Лабораторная работа №5

студента группы ИТ – 32

Курбатовой Софьи Андреевны

Выполнение:_____ Защита _____

БЕЗОПАСНОСТЬ В WINDOWS

Цель работы: изучение модели безопасности Windows. Получение практических навыков применения функций Win32 API для управления безопасностью Windows.

Содержание работы

Вариант 11

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

– должно выводить SID локального компьютера;

– должно выводить SID учетной записи текущего пользователя;

– должно выводить имена учетных записей для хороший известных SID, указанных в варианте задания;

– должно выводить список привилегий и прав учетной записи текущего пользователя;

– должно выводить списки привилегий и прав учетных записей хорошо известных SID указанных в варианте задания;

2. Разработать в Visual C++ оконное приложение Win32:

– должно выводить в своем окне список всех процессов;

– должно выводить имя и SID учетной записи пользователя, связанной с маркером доступа выбранного процесса;

– должно выводить список групп, связанных с маркером доступа выбранного процесса;

– должно выводить список привилегий, которыми обладает маркер доступа выбранного процесса;

– должно для выбранного процесса проверять наличие в его маркере доступа привилегий, указанных в варианте задания;

для привилегий, которые имеются в маркере доступа необходимо определить состояние.

3. Разработать в Visual C++ приложение Win32, которое для выбранного файла или каталога:

– должно выводить список разрешений (DACL);

– должно иметь возможность добавлять и удалять ACE;

– должно выводить и изменять имя учетной записи владельца;

4. Изменить приложениемWin32 разработанное в п.3 задания лабораторной работы №4, добавив возможность выполнять соответствующую операцию – **КОПИРОВАНИЕ ФАЙЛОВ И КАТАЛОГОВ** от имени задаваемой учетной записи.

Имя и пароль учетной записи пользователя должны запрашиваться всякий раз, когда приложение не может выполнить операцию с файлами и каталогами от имени текущего пользователя.

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

6. Исследовать работу разработанных приложений с помощью утилиты Process Explorer. Результаты исследовать отразить в отчете.

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

№	Хорошо известные SID (п. 1 задания)	Привилегии (п. 2 задания)
11,26	S-1-2-1 S-1-5-11 S-1-5-18 S-1-5-21-X-X-X-500 S-1-5-32-545 S-1-5-32-556	SeLoadDriverPrivilege SeLockMemoryPrivilege SeMachineAccountPrivilege SeManageVolumePrivilege SeProfileSingleProcessPrivilege

Рис. 5.1. задание для варианта 11

Ход работы

1. В соответствии с п.1. разработала приложение Win32, осуществляющего указанные действия: вывод SID, вывод списка привилегий и т.д.

Листинг 1. Приложение для определения информации по SID

```
#define LOCAL_FREE(ptr) if (NULL != (ptr)) LocalFree(ptr)
/*Функции и переменные*/
LSA_HANDLE OpenLocalPolicy(ACCESS_MASK AccessType);//открытие дескриптора политики безопасности
локального пк

BOOL GetAccountName_W(PSID psid, LPWSTR* AccountName);//определим имя аккаунта по его SID
BOOL GetWellKnowSID(WELL_KNOWN_SID_TYPE WellKnowSidType, PSID sidDomain, PSID
*sidWellKnow);//определение хорошо известного SID
BOOL GetAccountSID_W(LPCWSTR AccountName, PSID *ppsid); //узнаем SID ПК ИЛИ USER

void CoutSID(PSID psid);//вывод списков на экран
void ListOfPrivilegesAndRights_User(LSA_HANDLE PolicyHandle, PSID sidUser);//вывод списка
привелегий пользователя


int _tmain()
{
	_tsetlocale(LC_ALL, TEXT(""));

	/*Pointer Security ID*/
	PSID sidDomain = NULL;//security ID для локального ПК
	PSID sidUser = NULL;//security ID для локального пользователя
	PSID sidWellKnow;//для работы с хорошо известными SID


	//Вспомогательные переменные
	DWORD count_char;//для определения размеров
	BOOL RetRes = FALSE; //для результатов

	//Переменные для вывода имен ПК и пользователя*
	WCHAR ComputerName[MAX_COMPUTERNAME_LENGTH + 1] = TEXT("");
	WCHAR UserName[UNLEN + 1] = TEXT("");

	count_char = _countof(ComputerName);

	RetRes = GetComputerName(ComputerName, &count_char);//узнаем имя ПК

	if (RetRes != FALSE)
	{
		RetRes = GetAccountSID_W(ComputerName, &sidDomain);//определим SID ПК

		if ((RetRes != FALSE) && (sidDomain != NULL))
		{
			CoutSID(sidDomain);//выведем SID ПК на экран
		}
	}
/*-------------------------------------------------------------------------------------*/
	count_char = _countof(UserName);
	RetRes = GetUserName(UserName, &count_char);//узнаем имя пользователя

	LSA_HANDLE PolicyHandle = OpenLocalPolicy(POLICY_LOOKUP_NAMES);

	if (RetRes != FALSE)
	{
		RetRes = GetAccountSID_W(UserName, &sidUser);

		if ((RetRes != FALSE) && (sidUser != NULL))
		{
```

```
                    CoutSID(sidUser);
                    //вывод списка привелегий
                    if (PolicyHandle != NULL)
                           ListOfPrivilegesAndRights_User(PolicyHandle, sidUser);//НОРМ, ТАК И
ДОЛЖНО БЫТЬ.

                    LocalFree(sidUser);//повреждение кучи?
             }
      }
      /*---------------Работа с хорошо известными SID----------------------------*/
      constexpr WELL_KNOWN_SID_TYPE WellKnownType[] =
      {
             WinConsoleLogonSid,
             WinAuthenticatedUserSid,
             WinLocalSystemSid,
             WinAccountAdministratorSid,
             WinBuiltinUsersSid,
             WinBuiltinNetworkConfigurationOperatorsSid
      };

      for (int i = 0; i < _countof(WellKnownType); ++i)
      {
             RetRes = GetWellKnowSID(WellKnownType[i], sidDomain, &sidWellKnow);

             if ((RetRes != FALSE) && (sidWellKnow != NULL))
             {
                    CoutSID(sidWellKnow);
                    //вывод списка привелегий

                    if (PolicyHandle != NULL)
                    {
                           ListOfPrivilegesAndRights_User(PolicyHandle, sidWellKnow);
                    }
                    LocalFree(sidWellKnow);
             }
      }

      system("pause");
      LOCAL_FREE(sidDomain);

      LsaClose(PolicyHandle);

}

LSA_HANDLE OpenLocalPolicy(ACCESS_MASK AccessType)
{
      //POLICY_LOOKUP_NAMES - this access type is needed to translate between names and SIDs.
      LSA_HANDLE PolicyHandle;
      LSA_OBJECT_ATTRIBUTES ObjAtr;

      ZeroMemory(&ObjAtr, sizeof(ObjAtr));//Инициализация структуры LSA_OBJECT_ATTRIBUTES

      NTSTATUS ntstatus = LsaOpenPolicy(NULL, &ObjAtr, AccessType, &PolicyHandle);//получение
дескриптора политики безопасности

      SetLastError(LsaNtStatusToWinError(ntstatus)); //для ошибок

      return LSA_SUCCESS(ntstatus) ? PolicyHandle : NULL; //если завершено успешно, то вернуть
PolicyHandle

}

/*Определения функций определяющих SID*/
BOOL GetAccountSID_W(LPCWSTR AccountName, PSID *ppsid)
{
      BOOL RetRes = FALSE;
      SID_NAME_USE SidType;//переменная перечисляемого типа, сюда сохраним определенный тип SID
```

```
	/*Переменные для определения имени и SID*/
	LPWSTR RefDomainName = NULL;
	PSID psid = NULL;
	DWORD cbSID = 0, cchRefDomainName = 0;

	LookupAccountNameW(NULL, AccountName, NULL, &cbSID, NULL, &cchRefDomainName,
NULL);//определение размеров буфера под имена

	if ((cbSID > 0) && (cchRefDomainName > 0))
	{
		psid = (PSID)LocalAlloc(LMEM_FIXED, cbSID); //выделение памяти из локальной кучи
процесса
		RefDomainName = (LPWSTR)LocalAlloc(LMEM_FIXED, cchRefDomainName * sizeof(WCHAR));//
-||- для имени домена
	}

	if ((psid != NULL) && (RefDomainName != NULL))
	{
		RetRes = LookupAccountName(NULL, AccountName, psid, &cbSID, RefDomainName,
&cchRefDomainName, &SidType);
	}

	if (RetRes != FALSE)
	{
		*ppsid = psid;
	}
	else
	{
		if (psid != NULL)
		{
			LocalFree(psid);//освобождаем память
		}
	}

	if (RefDomainName != NULL)
	{
		LocalFree(RefDomainName);//освобождаем память
	}

	return RetRes;
}


BOOL GetAccountName_W(PSID psid, LPWSTR* AccountName)
{
	BOOL RetRes = FALSE;
	SID_NAME_USE SidType;//переменная перечисляемого типа, сюда сохраним определенный тип SID
	/*Переменные для вывода*/
	LPWSTR Name = NULL;
	DWORD cch = 0, cchRefDomainName = 0;

	if (IsValidSid(psid) == FALSE)
	{
		return FALSE;
	}

	LookupAccountSid(NULL, psid, NULL, &cch, NULL, &cchRefDomainName, NULL);//определим
размеры буферов

	DWORD cb = (cch + cchRefDomainName) * sizeof(TCHAR);

	if (cb > 0)
	{

		Name = (LPWSTR)LocalAlloc(LMEM_FIXED, cb);//выделение памяти из локальной кучи
процесса
```

```
	}

	if (Name != NULL)
	{
		RetRes = LookupAccountSid(NULL, psid, Name + cchRefDomainName, &cch, Name,
&cchRefDomainName, &SidType);
	}

	if (RetRes != FALSE)
	{
		if (SidTypeDomain != SidType)
		{
			if (cchRefDomainName > 0)
			{
				Name[cchRefDomainName] = '\\';
			}
			else
			{
				StringCbCopy(Name, cb, Name + 1);//копирование для вовзрата в основую
функциию
			}
		}
		*AccountName = Name; //вернем полученнное имя в программу
	}
	else
	{
		ConvertSidToStringSid(psid, AccountName);//если не получилось получить имя, то
вернем SID
		if (Name != NULL)
		{
			LocalFree(Name);
		}
	}
	return RetRes;
}

BOOL GetWellKnowSID(WELL_KNOWN_SID_TYPE WellKnowSidType, PSID sidDomain, PSID *sidWellKnow)
{
	DWORD MaxSidSize = SECURITY_MAX_SID_SIZE;

	PSID localSID = (PSID)LocalAlloc(LMEM_FIXED, MaxSidSize);

	if (localSID == NULL)
	{
		return FALSE; //определеить SID не удалось.
	}

	BOOL RetRes = CreateWellKnownSid(WellKnowSidType, sidDomain, localSID, &MaxSidSize);
	if (RetRes != FALSE)
	{
		*sidWellKnow = localSID;//возращаем созданный SID
	}
	else
	{
		LocalFree(localSID);
	}
	return RetRes;
}



void ListOfPrivilegesAndRights_User(LSA_HANDLE PolicyHandle, PSID sidUser)
{
	/*Переменные для вывода*/
	PLSA_UNICODE_STRING List_Rights; //список прав
	ULONG count_list;//элементы в списке прав
```

```
	/*Вспомогательные счетчики*/
	DWORD count_char, dwLangId;
	BOOL RetRes = FALSE;

	//Error: 0xc0000034 Определять права для группы?
	NTSTATUS ntstatus = LsaEnumerateAccountRights(PolicyHandle, sidUser, &List_Rights,
&count_list);

	if (LSA_SUCCESS(ntstatus))
	{
		wprintf(TEXT("\tСписок прав учетной записи:\n"));
		for (ULONG i = 0; i < count_list; ++i)
		{
			LPCWSTR UserRight = List_Rights[i].Buffer;

			wprintf(TEXT("\t  %2d. %s"), (i + 1), UserRight);

			TCHAR DisplayName[256];//дружественное имя

			count_char = _countof(DisplayName);//определеим количество символов в строке

			RetRes = LookupPrivilegeDisplayName(NULL, UserRight, DisplayName,
&count_char, &dwLangId);

			if (RetRes != FALSE)
			{
				wprintf(TEXT(" (%s)"), DisplayName);
			}
			else
			{
				/*Константы прав учетных записей*/
				constexpr LPCTSTR Right_array[20] =
				{
					SE_INTERACTIVE_LOGON_NAME, TEXT("Локальный вход в систему"),
					SE_DENY_INTERACTIVE_LOGON_NAME, TEXT("Запретить локальный
					вход"),
					SE_NETWORK_LOGON_NAME, TEXT("Доступ к компьютеру из сети"),
					SE_DENY_NETWORK_LOGON_NAME, TEXT("Отказать в доступе к этому
					компьютеру"),
					SE_BATCH_LOGON_NAME, TEXT("Вход в качестве пакетного задания"),
					SE_DENY_BATCH_LOGON_NAME, TEXT("Отказать во входе в качестве
					пакетного задания"),
					SE_SERVICE_LOGON_NAME, TEXT("Вход в качестве службы"),
					SE_DENY_SERVICE_LOGON_NAME, TEXT("Отказать во входе в качестве
					 службы"),
					SE_REMOTE_INTERACTIVE_LOGON_NAME, TEXT("Разрешать вход в
					систему через службы удаленных рабочих столов"),
					SE_DENY_REMOTE_INTERACTIVE_LOGON_NAME, TEXT("Запретить вход в
					систему через службы удаленных рабочих столов")
				};

				for (int j = 0; j < _countof(Right_array); j += 2)
				{
					if ((_tcscmp(Right_array[j], UserRight) == 0))
					{
						wprintf(TEXT(" (%s)"), Right_array[j + 1]);
						break;
					}
				}
			}
			wprintf(TEXT("\n"));
		}
		wprintf(TEXT("\n"));
		LsaFreeMemory(List_Rights);
	}
}
```

```
void CoutSID(PSID psid)
{
	LPWSTR lpSID = NULL, AccountName = NULL;//переменные для вывода
	ConvertSidToStringSidW(psid, &lpSID);
	GetAccountName_W(psid, &AccountName);
	if ((AccountName != NULL) && (psid != NULL))
	{
		wprintf(TEXT("%s (%s)\n\n"), AccountName, lpSID);
	}
	/*Освобождение памяти*/
	LOCAL_FREE(lpSID);
	LOCAL_FREE(AccountName);
	 }
```

2. Разработала программу в соответствии с указанным в п.2 заданием. Проверила ее с помощью утилиты Process Explorer.

