**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

**Лабораторна робота №4**

# «Файлова система (частина 1)»

з дисципліни  
«Операційні системи»

Виконала:

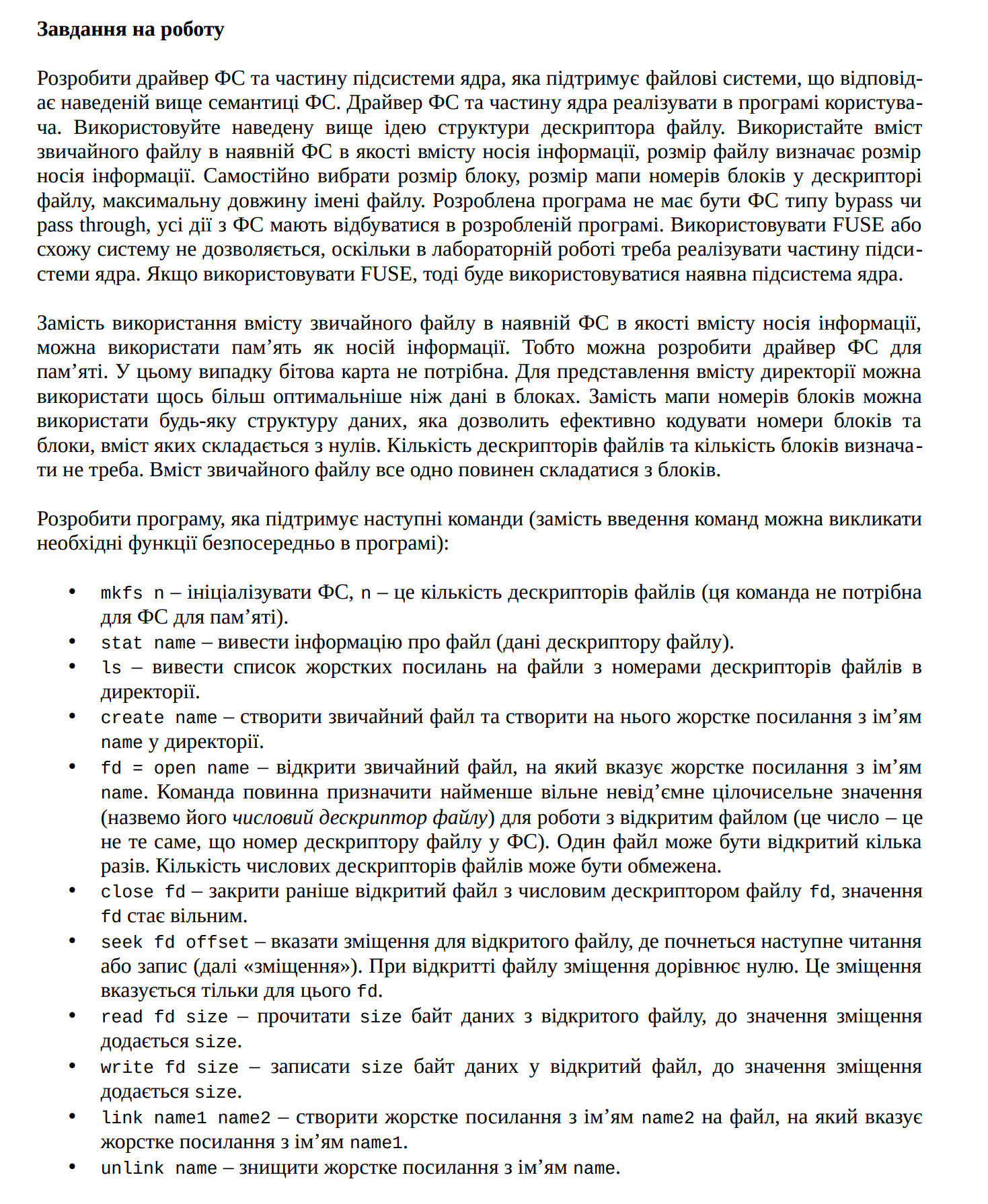
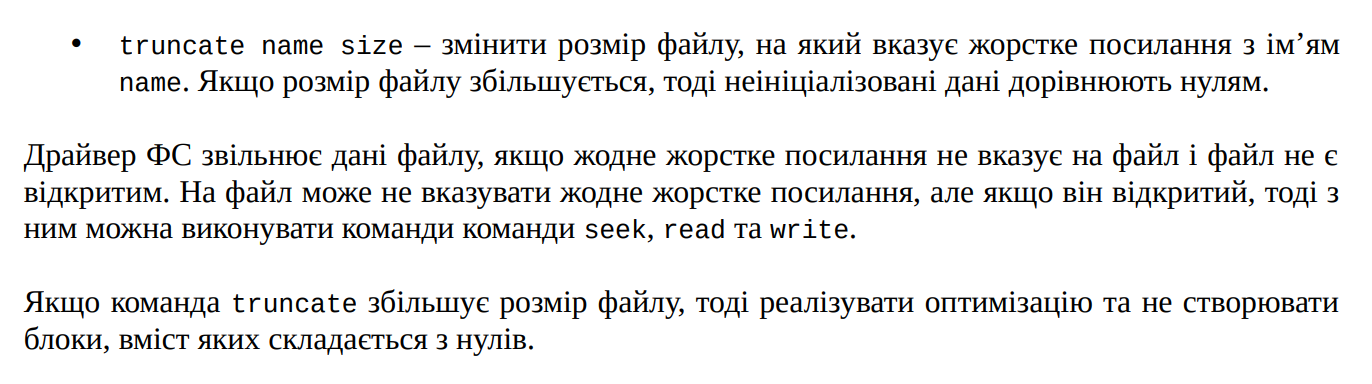
студентка групи ІМ-11   
Бащак Ярина Володимирівна

