✅ **Theory: LRU Page Replacement Algorithm**

📌 **What is Page Replacement?**

When a process tries to access a page that is not present in main memory (RAM), a page fault occurs.
The OS must bring that page from disk into memory.
But if memory (frames) is full, OS must replace one of the existing pages using a page replacement algorithm.

📌 **LRU (Least Recently Used) Algorithm**

- **LRU replaces the page which has not been used for the longest time.**
- **It assumes past behavior predicts future behavior (i.e., page not used recently will not be used soon).**
- **It maintains a timestamp or counter to track when each page was last used.**

✅ **Advantages**

- **More efficient than FIFO, because it considers page usage history.**
- **Reduces page faults compared to FIFO.**

❌ **Disadvantages**

- **Requires extra storage and computation to maintain timestamps.**
- **Hardware support (counters/stack) may be required for efficiency.**

✅ **Code Explanation (Line-by-Line)**

int frames, pages[100], frame[10], used[10], n, i, j, k;

int faults = 0, hits = 0, time = 0;

- **frames → number of physical memory frames.**
- **pages[] → page reference string entered by user.**
- **frame[] → current pages stored in memory.**
- **used[] → stores timestamp of when each frame was last used.**
- **faults and hits → counters.**
- **time → logical counter to simulate timestamps.**

---

printf("Enter number of frames: ");

scanf("%d", &frames);

Takes number of frames (e.g., 3 frames).

printf("Enter number of pages: ");

scanf("%d", &n);

Accepts length of reference string (e.g., 12 pages).

for(i = 0; i < n; i++)

** scanf("%d", &pages[i]);**

Reads each page number into array.

✅ **Initialization**

for(i = 0; i < frames; i++)

** frame[i] = -1, used[i] = -1;**

- **-1 means empty frame.**
- **used[] = -1 means no timestamp yet.**

✅ **Main Logic: Processing Each Page**

for(i = 0; i < n; i++) {

** int page = pages[i], hit = 0;**

- **page → current page request**
- **hit → becomes 1 if page already exists in memory**

✅ **Page Hit Checking**

for(j = 0; j < frames; j++) {

** if(frame[j] == page) {**

**   hit = 1;**

**   used[j] = time++;**

**   hits++;**

** }**

}

- **If page found → hit**
- **Update timestamp of that frame → used[j] = time++**

✅ **Page Fault Handling**

if(!hit) {

```
  int pos = 0;
  for(j = 0; j < frames; j++)
    if(frame[j] == -1 || used[j] < used[pos])
      pos = j;
```

- If page not found → Page Fault
- Select frame with least used timestamp → used[j] < used[pos]
- If frame is empty (-1) → directly use it

```
frame[pos] = page;
used[pos] = time++;
faults++;
}
```

- Insert new page into selected frame
- Update timestamp
- Increase fault count

✅ **Displaying Output**

```
printf("%d\t", page);
for(k = 0; k < frames; k++)
  printf("%s ", frame[k] == -1 ? "-" : (char [3]){frame[k]+'0',0});
```

Prints current memory state after every page request.

```
if(hit)
  printf("\t\tHit (%d)\n", page);
else
  printf("\t\tPage Fault\n");
```

Displays whether page was hit or fault.

✅ **Final Output**

```
printf("\nTotal Page Faults: %d", faults);
printf("\nTotal Page Hits: %d\n", hits);
```