CS 314 – Operating Systems Lab
Lab–8 Report

Student Name: Kavali Sri Vyshnavi Devi **Roll-No:** 200010023

1 Page Replacement Algorithms

The page replacement algorithm decides which page in memory is to be replaced. The process of replacement is sometimes called swap out or write to disk. When a new page needs to be loaded into the main RAM, the Page Replacement Algorithm decides which page to remove. Page Replacement occurs when a requested page is absent in the main memory, and there is insufficient free room to allocate the requested page. In this assignment three page replacement algorithms are implemented namely FIFO (first in first out), Random and LRU (least recently used).

1.1 FIFO

Operating systems use the page replacement algorithm FIFO (First-In-First-Out) to handle memory. The operating system replaces the page that has been in memory the longest using this method. The page that was initially put into memory is replaced with the new page when a page fault happens. The FIFO page replacement algorithm works like a queue. Pages are loaded into memory in the sequence that the process requests them. The oldest page in memory (the one at the front of the queue) is swapped out for the new page when memory is filled and a new page needs to be loaded.

1.1.1 Advantages

- Simple implementation: The FIFO algorithm is easy to implement and does not require complex data structures or algorithms.
- Low overhead: The overhead of the FIFO algorithm is low, as it only requires a simple queue data structure to keep track of the pages in memory.
- Fairness: The FIFO algorithm provides a fair allocation of memory to processes, as it follows a strict first-in-first-out order.
- No starvation: The FIFO algorithm ensures that each page in memory has an equal chance of being replaced, which prevents any page from being constantly swapped out, leading to starvation.

1.1.2 Disadvantages

- No consideration of page usage: The FIFO algorithm does not consider how often a page is used or how critical it is to the process. Therefore, it may replace important pages that are frequently used, leading to reduced performance.
- No adaptation to changing workloads: The FIFO algorithm does not adapt to changing workloads or process requirements, as it always replaces the oldest page in memory.
- Poor performance: The FIFO algorithm may not provide optimal performance in systems with a high page fault rate, as it does not take into account the frequency or importance of page references.

1.2 Random

This algorithm, as the name suggests, chooses any random page in the memory to be replaced by the requested page. This algorithm can behave like any of the algorithms based on the random page chosen to be replaced. Random replacement algorithm replaces a random page in memory. This eliminates the overhead cost of tracking page references. This algorithm don't need to maintain the frequency of pages or recency of a page is accessed.

1.2.1 Advantages

- It is easy to implement: Random page replacement requires minimal bookkeeping and does not require complex data structures or algorithms to operate. This simplicity makes it a popular choice for small embedded systems or low-resource environments.
- It is fair: Random page replacement treats all pages equally, regardless of their access history or content. This means that every page has an equal chance of being selected for eviction, ensuring that no page is prioritized over others.
- The random page replacement algorithm does not require any additional information about the state of the system or the access patterns of individual pages. This means that it can operate quickly and with minimal overhead.

1.2.2 Disadvantages

- No consideration for page importance: Random page replacement treats all pages equally, regardless of their importance to the system's performance. This means that important pages may be evicted, leading to a decrease in system performance.
- No consideration for the cost of page replacement: Random page replacement does not take into account the cost of replacing a page. Some pages may have a high cost of replacement, such as those containing kernel code or shared libraries. Evicting these pages can lead to a significant overhead cost.

- High variance: The randomness of the algorithm can lead to high variance in performance. The algorithm may randomly select a page to evict that is currently being heavily used, leading to a high rate of page faults and decreased performance.

1.3 LRU (least recently used)

As the name suggests, this algorithm is based on the strategy that whenever a page fault occurs, the least recently used page will be replaced with a new page. This algorithm can be implemented using time stamp for each page and we update the time stamp of each page when it is accessed but implementing LRU is more complex as compared to other algorithms. LRU can be implemented in software and hardware.

In software LRU implementation OS maintains ordered list of physical pages by reference time, when page is referenced - move page to front of list, when need a victim - remove the page from the front of the list. **Trade off:** Slow on memory reference, fast on replacement.

while implementing hardware LRU one need to associate timestamp register with each page, when page is referenced - move page to front of list, when need victim - scan through registers to find oldest clock. **Trade off:** Slow on memory reference, fast on replacement.