Перевірив:

ст. вик. Сімоненко А.В.

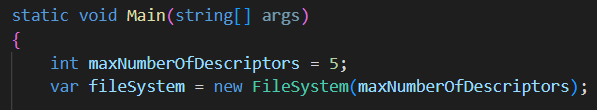
Київ 2023

**Завдання.**

**Приклади виконання всіх команд.**

* ініціалізація ФС. За умовою є одна директорія, тому при створенні ФС також створюється дескриптор цієї директорії.





* create name, ls. Більше 5 дескрипторів не доступно

fileSystem.Create("file1.txt");

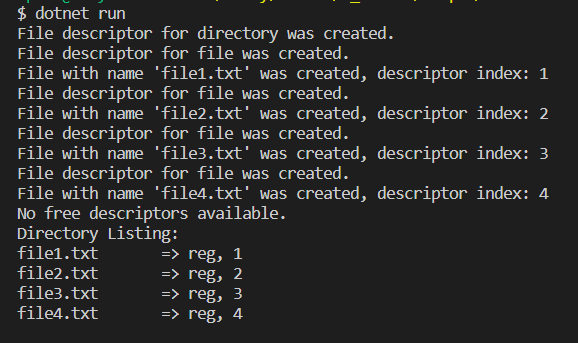
        fileSystem.Create("file2.txt");

        fileSystem.Create("file3.txt");

        fileSystem.Create("file4.txt");

        fileSystem.Create("file5.txt");

        fileSystem.Ls();



* open, close. Більше 3 одночасно відкритих файлів не доступно

        var fd1 = fileSystem.Open("file2.txt");

        var fd2 = fileSystem.Open("nonExistedFile.txt");

        var fd3 = fileSystem.Open("file1.txt");

        var fd4 = fileSystem.Open("file1.txt");

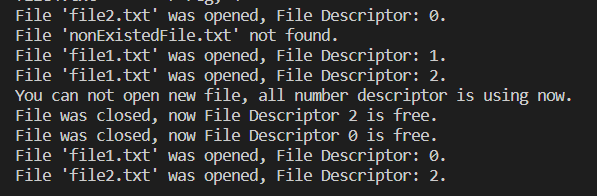
        var fd5 = fileSystem.Open("file1.txt");

        fileSystem.Close(fd4);

        fileSystem.Close(fd1);

        var fd6 = fileSystem.Open("file1.txt");

        var fd7 = fileSystem.Open("file2.txt");



* truncate, write, seek, read

        fileSystem.Truncate("file2.txt", 124);

        byte[] byteArray = new byte[60];

        for (int i = 0; i < byteArray.Length; i++)

        {

            byteArray[i] = (byte)i;

        }

        fileSystem.Write(fd1, byteArray);

        fileSystem.Seek(fd1, 20);

        var data = fileSystem.Read(fd1, 300);

        Console.WriteLine("Byte Array:");

        foreach (var item in data)

        {

            if (item == null)

