CSE 312 OPERATING SYSTEMS
HW-2 REPORT
200104004095
MEHMET HÜSEYİN YILDIZ

In this homework we would implement 3 page replacement algorithms:

- FIFO (First-In-First-Out)
- Second Chance
- LRU (Least Recently Used)

But to implement these algorithms we need to design a virtual memory system.
To implement virtual memory I designed a virtual memory entry table. For fifo and second chance algorithms we need a queue (linked list) structure so I held all entries in a linked list. The structure of entry is like this:

```
struct TableEntry{

    int page_index;
    void* physical_address;
    bool present;                        // if the entry used or not used.
    bool referenced;                     // is referenced by read or write. For Fifo and Second chance
    int last_referenced_counter;         // it holds the last usage time by holding a counter. For LRU

    TableEntry* nextEntry;
    TableEntry* prevEntry;
};
```

I held memory entries and disk entries in my VirtualMemoryManager class.

```
// Virtual memory manager class that manupulates virtual memory (memory and disk simulation)
class VirtualMemoryManager
{
private:

    int va_count;
    int reference_count;

    struct TableEntry* last_mem_page;                  // Last mem page
    struct TableEntry* last_disk_page;                  // Last disk page

    void* page_allocate(short disk_or_mem );           //  DISK | MEMORY
    TableEntry* entry_allocate( );                 //  DISK | MEMORY



public:
    int hit;
    int miss;
    int page_load;
    int page_write;

    int page_size;                                 // page size for every page in bytes.
    struct TableEntry* mem_page_table;             // Memory Page table
    struct TableEntry* disk_page_table;            // Disk Page table


    static MemoryManager* mem_manager;             // Holds the memory_manager
    static MemoryManager* disk_manager;            // Simulates the disk_manager
    static VirtualMemoryManager* activeVMM;

    VirtualMemoryManager(int page_size1, MemoryManager* mem_manager1, MemoryManager* disk_mana
    ~VirtualMemoryManager();                       // Destructor
```

I implemented linked list for TableEntry and their required functions like:

```
TableEntry* find_entry(short onDisk, int page_index);            // find the page entry by virtual address
TableEntry* pop_entry(bool on_disk);                              // removes the last entry of linked list and returns
void push_entry(int on_disk, TableEntry* new_entry);              // adds the given entry to head of linked_list
TableEntry* remove_entry(bool on_disk, int index);                // removes the entry with given index
```

These functions implemented for base table entry operations. They are implemented as we learned linked list in Data Structures and Algorithms course. If we want to apply the functions on disk entries then we make on_disk = 1 and for memory entries on_disk = 0.

After creating these structure. Then we need some virtual memory manager class methods like:

```
VA allocate(common::size_t size);                 // allocation of pages in given size
void free(VA virtual_addr);                       // deallocation of pages
void* operator[](VA virtual_address);             // converts the virtual adress to physical address.
bool is_on_disk(VA va);                           // returns if corresponding physical address of given virtual address is
int* get(VA baseVa, int index);                   // returns the int index based given virtual address.
```

The VirtualMemoryManagement class basically holds 2 MemoryManagement objects which implemented in 16^{th} wyos video. One of them is for memory pages (mem_manager) and the other one is for disk memory pages (disk_manager). Disk is simulated using MemoryManagement class.

allocate(): Allocates a page in memory by using memory manager malloc() function. If it has no place then it allocates in disk.

Operator[]: Converts virtual address to physical address. All reference bit or counter operations are made here. The specified replacement algorithm is called here. If there is no such virtual address, then it prints error message.

These are the basic 2 function I used. Allocation and accessing is realized with these.

The page replacement part is realized by using table entry functions like remove_entry(), pop_entry(), push_entry() etc.

Replacememnt Algorithms

Fifo

```
short VirtualMemoryManager::page_replace_fifo(TableEntry* disk_entry){
    TableEntry* oldMemEntry = pop_entry(0);
    page_replace(oldMemEntry, disk_entry);
    push_entry(1, oldMemEntry);
    push_entry(0, disk_entry);
}
```

As can be seen in the photo up-side, I used the basic linked list operations and page_repace() function which replaces the pages of given 2 entry.

