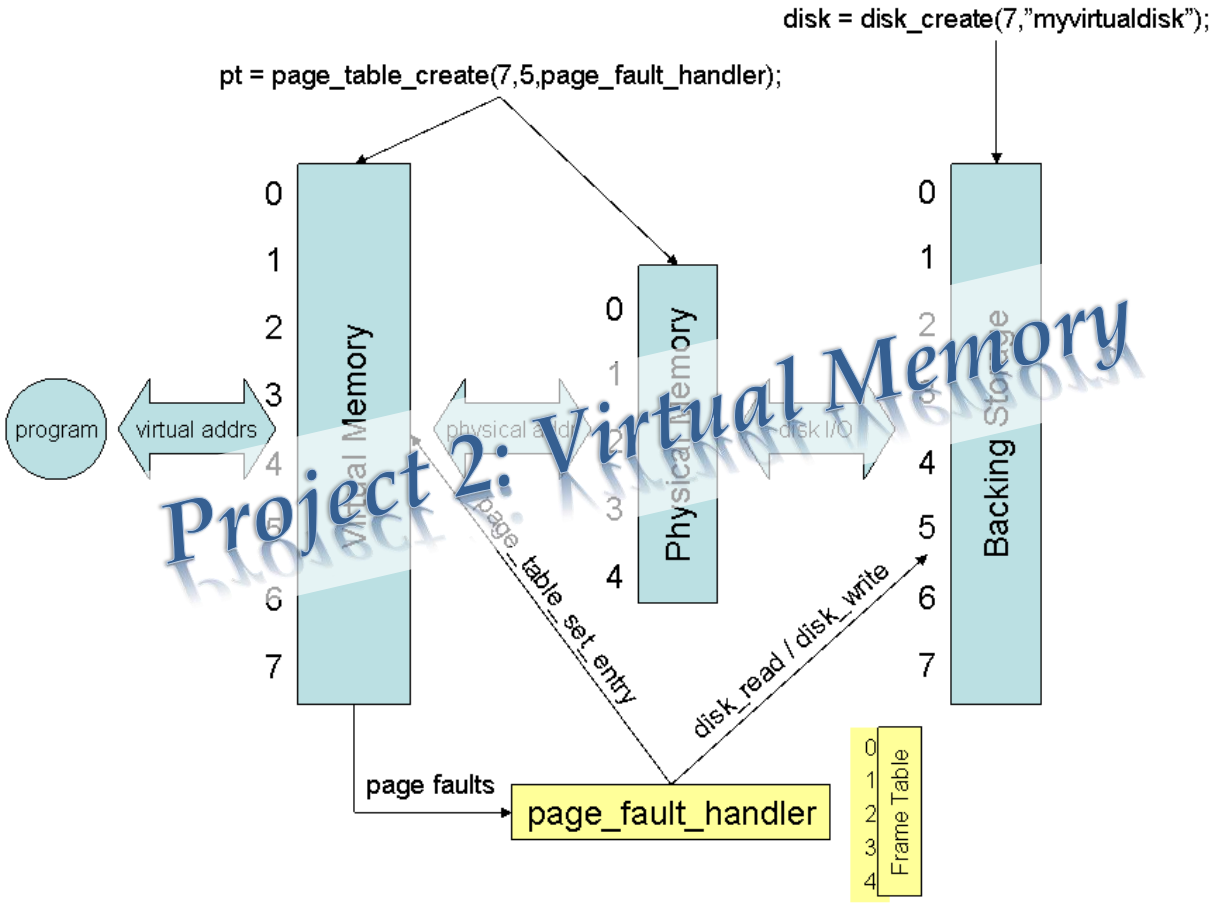
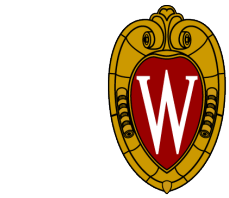
CS 537 - Introduction to Operating Systems



Project 2: Virtual Memory



Aman Rakesh Chadha
(9068354597)

1 Overview

a Purpose

The work seeks to implement a simple but fully functional demand paged virtual memory model with several page replacement policies:

- ✓ Random
- ✓ FIFO
- ✓ Second Chance FIFO
- ✓ Custom Algorithm (for flash drives)

b Experimental setup

We implemented the experiments on a Mumble CSL workstation. Fixing the number of pages to 100 and by incrementing the number of frames from 10 to 100 (in steps of 10), we obtained our results. A special case, i.e., # of frames = 5, was also taken into consideration for all page replacement algorithms and all programs.

