Изучение системных вызовов Win32 API работы с файлами.

Задача 1.

Написать программу, реализующую произвольный доступ к записям в файле двумя способами: с помощью указателя файла (file pointer).

Структура записи:

- номер записи;
- время создания записи (в формате FILETYME);
- текстовая строка заданной длины (80 символов);
- счетчик, показывающий, сколько раз запись изменялась.

Запись может быть пустая (инициализирована нулями).

В заголовке файла хранить количество непустых записей в файле и размер файла. Общее количество записей в файле задается из командной строки. Пользователь должен иметь возможность удалять и модифицировать существующие записи, обращаясь к ним по номеру. Интерфейс с пользователем реализуется на усмотрение студента.

Код программы:

```
#define _CRT_SECURE_NO_WARNINGS
#include<iostream>
#include<windows.h>
#include<string>
#include<stdio.h>
#include<string.h>
#include<tchar.h>

#define STEP 100
#define INFO_SIZE 80
#define FREE_ROW_ID -1
#define LINE_T_SIZE sizeof(LINE_T)

using namespace std;


typedef struct {
  DWORD id;
  FILETIME time;
  CHAR info[INFO_SIZE];
  DWORD changed;
}LINE_T;

typedef struct {
  DWORD lastId;
  DWORD fileSize;
  DWORD exists;
}FIRST_LINE_T;



void ModifyLine(INT);
DWORD FindLine(INT);
void DeleteLineFromFile(INT);
```

```
void ShowInformationFile();
void InitData();
void WriteLineToFile();

HFILE file;


int _tmain(INT argc, TCHAR** argv) {

  HANDLE std = GetStdHandle(STD_INPUT_HANDLE);
  DWORD mode;
  GetConsoleMode(std, &mode);
  SetConsoleMode(std, mode | ENABLE_PROCESSED_INPUT);

  LPTSTR act = NULL;
  LPCTSTR fileName = L"../fileData.txt";

  try {

    if (argv[1] == NULL) throw 1;
    //Create file
    file = (HFILE)CreateFile(
      (LPCTSTR)fileName, GENERIC_WRITE | GENERIC_READ, 0,
      NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
    int sizeOfFile = GetFileSize((HANDLE)file, NULL);
    if (sizeOfFile == 0) {
      InitData();
    }
    BYTE commmand = (BYTE)act[1];
    INT num;
    if (commmand == 's')  ShowInformationFile();
    else if (commmand == 'i') while (TRUE) WriteLineToFile();
    else if (commmand == 'd') {
      argv[2] == NULL ? throw 2 : num = _ttoi(argv[2]); DeleteLineFromFile(num);
    }
    else if (commmand == 'm') {
      argv[2] == NULL ? throw 2 : num = _ttoi(argv[2]);
      cout << "Input new info: ";
      ModifyLine(num);
    }
  }
  catch (int e) {
    cout << "Error: argument is wrong " << endl;
  }

  CloseHandle((HANDLE)file);
}

DWORD FindLine(INT num) {
  //Find free row
  DWORD count;
  BOOL res = TRUE;
  LINE_T line = {};
  SetFilePointer((HANDLE)file, LINE_T_SIZE, NULL, FILE_BEGIN);

  do res = ReadFile((HANDLE)file, (LPVOID)&line, LINE_T_SIZE, &count, NULL);
  while ((res & count != 0) && (line.id != num));

  if (line.id == num) {
    int offset = (LINE_T_SIZE);
    return SetFilePointer((HANDLE)file, -offset, NULL, FILE_CURRENT);
  }
  else {
    SetFilePointer((HANDLE)file, 0, NULL, FILE_END);
  }
```

```
  return FALSE;
}

void ModifyLine(INT num) {
  DWORD id = -1, count, newChang;
  if (FindLine(num)) {
    LINE_T line = {};
    ReadFile((HANDLE)file, (LPVOID)&line, LINE_T_SIZE, &count, NULL);

    //new inf
    if (!fgets(line.info, INFO_SIZE, stdin)) {
      return;
    }

    //set count of changes
    line.changed++;

    int offset = (LINE_T_SIZE);
