Batch: B2	Roll No.: 16010124107
Experiment / assignment / tutorial No. 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/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.

Pre Lab/ Prior Concepts:

Algorithm:

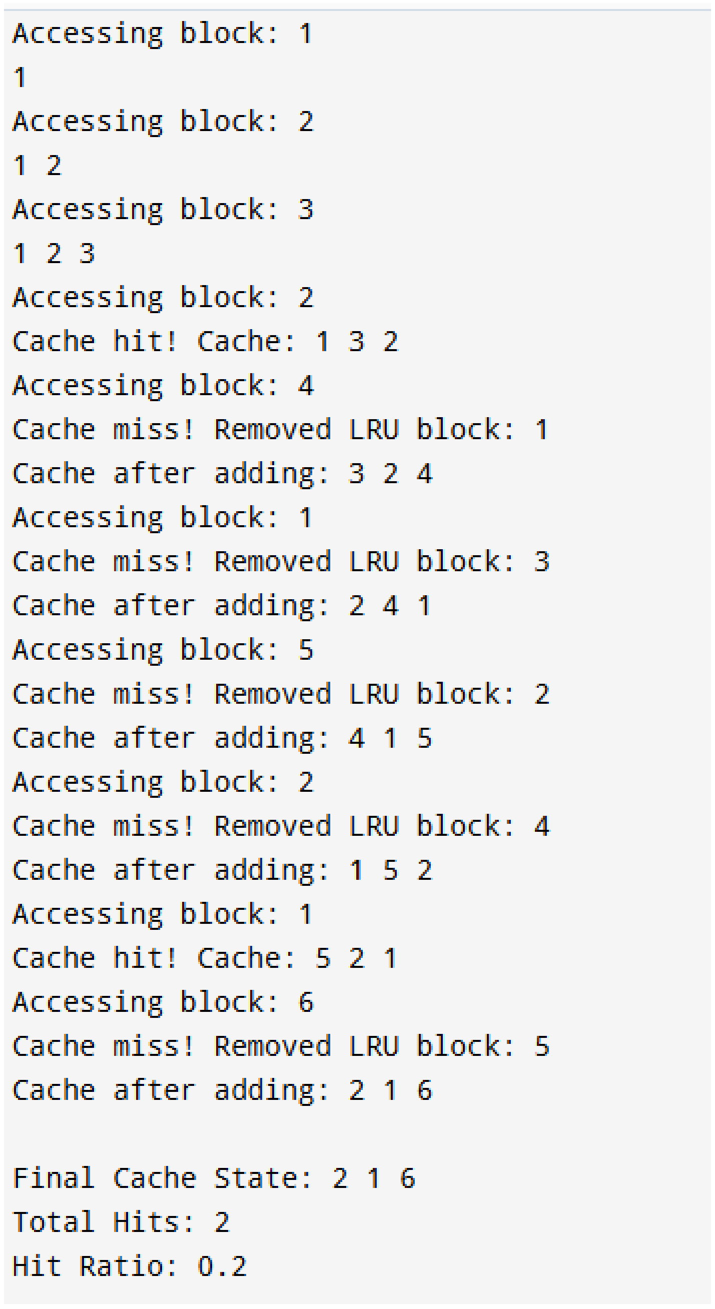
1. Start
2. Get input as memory block to be added to cache
3. Consider an element of the array
4. If cache is not full, add element to the cache array
5. If cache is full, check if element is already present
6. If it is hit is incremented
7. If not, element is added to cache removing least recently used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

Code:

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

```
using namespace std;

void lru_cache_simulation(const vector<int>& memory_blocks, int cache_capacity) {
    vector<int> cache;
    int hits = 0;
    int total_requests = memory_blocks.size();

    for (int block : memory_blocks) {
        cout << "Accessing block: " << block << endl;

        auto it = find(cache.begin(), cache.end(), block);

        if (it != cache.end()) {
            hits++;
            cache.erase(it);
            cache.push_back(block);
            cout << "Cache hit! Cache: ";
        } else {
            if ((int)cache.size() < cache_capacity) {
                cache.push_back(block);
            } else {
                int removed = cache.front();
                cache.erase(cache.begin());
                cache.push_back(block);
                cout << "Cache miss! Removed LRU block: " << removed << endl;
                cout << "Cache after adding: ";
            }
        }

        for (int c : cache) cout << c << " ";
        cout << endl;
    }

    double hit_ratio = (double)hits / total_requests;
    cout << "\nFinal Cache State: ";
    for (int c : cache) cout << c << " ";
    cout << "\nTotal Hits: " << hits << endl;
    cout << "Hit Ratio: " << hit_ratio << endl;
}

int main() {
    vector<int> memory_blocks = {1, 2, 3, 2, 4, 1, 5, 2, 1, 6};
    int cache_capacity = 3;
    lru_cache_simulation(memory_blocks, cache_capacity);
    return 0;
}
```

OUTPUT:



```
Accessing block: 1
1
Accessing block: 2
1 2
Accessing block: 3
1 2 3
Accessing block: 2
Cache hit! Cache: 1 3 2
Accessing block: 4
Cache miss! Removed LRU block: 1
Cache after adding: 3 2 4
Accessing block: 1
Cache miss! Removed LRU block: 3
Cache after adding: 2 4 1
Accessing block: 5
Cache miss! Removed LRU block: 2
Cache after adding: 4 1 5
Accessing block: 2
Cache miss! Removed LRU block: 4
Cache after adding: 1 5 2
Accessing block: 1
Cache hit! Cache: 5 2 1
Accessing block: 6
Cache miss! Removed LRU block: 5
Cache after adding: 2 1 6

Final Cache State: 2 1 6
Total Hits: 2
Hit Ratio: 0.2
```

Post Lab Descriptive Questions

1. Define hit rate and miss ratio?
Answer:

Hit Rate:
- The hit rate is the fraction of memory accesses that are found in the cache.

- It indicates the working efficiency of the cache
- A higher hit rate means better performance and fewer accesses to main memory.

Miss Ratio:

- The miss ratio is the fraction of memory accesses not found in the cache.
- It shows how often the system must fetch data from main memory.
- A lower miss ratio means the cache is performing efficiently.

2. What is the need for virtual memory?
Answer:

Virtual memory is needed to efficiently use main memory and allow execution of large programs that may not completely fit into physical RAM.

Main purposes:

1. Enables large programs to run even if they are bigger than the available physical memory.
2. Provides process isolation and protection.
3. Improves system performance.
4. Allows multitasking.
5. Simplifies memory management.

Conclusion:

The implementation of the LRU page replacement algorithm effectively manages memory by replacing the least recently used pages. It optimizes cache utilization, improving hit ratios and reducing page faults. This experiment demonstrates how LRU enhances system performance by maintaining relevant data in limited cache space efficiently.

Date: 10/10/2025