



VIRTUAL MEMORY MANAGEMENT SIMULATOR

Team Members

- Aditya Sharma
- Suyash Chintawar

Introduction and Objective

- This project deals with the creation of a Virtual Memory Management Simulator by utilising the various concepts of Virtual memory such as different fetch policies and page replacement algorithms.
- Fetch policy determines when a page should be brought into main memory. In our virtual memory management simulator, the following policies have been implemented:
 - Demand paging
 - Prepaging
- Page replacement algorithms are used to select the page that is to be replaced when the main memory is full and a new page has been requested.

Introduction and Objective (Cont...)

- The following page replacement algorithms will be implemented in our virtual memory management simulator
 - FIFO (First In First Out)
 - LRU (Least Recently Used)
 - Clock
- The entire simulation of Virtual memory management will be then displayed through a GUI.

Literature Survey

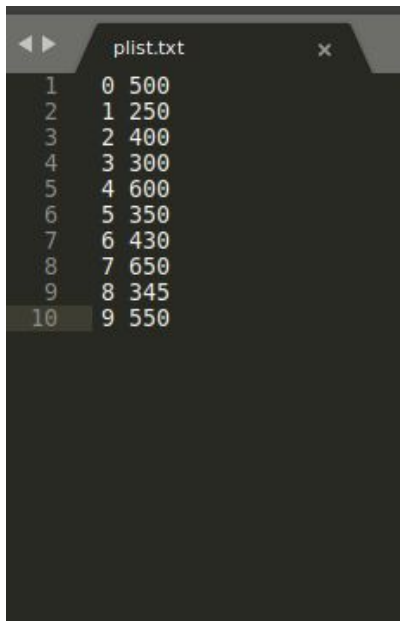
- In a paper, the virtual memory management system is simulated and its properties were verified with MSVL which is a parallel programming language.
- The system mentioned above manages the virtual memory by handling the pages and uses the aging algorithm to find a page to swap out when a page fault occurs and there is no idle page.
- Another study had modified the MOSS simulator and developed a virtual memory simulator which allowed the user to switch between different page replacement algorithms and analyse and compare the number of page faults that occur in each of the algorithms.
- The results in the above model showed that the optimal algorithm produces the least number of page faults and that LRU is generally better than FIFO.

Implementation

- Backend - C++
 - Input page size, fetching policy and replacement algorithm
 - Reading Plist and Ptrace
 - Calculating the number of page faults
- Frontend - Python + Tkinter
 - Getting data from backend using subprocess call
 - Running simulation for alternative algorithms
 - Generating statistics
 - Rendering GUI using Tkinter

Input Format

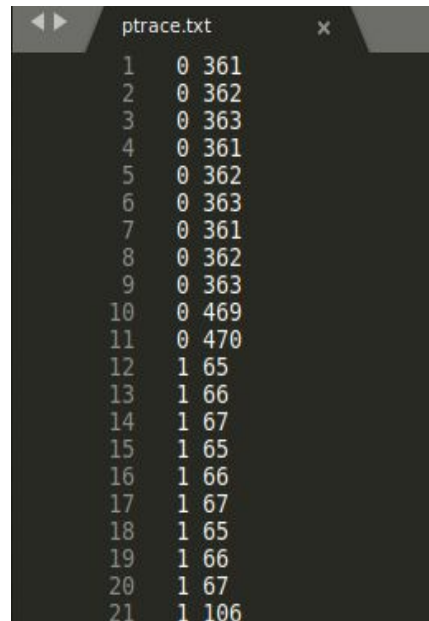
Process List



1	0	500
2	1	250
3	2	400
4	3	300
5	4	600
6	5	350
7	6	430
8	7	650
9	8	345
10	9	550

Each line has the format
(process_id, total # of memory locations)

Process Trace



1	0	361
2	0	362
3	0	363
4	0	361
5	0	362
6	0	363
7	0	361
8	0	362
9	0	363
10	0	469
11	0	470
12	1	65
13	1	66
14	1	67
15	1	65
16	1	66
17	1	67
18	1	65
19	1	66
20	1	67
21	1	106

Each line has the format
(process_id, referenced memory location)

Sample Input

Virtual Memory Management Simulator

VIRTUAL MEMORY MANAGEMENT SIMULATOR

Process list :