Second Chance

```
short VirtualMemoryManager::page_replace_second_chance(TableEntry* disk_entry){
    TableEntry* oldMemEntry;
    // finding the not referenced entry
    while( (oldMemEntry = pop_entry(0))->referenced )
        push_entry(0,oldMemEntry);

    // if the entry is not referenced then we replace it
    page_replace(oldMemEntry, disk_entry);
    push_entry(1, oldMemEntry);
    push_entry(0, disk_entry);
}
```

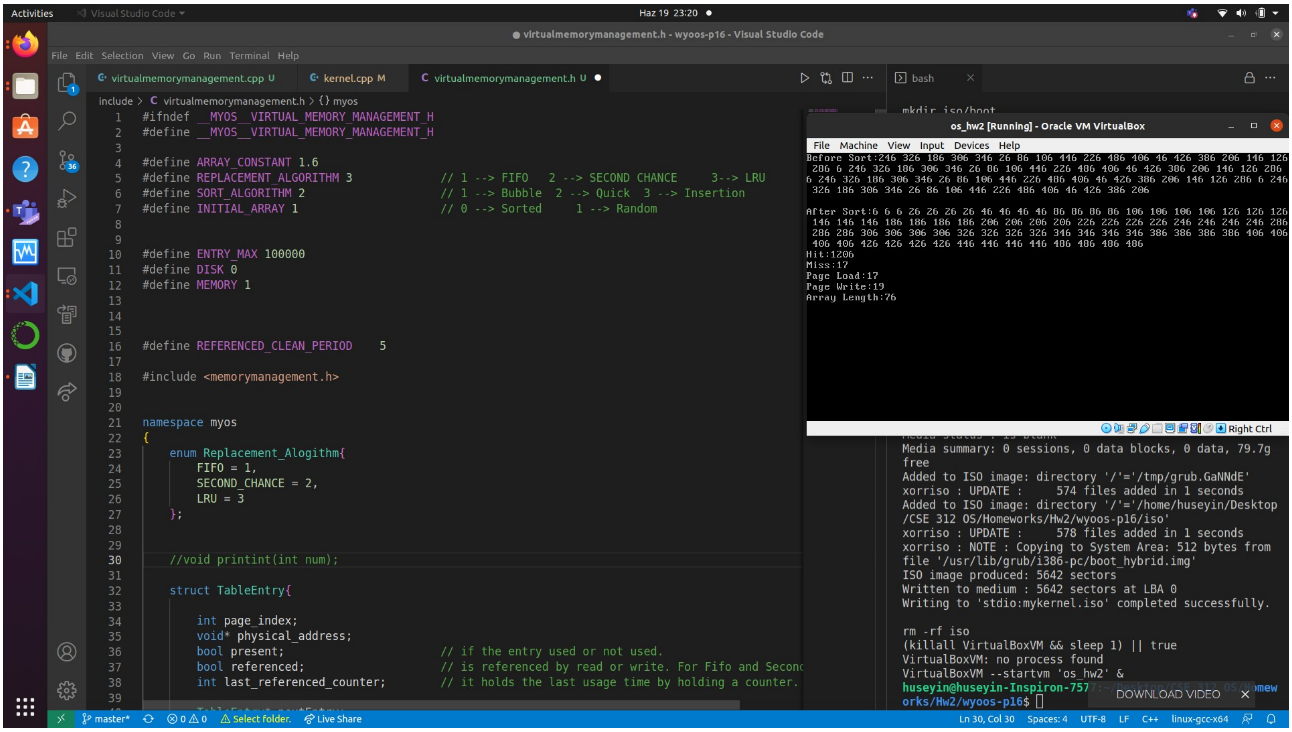
This algorithm is very similar to fifo. There is entry bit in every entry. When a page used (with operator[]) the entry reference bit is assigned as true. After every specific entry reference all reference bits are assigned false. If the popped entry is referenced, then it just pushes to the end of linked list. It makes this operation until it find the not referenced entry. Finally replaces the pages with disk entry and pushes them to the entries.