2 Custom Page Replacement Algorithm

In the custom part, we use a 2-Dimensional Array data structure to store state information. The number of rows is equal to the number of frames available in Physical Memory. It has four fields in the column section:

1) *Frame Number (FRAME_NUM)*
2) *Reference (REF)*: Incremented everytime a page is accessed (written to, or read from disk)
3) *Read/Write (READWRITE)*: Set to '1' for read-only permissions and '3' for read-write permission
4) *Valid Bit (VALID)*: Set whenever a page is accessed for the first time (cold-start miss) and remains set throughout.

This data structure is always updated whenever a page is evicted or written to. To find the frame to be replaced, we do the following:

1) In general, the algorithm is based on the principle that pages with READ permissions are given priority for eviction as compared to pages with READ|WRITE permissions. However, enforcing this policy strictly in all cases results in an inordinately large number of reads.
2) On the other hand, the number of reads decrease if we evict a certain number of READ|WRITE pages first and then begin to evict READ pages.
   Thus, two cases arise:
   Case 1: If the number of frames are equal to or more than 50% of the number of pages, for instance: 100 pages and 90 frames, evicting all READ pages first before starting to evict READ|WRITE pages gives lesser writes with a little more reads. Thus, in this case, the algorithm principle is being enforced strictly.
   Case 2: If the number of frames are less than 50% of the number of pages, for instance: 100 pages and 10 frames, evicting READ|WRITE pages first until the number of READ pages are 25% of the number of frames, and then starting to evict READ pages gives better performance. In this case, the threshold (25%) was obtained experimentally to serve as a tradeoff between least number of writes and maximum number of reads; as well as to comply with the performance requirements (< 10% more disk reads on average across all three programs as compared to 2FIFO, and a reduction in the number of disk writes as much as possible).
3) To select a page for eviction amongst pages with the same READ|WRITE permissions (either 1 or 3), we implemented FIFO using the recorded reference information.

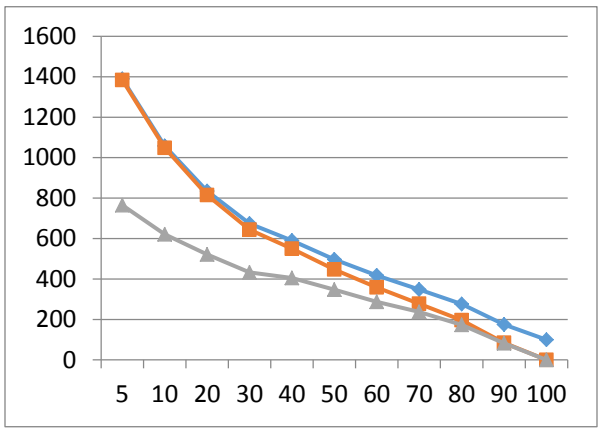
3 Results

Page Faults Disk Reads Disk Writes

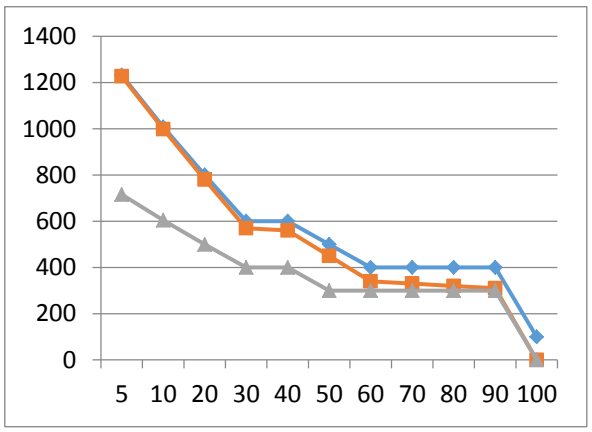
Command Invoked: **`./virtmem A B C D`**

