Yifan Yu
Prof. Douglas Thain
Operating Systems
April 12, 2019

Project 5: Virtual Memory

Introduction

The purpose of this project is to enhance our understanding of mapping between virtual and physical memory. At the same time, we monitor the performances of different ways to evict from physical memory given different programs. Specifically, I have implemented a Page fault handler. The page fault handler updates the page table and moves data between the physical memory and the disk. This handler then utilizes one of the three replacement algorithms. Random takes a random number of frames and evicted that entry in the page table. First-in-first-out implements a queue of page numbers where the page corresponding to the front of the queue is evicted first. A custom algorithm which will be detailed in the next section. For this experiment, each program and each page replacement algorithm's number of page faults, disk reads and disk writes are measured to examine their trade-offs under different conditions.

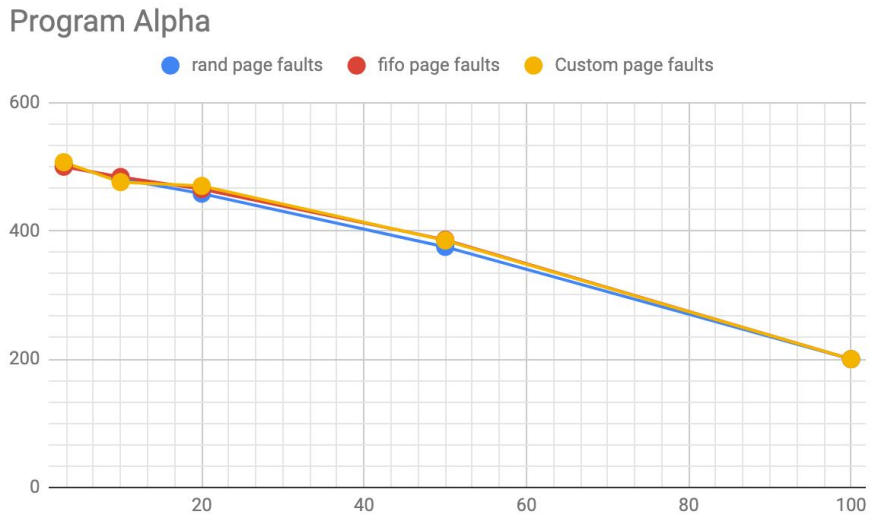
Experiments is performed on student machine `student00.cse.nd.edu` and here are the command line arguments.

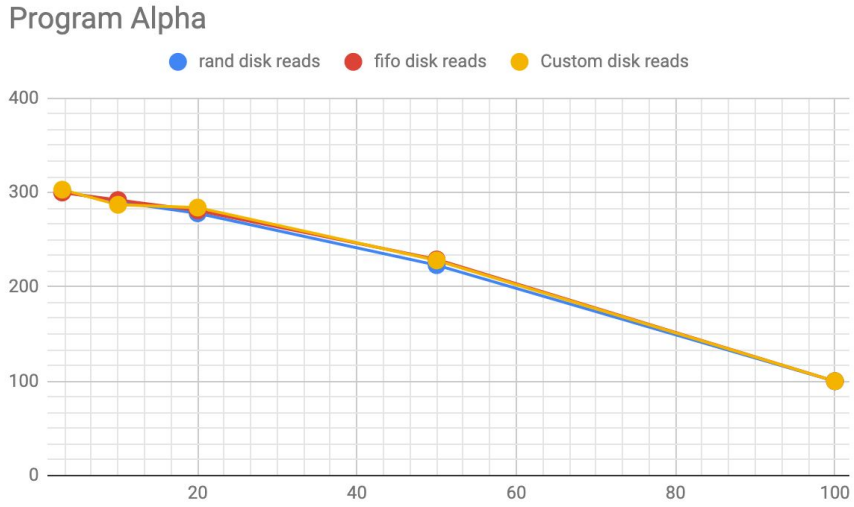
```
./virtmem <npages> <nframes> <rand|fifo|custom>
<alpha|beta|gamma|delta>
```

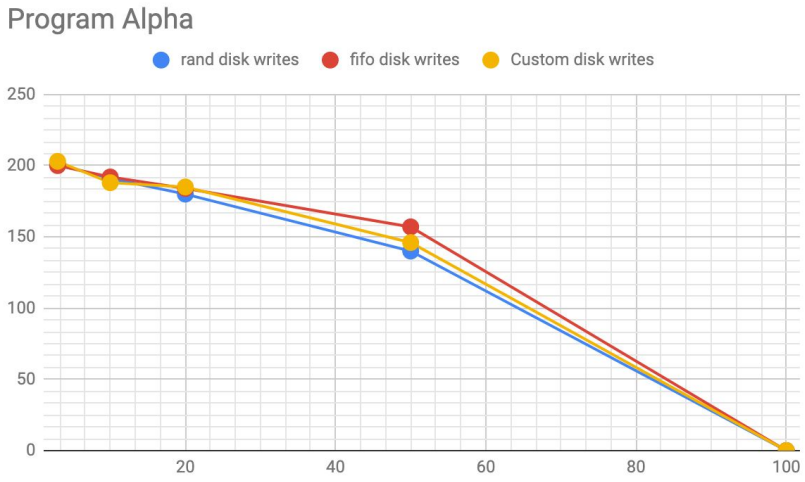
Custom Page Replacement Algorithm: NRU(Not Recently Used)

