

CSE 406 – Lab Report: Least Recently Used (LRU) Page Replacement

Submitted By:

Sharif Md. Yousuf
ID: 22101128
Section: C-2
4th Year, 1st Semester
Spring 2025

Submitted To:

Atia Rahman Orthi
Lecturer
Department of Computer Science & Engineering
University of Asia Pacific

**Date of Submission:
9 October, 2025 (Thursday)**

1 Problem Statement

In this lab, I implemented a demand paging simulator that applies the Least Recently Used (LRU) page replacement strategy. The program receives an initial configuration and then processes a complete reference string, replacing pages based on how recently they were accessed. The tool must trace each reference, track the content of the frames, and compute overall hit/fault statistics.

Input

For this experiment, I worked with the following configuration:

- Frame size: 4 frames
- Number of page references: 11
- Page reference string

7 0 1 2 0 3 0 4 2 3 0

Output

My simulator prints the frame state after every access and aggregates the final performance statistics. A sample run with the above parameters produces:

Page Reference | Frames Status | Hit/Miss

7		[7]	[]	[]	[]		MISS
0		[7]	[0]	[]	[]		MISS
1		[7]	[0]	[1]	[]		MISS
2		[7]	[0]	[1]	[2]		MISS
0		[7]	[0]	[1]	[2]		HIT
3		[3]	[0]	[1]	[2]		MISS
0		[3]	[0]	[1]	[2]		HIT
4		[3]	[0]	[4]	[2]		MISS
2		[3]	[0]	[4]	[2]		HIT
3		[3]	[0]	[4]	[2]		HIT
0		[3]	[0]	[4]	[2]		HIT

===== RESULTS =====

Total Page Requests: 11

Total Page Hits: 5

Total Page Faults: 6

Hit Ratio: 45.45%

Fault Ratio: 54.55%

2 Objective

Through this lab, I aimed to:

- Understand the mechanics behind the LRU page replacement policy
- Implement an accurate frame tracking system that updates recency information on every access
- Measure hit/fault statistics for a fixed reference string and memory size
- Contrast LRU behaviour with FIFO and MRU policies explored in earlier labs
- Appreciate how LRU improves hit ratio by evicting the page that has been unused the longest

3 Source Code Screenshot

A screenshot of a code editor window titled "LRU_Page_Replacement.cpp". The code is written in C++ and implements the Least Recently Used (LRU) page replacement algorithm. It includes headers, uses the std namespace, and defines a main function. The main function takes frame_size and num_pages as input, initializes a vector of pages, and then iterates through them. For each page, it checks if it is already in the frames. If it is, it updates the lastUsed index and increments pageHits. If it is not, it increments pageFaults and finds the index of the least recently used page (the one with the smallest lastUsed value) to replace it. The code uses cout to print the page reference and frames status at each step.

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  int main() {
6      int frame_size, num_pages;
7      cin >> frame_size >> num_pages;
8
9      vector<int> pages(num_pages);
10     for (int i = 0; i < num_pages; i++) {
11         cin >> pages[i];
12     }
13
14     vector<int> frames(frame_size, -1), lastUsed(frame_size, -1);
15     int pageFaults = 0, pageHits = 0;
16
17     cout << "\nPage Reference | Frames Status | Hit/Miss\n";
18     cout << "-----\n";
19     for (int i = 0; i < num_pages; i++) {
20         int currentPage = pages[i];
21         bool found = false;
22         int foundIndex = -1;
23
24         for (int j = 0; j < frame_size; j++) {
25             if (frames[j] == currentPage) {
26                 found = true;
27                 foundIndex = j;
28                 break;
29             }
30         }
31
32         cout << "      " << currentPage << "      | ";
33
34         if (found) {
35             pageHits++;
36             lastUsed[foundIndex] = i;
37         } else {
38             pageFaults++;
39
40             int replaceIndex = -1;
41             for (int j = 0; j < frame_size; j++) {
42                 if (frames[j] == -1) {
43                     replaceIndex = j;
44                     break;
45                 }
46             }
```

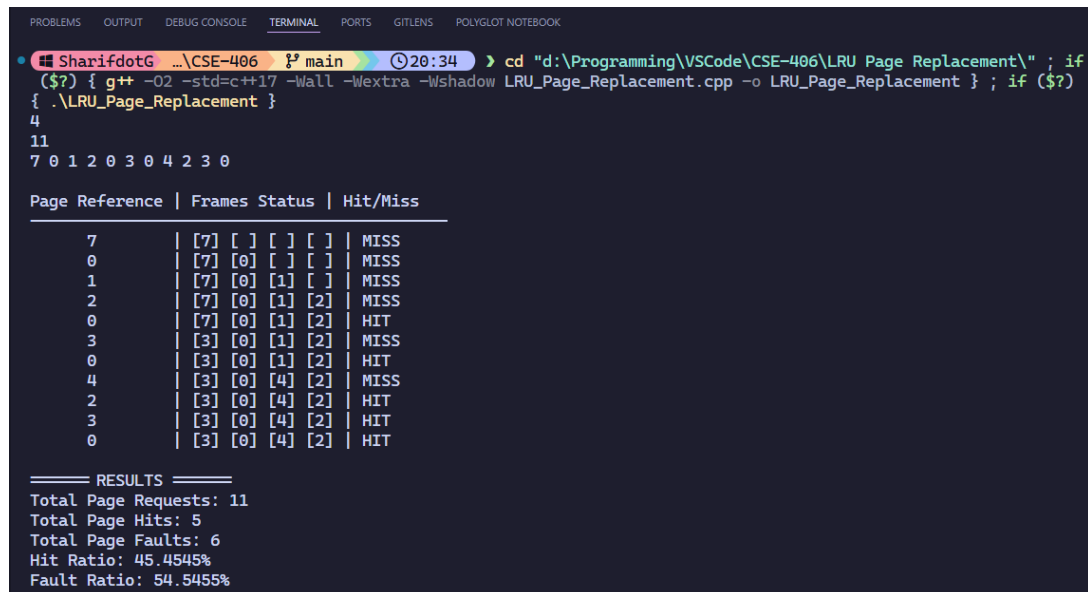
Figure 1: LRU Page Replacement Source Code (Part 1)