1.3.1 Advantages

- Minimizes page faults: LRU page replacement algorithm is designed to minimize page faults, which occur when the system needs to access a page that is not in memory. By keeping the most recently used pages in memory, LRU can reduce the number of page faults and improve system performance.
- Adaptable to different workloads: LRU page replacement algorithm can be adapted to different workloads and system configurations, making it suitable for a wide range of applications and systems.
- Maximizes hits: LRU page replacement algorithm ensures that the pages that are most recently used are kept in memory, resulting in a high hit rate. This means that the system can access the data it needs more quickly and efficiently.

1.3.2 Disadvantages

- Overhead: The LRU algorithm requires the maintenance of a page usage history for each page in memory. This history must be updated each time a page is accessed, which adds overhead to the system.
- Complexity: The LRU algorithm is more complex than other page replacement algorithms, such as FIFO or random page replacement. This complexity can result in slower performance and increased resource usage.
- The LRU algorithm is not always accurate in predicting future page accesses. In some cases, pages that have not been used for a long time may still be accessed, while pages that have been recently used may not be accessed again for some time.

1.4 Comparison Of Algorithms

In all the three above mentioned algorithms one can say coming to implementation FIFO and random are easy to implement where as, LRU need more data structure and involves a lot of book keeping as there is need to find the most recent pages to find a victim among all the pages present in the memory. LRU exploits locality as the main intuition behind it is if a page is accessed in the near past there are high possibilities of accessing the same in the near future. While coming to hit rates LRU perform near to optimal and sometimes random policy may perform better than LRU as it depends on the choice of the page being selected at the time of eviction. Sometimes random performs better than the other for example when there is a repeated linear sweep happening through out the memory slightly larger than the cache size in which the upcoming needed page will be just removed before from the memory in case of FIFO and LRU.

2 Implementation Of Algorithms

To implement all the 3 algorithms 2 vectors are maintained. One for the memory and other for the swap space.

- **Corner Cases**
  - If page number is not present in the disk then it is an invalid access.
  - If swapped out pages are incremented that is more than given number of disk blocks cannot swap in or out anymore and return from the function.
- **Page in Memory** - In case of LRU move the page to front of the list. In case of FIFO and Random here is no change in positions of pages in the queue. Hits count get incremented.
- **Page not in Memory**
  - Page in swap space - Remove the page from swap space and push it into the memory, page faults incremented by 1.
  - Page not in swap space - Find victim according to the page replacement algorithm and move it to the swap space and push the page needed into the memory, page faults incremented by 1.

3 Page Selection Strategies

There are various page selection strategies like demand paging and pre-fetching/pre-paging. In this assignment demand paging is used. Demand paging is a memory management technique used by operating systems to optimize memory usage. It allows the system to load only the necessary pages from disk into memory when they are actually needed, rather than loading all the pages into memory at once. In demand paging there is less scope for I/O.

4 Analyzing Graphs Obtained On Different Files

For generating all the below graphs number of addressable pages is set to 60, number of blocks in swaps space is set to 60 and number of pages frames in memory are varying.

The below figure depicts the results obtained based on req1.dat file which is provided as input along with assignment question. Here, at starting FIFO has lesser number of page faults as compared to page faults occurred in LRU this can happen in certain scenarios such as when processing data in a stream or when dealing with a queue of requests where the oldest item is always the next to be processed.

`Sequence - 1 10 32 2 12 5 2 16 9 30 21 35 6 8 7 17 22 38 45 53 43 10 8 20 30 16 18 56 60 57 53 27 35 24 32 13 17 4 5 18 20 52 28 25 18 9 19 3 31 59 11 6 23 28 37 48`

No.of Frames	FIFO	LRU	RANDOM
10	51	53	49
20	40	44	44
30	40	41	40
40	40	40	40
50	40	40	40