Листинг 2. Приложение для отображения привилегий процессов по их SID

```
LRESULT CALLBACK MainWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
/*Обработчики сообщений WM_CREATE WM_DESTROY WM_SIZE WM_COMMAND */

BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct);
void OnDestroy(HWND hwnd);
void OnSize(HWND hwnd, UINT state, int cx, int cy);
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);

// ----------------------------------------------------------------------------------------------
--

		/*для токенов*/

HANDLE OpenProcessTokenPID(UINT PID, DWORD AccessRIGHT);//открыть процесс по его PID
BOOL   GetAccountName_W(PSID psid, LPWSTR* AccountName);//получить имя пользователя по его SID

BOOL   GetTokenUser(HANDLE token, PSID* psid);//узнать маркер доступа пользователя по его SID
BOOL   GetTokenGR(HANDLE token, PTOKEN_GROUPS *tk_groups);//узнать список ГРУПП по маркеру
доступа
BOOL   GetTokenPR(HANDLE token, PTOKEN_PRIVILEGES *tk_PR);//узнать список ПРИВИЛЕГИЙ по маркеру
доступа
BOOL   ExistPR(HWND hwnd, HANDLE token, LPCWSTR PrivilName);//узнать если ли искомая привлегия и
у процесса

		/*Загрузка процессов и модулей этих процессов в LISTBOX*/
	void   ToLB_LoadProcesses(HWND hwndCtl);//процессы в LB
	void   ToLB_LoadTokenGroup(HWND hwndListGroups, HANDLE token);//группы привелегий в LB
	void   ToLB_LoadTokenPrivil(HWND hwndListPrivileges, HANDLE token);
// ----------------------------------------------------------------------------------------------
--

		/*Кнопки*/
		BOOL SwitchOnPrivil(HANDLE hToken, LPCWSTR PrivilName, BOOL bEnable);

int WINAPI _tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpszCmdLine, int nCmdShow)
{
	LoadLibrary(TEXT("ComCtl32.dll"));//для элементов общего пользования


	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
	}

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

	// создаем главное окно на основе нового оконного класса

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

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


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

	MSG  msg;
	BOOL bRet;

	while ((bRet = GetMessage(&msg, NULL, 0, 0)) != FALSE)
	{
		if (!TranslateAccelerator(hWnd, hAccel, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}


	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_DESTROY, OnDestroy);
		HANDLE_MSG(hWnd, WM_SIZE, OnSize);
		HANDLE_MSG(hWnd, WM_COMMAND, OnCommand);
	}
	return DefWindowProc(hWnd, uMsg, wParam, lParam);	// передача необработанного
сообщения
}

/*Создание оконного приложения с 2-мя listbox и меню*/
BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct)
{
	//Создаем список открытых процессов
	CreateWindowEx(0, TEXT("Static"), TEXT("Процессы:"), WS_CHILD | WS_VISIBLE | SS_SIMPLE,
		10, 10, 400, 20, hwnd, NULL, lpCreateStruct->hInstance, NULL);
	HWND hwndCtl = CreateWindowEx(0, TEXT("ListBox"), TEXT(""), WS_CHILD | WS_VISIBLE |
LBS_STANDARD,
		10, 30, 400, 400, hwnd, (HMENU)IDC_LB_PROCESSES, lpCreateStruct->hInstance, NULL);
	ToLB_LoadProcesses(hwndCtl);// получаем список процессов активных сейчас
```

```
	//список для перечисления групп
	CreateWindowEx(0, TEXT("Static"), TEXT("Группы"), WS_CHILD | WS_VISIBLE | SS_SIMPLE,
		420, 10, 300, 20, hwnd, NULL, lpCreateStruct->hInstance, NULL);

	CreateWindowEx(0, TEXT("ListBox"), TEXT(""), WS_CHILD | WS_VISIBLE | WS_HSCROLL |
WS_VSCROLL | WS_BORDER,
		420, 30, 300, 300, hwnd, (HMENU)IDC_LB_GROUPS, lpCreateStruct->hInstance, NULL);


	//Поле "Пользователь"
	CreateWindowEx(0, TEXT("Static"), TEXT("Пользователь:"), WS_CHILD | WS_VISIBLE |
SS_SIMPLE,
		420, 330, 200, 20, hwnd, NULL, lpCreateStruct->hInstance, NULL);

	CreateWindowEx(0, TEXT("Edit"), TEXT(""), WS_CHILD | WS_VISIBLE | ES_LEFT | ES_READONLY |
ES_AUTOHSCROLL,
		630, 330, 400, 20, hwnd, (HMENU)IDC_EDIT_ACCOUNT, lpCreateStruct->hInstance, NULL);

	// Поле "SID"
	CreateWindowEx(0, TEXT("Static"), TEXT("SID:"), WS_CHILD | WS_VISIBLE | SS_SIMPLE,
		420, 380, 200, 20, hwnd, NULL, lpCreateStruct->hInstance, NULL);

	CreateWindowEx(0, TEXT("Edit"), TEXT(""), WS_CHILD | WS_VISIBLE | ES_LEFT | ES_READONLY |
ES_AUTOHSCROLL,
		630, 380, 400, 20, hwnd, (HMENU)IDC_EDIT_SID, lpCreateStruct->hInstance, NULL);

	//список для перечисления привилегий
	CreateWindowEx(0, TEXT("Static"), TEXT("Привилегии:"), WS_CHILD | WS_VISIBLE | SS_SIMPLE,
		730, 10, 300, 20, hwnd, NULL, lpCreateStruct->hInstance, NULL);

	CreateWindowEx(0, TEXT("ListBox"), TEXT(""), WS_CHILD | WS_VISIBLE | LBS_STANDARD,
		730, 30, 300, 300, hwnd, (HMENU)IDC_LB_PRIVILEGES, lpCreateStruct->hInstance, NULL);


	// Кнопки Вкл/Выкл
	hwndCtl = CreateWindowEx(0, TEXT("Button"), TEXT("Включить"), WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
		630, 480, 110, 30, hwnd, (HMENU)IDC_BUTTON_PRIVILEGE_ENABLE, lpCreateStruct-
>hInstance, NULL);
	EnableWindow(hwndCtl, FALSE);	// отключение "Включить"

	hwndCtl = CreateWindowEx(0, TEXT("Button"), TEXT("Выключить"), WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
		730, 480, 110, 30, hwnd, (HMENU)IDC_BUTTON_PRIVILEGE_DISABLE, lpCreateStruct-
>hInstance, NULL);
	EnableWindow(hwndCtl, FALSE);	// отключение "Выключить"




	//список привелегий для поиска
	CreateWindowEx(0, TEXT("Static"), TEXT("Для проверки:"), WS_CHILD | WS_VISIBLE |
SS_SIMPLE,
		420, 400, 300, 20, hwnd, NULL, lpCreateStruct->hInstance, NULL);
	hwndCtl = CreateWindowEx(0, TEXT("ListBox"), TEXT(""), WS_CHILD | WS_VISIBLE |
LBS_STANDARD,
		420, 430, 300, 50, hwnd, (HMENU)IDC_PRIVILEGES, lpCreateStruct->hInstance, NULL);


	constexpr LPCTSTR Privilege[5] =
	{
		SE_LOAD_DRIVER_NAME, SE_LOCK_MEMORY_NAME,
		SE_MACHINE_ACCOUNT_NAME, SE_MANAGE_VOLUME_NAME,
		SE_PROF_SINGLE_PROCESS_NAME
	};
	for (int i = 0; i < _countof(Privilege); ++i)
	{
```

```
			int iItem = ListBox_AddString(hwndCtl, Privilege[i]);
		}
		ListBox_SetCurSel(hwndCtl, 0);

		return TRUE;
}

/*При завершении работы с приложением*/
void OnDestroy(HWND hwnd)
{
		PostQuitMessage(0); // отправляем сообщение WM_QUIT
		 }
/*Список меняет ширину и высоту*/
void OnSize(HWND hwnd, UINT state, int cx, int cy)
{
		if (state != SIZE_MINIMIZED)
		{
			// изменяем высоту списка для перечисления процессов
			MoveWindow(GetDlgItem(hwnd, IDC_LB_PROCESSES), 10, 30, 400, cy - 40, TRUE);
		}
}

void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
		switch (id)
		{
		case IDC_LB_PROCESSES:
		{

			if (LBN_SELCHANGE == codeNotify) // выбран другой элемент в списке процессов
			{

				/*Очистка полей*/
				SetDlgItemText(hwnd, IDC_EDIT_ACCOUNT, NULL);// очистим поле "Пользователь"
				SetDlgItemText(hwnd, IDC_EDIT_SID, NULL);// очистим поле "SID"

				/*Получение hwnd для групп*/
				HWND hwndListGroups = GetDlgItem(hwnd, IDC_LB_GROUPS);
				ListBox_ResetContent(hwndListGroups);// очистим список групп

				/*Получение hwnd для привилегий*/
				HWND hwndListPrivileges = GetDlgItem(hwnd, IDC_LB_PRIVILEGES);
				ListBox_ResetContent(hwndListPrivileges);// очистим список  привилегий


				int iItem = ListBox_GetCurSel(hwndCtl);//номер выделенной строки процесса
				if (iItem != -1)
				{
					UINT PID = (UINT)ListBox_GetItemData(hwndCtl, iItem);// определение ID
процесса
					/*Маркеры доступа процесса*/
					HANDLE token = OpenProcessTokenPID(PID, TOKEN_QUERY);

					if (token != NULL)
					{
						//Получим SID пользователя
						PSID psid = NULL;
						BOOL RetRes = GetTokenUser(token, &psid);//узнать SID по
маркеру доступа пользователя
							if (RetRes != FALSE)
							{
								LPWSTR Name = NULL;
								GetAccountName_W(psid, &Name);//получить имя
пользователя

								if (Name != NULL)
								{
```

```
                                                SetDlgItemText(hwnd, IDC_EDIT_ACCOUNT,
Name);//записать имя пользователя в поле
                                                LocalFree(Name), Name = NULL;
                                        }

                                        ConvertSidToStringSid(psid,
&Name);//преобразование SID в строку

                                        if (Name != NULL)
                                        {
                                                SetDlgItemText(hwnd, IDC_EDIT_SID, Name);
                                                LocalFree(Name), Name = NULL;
                                        }

                                        //LocalFree(psid), psid = NULL;
                                }
                                ToLB_LoadTokenGroup(hwndListGroups, token);
                                ToLB_LoadTokenPrivil(hwndListPrivileges,token);
                                CloseHandle(token);
                        }
                }
        }
        } break;

        case IDC_LB_PRIVILEGES: // Обновление списка процессов
        {
                if (LBN_SELCHANGE == codeNotify)
                {

                        int iItem = ListBox_GetCurSel(hwndCtl);//номер выбранной привилегии

                        if (iItem != -1)
                        {

                                DWORD atrb = (DWORD)ListBox_GetItemData(hwndCtl, iItem);// определяем
атрибуты привилегии

                                if (atrb & SE_PRIVILEGE_ENABLED)
                                {

                                        EnableWindow(GetDlgItem(hwnd, IDC_BUTTON_PRIVILEGE_ENABLE),
FALSE);// отключим кнопку "Включить"
                                        EnableWindow(GetDlgItem(hwnd, IDC_BUTTON_PRIVILEGE_DISABLE),
TRUE);// включим кнопку "Выключить"
                                }
                                else
                                {
                                        EnableWindow(GetDlgItem(hwnd, IDC_BUTTON_PRIVILEGE_ENABLE),
TRUE);// включим кнопку "Включить"
                                        EnableWindow(GetDlgItem(hwnd, IDC_BUTTON_PRIVILEGE_DISABLE),
FALSE);// отключим кнопку "Выключить"
                                }
                        }
                }
        }       break;
case IDC_BUTTON_PRIVILEGE_ENABLE: // включить привилегию
        case IDC_BUTTON_PRIVILEGE_DISABLE: // выключить привилегию
        {
                HWND hwnd_ProcessList = GetDlgItem(hwnd, IDC_LB_PROCESSES);
                HWND hwnd_PrivilegList = GetDlgItem(hwnd, IDC_LB_PRIVILEGES);

                UINT PID;

                int item = ListBox_GetCurSel(hwnd_ProcessList);

                if (item != -1)
                {
```

```
                    PID = (UINT)ListBox_GetItemData(hwnd_ProcessList, item);//выбранный процесс
                    item = ListBox_GetCurSel(hwnd_PrivilegList); //выбранная привилегеия
             }

             if (item != -1)
             {
                    //HANDLE token = OpenProcessTokenPID(PID, TOKEN_QUERY);//получить маркер
доступа процесса
                    HANDLE token = OpenProcessTokenPID(PID, TOKEN_QUERY |
TOKEN_ADJUST_PRIVILEGES);//получить маркер доступа процесса
                    if (token != NULL)
                    {
                           TCHAR NamePrivil[256];
                           ListBox_GetText(hwnd_PrivilegList, item, NamePrivil);//получение имени
привилегии

                           LPTSTR priv = _tcschr(NamePrivil, TEXT(' ')); //для удаления суффикса
состояния привилегии

                           if (priv != NULL)
                           {
                                  *priv = TEXT('\0');
                           }

