Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей Кафедра информатики

Дисциплина: Операционные среды и системное программирование

**ОТЧЕТ**

к лабораторной работе № 5

на тему «Реестр и журналы (Windows). Доступ к реестру Windows. Работа с журналами Windows. Другие вспомогательные средства управления»

Выполнил:

студент гр. 153504

Хрищанович А.К.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 6](#_Toc146631500)

[Выводы 10](#_Toc146631501)

[Список использованных источников 11](#_Toc146631502)

[Приложение А (обязательное) Листинг кода 12](#_Toc146631503)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является создание утилиты для установки и удаления программ с использование информации из реестра Windows.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений.[1]

Реестр Windows и журналы событий играют важную роль в управлении и мониторинге операционной системы Microsoft Windows. Реестр – это централизованное хранилище системных настроек, параметров и данных о приложениях. Он используется для хранения информации о конфигурации операционной системы, установленных приложениях, драйверах и других системных ресурсах. Реестр предоставляет доступ к этим данным и позволяет как системным администраторам, так и приложениям управлять системными параметрами.

Журналы событий Windows служат для регистрации и мониторинга событий и ошибок в операционной системе. Они предоставляют информацию о работе системы, приложениях и службах, а также могут содержать данные о безопасности и аудите. Журналы событий помогают администраторам и разработчикам выявлять проблемы, мониторить производительность и обнаруживать события, связанные с безопасностью.

Для управления реестром Windows и анализа журналов событий доступны различные инструменты и API. Они позволяют выполнять операции чтения, записи и анализа данных в реестре, а также мониторить и фильтровать события в журналах. Для обеспечения безопасности и надежности системы важно правильно использовать эти средства управления, чтобы предотвратить несанкционированный доступ и обнаруживать проблемы в работе операционной системы.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

1. Работа с реестром Windows: при написании кода использовались функции из Windows API (Application Programming Interface) для работы с реестром, такие как RegOpenKeyEx, RegCreateKey, RegSetValueEx, RegQueryValueEx, и RegCloseKey. Эти функции позволяют создавать, читать и записывать данные в реестр.

2. Процессы и потоки: при написании кода использовались функции для запуска инсталлятора и деинсталлятора программ с использованием ShellExecuteEx и WaitForSingleObject. Это связано с управлением процессами и ожиданием их завершения.

3. Ввод и вывод через командную строку: при написании кода были прописаны взаимодействия с пользователем, предлагая меню с опциями и принимая выбор пользователя.

4. Обработка ошибок и проверка результата операций: в коде используются условные операторы для проверки результатов операций, такие как открытие или создание ключей в реестре, выполнение инсталлятора или деинсталлятора.

5. Многозадачность: при написании кода использовались функции ожидания завершения процесса, что позволяет управлять многозадачным выполнением операций.

6. Работа с строками: при написании кода были использованы операции с C++ строками, включая ввод, вывод и манипуляции.

7. Интерфейс командной строки: Код предоставляет текстовый интерфейс, который позволяет пользователю выбирать опции и взаимодействовать с программой через команды.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано простое консольное приложение, представляющее собой утилиту для установки и удаления программ с использованием информации из реестра Windows. При использовании утилиты пользователь может взаимодействовать с меню, результат работы с которым представлен на рисунке 3.1.

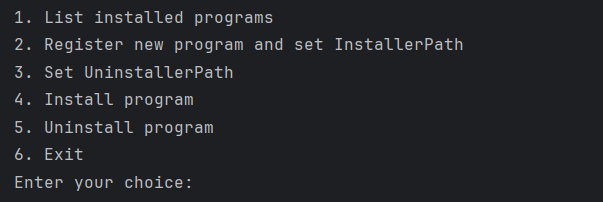


Рисунок 3.1 – Меню программы

При выборе в меню пункта номер 1, пользователю выводится писок существующих подключей в дочернем подключе выбранного родительского ключа. Результат работы вывода списка предоставлен на рисунке 3.2.



Рисунок 3.2 – Результат вывода списка подключей дочернего подключа родительского ключа

На данный момент в реестре на выбранном пути находиться всего лишь один подключ, который будет удален посредством утилиты regedit для последующего представления возможностей программы. При пункта номер 2 пользователю дается возможность ввести имя программы, а также путь до инсталлятора, которые будут занесены в реестр. Работа утилиты, а также результат регистрации новой программы и установки пути до инсталлятора будут приведены на рисунках 3.3 и 3.4 соответственно.

