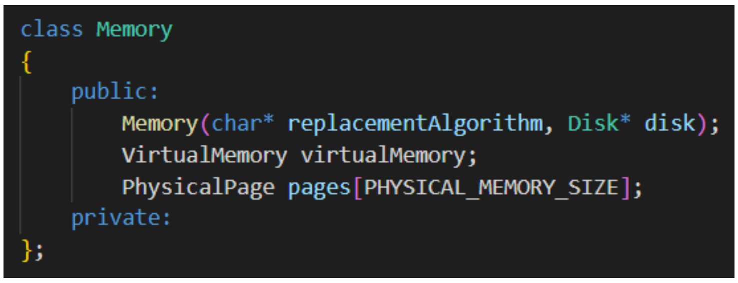
CSE 312/504 Operating Systems

Abdullah Çelik

171044002

Table of Contents

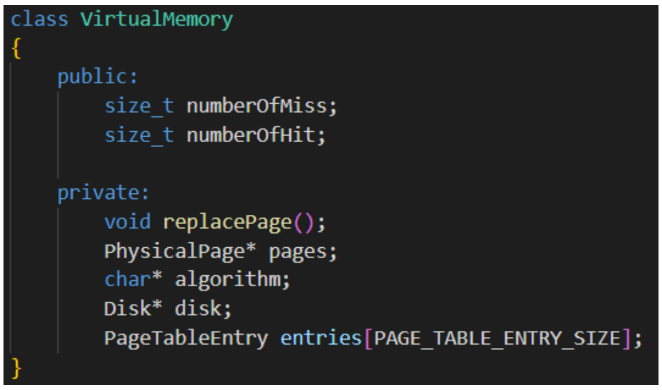
1. Memory Structure



```
class Memory
{
    public:
        Memory(char* replacementAlgorithm, Disk* disk);
        VirtualMemory virtualMemory;
        PhysicalPage pages[PHYSICAL_MEMORY_SIZE];
    private:
};
```

- Memory should involve two utilities in it.
- First utility is virtual memory that contains page table.
- Second utility is physical memory that keeps the actual memory pages which are indexed by the virtual memory page.

2. Virtual Memory Structure

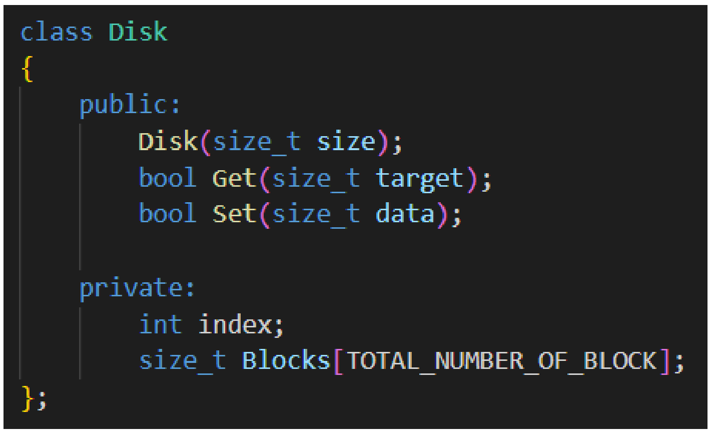


```
class VirtualMemory
{
    public:
        size_t numberOfMiss;
        size_t numberOfHit;

    private:
        void replacePage();
        PhysicalPage* pages;
        char* algorithm;
        Disk* disk;
        PageTableEntry entries[PAGE_TABLE_ENTRY_SIZE];
}
```

- Virtual memory should be responsible for address translation, page replacement, and also keeping record of the given data. For those reasons, virtual memory needs to access disk, physical memory.
- It should keep the statistics about number of hits and misses.

3. Disk Structure

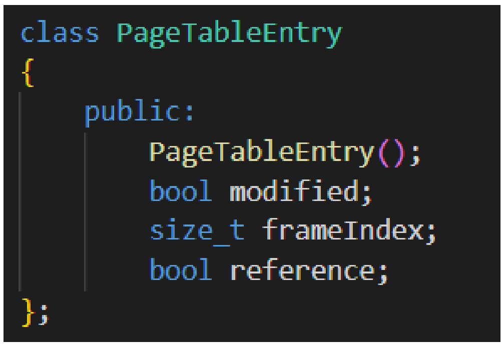


```
class Disk
{
    public:
        Disk(size_t size);
        bool Get(size_t target);
        bool Set(size_t data);

    private:
        int index;
        size_t Blocks[TOTAL_NUMBER_OF_BLOCK];
};
```

- Disk is a permanent storage. All data which is required should be located in the disk and in any emergent data access, memory can get those missed data from the disk.
- For this reason, I should put my data into the disk. When the page fault occurs, a missing page can be received from the disk block.
- There should be a sequential access from its interface.

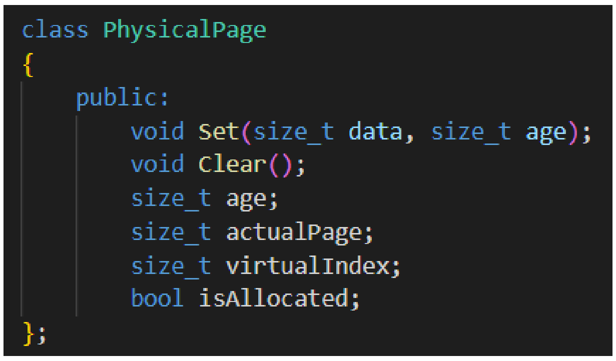
4. Page Table Entry Structure



```cpp
class PageTableEntry
{
    public:
        PageTableEntry();
        bool modified;
        size_t frameIndex;
        bool reference;
};
```

- PageTableEntry should be the shortest unit of the virtual page. Each of these entries should have some components.
- The most important one is the physical address frame index. Address frame keeps the index of physical pages. Pages are being kept sequentially as an array and that means constant time access for getting physical pages from the virtual page cells.
- PageTable entry size is way more bigger than actual physical memory size as it is mentioned in the homework document. Size abstraction is coming from this actually.
- There should be aging to apply the page replacements algorithms on the top of referenced and modified bits.

5. Physical Page Structure



```cpp
class PhysicalPage
{
    public:
        void Set(size_t data, size_t age);
        void Clear();
        size_t age;
        size_t actualPage;
        size_t virtualIndex;
        bool isAllocated;
};
```

- Physical page should represent the actual page that keeps the user data in the memory. Those pages are being accessed by the virtual page and both virtual page entry and the physical pages are being kept in the memory.
- Virtual page to physical page index is the key component for entire memory works.

6. Page Replacement Algorithms

a) Second-Chance (SC)

- I should implement the second chance algorithm. The second chance algorithm gives a second chance to the referenced pages.

- When the page replacement is required, the algorithm looks for an oldest page if the oldest page's second chance value is already made as true. That means that the replacer gives another chance to that page and it is time to replace it.
- If the second chance value is false, the replacer will give a second chance to that page and move on to another page to replace it with.

b) Least-Recently-Used (LRU)

- In order to implement an index for page table entries and perform page replacement, I should assign an index value to each entry in the page table. The index values start from 0 for the first entry, 1 for the second entry, and so on.
- When a page replacement is required, I should examine the index values associated with each page table entry and select the entry with the minimum index, indicating the least recently used (LRU) page. It is crucial to update the index whenever a page hit occurs.

c) Working set clock (WSClock)

- I should utilize an index for the page table. Starting from a given index, I can examine the page table entries and their corresponding reference bits. If the reference bit is 1, I will set it to 0 and continue checking the next entry. I will repeat this process until I find a page with a reference bit of 0.