CS 3031

OPERATING SYSTEMS LAB

VIRTUAL MEMORY MANAGER

AMAN VERMA (C11B002)

BHUPESH RAJ S (CS11B006)

RITVIK JAISWAL (CS11B031)

INTRODUCTION

The project was to implement a virtual memory manager to simulate TLB and Page Tables and mapping of logical addresses to physical addresses.

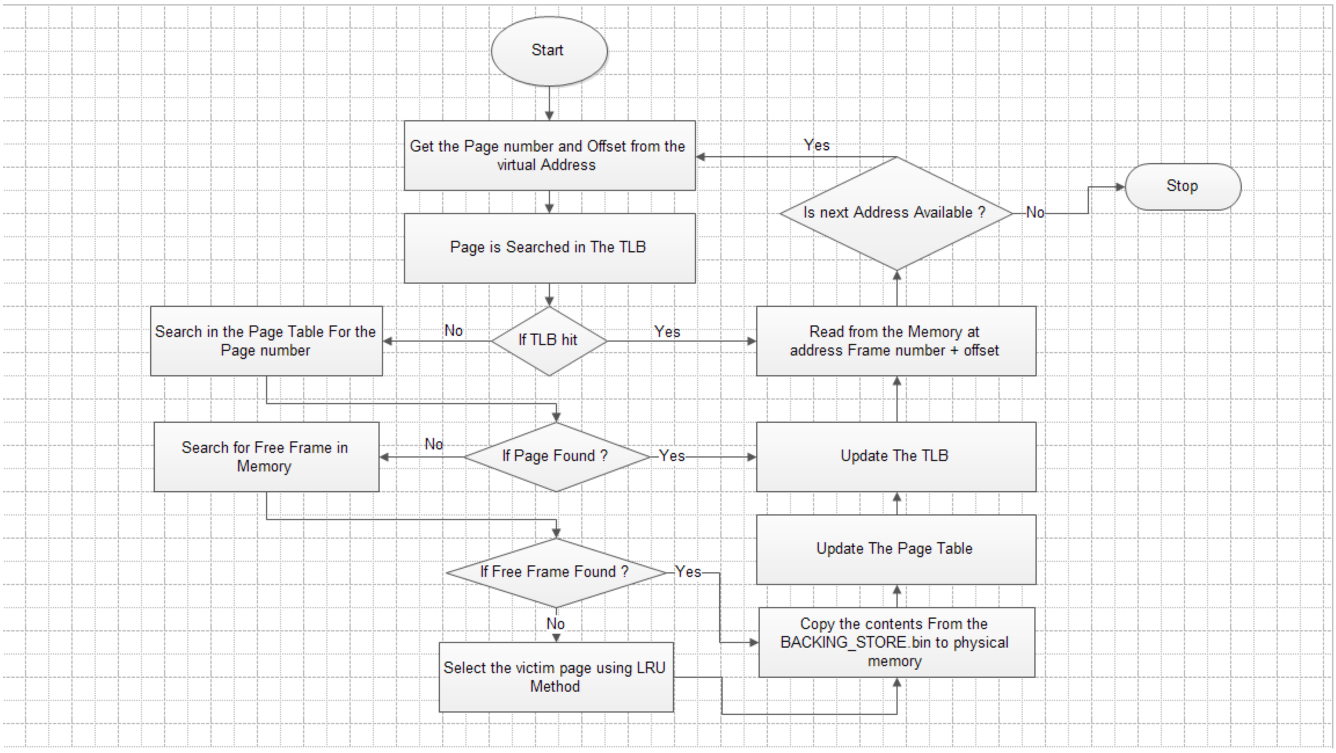
The simulation code was written in C Language.