We have implemented the Not Recently Used Custom Page Replacement Algorithm. Often times most recently used page numbers will be used again so this algorithm makes sure they are not the pages being replaced. Specifically, In this replacement algorithm, the page numbers are added to a buffer as a page fault is detected, size `nframes/10`. If the physical memory is not empty, we then randomly evict a page number from the physical memory. We evict a different random number if the page number is in the buffer. Thus protecting the most recently evicted pages.

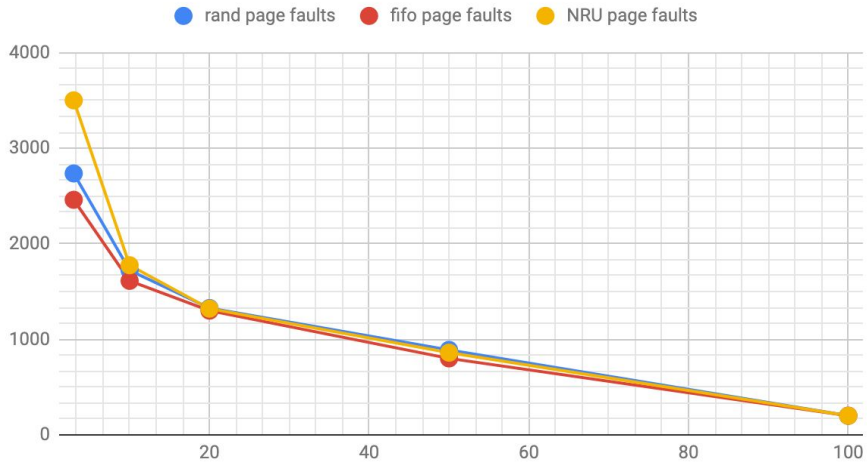
Algorithm Performances: (x vector is number of frames)



