Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы и системное программирование

Отчёт

к лабораторной работе

на тему

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

Студент: гр.153504

Пригожий К. А.

Проверил: Гриценко Н. Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146635545)

[2 Теоретические сведения 4](#_Toc146635546)

[3 Результат выполнения программы 5](#_Toc146635547)

[Список использованных источников 7](#_Toc146635548)

[Приложение А (обязательное) Исходный код программы 10](#_Toc146635549)

# 1 ЦЕЛЬ РАБОТЫ

Разработать приложение для мониторинга изменений в системных параметрах и автоматической реакции на изменения такие как: нажатие клавиш на клавиатуре, подключение и отключение питания, изменение разрешения экрана, изменение содержимого буфера обмена, подключение и отключение сторонних устройств к компьютеру.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Системные изменения можно обработать в функции *WndProc* конструкции *switch*.

Сообщение *WM\_DISPLAYCHANGE* отправляется во все окна при изменении разрешения экрана [1].

Сообщение *WM\_CLIPBOARDUPDATE* отправляется при изменении содержимого буфера обмена [2].

*WM*\_*KEYUP* отображается в окне с фокусом клавиатуры при освобождении несистемной клавиши. Несистемная клавиша — это клавиша, которая нажимается, когда клавиша *ALT* не нажата, или клавиша, которая нажимается, когда в окне находится фокус клавиатуры [3].

Сообщение *WM\_POWERBROADCAST* уведомляет приложения о возникновении события управления питанием [4].

Сообщение *WM\_DEVICECHANGE* уведомляет приложение об изменении конфигурации оборудования устройства или компьютера [5].

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Реализованная программа представляет собой приложение с полем для вывода, куда выводятся сообщения о системных событиях (рисунок 1).