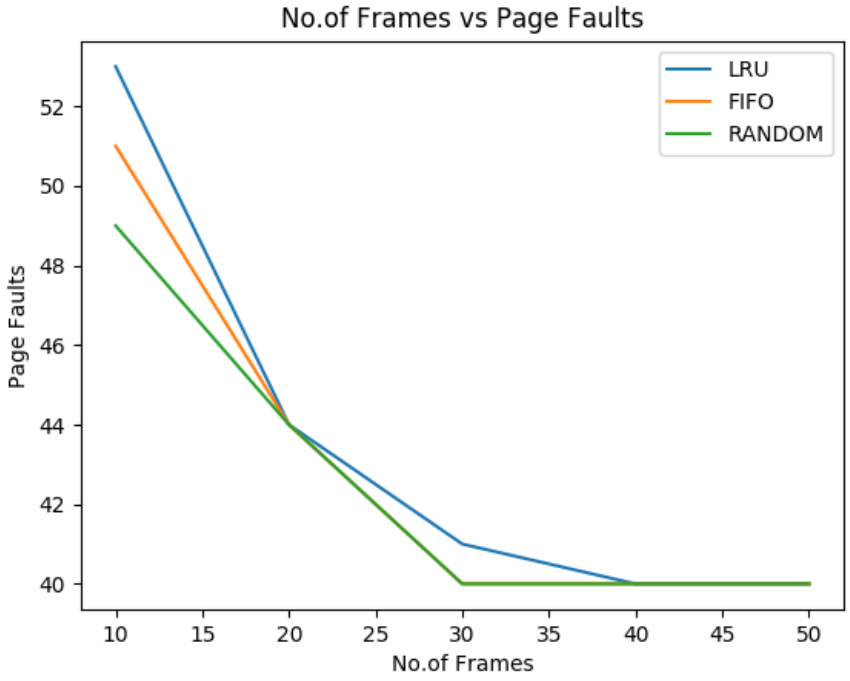

Figure 1: REQUEST1

Request2 file is a looping sequential access and in this at start 1 to 20 pages are accessed linearly, later again 1 to 20 pages are accessed later 1 to 10 pages are accessed linearly in this way at start when no.of frames in memory are less FIFO and LRU remove the page at front of the queue and this will be accessed later and so RANDOM remove the victim randomly which may not be the page at the front of the list which ensures some what less page faults. That is why in the below picture we can observe that the LRU and FIFO perform as same but random at started performed better than other two where as if the no.of frames in the memory greater than the no.of unique pages being accessed all pages will be present in the memory so, page faults will be same in all the three algorithms.

```
Sequence - 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4 5 6 7 8 9 10
```

No.of Frames	FIFO	LRU	RANDOM
10	50	50	46
20	20	20	20
30	20	20	20
40	20	20	20
50	20	20	20