                           BOOL RetRes = SwitchOnPrivil(token, NamePrivil,
(IDC_BUTTON_PRIVILEGE_ENABLE == id) ? TRUE : FALSE);
                           if (FALSE != RetRes)
                           {
                                  // загрузим список привилегий
                                  ToLB_LoadTokenPrivil(hwnd_PrivilegList, token);

                                  // задаём текущий элемент списка привилегий
                                  ListBox_SetCurSel(hwnd_PrivilegList, item);
                                  // передаём фокус клавиатуры списку привилегий
                                  SetFocus(hwnd_PrivilegList);

                                  if (IDC_BUTTON_PRIVILEGE_ENABLE == id) // привилегия была
включена
                                  {
                                         // отключим кнопку "Включить"
                                         EnableWindow(GetDlgItem(hwnd,
IDC_BUTTON_PRIVILEGE_ENABLE), FALSE);
                                         // включим кнопку "Выключить"
                                         EnableWindow(GetDlgItem(hwnd,
IDC_BUTTON_PRIVILEGE_DISABLE), TRUE);
                                  } // if
                                  else // привилегия была выключена
                                  {
                                         // включим кнопку "Включить"
                                         EnableWindow(GetDlgItem(hwnd,
IDC_BUTTON_PRIVILEGE_ENABLE), TRUE);
                                         // отключим кнопку "Выключить"
                                         EnableWindow(GetDlgItem(hwnd,
IDC_BUTTON_PRIVILEGE_DISABLE), FALSE);
                                  } // else
                           } // if
                           else
                           {
                                  MessageBox(hwnd, TEXT("Не удалось изменить состояние
привилегии"), NULL, MB_OK);
                           }
                           // закрываем дескриптор маркера доступа
                           CloseHandle(token);
                    }
             }

      }break;
```

```
	case IDC_PRIVILEGE_CHECK:
	{
		HWND hwnd_ProcessList = GetDlgItem(hwnd, IDC_LB_PROCESSES);
		HWND hwnd_PrivilegList = GetDlgItem(hwnd, IDC_PRIVILEGES);

		UINT PID;
		BOOL RetRes = FALSE;
		int item = ListBox_GetCurSel(hwnd_ProcessList);
		if (item != -1)
		{
			PID = (DWORD)ListBox_GetItemData(hwnd_ProcessList, item);
			item = ListBox_GetCurSel(hwnd_PrivilegList);
		}
		if (item != -1)
		{
			HANDLE token = OpenProcessTokenPID(PID, TOKEN_QUERY);
			if (token != NULL)
			{
				WCHAR PrivilName[256];
				ListBox_GetText(hwnd_PrivilegList, item, PrivilName);
				RetRes = ExistPR(hwnd,token, PrivilName);
				if (RetRes == -1)
				{
					MessageBox(hwnd, L"Наличие привилегии в маркере не определено",
L"НЕПОПРАВИМАЯ ОШИБКА" ,MB_OK);
				}
				else
					if (RetRes == FALSE)
					{
						MessageBox(hwnd, L"Такой привилегии у процесса нет
",L"Результат" ,MB_OK);
					}
					/*else
						if (RetRes == TRUE)
							{
								MessageBox(hwnd, L"Такая привелегия у
процесса есть и она включена", L"Результат", MB_OK);
							}*/
				CloseHandle(token);
			}
			else
			{
				MessageBox(hwnd, L"Маркер доступа открыть не удалось", L"Результат",
MB_OK);
			}
	}


	}break;

	}
}


/*Определение токена по идентификатору процесса*/
HANDLE OpenProcessTokenPID(UINT PID, DWORD AccessRIGHT)
{
	HANDLE token = NULL;
	HANDLE process = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, PID);

	if (process != NULL)
	{
		OpenProcessToken(process, AccessRIGHT , &token);//определение маркера доступа по PID
		CloseHandle(process);
	}
```

```
	return token;
}

/*Получение имени аккаунта по его SID*/
BOOL GetAccountName_W(PSID psid, LPWSTR* AccountName)
{
	BOOL RetRes = FALSE;
	SID_NAME_USE SidType;//переменная перечисляемого типа, сюда сохраним определенный тип SID
	/*Переменные для вывода*/
	LPWSTR Name = NULL;
	DWORD cch = 0, cchRefDomainName = 0;

	if (IsValidSid(psid) == FALSE)
	{
		return FALSE;
	}

	LookupAccountSid(NULL, psid, NULL, &cch, NULL, &cchRefDomainName, NULL);//определим
размеры буферов

	DWORD cb = (cch + cchRefDomainName) * sizeof(TCHAR);

	if (cb > 0)
	{

		Name = (LPWSTR)LocalAlloc(LMEM_FIXED, cb);//выделение памяти из локальной кучи
процесса
	}

	if (Name != NULL)
	{
		RetRes = LookupAccountSid(NULL, psid, Name + cchRefDomainName, &cch, Name,
&cchRefDomainName, &SidType);
	}

	if (RetRes != FALSE)
	{
		if (SidTypeDomain != SidType)
		{
			if (cchRefDomainName > 0)
			{
				Name[cchRefDomainName] = '\\';
			}
			else
			{
				StringCbCopy(Name, cb, Name + 1);//копирование для вовзрата в основую
функциию
			}
		}
		*AccountName = Name; //вернем полученнное имя в программу
	}
	else
	{
		ConvertSidToStringSid(psid, AccountName);//если не получилось получить имя, то
вернем SID
		if (Name != NULL)
		{
			LocalFree(Name);
		}
	}
	return RetRes;
}

/*To ListBox load's*/

void ToLB_LoadProcesses(HWND hwndCtl)
{
```

```
	ListBox_ResetContent(hwndCtl);//очистка списка

	DWORD PIDarray[1024], cbNeeded = 0;//массив для ID созданных процессов
	BOOL bRet = EnumProcesses(PIDarray, sizeof(PIDarray), &cbNeeded);//получение списка ID
созданных процессоров

	if (FALSE != bRet)
	{
		TCHAR ProcessName[MAX_PATH], szBuffer[300];

		for (DWORD i = 0,
			n = cbNeeded / sizeof(DWORD); i < n; ++i)
		{
			DWORD PID = PIDarray[i], cch = 0;
			if (0 == PID) continue;


			HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ,
FALSE, PID);//получение дескриптора процесса по его ID

			if (NULL != hProcess)
			{
				cch = GetModuleBaseName(hProcess, NULL, ProcessName,
_countof(ProcessName));// получаем имя главного модуля процесса

				CloseHandle(hProcess); // закрываем объект ядра
			}

			if (0 == cch)
				StringCchCopy(ProcessName, MAX_PATH, TEXT("Имя процесса не
определено"));

			StringCchPrintf(szBuffer, _countof(szBuffer), TEXT("%s (PID: %u)"),
ProcessName, PID);


			int iItem = ListBox_AddString(hwndCtl, szBuffer);

			ListBox_SetItemData(hwndCtl, iItem, PID);//запись в ListBox имени процесса
		}
	}
}

void ToLB_LoadTokenGroup(HWND hwndListGroups, HANDLE token)
{
	ListBox_ResetContent(hwndListGroups);
	TOKEN_GROUPS* tk_groups = NULL;

	if (GetTokenGR(token, &tk_groups) != FALSE)
	{
		for (UINT i = 0; i < tk_groups->GroupCount; ++i)
		{
			LPWSTR Name = NULL;
			GetAccountName_W(tk_groups->Groups[i].Sid, &Name);

			if (Name != NULL)
			{
				ListBox_AddString(hwndListGroups, Name);
				LocalFree(Name), Name = NULL;
			}
		}
		LocalFree(tk_groups);
	}
}

void ToLB_LoadTokenPrivil(HWND hwndListPrivileges, HANDLE token)
{
```

```
	ListBox_ResetContent(hwndListPrivileges);
	TOKEN_PRIVILEGES* tk_PR = NULL;

	if (GetTokenPR(token, &tk_PR) != FALSE)
	{
		TCHAR Name[256];
		for (UINT i = 0; i < tk_PR->PrivilegeCount; ++i)
		{
			//LUID -   structure is an opaque structure that specifies an
			// identifier that is guaranteed to be unique on the local machine
			LUID Luid = tk_PR->Privileges[i].Luid;

			DWORD count_char = _countof(Name);//определение имени привилегии
			LookupPrivilegeName(NULL, &Luid, Name, &count_char);

			DWORD atrb = tk_PR->Privileges[i].Attributes;

			if (atrb & SE_PRIVILEGE_ENABLED)
			{
				StringCchCat(Name, _countof(Name), TEXT(" (включена)")); //добавление
суффикса для отображения состояния привилегии
			}
				int item = ListBox_AddString(hwndListPrivileges, Name);//запись имени
привилегии
				ListBox_SetItemData(hwndListPrivileges, item, atrb);//запись
полученных атрибутов
		}
		LocalFree(tk_PR);
	}
}


/*Токены*/
BOOL GetTokenGR(HANDLE token, PTOKEN_GROUPS *tk_groups)
{
	DWORD tkSize;
	BOOL RetRes = FALSE;
	GetTokenInformation(token, TokenGroups, NULL, 0, &tkSize);//определение размера блока
памяти

	PTOKEN_GROUPS Groups = (PTOKEN_GROUPS)LocalAlloc(LPTR, tkSize);

	if (Groups == NULL) return FALSE;

	if (GetTokenInformation(token, TokenGroups, Groups, tkSize, &tkSize))//получение списка
привелегий в маркере доступа
	{
		RetRes = TRUE;
		*tk_groups = Groups;
	}
	else
	{
		LocalFree(Groups);
	}
	return RetRes;
}

BOOL GetTokenUser(HANDLE token, PSID *psid)
{
	DWORD tkSize;
	TOKEN_USER *tkUser;
	DWORD sidLeng;
	BOOL RetRes = FALSE;

	GetTokenInformation(token, TokenUser, NULL, 0, &tkSize);//определение размера
	tkUser = (TOKEN_USER*)malloc(tkSize); //выделение блока памяти
	if (tkUser == NULL)
```

```
	{
		return E_OUTOFMEMORY;
	}

	if (GetTokenInformation(token, TokenUser, tkUser, tkSize, &tkSize))
	{
		sidLeng = GetLengthSid(tkUser->User.Sid);
		PSID npsid = NULL;
		npsid = (PSID)malloc(sidLeng);
		if (npsid == NULL)
		{
			return E_OUTOFMEMORY;
		}

		RetRes = CopySid(sidLeng, npsid, tkUser->User.Sid); //копируем опередлнный SID
перед возвратом

		if (RetRes != FALSE)
		{
			*psid = npsid;
		}
		else
		{
			LocalFree(npsid);
			//free(tkUser)
		}
	}

	return RetRes;
}

BOOL GetTokenPR(HANDLE token, PTOKEN_PRIVILEGES *tk_PR)
{
	DWORD tkSize;
	BOOL RetRes = FALSE;
	GetTokenInformation(token, TokenPrivileges, NULL, 0, &tkSize);//определение размера блока
памяти

	PTOKEN_PRIVILEGES privil = (PTOKEN_PRIVILEGES)LocalAlloc(LPTR, tkSize);

	if (privil == NULL) return FALSE;

	if (GetTokenInformation(token, TokenPrivileges, privil, tkSize, &tkSize))//получение
списка привелегий в маркере доступа
	{
		RetRes = TRUE;
		*tk_PR = privil;
	}
	else
	{
		LocalFree(privil);
	}
	return RetRes;
}

BOOL ExistPR(HWND hwnd, HANDLE token, LPCWSTR PrivilName)
{
	LUID luid;
	BOOL RetRes, ResSetPriv, result;

	PTOKEN_PRIVILEGES tk_PR = NULL;
	PRIVILEGE_SET privil_set;//проверим состояние в которое установлена привилегия

	privil_set.PrivilegeCount = 1;//количество привилегий
	privil_set.Control = PRIVILEGE_SET_ALL_NECESSARY;//if all of the privileges must be
enabled;
	//or set it to zero if it is sufficient that any one of the privileges be enabled.
```

```
	RetRes = LookupPrivilegeValueW(NULL,PrivilName,&luid);
	ResSetPriv = LookupPrivilegeValueW(NULL, PrivilName, &privil_set.Privilege[0].Luid); //а они случайно одно и то же не делают?

	if (RetRes != FALSE)
	{
		RetRes = GetTokenPR(token, &tk_PR);
	}
	if (RetRes != FALSE)
	{
		for (UINT i = 0; i < tk_PR->PrivilegeCount; ++i)
		{
			int res = memcmp(&tk_PR->Privileges[i].Luid, &luid, sizeof(LUID));
			if (res == 0)
			{
				RetRes = TRUE;
				//выясним состояние привилегии -> включена/выключена
				if (ResSetPriv != FALSE)
				{
					ResSetPriv = PrivilegeCheck(token, &privil_set, &result);
					if (result != FALSE)
					{
						MessageBox(hwnd, L"Такая привелегия у процесса есть и она включена", L"Результат", MB_OK);
					}
					else
					{
						MessageBox(hwnd, L"Такая привелегия у процесса есть и она выключена", L"Результат", MB_OK);
					}

				}

			}
		}
	}
	else { return -1; }

	LocalFree(tk_PR);//освобождение памяти

	return RetRes;
}

/*Кнопки*/
BOOL SwitchOnPrivil(HANDLE hToken, LPCWSTR PrivilName, BOOL bEnable)
{
	TOKEN_PRIVILEGES	tokenPriv;

	tokenPriv.PrivilegeCount = 1; // количество привилегий

	// установим новое состояние указанной привилегии
	tokenPriv.Privileges[0].Attributes = (FALSE != bEnable) ? SE_PRIVILEGE_ENABLED : 0;

	// определяем LUID указанной привилегии
	BOOL bRet = LookupPrivilegeValue(NULL, PrivilName, &tokenPriv.Privileges[0].Luid);

	if (FALSE != bRet)
	{
		// изменяем состояние указанной привилегии
		bRet = AdjustTokenPrivileges(hToken, FALSE, &tokenPriv, sizeof(TOKEN_PRIVILEGES), NULL, NULL);//ошибка здесь
	} // if

	return bRet;
}
```