    SetFilePointer((HANDLE)file, -offset, NULL, FILE_CURRENT);
    WriteFile((HANDLE)file, (LPVOID)&line, LINE_T_SIZE, &count, NULL);
  }
  else {
    cout << "Record doesn't exist." << endl;
  }
}

void DeleteLineFromFile(INT num) {
  DWORD id = FREE_ROW_ID, count;
  if (FindLine(num)) {
    WriteFile((HANDLE)file, (LPVOID)&id, sizeof(DWORD), &count, NULL);
  }
  else {
    cout << "Record doesn't exist." << endl;
    return;
  }
  //dec index
  FIRST_LINE_T fline = {};
  DWORD red;
  SetFilePointer((HANDLE)file, 0, NULL, FILE_BEGIN);
  ReadFile((HANDLE)file, (LPVOID)&fline, sizeof(FIRST_LINE_T), &red, NULL);
  fline.exists--;
  SetFilePointer((HANDLE)file, 0, NULL, FILE_BEGIN);
  WriteFile((HANDLE)file, (LPVOID)&fline, sizeof(FIRST_LINE_T), &red, NULL);
}

void WriteLineToFile() {

  LINE_T fileline = {};
  FIRST_LINE_T fline = {};

  //init tmp memory
  DWORD red;
  SetFilePointer((HANDLE)file, 0, NULL, FILE_BEGIN);
  ReadFile((HANDLE)file, (LPVOID)&fline, sizeof(FIRST_LINE_T), &red, NULL);
  //correct dif between first and others lines
  SetFilePointer((HANDLE)file, LINE_T_SIZE, NULL, FILE_BEGIN);

  //get info from file
  if (!fgets(fileline.info, INFO_SIZE, stdin)) {
    return;
  }

  //update info
  fline.exists++;
  fline.lastId++;
```

```
  fline.fileSize = GetFileSize((HANDLE)file, NULL);

  //apply first line
  SetFilePointer((HANDLE)file, 0, NULL, FILE_BEGIN);
  WriteFile((HANDLE)file, (LPVOID)&fline, LINE_T_SIZE, &red, NULL);

  Veronika Abarnikova, [09.05.20 00:26]
    //struct init
    fileline.id = fline.lastId;

  //get time
  SYSTEMTIME st;
  GetSystemTime(&st); // gets current time
  SystemTimeToFileTime(&st, &fileline.time); // converts to file time format

  //changed are 0 already

  //find free row
  FindLine(FREE_ROW_ID);
  WriteFile((HANDLE)file, (LPVOID)&fileline, LINE_T_SIZE, &red, NULL);
}

void ShowInformationFile() {

  DWORD count = 0;
  FIRST_LINE_T fline = {};
  //set file pointer
  SetFilePointer((HANDLE)file, 0, NULL, FILE_BEGIN);
  //read file
  ReadFile((HANDLE)file, (LPVOID)&fline, sizeof(FIRST_LINE_T), &count, NULL);
  //correct dif between first and others lines
  SetFilePointer((HANDLE)file, LINE_T_SIZE, NULL, FILE_BEGIN);
  BOOL result = FALSE;
  LINE_T line = {};
  while (TRUE) {
    result = ReadFile((HANDLE)file, (LPVOID)&line, LINE_T_SIZE, &count, NULL);

    if (result & count == 0) {
      break;
    }

    if (line.id == FREE_ROW_ID) {
      continue;
    }

    SYSTEMTIME st = {};
    FileTimeToSystemTime(&line.time, (LPSYSTEMTIME)&st);

    cout << "#" << line.id << endl;
    cout << "CreatedTime: " << st.wHour << ":" << st.wMinute
      << " " << st.wDay << "." << st.wMonth << "." << st.wYear << endl;
    cout << "Message: " << line.info << endl;
    cout << "Number of change : " << line.changed << " times" << endl;
    cout << "\n\t***\n";
  }

  cout << "\n-----------------" << endl;
  cout << "Total: " << fline.exists << " records, " << fline.fileSize << " bytes." << endl;
}
//data initialize
void InitData() {
  LPSTR bufer = new CHAR[LINE_T_SIZE];
  memset(bufer, 0, LINE_T_SIZE);
  DWORD written;
  BOOL res = WriteFile((HANDLE)file, bufer, LINE_T_SIZE, &written, NULL);
```

```
delete[](bufer);
    }
```

```
C:\Users\Veronika>C:\Users\Veronika\Desktop\FileWorkLab2\x64\Debug\FileAccess.exe -i
Input of info:
hello
i am
Veronika
```