```
LRU_Page_Replacement.cpp

1      if (replaceIndex == -1) {
2          int lruValue = numeric_limits<int>::max();
3          for (int j = 0; j < frame_size; j++) {
4              if (lastUsed[j] < lruValue) {
5                  lruValue = lastUsed[j];
6                  replaceIndex = j;
7              }
8          }
9      }
10
11      frames[replaceIndex] = currentPage;
12      lastUsed[replaceIndex] = i;
13  }
14
15  for (int j = 0; j < frame_size; j++) {
16      if (frames[j] == -1) {
17          cout << "[ ] ";
18      } else {
19          cout << "[" << frames[j] << " ] ";
20      }
21  }
22
23  if (found) {
24      cout << "| HIT";
25  } else {
26      cout << "| MISS";
27  }
28  cout << endl;
29  }
30
31  cout << "\n===== RESULTS =====\n";
32  cout << "Total Page Requests: " << num_pages << endl;
33  cout << "Total Page Hits: " << pageHits << endl;
34  cout << "Total Page Faults: " << pageFaults << endl;
35  cout << "Hit Ratio: " << (double)pageHits / num_pages * 100 << "%" << endl;
36  cout << "Fault Ratio: " << (double)pageFaults / num_pages * 100 << "%" << endl;
37
38  return 0;
39 }
```

Figure 2: LRU Page Replacement Source Code (Part 2)

4 Output Screenshot



```
## SharifdotG ...\CSE-406 ? main 20:34 > cd "d:\Programming\VSCode\CSE-406\LRU Page Replacement\" ; if ($?) { g++ -O2 -std=c++17 -Wall -Wextra -Wshadow LRU_Page_Replacement.cpp -o LRU_Page_Replacement } ; if ($?) { .\LRU_Page_Replacement }
4
11
7 0 1 2 0 3 0 4 2 3 0

Page Reference | Frames Status | Hit/Miss
-----
7 | [7] [ ] [ ] [ ] | MISS
0 | [7] [0] [ ] [ ] | MISS
1 | [7] [0] [1] [ ] | MISS
2 | [7] [0] [1] [2] | MISS
0 | [7] [0] [1] [2] | HIT
3 | [3] [0] [1] [2] | MISS
0 | [3] [0] [1] [2] | HIT
4 | [3] [0] [4] [2] | MISS
2 | [3] [0] [4] [2] | HIT
3 | [3] [0] [4] [2] | HIT
0 | [3] [0] [4] [2] | HIT

===== RESULTS =====
Total Page Requests: 11
Total Page Hits: 5
Total Page Faults: 6
Hit Ratio: 45.4545%
Fault Ratio: 54.5455%
```

Figure 3: LRU Program Output

5 Discussion

Working with the LRU policy highlighted how valuable temporal locality is for page replacement. Some of the key insights I recorded during the experiment are:

- **Recency Tracking:** Maintaining the access timestamp for every frame guarantees that the algorithm always evicts the page that has stayed idle the longest, which aligns well with real workloads that reuse recently referenced pages.
- **Improved Hit Ratio:** Compared to FIFO, LRU preserved frequently accessed pages like 0 and 3, leading to five hits out of eleven references even with just four frames.
- **Overhead Considerations:** The implementation needs additional bookkeeping (the `lastUsed` array) to keep recency information up to date, adding a small computational overhead per access.
- **Deterministic Behaviour:** Given a fixed reference string, LRU produces a deterministic trace, making it easy to validate the simulator against manual calculations.
- **Comparison with MRU:** In scenarios where scans sweep through large data sets, MRU can outperform LRU, but for this string LRU clearly avoids wasting frames on rarely reused pages.

6 Conclusion

This lab solidified my understanding of the LRU page replacement approach and illustrated how intelligent eviction decisions can substantially reduce page faults. By tracking

access recency, the simulator managed to retain hot pages and limit total faults to six for the given workload. Although LRU incurs bookkeeping overhead, the resulting hit ratio and predictable behaviour make it a strong baseline policy for demand paging systems. The experience also prepared me to evaluate more advanced approximations of LRU in future experiments.