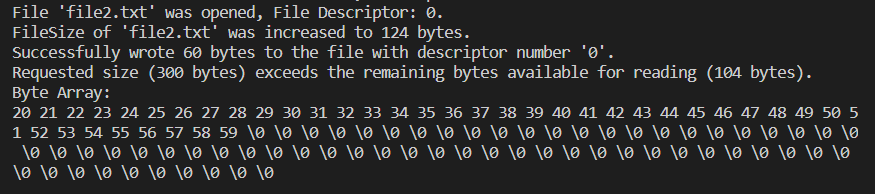
                Console.Write("\\0 ");

            else

                Console.Write(item + " ");

        }

        Console.WriteLine();



* stat, trancate. Розмір файлу збільшили до 128 байтів (trancate). Розмір одного блоку: 64 байти. У файл починаючи з 64-го індексу (seek) записали 20 байтів (write), тобто мав зайнятись лише один блок (stat). Потім скоротили файл до розміру 60 байтів (trancate), які є нульовими, тому зайнятих блоків стало 0 (stat).

        fileSystem.Truncate("file2.txt", 128);

        fileSystem.Seek(fd1, 64);

        byte[] byteArray = new byte[20];

        for (int i = 0; i < byteArray.Length; i++)

        {

            byteArray[i] = (byte)i;

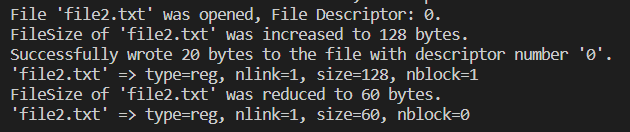
        }

        fileSystem.Write(fd1, byteArray);

        fileSystem.Stat("file2.txt");

        fileSystem.Truncate("file2.txt", 60);

        fileSystem.Stat("file2.txt");



* link, unlink. Якщо жодного жорсткого посилання не вказує на файл, але він відкритий, то звільнення не відбувається. У файл все ще можна, наприклад, щось записати. І тільки після закриття файлу, звільняються блоки, які займав цей файл.

        fileSystem.Link("file2.txt", "file3.txt");

        fileSystem.Stat("file2.txt");

        var fd1 = fileSystem.Open("file2.txt");

        fileSystem.Unlink("file2.txt");

        fileSystem.Unlink("file3.txt");

        byte[] byteArray = new byte[20];

        for (int i = 0; i < byteArray.Length; i++)

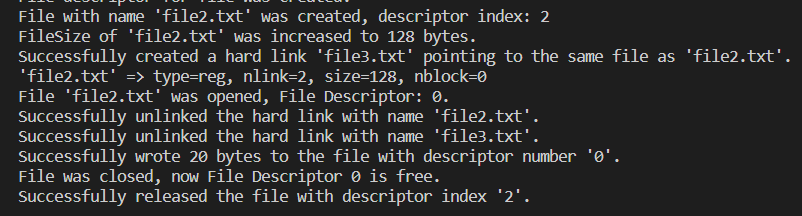
        {

            byteArray[i] = (byte)i;

        }

        fileSystem.Write(fd1, byteArray);

        fileSystem.Close(fd1);



**Лістинг.**

Я використовувала мову C# (.NET). Повний код класів і методів наведений нижче або за посиланням на [github](https://github.com/yaryna-bashchak/Operating-systems-labs/tree/main/lab4).

Коди класів, які реалізують потрібну поведінку:

public class FileSystem

{

    public bool[] Bitmap { get; set; }

    public List<FileDescriptor?> Descriptors { get; set; }

    private List<int> FreeFileDescriptorNumbers { get; set; } = new List<int>();

    private List<OpenedFile> OpenedFiles { get; set; } = new List<OpenedFile>();

    private int BlockSize { get; set; } = 64;

    public Directory Directory { get; set; }

    public FileSystem(int maxNumberOfDescriptors)

    {

        int numberOfBlocks = 20;

        int maxNumberOfNumberDescriptors = 3;

        Bitmap = new bool[numberOfBlocks];

        Descriptors = new List<FileDescriptor?>(maxNumberOfDescriptors);

        Directory = new();

        for (int i = 0; i < maxNumberOfDescriptors; i++)

        {

            Descriptors.Add(null);

        }

        for (int i = 0; i < maxNumberOfNumberDescriptors; i++)

        {

            FreeFileDescriptorNumbers.Add(i);