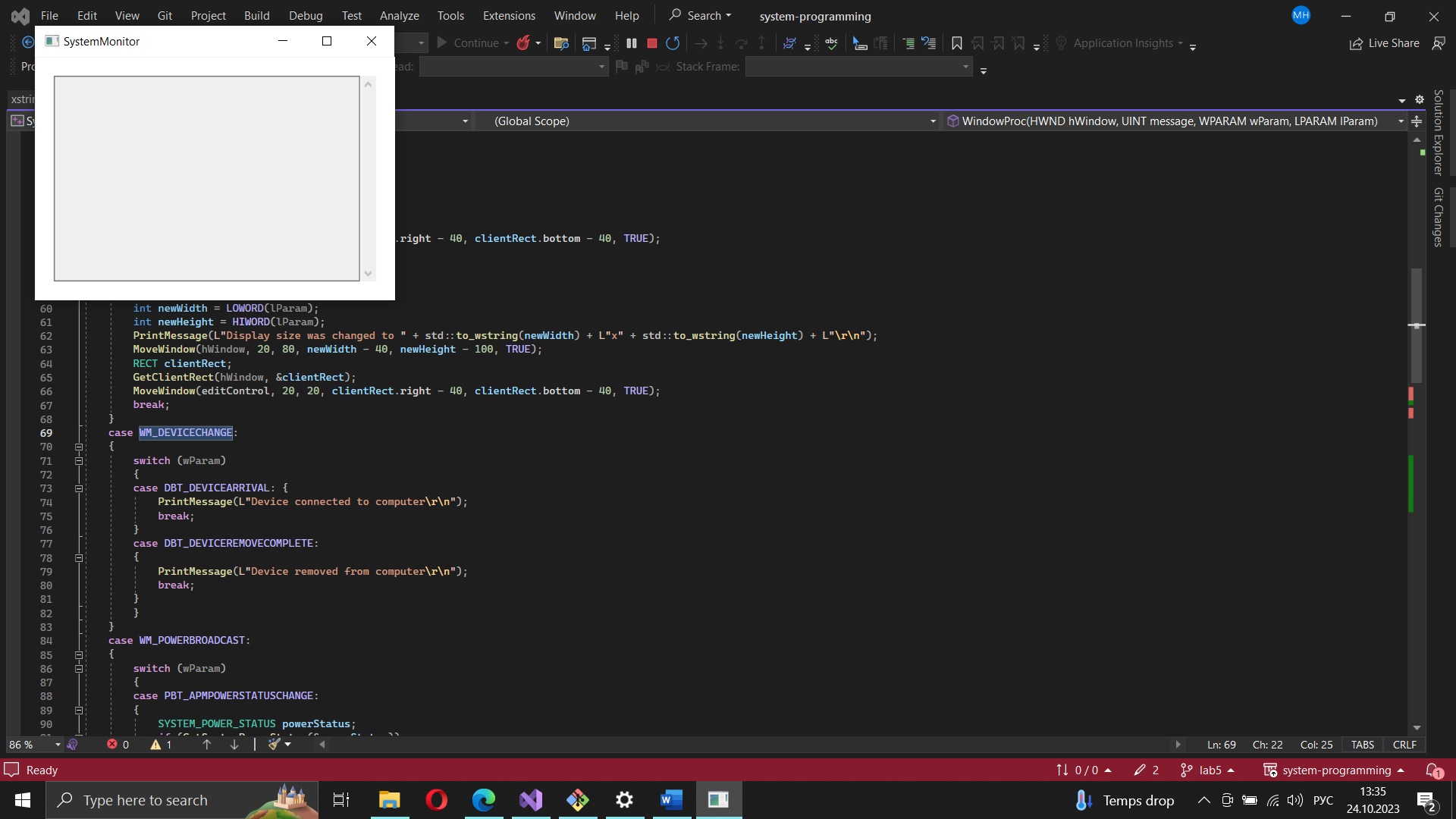


Рисунок 1 – Главное окно приложения

При отключении и подключении питательного устройства к компьютеру выводится сообщение (рисунок 2).

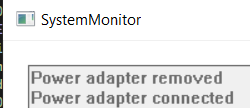


Рисунок 2 – Вывод при отключении и подключении питательного устройства

При нажатии и отпускании выводятся соответствующие сообщения (рисунок 3).



Рисунок 3 – Вывод при нажатии и отпускании клавиши на клавиатуре

При изменении разрешения экрана меняется размер окна приложения и выводится сообщение(рисунок 4)*.*



Рисунок 4 – Вывод при изменении разрешения экрана

При вставке и извлечении портативного носителя выводится сообщение (рисунок 5).



Рисунок 5 – Вывод при вставке и извлечении внешнего носителя

При изменении содержимого буфера обмена выводится сообщение (рисунок 5).



Рисунок 6 – Вывод изменении буфера обмена

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Сообщение *WM\_DISPLAYCHANGE* (*Winuser.h*) - *Win32 apps* [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/gdi/wm-displaychange>

[2] Сообщение *WM\_CLIPBOARDUPDATE (Winuser.h) - Win32 apps* [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-clipboardupdate](https://learn.microsoft.com/en-us/windows/win32/dataxchg/wm-clipboardupdate%20)

[3] Сообщение *WM\_KEYUP* (*Winuser.h*) - *Win32 apps* [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/inputdev/wm-keyup>

[4] Cообщение *WM\_POWERBROADCAST* (*WinUser.h*) - *Win32 apps* | *Microsoft Learn* [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/power/wm-powerbroadcast>

[5] Сообщение *WM\_DEVICECHANGE* (*Winuser.h*) - *Win32 apps* | *Microsoft Learn* [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/devio/wm-devicechange>

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл main.h**

#pragma once

#include <Windows.h>

#include <string>

#include <Dbt.h>

// constants

const wchar\_t APPLICATION\_NAME[] = L"SystemMonitor";

const wchar\_t MAIN\_WINDOW\_CLASS\_NAME[] = L"MainWindowClassName";

const int MAIN\_WINDOW\_INITIAL\_WIDTH = 400;

const int MAIN\_WINDOW\_INITIAL\_HEIGHT = 300;

HWND editControl;

HBRUSH backgroundBrush;

//functions

LRESULT CALLBACK WindowProc(HWND hWindow, UINT message, WPARAM wParam, LPARAM lParam);

HWND InstantiateMainWindow(HINSTANCE hInstance);

WNDCLASSEX CreateMainWindowClass(HINSTANCE hInstance);

void PrintMessage(std::wstring message);

**Файл main.cpp**

#include "main.h"

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR szCmdLine, int nCmdShow)

{

WNDCLASSEX mainWindowClass = CreateMainWindowClass(hInstance);

if (not RegisterClassEx(&mainWindowClass)) {

MessageBoxA(NULL, "Main window hasn't registered", "Error", MB\_OK);

return EXIT\_FAILURE;

}

HWND mainWindow = InstantiateMainWindow(hInstance);

editControl = CreateWindow(

L"EDIT",

L"",

WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_AUTOVSCROLL | WS\_DISABLED | ES\_MULTILINE | WS\_VSCROLL,

20,

80,

500,

300,

mainWindow,

NULL,

(HINSTANCE)GetWindowLongPtr(mainWindow, GWLP\_HINSTANCE),

NULL

);

