OS HW2

Zafer ALTAY

Structures

```
int arrayForSorting[40]={40,39,38,37,36,35,34,33,32,31,
                         30,29,28,27,26,25,24,23,22,21,
                         20,19,18,17,16,15,14,13,12,11,
                         10,9,8,7,6,5,4,3,2,1};

const int arraySize=40;
const int memorySize=10;
int replacementCounter=0;
int hitCounter=0;
int missCounter=0;
int fifolist[memorySize];
int pmCounter=0; //Memoryde ki eleman sayısı, memorysize'dan küçükse direkt oraya eklenir.


struct{
    int val;
    int rBit;
    int validBit;
    int counter; //for LRU
    int physicalFrame;

}typedef PageEntry;


struct
{
    PageEntry pagetable[arraySize];

}typedef VirtualMemory;
VirtualMemory vm;

struct
{
    int pages[memorySize];

}typedef PhysicalMemory;
PhysicalMemory pm;


struct{

    int diskPages[arraySize];

}typedef Disk;
Disk disk;
```

Here some values are used for demo.Virtual memory holds page table that consist of page entry and physical memory only hold page numbers. Page information is kept in the page table.If an operation is applied on the page numbers kept in the Memory, the page table is updated.

Algorithms

FirstInFirstOut

```c
void firstInFirstOut(int newpage){                          //Fifolisttin ilk elemanındaki page'i memoryde bulur kaldırır.
                                                            //Yeni gelecek page'i memory'de onun yerine ekler.
    //SİLİNECEK                                             //Fifo listi günceller.
    printf("Mem: ");
    printMemory();
    printf("\n");
    printf("Fif: ");
    printFifoList();
    printf("\n");
    /*for(int j=0;j<80000;j++){
        for (int k = 0; k < 20000; k++)
        {
            //SLEEP
        }
    }
    */
    int i=0;
    int temp=fifolist[i];
    int vmIndex=findInVM(temp);
    vm.pagetable[vmIndex].physicalFrame=-1;
    vm.pagetable[vmIndex].validBit=0;

    for (i = 0; i < memorySize-1; i++)
    {
        fifolist[i]=fifolist[i+1];

    }
    fifolist[memorySize-1]=newpage;

    for (int i = 0; i < memorySize; ++i)
    {
        if (pm.pages[i]==temp)
        {
            pm.pages[i]=newpage;
            int vmIndex=findInVM(newpage);
            vm.pagetable[vmIndex].physicalFrame=i;
            vm.pagetable[vmIndex].rBit=1;
            vm.pagetable[vmIndex].validBit=1;
            replacementCounter++;
        }
    }
}
```

Here I am backing up the first element of fifolist first. Because this is the page to be removed.Then I find this page in pagetable and memory and add the new page in its place.I am updating the page table according to these operations. Then I add the new page to the end of the fifolist.

Second Change

```
void secondC(int newpage){


    int i=0;

    while(1){                                   //Fifo listteki ilk elemanın rbiti 0'sa fifo fonskiyonunu çağırır,1 ise 0 yapıp en sona atar kaydırır.
        int temp=fifolist[0];
        int vmIndex=findInVM(temp);             //0 olan fifoda kaldırılır. Memory'e eklenir. Page table güncellenir.FİFo list güncellenir.

        printf("Fif: ");
        printFifoList();
        printf("\n");


        if (vm.pagetable[vmIndex].rBit==0)     //
        {
            firstInFirstOut(newpage);
            return;
        }
        else{
            vm.pagetable[vmIndex].rBit=0;
            for (int i = 0; i < memorySize-1; i++)
            {
                fifolist[i]=fifolist[i+1];
            }
            fifolist[memorySize-1]=temp;
        }
    }

}
```

Here I find the first element of our fifo list in pagetable.If the Rbit is 1, I set the rbit to 0 and put it at the end of the list. And I scroll the list 1 time. So I'm looking at the next element.If rbit is 0, I send the new element to fifo, the old element is removed.

LRU

```
void LRUSoftware(int newpage){          //Minimum counter'a sahip page'i arar. Onu kaldırır. Yeni gelen page'i onun yerine ekler.

    int min=99999;                          //Page table'ı günceller.
    int index=-1;
    int pageNum=-1;

    for (int i = 0; i < memorySize; i++)
    {
        int xx=pm.pages[i];
        int vmIndex=findInVM(xx);
        if (min>vm.pagetable[vmIndex].counter)
        {
           min=vm.pagetable[vmIndex].counter;
           index=i;
           pageNum=xx;
        }
    }
        //SİLİNECEK
    printf("Mem: ");
    printMemory();
    printf("\n");
    printf("Fif: ");
    printFifoList();
    printf("\n");
    /*  for(int j=0;j<80000;j++){
            for (int k = 0; k < 20000; k++)
            {
                //SLEEP
            }
        }
    */
    int vmIndex=findInVM(pageNum);
    vm.pagetable[vmIndex].physicalFrame=-1;
    int vmIndex2=findInVM(newpage);
    vm.pagetable[vmIndex2].physicalFrame=index;
    pm.pages[index]=newpage;
    replacementCounter++;

}
```

Here I find the page with the lowest counter in memory. I remove it and add the new page. Then I update the page table.

