Performance of Page Replacement Algorithms

by: Alec Braynen, Matthew Kramer, Nghiem Ngo, and Sterling Price

Abstract

In this project, we implement five page replacement algorithms: first-in first-out, least recently used, least frequently used, most frequently used, and optimal. This is done by simulating how the page table and memory trace interact through code in the C++ programming language. Using a variety of data structures, we were able to compare the performance of each algorithm and determine the cases in which each was strongest and weakest.

Introduction

The primary goal of this project was to design and implement a simulator that will take in a memory trace as input, simulate the actions of a system using virtual memory with a page table, and output the number of page faults, number of page hits, and the hit and miss ratios. The simulator should ideally be able to keep track of what pages are loaded in memory. As the simulator processes each page of the memory trace, it should first check to see if the page is currently loaded into memory already. If the page to be inserted is not currently loaded into memory, then one of the page replacement algorithms is used to select the page to be replaced. If the page to be replaced is dirty - has previous accesses that involved writing - then it must be saved to the disk. Lastly, the new page is loaded into memory from disk before the page table is updated. In our project, however, we were just required to simulate the replacement of pages, not including the dirty bit of each frame or whether pages must be read from the disk.

First-In First-Out [FIFO]

Integers were created to store the number of hits and misses. Additionally, a vector was created to simulate the virtual memory. The vector for virtual memory in this case acts as a stack. Using a for loop, the data in the array of the simulated memory is iterated through and put into the virtual memory vector. Before being placed into the virtual memory, the program checks to see if the data being placed into the virtual memory is already there; if it is there, the hits integer is incremented, and the data is removed from the main memory vector. If the data isn't in the virtual memory vector and the virtual memory vector is not full, the data is placed into the virtual memory vector and the miss integer is incremented. Finally, if the virtual memory vector is full and the data being placed in the virtual memory vector from main memory is not in the virtual memory vector, the first data out of all the data currently in the virtual memory vector is replaced with the new data from main memory.

Least Recently Used [LRU]

Integers were created to store the number of hits and misses. Additionally, a vector was created to simulate the virtual memory. Each virtual memory block also has a counter that keeps track of how recently that block of memory was accessed. Using a for loop, the data in the array of the simulated memory is iterated through and put into the virtual memory vector. Before being placed into the virtual memory, the program checks to see if the data being placed into the virtual memory is already there; if it is there, the hits integer is incremented, the count that keeps track of if the block was recently accessed is incremented, and the data is removed from the main memory array. If the data isn't in the virtual memory vector and the virtual memory vector is not full, the data is placed into the virtual memory vector, the miss integer is incremented and the counter for recently accessed is incremented. Finally, if the virtual memory vector is full and the data being placed in the virtual memory is not currently there, the memory block with the least recently accessed counter is replaced with the data from the main memory array and the counter is updated to show that the block was the most recently accessed.

Least-Frequently Used [LFU]

Integers were created to store the number of hits and misses. Additionally, an array was created to simulate the virtual memory. Each virtual memory block also has a counter that keeps track of how often that block of memory was accessed. Using a for loop, the data in the array of the simulated memory is iterated through and put into the virtual memory vector. Before being placed into the virtual memory, the program checks to see if the data being placed into the virtual memory is already there; if it is there, the hits integer is incremented, the count that keeps track of how often the block was accessed is incremented, and the data is removed from the main memory array. If the data isn't in the virtual memory vector and the virtual memory vector is not full, the data is placed into the virtual memory vector, the miss integer is incremented and the counter for frequency of access is incremented. Finally, if the virtual memory vector is full and the data being placed in the virtual memory is not currently there, the memory block with the least frequently accessed counter is replaced with the data from the main memory array and the counter is reset.

Most Frequently Used [MFU]

Integers were created to store the number of hits and misses. Additionally, a vector was created to simulate the virtual memory. Each virtual memory block also has a counter that keeps track of how often that block of memory was accessed. Using a for loop, the data in the array of the simulated memory is iterated through and put into the virtual memory vector. Before being placed into the virtual memory, the program checks to see if the data being placed into the virtual memory is already there; if it is there, the hits integer is incremented, the count that keeps track of how often the block was accessed is incremented, and the data is removed from the main memory array. If the data isn't in the virtual memory vector and the virtual memory vector is not full, the data is placed into the virtual memory vector, the miss integer is incremented and the counter for frequency of access is incremented. Finally, if the virtual memory vector is full and the data being placed in the virtual memory is not currently there, the memory block with the most frequently accessed counter is replaced with the data from the main memory vector and the counter is reset.

