МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп'ютерних систем та інфокомунікацій

Кафедра комп'ютерних систем, мереж і кібербезпеки

Лабораторна робота

з Системного програмування
(назва дисципліни)

на тему: «Вивчення системних викликів Win32 API роботи з файлами.»

Виконала: студентка 3-го курсу групи №525ст2
напряму підготовки (спеціальності)
123-«Комп'ютерна інженерія»
(шифр і назва напряму підготовки (спеціальності))

Коваленко Я.О.
(прізвище й ініціали студента)

Прийняв: асистент каф.503
Мозговий М.В.
(посада, науковий ступінь, прізвище й ініціали)

Національна шкала: _____
Кількість балів: _____
Оцінка: ECTS _____

Харків – 2020

Цель работы:

Изучение системных вызовов Win32 API работы с файлами.

Постановка задачи:

Программа 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.

Программа 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

Ход работы:

Код программы:

```
#include "pch.h"
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include "windows.h"
#include <fstream>
#include <string>

using namespace std;

struct record {
unsigned char index;
FILETIME creationTime;
char content[80];
unsigned char accessCount;
};

string SDate(unsigned short day, unsigned short month, unsigned short year) {
string rs = std::to_string(day) + "th of "s;
switch (month)
{
case 1:
	rs += "January";
	break;
case 2:
	rs += "February";
	break;
case 3:
	rs += "March";
	break;
case 4:
	rs += "April";
	break;
case 5:
	rs += "May";
	break;
case 6:
	rs += "June";
	break;
case 7:
	rs += "July";
	break;
case 8:
	rs += "August";
	break;
case 9:
	rs += "September";
	break;
case 10:
	rs += "October";
	break;
case 11:
	rs += "November";
```

```
		break;
	case 12:
		rs += "December";
		break;
	default:
		rs += "???";
		break;
	}
	rs += " "s + std::to_string(year);
	return rs;
	}
	string STime(unsigned short hour, unsigned short minute, unsigned short second) {
	return std::string(std::to_string(hour) + ":"s + std::to_string(minute) + ":"s + std::to_string(second));
	}
	#pragma region UserCases
	void cs_create_file() {
	unsigned short FileSize;
	unsigned char FRecordCount;

	SYSTEMTIME system_time;
	FILETIME file_time;

	FILE *file = fopen("RecordList", "w");
	cout << "Enter number of records: ";
	scanf("%hhu", &FRecordCount);
	unsigned char b = '0';
	fwrite(&b, sizeof(char), 1, file);
	fwrite(&FileSize, sizeof(short), 1, file);
	for (unsigned char i = 0; i < (int)FRecordCount; i++)
	{
		GetSystemTime(&system_time);
		SystemTimeToFileTime(&system_time, &file_time);
		struct record rec = { 1, file_time,"",0 };
		rec.index = i;
		fwrite(&rec, sizeof(struct record), 1, file);
	}
	fseek(file, 0, SEEK_END);
	FileSize = ftell(file);
	fseek(file, sizeof(char), SEEK_SET);
	fwrite(&FileSize, sizeof(short), 1, file);
	fclose(file);
	cout << "A file was created.";
	}
	bool cs_change_file(record *inp_rec, unsigned char *ind) {

	FILE *file = fopen("RecordList", "r+");
	SYSTEMTIME local_time;

	bool search_end_trigger = false;
	cout << "Enter record's Index to access it: ";
	scanf("%hhu", ind);
```

```
	fseek(file, sizeof(char) + sizeof(short), SEEK_SET);
	while (!search_end_trigger && fread(inp_rec, sizeof(struct record), 1, file))
	{
		if (inp_rec->index == *ind) {
			search_end_trigger = true;
			cout << "Requested record was found.\n";
			if ((string)inp_rec->content == "") {
				cout << "No content was found\n";
			}
			else {
				cout << "Content: " << inp_rec->content << "\n";
			}

			FileTimeToSystemTime(&(inp_rec->creationTime), &local_time);
			cout << "Last modified: " << SDate(local_time.wDay, local_time.wMonth,
local_time.wYear) << " ";
			cout << STime(local_time.wHour, local_time.wMinute, local_time.wSecond)
<< endl;
			cout << "Number of modifications: " << (int)inp_rec->accessCount << endl;
		}
	}
	fclose(file);
	return search_end_trigger;
	}
	void cs_delete_record(record inp_rec, unsigned char ind) {
	FILE *file = fopen("RecordList", "r+");
	SYSTEMTIME system_time;
	FILETIME file_time;

	if ((string)inp_rec.content != "") {
		unsigned char b = '0';
		fseek(file, 0, SEEK_SET);
		fread(&b, sizeof(char), 1, file);
		b -= 1;
		fseek(file, 0, SEEK_SET);
		fwrite(&b, sizeof(char), 1, file);
	}
	fseek(file, sizeof(char) + sizeof(short) + (int)ind * sizeof(struct record), SEEK_SET);
	GetSystemTime(&system_time);
	SystemTimeToFileTime(&system_time, &file_time);
	struct record empty = { ind, file_time,"",0 };
	fwrite(&empty, sizeof(struct record), 1, file);
	cout << "Record was deleted.\n";
	fclose(file);
	}
	void cs_change_record(record inp_rec, unsigned char ind) {
	FILE *file = fopen("RecordList", "r+");
	char cont[80];
	cout << "Input new content: ";
	cin.getline(cont, sizeof(cont));
	cin.getline(cont, sizeof(cont));
	cin.clear();
```

```
	cin.ignore(INT_MAX, '\n');
	if ((string)cont != "") {
		unsigned char b = '0';
		fseek(file, 0, SEEK_SET);