`A: # of pages; B: # of frames; C: rand/fifo/2fifo/custom; D: sort/focus/scan`

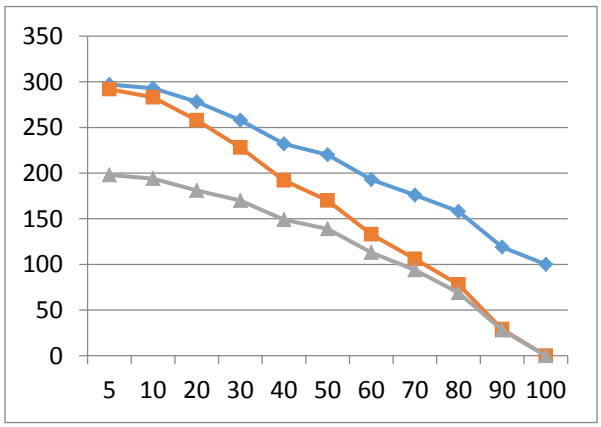
a RAND: Sort



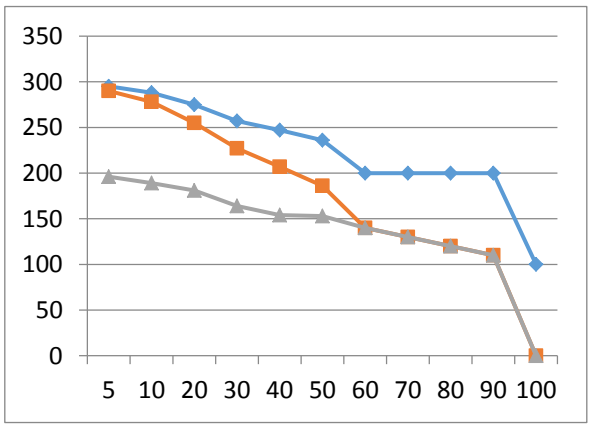
e FIFO: Sort



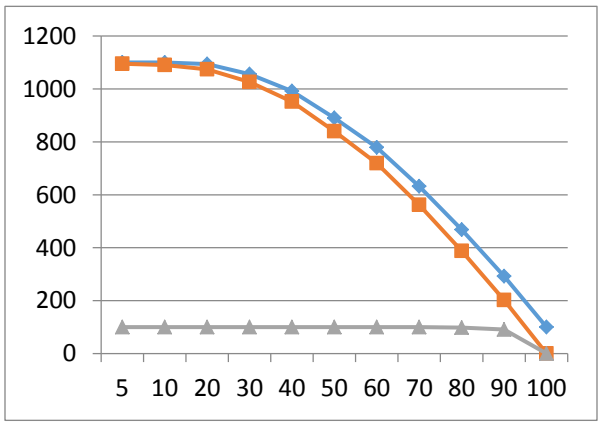
b RAND: Focus



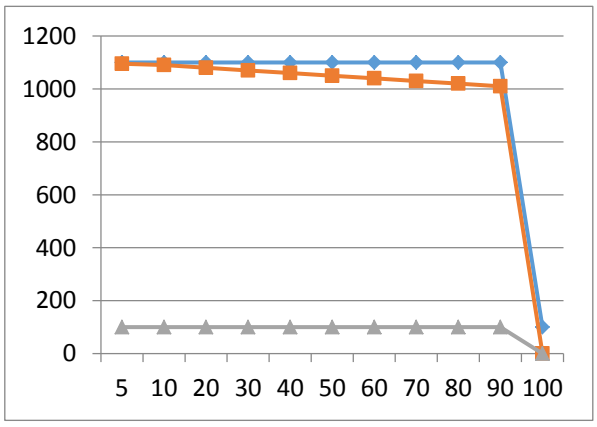
f FIFO: Focus



c RAND: Scan

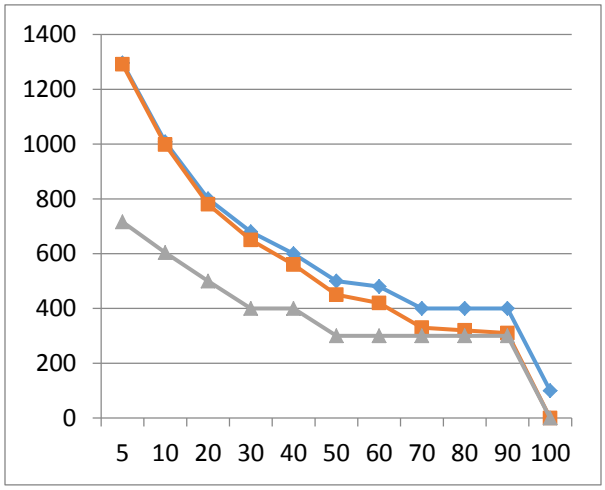


g FIFO: Scan

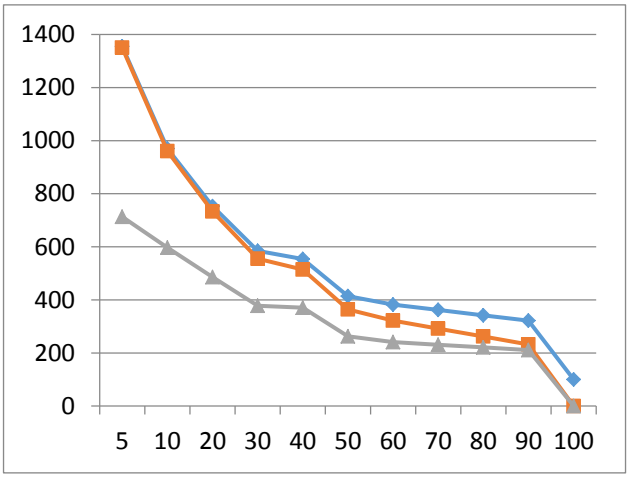


4 Results (Continued…)

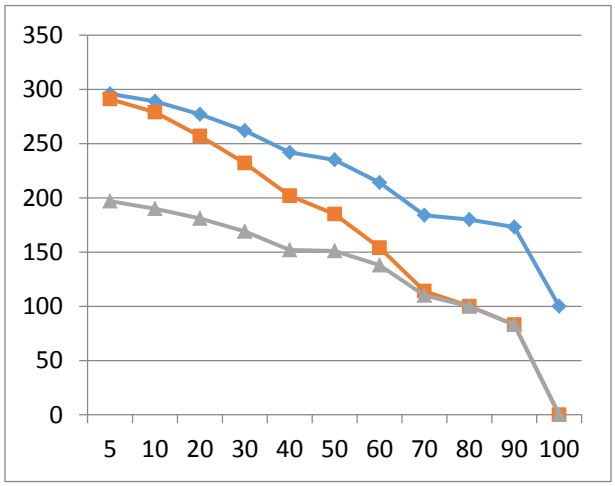
h 2FIFO: Sort



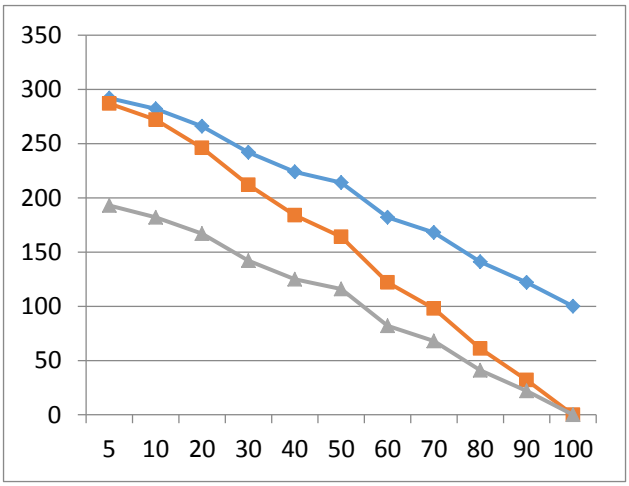
k Custom: Sort



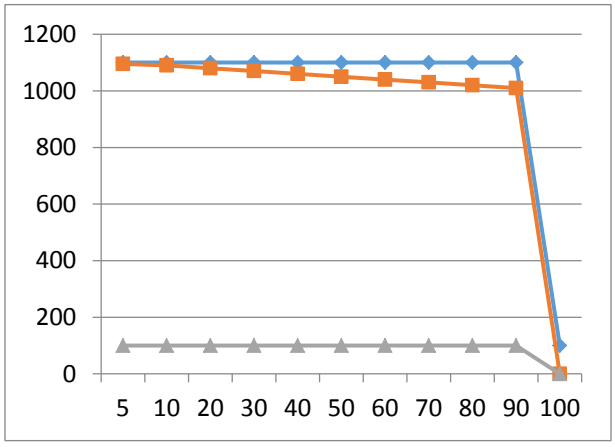
i 2FIFO: Focus



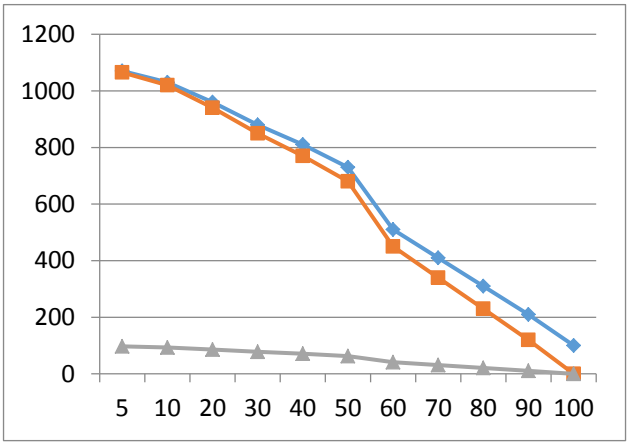
l Custom: Focus



j 2FIFO: Scan



m Custom: Scan



5 Analysis of Results

For each of these programs, there are loops that simply iterate through addresses making both the FIFO and 2nd Chance FIFO Algorithms perform more poorly. Each of these algorithms tended to result in pages getting evicted from memory just before they were about to be accessed again. However, this did not affect the Random Algorithm, allowing it to perform better in most cases. All of the algorithms performed better when more frames were available. Both hard and soft page faults were taken into account while drafting and analyzing results.

`#1 = Random; #2 = FIFO; #3 = 2FIFO; #4 = Custom; R = Reads; W = Writes; FR = Frames`

SORT

FR	R #1	R #2	R #3	R #4	W #1	W #2	W #3	W #4
