Names: Ashritha Nagavaram (nagav001) , Gaurav Khandelwal (khand052), Gopal Sarda (sarda014)

The purpose of the experiment is to understand the concept of memory mapped files and demand paging, and evaluate the performance of different page replacement algorithms. Code that implements “virtual” page table and “virtual” disk has been provided. We are supposed to write a page fault handler that traps page faults, fetches corresponding data from the disk and updates the page table with the required page. In case, the page table is full, we need to replace an existing page table entry using user specified replacement algorithm.
The experiments were performed on Linux machine, Ubuntu (Linux quasar 3.11.0-26-generic).
Compile command : Make -f Makefile.txt
Command line arguments : ./virtmem npages nframes rand|fifo|custom scan|sort|focus
example - ./virtmem 100 10 rand scan

In our experiment, if there are free available frames, we will grant it to a page. For that, we have maintained an array, whose index gives us frame number and value at that index gives us page number at that frame. If all the frames are occupied, we will replace page, based on replacement algorithm mentioned by user in command line arguments. During page fault, if the page has no permissions we are granting it as PROT_READ permissions and if the page has PROT_READ, then we are granting it PROT_WRITE permission. Also, when replacing pages with PROT_WRITE permission, we need to first write back the updated value to the disk. The different replacement algorithms implemented are :
1) Random : We randomly select a frame using rand() and replace the page in that frame.
2) FIFO : We are maintaining a linked list to keep track of the order in which frames are accessed. The frame present at the head of the linked list will be the one which was accessed earliest and can be replaced. Once replaced, the frame is removed from the linked list.
3) Custom : The custom page replacement algorithm works on the basis of space locality i.e. the pages around recently accessed page will be accessed again so we need to replace the page that is far from the recently accessed page. Whenever a page is added to the frame, the frame number is stored in the list. If a page fault occurs,the algorithm starts by sorting the pages present in the physical memory based on the page number. Once we have the sorted list, we then check for the frame that is recently added. The algorithm searches for the page that is located spatially far from the page recently accessed and then replaces that particular page.

1. Sort the existing pages based on the page numbers
2. Check for the page number that is added recently.
3. From the sorted list search for the page that is far away from the recently accessed page and replace it i.e. if the index of the recently accessed page in the sorted list is i , compare difference between the index i, 0 (starting index) and index i, lastindex. we then get the page that is far away (having maximum difference) from recently accessed one, return the frame number to be replaced.

In general, we observed that custom algorithm implemented by us performed better than fifo in terms of the number of page faults, disk reads and disk writes. Another pattern observed was that if the data is sequential, fifo and custom performed similar. In cases where the access pattern is repeated, custom algorithm performs better than fifo. The reason for better performance is because if your process is frequently accessing a page, it should not be paged to disk, even if it was the very first accessed page. Additionally, there are greater chances of a process accessing a page closer to the last accessed page because of spatial locality.

For eg: If there are 4 pages and 3 frames , and the access pattern of pages is 1 2 3 4 1 2 3

Page accessed	fifo algorithm | list	custom
1	1	1
2	1 2	1 2
3	1 2 3	1 2 3
4	2 3 4	2 3 4
1	3 4 1	3 4 1
2	4 1 2	3 1 2
3	1 2 3	No replacement
Total # of page faults	7	6

replacement Algorithm	Program	# of Pages	# of Frames	# of page faults	# of disk reads	# of disk writes
rand	sort	100	35	1053	624	429
fifo	sort	100	35	997	599	398
custom	sort	100	35	981	602	362
rand	scan	100	50	995	895	100
fifo	scan	100	50	1200	1100	100
custom	scan	100	50	960	860	76
rand	focus	100	12	486	293	193
fifo	focus	100	12	475	287	188
custom	focus	100	12	477	286	191
custom	scan	13	10	96	83	8
fifo	focus	100	90	311	200	110
rand	sort	45	3	1048	705	343
custom	sort	90	9	1473	954	515
custom	focus	240	100	778	478	280

Testing description, including a list of cases attempted (also must include negative cases) and the results

a. When number of pages and frames are negative.
b. When number of pages are equal to number of frames
c. When number of pages are less than number of frames
d. When number of pages are slightly greater than number of frames and both are small
e. When number of pages are slightly greater than number of frames and both are large

f. When number of pages are significantly greater than number of frames and number of frames are small and number of pages are not very significantly high.
g. When number of pages are significantly greater than number of frames and number of frames are small and number of pages are significantly high.
h. When number of pages are significantly greater than number of frames and number of frames are high and number of pages are also very high.
i. All above experiments are repeated for all replacement algorithms and all program type
j. When we intentionally type incorrect command parameter