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

ОТЧЁТ

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. 153503

Звягинцева В.А.

Проверил:

Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Постановка задачи3

2 Краткие теоретические сведения4

2.1 Реестр4

2.2 Журнал4

3 Результаты выполнения лабораторной работы6

Вывод8

Список использованных источников9

Приложение А10

**1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является разработка приложения для отслеживания уровня звука системы и изменение размера и цвета окна, в зависимости от него, с сохранением цвета в реестр.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**2.1 Реестр**

Центральная иерархическая база данных, используемая в *Windows*98, *Windows CE*, *Windows* *NT* и *Windows*2000, используется для хранения сведений, необходимых для настройки системы для одного или нескольких пользователей, приложений и аппаратных устройств.

Реестр содержит сведения, на которые *Windows* постоянно ссылается во время операции, такие как профили для каждого пользователя, приложения, установленные на компьютере, типы документов, которые могут создаваться, параметры таблицы свойств для папок и значков приложений, оборудование, которое установлено в системе, и используемые порты.

Реестр заменяет большинство текстовых *INI*-файлов, используемых в файлах конфигурации *Windows*3.*x* и *MS-DOS*, таких как *Autoexec.bat* и *Config.sys*. Хотя реестр является общим для нескольких операционных систем *Windows*, между ними существуют некоторые различия. Куст реестра — это группа ключей, подразделов и значений в реестре с набором вспомогательных файлов, содержащих резервные копии данных. Вспомогательные файлы для всех кустов, кроме *HKEY\_CURRENT\_USER*, находятся в папке %*SystemRoot*%\*System*32\*Config* в *Windows NT* 4.0, *Windows* 2000, *Windows XP*, *Windows Server* 2003 и *Windows Vista*. Вспомогательные файлы для *HKEY\_CURRENT\_USER* находятся в папке %*SystemRoot*%\*Profiles*\*Username*. Расширения имён файлов в этих папках указывают тип содержащихся в них данных. Кроме того, отсутствие расширения иногда может указывать на тип содержащихся в них данных. [1]

**2.2 Журнал**

*API* журнала событий *Windows* определяет схему, используемую для написания манифеста инструментирования. Манифест инструментирования идентифицирует поставщика событий и события, которые он регистрирует. Просмотр событий, будет использовать для чтения и отрисовки событий. Чтобы записать события, определённые в манифесте, можно использовать функции, включённые в *API* трассировки событий (*ETW*).

Журнал событий *Windows* заменяет *API* ведения журнала событий, начиная с операционной системы *Windows Vista*.

Приложения и библиотеки *DLL* используют манифест инструментирования для определения поставщиков инструментирования и событий, которые они записывают. Манифест — это *XML*-файл, содержащий элементы, определяющие поставщика. Соглашение заключается в том, чтобы использовать *.man* в качестве расширения для манифеста. Манифест должен соответствовать *XSD* манифесту события.

События можно использовать из каналов или из файлов журнала. Для использования событий можно использовать все события или указать выражение *XPath*, определяющее события, которые вы хотите использовать. Сведения об определении элементов и атрибутов события, которые можно использовать в выражении *XPath*. Для запросов используются следующие функции:

1 *EvtQuery*: функция запроса события. Можно указать порядок, в котором возвращаются события (от старых к новым (по умолчанию) или от новых к старым), а также указать, следует ли допускать неправильно сформированные выражения *XPath* в запросе (дополнительные сведения о том, как функция игнорирует неправильно сформированные выражения.

2 *EvtGetQueryInfo*: функция для определения успешных и неудачных запросов. Если не передать флаг *EvtQueryTolerateQueryErrors*, функция *EvtQuery* завершится ошибкой с первой ошибкой, найденной в запросе. Если запрос завершается сбоем с *ERROR\_EVT\_INVALID\_QUERY*.

3 *EvtNext*: функция перечисляет события в результирующем наборе если вызвать её в цикле, пока она не вернёт *false*, а функция *GetLastError* *ERROR\_NO\_MORE\_ITEMS*. [2]

**3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ**

В ходе выполнения лабораторной работы было разработано приложение для отслеживания уровня звука системы и изменение размера и цвета окна, в зависимости от него, с сохранением цвета в реестр.

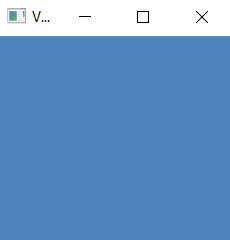


Рисунок 1. – Результат работы программы

Данный цвет окно имеет при значении уровня громкости 34. На рисунке 2 же, уровень громкости равен 100.