5	1384	1227	1291	**1350**	765	716	716	**713**
10	1048	998	998	**960**	621	604	604	**597**
20	815	780	780	**733**	521	500	500	**486**
30	644	570	650	**555**	433	400	400	**378**
40	550	560	560	**514**	405	400	400	**371**
50	447	450	450	**364**	347	300	300	**263**
60	358	340	420	**322**	287	300	300	**241**
70	278	330	330	**292**	237	300	300	**231**
80	196	320	320	**262**	173	300	300	**221**
90	85	310	310	**232**	82	300	300	**211**
100	0	0	0	**0**	0	0	0	**0**

The sort program performs better for both of the FIFO Algorithms. Sort uses quicksort to sort a number of random integers. The quicksort algorithm loops through addresses in the same area more often than the scan program does. For FIFO and 2nd Chance FIFO, this allows pages to be referenced many times before they are removed from the end of their queues. It also takes advantage of the 2nd Chance FIFO more often than scan does.

FOCUS

FR	R #1	R #2	R #3	R #4	W #1	W #2	W #3	W #4
5	292	290	291	**287**	198	196	197	**193**
10	283	278	279	**272**	194	189	190	**182**
20	258	255	257	**246**	181	181	181	**167**
30	228	227	232	**212**	170	164	169	**142**
40	192	207	202	**184**	149	154	152	**125**
50	170	186	185	**164**	139	153	151	**116**
60	133	140	154	**122**	113	140	138	**82**
70	106	130	114	**98**	94	130	110	**68**
80	78	120	100	**61**	69	120	100	**41**
90	29	110	83	**32**	28	110	83	**22**
100	0	0	0	**0**	0	0	0	**0**

