CS1550 – Introduction to Operating System
Project3 – Virtual Memory Simulator

Haoyue Cui (hac113@pitt.edu)

11/8/2019

Introduction

In this report, we investigate 3 different page replacement algorithms – Fifo, Opt, Aging based on a 32 bit implementation of a virtual memory in java (visim program). The page replacement algorithms are as follows:

Fifo: The page replacement algorithm which will Implement first-in, first-out.

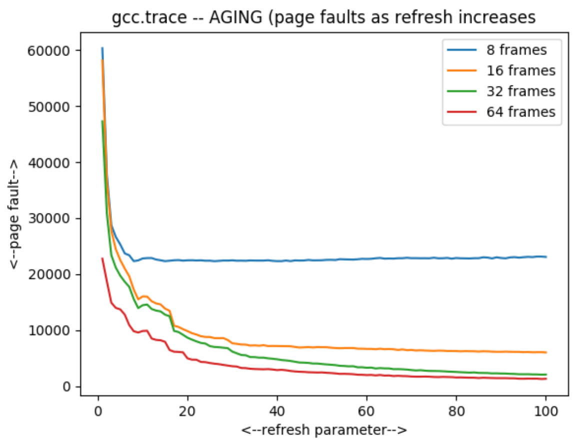
Optimal: The page replacement algorithm which knows the future memory accesses. Simulate what the optimal page replacement algorithm would choose if it had perfect knowledge. When RAM is full, the program will evict the largest future memory accesse since it is the furthest memory needed.

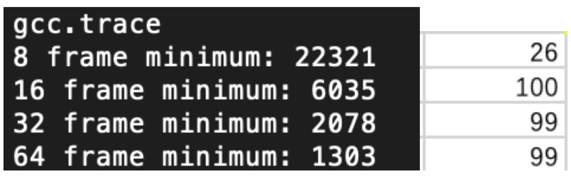
Aging: The algorithm is a variant of Not Recently Used(NRU) algorithm with reference bits and counters that shows the page recent uses by shifting and adding the reference bits. The reference bit is used to record the previously visited memory. Implement the aging algorithm that approximates LRU with an 8-bit counter. Do a refresh of the R bits every P CPU cycles.

In the following of this report, we conduct two experiments. The first one tries to determine an approximately minimum refresh parameter for 8, 16, 32, 64 frames and plot the results in the graph in the second part. The second one is to determine which algorithm might be most appropriate for use in an actual operating system. The third one is to determine if there are any instances of Belady's anomaly in three traces using FIFO algorithms.

Experiment 1: Finding the approximately minimum refresh parameter for different frames

I plot all the test outputs in python and find the minimum refresh parameters for each frame numbers in **gcc.trace, gzip.trace and swim.trace.**



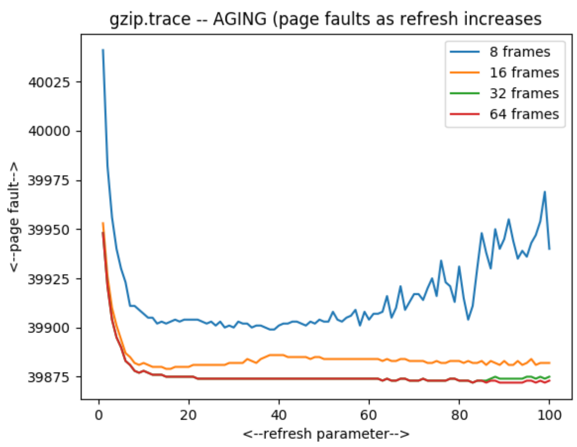


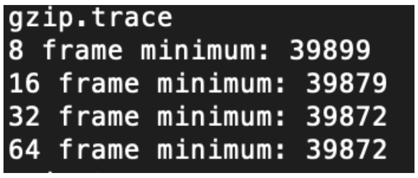
```
gcc.trace
8 frame minimum: 22321
16 frame minimum: 6035
32 frame minimum: 2078
64 frame minimum: 1303
```

26
100
99
99

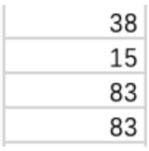
Figure 1

The refresh parameter used for next experienement is **26,100,99,99**





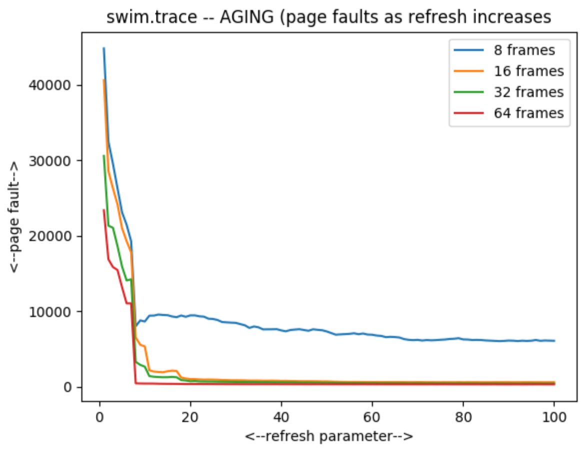
```
gzip.trace
8 frame minimum: 39899
16 frame minimum: 39879
32 frame minimum: 39872
64 frame minimum: 39872
```

