Experiment No. 8

Aim : To implement the least Recently used (LRU) page Replacement Algorithm.

Objective : To study & implement the least Recently used (LRU) page replacement algorithm and evaluate it's efficieny in terms of page faults.

Aim : To implement the least Recently Used (LRU) Page Replacement Algorithm.

Objective : To study & implement the least Recently used (LRU) page replacement algorithm & evaluate it's efficiency in terms of page faults.

About Page Replacement in OS: When a process executes & requests a page not present in main memory (a page fault), the operating system must load the requested page into memory. If the memory is full, a page replacement algorithm decides which page to remove

LRU / Least Recently Used Page ~~Alg~~ replacement Algorithm :

LRU replaces the page that has not been used for the longest time.

It works on the principle of temporary locality - recently accessed pages are more likely to be accessed again.

LRU is more efficient than FIFO in many cases because it considers the usage history of pages.

Algorithm : i) Initialize an empty page frame list.

ii) Traverse the page reference string one by one :

- If the page is in memory (a hit), move it to the most recently used position.
- if the page is NOT in the memory (a Pge Page fault) :
  - If the memory has space, add the page.
  - if the memory is full, remove the latest used page & add a new page.

iii) Keep the track of the total number of page faults.

Key Terms : i) Page frame : Fixed size memory block in the main memory.

ii) Page fault : Occurs when the requested page is not in the memory.

iii) Page Hit : ~~Hit~~ Requested page is already in memory.

Code implementation in C++ :

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <list>
using namespace std;
```

```
void LRU (int pages [], int n, int capacity) {
  unordered map <int, list<int> :: iterator > pagemap;
  list <int> pageList ;
  int pageFaults = 0;

  for (int i = 0; i<n; i++) {
      int page = pages [i];

  if (pageMap.find (page) != pageMap.end ()) {
        pageList.erase (pageMap [page]);
  } else {
        if (pageList.size () == capacity) {
            int lru = pageList.back ();
            pageList.pop_back ();
            pageMap.erase (lru);
        }

} pagefaults ++;
}

  pageList.push_front (page);
 } pageMap [page] = pageList.begin ();

  cout << "Total page faults : " << pageFaults << endl;
}

  int main () {
     int pages [] = {7,0,1,2,0,3,0,4,2,3,0,3,
                     2};
```

Conclusion: The LRU page replacement algorithm helps in minimizing page faults by tracking usage history It is a realistic & widely used strategy, especially in systems with frequent memory access.

22/4/25

```
    int n = sizeof(pages) / sizeof(pages[0]);
    int capacity;

    cout << "Enter the number of Page frames: ";
    cin >> capacity;

    LRU(pages, n, capacity);

    return 0;
}
```

Execution & output:

```
Enter the number of Page frames : 4
Total Page faults = 9
```

Conclusion: The LRU page replacement Algorithm helps in minimizing page faults by tracking usage history It is a realistic & widely used strategy, especially in systems with frequent memory access.

22/4/25