Chullipparambil, Chinnu Padman

Class Account: masc1445

CS 570, Summer 2014

Assignment #2, Analysis of Page Replacement Algorithms.

Filename: README

ANALYSIS OF PAGE REPLACEMENT ALGORITHMS

**Abstract:** Analysis of the following four page replacement algorithms using simulation of the page replacement function of the page fault interrupt handler routine for a paged memory management system.

1. OPT (Optimal)
2. LRU (Least Recently Used)
3. FIFO
4. CLOCK
5. **INTRODUCTION**

Goal of the page replacement algorithm is to reduce page fault rate by selecting the “best” page to evict. The best pages are those that will never be used again or those that will be used in furthest in the future.

OPT (Optimal)

* Selects that page for replacement whose next use will occur furthest in the future.
* Guarantees the lowest possible page-fault rate (optimal) for a fixed number of frames.
* Impossible to know future events in a run time system.
* Useful as a “yardstick” to compare the performance of other (implementable) algorithms against.

LRU (Least Recently Used)

* Replaces the page that has not been referenced for the longest time.
* By the principle of locality, this should be the page least likely to be referenced in the near future. Hence LRU gives a good approximation to the optimal algorithm.
* Though theoretically realizable, expensive to implement in practice.

FIFO

* When a page must be replaced, the oldest page is chosen. Throw out pages in the order that they were allocated.
* Simple to implement.
* Performs poorly in practical applications. A page which is still in constant use may get replaced.

CLOCK

* Keeps a circular list of pages in memory and when a frame is needed, the pointer (hand on the clock) advances until it finds a page with use bit=0 and replaces it. As it advances it clears the use bits giving a second chance to each page. Thus a page that is given a second chance will not be replaced until all other pages are replaced or given second chances.
* An improved form of FIFO where each page is given a second chance.
* Approximates aging of the pages referenced/used longest ago in history.

Locality of Reference or Principle of Locality

* Temporal locality: Memory accessed recently tends to be accessed again soon
* Spatial locality: Memory locations near recently-accessed memory is likely to be referenced soon

1. **ANALYSIS USING SIMULATION**

**Data Set 1:**

Random numbers generated between 1-100, 1-50 and 1-25.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Page references** | **Frame #** | **OPT** | **LRU** | **CLOCK** | **FIFO** |
| **100 numbers between 1-100** | 5 | 82 | 95 | 95 | 95 |
| 10 | 72 | 90 | 91 | 89 |
| 15 | 64 | 86 | 86 | 84 |
| **100 numbers between 1-50** | 5 | 64 | 83 | 83 | 84 |
| 10 | 51 | 73 | 75 | 74 |
| 15 | 43 | 64 | 62 | 63 |
| **100 numbers between 1-25** | 5 | 54 | 75 | 73 | 76 |
| 10 | 35 | 58 | 54 | 51 |
| 15 | 28 | 41 | 46 | 45 |
| 20 | 25 | 32 | 35 | 29 |

**Observations:**

* Each sample had more repetitions compared to previous one. Observed that as the repetitions increases, no of page faults decreases. 100 random page numbers generated between 1 and 25 had more repetitions than 100 random page numbers generated between 1 and 100. Number of pages faults is more for later dataset.
* Observed that if the page numbers are completely random with minimum repetitions, not much difference in performance among the algorithms. We can observe that in third set of data, where there are more repetitions, variations in performance begin to appear. But here still the repetitions are random and does not have locality of reference, hence for certain frame size FIFO may even perform better than others as shown above. But in real time scenarios, patterns with locality of reference is observed. Hence in the subsequent data sets, introducing certain patters are taken into consideration.