        }

        int rootDescriptorIndex = FindFreeDescriptorIndex();

        if (rootDescriptorIndex != -1)

        {

            Descriptors[rootDescriptorIndex] = new FileDescriptor(false);

        }

    }

    public bool Create(string fileName)

    {

        if (Directory.Contains(fileName))

        {

            Console.WriteLine($"File '{fileName}' already exists in the directory.");

            return false;

        }

        int descriptorIndex = FindFreeDescriptorIndex();

        if (descriptorIndex == -1)

        {

            Console.WriteLine("No free descriptors available.");

            return false;

        }

        var fileDescriptor = new FileDescriptor();

        Descriptors[descriptorIndex] = fileDescriptor;

        Directory.AddEntry(fileName, descriptorIndex);

        return true;

    }

    public void Stat(string filename)

    {

        int descriptorIndex = Directory.FindDescriptorIndex(filename);

        if (descriptorIndex != -1)

        {

            FileDescriptor fileDescriptor = Descriptors[descriptorIndex]!;

            Console.Write($"'{filename}' =>");

            Console.Write($" type={(fileDescriptor.IsRegularFile ? "reg" : "dir")}");

            Console.Write($", nlink={fileDescriptor.HardLinkCount}");

            Console.Write($", size={fileDescriptor.FileSize}");

            Console.WriteLine($", nblock={fileDescriptor.BlockMap.Count}");

        }

        else

        {

            Console.WriteLine($"File '{filename}' not found.");

        }

    }

    public void Ls()

    {

        Directory.Ls();

    }

    public int Open(string filename)

    {

        int descriptorIndex = Directory.FindDescriptorIndex(filename);

        if (descriptorIndex != -1)

        {

            FileDescriptor fileDescriptor = Descriptors[descriptorIndex]!;

            int fileDescriptorNumber;

            if (FreeFileDescriptorNumbers.Count != 0)

            {

                fileDescriptorNumber = FreeFileDescriptorNumbers.Min();

                FreeFileDescriptorNumbers.Remove(fileDescriptorNumber);

            }

            else

            {

                Console.WriteLine($"You can not open new file, all number descriptor is using now.");

                return -1;

            }

            OpenedFiles.Add(new OpenedFile

            {

                FileDescriptorNumber = fileDescriptorNumber,

                DescriptorIndex = descriptorIndex,

                Position = 0,

            });

            Console.WriteLine($"File '{filename}' was opened, File Descriptor: {fileDescriptorNumber}.");

            return fileDescriptorNumber;

        }

        else

        {

            Console.WriteLine($"File '{filename}' not found.");

            return -1;

        }

    }

    public void Close(int fileDescriptorNumber)

    {

        var openedFile = OpenedFiles.FirstOrDefault(f => f.FileDescriptorNumber == fileDescriptorNumber);

        if (openedFile != null)

        {

            FreeFileDescriptorNumbers.Add(fileDescriptorNumber);

            OpenedFiles.Remove(openedFile);

            Console.WriteLine($"File was closed, now File Descriptor {fileDescriptorNumber} is free.");

            var fileDescriptorIndex = openedFile.DescriptorIndex;

            if (Descriptors[fileDescriptorIndex]!.HardLinkCount == 0 && !IsFileOpened(fileDescriptorIndex))

                FreeFile(fileDescriptorIndex);

        }

        else

        {

            Console.WriteLine($"File with descriptor number '{fileDescriptorNumber}' not found or already closed.");

        }

    }

    public void Seek(int fileDescriptorNumber, int offset)

    {

        var openedFile = OpenedFiles.FirstOrDefault(f => f.FileDescriptorNumber == fileDescriptorNumber);

        if (openedFile != null)

        {

            FileDescriptor fileDescriptor = Descriptors[openedFile.DescriptorIndex]!;

            if (offset >= 0 && offset < fileDescriptor.FileSize)

            {

                openedFile.Position = offset;

            }

            else

            {

                Console.WriteLine($"Invalid offset for file with descriptor number '{fileDescriptorNumber}'. Offset must be between 0 and file size.");

            }

        }

        else

        {

            Console.WriteLine($"File with descriptor number '{fileDescriptorNumber}' not found or not opened.");

        }

    }

    public byte?[] Read(int fileDescriptorNumber, int size)

    {

        var openedFile = OpenedFiles.FirstOrDefault(f => f.FileDescriptorNumber == fileDescriptorNumber);

        if (openedFile != null)

