Содержание

[ВВЕДЕНИЕ 5](#_Toc28901419)

[1 ОБЗОР СОСТОЯНИЕ ВОПРОСА 7](#_Toc28901420)

[1.1 Общее понятие о средствах защиты информации 7](#_Toc28901421)

[1.2 Структура реестра 7](#_Toc28901422)

[1.3 Привязка программного обеспечения к уникальным значениям реестра 9](#_Toc28901423)

[2 ЦЕЛЬ И ЗАДАЧИ ПРОЕКТА 10](#_Toc28901424)

[3 ОПИСАНИЕ ПРОГРАММЫ 11](#_Toc28901425)

[3.1 Общие сведения 11](#_Toc28901426)

[3.2 Функциональное назначение 12](#_Toc28901427)

[3.3 Описание логической структуры 12](#_Toc28901428)

[3.3.1 Проверка принадлежности приложения к устройству 12](#_Toc28901429)

[3.3.2 Графическая часть приложения 14](#_Toc28901430)

[3.4 Используемые технические средства 18](#_Toc28901431)

[3.5 Вызов и загрузка 18](#_Toc28901432)

[3.6 Входные данные 18](#_Toc28901433)

[3.7 Выходные данные 18](#_Toc28901434)

[ВЫВОДЫ 19](#_Toc28901435)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc28901436)

[ПРИЛОЖЕНИЕ А 21](#_Toc28901437)

[ПРИЛОЖЕНИЕ Б 27](#_Toc28901438)

# **ВВЕДЕНИЕ**

Каждый человек, выполняя свою работу, рассчитывает на получение какой-либо компенсации за свой труд. И разработчики программного обеспечения не исключение. Те разработчики, которые работают на компании, усилия которых направлены на выпуск собственной продукции, сильно зависят от рынка, а если быть точнее – от действий конечного пользователя. Использование взломанного программного обеспечения в наше время не редкость, следовательно, для предотвращения кражи продуктов, выставленных на продажу на цифровых площадках, нужно добавлять защиту от несанкционированного доступа к приложению.

Ярким примером является игровая индустрия. Приложения, купленные на цифровых площадках, зачастую являются привязанными к этим площадкам, которые в свою очередь являются посредниками между производителем и конечным пользователем. Не имеет значения, продаётся игра, или, например, в качестве рекламной акции производитель решил раздать n-ное количество копий продукта, у каждого цифрового экземпляра приложения есть свой уникальный ключ, с помощью которого продукта идентифицируется может проходить дальнейшую валидацию.

Алгоритм работы данного механизма приблизительно следующий: после покупки игры прользователю приходит случайный постоянный ключ, сгенерированный сервером цифровой площадки; получив ключ, клиент отправляет его на сервер для подтверждения прав на владение продуктом. Если значения ключей, которые сервер отправил клиенту, и, впоследствии, получил от него, совпадают, то клиенту предоставляется цифровая копия продукта, которую он уже может использовать, окончательно привязывая ключ к определённому клиенту.

Слабостью такого метода является часть с созданием сервером постоянного ключа. При генерации значения сервер зачастую следует определённому паттерну, из-за чего злоумышленники получают возможность написать собственный генератор ключей, проанализировав некоторое количество созданных сервером[1].

Однако, несмотря на такую брешь в безопасности, валидация с использованием ключей является одним из наиболее популярных вариантов лицензированного распространения программного обеспечения. Тем не менее, существуют и другие способы ограничения возможностей нелегального распространения программного обеспечения.

Задачей данного курсового проекта является написать приложение, имеющее защиту от несанкционированного доступа.

Записка по курсовому содержит более подробную информацию о различных аспектах программы и безопасности в целом. Она состоит из следующих разделов:

* обзор состояния вопроса: содержит общую информацию о способах защиты приложения от несанкционированного доступа;
* цель и задачи проекта: содержит цель, задачи и требования к проекту;
* описание программы: содержит детализированные сведения об алгоритме программы, его реализации, информацию о входных и выходных данных;
* выводы: состоит из подведения итогов курсового проекта;
* список использованной литературы: содержит список источников с информацией, использованной для написания курсового проекта;
* приложение А: содержит полный исходный код приложения;
* приложение Б: содержит формы оформления блок-схем.

# **ОБЗОР СОСТОЯНИЕ ВОПРОСА**

## Общее понятие о средствах защиты информации

Средства защиты информации– это совокупность инженерно-технических, электрических, электронных, оптических и других устройств и приспособлений, приборов и технических систем, а также иных элементов, используемых для решения различных задач по защите информации, в том числе предупреждения утечки и обеспечения безопасности защищаемой информации.

В целом средства обеспечения защиты информации в части предотвращения преднамеренных действий в зависимости от способа реализации можно разделить на группы: технические (аппаратные), программные, смешанные аппаратно - программные, организационные.

Технические (аппаратные) средства- это различные по типу устройства (механические, электромеханические, электронные и другие), которые аппаратными средствами решают задачи защиты информации. Они препятствуют физическому проникновению, либо, если проникновение все же состоялось, препятствуют доступу к информации, в том числе с помощью ее маскировки. Первую часть задачи решают замки, решетки на окнах, защитная сигнализация и др. Вторую - генераторы шума, сетевые фильтры, сканирующие радиоприемники и множество других устройств, «перекрывающих» потенциальные каналы утечки информации или позволяющих их обнаружить. Преимущества технических средств связаны с их надежностью, независимостью от субъективных факторов, высокой устойчивостью к модификации.

К аппаратным средствам защиты относятся различные электронные, электронно-механические, электронно-оптические устройства. К настоящему времени разработано значительное число аппаратных средств различного назначения, однако наибольшее распространение получили следующие:

* специальные реестры для хранения реквизитов защиты: пароли, идентифицирующие коды, грифы или уровни секретности;
* устройства измерения индивидуальных характеристик человека (голоса, отпечатков) с целью его идентификации;
* схемы прерывания передачи информации в линии связи с целью периодической проверки адреса выдачи данных.
* устройства для шифрования информации (криптографические методы)[2].

В данном приложении будет реализована аппаратная защита с использованием уникальных значений, получаемых из реестра.

## Структура реестра

Реестр Windows, или системный реестр — иерархически построенная база данных параметров и настроек в большинстве операционных систем семейства Microsoft Windows.

Как показано на рисунке 1.2.1, реестр имеет иерархическую структуру, которая напоминает файловую систему жесткого диска – с его каталогами, подкаталогами и файлами. Но называются элементы реестра по-другому: верхний уровень иерархии составляют разделы, каждый из которых может содержать вложенные подразделы, а также параметры. Именно в параметрах хранится основное содержимое реестра, разделы служат лишь для группировки схожих по назначению параметров.