SORTS

Bubble Sort

```
//------------SORT ALGORITHM-----------------------------------------------------------------
void bubbleSort(int arr[], int n,int algorithm){              // bubble sort function

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n-i-1; j++){         // loop until n-i-1(further is unnecessary)

            if (!findInMemory(arr[j]))  //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
            {
                int vmIndex=findInVM(arr[j]);
                int diskIndex=findInDisk(arr[j]);
                if (pmCounter<memorySize)
                {
                    pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
                    vm.pagetable[vmIndex].physicalFrame=pmCounter;
                    vm.pagetable[vmIndex].rBit=1;
                    vm.pagetable[vmIndex].validBit=1;
                    fifolist[pmCounter]=disk.diskPages[diskIndex];
                    pmCounter++;

                }
                else{                                                   //Eğer memory doluysa algoritmaya göre page replacement yapılır.
                    if (algorithm==1)
                    {
                        firstInFirstOut(arr[j]);
                    }
                    else if(algorithm==2){
                        secondC(arr[j]);
                    }
                    else{
                        LRUSoftware(arr[j]);
                    }
                }

            }
            if (!findInMemory(arr[j+1]))  //Page Replacement for arr[j+1] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
            {
                int vmIndex=findInVM(arr[j+1]);
                int diskIndex=findInDisk(arr[j+1]);
                if (pmCounter<memorySize)
                {
                    pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
                    vm.pagetable[vmIndex].physicalFrame=pmCounter;
                    vm.pagetable[vmIndex].rBit=1;
                    vm.pagetable[vmIndex].validBit=1;
                    fifolist[pmCounter]=disk.diskPages[diskIndex];
                    pmCounter++;

                }
```

```
                }
                else{                                   //Eğer memory doluysa algoritmaya göre page replacement yapılır.
                    if (algorithm==1)
                    {
                        firstInFirstOut(arr[j+1]);
                    }
                    else if(algorithm==2){
                        secondC(arr[j+1]);
                    }
                    else{
                        LRUSoftware(arr[j+1]);
                    }
                }

            }
            if( arr[j] > arr[j+1]){
                int temp = arr[j];              // swap elements for sorting them
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

Classic bubble sort algorithm is applied.It has been checked whether the elements of the array to which the sort algorithm will be applied are in memory or not.We do not need to have all of them. Because we are paging here. Thus, it is sufficient for us to have only the element used at that moment.If the memory is not completely full at that moment, it is added directly to the memory. If the memory is full and the page is not in memory, page replacement is performed.

Insertion Sort

```
void insertionSort(int arr[], int n,int algorithm)
{

    //-----------------------------
    int i, key, j;
    for (i = 1; i < n; i++)
    {
    //--------------------------------

        if (!findInMemory(arr[i]))  //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
        {
            int vmIndex=findInVM(arr[i]);
            int diskIndex=findInDisk(arr[i]);
            if (pmCounter<memorySize)
            {
                pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
                vm.pagetable[vmIndex].physicalFrame=pmCounter;
                vm.pagetable[vmIndex].rBit=1;
                vm.pagetable[vmIndex].validBit=1;
                fifolist[pmCounter]=disk.diskPages[diskIndex];
                pmCounter++;
            }
            else{                                                       //Eğer memory doluysa algoritmaya göre page replacement yapılır.
                if (algorithm==1)
                {
                    firstInFirstOut(arr[i]);
                }
                else if(algorithm==2){
                    secondC(arr[i]);
                }
                else{
                    LRUSoftware(arr[i]);
                }
            }
        }

        //-------------------
        key = arr[i];
        j = i - 1;
        //-------------------

        if (!findInMemory(arr[j]))  //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
        {
            int vmIndex=findInVM(arr[j]);
            int diskIndex=findInDisk(arr[j]);
            if (pmCounter<memorySize)
            {
                pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
                vm.pagetable[vmIndex].physicalFrame=pmCounter;
                vm.pagetable[vmIndex].rBit=1;
                vm.pagetable[vmIndex].validBit=1;
                fifolist[pmCounter]=disk.diskPages[diskIndex];
                pmCounter++;
            }
            else{                                                       //Eğer memory doluysa algoritmaya göre page replacement yapılır.
                if (algorithm==1)
                {
                    firstInFirstOut(arr[j]);
                }
```

```
                else if(algorithm==2){
                    secondC(arr[j]);
                }
                else{
                    LRUSoftware(arr[j]);
                }
            }
        }

        // Move elements of arr[0..i-1],
        // that are greater than key, to one
        // position ahead of their
        // current position
        //-------------------------------------
        while (j >= 0 && arr[j] > key)
        {
        //-------------------------------------

            if (!findInMemory(arr[j+1]))  //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
            {
                int vmIndex=findInVM(arr[j+1]);
                int diskIndex=findInDisk(arr[j+1]);
                if (pmCounter<memorySize)
                {
                    pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
                    vm.pagetable[vmIndex].physicalFrame=pmCounter;
                    vm.pagetable[vmIndex].rBit=1;
                    vm.pagetable[vmIndex].validBit=1;
                    fifolist[pmCounter]=disk.diskPages[diskIndex];
                    pmCounter++;
                }
                else{                                                   //Eğer memory doluysa algoritmaya göre page replacement yapılır.
                    if (algorithm==1)
                    {
                        firstInFirstOut(arr[j+1]);
                    }
                    else if(algorithm==2){
                        secondC(arr[j+1]);
                    }
                    else{
                        LRUSoftware(arr[j+1]);
                    }
                }
            }
            if (!findInMemory(arr[j]))  //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
            {
                int vmIndex=findInVM(arr[j]);
                int diskIndex=findInDisk(arr[j]);
                if (pmCounter<memorySize)
                {
                    pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
                    vm.pagetable[vmIndex].physicalFrame=pmCounter;
                    vm.pagetable[vmIndex].rBit=1;
                    vm.pagetable[vmIndex].validBit=1;
                    fifolist[pmCounter]=disk.diskPages[diskIndex];
                    pmCounter++;
                }
```

