Лабораторная работа №4

студента группы ИТ – 32

Курбатовой Софьи Андреевны

Выполнение:_____ Защита _____

ФАЙЛЫ И КАТАЛОГИ. СИСТЕМНЫЙ РЕЕСТР WINDOWS

Цель работы: получение практических навыков работы с файлами, каталогами и системным реестром в Windows с применением функций Win32 API .

Содержание работы

Вариант 11

1. Разработать в Visual C++ оконное приложение Win32, которое:

– должно иметь возможность создания, чтения и редактирования текстовых файлов. Чтение и редактирование файлов должно осуществляться с помощью асинхронных операций ввода/вывода;

– должно сохранять в файл инициализации размер и положение окна, а также имя последнего редактируемого текстового файла, чтобы использовать их при повторном запуске. Можно также сохранять в файле инициализации и другие параметры приложения (например, параметры шрифта);

2. Разработать в Visual C++ приложение Win32:

– должно для выбранного файла или каталога выводить его имя, атрибуты и размер, а также время его создания, изменения и последнего обращения;

– должно иметь возможность переименования выбранного файла или каталога;

– должно иметь возможность изменять атрибуты выбранного файл или каталога (кроме атрибутов системный и временный файл, а также атрибутов сжатия и шифрования);

– должно сохранять в системном реестре размер и положение окна, а также имя последнего выбранного файла или каталога, чтобы использовать их при повторном запуске. Сохранение в системном реестре должно осуществляться в разделе HKCU\Software\IT-311;

3. Разработать в Visual C++ приложение Win32, которое должно выполнять указанную в варианте задания операцию с файлами и каталогами. Каталоги должны перемещаться, копироваться и удаляться вместе с вложенными файлами и каталогами.

4. Разработать в Visual C++ приложение Win32, которое должно выводить следующую информацию из системного реестра:

– список установленных программ HKLM\ SOFTWARE\ MICROSOFT\ WINDOWS\ CURRENTVERSION\ UNISTALL;

– список программ автозапуска HKLM\ SOFTWARE\ MICROSOFT\ WINDOWS\CurrentVersion\Run , HKCU\ SOFTWARE\ MICROSOFT\ WINDOWS\CurrentVersion\Run

5. Протестировать работу разработанных приложений на компьютере под управлением Windows. Результаты отразить в отчете.

6. Включить в отчет исходный программный код и выводы о проделанной работе.

11,26	Копирование файлов и каталогов

Рис. 4.1. задание для варианта 11

Ход работы

1. В Visual Studio было разработано оконное приложение win32, которое имитирует работу текстового редактора с возможностью ввода текста, изменения его положения в рабочей области: выравнивание по левой границе, правой границе, центру. Среди реализованных функций также присутствуют: сохранение изменений, сохранение нового файла с расширением .txt, открытие ранее созданного файла. В дополнение к этому, при повторном запуске приложения открывается ранее редактируемый файл. Это возможно благодаря наличию файла инициализации, который формируется при первом открытии приложения и при повторных запусках обновляется новыми значениями.

