Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Операционные среды и системное программирование

ОТЧЁТ
к лабораторной работе №2
на тему

РАБОТА С ФАЙЛАМИ

Выполнил: студент гр. 253503
Котова К.А.
Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

1. ФОРМУЛИРОВКА ЗАДАЧИ

Целью данной лабораторной работы является создание файла прямого доступа – упрощенной базы данных.

Упрощенная база данных может использоваться для хранения рабочих данных, конфигураций и т.п. Данные структурированные. Нужно создать примитивы (процедуры) доступа к структурам данных в файле (воспроизводить язык запросов не требуется).

Доступ осуществляется путем отображения файла в память. Требуется продумать описание структуры базы (простейшее минимальное), удаление объектов из базы («дырки», сжатие/дефрагментация базы), расширение базы (увеличение размера файла).

2. ОПИСАНИЕ ОСНОВНЫХ ФУНКЦИЙ ПРОГРАММЫ

2.1 CreateAndMapFile

CreateAndMapFile — это функция, которая выполняет создание файла и его отображение в память. Сначала она вызывает функцию CreateFile, чтобы создать или открыть файл с правами на чтение и запись. Если создание файла завершилось неудачно, функция выводит сообщение об ошибке и завершает выполнение. Далее она использует функции SetFilePointer [3] и SetEndOfFile [5], чтобы установить начальный размер файла. Это нужно для того, чтобы файл был готов к использованию с определенным размером. После этого, с помощью функции CreateFileMapping [1], создается отображение файла в память.

Если операция создания отображения завершилась успешно, функция вызывает MapViewOfFile [2], которая отображает файл в виртуальной памяти. Это позволяет программе работать с файлом как с обычной памятью. Если на любом этапе происходит ошибка, выводится сообщение, и файл не создается.

Цель этой функции — предоставить прямой доступ к файлу через память для оптимизации работы с большими объемами данных.

2.2 WriteRecord

WriteRecord предназначена для записи данных в определенное место в отображении файла. Эта функция принимает индекс, уникальный идентификатор записи (ID) и строку с именем. Она преобразует указатель на отображение в массив структур Record, где каждая структура содержит ID и строку с именем. Функция записывает переданные данные в структуру на указанной позиции, копируя строку имени с помощью безопасной функции strncpy, чтобы избежать переполнения буфера. Таким образом, данные записываются прямо в память, которая отображает файл, что упрощает и ускоряет процесс работы с файлами.

2.3 DeleteRecord

Функция удаляет запись из файла логически, заменяя удаляемую запись последней в массиве. Это минимизирует накладные расходы на удаление и поддерживает целостность данных. Вначале проверяется, корректен ли переданный индекс и существует ли запись для удаления. Если всё в порядке, последняя запись копируется на место удаляемой, а последняя запись обнуляется (значения идентификатора и имени очищаются). После этого общее количество записей уменьшается на единицу. Такая реализация удаления данных помогает избежать фрагментации и излишнего перемещения данных. Основные функции, которые обеспечивают этот процесс – это

parallelSort, выполняющая сортировку частей массива, и parallelMerge, которая отвечает за слияние отсортированных частей.

2.4 ResizeFileMapping

Эта функция изменяет размер файла и пересоздает его отображение в память. Сначала старое отображение удаляется с помощью UnmapViewOfFile [4], освобождая занятую память. Затем с помощью SetFilePointer [3] и SetEndOfFile [5] изменяется размер файла, после чего создается новое отображение файла в память с обновленным размером с помощью CreateFileMapping [1]. Если на каком-либо этапе возникает ошибка, программа выводит соответствующее сообщение. После успешного создания нового отображения файл снова отображается в память с помощью MapViewOfFile [2]. Эта функция позволяет динамически управлять размером файла, делая его больше или меньше в зависимости от текущих нужд программы.

2.5 ManageFileSize

Функция контролирует динамическое изменение размера файла. Если количество записей превышает текущий лимит, она вызывает ResizeFileMapping, чтобы расширить файл, добавив место для новых данных. Если количество записей уменьшается (например, из-за удаления) до определенного порога, функция уменьшает размер файла. Это помогает оптимизировать использование дискового пространства и ресурсов. После каждого изменения файла лимит записей обновляется, а пользователю выводится сообщение о новом размере файла.