```
                    firstInFirstOut(arr[j]);
                }
                else if(algorithm==2){
                    secondC(arr[j]);
                }
                else{
                    LRUSoftware(arr[j]);
                }
            }
        }

    //-----------------------------
        arr[j + 1] = arr[j];
        j = j - 1;
    //-----------------------------
    }

    if (!findInMemory(arr[j]))  //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
    {
        int vmIndex=findInVM(arr[j]);
        int diskIndex=findInDisk(arr[j]);
        if (pmCounter<memorySize)
        {
            pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
            vm.pagetable[vmIndex].physicalFrame=pmCounter;
            vm.pagetable[vmIndex].rBit=1;
            vm.pagetable[vmIndex].validBit=1;
            fifolist[pmCounter]=disk.diskPages[diskIndex];
            pmCounter++;
        }
        else{                                                   //Eğer memory doluysa algoritmaya göre page replacement yapılır.
            if (algorithm==1)
            {
                firstInFirstOut(arr[j]);
            }
            else if(algorithm==2){
                secondC(arr[j]);
            }
            else{
                LRUSoftware(arr[j]);
            }
        }
    }
    arr[j + 1] = key;
  }
}
```

Classic insertion sort is applied.The replacement rules that I applied in the previous algorithm were also applied.

Bubble Sort

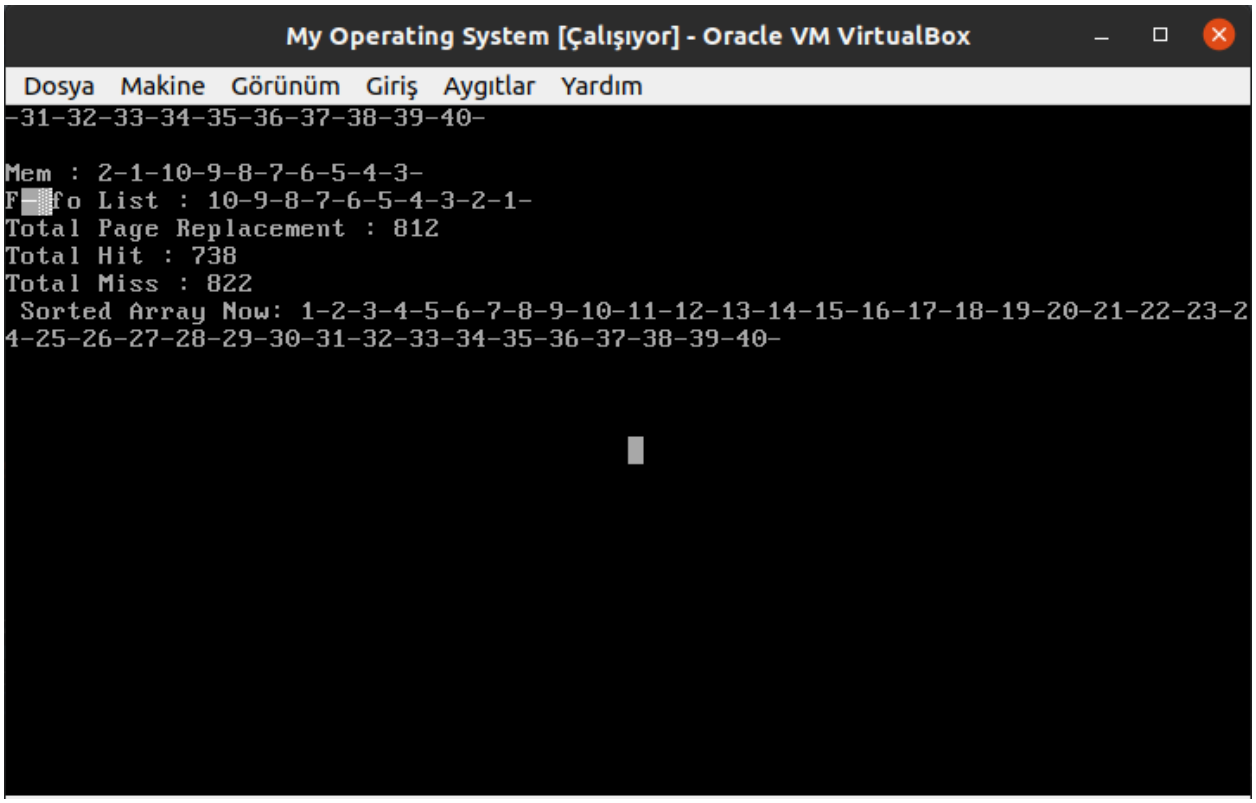
```
//------------SORT ALGORITHM-----------------------------------------------------------
void bubbleSort(int arr[], int n,int algorithm){              // bubble sort function

    for(int i = 0; i < n; i++){
        for(int j = 0; j < n-i-1; j++){         // loop until n-i-1(further is unnecessary)

            if (!findInMemory(arr[j]))  //Page replacement for arr[j] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
            {
                int vmIndex=findInVM(arr[j]);
                int diskIndex=findInDisk(arr[j]);
                if (pmCounter<memorySize)
                {
                    pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
                    vm.pagetable[vmIndex].physicalFrame=pmCounter;
                    vm.pagetable[vmIndex].rBit=1;
                    vm.pagetable[vmIndex].validBit=1;
                    fifolist[pmCounter]=disk.diskPages[diskIndex];
                    pmCounter++;

