УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

**“ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО”**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 11

по дисциплине «Операционные системы»

На тему **«**Управление файловой системой»

Выполнил: студент гр. ИТП-11

Козлов О. В.

Принял: преподаватель-стажер

Карась О. В.

Гомель 2022

**Цель**: реализовать вспомогательную утилиту для файловой системы.

**Ход работы**

**Задание 1.** Реализовать вспомогательную утилиту для управления файловой системой.

В соответствии с вариантом необходимо реализовать утилиту «Журнализация».

В качестве файловой системы была выбрана последовательная файловая система с выделением непрерывной последовательности блоков.

Простейший способ – хранить каждый файл как непрерывную последовательность блоков диска. При непрерывном расположении файл характеризуется адресом и длиной (в блоках). Файл, стартующий с блока *b,* занимает затем блоки *b*+1, *b*+2, ... *b*+*n*-1.

Эта схема имеет два преимущества. Во-первых, ее легко реализовать, так как выяснение местонахождения файла сводится к вопросу, где находится первый блок. Во-вторых, она обеспечивает хорошую производительность, так как целый файл может быть считан за одну дисковую операцию.

Другим средством поддержки целостности является заимствованный из систем управления базами данных прием, называемый журнализация (иногда употребляется термин "журналирование"). Последовательность действий с объектами во время файловой операции протоколируется, и если произошла остановка системы, то, имея в наличии протокол, можно осуществить откат системы назад в и сходное целостное состояние, в котором она пребывала до начала операции.

Подобная избыточность может стоить дорого, но она оправданна, так как в случае отказа позволяет реконструировать потерянные данные. Для отката необходимо, чтобы для каждой протоколируемой в журнале операции существовала обратная. Например, для каталогов реляционных СУБД это именно так. По этой причине, в отличие от СУБД, в файловых системах протоколируются не все изменения, а лишь изменения метаданных (индексных узлов, записей в каталогах и др.). Изменения в данных пользователя в протокол не заносятся. Кроме того, если протоколировать изменения пользовательских данных, то этим будет нанесен серьезный ущерб производительности системы, поскольку кэширование потеряет смысл. Журнализация реализована в *NTFS*, *Ext3FS*, *ReiserFS* и других системах.

При запуске программы мы увидим следующее окно, в котором нам предложат ввести размер памяти, например, введем 32 и добавим несколько файлов, результат данных действий представлен ниже:

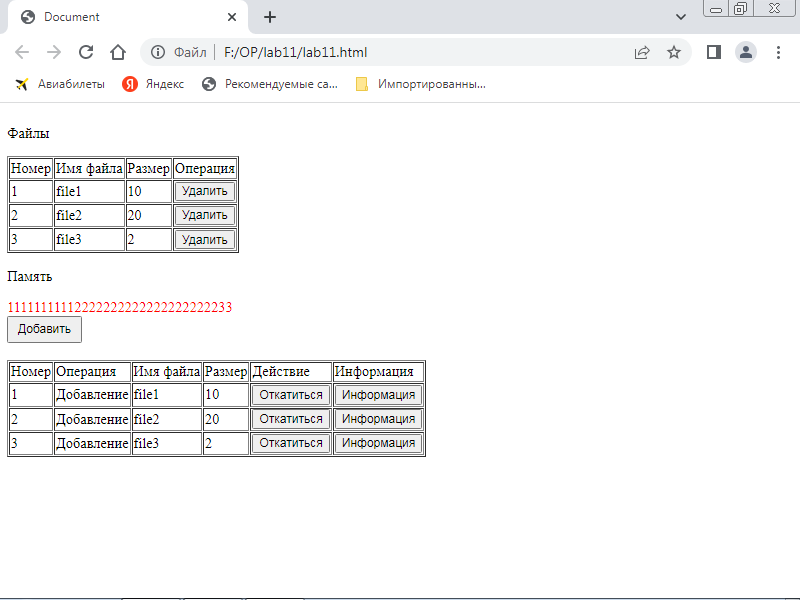


Рисунок 1 – Окно программы

Мы можем увидеть несколько таблиц: в первой демонстрируется список файлов, во второй – Журнал операций. И согласно заданию, необходимо реализовать утилиту «Журнализация», суть которой состоит в том, что утилита сохраняет действия пользователя, и в нужный момент можно откатиться до необходимого нам состояния. Например, в нашей программе мы совершили три действия: создали три файла, и это означает, что у нас есть три состояния: до создания файла *file1*, *file2*, *file3.* Откатившись до состояния 3, например, мы вернемся к тому моменту, когда еще не был создан *file3* и присутствовало только два файла. Чтобы убедиться в этом, мы можем посмотреть состояние нажав на кнопку информация, например, мы хотим откатиться до состояния 3, тогда посмотрим информацию о откате:

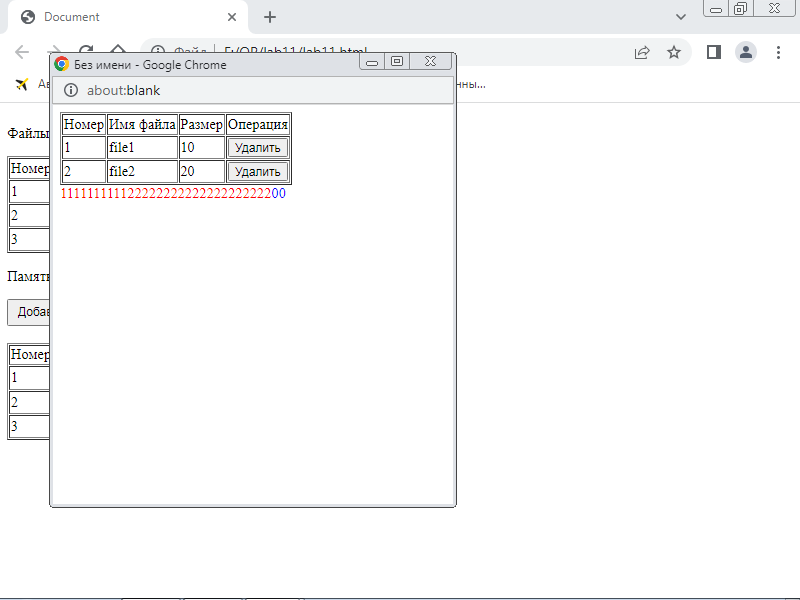


Рисунок 2 – Состояние памяти до

добавления 3-го файла

Теперь мы можем вернуться назад к предыдущему состоянию, в результате мы увидим только два файла: *file1*, *file2*, так как третий на тот момент времени еще не добавлен. Результат отката представлен ниже:

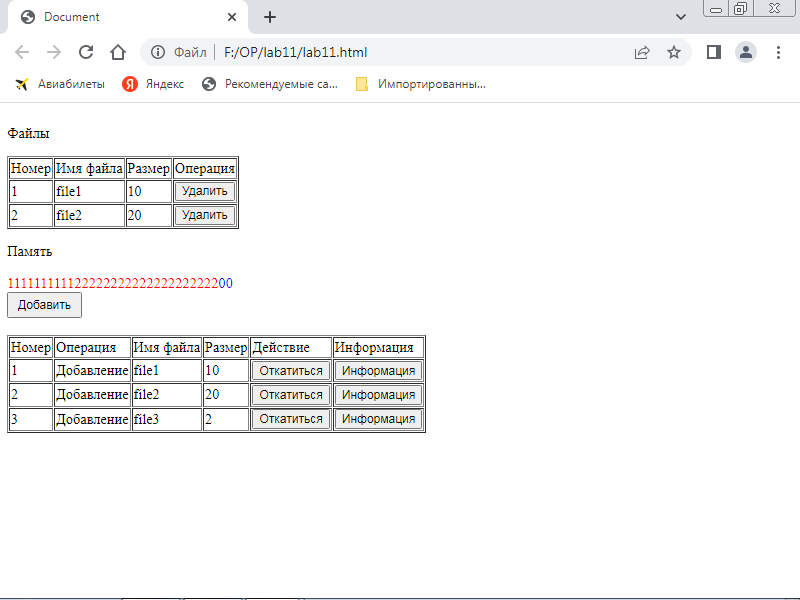


Рисунок 3 – Результат отката

Код программы представлен в приложении А.

**Вывод**: в ходе работы была реализована вспомогательная утилита «Журнализация».

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

**Код утилиты «Журнализация»**

"use strict";

let addFileBtn = document.getElementById("addFile");

let filesInfo = document.querySelector(".info\_\_files");

let memoryInfo = document.querySelector(".info\_\_memory");

let journalTable = document.querySelector(".journal\_\_table");

let memory = {

    blocksAmount: 0,

    blocks: [],

    files: [],

    ff: 0

}

let journal = []

memory.blocksAmount = +prompt("Введите размер памяти");

for (let i = 0; i < memory.blocksAmount; i++)

    memory.blocks[i] = 0;

addFileBtn.addEventListener("click", (e) => {

    let name = prompt("Имя файла: ");

    let size = +prompt("Размер файла:");

    addFile(name, size);

    showFilesTable(memory, true);

    showMemory(memory, true);

    showJournalTable(true);

