seg1: 1
Seg2: 1
Total Page Counts:11
---------------------------
Looking For Seg: 2, Page: 0
TLB:
SEG     PAGE
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
-1      -1
```

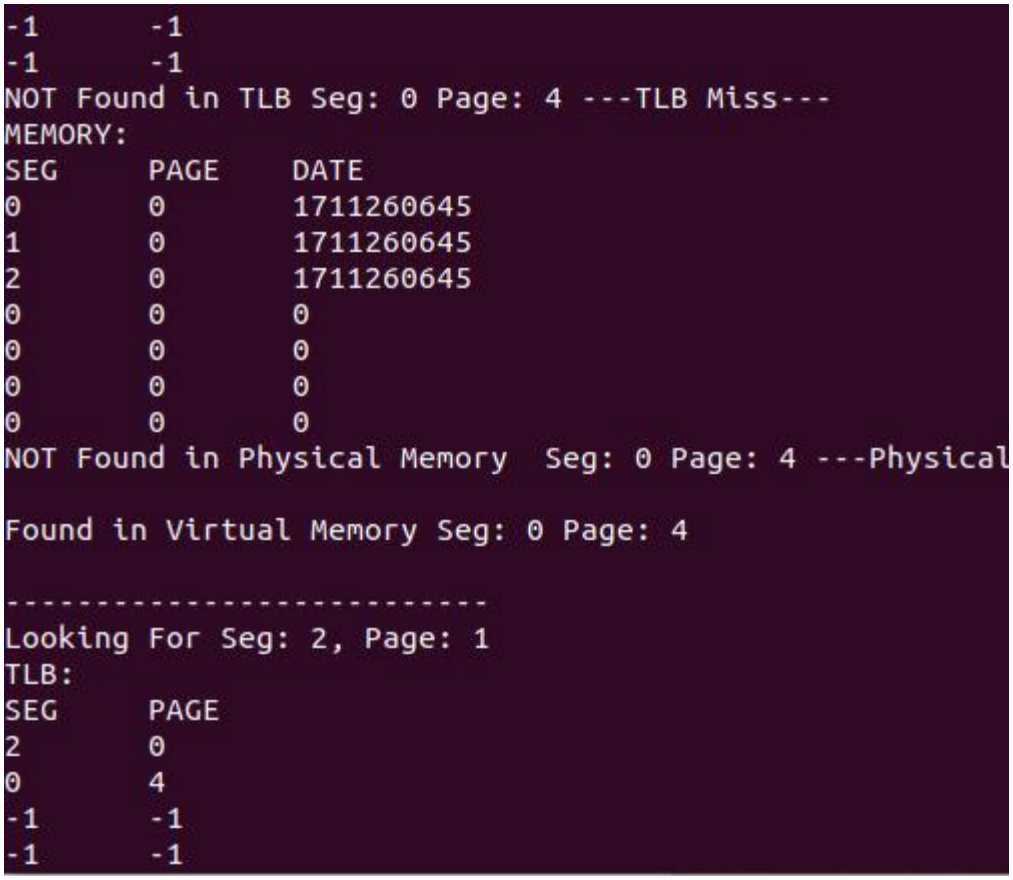
Page count of each segment and total page count

```
-1      -1
NOT Found in TLB Seg: 2 Page: 0 ---TLB Miss---
MEMORY:
SEG     PAGE    DATE
0       0       1711260645
1       0       1711260645
2       0       1711260645
Found in Physical Memory Seg: 2 Page: 0

----------------------------
Looking For Seg: 2, Page: 0
TLB:
SEG     PAGE
2       0
Found in TLB Seg: 2 Page: 0