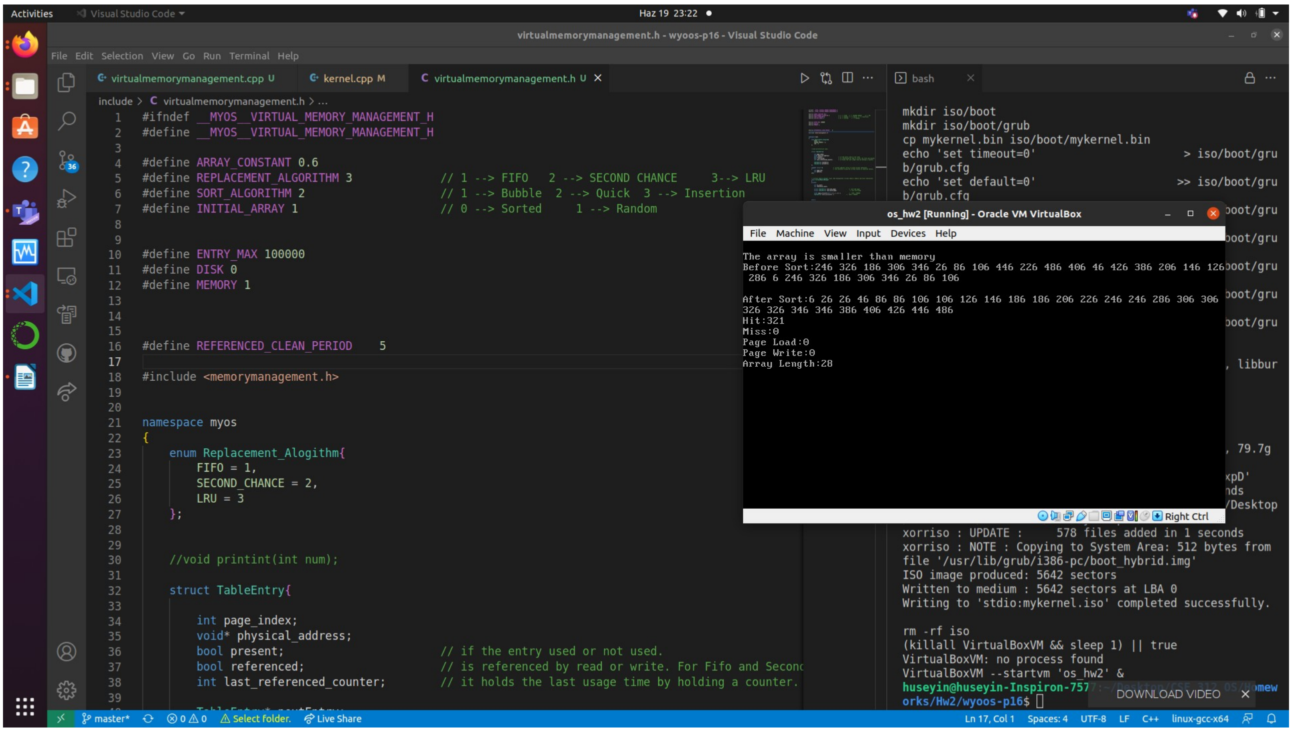
Least Recently Used (LRU)

```cpp
short VirtualMemoryManager::page_replace_LRU(TableEntry* disk_entry){

    int lru_index = find_least_recently_used_mem_entry();
    TableEntry* oldMemEntry = remove_entry(0,lru_index);

    page_replace(oldMemEntry,disk_entry);
    push_entry(1, oldMemEntry);
    push_entry(0, disk_entry);

}
```

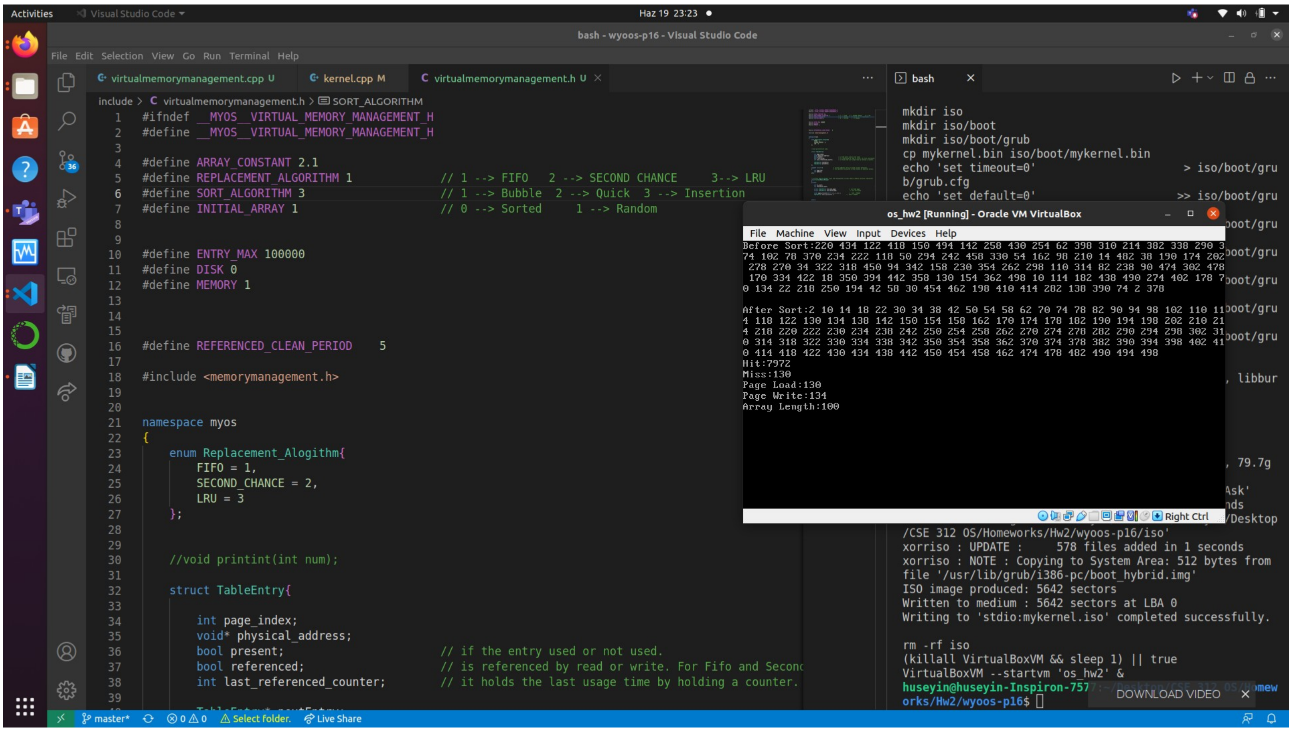
For this algorithm, every entry has last_referenced_counter. Every entry counter is increased on every referencing in operator[] function. The called entry is made 0. In page_replace_LRU() function find_least_recently_used_mem_entry() function is traverses all memory entries and finds the entry has maximum counter and returns its index. Then the entry is removed from the linked list and page replaced. Lastly both entries pushed to the entry lists again.