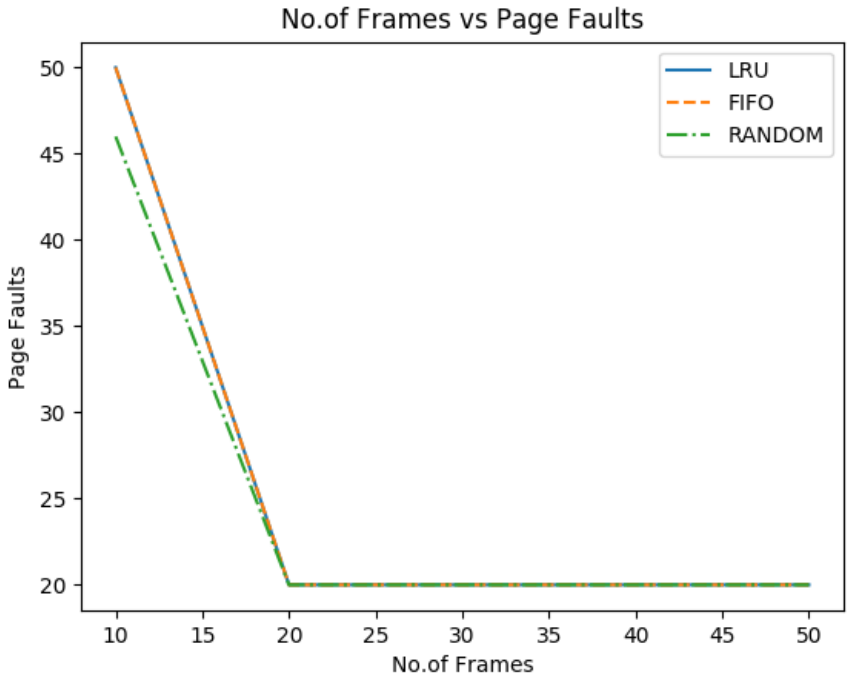

Figure 2: REQUEST2

In request3 the access pattern is like 80 percent of accesses are made to 20 percent of the pages, 20 percent of the accesses are made to 80 percent of the pages. This means that a small set of frequently accessed pages are responsible for the majority of memory hits, while the remaining pages are accessed infrequently. LRU performs well in this scenario because it prioritizes evicting the least recently used data from the memory, which is likely to be among the less frequently accessed data. This allows the cache to retain the frequently accessed data, leading to a lesser number of page faults and better overall performance. From the graph also one can observe that the page faults are more in case of Random as compared to FIFO and LRU.

```
Sequence - 3 10 1 6 5 3 7 8 9 4 41 3 2 3 4 49 42 25 2 9 5 6 2 7 8 34 40 46
1 4 10 39 29 22 2 7 5 6 8 9 1 10 4 5 1 9 8 6 7 10
```

No.of Frames	FIFO	LRU	RANDOM
10	37	36	38
20	20	20	20
30	20	20	20
40	20	20	20
50	20	20	20

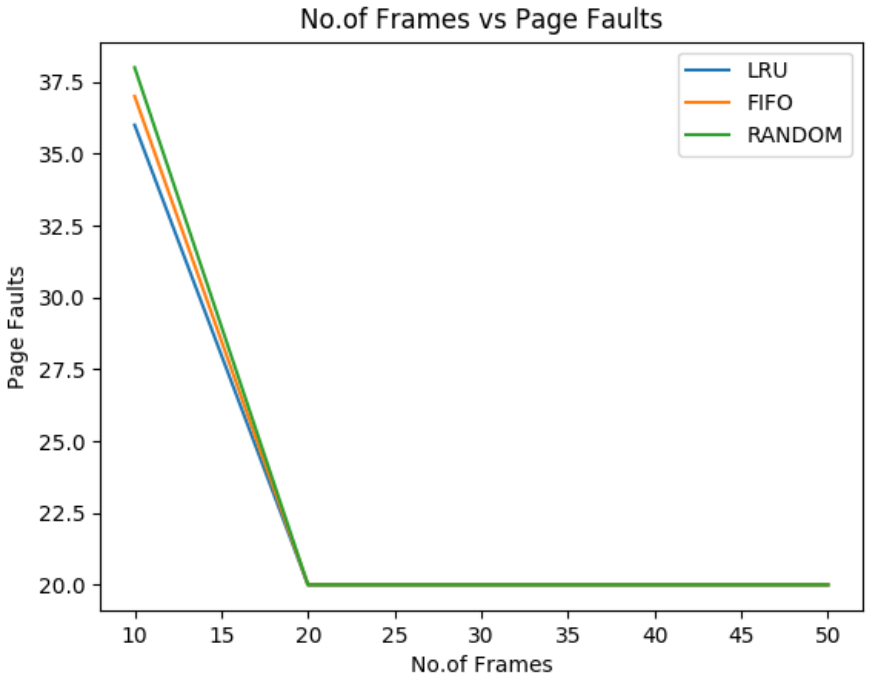

Figure 3: REQUEST3

In request4 accesses are made randomly in which there is no concept of locality. Here, the pages being accessed are not located in a contiguous or predictable location, and there is no pattern to when and where the pages will be accessed. In a workload with no locality that is in which accesses are random, both LRU and FIFO may not perform as well as Random since they are designed to take advantage of the temporal locality of reference, which may not be present in this type of accesses. That is why in the below graph also we can observe that there are many page faults in case of FIFO and LRU but they are less in Random.

```
Sequence - 2 18 13 8 15 16 12 1 7 10 5 3 6 17 11 14 19 20 4 9 17 15 12 20
2 1 11 7 6 4 3 16 19 13 18 8 10 14 5 9 20 1 2 6 11 15 12 7 3 16
```

No.of Frames	FIFO	LRU	RANDOM
10	46	48	38
20	20	20	20
30	20	20	20
40	20	20	20
50	20	20	20