		fread(&b, sizeof(char), 1, file);
		b += 1;
		fseek(file, 0, SEEK_SET);
		fwrite(&b, sizeof(char), 1, file);
	}
	strncpy(inp_rec.content, cont, sizeof(inp_rec.content));
	inp_rec.accessCount++;
	fseek(file, sizeof(char) + sizeof(short) + (int)ind * sizeof(struct record), SEEK_SET);
	fwrite(&inp_rec, sizeof(struct record), 1, file);
	fclose(file);
	}
	#pragma endregion UserCases
	int main()
	{
	#pragma region variables
	enum program_mode { cs_exit, create_file, change_file };
	enum record_mode { rcd_exit, delete_record, change_record };
	int mode;
	bool program_exit_trigger;

	#pragma endregion variables
	program_exit_trigger = false;
	do {
		cout << "\nChoose work mode:\n\n1 - create new file;\n2 - change previous one;\n0 -
close program.\n";
		cin >> mode;
		switch (mode)
		{
		case cs_exit: {
			program_exit_trigger = true;
			break;
		}
		case create_file: {
			cs_create_file();
			break;
		}
		case change_file: {
			record inp_rec;
			unsigned char ind;
			bool found_record_trigger = cs_change_file(&inp_rec, &ind);
			if (found_record_trigger) {
				int rcd_mode;
				cout << "Choose operation: 1 - delete record; 2 - change it's content; 0 -
cancel operation;\n";
				cin >> rcd_mode;
				switch (rcd_mode)
				{
				case rcd_exit: {
```

```
                        break;
                }
                case delete_record: {
                        cs_delete_record(inp_rec, ind);
                        break;
                }
                case change_record: {
                        cs_change_record(inp_rec, ind);
                        break;
                }
                default:
                        cout << "No such operation.\n";
                        break;
                }
        }
        else {
                cout << "A record with given Index doesn't exist.\n";
        }
        break;
    }
    default:
        cout << "No such operation. Try again.\n";
        break;
    }
} while (!program_exit_trigger);
}

#include "pch.h"
#include <iostream>
#include <Windows.h>
#include <stdio.h>
#include <tchar.h>
#include <string>
#include <strsafe.h>
#include <string>

#define BUFSIZE MAX_PATH

using namespace std;
void wrap_error() {
LPVOID lpMsgBuf;
LPVOID lpDisplayBuf;
DWORD dw = GetLastError();

FormatMessage(
    FORMAT_MESSAGE_ALLOCATE_BUFFER |
    FORMAT_MESSAGE_FROM_SYSTEM |
    FORMAT_MESSAGE_IGNORE_INSERTS,
    NULL,
    dw,
    MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
    (LPTSTR)&lpMsgBuf,
    0, NULL);
wprintf(L"error %d: %s\n", dw, lpMsgBuf);
}
```

```
string SDate(unsigned short day, unsigned short month, unsigned short year) {
string rs = std::to_string(day) + "th of "s;
switch (month)
{
case 1:
	rs += "January";
	break;
case 2:
	rs += "February";
	break;
case 3:
	rs += "March";
	break;
case 4:
	rs += "April";
	break;
case 5:
	rs += "May";
	break;
case 6:
	rs += "June";
	break;
case 7:
	rs += "July";
	break;
case 8:
	rs += "August";
	break;
case 9:
	rs += "September";
	break;
case 10:
	rs += "October";
	break;
case 11:
	rs += "November";
	break;
case 12:
	rs += "December";
	break;
default:
	rs += "???";
	break;
}
rs += " "s + std::to_string(year);
return rs;
}
string STime(unsigned short hour, unsigned short minute, unsigned short second) {
return std::string(std::to_string(hour) + ":"s + std::to_string(minute) + ":"s + std::to_string(second));
}
void CurrentDirectory(TCHAR* Path) {
TCHAR Buffer[BUFSIZE];
DWORD dwRet;
dwRet = GetCurrentDirectory(BUFSIZE, Buffer);
if (dwRet == 0)
{
	printf("GetCurrentDirectory failed (%d)\n", GetLastError());
```

```
		return;
	}
	if (dwRet > BUFSIZE)
	{
		printf("Buffer too small; need %d characters\n", dwRet);
		return;
	}
	for (int i = 0; i < BUFSIZE; ++i) {
		Path[i] = Buffer[i];
	}
}
int fileExists(TCHAR* file)
{
WIN32_FIND_DATA FindFileData;
HANDLE handle = FindFirstFile(file, &FindFileData);
int found = handle != INVALID_HANDLE_VALUE;
if (found)
{
	FindClose(handle);
}
return found;
}
#pragma region UserCases
void cs_change_directory() {
string new_Path;
cout << "Change directory to: ";
getline(cin, new_Path);
getline(cin, new_Path);
wstring stemp = wstring(new_Path.begin(), new_Path.end());
LPCWSTR error_switch = stemp.c_str();
if (!SetCurrentDirectory(error_switch))
{
	printf("SetCurrentDirectory failed (%d)\n", GetLastError());
}
}
void cs_list_files(TCHAR Path[BUFSIZE]) {
HANDLE hFind = INVALID_HANDLE_VALUE;
WIN32_FIND_DATA found_file_data;
LARGE_INTEGER filesize;
SYSTEMTIME local_time;
FILETIME system_time;

StringCchCat(Path, MAX_PATH, TEXT("\\*"));
hFind = FindFirstFile(Path, &found_file_data);
if (INVALID_HANDLE_VALUE == hFind)
{
	wrap_error();
}
else {
	do
	{
		if (found_file_data.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
		{
			_tprintf(TEXT("  %s   <DIR>\n"), found_file_data.cFileName);
		}
		else
		{
```