Листинг 1. Оконное приложение FileEditor

```
#include "FileEditorHeader.h"

#define IDC_EDIT_TEXT		2001
#pragma comment(linker,"\"/manifestdependency:type='win32' \
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \
processorArchitecture='*' publicKeyToken='6595b64144ccf1df' language='*'\"")


int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpszCmdLine, int nCmdShow)
{
	HINSTANCE relib = LoadLibrary(TEXT("riched32.dll"));	//load the dll don't forget this
								//and don't forget to free it (see wm_destroy)
	if (relib == NULL)
		MessageBox(NULL, TEXT("Failed to load riched32.dll!"), TEXT("Error"),
MB_ICONEXCLAMATION);
	HACCEL hAccel = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR_ACCELERATOR1));

	WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

	wcex.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
	wcex.lpfnWndProc = MainWindowProc; // оконная процедура
	wcex.hInstance = hInstance;
	wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
	wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground = (HBRUSH)(COLOR_BTNFACE + 2);
	wcex.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
	wcex.lpszClassName = TEXT("MainWindowClass"); // имя класса
	wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

	if (0 == RegisterClassEx(&wcex)) // регистрируем класс
	{
		return -1; // завершаем работу приложения
	}

	LoadLibrary(TEXT("ComCtl32.dll"));//для элементов общего пользования

	TCHAR InitFN[MAX_PATH];//имя файла инициализации
		{
			GetModuleFileName(NULL, InitFN, MAX_PATH);
			LPTSTR str = _tcsrchr(InitFN, TEXT('.'));
			if (NULL != str) str[0] = TEXT('\0');
			StringCchCat(InitFN, MAX_PATH, TEXT(".ini"));
		}
	// загружаем параметры приложения из файла инициализации
	LoadProfile(InitFN);

	HWND hWnd = CreateWindowEx(0, TEXT("MainWindowClass"), TEXT("Process"),
WS_OVERLAPPEDWINDOW,
```

```
		WindowPosition.x, WindowPosition.y, WindowSize.cx, WindowSize.cy, NULL, NULL,
hInstance, NULL);

	if (NULL == hWnd)
	{	return -1; }// завершаем работу приложения
	ShowWindow(hWnd, nCmdShow); // отображаем главное окно
	MSG  msg;
	BOOL Ret;
	for (;;)
	{
		// определяем наличие сообщений в очереди
		while (!PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE))
		{
			OnIdle(hWnd);
		}
		// извлекаем сообщение из очереди
		Ret = GetMessage(&msg, NULL, 0, 0);
		if ( Ret == FALSE )
		{
			break; // получено WM_QUIT, выход из цикла
		}
		else if (!TranslateAccelerator(hWnd, hAccel, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}
	SaveProfile(InitFN);//сохранение параметров
	return (int)msg.wParam;
}
LRESULT CALLBACK MainWindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
	switch (uMsg)
	{
		HANDLE_MSG(hwnd, WM_CREATE, OnCreate);
		HANDLE_MSG(hwnd, WM_COMMAND, OnCommand);

	case WM_SIZE:
	{
			HWND hwndCtl = GetDlgItem(hwnd, IDC_EDIT_TEXT);
			MoveWindow(hwndCtl, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE); // изменяем
размеры поля ввода
	}

		break;
	case WM_DESTROY:
	{
		if (INVALID_HANDLE_VALUE != hFile)
		{
			FinishIo(&ovlWrite);// ожидаем завершения операции ввода/вывода
			CloseHandle(hFile), hFile = INVALID_HANDLE_VALUE;// закрываем дескриптор
файла
		}

		if (NULL != hFont)
			DeleteObject(hFont), hFont = NULL;// удаляем созданный шрифт
		PostQuitMessage(0); // отправляем сообщение WM_QUIT
	}break;
	case WM_CLOSE:
		RECT rect;
		GetWindowRect(hwnd, &rect);

		WindowPosition.x = rect.left;
		WindowPosition.y = rect.top;

		WindowSize.cx = rect.right - rect.left;
		WindowSize.cy = rect.bottom - rect.top;
```

```
			DestroyWindow(hwnd); // уничтожаем окно
			break;
		}
		return DefWindowProc(hwnd, uMsg, wParam, lParam);
}


void OnIdle(HWND hwnd)
{
		if (NULL != lpBuffReWri)
		{
			if (TryFinishIo(&ovlRead) != FALSE) // асинхронное чтение данных из файла завершено
			{
				if (ERROR_SUCCESS == ovlRead.Internal) // чтение завершено успешно
				{
					WORD bom = *(LPWORD)lpBuffReWri; // маркер последовательности байтов
					if (0xFEFF == bom) // Unicode-файл
					{
						LPWSTR lpszText = (LPWSTR)(lpBuffReWri + sizeof(WORD)); //
Unicode-строка
						// вычисляем длину Unicode-строки
						DWORD cch = (ovlRead.InternalHigh - sizeof(WORD)) /
sizeof(WCHAR);
						lpszText[cch] = L'\0';// задаём нуль-символ в конце строки

						SetDlgItemTextW(hwnd, IDC_EDIT_TEXT, lpszText);// копируем
Unicode-строку в поле ввода
					}
					else // ANSI-файл
					{
						lpBuffReWri[ovlRead.InternalHigh] = '\0';// задаём нуль-символ
в конце строки
						SetDlgItemTextA(hwnd, IDC_EDIT_TEXT, lpBuffReWri);// копируем
ANSI-строку в поле ввода
					}
				}
				delete[] lpBuffReWri, lpBuffReWri = NULL;	// освобождаем выделенную
память
			}
			else if (TryFinishIo(&ovlWrite) != FALSE) // асинхронная запись данных в файл
завершена
			{
				if (ERROR_SUCCESS == ovlWrite.Internal) // запись завершена успешно
				{
					// заставим операционную систему записать данные в файл не дожидаясь
его закрытия
					FlushFileBuffers(hFile);
				}
				delete[] lpBuffReWri, lpBuffReWri = NULL;// освобождаем выделенную память
			}
		}
}
BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct)
{
		// создаёи поле ввода для редактирования текста
		DWORD Styles = WS_VISIBLE | WS_CHILD | WS_BORDER | WS_HSCROLL | WS_VSCROLL | ES_NOHIDESEL
| ES_AUTOVSCROLL | ES_MULTILINE | ES_SAVESEL| ES_SUNKEN;

		// Создаем орган управления Rich Edit
		HWND hwndCtl = CreateWindowEx(0, TEXT( "RICHEDIT"), TEXT(""), Styles, 0, 0, 100, 100,
hwnd, (HMENU)IDC_EDIT_TEXT, lpCreateStruct->hInstance, NULL);

		if (hwndCtl == NULL)
			return FALSE;

		// Передаем фокус ввода органу управления Rich Edit
```

```
	SetFocus(hwndCtl);
//установка значений font по умолчанию при открытии приложения
	logFont.lfCharSet = DEFAULT_CHARSET; //значение по умолчанию
	logFont.lfPitchAndFamily = DEFAULT_PITCH; //значения по умолчанию
	wcscpy_s(logFont.lfFaceName, L"Times New Roman"); //копируем в строку название шрифта
	logFont.lfHeight = 20; //высота
	logFont.lfWidth = 10; //ширина
	logFont.lfWeight = 40; //толщина
	logFont.lfEscapement = 0; //шрифт без поворота

	// создаём шрифт
	hFont = CreateFontIndirect(&logFont);
	static HWND hEdit = GetDlgItem(hwnd, IDC_EDIT_TEXT);
	if (NULL != hFont)
	{
		// устанавливаем шрифт для поля ввода
		SendMessage(hwndCtl, WM_SETFONT, (WPARAM)hFont, (LPARAM)TRUE);
	}

	// открываем последний редактируемый текстовый фал
	if (OpenFileAsync(hwndCtl) != FALSE)
	{
		// задаём заголовок главного окна
		SetWindowText(hwnd, FileName);
		SendMessage(hEdit, EM_SETPARAFORMAT, 0, (LPARAM)&pf);//выравнивание текста
	}
	else
	{
		FileName[0] = _T('\0');// очищаем имя редактируемого текстового файла
		SetWindowText(hwnd, TEXT("Безымянный"));// задаём заголовок главного окна
	}

	return TRUE;
}
//команды приложения
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
	static HWND hEdit = GetDlgItem(hwnd, IDC_EDIT_TEXT);
	CHOOSEFONT choosef = { sizeof(CHOOSEFONT) };
	HDC hDC;


	switch (id)
	{
	case ID_NEW_FILE: // Создать
	{
		if (INVALID_HANDLE_VALUE != hFile)
		{

			FinishIo(&ovlWrite);// ожидаем завершения операции ввода/вывода
			CloseHandle(hFile), hFile = INVALID_HANDLE_VALUE;
		}
		Edit_SetText(hEdit, NULL);// удаляем текст из поля ввода

		FileName[0] = _T('\0');
		SetWindowText(hwnd, TEXT("Безымянный"));
	}
	break;

	case ID_OPEN: // Открыть
	{
		OPENFILENAME openfile = { sizeof(OPENFILENAME) };

		openfile.hInstance = GetWindowInstance(hwnd);
		openfile.lpstrFilter = TEXT("Текстовые документы (*.txt)\0*.txt\0");
		openfile.lpstrFile = FileName;
		openfile.nMaxFile = _countof(FileName);
```

```
			openfile.lpstrTitle = TEXT("Открыть");
			openfile.Flags = OFN_EXPLORER | OFN_ENABLESIZING | OFN_FILEMUSTEXIST;
			openfile.lpstrDefExt = TEXT("txt");

			if (GetOpenFileName(&openfile) != FALSE)
			{
				if (OpenFileAsync(hEdit) != FALSE) // открываем файл
				{
					SetWindowText(hwnd, FileName);// задаём заголовок главного окна
				}
				else
				{
					MessageBox(NULL, TEXT("Не удалось открыть текстовый файл."), NULL,
MB_OK | MB_ICONERROR);
					FileName[0] = _T('\0');
					SetWindowText(hwnd, TEXT("Безымянный"));
				}
			}
		}
		break;

		case ID_SAVE: // Сохранить
			if (SaveFileAsync(hEdit) != FALSE) // сохраняем файл
			{
				break;
			}
		case ID_SAVE_AS: // Сохранить как
		{
			OPENFILENAME savefile = { sizeof(OPENFILENAME) };

			savefile.hInstance = GetWindowInstance(hwnd);
			savefile.lpstrFilter = TEXT("Текстовые документы (*.txt)\0*.txt\0");
			savefile.lpstrFile = FileName;
			savefile.nMaxFile = _countof(FileName);
			savefile.lpstrTitle = TEXT("Сохранить как");
			savefile.Flags = OFN_EXPLORER | OFN_ENABLESIZING | OFN_CREATEPROMPT |
OFN_OVERWRITEPROMPT;
			savefile.lpstrDefExt = TEXT("txt");

			if (GetSaveFileName(&savefile) != FALSE)
			{
				if (SaveFileAsync(hEdit, TRUE) != FALSE) // пересохраняем файл
				{
					SetWindowText(hwnd, FileName);// задаём заголовок главного окна
				}
				else
				{
					MessageBox(NULL, TEXT("Не удалось сохранить текстовый файл."), NULL,
MB_OK | MB_ICONERROR);
				}
			}
		}
		break;

		case ID_EXIT:
			SendMessage(hwnd, WM_CLOSE, 0, 0);
			break;

		case ID_UNDO: // Отменить
		{

			Edit_Undo(hEdit);// отменяем последне изменения в поле ввода
			SetFocus(hEdit);// передаём фокус клавиатуы в поле ввода
		}
		break;

		case ID_SELECT_ALL: // Выделить все
```

```
	{
		Edit_SetSel(hEdit, 0, -1);// выделяем текст в поле ввода
		SetFocus(hEdit);// передаём фокус клавиатуы в поле ввода
	}
	break;


case IDM_EDCUT:
		SendMessage(hEdit, WM_CUT, 0, 0);
		break;

	case IDM_EDCOPY:
		SendMessage(hEdit, WM_COPY, 0, 0);
		break;

	case IDM_EDPASTE:
		SendMessage(hEdit, WM_PASTE, 0, 0);
		break;

	// Устанавливаем выравнивание параграфа по правой границе
	// окна органа управления Rich Edit
	case ID_FORMAT_PARAGRAPH_RIGHT:
		{
		pf.cbSize = sizeof(pf);
		pf.dwMask = PFM_ALIGNMENT;
		pf.wAlignment = PFA_RIGHT;
		SendMessage(hEdit, EM_SETPARAFORMAT, 0, (LPARAM)&pf);
		break;
		}

	// Выполняем центровку текущего параграфа
	case ID_FORMAT_PARAGRAPH_CENTER:
	{
		pf.cbSize = sizeof(pf);
		pf.dwMask = PFM_ALIGNMENT;
		pf.wAlignment = PFA_CENTER;
		SendMessage(hEdit, EM_SETPARAFORMAT, 0, (LPARAM)&pf);

		break;
	}
	// Устанавливаем выравнивание параграфа по левой границе
// окна органа управления Rich Edit
	case ID_FORMAT_PARAGRAPH_LEFT:
		{
			pf.cbSize = sizeof(pf);
			pf.dwMask = PFM_ALIGNMENT;
			pf.wAlignment = PFA_LEFT;

			// Изменяем тип выравнивания текущего параграфа
			SendMessage(hEdit, EM_SETPARAFORMAT, 0, (LPARAM)&pf);
			break;
		}
	}
}

void LoadProfile(LPCTSTR lpFileName)
{
	// загружаем положение и размер окна

	WindowPosition.x = GetPrivateProfileInt(TEXT("Window"), TEXT("X"), CW_USEDEFAULT,
lpFileName);
	WindowPosition.y = GetPrivateProfileInt(TEXT("Window"), TEXT("Y"), 0, lpFileName);

	WindowSize.cx = GetPrivateProfileInt(TEXT("Window"), TEXT("Width"), CW_USEDEFAULT,
lpFileName);
	WindowSize.cy = GetPrivateProfileInt(TEXT("Window"), TEXT("Height"), 600, lpFileName);
```

```
	/*Dont init without this items*/
	pf.cbSize = sizeof(pf);
	pf.dwMask = PFM_ALIGNMENT;

	//загрузка типа выравнивания
	if (GetPrivateProfileInt(TEXT("Paraformat"), TEXT("wAlignment"), 0, lpFileName)==3)
	{
		pf.wAlignment = PFA_CENTER;
	}
	if (GetPrivateProfileInt(TEXT("Paraformat"), TEXT("wAlignment"), 0, lpFileName) == 2)
	{
			pf.wAlignment = PFA_RIGHT;
	}
	if (GetPrivateProfileInt(TEXT("Paraformat"), TEXT("wAlignment"), 0, lpFileName) == 1)
	{
		pf.wAlignment = PFA_LEFT;
	}
	// загружаем имя последнего редактируемого текстового файла
	GetPrivateProfileString(TEXT("File"), TEXT("Filename"), NULL, FileName, MAX_PATH,
lpFileName);
}

void SaveProfile(LPCTSTR lpFileName)
{
	TCHAR szString[10];

	// сохраняем положение и размер окна

	StringCchPrintf(szString, 10, TEXT("%d"), WindowPosition.x);
	WritePrivateProfileString(TEXT("Window"), TEXT("X"), szString, lpFileName);

	StringCchPrintf(szString, 10, TEXT("%d"), WindowPosition.y);
	WritePrivateProfileString(TEXT("Window"), TEXT("Y"), szString, lpFileName);

	StringCchPrintf(szString, 10, TEXT("%d"), WindowSize.cx);
	WritePrivateProfileString(TEXT("Window"), TEXT("Width"), szString, lpFileName);

	StringCchPrintf(szString, 10, TEXT("%d"), WindowSize.cy);
	WritePrivateProfileString(TEXT("Window"), TEXT("Height"), szString, lpFileName);

	//сохраняем параметры выравнивания текста
	/*	warning! now this line describe all text in program.
		so, if you have first string with PFA_LEFT and second string with PFA_CENTER
		saveprofile write wAlignment = 3 (it is PFA_CENTER) in .ini file
	*/
	StringCchPrintf(szString, 10, TEXT("%d"), pf.dwMask);
	WritePrivateProfileString(TEXT("Paraformat"), TEXT("dwMask"), szString, lpFileName);

	StringCchPrintf(szString, 10, TEXT("%d"), pf.wAlignment);
	WritePrivateProfileString(TEXT("Paraformat"), TEXT("wAlignment"), szString, lpFileName);

	// сохраняем имя последнего редактируемого текстового файла

	WritePrivateProfileString(TEXT("File"), TEXT("Filename"), FileName, lpFileName);



}

BOOL OpenFileAsync(HWND hwndCtl)
{
	HANDLE hExistingFile = CreateFile(FileName,
		GENERIC_READ | GENERIC_WRITE,//запись и чтние
		FILE_SHARE_READ, //для совместного чтения
		NULL, //защиты нет
		OPEN_EXISTING, //открыть существующий
		FILE_FLAG_OVERLAPPED,//асинхронный доступ к файлу
```

```
		NULL); //шаблона нет

	if (INVALID_HANDLE_VALUE == hExistingFile) // не удалось открыть файл
	{
		CloseHandle(hExistingFile);
		return FALSE;
	}

	Edit_SetText(hwndCtl, NULL);// удаляем текст из поля ввода

	if (INVALID_HANDLE_VALUE != hFile)
	{
		FinishIo(&ovlWrite);// ожидаем завершения операции ввода/вывода
		CloseHandle(hFile);
	}

	hFile = hExistingFile;

	LARGE_INTEGER size;// определяем размер файла
	BOOL bRet = GetFileSizeEx(hFile, &size);

	if ((FALSE != bRet) && (size.LowPart > 0))
	{
		// выделяем память для буфера, в который будет считываться данные из файла
		lpBuffReWri = new CHAR[size.LowPart + 2];

		bRet = ReadAsync(hFile, lpBuffReWri, 0, size.LowPart, &ovlRead);// асинхронное
чтение данных из файла

		if (FALSE == bRet) // возникла ошибка
		{
			delete[] lpBuffReWri, lpBuffReWri = NULL;// освобождаем выделенную память
		}
	}
	return bRet;
}

BOOL SaveFileAsync(HWND hwndCtl, BOOL fSaveAs)
{
	if (fSaveAs != FALSE)
	{
		// создаём и открываем файл для чтения и записи
		HANDLE hNewFile = CreateFile(FileName,
			GENERIC_READ | GENERIC_WRITE,//запись и чтение
			FILE_SHARE_READ, //для совместного чтения
			NULL, //защиты нет
			CREATE_ALWAYS, //создание нового файла
			FILE_FLAG_OVERLAPPED,//асинхронный доступ к файлу
			NULL); //шаблона нет


		if (hNewFile == INVALID_HANDLE_VALUE) // не удалось открыть файл
		{
			CloseHandle(hNewFile);
			return FALSE;
		}

		if (hFile != INVALID_HANDLE_VALUE)
		{
			FinishIo(&ovlWrite);// ожидаем завершения операции ввода/вывода
			CloseHandle(hFile);
		}

		hFile = hNewFile;
	}
	else if (hFile != INVALID_HANDLE_VALUE)
	{
```

```
			FinishIo(&ovlWrite);// ожидаем завершения операции ввода/вывода
		}
		else
		{
			// создаём и открываем файл для чтения и записи
			hFile = CreateFile(FileName,
				GENERIC_READ | GENERIC_WRITE,
				FILE_SHARE_READ,
				NULL,
				CREATE_ALWAYS,
				FILE_FLAG_OVERLAPPED, NULL);

			if (hFile == INVALID_HANDLE_VALUE) // не удалось открыть файл
			{
				CloseHandle(hFile);
				return FALSE;
			}
		}

		LARGE_INTEGER size;// определяем размер текста
		size.QuadPart = GetWindowTextLengthA(hwndCtl);

		BOOL bRet = SetFilePointerEx(hFile, size, NULL, FILE_BEGIN);// изменяем положение
указателя файла

		if (FALSE != bRet)
			bRet = SetEndOfFile(hFile);// устанавливаем конец файла

		if ((FALSE != bRet) && (size.LowPart > 0))
		{

			lpBuffReWri = new CHAR[size.LowPart + 1];	// выделяем память для буфера, из
которого будут записываться данные в файл

			GetWindowTextA(hwndCtl, lpBuffReWri, size.LowPart + 1);		// копируем
ANSI-строку из поля ввода в буффер

			bRet = WriteAsync(hFile, lpBuffReWri, 0, size.LowPart, &ovlWrite);// асинхронная
запись данных в файл

			if (FALSE == bRet)
			{
				delete[] lpBuffReWri, lpBuffReWri = NULL;// освобождаем выделенную память
			}
		}

		return bRet;
}
/*Asynch work*/

BOOL ReadAsync(HANDLE hFile, LPVOID lpBuffer, DWORD dwOffset, DWORD dwSize, LPOVERLAPPED ovl)
{
		// инициализируем структуру OVERLAPPED
		ZeroMemory(ovl, sizeof(ovl));

		ovl->Offset = dwOffset; // младшая часть смещения
		ovl->hEvent = CreateEvent(NULL, FALSE, FALSE, NULL); //событие для оповещения завершения
записи

		// начинаем асинхронную операцию чтения данных из файла
		BOOL bRet = ReadFile(hFile, lpBuffer, dwSize, NULL, ovl);
		DWORD  dwRet = GetLastError();
		if (FALSE == bRet && ERROR_IO_PENDING != dwRet)
		{
			CloseHandle(ovl->hEvent), ovl->hEvent = NULL;
			return FALSE;
		}
```

```
	return TRUE;
}

BOOL WriteAsync(HANDLE hFile, LPCVOID lpBuffer, DWORD dwOffset, DWORD dwSize, LPOVERLAPPED ovl)
{
	// инициализируем структуру OVERLAPPED
	ZeroMemory(ovl, sizeof(ovl));
	ovl->Offset = dwOffset; // младшая часть смещения
	ovl->hEvent = CreateEvent(NULL, FALSE, FALSE, NULL); //событие для оповещения завершения
записи
	// начинаем асинхронную операцию записи данных в файл
	BOOL bRet = WriteFile(hFile, lpBuffer, dwSize, NULL, ovl);
	DWORD  dwRet = GetLastError();
	if (FALSE == bRet && ERROR_IO_PENDING != dwRet)
	{
		CloseHandle(ovl->hEvent), ovl->hEvent = NULL;
		return FALSE;
	}
	return TRUE;
}

BOOL FinishIo(LPOVERLAPPED ovl)
{
	if (NULL != ovl->hEvent)
	{
		// ожидаем завершения операции ввода/вывода
		DWORD dwRes = WaitForSingleObject(ovl->hEvent, INFINITE);

		if (WAIT_OBJECT_0 == dwRes) // операция завершена
		{
			CloseHandle(ovl->hEvent), ovl->hEvent = NULL;
			return TRUE;
		}
	}

	return FALSE;
}


BOOL TryFinishIo(LPOVERLAPPED ovl)
{
	if (NULL != ovl->hEvent)
	{

		DWORD dwResult = WaitForSingleObject(ovl->hEvent, 0);// определяем состояние
операции ввода/вывода

		if (WAIT_OBJECT_0 == dwResult) // операция завершена
		{
			CloseHandle(ovl->hEvent), ovl->hEvent = NULL;
			return TRUE;
		}
	}

	return FALSE;
	}
```