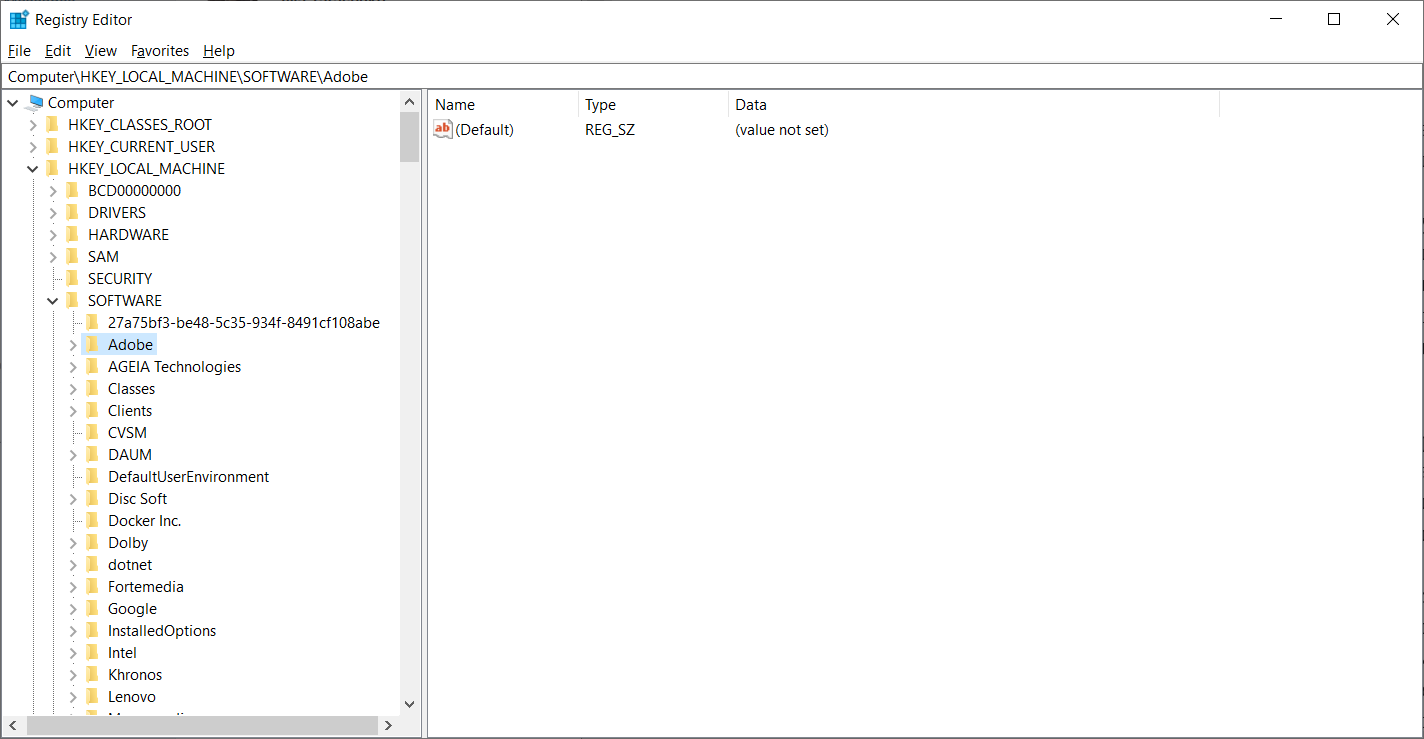


Рисунок 1.2.1 – реестр Windows 10

Далее приведен краткий перечень и краткое описание стандартных разделов реестра. Максимальная длина имени раздела составляет 255 символов.

* **HKEY\_CURRENT\_USER –** данный раздел является корневым для данных конфигурации пользователя, вошедшего в систему в настоящий момент. Здесь хранятся папки пользователя, цвета экрана и параметры панели управления. Эти сведения сопоставлены с профилем пользователя. Вместо полного имени раздела иногда используется аббревиатура HKCU;
* **HKEY\_USERS –** данный раздел содержит все активные загруженные профили пользователей компьютера. Раздел HKEY\_CURRENT\_USER является подразделом раздела HKEY\_USERS. Вместо полного имени раздела иногда используется аббревиатура HKU;
* **HKEY\_LOCAL\_MACHINE –** раздел содержит параметры конфигурации, относящиеся к данному компьютеру (для всех пользователей). Наиболее интересным является подраздел Software, который включает в себя настройки всех установленных в системе приложений. Вместо полного имени раздела иногда используется аббревиатура HKLM;
* **HKEY\_CLASSES\_ROOT –** хранящиеся здесь сведения обеспечивают выполнение необходимой программы при открытии файла с использованием проводника. Вместо полного имени раздела иногда используется аббревиатура HKCR. Начиная с Windows 2000, эти сведения хранятся как в HKEY\_LOCAL\_MACHINE, так и в HKEY\_CURRENT\_USER;
* **HKEY\_CURRENT\_CONFIG** – данный раздел содержит сведения о профиле оборудования, используемом локальным компьютером при запуске системы[3].

Разделы в реестре называются ключами. Значения ключей определяют поведеение компьютера в тех или иных ситуациях. Данные могут храниться с такими типами, как REG\_DWORD, REG\_SZ, REG\_BINARY и т.п.[4].

На рисунке 1.2.2 показан пример значений ключа, где 1 – столбец с именами значения, 2 – столбец типов данных, а 3 – данные, записанные в значении.