3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ

3.1 Добавление записей, превышающих заданное количество

При добавлении пятнадцати записей в файл, максимальный размер которого ограничен десятью записями, программа сначала заполняет файл до предельного размера. Как только количество записей достигает установленного лимита, происходит расширение файла, чтобы вместить дополнительные данные.

В процессе работы программы файл расширяется автоматически, позволяя сохранить все пятнадцать записей. Это расширение происходит, как только файл превышает свой первоначальный размер, демонстрируя гибкость в работе с большим количеством данных. Таким образом, файл динамически увеличивается, обеспечивая сохранение всех добавленных записей, превышающих исходный лимит.

Результат работы программы представлен на рисунке 3.1.

```
Record added. Total records: 1
File shrunk. New limit: 5 records.
Record added. Total records: 2
Record added. Total records: 3
Record added. Total records: 4
Record added. Total records: 5
File resized. New limit: 7 records.
Record added. Total records: 6
Record added. Total records: 7
File resized. New limit: 10 records.
Record added. Total records: 8
Record added. Total records: 9
Record added. Total records: 10
File resized. New limit: 15 records.
Record added. Total records: 11
Record added. Total records: 12
Record added. Total records: 13
Record added. Total records: 14
Record added. Total records: 15
```

Рисунок 3.1 – Результат работы программы при добавлении записей в файл

3.2 Удаление данных из файла

В ходе работы программы мы можем удалять часть записей из файла. После удаления программа изменит размер файла, сохранив только оставшиеся данные, так как их количество было слишком мало для данного

размера файла. Если количество записей после удаления значительно не уменьшится, то и размер файла изменяться не будет.

В данном примере изначальный размер файла составлял двадцать записей. В результате работы программы мы удалили все записи в файле. В тот момент, когда количество записей стало меньше минимального допустимого, файл ужался.

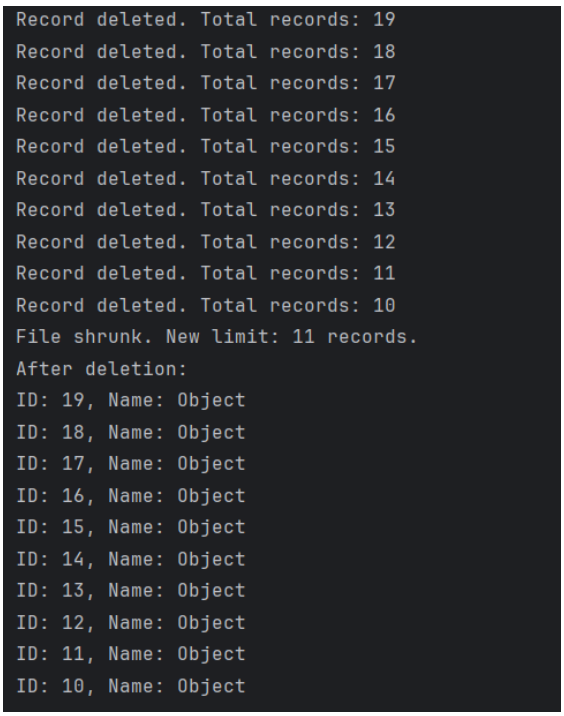

Рисунок 3.2 – Результат работы программы при удалении записей

ВЫВОД

В ходе выполнения лабораторной работы была разработана программа, которая позволяет работать с файлами в режиме прямого доступа, поддерживая динамическое изменение их размера в зависимости от добавляемых или удаляемых записей. Основная задача программы — эффективно управлять файлом, который ограничен по количеству записей, и автоматически изменять его размер при изменении числа хранимых данных.

На этапе добавления записей, программа сначала заполняет файл до предельно допустимого количества записей. Как только лимит в десять записей достигается, программа динамически расширяет файл, чтобы вместить дополнительные данные. При этом расширение происходит автоматически, что упрощает работу пользователя и позволяет сохранять новые записи без потерь данных. Такой подход демонстрирует высокую гибкость программы в работе с большими объемами информации, так как она способна адаптироваться к изменению объема данных, изменяя размер файла по мере необходимости.