AVAILABLE RESOURCES

We were provided with the following files –

Addresses.txt	- contains the list of virtual addresses
Correct.txt	- contains the correct output
BACKING_STORE.bin	- simulates pages stored in secondary storage

FLOWCHART


Start
Get the Page number and Offset from the virtual Address
Yes
Is next Address Available ?
No
Stop
Page is Searched in The TLB
Search in the Page Table For the Page number
No
If TLB hit
Yes
Read from the Memory at address Frame number + offset
Search for Free Frame in Memory
No
If Page Found ?
Yes
Update The TLB
Update The Page Table
If Free Frame Found ?
Yes
No
Copy the contents From the BACKING_STORE.bin to physical memory
Select the victim page using LRU Method

DESIGN

One structure each for the TLB and Page Table Entries was defined.

- TLB structure named T_L_B containing
  - Page_number - stores the virtual page number of the process
  - Frame_number - stores the physical page frame number of the DRAM
  - Valid_bit - +1 if corresponding entry is valid, 0 otherwise
- Page table entries structure named Page_Table_Entry, containing
  - Frame_number - stores the frame number allotted to the virtual page
  - Valid_bit - +1 if the page is in DRAM, 0 otherwise
  - Count - keeps track of when the page was last used

The following macros were defined –

- TLB_SIZE - Defines the size of the TLB
- PAGE_TABLE_SIZE - Number of Page Table Entries
- NO_OF_FRAMES - Number of Frames in DRAM
- FRAME_SIZE - Size of each frame (Typically 256 bytes)

The following arrays were declared –

- TLB - Array of type T_L_B and size TLB_SIZE
- Page_Table - array of type Page_Table_Entry and size PAGE_TABLE_SIZE
- Physical_Memory - 2D character array which simulates the DRAM.
- Free_Frame - integer array with values 0 if free, 1 if occupied
- TLB_head - integer which holds the index of TLB entry treated as head

The following functions were implemented –

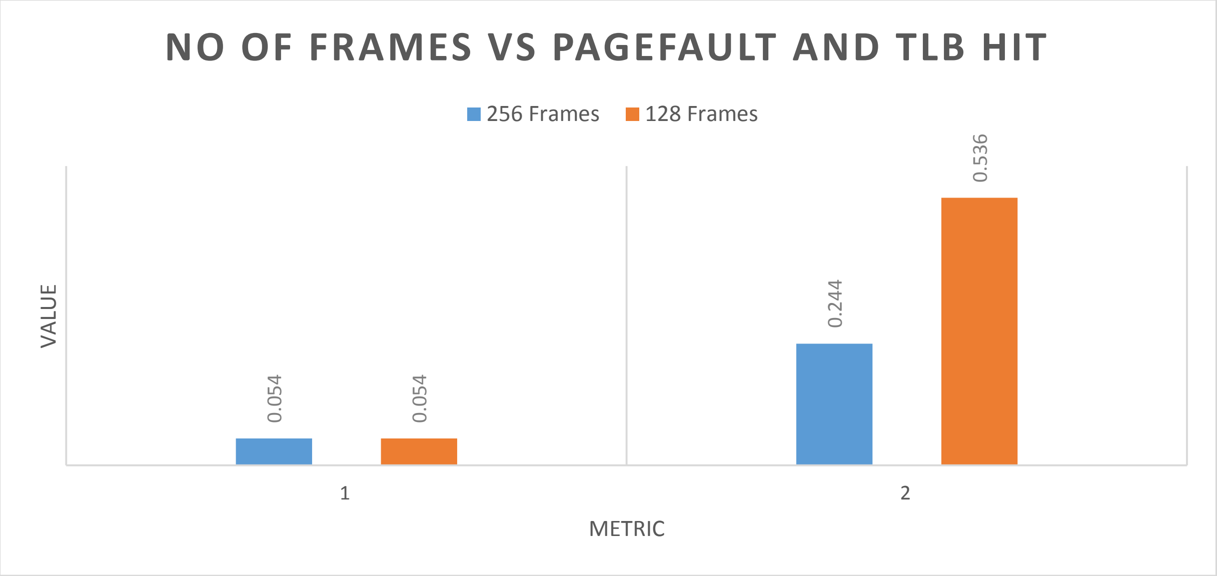
- void initialization()
  - Initializes data members of the two structures.
- void print_TLB (FILE *fp, int count)
  - Takes as arguments a file pointer to **steps.txt** and the number of iterations completed.
  - Prints the current state of the TLB, redirected to file **steps.txt**.
- void print_Page_Table (FILE *fp, int count)
  - Takes as arguments a file pointer to **steps.txt** and the number of iteration completed.
  - Prints the current state of the Page Table, redirected to file **steps.txt**.
- int TLB_Search (int page_number)
  - Linear search is performed to search the TLB.
  - The frame number of the requested page is returned on a TLB hit, -1 is returned on a TLB miss.
  - TLB hit occurs if the requested page number is found in the TLB and the corresponding valid bit is 1.
- int TLB_index_Search (int page_number)
  - This function returns the index of the TLB on a TLB hit, and returns -1 otherwise.

