# WEEK-6

## AIM: To implement LRU algorithm

## DESCRIPTION:

In operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

The **L**east **R**ecently **U**sed (LRU) algorithm is a Greedy algorithm where the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page is not likely

Let say the page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 . Initially we have 4 page slots empty.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**

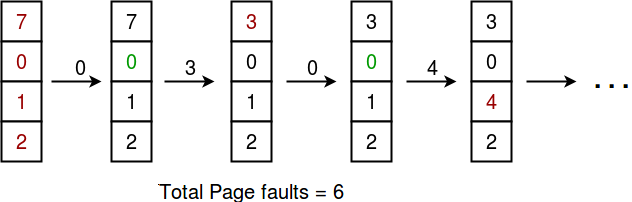
0 is already their so —> **0 Page fault.**

when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**

0 is already in memory so —> **0 Page fault**.

4 will takes place of 1 —> **1 Page Fault**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.



## IMPLEMENTATION:

slots = int(input("Enter the number of page slots"))

pagestring = input("Enter the page reference string")

pagelist = list(map(int,pagestring.split()))

pf = 0

if len(pagelist) > slots:

s = pagelist[:slots]

pf += len(s)

for i in range(slots,len(pagelist)):

if pagelist[i] not in s:

pf += 1

lookup = pagelist[i-1::-1]

mi = 0

for x in s:

k = lookup.index(x)

if k > mi:

mi = k

s[s.index(lookup[mi])] = pagelist[i]

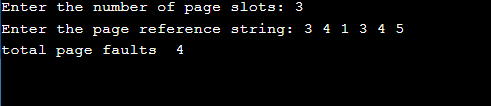
print('total page faults ',pf)

else:

pf = len(pagelist)

print('total page faults',pf)

## OUTPUT:



## C implementation:

#include<stdio.h>

int findLRU(int time[], int n){

int i, minimum = time[0], pos = 0;

for(i = 1; i < n; ++i){

if(time[i] < minimum){

minimum = time[i];

pos = i;

}

}

return pos;

}

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter reference string: ");

for(i = 0; i < no\_of\_pages; ++i){

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

}

for(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == pages[i]){

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == -1){

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j){

printf("%d\t", frames[j]);

}

}

printf("\n\nTotal Page Faults = %d", faults);

return 0;

}

## OBSERVATIONS:

One important advantage of the LRU algorithm is that it is amenable to full statistical analysis. It has been proven, for example, that LRU can never result in more than N-times more page faults than OPT algorithm, where N is proportional to the number of pages in the managed pool.

On the other hand, LRU's weakness is that its performance tends to degenerate under many quite common reference patterns. For example, if there are N pages in the LRU pool, an application executing a loop over array of N + 1 pages will cause a page fault on each and every access. As loops over large arrays are common, much effort has been put into modifying LRU to work better in such situations