COMP2240 A3 - A Comparison of Page Replacement Policies

Jeremiah Smith

c3238179

University of Newcastle, Australia

Author Note

First paragraph: Least Recently Used (LRU) Replacement Policy

Second paragraph: Clock Replacement Policy

Third paragraph: Summary of policies and program

Fourth paragraph: Program testing techniques

Programming Assignment 3 - Report

The LRU replacement policy is a simple yet effective algorithm used by processes when a process memory is full and the incoming page to be read from is not currently in memory. For this to be necessary, page to be ready from must not be in memory and all existing frames in memory must already contain a page. The algorithm works by selecting and replacing the page that was used least recently. This meant that every time a process accessed or "used" a page in one of its frames, then it was necessary to save the time of the event for that process so that when the process needs to replace an existing page, it can find the page with the smallest value and overwrite that page with the new one. Testing this policy involved taking the number of frames for a process into consideration, as a larger number of frames with a relatively small number of pages to read from would mean the replacement policy may never be used or will be used very in the program execution. So tracing program output to find the time that the replacement policy was first used was necessary. The results would then be evaluated as to whether they complied with the expected values of the LRU policy. Specific test input was created to test the correctness of the policy by considering the scenarios when the LRU policy would create unique results and testing whether the program would handle those scenarios. An example was the scenario where each process had a very small number of available frames, in which case the policy was used quite frequently and needed to be understood well in order to coherently trace the results. The LRU policy is easy to implement and also is quite effective, however, it involves a bit of overhead when finding the least recently used page and saving information about when pages are accessed.

The Clock Replacement Policy is a slightly more complicated page replacement algorithm which searches frames to replace like the hand of a clock, moving from one page to the next and looping until it finds the one which it must replace. Frames are referenced with a bit value of 1 or 0 which represents what kind of event last referenced that page. Because of the complexity of this algorithm, it was more difficult to implement than the LRU policy. Mistakes which lead to incorrect output included not understanding the concept of the clock hand properly in that a process needs to save the position of the hand/pointer so that the next time the policy is used, it starts from where it left off. The last mistake that required correction also related to this as the program would find a page to replace, update the frame's use bit value to 1, and move on, resulting in the replacement policy setting that same use bit back to 1 as soon as the policy is called again. To avoid this redundant process of switching

the value of the same use bit twice, the process should, in fact, move its pointer to the next frame as the last action after replacing a page. Once again, testing for this replacement policy involved creating custom input files and evaluating the results in accordance with the specifications of the Clock Replacement Policy.

When reviewing the results of both of these replacement policies, it was apparent that the LRU replacement policy is usually more effective than the Clock policy in the sense that it results in the least number of page faults during process execution. This results in processes being more productive under an LRU policy especially where there are some page numbers which are used far less frequently than others, as the more frequently used pages can stay in memory for longer or never be replaced at all. In contrast, Clock replacement still minimizes page faults but not as effectively as it does not take as much relevant information into consideration during this process despite its complexity. The benefit of the Clock policy lies in the fact that it has less overhead than the LRU policy as it only uses a few simple bits of data to execute. So there may be specific use cases where Clock policy is preferred to minimize overhead. This would likely be when the the number of times that each page a process needs to access is relatively equal for every page, such that using Clock over LRU as the replacement policy does not yield a significant enough difference in the number of page faults to necessitate using LRU over Clock. The Clock policy may be preferable in these instances to reduce overhead and further increase the productivity of the CPU.

The primary technique used when testing for the correctness of the policies in the program was to manually trace through the results and calculate whether the results were expected. To do this, print statements were made at each time interval to record data pertaining to the state of each process and the events that occurred at that time such as page faults and page executions. Further testing the correctness of the replacement policies was best done by comparing the results of more large and complicated custom input files with other students. The algorithms had to be analyzed and the implementations tweaked until all of our results were the same and any misunderstandings about the policies were smoothed out.