DUFS v1

Kaden VanPortfleet

Davenport University

CSCI430: Operating Systems

Mike Viola

April 28, 2023

Goal:

The goal of this project is to develop a system for storing files. The files will be stored in volumes that can be created from the terminal. Commands mount, unmount, truncate, delete, allocate, deallocate, info, set, dump, catalog, read, and write should be implemented at the volume and/or the file level for manipulation of data in the file system. The operating system is to operate by manipulating the bytes of each file, where each unique file is a volume.

Approach:

The approach taken is quite complex. To accurately include all desired functionality, multiple processes need to make predictions and estimates about file sizes, free space, and volume size to alleviate issues of corrupt data.

Algorithm foundation:

The backbone of the DUFS program is the linear search included in most of its methods. This linear search is built to not only find files of *unique naming*, but also is modified in methods, like the *create(),* to recognize how big a file is going to be and then find a space large enough in the volume to fit it. This linear search system is included in all methods where locating a file is necessary. The file system automatically handles the placing of files in a given volume.

This linear search function is uniquely different from a common file system like FAT as it does not use a table to find files. The major difference between the two comes in the form of how data is physically stored. In the FAT file system, the data can be segmented and is referenced through a pointer to clusters in the storage. In my system, all files are stored contiguously. This means that all the data for a file is stored in byte regions *right next to each other.* This has a disadvantage of being slower on random read/write access BUT could provide a benefit in sustained reads of large files. This advantage is particularly helpful with spinning hard drives as it would prevent the read-head from skipping around to different disk regions to read one file. There would be no need for defragmentation.

Identifying files and their attributes:

Each file in a volume follows a predefined format for defining how it is read. Multiple bytes values are used in order to define where one region of a file begins and another ends. To start from the beginning, let's assume that a user allocates some storage to a volume of 100 bytes. The user then mounts this volume using the command line and then wants to create a file of 10 bytes in the volume. When the user issues a command to create the file, the file system program begins a linear search in the volume space. When it finds an available spot in the volume (the first unused 0), it then extrapolates the files' size and checks if the space that it found is long enough to fit the file. If it is, it begins writing the file. If not, it continues its search. The program will return a message warning the user if no space in the volume was found.

Writing a file:

Once an eligible space has been found the program begins writing the file. First, we place the start byte which is signified by a byte value of '1'. This byte allows the file system to recognize the beginning of a file in the volume. Next, the file system places the byte values of the file name into the region one byte at a time. Once the name is written out as bytes, the program ends this region with a '2' byte. The '2' byte marks the end of the name section and the beginning of the properties section. Again, here we write the ascii byte values for these properties into the file space on the volume. Each property (size, create data, create time, last modified date, last modified time, readonly) is separated in this region by a byte value of '1'. The last property, readonly, uses byte values of '4' and '5' for readonly=false and readonly=true respectively. After these

properties, this section is delimited by a byte value of ‘3’. This marks the end of the properties section and the beginning of the “data” section. This section is to hold the actual data the user wants to store. Finally, at the end of the data section a byte value of ‘6’ is used to mark the end of file (EoF).

This file structure is crucial to the operation of the file system as it is how the program is able to find sections in each file. The search algorithm first finds the file in the volume space then uses these bytes as flags to identify each section. Then from that flag we can either find other flags or manipulate that region's values.

Commands:

Commands for the file system come in two hierarchies, volume level and file level. As their names suggest, commands done at the volume level are designed to manipulate volumes. While commands at the file level are designed to manipulate files in a given volume. To access the file commands, one must mount a volume. To return to volume commands, you simply need to issue the unmount command. The commands are as listed below:

Note: Regions in the command line examples surrounded by (*) are for user determined inputs. Do not include (*)s in your commands.

Volume:

Allocate - This creates a volume on the disk of specified size. Usage: *allocate *volume name**

Deallocate - This deletes a volume from the disk. Usage: *deallocate *volume name**

Truncate - This clears a volume of all data. Usage: *truncate *volume name**

Dump - This dumps all bytes in the volume to the terminal as their hex values. Usage: *dump *volume name**

Mount - This mounts a volume for manipulation of its files. Usage: *mount *volume name**

Info - This displays the size of a volume. Usage: *info *volume name**

File:

Create - This creates a file in the volume of specified name and size. Usage:

*create *file name* *file size**

Read - This reads the data from a file from a specified start offset to an end offset in the file. Usage:

*read *file name* *start offset* *end offset**

Write - This writes data to a file starting at a specified start offset. Usage:

*write *file name* *start offset**

Delete - This completely deletes a file from the volume. Usage:

*delete *file name**

Truncate - This clears and resets a file in the volume. Usage:

*truncate *file name**

Catalog - This lists all of the files in the mounted volume. Usage:

catalog

Info - This displays the attributes of a specified file. Usage:

*info *file name**

Set - This sets the readonly status of a specified file. There are two options for commands “readonly=true” and “readonly=false”. Setting a file as readonly prevents it from being deleted. Usage:

*set *file name* *readonly=X**

Unmount - This unmounts the current volume and returns control to the volume command list. Usage:

unmount

Code:

```
using System.ComponentModel;
using System.IO;
using System.Runtime.CompilerServices;
using System.Security.Cryptography;
using System.Xml.Linq;

namespace DUFS
{

    class Program
    {
        string dir = "..\\..\\..\\FS\\";
        public static void Main(string[] args)
        {
            Program p = new Program();

            Console.WriteLine("Welcome to the Davenport University File System (DUFS)");

            while (true)
            {

                Console.WriteLine("Please enter a command:");
                string input = Console.ReadLine();
                string[] inputArgs = input.Split(" ");

                switch (inputArgs[0])
                {
                    case "allocate"://ALLOCATE <VOLUMENAME>, <SIZE>: Create a file system called VOLUMENAME that contains SIZE bytes.
                        p.allocate(inputArgs[1], Convert.ToInt32(inputArgs[2]));
                        break;
                    case "deallocate"://DEALLOCATE <VOLUMENAME>: Physically deletes the volume called VOLUMENAME.
                        p.deallocate(inputArgs[1]);
                        break;
                    case "truncate"://TRUNCATE <VOLUMENAME>: Initializes (erases) the volume called VOLUMENAME.
                        p.truncate(inputArgs[1]);
                        break;
                    case "dump"://DUMP <VOLUMENAME>: Displays the contents of the volume called VOLUMENAME on the screen.
                        p.dump(inputArgs[1]);
                        break;
                    case "mount":
                        Console.WriteLine($"Mounting {inputArgs[1]}...");
                        p.volume(p, inputArgs[1]);
                        break;
                    case "info"://Display size of specified volume.
                        p.volInfo(inputArgs[1]);
                        break;
                }
            }


        }

        public void volume(Program p, string inputVolume)
        {
            string path = dir + inputVolume;
            try
```

```
        {
            var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite);
            stream.Close();
        }
        catch
        {
            Console.WriteLine("Volume not found! Please mount a exisiting volume!");
            return;
        }

        Console.WriteLine("Volume Mounted!");
        bool endFlag = false;

        while (endFlag == false)
        {
            Console.WriteLine($"Please enter command for {inputVolume}:");
            string input2 = Console.ReadLine();
            string[] inputArgs2 = input2.Split(" ");
            switch (inputArgs2[0])
            {
                case "catalog"://Display files in the current mounted volume.
                    p.catalog(inputVolume);
                    break;
                case "unmount"://Unmount current volume.
                    endFlag = true;
                    break;
                case "info"://display size of specified file.
                    p.info(inputVolume, inputArgs2[1]);
                    break;
                case "create"://Create new, empty file with specified name and size.
                    p.create(inputVolume, inputArgs2[1], Convert.ToInt32(inputArgs2[2]));
                    break;
                case "write"://Write data to specified filename starting at specified offset.
                    p.write(inputVolume, inputArgs2[1], Convert.ToInt32(inputArgs2[2]), inputArgs2[3]);
                    break;
                case "read"://Reads specified file from specified start to end.
                    Console.WriteLine("Output from file: " + p.read(inputVolume, inputArgs2[1], Convert.ToInt32(inputArgs2[2]),
Convert.ToInt32(inputArgs2[3])));
                    break;
                case "delete"://Removes specified file from volume ONLY IF READONLY IS FALSE!
                    p.delete(inputVolume, inputArgs2[1]);
                    break;
                case "truncate"://Same as DELETE <FILENAME> followed by CREATE < FILENAME >
                    p.truncate(inputVolume, inputArgs2[1]);
                    break;
                case "set"://SET <FILENAME> READOLNY=TRUE|FALSE: Sets the read-only flag for FILENAME.
                    p.set(inputVolume, inputArgs2[1], inputArgs2[2]);
                    break;


            }
        }
    }

    public void allocate(string name, int size)
    {
        Console.WriteLine($"Allocating {size} bytes for '{name}'...");
        string path = dir + name;
        File.WriteAllBytes(@path, new byte[size]);
```

```
}

public void deallocate(string name)
{
    Console.WriteLine($"Deallocating '{name}'...");
    string path = dir + name;
    File.Delete(@path);
}
public void truncate(string name)
{
    Console.WriteLine($"Truncating '{name}'...");
    string path = dir + name;
    byte[] content;
    try
    {
        content = File.ReadAllBytes(@path);
    }
    catch
    {
        Console.WriteLine("VOLUME NOT FOUND");
        return;
    }
    int size = content.Length;
    File.WriteAllBytes(path, new byte[size]);
    Console.WriteLine($"Done!");
}
public void dump(string name)
{
    string path = dir + name;
    try
    {
        var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite);
        stream.Close();
    }
    catch
    {
        Console.WriteLine("Volume not found! Please mount a exisiting volume!");
        return;
    }
    Console.WriteLine($"Dumping '{name}'...");

    byte[] content;
    try
    {
        content = File.ReadAllBytes(@path);
    }
    catch
    {
        Console.WriteLine("VOLUME NOT FOUND");
        return;
    }
    foreach (byte b in content)
    {
        Console.Write("0x{0:x2}", b);
        Console.Write(" ");

    }
    Console.Write("\n");
}
```

```
public void create(string volName, string fileName, int fileSize)
{

    string path = dir + volName;
    byte[] content;
    content = File.ReadAllBytes(@path);
    bool flag = false;
    var fi = new FileInfo(path);
    long volumeSize = fi.Length;
    int projectedSize = fileName.Length + fileSize.ToString().Length + 46 + fileSize;
    Console.WriteLine($"Projected File Size: {projectedSize}");


    using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
    {
        for (int i = 0; i < content.Length; i++)
        {
            if (i + projectedSize > volumeSize)
            {
                Console.WriteLine("PROJECTED FILE SIZE LARGER THAN VOLUME OR NOT ENOUGH SPACE IN VOLUME! ABORTING CREATION");
                return;
            }
        start:
            stream.Position = i;
            if (stream.ReadByte() != 1 && flag == false)
            {

                for (int j = 0; j < projectedSize - 1; j++)
                {
                    if (stream.ReadByte() != 0)
                    {
                        i = Convert.ToInt16(stream.Position);
                        Console.WriteLine("Not enough space!");
                        goto start;
                    }
                }
                stream.Position = i;

                break;
            }
            stream.Position = i;
            if (stream.ReadByte() == 1)
            {
                flag = true;
            }
            stream.Position = i;
            if (stream.ReadByte() == 6)
            {
                flag = false;
            }
        }

        stream.WriteByte(1); //Mark beginning of file.

        //-----------------------------------------------------
        //------------WRITE FILE NAME--------------------------
        //-----------------------------------------------------
        foreach (char c in fileName)
        {
```

```
    stream.WriteByte((byte)c);

}

stream.WriteByte(2);//Mark end of name section and start of the properties section.

//------------------------------------------------------
//------------WRITE FILE SIZE---------------------------
//------------------------------------------------------
foreach (char c in fileSize.ToString())
{
    stream.WriteByte(((byte)c));
}


stream.WriteByte(1);


//------------------------------------------------------
//------------WRITE FILE CREATE DATE--------------------
//------------------------------------------------------
string date = DateTime.Now.Month.ToString();
if (date.Length < 2)//Fix date length to xx/xx/xxxx
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Day.ToString();
if (date.Length < 2)//Fix date lenght to xx/xx/xxxx
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Year.ToString();
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}

stream.WriteByte(1);

//------------------------------------------------------
//------------WRITE FILE CREATE Time--------------------
//------------------------------------------------------
string time = DateTime.Now.Hour.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
```

```
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)':');
time = DateTime.Now.Minute.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)':');
time = DateTime.Now.Second.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte(1);