----------------------------
Looking For Seg: 1, Page: 1
TLB:
SEG     PAGE
2       0
-1      -1
-1      -1
-1      -1
```

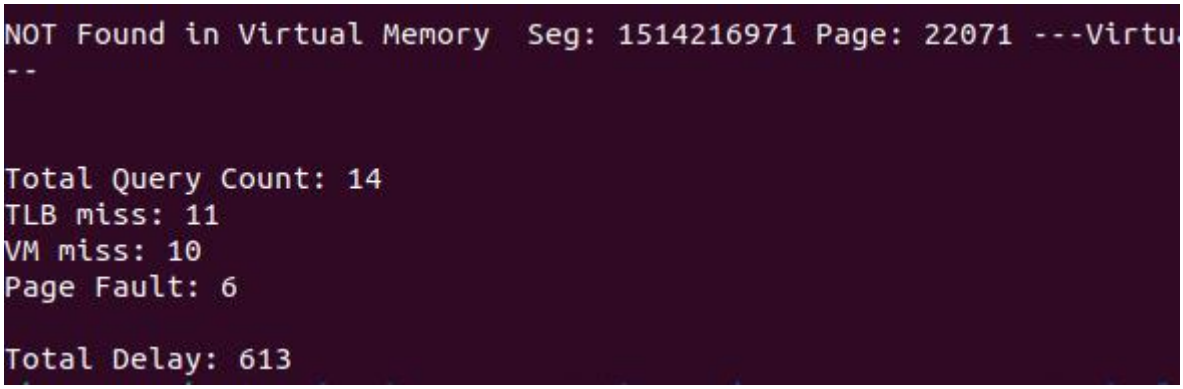
Accessing a page: Page found in the physical memory

```
-1      -1
-1      -1
-1      -1
-1      -1
NOT Found in TLB Seg: 1 Page: 1 ---TLB Miss---
MEMORY:
SEG     PAGE    DATE
0       0       1711260645
1       0       1711260645
2       0       1711260645
0       0       0
0       0       0
0       0       0
0       0       0
NOT Found in Physical Memory  Seg: 1 Page: 1 ---Physical

NOT Found in Virtual Memory  Seg: 1 Page: 1 ---Virtual M
----------------------------
Looking For Seg: 0, Page: 4
TLB:
SEG     PAGE
2       0
-1      -1
-1      -1
```

TLB miss



Not found in physical memory but found in Virtual memory



Query count, TLB miss count, VM miss , Page fault and Total delay

APPLICATIONS:

1. **Operating Systems:** Modern operating systems often use paging with segmentation to manage memory efficiently. This approach allows for better memory protection, virtual memory management, and process isolation.
2. **Virtualization:** Virtualization platforms use paging with segmentation to provide virtual machines (VMs) with isolated memory spaces. This allows multiple VMs to run concurrently on the same physical hardware, each with its own segmented memory space.
3. **Programming Languages:** Some high-level programming languages, such as Ada and Modula-3, use segmentation to organize program code and data into logical units. Paging is then used to manage memory allocation and provide virtual memory support.
4. **Graphics Processing:** Graphics processing applications often use segmentation to organize graphical objects and textures into logical units. Paging can then be used to efficiently manage memory allocation for rendering and texture mapping operations.
5. **Network Protocols:** Network protocols may use segmentation to divide data packets into logical units, with paging used to manage memory buffers efficiently. This allows for the efficient transmission and processing of network data.

CONCLUSION

Through our project, we extensively explored various memory management techniques, including paging, segmentation, single-level paging, multi-level paging, and segmented paging. We delved into the intricacies of each method, understanding their advantages, limitations, and practical applications in operating systems.

Paging, a memory management scheme, divides the physical memory into fixed-size blocks called pages, enabling efficient memory allocation and retrieval. Single-level and multi-level paging further optimize memory access by organizing pages hierarchically, reducing the overhead associated with large address spaces.

Segmentation, another memory management technique, divides the logical memory into variable-sized segments based on program structure, enhancing memory utilization and program flexibility. However, segmentation alone may lead to fragmentation issues and inefficient memory allocation.

To address the limitations of individual methods, we proposed a novel approach called paged segmentation. In paged segmentation, we combine the benefits of paging and segmentation, allowing for efficient memory management by dividing the logical memory into segments and further subdividing them into pages.

By implementing paged segmentation, we achieve the flexibility of segmentation while mitigating fragmentation concerns through page-based memory allocation. This hybrid approach optimizes memory utilization, enhances address space management, and improves system performance in scenarios with diverse memory requirements.

Throughout our project, we conducted comprehensive experiments, simulations, and analyses to evaluate the effectiveness and efficiency of paged segmentation compared to traditional memory management techniques. Our findings demonstrate that paged segmentation offers a balanced solution, providing the benefits of both paging and segmentation while minimizing their drawbacks. The insights gained from this project pave the way for our learning into the field of operating system