- int Page_Table_Search (int page_number)
  - This function indexes into the page table and returns the frame number of the requested page if the corresponding valid bit is 1, and returns -1 otherwise.
- int search_for_free_slot_in_physical_memory()
  - Linear search is performed on the array Free_Frame.
  - The index of the array is returned if corresponding value is 1 (i.e. the page frame is free).
  - If no free frame is found, the function returns -1.
- int select_victim_page()
  - LRU replacement policy is chosen to select a victim page from the DRAM.
  - Since all pages of the DRAM are valid in the page table, the page table is searched for entries with valid bits set to 1.
  - Of these entries, the one with the highest value of count is selected as victim.
  - The corresponding index of the page table is returned.
- int Update_Count (int page_number)
  - This function is called each time a page is requested, i.e. a virtual page id from **addresses.txt** is read.
  - This function sets the count variable of the requested page to 0 and increments the count of all other pages by 1.
  - This ensures that the most frequently used page has the least value of count, which is zero, and the least frequently used page has the highest value of count.
- int Update_TLB (int page_number, int frame_number)
  - This function is called in case of a TLB miss.
  - The TLB is searched for any invalid entries to replace them.
  - If no invalid entry is found, then FIFO algorithm is implemented to choose which entry in the TLB will be updated.
- int Update_Page_Table (int page_number, int frame_number)
  - This function is called when a page fault occurs.
  - The valid bit of the page table entry corresponding to the requested page is set to 1, and the frame number is assigned.
- int Update_Free_Frame (int frame_number)
  - This function is called when a free page frame is assigned to the process.
  - The corresponding entry in the array Free_Frame is reset to 0.
- int Read_from_Memory (int number, int frame_number, int offset, FILE *output_file)
  - Given frame number and offset this function reads from the physical memory and writes them and the physical address and the value stored at that address in the output file.
- int Copy_from_disk_to_memory (int frame_number, int page_number)
  - This function is used to copy the data stored in BACKING_STORE.bin stored at given page number into the physical memory at the corresponding frame number
- int main()
  - In this function, virtual addresses are read from **addresses.txt**.
  - These addresses are first searched in the TLB using the TLB_Search() function.
    - In case of TLB hit, the page is directy read from memory using the Read_from_Memory() function.
    - In case of TLB miss, the page table is checked using the Page_Table_Search() function.

- In case of a valid entry, the TLB is updated using Update_TLB(), **count** variable of the page table entries is updated using Update_Count() and the page is read from memory using Read_from_Memory().
- In case of an invalid entry, we look for a free frame using search_for_free_slot_in_physical_memory().
  - In case a free frame is found, we copy the page into the free frame using Copy_from_disk_to_memory().
    - Update_TLB(), Update_Page_Table() and Update_Count() functions are called to update the state of the simulation.
  - In case a free frame is not found, we select a victim page using select_victim_page() and the above functions are called.
- This is repeated until all the addresses from **addresses.txt** have been read.

RESULT

Reference strings are obtained from **Addresses.txt**.



1 Represents TLB Hit Rate

2 Represents Page Fault Rate.