Batch: A1	**Roll No.: 16010123012**
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:

It follows a simple logic, while replacing it will replace that page which has least recently used out of all.

a) A hit is said to be occurred when a memory location requested is already in the cache.

b) When cache is not full, the number of blocks is added.

c) When cache is full, the block is replaced which is recently used

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 <iostream>
using namespace std;

int hit = 0, access = 0;

void insert(int x, int* cache, int* age, int cacheSize) {
   access++;
   int i = 0;
   while (i < cacheSize && x != cache[i]) {
      i++;
   }

   if (i != cacheSize) {
      hit++;
      age[i] = 1;
      for (int j = 0; j < cacheSize; j++) {
         if (j != i) age[j]++;
      }
   } else {
      int old = 0;
      for (int j = 1; j < cacheSize; j++) {
         if (age[j] > age[old]) {
            old = j;
         }
      }
      cache[old] = x;
      age[old] = 1;
      for (int j = 0; j < cacheSize; j++) {
         if (j != old) age[j]++;
      }
   }

   for (int j = 0; j < cacheSize; j++) {
      cout << cache[j] << " ";
   }
   cout << endl;
}
```

```
void hitratio(int access, int hit) {
    double hitr = (access > 0) ? (double)hit / access : 0.0;
    cout << "Hit Ratio: " << hitr << endl;
}

int main() {
    int cacheSize;
    cout << "Enter the desired cache size: ";
    cin >> cacheSize;

    int* cache = new int[cacheSize];
    int* age = new int[cacheSize];

    cout << "Enter the first " << cacheSize << " cache elements: " << endl;
    for (int i = 0; i < cacheSize; i++) {
        cin >> cache[i];
        age[i] = i + 1;
    }

    int inpt = 0;
    cout << "Enter numbers to add to cache (enter -1 to stop): " << endl;
    while (inpt != -1) {
        cin >> inpt;
        if (inpt != -1) {
            insert(inpt, cache, age, cacheSize);
        }
    }

    hitratio(access, hit);

    delete[] cache;
    delete[] age;
    return 0;
}
```

Output:

```
Enter the desired cache size: 3
Enter the first 3 cache elements:
1
2
3
Enter numbers to add to cache (enter -1 to stop):
2
1 2 3
4
1 2 4
6
6 2 4
5
6 5 4
4
6 5 4
2
2 5 4
1
2 1 4
0
2 1 0
8
8 1 0
3
8 3 0
-1
Hit Ratio: 0.2
```

Example:

Input Number	Cache State	Age State	Action	Hit Count	Access Count
Initial	5 6 7	1 2 3	-	0	0
5	5 6 7	1 3 4	Hit (5 already in cache)	1	1
8	5 6 8	1 2 1	Miss (replace 7)	1	2
6	5 6 8	2 1 2	Hit (6 already in cache)	2	3
9	9 6 8	1 2 3	Miss (replace 5)	2	4
5	9 6 5	2 3 1	Miss (replace 8)	2	5
8	8 6 5	1 2 2	Miss (replace 9)	2	6
-1	-	-	End of Input	2	6

Post Lab Descriptive Questions

1. Define hit rate and miss ratio?

Hit rate and miss ratio are terms commonly used in the context of cache memory systems, especially in computer architecture and computer systems. They help evaluate the effectiveness of a cache in reducing memory access latency and improving overall system performance.

Hit Rate:

- The hit rate, also known as the cache hit rate, is a metric that measures the percentage of memory accesses that are successfully serviced by the cache without having to access the main memory or a lower-level cache.
- A high hit rate indicates that the cache is doing a good job of storing frequently used data and reducing the number of accesses to slower memory levels (such as RAM or disk).
- Hit Rate (%) = (Number of Cache Hits / Total Number of Memory Accesses) * 100

Miss Ratio:

- The miss ratio, also known as the cache miss ratio or cache miss rate, is a metric that measures the percentage of memory accesses that result in cache misses. In other words, it represents the proportion of memory accesses that cannot be serviced by the cache and require accessing a slower memory level.
- A lower miss ratio is desirable because it indicates that the cache is effective in storing the most frequently used data and minimizing the need to access slower memory.
- Miss Ratio (%) = (Number of Cache Misses / Total Number of Memory Accesses) * 100

2. What is the need for virtual memory?

1. Extended Memory Capacity: It allows a system to use more memory than physically available by using disk space as an extension of RAM, enabling larger applications to run.
2. Isolation and Protection: Virtual memory provides each process with its own address space, isolating them from one another and preventing accidental interference, which enhances security and stability.
3. Efficient Memory Management: It enables more efficient use of memory through techniques like paging and segmentation, allowing the system to allocate memory dynamically based on need.
4. Simplified Memory Allocation: Programmers can write applications without worrying about the limitations of physical memory, as the operating system manages memory allocation and addresses.
5. Multitasking Support: Virtual memory facilitates multitasking by allowing multiple processes to run simultaneously without exhausting physical memory resources.

Conclusion

In this experiment, we implemented the LRU Page Replacement Algorithm using Java and showcased its real-life applications. LRU demonstrates its effectiveness in minimizing page faults by prioritizing frequently accessed pages, which enhances the overall performance of the system.

Date: 16 / 09 / 24