Optimal

Integers we're created to store the number of hits and misses. Additionally, a vector was created to simulate the virtual memory. Each virtual memory block also has an index that correlates with the index of identical data in the main memory array. Using a for loop, the data in the array of the simulated memory is iterated through and put into the virtual memory vector. Before being placed into the virtual memory, the program checks to see if the data being placed into the virtual memory is already there; if it is there, the hits integer is incremented, the index of the memory block is matched to the index of the next block in main memory with identical data (or set to infinity if it is not in the main memory array), and the data is removed from the main memory array. If the data isn't in the virtual memory vector and the virtual memory vector is not full, the data is placed into the virtual memory vector, the miss integer is incremented and the index of the memory block is matched to the index of the next block in main memory with identical data (or set to infinity if it is not in the main memory array.) Finally, if the virtual memory vector is full and the data being placed in the virtual memory is not currently there, the memory block with the largest index value is replaced with the data from the main memory vector and the index of the memory block is matched to the index of the next block in main memory with identical data (or set to infinity if it is not in the main memory array.)

Results

```
Please enter the number of frames: 3

1 2 3 4 2 1 5 6 2 1 2 3 7 6 3 2 6

###################################

  First-In First-Out

###################################

1 : 1 X X
2 : 2 1 X
3 : 3 2 1
4 : 4 3 2
2 : 4 3 2
1 : 1 4 3
5 : 5 1 4
6 : 6 5 1
2 : 2 6 5
1 : 1 2 6
2 : 1 2 6
3 : 3 1 2
7 : 7 3 1
6 : 6 7 3
3 : 6 7 3
2 : 2 6 7
6 : 2 6 7

Page Faults: 13
Page Hits: 4
Hit Ratio: 23.5294%
Miss Ratio: 76.4706%
```

```
###################################

  Least Recently Used

###################################

1 : 1 X X
2 : 2 1 X
3 : 3 2 1
4 : 4 3 2
2 : 2 4 3
1 : 1 2 4
5 : 5 1 2
6 : 6 5 1
2 : 2 6 5
1 : 1 2 6
2 : 2 1 6
3 : 3 2 1
7 : 7 3 2
6 : 6 7 3
3 : 3 6 7
2 : 2 3 6
6 : 6 2 3

Page Faults: 13
Page Hits: 4
Hit Ratio: 23.5294%
Miss Ratio: 76.4706%
```

```
###################################

  Least Frequently Used

###################################

1 : 1 X X
2 : 2 1 X
3 : 3 2 1
4 : 4 3 2
2 : 4 3 2
1 : 1 4 2
5 : 5 1 2
6 : 6 5 2
2 : 6 5 2
1 : 1 6 2
2 : 1 6 2
3 : 3 1 2
7 : 7 3 2
6 : 6 7 2
3 : 3 6 2
2 : 3 6 2
6 : 3 6 2
```

Page Faults: 12
Page Hits: 5
Hit Ratio: 29.4118%
Miss Ratio: 70.5882%

```
###################################

  Most Frequently Used

###################################

1 : 1 X X
2 : 2 1 X
3 : 3 2 1
4 : 4 3 2
2 : 4 3 2
1 : 1 4 3
5 : 5 1 4
6 : 6 5 1
2 : 2 6 5
1 : 1 2 6
2 : 1 2 6
3 : 3 1 6
7 : 7 3 1
6 : 6 7 3
3 : 6 7 3
2 : 2 6 7
6 : 2 6 7
```

Page Faults: 13
Page Hits: 4
Hit Ratio: 23.5294%
Miss Ratio: 76.4706%

```
###################################

  Optimal

###################################

1 : 1 X X
2 : 2 1 X
3 : 3 2 1
4 : 4 2 1
2 : 4 2 1
1 : 4 2 1
5 : 5 2 1
6 : 6 2 1
2 : 6 2 1
1 : 6 2 1
2 : 6 2 1
3 : 3 6 2
7 : 7 3 6
6 : 7 3 6
3 : 7 3 6
2 : 2 7 6
6 : 2 7 6

Page Faults: 9
Page Hits: 8
Hit Ratio: 47.0588%
Miss Ratio: 52.9412%
```

Conclusion

Through the use of vectors, stacks, queues, counters, indices and a little thinking, the page replacement algorithms - first-in first-out, least recently used, least frequently used, most frequently used, and optimal - were successfully implemented and provided a clear portrait of their performance within the operating system. While the optimal page replacement algorithm proved to be the best performing, its use is simply theoretical, as it is impossible to implement in the real world.