Удаление данных из файла также происходит эффективно. Когда программа удаляет часть записей, она корректирует размер файла, если количество оставшихся данных значительно меньше текущего размера файла.

Это означает, что программа освобождает неиспользуемое дисковое пространство, что позволяет избежать фрагментации файла и избыточного использования ресурсов. Когда количество оставшихся записей в файле уменьшается до определенного порога, программа автоматически уменьшает размер файла, освобождая память. В этом проявляется важная особенность программы — она не только способна расширять файл для добавления новых записей, но и уменьшать его, если данные удаляются.

Таким образом, программа эффективно решает задачу динамического управления файлами. Она демонстрирует, как можно использовать механизмы отображения файла в память для быстрой работы с данными, а также реализует подход к управлению размером файла, который позволяет экономить дисковое пространство и избегать проблем с фрагментацией.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Документация Microsoft для функции CreateFileMapping [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createfilemappinga – Дата доступа: 01.10.2024.

[2] Документация Microsoft для функции MapViewOfFile [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile –Дата доступа: 01.10.2024.

[3] Документация Microsoft для функции SetFilePointer [Электронный ресурс]. – Электронные данные. – Режим доступа https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-setfilepointer – Дата доступа: 01.10.2024.

[4] Документация Microsoft для функции UnmapViewOfFile [Электронный ресурс]. – Электронные данные. – Режим доступа https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-unmapviewoffile –Дата доступа: 14.09.2024.

[5] Документация Microsoft для функции SetEndOfFile [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/de-de/windows/win32/api/fileapi/nf-fileapi-setendoffile –Дата доступа: 15.09.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Код программы

