CSE312 OPERATING SYSTEM
HOMEWORK 2

Mehmet Avni ÇELİK
1801042630

This homework is about implementing some of the page replacement algorithms to required sorting algorithms.

Page Replacement Algorithms

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

1) **First-In First-Out(FIFO):**

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced, page in the front of the queue is selected for removal.

2) **Second Chance:**

In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected, if it's referenced recently(if referenced bit is 1) this page will be assigned to the head of the list and will have a second chance. If the oldest page's reference bit is 0, it will be selected for removal.

3) **Least Recently Used(LRU):**

In this algorithm, page will be replaced which is least recently used. It will be decided by referenced bit and keeping a clock in addition.

In this assignment, I assumed that the virtual memory and the dis as an array of "Pages", I could only implemented first-in first-out algorithm.

Implementation of First-In First-Out

Here you can see some required definitions and global variables.

```
#define MY_VIRTUAL_MEMORY_SIZE 8
#define MY_ARRAY_SIZE 10
#define MY_DISK_SIZE 16


int virtualSize=0;
int diskSize=0;
int hit=0;
int miss=0;
int pagesLoaded=0;
int back2disk=0;
int diskIndex=0;
```

```
class Page{
    public:
    int data;
    int reference;
    bool modified;
    int size=0;
    void setData(int i){
        data=i;
    };
    int getData(){
        return data;
    }

};

int myArray[MY_ARRAY_SIZE]={5,9,6,0,3,2,1,7,4,8};
Page myVirtualMemory[MY_VIRTUAL_MEMORY_SIZE];
Page myDisk[MY_DISK_SIZE];
```

Then, virtual memory is filled by statically given array. If there is no space in the virtual memory, all the other elements are filled to the assumed disk.

```
int virtualMemoryFiller(){      //fills virtual memory and then disk.
    diskIndex=MY_ARRAY_SIZE-MY_VIRTUAL_MEMORY_SIZE;
    for(int i=0;i<MY_VIRTUAL_MEMORY_SIZE;i++){
        myVirtualMemory[i].setData(myArray[i]);
        virtualSize++;
    }
    for(int i=0;i<diskIndex;i++){
        myDisk[i].setData(myArray[diskIndex+i]);
        diskSize++;
    }
}
```

When any index of an array will be used, it will be checked that if the required data is in the memory or not.

```
bool check_is_in_virtual_mem(int x){
    for(int i=0;i<MY_VIRTUAL_MEMORY_SIZE;i++){
        if(myVirtualMemory[i].data==x){
            return true;
        }
        else{
            return false;
        }
    }
}
```

If it is not in the memory, the pages will be replaced according to first-in first-out page replacement algorithm that is showed below.

```
void FIFO(int x){
    int temp= myVirtualMemory[0].data;
    for(int j=1;j<MY_VIRTUAL_MEMORY_SIZE;j++){
        myVirtualMemory[j-1].setData(myVirtualMemory[j].data);
    }
    for(int i=0;i<MY_DISK_SIZE;i++){
        if(myDisk[i].data==x){
            myVirtualMemory[MY_VIRTUAL_MEMORY_SIZE-1].setData(myDisk[i].data);
            myDisk[diskIndex].setData(temp);
            diskIndex++;
        }
    }
}
```

The usage of all the functions in main is shown in the picture below. Some information,such as; hit number,miss number,pages loaded and pages written back is also implemented. Each of the sorting algorithms can be tested by removing comment lines.

```
extern "C" void kernelMain(const void* multiboot_structure, uint32_t /*multiboot_magic*/)
{
    virtualMemoryFiller();
    printArray(myArray,MY_ARRAY_SIZE);
    printf("\n");
    bubbleSort(myArray,MY_ARRAY_SIZE);
    //insertionSort(myArray,MY_ARRAY_SIZE);
    //quickSort(myArray,0,MY_ARRAY_SIZE-1);
    printf("\n");
    printArray(myArray,MY_ARRAY_SIZE);
    printf("---");
    printf("Hit Number: ");
    printfHex(hit);
    printf("---");
    printf("Miss Number: ");
    printfHex(miss);
    printf("---");
    printf("Pages Loaded: ");
    printfHex(miss);
    printf("---");
    printf("Pages Written Back: ");
    printfHex(miss);
    printf("---\n");
```

Implementations of Sorting Algorithms'

Standard implementations of sorting algorithms' are used.

