Assignment 3 – Comparison of Different Resident Set Management Policies

By Jacob Boyce
4.10.2022

Testing Methods, Specific Tricks Applied

Debug Statements: By strategically placing debug statements around the program, the inner workings of the program can be tested far easier. The debug output can be compared to GANTT charts or the expected working of the algorithm to see if any error occurred.

Separation of Concerns: By separating all the required components into modules (classes), it allowed them to be tested individually in isolation. For example, the dispatcher, resident set policy and replacement policy were all separate classes. This made testing and developing the assignment significantly faster and easier. For reference you can view ‘`TestClock1()`’ in ‘`A3.java`’ which tests the GCLOCK replacement policy individually

Edge Cases

All test cases can be viewed in `` `A3.java` ``. Notable edge cases are shown here *Volume Test.* In this test case there is 100 processes. Each process had 10 lots of the three instructions with a time quanta of three (1,1,1, 2,2,2, …). As expected, all 100 processes page fault at time stamp 0. All processes begin running for their time quanta. All processes then page fault again at time stamp 306 and so forth until termination.

No Page Faults: In this test case all processes have a singular instruction. As expected, all processes page fault at time stamp 0. But after this singular fault a simple round robin occurs with each process only executing three instructions.

Always Page Fault: In this case every instruction in every process is different and each process had exactly 1 frame. As expected after every instruction occurs a page fault occurs. As there are only just enough frames for 1 per process the frame always gets replaced. In the global replacement policy this results in some processes frames being overridden requiring a second page fault.

Comparison of Replacement Methods

Due to the brevity of the report specific testing examples to show these comparisons are not included!

Global Replacement: The page fault rate of a given process is not just affected by the process itself by the nature of all processes. For example, if there was a single process with a large memory requirement and several other processes with a lower memory requirement, the large process would begin replacing smaller processes frames causing them to fault more often, however by allocating more space to the larger processes, their turnaround time was much faster as they were not hindered by the limited memory space caused by a fixed replacement policy.

Fixed Replacement: The page fault rate of a given process is affected solely by the process itself and the number of frames the process was allocated. Additionally, this method resulted in a lot of internal fragmentation as smaller processes do not use all the memory they were allocated and external fragmentation as not all frames were allocated. This meant that smaller less memory intensive processes had a lower turnaround time, whereas larger processes had a significantly higher turnaround time, even if all other processes had terminated when compared to global replacement.

Specific Issues

One issue that occurred is that if multiple page faults unblocked at the same timestamp, they would not enter the ready queue in the same order they were blocked. This is because the priority queue in java does not enforce FIFO. This was resolved by adding a unique id for each event. Each event that was created would have an id higher than the previous event. The issue could then be resolved by comparing these id's if the timestamp of the events was equal.

Another issue that occurred was with the GLOCKS policy. Originally the counter would not be incremented after a page had been replaced causing a newly replaced page to be immediately written over. This was detected when testing this policy in isolation.