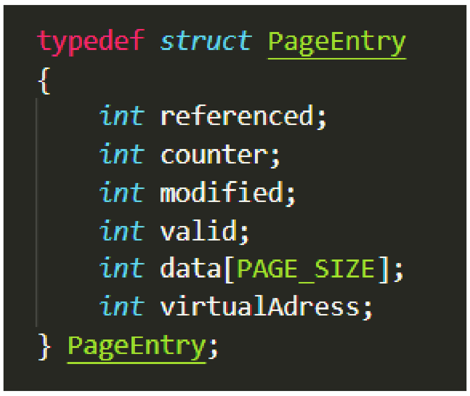
GIT Department of Computer Engineering CSE 312/504 - Spring 2022
Homework #2
Report

Sefa Çiçek
1801042657

1. Design Decisions and Main Structure

Page Entry structure for pages :

Referenced bit : If page is in physical memory, referenced bit equals to 1, otherwise 0. When page replacement occurs and a page is removed from physical memory, we must set the referenced bit of the page to 0.



```c
typedef struct PageEntry
{
    int referenced;
    int counter;
    int modified;
    int valid;
    int data[PAGE_SIZE];
    int virtualAdress;
} PageEntry;
```

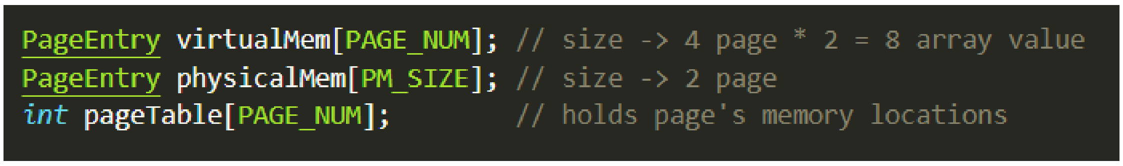
Modified bit : If data's of a page has changed because of the sorting, must set the modified bit of the page to 1. According to the modified bit of a page, determine whether page must be written to the disk or not, when page replacement occurs.

Counter : It is used for LRU Algorithm. On swap operations(clock tick), referenced bit of the page is added to the counter.

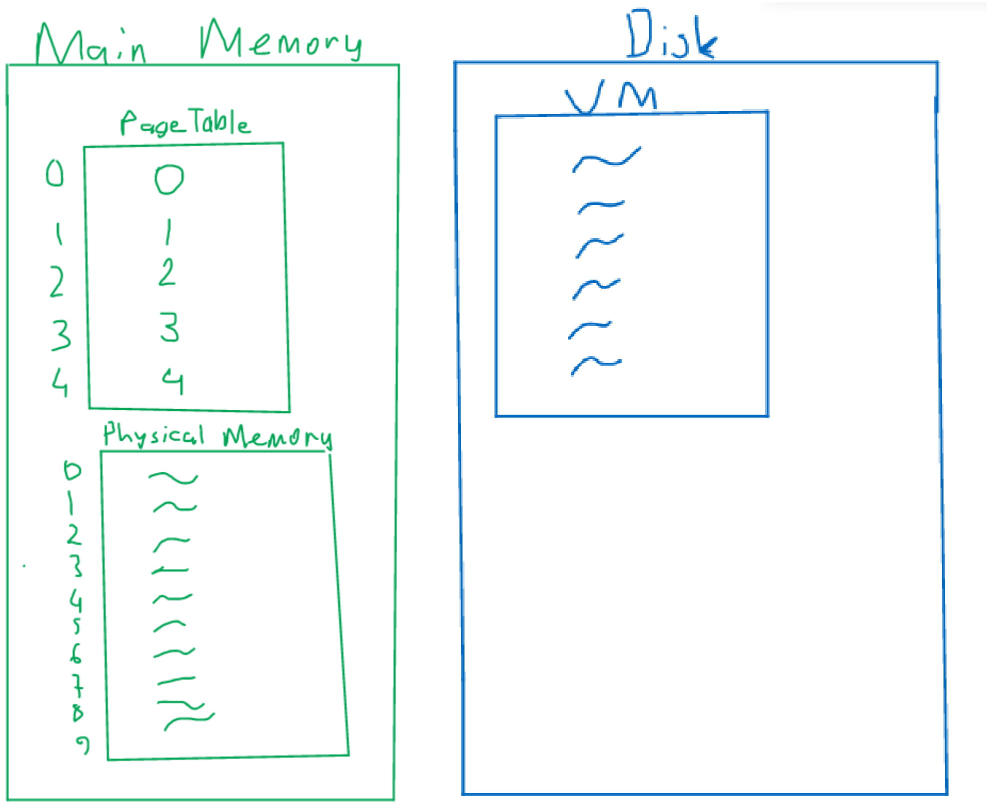
Valid : It checks whether the page is in memory or not.