3. В соответствии с указанным в п. 3 заданием разработала приложение Win32, с возможностью просмотра списка контроля доступа для файла/каталога и установки элементов контроля доступа.

Листинг 3. Приложение для просмотра DACL

```
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);

	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 (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);
}

// ----------------------------------------------------------------------------------------------
INT_PTR CALLBACK DialogAceProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
	switch (uMsg)
	{
	case WM_INITDIALOG:
	{
		BOOL bRet = HANDLE_WM_INITDIALOG(hwndDlg, wParam, lParam, DialogAce_OnInitDialog);
		return SetDlgMsgResult(hwndDlg, uMsg, bRet);
	}

	case WM_CLOSE:
		EndDialog(hwndDlg, IDCLOSE);
		return TRUE;

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

	return FALSE;
}

BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCRStr)
{
	CreateWindowEx(0, TEXT("Edit"), NULL, WS_CHILD | WS_VISIBLE | WS_BORDER, 30, 10, 400, 20,
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, 50, 400, 150, hwnd, (HMENU)IDC_LIST1, 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, 100, (LPTSTR)TEXT("Тип") },
		{ LVCF_WIDTH | LVCF_TEXT, 0, 100, (LPTSTR)TEXT("Имя") },
		{ LVCF_WIDTH | LVCF_TEXT, 0, 100, (LPTSTR)TEXT("Применять к:") },

	};

	for (int i = 0; i < _countof(lvColumns); ++i)
	{
		// вставляем столбец
		ListView_InsertColumn(hwndLV, i, &lvColumns[i]);
	}

	CreateWindowEx(0, TEXT("Static"), TEXT("Владелец:"), WS_CHILD | WS_VISIBLE | SS_SIMPLE,
		30, 220, 80, 20, hwnd, NULL, lpCRStr->hInstance, NULL);
	CreateWindowEx(0, TEXT("Edit"), NULL, WS_CHILD | WS_VISIBLE | WS_BORDER, 110, 220, 310,
20, hwnd, (HMENU)IDC_EDIT_OWNER, lpCRStr->hInstance, NULL);

	CreateWindowEx(0, TEXT("Static"), TEXT("Изменить на:"), WS_CHILD | WS_VISIBLE | SS_SIMPLE,
		30, 250, 80, 20, hwnd, NULL, lpCRStr->hInstance, NULL);
	CreateWindowEx(0, TEXT("Edit"), NULL, WS_CHILD | WS_VISIBLE | WS_BORDER, 110, 250, 310,
20, hwnd, (HMENU)IDC_NEW_OWNER, lpCRStr->hInstance, NULL);

	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
	{
		BROWSEINFO bi;//structure for open special box with folder in treview
		LPITEMIDLIST pidl;
		ZeroMemory(&bi, sizeof(bi));
		bi.hwndOwner = NULL;
		bi.pszDisplayName = FileName;
		bi.lpszTitle = TEXT("Select file");
		bi.ulFlags = BIF_BROWSEINCLUDEFILES;
		pidl = SHBrowseForFolder(&bi);//open window for select
		if (pidl)
		{
			SHGetPathFromIDList(pidl, FileName);//get path
			if (!(ListViewInit(FileName, hwnd)))
			{
				GetLastError();
				break;
			}
		}
	}
	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
       {

       }

       case ID_CHANGE_OWNER://переименование без сохранения атрибутов
       {
             LPWSTR Owner = NULL;
             GetDlgItemText(hwnd, IDC_EDIT_OWNER, Owner, sizeof(Owner));// копируем имя учетной
записи владельца в поле "Текущий владелец"

             TCHAR NewOwner[UNLEN + 1]; // новое имя владельца

             GetDlgItemText(hwnd, IDC_NEW_OWNER, NewOwner, _countof(NewOwner));//это имя и его к
указателю lpszFileName
             /*новый код*/
             PSID psid;
             BOOL bret = GetAccountSID_W(NewOwner, &psid);
             if (bret != FALSE)
             {
                    if (SetFileSecurityInfo(FileName, NewOwner, 0, NULL, FALSE) == TRUE)
                    {
                           SetDlgItemText(hwnd, IDC_NEW_OWNER, NULL);
                    }
                    else
                    {
                           DWORD dwRetCode = GetLastError();

                           if (dwRetCode == ERROR_INVALID_OWNER)
                           {
                                  MessageBox(hwnd, TEXT("Этот пользователь не может быть
владельцем файла или каталога"), NULL, MB_OK | MB_ICONERROR);
                                  //Тогда добавим этого пользователя в список DALC. Он будет без
прав делать что либо
                                  EXPLICIT_ACCESS ea;
                                  ea.grfAccessPermissions = 0;
                                  ea.grfAccessMode = GRANT_ACCESS;
                                  ea.grfInheritance = NO_INHERITANCE;
                                  ea.Trustee.TrusteeForm = TRUSTEE_IS_SID;
                                  ea.Trustee.ptstrName = (LPWSTR)psid;
                                  if (SetFileSecurityInfo(FileName, NULL, 1, &ea, TRUE) == TRUE)
                                  {
                                         SetDlgItemText(hwnd, IDC_NEW_OWNER, NULL);
                                  }
                                  else
                                  {
                                         MessageBox(hwnd, TEXT("Не удалось сменить владельца.
Проверьте правильность введенного имени пользователя."), NULL, MB_OK | MB_ICONERROR);
                                  }
                           }
                           else
                           {
                                  MessageBox(hwnd, TEXT("Не удалось сменить владельца. Проверьте
правильность введенного имени пользователя."), NULL, MB_OK | MB_ICONERROR);
                           }

                    }
                    LocalFree(psid);
```

```
                    Edit_SetText(GetDlgItem(hwnd, IDC_NEW_OWNER), TEXT(""));
                    //BOOL RetRes = SetFileSecurityInfo(FileName, NewOwner, 0, NULL, FALSE);
             }
             ListViewInit(FileName, hwnd);
      }
      break;

      case  ID_ADD_ACE:
      {
             int mdRes = DialogBox(GetWindowInstance(hwnd), MAKEINTRESOURCE(IDD_DIALOG_ACE),
hwnd, DialogAceProc);// открытие диалоговое окно "Элемент разрешения"

             if (IDOK == mdRes)
             {
                    ListViewInit(FileName, hwnd);// выведем список с разрешениями
             }
      }
      break;
      case ID_DELETE_ACE:
      {
             // список просмотра DACL - список разграничиваемого контроля доступа
             HWND hwndLV = GetDlgItem(hwnd, IDC_DACL);
             BOOL RetRes = FALSE;
             PACL lpDacl;
             BOOL bDaclPresent, bDaclDefaulted;
             DWORD dwRetCode;       // код возврата
             /*for (;;)
             {*/

             int item = ListView_GetNextItem(hwndLV, -1, LVNI_SELECTED);// находим выделенный
элемент в списке просмотра DACL

             if (item == -1) // если нет выделенных элементов
             {
                    break;
             }

      // получаем список DACL из дескриптора безопасности
             if (!GetSecurityDescriptorDacl(
                    Sec_Descriptor,              // адрес дескриптора безопасности
                    &bDaclPresent,     // признак присутствия списка DACL
                    &lpDacl,           // адрес указателя на DACL
                    &bDaclDefaulted))  // признак списка DACL по умолчанию
                           {
                                  dwRetCode = GetLastError();
                                  MessageBox(hwnd, TEXT("Не удалось сменить владельца. Проверьте
правильность введенного имени пользователя."), dwRetCode), NULL, MB_OK | MB_ICONERROR);
                                  //printf("Get security descriptor DACL failed.\n");
                                  //printf("Error code: %d\n", dwRetCode);
                                  break;
                           }
             else
             {
                           RetRes = DeleteAce(lpDacl, item);// удаляем элемент из DACL
             }


             ListView_DeleteItem(hwndLV, item);// удаляем элемент из списка просмотра DACL
             //} // for

             SetFileSecurity(FileName, DACL_SECURITY_INFORMATION, Sec_Descriptor);            //
изменяем дескриптор безопасности в файле/каталоге
      }
             break;
      case ID_LOAD:
             ListViewInit(FileName, hwnd);
             break;
```

```
	case ID_EXIT:
		SendMessage(hwnd, WM_CLOSE, 0, 0);
		RegCloseKey(hKey);
		break;
	}
}
BOOL ListViewInit(LPTSTR path, HWND hwnd)
{
	WIN32_FILE_ATTRIBUTE_DATA bhfi;
	BOOL RetRes;

	if (!GetFileAttributesEx(path, GetFileExInfoStandard, &bhfi))
	{
		GetLastError();
	}
	/*Добавление имени*/
	LPTSTR lpFN = PathFindFileNameW(path);
	SetDlgItemText(hwnd, IDC_EDIT_FILENAME, lpFN);

	/*Инициализация списка*/
	HWND hwndLV = GetDlgItem(hwnd, IDC_LIST1);
	ListView_DeleteAllItems(hwndLV); //очистка списка просмотра

	// освобождаем выделенную память
	if (NULL != Sec_Descriptor)
		LocalFree(Sec_Descriptor),
		Sec_Descriptor = NULL;


	RetRes = GetFileSecurityDescriptor(FileName, OWNER_SECURITY_INFORMATION |
DACL_SECURITY_INFORMATION, &Sec_Descriptor);// получение дескриптора безопасности

	if (RetRes != FALSE)
	{
		LPWSTR Owner = NULL;
		RetRes = GetOwnerName_W(Sec_Descriptor, &Owner);//получение имени владельца
		SetDlgItemText(hwnd, IDC_EDIT_OWNER, Owner);// копируем имя учетной записи владельца
в поле "Текущий владелец"
		LocalFree(Owner);
	}

	ULONG uCount = 0; // количество элементов в массиве ACE
	PEXPLICIT_ACCESS pEA = NULL; // массив ACE

	RetRes = GetItemFromDACL(Sec_Descriptor, &uCount, &pEA);//узнаем количество элементов в
DACL

	if (RetRes != FALSE)
	{
		for (ULONG i = 0; i < uCount; ++i)
		{
			LVITEM lvItem = { LVIF_TEXT | LVIF_PARAM };

			lvItem.iItem = (int)i;

			lvItem.lParam = (LPARAM)pEA[i].grfAccessPermissions;		//
определим права доступа

			// определим тип ACE
			switch (pEA[i].grfAccessMode)
			{
			case GRANT_ACCESS:
				lvItem.pszText = (LPTSTR)TEXT("Разрешить");
				break;
			case DENY_ACCESS:
				lvItem.pszText = (LPTSTR)TEXT("Запретить");
```

```
                        break;
                }

                // добавляем новый элемент в список просмотра DACL
                int iItem = ListView_InsertItem(hwndLV, &lvItem);
                if (iItem == -1)
                        continue; // не удалось добавить новый элемент

                 // определим имя учетной записи
                if (TRUSTEE_IS_SID == pEA[i].Trustee.TrusteeForm)
                {
                        LPTSTR lpszName = NULL; // имя учетной записи

                        GetAccountName_W(pEA[i].Trustee.ptstrName, &lpszName);// получим имя
учетной записи

                        if (NULL != lpszName)
                        {

                                ListView_SetItemText(hwndLV, iItem, 1, lpszName);// копируем
имя учетной записи в ячейку списка просмотра DACL

                                LocalFree(lpszName);
                        }
                }

                DWORD grfInheritance = pEA[i].grfInheritance & (~INHERIT_NO_PROPAGATE);//
определим к каким обьектам применяются права доступа

                for (int j = 0; j < _countof(dwInherit); ++j)
                {
                        if (grfInheritance == dwInherit[j]) // найдено значение
                        {

                                ListView_SetItemText(hwndLV, iItem, 2,
(LPTSTR)szInheritText[j]);// копируем в ячейку списка просмотра DACL описание для значения поля
grfInheritance

                                break; // выходим из цикла
                        }
                }
        }
    }

    return TRUE;
}
/*Дескрипторы*/
BOOL GetFileSecurityDescriptor(LPCWSTR lpFileName, SECURITY_INFORMATION RequestedInformation,
PSECURITY_DESCRIPTOR *ppSD)
{
    DWORD lpnLengthNeeded = 0;

    GetFileSecurity(lpFileName, RequestedInformation, NULL, 0, &lpnLengthNeeded);    //
определим размер дескриптора безопасности

    PSECURITY_DESCRIPTOR pSD = (PSECURITY_DESCRIPTOR)LocalAlloc(LMEM_FIXED,
lpnLengthNeeded);// выделение памяти для дескриптора безопасности
    if (NULL == pSD) return FALSE;

    BOOL RetRes = GetFileSecurity(lpFileName, RequestedInformation, pSD, lpnLengthNeeded,
&lpnLengthNeeded);// получим дескриптор безопасности

    if (FALSE != RetRes)
    {
        *ppSD = pSD; // возвращаем полученный дескриптор безопасности
    }
    else
```

```
	{
		LocalFree(pSD); // освобождаем выделенную память
	}

	return RetRes;
}

BOOL GetItemFromDACL(PSECURITY_DESCRIPTOR pSecurityDescriptor, PULONG pcCountOfEntries,
PEXPLICIT_ACCESS *pListOfEntries)
{
	PACL pDacl = NULL;// указатель на список управления доступом
	BOOL lpbDaclPresent = FALSE; // признак присутствия списка DACL
	BOOL lpbDaclDefaulted = FALSE; // признак списка DACL по умолчанию
	BOOL RetRes;
	DWORD Result; //код результата извлечения элментов из DALC



	// получаем DACL
	RetRes = GetSecurityDescriptorDacl(pSecurityDescriptor, &lpbDaclPresent, &pDacl,
&lpbDaclDefaulted);

	if (RetRes != FALSE && lpbDaclPresent != FALSE)
	{
		// читаем элементы из списка DACL
		Result = GetExplicitEntriesFromAcl(
			pDacl,                    // адрес списка DACL
			pcCountOfEntries,    // адрес для количества элементов
			pListOfEntries);     // адрес указателя на буфер

		RetRes = (ERROR_SUCCESS == Result) ? TRUE : FALSE;
	}
	else
	{
		*pcCountOfEntries = 0; // возвращаем 0 элементов
	}

	return RetRes;
}
/*Получение имени аккаунта по его SID и по имени аккаунта*/
