Operating Systems

Project Report

DEPT: Bio 3rd year

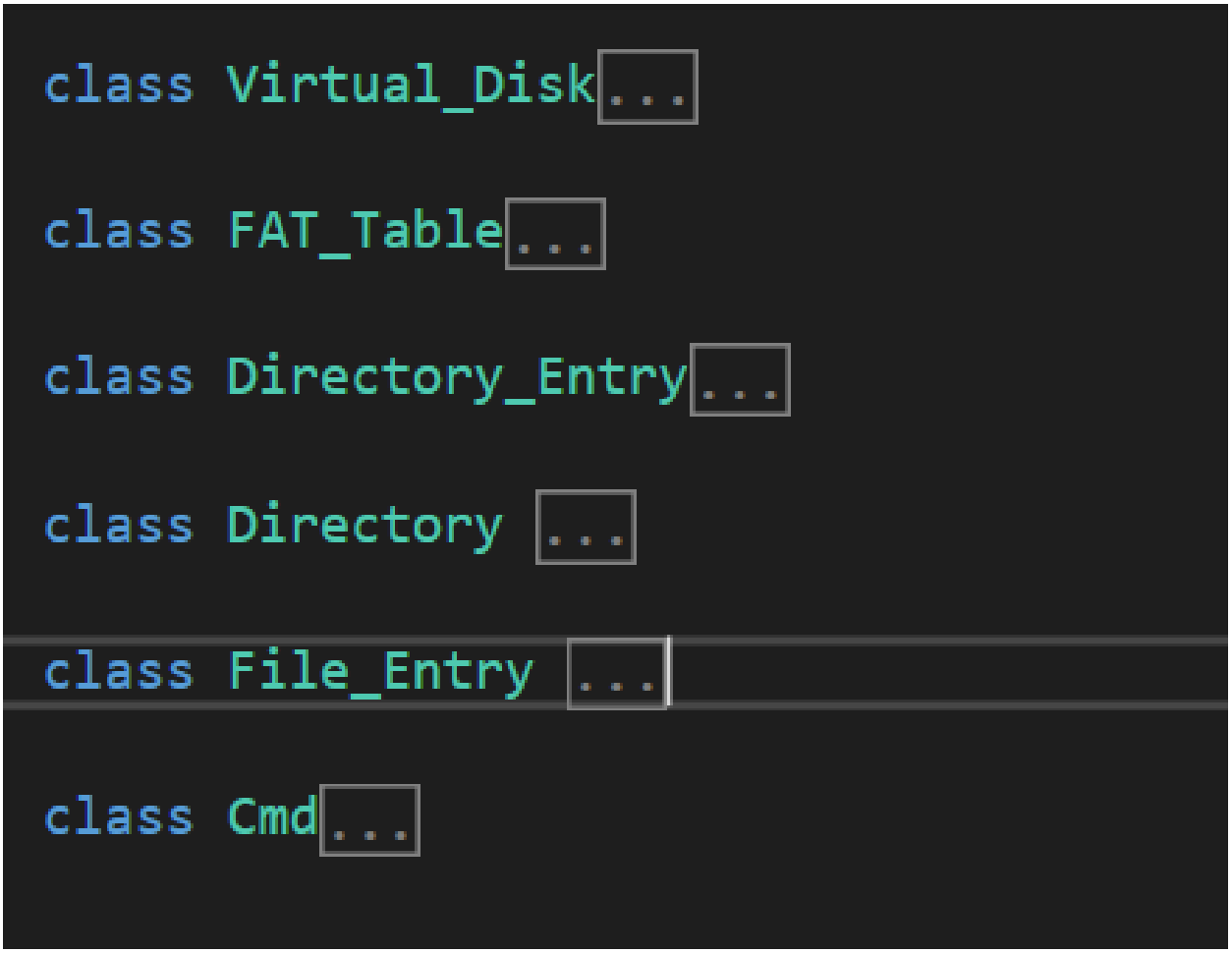
Team members:

-Ahmed Saleh

-Al-Hassan Mohamed

-Gamila Mohamed

The project was made using C# and it consists of 5 main classes as shown in the following picture.



This report will contain a brief description of each class and its methods.