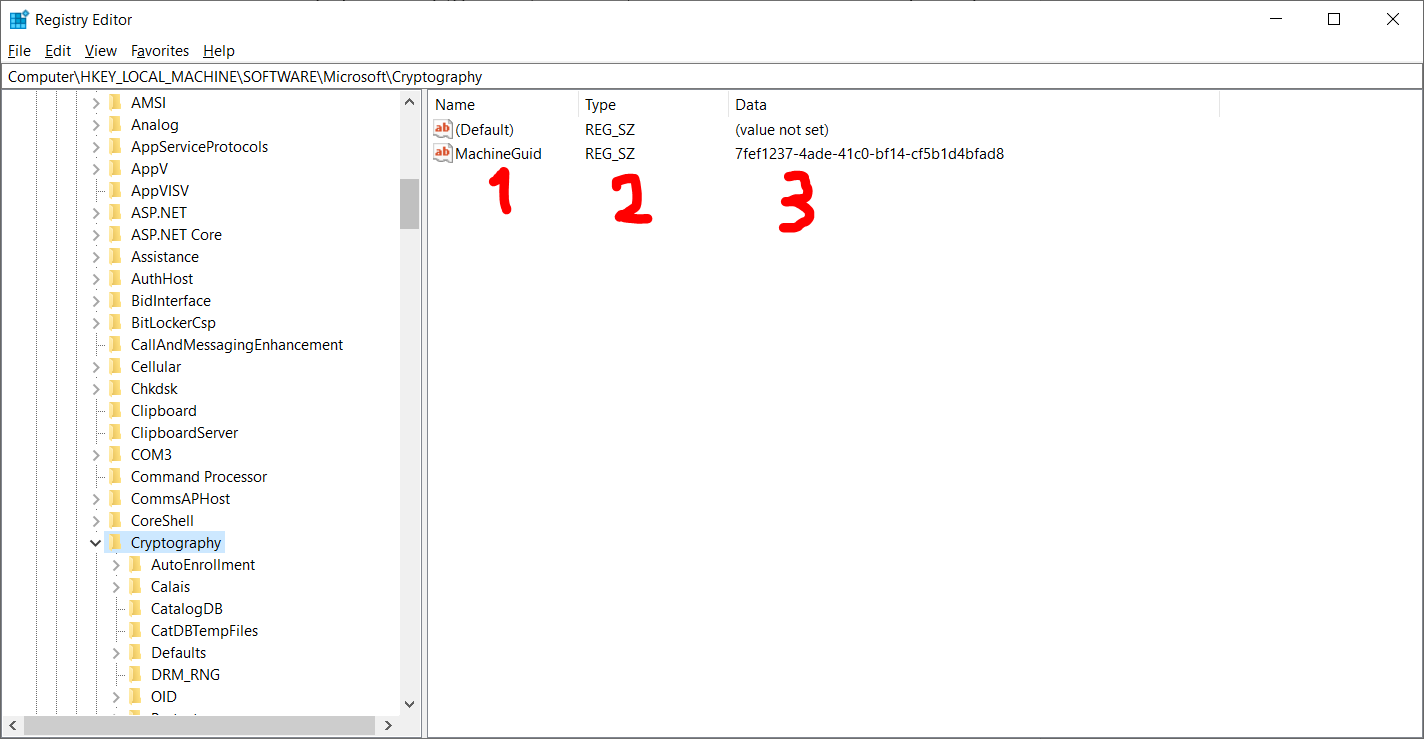


Рисунок 1.2.2 – значения ключей реестра Windows

## Привязка программного обеспечения к уникальным значениям реестра

В реестре содержится масса уникальных значений, которые можно найти только на данном устройстве. Это такие данные, как, например, номер сетевой карты, или код операционной системы. Следовательно, выставив защиту таким образом, что программа будет получать одно (или несколько) значение, которое будет уникально идентифицировать устройство, не позволяя запускать приложенте при других значениях.

Однако недостатком такого подхода является факт, что пользователи, которые регулярно обновляют состовляющие своего ПК (как аппаратные, так и программные), могут столкнуться с ситуацией, в которой они не смогут запустить приложение по причине отсутствия в комплектации компьютера устройства, на которое привязывалась программа.

Ввиду вышеперечисленных причин такой метод защиты может подойти для организаций или учреждений, в который такие обновления не проводятся часто. Под такой типаж попадают как офисные компании, так и учебные заведения.

# **ЦЕЛЬ И ЗАДАЧИ ПРОЕКТА**

Целью данной работы является разработка приложения с защитой от несанкционированного доступа к исполняемому файлу.

Задача проекта – разработать Win 32/64 API программу на языке C++, защищённую с использованием динамически подключаемой библиотеки.

Целевые действия программы:

* проверка на наличие подключаемого \*.dll файла;
* в случае успеха, подключаемая библиотека проводит проверку по уникальному значению операционной системы, взятому из реестра. В случае успеха запускается программный интерфейс.

Дополнительно необходимо просмотреть следующие свойства:

* организовать интуитивно-понятный пользовательский интерфейс для приложения;
* в случае возникновения ошибок при работе программы, уведомлять пользователя об ошибках.

# **ОПИСАНИЕ ПРОГРАММЫ**

## Общие сведения

Программа разработана для использования в пределах семейства операционных систем Windows. В процессе написания программы использовался язык программирования C++, работа проходила в среде разработки Microsoft Visual Studio 2017.

При успешном запуске приложения перед пользовтателем появляется окно с рисунка 3.1.1, из которого он может получить краткую информацию о приложении, нажав кнопку About. Результат виден на рисунке 3.1.2. Для получения помощи и описания действий, которые нужно выполнить для успешного выполнения основного цикла программы, нужно кликнуть на More→Help, после чего появится окно с рисунка 3.1.3. Чтобы запустить меню взаимодействия с атрибутами доступа надо нажать на кнопку Run.

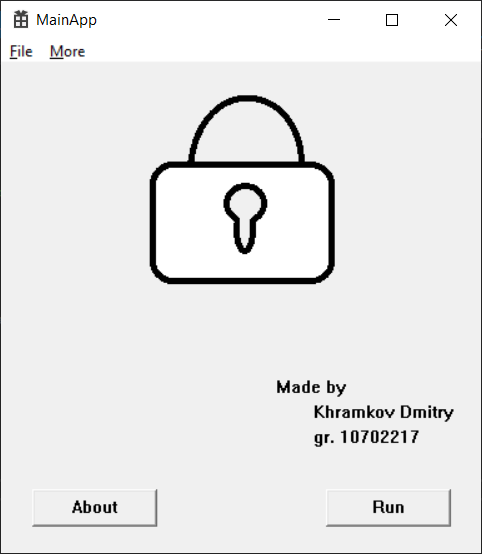


Рисунок 3.1.1 – Главное меню программы

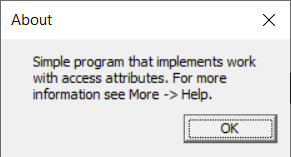


Рисунок 3.1.2 – Меню краткого описания

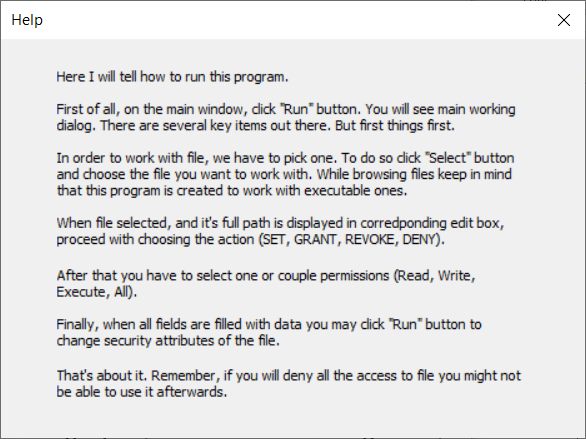


Рисунок 3.1.3 – Меню помощи

## Функциональное назначение

Приложение позволяет изменять атрибуты доступа файлов. К примеру, есть тестовый файл patch\_4gb.exe, выбрав который мы сможем выполнять следующие действия:

* задать доступ – удаляет режимы доступа, имеющиеся у файла, и задаёт указанный;
* предоставить доступ – к уже имеющимся режимам доступа добавляется указанный. Если указанный режим уже задан – состояние файла не меняется;
* отменить доступ – отменяет существующий режим доступа, соответствующий указанному. Если указанный режим не задан – состояние файла не меняется;
* запретить доступ – ограничивает доступ к файлу для указанного режима. Если доступ для указанного режима запрещён – состояние файла не меняется.

