**Measuring the Performance of Page**

**Replacement Algorithms on Real Traces**

Medhi Welsh -Brito

**intro**

The aim of this project is to create a simulation system that implements page replacement algorithms. This system is designed to process a memory trace and display the various steps involved, including the number of reads, writes, and traces in the input file. The simulation will depict how the virtual memory system handles the memory trace and tracks which pages are loaded into memory.

To achieve this, the program reads traces from an input file and processes one page at a time by dividing each trace by 4KB. If a page has not been loaded into memory, it is recorded and the read counter is incremented to indicate a successful page insertion. If the page is already present, it is replaced with the current page being inputted. Additionally, if the previous page was read-only (R) and the current page is a write (W), the previous operation is updated to write (W), but not vice versa.

The collected data is then used to evaluate the performance of the three different algorithms used in the simulation. The algorithms process the memory trace and display the steps involved in evaluating the pages, allowing for a performance comparison between the different algorithms used.

**methods**

These are the three algorithms I used to implement page replacements.

**first in, first out (fifo)**

In this algorithm, the page table is stored in a deque where pages are inserted sequentially based on the input trace. First, the algorithm checks if the page table is not already full and whether the current input has been loaded into memory. If the input has not been loaded, it is loaded into an empty frame, and the read counter is increased. This process continues until the table is full, and a new page is added to the deque.

The algorithm then searches through both the page table and the current input to determine if the page is already loaded into memory. If it is, the last occurrence of that page is replaced with the current page, and the algorithm continues. If a match is not found, the algorithm removes the top page in the page table from the front of the deque and loads the current input into the back of the deque. Additionally, if the top page has a write operation (W), the write counter is incremented before it is removed from the deque.

**least recently used (lru)**

Similar to the previous algorithm, this algorithm also inserts pages sequentially. However, in this case, pages are stored in a vector as the page table.

The algorithm first checks if the page table is not full and whether the page has already been loaded into memory. If the page has not been loaded, it is added to the table, and the read counter is incremented.

The algorithm then searches the page table to see if the current input has already been loaded into memory. If a match is found, the current page is swapped with the matched page. However, in this algorithm, it is necessary to track the page table and access the least recently used frame.

If a match is not found and the page table is full, the algorithm removes the least recently used page from the back of the table and inserts the new page at the top of the page table.

**segmented fifo**

In segmented fifo, the algorithm that works by dividing the physical memory into a set of fixed-size segments. The algorithm maintains a separate FIFO (First-In-First-Out) queue for each segment, where pages are added to the back of the queue when they are first loaded into memory. In our case, one buffer will implement a FIFO structure and the other will use a LRU structure.

When a new page needs to be loaded into memory, the algorithm first checks whether the segment containing the page is already loaded. If the segment is already in memory, the algorithm simply adds the new page to the back of the corresponding segment queue.

If the segment is not already in memory, the algorithm removes the oldest segment from memory, which is the one at the front of the queue with the lowest priority, to make space for the new segment. That page is moved into the secondary buffer (LRU) and the new segment is loaded into memory, and its pages are added to the back of the corresponding queue.

**results**

In most cases, LRU (Least Recently Used) is faster than FIFO (First-In-First-Out) because LRU is able to make better use of the available memory.

In FIFO, the page that is loaded first is the first page to be removed from memory when a new page needs to be loaded. This can result in a situation where important pages are removed from memory and less important pages are kept in memory, leading to more page faults and slower performance.

On the other hand, LRU works by keeping track of the pages that have been accessed most recently and removing the least recently used pages when memory needs to be freed. This ensures that the most important and frequently used pages remain in memory, resulting in fewer page faults and faster performance.

However, it's worth noting that LRU requires more processing power and memory to keep track of the access history of each page, which may not be feasible in all systems.

As for segmented FIFO, it is not necessarily better than LRU because it operates at a higher level of memory abstraction than LRU. While segmented FIFO is designed to manage memory at the segment level, LRU operates at the page level, providing finer-grained control over memory allocation.

Segmented FIFO can be useful in certain cases, such as when dealing with variable-sized pages or segments, but LRU remains a widely used and effective algorithm for managing memory in most systems. Ultimately, the choice of memory management algorithm depends on the specific needs of the system and the characteristics of the workload it is expected to handle.

Here in the following three figures you can see the difference between the algorithms using the same number of memory frames.

fifo:

Text

Description automatically generated

vms:

Text

Description automatically generated

lru:

Text

Description automatically generated