**R. V. College of Engineering,**

Bangalore 560059

(Autonomous Institution Affiliated to VTU, Belgaum)



**Page Replacement Algorithms**

**CS235AI**

Operating Systems

Submitted to:

**Assistant Prof. Dr. Pavithra H.**

Submitted By:

|  |  |  |
| --- | --- | --- |
| **USN** | **Name** | **E-Mail** |
| 1RV22CS014 | Akash M Tambake | [akashmtambake.cs22@rvce.edu.in](mailto:akashmtambake.cs22@rvce.edu.in) |
| 1RV22CS019 | Amol Sahu | [amolsahu.cs22@rvce.edu.in](mailto:amolsahu.cs22@rvce.edu.in) |
| 1RV22CS023 | Anirudh S | [anirudhs.cs22@rvce.edu.in](mailto:anirudhs.cs22@rvce.edu.in) |
| 1RV22CS034 | Aviral Singh | [aviralsingh.cs22@rvce.edu.in](mailto:aviralsingh.cs22@rvce.edu.in) |

**Performance Comparison of Page Replacement Algorithms: FIFO, LRU, LFU, and Second Chance**

**Abstract**

Efficient virtual memory management relies heavily on effective page replacement algorithms. This paper analyzes and compares the performance of four common algorithms: **First-In-First-Out (FIFO), Least Recently Used (LRU), Least Frequently Used (LFU)** and **Second Chance (SC)**. We will evaluate their performances using key metrics like page faults and hit ratios under various access patterns, including random, sequential, and working set models. The goal is to understand the strengths and weaknesses of each algorithm to find out the choice of the most suitable approach for different workload scenarios.

**1.Introduction**

Modern operating systems utilize virtual memory to create the illusion of having more RAM than physically available. When a program requests a page not currently in memory, a page fault occurs. The operating system then employs a page replacement algorithm to evict an existing page from memory, making space for the requested one. This decision significantly impacts system performance, as page faults trigger slower data retrieval from secondary storage (e.g. SSDs and hard drives).

This paper delves into the performance comparison of four prominent page replacement algorithms:

* **FIFO (First-In-First-Out):** This algorithm evicts the oldest loaded page, regardless of its recent usage. While simple to implement, it can lead to suboptimal performance under certain access patterns.
* **LRU (Least Recently Used):** LRU prioritizes recently used pages, evicting the page that has not been accessed for the longest time. This strategy leverages the principle of \*temporal locality of reference, where recently accessed pages are more likely to be accessed again soon.

\*temporal locality of reference: If at one point a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future

* **LFU (Least Frequently Used):** LFU focuses on access frequency, evicting the page with the lowest overall access count. This approach can be beneficial for workloads with a mix of frequently and infrequently accessed pages.
* **Second Chance (SC):** SC introduces a twist on FIFO. It maintains a reference bit for each page, initially set to 0. When a page is accessed, its reference bit is set to 1. During eviction, FIFO chooses a candidate for replacement. If the candidate's reference bit is 1, SC gives it a "second chance" by resetting the bit to 0 and skipping eviction. This allows potentially useful pages, even if loaded earlier, to remain in memory if recently accessed.

**2.Performance Metrics**

The performance of these algorithms will be evaluated on the basis of the following metrics:

* **Page Faults:** As mentioned earlier, page faults occur when a requested page is not found in physical memory. Minimizing page faults is crucial for efficient system performance.
* **Hit Ratio:** The hit ratio represents the percentage of memory accesses that are satisfied by finding the requested page in physical memory, indicating faster processing without requiring data retrieval from secondary storage.

**3.Data and Methodology**

The effectiveness of a page replacement algorithm depends on the workload it handles. To comprehensively evaluate their performance, we will utilize data sets representing different access patterns:

* **Random Access Pattern:** This scenario simulates situations where memory accesses occur randomly, with no specific order or locality.
* **Sequential Access Pattern:** This scenario reflects workloads where memory accesses follow a sequential order, where a page is likely to be followed by its neighbors.
* **Working Set Model:** This model represents a scenario where a program exhibits a specific locality of reference, accessing a limited set of pages for a certain period.

We will employ a simulation tool to model the behavior of each algorithm under these access patterns. The simulation will track page faults and hit ratios, for various memory size configurations (provide specific memory sizes you will simulate).