## Описание логической структуры

Не смотря на то, что в качестве основной части приложения пользователю предоставляется диалоговое окно для изменения атрибутов доступа файлов, основной частью курсового проекта является неявно выполняющаяся проверка, подтверждающая или опровергающая факт, что приложение выполняется на устройстве, которому оно принадлежит. Сам факт запуска графической части приложения является свидетельством успешного выполнения проверки. Рассмотрим каждую из частей более подробно в разделах 3.3.1 и 3.3.2 для части с проверкой и части с графическим интерфейсом соответственно.

### **Проверка принадлежности приложения к устройству**

Алгоритм проверки изображён на рисунке 3.3.1.1. Объявляем переменную, в которой будет храниться значение MachineGuid ключа HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Cryptography, содержащего уникальный ключ операционной системы. Операция выполняется кодом из листинга 3.3.1.1.

char value[37];

DWORD BufferSize = sizeof(value);

operationResult = RegGetValueA(HKEY\_LOCAL\_MACHINE, "SOFTWARE\\Microsoft\\Cryptography", "MachineGuid", RRF\_RT\_REG\_SZ, NULL, value, &BufferSize);

Листинг 3.3.1.1 – получение значения MachineGuid[5]

В случае успешного получения значения (1), оно сравнивается (2) с заранее указанным ключом, введённым вручную на этапе помещения приложения на новый персональный компьютер. Если в случае (1) или (2) алгоритм получает в результате проверки FALSE – из функции возвращается false, иначе, если все проверки вернули TRUE, то результатом выполнения функции будет true. Реализация приведена в листинге 3.3.1.2.

if (operationResult != ERROR\_SUCCESS) {

return false;

}

if (strcmp(value, “7fef1237-4ade-41c0-bf14-cf5b1d4bfad8”) != 0) {

return false;

}

return true;

Листинг 3.3.1.2 – проверка валидности значения MachineGuid

Теперь, разобравшись с алгоритмом проверки, мы можем продвигаться в часть 3.3.2, описывающую взаимодействие с пользовательским интерфейсом.

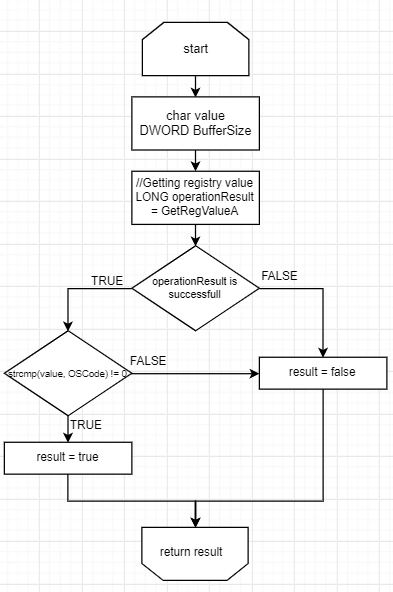


Рисунок 3.3.1.1 – блок-схема алгоритма проверки

### **Графическая часть приложения**

На рисунке 3.3.2.1 указан алгоритм работы графической части приложения. Рисунок 3.3.2.2. отображает окно, с которым пользователь работает, нажав на кнопку Run в главном меню.

Сперва нужно выбрать файл, нажав на кнопку Select, которя показана на рисунке 3.3.2.3, после чего выполняется код из листинга 3.3.2.1.

case ID\_F\_SELECT\_BTN: {

file.lpstrTitle = L"Choose file to open";

if (!GetOpenFileName(&file)) {

MessageBox(hDlg, L"Something went wrong", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

SetWindowText(fNameEdit, file.lpstrFile);

break;

}

Листинг 3.3.2.1 – выбор файла

В результате успешного выполнения в окно 1 рисунка 3.3.2.2 (далее – окно вывода имени файла) будет выведено имя выбранного файла. Далее, в окне 2 рисунка 3.3.2.2 (далее – окно выбора установки режима) нужно выбрать один из режимов. Один из способов показан в листинге 3.3.2.2.

GetWindowTextA(accessCB, access\_mode\_choice, access\_mode\_length+1);

ACCESS\_MODE access\_mode;

if (strcmp(access\_mode\_choice, "Set Access") == 0) {

access\_mode = SET\_ACCESS;

}

else if (strcmp(access\_mode\_choice, "Grant Access") == 0) {

access\_mode = GRANT\_ACCESS;

}

else if (strcmp(access\_mode\_choice, "Deny Access") == 0) {

access\_mode = DENY\_ACCESS;

}

else if (strcmp(access\_mode\_choice, "Revoke Access") == 0) {

access\_mode = REVOKE\_ACCESS;

}

else {

MessageBox(hDlg, L"You have to choose access permission", L"Error", MB\_OK);

return -1;

}

Листинг 3.3.2.2 – установка режима

Затем выбираем желаемые атрибуты в окне 3 рисунка 3.3.2.2 (далее – окно выбора атрибутов) используя код из листинга 3.3.2.3.

DWORD file\_permission = 0;

if (SendDlgItemMessage(hDlg, ID\_FILE\_PERMISSION\_READ, BM\_GETCHECK, NULL, NULL)) {

file\_permission |= GENERIC\_READ;

}

if (SendDlgItemMessage(hDlg, ID\_FILE\_PERMISSION\_WRITE, BM\_GETCHECK, NULL, NULL)) {

file\_permission |= GENERIC\_WRITE;

}

if (SendDlgItemMessage(hDlg, ID\_FILE\_PERMISSION\_EXECUTE, BM\_GETCHECK, NULL, NULL)){

file\_permission |= GENERIC\_EXECUTE;

}

if (SendDlgItemMessage(hDlg, ID\_FILE\_PERMISSION\_ALL, BM\_GETCHECK, NULL, NULL)) {

file\_permission = GENERIC\_ALL;

}

if (file\_permission == 0) {

MessageBox(hDlg, L"You have to choose at least one permission", L"Error", MB\_OK);

return -1;

}

Листинг 3.3.2.3 – выбор атрибутов

И наконец, когда пользователь нажимает на кнопку Run, при условии, что заполнены окна вывода имени файла, выбора установки режима и выбора установки атрибутов доступа с рисунока 3.3.2.5 с файла считываются существующие атрибуты доступа, а затем, в зависимости от указанных пользователем параметров, выполняются соответствтующие действия из листинг 3.3.2.4.