        {

            FileDescriptor fileDescriptor = Descriptors[openedFile.DescriptorIndex]!;

            int remainingBytes = fileDescriptor.FileSize - openedFile.Position;

            if (size > remainingBytes)

            {

                Console.WriteLine($"Requested size ({size} bytes) exceeds the remaining bytes available for reading ({remainingBytes} bytes).");

                size = remainingBytes;

            }

            byte?[] data = new byte?[size];

            for (int i = 0; i < size; i++)

            {

                int blockIndex = openedFile.Position / BlockSize;

                int blockOffset = openedFile.Position % BlockSize;

                data[i] = fileDescriptor.FileData[blockIndex \* BlockSize + blockOffset];

                openedFile.Position++;

            }

            return data;

        }

        else

        {

            Console.WriteLine($"File with descriptor number '{fileDescriptorNumber}' not found or not opened.");

            return Array.Empty<byte?>();

        }

    }

    public void Write(int fileDescriptorNumber, byte[] data)

    {

        var openedFile = OpenedFiles.FirstOrDefault(f => f.FileDescriptorNumber == fileDescriptorNumber);

        if (openedFile != null)

        {

            FileDescriptor fileDescriptor = Descriptors[openedFile.DescriptorIndex]!;

            int remainingBytes = fileDescriptor.FileSize - openedFile.Position;

            if (data.Length > remainingBytes)

            {

                Console.WriteLine($"Not enough space available for writing {data.Length} bytes.");

                return;

            }

            for (int i = 0; i < data.Length; i++)

            {

                int blockIndex = openedFile.Position / BlockSize;

                int blockOffset = openedFile.Position % BlockSize;

                int byteIndex = blockIndex \* BlockSize + blockOffset;

                if (IsBlockConsistsOnlyOfNulls(fileDescriptor.FileData, blockIndex))

                {

                    fileDescriptor.BlockMap.Add(FindFreeBlockIndex());

                }

                fileDescriptor.FileData[byteIndex] = data[i];

                openedFile.Position++;

            }

            Console.WriteLine($"Successfully wrote {data.Length} bytes to the file with descriptor number '{fileDescriptorNumber}'.");

        }

        else

        {

            Console.WriteLine($"File with descriptor number '{fileDescriptorNumber}' not found or not opened.");

        }

    }

    public void Link(string name1, string name2)

    {

        int fileDescriptorIndex1 = Directory.Entries?.FirstOrDefault(entry => entry.FileName == name1)?.FileDescriptorIndex ?? -1;

        if (fileDescriptorIndex1 != -1)

        {

            Directory.Entries?.Add(new DirectoryEntry(name2, fileDescriptorIndex1));

            Descriptors[fileDescriptorIndex1]!.HardLinkCount++;

            Console.WriteLine($"Successfully created a hard link '{name2}' pointing to the same file as '{name1}'.");

        }

        else

        {

            Console.WriteLine($"File with name '{name1}' not found in the directory.");

        }

    }

    public void Unlink(string name)

    {

        var directoryEntry = Directory.Entries?.FirstOrDefault(entry => entry.FileName == name);

        if (directoryEntry != null)

        {

            int fileDescriptorIndex = directoryEntry.FileDescriptorIndex;

            Descriptors[fileDescriptorIndex]!.HardLinkCount--;

            Console.WriteLine($"Successfully unlinked the hard link with name '{name}'.");

            Directory.Entries?.Remove(directoryEntry);

            if (Descriptors[fileDescriptorIndex]!.HardLinkCount == 0 && !IsFileOpened(fileDescriptorIndex))

                FreeFile(fileDescriptorIndex);

        }

        else

        {

            Console.WriteLine($"Hard link with name '{name}' not found in the directory.");

        }

    }

    public void Truncate(string filename, int newSize)

    {

        int descriptorIndex = Directory.FindDescriptorIndex(filename);

        if (descriptorIndex != -1)

        {

            FileDescriptor fileDescriptor = Descriptors[descriptorIndex]!;

            var isFileReduced = newSize < fileDescriptor.FileSize;

            if (isFileReduced)

            {

                int oldMaxIndexOfBlock = fileDescriptor.FileSize / BlockSize;

                int newMaxIndexOfBlock = newSize / BlockSize;

                int numberOfReleasedBlocks = oldMaxIndexOfBlock - newMaxIndexOfBlock;

                for (int i = 0; i < numberOfReleasedBlocks; i++)

