----------WRITE UP/READ ME----------

Name: Shawn Banish

Project Due Date: 11/13/2016 11:59 PM

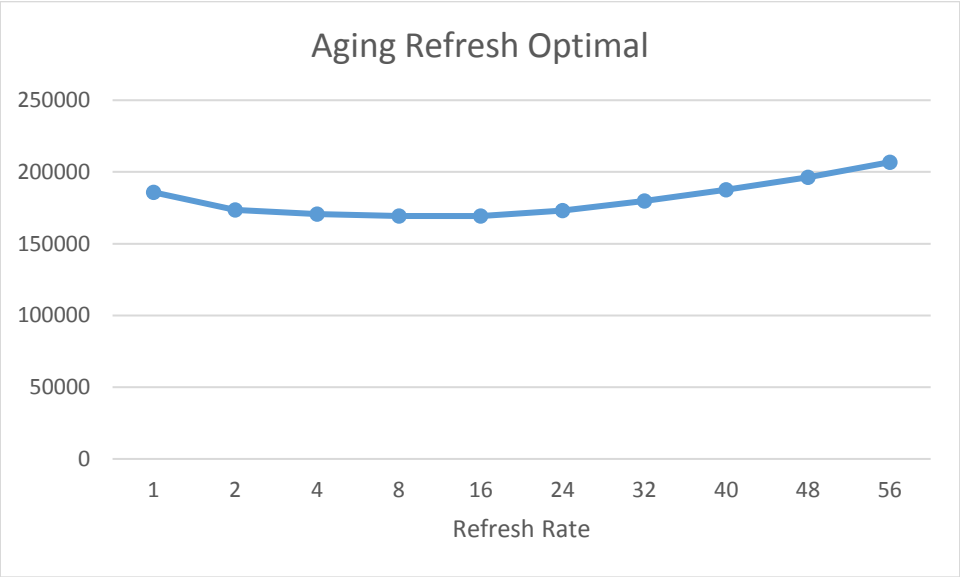
Source File Names: vmsim, Page, PageTable, FrameTable, Opt, Clock, Aging, Work

Compile Instructions:

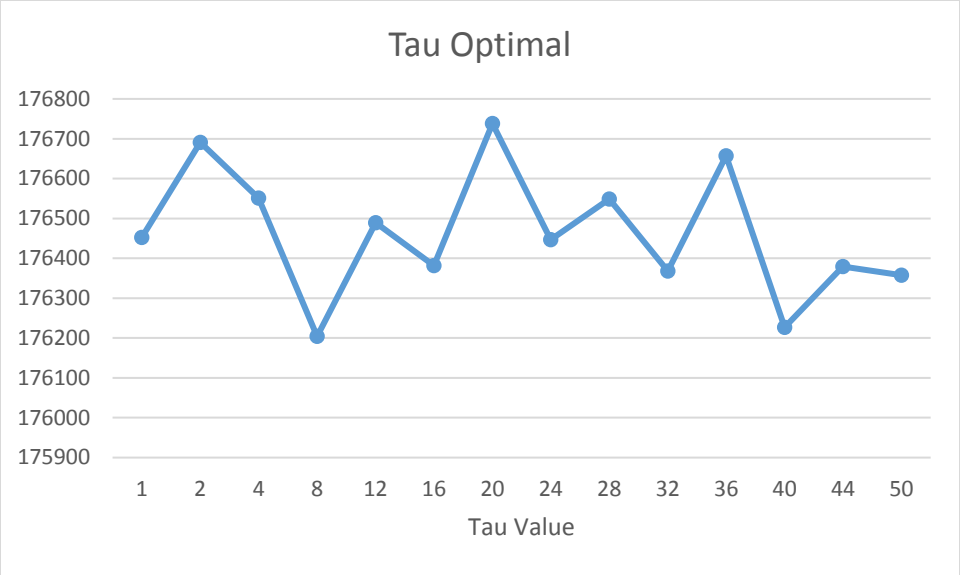
Compile with javac: javac vmsim.java

Running: java vmsim –n <numframes> -a <opt|clock|aging|work> [-r <refresh>] [-t <tau>] <tracefile>

