**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И РОБОТОТЕХНИКИ

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

**ОТЧЁТ**

**по лабораторной работе №4**

# «Работа с файлами»

**по дисциплине «Операционные системы и системное программирование»**

Выполнил: ст. гр.

Принял: Разоренов Н. А.

Минск 2020

**Лабораторная работа №4**

**Работа с файлами**

**1. Цель работы.**

Изучить основы работы с двоичным и текстовым файлами базе WIN32 API.

**2. Изучаемые вопросы.**

1. Создание текстового файла. Структура текстового файла.
2. Запись/Чтение текстового файла.
3. Создание двоичного файла. Структура двоичного файла.
4. Запись/Чтение двоичного файла.
5. Атрибуты файла. Чтение/Установка атрибутов файла.
6. Преобразование информации при записи и чтении в/из файла.
7. Внутреннее представление информации разного типа в оперативной и дисковой памяти (файлах).
8. Дампы памяти.

**3. Постановка задачи.**

Разработать WIN32 Application с диалогами, которое должно обеспечить:

1. Записать в оба типа файлов данные следующих типов (значение данных задаются через диалоговое окно):
   * BYTE, UINT;
   * INT32/ INT64;
   * WORD, DWORD;
   * LONG, BOOL;
   * Float, Double;
   * Char;.
   * Строки (в файле должно быть 3 и более строк).
2. Чтение данных из двоичного файла и их отображение в диалоговом окне.
3. Чтение данных из текстового файла и их отображение в диалоговом окне.
4. Отображать дампы значений данных файлов.
5. Для сохранения файла и загрузки файла использовать стандартные диалоги посредством вызова функций GetOpenFileName() и GetSaveFileName().

В результате выполнения работы студент должен уметь анализировать кодировку данных в символьном/текстовом и двоичном форматах на основе дампов памяти (файлов). Дамп памяти – отображение байтов памяти (оперативной или дисковой) в шестнадцатеричной системе.

**4. Ход выполнения работы.**

1. **Создание текстового файла. Структура текстового файла.**

Текстовые файлы представляют собой набор однобайтовых (ANSI) или двухбайтовых данных (UNICODE), содержащих коды символов, т. е. текстовую информацию. Открыть текстовый файл или создать новый можно вызвав стандартный диалог открытия файла OPENFILENAME ofn. Заполнение его полей:

ofn.lStructSize = sizeof(OPENFILENAME); // размер структуры

ofn.hwndOwner = hwnd; // дескриптор род. окна

ofn.nFilterIndex = 1; // количество фильтров

ofn.lpstrFile = fileName; //адрес буфера имени файла

ofn.lpstrFilter = "Текстовый файл\0\*.txt;\0Все\0\*.\*\0"; // фильтры

Вызов стандартного диалога:

GetOpenFileName(&ofn);

Для создания текстового файла нужно указать расширение .txt. Создание текстового файла, ofn.lpstrFile – имя файла:

HANDLE hwndText;

hwndText = CreateFile(

fileName, // имя файла

GENERIC\_WRITE, // режим доступа

0, // совместный доступ

NULL, // SD (дескр. защиты)

CREATE\_ALWAYS, // как действовать

FILE\_ATTRIBUTE\_NORMAL, // атрибуты файла

NULL // дескр. Шаблона файла

);

1. **Запись / Чтение текстового файла.**

Получаем длину файла:

end = GetFileSize(hwndText, NULL);

Устанавливаем указатель файла на начало файла:

SetFilePointer(hwndText, 0, NULL, FILE\_BEGIN);

Считываем из него данные:

ReadFile(hwndText, TxtFile, end, &dwBytes, NULL);

Запись в текстовый файл:

size = GetWindowTextLength(GetDlgItem(hwnd, IDC\_EDIT[i])); GetWindowText(GetDlgItem(hwnd, IDC\_EDIT[i]), Buffer, size + 1); WriteFile(hwndText, Buffer, size, &size, NULL);

1. **Создание двоичного файла. Структура двоичного файла.**

HANDLE hwndBin;

hwndBin = CreateFile(fileName, GENERIC\_WRITE, 0, 0, CREATE\_ALWAYS, 0, 0); // параметры указаны в первом ответе

if (hwndBin == INVALID\_HANDLE\_VALUE)

MessageBox(hwndMain, "Ошибка", "Ошибка", MB\_OK | MB\_ICONWARNING);

Бинарные файлы содержат массивы длинных чаще всего нетекстового вида, это могут быть как целые, так и вещественные числа, пользовательские структуры данных и т. п.

1. **Запись/Чтение двоичного файла.**

Мы записываем в бинарный файл нетекстовые данные. Для этого мы их считываем из edit в буфер, после этого преобразуем их в данные соответствующего типа и записываем в файл:

BYTE Byte;

INT32 Int32;

char Buffer[100] = "";

Byte = GetDlgItemInt(hwnd, IDC\_EDIT1, NULL, TRUE);

WriteFile(hwndBin, &Byte, sizeof(Byte), &dwBytes, NULL);

Int32 = GetDlgItemInt(hwnd, IDC\_EDIT3, NULL, TRUE);

WriteFile(hwndBin, &Int32, sizeof(Int32), &dwBytes, NULL);

Аналогично происходит запись остальных типов данных. Чтение бинарного файла

происходит побайтно.