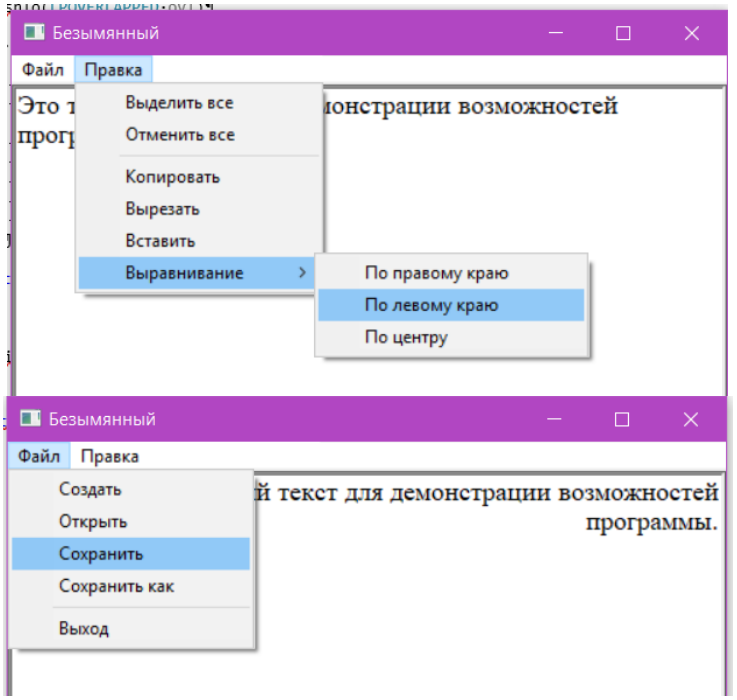

Рис. 4.2. Общий внешний вид

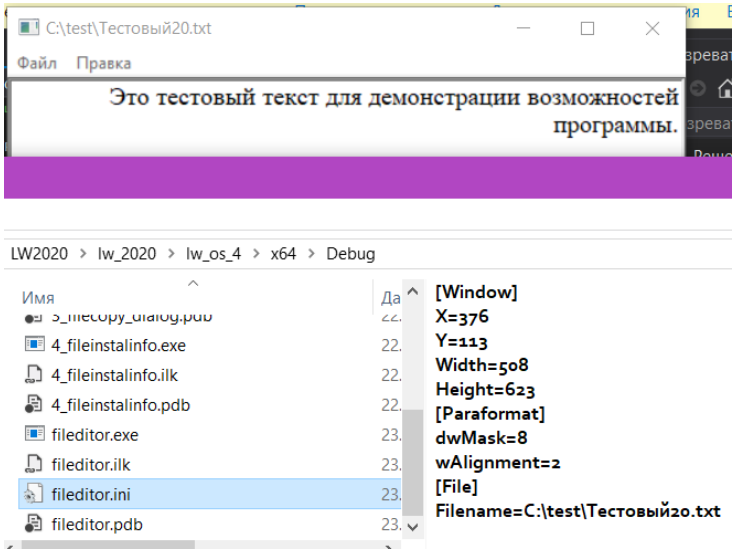

Рис. 4.3. Файл инициализации

2. В Visual Studio было разработано оконное приложение, в котором можно выбрать файл или каталог и узнать имя, атрибуты и размер. На экран пользователю также выводится информация о времени создания, изменения и последнего обращения. Имя и атрибуты файла или каталога можно изменять и сохранять. Кроме того, как и в случае первого приложения, существует возможность сохранения информации о последней работе в приложении. Здесь, информация сохраняется в реестре в разделе: HKCU\Software\IT-311.