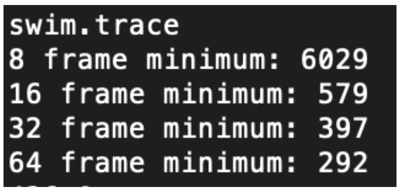


38
15
83
83

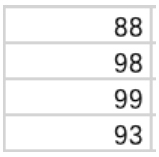
Figure 2

The refresh parameter used for next experienement is **38,15,83,83**





```
swim.trace
8 frame minimum: 6029
16 frame minimum: 579
32 frame minimum: 397
64 frame minimum: 292
```



88
98
99
93

Figure 3

The refresh parameter used for next experienement is **88,98,99,93**

After experiments on three trace files, we can see the graphs that the number of page faults gives a trend. Page faults go down when refresh parameter increases. And after the dramatically decreases of page faults, there is no much difference in pages faults as refresh parameter increases. But we still use the absolute minimum refresh parameters for all the cases and are going to use it in experience 2.

Experiment 2: Comparing different page replacement algorithms

In this section we report the results for different page replacement algorithms using gcc.trace, gzip.trace and swim.trace. The experiments used 8,16,32 and 64 frames. The following discussion mainly tries to point the behavior of different algorithms on both of these figures. From Figure 4, Figure 5 and Figure 6, the Optimal page replacement algorithm has the best results whereas the Fifo algorithm has worst performance normally. The Optimal algorithm can predict the future page requests and it can ultimately decide which one to replace. Fifo algorithm fails in the gzip.trace file. Fifo algorithm evicts the first memory goes in RAM. This means gzip trace file have many repeated memory accesses. Since fifo algorithm does not look at the recently used memory access, when there are many repeated memories first coming into RAM, aging algorithm will not evict that frequently used memory. But fifo algorithm performs better than aging algorithm in gcc.trace and swim.trace when frame

number gets bigger. The aging algorithm has a better start with 8 frames, but it cannot maintain its performance while frame numbers increase. This observation is mainly due to the span of use feature of aging algorithm with the current page reference and tends to fail the algorithm while a larger frame numbers give not correctly correlated with the future use of pages.
In gzip trace file, opt and aging perform basically same results. This means this trace works better for future predicted memory access trace files. The access pattern of the trace files follows a specific long-term pattern. Maybe this is because the memories accessed are always new. So the drawback of aging algorithm does not give bad results.
In swim trace file, three algorithms give approximately same results while increasing frame numbers. With small frame number, optimal does better than aging and fifo is the worst algorithm when there are small frame numbers. Because the evict algorithm is used often with small frame numbers. Better predicted evicted page will give better results.