ReadFile(hwndBin, &Byte, sizeof(BYTE), &dwBytes, NULL);

printf\_s(q, "%d ", Byte);

strcat\_s(BinnFile, q);

SetDlgItemTextA(hwnd, IDC\_EDIT13, BinnFile);

1. **Атрибуты файла. Чтение/Установка атрибутов файла.**

Атрибуты файла можно задавать при создании файла или непосредственно при выполнении программы. В нашей работе мы проверяем, является ли файл скрытым или открытым только для чтения:

DWORD attribute;

Получаем атрибуты файла:

attribute = GetFileAttributesW(LPCWSTR(fileName));

Проверка установленных атрибутов:

if (attribute & FILE\_ATTRIBUTE\_READONLY)

MessageBox(hwndMain, "Ошибка", "FILE\_ATTRIBUTE\_READONLY", MB\_OK);

Также можно установить атрибут файлу. В данном случае мы делаем файл скрытым:

SetFileAttributes(fileName, attribute | FILE\_ATTRIBUTE\_HIDDEN);

1. **Преобразование информации при записи и чтении в/из файла.**

При записи и чтения в/из файла мы используем функцию и atof, которые преобразуют строку из edit во float:

Float = atof(Buffer);

А также функцию sprintf\_s для преобразования данных определенного формата в строку:

sprintf\_s(q, "%d ", Byte);

sprintf\_s(q, "%.2f ",Float);

1. **Внутреннее представление информации разного типа в оперативной и дисковой памяти (файлах).**

Представление данных в бинарных файлах ничем не отличается от представления в оперативной памяти. В оперативной памяти Арифметические данные занимает определенное количество байт, которое зависит от типа данных. При этом данные в памяти (оперативная память, файлы) располагаются младшими байтами вперед, т. е. при выводе его дампа мы их наблюдаем в «перевернутом» виде. Текстовые данные занимают 1 байт под символ. В этом байте хранится шестнадцатеричный код данного символа.

1. **Дампы памяти.**

Для вывода дампа памяти на экран преобразуем данные в шестнадцатеричный формат и выводим на экран:

Int i = 0;

Int k = 0;

char dump[300] = "";

char BinFile[200] = "";

while (i < end)

{

if (i % 16 == 0 && i != 0)

k += sprintf(dump + k, "\r\n");

k += sprintf(dump + k, "%.2X ", (unsigned char)BinFile[i]);

i++;

}

SetDlgItemTextA(hwnd, IDC\_EDIT14, dump);

**5. Результат выполнения работы программы:**

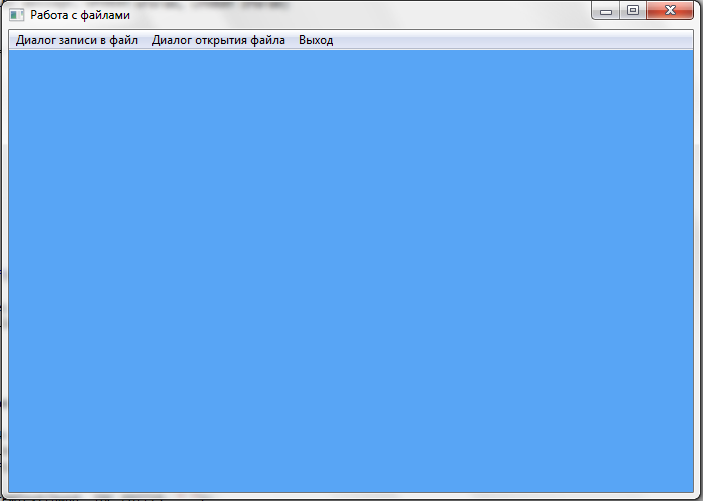
****

Рисунок 1 – Работа с немодальным окном

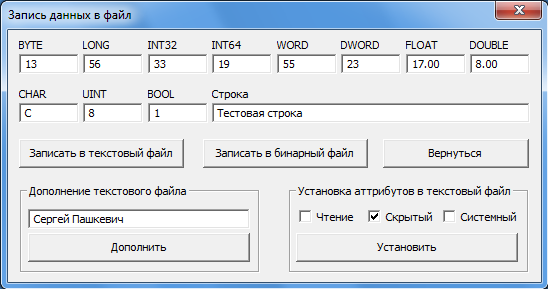


Рисунок 2 – Диалоговое окно «Запись в файл»

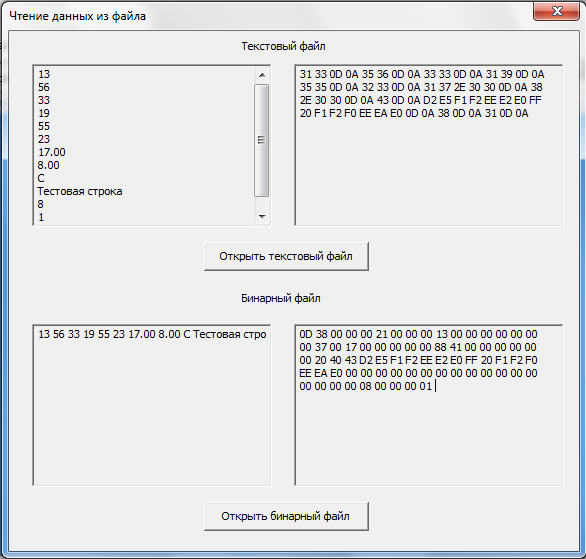


Рисунок 3 – Диалоговое окно «Дампы файлов»