                }
                else{                                                       //Eğer memory doluysa algoritmaya göre page replacement yapılır.
                    if (algorithm==1)
                    {
                        firstInFirstOut(arr[j]);
                    }
                    else if(algorithm==2){
                        secondC(arr[j]);
                    }
                    else{
                        LRUSoftware(arr[j]);
                    }
                }

            }
            if (!findInMemory(arr[j+1]))  //Page Replacement for arr[j+1] + Eğer memory dolu değilse direkt memory'e ekleme yapılır.
            {
                int vmIndex=findInVM(arr[j+1]);
                int diskIndex=findInDisk(arr[j+1]);
                if (pmCounter<memorySize)
                {
                    pm.pages[pmCounter]=disk.diskPages[diskIndex]; //get page from disk
                    vm.pagetable[vmIndex].physicalFrame=pmCounter;
                    vm.pagetable[vmIndex].rBit=1;
                    vm.pagetable[vmIndex].validBit=1;
                    fifolist[pmCounter]=disk.diskPages[diskIndex];
                    pmCounter++;

                }
                else{                                   //Eğer memory doluysa algoritmaya göre page replacement yapılır.
                    if (algorithm==1)
                    {
                        firstInFirstOut(arr[j+1]);
                    }
                    else if(algorithm==2){
                        secondC(arr[j+1]);
                    }
                    else{
                        LRUSoftware(arr[j+1]);
                    }
```

```

                }
                else{                                   //Eğer memory doluysa algoritmaya göre page replacement yapılır.
                    if (algorithm==1)
                    {
                        firstInFirstOut(arr[j+1]);
                    }
                    else if(algorithm==2){
                        secondC(arr[j+1]);
                    }
                    else{
                        LRUSoftware(arr[j+1]);
                    }
                }

            }
            if( arr[j] > arr[j+1]){
                int temp = arr[j];              // swap elements for sorting them
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

Classic bubble sort is applied.The replacement rules that I applied in the first algorithm were also applied.

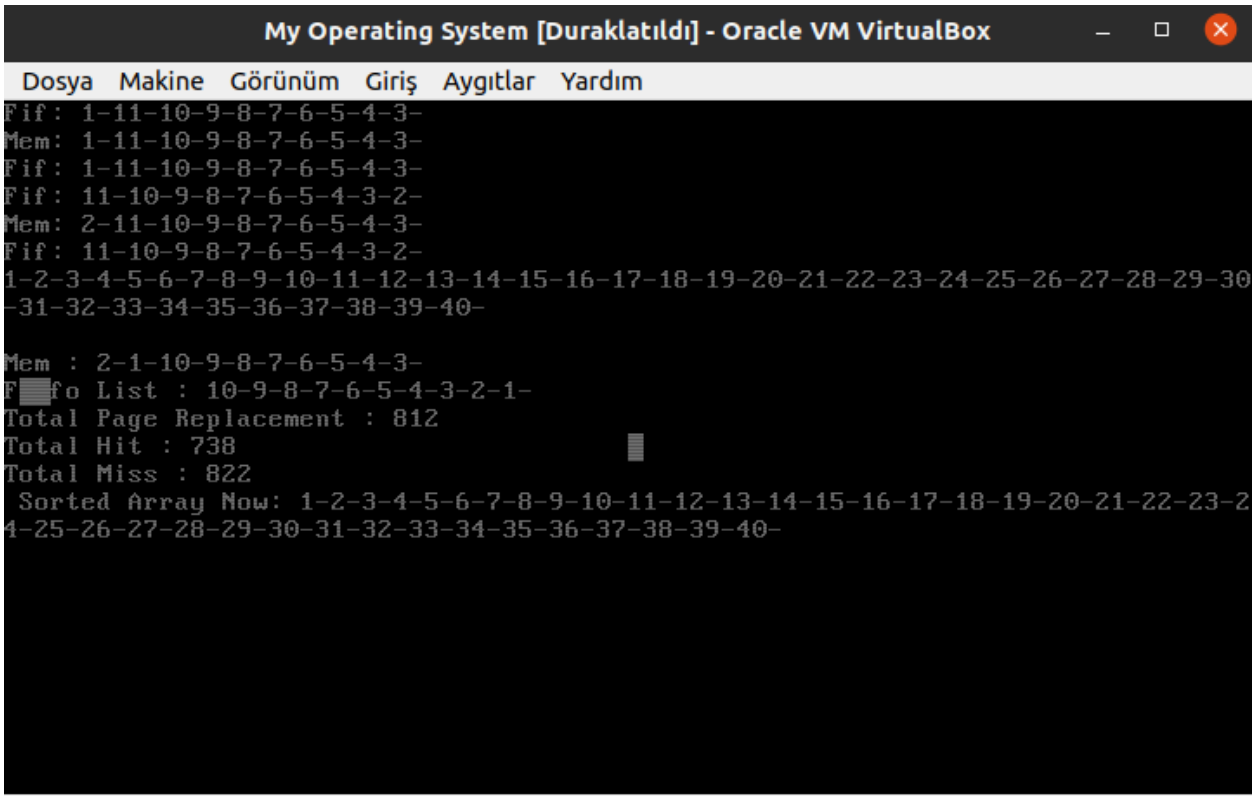
SCREEN S.

