LAB 4

Id : 5692

Name: ahmed elsayed

Problem statement:

implementing page replacement algorithm OPTIMAL, FIFO, LRU, or CLOCK

**Main function** :

We start by declaring ( space = which holds the amount of frames in the memory, method = which holds which type of algorithm to use , array = which holds the page replacements requested from the main memory )

Then we compare the value with method to execute one of the 4 algo. Functions

**Some helpful functions** :

Searcharray function is used to find if a given value is present in the array or not

Biggestcountindex function is used to find the index of the biggest value in an array

Findindex function is used to find the index of a given value in the array

**fifo function** :

start by declaring memory array , faults and currentindex -> used to point to which frame will be used to put page number in it

first for loop to iterate over the page numbers in the array of pagenumbers

now we have two situations either the main memory is full or not

**if not full** then search if the value to be put is in the memory if it print it if not then add it to the memory and then print it

**if full** search if the value is in the memory if it is then print it if not then

add it to the memory in the current index location and inc current index

**clock function** : similar to fifo same variable the difference in line 318

case where the memory is full and we start replacing pages

here we have an array called counts which holds the used bit for each frame in the memory that array is initlized with 1 value

if the page is not in the memory then

do an inifinity loop while(1)

if the counts of the current index is 0 replace it and set the count of that index to 1 and inc the current index

if counts is 1 then set it to 0 and inc the current index

infinity loop until we reach the index which has counts 0 and break from

that loop

**Optimal function** :

Same as other functions until we reach line 139 where the memory is full and we need to replace

Here the counts array will hold how many steps we took to find the value of the index in the array of page replacments

We start by looping over the memory frames

In the loop we have another loop that loops over the rest of the array of page replacments and search for the value if its found add to counts how many steps we needed to get to that value

After looping over all the memory frames we compare the steps we needed to find them in the array of page replacement and the memory frame who’s page value has the biggest step count will be removed and replaced with another page .

**LRU function** : the same as the optimal algorithm but instead of searching the steps forward we search for them backwards and the one with the biggest steps will be removed