Листинг 2. FileInfo

```
#include "FileInfoFuncHeader.h"


int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpszCmdLine, int nCmdShow)
{

	HACCEL hAccel = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR_ACCELERATOR1));
```

```
	WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

	wcex.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
	wcex.lpfnWndProc = MainWindowProc; // оконная процедура
	wcex.hInstance = hInstance;
	wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
	wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground = (HBRUSH)(COLOR_BTNFACE + 2);
	wcex.lpszMenuName = MAKEINTRESOURCE(IDR_MENU1);
	wcex.lpszClassName = TEXT("MainWindowClass"); // имя класса
	wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

	DWORD dwDisposition;

	// создаём и открываем ключ реестра для сохранения параметров приложения
	RegCreateKeyEx(HKEY_CURRENT_USER, TEXT("Software\\IT-311"),
		0, NULL, REG_OPTION_NON_VOLATILE, KEY_QUERY_VALUE | KEY_SET_VALUE, NULL, &hKey,
&dwDisposition);

	// копируем имя файла/каталога из системного реестра
	RegGetValueSZ(hKey, TEXT("Path"), FileName, _countof(FileName), NULL);
	// копируем положение окна из системного реестра
	RegGetValueBinary(hKey, TEXT("rect"), (LPBYTE)&rect, sizeof(rect), NULL);

	LoadLibrary(TEXT("ComCtl32.dll"));//для элементов общего пользования


	if (0 == RegisterClassEx(&wcex)) // регистрируем класс
	{
		return -1; // завершаем работу приложения
	}
	RECT wr = { 0, 0, 500, 500 };    // set the size, but not the position

	// создаем главное окно на основе нового оконного класса
	HWND hWnd = CreateWindowEx(0, TEXT("MainWindowClass"), TEXT("Process"),
WS_OVERLAPPEDWINDOW^WS_THICKFRAME^WS_MINIMIZEBOX^WS_MAXIMIZEBOX, 300,300,
		wr.right - wr.left,   wr.bottom - wr.top, NULL, NULL, hInstance, NULL);
	if (IsRectEmpty(&rect) == FALSE)
		{
			// изменяем положение окна
			SetWindowPos(hWnd, NULL, rect.left, rect.top, 0, 0, SWP_NOSIZE|
SWP_SHOWWINDOW);
	} // if


	if (NULL == hWnd)
	{
		return -1; // завершаем работу приложения
	}

	ShowWindow(hWnd, nCmdShow); // отображаем главное окно

	MSG  msg;
	BOOL Ret;

	for (;;)
	{

		// извлекаем сообщение из очереди
		Ret = GetMessage(&msg, NULL, 0, 0);
		if (Ret == FALSE)
		{
			break; // получено WM_QUIT, выход из цикла
		}
		else if (!TranslateAccelerator(hWnd, hAccel, &msg))
		{
			TranslateMessage(&msg);
```

```
                DispatchMessage(&msg);
            }
        }
        return (int)msg.wParam;
}
LRESULT CALLBACK MainWindowProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
        switch (msg)
        {
            HANDLE_MSG(hwnd, WM_CREATE, OnCreate);
            HANDLE_MSG(hwnd, WM_COMMAND, OnCommand);
        case WM_SIZE:
        {
            HWND hwndCtl = GetDlgItem(hwnd, IDC_EDIT_TEXT);
            MoveWindow(hwndCtl, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE); // изменяем размеры
поля ввода
        }

        break;
        case WM_DESTROY:
        {
            PostQuitMessage(0); // отправляем сообщение WM_QUIT
        }break;
        case WM_CLOSE:

            DestroyWindow(hwnd); // уничтожаем окно
            break;
        }
        return DefWindowProc(hwnd, msg, wParam, lParam);
}



BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCRStr)
{
        CreateWindowEx(0, TEXT("Edit"),NULL, WS_CHILD | WS_VISIBLE | WS_BORDER, 30, 10, 400, 30,
hwnd, (HMENU)IDC_EDIT_FILENAME, lpCRStr->hInstance, NULL);
        HWND hwndLV = CreateWindowEx(0, TEXT("SysListView32"), NULL,WS_CHILD | WS_VISIBLE |
WS_BORDER | LVS_REPORT | LVS_SHOWSELALWAYS , 30, 40, 400, 150, hwnd, (HMENU)IDC_LIST1, lpCRStr-
>hInstance, NULL);

        //значения атрибутов
        CreateWindowEx(0, TEXT("button"), TEXT("Только для чтения"), WS_CHILD | WS_VISIBLE |
BS_AUTOCHECKBOX, 30, 200, 400, 30, hwnd, (HMENU)IDC_ATTRIBUTE_READONLY, lpCRStr->hInstance,
NULL);
        CreateWindowEx(0, TEXT("button"), TEXT("Скрытый"), WS_CHILD | WS_VISIBLE |
BS_AUTOCHECKBOX, 30, 230, 400, 30, hwnd, (HMENU)IDC_ATTRIBUTE_HIDDEN, lpCRStr->hInstance, NULL);
        CreateWindowEx(0, TEXT("button"), TEXT("Файл готов для архивирования"), WS_CHILD |
WS_VISIBLE | BS_AUTOCHECKBOX, 30, 260, 400, 30, hwnd, (HMENU)IDC_ATTRIBUTE_ARCHIVE, lpCRStr-
>hInstance, NULL);
        CreateWindowEx(0, TEXT("button"), TEXT("Системный"), WS_CHILD | WS_VISIBLE |
BS_AUTOCHECKBOX, 30, 290, 400, 30, hwnd, (HMENU)IDC_ATTRIBUTE_SYSTEM, lpCRStr->hInstance, NULL);
        CreateWindowEx(0, TEXT("button"), TEXT("Временный"), WS_CHILD | WS_VISIBLE |
BS_AUTOCHECKBOX, 30, 320, 400, 30, hwnd, (HMENU)IDC_ATTRIBUTE_TEMPORARY, lpCRStr->hInstance,
NULL);
        CreateWindowEx(0, TEXT("button"), TEXT("Сжимать для экономии места"), WS_CHILD |
WS_VISIBLE | BS_AUTOCHECKBOX, 30, 350, 400, 30, hwnd, (HMENU)IDC_ATTRIBUTE_COMPRESSED, lpCRStr-
>hInstance, NULL);
        CreateWindowEx(0,TEXT("button"), TEXT("Шифровать содержимое для защиты"), WS_CHILD |
WS_VISIBLE | BS_AUTOCHECKBOX,30, 380, 400, 30, hwnd,(HMENU)IDC_ATTRIBUTE_ENCRYPTED,lpCRStr-
>hInstance, NULL);

        // задем расширенный
        ListView_SetExtendedListViewStyle(hwndLV, LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);

        // вставляем три столбца в список просмотра
```

```
	LVCOLUMN lvColumns[] = {
		{ LVCF_WIDTH | LVCF_TEXT, 0, 200, (LPTSTR)TEXT("Свойство") },
		{ LVCF_WIDTH | LVCF_TEXT, 0, 200, (LPTSTR)TEXT("Значение") },
	};

	for (int i = 0; i < _countof(lvColumns); ++i)
	{
		// вставляем столбец
		ListView_InsertColumn(hwndLV, i, &lvColumns[i]);
	}
	if (FileName != NULL) //если путь не пустой, то заполнить список
	{
		ListViewInit(FileName, hwnd);
	}

	return TRUE;
}

void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
	static HWND hEdit = GetDlgItem(hwnd, IDC_EDIT_TEXT);

	switch (id)
	{
	case ID_OPEN_FILE: // Открыть
		{
			OPENFILENAME openfile = { sizeof(OPENFILENAME) };

			openfile.hInstance = GetWindowInstance(hwnd);
			openfile.lpstrFilter = TEXT("Текстовые документы (*.txt)\0*.txt\0");
			openfile.lpstrFile = FileName;

			openfile.nMaxFile = _countof(FileName);
			openfile.lpstrTitle = TEXT("Открыть");
			openfile.Flags = OFN_EXPLORER | OFN_ENABLESIZING | OFN_FILEMUSTEXIST;
			openfile.lpstrDefExt = TEXT("txt");

			if (GetOpenFileName(&openfile) != FALSE)
			{
				// получаем информацию о файле
				//get info about file
				if (!(ListViewInit(FileName, hwnd)))
				{
					GetLastError();
					break;
				}
				else
				{
					break;
				}
			}
			else
			{
				MessageBox(NULL, TEXT("Не удалось открыть текстовый файл."),
NULL, MB_OK | MB_ICONERROR);
				FileName[0] = _T('\0');
			}

		}
	break;
	case ID_OPEN_DIR://Открыть папку
	{
		BROWSEINFO bi;//structure for open special box with folder in treview
		LPITEMIDLIST            pidl;
		LPMALLOC  pMalloc = NULL;
```

```
			ZeroMemory(&bi, sizeof(bi));
			bi.hwndOwner = NULL;
			bi.pszDisplayName = FileName;
			bi.lpszTitle = TEXT("Select folder");
			bi.ulFlags = BIF_RETURNONLYFSDIRS;

			pidl = SHBrowseForFolder(&bi);//open window for select
			if (pidl)
			{
				SHGetPathFromIDList(pidl, FileName);//get path

				if (!(ListViewInit(FileName, hwnd)))
				{
					GetLastError();
					break;
				}
			}
		}
			break;
		case ID_CHANGE_ATR://Изменение атрибутов
		{
			TCHAR NewFileName[MAX_PATH]; // новое имя файла/каталога
			GetDlgItemText(hwnd, IDC_EDIT_FILENAME, NewFileName, _countof(NewFileName));//это
имя и его к указателю lpszFileName

				// найдём имя в пути к файлу/каталогу
			lpszFileName = PathFindFileName(FileName);

			// вычисляем длину пути к файлу/каталогу
			cchPath = (DWORD)(lpszFileName - FileName) - 1;
			// разделяем нуль-символом путь и имя файла/каталога
			FileName[cchPath] = _T('\0');

			if (CompareString(LOCALE_USER_DEFAULT, 0, lpszFileName, -1, NewFileName, -1) !=
CSTR_EQUAL) // (!) изменилось имя файла/каталога
			{
				TCHAR ExistingFileName[MAX_PATH]; // старое имя файла/каталога
				StringCchPrintf(ExistingFileName, _countof(ExistingFileName), TEXT("%s\\%s"),
FileName, lpszFileName);

				// формируем новый путь к файлу/каталогу
				PathAppend(FileName, NewFileName);
				// переименовываем файл/каталог
				MoveFile(ExistingFileName, FileName);
			} // if
			else
			{
				// заменим нуль-символ, разделяющий путь и имя файла/каталога
				FileName[cchPath] = _T('\\');
			} // else

			// массив атрибутов
			constexpr DWORD attr[] = {
				FILE_ATTRIBUTE_READONLY, FILE_ATTRIBUTE_HIDDEN, FILE_ATTRIBUTE_ARCHIVE,
				FILE_ATTRIBUTE_SYSTEM, FILE_ATTRIBUTE_TEMPORARY, FILE_ATTRIBUTE_COMPRESSED,
FILE_ATTRIBUTE_ENCRYPTED
			};

			// массив идентификаторов флажков для атрибутов
			constexpr DWORD ids[] = {
				IDC_ATTRIBUTE_READONLY, IDC_ATTRIBUTE_HIDDEN, IDC_ATTRIBUTE_ARCHIVE,
				IDC_ATTRIBUTE_SYSTEM, IDC_ATTRIBUTE_TEMPORARY, IDC_ATTRIBUTE_COMPRESSED,
IDC_ATTRIBUTE_ENCRYPTED
			};

			DWORD dwFileAttributes = 0; // атрибуты файла/каталога
```

```
			for (int i = 0; i < _countof(attr); ++i)
			{
				if (IsDlgButtonChecked(hwnd, ids[i]) == BST_CHECKED) // флажок установлен
				{
					dwFileAttributes |= attr[i]; // добавим соответствующий атрибут
				}
			}

			// зададим атрибуты
			SetFileAttributes(FileName, dwFileAttributes);

			ListViewInit(FileName, hwnd);
		}

		case ID_RENAME://переименование без сохранения атрибутов
		{
			TCHAR NewFileName[MAX_PATH]; // новое имя файла/каталога
			GetDlgItemText(hwnd, IDC_EDIT_FILENAME, NewFileName, _countof(NewFileName));//это
имя и его к указателю lpszFileName

				// найдём имя в пути к файлу/каталогу
			lpszFileName = PathFindFileName(FileName);

			// вычисляем длину пути к файлу/каталогу
			cchPath = (DWORD)(lpszFileName - FileName) - 1;
			// разделяем нуль-символом путь и имя файла/каталога
			FileName[cchPath] = _T('\0');

			if (CompareString(LOCALE_USER_DEFAULT, 0, lpszFileName, -1, NewFileName, -1) !=
CSTR_EQUAL) // (!) изменилось имя файла/каталога
			{
				TCHAR ExistingFileName[MAX_PATH]; // старое имя файла/каталога
				StringCchPrintf(ExistingFileName, _countof(ExistingFileName), TEXT("%s\\%s"),
FileName, lpszFileName);