FİFO - BUBBLE



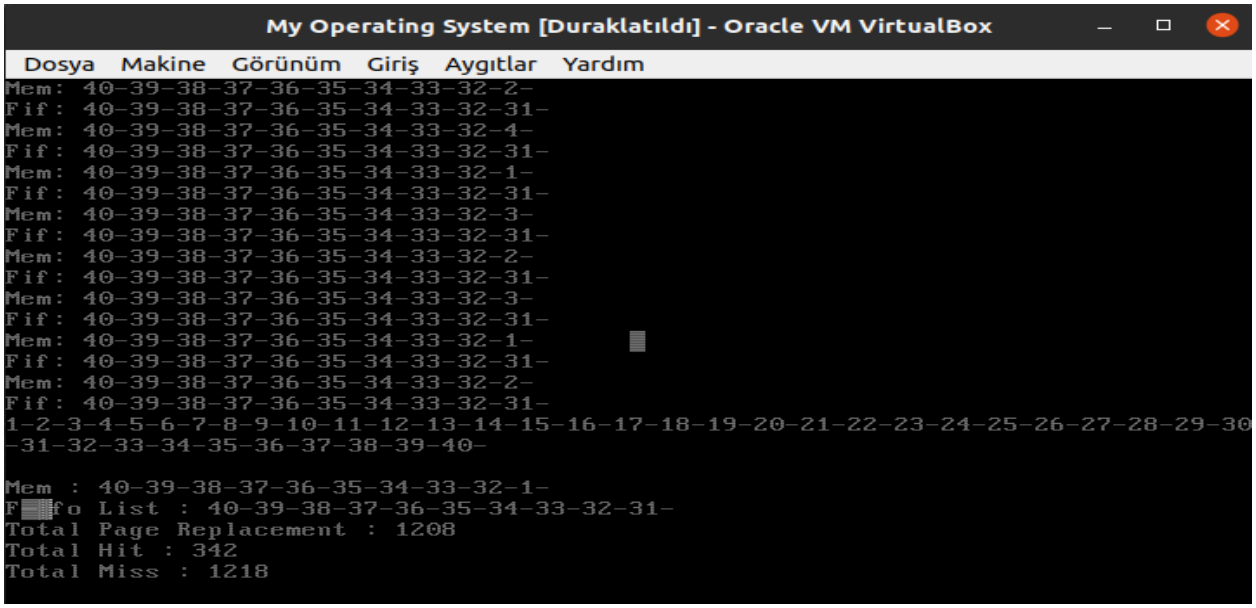
There are sorted versions of Array, some information, and final versions of fifo list memory.

Second Change - BUBBLE



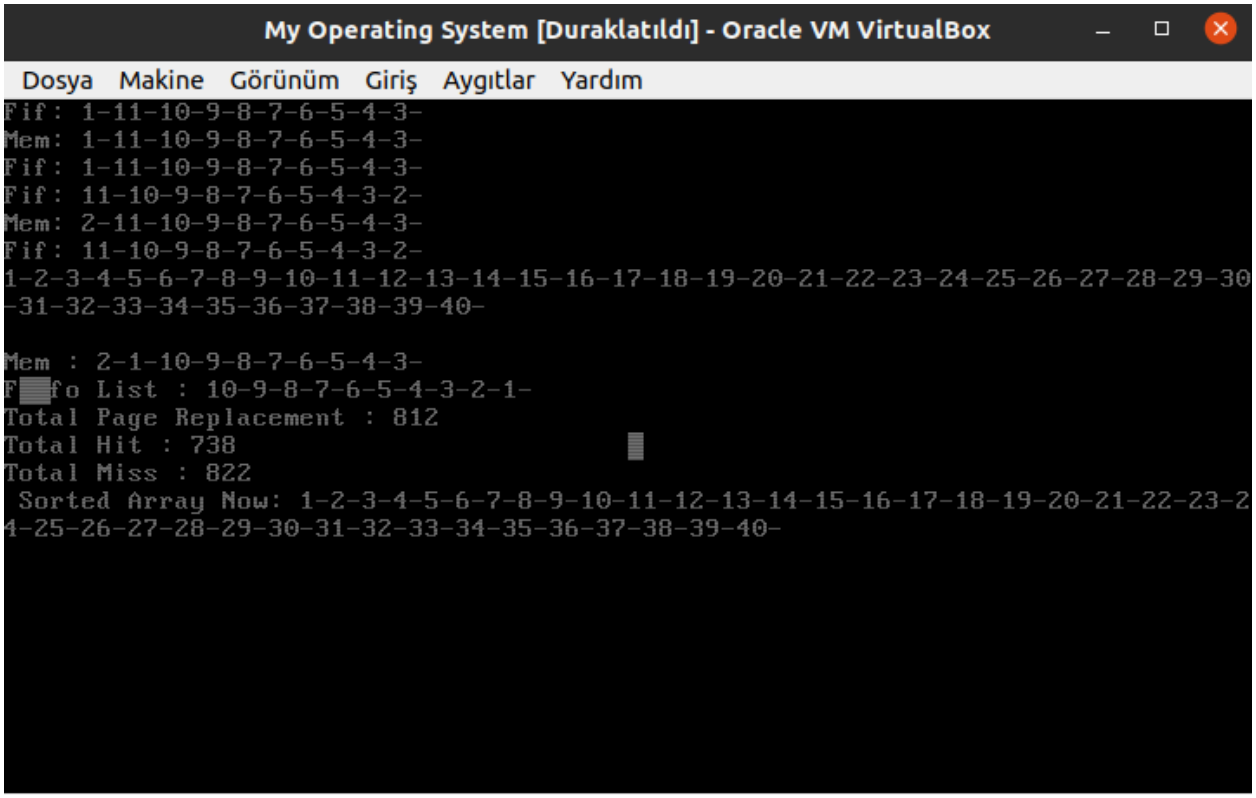
There are sorted versions of Array, some information, and final versions of fifo list memory.

