Page Replacement Policies Simulation

AOS Project | Team - Aos_project_1234

Introduction

In an OS that uses paging for memory management, page replacement algorithms decide which memory pages to page out (sometimes called swap out) or write to disk when a page of memory needs to be allocated. Page replacement happens when a requested page is not in memory (page fault) and a free page cannot be used to satisfy the allocation, either because there are none, or because the number of free pages is lower than some threshold.

When the page that was selected for replacement and paged out is referenced again it has to be paged in (read from disk) and this involves waiting for I/O completion. This determines the quality of the page replacement algorithm: the less time waiting for page-ins, the better the algorithm. A page replacement algorithm looks at the information about accesses to the pages provided by hardware, and tries to guess which pages should be replaced to minimise the total number of page misses, while balancing this with the costs (primary storage and processor time) of the algorithm itself.

This project simulates and measures the performance of page replacement algorithms such as :

- Aging (approximate LRU)
- Clock
- FIFO (First-In First-Out)
- FIFO with Second Chance
- LRU (Least Recently Used)
- NFU (Not Frequently Used)
- NRU (Not Recently Used)
- Optimal
- Random
- Working Set
- WSClock

While keeping track of the pages that are being loaded, the performance of all the above mentioned algorithms was observed and represented in the form of graphs.

Approach

For each page replacement algorithm, we have written C++ program that reads a memory trace, implements the actions of that algorithm and writes the hit ratio and fault ratio to a file.

To analyse the performance of each page replacement algorithm, we have written a Python code (using matplotlib) to plot a graph of hit ratio and fault ratio vs the numbers of frames.

Additionally, we have written a bash script (Project.sh) that first runs C++ programs for each algorithm and then runs the Python program to plot the graph.

Learning

While working on this project, we got a better understanding of the different page replacement algorithms and the use case for each algorithm.

Implementing and analysing these algorithms allowed us to learn more about the role the operating system plays in memory management. It also helped us to find and understand the difference between the different algorithms and their behaviour in different conditions by plotting their graphs.

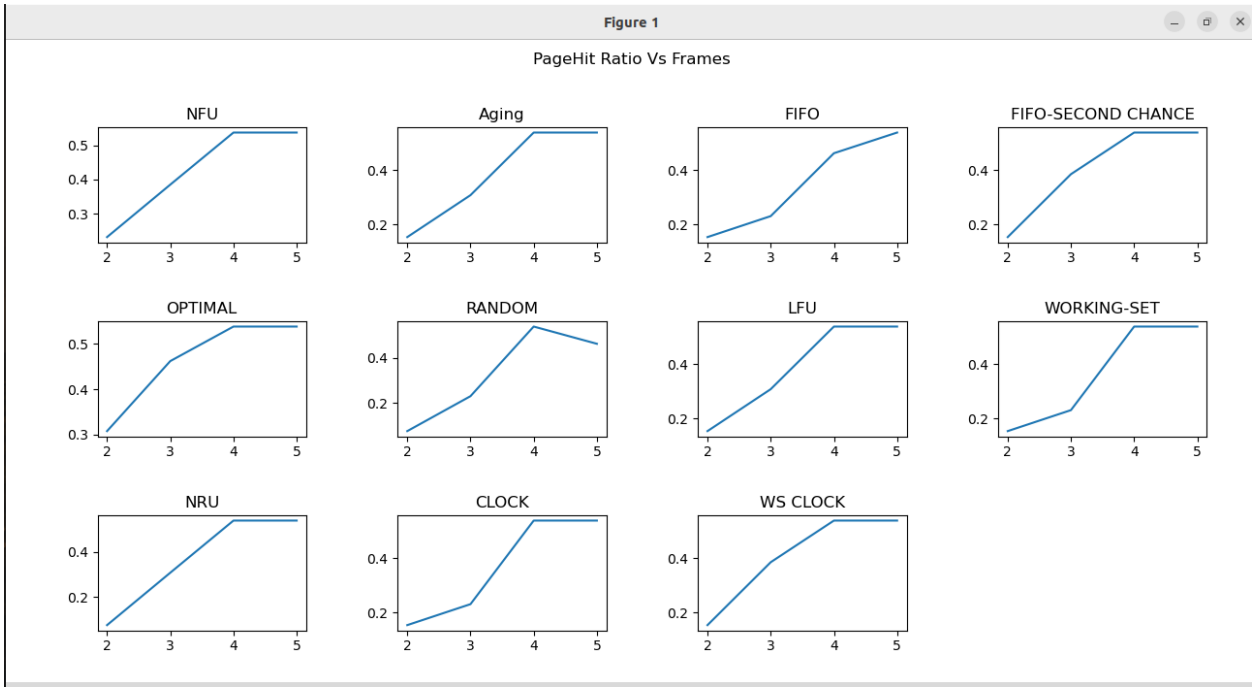
The project helped us understand the concept of how virtual memory can address more memory than the total amount of physical memory (RAM). Virtual memory provides an abstraction of the storage resources that are actually available on a given machine, which creates the illusion to the users of a very large main memory.

Lastly, we learnt how to plot graphs in Python using Matplotlib.

