TRIBHUWAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A LAB REPORT
ON

Page Replacement Algorithms

Lab No: 7
Experiments Date:
Submission Date:

<u>Submitted By:</u>
Name: Nabin Khanal
Group: B
Roll No: 076BCT036

<u>Submitted To:</u>
Department of
Electronics and computer engineering

TITLE: PAGE REPLACEMENT ALGORITHMS

OBJECTIVE

- To simulate FIFO page replacement algorithm
- To simulate LRU page replacement algorithm
- To simulate LFU page replacement algorithm

THEORY

In an operating system page replacement is referred to the scenario in which a page from main memory should be replaced by a page from secondary memory. Page replacement is needed in operating systems that use virtual memory using Demand Paging. As we know that in Demand paging, only a set of pages of a process is loaded into the memory. This is done so that we can have more processes in the memory at the same time.

When a page that is residing in virtual memory is requested by a process for its execution, the operating system needs to decide which page will be replaced by ~~these~~ this requested page. This process is known as page replacement and is a vital component in virtual memory management.

A page fault occurs when a program running in the CPU tries to access the page that is in address space of that program but that ~~program~~ page is not currently loaded into the main physical memory.

Since the actual RAM is much less than the virtual memory, the page fault occurs. So whenever a page fault occurs operating system has to replace an existing page in RAM with newly requested page. In this scenario, page replacement algorithm helps the OS decide which page to replace. The primary objective of all the page replacement algorithms is to minimize the number of page faults.

Page Replacement Algorithms.

First In First Out

FIFO algorithm is the simplest of all the page replacement algorithms. In this, we maintain a queue of all pages that are in the memory currently. The oldest page in memory is at the front end of the queue and the most recent page is at the back end of the queue.

whenever a page fault occurs, the operating system looks at the front end of the queue to know the page to be replaced by the newly requested page. It also adds this newly requested page at the rear end and removes the oldest page from front end of the queue.

Consider the page reference string as
3 1 2 1 6 5 1 3.

Pages	3	1	2	1	6	5	1	3
frames	3	3	3	3	6	6	6	3
		1	1	1	1	5	5	5
			2	2	2	2	1	1
	Miss	Miss	miss	Hit	miss	Miss	Miss	Miss

Hit ratio = $\frac{1}{8}$

Least Recently used (LRU) Page Replacement Algorithm

The least recently used page replacement algorithm keeps the track of usage of pages over a period of time. This algorithm works on the basis of principle of locality of reference. which states that a program has tendency to access the same set of memory locations repetitively over a short period of time. So pages that have been used heavily in the past are most likely to be used in the future also.

In this algorithm, when a page fault occurs, then the page that has not been used for the longest duration of time is replaced by the newly requested page.

Let's see the performance of LRU on the same reference string of 3 1 2 1 6 5 1 3 with 3 page frames.

Pages:	3	1	2	1	6	5	1	3
Frames:	3	3	3	3	6	6	6	3
		1	1	1	1	1	1	1
			2	2	2	5	5	5
	Miss	Miss	miss	HIT	Miss.	Miss.	Hit	Miss.

Hit ratio = $\frac{2}{8}$

LFU (Least Frequently Used) Page Replacement algorithm.

This method keeps the track of number of times a block is referenced. When page replacement is required, the item with the least frequency is replaced with the current item. It can be implemented by assigning counter to every block that is ~~loaded~~ loaded into the cache. Every time a reference is made the counter is increased.

Consider the page reference string as
3 1 2 1 6 5 1 3

Pages	3	1	2	1	6	5	1	3
frames	3	3	3	3	1	1	1	1
		1	1	1	2	6	6	5
			2	2	6	5	5	3
				HIT			HIT	

Hit ratio = $\frac{2}{8}$

Source Code:

```
def page replacement_fifo (data, no_of_frames):
    frame = []
     no_of_hits = 0
      all_frames = []
      current_index = 0
      for _ in rangecno_of_items):
            all_frames.append ([])
      for element in data:
            if element in frame:
                  no_of_hits += 1
            else
                  if len(frame) < no_of_frames:
                        frame.append(element)
                  else:
                        frame[current_index] = element
                        current_index = (current_index
                                             + 1)% no_of_frames
```

```
        for i in range(no_of_frames):
            if (len(frames) > i):
                all_frames[i].append(frame[i])
            else:
                all_frames[i].append('_')
    return all_frames, no_of_hits


def page_replacement_lru(data, no_of_frames):
    frames = []
    all_frames = []

    for _ in range(no_of_frames):
        ~~aff~~ all_frames.append([])
    for index, element in enumerate(data):
        if element in frame:
            no_of_elements += 1
        else:
            present = []
            for i in range(index-1, -1, -1):
                if data[i] in present:
                    pass
                elif len(present) < (no_of_frames -
                    present.append(data[i])
                else:
                    to_remove = data[i]
                    break

            to_remove_index = frame.index(to_remove
                                    0, len(frame))

            Frame[to_remove_index] = element
    for i in range(no_of_frames):
        if (len(frame) > i):
            all_frames[i].append(frame[i])
        else:
            all_frames[i].append('_')
return all_frames, no_of_hits.


def_page replacement_lfu (data, no_of frames):
    frame = {}
    no_of_hits =0
    all_frames = []
    for _ in range (no_of_frames):
        all_frames. append([])
    for index, element in enumerate(data):
        if element in frame. keys():
            no_of_hits += 1
            frame[element]+=1
        else
            if len(frame) < no_of_frames:
                frame [element] = 1.
            else:
                minval = 1000
                minkey = ''
                for key, value in frame.items():
                    if value < minval :
                        minval = value
                        minkey = key
                frame.pop(minkey)
                frame[element]=1
        framelist = [(k,v) for k,v in frame.items()]
```

```
            for ia in range (no_of_frames):
                if len(frames) >i:
                    all_frames[i].append (framelist
                                          [i][0])
                else:
                    all_frames[i].append(-)
    return all_frames, no_of_hits


def main():
    no_of frames = 3
    data = ['2','3','2','1','5','2','4','5','3','2',
            '5', '2']
    print (data)
    print()
    print("FIFO")
    all_frames,no_of_hits = page_replacement_
                            fifo(data,no_of_frames)
    for i in all_frames:
        print (i)

    print ("No of hits:",no_of_hits)
    print ("Hit Ratio", no_of_hits/len(data))
main()
```

Output

```
F-2   3   2   1   5   2   4   5   3   2   5   2

FIFO.
2   2   2   2   ~~2~~5   5   5   5   3   3   3   3
-   3   3   3   3    2   2   2   2   2   5   5
-   -   1   1   1    4   4   4   4   4   4   2
No of hits: 3
Hit ratio: 0.25
```

Discussion and Conclusion

Page Replacement algorithms are used to decide which memory pages to page out in the systems using paging for virtual memory management. commonly used page replacement algorithms are FIFO, LRU, LFU. .

In this lab, we learnt about the working of these algorithms. we implemented these algorithms and tested the output.