1)Bubble Sort:

```
// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)
        for (j = 0; j < n - i - 1; j++){
            if(check_is_in_virtual_mem(arr[j])==true){
                hit++;
            }
            else{
                FIFO(myArray[arr[j]]);
                miss++;
            }
            if(check_is_in_virtual_mem(arr[j+1])==true){
                hit++;
            }
            else{
                FIFO(myArray[arr[j+1]]);
                miss++;
            }
            if (arr[j] > arr[j + 1]){
                swap(&arr[j], &arr[j + 1]);
            }
        }
}
```

2)Insertion Sort:

```
/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        if(check_is_in_virtual_mem(arr[i])==true){
            hit++;
        }
        else{
            FIFO(myArray[arr[i]]);
            miss++;
        }
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            if(check_is_in_virtual_mem(arr[j])==true){
                hit++;
            }
            else{
                miss++;
            }
            if(check_is_in_virtual_mem(arr[j+1])==true){
                hit++;
            }
            else{
                miss++;
            }
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

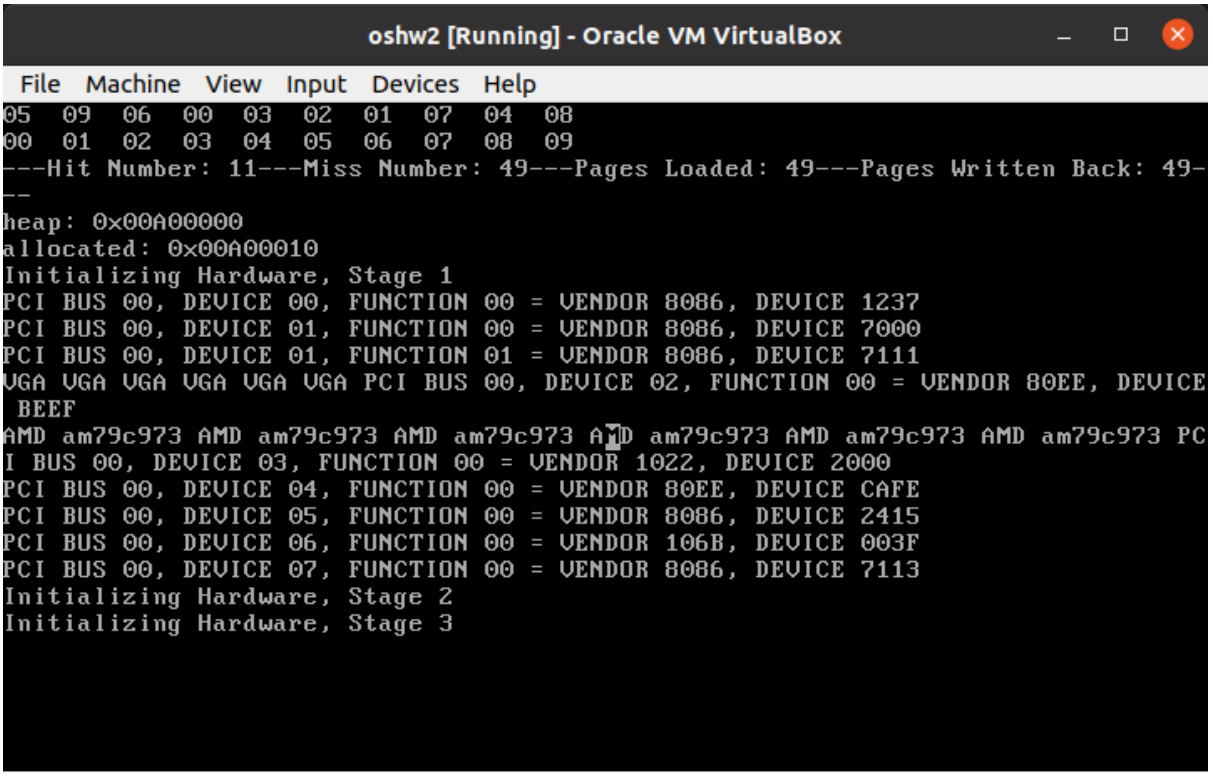
3)Quick Sort:

```
int partition (int arr[], int low, int high)
{
    if(check_is_in_virtual_mem(arr[high])==true){
        hit++;
    }
    else{
        miss++;
    }
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element and indicates the right position of pivot found so far

    for (int j = low; j <= high - 1; j++)
    {
        if(check_is_in_virtual_mem(arr[i])==true){
            hit++;
        }
        else{
            miss++;
        }
        if(check_is_in_virtual_mem(arr[j])==true){
            hit++;
        }
        else{
            miss++;
        }

        // If current element is smaller than the pivot
        if (arr[j] < pivot)
        {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    if(check_is_in_virtual_mem(arr[i])==true){
        hit++;
    }
    else{
        miss++;
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

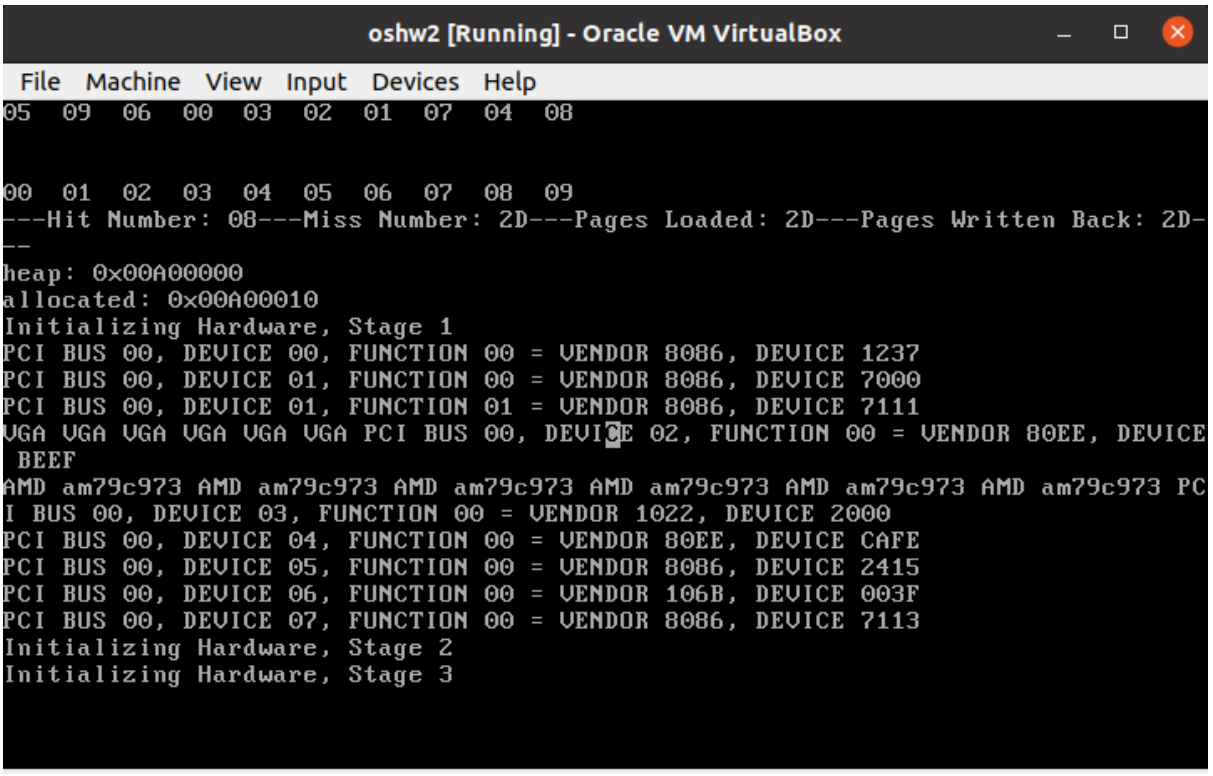
Tests

Each of the sorting algorithms are tested with the FIFO page replacement algorithm.