				// формируем новый путь к файлу/каталогу
				PathAppend(FileName, NewFileName);
				// переименовываем файл/каталог
				MoveFile(ExistingFileName, FileName);
			}
			else
			{
				// заменим нуль-символ, разделяющий путь и имя файла/каталога
				FileName[cchPath] = _T('\\');
			}
			// запомним размер и положение окна
			GetWindowRect(hwnd, &rect);

			ListViewInit(FileName, hwnd);
		}
		break;
		case ID_SAVE_PARAM://сохранение параметров файла в реестре
		{
			// запомним размер и положение окна
			GetWindowRect(hwnd, &rect);
			// вычисляем размер строкового значения (в байтах)
			DWORD cb = (_tcslen(FileName) + 1) * sizeof(TCHAR);
			// изменяем значение параметра
			RegSetValueEx(hKey, TEXT("Path"), 0, REG_SZ, (LPCBYTE)FileName, cb);
			// сохраняем положение окна в системный реестр
			RegSetValueEx(hKey, TEXT("rect"), 0, REG_BINARY, (LPCBYTE)&rect, sizeof(rect));


		}
		break;
		case ID_EXIT:
			SendMessage(hwnd, WM_CLOSE, 0, 0);
```

```
			RegCloseKey(hKey);
			break;

		}
}

BOOL ListViewInit(LPTSTR path, HWND hwnd)
{
		WIN32_FILE_ATTRIBUTE_DATA bhfi;
		TCHAR TimeBuffer[100], Buffer[100];
		if (!GetFileAttributesEx(path, GetFileExInfoStandard, &bhfi))
		{
			GetLastError();
		}

		//получение информации о размере файла
		//get info about size of file
		LARGE_INTEGER LI_Size;
		ULARGE_INTEGER sizeDir = { 0 };
		hFile = CreateFile(
			FileName,   // имя файла
			GENERIC_READ,          // чтение из файла
			FILE_SHARE_READ,       // совместный доступ к файлу
			NULL,                  // защиты нет
			OPEN_EXISTING,         // открываем существующий файл
			FILE_ATTRIBUTE_NORMAL,  // асинхронный ввод
			NULL                   // шаблона нет
		);
		if (!GetFileSizeEx(hFile, &LI_Size))
		{
			//обработка ошибки
		}
		if (bhfi.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
		{
			//расчет для папки
			CalculateSize(path, &bhfi, &sizeDir);
			ConvertDirectSize(Buffer, _countof(Buffer), sizeDir);
		}
		else
		{

			ConvertFileSize(Buffer, _countof(Buffer), LI_Size);

		}

		LPTSTR lpFN = PathFindFileNameW(path);
		SetDlgItemText(hwnd, IDC_EDIT_FILENAME, lpFN);
		/*Работает, не трогать*/

		//Добавление найденных атрибутов в список просмотра
		HWND hwndLV = GetDlgItem(hwnd, IDC_LIST1);
		// добавляем новый элемент в список просмотра
		ListView_DeleteAllItems(hwndLV);
		LVITEM lvItem = { LVIF_TEXT | LVIF_PARAM };
		lvItem.iItem = ListView_GetItemCount(hwndLV);
		//Размер
		lvItem.pszText = (LPWSTR)(L"Размер:");
		lvItem.iItem = ListView_InsertItem(hwndLV, &lvItem);
		if ((lvItem.iItem != -1))
		{
			ListView_SetItemText(hwndLV, lvItem.iItem, 1, Buffer);
		}
		//третий параметр
		lvItem.pszText = (LPWSTR)(L"Время изменения:");
		lvItem.iItem = ListView_InsertItem(hwndLV, &lvItem);
		if ((lvItem.iItem != -1))
		{
```

```
		GetFileTimeFormat(&bhfi.ftCreationTime, TimeBuffer, _countof(TimeBuffer));//время
создания
		ListView_SetItemText(hwndLV, lvItem.iItem, 1, TimeBuffer);
	}
	//второй параметр
	lvItem.pszText = (LPWSTR)(L"Время последнего обращения:");
	lvItem.iItem = ListView_InsertItem(hwndLV, &lvItem);
	if ((lvItem.iItem != -1))
	{
		GetFileTimeFormat(&bhfi.ftLastAccessTime, TimeBuffer, _countof(TimeBuffer));//время
последнего обращения
		ListView_SetItemText(hwndLV, lvItem.iItem, 1, TimeBuffer);
	}
	//первый параметр
	lvItem.pszText = (LPWSTR)(L"Время создания:");
	lvItem.iItem = ListView_InsertItem(hwndLV, &lvItem);
	if ((lvItem.iItem != -1))
	{
		GetFileTimeFormat(&bhfi.ftLastWriteTime, TimeBuffer, _countof(TimeBuffer));//время
изменения
		ListView_SetItemText(hwndLV, lvItem.iItem, 1, TimeBuffer);
	}
	//Расположение
	lvItem.pszText = (LPWSTR)(L"Расположение:");
	lvItem.iItem = ListView_InsertItem(hwndLV, &lvItem);
	if ((lvItem.iItem != -1))
	{
		ListView_SetItemText(hwndLV, lvItem.iItem, 1, path);
	}
	//Тип
	lvItem.pszText = (LPWSTR)(L"Тип:");
	lvItem.iItem = ListView_InsertItem(hwndLV, &lvItem);
	if ((lvItem.iItem != -1))
	{
		if (!(bhfi.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
		{
			ListView_SetItemText(hwndLV, lvItem.iItem, 1, (LPWSTR)(L"Файл"));
		}
		else
			if(bhfi.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
			ListView_SetItemText(hwndLV, lvItem.iItem, 1, (LPWSTR)(L"Папка с файлами"));
	}

	// массив атрибутов
	constexpr DWORD attr[] = {
		FILE_ATTRIBUTE_READONLY, FILE_ATTRIBUTE_HIDDEN, FILE_ATTRIBUTE_ARCHIVE,
		FILE_ATTRIBUTE_SYSTEM, FILE_ATTRIBUTE_TEMPORARY, FILE_ATTRIBUTE_COMPRESSED,
FILE_ATTRIBUTE_ENCRYPTED
	};
	// массив идентификаторов флажков для атрибутов
	constexpr DWORD ids[] = {
		IDC_ATTRIBUTE_READONLY, IDC_ATTRIBUTE_HIDDEN, IDC_ATTRIBUTE_ARCHIVE,
		IDC_ATTRIBUTE_SYSTEM, IDC_ATTRIBUTE_TEMPORARY, IDC_ATTRIBUTE_COMPRESSED,
IDC_ATTRIBUTE_ENCRYPTED
	};

	// раставим флажки соответственно с установленными атрибутами файла/каталога

	for (int i = 0; i < _countof(attr); ++i)
	{
		UINT uCheck = (bhfi.dwFileAttributes & attr[i]) ? BST_CHECKED : BST_UNCHECKED;
		CheckDlgButton(hwnd, ids[i], uCheck);
	}

	// закрываем дескриптор файла */
	CloseHandle(hFile);
	return TRUE;
```

```
}

BOOL __stdcall CalculateSize(LPCTSTR lpszFileName, const LPWIN32_FILE_ATTRIBUTE_DATA
lpFileAttributeData, LPVOID lpvParam)
{
	if (lpFileAttributeData->dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
	{
		// продолжим поиск внутри каталога
		return FileSearch(TEXT("*"), lpszFileName, CalculateSize, lpvParam);
	} // if
	return TRUE; // возвращаем TRUE, чтобы продолжить поиск
}

BOOL FileSearch(LPCTSTR lpszFileName,LPCTSTR path, LPSEARCHFUNC lpSearchFunc, LPVOID lpvParam)
{
	WIN32_FIND_DATA ffd;
	LARGE_INTEGER filesize;
	LARGE_INTEGER size;
	TCHAR szDir[MAX_PATH];
	size_t length_of_arg;
	HANDLE hFind = INVALID_HANDLE_VALUE;
	StringCchCopy(szDir, MAX_PATH, path);
	StringCchCat(szDir, MAX_PATH, TEXT("\\*"));
	hFind = FindFirstFile(szDir, &ffd);

	if (INVALID_HANDLE_VALUE == hFind)
	{
		//error. terminator . later.
	}
	do
	{
		if (!(ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
		{
			filesize.LowPart = ffd.nFileSizeLow;
			filesize.HighPart = ffd.nFileSizeHigh;
			((ULARGE_INTEGER *)lpvParam)->QuadPart += filesize.QuadPart;
		}
	} while (FindNextFile(hFind, &ffd) != 0);

	FindClose(hFind);
	return TRUE;
}

BOOL GetFileTimeFormat(const LPFILETIME FileTime, LPTSTR lptstrFileTime, DWORD cchFileTime)
{
	SYSTEMTIME st;

	// преобразуем дату и время из FILETIME в SYSTEMTIME
	BOOL bRet = FileTimeToSystemTime(FileTime, &st);

	// приведем дату и время к текущему часовому поясу
	if (FALSE != bRet)
		bRet = SystemTimeToTzSpecificLocalTime(NULL, &st, &st);

	if (FALSE != bRet)
	{
		// скопируем дату в результирующую строку
		GetDateFormat(LOCALE_USER_DEFAULT, DATE_LONGDATE, &st, NULL, lptstrFileTime,
cchFileTime);

		// добавим время в результирующую строку

		StringCchCat(lptstrFileTime, cchFileTime, TEXT(", "));
		DWORD len = _tcslen(lptstrFileTime);

		if (len < cchFileTime)
```

```
                    GetTimeFormat(LOCALE_USER_DEFAULT, TIME_FORCE24HOURFORMAT, &st, NULL,
lptstrFileTime + len, cchFileTime - len);
        }

        return bRet;
}

void ConvertFileSize(LPTSTR lpszBuffer, DWORD cch, LARGE_INTEGER size)
{
        if (size.QuadPart >= 0x40000000ULL)
        {
                StringCchPrintf(lpszBuffer, cch, TEXT("%.1f ГБ"), (size.QuadPart /
(float)0x40000000ULL));
        }
        else if (size.QuadPart >= 0x100000ULL)
        {
                StringCchPrintf(lpszBuffer, cch, TEXT("%.1f МБ"), (size.QuadPart /
(float)0x100000ULL));
        }
        else if (size.QuadPart >= 0x0400ULL)
        {
                StringCchPrintf(lpszBuffer, cch, TEXT("%.1f КБ"), (size.QuadPart /
(float)0x0400ULL));
        }
        else
        {
                StringCchPrintf(lpszBuffer, cch, TEXT("%u байт"), size.LowPart);
        }

        size_t len = _tcslen(lpszBuffer);

        if (len < cch)
        {
                StringCchPrintf((lpszBuffer + len), (cch - len), TEXT(" (%llu байт)"),
size.QuadPart);
        }
}

void ConvertDirectSize(LPTSTR lpszBuffer, DWORD cch, ULARGE_INTEGER size)
{
        if (size.QuadPart >= 0x40000000ULL)
        {
                StringCchPrintf(lpszBuffer, cch, TEXT("%.1f ГБ"), (size.QuadPart /
(float)0x40000000ULL));
        }
        else if (size.QuadPart >= 0x100000ULL)
        {
                StringCchPrintf(lpszBuffer, cch, TEXT("%.1f МБ"), (size.QuadPart /
(float)0x100000ULL));
        }
        else if (size.QuadPart >= 0x0400ULL)
        {
                StringCchPrintf(lpszBuffer, cch, TEXT("%.1f КБ"), (size.QuadPart /
(float)0x0400ULL));
        }
        else
        {
                StringCchPrintf(lpszBuffer, cch, TEXT("%u байт"), size.LowPart);
        }

        size_t len = _tcslen(lpszBuffer);

        if (len < cch)
        {
                StringCchPrintf((lpszBuffer + len), (cch - len), TEXT(" (%llu байт)"),
size.QuadPart);
        }
```

```
}


//work wit regist
LSTATUS RegGetValueSZ(HKEY hKey, LPCTSTR lpValueName, LPTSTR lpszData, DWORD cch, LPDWORD
lpcchNeeded)
{
	// определяем тип получаемого значения параметра
	DWORD RegType;
	LSTATUS retCode = RegQueryValueEx(hKey, lpValueName, NULL, &RegType, NULL, NULL);
	if (ERROR_SUCCESS == retCode && REG_SZ == RegType)
	{
		DWORD DataBuffer = cch * sizeof(TCHAR);
		// получаем значение параметра
		retCode = RegQueryValueEx(hKey,lpValueName, NULL, NULL, (LPBYTE)lpszData,
&DataBuffer);
	}
	else if (ERROR_SUCCESS == retCode)
	{
		retCode = ERROR_UNSUPPORTED_TYPE; // неверный тип данных
	}

	return retCode; }

LSTATUS RegGetValueBinary(HKEY hKey, LPCTSTR lpValueName, LPBYTE lpData, DWORD cb, LPDWORD
lpcbNeeded)
{
	DWORD dwType;
	// определяем тип получаемого значения параметра
	LSTATUS lStatus = RegQueryValueEx(hKey, lpValueName, NULL, &dwType, NULL, NULL);

	if (ERROR_SUCCESS == lStatus && REG_BINARY == dwType)
	{
		// получаем значение параметра
		lStatus = RegQueryValueEx(hKey, lpValueName, NULL, NULL, lpData, &cb);

		if (NULL != lpcbNeeded) *lpcbNeeded = cb;
	} // if
	else if (ERROR_SUCCESS == lStatus)
	{
		lStatus = ERROR_UNSUPPORTED_TYPE; // неверный тип данных
	} // if

	return lStatus;
}
```