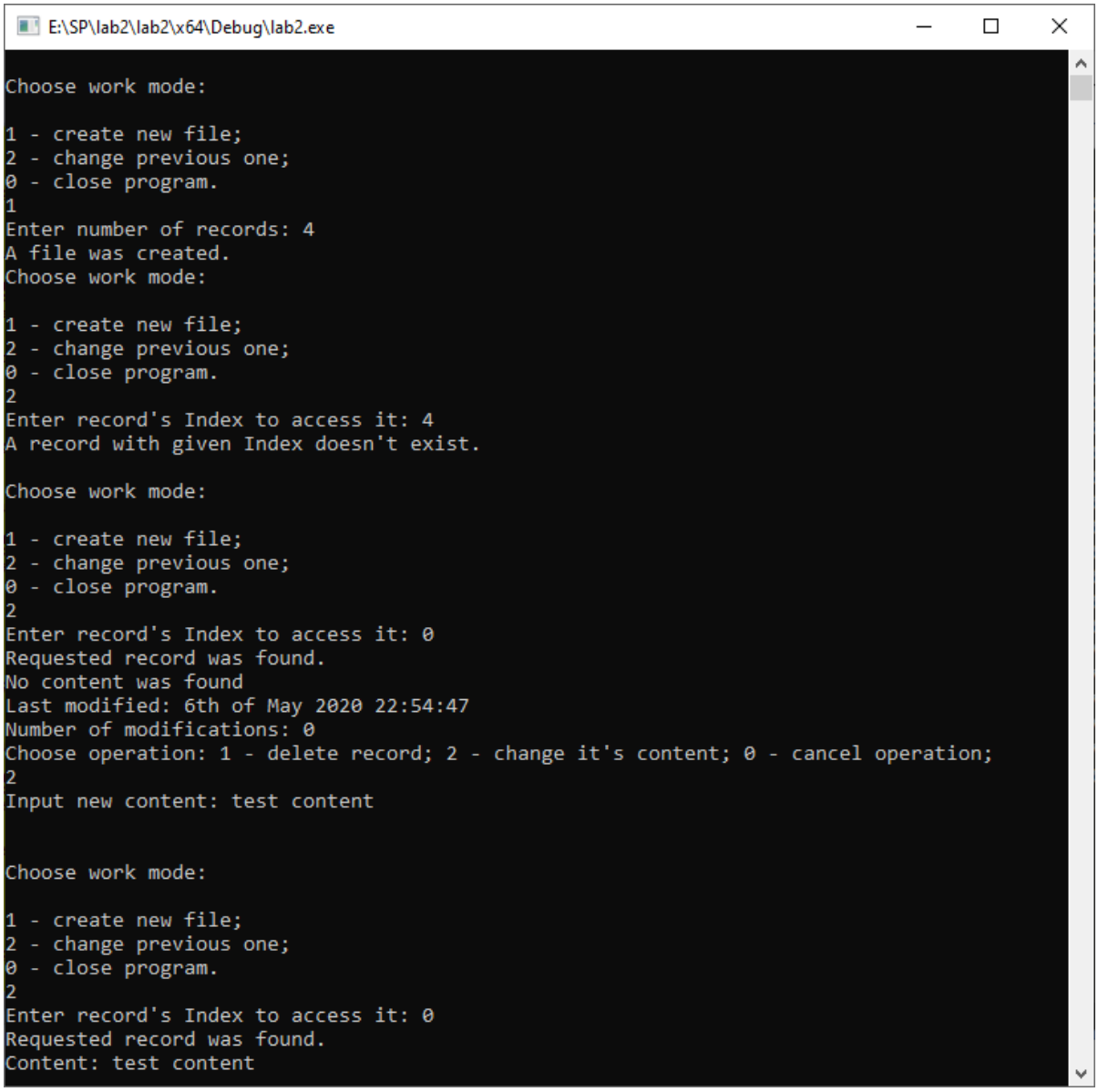
```
                                filesize.LowPart = found_file_data.nFileSizeLow;
                                filesize.HighPart = found_file_data.nFileSizeHigh;
                                FileTimeToLocalFileTime(&found_file_data.ftCreationTime, &system_time);
                                FileTimeToSystemTime(&system_time, &local_time);
                                _tprintf(TEXT("  %s  %ld  bytes  "),  found_file_data.cFileName,
filesize.QuadPart);
                                cout << SDate(local_time.wDay, local_time.wMonth, local_time.wYear) << "
";
                                cout << STime(local_time.wHour, local_time.wMinute, local_time.wSecond)
<< endl;
                        }
                } while (FindNextFile(hFind, &found_file_data) != 0);

                if (GetLastError() != ERROR_NO_MORE_FILES)
                {
                        wrap_error();
                }
                FindClose(hFind);
        }
        }
        void cs_copy_file() {
        string new_Path;

        cout << "Copy file (path): ";
        getline(cin, new_Path);
        getline(cin, new_Path);
        wstring stemp1 = wstring(new_Path.begin(), new_Path.end());
        LPCWSTR source = stemp1.c_str();
        cout << " to ";
        getline(cin, new_Path);
        wstring stemp = wstring(new_Path.begin(), new_Path.end());
        LPCWSTR destination = stemp.c_str();
        if (CopyFile(source, destination, 1) == 0) {
                wrap_error();
        }
        }
        void cs_create_directory() {
        string new_Path;

        cout << "Directory (path): ";
        getline(cin, new_Path);
        getline(cin, new_Path);
        wstring stemp = wstring(new_Path.begin(), new_Path.end());
        LPCWSTR destination = stemp.c_str();
        if (CreateDirectoryW(destination, NULL) == 0) {
                if (GetLastError() == ERROR_ALREADY_EXISTS) {
                        cout << "The specified directory already exists." << endl;
                }
                else {
                        cout << "One or more intermediate directories do not exist" << endl;
                }
        }
        }
        void cs_delete_file_or_empty_directory() {
        string new_Path;

        cout << "Path: ";
```

```
getline(cin, new_Path);
getline(cin, new_Path);
wstring stemp = wstring(new_Path.begin(), new_Path.end());
LPCWSTR destination = stemp.c_str();
DWORD ftyp = GetFileAttributesW(destination);
if (ftyp == INVALID_FILE_ATTRIBUTES) {
	wrap_error();
}
else if (ftyp & FILE_ATTRIBUTE_DIRECTORY) {
	if (RemoveDirectoryW(destination) == 0) {
		wrap_error();
	}
}
else {
	if (DeleteFileW(destination) == 0) {
		if (GetLastError() == ERROR_FILE_NOT_FOUND) {
			cout << "Requested file wasn't found.\n";
		}
		else {
			cout << "Requested file is inaccessible.\n";
		}
	}
}
}
void cs_extended_file_info() {
