Performance of Page Replacement Algorithms

Justin Findley, Karshan Arjun, Maharshi Shah, Hitarthi Shah

Abstract- The purpose of this project is to write a memory management simulator that uses paging and measure the performances of page replacement algorithms such as FIFO (first in first out), LRU (least recently used), LFU (least frequently used), MFU (most frequently used), and OPT (optimal page replacement). We kept track of what pages were being loaded and we increment a counter to keep track of disk reads/writes when a simulated disk read/write must occur. We observed that the performance of all these page replacement algorithms depend on the situation that they are handling.

I. Introduction

To evaluate how applications respond to a variety of page replacement algorithms such as FIFO, LRU, LFU, MFU, and OPT, we were required to build a simulator that is able to read a memory trace as well as stimulating the action of a virtual memory system with a single level page table. This simulator keeps track of what pages are loaded into memory and and as it processes each memory event from the trace, it checks to see if the corresponding page is loaded. If it is not it, it chooses a page to remove from memory. Any pages considered “dirty”, it is saved to the disk. Once the new page is loaded in memory from disk, the page table is updated.

II. Methodology

FIFO: This is the simplest page replacement algorithm. The implementation of this algorithm is done by creating a vector to simulate virtual memory. The vector acts as a stack. The pages in the array of the simulated memory is iterated using a for loop and is transferred to the virtual memory. During this process, the program checks if the page being placed already exists in the virtual memory. If it does, the hit counter is incremented and the page is removed from the main memory vector. If the page is not present in the virtual memory vector and the vector is not full, it is placed into the virtual memory vector and the miss counter is incremented. If the virtual memory vector is full, the first page out of all the page currently present in the virtual memory vector is replaced with the new page from main memory.

LRU: Creating the LRU method was done by using the same implementation of the FIFO method by using a vector to store and to keep track of the data. Once the data is already in the vector, the hits are then calculated if the integer is already in there. If the integer is not in the vector, then it is added to the next available spot, this is where the misses calculations are conducted. Another operation of the method, is solving the problem of a full vector that does not have the current integer inside of it. This is where the least added integer is then replaced with the new integer. Once all the numbers are process the hits and misses are displayed with the LRU page faults counts.

LFU: Creating the LFU method is done by implementation of a vector to store and to keep track of data. We used a for loop to iterate through the data and before being placed, our program calculates the hits integers and a counter is incremented if the integer is already there. If the integer is not there and the vector is not yet filled all the way, then the miss counter is incremented and the data is placed into the vector. If the vector is full then the data being placed in the vector is

not there, then the memory block with the least frequently used integer is replaced with the data from the main memory.

MFU: Most Frequently Used (MFU) algorithm assumes that the most frequently used page will not be needed immediately and hence, will be replaced. Implementing this algorithm is done using a vector that acts as a virtual memory block that can keep track of the times the memory was accessed. The number of hits and misses are initialized to zero and stored as counters. Before entering the page value into the vector, the algorithm checks if that value is already present in the array. If so, the number of hits is incremented by one. If that value is not present, a page miss has occurred and that counter is incremented. This, in turn, also increases the memory access counter by one. This is an important step as the memory access counter helps make the decision for page replacement once the vector is full and a page miss has occurred. If this condition is true, the value of the most recently used page value is replaced and the counter is set to zero. This process is iterated through the vector and the simulation is complete once all values have been dealt with.

Optimal: This replacement algorithm works on the idea that the page to be replaced is the one that will not be used for the longest period of time. Theoretically, this replacement algorithm has the best performance among all other algorithms as it does not suffer from Belady's anomaly. A vector acts as a block of memory and the counter for page hits and misses is initialized. All memory blocks have an index along with all data values that are repeated in the memory array. If the value being inserted in memory is already present, the hit counter is incremented by one. The indices of the memory block and the next block in memory are matched that have the identical data and that data value is deleted from main memory. If the value is not present in memory, a page miss has occurred. If the memory block is full and a page miss occurs, the memory block index with the largest value is replaced with the value from the memory vector and the index of the memory block is matched to the index of the next block in memory with identical data. This replacement algorithm requires future knowledge of the memory page values that need to be placed in memory and hence, not feasible for actual implementation in an Operating System.

III. Results

To evaluate the performance of these five algorithms, we chose three random string lengths and two random frame numbers. The following tables show the performance of each algorithm relative to the string lengths and the number of frames in the page table:

- The string length chosen for the below table is 10.

	Frames: 3		Frames: 7	
Algorithm	Miss ratio	Hit ratio	Miss ratio	Hit ratio
FIFO	40 %	60 %	60 %	40 %
LRU	40 %	60 %	60 %	40 %
LFU	40 %	60 %	60 %	40 %
MFU	50 %	50 %	60 %	40 %
Optimal	40 %	60 %	60 %	40 %

- The string length chosen for the below table is 15.

	Frames: 3		Frames: 7	
Algorithm	Miss ratio	Hit ratio	Miss ratio	Hit ratio
FIFO	80 %	20 %	60 %	40 %
LRU	80 %	20 %	60 %	40 %
LFU	67.7 %	33.3 %	60 %	40 %
MFU	80 %	20 %	43.3 %	46.7 %
Optimal	60 %	40 %	43.3 %	46.7 %

- The string length chosen for the below table is 20.

	Frames: 3		Frames: 7	
Algorithm	Miss ratio	Hit ratio	Miss ratio	Hit ratio
FIFO	80 %	20 %	60 %	40 %
LRU	80 %	20 %	55 %	45 %
LFU	70 %	30 %	55 %	45 %
MFU	85 %	15 %	70 %	30 %
Optimal	55 %	45 %	45 %	55 %

IV. Conclusion

As per the results, there was a noticeable trend based on the number of frames and the length of string used in the simulator. LFU performs better than LRU when fewer number of frames are being used in the paging system. On the other hand, MFU performs better when the number frames being used are more. We can observe that Optimal page replacement algorithm has the best performance irrespective of the chosen string length and the number of frames in the page table. In general, FIFO has a high number of page faults as compared to all other algorithms. Lastly, the performance of all page replacement algorithms vary depending on factors like the number of page frames, the number of page references, etc.