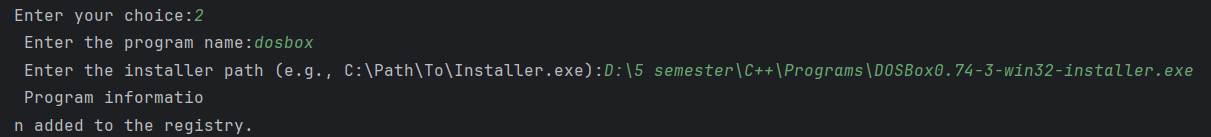


Рисунок 3.3 – Результат работы пункта номер 2

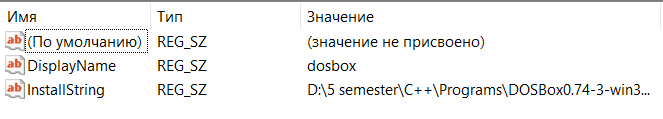


Рисунок 3.4 – Результат регистрации новой программы и установки пути до инсталлятора

Пока мы не имеем пути до деинсталлятора, поэтому пропустим пункт номер 3. Если же путь до инсталлятора не будет прописан, то в реестр в значении UninstallString добавится пустая строка. Выберем пункт номер 4, который дает возможность пользователю установить программу, используя имя программы и прописанный путь к инсталлятору. Результат работы пункта номер 4 будет приведен на рисунках 3.5 и 3.6 соответственно.

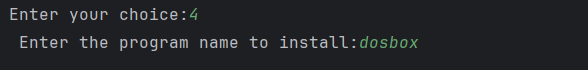


Рисунок 3.5 – Выбор из меню пункта номер 4

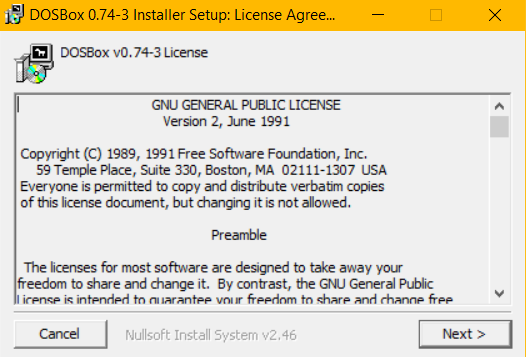


Рисунок 3.6 – Результат работы инсталлятора программы

При выборе пользователем из меню пункта номер 3, ему дается возможность ввести путь к деинсталлятору для выбранной программы, который после будет занесен в реестр. Результаты работы выборки из меню пункта номер 3 будут представлены на рисунках 3.7 и 3.8 соответственно.

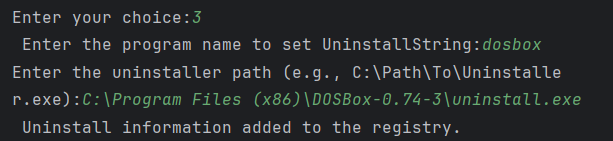


Рисунок 3.7 – Выборка из меню пункта номер 3 и ввод пути к деинсталлятору

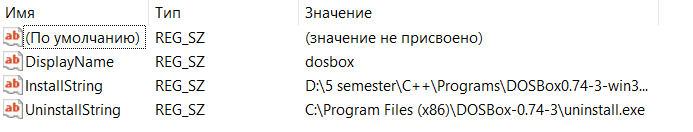


Рисунок 3.8 – Результат добавления пути к деинсталлятору

После добавления пути к деинсталлятору пользователь может удалить программу, выбрав пункт номер 5 из меню. Результаты работы выборки пункта 5 из меню будут предоставлены на рисунках 3.9 и 3.10 соответственно.

