­CS 4328.001

Operating Systems

Fall 2021

Dylan W. Ray(dwr48), Brett M. Owen(bmo23)

Programming Assignment #3: Page Replacement Algorithms

Due: *Thursday, December 2 @ 11:55 PM*

**Division of Work:**

Dylan Ray:

-Source code implementation for OPT.py.

-Report; “Division of Work”, “Design and Implementation”, “Compilation/Runtime Instructions”, “Execution Data and Interpretation”.

Brett Owen:

-Source code implementation for program3.py, FIFO.py, LRU.py.

-Program Git repository management.

**Design and Implementation:**

For the design of our program, we decided to use a multi-file python implementation. In total we have four files. Three of the four source code files each implement one of the page replacement algorithms; First-In First-Out FIFO.PY, Least Recently Used LRU.py, Optimal OPT.py. The fourth source code file, program3.py, executes the previously mentioned source code files in the proper manner to produce the desired data results. Let us first analyze program3.py. Program3.py defines two functions genReferenceString() and main(). genReferenceString() is used to declare and initialize page-reference data in the proper format, namely a python List of size 100 where each element is a randomly generated integer in range [0, 49]. main() is a programmer defined entry point. It begins by calling genReferenceString() to get the page-reference List, and then prints the List to the terminal. Next, three new Lists are declared to store the results of the three separate page replacement algorithms. Then three for-loops which each iterate a total of 30 times are declared. Through each iteration the parameter for the free frames count in each algorithms call is incremented, 1, 2, 3, . . ., 30. The return page-fault value for each algorithm function call is as well appended to the proper python List. Finally, the calculated results are printed to the terminal. Let us now look at the general design of the three page-replacement algorithm source code files. Each file is similarly structured. All begin by making a copy of the passed page-reference List, declaring and empty free frames List, and declaring an integer counter for the number of page faults. This set up is followed by a while-loop which loops up until each element of the page-reference List is consumed. Within the while-loop an if-else-if block is used to handle three cases during execution. These cases are as follows; The current page being processed from the reference List is already loaded in a frame, not loaded in a frame but a free frame exists or is not loaded in a frame and no free frame exists. How each algorithm handles these cases is where they differ the most. FIFO simply pops the current page in the page-reference List if it already loaded. If the current page being processed in not in the frames List and a free frame exists, the page is appended to the frames List and popped from the reference List. In the last case, no free frame exists, the front frame is evicted, the page being processed is appended to the frames List, and again the page is popped from the reference List. In LRU, for the first case of a page-hit, we remove the page from its current position in the frames List and append it to the back of this List effectively marking it as the most recently used page. Doing this has the added side effect of making the least recently used page bubble up to the front of the frames List. The remaining two page-fault cases are handled the same as in the FIFO implementation, but because of the previously mentioned property of the page-hit case, evicting the first page of the frames List means we are evicting the least recently used page by default. Now we shall analyze OPT. The page-hit and page-fault with a free frame case are implemented in OPT the same way as they are in FIFO. Only the page-fault with no free frame case is implemented differently. When this case in OPT is met a for-loop which traverses the current configuration of the free frames List is made. For each page element in the frames List the page-reference List is traversed to find the index of the next occurrence of that page. The maximum index found in the page-reference string is kept track of and the corresponding page to this maximum index in the frames List is the page which will be evicted. Once this page is evicted the new page is removed from the page-reference List and appended to the frames List. In each page replacement algorithms source code file for the two page-fault cases the page-fault counter is implemented and returned at the end of the code once the page-reference List is fully consumed.

**Compilation/Runtime Instructions:**

To run our program first ensure you have the most recent version of the python interpreter, 3.9.1, installed on your machine. Next make sure that all constituent files (program3.py, FIFO.py, LRU.py, OPT.py) are in the same directory. We steered slightly away from the suggestion that “the number of page frames available [should] be passed in as an argument”. Our implementation is not facilitated using such a command-line argument. Instead, we built the required 30 executions per page replacement algorithm directly into the program3.py driver code. To run the overall program simply call in the terminal “**$ python3 program3.py”** when the present working directory is where you have all required source code files**.** The data should then be printed to the screen.

**Execution Data and Interpretation:**

Reference String: 42, 6, 11, 3, 18, 45, 14, 5, 11, 8, 40, 1, 22, 39, 11, 40, 11, 5, 44, 35, 20, 49, 41, 2, 46, 38, 32, 0, 0, 39, 1, 35, 38, 1, 0, 32, 35, 24, 5, 30, 11, 44, 39, 44, 11, 43, 35, 41, 33, 7, 16, 3.

FIFO: 98, 96, 94, 92, 87, 84, 82, 78, 78, 77, 71, 69, 67, 65, 58, 58, 58, 58, 58, 58, 58, 57, 53, 52, 51, 48, 47, 47, 46, 45.

LRU: 98, 96, 94, 93, 87, 83, 81, 77, 76, 75, 73, 70, 68, 64, 63, 58, 58, 57, 53, 51, 51, 51, 50, 50, 50, 50, 49, 49, 47, 46.

OPT: 98, 88, 80, 73, 68, 64, 61, 58, 55, 52, 50, 48, 46, 45, 44, 43, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42.

Let us analyze the output of our three algorithm runs. Beginning with OPT we see a smooth downward trending curve, seemingly hitting a horizontal asymptote and bottoming out at 42 page faults. In fact for frame sizes of 17-30 the number of page faults is consistently maintained at 42. This indicates that as the size of the free frames List increases the number of page faults follows a regular inverse relationship and correspondingly decreases. The Optimal Page Replacement Algorithm is named as such because it is the ideal, yet practically unattainable, solution to page replacement. Any page replacement algorithm that has a graph curve closer to the OPT graph curve would likely be more efficient. How do both FIFO and LRU compare to OPT? We can plainly see that for a frame size of only 1 all algorithms perform the same, trivial. But surprisingly both LRU and FIFO have a much more similar performance then expected. In fact, the average difference between LRU and FIFO is only 1.9 page faults. Most of the difference in page faults, from theses specific results, occurred between free frame sizes of 15-23. Both LRU and FIFO trend down in page fault occurrence as the free frames list size increases. This is like the OPT algorithm, but with the distinct difference of a more sporadic jagged curve and an all-around higher page fault count. This data is the result of only one execution of our entire simulation with frame sizes from 1 to 30. Ideally, we should have run the program multiple times and taken the average of each result for each frames list size for each algorithm and used that in our graph. For this singular set of results, it is unclear weather FIFO or LRU would be the desirable choice of algorithm, already knowing that the OPT algorithm is practically infeasible.