//-----------------------------------------------------
//------------WRITE FILE LAST MODIFIED DATE------------------
//-----------------------------------------------------
date = DateTime.Now.Month.ToString();
if (date.Length < 2)
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Day.ToString();
if (date.Length < 2)
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Year.ToString();
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}

stream.WriteByte(1);

//-----------------------------------------------------
//------------WRITE FILE LAST MODIFIED Time------------------
//-----------------------------------------------------
time = DateTime.Now.Hour.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
```

```
        {
            time = "0" + time;
        }
        foreach (char c in time)
        {
            stream.WriteByte(((byte)c));
        }
        stream.WriteByte((byte)':');
        time = DateTime.Now.Minute.ToString();
        if (time.Length < 2)//fix time format to xx:xx:xx
        {
            time = "0" + time;
        }
        foreach (char c in time)
        {
            stream.WriteByte(((byte)c));
        }
        stream.WriteByte((byte)':');
        time = DateTime.Now.Second.ToString();
        if (time.Length < 2)//fix time format to xx:xx:xx
        {
            time = "0" + time;
        }
        foreach (char c in time)
        {
            stream.WriteByte(((byte)c));
        }
        stream.WriteByte(1);


        stream.WriteByte(4);//Mark as NOT readonly by default.

        stream.WriteByte(3);//Mark end of property section AND beginning of data section.
        while (fileSize > 0)
        {
            fileSize--;
            stream.WriteByte(0);
        }
        stream.WriteByte(6);//Mark end of file.
        stream.Close();
    }
}

public void write(string volName, string fileName, int offset, string fileData)
{
    string path = dir + volName;
    byte[] content;
    content = File.ReadAllBytes(@path);
    bool flag = false;
    int desiredFilePos = 0;
    using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
    {

        for (int i = 0; i < content.Length; i++)
        {
        start:
            if (i > content.Length)
            {
                return;
```

```
        }
        stream.Position = i;
        if (stream.ReadByte() == 1 || flag == false)
        {
            desiredFilePos = i;
            foreach (char c in fileName)
            {
                if ((byte)c == stream.ReadByte())
                {
                    continue;
                }
                else
                {
                    //desiredFilePos = 0;
                    i++;
                    goto start;
                }
            }
            if (stream.ReadByte() != 2)
            {
                i++;
                goto start;
            }
            stream.Position--;
            goto label;
        }
        stream.Position = i;
        if (stream.ReadByte() != 1)
        {
            flag = true;
        }
        stream.Position = i;
        if (stream.ReadByte() == 6)
        {
            flag = false;
        }
    }

label:
    while (stream.ReadByte() != 2)
    { }
    while (stream.ReadByte() != 1) { }
    while (stream.ReadByte() != 1) { }
    while (stream.ReadByte() != 1) { }

    //-----------------------------------------------------
    //------------WRITE FILE LAST MODIFIED DATE------------------
    //-----------------------------------------------------
    string date = DateTime.Now.Month.ToString();
    if (date.Length < 2)
    {
        date = "0" + date;
    }
    foreach (char c in date)
    {
        stream.WriteByte(((byte)c));
    }
    stream.WriteByte((byte)'/');
    date = DateTime.Now.Day.ToString();
```

```
if (date.Length < 2)
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Year.ToString();
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}

while (stream.ReadByte() != 1) { }

//-----------------------------------------------------
//-------------WRITE FILE LAST MODIFIED Time-----------------
//-----------------------------------------------------
string time = DateTime.Now.Hour.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)':');
time = DateTime.Now.Minute.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)':');
time = DateTime.Now.Second.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
    stream.WriteByte(((byte)c));
}



while (stream.ReadByte() != 3)
{ }
stream.Position += offset;
foreach (char c in fileData)
{
    if (stream.ReadByte() != 6)
    {
```

```
                stream.Position--;
                stream.WriteByte((byte)c);
            }
            else
            {
                Console.WriteLine("FILE SPACE NOT LARGE ENOUGH TO WRITE DATA TO! DATA HAS BEEN CROPPED TO FIT!");
                break;
            }
        }
        stream.Close();
        return;

    }
}

public string read(string volName, string fileName, int offsetStart, int offsetEnd)
{
    string path = dir + volName;
    byte[] content;
    content = File.ReadAllBytes(@path);
    bool flag = false;
    int desiredFilePos = 0;
    using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
    {

        for (int i = 0; i < content.Length; i++)
        {
        start:
            if (i > content.Length)
            {
                return "";
            }
            stream.Position = i;
            if (stream.ReadByte() == 1 || flag == false)
            {
                desiredFilePos = i;
                foreach (char c in fileName)
                {
                    if ((byte)c == stream.ReadByte())
                    {
                        continue;
                    }
                    else
                    {
                        //desiredFilePos = 0;
                        i++;
                        goto start;
                    }
                }
                if (stream.ReadByte() != 2)
                {
                    i++;
                    goto start;
                }
                stream.Position--;
                goto label;
            }
            stream.Position = i;
            if (stream.ReadByte() != 1)
```

```
            {
                flag = true;
            }
            stream.Position = i;
            if (stream.ReadByte() == 6)
            {
                flag = false;
            }
        }

    label:
        while (stream.ReadByte() != 3) { }
        string output = "";
        stream.Position += offsetStart;
        for (int i = 0; i < (offsetEnd - offsetStart); i++)
        {
            if (stream.ReadByte() == 6)
            {
                break;
            }
            else
            {
                stream.Position--;
                output = output + (char)stream.ReadByte();
            }

        }
        stream.Close();
        return output;
    }
}

public string info(string volName, string fileName)
{
    string path = dir + volName;
    byte[] content;
    content = File.ReadAllBytes(@path);
    bool flag = false;
    int desiredFilePos = 0;
    using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
    {

        for (int i = 0; i < content.Length; i++)
        {
        start:
            if (i > content.Length)
            {
                return "";
            }
            stream.Position = i;
            desiredFilePos = i;
            if (stream.ReadByte() == 1 || flag == false)
            {

                foreach (char c in fileName)
                {
                    if ((byte)c == stream.ReadByte())
                    {
                        continue;
```

```
                }
                else
                {
                    //desiredFilePos = 0;
                    i++;
                    goto start;
                }
            }
            if (stream.ReadByte() != 2)
            {
                i++;
                goto start;
            }
            stream.Position--;
            goto label;
        }
        stream.Position = i;
        if (stream.ReadByte() != 1)
        {
            flag = true;
        }
        stream.Position = i;
        if (stream.ReadByte() == 6)
        {
            flag = false;
        }
    }

label:
    string output = "";
    stream.Position = desiredFilePos + 1;
    while (stream.ReadByte() != 2)
    {
        stream.Position--;
        output += (char)stream.ReadByte();
    }
    Console.WriteLine($"File Name: '{output}'");
    output = "";

    //---File Size---
    while (stream.ReadByte() != 1)
    {
        stream.Position--;
        output += (char)stream.ReadByte();
    }
    Console.WriteLine($"File Size: {output} Bytes");
    output = "";
    //---Date Created---
    while (stream.ReadByte() != 1)
    {
        stream.Position--;
        output += (char)stream.ReadByte();
    }
    Console.WriteLine($"Date Created: {output}");
    output = "";

    //---Time Created---
    while (stream.ReadByte() != 1)
    {
```

```
            stream.Position--;
            output += (char)stream.ReadByte();
        }
        Console.WriteLine($"Time Created: {output}");