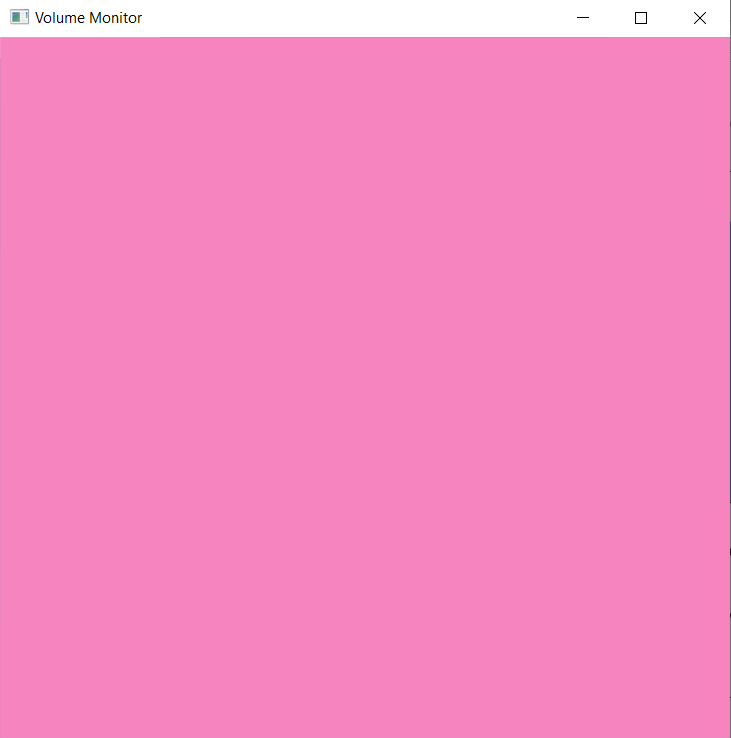


Рисунок 2. – Работа программы при изменении уровня звука пользователем

Далее завершим программу, цвет окна сохранится в реестр и при следующем запуске программы изначальный цвет будет взят из него. Цвет окна изменяется в соотношении: чем ниже уровень звука, тем холоднее цвет и, наоборот, чем выше уровень звука, тем теплее цвет. Также есть прямая пропорциональность уровня звука и размера окна, чем выше уровень, тем больше окно.

Таким образом, при сохранении параметра цвета на уровне звука равного 90 (розовый цвет) и последующем запуске приложения на уровне звука 18 изначальный цвет окна будет розовый, т.к. значение будет взято из реестра и уже после изменится на голубой.



Рисунок 3. – Цвет окна был взят из реестра

**ВЫВОД**

В результате выполнения лабораторной работы были изучены реестр (*Windows*), а также другие вспомогательные средства управления. Также было создано приложение, использующее их для регулирования размера и цвета окна в зависимости от уровня громкости устройства.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Сведения о реестре Windows для опытных пользователей [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/troubleshoot/windows-server/performance/windows-registry-advanced-users.
2. Синхронизация процессов [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/interprocess-synchronization>.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

**main.h**

#pragma once

#include <windows.h>

#include <mmdeviceapi.h>

#include <endpointvolume.h>

#include <iostream>

#include <thread>

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

void MonitorVolumeChanges();

void OnVolumeChange(float);

void CALLBACK TimerProc(HWND, UINT, UINT\_PTR, DWORD);

void SaveColorToRegistry(COLORREF);

bool ReadColorFromRegistry(COLORREF&);

void RestoreColorFromRegistry();

HWND hWnd; // Глобальная переменная для хранения дескриптора окна

COLORREF currentColor = RGB(178, 132, 190);

COLORREF targetColor = RGB(178, 132, 190);

const int timerInterval = 100; // Интервал таймера в миллисекундах

**main.cpp**

#include "main.h"

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };

wcex.lpfnWndProc = WndProc;

wcex.hInstance = hInstance;

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

//wcex.hbrBackground = CreateSolidBrush(RGB(178, 132, 190));

wcex.lpszClassName = L"Volume Monitor";

RegisterClassEx(&wcex);

hWnd = CreateWindow(L"Volume Monitor", L"Volume Monitor", WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 1000, 1000, nullptr, nullptr, hInstance, nullptr);

if (!hWnd) {

return false;

}

RestoreColorFromRegistry();

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

// Создаем отдельный поток для мониторинга громкости

std::thread monitorThread(MonitorVolumeChanges);

// Запускаем таймер для плавного изменения цвета фона

SetTimer(hWnd, 1, timerInterval, nullptr);

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

// Завершаем поток и закрываем программу

KillTimer(hWnd, 1); // Останавливаем таймер перед выходом

monitorThread.detach();

return static\_cast<int>(msg.wParam);

}

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

switch (message) {

case WM\_CLOSE:

DestroyWindow(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

case WM\_TIMER:

TimerProc(hWnd, WM\_TIMER, 1, GetTickCount64());

break;

default:

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

}

return 0;