EXPLICIT\_ACCESS ea;

PACL pDACL = NULL;

PSID pSIDEveryone = NULL;

PSECURITY\_DESCRIPTOR pSD = NULL;

SecureZeroMemory(&pSD, sizeof(PSECURITY\_DESCRIPTOR));

SID\_IDENTIFIER\_AUTHORITY SIDAuthWorld = SECURITY\_WORLD\_SID\_AUTHORITY;

SID\_IDENTIFIER\_AUTHORITY SIDAuthNT = SECURITY\_NT\_AUTHORITY;

if (!AllocateAndInitializeSid(&SIDAuthWorld, 1, SECURITY\_WORLD\_RID,

0, 0, 0, 0, 0, 0, 0, &pSIDEveryone)) {

MessageBox(hDlg, (LPCWSTR)GetLastError(), L"AllocateAndInitializeSid (Everyone) error", MB\_OK);

Cleanup(pSIDEveryone, pSD, pDACL);

return -1;

}

ZeroMemory(&ea, sizeof(EXPLICIT\_ACCESS));

ea.grfAccessPermissions = file\_permission;

ea.grfAccessMode = access\_mode;

ea.grfInheritance = NO\_INHERITANCE;

ea.Trustee.TrusteeForm = TRUSTEE\_IS\_SID;

ea.Trustee.TrusteeType = TRUSTEE\_IS\_WELL\_KNOWN\_GROUP;

ea.Trustee.ptstrName = (LPTSTR)pSIDEveryone;

dwRes = SetEntriesInAcl(1, &ea, NULL, &pDACL);

Листинг 3.3.2.4 – установка атрибутов

В случае, если одно из окон не было заполнено, пользователь получит ошибку с соответствующим сообщением, отображённом на рисуноке 3.3.2.4. Для отображения данного сообщения нужно добавить код из листинга 3.3.2.5.

if (dwRes != ERROR\_SUCCESS) {

MessageBox(hDlg, (LPCWSTR)dwRes, L"SetEntriesInAcl failed", MB\_OK);

Cleanup(pSIDEveryone, pSD, pDACL);

return -1;

}

Листинг 3.3.2.5 – вывод ошибки

Если во время смены атрибутов не случилось никаких ошиобок, то пользователь получает сообщение об успешной смене атрибутов доступа. Сообщение показано на рисуноке 3.3.2.6, его реализация приведена в листинге 3.3.2.6.

if (dwRes == ERROR\_SUCCESS) {

MessageBox(hDlg, L"Successfully changed DACL", L"Info", MB\_OK);

Cleanup(pSIDEveryone, pSD, pDACL);

return 0;

}

Листинг 3.3.2.6 – вывод сообщения об успешном завершении операции

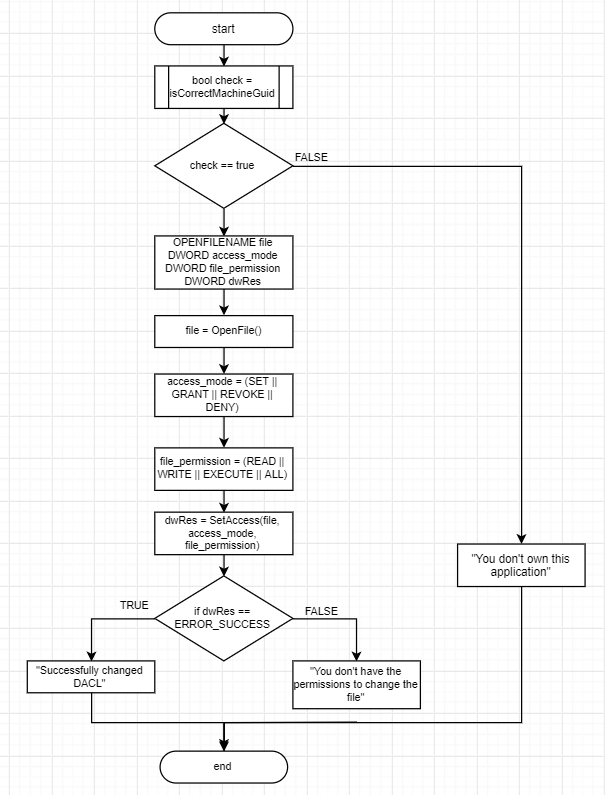


Рисунок 3.3.2.1 – блок-схема алгоритма графической части

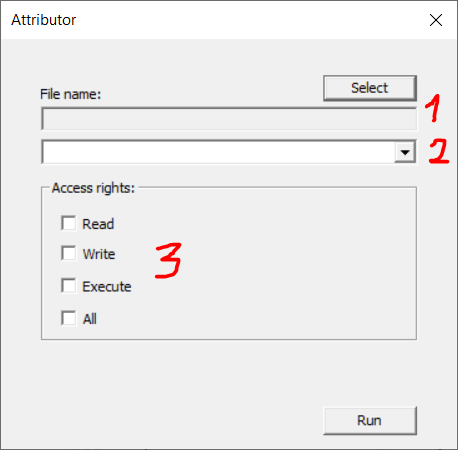


Рисунок 3.3.2.2 – окно изменения атрибутов доступа

Описание элементов рисунка 3.3.2.2:

* 1 – окно вывода имени файла;
* 2 – окно выбора установки режима;
* 3 – окно выбора установки атрибутов доступа.