Virtual_Disk class:

This class is responsible for making our 1 mb virtual file.

```
public static void Initialize()
{

    if (!File.Exists("Virtual Disk.txt"))
    {
        using (FileStream stream = new FileStream("Virtual Disk.txt", FileMode.Create, FileAccess.ReadWrite))
        {
            for (int j = 0; j < 1024; j++)
                stream.WriteByte((byte)'0');

            for (int i = 0; i < 4; i++)
                for (int j = 0; j < 1024; j++)
                    stream.WriteByte((byte)'*');

            for (int i = 0; i < 1019; i++)
                for (int j = 0; j < 1024; j++)
                    stream.WriteByte((byte)'#');

            FAT_Table.Initialize_FAT();
            Directory root = new Directory("C".ToCharArray(), 0x10, 5);
            stream.Close();
            root.Write_Directory();
            FAT_Table.Write_FAT();
            Program.current_dir = root;
        }
    }

    else
    {
        FAT_Table.Read_FAT();
        Directory root = new Directory("C".ToCharArray(), 0x10, 5);
        if (FAT_Table.Get_FAT_Value(5) != 0)
            root.Read_Directory();
        Program.current_dir = root;

    }
```

`Initialize()`

-Creates 1mb virtual disk file if doesn`t exist and fills it with:

1- Super cluster of zeros in the first cluster.

2- FAT information in the next 4 clusters.

3- Root directory in the 5^{th} cluster.

-If the file is already created then it loads the FAT information and the root directory from it directly to begin working.

```
public static void Write_Cluster(byte[] block, int index)
{
    using (FileStream stream = new FileStream("Virtual Disk.txt", FileMode.Open, FileAccess.ReadWrite))
    {

        stream.Seek(index * 1024, SeekOrigin.Begin);

        for (int i = 0; i < 1024; i++)
            stream.WriteByte(block[i]);

    }

}
public static byte[] Read_Cluster(int index)
{
    using (FileStream stream = new FileStream("Virtual Disk.txt", FileMode.Open, FileAccess.ReadWrite))
    {
        byte[] Readblock = new byte[1024];

        stream.Seek(1024 * index, SeekOrigin.Begin);
        stream.Read(Readblock, 0, 1024);

        return Readblock;

    }
}
```

Write_Cluster(byte[] block, int index)

This method writes a block of bytes (cluster) that contain data in a specific position into the virtual disk file.

Read_Cluster(int index)

This method reads a block of bytes (cluster) with that index in the file.

FAT_Table class

```
static int[] FAT = new int[1024];

public static void Initialize_FAT()
{
    FAT[0] = -1;FAT[1] = 2;FAT[2] = 3;FAT[3] = 4;FAT[4] = -1;
    for (int i = 5; i < FAT.Length; ++i)
        FAT[i] = 0;
}
```

This class contains FAT array which is an array of 1024 integer to store the FAT information within the file and its methods serve the purpose of saving these information and update it if any changes occur in the file system.

Initialize_FAT()

Initializes the fat information of the first 5 clusters with -1,2,3,4,-1 and the rest of them with 0 .

```
public static void Write_FAT()
{
    using (FileStream file = new FileStream("VIRTUAL DISK.txt", FileMode.Open, FileAccess.Write))
    {
        byte[] FAT_in_bytes = new byte[4096];
        Buffer.BlockCopy(FAT, 0, FAT_in_bytes, 0, FAT.Length);

        file.Seek(1024, SeekOrigin.Begin);

        for (int i = 0; i < FAT_in_bytes.Length; i++)
            file.WriteByte(FAT_in_bytes[i]);

        file.Close();

    }
}
```

Write_fat()

Writes FAT information into the virtual disk file these information specifies which clusters are empty and the ones are full of data and the clusters that points to other clusters.

```
public static void Read_FAT()
{
    using (FileStream file = new FileStream("VIRTUAL DISK.txt", FileMode.Open, FileAccess.Read))
    {
        byte[] ReadFat = new byte[4096];

        file.Seek(1024, SeekOrigin.Begin);

        file.Read(ReadFat, 0, 4096);

        Buffer.BlockCopy(ReadFat, 0, FAT, 0, 4096);

        file.Close();
    }
}
```

Read_fat()

