Matrix Analysis

Page Fault Analysis

Project 4

COP 4634 Systems & Networks 1

Fall 2016

Ashley Miller/Torrey Bettis

3 November 2016

This analysis will be of a large two dimensional array, and what happens in memory when it iterates through the major aspect. It will also look at precautions that can be taken to minimize any negative impacts to processing time and memory manipulation.

When accessing a page that is not currently in memory, a page fault is generated. The array used in this experiment was 20480*i* rows by 4096*j* columns. 4096 was also designated as the page size. There are 15 frames when starting this experiment. Two frames are taken up by the code and the stack, leaving 13 frames free for page storage. When traversing this matrix in row major fashion *i* is incremented, and new item loaded creates a page fault. In this case since each row is a page size of 4096, then each row creates a page fault. There are 20480 rows in our experiment, so row major operations generates 20480 page faults.

Navigating the matrix using column major operations presents more problems. When using this method, you really get dinged twice for page faults. Every bit in the column causes a page fault, since you are only reading 1-bit from each page. So using column major operation we fins we are faced with 20480 \* 4096 page faults or a staggering 223 page faults. This is evident is you look at the times for column major operations, with their much longer run times being 10 times or more longer than row major operations.

To eliminate some of the page faults in this experiment we would need to run read operations immediately after write operation, but this would only be beneficial in the row major operations. Since the page is already in the memory from the write operations, we would be able to read it for “free”. The term free in this case means no page fault would be generated, since the page fault we be only generated on the write actions.