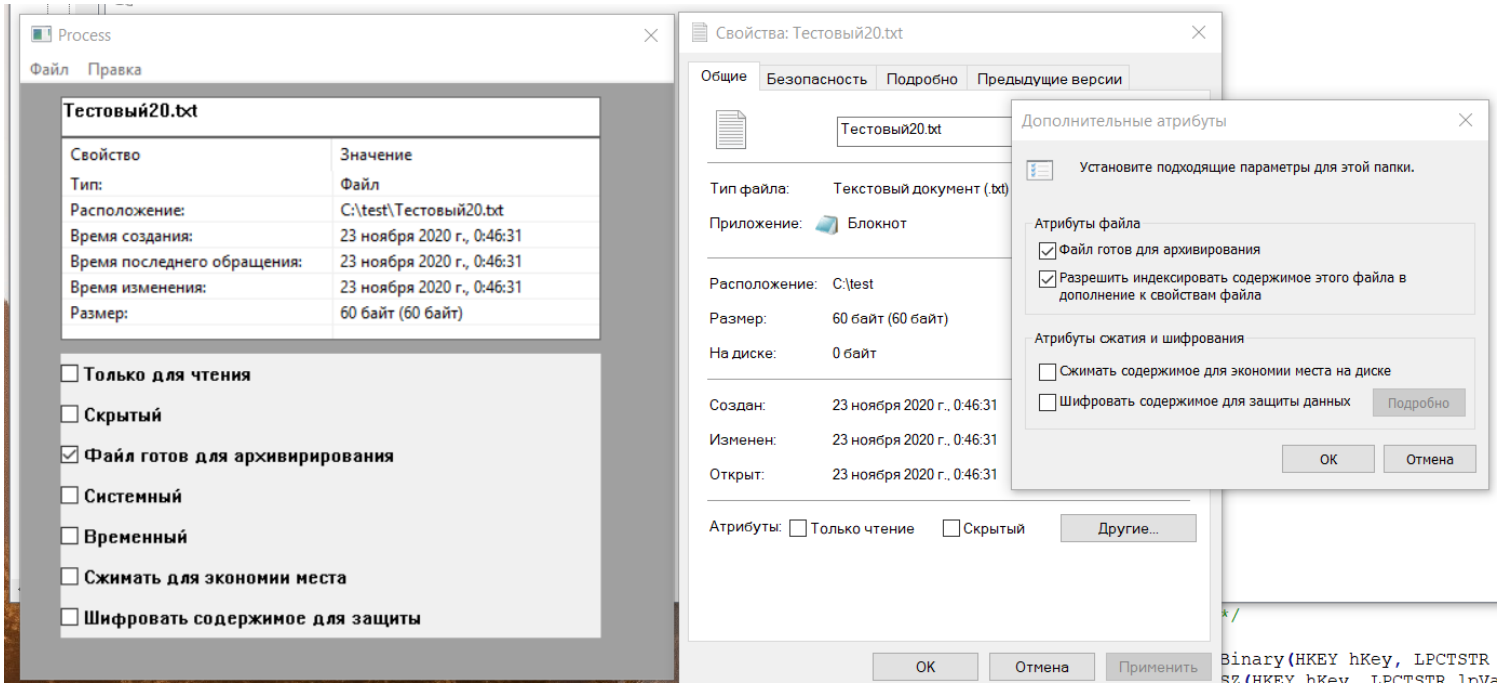

Рис. 4.4. Свойства файла

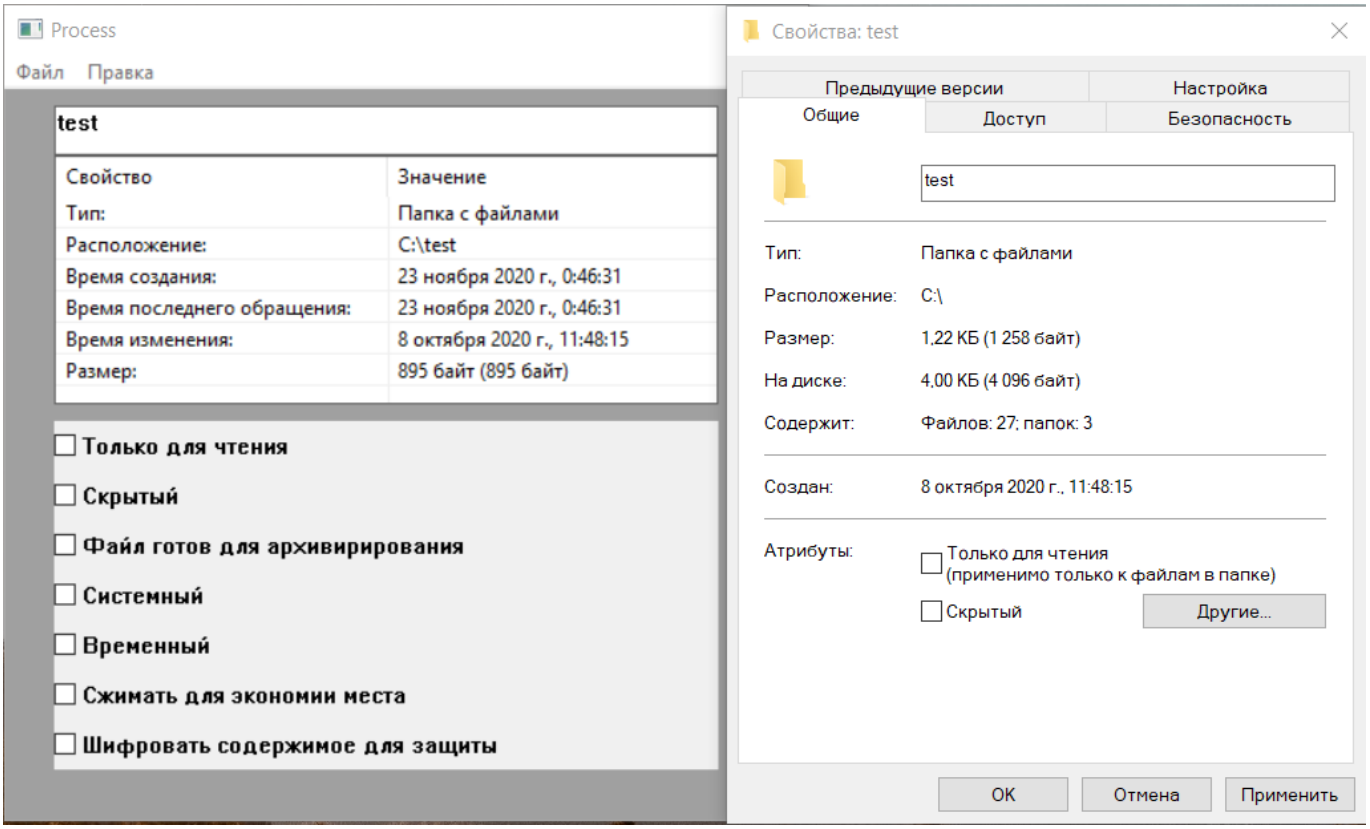

Рис. 4.5. Свойства директории

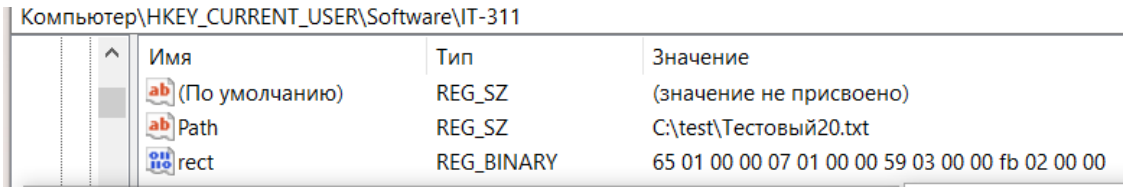

Рис. 4.6. Данные в реестре

3. Разработано в Visual C++ оконное приложение Win32, которое выполняет копирование файлов и каталогов вместе с вложенными файлами и каталогами.

```
#include "FileCopeDialogHeader.h"

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpszCmdLine, int nCmdShow)
{

	HINSTANCE relib = LoadLibrary(TEXT("riched32.dll"));	//load the dll don't forget this
								//and don't forget to
free it (see wm_destroy)
	if (relib == NULL)
		MessageBox(NULL, TEXT("Failed to load riched32.dll!"), TEXT("Error"),
MB_ICONEXCLAMATION);
	HACCEL hAccel = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR_ACCELERATOR1));

	WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

	wcex.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
	wcex.lpfnWndProc = MainWindowProc; // оконная процедура
	wcex.hInstance = hInstance;
	wcex.hIcon = LoadIcon(NULL, IDI_APPLICATION);
	wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground = (HBRUSH)(COLOR_BTNFACE + 2);

	wcex.lpszClassName = TEXT("MainWindowClass"); // имя класса
	wcex.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

	if (0 == RegisterClassEx(&wcex)) // регистрируем класс
	{
		return -1; // завершаем работу приложения
	}

	LoadLibrary(TEXT("ComCtl32.dll"));//для элементов общего пользования
	// создаем главное окно на основе нового оконного класса


	HWND hWnd = CreateWindowEx(0, TEXT("MainWindowClass"), TEXT("Process"),
WS_OVERLAPPEDWINDOW,
		100, 100, 10, 10, NULL, NULL, hInstance, NULL);

	if (NULL == hWnd)
	{
		return -1; // завершаем работу приложения
	}

	ShowWindow(hWnd, nCmdShow); // отображаем главное окно



	MSG  msg;
	BOOL Ret;

	for (;;)
	{


		// извлекаем сообщение из очереди
		Ret = GetMessage(&msg, NULL, 0, 0);
		if (Ret == FALSE)
		{
			break; // получено WM_QUIT, выход из цикла
		}
		else if (!TranslateAccelerator(hWnd, hAccel, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
```

```
		}
	}

	return (int)msg.wParam;
	 }
LRESULT CALLBACK MainWindowProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
	switch (msg)
	{
	case WM_CREATE:
	{
		HINSTANCE hInstance = GetWindowInstance(hwnd);
		HWND hDlg = CreateDialog(hInstance, MAKEINTRESOURCE(IDD_DIALOG1), hwnd, DialogProc);
		ShowWindow(hDlg, SW_SHOW);
	}
	break;

	}

	return DefWindowProc(hwnd, msg, wParam, lParam);
}


INT_PTR CALLBACK DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
	TCHAR FileName[260];
	BROWSEINFO bi;//structure for open special box with folder in treview
	HDC hdc;
	LPITEMIDLIST pidl;
	LPMALLOC  pMalloc = NULL;
	switch (uMsg)
	{
	case WM_LBUTTONDOWN:
	{
		DWORD xPos, yPos, nSize;
		TCHAR szBuf[80];

		// Сохраняем координаты курсора мыши
		xPos = LOWORD(lParam);
		yPos = HIWORD(lParam);

		/*Отследим точки над первым и вторым editbox
		Если да, то откроем для соответствующего editbox окна для их заполнения*/
		if ((xPos > 312 & xPos < 544)&(yPos>39&yPos<81))
		{
			//В какую директорию скопировать
				//TextOut(hdc, xPos, yPos, szBuf, nSize);
				ZeroMemory(&bi, sizeof(bi));
				bi.hwndOwner = NULL;
				bi.pszDisplayName = FileName;
				bi.lpszTitle = TEXT("Select folder");
				bi.ulFlags = BIF_RETURNONLYFSDIRS;

				pidl = SHBrowseForFolder(&bi);//open window for select
				if (pidl)
				{
					SHGetPathFromIDList(pidl, FileName);//get path
				}
				SetDlgItemText(hwndDlg, IDC_EDIT_TO,FileName);
		}
		else
			if ((xPos > 36 & xPos < 250)&(yPos > 39 & yPos < 81))
			{
					ZeroMemory(&bi, sizeof(bi));
					bi.hwndOwner = NULL;
					bi.pszDisplayName = FileName;
					bi.lpszTitle = TEXT("Select folder");
```

```
                                bi.ulFlags = BIF_BROWSEINCLUDEFILES;
                                pidl = SHBrowseForFolder(&bi);//open window for select
                                if (pidl)
                                {
                                        SHGetPathFromIDList(pidl, FileName);//get path
                                }
                                SetDlgItemText(hwndDlg, IDC_EDIT_FROM, FileName);
                        }
        }break;
        case WM_INITDIALOG:
        {
                BOOL bRet = HANDLE_WM_INITDIALOG(hwndDlg, wParam, lParam, Dialog_OnInitDialog);
                return SetDlgMsgResult(hwndDlg, uMsg, bRet);
        }

        case WM_CLOSE:
                HANDLE_WM_CLOSE(hwndDlg, wParam, lParam, Dialog_OnClose);

                return TRUE;

        case WM_COMMAND:
                HANDLE_WM_COMMAND(hwndDlg, wParam, lParam, Dialog_OnCommand);
                return TRUE;
        } // switch

        return FALSE;
}

BOOL Dialog_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)
{
        return TRUE;
}

void Dialog_OnClose(HWND hwnd)
{
        EndDialog(hwnd, IDCLOSE);
        DestroyWindow(hwnd); // уничтожаем окно
        PostQuitMessage(0); // отправляем сообщение WM_QUIT
}

BOOL Copy(LPCTSTR szInDirName, LPCTSTR szOutDirName)
{
        WIN32_FIND_DATA ffd;
        HANDLE hFind;

        TCHAR szFind[MAX_PATH + 1];
        TCHAR szInFileName[MAX_PATH + 1];
        TCHAR szOutFileName[MAX_PATH + 1];

        lstrcpy(szFind, szInDirName);
        lstrcat(szFind, L"\\*.*"); //ищем файлы с любым именем и рысширением

        hFind = FindFirstFile(szFind, &ffd);

        do
        {
                //Формируем полный путь (источник)
                lstrcpy(szInFileName, szInDirName);
                lstrcat(szInFileName, L"\\");
                lstrcat(szInFileName, ffd.cFileName);

                //Формируем полный путь (результат)

                lstrcpy(szOutFileName, szOutDirName);
                lstrcat(szOutFileName, L"\\");
                lstrcat(szOutFileName, ffd.cFileName);
```

```
                        if (ffd.dwFileAttributes & 0x00000010)
                        {
                                if (lstrcmp(ffd.cFileName, L".") == 0 || lstrcmp(ffd.cFileName, L"..")
== 0) continue;

                                CreateDirectory(szOutFileName, NULL);
                                Copy(szInFileName, szOutFileName);
                        }

                CopyFile(szInFileName, szOutFileName, TRUE);

        } while (FindNextFile(hFind, &ffd));

        FindClose(hFind);
        return TRUE;
}

void Dialog_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
        switch (id)
        {

        case IDOK:
{               TCHAR FromName[260];
                TCHAR ToName[260];

                GetDlgItemText(hwnd, IDC_EDIT_FROM, FromName, _countof(FromName));
                GetDlgItemText(hwnd, IDC_EDIT_TO, ToName, _countof(ToName));
/*Выясним, что копируется, файл или папка.
                Если папка, то сформируем с ней маршрут и продолжим поиск*/
                WIN32_FIND_DATA ffd;
                HANDLE hFind;
                BOOL BRET;
                LPCTSTR FILE = PathFindFileNameW(FromName);
                hFind = FindFirstFile(FromName, &ffd); //Ищем файл/каталог
                if (ffd.dwFileAttributes == FILE_ATTRIBUTE_DIRECTORY)
                {
                        lstrcat(ToName, L"\\");
                        lstrcat(ToName, FILE);
                        CreateDirectory(ToName, NULL);
                        BRET = Copy(FromName, ToName);
                }
                else
                {
                        lstrcat(ToName, L"\\");
                        lstrcat(ToName, ffd.cFileName);
                        BRET = CopyFile(FromName, ToName, TRUE);

                }
                TCHAR Message[MAX_PATH];
                if (BRET == 0)
                {
                        lstrcpy(Message, L"Файлы не скопированы в папку: ");
                        lstrcat(Message, ToName);
                        MessageBox(hwnd, Message, L"Ошибка", MB_OK);
                        SetDlgItemText(hwnd, IDC_EDIT_FROM, L" ");
                        SetDlgItemText(hwnd, IDC_EDIT_TO, L" ");
                        //ShowMessage(IntToStr(result));
                }
                else
                {

                        lstrcpy(Message, L"Файлы скопированы. Проверьте папку: ");
                        lstrcat(Message, ToName);
                        MessageBox(hwnd,Message, L" Успех!", MB_OK);
                        SetDlgItemText(hwnd, IDC_EDIT_FROM, L" ");
                        SetDlgItemText(hwnd, IDC_EDIT_TO, L" ");
```

```
            }
}
break;
	case IDCANCEL:
	{
		// завершаем работу диалогового окна
		EndDialog(hwnd, IDCANCEL);
		DestroyWindow(hwnd); // уничтожаем окно
		PostQuitMessage(0);
	}
	break;
	}
}
```