Reads FAT information from the virtual file system and it`s the opposite of the method Write_Cluster().

```
public static int Get_FreeCluster()
{

    for (int i = 0; i < FAT.Length; i++)
    {
        if (FAT[i] == 0)
            return i;
    }

    return -1;
}

public static int Get_FAT_Value(int index)
{
    return FAT[index];
}

public static void Set_FAT_Value(int index, int value)
{
    FAT[index] = value;
}

public static int Get_AvailbaleClusters()
{
    int counter = 0;

    for (int i = 0; i < FAT.Length; i++)
    {
        if (FAT[i] == 0)
            counter += 1;

    }

    return counter;
}

public static int Get_FreeSpace()
{
    return (Get_AvailbaleClusters() * 1024);
}
```

Get_FreeCluster()

Returns the positions of the first free cluster in the file.

Get_FAT_Value(int index)

Gets the value of the FAT array with that index.

Set_FAT_Value(int index ,int value)

Sets a value to a FAT element of that index.

Get_Available_clusters()

Counts the number of free clusters of the file.

Get_Available_Space()

calculates the free space left in the file.

Directory_Entry Class

```
public char[] name = new char[11];
public byte attribute;
byte[] empty = new byte[12];
public int size;
public int first_cluster;

Directory_Entry() { }

public Directory_Entry(char[] name, byte attribute, int first_cluster, int size = 0)
{
    if (name.Length > 11)
        for (int i = 0; i < 11; i++)
            this.name[i] = name[i];

    else
        this.name = name;

    this.attribute = attribute;
    this.first_cluster = first_cluster;
    this.size = size;

}
```

This class purpose is to make an entry of each directory or file in the file system, this class is very important as there are 2 other classes that inherit from him.

The parameterized constructor initiates the field of the class with these parameters.

```
public Directory_Entry Read_Directory_Entry()
{
    Directory_Entry obj = new Directory_Entry();

    obj.name = this.name;

    obj.attribute = this.attribute;

    obj.empty = this.empty;

    obj.first_cluster = this.first_cluster;

    obj.size = this.size;

    return obj;
}

public Directory_Entry Read_Directory_Entry(byte[] b)...
```

Those two methods have the same purpose which is reading directory entry from a block of bytes or from the object that called the method.

Directory class inherits from Directory_Entry Class

This class purpose is to write, read, search, update and delete directories and it contains list of directory entries and a parent directory as attributes.

```
public void Write_Directory()//
{

    byte[] DTB = new byte[32 * Directory_table.Count];
    byte[] DEB = new byte[32];

    for (int i = 0; i < Directory_table.Count; i++)
    {

        DEB = Directory_table[i].Get_Bytes_of_DirEntry();

        for (int j = i * 32, c = 0; c < 32; j++, c++)
        {
            DTB[j] = DEB[c];
        }

    }

    double num_of_req_blocks = Math.Ceiling(DTB.Length / 1024.0);

    if (num_of_req_blocks <= FAT_Table.Get_FreeSpace())
    {
        int FI;
        int LI = -1;
```

```
        if (first_cluster != 0)
            FI = first_cluster;

        else
        {

            FI = FAT_Table.Get_FreeCluster();
            first_cluster = FI;
        }


        byte[] clone = new byte[1024];

        for (int i = 0; i < num_of_req_blocks; i++)
        {
            for (int j = i * 1024, c = 0; c < 1024; j++, c++)
            {

                if (j < DTB.Length)
                    clone[c] = DTB[j];

                else
                    clone[c] = (byte)'*';

            }

            Virtual_Disk.Write_Cluster(clone, FI);
            FAT_Table.Set_FAT_Value(FI, -1);

            if (LI != -1)
                FAT_Table.Set_FAT_Value(LI, FI);


            LI = FI;
            FI = FAT_Table.Get_FreeCluster();

        }
```

Write_Directory()

-Creates a directory table that will store all the directory entries that will be created and then checks if there is enough space to write that directory and if so it writes that directory and then update the FAT information.

