Gebze Technical University - Computer Engineering Department

CSE 312 – 2022 Operating System Course

Muhammed Bedir ULUCAY 1901042697

Problem:

Design a memory management structure creating virtual memory, page table, main memory and disk.

In my program all our objects are variable, and methods are all static. Because there is just one object that could be these structures (Singleton).

Sizes:

```
#define PAGE_SIZE 2
#define MEMORY_SIZE 4
#define DISK_SIZE 16
#define PAGE_TABLE_SIZE (DISK_SIZE + MEMORY_SIZE)
```

Virtual Memory and Page Table:

Our base structure is virtual table and its objects so page table entry and page table.

```
typedef struct PageEntry{

    common::int32_t memory_index;
    common::int32_t disk_index;         // bulundugu index
    common::int32_t reference;          // son çiçyle da kullanıldı mı?
    common::int32_t modified;           // is modified or not
    common::int32_t present;            // is present or not
    common::int32_t pageNumber;         // programin kacinci page i olduguna dair
    common::int32_t isFree;             // in use or not

}PageEntry;


class PageTable{


    public:
        PageTable();
        ~PageTable();

        static void setInitialPageTable();

        static PageEntry getPageEntry(int i);
        static int setPageEntry(int i, PageEntry p);

        static int getNextFreePage();
        static common::int32_t getPageUsingPageNumber(int pageFrame);

        static void removePresentFromMemory(int pageNumber);

        static common::int32_t inPage(common::int32_t pageNumber, common::int32_t reference);

        static common::int32_t writeBackToDisk(int pageNumber);
        static common::int32_t getFromDisk(int pageNumber);

        static common::int32_t getMemoryIndex(int pageNumber);
        static common::int32_t getDiskIndex(int pageNumber);

        static void printPageTable();

        static PageEntry pageTable[MEMORY_SIZE + DISK_SIZE];

};
```

Page table contains a page table entry array to keep track of the page of the data or a program. Which page is located where and its some special properties. It is the most needed part for the virtual memory management system.

Memory:

```
class Memory{

        Memory();
        ~Memory();

    public:

        static void setInitialMemory();

        static common::int32_t writePage(int index, int pageNumber, common::int32_t page[PAGE_SIZE]);

        static PageEntry getPageStatus(int pageNumber);
        static common::int32_t isInMemory(int pageNumber);

        static common::int32_t* getPage(int index);
        static common::int32_t getNextFreePage();

        static void printMemory();

        static common::int32_t memory[MEMORY_SIZE][PAGE_SIZE];
        static common::int32_t isFree[MEMORY_SIZE];

};
```

Memory keeps data getting from the disk and according to the page table entry that is memory located data decided what we are going to do about the page in memory.

We can write back to disk if modified bit 1. Or directly delete …

If a page is in the memory its present bit in the page table entry is 1 if the page is not in the memory the present bit of the page is 0.

And the page table entry keeps track of the index in the memory and the disk.

Disk:

```
class MEM{

        MEM();
        ~MEM();

    public:

        static void setInitialMEM();

        static common::int32_t writePage(int index, int pageNumber, common::int32_t page[PAGE_SIZE]);
        static common::int32_t writePage2(int index, int pageNumber, common::int32_t page[PAGE_SIZE]);

        static PageEntry getPageStatus(int pageNumber);
        static common::int32_t isInDisk(int pageNumber);

        static common::int32_t* getPage(int index);
        static common::int32_t getNextFreePage();

        static common::int32_t getPageUsingPageFrame(int pageFrame);

        static void printDisk();

        static common::int32_t LMEM[DISK_SIZE][PAGE_SIZE];
        static common::int32_t isFree[DISK_SIZE];

};
```

The disk stores the data before the sorting algorithm is started. And adding the page table entries to the page table allows us to keep track of the page numbers and the indexes where they are ed in the disk.

Page Replacement:

```
int page_1 = j / PAGE_SIZE;
if(in_fifo(fifo, MEMORY_SIZE, page_1) == -1){
    miss++;
```

```
    next++;
}else{
    hit++;
}
```

If the wanted page is not in the memory,

```
for(int k=0; k<PAGE_SIZE; ++k){
    Memory::memory[PageTable::pageTable[page_1].memory_index][k] = MEM::LMEM[page_1][k];
}
```

we are getting it from the disk. And set the present bit as 1.

After the comparison of the values in the queue [j] > [j+1]

Then we are right back to disk.

```
    for(int i=0; i<PAGE_SIZE; ++i){
        MEM::LMEM[page_1][i] = Memory::memory[page_1][i];
        MEM::LMEM[page_2][i] = Memory::memory[page_2][i];
    }
```

After the sorting is done the output;