Содержимое файла database.cpp

```
#include <windows.h>
#include <iostream>
#include <string>

struct Record {
    int id;
    char name[50];
};

HANDLE CreateAndMapFile(LPCSTR filename, DWORD size, HANDLE
&hFile, LPVOID &mappedView) {
    hFile = CreateFile(filename, GENERIC_READ | GENERIC_WRITE,
0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == INVALID_HANDLE_VALUE) {
        std::cerr << "Failed to create file" << std::endl;
        return NULL;
    }

    if (SetFilePointer(hFile, size, NULL, FILE_BEGIN) ==
INVALID_SET_FILE_POINTER && GetLastError() != NO_ERROR) {
        std::cerr << "Failed to set file pointer with error: "
<< GetLastError() << std::endl;
        CloseHandle(hFile);
        return NULL;
    }
    if (!SetEndOfFile(hFile)) {
        std::cerr << "Failed to set end of file with error: " <<
GetLastError() << std::endl;
        CloseHandle(hFile);
        return NULL;
    }

    HANDLE hMapFile = CreateFileMapping(hFile, NULL,
PAGE_READWRITE, 0, size, NULL);
    if (!hMapFile) {
        std::cerr << "Failed to create file mapping" <<
std::endl;
        CloseHandle(hFile);
        return NULL;
    }

    mappedView = MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0,
0, size);
    if (!mappedView) {
        std::cerr << "Failed to map file view" << std::endl;
        CloseHandle(hMapFile);
```

```
        CloseHandle(hFile);
        return NULL;
    }

    return hMapFile;
}

void WriteRecord(LPVOID mappedView, int index, int id, const
char* name) {
    Record* records = static_cast<Record*>(mappedView);
    records[index].id = id;
    strncpy(records[index].name, name,
sizeof(records[index].name) - 1);
    records[index].name[sizeof(records[index].name) - 1] = '\0';
}

void DeleteRecord(LPVOID mappedView, int index, int
&recordCount) {
    if (recordCount <= 0 || index >= recordCount) {
        std::cerr << "Invalid index or no records to delete." <<
std::endl;
        return;
    }

    Record* records = static_cast<Record*>(mappedView);

    if (index != recordCount - 1) {
        records[index] = records[recordCount - 1];
    }

    records[recordCount - 1].id = -1;
    strcpy(records[recordCount - 1].name, "");

    recordCount--;
}


HANDLE ResizeFileMapping(HANDLE hFile, HANDLE hMapFile, LPVOID
&mappedView, DWORD newSize) {

    if (!UnmapViewOfFile(mappedView)) {
        std::cerr << "Failed to unmap view of file with error: "
<< GetLastError() << std::endl;
        return NULL;
    }
    CloseHandle(hMapFile);  // Закрываем старое отображение

    if (SetFilePointer(hFile, newSize, NULL, FILE_BEGIN) ==
INVALID_SET_FILE_POINTER && GetLastError() != NO_ERROR) {
        std::cerr << "Failed to set file pointer with error: "
<< GetLastError() << std::endl;
        return NULL;
    }
```

```
    if (!SetEndOfFile(hFile)) {
        std::cerr << "Failed to set end of file with error: " <<
GetLastError() << std::endl;
        return NULL;
    }


    hMapFile = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0,
newSize, NULL);
    if (hMapFile == NULL) {
        std::cerr << "CreateFileMapping failed with error: " <<
GetLastError() << std::endl;
        return NULL;
    }


    mappedView = MapViewOfFile(hMapFile, FILE_MAP_ALL_ACCESS, 0,
0, newSize);
    if (!mappedView) {
        std::cerr << "Failed to map view of file with error: "
<< GetLastError() << std::endl;
        CloseHandle(hMapFile);
        return NULL;
    }

    return hMapFile;
}


void ManageFileSize(HANDLE hFile, HANDLE &hMapFile, LPVOID
&mappedView, int &recordCount, int &maxRecords) {
    const int increaseStep = static_cast<int>(maxRecords / 2);
    DWORD newSize;

    if (recordCount >= maxRecords) {
        newSize = sizeof(Record) * (maxRecords + increaseStep);
        hMapFile = ResizeFileMapping(hFile, hMapFile,
mappedView, newSize);
        if (hMapFile) {
            maxRecords += increaseStep;
            std::cout << "File resized. New limit: " <<
maxRecords << " records." << std::endl;
        } else {
            std::cerr << "Failed to resize the file!" <<
std::endl;
            return;
        }
    }


    if (recordCount < maxRecords / 2 && maxRecords >
increaseStep) {
```

```
        newSize = sizeof(Record) * (maxRecords - increaseStep);
        hMapFile = ResizeFileMapping(hFile, hMapFile,
mappedView, newSize);
        if (hMapFile) {
            maxRecords -= increaseStep;
            std::cout << "File shrunk. New limit: " <<
maxRecords << " records." << std::endl;
        } else {
            std::cerr << "Failed to shrink the file!" <<
std::endl;
            return;
        }
    }
}

int main() {
    LPVOID mappedView;
    HANDLE hFile;
    int maxRecords = 10;
    const DWORD initialSize = sizeof(Record) * maxRecords;

    HANDLE hMapFile = CreateAndMapFile("database2.bin",
initialSize, hFile, mappedView);

    if (hMapFile) {
        int recordCount = 0;

        for(int i = 0; i < 20; i++) {
            WriteRecord(mappedView, recordCount++, i, "Object");
            std::cout << "Record added. Total records: " <<
recordCount << std::endl;
            ManageFileSize(hFile, hMapFile, mappedView,
recordCount, maxRecords);
        }

        std::cout << "Before deletion:" << std::endl;
        Record* records = static_cast<Record*>(mappedView);
        for (int i = 0; i < recordCount; i++) {
            std::cout << "ID: " << records[i].id << ", Name: "
<< records[i].name << std::endl;
        }

        for(int i = 0; i < recordCount; i++) {
            DeleteRecord(mappedView, i, recordCount);
            std::cout << "Record deleted. Total records: " <<
recordCount << std::endl;
            ManageFileSize(hFile, hMapFile, mappedView,
recordCount, maxRecords);
        }



        std::cout << "After deletion:" << std::endl;
```

```
        for (int i = 0; i < recordCount; i++) {
            std::cout << "ID: " << records[i].id << ", Name: "
<< records[i].name << std::endl;
        }

        UnmapViewOfFile(mappedView);
        CloseHandle(hMapFile);
        CloseHandle(hFile);
    }

    return 0;
}
```