ShowWindow(mainWindow, nCmdShow);

UpdateWindow(mainWindow);

if (AddClipboardFormatListener(mainWindow))

{

OutputDebugStringA("clipboard listener connected");

}

MSG message;

while (GetMessage(&message, NULL, 0, 0) > 0)

{

TranslateMessage(&message);

DispatchMessage(&message);

}

}

LRESULT CALLBACK WindowProc(HWND hWindow, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message) {

case WM\_CLOSE:

DestroyWindow(hWindow);

RemoveClipboardFormatListener(hWindow);

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

case WM\_SIZE: {

RECT clientRect;

GetClientRect(hWindow, &clientRect);

MoveWindow(editControl, 20, 20, clientRect.right - 40, clientRect.bottom - 40, TRUE);

return 0;

}

case WM\_DISPLAYCHANGE:

{

int newWidth = LOWORD(lParam);

int newHeight = HIWORD(lParam);

PrintMessage(L"Display size was changed to " + std::to\_wstring(newWidth) + L"x" + std::to\_wstring(newHeight) + L"\r\n");

MoveWindow(hWindow, 20, 80, newWidth - 40, newHeight - 100, TRUE);

RECT clientRect;

GetClientRect(hWindow, &clientRect);

MoveWindow(editControl, 20, 20, clientRect.right - 40, clientRect.bottom - 40, TRUE);

break;

}

case WM\_DEVICECHANGE:

{

switch (wParam)

{

case DBT\_DEVICEARRIVAL: {

PrintMessage(L"Device connected to computer\r\n");

break;

}

case DBT\_DEVICEREMOVECOMPLETE:

{

PrintMessage(L"Device removed from computer\r\n");

break;

}

}

}

case WM\_POWERBROADCAST:

{

switch (wParam)

{

case PBT\_APMPOWERSTATUSCHANGE:

{

SYSTEM\_POWER\_STATUS powerStatus;

if (GetSystemPowerStatus(&powerStatus))

{

if (powerStatus.ACLineStatus == AC\_LINE\_ONLINE)

PrintMessage(L"Power adapter connected\r\n");

else

PrintMessage(L"Power adapter removed\r\n");

}

return 0;

}

break;

}

}

case WM\_CLIPBOARDUPDATE:

{

if (OpenClipboard(hWindow))

{

HANDLE hClipboardData = GetClipboardData(CF\_TEXT);

if (hClipboardData)

{

char\* pszData = static\_cast<char\*>(GlobalLock(hClipboardData));

if (pszData)

{

std::string data = pszData;

PrintMessage(L"Clipboard updated: " + std::wstring(data.begin(), data.end()) + L"\r\n");

GlobalUnlock(hClipboardData);

}

}

CloseClipboard();

}

break;

}

case WM\_KEYUP:

{

int virtualKeyCode = (int)wParam;

std::string keyChar = std::string(1, MapVirtualKey(virtualKeyCode, MAPVK\_VK\_TO\_CHAR));

if (isprint(keyChar[0]))

PrintMessage(L"Key up: " + std::wstring(keyChar.begin(), keyChar.end()) + L"\r\n");

else

PrintMessage(L"Key up!\r\n");

break;

}

case WM\_KEYDOWN:

{

int virtualKeyCode = (int)wParam;

std::string keyChar = std::string(1, MapVirtualKey(virtualKeyCode, MAPVK\_VK\_TO\_CHAR));

if (isprint(keyChar[0]))

PrintMessage(L"Key down: " + std::wstring(keyChar.begin(), keyChar.end()) + L"\r\n");

else

PrintMessage(L"Key down!\r\n");

break;

}

default:

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

}

}

WNDCLASSEX CreateMainWindowClass(HINSTANCE hInstance)

{

WNDCLASSEX wc{};

wc.cbSize = sizeof(WNDCLASSEX);

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

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

wc.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

wc.lpszClassName = MAIN\_WINDOW\_CLASS\_NAME;

return wc;

}

HWND InstantiateMainWindow(HINSTANCE hInstance)

{

return CreateWindowEx(

WS\_EX\_CLIENTEDGE,

MAIN\_WINDOW\_CLASS\_NAME, APPLICATION\_NAME,

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, MAIN\_WINDOW\_INITIAL\_WIDTH, MAIN\_WINDOW\_INITIAL\_HEIGHT,

NULL, NULL, hInstance, NULL

);

}

void PrintMessage(std::wstring message)

{

SendMessage(editControl, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

SendMessage(editControl, EM\_REPLACESEL, 0, (LPARAM)message.c\_str());

}