CS 314 - OS Laboratory Assignment 8

Cebajel Tanan (210010055)
Tanishq Trivedi (210010056)

March 17, 2024

Contents

1 Introduction

1.1 First In First Out (FIFO)

In FIFO, the item that has been in the system the longest is the one to be replaced. Here's how it works:

- **Arrival Order:** Each item that enters the system is placed in a queue. The first item to arrive is placed at the front of the queue, and subsequent items are placed behind it.
- **Replacement Decision:** When the system needs to replace an item (for example, when a cache is full and a new item needs to be cached), the item at the front of the queue, which has been in the system the longest, is chosen for replacement.
- **Advantages:** FIFO is simple to implement and easy to understand. It ensures fairness as it treats all items equally based on their arrival order.
- **Disadvantages:** One of the main drawbacks of FIFO is its lack of consideration for the relevance or frequency of access of items. It may replace frequently accessed or critical items simply because they were the first to arrive, leading to potential inefficiencies in resource usage.

Overall, while FIFO is straightforward and fair, it might not always be the most efficient method for resource management, especially in scenarios where access patterns vary or where certain items are more critical than others.

1.2 Least Recently Used (LRU)

Here's how LRU works:

- **Tracking Usage:** The system keeps track of when each item in the cache was last accessed or used.
- **Replacement Decision:** When the cache is full and a new item needs to be added, the system identifies the item that has not been accessed for the longest period of time. This item is then replaced with the new item.
- **Advantages:** LRU is effective in evicting items that are least likely to be used again in the near future, thus maximizing cache performance by keeping frequently accessed items in memory.
- **Disadvantages:** Implementing LRU can require additional overhead to track usage information for each item in the cache. This overhead can become significant, especially in large-scale systems with numerous items in the cache.

Overall, LRU is a popular choice for cache management due to its effectiveness in minimizing cache misses and improving overall system performance by prioritizing the retention of frequently accessed data.

1.3 Random Replacement

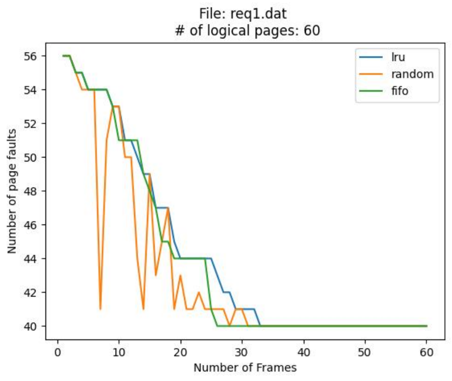
In this policy, when a cache line needs to be replaced, a randomly chosen line is evicted from the cache, regardless of its usage history or any other factors.

Here's how it works:

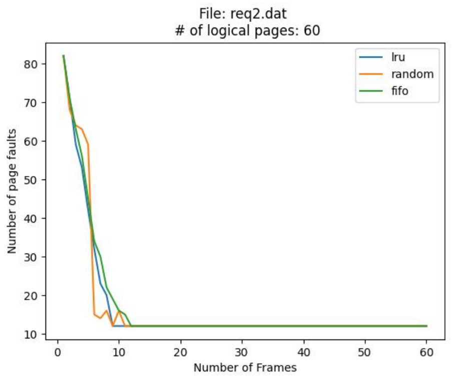
- **Random Selection:** When a cache line needs to be replaced due to a cache miss or cache eviction, the system randomly selects a line from the cache to be replaced.
- **No Consideration of Usage:** Unlike other replacement policies such as LRU (Least Recently Used) or FIFO (First-In-First-Out), the Random Replacement Policy does not consider the usage history or access patterns of the cache lines. It treats all cache lines equally in terms of their likelihood of eviction.
- **Advantages:** The main advantage of the Random Replacement Policy is its simplicity. It requires minimal computational overhead and is easy to implement.
- **Disadvantages:** One of the main drawbacks of this policy is its lack of optimality. Since cache lines are selected for replacement randomly, there's no guarantee that frequently accessed or more important data will be retained in the cache. This can lead to lower cache hit rates and potentially reduced system performance compared to more sophisticated replacement policies.

Overall, the Random Replacement Policy is a basic approach that can be suitable for scenarios where simplicity is prioritized over optimization, or when the access patterns of the data are unpredictable. However, in many cases, more advanced replacement policies such as LRU or LFU (Least Frequently Used) are preferred for better cache performance.