Data[] : It holds data's of the page which are values in the array. (5 , 8, 3…)

Virtual address : Thanks to this parameter, we can easily find the virtual memory location of a page although the page is in physical memory.



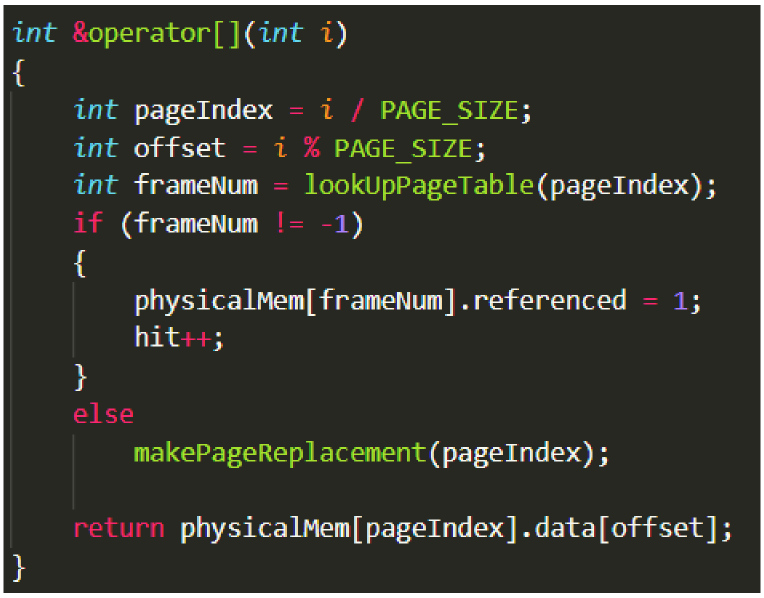
```c
PageEntry virtualMem[PAGE_NUM]; // size -> 4 page * 2 = 8 array value
PageEntry physicalMem[PM_SIZE]; // size -> 2 page
int pageTable[PAGE_NUM];        // holds page's memory locations
```



- Main structure consists of virtual memory, page table and physical memory.
- Firstly all pages are loaded into virtual memory. When sorting operation is called, physical memory is filled by some pages according to the order of pages (page0, page1, …). Page replacement algorithms have to be considered if physical memory is full.

2. Algorithm

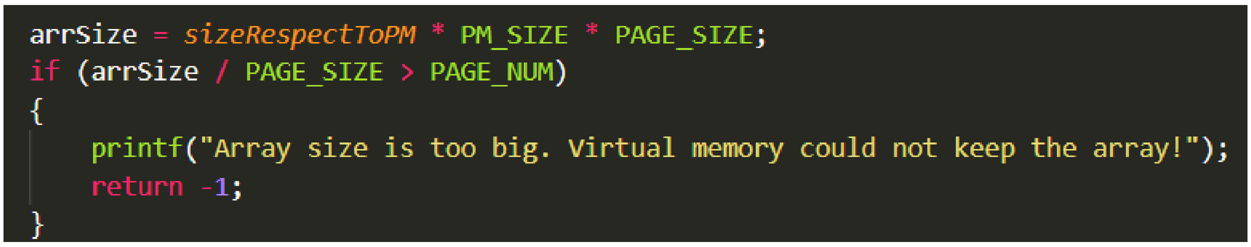
- Overloading “[]” operator makes things easier. All sorting operations use this operator to get value. If we use “[]” operator to manipulate pages, we can easily handle with sorting situations. According to the index of the value, we can find “pageIndex” and “offset” of the value. After that, we look up the page table to check whether the page is in physical memory or not. If the page is in physical memory “hit” increases, otherwise page replacement occurs. We must return the new data of the page for swap operations.



```
int &operator[](int i)
{
    int pageIndex = i / PAGE_SIZE;
    int offset = i % PAGE_SIZE;
    int frameNum = lookUpPageTable(pageIndex);
    if (frameNum != -1)
    {
        physicalMem[frameNum].referenced = 1;
        hit++;
    }
    else
        makePageReplacement(pageIndex);

    return physicalMem[pageIndex].data[offset];
}
```

- In page replacement function “miss” and “pageLoaded” increase. If physical memory is full, FIFO, Second Chance or LRU algorithm are used according to the given choice.
- Page replacement algorithms have some common features. For example, updating the page table, writing modified page to the virtual memory (modified bit is checked) etc.
- I used “integer” value to remind last position of the new page in the memory to apply FIFO algorithm correctly. Also, I’ve implemented Queue for Second Chance algorithm.
- In the other parts, I tried to stick to algorithm representations in the lecture.
- I’ve checked the errors as best I could.



```
arrSize = sizeRespectToPM * PM_SIZE * PAGE_SIZE;
if (arrSize / PAGE_SIZE > PAGE_NUM)
{
    printf("Array size is too big. Virtual memory could not keep the array!");
    return -1;
}
```