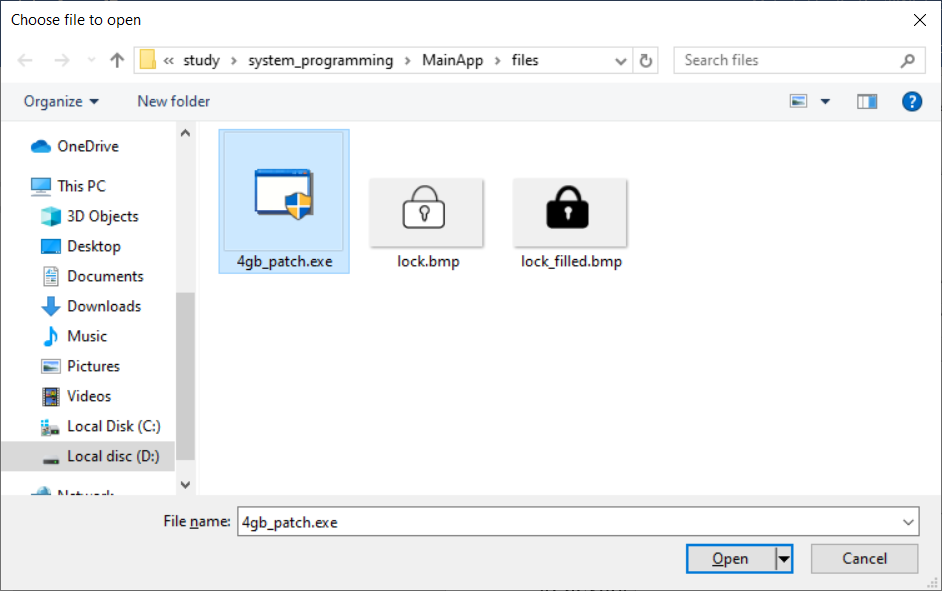


Рисунок 3.3.2.3 – окно выбора файла

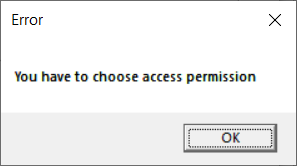


Рисунок 3.3.2.4 – окно ошибки

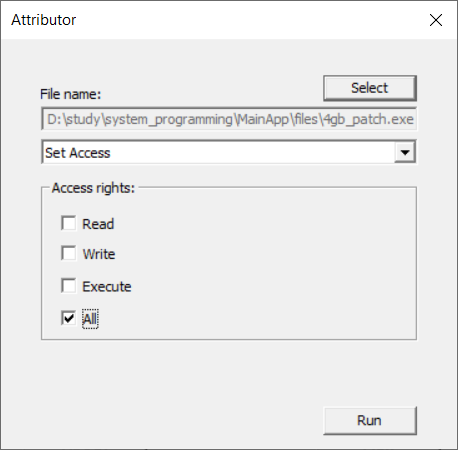


Рисунок 3.3.2.5 – заполненное окно изменения атрибутов доступа

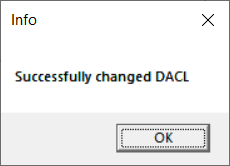


Рисунок 3.3.2.6 – успешная смена атрибутов доступа

## Используемые технические средства

На рисунке 3.4.1 представлены характеристики системы ПК.

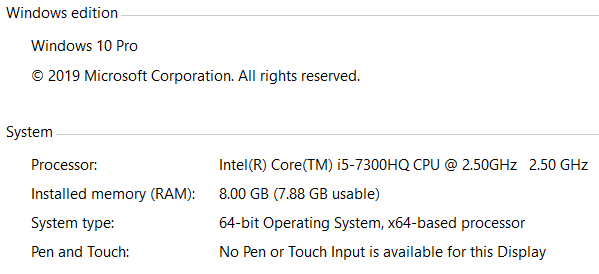


Рисунок 3.4.1 – используемые технические средства

## Вызов и загрузка

Для загрузки приложения достаточно запустить исполняемый MainApp.exe файл, кликнув по нему дважды.

## Входные данные

Входными данными для приложения являются пользовательский выбор в окнах вывода имени файла, выбора установки режима и выбора установки атрибутов доступа.

## Выходные данные

В качестве выходных данных для пользователя служат сообщения об ошибках, или об успешном проведении операции, как на рисунках 3.3.2.4 и 3.3.2.6.

**ВЫВОДЫ**

В ходе выполнения курсового проекта было разработано приложение, имеющее встроенную защиту от несанкционированного доступа, реализованную через проверку уникального кода операционной системы, находящегося в реестре Windows. Для получения значения из реестра была изучена функция GetRegValueA, задающее передаваемой в функцию переменной значение указанной записи.

Метод привязки приложения к уникальным номерам имеет как плюсы, так и минусы. Из преимуществ можно выделить простоту в реализации, а также более высокую устойчивость против генераторов ключей по сравнению с использованием передачи пользователю ключа, сгенерированного сервером. Однако, значительным недостатком является фактор отсутствия возможности массового распространеения. К тому же, чтобы пользоваться программой, нужно иметь устройство, на которое оно установлено, при себе, что не всегда является возможным. Также, опытный пользователь сможет обойти такую защиту, сэмулировав нужные характеристики.

Таким образом, учитывя вышеперечисленные плюсы и минусы, использование такого средства защиты подходит для компьютерных клубов, или оффисных зданий, иными словами для помещений, в которых обновление оборудования и программного обеспечения часто не происходит.

Были изучены функции по работе с реестром Windows, в частности GetRegValueA, записывающая значение указанной записи в передаваемую переменную.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. How does a keygen generator actually comes up with a valid registration key?: askscience [Электронный ресурс]. Режим доступа:

<https://www.reddit.com/r/askscience/comments/25594a/how_does_a_keygen_generator_actually_come_up_with/>

1. Защита информации [Электронный ресурс]. Режим доступа: <https://stud-baza.ru/zaschita-dannyih-ot-nesanktsionirovannogo-dostupa-diplomnaya-rabota-informatika-programmirovanie-79662>
2. Реестр Windows – Документация [Электронный ресурс]. Режим доступа: <https://regedit.readthedocs.io/introduce.html#windows>
3. Сведения о реестре Windows для опытных пользователей [Электронный ресурс]. Режим доступа: <https://support.microsoft.com/ru-ru/help/256986/windows-registry-information-for-advanced-users>
4. RegGetValueA function (winreg.h) – Win32 apps | Microsoft Docs [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/en-us/windows/win32/api/winreg/nf-winreg-reggetvaluea>
5. ГОСТ 2.105–95 Единая система конструкторской документации. Общие требования к текстовым документам.
6. ГОСТ 2.104–2006 Единая система конструкторской документации. Основные надписи.

# **ПРИЛОЖЕНИЕ А**

Файл DllApp.h:

#pragma once

#define EXPORT \_\_declspec(dllexport)

EXPORT bool IsCorrectMachineGuid();

Файл DllApp.cpp:

// DllApp.cpp : Defines the exported functions for the DLL application.

//

#include "stdafx.h"

#include "DllApp.h"

bool IsCorrectMachineGuid() {

char value[37];

DWORD BufferSize = sizeof(value);

LONG operationResult = RegGetValueA(HKEY\_LOCAL\_MACHINE, "SOFTWARE\\Microsoft\\Cryptography", "MachineGuid", RRF\_RT\_REG\_SZ, NULL, value, &BufferSize);

if (operationResult != ERROR\_SUCCESS) {

return false;

}

if (strcmp(value, "7fef1237-4ade-41c0-bf14-cf5b1d4bfad8") != 0) {

return false;

}

return true;

}

Файл MainApp.cpp:

// MainApp.cpp : Defines the entry point for the application.