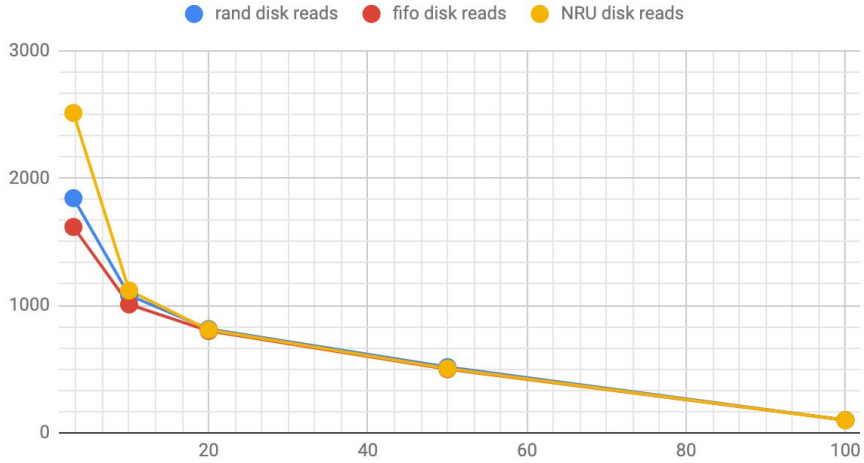




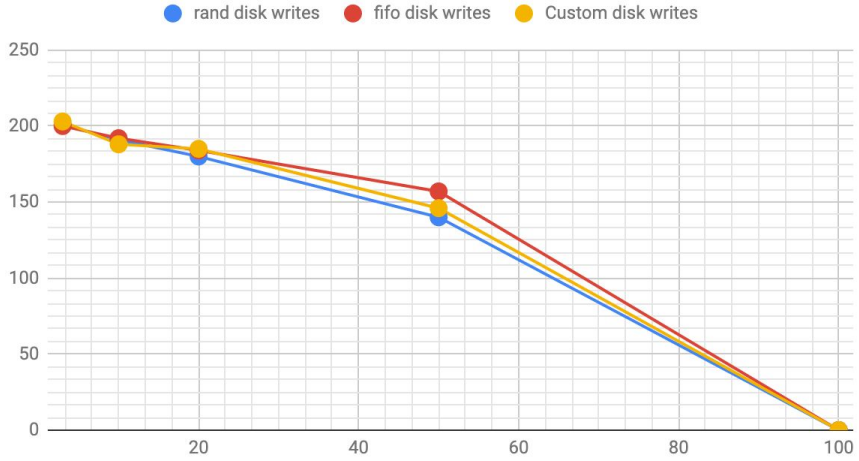
Program Beta



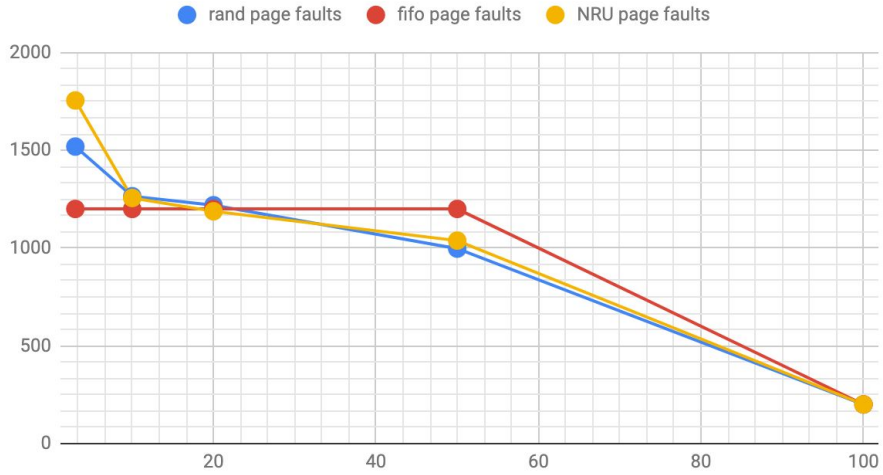
Program Beta



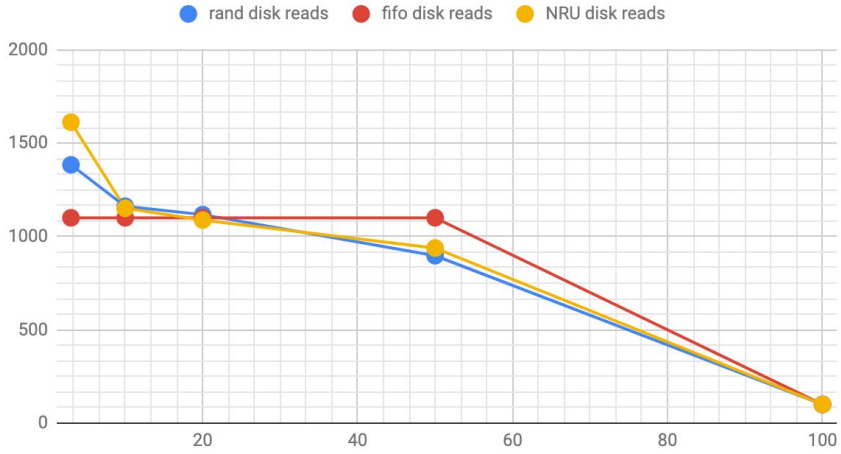
Program Beta



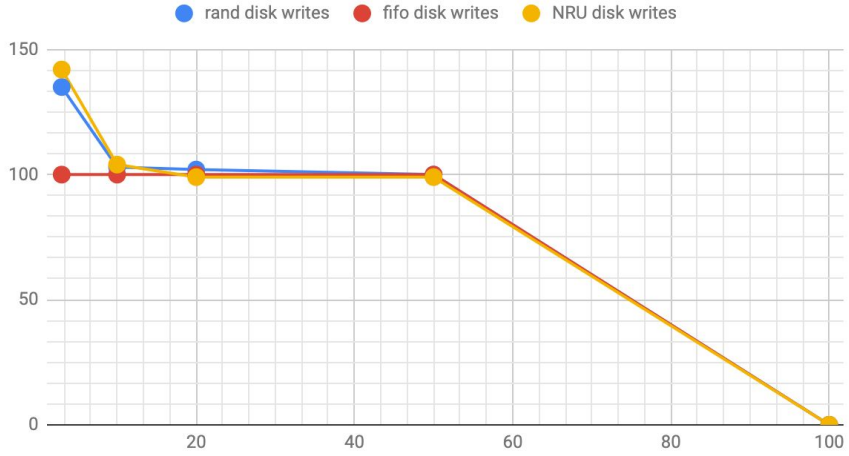
Program Gamma



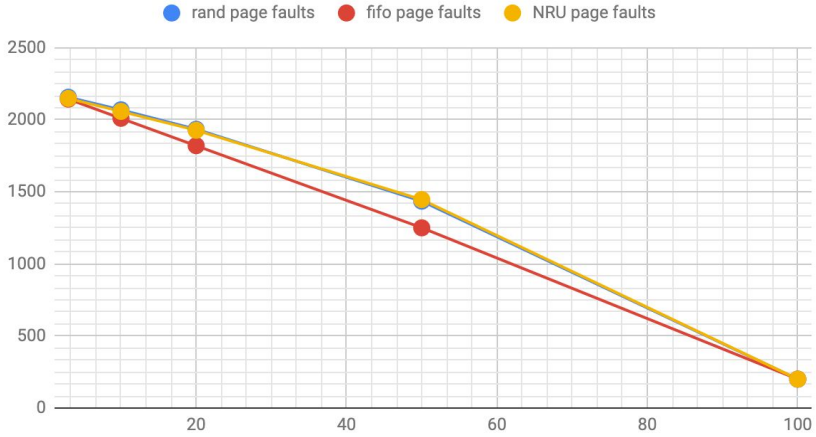
Program Gamma



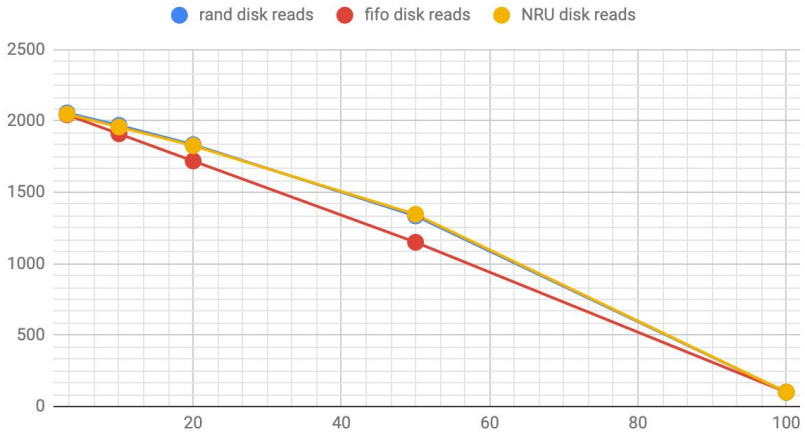
Program Gamma



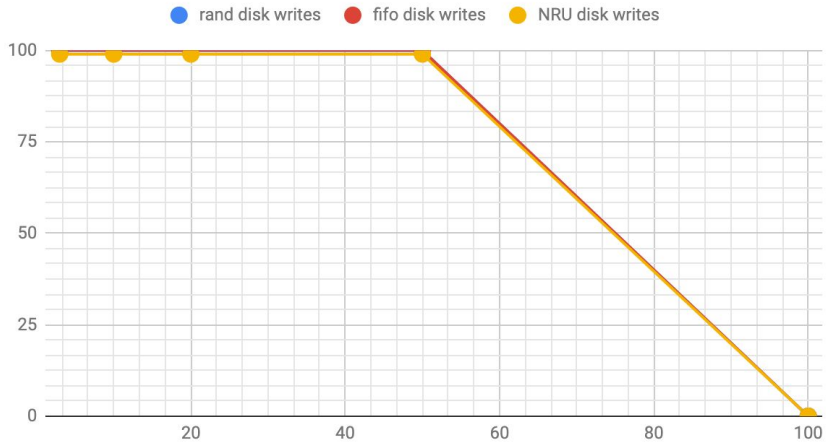
Program Delta



Program Delta



Program Delta



For Alpha program, the number of page faults, disk writes and disk reads slowly decreased as the nframes increased. As the number of frames reach closer to the number of pages, the random page replacement algorithm had the least amount of disk writes and the fifo algorithm had the most. This likely a result of fewer evictions and fewer opportunities for a page evicted to have a dirty bit. All three algorithms demonstrated similar performances in page faults and disk reads.

For the beta program, the number of page faults, disk writes significantly decreased as the nframes increased. The custom algorithm had more page faults and disk reads than the rand and the fifo algorithm. This is likely a result of lack of optimization of buff size and a bad voodoo value. However, as page frame start to increase, fifo had more disk writes compared to the other algorithms and random algorithm had more page faults.

For the gamma program, the number of page faults, disk writes and disk reads remain leveled as the nframes increased. The gamma program consists of matrix access. FIFO had the same overall number of page faults, disk reads and disk writes. This can be explained as the number of page faults where the number of things with the write bit is decreasing while the number of evictions with the read-only bit is increasing.

For the delta program, the number of page faults, disk writes linearly decreased as the nframes increased. The delta program consists of array access. The graph for page faults and disk reads is almost linear with first in first all algorithm having fewer page faults and disk reads. The fifo simply evicts the earliest page in memory requested and this allows sequential programs to have the right page.