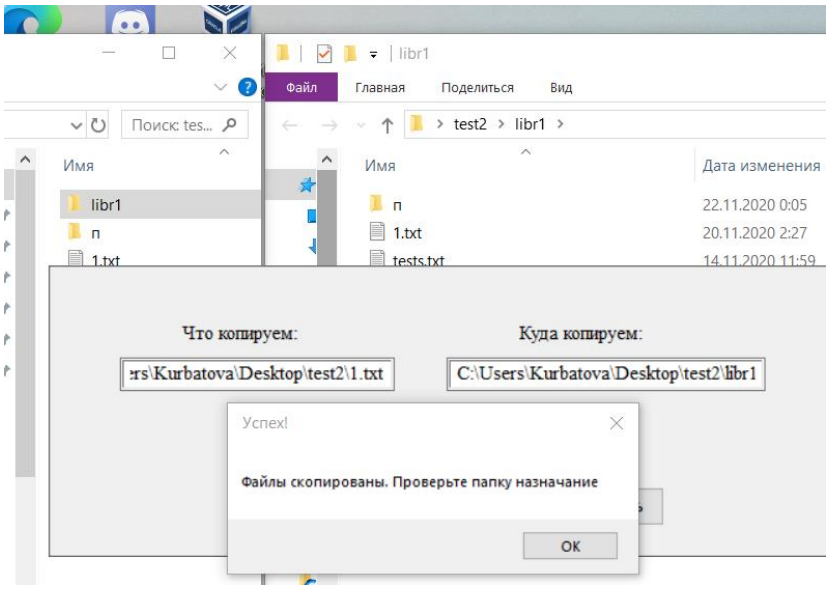

Рис. 4.7. Копирование файлов

4. Разработано в Visual C++ приложение Win32, которое выводит информацию из системного реестра.

Листинг 4. FileRegistrInfo

```
#include <Windows.h>
#include <tchar.h>
#include <stdio.h>
#include <locale.h>
#include <iostream>
#include <shlwapi.h>
#pragma comment(lib, "shlwapi.lib")


// функция для вывода списка установленных программ
void InstallAppList(REGSAM samDesired);
// функция для вывода списка программ автозапуска
void AutorunAppList(HKEY hRootKey, REGSAM samDesired);

void SubFind(LSTATUS retCode, DWORD CMVlen, HKEY hSubKey);

LSTATUS RegGetValueSZ(HKEY hKey, LPCTSTR lpValueName, LPTSTR lpszData, DWORD cch, LPDWORD
lpcchNeeded);
// функция для определения, является ли операционная система 64-разрядной версией Windows
BOOL IsWin64();

// ----------------------------------------------------------------------------------------
int _tmain()
{
	setlocale(LC_ALL, "");

	if (IsWin64()) // 64-разрядная версия Windows
```

```
	{
		std::cout<<"Список установленных программ:\n\n";

		// выведем список установленных программ (Win64)
		InstallAppList(KEY_ENUMERATE_SUB_KEYS | KEY_QUERY_VALUE | KEY_WOW64_64KEY);

		std::cout << "Список программ автозапуска
HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run:\n\n";

		AutorunAppList(HKEY_LOCAL_MACHINE, KEY_ENUMERATE_SUB_KEYS | KEY_QUERY_VALUE |
KEY_WOW64_64KEY);

		std::cout << "Список программ автозапуска
HKCU\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run:\n\n";

		AutorunAppList(HKEY_CURRENT_USER, KEY_ENUMERATE_SUB_KEYS | KEY_QUERY_VALUE |
KEY_WOW64_64KEY);
	}

}
// ----------------------------------------------------------------------------------------
void InstallAppList(REGSAM AccessRights)
{
	HKEY hKey; // дескриптор ключа реестра

	// открываем ключ реестра
	LSTATUS retCode = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
		TEXT("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall"),
REG_OPTION_NON_VOLATILE, AccessRights, &hKey);

	if (ERROR_SUCCESS == retCode)
	{
		DWORD cSubKeys, cbMaxSubKey,i; // вложенные ключи,их максимальный размер,переменная
для перехода по ним

		retCode = RegQueryInfoKey(hKey, NULL, NULL, NULL, &cSubKeys, &cbMaxSubKey, NULL,
NULL, NULL, NULL, NULL, NULL);

		if ((ERROR_SUCCESS == retCode) && (cSubKeys > 0))
		{
			printf("\nNumber of values: %u\n", cSubKeys);
			// выделим память под буфер для имени вложенного ключа
			LPTSTR szSubKeyName = new TCHAR[cbMaxSubKey + 1];

			for ( i = 0;  i< cSubKeys; ++i)
			{
				HKEY hSubKey = NULL; // дескриптор вложенного ключа реестра
				// получим имя вложенного ключа с индексом dwIndex
				DWORD cchValue = cbMaxSubKey + 1;

				if (ERROR_SUCCESS == RegEnumKeyEx(hKey, i, szSubKeyName, &cchValue,
NULL, NULL, NULL, NULL))
				{
					//начнем поиск по вложенным ключам
					if (ERROR_SUCCESS == RegOpenKeyEx(hKey, szSubKeyName,
REG_OPTION_NON_VOLATILE, AccessRights, &hSubKey))
					{
						DWORD cMaxValueLen = 0;//для определения максимальной
длины значения
						retCode = RegQueryInfoKey(hSubKey, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, &cMaxValueLen, NULL, NULL);
						SubFind(retCode, cMaxValueLen, hSubKey);//демонстрация
содержимого по вложенному ключу
						RegCloseKey(hSubKey), hSubKey = NULL;// закрываем
дескриптор вложенного ключа реестра
					}
				}
```

```
                }
                delete[] szSubKeyName;
            }

            RegCloseKey(hKey), hKey = NULL;// закрываем дескриптор ключа реестра
        }
}

void SubFind(LSTATUS retCode, DWORD CMVlen, HKEY hSubKey)
{

        DWORD RegType;//для определения типа данных параметров
        DWORD DataBuffer;//размер содержимого значений
        LPTSTR lpData = new TCHAR[CMVlen];//для определения содержимого значений

        if ((ERROR_SUCCESS == retCode) && (CMVlen > 0))
        {

        /*-------------Отобразить наименование ----------------------*/
                retCode = RegQueryValueEx(hSubKey, TEXT("DisplayName"), NULL, &RegType, NULL, NULL);
                if (RegType == REG_DWORD || retCode == ERROR_SUCCESS)
                {
                        DataBuffer = CMVlen * sizeof(TCHAR);
                        // получаем значение параметра
                        retCode = RegQueryValueEx(hSubKey, TEXT("DisplayName"), NULL, NULL,
(LPBYTE)lpData, &DataBuffer);
                        _tprintf(TEXT("%s\n"), lpData);
                }
                else
                        if (retCode != ERROR_SUCCESS)
                        {
                                retCode = ERROR_UNSUPPORTED_TYPE;
                        }
        /*-----------------------------------------------------------*/

        /*-------------Отобразить издателя----------------------*/
                retCode = RegQueryValueEx(hSubKey, TEXT("Publisher"), NULL, &RegType, NULL, NULL);
                if (RegType == REG_DWORD || retCode == ERROR_SUCCESS)
                {
                        DataBuffer = CMVlen * sizeof(TCHAR);
                        // получаем значение параметра
                        retCode = RegQueryValueEx(hSubKey, TEXT("Publisher"), NULL, NULL,
(LPBYTE)lpData, &DataBuffer);
                        _tprintf(TEXT("%s\n"), lpData);
                }
                else
                        if (retCode != ERROR_SUCCESS)
                        {
                                retCode = ERROR_UNSUPPORTED_TYPE;
                        }
                /*-----------------------------------------------------------*/

                /*-------------Отобразить номер версии----------------------*/
                retCode = RegQueryValueEx(hSubKey, TEXT("DisplayVersion"), NULL, &RegType, NULL,
NULL);
                if (RegType == REG_DWORD || retCode == ERROR_SUCCESS)
                {
                        DataBuffer = CMVlen * sizeof(TCHAR);
                        // получаем значение параметра
                        retCode = RegQueryValueEx(hSubKey, TEXT("DisplayVersion"), NULL, NULL,
(LPBYTE)lpData, &DataBuffer);
                        _tprintf(TEXT("%s\n"), lpData);
                }
                else
                        if (retCode != ERROR_SUCCESS)
                        {
                                retCode = ERROR_UNSUPPORTED_TYPE;
```

```
                }
            /*-------------------------------------------------------*/

                std::cout <<  "\n";
        }
        delete[] lpData;// освободим выделенную память}
}


void AutorunAppList(HKEY hRootKey, REGSAM AccessRights)
{
        HKEY hKey = NULL; // дескриптор ключа рееста

        // открываем ключ реестра
        LSTATUS lStatus = RegOpenKeyEx(hRootKey,
TEXT("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"), REG_OPTION_NON_VOLATILE, AccessRights,
&hKey);

        if (ERROR_SUCCESS == lStatus)
        {
            DWORD Val, MaxNameLen, MaxVallLen; // число параметров ключа, буфер для имен, буфер
для значений

            // определим число параметров ключа и
            // максимальный размер буферов для имён и значений параметров
            lStatus = RegQueryInfoKey(hKey, NULL, NULL, NULL, NULL, NULL, NULL, &Val,
&MaxNameLen, &MaxVallLen, NULL, NULL);

            if ((ERROR_SUCCESS == lStatus) && (Val > 0))
            {
                // выделим память под буферы
                LPTSTR NameKey = new TCHAR[MaxNameLen + 1];
                LPBYTE lpData = new BYTE[MaxVallLen];

                for (DWORD i = 0; i < Val; ++i)
                {

                    DWORD cchValueName = MaxNameLen + 1, dwType, cbData = MaxVallLen;
                    lStatus = RegEnumValue(hKey, i, NameKey, &cchValueName, NULL, &dwType,
lpData, &cbData);

                    if ((ERROR_SUCCESS == lStatus) && (cchValueName > 0))
                    {
                        //reg_sz - тип данных. его значение UNICODE-строка
                        if ((REG_SZ == dwType) || (REG_EXPAND_SZ == dwType))
                        {
                            _tprintf(TEXT("-----\n%s\n"), NameKey);//имя параметра
                            _tprintf(TEXT("%s\n\n"), (LPTSTR)lpData);//значение
параметра
                        }
                    }
                }

                delete[] lpData;//освобождение памяти
                delete[] NameKey;//освобождение памяти
            }

            RegCloseKey(hKey), hKey = NULL;          // закрываем дескриптор ключа реестра
        }
}

  //work wit regist

        LSTATUS RegGetValueSZ(HKEY hKey, LPCTSTR section, LPTSTR key, DWORD cch, LPDWORD
lpcchNeeded)
        {
            DWORD dwType;
```

```
		LSTATUS res = RegQueryValueEx(hKey, section, NULL, &dwType, NULL, NULL);
			if (res != ERROR_SUCCESS)
			{
					res = ERROR_UNSUPPORTED_TYPE; // неверный тип данных
			}
			else
				if (ERROR_SUCCESS == res)
				{
					DWORD cb = cch * sizeof(TCHAR);
						// получаем значение параметра
					res = RegQueryValueEx(hKey, section, NULL, NULL, (LPBYTE)key,
&cb);
					if (NULL != lpcchNeeded) *lpcchNeeded = cb / sizeof(TCHAR);
				}
		return res;
	}


	BOOL IsWin64()
	{
#ifdef _WIN64

		return TRUE;

#else // _WIN64

		// (!) функция IsWow64Process поддерживается не во всех версиях Windows.

		typedef BOOL(WINAPI *LPFN_ISWOW64PROCESS) (HANDLE, PBOOL);
		LPFN_ISWOW64PROCESS fnIsWow64Process =
(LPFN_ISWOW64PROCESS)GetProcAddress(GetModuleHandle(TEXT("kernel32")), "IsWow64Process");

		if (NULL != fnIsWow64Process)
		{
			BOOL bIsWow64;
			return (fnIsWow64Process(GetCurrentProcess(), &bIsWow64) != FALSE) &&
(bIsWow64 != FALSE);
		} // if

		return FALSE;

#endif // !_WIN64
		}
```