Рисунок 1 – Ввод инф в файл.

```
C:\Users\Veronika>C:\Users\Veronika\Desktop\FileWorkLab2\x64\Debug\FileAccess.exe -s
#1
Time: 21:48 8.5.2020
Message: hello

Changed: 0 times

        ***
#2
Time: 21:48 8.5.2020
Message: i am

Changed: 0 times

        ***
#3
Time: 21:48 8.5.2020
Message: Veronika

Changed: 0 times

        ***
```

Рисунок 2 – Вывод содержимого.

```
C:\Users\Veronika>C:\Users\Veronika\Desktop\FileWorkLab2\x64\Debug\FileAccess.exe -m 3 change messag
Input of new info: Vitaliy
```

Рисунок 3 – Модификация записи.

```
#3
CreatedTime: 21:48 8.5.2020
Message: Vitaliy

Number of changes: 1 times
```

Рисунок 4 – Вывод содержимого.

```
C:\Users\Veronika>C:\Users\Veronika\Desktop\FileWorkLab2\x64\Debug\FileAccess.exe -d 1
```

Рисунок 5 – Удаление первой записи.

```
#2
CreatedTime: 21:48 8.5.2020
Message: i am

Number of changes: 0 times

        ***
#3
CreatedTime: 21:48 8.5.2020
Message: Vitaliy

Number of changes: 1 times

        ***
#4
CreatedTime: 21:54 8.5.2020
Message: Hello

Number of changes: 0 times

        ***
#5
CreatedTime: 21:54 8.5.2020
Message: i am

Number of changes: 0 times
```

Рисунок 6 – Вывод содержимого.

Задача 2.

Написать программу, реализующую функцию файлового менеджера. Программа должна выдавать приглашение на ввод команды. Поддерживаемые команды:

- Сменить директорию
- Распечатать директорию
- Скопировать файл
- Создать директорию
- Удалить файл (пустую директорию)
- Вывести подробную информацию о файле

