CSE 312 FINAL

REPORT

İBRAHİM AKBULUT
151044077

First of all, i will say what algorithms that i didn't implement. I didn't implement WSClock algorithm.

Page table I designed for the other 4 algorithms to work:

Page table entry;

char isPageModified	**char** isPageReferenced	**char** isPagePresent	**int** pageNumberInRam	**long long int** İnstruction_counter_number

isPageModified: This variable determine whether a page is modified or not.

isPageReferenced: This variable determine whether a page is referenced or not.

isPagePresent: This variable determine whether a page is in the pysicall memory or not.

İnstruction_counter_number: This variable is used for aging the page. Which mean whenever a page is loaded to memory current instruction number is writed to that page. So we know a page's age. This field is meaningful only for **LRU**. This counter is 64 bit.

I am using a queue for FIFO and SC algorithm. So this queue object is created before program starts. Of course this queue object is not used for other algorithms.

I have a instruction counter as global so i can count instructions which mean accessing the memory with **get** and **set** functions.

I guess i did mention about necessary elements for page table and replacement algorithms.

So, i am using a page table entry C array. I allocate space dinamically for this array after my program starts according to given user parameters.

My pysicall memory is also an array, int C array. Size of this array determined after program starts acording to given user parameters.

Functions;

set(unsigned int index, int value, char *tName):

This function access the memory for changing. According to tName different behaviours occurs in this function.

For 4 sorting thread ; it convert index to page table index. Then check page_table[page table index]'s isPagePresent value. Mean it checks that correspondenig page is in memory or not. If page is in the memory then it changes corresponding index's value and set isPageModified of page table[page itable index] to 1.

If page is not in the memory then it call page replacement function. Then set isPageModified of page table[page itable index] to 1. By the way number of read and number of writes counters are incremented. If page replacement algorithm is NRU then isPageReferenced field of all page table entries are set to 0(line 1894 in sortArrays.cpp). Of course if it's time.

For check thread, function returns a dump value immediately. Because check thread is never gonna call the **set** function.

For fill thread, just write the value to virtual memory(disk). Another important thing about this function; more than 1 thread can not enter to this function on same time. Because, all works in this function are critic. So i am using a lock mechanism. Also i call print_page_table function(line 1665 in sortArrays.cpp) in the begin.

Since I saw fill and check functions as creating and controlling virtual memory, I did not keep any statistical information for these functions other than number of write and number of read.

get(unsigned int index, char *tName):

It is very similar to the set function. One of the differences; While writing a value in the set function, a value is drawn in the get function. Page replacement processes that need to be done here are done in the same place and conditions as the set. The other difference is while returning a dump value for check thread(actually this not a thread but in the hw documention it is) in the set , dump value is returned for fill thread in this functions.

page_replacement (int index_of_page_in_virtual_memory,char *tName,char *pageReplacementt):

Behaviour of this function changes according to given page replacement algorithm and allocation policy.

- **Case page replacemnt algorithms is FIFO(line 82 in sortArrays.cpp):**

-if pysicall memory is not full then just draw page to memory

-if pysicall memory is full and allocation policy is global:

There is queue as fifo. Remove head of this fifo and add related page's info to queue.

-if pysicall memory is full and allocation policy is local:

Look for a page that has related to current thread. İf that page is found then replace it with corresponding page. But if a related page is not found then apply like global FIFO page replacement.

- **Case page replacement algorithm is SC (line 644 in sortArrays.cpp)**

-if pysicall memory is not full then just draw intended page to memory

-if pysicall memory is full and allocation policy is global:

Take head of FIFO. Check it's referenced value. İf it is 1 then set referenced bit of it to 0 and add these to end of fifo. Then check next head of FIFO. This process goes until a page found that has 0 referenced value. Then remove that page from FIFO and intended page's info FIFO.

-if pysicall memory is full and allocation policy is local:

Look for a page that has related to current thread. İf that page is found then replace it with corresponding page. But if a related page is not found then apply like global SC page replacement.

- **Case page replacement algorithm is LRU (line 933 in sortArrays.cpp)**

Fort this algorithm i used instruction counter. I increment instruction counter on each set and get. And assign current counter to new page in the memory(taked to memory with page replacement function).

-if pysicall memory is not full then just draw intended page to memory

-if pysicall memory is full and allocation policy is global:

Search for page table for page that has lowest instruction counter and has isPagePresent value as 1. If that page is found. Then set is pagePresent value to 0. Because it wont be in the memory. Set isPagePresent value of intended page in page tablet to 1. Because we take this page to memory. Of course modified and referenced values are also set to 0;

-if pysicall memory is full and allocation policy is local:

Look for a page that has related to current thread. İf that page is found then replace it with corresponding page. But if a related page is not found then apply like global LRU page replacement.

- **Case page replacement algorithm is NRU (line 1144 in sortArrays.cpp)**

this algorithm actually works very simply. The page with the desired feature is searched with 4 loops in a row. The order of the loops is determined by the priority of the desired feature. This feature sequence was shown as 4 classes in the course.

From time to time, all r bits in the page table are reset.

-if pysicall memory is not full then just draw intended page to memory

-if pysicall memory is full and allocation policy is global:

Search for page in table that has desired class properties. Apply changing process for founded page and intended page.

-if pysicall memory is full and allocation policy is local:

Look for a page that has related to current thread. İf that page is found then replace it with corresponding page. But if a related page is not found then apply like global NRU page replacement.

And there are other sorting algorithms which i dont mention about them in here.