Read_Directory()

Reads a directory from the virtual file and it`s the opposite of the method Write_Directory().

```
public int Search(string name)//
{
    name = name.TrimEnd(new char[] { '\0' });
    string str;

    for (int i = 0; i < Directory_table.Count; i++)
    {

        str = new string(Directory_table[i].name).TrimEnd(new char[] { '\0' });

        if (str == name)
            return i;

    }

    return -1;

}
```

Search()

The most important method in the program it takes the name of the directory and searches for it in the directory table which contains directory entry if it’s found then it will return its position in the directory table If not found it will return -1.

```
public void Delete_Directory()//
{
    if (first_cluster != 0)
    {
        int index = first_cluster;
        int next = FAT_Table.Get_FAT_Value(index);

        do
        {
            FAT_Table.Set_FAT_Value(index, 0);
            index = next;
            if (index != -1)
                next = FAT_Table.Get_FAT_Value(index);

        } while (next != -1);

        FAT_Table.Write_FAT();
    }
    if (Parent != null)
    {
        string file_name = new string(name);
        Parent.Read_Directory();
        int i = Parent.Search(file_name);

        if (i != -1)
        {
            Parent.Directory_table.RemoveAt(i);
            Parent.Write_Directory();
        }
    }

}
```

Delete_Directory()

Deletes the directory with its contents and if it has a parent so it will remove its information from the parent directory entry.

File_Entry class inherits from Directory_Entry

This class is responsible for creating, reading, deleting files from the virtual file system. It has a two fields: parent directory and the content of the file.

```
public void Write_File()
{

    double num_of_req_blocks = Math.Ceiling(content.Length / 1024.0);


    if (num_of_req_blocks <= FAT_Table.Get_FreeSpace())
    {
        int FI;
        int LI = -1;

        if (first_cluster != 0)
            FI = first_cluster;

        else
        {
            FI = FAT_Table.Get_FreeCluster();
            first_cluster = FI;
        }
```

```
    byte[] tempCLuster = new byte[1024];
    for (int i = 0; i < num_of_req_blocks; i++)
    {
        for (int j = i * 1024, c = 0; c < 1024; j++, c++)
        {

            if (j < this.content.Length)
                tempCLuster[c] = (byte)this.content[j];

            else
                tempCLuster[c] = (byte)'*';

        }


        Virtual_Disk.Write_Cluster(tempCLuster, FI);
        FAT_Table.Set_FAT_Value(FI, -1);

        if (LI != -1)
            FAT_Table.Set_FAT_Value(LI, FI);

        LI = FI;

        FI = FAT_Table.Get_FreeCluster();

    }
    FAT_Table.Write_FAT();
}
```

Write_File()

Writes a file into the virtual file system and updates its FAT information.

Read_File()

Reads a file from the virtual file system.

```
public void Delete_File()
{
    if (first_cluster != 0)
    {
        int index = first_cluster;
        int next = FAT_Table.Get_FAT_Value(index);

        do
        {
            FAT_Table.Set_FAT_Value(index, 0);
            index = next;
            if (index != -1)
                next = FAT_Table.Get_FAT_Value(index);

        } while (next != -1);

        FAT_Table.Write_FAT();
    }
    if (Parent != null)
    {
        string file_name = new string(name);
        Parent.Read_Directory();
        int i = Parent.Search(file_name);

        if (i != -1)
        {
            Parent.Directory_table.RemoveAt(i);
            Parent.Write_Directory();
        }
    }

}
```

Delete_File()

Deletes a file from the virtual file system and update its parent directory information if it a has a parent directory.

CommadLine class

This class is responsible for creating the environment which will take the commands and its methods will interact with the virtual file system.

Initialize()

Setting up the environment which will take the commands and checks it validity and if it's a valid command then it checks its arguments.

md(string name)

It makes a new directory in the file system on condition that it's doesn't already exist in the virtual file system.

rd(string name)

It removes a directory from the file system on condition that it's in the virtual file system.

dir()

Lists the contents of the current directory of directories and files and counts the number of each of them and the space occupied by this directory and the free space left in the virtual file system.

rename(string old, string new)

Renames a file or a directory.

del(string name)

Deletes a file from the virtual file system.

copy(string source, string destination)

Copies files and directories from the source to the destination.

import(string path)

Imports a file from the virtual disk to the virtual file system.

export(string source, string destination)

Exports a file from the virtual file system to the virtual disk.

type()

Shows the contents of the file.

Cd(string name)

Displays the name of or changes the current directory.

help()

Provides Help information for Windows commands.

Cls()

Clears the screen.

Quit()

Quits the CMD.EXE program (command interpreter).