Batch: C-2	**Roll No.: 16010122267**
Experiment / assignment / tutorial No: 7	

TITLE : Implementation of FIFO Page Replacement Algorithm

AIM: The FIFO algorithm uses the principle that the block in the set which has been in for the longest time will be replaced

Expected OUTCOME of Experiment: (Mention CO/CO's attained here)

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.

3. Dr. M. Usha, T. S. Srikanth, "Computer System Architecture and Organization", First Edition, Wiley-India.

Pre Lab/ Prior Concepts:

T he FIFO algorithm uses the principle that the block in the set which has been in the block for the longest time is replaced. FIFO is easily implemented as a round robin or criteria buffer technique. The data structure used for implementation is a queue. Assume that the number of cache pages is three. Let the request to this cache is shown alongside.

Algorithm:

1. A hit is said to be occurred when a memory location requested is already in the cache.
2. When cache is not full, the number of blocks is added.
3. When cache is full, the block is replaced which was added first

Design Steps:

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 first element (which is in first).
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.

Example:

0	2	1	6	4	0	1	0	3	1	2	1
0	0	0	0	4	4			4	4	2	
	2	2	2	2	0		hit	0	0	0	
		1	1	1	1	hit		3	3	3	
			6	6	6			6	1	1	hit

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    std::vector<int> cache;
    const int maxSize = 4;
    int index = 0;
    int cacheHits = 0;
    int cacheMisses = 0;
    int* mostRecent = nullptr; // most recently added element

    auto displayCache = [&cache, &mostRecent]() {
        cout << "Cache: " << endl;
        for (const int& value : cache) {
            if (&value == mostRecent) {
                cout <<  value << "*";

            } else {
                cout << value << " ";
            }
            cout << endl;
        }
        cout << endl;
    };
```

```
    auto addToCache = [&cache, maxSize, &index, &cacheHits, &cacheMisses,
&mostRecent](int element) {
        auto it = std::find(cache.begin(), cache.end(), element);
        if (it != cache.end()) {
            // hit
            cout << element << " is already in the cache." << endl;
            cacheHits++;
        } else {
            // miss
            cacheMisses++;
            if (cache.size() < maxSize) {
                cache.push_back(element);

            } else {
                mostRecent = &cache[index];
                cache[index] = element;
                index = (index + 1) % maxSize; //cyclic indexing
            }
            cout << element << " added to the cache." << endl;
        }
    };

    int choice;
    while (true) {
        cout << "Menu:" << endl;
        cout << "1. Add element to cache" << endl;
        cout << "2. Display cache" << endl;
        cout << "3. Show Hit Rate and Hit Ratio" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                int element;
                cout << "Enter an element to add to the cache: ";
                cin >> element;
                addToCache(element);
                break;
            case 2:
                displayCache();
                break;
            case 3:
```

```
                if (cacheHits + cacheMisses > 0) {
                    double hitRate = static_cast<double>(cacheHits) /
(cacheHits + cacheMisses);
                    double hitRatio = static_cast<double>(cacheHits) /
cacheMisses;
                    cout << "Hit Rate: " << hitRate << endl;
                    cout << "Hit Ratio: " << hitRatio << endl;
                } else {
                    cout << "No cache accesses yet." << endl;
                }
                break;
            case 4:
                return 0;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    }

    return 0;
}
```

Output:

Let's consider reference string

7 0 1 2 0 3 0 4 2 3 0 3 1 2 0

Added elements: 7, 0, 1

```
Menu:
1. Add element to cache
2. Display cache
3. Show Hit Rate and Hit Ratio
4. Exit
Enter your choice: 2
Cache:
7
0
1
```

```
2 added to the cache.
```

```
Cache:
2*
0
1
```

```
Enter an element to add to the cache: 0
0 is already in the cache.
```

```
Enter an element to add to the cache: 3
3 added to the cache.
```

```
Cache:
2
3*
1
```

```
Enter an element to add to the cache: 0
0 added to the cache.
```

```
Cache:
2
3
0*
```

```
Enter an element to add to the cache: 4
4 added to the cache.
```

```
Cache:
4*
3
0
```

```
Enter an element to add to the cache: 2
2 added to the cache.
```

```
Cache:
4
2*
0
```

```
Enter an element to add to the cache: 3
3 added to the cache.
```

```
Cache:
4
2
3*
```

```
Enter an element to add to the cache: 0
0 added to the cache.
```

```
Cache:
0*
2
3
```

```
Enter an element to add to the cache: 3
3 is already in the cache.
```

```
Enter an element to add to the cache: 1
1 added to the cache.
```

```
Cache:
0
1*
3
```

```
Enter an element to add to the cache: 2
2 added to the cache.
```

```
Cache:
0
1
2*
```

```
Enter an element to add to the cache: 0
0 is already in the cache.
```

Final Page Frame:

```
Cache:
0
1
2*
```

Hit Rate and Hit Ratio:

```
Menu:
1. Add element to cache
2. Display cache
3. Show Hit Rate and Hit Ratio
4. Exit
Enter your choice: 3
Hit Rate: 0.2
Hit Ratio: 0.25
```

Post Lab Descriptive Questions

1. What is meant by memory interleaving?

Ans:

- Memory interleaving is a technique used to enhance memory access performance in computer systems.
- It divides the memory address space into multiple interleaved banks or modules.
- Memory requests are distributed across these banks, enabling parallel memory operations.
- Reduces contention for memory access and mitigates bottlenecks in high-memory bandwidth systems.
- Various interleaving methods exist, such as byte, word, and block interleaving.
- Proper configuration is crucial to match the system's requirements and architecture.
- Commonly used in servers, workstations, and other systems with high memory bandwidth demands.

3. Explain Paging Concept?

Ans:

Paging is a memory management technique used in computer operating systems to efficiently manage and allocate physical memory (RAM) to processes running on a computer. It provides a way to abstract and isolate the physical memory from the logical memory used by programs, making the best use of available resources and simplifying memory management. Here's an explanation of paging:

- **Memory Abstraction**: Paging divides memory into fixed-size blocks called "pages" for both logical (program) and physical (RAM) memory.
- **Page Table**: Each process has a page table to map logical addresses to physical addresses.
- **Benefits**: Paging provides isolation, flexibility, and reduces memory

fragmentation.

- **Page Faults**: Occur when a program accesses a page not in physical memory, prompting retrieval from secondary storage.
- **Page Replacement**: When physical memory is full, the OS selects pages for replacement to minimize performance impact.
- **Demand Paging**: Pages are loaded into memory only when needed, conserving resources.
- **Sharing and Copy-on-Write**: Allows memory sharing among processes and optimizes memory use during process forking.

Conclusion:

In conclusion, the FIFO algorithm effectively replaces the oldest block in a set, demonstrating its simplicity and fairness in managing cache memory, but it may not always provide optimal performance in scenarios where more intelligent caching strategies are required.

Date:10/10/23