BOOL GetAccountName_W(PSID psid, LPWSTR* AccountName)
{
	BOOL RetRes = FALSE;
	SID_NAME_USE SidType;//переменная перечисляемого типа, сюда сохраним определенный тип SID
	/*Переменные для вывода*/
	LPWSTR Name = NULL;
	DWORD cch = 0, cchRefDomainName = 0;

	if (IsValidSid(psid) == FALSE)
	{
		return FALSE;
	}

	LookupAccountSid(NULL, psid, NULL, &cch, NULL, &cchRefDomainName, NULL);//определим
размеры буферов

	DWORD cb = (cch + cchRefDomainName) * sizeof(TCHAR);

	if (cb > 0)
	{

		Name = (LPWSTR)LocalAlloc(LMEM_FIXED, cb);//выделение памяти из локальной кучи
процесса
	}

	if (Name != NULL)
	{
```

```
		RetRes = LookupAccountSid(NULL, psid, Name + cchRefDomainName, &cch, Name,
&cchRefDomainName, &SidType);
	}

	if (RetRes != FALSE)
	{
		if (SidTypeDomain != SidType)
		{
			if (cchRefDomainName > 0)
			{
				Name[cchRefDomainName] = '\\';
			}
			else
			{
				StringCbCopy(Name, cb, Name + 1);//копирование для вовзрата в основую
функциию
			}
		}
		*AccountName = Name; //вернем полученнное имя в программу
	}
	else
	{
		ConvertSidToStringSid(psid, AccountName);//если не получилось получить имя, то
вернем SID
		if (Name != NULL)
		{
			LocalFree(Name);
		}
	}
	return RetRes;
}
/*Определения функций определяющих SID*/
BOOL GetAccountSID_W(LPCWSTR AccountName, PSID *ppsid)
{
	BOOL RetRes = FALSE;
	SID_NAME_USE SidType;//переменная перечисляемого типа, сюда сохраним определенный тип SID

	/*Переменные для определения имени и SID*/
	LPWSTR RefDomainName = NULL;
	PSID psid = NULL;
	DWORD cbSID = 0, cchRefDomainName = 0;

	LookupAccountNameW(NULL, AccountName, NULL, &cbSID, NULL, &cchRefDomainName,
NULL);//определение размеров буфера под имена

	if ((cbSID > 0) && (cchRefDomainName > 0))
	{
		psid = (PSID)LocalAlloc(LMEM_FIXED, cbSID); //выделение памяти из локальной кучи
процесса
		RefDomainName = (LPWSTR)LocalAlloc(LMEM_FIXED, cchRefDomainName * sizeof(WCHAR));//
-||- для имени домена
	}

	if ((psid != NULL) && (RefDomainName != NULL))
	{
		RetRes = LookupAccountName(NULL, AccountName, psid, &cbSID, RefDomainName,
&cchRefDomainName, &SidType);
	}

	if (RetRes != FALSE)
	{
		*ppsid = psid;
	}
	else
	{
		if (psid != NULL)
		{
```

```
                        LocalFree(psid);//освобождаем память
                }
        }

        if (RefDomainName != NULL)
        {
                LocalFree(RefDomainName);//освобождаем память
        }

        return RetRes;
}


/*Получение имени пользователя по дескриптору безопасности*/
BOOL GetOwnerName_W(PSECURITY_DESCRIPTOR Sec_Descriptor, LPWSTR *OwnerName)
{
        PSID psid;
        BOOL bDefaulted;

        BOOL bRet = GetSecurityDescriptorOwner(Sec_Descriptor, &psid, &bDefaulted);// получаем SID
владельца

        if (FALSE != bRet)
        {
                bRet = GetAccountName_W(psid, OwnerName);// определяем имя учетной записи владельца
        }
        return bRet;
}

/*Изменение информации в дескрипторе безопасности*/
BOOL SetFileSecurityInfo(LPCTSTR FileName, LPWSTR NewOwner,ULONG CountOfEntries, PEXPLICIT_ACCESS
pListOfEntries, BOOL bMergeEntries)
{
        BOOL RetRes = FALSE;
        SECURITY_DESCRIPTOR secur_desc;

        /*Выделяем буферы для новых значений*/
        PSID psid_Owner = NULL;
        PACL pNewDacl = NULL;

        RetRes = InitializeSecurityDescriptor(&secur_desc, SECURITY_DESCRIPTOR_REVISION);

        if (RetRes != FALSE && NewOwner != NULL)
        {

                RetRes = GetAccountSID_W(NewOwner, &psid_Owner);//по имени владельца получим его SID


                if (RetRes != FALSE)
                {
                        RetRes = SetSecurityDescriptorOwner(&secur_desc, psid_Owner, FALSE);//для
связи дескриптора с SID
                }
        }

        if (RetRes != FALSE && CountOfEntries > 0 && pListOfEntries != NULL)
        {
                PSECURITY_DESCRIPTOR OldSD = NULL;
                PACL pOldDacl = NULL; // указатель на буфер для DACL

                BOOL DaclDefaulted = FALSE;
                BOOL DaclPresent;

                if (bMergeEntries != FALSE)
                {
                        RetRes = GetFileSecurityDescriptor(FileName, DACL_SECURITY_INFORMATION,
&OldSD);
```

```
                    if (RetRes != FALSE)
                    {
                          GetSecurityDescriptorDacl(OldSD, &DaclPresent, &pOldDacl,
&DaclDefaulted);
                    }
             }

             DWORD result = SetEntriesInAcl(CountOfEntries, pListOfEntries, pOldDacl, &pNewDacl);
             RetRes = (ERROR_SUCCESS == result) ? TRUE:FALSE;
             if (RetRes != FALSE)
             {
                    RetRes = SetSecurityDescriptorDacl(&secur_desc, TRUE, pNewDacl,
DaclDefaulted);
             }
             if (OldSD != NULL)
                    LocalFree(OldSD);
      }

      // проверяем структуру дескриптора безопасности
      if (!IsValidSecurityDescriptor(&secur_desc))
      {
             DWORD dwRetCode = GetLastError();

      }



      if (RetRes != NULL)
      {
             SECURITY_INFORMATION si = 0;
             if (psid_Owner != NULL) si |= OWNER_SECURITY_INFORMATION;
             if (pNewDacl != NULL) si |= DACL_SECURITY_INFORMATION;

             /*Если пользователь который не принадлежит группе администраторов пытается изменить
себя на владельца
             То это приведет к появлению ошикби INVALID_OWNER*/
             RetRes = SetFileSecurity(FileName, si, &secur_desc); //изменяем дескриптор
безопасности для файла
      }

      /*Освобождение памяти*/
      if (psid_Owner != NULL)
      {
             LocalFree(psid_Owner);
      }
      if (pNewDacl != NULL)
      {
             LocalFree(pNewDacl);
      }

      return RetRes;

}
BOOL DialogAce_OnInitDialog(HWND hwnd, HWND hwndFocus, LPARAM lParam)
{
      /*Variable*/
      PEXPLICIT_ACCESS pEA = (PEXPLICIT_ACCESS)lParam; // указатель на массив элементов типа
                                                                                  //
EXPLICIT_ACCESS
      DWORD grfAccessMode;// для получения режима доступа
      HWND hwndCtl;//дескриптор для списка

      BOOL fEditable = (NULL == pEA) ? TRUE : FALSE; // флаг, определяющий возможность
редактировать элемент ACE


      grfAccessMode = (NULL != pEA) ? pEA->grfAccessMode : GRANT_ACCESS;
```

```
	// заполняем выпадающий список "Тип": разрешить или запретить
	hwndCtl = GetDlgItem(hwnd, IDC_ACCESS_MODE);
	EnableWindow(hwndCtl, fEditable);

	int iItem = ComboBox_AddString(hwndCtl, TEXT("Разрешить"));

	if (iItem != -1)
	{
		ComboBox_SetItemData(hwndCtl, iItem, GRANT_ACCESS);
		if (GRANT_ACCESS == grfAccessMode) ComboBox_SetCurSel(hwndCtl, iItem);
	} // if

	iItem = ComboBox_AddString(hwndCtl, TEXT("Запретить"));

	if (iItem != -1)
	{
		ComboBox_SetItemData(hwndCtl, iItem, DENY_ACCESS);
		if (DENY_ACCESS == grfAccessMode) ComboBox_SetCurSel(hwndCtl, iItem);
	}

	hwndCtl = GetDlgItem(hwnd, IDC_EDIT_NAME);
	Edit_SetText(hwndCtl, NULL);

	if (pEA != NULL)
	{
		Edit_SetReadOnly(hwndCtl, TRUE);

		LPTSTR AccountName = NULL; // имя учетной записи

		GetAccountName_W(pEA->Trustee.ptstrName, &AccountName);	// получим имя учетной
записи

		if (AccountName != NULL)
		{
			// копируем имя учетной записи в поле "Имя"
			Edit_SetText(hwndCtl, AccountName);

			LocalFree(AccountName);
		}
	}

	EnableWindow(GetDlgItem(hwnd, IDC_BUTTON_NAME_TEST), fEditable);

	if (NULL != pEA)
	{
		for (int i = 0; i < 13; ++i)
		{
			if (pEA->grfAccessPermissions & dwPermissions[i])
			{
				CheckDlgButton(hwnd, idcPermissions[i], BST_CHECKED);
			}

			EnableWindow(GetDlgItem(hwnd, idcPermissions[i]), fEditable);
		}
	}


	if (FALSE != fEditable) // разрешено редактировать элемент ACE
	{
		// получим атрибуты файла/каталога
		DWORD dwFileAttributes = GetFileAttributes(FileName);

		// определим можно, ли редактировать элемент ACE (только для каталогов)
		fEditable = ((INVALID_FILE_ATTRIBUTES != dwFileAttributes) && (dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY)) ? TRUE : FALSE;
	}
```

```
	// заполняем выпадающий список "Применять"

	DWORD selInheritance = (NULL != pEA) ? (pEA->grfInheritance & (~INHERIT_NO_PROPAGATE)) :
SUB_CONTAINERS_AND_OBJECTS_INHERIT;

	hwndCtl = GetDlgItem(hwnd, IDC_INHERIT);
	EnableWindow(hwndCtl, fEditable);

	for (int i = 0; i < 7; ++i)
	{
		int iItem = ComboBox_AddString(hwndCtl, szInheritText[i]);

		if (iItem != -1)
		{
			ComboBox_SetItemData(hwndCtl, iItem, dwInherit[i]);
			if (selInheritance == dwInherit[i]) ComboBox_SetCurSel(hwndCtl, iItem);
		}
	}

	if ((NULL != pEA) && (pEA->grfInheritance & INHERIT_NO_PROPAGATE))
	{
		CheckDlgButton(hwnd, IDC_CHECK_INHERIT_NO_PROPAGATE, BST_CHECKED);
	}

	EnableWindow(GetDlgItem(hwnd, IDC_CHECK_INHERIT_NO_PROPAGATE), fEditable);


	return TRUE;
}

// ----------------------------------------------------------------------------------------------
void DialogAce_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
	switch (id)
	{
	case IDOK:
	{
		HWND hwndCtl = GetDlgItem(hwnd, IDC_EDIT_NAME);

		if (IsWindowEnabled(hwndCtl) != FALSE) // добавить новый ACE
		{
			PSID pSid = NULL; // указатель на дескриптор безопасности

			int cch = GetWindowTextLength(hwndCtl);

			if (cch > 0)
			{
				LPTSTR lpszName = new TCHAR[cch + 1]; // имя учётной записи

				// копируем имя учётной записи из поля ввода
				GetDlgItemText(hwnd, IDC_EDIT_NAME, lpszName, cch + 1);
				// получим SID учётной записи
				GetAccountSID_W(lpszName, &pSid);

				// освобождаем выделенную память
				delete[] lpszName, lpszName = NULL;
			}

			if (NULL != pSid)
			{
				EXPLICIT_ACCESS ea;

				ea.Trustee.TrusteeForm = TRUSTEE_IS_SID; // доверенный объект задается
с помощью SID
				ea.Trustee.ptstrName = (LPTSTR)pSid; // SID
```

```
                        // формируем разрешения

                        ea.grfAccessPermissions = 0;

                        for (int i = 0; i < 13; ++i)
                        {
                                if (IsDlgButtonChecked(hwnd, idcPermissions[i]) == BST_CHECKED)
                                {
                                        ea.grfAccessPermissions |= dwPermissions[i];
                                }
                        }

                        // определим тип разрешения

                        hwndCtl = GetDlgItem(hwnd, IDC_ACCESS_MODE);
                        ea.grfAccessMode = (ACCESS_MODE)ComboBox_GetItemData(hwndCtl,
ComboBox_GetCurSel(hwndCtl));


                        hwndCtl = GetDlgItem(hwnd, IDC_INHERIT);
                        ea.grfInheritance = (DWORD)ComboBox_GetItemData(hwndCtl,
ComboBox_GetCurSel(hwndCtl));

                        if (( ea.grfInheritance != NO_INHERITANCE) &&
                                (IsDlgButtonChecked(hwnd, IDC_CHECK_INHERIT_NO_PROPAGATE) ==
BST_CHECKED))
                        {
                                ea.grfInheritance |= INHERIT_NO_PROPAGATE;
                        }


                        if (0 != ea.grfAccessPermissions)
                        {
                                // изменяем информацию в дескрипторе безопасности
                                BOOL RetRes = SetFileSecurityInfo(FileName, NULL, 1, &ea,
TRUE);

                                if (RetRes != FALSE)
                                {
                                        EndDialog(hwnd, IDOK);// закрываем диалоговое окно
                                }
                        }
                        LocalFree(pSid);
                }
                else
                {
                        MessageBox(hwnd, TEXT("Не удалось найти учетную запись. Проверьте
правильность введенного имени."), NULL, MB_OK | MB_ICONERROR);
                }
        }
        else
        {
                EndDialog(hwnd, IDOK);      // закрываем диалоговое окно
        }
    }
    break;

    case IDCANCEL:
            EndDialog(hwnd, IDCANCEL);
            break;

    case IDC_BUTTON_NAME_TEST: // Проверить
    {
            int cch = GetWindowTextLength(GetDlgItem(hwnd, IDC_EDIT_NAME));

            if (cch > 0)
```

```
		{
			PSID pSid = NULL;

			LPTSTR AccountName = new TCHAR[cch + 1]; // имя учётной записи

			GetDlgItemText(hwnd, IDC_EDIT_NAME, AccountName, cch + 1);// копируем имя
учётной записи из поля ввода

			GetAccountSID_W(AccountName, &pSid);		// получим SID учётной записи

			delete[] AccountName, AccountName = NULL;	// освобождаем выделенную
память

			if ( pSid != NULL)
			{
				GetAccountName_W(pSid, &AccountName);	// получим имя учётной записи

				SetDlgItemText(hwnd, IDC_EDIT_NAME, AccountName);	// копируем имя
учётной записи в поле ввода

				if (AccountName != NULL)
					LocalFree(AccountName);// освобождаем выделенную память

				LocalFree(pSid);

				MessageBox(hwnd, TEXT("Учетная запись есть."), NULL, MB_OK |
MB_ICONERROR);
			}
			else
			{
				MessageBox(hwnd, TEXT("Не удалось найти учетную запись. Проверьте
правильность введенного имени."), NULL, MB_OK | MB_ICONERROR);

				// удаляем текст из поля "Имя"
				SetDlgItemText(hwnd, IDC_EDIT_NAME, NULL);
			}
		}
	}
	break;
	}
}
```

