For this assignment we had to implement four virtual memory page replacement algorithms using a simulation of page requests. The four algorithms are optimum, clock, first-in-first-out (FIFO), and not recently used (NRU). After debugging and ensuring that the implementations were working in an acceptable manner, the last task was to compare the performance of the 3 algorithms against the optimal situation and to check if we could find any instances of Bélády's anomaly for the FIFO algorithm.

Optimal

The optimal page replacement algorithm that I implemented begins by preprocessing the file. It does this by not only maintaining the read order, using a linked list of nodes, but also individual linked lists for each instance of a page and its associated offsets which are all stored in hash map for quick access. During a simulated page request, the program “reads” the next memory address by traversing the main linked list. It then removes an individual node from the individual “page” linked listed from the hash map. The first node of the “page” linked listed is also checked whenever a fault occurs by looking at its access number (essentially the line of the file it was read from). This peek is done for every “entry” currently in the page table (PT). If a given page table entry (PTE) doesn’t have any more nodes in its associate “page” linked list, or the value it finds is the largest among all of the PTEs, then that page will be evicted from the PT and a new PTE added in its place.

The hardest part of this implementation this algorithm was figuring out the logic and how to efficiently “look into the future.” From a practical standpoint, the “optimal” algorithm is basically impossible to use in the real world. All of the preprocessing necessary to minimize the number of page faults makes it impractical to use except in learning. See Table 1 for the associated testing data obtained using the three test files.

Clock

The clock algorithm was by far the best of the tree practical algorithms. In all twelve test cases, it was only beaten by the optimal algorithm. The superiority of this algorithm over FIFO and NRU is most evident in the tests cases where there were only 8 frames. In those scenarios, it generally produced several thousand less page faults. This would suggest that the Clock would be ideal in systems with limited amount of memory since it consistently produces the best results.

The clock was also incredibly simple to implement using a circular linked list. Given its ease of implementation and its relatively good performance, it seems the most likely of the three to be used in an actual operating system. See Table 1 for the list of page fault data collected during testing.

First-In-First-Out (FIFO)

The FIFO algorithm is the simplest of the three and operates the same way that a traditional queue works. The first PTE into the PT is the first one that is evicted whenever a fault occurs. While easy to implement, it is the least intelligent of the algorithms and offers benefit only by chance. In the twelve tests, it was the worst in all but four.

During testing, I also ran the FIFO algorithm using eight additional page frame sizes ranging from 2 to 100. The results of my observations are in Table 2. While the overall trend indicated that increasing the number of page frames leads to better performance (i.e. less faults), there can be instances where increasing the number of frames actually creates a higher number of faults. This phenomenon is called Bélády's anomaly. During my extra testing I did not notice the phenomenon, but none the less it still can occur in certain situations and makes FIFO the worst candidate for a real-world paging system.

Not Recently Used (NRU)

The NRU was the more complex of the three practical algorithms and relies on grading pages into four classes based in information contained in their dirty and reference bits. The four classes are then used to decide which PTE to evict in the event of a fault. This algorithm also periodically scans the entire page table at a predetermined refresh rate, in order to set the reference bit of every entry to false. It does this extra stop to ensure that as PTE age, they eventually fall into the lower classifications and can evicted first, ensuring that more recently accessed pages are kept in the PT.

The benefit of the periodic refresh rate isn’t universal and seems to be more file specific. While testing different refresh rates in order to find the best one, I used the gcc.trace file with frame sizes of eight, sixteen, thirty-two, and sixty-four in order to obtain the optimal refresh rates listed in Table 1. I have included a graph (Graph 1), of the data I obtained from using a frame size of 8. The impact of varying the refresh rate on the performance of this algorithm is most noticeable when compared against the FIFO results for the gcc and swim files and is more noticeable for the lowest frame size.

The data in Table 1 shows that overall NRU is the second best of the three. However, given its complexity and the overhead it requires, it would not make an ideal page replacement algorithm for a real operating system

Table 1. Comparison of Page Faults for Four Different Page Replacement Strategies

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File** | **Frames** | **Optimal** | **Clock** | **FIFO** | **NRU** | **Refresh** |
| Bzip | 8 | 18251 | 34926 | 47828 | 51554 | 19 |
| Bzip | 16 | 2427 | 3416 | 3820 | 23180 | 57 |
| Bzip | 32 | 1330 | 2119 | 2497 | 2835 | 160 |
| Bzip | 64 | 527 | 1280 | 1467 | 1958 | 520 |
| Gcc | 8 | 118480 | 168290 | 205368 | 175329 | 19 |
| Gcc | 16 | 80307 | 115521 | 138539 | 124480 | 57 |
| Gcc | 32 | 55802 | 83859 | 98067 | 89868 | 160 |
| Gcc | 64 | 38050 | 57682 | 70315 | 63052 | 520 |
| Swim | 8 | 171244 | 278026 | 330893 | 299361 | 19 |
| Swim | 16 | 78312 | 160177 | 214295 | 174881 | 57 |
| Swim | 32 | 28826 | 46636 | 81638 | 60407 | 160 |
| Swim | 64 | 14289 | 21380 | 30422 | 30518 | 520 |

Table 2. Various Fault Counts for the FIFO Algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| **Frames** | **Bzip** | **Gcc** | **Swim** |
| 2 | 228838 | 460912 | 541458 |
| 8 | 47828 | 205368 | 330893 |
| 16 | 3820 | 138539 | 214295 |
| 25 | 3188 | 110793 | 115014 |
| 32 | 2497 | 98067 | 81638 |
| 35 | 2162 | 93505 | 71546 |
| 47 | 1719 | 80851 | 46908 |
| 50 | 1667 | 78462 | 42895 |
| 64 | 1467 | 70315 | 30422 |
| 75 | 1276 | 64560 | 25335 |
| 98 | 1032 | 54835 | 20185 |
| 100 | 1026 | 54194 | 19903 |

Graph 1. Scatterplot of Faults vs Refresh Rates for the NRU Algorithm Using the gcc.trace File.