Код программы

```
#define _CRT_SECURE_NO_WARNINGS
#define _CRT_NON_CONFORMING_WCSTOK
#define _CRT_NON_CONFORMING_SWPRINTFS

#include <windows.h>
#include <string.h>
#include <tchar.h>
#include <iomanip>
#include <iostream>
#include <fcntl.h>
#include <io.h>
```

```
#define MAX_BUFFER 1000
#define N_MEASURES 4
#define MAX_ARGC 3
#define FILE_NAME 40
#define DISP_STEP 15


using namespace std;


//Creating file
void CreateDir(LPTSTR dirPath) {
         if (!*dirPath) throw (INT)2;
         if (!CreateDirectory(dirPath, NULL)) {
           wcout << "Sorry, error occured while creating directory." << endl;
           throw GetLastError();
   }
         wcout << "File was created" << endl;
}

//Change directory
void ChangeDir(LPTSTR dirPath){
  if (!*dirPath) throw (INT)2;
  SetCurrentDirectory(dirPath);
  GetCurrentDirectory(MAX_BUFFER, dirPath);
  wcout << "Directory has been changed to " << dirPath << endl;
}

// Copy file
void Copyfile(LPTSTR dirPath, LPTSTR src) {

    if (!*dirPath || !*src) throw (INT)1;
    if (!CopyFile(src, dirPath, FALSE)) {
      wcout << "Sorry, can't copy the file." << endl;
      throw GetLastError();
    }
    wcout << "Successful operation" << endl;
}
//Delete file or Directory
void DeleteFileOrDirectory(LPTSTR dirPath) {

  if (!*dirPath) throw (INT)2;

  DWORD attr = GetFileAttributes(dirPath);

  if (attr & FILE_ATTRIBUTE_DIRECTORY) {
    if (!RemoveDirectory(dirPath)) {
      throw GetLastError();
    }
  }
  else {
    if (!DeleteFile(dirPath)) {
      throw GetLastError();
    }
  }
  wcout << "File or directory was deleted!" << endl;
}
// Directory inf
void DirectoryInformaion(LPTSTR dirPath) {
    dirPath = new TCHAR[MAX_BUFFER];
    GetCurrentDirectory(MAX_BUFFER, dirPath);
    wcout << dirPath << endl;
}
```

```
//Check path
BOOL IsNormal(LPCTSTR path) {
  INT i = 0;
  while (path[i] != TEXT('\0')) {
    if (path[i] == TEXT(':')) {
      return TRUE;
    }
    i++;
  }
  return FALSE;
}

//Decoding information
BOOL SplitInputAndDecode(LPTSTR input, LPTSTR com, LPTSTR dirPath, LPTSTR src, LPTSTR
pathfile) {

  //split input string
  LPWSTR sep = LPWSTR(L" \n");
  LPTSTR token;
  INT argc = 0;
  LPTSTR argv[MAX_ARGC * sizeof(LPWSTR)] = {};
  token = wcstok(input, sep);
  if (!token) return FALSE;
  while (token) {
    argv[argc] = token;
    argc++;
    token = wcstok(NULL, sep);
  }

  wcscpy(com, argv[0]);

  if (argc > 1) {
    if (!IsNormal(argv[1])) {
      GetFullPathName(argv[1], MAX_BUFFER, dirPath, &pathfile);
    }
    else {
      wcscpy(dirPath, argv[1]);
    }
  }
  if (argc > 2) {
    if (!IsNormal(argv[2])) {
      GetFullPathName(argv[2], MAX_BUFFER, src, &pathfile);
    }
    else {
      wcscpy(src, argv[2]);
    }
  }
  return TRUE;
}
//File inf
void FileInf(LPTSTR dirPath) {
        if (!*dirPath) throw (INT)2;
        WIN32_FIND_DATA info;
        if (FindFirstFile(dirPath, &info) == INVALID_HANDLE_VALUE) throw (DWORD)2;
        wcout << "File information:\n" << endl;

        //if it's a directory
        if (info.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
          wcout << "<directory>" << endl;
        }

        wcout << "NameFile: " << info.cFileName << endl;
        wcout << "Attribs mask: " << info.dwFileAttributes << endl;
        wcout << "Alter: " << info.cAlternateFileName << endl;
```

```
        unsigned long long size = (unsigned long long)info.nFileSizeHigh << 32 |
info.nFileSizeLow;
        LPCSTR measures[] = { "B", "KB" ,"MB" ,"GB" };
        INT i = 0;
        while (size >> 10 * ++i);
        i--;

        wcout << "SizeFile: " << (size >> 10 * i) << " " << measures[i] << endl;

        SYSTEMTIME stime;
        FileTimeToSystemTime(&info.ftCreationTime, &stime);

        wcout << "Created: " << stime.wHour << ":" << stime.wMinute << " "
          << stime.wDay << "." << stime.wMonth << "." << stime.wYear << endl;

}

void PrDirectory(LPCTSTR cdir) {

  LPTSTR dir = new TCHAR[MAX_BUFFER];
  memcpy(dir, cdir, MAX_BUFFER * sizeof(TCHAR));

  wcscat(dir, TEXT("/*.*"));
  WIN32_FIND_DATA fd;
  HANDLE hFind = ::FindFirstFile(dir, &fd);
  if (hFind != INVALID_HANDLE_VALUE) {

    do {
      // read all (real) files in current folder