Process trace:

Fetch Policy: Replacement Policy:

Page Size:

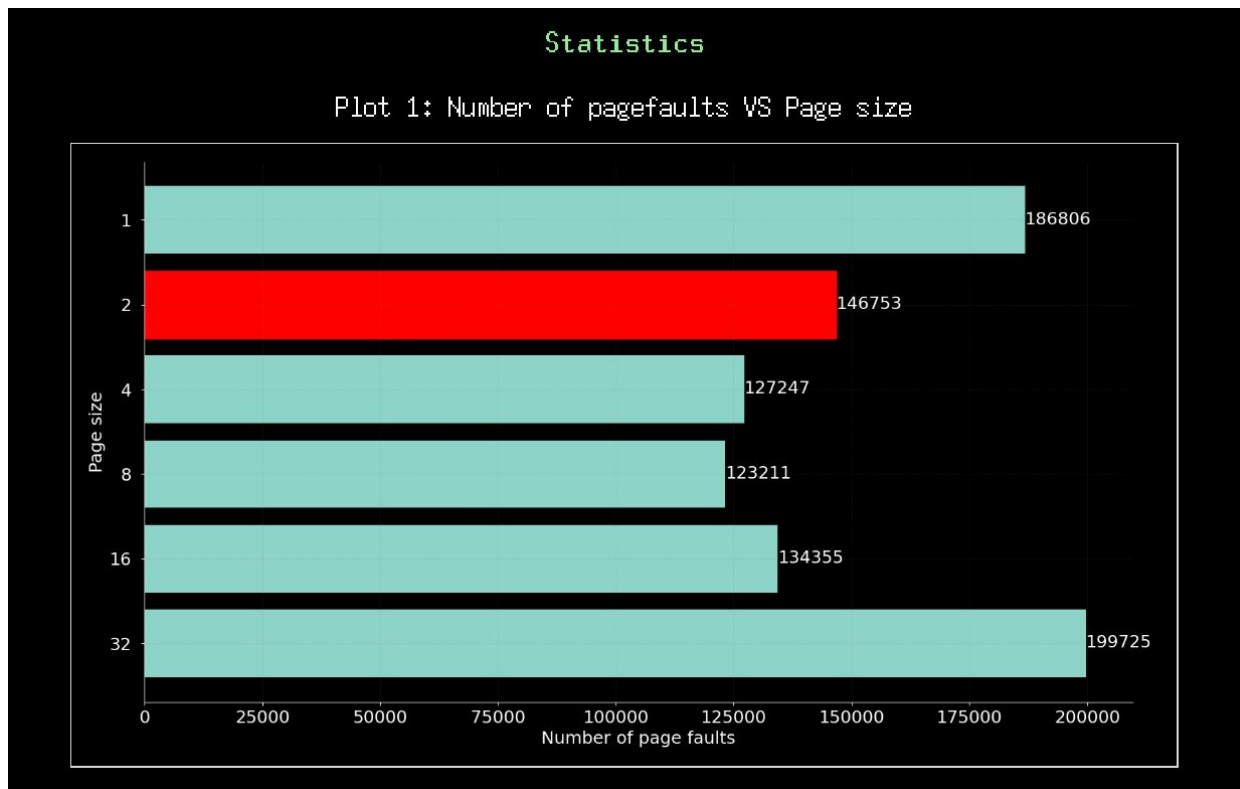
Sample Output

VIRTUAL MEMORY MANAGEMENT SIMULATOR

Results

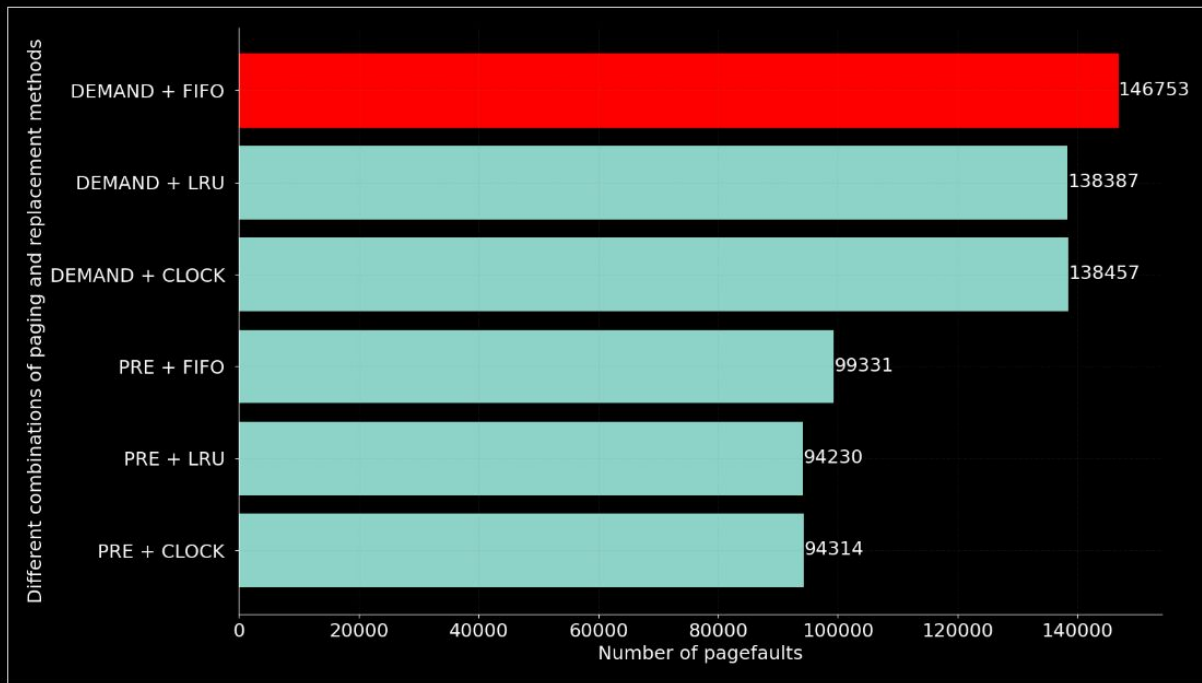
Number of Processes	:	10
Number of Memory Requests	:	1100110
Fetch Policy	:	DEMAND
Replacement Policy	:	FIFO
Page size	:	2
Number of Page Faults	:	146753

Sample Output - Plot 1



Sample Output - Plot 2

Plot 2: Number of pagefaults for different combinations of paging and replacement methods



Experimentation and Analysis

Page Size	Demand Paging			Pre-paging		
	FIFO	LRU	Clock	FIFO	LRU	Clock
1	186806	185262	185442	100900	99712	99887
2	146753	138387	138457	99331	94230	94314
4	127247	114975	115015	106345	93063	93069
8	123211	103523	103645	125463	93768	94387
16	134355	97981	103491	190575	177377	190940
32	199725	200234	199992	184549	184549	184549

Fig. Number of page faults for different combinations of fetch policies and replacement algorithms

Experimentation and Analysis

- When the page size is extreme, i.e. too high or too low, then, usually all the replacement policies show similar behavior, for a given page fetch policy. Observing the collected statistics for page sizes 1 and 32, it is evident that the number of page faults is almost close for each of the page replacement policies for a given page fetch policy.
- Further, when the page size is small, it can be observed that, generally, pre-paging produces much better performance as compared to demand paging.
- Also, when the page size becomes too large, the performance of pre-paging reduces and in some cases it is out-performed by demand paging.

Experimentation and Analysis

- When the page size is intermediate, i.e. neither too high nor too low, then, it can be observed that FIFO replacement policy produces the maximum number of page faults whereas Clock and LRU generally produce a lesser number of page faults. Although the number of page faults produced by both Clock and LRU replacement policies is both close, generally, LRU produces a lesser number of page faults as compared to Clock. This shows that in most cases, LRU performs the best, closely followed by Clock whereas FIFO generally gives the worst performance.
- Also, in case of intermediate page size, again pre-paging usually produces better results as compared to demand paging as the number of page faults produced for pre-paging for a given replacement algorithm is generally lesser than the number of page faults produced by demand paging.
- Further, for a given combination of page fetch policy and page replacement policy, it is observed that as the page size increases, the number of page faults initially decreases and then increases (Fig 1 - 6). When page size is very small, then the number of page faults is quite high. The number of page faults decreases as the page size is increased till a certain point, after which, the number of page faults again increases.

Thank You