**Data Set 2:**

Reference string: 1 2 3 4 5 1 2 6 7 8 1 2 9 10 11 1 2 12 13 14 1 2 15 16 17 1 2 18 19 20

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Frame #** | **OPT** | **LRU** | **CLOCK** | **FIFO** |
| 3 | 20 | 30 | 30 | 30 |
| 5 | 20 | 20 | 20 | 24 |
| 6 | 20 | 20 | 20 | 24 |

**Observations:**

* LRU and CLOCK

When will LRU and CLOCK will choose different pages to evict? When a page already exists in a frame, in CLOCK, the pointer always moves to its next immediate frame. If a page fault occurs next, there is a chance that a page which was in between that movement, which already got a second chance may get replaced before other pages which already got second chance in the current clock cycle. In LRU this will never happen. It will always evict the page which was least recently used.

But in the above reference string, and repeated numbers always comes in the same order and same interval and hence the in between jumps within a clock cycle will not occur. Hence with the above data set, LRU and CLOCK will always yield same number of page faults.

* FIFO

Unlike data set 1, here we can observe that when data set reflects locality of reference FIFO performs worse than LRU and CLOCK. FIFO does not consider locality of reference, hence not suitable for real time scenarios.

**Data Set 3:**

Reference string: 7 0 1 1 2 0 3 0 2 4 2 3 0 3 2 1 2 0 1 7 2 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Frame #** | **OPT** | **LRU** | **CLOCK** | **FIFO** |
| 3 | 15 | 22 | 23 | 27 |
| 5 | 9 | 9 | 13 | 15 |

**Observations:**

* LRU and CLOCK

We can observe that for frame size 5, LRU gives performance as good as OPT for the given reference string. CLOCK has more page faults than LRU and this is because more jumps to already existing pages occurs during clock cycles and this does not ensure page replacement as efficient as LRU.

* FIFO

Again with this data set, FIFO performs worse when pattern reflects locality of reference. We can observe its performance compared to the optimal case. For instance for frame size 3, if we ignore the first 3 page faults, which every algorithm must suffer, number of page faults in FIFO is double than that of OPT.

**Data Set 4:**

Belady’s Anomaly

For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases. We would expect that giving more memory to a process would improve its performance. This assumption may not always be true.

Reference String: 1 2 3 4 1 2 5 1 2 3 4 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Frames #** | **OPT** | **LRU** | **CLOCK** | **FIFO** |
| 2 | 9 | 12 | 12 | 12 |
| 3 | 7 | 10 | 10 | 9 |
| 4 | 6 | 10 | 10 | 10 |
| 5 | 5 | 5 | 5 | 5 |
| 6 | 5 | 5 | 5 | 5 |

For FIFO, the number of faults for four frames (10) is greater than the number of faults for three frames (9). Thus FIFO suffer from Belady’s Anomaly.

OPT and LRU belong to a class of page replacement algorithms called stack algorithms that can never exhibit Belady’s anomaly. A stack algorithm is an algorithm for which it can be shown that the set of pages in memory for n frames is always a subset of the set of pages that would be in memory with n+1 frames. For example, in LRU, the set of pages in memory would be n most recently referenced pages. If the number of frames increased to n+1, these n pages will still be a subset of those n+1 most recently referenced pages and so will still be in memory.

Clock algorithm may suffer from Belady’s Anomaly as it degenerates to FIFO replacement if all use bits are set.

1. **Summary**

We expect to see patterns based on principle of locality in real programs. Hence LRU will perform better and would approximate OPT compared to FIFO and CLOCK as it always evicts least recently used page. CLCOK will have a slight low performance than LRU. Though it takes recent use into consideration, it will not work as efficient as LRU. The pointer jumps back to the frame next to the recently used page ensuring the page just referenced is furthest from the pointer. But the pages in between that movement may lose the fair chance to get replaced only after other candidates in the current clock cycle are replaced if a page fault occurs next. Whereas in LRU, least recently used page will always get evicted. But CLOCK implementation is simpler than that of LRU. Since FIFO does not take locality of reference into consideration, it will give the least performance among these algorithms in real time scenarios.

**References:**

Operating System Concepts, Silberschats, Galvin, Gange

Modern Operating Systems, Tanenbaum

<http://www.eecs.harvard.edu/~mdw/course/cs161/notes/pagerepl.pdf>