        output = "";

        //---Date Last Edited---
        while (stream.ReadByte() != 1)
        {
            stream.Position--;
            output += (char)stream.ReadByte();
        }
        Console.WriteLine($"Date Last Edited: {output}");
        output = "";

        //---Time Last Edited---
        while (stream.ReadByte() != 1)
        {
            stream.Position--;
            output += (char)stream.ReadByte();
        }
        Console.WriteLine($"Time Last Edited: {output}");
        output = "";

        string readOnly ="";
        if (stream.ReadByte() == 4)
        {
            readOnly = "false";
        }
        else
        {
            readOnly = "true";
        }
        Console.WriteLine($"Read-Only: {readOnly}");

        stream.Close();
        return output;
    }
}

public void set(string volName, string fileName, string fileRead)
{
    string path = dir + volName;
    byte[] content;
    content = File.ReadAllBytes(@path);
    bool flag = false;
    int desiredFilePos = 0;
    using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
    {

        for (int i = 0; i < content.Length; i++)
        {
        start:
            if (i > content.Length)
            {
                return;
            }
            stream.Position = i;
            if (stream.ReadByte() == 1 || flag == false)
```

```
            {
                desiredFilePos = i;
                foreach (char c in fileName)
                {
                    if ((byte)c == stream.ReadByte())
                    {
                        continue;
                    }
                    else
                    {
                        //desiredFilePos = 0;
                        i++;
                        goto start;
                    }
                }
                if (stream.ReadByte() != 2)
                {
                    i++;
                    goto start;
                }
                stream.Position--;
                goto label;
            }
            stream.Position = i;
            if (stream.ReadByte() != 1)
            {
                flag = true;
            }
            stream.Position = i;
            if (stream.ReadByte() == 6)
            {
                flag = false;
            }
        }

    label:
        while (stream.ReadByte() != 3) { }
        stream.Position -= 2;
        if (fileRead == "readonly=true")
        {
            stream.WriteByte(5);
        }
        else if (fileRead == "readonly=false")
        {
            stream.WriteByte(4);
        }
        else
        {
            Console.WriteLine("ERROR IN INPUT");
            return;
        }
        Console.WriteLine($"Read-Only Flag Set for {fileName}!");
        stream.Close();
    }
}

public void delete(string volName, string fileName)
{
    string path = dir + volName;
```

```
byte[] content;
content = File.ReadAllBytes(@path);
bool flag = false;
int desiredFilePos = 0;
using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
{

    for (int i = 0; i < content.Length; i++)
    {
    start:
        if (i > content.Length)
        {
            return;
        }
        stream.Position = i;
        desiredFilePos = i;
        if (stream.ReadByte() == 1 || flag == false)
        {

            foreach (char c in fileName)
            {
                if ((byte)c == stream.ReadByte())
                {
                    continue;
                }
                else
                {
                    //desiredFilePos = 0;
                    i++;
                    goto start;
                }
            }
            if (stream.ReadByte() != 2)
            {
                i++;
                goto start;
            }
            stream.Position--;
            goto label;
        }
        stream.Position = i;
        if (stream.ReadByte() != 1)
        {
            flag = true;
        }
        stream.Position = i;
        if (stream.ReadByte() == 6)
        {
            flag = false;
        }
    }

label:
    while (stream.ReadByte() != 3)
    { }
    stream.Position -= 2;

    if (stream.ReadByte() == 4)
    {
```

```
            while (stream.ReadByte() != 2)//Back up to beginning of file.
            {
                stream.Position -= 2;
            }
            while (stream.ReadByte() != 1)//Back up to beginning of file.
            {
                stream.Position -= 2;
            }
            stream.Position--;
            while (stream.ReadByte() != 6)
            {
                stream.Position--;
                stream.WriteByte(0);
            }
            stream.Position--;
            stream.WriteByte(0);
            stream.Close();
        }
        else
        {
            Console.WriteLine("FILE IS SET READONLY! CANNOT DELETE!");
            stream.Close();
            return;
        }

    }
}


public void truncate(string volName, string fileName)
{
    string path = dir + volName;
    byte[] content;
    content = File.ReadAllBytes(@path);
    bool flag = false;
    int desiredFilePos = 0;
    using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
    {

        for (int i = 0; i < content.Length; i++)
        {
        start:
            if (i > content.Length)
            {
                return;
            }
            stream.Position = i;
            desiredFilePos = i;
            if (stream.ReadByte() == 1 || flag == false)
            {

                foreach (char c in fileName)
                {
                    if ((byte)c == stream.ReadByte())
                    {
                        continue;
                    }
                    else
                    {
```

```
                    //desiredFilePos = 0;
                    i++;
                    goto start;
                }
            }
            if (stream.ReadByte() != 2)
            {
                i++;
                goto start;
            }
            stream.Position--;
            goto label;
        }
        stream.Position = i;
        if (stream.ReadByte() != 1)
        {
            flag = true;
        }
        stream.Position = i;
        if (stream.ReadByte() == 6)
        {
            flag = false;
        }
    }

label:
    while (stream.ReadByte() != 2)//Back up to size field
    {

    }


    string strFileSize = "";

    while (stream.ReadByte() != 1)
    {
        stream.Position--;
        strFileSize += (char)stream.ReadByte();
    }



    int fileSize = Convert.ToInt32(strFileSize);

    while (stream.ReadByte() != 1)//Back up to beginning of file.
    {
        stream.Position -= 2;
    }
    //stream.Position--; Keep the 1 at the beginning!
    while (stream.ReadByte() != 6)
    {
        stream.Position--;
        stream.WriteByte(0);
    }



    stream.Position = desiredFilePos + 1;
```

```
//-------------------------------------------------------
//------------WRITE FILE NAME----------------------------
//-------------------------------------------------------
foreach (char c in fileName)
{
    stream.WriteByte((byte)c);

}

stream.WriteByte(2);//Mark end of name section and start of the properties section.

//-------------------------------------------------------
//------------WRITE FILE SIZE----------------------------
//-------------------------------------------------------
foreach (char c in fileSize.ToString())
{
    stream.WriteByte(((byte)c));
}


stream.WriteByte(1);


//-------------------------------------------------------
//------------WRITE FILE CREATE DATE---------------------
//-------------------------------------------------------
string date = DateTime.Now.Month.ToString();
if (date.Length < 2)//Fix date lenght to xx/xx/xxxx
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Day.ToString();
if (date.Length < 2)//Fix date lenght to xx/xx/xxxx
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Year.ToString();
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}

stream.WriteByte(1);

//-------------------------------------------------------
//------------WRITE FILE CREATE Time---------------------
//-------------------------------------------------------
string time = DateTime.Now.Hour.ToString();
```

```
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)':');
time = DateTime.Now.Minute.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)':');
time = DateTime.Now.Second.ToString();
if (time.Length < 2)//fix time format to xx:xx:xx
{
    time = "0" + time;
}
foreach (char c in time)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte(1);
//-------------------------------------------------
//------------WRITE FILE LAST MODIFIED DATE----------------
//-------------------------------------------------
date = DateTime.Now.Month.ToString();
if (date.Length < 2)
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Day.ToString();
if (date.Length < 2)
{
    date = "0" + date;
}
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}
stream.WriteByte((byte)'/');
date = DateTime.Now.Year.ToString();
foreach (char c in date)
{
    stream.WriteByte(((byte)c));
}

stream.WriteByte(1);
```