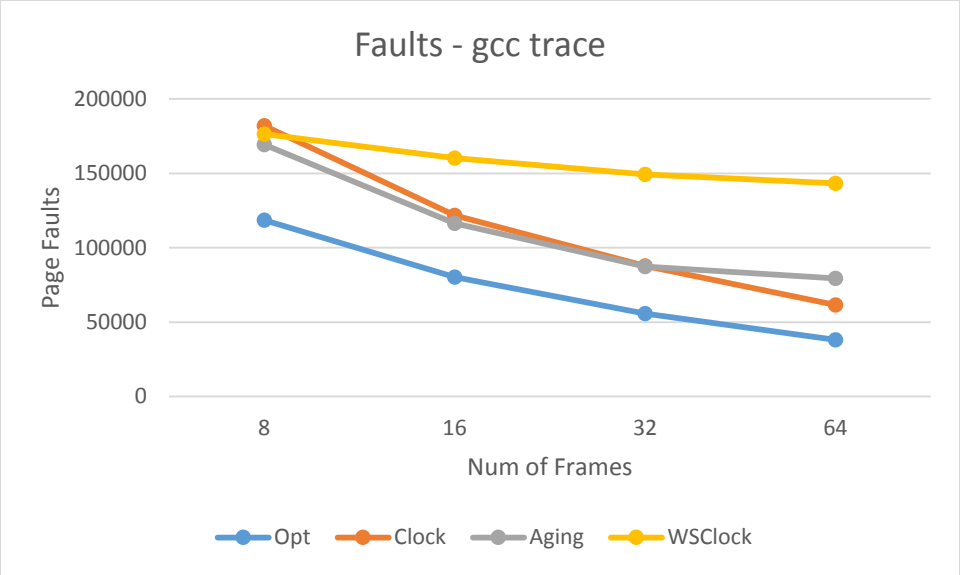
Notes to Grader:

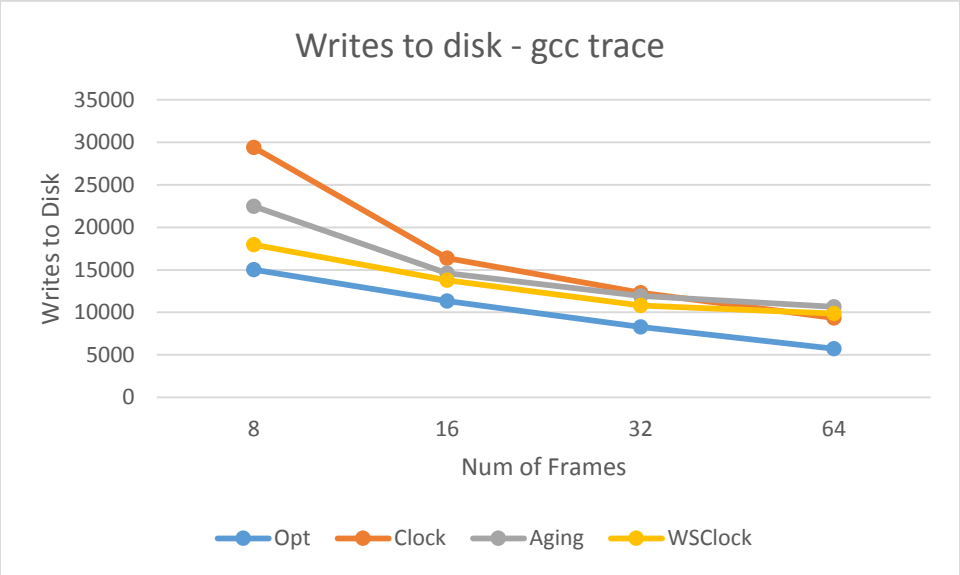


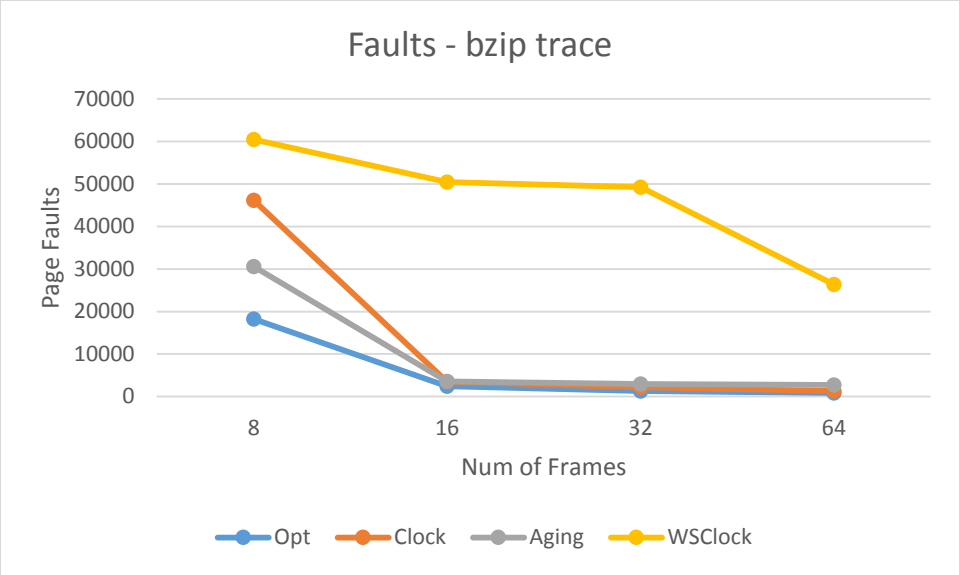
Response: I decided to make my refresh rate 16. I tested the refresh rates mostly in patterns of four, and after doing that I noted down the result in an Excel table. Upon creating a graph, it appeared that the minimum value aligned with the refresh rate of 16, so I decided to choose that.

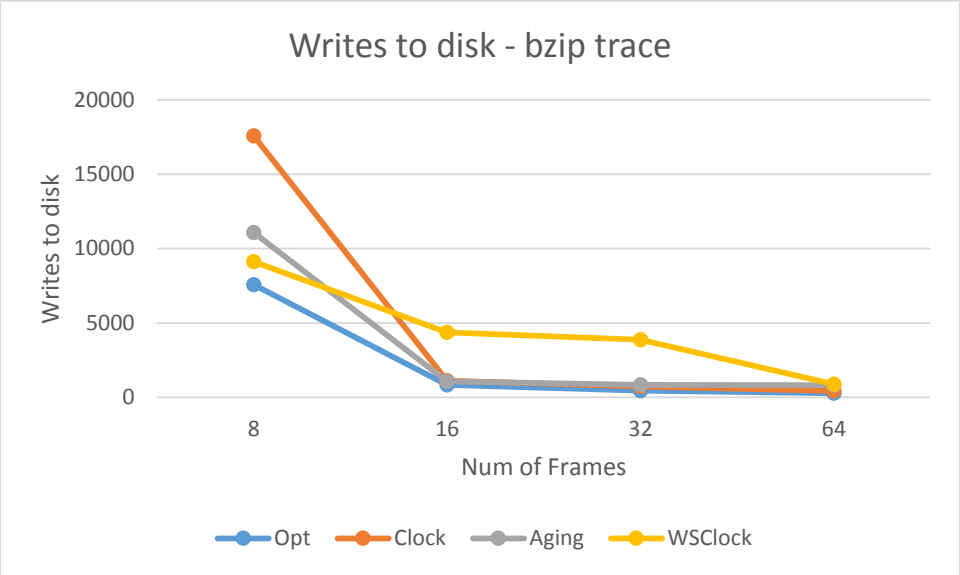


Response: After deciding to use the refresh rate from aging (8) I began searching for the most efficient value for tau by tracking the results after increasing the value of tau by four each time I ran vmsim. This result was less conclusive than the refresh rate, however, to me it appeared that the lowest value occurred at 8, so I decided to use that as my tau.


Faults - gcc trace
200000
150000
100000
50000
0
Page Faults
8
16
32
64
Num of Frames
Opt
Clock
Aging
WSClock


Writes to disk - gcc trace
35000
30000
25000
20000
15000
10000
5000
0
Writes to Disk
8
16
32
64
Num of Frames
Opt
Clock
Aging
WSClock





Response: Upon completely testing all four algorithms with both gcc trace and bzip trace, it is clear that the optimal algorithm works best. I would next place the aging algorithm second. In the results from gcc trace, it is fairly inconclusive whether aging or clock is better, however, in the results from bzip trace it is clear the aging has less page faults and writes to disk less. After aging, I would rank clock, and lastly I would put WSClock.