//

#include "stdafx.h"

#include "MainApp.h"

#include "aclapi.h"

#include <commdlg.h>

#include "..\\DllApp\\DllApp.h"

#pragma comment(lib, "..\\x64\\Debug\\DllApp.lib")

using namespace std;

#define MAX\_LOADSTRING 100

#define ID\_RUN\_BTN 1101

#define ID\_ABOUT\_BTN 1102

// Global Variables:

HINSTANCE hInst; // current instance

WCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

WCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK Help(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK Attributor(HWND, UINT, WPARAM, LPARAM);

HWND hWnd;

HBITMAP hBmp;

void Cleanup(PSID pSIDEveryone, PSECURITY\_DESCRIPTOR pSD, PACL pNewDACL);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

// Initialize global strings

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_MAINAPP, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_MAINAPP));

MSG msg;

// Main message loop:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// FUNCTION: MyRegisterClass()

//

// PURPOSE: Registers the window class.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_MAINAPP));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_MAINAPP);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

//

// FUNCTION: InitInstance(HINSTANCE, int)

//

// PURPOSE: Saves instance handle and creates main window

//

// COMMENTS:

//

// In this function, we save the instance handle in a global variable and

// create and display the main program window.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Store instance handle in our global variable

DWORD btnStyle = WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON;

DWORD txtStyle = WS\_CHILD | WS\_VISIBLE;

int wndWidth = 400;

int wndHeight = 450;

int btnWidth = 100;

int btnHeight = 30;

hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

500, 100, wndWidth, wndHeight, nullptr, nullptr, hInstance, nullptr);

HWND bgsStatic = CreateWindow(L"STATIC", NULL, txtStyle,

0, 240, wndWidth, wndHeight, hWnd, NULL, hInstance, NULL);

HWND madeByStatic = CreateWindow(L"STATIC", L"Made by", txtStyle,

220, 250, 100, 20, hWnd, NULL, hInstance, NULL);

HWND nameStatic = CreateWindow(L"STATIC", L"Khramkov Dmitry", txtStyle,

250, 270, 150, 20, hWnd, NULL, hInstance, NULL);

HWND groupStatic = CreateWindow(L"STATIC", L"gr. 10702217", txtStyle,

250, 290, 150, 20, hWnd, NULL, hInstance, NULL);

HWND startBtn = CreateWindow(L"button", L"Run", btnStyle, 260, 340, btnWidth, btnHeight,

hWnd, (HMENU)ID\_RUN\_BTN, hInstance, NULL);

HWND aboutBtn = CreateWindow(L"Button", L"About", btnStyle, 25, 340, btnWidth, btnHeight,

hWnd, (HMENU)ID\_ABOUT\_BTN, hInstance, NULL);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// FUNCTION: Cleanup(PSID, PSECURITY\_DESCRIPTOR, PACL)

//

// PURPOSE: Free all dynamic resources

//

//

void Cleanup(PSID pSIDEveryone, PSECURITY\_DESCRIPTOR pSD, PACL pNewDACL) {

if (pSIDEveryone) {

FreeSid(pSIDEveryone);

}

if (pSD) {

LocalFree((HLOCAL)pSD);

}

if (pNewDACL) {

LocalFree((HLOCAL)pNewDACL);

}

}

//

// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// PURPOSE: Processes messages for the main window.

//

// WM\_COMMAND - process the application menu

// WM\_PAINT - Paint the main window

// WM\_DESTROY - post a quit message and return

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE: {

// Here we are checking if the application is runned at specific machine by its GUID

if (!IsCorrectMachineGuid()) {

MessageBox(hWnd, L"This software doesn't belong to you", L"Error", MB\_OK);

return -1;

}

hBmp = (HBITMAP)LoadImageA(NULL, "D:\\study\\system\_programming\\MainApp\\files\\lock.bmp", IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

break;

}

case WM\_COMMAND: {

int wmId = LOWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case ID\_RUN\_BTN: {

hBmp = (HBITMAP)LoadImageA(NULL, "D:\\study\\system\_programming\\MainApp\\files\\lock\_filled.bmp", IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

DialogBox(hInst, MAKEINTRESOURCE(IDD\_MAIN\_MENU), hWnd, Attributor);

InvalidateRect(hWnd, NULL, FALSE);

break;

}

case ID\_MORE\_HELP: {

DialogBox(hInst, MAKEINTRESOURCE(ID\_HELPBOX), hWnd, Help);

break;

}

case ID\_ABOUT\_BTN: {

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

}

case IDM\_EXIT: {

DestroyWindow(hWnd);

break;

}

default: {

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

}

case WM\_PAINT: {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

BITMAP bm;

HDC hMemDc = CreateCompatibleDC(hdc);

SelectObject(hMemDc, hBmp);

GetObject(hBmp, sizeof(bm), &bm);

BitBlt(hdc, 0, 0, bm.bmWidth, bm.bmHeight, hMemDc, 0, 0, SRCCOPY);

DeleteDC(hMemDc);

// TODO: Add any drawing code that uses hdc here...

EndPaint(hWnd, &ps);

break;

}

case WM\_DESTROY: {

PostQuitMessage(0);

break;

}

default: {

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

return 0;

}

//

// FUNCTION: Attributor(HWND, UINT, WPARAM, LPARAM)

//

// PURPOSE: allows us to play around with file access attributes.

//

// WM\_INITDIALOG - adds values in ComboBox and sets default file

// WM\_COMMAND - process the messages

// WM\_DESTROY - post a quit message and return

//

//

INT\_PTR CALLBACK Attributor(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {

HWND fNameEdit = GetDlgItem(hDlg, ID\_FNAME\_EDIT);

HWND accessCB = GetDlgItem(hDlg, ID\_ACCESS\_COMBO\_BOX);

const DWORD MaxLength = 0x7fff;

static TCHAR name[256] = L"";

static OPENFILENAME file;

static HANDLE hFile;

static char text[MaxLength];

static DWORD nCharRead;

file.lStructSize = sizeof(OPENFILENAME);

file.hInstance = hInst;

file.lpstrFile = name;

file.nMaxFile = 256;

file.lpstrInitialDir = L"..\\files";

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG: {

// Adding values to ComboBox

SendMessage(accessCB, CB\_ADDSTRING, NULL, (LPARAM)L"Set Access");

SendMessage(accessCB, CB\_ADDSTRING, NULL, (LPARAM)L"Grant Access");

SendMessage(accessCB, CB\_ADDSTRING, NULL, (LPARAM)L"Deny Access");