LRU - BUBBLE

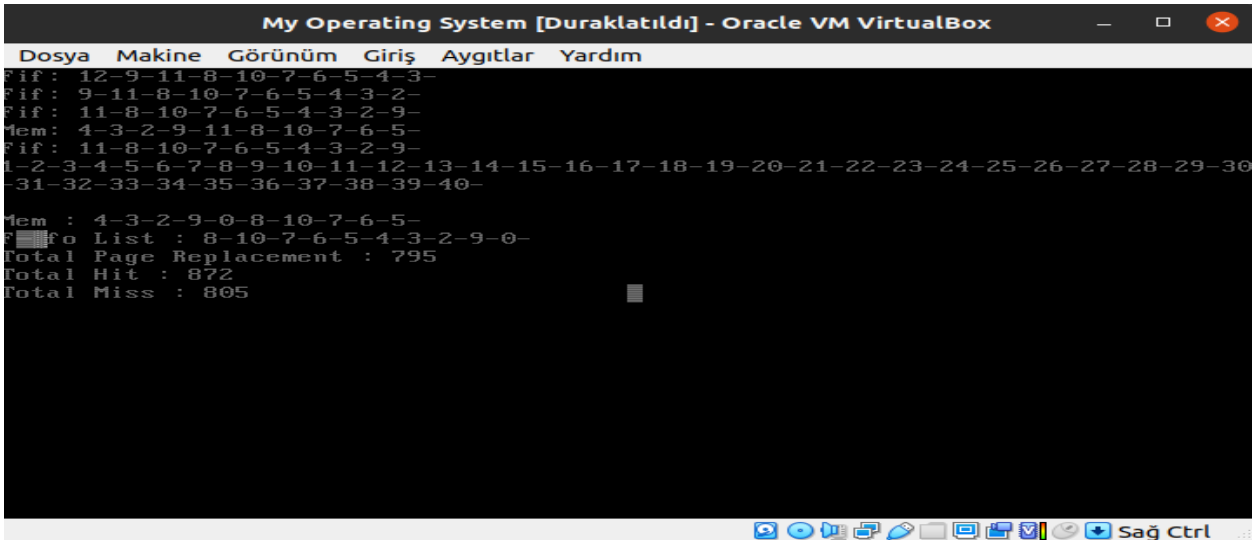


There are sorted versions of Array, some information, and final versions of fifo list memory.