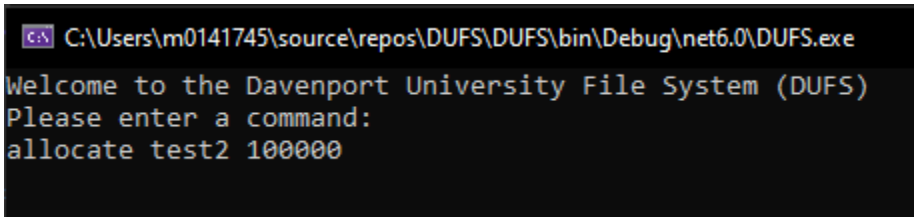
```
        //-------------------------------------------------------
        //-------------WRITE FILE LAST MODIFIED Time-------------------
        //-------------------------------------------------------
        time = DateTime.Now.Hour.ToString();
        if (time.Length < 2)//fix time format to xx:xx:xx
        {
            time = "0" + time;
        }
        foreach (char c in time)
        {
            stream.WriteByte(((byte)c));
        }
        stream.WriteByte((byte)':');
        time = DateTime.Now.Minute.ToString();
        if (time.Length < 2)//fix time format to xx:xx:xx
        {
            time = "0" + time;
        }
        foreach (char c in time)
        {
            stream.WriteByte(((byte)c));
        }
        stream.WriteByte((byte)':');
        time = DateTime.Now.Second.ToString();
        if (time.Length < 2)//fix time format to xx:xx:xx
        {
            time = "0" + time;
        }
        foreach (char c in time)
        {
            stream.WriteByte(((byte)c));
        }
        stream.WriteByte(1);


        stream.WriteByte(4);//Mark as NOT readonly by default.

        stream.WriteByte(3);//Mark end of property section AND beginning of data section.
        while (fileSize > 0)
        {
            fileSize--;
            stream.WriteByte(0);
        }
        stream.WriteByte(6);//Mark end of file.
        stream.Close();
    }
}


public void catalog(string volName)
{
    string path = dir + volName;
    byte[] content;
    content = File.ReadAllBytes(@path);
    bool flag = false;
    int desiredFilePos = 0;

    Console.WriteLine($"Files in {volName}:");
```