                {

                    if (!IsBlockConsistsOnlyOfNulls(fileDescriptor.FileData, oldMaxIndexOfBlock - i))

                        fileDescriptor.BlockMap.RemoveAt(0);

                }

                fileDescriptor.FileData = fileDescriptor.FileData.Take(newSize).ToList();

            }

            else

            {

                fileDescriptor.FileData.Capacity = newSize;

                for (int i = fileDescriptor.FileData.Count; i < newSize; i++)

                {

                    fileDescriptor.FileData.Add(null);

                }

            }

            fileDescriptor.FileSize = newSize;

            Console.WriteLine($"FileSize of '{filename}' was {(isFileReduced ? "reduced" : "increased")} to {newSize} bytes.");

        }

        else

        {

            Console.WriteLine($"File '{filename}' not found.");

        }

    }

    private bool IsBlockConsistsOnlyOfNulls(List<byte?> data, int blockIndex)

    {

        int startIndex = blockIndex \* BlockSize;

        int endIndex = startIndex + BlockSize - 1;

        if (endIndex > data.Count - 1)

            endIndex = data.Count - 1;

        for (int i = startIndex; i <= endIndex; i++)

        {

            if (data[i] != null)

            {

                return false;

            }

        }

        return true;

    }

    private int FindFreeBlockIndex()

    {

        for (int i = 0; i < Bitmap.Length; i++)

        {

            if (!Bitmap[i])

            {

                Bitmap[i] = true;

                return i;

            }

        }

        return -1;

    }

    private int FindFreeDescriptorIndex()

    {

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

        {

            if (Descriptors[i] == null)

            {

                return i;

            }

        }

        return -1;

    }

    private void FreeFile(int fileDescriptorIndex)

    {

        if (fileDescriptorIndex >= 0 && fileDescriptorIndex < Descriptors.Count)

        {

            foreach (int blockIndex in Descriptors[fileDescriptorIndex]!.BlockMap)

            {

                Bitmap[blockIndex] = false;

            }

            Descriptors[fileDescriptorIndex] = null;

            Console.WriteLine($"Successfully released the file with descriptor index '{fileDescriptorIndex}'.");

        }

        else

        {

            Console.WriteLine($"Invalid file descriptor index '{fileDescriptorIndex}'.");

        }

    }

    private bool IsFileOpened(int fileDescriptorIndex)

    {

        return OpenedFiles.Any(openedFile => openedFile.DescriptorIndex == fileDescriptorIndex);

    }

}

public class FileDescriptor

{

    public bool IsRegularFile { get; set; }

    public int HardLinkCount { get; set; } = 1;

    public int FileSize { get; set; } = 0;

    public List<byte?> FileData { get; set; } = new List<byte?>();

    public List<int> BlockMap { get; set; } = new List<int>();

    public FileDescriptor(bool isRegularFile = true)

    {

        IsRegularFile = isRegularFile;

        Console.WriteLine($"File descriptor for {(IsRegularFile ? "file" : "directory")} was created.");

    }

}

public class Directory

{

    public List<DirectoryEntry> Entries { get; } = new List<DirectoryEntry>();

    public bool Contains(string fileName)

    {

        return Entries.Any(entry => entry.FileName == fileName);

    }

    public void AddEntry(string fileName, int fileDescriptorIndex)

    {

        Entries.Add(new DirectoryEntry(fileName, fileDescriptorIndex));

        Console.WriteLine($"File with name '{fileName}' was created, descriptor index: {fileDescriptorIndex}");

    }

    public int FindDescriptorIndex(string filename)

    {

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

        {

            if (Entries[i].FileName == filename)

            {

                return Entries[i].FileDescriptorIndex;

            }

        }

        return -1;

    }

    public void Ls()

    {

        Console.WriteLine("Directory Listing:");

        foreach (var entry in Entries)

        {

            Console.WriteLine($"{entry.FileName}\t=> reg, {entry.FileDescriptorIndex}");

        }

    }

}

public class DirectoryEntry

{

    public string FileName { get; set; }

    public int FileDescriptorIndex { get; set; }

    public DirectoryEntry(string fileName, int fileDescriptorIndex)

    {

        FileName = fileName;

        FileDescriptorIndex = fileDescriptorIndex;

    }

}

public class OpenedFile

{

    public int FileDescriptorNumber { get; set; }

    public int DescriptorIndex { get; set; }

    public int Position { get; set; }

}