Harrithha B
200010018

CS314 Operating Systems Lab
Lab 8

---

1 Program Implementation

All the programs were similar in their implementation apart from the page replacement policy.

- Arguments were passed to the program in the following order: `number of logical pages, number of physical frames, number of swap disk blocks, path to page requests file`.
- We will maintain a `page_table` using unordered_map which maps the logical page number to the physical frame number.
- For each page request from the page request input file, we will check whether the logical page number exists in the `page_table`.
- If it exists, increment number of hits and continue to fulfill next request.
- If page not found in `page_table`, page fault is incremented and we will check whether the physical frames are full. If yes, we will have to select a victim frame using one of the below replacement policies and insert the new page.
- If the physical frames are not full, we can simply add the new page to the next free slot in the physical frame.

2 FIFO Replacement Policy

- Here, a queue `loaded_frames` was maintained and each time a page was added to the physical frame, the page was pushed to the queue.
- During eviction, we select the first page in the queue as the victim page to evict since it arrived first in the page table.

3 LRU Replacement Policy

- Here, a vector `page_queue` is maintained to handle the page requests and eviction. Whenever a page is requested, we will check if this page exists in the `page_queue`.
- If present, we will move the page to the end of the vector, indicating that it is the recently accessed page.
- For eviction, we select the page which is at the beginning of the vector as it is the oldest accessed page among all the pages in the `page_queue`.

4 Random Replacement Policy

- Here during eviction, we select a random physical frame as the victim frame to be replaced and written to disk.

5 Test Cases, Plots and Analysis

All the below test cases were simulated such that for each request file, experiments were run with frame size ranging from 1-61 for each of the three replacement policies. A single graph capturing the comparison between the 3 replacement policies was plotted.

5.1 No pattern of requests - req1.dat

```
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
```

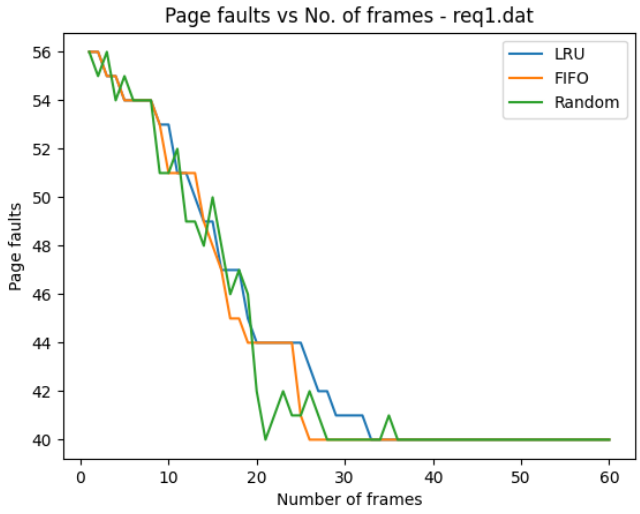

Figure 1: req1.dat

Analysis

- Here we can see that lru is performing best and the results are not too consistent. This is probably due to the fact that the request pattern is not consistent or predictable, which makes it difficult for LRU and FIFO to effectively predict which items are more likely to be accessed in future.
- Random replacement policy does not rely on any assumptions about access patterns and is thus able to handle irregular and unpredictable access patterns.

5.2 Belady's Anomaly - req2.dat

```
1 2 3 4 1 2 5 1 2 3 4 5 6 7 8 6 7 8 9 10 9 10 11 12 11 12 13 14 13 14 15 16 15 16 17 18
    17 18 19 20 19 20 21 22 21 22 23 24 23 24 25 26 25
```

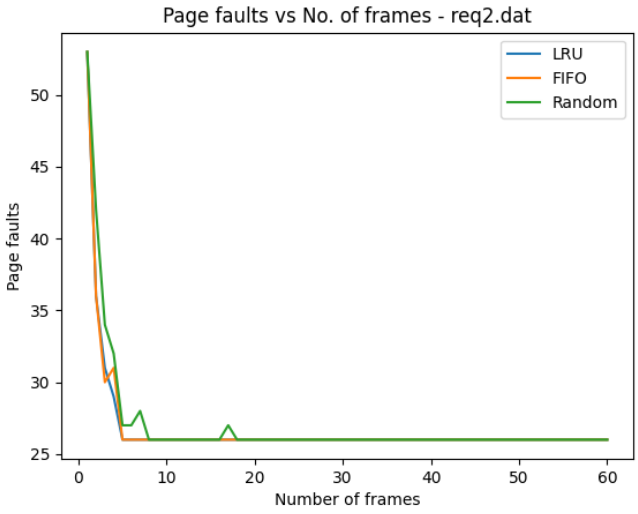

Figure 2: req2.dat

Analysis

- Belady's anomaly is a phenomenon where increasing the frame size would increase the number of page faults, in contrast to what we would usually expect to happen.
- It occurs in replacement policies which do not take into account the past history of memory accesses, i.e. **Random replacement policy**. Thus, a page accessed recently has a chance of being evicted before pages which were accessed long ago in the past.
- In the graph, we can observe that as the frame size increases from 6 to 8 and 16 to 18, the number of page faults is increasing due to the randomness in page eviction policy.