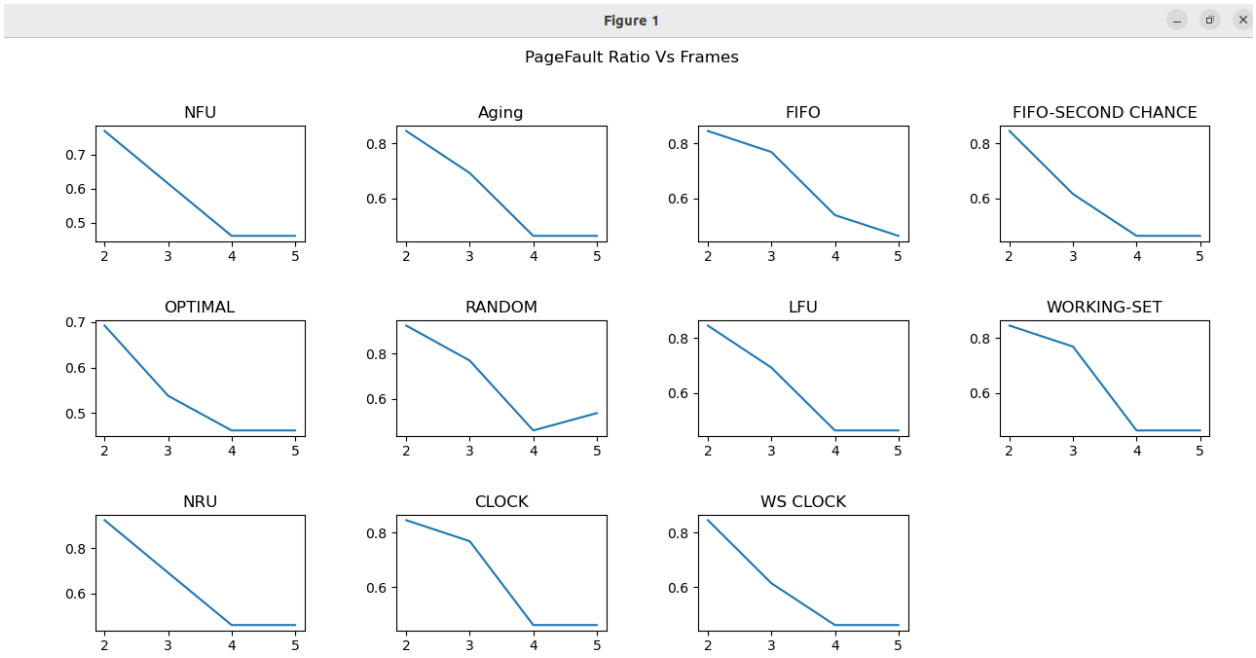
Result

For each algorithm, the page reference order and dirty bits are read and page hit and page fault ratios for different numbers of frames are written to an output file, which is further used to create the graphs shown below.

Number of Frames vs Page Hit ratio



Number of Frames vs Page Fault ratio



Assumptions

Reference bit is set when a page already in memory is accessed again.

Input format :

- 1st line : 2 space separated integers N, M mentioning number of frames and the number of pages in the stream
- 2nd line : M space separated integers denoting the order of page requests
- 3rd line : M space separated integers denoting the dirty bit (0/1) for the ith page.

Conclusion

Theoretically, Optimal page replacement algorithm (OPT) is the best performing because it replaces the page that will be referenced last from among the current pages. This algorithm cannot be implemented in a general OS because it is impossible to compute reliably how long it will be before a page is going to be used. It is useful as a benchmark against which performance of other algorithms can be compared.

NRU favours keeping pages in memory that have been recently used. To this end, it divides the pages into 4 classes depending on the value of reference bit (R) and modify bit (M) and replaces a random page from the lowest class possible.

FIFO is a low-overhead algorithm. OS keeps track of all the pages in memory in a queue, with the least recent in the front and the most recent at the back. When replacement is required, the page at the front of the queue (oldest page) is replaced. While it is cheap and intuitive, it performs poorly in practical applications. Thus, it is rarely used in its unmodified state.

FIFO Second-chance is a modified version of FIFO and it fares better than FIFO at little cost of improvement. An old page that has been referenced is probably in use, and should not be swapped out over a new page that has not been referenced.

Clock is a more efficient version of FIFO than Second-chance because pages don’t have to be constantly pushed to the back of the list.

LRU works on the idea that the pages that have been most heavily used in the past few instructions are most likely to be used heavily in

the next few instructions too. While LRU can provide near-optimal performance in theory, it is rather expensive to implement in practice.

Random replacement algorithm replaces a random page in memory. This eliminates the cost of tracking page references. Usually, it fares better than FIFO.

NFU requires a counter and every page has a counter associated with it (initially 0) that keeps track of how frequently a page has been used. The page with the lowest counter is swapped out when required. Its main problem is that it keeps track of frequency of use without regard to the time span of use. This results in poor performance.

Aging algorithm is a variant of NFU with modifications to make it aware of the time span of use. Page references closer to the present time have more impact than page references long ago. This ensures that pages referenced more recently , though less frequently referenced, will have higher priority over pages more frequently referenced in the past. Aging can offer near-optimal performance for a moderate price.

Working set algorithm has reasonable performance but it is somewhat expensive to implement.

WSClock is a variant of Clock and Working-set which gives good performance and is efficient to implement.

Overall, aging and WSClock give near-optimal performance and can be implemented efficiently. These two are probably the most important page replacement algorithms.

Team - Aos_project_1234

Dishant Sharma (2022202019) - LRU, NRU, Random, Working Set
Jugnu Gill (2022201011) - Clock, WSClock
Udrasht Pal (2022201020) - Aging, NFU, Optimal
Vikram Raj Bakshi (2022201044) - FIFO, FIFO Second Chance

https://github.com/Udrasht/Page-replacement-policies-simulation