**4.Results**

This section will present the results of the performance evaluation, analyzing how each algorithm performs under different workload scenarios using the defined metrics.

**4.1 Performance under Random Access Patterns:**

We expect FIFO to exhibit a high number of page faults due to its inability to exploit any access locality. Pages get evicted based on their arrival time, not their usage. This can lead to situations where recently used pages are evicted to make space for older ones that haven't been used in a long time. LRU, on the other hand, should perform better by keeping recently used pages in memory, leading to a higher hit ratio and potentially faster execution times. LFU's performance under random access patterns might be less predictable compared to LRU. LFU considers overall access frequency, but in a scenario of random access, this might not accurately reflect which pages are likely to be used soon. SC can offer some improvement over FIFO under random access. The reference bit mechanism in SC can potentially prevent eviction of recently used pages that were loaded earlier, leading to a slightly better hit ratio compared to FIFO.

**4.2 Performance under Sequential Access Patterns:**

In this scenario, all four algorithms are likely to perform well. FIFO can benefit from the sequential nature of accesses, keeping the most recently used pages (which in a sequential pattern tend to be the ones used most frequently) in memory. LRU and LFU should also exhibit high hit ratios due to the predictable access pattern. As pages are accessed sequentially, they are less likely to be replaced before being used again. SC's reference bit mechanism can further enhance performance in this scenario. Pages accessed sequentially will have their reference bits set, and SC is more likely to retain these pages in memory compared to a strict FIFO approach.

**4.3 Performance under Working Set Model:**

The working set model represents a workload where a program exhibits a specific locality of reference, accessing a limited set of pages for a certain period. This scenario favors algorithms that can effectively capture this access locality. LRU is well-suited for this workload as it prioritizes recently used pages. Pages within the working set are likely to be referenced frequently, and LRU keeps them in memory, minimizing page faults.

However, LFU might not perform as well under the working set model. While it considers access frequency, it doesn't account for the recency of access. Pages within the working set might be accessed frequently, but not necessarily the most frequently accessed pages overall. LFU could potentially evict a relevant page within the working set to keep a less frequently used page but with a higher overall access count.

SC can offer a balance between LRU and FIFO in the working set model. It leverages the reference bit mechanism to potentially save recently used pages within the working set from eviction, even if they were loaded earlier than other frequently accessed pages. This can improve performance compared to FIFO, but might not achieve the optimal hit ratio of LRU.

**4.4 Impact of Memory Size:**

The impact of memory size on page faults can vary depending on the algorithm. With a larger memory size, all algorithms might experience fewer page faults as there's more space to accommodate frequently accessed pages. This can lead to a smaller performance gap between the algorithms. However, the relative performance difference between the algorithms might become less pronounced with abundant memory. LRU's advantage in exploiting temporal locality might be less significant when there's enough physical memory to hold most recently used pages. Similarly, SC's benefit of giving a second chance to potentially useful pages might become less crucial with ample memory.

4.5 Test Cases

Reference String: 3 2 1 3 4 1 6 2 4 3 4 2

Frames:3

FIFO algorithm with three frames for distribution

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pages | 3 | 2 | 1 | 3 | 4 | 1 | 6 | 2 | 4 | 3 | 4 | 2 |
| Frame 1 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 3 | 3 | 3 |
| Frame 2 | - | 2 | 2 | 2 | 2 | 2 | 6 | 6 | 6 | 6 | 4 | 4 |
| Frame 3 | - | - | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

No. of Faults – 8 Hit Ratio: 33.33%

LRU algorithm with three frames for distribution

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pages | 3 | 2 | 1 | 3 | 4 | 1 | 6 | 2 | 4 | 3 | 4 | 2 |
| Frame 1 | 3 | 3 | 3 | 2 | 1 | 3 | 4 | 1 | 6 | 2 | 2 | 3 |
| Frame 2 | - | 2 | 2 | 1 | 3 | 4 | 1 | 6 | 2 | 4 | 3 | 4 |
| Frame 3 | - | - | 1 | 3 | 4 | 1 | 6 | 2 | 4 | 3 | 4 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

No. of Faults – 8 Hit Ratio: 33.33%

LFU algorithm with three frames for distribution

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pages | 3 | 2 | 1 | 3 | 4 | 1 | 6 | 2 | 4 | 3 | 4 | 2 |
| Frame 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| Frame 2 | - | 2 | 2 | 2 | 4 | 4 | 6 | 2 | 4 | 4 | 4 | 4 |
| Frame 3 | - | - | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

