# Experiment no: 09.

**Name: Meet Brijwani**
**Roll no: 14**
**Batch: S11**

**Aim:** Write a program in C demonstrate the concept of page replacement policies for handling page faults eg: FIFO, LRU

## Theory:

Page replacement policies are an integral part of virtual memory management in operating systems. When a program accesses data or instructions, the operating system loads the corresponding pages into physical memory (RAM) from secondary storage (usually a hard disk). However, physical memory has limited capacity, so not all pages can reside in memory simultaneously. When a program requests a page that is not in memory, a page fault occurs, triggering the need for a page replacement policy to determine which page to evict from memory to make room for the new one. Two commonly used page replacement policies are First-In-First-Out (FIFO) and Least Recently Used (LRU). Let's delve into these policies and understand their mechanisms and trade-offs.

First-In-First-Out (FIFO):
FIFO is one of the simplest page replacement algorithms. It evicts the oldest page in memory, based on the assumption that the page that has been in memory the longest is least likely to be needed in the near future.

Mechanism:
When a page fault occurs and memory is full, the operating system selects the page that entered memory earliest (the oldest page) for replacement.
The selected page is evicted from memory, and the new page is loaded in its place.

The page table is updated accordingly.
Advantages:
Simplicity: FIFO is easy to implement and understand.
Low Overhead: The overhead of maintaining data structures is minimal.
Disadvantages:
Belady's Anomaly: FIFO can suffer from Belady's anomaly, where increasing the number of frames can actually increase the number of page faults.
Poor Performance: FIFO does not consider the access history of pages, leading to suboptimal performance in many cases, especially when the access patterns are irregular.
Least Recently Used (LRU):
LRU is based on the principle that the page that has not been accessed for the longest time is least likely to be used in the near future.

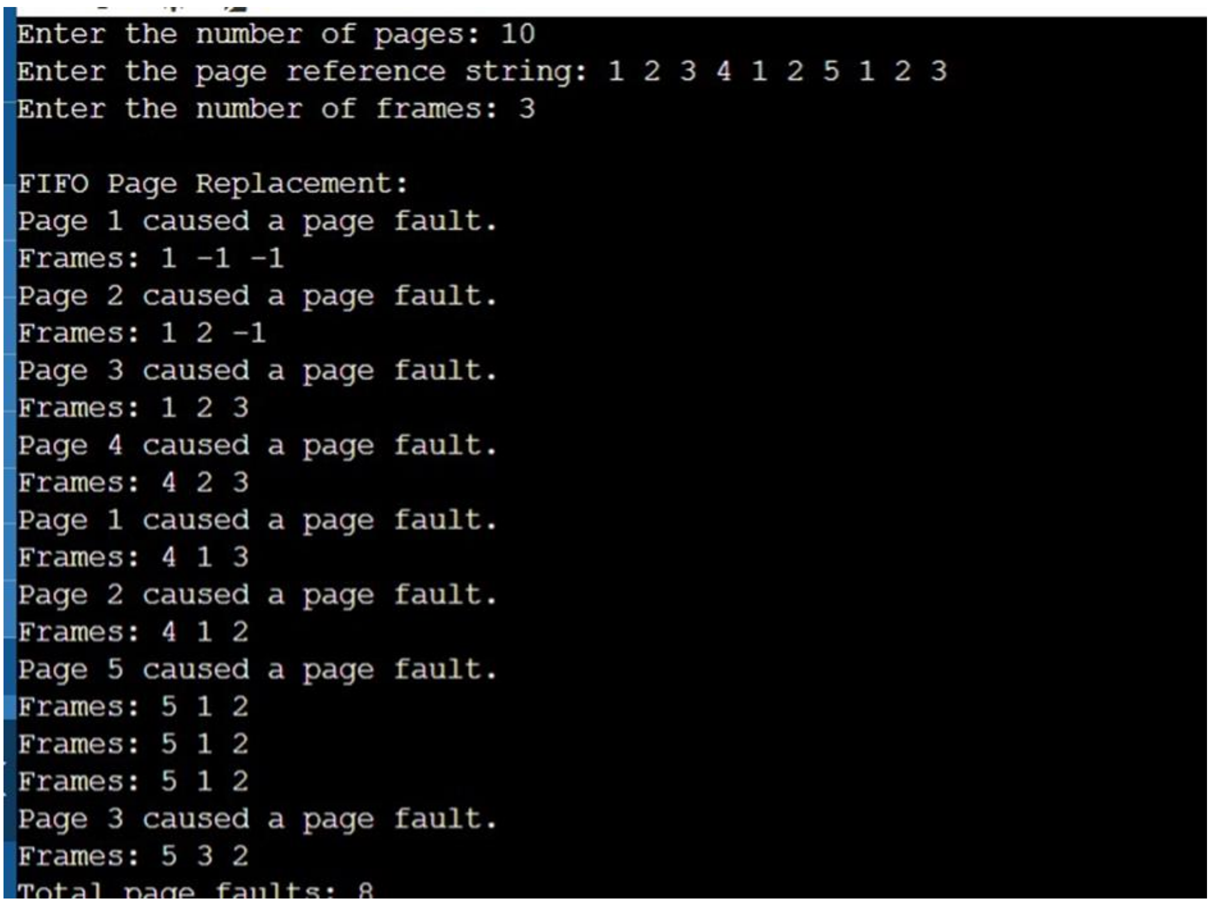
Mechanism:
LRU keeps track of the time of the last access for each page.
When a page fault occurs, the operating system selects the page that was least recently accessed for replacement.
The selected page is evicted from memory, and the new page is loaded in its place.
The page table is updated accordingly, and the access time of the new page is recorded.
Advantages:
Optimality: LRU provides better performance than FIFO in terms of reducing the number of page faults in many scenarios.
Flexibility: LRU can adapt to varying access patterns by considering the access history of pages.
Disadvantages:
Implementation Complexity: Implementing an efficient LRU algorithm requires maintaining a data structure to track the access times of pages, which can be resource-intensive.
High Overhead: The overhead of maintaining access times for each page can be significant, especially in systems with a large number of pages.
Comparison and Trade-offs:

Optimality: LRU is generally considered more optimal than FIFO because it takes into account the actual access history of pages rather than just the order of arrival. However, implementing a true LRU algorithm can be complex and resource-intensive.
Overhead: FIFO has lower overhead compared to LRU since it does not require tracking access times for each page. However, this simplicity comes at the cost of potentially poorer performance.
Belady's Anomaly: FIFO can suffer from Belady's anomaly, where increasing the number of frames can paradoxically increase the number of page faults. LRU does not suffer from this anomaly.
Adaptability: LRU is more adaptable to varying access patterns since it considers the actual access history of pages. FIFO, on the other hand, may perform poorly in scenarios with irregular access patterns.

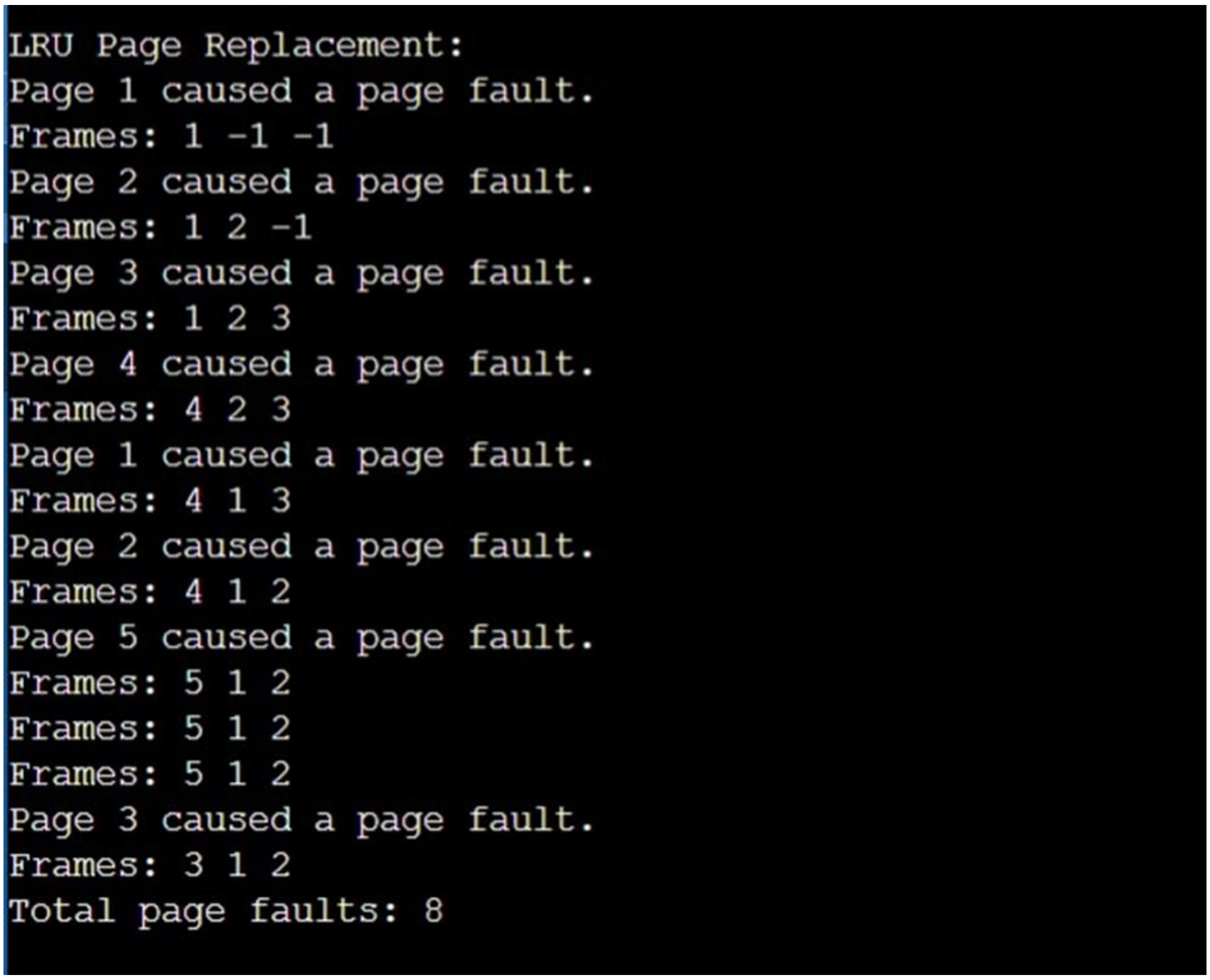
## Program:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_FRAMES 3 // Maximum number of page frames
#define MAX_PAGES 10 // Maximum number of pages in reference
string
// Function prototypes void fifo(int pages[], int n, int frames); void lru(int
pages[], int n, int frames);