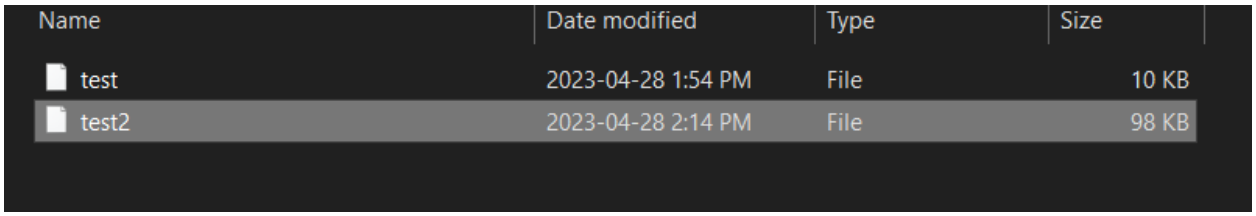
```
    using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
    {
        for (int i = 0; i < content.Length; i++)
        {
            stream.Position = i;
            if (stream.ReadByte() == 1 && flag == false)
            {
                while (stream.ReadByte() != 2)
                {
                    stream.Position--;
                    Console.Write((char)stream.ReadByte());
                }
                Console.Write("\n");
            }

            stream.Position = i;
            if (stream.ReadByte() != 1)
            {
                flag = true;
            }

            stream.Position = i;
            if (stream.ReadByte() == 6)
            {
                flag = false;
            }

        }
    }
}

public void volInfo(string volName)
{
    string path = dir + volName;
    byte[] content;
    content = File.ReadAllBytes(@path);
    bool flag = false;
    int freeSpace = 0;
    int fileCount = 0;

    var fi = new FileInfo(path);
    long volSize = fi.Length;
    Console.WriteLine($"Volume Size: {volSize} Bytes");
    using (var stream = new FileStream(path, FileMode.Open, FileAccess.ReadWrite))
    {
        for (int i = 0; i < content.Length; i++)
        {
            stream.Position = i;
            if (stream.ReadByte() == 1 && flag == false)
            {
                fileCount++;
                continue;
            }

            stream.Position = i;
            if (stream.ReadByte() == 0 && flag == false)
            {
                freeSpace++;
                continue;
```

```
                    }

                    stream.Position = i;
                    if (stream.ReadByte() == 6)
                    {
                        flag = false;
                        continue;
                    }

                    stream.Position = i;
                    if (stream.ReadByte() != 1)
                    {
                        flag = true;
                        continue;
                    }


                }
                stream.Close();
            }

            Console.WriteLine($"Free Space Available in '{volName}': {freeSpace} Bytes");
            Console.WriteLine($"Number of files in '{volName}': {fileCount}");
        }
    }

}
```