No. of Faults – 8 Hit Ratio: 33.33%

Second chance algorithm with three frames for distribution

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pages | 3 | 2 | 1 | 3 | 4 | 1 | 6 | 2 | 4 | 3 | 4 | 2 |
| Frame 1 | 3 | 3 | 3 | 3 | 3 | 3 | 6 | 6 | 6 | 3 | 3 | 3 |
| Frame 2 | - | 2 | 2 | 2 | 4 | 4 | 4 | 2 | 2 | 2 | 2 | 2 |
| Frame 3 | - | - | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 4 | 4 | 4 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

No. of Faults – 8 Hit Ratio: 33.33%

|  |  |
| --- | --- |
| Reference String: 3 2 1 3 4 1 6 2 4 3 4 2  Frames:4 | |
| **Algorithm** | **Page Faults** |
| FIFO | 7 |
| LRU | 7 |
| LFU | 7 |
| Second Chance | 8 |

|  |  |
| --- | --- |
| Reference String: 7 0 1 2 0 3 0 4 2 3 0 3 2  Frames:3 | |
| **Algorithm** | **Page Faults** |
| FIFO | 10 |
| LRU | 9 |
| LFU | 8 |
| Second Chance | 8 |

|  |  |
| --- | --- |
| Reference String: 7 0 1 2 0 3 0 4 2 3 0 3 2  Frames:4 | |
| **Algorithm** | **Page Faults** |
| FIFO | 7 |
| LRU | 6 |
| LFU | 6 |
| Second Chance | 6 |

|  |  |
| --- | --- |
| Reference String: 1 2 3 4 5 1 2 3 4 5  Frames:3 | |
| **Algorithm** | **Page Faults** |
| FIFO | 10 |
| LRU | 10 |
| LFU | 10 |
| Second Chance | 10 |

|  |  |
| --- | --- |
| Reference String: 1 2 3 4 5 1 2 3 4 5  Frames:5 | |
| **Algorithm** | **Page Faults** |
| FIFO | 5 |
| LRU | 5 |
| LFU | 5 |
| Second Chance | 5 |

|  |  |
| --- | --- |
| Reference String: 1 2 3 4 1 2 5 1 2 3 4 5  Frames:4 | |
| **Algorithm** | **Page Faults** |
| FIFO | 10 |
| LRU | 8 |
| LFU | 8 |
| Second Chance | 8 |

**5. Discussion**

This section will discuss the key findings from the performance evaluation and analyze the strengths and weaknesses of each algorithm. We will consider factors like:

* **Suitability for Different Workloads:** Based on the results, we will discuss which algorithm is most suitable for workloads with random access patterns versus those with sequential access patterns or working set behavior.
* **Trade-offs:** Each algorithm has its own trade-offs. We will discuss the implementation complexity of each approach, along with any limitations. For instance, LRU requires maintaining timestamps or access counters for all pages, which can add overhead. SC introduces the reference bit mechanism, adding complexity compared to FIFO but potentially improving performance.
* **Belady's Anomaly:** We can also explore the concept of Belady's Anomaly, which demonstrates that FIFO can exhibit worse page fault behavior than some other algorithms even with knowledge of future access patterns (which is not achievable in real systems). This reinforces the limitations of FIFO in exploiting access locality.

**6. Conclusion**

This paper compared the performance of four common page replacement algorithms: FIFO, LRU, LFU, and Second Chance. Our evaluation explored their performance under different access patterns and memory size configurations. The results highlight that the optimal choice of algorithm depends on the specific workload characteristics and available memory resources. LRU generally offers a good balance between simplicity and effectiveness, particularly for workloads with locality of reference. However, for workloads with specific access patterns, SC, FIFO, or LFU might be more suitable alternatives.

**7. References**

1. Chawan, Pramila. (2011). A Comparison of Page Replacement Algorithms. International Journal of Engineering and Technology. 3. 10.7763/IJET.2011.V3.218.
2. Irum, Isma & Raza, Mudassar & Sharif, Muhammad. (2012). File Systems for Various Operating Systems: A Review. Research journal of Applied Sciences, Engineering and Technology. 4. 2934-2947.