4. В соответствии с заданием в п.4 изменила разработанную ранее программу для копирования файлов/каталогов.

Листинг 4. Приложения для копирования файлов с запросом пароля

```
HANDLE LogonUserToLocalComputer();
HANDLE OpenUserToken(LPCTSTR lpUserName, LPCTSTR lpDomain, LPCTSTR lpPassword,
	DWORD LogonType, DWORD DesireAcces, PSECURITY_ATTRIBUTES PSECUR_ATTRIB,
	TOKEN_TYPE TOKEN_TYP, SECURITY_IMPERSONATION_LEVEL IMPERSONATION_LEVEL);


TCHAR szUserName[UNLEN + 1];
TCHAR szPassword[51];

HANDLE hToken;

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, SW_HIDE); // скрываем 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 ChildDlgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM)
```

```
{
	switch (uMsg) {
	case WM_COMMAND:
		switch (LOWORD(wParam)) {
		case IDOK:
		{
			/*TCHAR szUserName[UNLEN + 1];
			TCHAR szPassword[51];*/
			GetDlgItemText(hWnd, IDC_USER, szUserName, _countof(szUserName));//это имя и
			GetDlgItemText(hWnd, IDC_PASSWORD, szPassword, _countof(szUserName));//это
пароль
			hToken = LogonUserToLocalComputer();
			EndDialog(hWnd, IDOK);
			return TRUE;
		}break;
		case IDCANCEL:
			EndDialog(hWnd, 0);
			break;
		}
		break;
	case WM_CLOSE:
		EndDialog(hWnd, 0);
		break;
	}
	//return (int)uMsg.wParam;
return FALSE;
}

HANDLE LogonUserToLocalComputer()
{
	for (int j = 0; j < 3; ++j)
	{
		// получение маркера доступа пользователя
		HANDLE hToken = OpenUserToken(szUserName, TEXT("."), szPassword,
			LOGON32_LOGON_INTERACTIVE,
			TOKEN_QUERY | TOKEN_IMPERSONATE, //для получения информации о содержимом
маркера доступа | разрешение замещать маркер доступа процесса
			NULL, TokenImpersonation, SecurityImpersonation);

		if (NULL != hToken)
		{
			return hToken;
		} // if
	} // for

	return NULL;
}

HANDLE OpenUserToken(LPCTSTR lpUserName, LPCTSTR lpDomain, LPCTSTR lpPassword, DWORD LogonType,
DWORD DesireAcces, PSECURITY_ATTRIBUTES PSECUR_ATTRIB, TOKEN_TYPE TOKEN_TYP,
SECURITY_IMPERSONATION_LEVEL IMPERSONATION_LEVEL)
{
	HANDLE TOKEN = NULL;
	BOOL BRET = LogonUser(lpUserName, lpDomain, lpPassword, LogonType,
LOGON32_PROVIDER_DEFAULT, &TOKEN);//получение маркера доступа указанного пользователя
	if (FALSE != BRET)
	{
		HANDLE newTOKEN = NULL;
		BRET = DuplicateTokenEx(TOKEN, DesireAcces, PSECUR_ATTRIB, IMPERSONATION_LEVEL,
TOKEN_TYP, &newTOKEN);//duble marker of acces
		CloseHandle(TOKEN);//КАК ОН БУДЕТ РАБОТАТЬ ЕСЛИ МЫ ЕГО ЗАКРЫЛИ???
		TOKEN = (FALSE != BRET) ? newTOKEN : NULL;
	}
	return TOKEN;
}
```

```
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
			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;
} // DialogProc
/ ----------------------------------------------------------------------------------------------
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 CopyDirectoryContent(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 расширением

      hFind = FindFirstFile(szFind, &ffd);
      if (hFind == INVALID_HANDLE_VALUE)
      {
            return FALSE;
      }
      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 & FILE_ATTRIBUTE_DIRECTORY)
            {
                  if (lstrcmp(ffd.cFileName, L".") == 0 || lstrcmp(ffd.cFileName, L"..") == 0)
continue;
                  CreateDirectory(szOutFileName, NULL);//копирование со всеми атрибутами??
                  CopyDirectoryContent(szInFileName, szOutFileName);
            }
            else
            {
                  CopyFile(szInFileName, szOutFileName, TRUE);
            }


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

      FindClose(hFind);
      return TRUE;
```

```
}

BOOL CopyDirectoryContent_Dir(LPCTSTR szInDirName, LPCTSTR szOutDirName)
{
	if (!DirectoryExists(szOutDirName))
	{
		BOOL RetRes = CreateDirectory(szOutDirName, NULL);
		if ((GetLastError() != ERROR_ACCESS_DENIED))
		{
			return RetRes = CopyDirectoryContent(szInDirName, szOutDirName);
		}
		else
		{
			return RetRes;
		}
	}
	else
	{
		MessageBox(hwnd, L"Папка уже существует", L" !", MB_OK);
		return FALSE;
	}
}

BOOL DirectoryExists(LPCTSTR szPath)
{
	DWORD dwAttrib = GetFileAttributes(szPath);

	return (dwAttrib != INVALID_FILE_ATTRIBUTES &&
		(dwAttrib & FILE_ATTRIBUTE_DIRECTORY));
}
void Dialog_OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify)
{
	switch (id)
	{

	case IDOK:
	{
		TCHAR FromName[260];
		TCHAR ToName[260];
		//TCHAR NewName[MAX_PATH + 1];

		GetDlgItemText(hwnd, IDC_EDIT_FROM, FromName, _countof(FromName));
		GetDlgItemText(hwnd, IDC_EDIT_TO, ToName, _countof(ToName));//каталог куда копируем

		/*Выясним, что копируется, файл или папка.
		Если папка, то сформируем с ней маршрут и продолжим поиск*/
		WIN32_FIND_DATA ffd;
		HANDLE hFind;
		BOOL BRET = FALSE;
		LPCTSTR FILE = PathFindFileNameW(FromName);


		DWORD FromAttributes = GetFileAttributes(FromName);//определим, что копируем
		DWORD ToNameAttributes = GetFileAttributes(ToName);//выясним, выясним куда копируем

		if (INVALID_FILE_ATTRIBUTES == ToNameAttributes || INVALID_FILE_ATTRIBUTES ==
FromAttributes) // (!) если нет файла или каталога
		{
			break;
		}
		else
			if ((ToNameAttributes & FILE_ATTRIBUTE_DIRECTORY) == 0 && (FromAttributes&
FILE_ATTRIBUTE_DIRECTORY) != 0)//не каталог
			{
				SetLastError(ERROR_PATH_NOT_FOUND);
				MessageBox(hwnd, L"Нельзя скопировать папку в файл!", L" !", MB_OK);
			}
```

```
            if ((FromAttributes& FILE_ATTRIBUTE_DIRECTORY) != 0)//является каталогом
            {
                        // выполняем операцию с файлом/каталогом
                        lstrcat(ToName, L"\\");
                        lstrcat(ToName, FILE);
                        //CreateDirectory(ToName, NULL);
                        BRET = CopyDirectoryContent_Dir(FromName, ToName);
                        // получим код последней ошибки

                        /*Блок, где будет вызван ввод пароля в случае отказа в доступе*/
                        DWORD dwError = (FALSE == BRET) ? GetLastError() : ERROR_SUCCESS;
                        // завершаем олицитворение
                        RevertToSelf();
                        if (ERROR_ACCESS_DENIED == dwError) // (!) ошибка: отказано в доступе
                        {
                              MessageBox(hwnd, L"Отказано в доступе", L" !", MB_OK);
                              // получаем маркер доступа пользователя
                              DialogBox(GetWindowInstance(hwnd),
MAKEINTRESOURCE(IDD_PASSWORD), hwnd, ChildDlgProc);//окно для запроса пароля и логина
                              //HANDLE hToken = LogonUserToLocalComputer();
                              if (NULL != hToken)//глобальный
                              {

                                    ImpersonateLoggedOnUser(hToken);// начинаем олицитворение

                                    CloseHandle(hToken);// закрываем маркер доступа

                                    BRET = CopyDirectoryContent_Dir(FromName, ToName);
                                    TCHAR Message[MAX_PATH];
                                    if (BRET == 0)
                                    {
                                          lstrcpy(Message, L"Файлы не скопированы в папку:
");

                                          lstrcat(Message, ToName);
                                          MessageBox(hwnd, Message, L"Ошибка\0", MB_OK);
                                          SetDlgItemText(hwnd, IDC_EDIT_FROM, L" ");
                                          SetDlgItemText(hwnd, IDC_EDIT_TO, L" ");
                                    }
                                    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" ");
                                    }

                              }
                              else
                              {
                                    break; // (!) выходим из цикла
                              }
                        }
                        else
                        {
                              SetLastError(dwError);
                              break; // (!) выходим из цикла
                        }
                        /*-------------------------------------------------------*/
            }
            else
            {
                  lstrcat(ToName, L"\\");
                  LPCTSTR FILE = PathFindFileNameW(FromName);
                  lstrcat(ToName, FILE);
```

```
                    BRET = CopyFile(FromName, ToName, TRUE);
                    /*Блок, где будет вызван ввод пароля в случае отказа в доступе*/
                    DWORD dwError = (FALSE == BRET) ? GetLastError() : ERROR_SUCCESS;
                    // завершаем олицитворение
                    RevertToSelf();
                    if (ERROR_ACCESS_DENIED == dwError) // (!) ошибка: отказано в доступе
                    {
                           MessageBox(hwnd, L"Отказано в доступе", L" !", MB_OK);
                           // получаем маркер доступа пользователя

                           DialogBox(GetWindowInstance(hwnd), MAKEINTRESOURCE(IDD_PASSWORD),
hwnd, ChildDlgProc);//окно для запроса пароля и логина

                           if (NULL != hToken)//глобальный
                           {
                                  // начинаем олицитворение
                                  ImpersonateLoggedOnUser(hToken);
                                  // закрываем маркер доступа
                                  CloseHandle(hToken);
                                  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\0", MB_OK);
                                                SetDlgItemText(hwnd, IDC_EDIT_FROM, L" ");
                                                SetDlgItemText(hwnd, IDC_EDIT_TO, L" ");

                                         }
                                         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" ");
                                         }
                           }
                           else
                           {
                                  break; // (!) выходим из цикла
                           }
                    }
                    else
                    {
                           SetLastError(dwError);
                           break; // (!) выходим из цикла
                    }
                    /*-------------------------------------------------------*/
             }



      }
      break;

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