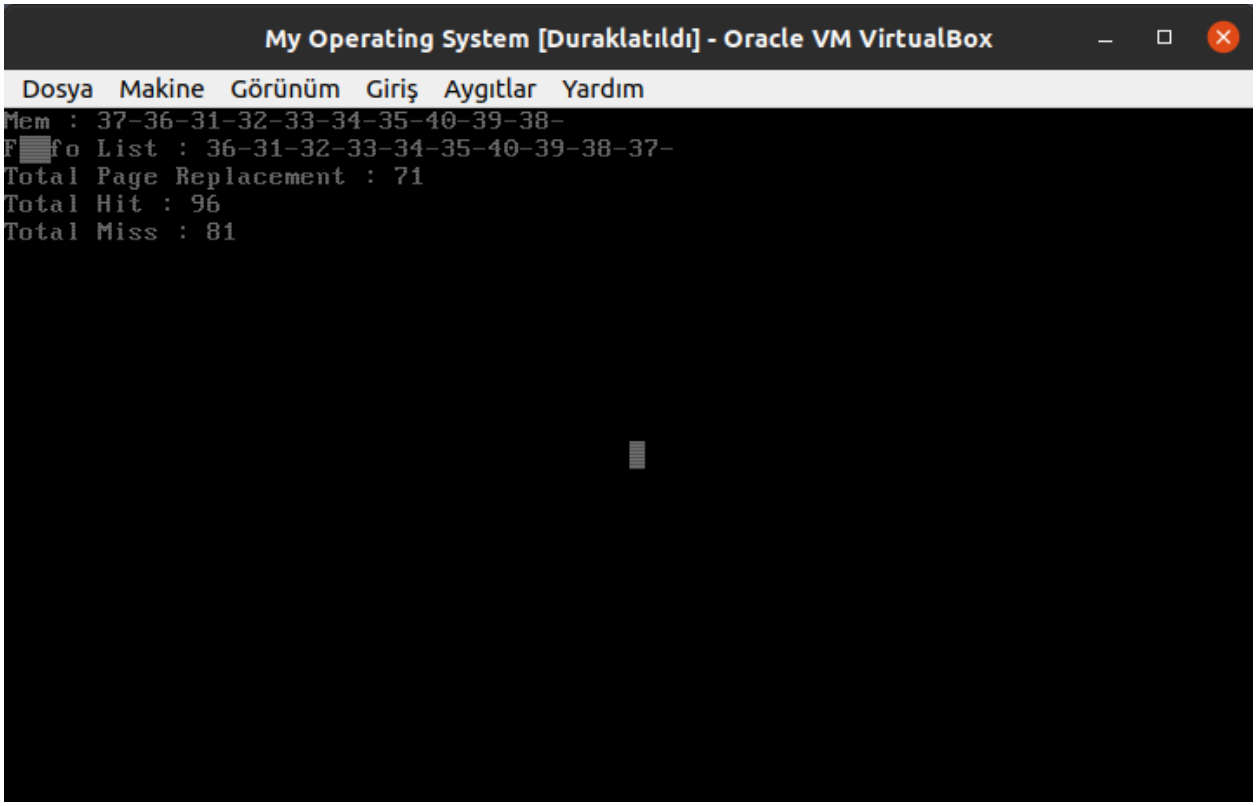
Second Chance - BUBBLE



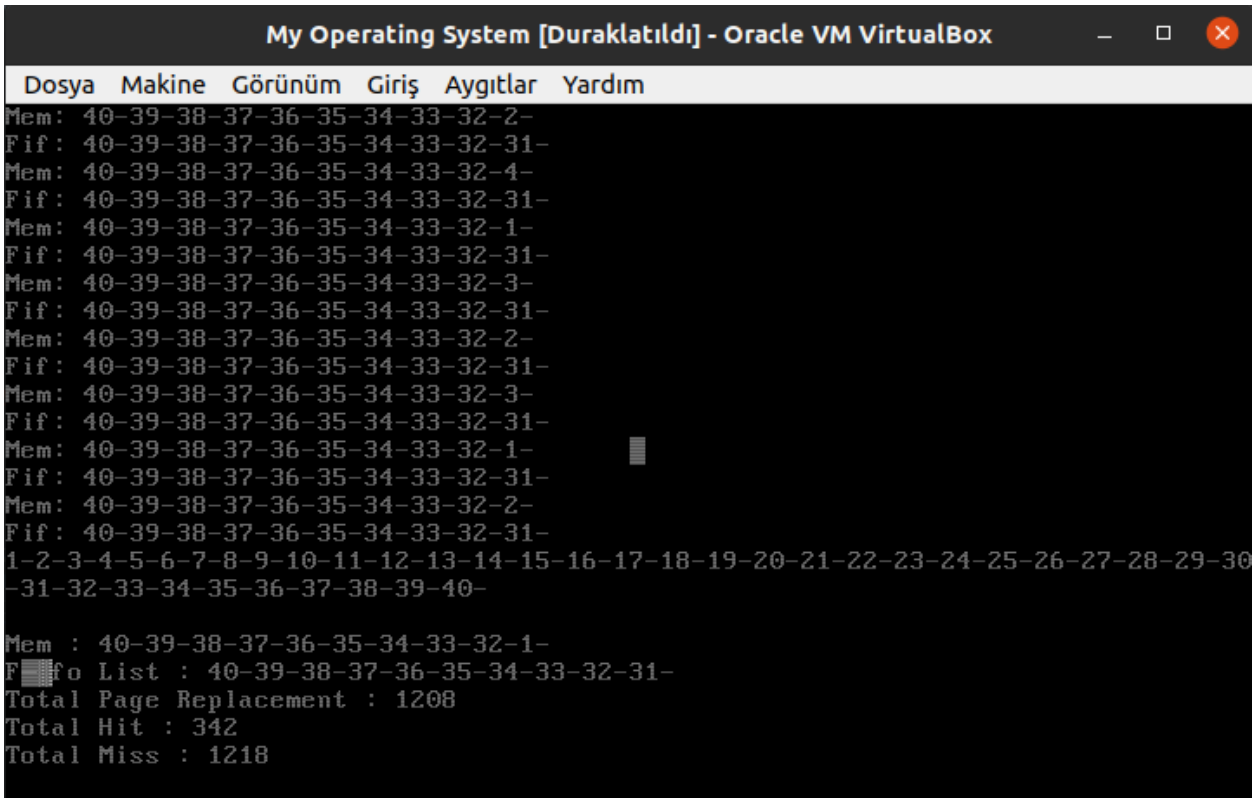
Second Chance - INSERTION



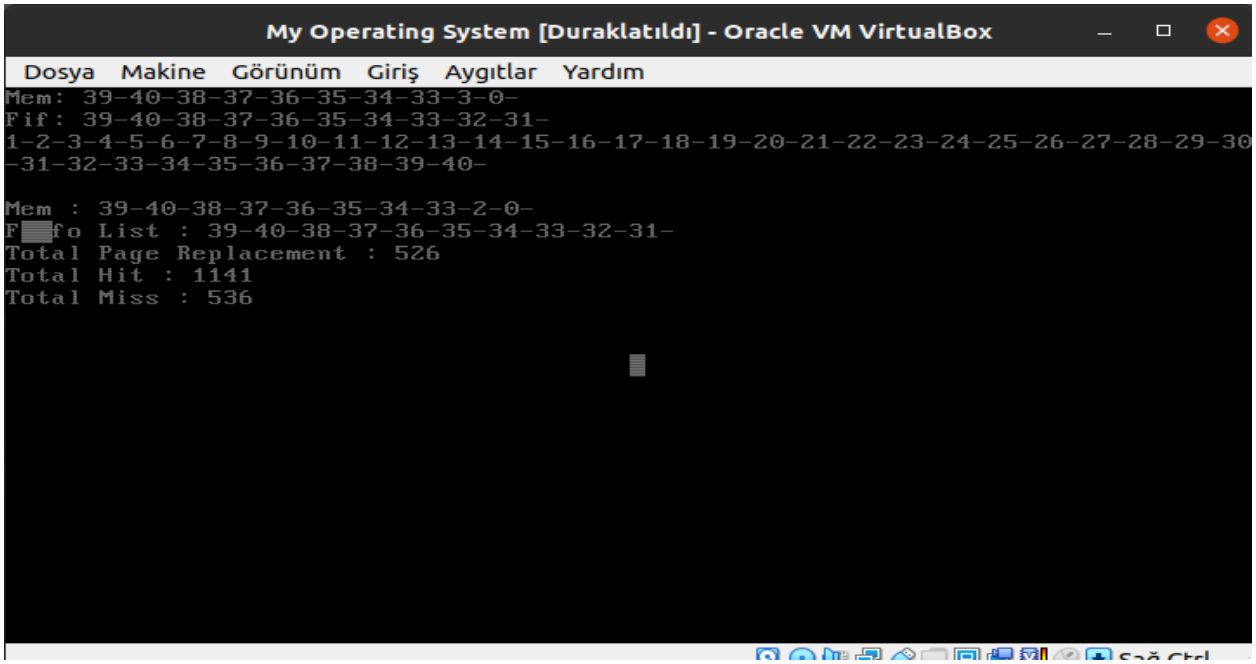
Second Chance - QUICK



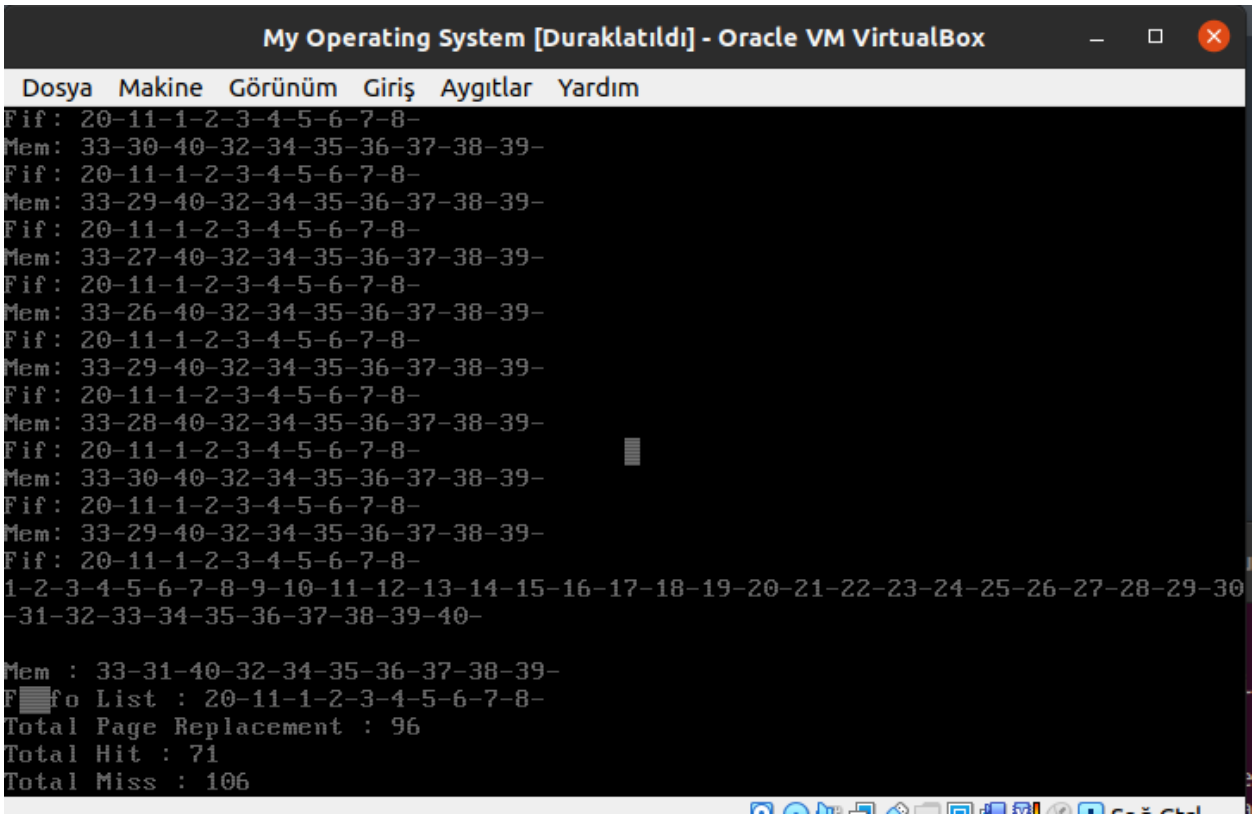
LRU - BUBBLE



LRU- INSERTION



LRU - QUICK



General Summary

In this algorithm, I get an array to be sorted.Then I find each element of this array as a page and initialize it to virtual memory and disk.Then the sort algortihm and page replacement algorithm to be used are selected.Then, if each index used in the array is not in the memory, it is brought from the disk and the page table is updated.Here, the element in the index of the array is represented by page.If the used page is in memory, it is used directly. If not, and there is space in the memory, it is added to the memory and used.If the used page is in memory, it is used directly. If not, and there is space in the memory, it is added to the memory and used.This continues until the entire array has been sorted.