}

void OnVolumeChange(float newVolume) {

// Считать цвет из реестра при первом изменении громкости

static bool firstChange = true;

if (firstChange) {

COLORREF savedColor;

if (ReadColorFromRegistry(savedColor)) {

currentColor = savedColor;

targetColor = savedColor;

}

firstChange = false;

}

// Пример: увеличьте ширину и высоту окна на значение громкости

int newSize = static\_cast<int>(600 \* newVolume);

// Установить новый размер окна

SetWindowPos(hWnd, nullptr, 0, 0, newSize, newSize, SWP\_NOMOVE | SWP\_NOZORDER);

// Обновить целевой цвет в зависимости от уровня громкости

int red = static\_cast<int>(255 \* newVolume);

targetColor = RGB(red, 132, 190);

// Запустить таймер для плавного изменения цвета

SetTimer(hWnd, 1, 100, nullptr);

// Сохраняем цвет в реестре

SaveColorToRegistry(targetColor);

}

void MonitorVolumeChanges() {

CoInitialize(nullptr);

IMMDeviceEnumerator\* enumerator = nullptr;

HRESULT hr = CoCreateInstance(\_\_uuidof(MMDeviceEnumerator), nullptr, CLSCTX\_ALL, \_\_uuidof(IMMDeviceEnumerator), reinterpret\_cast<void\*\*>(&enumerator));

if (SUCCEEDED(hr)) {

IMMDevice\* defaultDevice = nullptr;

hr = enumerator->GetDefaultAudioEndpoint(eRender, eConsole, &defaultDevice);

if (SUCCEEDED(hr)) {

IAudioEndpointVolume\* endpointVolume = nullptr;

hr = defaultDevice->Activate(\_\_uuidof(IAudioEndpointVolume), CLSCTX\_ALL, nullptr, reinterpret\_cast<void\*\*>(&endpointVolume));

if (SUCCEEDED(hr)) {

while (true) {

float currentVolume = 0.0f;

hr = endpointVolume->GetMasterVolumeLevelScalar(&currentVolume);

if (SUCCEEDED(hr)) {

OnVolumeChange(currentVolume);

}

Sleep(1000); // Пауза на 1 секунду

}

endpointVolume->Release();

}

defaultDevice->Release();

}

enumerator->Release();

}

CoUninitialize();

}

void CALLBACK TimerProc(HWND hwnd, UINT uMsg, UINT\_PTR idEvent, DWORD dwTime) {

// Изменить текущий цвет в сторону целевого цвета

int rDiff = (GetRValue(targetColor) - GetRValue(currentColor)) / 10;

int gDiff = (GetGValue(targetColor) - GetGValue(currentColor)) / 10;

int bDiff = (GetBValue(targetColor) - GetBValue(currentColor)) / 10;

currentColor = RGB(GetRValue(currentColor) + rDiff,

GetGValue(currentColor) + gDiff,

GetBValue(currentColor) + bDiff);

// Установить новый цвет фона окна

HBRUSH hBrush = CreateSolidBrush(currentColor);

SetClassLongPtr(hwnd, GCLP\_HBRBACKGROUND, reinterpret\_cast<LONG\_PTR>(hBrush));

InvalidateRect(hwnd, nullptr, TRUE);

// Если достигнут целевой цвет, остановить таймер

if (currentColor == targetColor) {

KillTimer(hwnd, 1);

}

}

void SaveColorToRegistry(COLORREF color) {

HKEY hKey;

if (RegCreateKeyEx(HKEY\_CURRENT\_USER, L"Software\\YourAppName", 0, nullptr, 0, KEY\_WRITE, nullptr, &hKey, nullptr) == ERROR\_SUCCESS) {

DWORD dwData = color;

RegSetValueEx(hKey, L"WindowColor", 0, REG\_DWORD, (BYTE\*)&dwData, sizeof(dwData));

RegCloseKey(hKey);

}

}

bool ReadColorFromRegistry(COLORREF& color) {

HKEY hKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"Software\\YourAppName", 0, KEY\_READ, &hKey) == ERROR\_SUCCESS) {

DWORD dwData;

DWORD dwSize = sizeof(DWORD);

if (RegQueryValueEx(hKey, L"WindowColor", nullptr, nullptr, reinterpret\_cast<LPBYTE>(&dwData), &dwSize) == ERROR\_SUCCESS) {

color = static\_cast<COLORREF>(dwData);

RegCloseKey(hKey);

return true;

}

RegCloseKey(hKey);

}

return false;

}

void RestoreColorFromRegistry() {

COLORREF restoredColor;

if (ReadColorFromRegistry(restoredColor)) {

// Установить цвет фона окна

HBRUSH hBrush = CreateSolidBrush(restoredColor);

SetClassLongPtr(hWnd, GCLP\_HBRBACKGROUND, reinterpret\_cast<LONG\_PTR>(hBrush));

InvalidateRect(hWnd, nullptr, TRUE);

}

}