```
        }
}
```

5. Тестирование:

Первую программу необходимо запустить от имени учетной записи «Администратор».

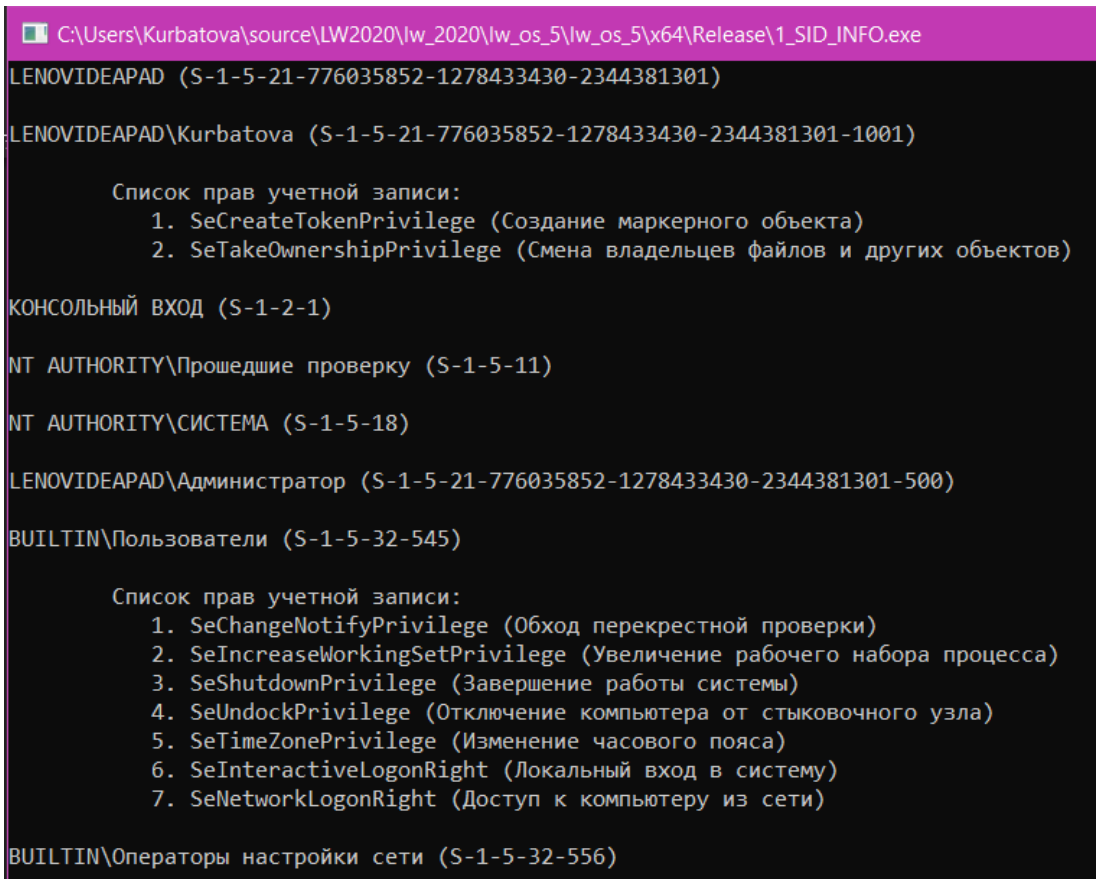

Рис. 5.2. Программа 1. Информация по SID

Вторая программа должна быть запущена от имени учетной записи «Администратор», как и программа «Process Explore», которая используется для проверки точности отображаемых разработанной программой сведений.

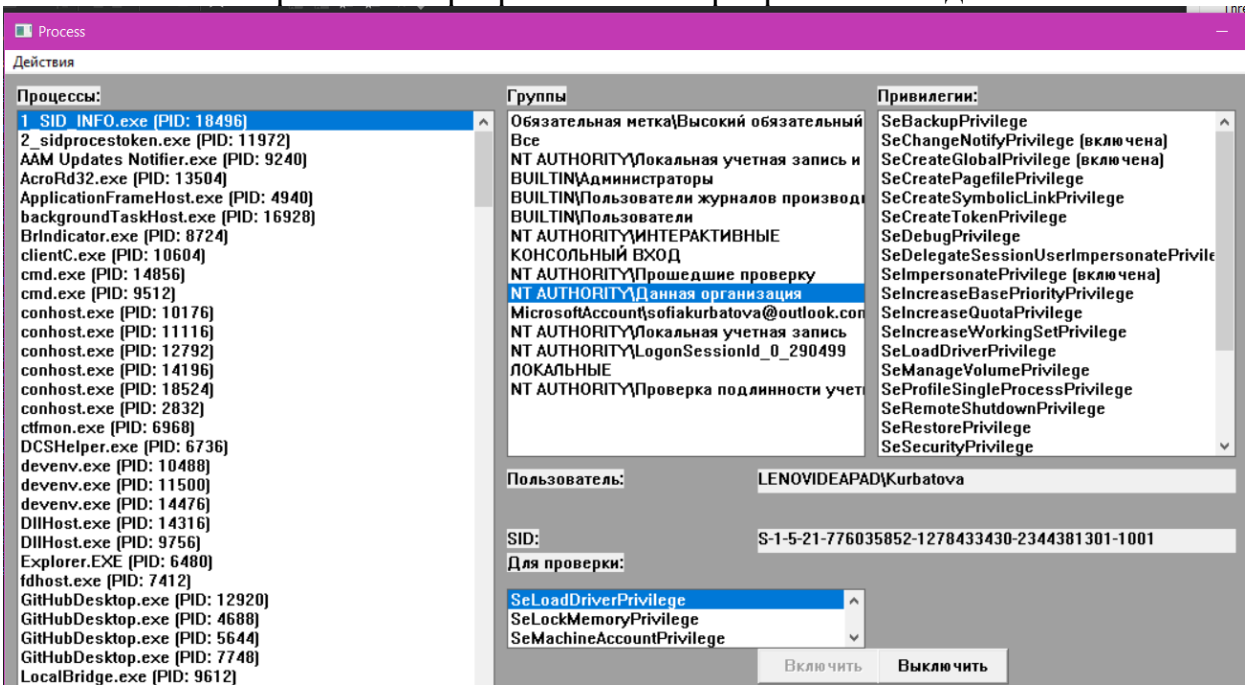

Рис. 5.3. Программа 2. Отображение групп и привилегий для программы 1

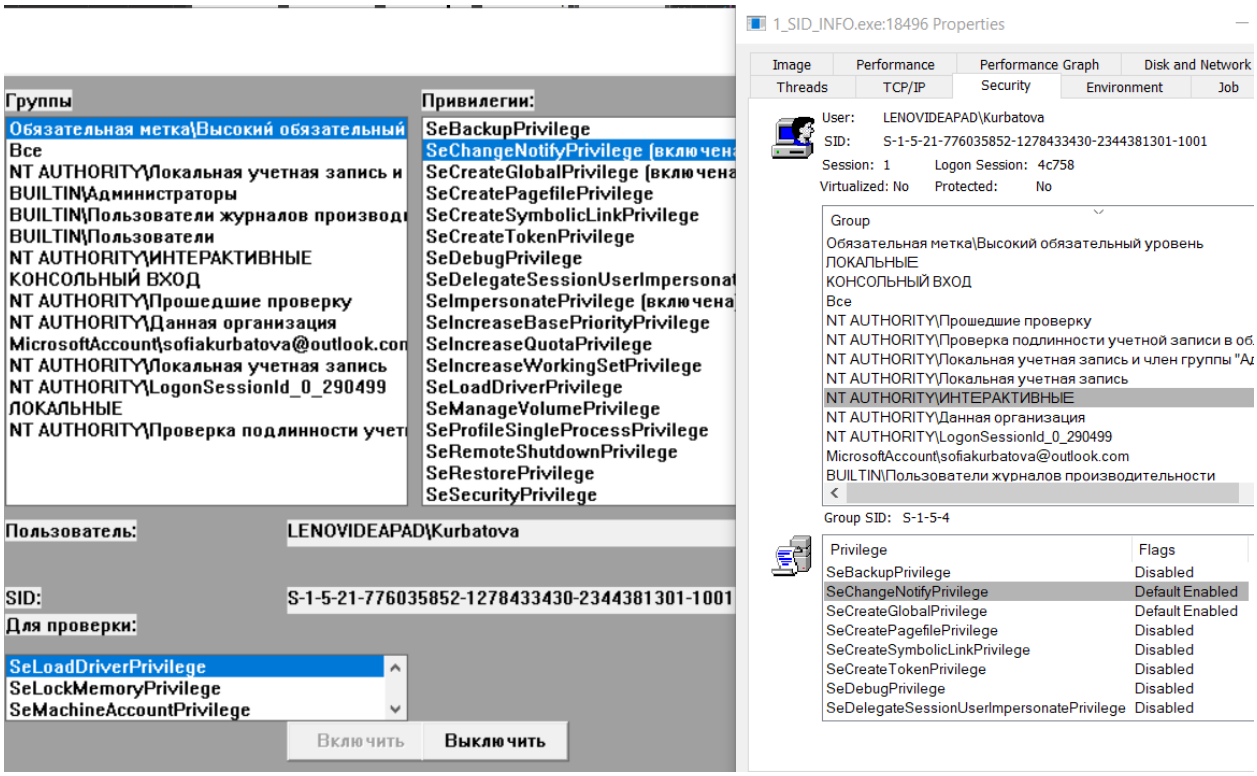

Рис. 5.4. Проверка корректности вывода данных

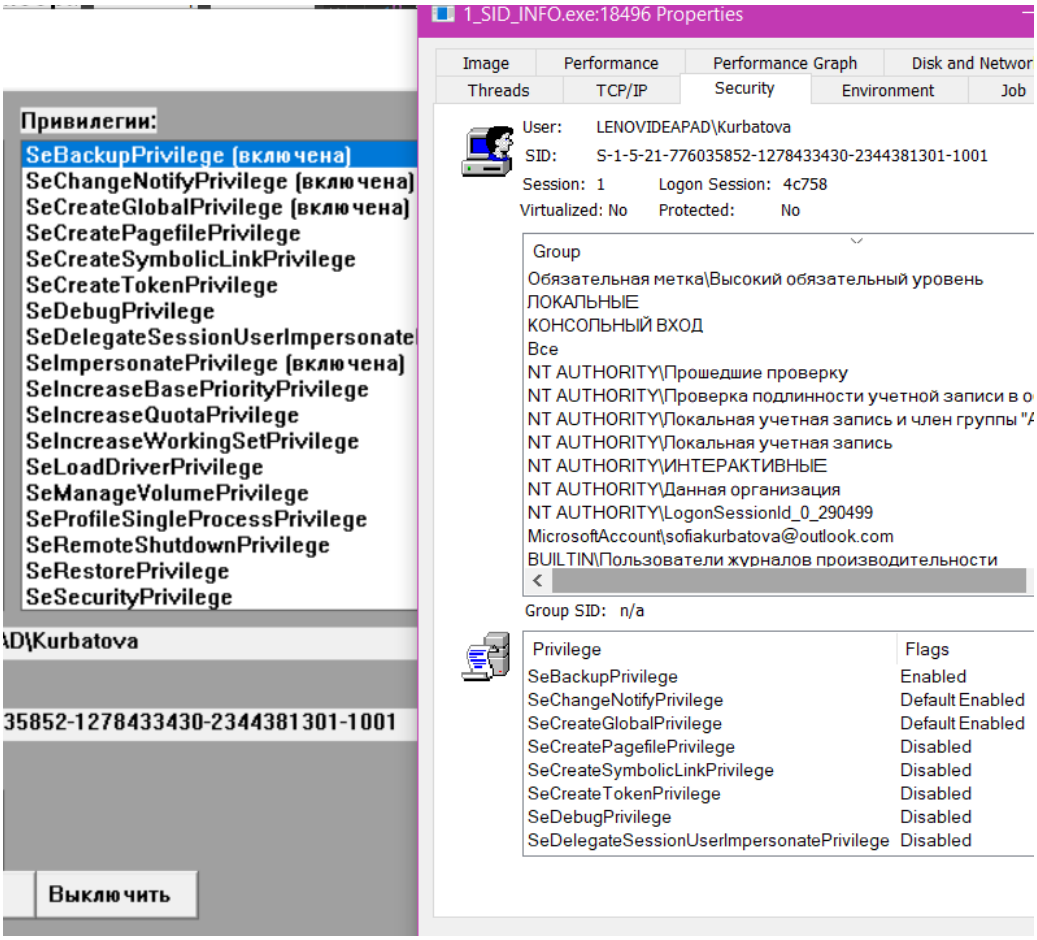

Рис. 5.5. Включена привилегия SeBackupPrivilege

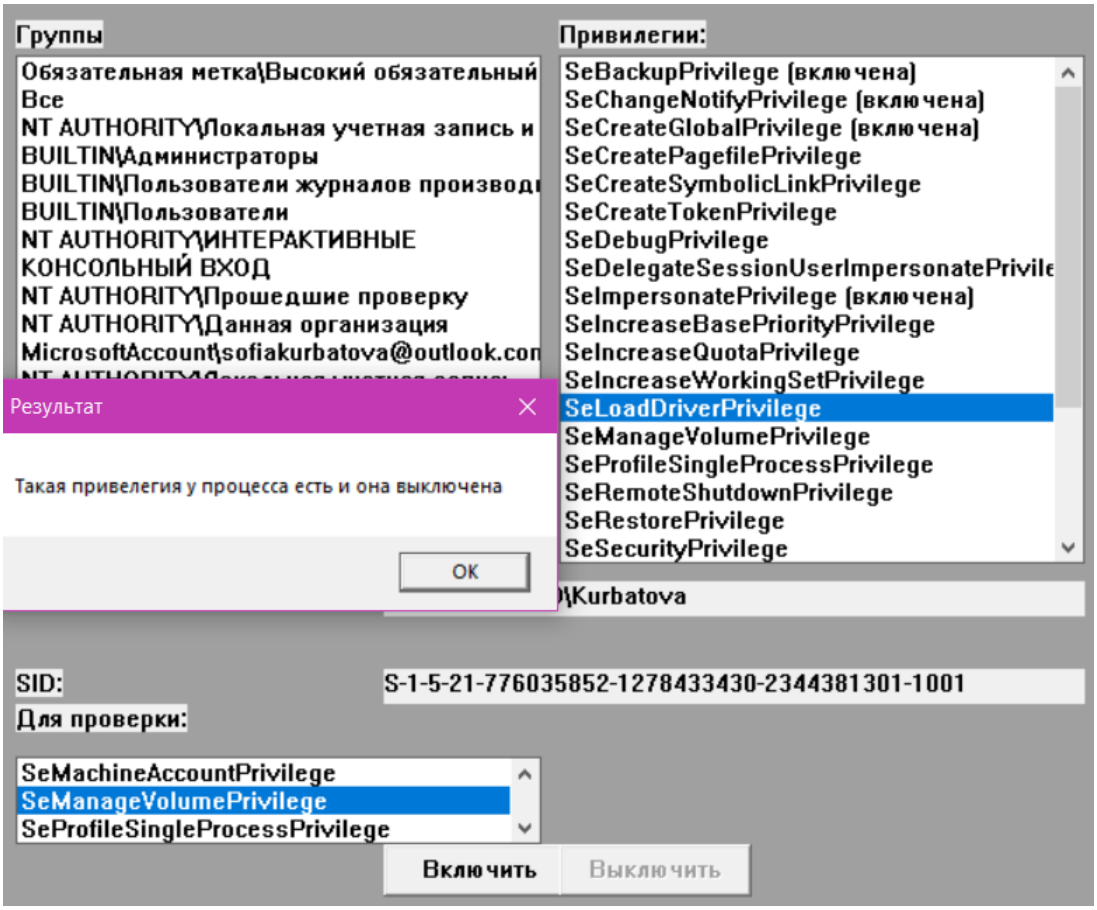

Рис. 5.6. Проверка наличия привилегии по варианту

Третья программа:

Предварительно настроила каталог, ограничив количество субъектов обладающих доступом к выбранному каталогу до 2.

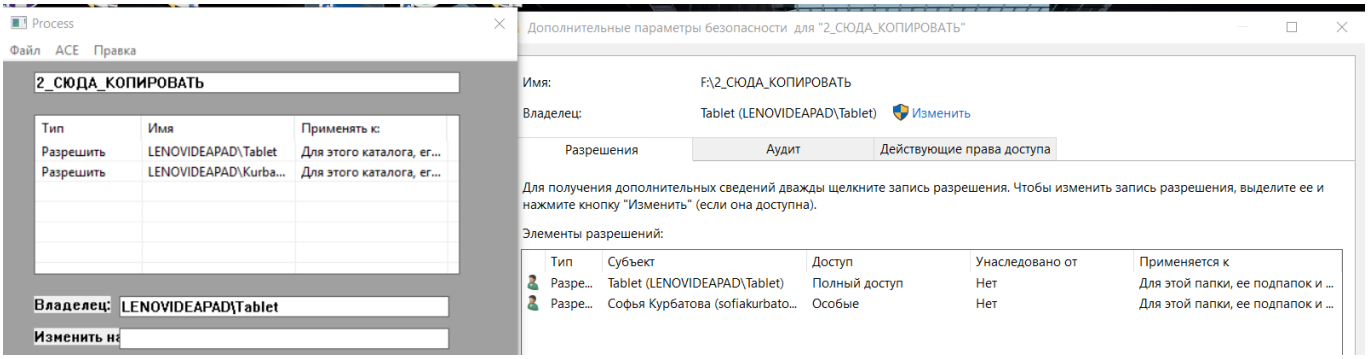

Рис. 5.7. Программа 3. Просмотр DACL

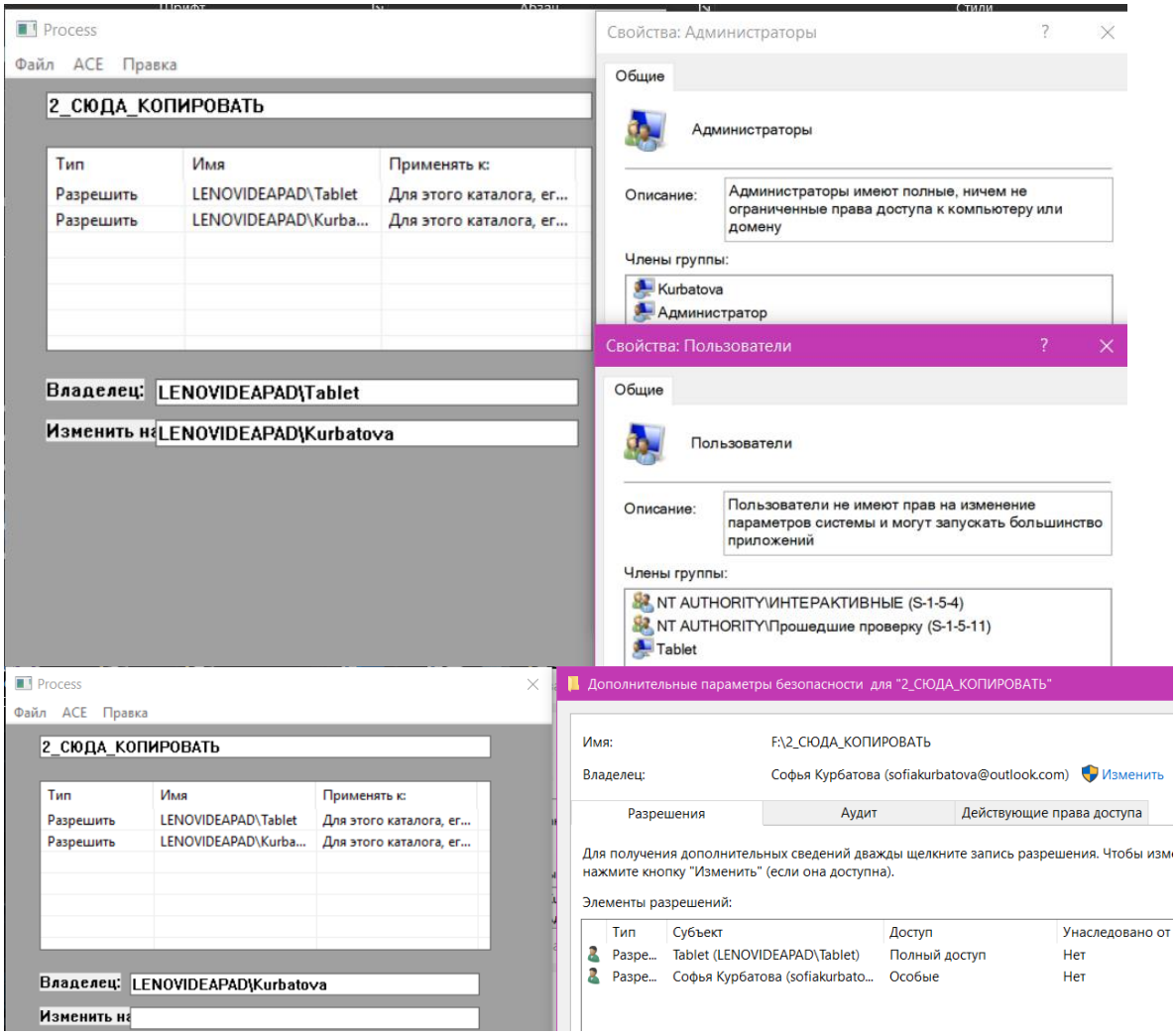

Рис. 5.8. Изменение владельца папки

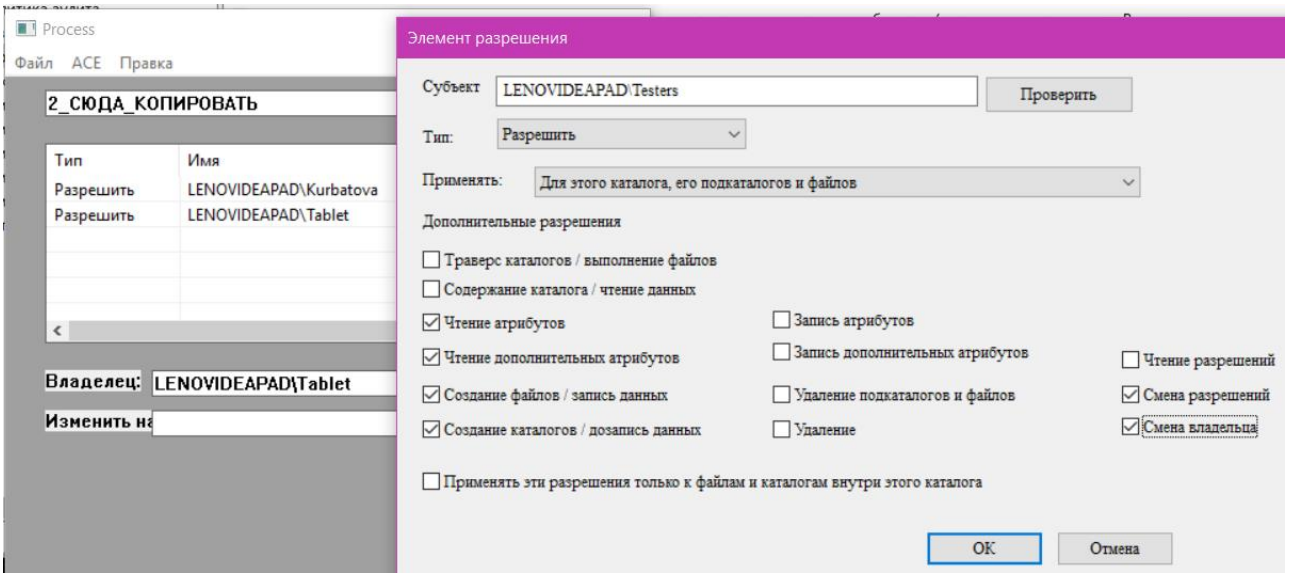