SendMessage(accessCB, CB\_ADDSTRING, NULL, (LPARAM)L"Revoke Access");

break;

} case WM\_COMMAND: {

int wmId = LOWORD(wParam);

switch (wmId)

{

case ID\_MAIN\_RUN\_BTN: {

int filename\_length = GetWindowTextLength(fNameEdit);

char \*filename = new char[filename\_length];

GetWindowTextA(fNameEdit, filename, filename\_length+1);

if (strcmp(filename, "") == 0) {

MessageBox(hWnd, L"You have to choose the file first", L"Error", MB\_OK);

return -1;

}

// Defines the access mode

const int access\_mode\_length = GetWindowTextLength(accessCB);

char \*access\_mode\_choice = new char[access\_mode\_length];

GetWindowTextA(accessCB, access\_mode\_choice, access\_mode\_length+1);

ACCESS\_MODE access\_mode;

if (strcmp(access\_mode\_choice, "Set Access") == 0) {

access\_mode = SET\_ACCESS;

}

else if (strcmp(access\_mode\_choice, "Grant Access") == 0) {

access\_mode = GRANT\_ACCESS;

}

else if (strcmp(access\_mode\_choice, "Deny Access") == 0) {

access\_mode = DENY\_ACCESS;

}

else if (strcmp(access\_mode\_choice, "Revoke Access") == 0) {

access\_mode = REVOKE\_ACCESS;

}

else {

MessageBox(hDlg, L"You have to choose access permission", L"Error", MB\_OK);

return -1;

}

// Defines the file permission

DWORD file\_permission = 0;

if (SendDlgItemMessage(hDlg, ID\_FILE\_PERMISSION\_READ, BM\_GETCHECK, NULL, NULL)) {

file\_permission |= GENERIC\_READ;

}

if (SendDlgItemMessage(hDlg, ID\_FILE\_PERMISSION\_WRITE, BM\_GETCHECK, NULL, NULL)) {

file\_permission |= GENERIC\_WRITE;

}

if (SendDlgItemMessage(hDlg, ID\_FILE\_PERMISSION\_EXECUTE, BM\_GETCHECK, NULL, NULL)) {

file\_permission |= GENERIC\_EXECUTE;

}

if (SendDlgItemMessage(hDlg, ID\_FILE\_PERMISSION\_ALL, BM\_GETCHECK, NULL, NULL)) {

file\_permission = GENERIC\_ALL;

}

if (file\_permission == 0) {

MessageBox(hDlg, L"You have to choose at least one permission", L"Error", MB\_OK);

return -1;

}

DWORD dwRes = 0;

EXPLICIT\_ACCESS ea;

// Existing and new DACL pointers...

PACL pDACL = NULL;

// Reserving memory for security ID's

PSID pSIDEveryone = NULL;

// Security descriptor

PSECURITY\_DESCRIPTOR pSD = NULL;

SecureZeroMemory(&pSD, sizeof(PSECURITY\_DESCRIPTOR));

SID\_IDENTIFIER\_AUTHORITY SIDAuthWorld = SECURITY\_WORLD\_SID\_AUTHORITY;

SID\_IDENTIFIER\_AUTHORITY SIDAuthNT = SECURITY\_NT\_AUTHORITY;

// Specify the DACL to use

// Create a SID for the Everyone group

if (!AllocateAndInitializeSid(&SIDAuthWorld, 1, SECURITY\_WORLD\_RID,

0, 0, 0, 0, 0, 0, 0, &pSIDEveryone)) {

MessageBox(hDlg, (LPCWSTR)GetLastError(), L"AllocateAndInitializeSid (Everyone) error", MB\_OK);

Cleanup(pSIDEveryone, pSD, pDACL);

return -1;

}

ZeroMemory(&ea, sizeof(EXPLICIT\_ACCESS));

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

// block access by certain user

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

ea.grfAccessPermissions = file\_permission;

ea.grfAccessMode = access\_mode;

ea.grfInheritance = NO\_INHERITANCE;

ea.Trustee.TrusteeForm = TRUSTEE\_IS\_SID;

ea.Trustee.TrusteeType = TRUSTEE\_IS\_WELL\_KNOWN\_GROUP;

ea.Trustee.ptstrName = (LPTSTR)pSIDEveryone;

// Creates a new ACL that merges the new ACE into the existing DACL.

dwRes = SetEntriesInAcl(1, &ea, NULL, &pDACL);

// Verify

if (dwRes != ERROR\_SUCCESS) {

MessageBox(hDlg, (LPCWSTR)dwRes, L"SetEntriesInAcl failed", MB\_OK);

Cleanup(pSIDEveryone, pSD, pDACL);

return -1;

}

wchar\_t \*filename\_buf = new wchar\_t[filename\_length];

mbstowcs(filename\_buf, filename, filename\_length+1); // converts char\* into wchar\_t\*

// Attach the new ACL as the object's DACL.

dwRes = SetNamedSecurityInfo((LPWSTR)filename\_buf, SE\_FILE\_OBJECT,

DACL\_SECURITY\_INFORMATION, NULL, NULL, pDACL, NULL);

if (dwRes == ERROR\_ACCESS\_DENIED) {

MessageBox(hDlg, L"You don't have the permissions to change the file", L"Error", MB\_OK);

Cleanup(pSIDEveryone, pSD, pDACL);

return -1;

}

if (dwRes == ERROR\_SUCCESS) {

MessageBox(hDlg, L"Successfully changed DACL", L"Info", MB\_OK);

Cleanup(pSIDEveryone, pSD, pDACL);

return 0;

}

break;

}

case ID\_F\_SELECT\_BTN: {

file.lpstrTitle = L"Choose file to open";

if (!GetOpenFileName(&file)) {

MessageBox(hDlg, L"Something went wrong", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

SetWindowText(fNameEdit, file.lpstrFile);

break;

}

}

break;

}

case WM\_CLOSE: {

EndDialog(hDlg, LOWORD(wParam));

break;

}

}

return 0;

}

//

// FUNCTION: About(HWND, UINT, WPARAM, LPARAM)

//

// PURPOSE: displays About dialog

//

// WM\_COMMAND - process the messages

//

//

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_COMMAND: {

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

}

return (INT\_PTR)FALSE;

}

//

// FUNCTION: Help(HWND, UINT, WPARAM, LPARAM)

//

// PURPOSE: displays Help dialog with the information about how to use the application

//

// WM\_COMMAND - process the messages

// WM\_DESTROY - post a quit message and return

//

//

INT\_PTR CALLBACK Help(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_COMMAND: {

break;

}

case WM\_CLOSE: {

EndDialog(hDlg, LOWORD(wParam));

break;

}

}

return 0;

}

# **ПРИЛОЖЕНИЕ Б**