3 Write Ups for Project 3

Write up part 1: The graph below is for the 3 algorithms when ran on gzip.trace. All three algorithms approach a limit to the page faults. LRU and SECOND perform very similarly and OPT is predictably better than those 2. However, the benefit of OPT evicting the ideal page does not result in much better performance, only ~40 less page faults at 8 frames, and settles down to 18. This difference is very minor compared to the cost of finding and recording all pages uses. Therefore, I would use LRU in a real operating system. It does not require the extra bit to be used like Second, and LRU provides better or equal performance to Second. Without the extra bit, space can be saved in the page table. The page faults for 8, 16, 32, and 64 are described in the graph.

Writeup part 2:

Belday’s anomaly is present in gcc.trace is present when going from 49 to 50 (483 to 486), 59 to 60(419 to 422), 61 to 62 (418 to 424), 69 to 70 (396 to 400), 72 to 73 (390 to 392), 79 to 80 (375 to 376), 82 to 83 (372 to 374), 84 to 85 (368 to 371), 87 to 88 (363 to 365), 88 to 89 (365 to 366), 89 to 90 (366 to 367) and 98 to 99 (356 to 358).

In file gzip.trace, there are again instances of Belday’s anomaly until the page faults hit an asymptote at 32 (39874) and stay there for the rest of the frame numbers.

In file swim.trace, less instances of Belday’s anomaly are present. Once the page faults get under 150, they drop very slowly and Belday’s anomaly occasionally occurs in this range.

Writeup part 3:

OPT is implemented using a hashmap and linked list. During the first pass through of the file, whenever a new page is seen, a new page object is created and stored in the hashmap with its address as the key. Anytime a page is seen, the access number is added to a linked list of accesses stored in the page object. Thus, each page has a sorted list of accesses that can be referenced later. The second pass through of the file, when a page is accessed, the first element of the list is removed. This access just happened, so it should be removed from the list. To find which page to remove, simply look at the head of the list for all pages in the page table.

The runtime of this is O(N) for a pagefault where N is the number of pages in the page table. It is O(1) to find the page object from the hashmap, O(N) to loop over all pages, and O(1) to check the next access. If the page table is not full, it is O(1) to add to the page table, as all that needs done is a hashmap lookup to find the page object.