Graham Matthews
CS470
Project 4

Upon initialization, each line from the input is read in and made into a page. Each page has a process ID and page number, which are used to identify itself. Each page is added to the inputVector, which keeps the pages in the same order they were received in.

The least recently used algorithm keeps an integer counter for the number of page faults. A queue is used to keep track of the page table. The algorithm loops through the inputVector. If the pageTable queue is smaller than the specified frame size, and the page is not currently in the page table, the page is added to the page table and the page faults counter is incremented. If the page table queue is greater than or equal to the specified frame size, and the page table does not contain the page being accessed, the page table removes the page at the front of the line, the least recently used page, adds the new page to the page table and increments the page fault counter. If the page is already in the page table, it is removed from the queue and added back on, making it the most recently used page. This does not increment the page fault counter, as the page was already in the page table.

The FIFO algorithm also keeps an integer counter for the number of page faults. A queue is also used to keep track of the page table, since a queue's very nature is to add elements to the back of the queue and remove elements at the front of the queue, as the FIFO algorithm should perform. If the page table queue is less than the specified frame size, and the page is not currently in the page table, the page is added to the queue and the page faults counter is incremented. If the page table size is equal to or greater than the frame size, and the page table does not contain the page being accessed, the page at the front of the queue is removed and the page being accessed is added, and the page fault counter is incremented.

The optimal algorithm uses an integer counter for the number of page faults. A vector is used to keep track of the page table, since in the optimal algorithm, any element at any index may be removed.

If the page table vector is less than the specified frame size and the page table does not contain the page being accessed, the page is added to the page table and the page faults counter is incremented. If the page table is greater than or equal to the specified frame size, and the page table does not contain the page being accessed, the algorithm looks ahead as to what page will be accessed the furthest in the future. The findVictimPage function does this by scanning the remaining portion of the input vector, and selecting the element that occurs furthest in the future that is also in the page table. That page is removed from the page table and the page being accessed is added, and the page faults counter is incremented.

1. Which replacement strategy I would choose would depend on how many page frames I could implement in the page table. For the trials where 100 page frames were implemented, all three algorithms returned either an equal or nearly equal amount of page faults. If this were the case, I would implement a least recently used algorithm, since its implementation was about as difficult as FIFO's implementation, but consistently outperforms FIFO. If I had a limited number of page frames, I would choose to implement the optimal algorithm since its performance outweighs the difficulty of its implementation.
2. The optimal algorithm was the most difficult to implement. Since this required searching the remaining input vector and finding the first occurrence of each page in the page table to determine which page should be removed proved to be a burden. In my current implementation, it was made more difficult by the fact that I made every page a new object, even if their process ID and page number were the same.
3. The FIFO algorithm was the least difficult to implement. The algorithm really is as simple as implementing a queue and checking to see if the page is already in the queue or not.

4. I would not make each input line its own page. This caused problems when trying to remove pages from the page table queue or vector, as I would have to find another object with the same process ID and page number, when I could have just searched for the same object if each page was only created once.
5. I found the increase in performance without consequence when adding additional page frames to the page table surprising. Surely in a truly realistic application, there is downside to increasing the number of pages available to the page table, but there was nothing to discourage from increasing the number of pages available to several hundred or thousands if desired in this simulation.