      // , delete '!' read other 2 default folder . and ..
      //do printing
      wcout << left << setw(FILE_NAME) << fd.cFileName << " ";

      unsigned long long sizeOfFile = (unsigned long long)fd.nFileSizeHigh << 32 |
fd.nFileSizeLow;
      LPCTSTR measures[] = { TEXT("B"), TEXT("KB") ,TEXT("MB") ,TEXT("GB") };
      INT i = 0;
      while (sizeOfFile >> 10 * ++i);
      i--;

      wcout << left << setw(DISP_STEP);
      if (!(fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)) {

        TCHAR buf[50];
        memset(buf, 0, 100);
        swprintf(buf, TEXT("%d "), (sizeOfFile >> 10 * i));
        wcscat(buf, measures[i]);
        wcout << buf;
      }
      else {
        wcout << "<dir>";
      }

      SYSTEMTIME stime;
      FileTimeToSystemTime(&fd.ftCreationTime, &stime);

      wcout<< stime.wDay << "." << stime.wMonth << "." << stime.wYear;
      wcout << endl;

    } while (::FindNextFile(hFind, &fd));
    FindClose(hFind);
  }
  else {
    wcout << "Error code  :" << GetLastError() << endl;
  }
```

```
  delete dir;
}
//Print directory
void PrintDir(LPTSTR dirPath) {
  if (!GetCurrentDirectory(MAX_BUFFER, dirPath)) {
    throw;
  }
  PrDirectory(dirPath);
}
int _tmain(INT argc, TCHAR** argv)
{
  // enabling all Unicode chars in console
  _setmode(_fileno(stdout), _O_U16TEXT);

  LPTSTR dirPath = new TCHAR[MAX_BUFFER], src = new TCHAR[MAX_BUFFER], pfile = new
TCHAR[MAX_BUFFER];
  LPTSTR com = new TCHAR[MAX_BUFFER];
  LPTSTR input = new TCHAR[MAX_BUFFER];
  int operation;
  memset(src, 0, MAX_BUFFER * sizeof(TCHAR));
  memset(dirPath, 0, MAX_BUFFER * sizeof(TCHAR));
  memset(input, 0, MAX_BUFFER * sizeof(TCHAR));
  memset(com, 0, MAX_BUFFER * sizeof(TCHAR));
  memset(pfile, 0, MAX_BUFFER * sizeof(TCHAR));
  wcout << "\t <File Manager>" << endl;
  wcout << " Create dirirectory : mkdir name  " << endl;
  wcout << " Change current directory : cd dirPath " << endl;
  wcout << " Information about directory : infdir" << endl;
  wcout << " Print current directory : ls  " << endl;
  wcout << " Copy file : cp  dirPath src " << endl;
  wcout << " Delete file or empty directory : rm name  " << endl;
  wcout << " Full info about file : inffile name  " << endl;
  while (TRUE) {
    try {
      int k =0 ;
      if (!fgetws(input, MAX_BUFFER, stdin)) {
        continue;
      }
      if (!SplitInputAndDecode(input, com, dirPath, src, pfile)) {
        continue;
      }

       int i = 0;
       string strCommand;
      while (com[i]!= '\0')
      {
        strCommand += com[i];
        i++;
      }
      if (strCommand == "mkdir") {
        CreateDir(dirPath);
        continue;
      }
      else if (strCommand == "cd") {
        ChangeDir(dirPath);
      }
      else if (strCommand == "ls") {

        DirectoryInformaion(dirPath);
      }
      else if (strCommand == "cp") {
        Copyfile(dirPath, src);
      }
      else if (strCommand == "rm") {
        DeleteFileOrDirectory(dirPath);
```

```
    }
    else if (strCommand == "infdir") {
      PrintDir(dirPath);
    }
    else if (strCommand == "inffile") {
      FileInf(dirPath);
    }
    else {
      wcout << L"Command doesn't exist." << endl;
    }
  }
  catch (INT e) {
    switch (e) {
    case 1:
      wcout << "Source folder not  found." << endl;
      break;
    case 2:
      wcout << "Destination folder not found." << endl;
      break;
    }
  }

  catch (DWORD e) {
    LPTSTR message = new TCHAR[MAX_BUFFER];
    FormatMessage(
      // use system message tables to retrieve error text
      FORMAT_MESSAGE_FROM_SYSTEM
      // allocate buffer on local heap for error text
      | FORMAT_MESSAGE_ALLOCATE_BUFFER
      | FORMAT_MESSAGE_IGNORE_INSERTS,
      NULL, e,
      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
      (LPWSTR)&message,
      0, NULL);
    wcout << L"Error: " << message << endl;
  }
}


getchar();
return 0;
```

```
        <File Manager>
Create dirirectory : mkdir name
Change current directory : cd dirPath
Information about directory : infdir
Print current directory : ls
Copy file : cp  dirPath src
Delete file or empty directory : rm name
Full info about file : inffile name
mkdir  word
File was created
```