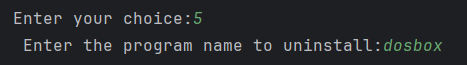


Рисунок 3.9 – Выборка из меню пункта номер 5

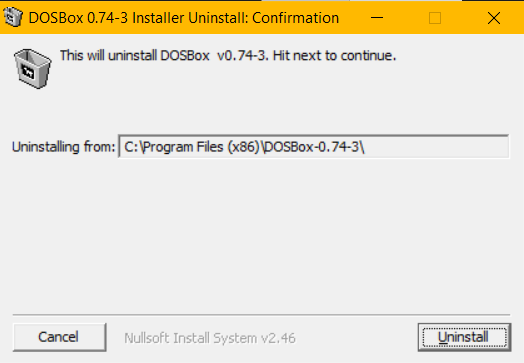


Рисунок 3.10 – Результат работы деинсталлятора программы

При последующих взаимодействиях с утилитой, пользователь без лишних взаимодействий с программой может использовать пути к инсталлятору и деинсталлятору для установки и удаления программы соответственно, если они будут располагаться на тех же путях в системе.

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы было создано простое консольное приложение, являющееся утилитой для установки и удаления программ с использованием информации из реестра.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.
2. [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/debug/system-error-codes--0-499 – Дата доступа 21.10.2023
3. [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecutea – Дата доступа: 23.10.2023

**ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

**main.cpp**

#include <Windows.h>

#include <iostream>

#include <string>

// Регистрируем новую программу в реестре с установкой пути к инсталятору

bool SetProgramRegistry(const std::string& programName, const std::string& installerPath) {

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, "SOFTWARE\\MyProject", 0, KEY\_WRITE, &hParentKey) == ERROR\_SUCCESS) {

HKEY hAppKey;

if (RegCreateKey(hParentKey, programName.c\_str(), &hAppKey) == ERROR\_SUCCESS) {

RegSetValueEx(hAppKey, "DisplayName", 0, REG\_SZ, (const BYTE\*)programName.c\_str(), programName.size());

RegSetValueEx(hAppKey, "InstallString", 0, REG\_SZ, (const BYTE\*)installerPath.c\_str(), installerPath.size());

RegCloseKey(hAppKey);

RegCloseKey(hParentKey);

return true;

}

}

return false;

}

// Запускаем инсталятор

bool RunInstaller(const std::string& installerPath) {

SHELLEXECUTEINFO ShExecInfo = {0};

ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);

ShExecInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;

ShExecInfo.lpFile = installerPath.c\_str();

ShExecInfo.nShow = SW\_SHOWNORMAL;

if (ShellExecuteEx(&ShExecInfo) == TRUE) {

WaitForSingleObject(ShExecInfo.hProcess, INFINITE);

CloseHandle(ShExecInfo.hProcess);

return true;

}

return false;

}

// Устаналиваем путь к деинсталятору

bool SetUninstallRegistry(const std::string& programName, const std::string& uninstallerPath) {

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, "SOFTWARE\\MyProject", 0, KEY\_WRITE, &hParentKey) == ERROR\_SUCCESS) {

HKEY hAppKey;

if (RegCreateKey(hParentKey, programName.c\_str(), &hAppKey) == ERROR\_SUCCESS) {

RegSetValueEx(hAppKey, "UninstallString", 0, REG\_SZ, (const BYTE\*)uninstallerPath.c\_str(), uninstallerPath.size());

RegCloseKey(hAppKey);

RegCloseKey(hParentKey);

return true;

}

}

return false;

}

// Запускаем деинсталятор

bool RunUninstaller(const std::string& programName) {

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, "SOFTWARE\\MyProject", 0, KEY\_READ, &hParentKey) == ERROR\_SUCCESS) {

HKEY hAppKey;

if (RegOpenKeyEx(hParentKey, programName.c\_str(), 0, KEY\_READ, &hAppKey) == ERROR\_SUCCESS) {

DWORD dataValueSize = 1024;

char uninstallerPath[1024];

if (RegQueryValueEx(hAppKey, "UninstallString", 0, NULL, (LPBYTE)uninstallerPath, &dataValueSize) == ERROR\_SUCCESS) {

SHELLEXECUTEINFO ShExecInfo = {0};

ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);

ShExecInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;

ShExecInfo.lpFile = uninstallerPath;

ShExecInfo.nShow = SW\_SHOWNORMAL;

if (ShellExecuteEx(&ShExecInfo) == TRUE) {

WaitForSingleObject(ShExecInfo.hProcess, INFINITE);

CloseHandle(ShExecInfo.hProcess);

return true;

} else {

std::cout << "Failed to run the uninstaller.\n\*\*\*\n" << std::endl;

return false;

}

}

RegCloseKey(hAppKey);