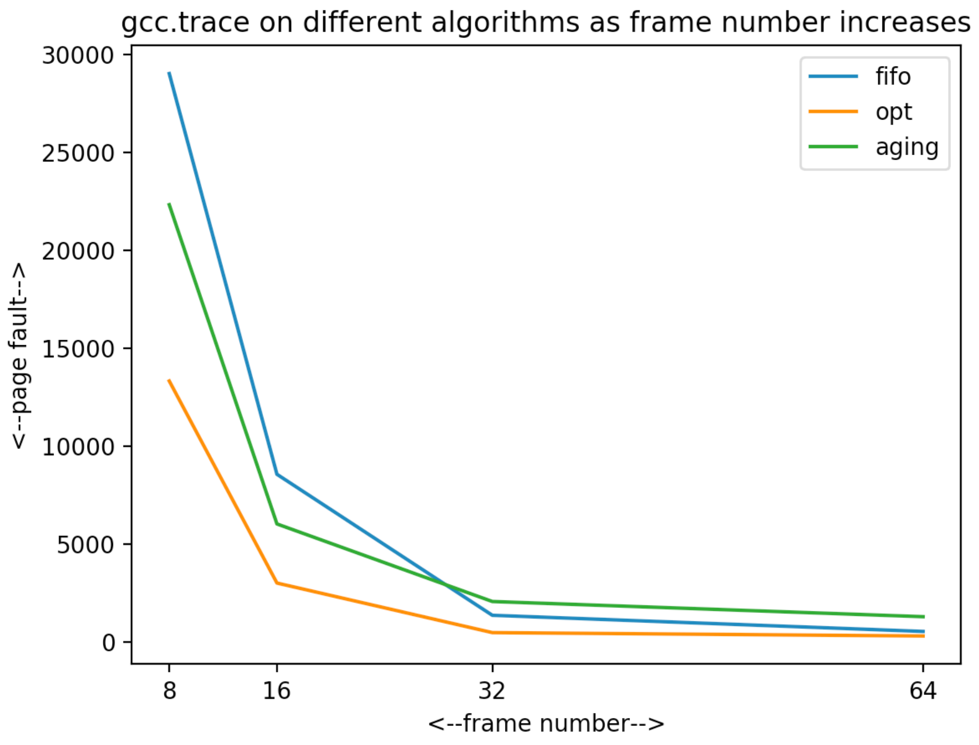


Figure 4

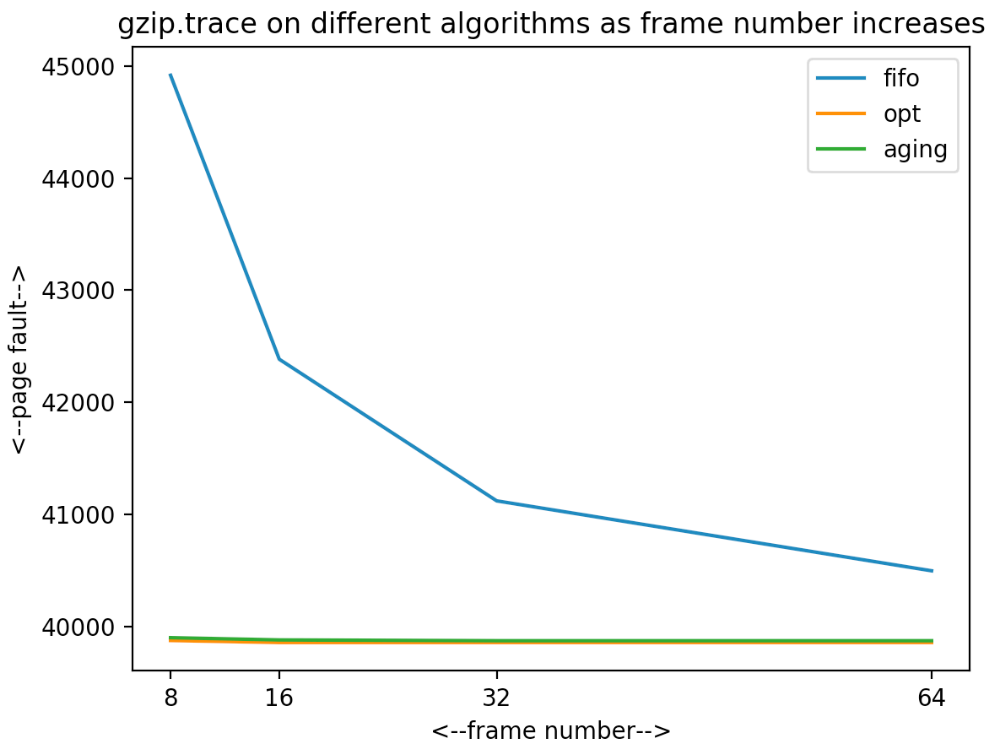


Figure 5

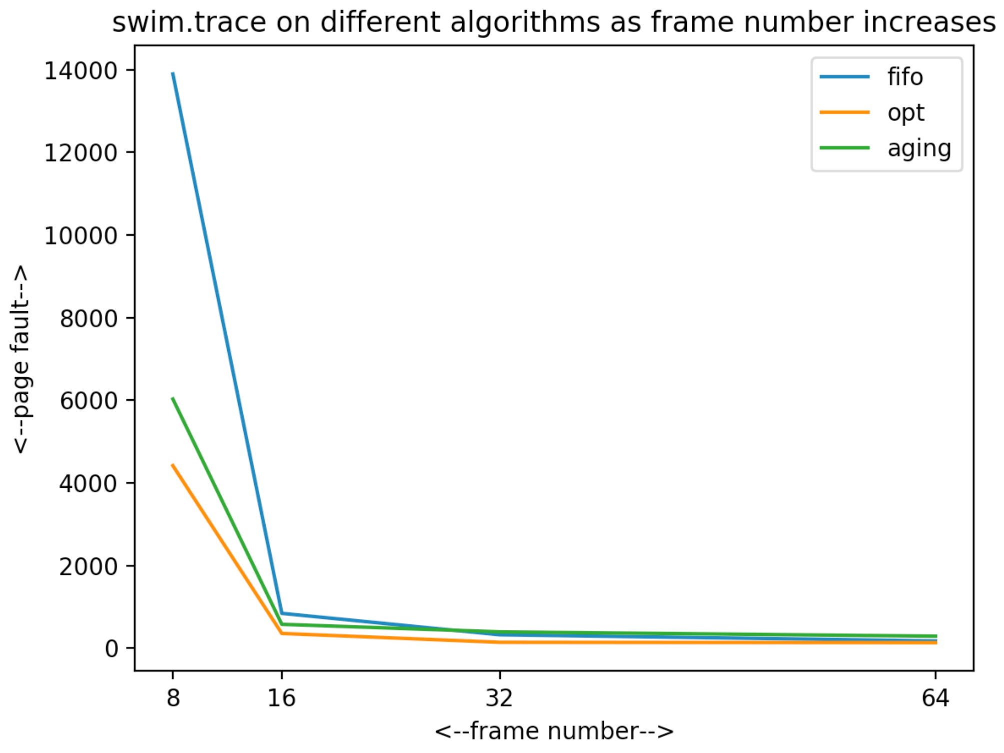
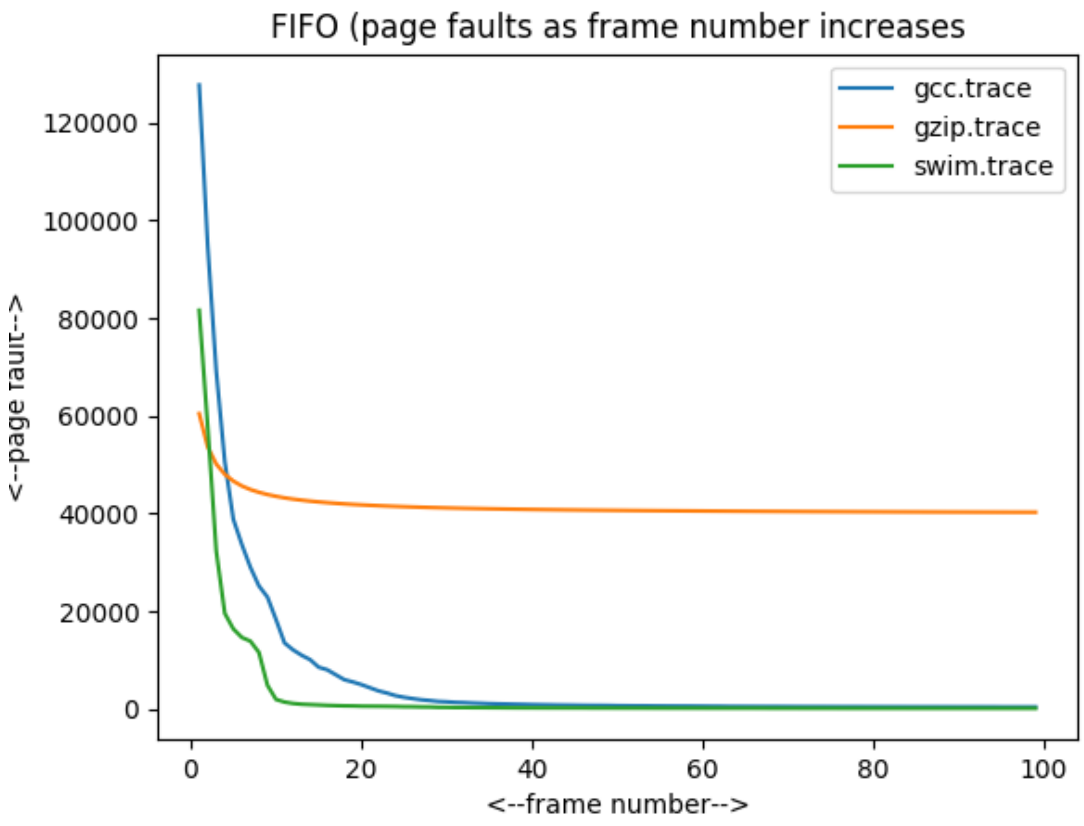


Figure 6

Experiment 3: Determine if there are Belady's anomaly with Fifo algorithms

With the three traces and varying the total number of frames from 2 to 100, determine if there are Belady's anomaly with fifo algorithms. I compute the values of all page faults and there are **no** Belady's anomaly for three trace files.


FIFO (page faults as frame number increases
gcc.trace
gzip.trace
swim.trace
<--page fault-->
<--frame number-->
0
20000
40000
60000
80000
100000
120000
0
20
40
60
80
100