});

function getBackMemoryTo(number) {

    memory = journal[number].memory;

    showFilesTable(memory, true);

    showMemory(memory, true);

    showJournalTable(true);

}

function addFile(name, size) {

    let startPos = 0, endPos = 0, isFound = false, isEnoughMemory;

    if (memory.files.some(file => file.name == name))

    {

        alert("Файл с таким именем уже есть!");

        return;

    }

    while (!isFound) {

        for (let i = startPos; i < memory.blocks.length; i++)

            if (memory.blocks[i] == 0)

            {

                startPos = i++;

                break;

            }

        for (let i = startPos; i < memory.blocks.length; i++)

            if (memory.blocks[i] == 0 && endPos - startPos + 1 !== size)

                endPos = i;

            else if (memory.blocks[i] !== 0)

                break;

        if (endPos - startPos + 1 < size) {

            startPos = endPos + 1;

        } if (endPos - startPos + 1 == size) {

            isFound = true;

            isEnoughMemory = true;

        } else {

           isFound = true;

           isEnoughMemory = false;

        }

    }

    if (!isEnoughMemory) {

        alert("Не достаточно памяти!");

        return;

    }

    let file = {

        name: name,

        size: size,

        start: startPos,

        end: endPos

    };

    journal.push({

        type: "Добавление",

        file: file,

        memory: copyObj(memory)

    });

    memory.files.push(file);

    memory.ff++;

    for (let i = startPos; i <= endPos; i++)

        memory.blocks[i] = memory.ff;

}

function deleteFile(name) {

    if (!memory.files.some(file => file.name == name)) {

        alert("Такого файла нет!");

        return;

    }

    let fileI = 0;

    memory.files.forEach((file, ind) => file.name == name && (fileI = ind));

    journal.push({

        type: "Удаление",

        file: memory.files[fileI],

        memory: copyObj(memory)

    });

    for (let i = memory.files[fileI].start; i <= memory.files[fileI].end; i++)

        memory.blocks[i] = 0;

    memory.files = del(memory.files, fileI);

    showFilesTable(memory, true);

    showMemory(memory, true);

    showJournalTable(true);

}

function copyObj(obj) {

    let newObj = {};

    for (let prop in obj) {

        if (Array.isArray(obj[prop]))

            newObj[prop] = [...obj[prop]];

        else if (typeof obj[prop] == "object")

            newObj[prop] = copyObj(obj[prop])

        else

            newObj[prop] = obj[prop];

    }

    return newObj;

}

function del(arr, pos) {

    if (pos + 1 == arr.length)

        return [...arr.slice(0, pos)]

    else if (pos == 0)

        return [...arr.slice(1)]

    else

        return [...arr.slice(0, pos), ...arr.slice(pos + 1, arr.length)];

}

function showFilesTable(memory, shouldShow) {

    let str = "";

    str += `<table border='1'><thead><tr><td>Номер</td><td>Имя файла</td><td>Размер</td><td>Операция</td></tr></thead><tbody>`;

    memory.files.forEach((file, ind) => {

        str += "<tr>";

        str += `<td>${ind + 1}</td><td>${file.name}</td><td>${file.size}</td><td><input type='button' value='Удалить' onclick='deleteFile("${file.name}")'></td></tr>`;

    });

    str += "</tbody></table>";

    if (shouldShow)

        filesInfo.innerHTML = str;

    return str;

}

function showMemory(memory, shouldShow) {

    let str = "";

    let c = 0;

    memoryInfo.innerHTML = "";

    memory.blocks.forEach(block => {

        str += `<span ${block !== 0 ? "style='color:red;'" : "style='color:blue;'"}>${block}</span>`

    });

    if (shouldShow)

        memoryInfo.innerHTML += str + "<br>";

    return str;

}

function showJournalTable(shouldShow) {

    let str = "";

    str += `<table border='1'><thead><tr><td>Номер</td><td>Операция</td><td>Имя файла</td><td>Размер</td><td>Действие</td><td>Информация</td></tr></thead><tbody>`;

    journal.forEach((j, ind) => {

        str += "<tr>";

        str += `<td>${ind + 1}</td><td>${j.type}</td><td>${j.file.name}</td><td>${j.file.size}</td><td><input type='button' value='Откатиться' onclick='getBackMemoryTo(${ind})'</td><td><input type='button' value='Информация' onclick='showPrevMemory(${ind})'</td></tr>`;

    });

    str += "</tbody></table>";

    if (shouldShow)

        journalTable.innerHTML = str;

    return str;

}

function showPrevMemory(number) {

    let w = window.open("", "", "width=400, height=400");

    let memory = journal[number].memory

    w.document.write(showFilesTable(memory) + showMemory(memory));

}