}

RegCloseKey(hParentKey);

}

return false;

}

// Список подключей в ключе реестра

void ListInstalledPrograms() {

HKEY hParentKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, "SOFTWARE\\MyProject", 0, KEY\_READ, &hParentKey) == ERROR\_SUCCESS) {

DWORD subkeyIndex = 0;

char subkeyName[255];

while (RegEnumKey(hParentKey, subkeyIndex, subkeyName, sizeof(subkeyName)) == ERROR\_SUCCESS) {

HKEY hAppKey;

if (RegOpenKeyEx(hParentKey, subkeyName, 0, KEY\_READ, &hAppKey) == ERROR\_SUCCESS) {

std::cout << "Program: " << subkeyName << std::endl;

RegCloseKey(hAppKey);

}

subkeyIndex++;

}

RegCloseKey(hParentKey);

}

}

int main() {

while (true) {

int choice;

std::string programName;

std::string installerPath;

std::string uninstallerPath;

std::cout << "1. List installed programs\n";

std::cout << "2. Register new program and set InstallerPath\n";

std::cout << "3. Set UninstallerPath\n";

std::cout << "4. Install program\n";

std::cout << "5. Uninstall program\n";

std::cout << "6. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

if (choice == 1) {

ListInstalledPrograms();

} else if (choice == 2) {

std::cout << "Enter the program name: ";

std::cin.ignore();

std::getline(std::cin, programName);

std::cout << "Enter the installer path (e.g., C:\\Path\\To\\Installer.exe): ";

std::getline(std::cin, installerPath);

if (SetProgramRegistry(programName, installerPath)) {

std::cout << "Program information added to the registry." << std::endl;

} else {

std::cout << "Failed to add program information to the registry." << std::endl;

}

} else if (choice == 3) {

std::cout << "Enter the program name to set UninstallString:";

std::cin.ignore();

std::getline(std::cin, programName);

std::cout << "Enter the uninstaller path (e.g., C:\\Path\\To\\Uninstaller.exe): ";

std::getline(std::cin, uninstallerPath);

if (SetUninstallRegistry(programName, uninstallerPath)) {

std::cout << "Uninstall information added to the registry." << std::endl;

} else {

std::cout << "Failed to add uninstall information to the registry." << std::endl;

}

} else if (choice == 4) {

std::cout << "Enter the program name to install: ";

std::cin.ignore();

std::getline(std::cin, programName);

HKEY hInstallKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, "SOFTWARE\\MyProject", 0, KEY\_READ, &hInstallKey) == ERROR\_SUCCESS) {

HKEY hAppKey;

if (RegOpenKeyEx(hInstallKey, programName.c\_str(), 0, KEY\_READ, &hAppKey) == ERROR\_SUCCESS) {

DWORD valueSize = 1024;

char installPath[1024];

if (RegQueryValueEx(hAppKey, "InstallString", 0, NULL, (LPBYTE)installPath, &valueSize) == ERROR\_SUCCESS) {

if (RunInstaller(installPath)) {

std::cout << "Program installed successfully!" << std::endl;

} else {

std::cout << "Failed to run the installer." << std::endl;

}

}

RegCloseKey(hAppKey);

}

RegCloseKey(hInstallKey);

}

} else if (choice == 5) {

std::cout << "Enter the program name to uninstall: ";

std::cin.ignore();

std::getline(std::cin, programName);

HKEY hUninstallKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, "SOFTWARE\\MyProject", 0, KEY\_READ, &hUninstallKey) == ERROR\_SUCCESS) {

HKEY hAppKey;

if (RegOpenKeyEx(hUninstallKey, programName.c\_str(), 0, KEY\_READ, &hAppKey) == ERROR\_SUCCESS) {

DWORD valueSize = 1024;

char uninstallPath[1024];

if (RegQueryValueEx(hAppKey, "UninstallString", 0, NULL, (LPBYTE)uninstallPath, &valueSize) == ERROR\_SUCCESS) {

if (RunUninstaller(programName)) {

std::cout << "Program uninstalled successfully!" << std::endl;

} else {

std::cout << "Failed to run the uninstaller." << std::endl;

}

}

RegCloseKey(hAppKey);

}

RegCloseKey(hUninstallKey);

}

} else if (choice == 6) {

break;

}

}

return 0;

}