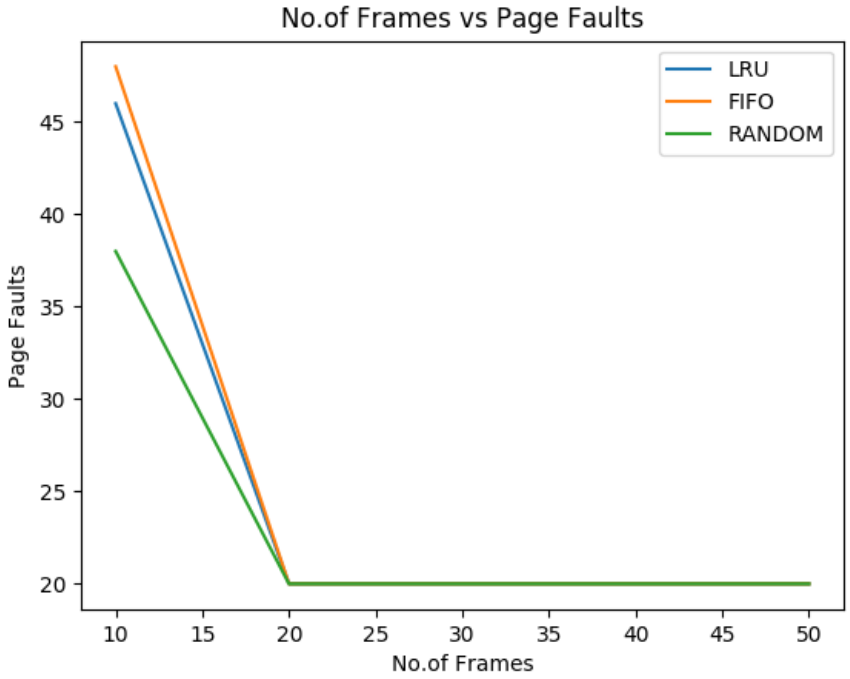

Figure 4: REQUEST4

In request5 the page accesses are following Belady's anomaly. From this type of sequential access patterns we can observe that the number of pages faults increase as the cache size increases in case of FIFO. In the above access pattern if the no.of frames in memory is 9 the number of page faults are equal to 21, when no.of frames in memory is increased to 10 the number of page faults increased to 22. In general this don't happen in case of LRU using k frames will always be a subset of k + n frames for LRU. Thus, any page-faults that may occur for k + n frames will also occur for k frames, which in turn means that LRU doesn't suffer Belady's anomaly. In the graph one can observe that the number of page faults in case of FIFO increase when cache size size increases from 9 to 10.

`Sequence - 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 10 11 1 2 3 4 5 6 7 8 9 10 11`

No.of Frames	FIFO	LRU	RANDOM
9	21	22	18
10	22	15	12
20	11	11	11
30	11	11	11
40	11	11	11
50	11	11	11

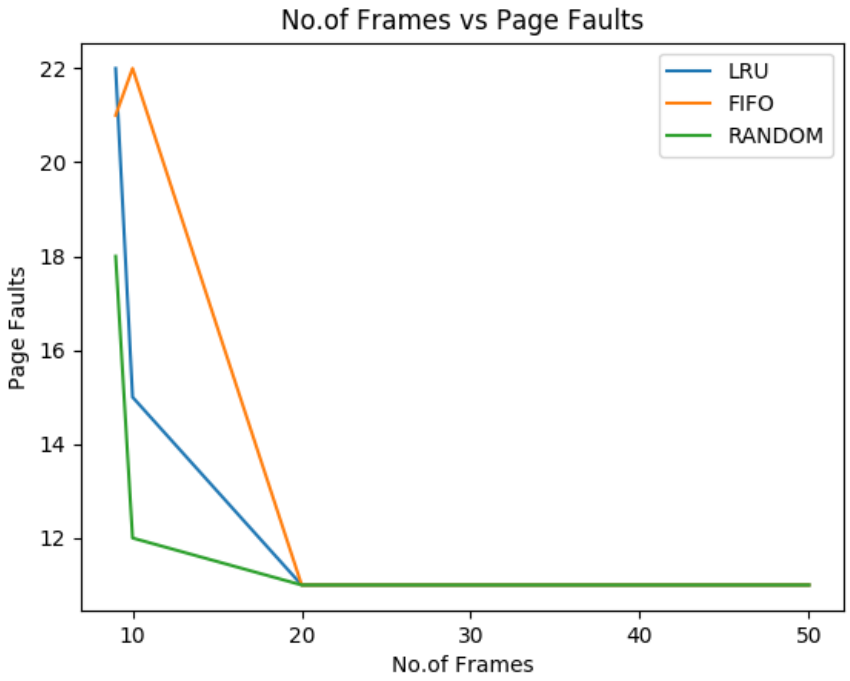

Figure 5: REQUEST5