string new_Path;
HANDLE hFind = INVALID_HANDLE_VALUE;
WIN32_FIND_DATA found_file_data;
LARGE_INTEGER filesize;
SYSTEMTIME local_time;
FILETIME system_time;

cout << "File name (path): ";
getline(cin, new_Path);
getline(cin, new_Path);
wstring stemp = wstring(new_Path.begin(), new_Path.end());
LPCWSTR destination = stemp.c_str();
hFind = FindFirstFileW(destination, &found_file_data);
if (INVALID_HANDLE_VALUE == hFind)
{
	wrap_error();
}
else {
	filesize.LowPart = found_file_data.nFileSizeLow;
	filesize.HighPart = found_file_data.nFileSizeHigh;
	_tprintf(TEXT("Name: %s\n Size: %ld bytes\n"), found_file_data.cFileName,
filesize.QuadPart);
	FileTimeToLocalFileTime(&found_file_data.ftCreationTime, &system_time);
	FileTimeToSystemTime(&system_time, &local_time);
	cout << "Creation Time: ";
	cout << SDate(local_time.wDay, local_time.wMonth, local_time.wYear) << " ";
	cout << STime(local_time.wHour, local_time.wMinute, local_time.wSecond) << endl;
	FileTimeToLocalFileTime(&found_file_data.ftLastAccessTime, &system_time);
	FileTimeToSystemTime(&system_time, &local_time);
	cout << "Last Access Time: ";
	cout << SDate(local_time.wDay, local_time.wMonth, local_time.wYear) << " ";
	cout << STime(local_time.wHour, local_time.wMinute, local_time.wSecond) << endl;
```

```
		FileTimeToLocalFileTime(&found_file_data.ftLastWriteTime, &system_time);
		FileTimeToSystemTime(&system_time, &local_time);
		cout << "Last Write Time: ";
		cout << SDate(local_time.wDay, local_time.wMonth, local_time.wYear) << " ";
		cout << STime(local_time.wHour, local_time.wMinute, local_time.wSecond) << endl;
		cout << "File Attributes: " << found_file_data.dwFileAttributes << endl;
	}
	}
	#pragma endregion UserCases
	int main()
	{
	#pragma region Variables
	enum cases { exit, change_dir, list_files, copy_file, create_dir, delete_file_or_empty_dir,
extended_file_info };
	int mode;
	bool non_program_exit_trigger = true;
	TCHAR Path[BUFSIZE];

	#pragma endregion Variables
	do
	{
		CurrentDirectory(Path);
		_tprintf(TEXT("Current directory: %s\n"), Path);
		cout << "Commands:\n1 - change directory;\n2 - list files in current directory;\n3 - copy
file;\n";
		cout << "4 - create directory;\n5 - delete file (empty directory);\n6 - extended file info;\n0 -
close program;\n";
		cin >> mode;
		switch (mode)
		{
		case exit:
			non_program_exit_trigger = false;
			break;
		case change_dir: {
			cs_change_directory();
			break;
		}
		case list_files: {
			cs_list_files(Path);
			break;
		}
		case copy_file: {
			cs_copy_file();
			break;
		}
		case create_dir: {
			cs_create_directory();
			break;
		}
		case delete_file_or_empty_dir: {
			cs_delete_file_or_empty_directory();
			break;
		}
		case extended_file_info: {
			cs_extended_file_info();
			break;
		}
```

```
        default:
                non_program_exit_trigger = false;
                break;
        }
} while (non_program_exit_trigger);
}
```