The focus program looked up addresses in memory at random. This again allows FIFO and 2nd Chance FIFO to perform better than they did with scan. The 2nd Chance FIFO caught more frames on its second chance list being referenced and often had fewer page faults than FIFO because of this. Random performed well again and was fairly consistent across all of the programs.

SCAN

FR	R #1	R #2	R #3	R #4	W #1	W #2	W #3	W #4
5	1095	1095	1095	**1065**	100	100	100	**97**
10	1090	1090	1090	**1020**	100	100	100	**93**
20	1074	1080	1080	**940**	100	100	100	**86**
30	1026	1070	1070	**850**	100	100	100	**78**
40	952	1060	1060	**770**	100	100	100	**71**
50	840	1050	1050	**680**	100	100	100	**63**
60	719	1040	1040	**450**	100	100	100	**41**
70	562	1030	1030	**340**	100	100	100	**31**
80	388	1020	1020	**230**	98	100	100	**21**
90	202	1010	1010	**120**	91	100	100	**11**
100	0	0	0	**0**	0	0	0	**0**

The scan program does not have much of a random element, but simply iterates through addresses. For this reason, FIFO and 2nd Chance FIFO tend to evict pages before they are referenced a second time perform very poorly while random increases the chances that a page will still be in memory the next time it is referenced.

The Custom Algorithm offers 20.98% lesser reads and 24.80% lesser writes than 2FIFO on average across all three programs. In all cases, it offers the least number of writes compared to other algorithms and in most (99%) cases, it offers the least number of reads and page faults.