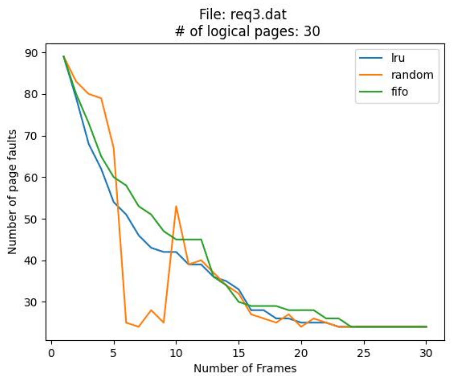
2 Figures



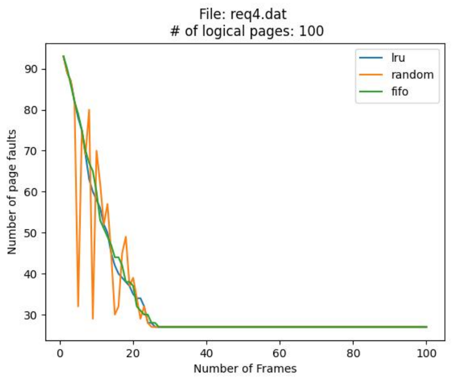
(a) Output for req1.dat



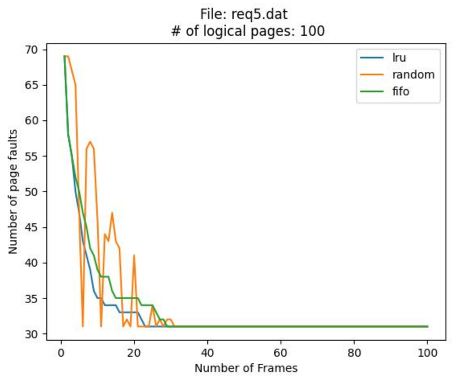
(b) Output for req2.dat



(c) Output for req3.dat



(d) Output for req4.dat



(e) Output for req5.dat

3 Analysis

As number of frames reach infinity all replacement policies converge to optimal policy with minimum possible page faults.

- “req1.dat” does not show any particular access pattern so random replacement policy works best in this scenario.
- “req2.dat” shows some temporal locality in access pattern, so, lru replacement policy has an edge over fifo and random in this scenario.
- “req3.dat” shows good temporal locality in access pattern, so, lru replacement policy works best in this scenario.
- “req4.dat” shows distribution of page accesses is such a way that the pages being replaced by FIFO are similar to the pages that would be replaced by LRU. So, lru replacement policy or fifo works best in this scenario.
- “req5.dat” shows good temporal locality in access pattern, so, lru replacement policy works best in this scenario.

4 Conclusion

We have successfully implemented different replacement policies along with generating outputs and the analysis of the performance of each policy.

5 Appendix

5.1 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
```

5.2 req2.dat

```
 14 14 19 16 18 13 12 14 15 15 18 16 13 12 12 13 17 13 9 17 16 13 13 19 17 12 12 16
17 15 15 17 17 15 16 16 14 13 17 10 12 17 12 15 16 16 20 19 14 15 16 16 15 17 17 14
13 11 12 15 14 16 15 13 17 13 15 14 15 13 13 14 16 16 11 17 16 17 13 16 15 15 14 16
13 15 15 15 14 17 14 15 15 14 19 20 16 12 13 13
```

5.3 req3.dat

```
 3 5 1 5 5 5 3 5 11 3 14 5 11 16 5 9 3 30 24 3 3 5 25 3 7 7 28 7 3 5 3 10 12 5 12 25
21 3 6 13 5 5 11 3 28 2 5 5 24 12 3 5 12 12 30 16 22 18 9 5 3 5 5 25 3 21 6 5 8 5 6
5 27 2 5 7 3 16 3 22 5 5 5 11 3 19 8 11 5 5 1 18 22 19 5 20 5 11 3 5
```

5.4 req4.dat

```
 23 18 11 7 24 20 7 1 11 1 29 1 4 11 22 17 8 15 8 29 11 7 1 16 3 8 27 19 10 8 4 21
9 23 8 15 13 4 5 28 22 16 12 23 8 2 10 5 8 2 21 21 21 27 28 10 15 21 29 16 22 20 20
3 3 23 2 8 16 29 29 22 30 22 4 29 23 5 17 14 7 21 23 19 1 17 17 7 22 15 4 17 19 23
9 9 7 5 9 24
```

5.5 req5.dat

```
 43 15 3 2 20 5 51 29 3 5 5 3 5 3 3 5 5 3 5 27 3 17 3 3 58 5 54 37 3 5 26 5 3 1 14
4 7 42 32 5 46 3 12 3 59 32 5 15 9 19 55 5 5 3 5 40 15 3 3 3 59 29 44 21 8 5 5 3 9
40 5 41 46 3 41 3 5
```