Результат работы:



E:\SP\lab2\lab2\x64\Debug\lab2.exe

```
Input new content: test content


Choose work mode:

1 - create new file;
2 - change previous one;
0 - close program.
2
Enter record's Index to access it: 0
Requested record was found.
Content: test content
Last modified: 6th of May 2020 22:54:47
Number of modifications: 1
Choose operation: 1 - delete record; 2 - change it's content; 0 - cancel operation;
1
Record was deleted.

Choose work mode:

1 - create new file;
2 - change previous one;
0 - close program.
2
Enter record's Index to access it: 3
Requested record was found.
No content was found
Last modified: 6th of May 2020 22:54:47
Number of modifications: 0
Choose operation: 1 - delete record; 2 - change it's content; 0 - cancel operation;
2
Input new content: test content2


Choose work mode:

1 - create new file;
2 - change previous one;
0 - close program.
```

```
31 83 01 00 CC CC CC B0  39 1F 7F F9 23 D6 01 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
CC CC CC 01 CC CC CC 20  CD 2A 57 F9 23 D6 01 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
CC CC CC 02 CC CC CC 20  CD 2A 57 F9 23 D6 01 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
CC CC CC 03 CC CC CC 20  CD 2A 57 F9 23 D6 01 74
65 73 74 20 63 6F 6E 74  65 6E 74 32 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 01
CC CC CC
```