int main() {     int pages[MAX_PAGES];
    int n, frames;

    printf("Enter the number of pages: ");     scanf("%d", &n);

    printf("Enter the page reference string: ");     for (int i = 0; i < n; i++) {
scanf("%d", &pages[i]);
    }

    printf("Enter the number of frames: ");     scanf("%d", &frames);

    printf("\nFIFO Page Replacement:\n");     fifo(pages, n, frames);
    printf("\nLRU Page Replacement:\n");     lru(pages, n, frames);
    return 0;
}

void fifo(int pages[], int n, int frames) {     int frame[MAX_FRAMES];
    int page_faults = 0;     int current_frame = 0;

    for (int i = 0; i < frames; i++) {         frame[i] = -1; // Initialize frames as
empty
    }

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

        // Check if page is already in a frame         for (int j = 0; j < frames;
j++) {             if (frame[j] == page) {                 found = 1;                 break;
            }
        }

       // Page fault: replace the oldest page
        if (!found) {             printf("Page %d caused a page fault.\n", page);
frame[current_frame] = page;             current_frame = (current_frame +
1) % frames; // Circular queue
            page_faults++;
        }

        // Display current state of frames         printf("Frames: ");         for
(int j = 0; j < frames; j++) {             printf("%d ", frame[j]);
        }        printf("\n");
    }

    printf("Total page faults: %d\n", page_faults);
}

void lru(int pages[], int n, int frames) {     int frame[MAX_FRAMES];
    int page_faults = 0;     int counter[MAX_FRAMES] = {0};

    for (int i = 0; i < frames; i++) {
        frame[i] = -1; // Initialize frames as empty
    }

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

        // Check if page is already in a frame         for (int j = 0; j < frames;
j++) {             if (frame[j] == page) {                 found = 1;
counter[j] = i + 1; // Update counter to indicate recent access
                break;
            }
        }

        // Page fault: replace the least recently used page         if (!found) {
printf("Page %d caused a page fault.\n", page);             int min_counter =
counter[0];             int min_index = 0;             for (int j = 1; j < frames; j++)
{                 if (counter[j] < min_counter) {                     min_counter =
counter[j];
                    min_index = j;
                }
            }
            frame[min_index] = page;             counter[min_index] = i + 1; //
Update counter to indicate recent access             page_faults++;
        }

        // Display current state of frames         printf("Frames: ");         for
(int j = 0; j < frames; j++) {             printf("%d ", frame[j]);
        }         printf("\n");
    }

    printf("Total page faults: %d\n", page_faults);
}
```

**Output:**



```
Enter the number of pages: 10
Enter the page reference string: 1 2 3 4 1 2 5 1 2 3
Enter the number of frames: 3

FIFO Page Replacement:
Page 1 caused a page fault.
Frames: 1 -1 -1
Page 2 caused a page fault.
Frames: 1 2 -1
Page 3 caused a page fault.
Frames: 1 2 3
Page 4 caused a page fault.
Frames: 4 2 3
Page 1 caused a page fault.
Frames: 4 1 3
Page 2 caused a page fault.
Frames: 4 1 2
Page 5 caused a page fault.
Frames: 5 1 2
Frames: 5 1 2
Frames: 5 1 2
Page 3 caused a page fault.
Frames: 5 3 2
Total page faults: 8
```



```
LRU Page Replacement:
Page 1 caused a page fault.
Frames: 1 -1 -1
Page 2 caused a page fault.
Frames: 1 2 -1
Page 3 caused a page fault.
Frames: 1 2 3
Page 4 caused a page fault.
Frames: 4 2 3
Page 1 caused a page fault.
Frames: 4 1 3
Page 2 caused a page fault.
Frames: 4 1 2
Page 5 caused a page fault.
Frames: 5 1 2
Frames: 5 1 2
Frames: 5 1 2
Page 3 caused a page fault.
Frames: 3 1 2
Total page faults: 8
```

## Conclusion:

In summary, page replacement policies such as FIFO and LRU play a crucial role in virtual memory management by determining which pages to evict from memory when page faults occur. While FIFO is simple to implement and has low overhead, it may perform poorly in scenarios with irregular access patterns and can suffer from Belady's anomaly. LRU, on the other hand, provides better performance by considering the actual access history of pages, but it comes with higher implementation complexity and overhead. The choice between FIFO and LRU depends on factors such as the system's requirements, workload characteristics, and available resources.