5.3 No locality workload - req3.dat

```
31 23 17 57 58 42 28 13 45 4 34 8 2 54 21 20 53 11 56 46 29 59 3 49 50 15 37 9 12 40 47
    41 55 38 26 16 22 5 10 36 43 51 35 25 14 7 39 18 48 30 52 1 33 6 44
```

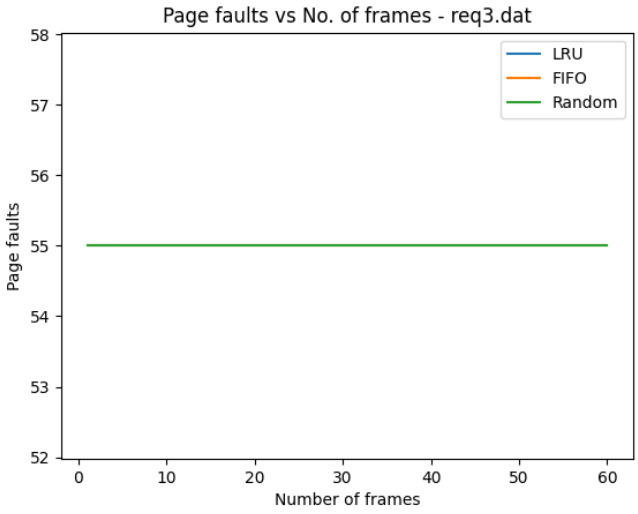

Figure 3: req3.dat

Analysis

- This is a no-locality workload which means that, each page request is to a unique page.
- From the graph, we can observe that since each page request is to a unique page, the page will not be available in cache as it was not accessed previously and this would result in a page fault. Thus, all requests are page faults and the 3 policies have a coinciding line as there is no notion of spatial or temporal locality in the pattern of page requests.

5.4 80-20 workload - req4.dat

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

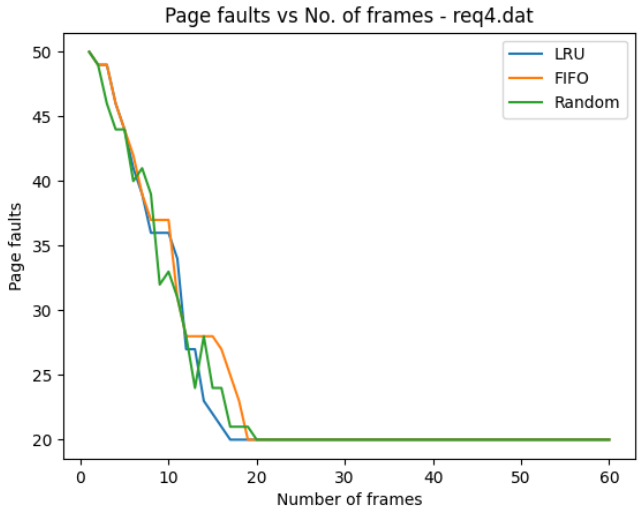

Figure 4: req4.dat

Analysis

- This simulates the 80-20 workload which means that 80% of the references will be made to 20% of the pages, and 20% of the references will be made to 80% of the pages.
- In this case, temporal locality is there meaning, we are accessing the recently accessed pages frequently.
- As we can see from the graph, LRU performs best for most of the frame sizes due to the fact that it takes into account the recency of the access of each page.

5.5 Looping sequence - req5.dat

```
1 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
```

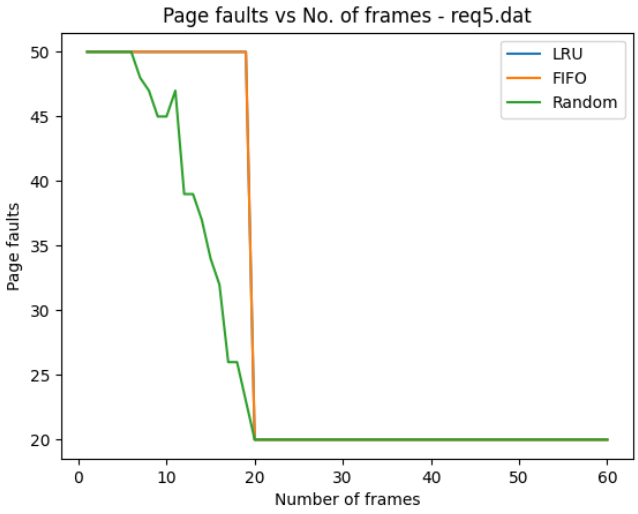

Figure 5: req5.dat

Analysis

- This looping sequence would mean that the earlier accessed pages would be accessed before the recently accessed pages. This is in contrast to the pattern of page requests in which FIFO and LRU generally perform better (i.e those patterns which exhibit temporal locality in accessing pages).
- As seen from the graph, LRU and FIFO perform similar - poorly when compared with random policy as random policy does not take into account the history of the page access pattern during eviction.
- For the first 20 requests, the pages are uniquely accessed and the looping begins after this.