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

Князев Н.Д.

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: программа 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 графическое приложение, представляющее собой утилиту для поиска устаревших записей 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 кнопки “Анализировать”, пользователю выводится список устаревших записей в реестре. Результат работы вывода списка предоставлен на рисунке 3.2.

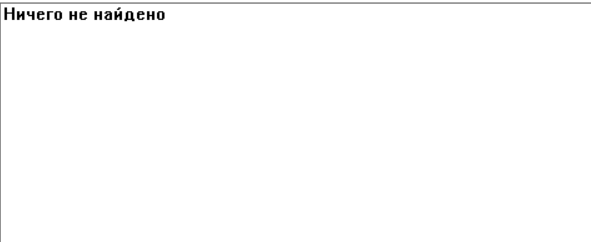


Рисунок 3.2 – Результат вывода списка устаревших записей реестра

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы было создано графическое приложение, являющееся утилитой для поиска устаревших значений 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

**lab5.cpp**

#include <windows.h>

#include <string>

#include <vector>

// Объявление функций

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

std::vector<std::wstring> GetObsoleteRegistryEntries();

void EnumerateRegistryEntries(HKEY hKey, const std::wstring& subKey, std::vector<std::wstring>& obsoleteEntries);

bool IsObsoleteRegistryEntry(HKEY hKey, const std::wstring& valueName);

void AnalyzeRegistry(HWND hwnd);

// Глобальная переменная для хранения дескриптора кнопки

HWND g\_AnalyzeButton;

HWND g\_ListBox;

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow) {

// Регистрация класса окна

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = GetModuleHandle(NULL);

wc.lpszClassName = L"RegistryAnalyzer";

RegisterClass(&wc);

// Создание окна

HWND hwnd = CreateWindowEx(

0, L"RegistryAnalyzer", L"Registry Analyzer", WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 500, 300, NULL, NULL, GetModuleHandle(NULL), NULL

);

if (hwnd == NULL) {

MessageBox(NULL, L"Window creation failed", L"Error", MB\_ICONERROR);

return 1;

}

// Создание кнопки "Проанализировать"

g\_AnalyzeButton = CreateWindow(

L"BUTTON", L"Проанализировать", WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

10, 10, 150, 30, hwnd, (HMENU)1, GetModuleHandle(NULL), NULL

);

// Отображение окна

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

// Основной цикл сообщений

MSG msg = {};

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

switch (uMsg) {

case WM\_CREATE:

return 0;

case WM\_COMMAND:

if (HIWORD(wParam) == BN\_CLICKED) {

// Обработка нажатия кнопки

if (reinterpret\_cast<HWND>(lParam) == g\_AnalyzeButton) {

AnalyzeRegistry(hwnd);

}

}

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

default:

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

}

void AnalyzeRegistry(HWND hwnd) {

std::vector<std::wstring> obsoleteEntries = GetObsoleteRegistryEntries();

// Если список уже создан, удалим его

if (g\_ListBox) {

DestroyWindow(g\_ListBox);

}

// Создаем список

g\_ListBox = CreateWindow(

L"LISTBOX", L"", WS\_VISIBLE | WS\_CHILD | LBS\_STANDARD | LBS\_NOTIFY,

10, 50, 480, 200, hwnd, (HMENU)2, GetModuleHandle(NULL), NULL

);

if (!obsoleteEntries.empty()) {

// Если есть устаревшие записи, добавляем их в список

for (const auto& entry : obsoleteEntries) {

SendMessage(g\_ListBox, LB\_ADDSTRING, 0, (LPARAM)entry.c\_str());

}

}

else {

// Если устаревших записей нет, добавляем сообщение

SendMessage(g\_ListBox, LB\_ADDSTRING, 0, (LPARAM)L"Ничего не найдено");

}

}

std::vector<std::wstring> GetObsoleteRegistryEntries() {

std::vector<std::wstring> obsoleteEntries;

HKEY hKey;

if (RegOpenKeyEx(HKEY\_LOCAL\_MACHINE, L"Software", 0, KEY\_READ, &hKey) == ERROR\_SUCCESS) {

EnumerateRegistryEntries(hKey, L"", obsoleteEntries);

RegCloseKey(hKey);

}

return obsoleteEntries;

}

void EnumerateRegistryEntries(HKEY hKey, const std::wstring& subKey, std::vector<std::wstring>& obsoleteEntries) {

HKEY hSubKey;

if (RegOpenKeyEx(hKey, subKey.c\_str(), 0, KEY\_READ, &hSubKey) == ERROR\_SUCCESS) {

DWORD index = 0;

constexpr DWORD bufferSize = 256;

wchar\_t valueName[bufferSize];

DWORD valueNameSize = bufferSize;

while (RegEnumValue(hSubKey, index, valueName, &valueNameSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS) {

std::wstring fullKey = subKey.empty() ? valueName : subKey + L"\\" + valueName;

if (IsObsoleteRegistryEntry(hSubKey, valueName)) {

obsoleteEntries.push\_back(fullKey);

}

// Сброс размера буфера перед следующей итерацией

valueNameSize = bufferSize;

index++;

}

// Рекурсивный вызов для подразделов

DWORD subIndex = 0;

constexpr DWORD subKeySize = 256;

wchar\_t subKeyName[subKeySize];

DWORD subKeyNameSize = subKeySize;

while (RegEnumKeyEx(hSubKey, subIndex, subKeyName, &subKeyNameSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS) {

std::wstring fullSubKey = subKey.empty() ? subKeyName : subKey + L"\\" + subKeyName;

EnumerateRegistryEntries(hSubKey, fullSubKey, obsoleteEntries);

// Сброс размера буфера перед следующей итерацией

subKeyNameSize = subKeySize;

subIndex++;

}

RegCloseKey(hSubKey);

}

}

bool IsObsoleteRegistryEntry(HKEY hKey, const std::wstring& valueName) {

constexpr DWORD bufferSize = 256;

wchar\_t data[bufferSize];

DWORD dataSize = bufferSize;

DWORD dataType;

if (RegQueryValueEx(hKey, valueName.c\_str(), NULL, &dataType, reinterpret\_cast<BYTE\*>(data), &dataSize) == ERROR\_SUCCESS) {

if (dataType == REG\_SZ || dataType == REG\_EXPAND\_SZ) {

// Проверка, начинается ли значение с "Obsolete"

if (wcsncmp(data, L"Obsolete", wcslen(L"Obsolete")) == 0) {

return true;

}

}

}

return false;

}