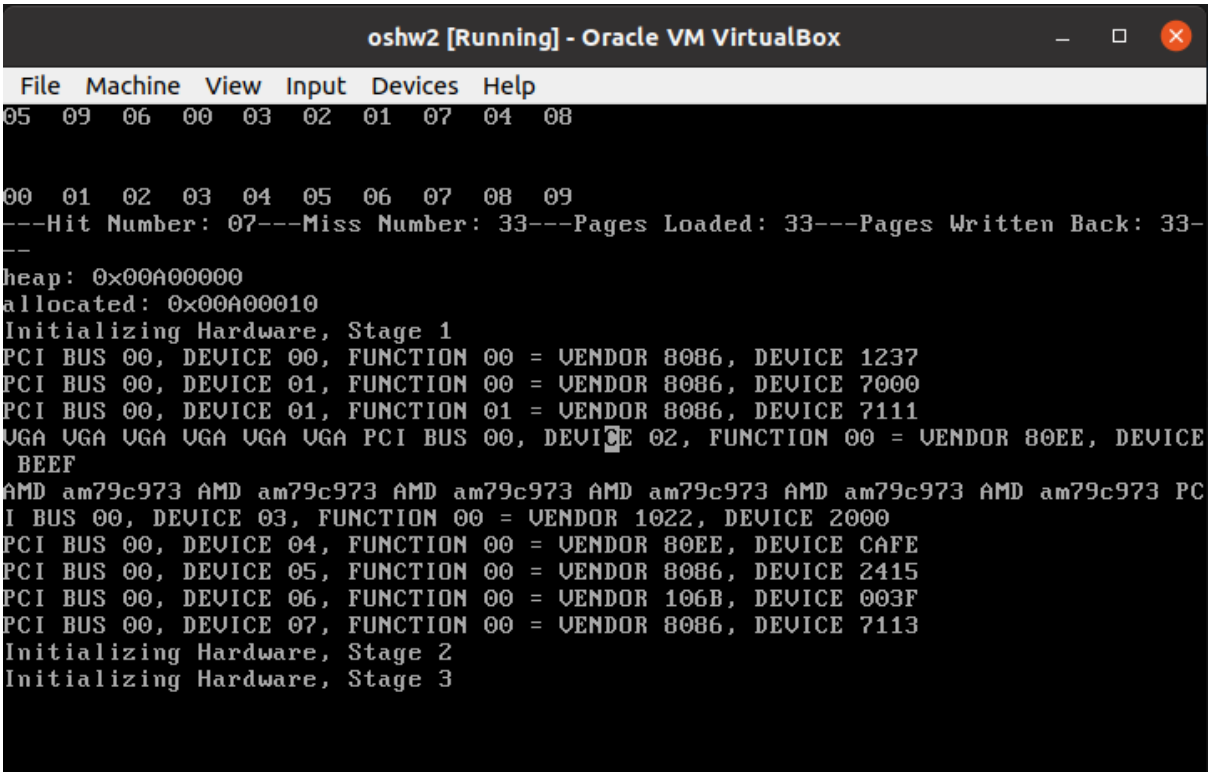
1)Bubble Sort Output:



2)Insertion Sort Output:



3)Quick Sort Output:



Results

-(Hexadecimal)	Hit Number	Miss Number	Pages Loaded	Pages Written Back
Bubble Sort	11	49	49	49
Insertion Sort	08	2D	2D	2D
Quick Sort	07	33	33	33

-(Decimal)	Hit Number	Miss Number	Pages Loaded	Pages Written Back
Bubble Sort	17	73	73	73
Insertion Sort	8	45	45	45
Quick Sort	7	51	51	51

For my statically defined array, it is seen that;

- The bubble sort has the most hit rate and also worst miss rate.
- The insertion sort has the second best hit rate and best miss rate.
- The quick sort has the lowest hit rate but second in miss rate.
- For the FIFO algorithm, miss number,loaded pages and written back pages are in the same number. Because when there is a page fault it is taken from disk and written to memory. The future removed page will be loaded and written back to disk. Thus, they are all equal.

Thank you in advance…