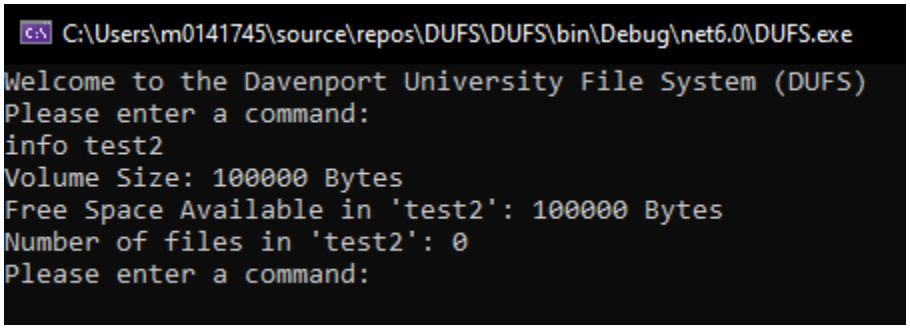
Achieved:

Allocate -

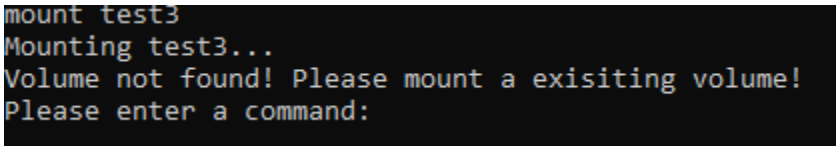


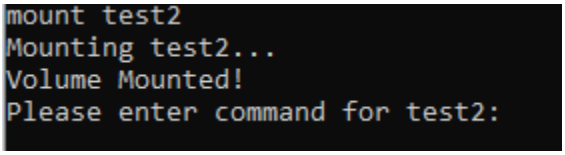


Volume Info -



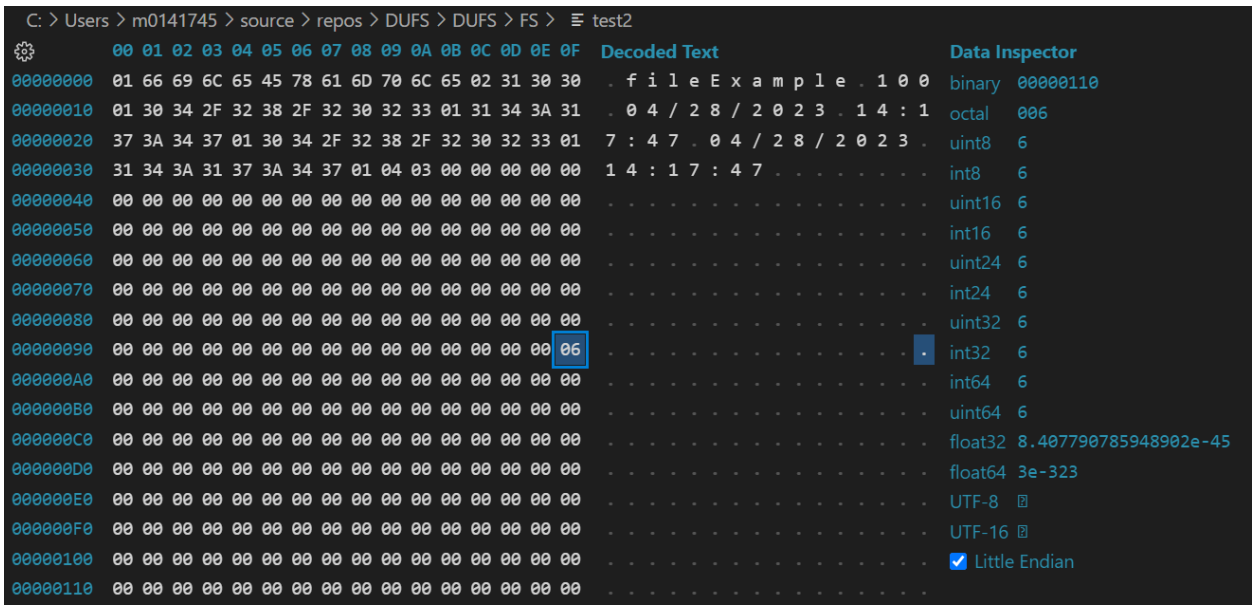
Mount -





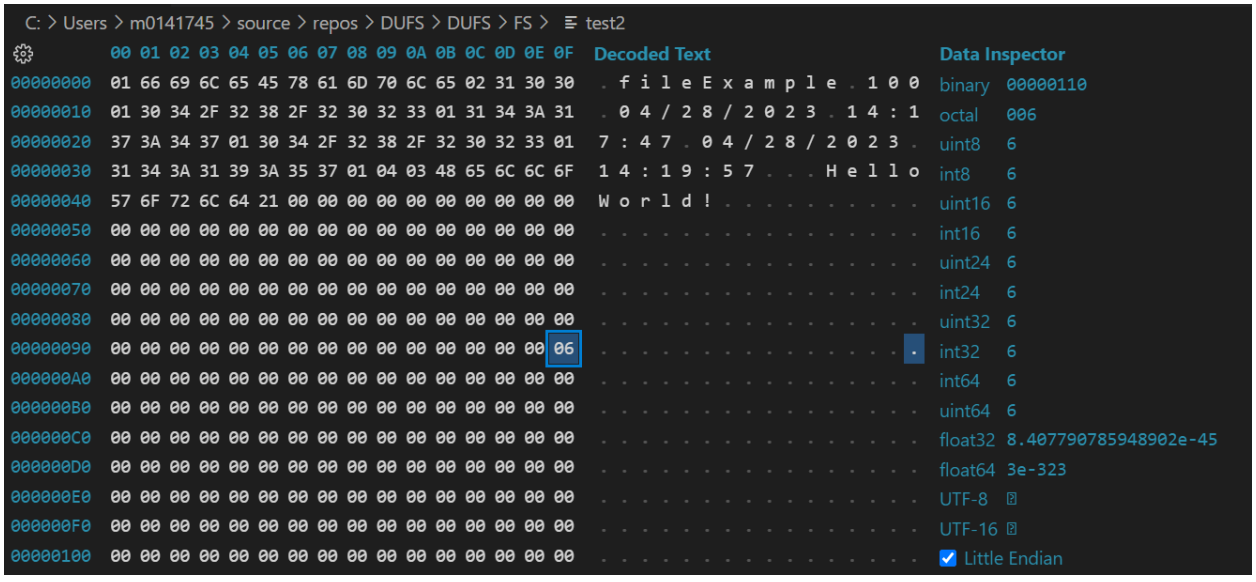
Create -

```
create fileExample 100
Projected File Size: 160
Please enter command for test2:
```



Write -

```
Please enter command for test2:
write fileExample 0 HelloWorld!
Please enter command for test2:
```



Read -

```
Please enter command for test2:
read fileExample 0 10
Output from file: HelloWorld
Please enter command for test2:
read fileExample 0 11
Output from file: HelloWorld!
Please enter command for test2:
read fileExample 0 5
Output from file: Hello
Please enter command for test2:
read fileExample 1 8
Output from file: elloWor
Please enter command for test2:
```

Info -

```
info fileExample
File Name: 'fileExample'
File Size: 100 Bytes
Date Created: 04/28/2023
Time Created: 14:17:47
Date Last Edited: 04/28/2023
Time Last Edited: 14:19:57
Read-Only: false
Please enter command for test2:
```

Set -

```
set fileExample readonly=true
Read-Only Flag Set for fileExample!
Please enter command for test2:
info fileExample
File Name: 'fileExample'
File Size: 100 Bytes
Date Created: 04/28/2023
Time Created: 14:17:47
Date Last Edited: 04/28/2023
Time Last Edited: 14:19:57
Read-Only: true
Please enter command for test2:
```

Catalog -

```
create fileExample2 250
Projected File Size: 311
Please enter command for test2:
catalog
Files in test2:
fileExample
fileExample2
Please enter command for test2:
```

Delete -

C: > Users > m0141745 > source > repos > DUFS > DUFS > FS > test2

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded Text
00000000  01 66 69 6C 65 45 78 61 6D 70 6C 65 02 31 30 30  .fileExample.100
00000010  01 30 34 2F 32 38 2F 32 30 32 33 01 31 34 3A 31  .04/28/2023.14:1
00000020  37 3A 34 37 01 30 34 2F 32 38 2F 32 30 32 33 01  7:47.04/28/2023.
00000030  31 34 3A 31 39 3A 35 37 01 05 03 48 65 6C 6C 6F  14:19:57...Hello
00000040  57 6F 72 6C 64 21 00 00 00 00 00 00 00 00 00 00  World!..........
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06  ................
000000A0  01 66 69 6C 65 45 78 61 6D 70 6C 65 32 02 32 35  .fileExample2.25
000000B0  30 01 30 34 2F 32 38 2F 32 30 32 33 01 31 34 3A  0.04/28/2023.14:
000000C0  32 33 3A 34 35 01 30 34 2F 32 38 2F 32 30 32 33  23:45.04/28/2023
000000D0  01 31 34 3A 32 33 3A 34 35 01 04 03 00 00 00 00  .14:23:45.......
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001D0  00 00 00 00 00 00 06 00 00 00 00 00 00 00 00 00  ................
```