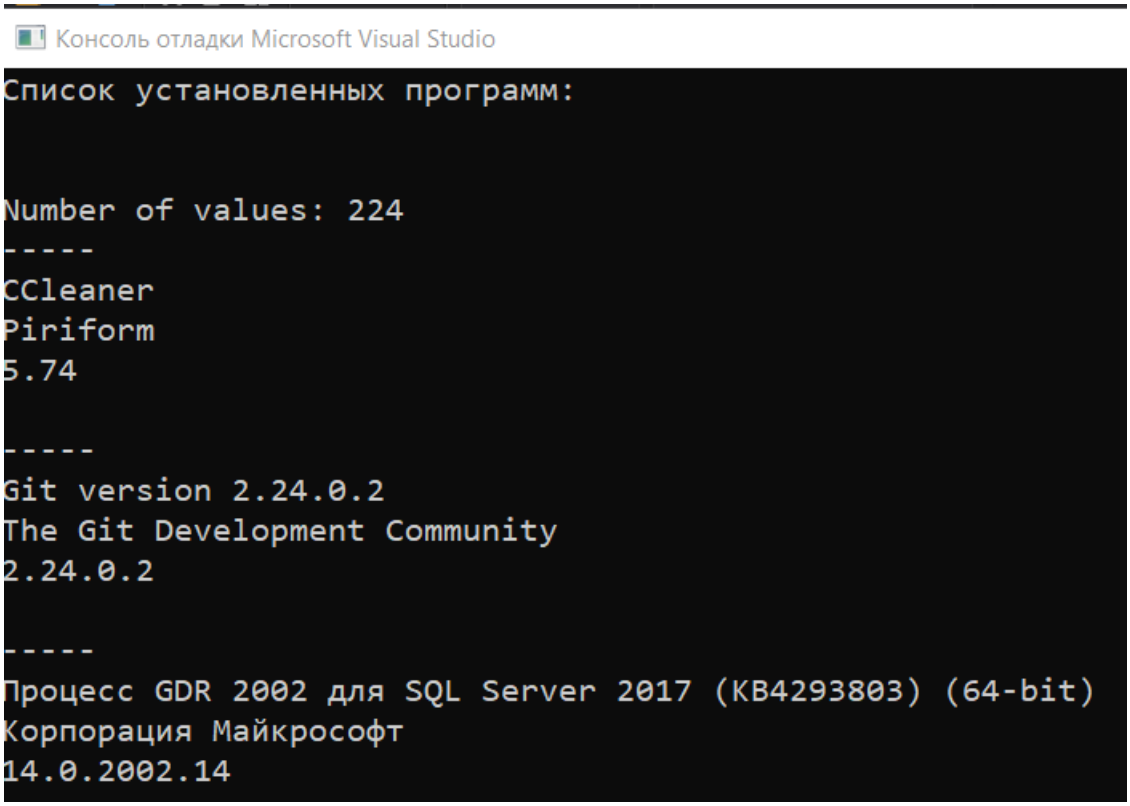

Рис. 4.8. Список установленных программ

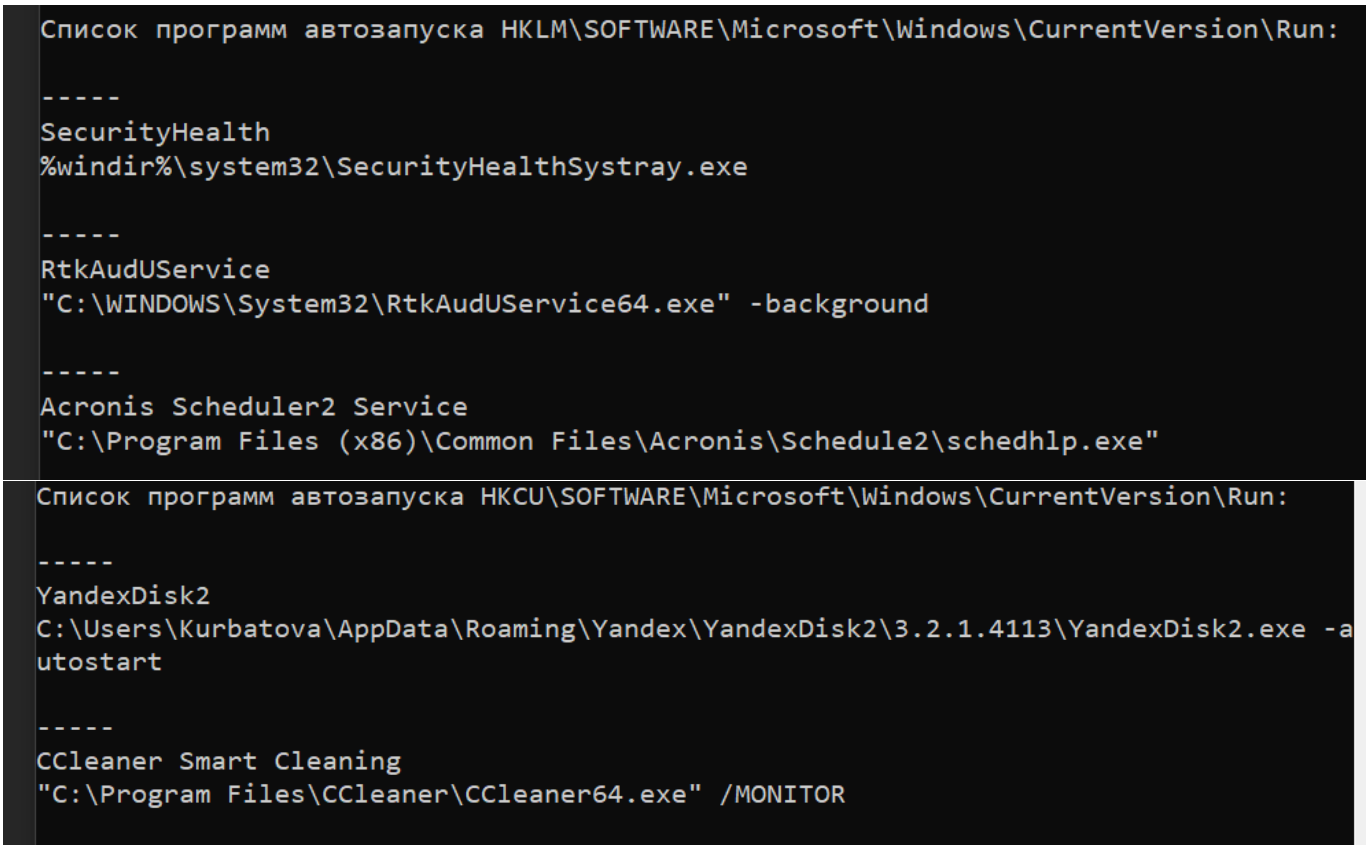

Рис. 4.9. Список программ автозапуска

Вывод: Таким образом, в ходе выполнения лабораторной работы было осуществлено создание 4 приложений для получения практических навыков работы с файлами, каталогами и системным реестром в Windows с применением функций Win32 API.