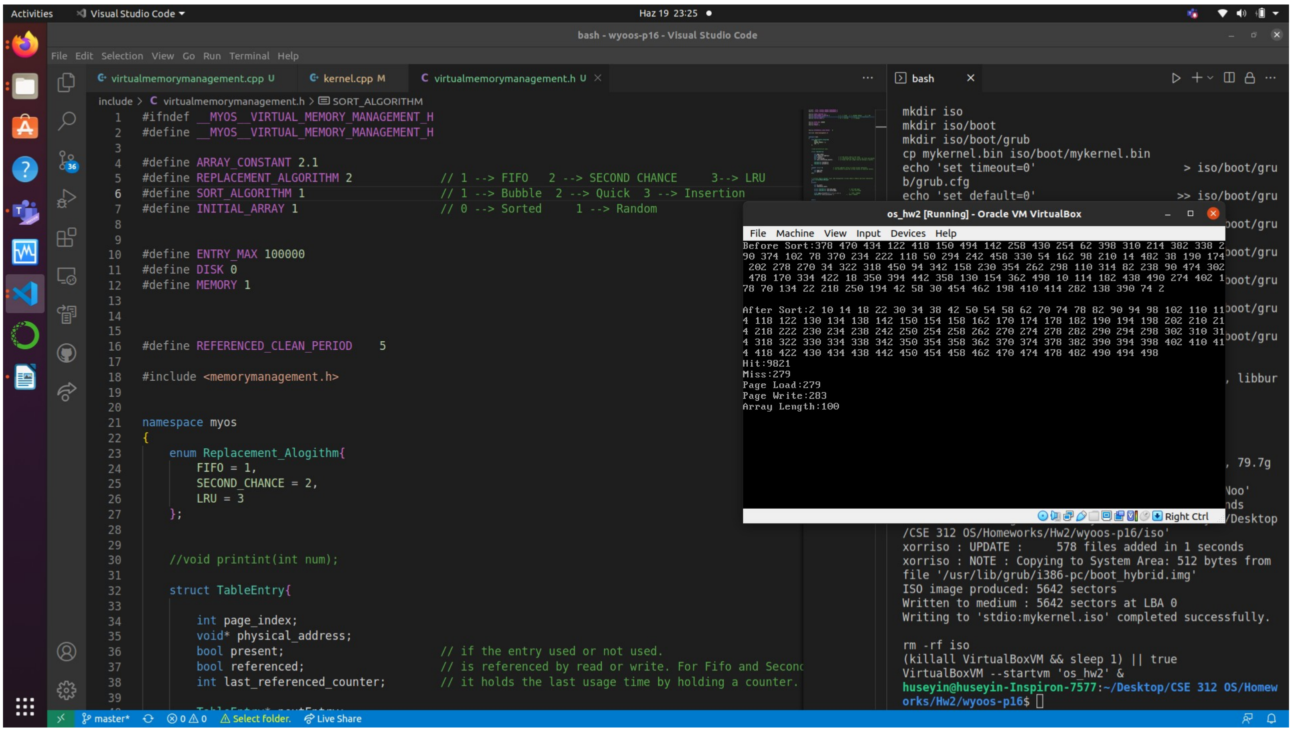
Example outputs





As can be seen when array size is smaller than memory error message is printed.





Conclusion

This is all I done in this homework. I believe that I done best. Everything is designed as we learned in the lectures. And I designed these all on my own completely. In this report I tried to be clear and short as much as possible. I hope everything is clear.