```
delete fileExample
FILE IS SET READONLY! CANNOT DELETE!
Please enter command for test2:
set fileExample readonly=false
Read-Only Flag Set for fileExample!
Please enter command for test2:
delete fileExample
Please enter command for test2:
```

C: > Users > m0141745 > source > repos > DUFS > DUFS > FS > test2

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded Text
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0  01 66 69 6C 65 45 78 61 6D 70 6C 65 32 02 32 35  .fileExample2.25
000000B0  30 01 30 34 2F 32 38 2F 32 30 32 33 01 31 34 3A  0.04/28/2023.14:
000000C0  32 33 3A 34 35 01 30 34 2F 32 38 2F 32 30 32 33  23:45.04/28/2023
000000D0  01 31 34 3A 32 33 3A 34 35 01 04 03 00 00 00 00  .14:23:45.......
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001D0  00 00 00 00 00 00 06 00 00 00 00 00 00 00 00 00  ................
```

Truncate -

```
truncate fileExample2
Please enter command for test2:
```

```
000000A0  01 66 69 6C 65 45 78 61 6D 70 6C 65 32 02 32 35  .fileExample2.25
000000B0  30 01 30 34 2F 32 38 2F 32 30 32 33 01 31 34 3A  0.04/28/2023.14:
000000C0  32 37 3A 30 32 01 30 34 2F 32 38 2F 32 30 32 33  27:02.04/28/2023
000000D0  01 31 34 3A 32 37 3A 30 32 01 04 03 00 00 00 00  .14:27:02.......
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001D0  00 00 00 00 00 00 06 00 00 00 00 00 00 00 00 00  ................
```

Unmount -

```
unmount
Please enter a command:
```

Volume info -

```
info test2
Volume Size: 100000 Bytes
Free Space Available in 'test2': 99689 Bytes
Number of files in 'test2': 1
Please enter a command:
```

Volume dump -

```
dump test2
Dumping 'test2'...
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x66 0x69 0x6c 0x65 0x45 0x78 0x61
0x6d 0x70 0x6c 0x65 0x32 0x02 0x32 0x35 0x30 0x01 0x30 0x34 0x2f 0x32 0x38 0x2f 0x32 0x30 0x32 0x33 0x01 0x31 0x34 0x3a
0x32 0x37 0x3a 0x30 0x32 0x01 0x30 0x34 0x2f 0x32 0x38 0x2f 0x32 0x30 0x32 0x33 0x01 0x31 0x34 0x3a 0x32 0x37 0x3a 0x30
0x32 0x01 0x04 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

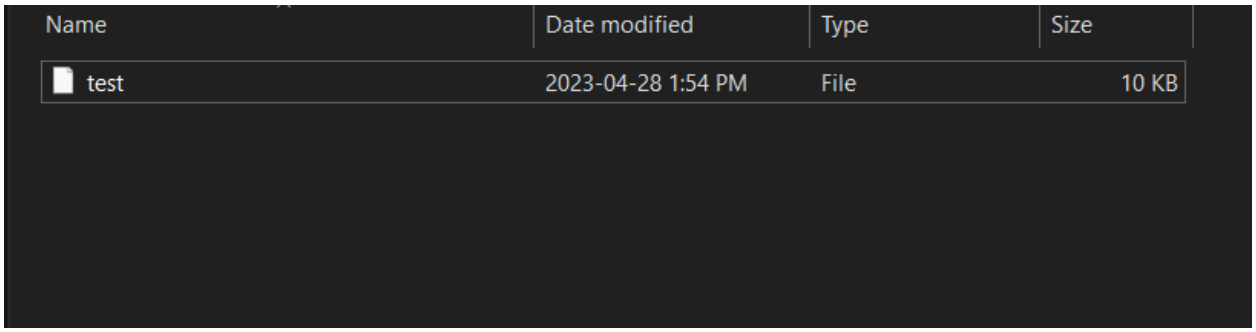
Volume truncate -

```
truncate test2
Truncating 'test2'...
Done!
Please enter a command:
```

```
C: > Users > m0141745 > source > repos > DUFS > DUFS > FS > test2
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded Text
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  . . . . . . . . . . . . . . . .
```

Volume delete -

```
deallocate test2
Deallocating 'test2'...
Please enter a command:
```



As seen from the process above, the file system successfully runs all of our commands. We are able to create volumes, mount them, create files in these volumes, write to them, read them, display info about them, delete them, truncate them, unmount the volume, truncate the volume, and delete it.

While this file system is very basic, it meets our design goals by allowing users to manage files. And write basic strings of text to them.

Bugs:

Currently, there are a couple of bugs. Firstly, you cannot write data to a file if it contains a space. This is due to how arguments are parsed in the command line. A small bit of logic can be added so that all arguments after the text argument in write() is concatenated to that argument in the inputArgs2[] array.

The other major bug in the program comes down to incorrect commands crashing the program. If a command is issued to the program with bad syntax or is missing arguments, this may cause the program to crash. The use of try-catch statements is used to catch some of them, but not all are caught. Inclusion of these as crash avoidance would be top priority for the following version of DUFS.

This Davenport University File System (or DUFS) is a student made project made to explore and emulate the abilities of a file system structure on an OS. The project was developed, designed, and written by Kaden VanPortfleet. Please do your best to break this program in any way not already described. Thank you.