Рисунок 1 – Создать директорию.

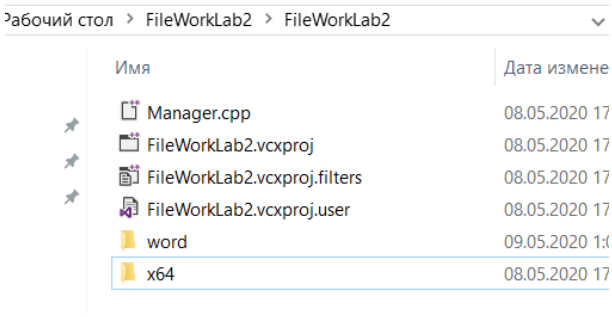

Рисунок 2 – Создать директорию.

```
cd C:\Users\Veronika\Desktop\FileWorkLab2
Directory has been changed to C:\Users\Veronika\Desktop\FileWorkLab2
```

Рисунок 3 – Сменить директорию.

```
infdir
.                                       <dir>          8.5.2020
..                                      <dir>          8.5.2020
.vs                                     <dir>          8.5.2020
FileAccess                              <dir>          8.5.2020
fileData.txt                            0 B            8.5.2020
FileWorkLab2                            <dir>          8.5.2020
FileWorkLab2.sln                        2 KB           8.5.2020
new                                     <dir>          8.5.2020
x64                                     <dir>          8.5.2020
```

Рисунок 4 – информация про директорию.

```
ls
C:\Users\Veronika\Desktop\FileWorkLab2
```

Рисунок 5 – Текущая директория.

```
rm new
File or directory was deleted!
```

Рисунок 6 – Удаление директориии.

```
infdir
.                                       <dir>          8.5.2020
..                                      <dir>          8.5.2020
.vs                                     <dir>          8.5.2020
FileAccess                              <dir>          8.5.2020
fileData.txt                            0 B            8.5.2020
FileWorkLab2                            <dir>          8.5.2020
FileWorkLab2.sln                        2 KB           8.5.2020
x64                                     <dir>          8.5.2020
```

Рисунок 7 – Удаление директориии.

```
NameFile: fileData.txt
Attribs mask: 32
Alter:
SizeFile: 0 B
```

Рисунок 8 – Информация о файле.