Рис. 5.9. Добавление разрешений

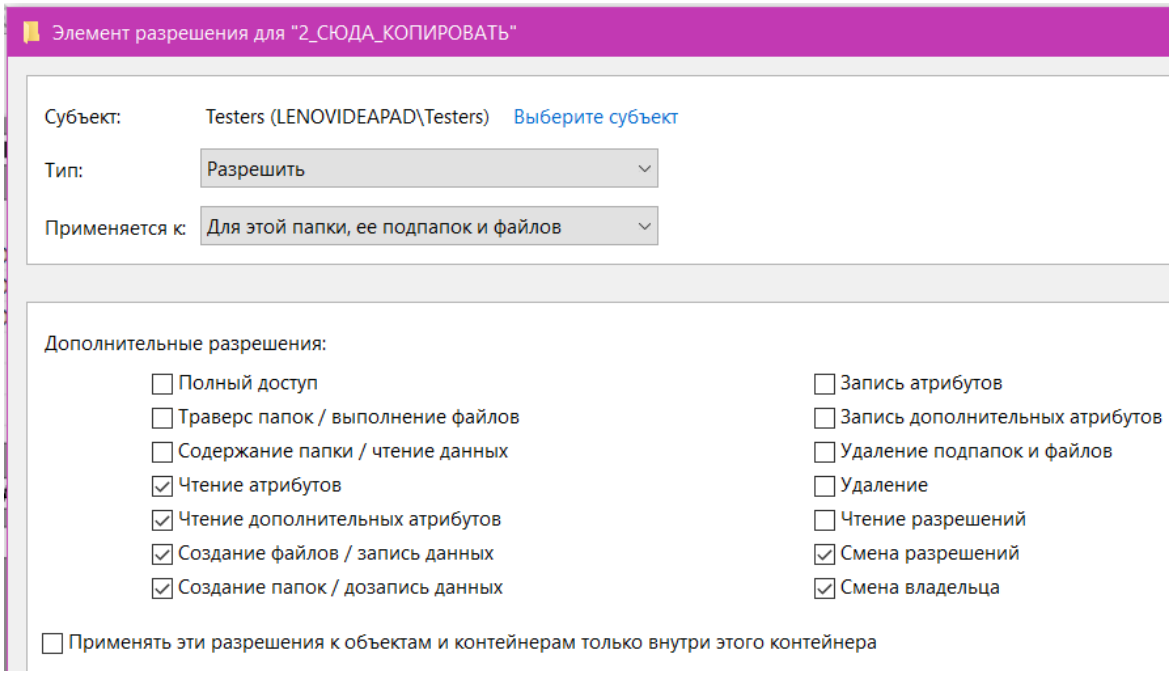

Рис. 5.10. Проверка установленных разрешений

Для двух папок были установлены разграничивающие доступ разрешения. Так пользователю, который осуществляет копирование из папки 1_Копирование в 2_Копировать сюда запрещено данное действие.

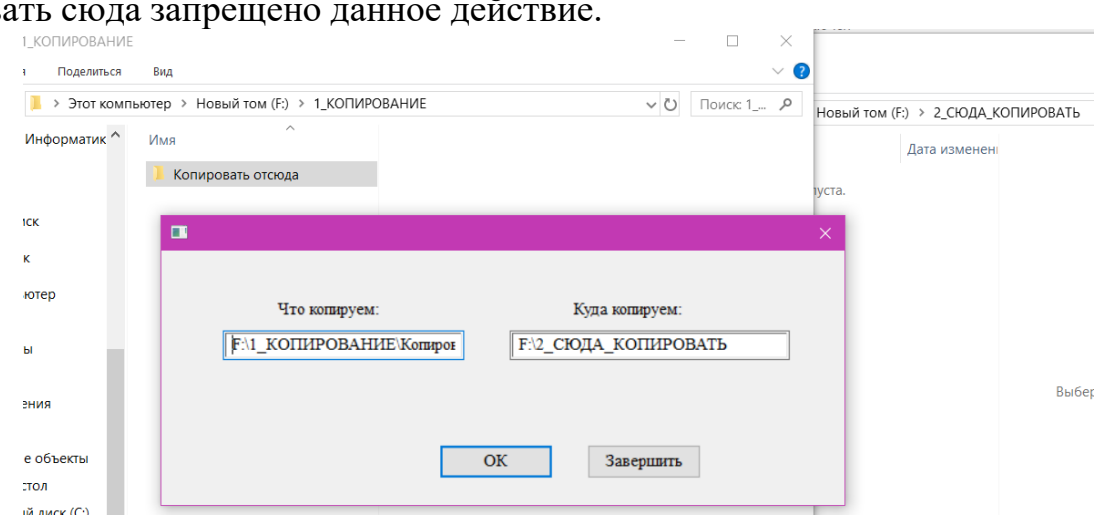

Рис. 5.11. Программа 4. Начало выполнения копирования каталога

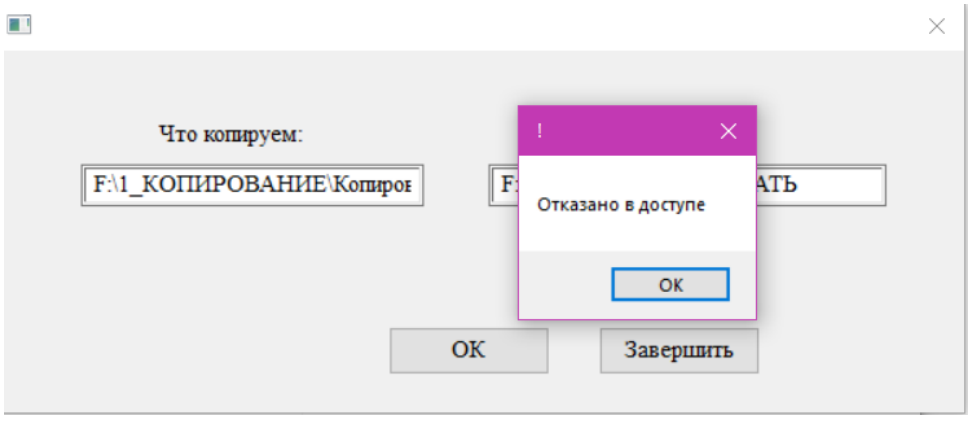

Рис. 5.12. Информирование об отказе

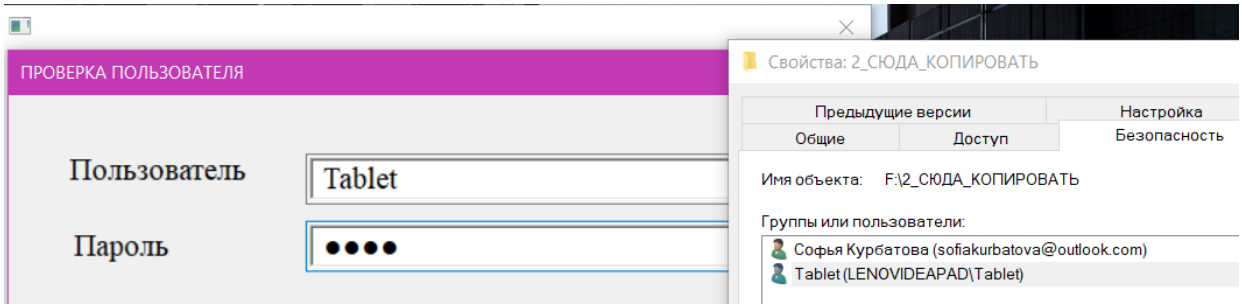

Рис. 5.13. Ввод данных пользователя с нужным разрешением

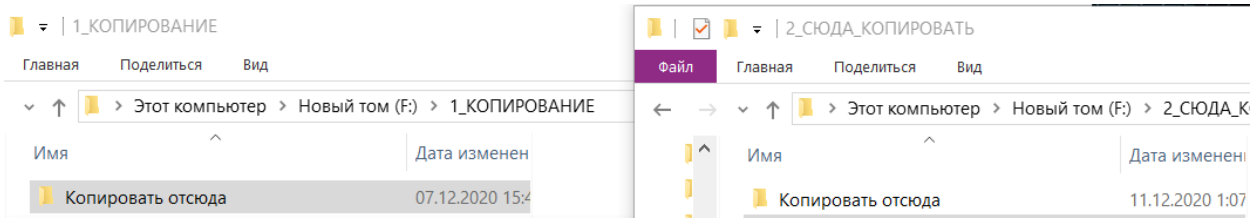

Рис. 5.14. Результат копирования

Вывод: Таким образом, в ходе выполнения лабораторной работы было разработано 4 приложения, которые позволили изучить модель безопасности Windows. Так были получены практические навыки определения прав и привилегий пользователей, осуществлено разграничение доступа к одному и тому же каталогу для разных пользователей. Из-за этого, для того, чтобы выполнить заданную операцию копирования файлов, было необходимо ввести данные для подтверждения пользователя.