**Batch: C3 Roll No.: 16010123217**

**Experiment / assignment / tutorial No. 6**

6

|  |
| --- |
| **TITLE: Implementation of LRU Page Replacement Algorithm.** |

**AIM:** The LRU algorithm replaces the least recently used that is the last accessed memory block from user.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO/COs attained here)**

CO 3 Learn and evaluate memory organization and cache structure

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, TataMcGraw-Hill.
2. William Stallings, “Computer Organization and Architecture: Designing for Performance”, Eighth Edition, Pearson.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

**Algorithm:**

1. Start
2. Get input as memory block to be added to cache
3. Consider an element of the array
4. If cache is not full, add element to the cache array
5. If cache is full, check if element is already present
6. If it is hit is incremented
7. If not, element is added to cache removing least recently used element
8. Repeat step 3 to 7 for remaining elements
9. Display the cache at very instance of step 8
10. Print hit ratio
11. End

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX\_SIZE 1000

int\* Main;

int\* order;

int\* list;

*// Function to check if a page is in the memory (Main)*

bool isPageInMemory(int page, int size) {

    for (int i = 0; i < size; i++) {

        if (Main[i] == page) {

            return true;

        }

    }

    return false;

}

*// Function to find the index of the least recently used page*

int findLRUIndex(int size) {

    int min = 0;

    for (int i = 1; i < size; i++) {

        if (order[i] < order[min]) {

            min = i;

        }

    }

    return min;

}

int main(void) {

    int n, a;

*// Input size of Main*

    printf("Enter the size of Main arr: ");

    scanf("%d", &n);

    Main = (int\*)malloc(n \* sizeof(int));

    order = (int\*)malloc(n \* sizeof(int));

*// Initialize Main with -1 (indicating empty)*

    for (int i = 0; i < n; i++) {

        Main[i] = -1;

        order[i] = 0;

    }

*// Input number of elements in list*

    printf("Enter the number of elements in List: ");

    scanf("%d", &a);

    list = (int\*)malloc(a \* sizeof(int));

    printf("Enter the elements of the list: ");

    for (int i = 0; i < a; i++) {

        scanf("%d", &list[i]);

    }

    int hits = 0;

    int pageFaults = 0;

    for (int i = 0; i < a; i++) {

        int page = list[i];

        if (isPageInMemory(page, n)) {

*// It's a hit*

            hits++;

*// Update the order of usage*

            for (int j = 0; j < n; j++) {

                if (Main[j] == page) {

                    order[j] = i;

                    break;

                }

            }

        } else {

*// It's a page fault*

            pageFaults++;

            int lruIndex = findLRUIndex(n);

            Main[lruIndex] = page;

            order[lruIndex] = i;

        }

    }

    printf("Hits: %d\n",hits);

    float hitRatio = (float)hits / (a-1);

    printf("Hit Ratio: %.2f\n", hitRatio);

*// Display the contents of Main*

    printf("Contents of Main array:\n");

        printf("%d ",Main[n-1]);

    for (int i = 0; i < n-1; i++) {

        printf("%d ", Main[i]);

    }

    printf("\n");

*// Clean up*

    free(Main);

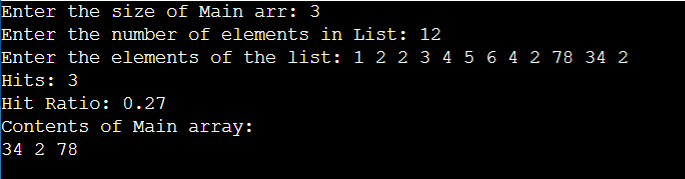
    free(order);

    free(list);

    return 0;

}

**Output:**

****

**Post Lab Descriptive Questions**

**1. Define hit rate and miss ratio?**

1. **Hit rate: -** The hit rate is the fraction of cache accesses that result in a cache hit. When a requested item is found in the cache, it’s a cache hit. The hit rate measures how often this happens. A higher hit rate indicates better cache performance.

Formula: Hit Rate = (Number of Cache Hits) / Total Number of Cache Accesses

1. Miss Ratio:- The miss ratio is the fraction of cache accesses that result in a cache miss. When a requested item is not found in the cache and must be fetched from a slower memory or storage, it’s a cache miss. The miss ratio is the complement of the hit rate, so it can also be calculated as Miss Ratio=1−Hit Rate or

Miss Ratio = (Number of Cache Misses )/(Total Number of Cache Accesses).

**2. What is the need for virtual memory**?

Virtual memory is needed for:

1. Larger Address Space: Allows programs to use more memory than physically available by providing an illusion of a large, contiguous block of memory.

2. Process Isolation: Ensures that each process has its own address space, improving security and stability.

3.Efficient Memory Use: Manages physical memory more effectively by swapping data between RAM and disk storage.

4. Multitasking: Supports running multiple processes concurrently without interference.

5. Simplified Programming: Developers can write programs without worrying about physical memory limitations.

**Conclusion:**

The LRU page-replacement algorithm has been implemented correctly with proper management of the memory blocks by replacing entries not recently accessed, and a case wherein all of the cache blocks are full. This ensures better utilization of memory and boosts the cache for longer hit rates. This overall experiment demonstrates how the strengthening of virtual memory and the management techniques increase the efficiency and stability of the system.

**Date: 11/7/2024**