E:\SP\lab2\lab2.2\Debug\lab2.2.exe

```
Current directory: E:\SP\lab2\lab2.2\lab2.2
Commands:
1 - change directory;
2 - list files in current directory;
3 - copy file;
4 - create directory;
5 - delete file (empty directory);
6 - extended file info;
0 - close program;
1
Change directory to: E:\Ubuntu\Ubuntu
Current directory: E:\Ubuntu\Ubuntu
Commands:
1 - change directory;
2 - list files in current directory;
3 - copy file;
4 - create directory;
5 - delete file (empty directory);
6 - extended file info;
0 - close program;
2
  .   <DIR>
  ..   <DIR>
  Logs   <DIR>
  Ubuntu.vbox   3848 bytes   25th of September 2019 20:32:44
  Ubuntu.vbox-prev.xml   3859 bytes   25th of September 2019 20:32:44
  Ubuntu.vdi   1612709888 bytes   25th of September 2019 20:32:44
Current directory: E:\Ubuntu\Ubuntu
```

E:\SP\lab2\lab2.2\Debug\lab2.2.exe

```
  Ubuntu.vbox   3848 bytes   25th of September 2019 20:32:44
  Ubuntu.vbox-prev.xml   3859 bytes   25th of September 2019 20:32:44
  Ubuntu.vdi   1612709888 bytes   25th of September 2019 20:32:44
Current directory: E:\Ubuntu\Ubuntu
Commands:
1 - change directory;
2 - list files in current directory;
3 - copy file;
4 - create directory;
5 - delete file (empty directory);
6 - extended file info;
0 - close program;
3
Copy file (path): Ubuntu.vbox
 to PathUbuntu.vbox
Current directory: E:\Ubuntu\Ubuntu
Commands:
1 - change directory;
2 - list files in current directory;
3 - copy file;
4 - create directory;
5 - delete file (empty directory);
6 - extended file info;
0 - close program;
```

Name	Date modified	Type	Size
Logs	9/25/2019 8:32 PM	File folder	
Ubuntu.vbox	9/12/2019 2:12 AM	VBOX File	4 KB
Ubuntu.vbox-prev.xml	9/12/2019 2:12 AM	XML Document	4 KB
Ubuntu.vdi	12/22/2019 6:09 PM	VDI File	1,574,912 KB

Logs	9/25/2019 8:32 PM	File folder	
PathUbuntu.vbox	9/12/2019 2:12 AM	VBOX File	4 KB
Ubuntu.vbox	9/12/2019 2:12 AM	VBOX File	4 KB
Ubuntu.vbox-prev.xml	9/12/2019 2:12 AM	XML Document	4 KB
Ubuntu.vdi	12/22/2019 6:09 PM	VDI File	1,574,912 KB

Выводы:

В результате выполнения данной лабораторной работы были изучены системныt вызовs Win32 API работы с файлами.