Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

**МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ**

Кафедра програмування та захисту інформації

**Звіт**

з виконаної лабораторної роботи № 4

дисципліни “ Системне програмування”

на тему

«**ОСНОВИ ВИКОРИСТАННЯ WIN API (частина 2)**»

Виконав :

студент академічної групи КІ-15

Аннаєв А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірив :

Викладач

Константинова K.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кропивницкий- 2017

Лабораторна робота № 4

**ТЕМА**: ОСНОВИ ВИКОРИСТАННЯ WIN API (частина 2)

**МЕТА**: Отримати практичні навики в використанні функцій WIN API 32, використання системи повідомлень ОС Windows.

ЗАВДАННЯ Використовуючи наявну електронну документацію по напрямку

програмування з використанням WIN API (функції наведені у завданні), виконати

наступні маленькі завдання.

1. Відключити кнопку закриття будь-якого вказаного вікна (GetSystemMenu).

2. Визначити числа завдань SPOOLERа друку (WM\_SPOOLERSTATUS).

3. Глобально перехопити натискання кнопки Printscreen (WM\_HOTKEY).

4. Визначити факт зміни системного часу іншим ПЗ (WM\_TIMECHANGE).

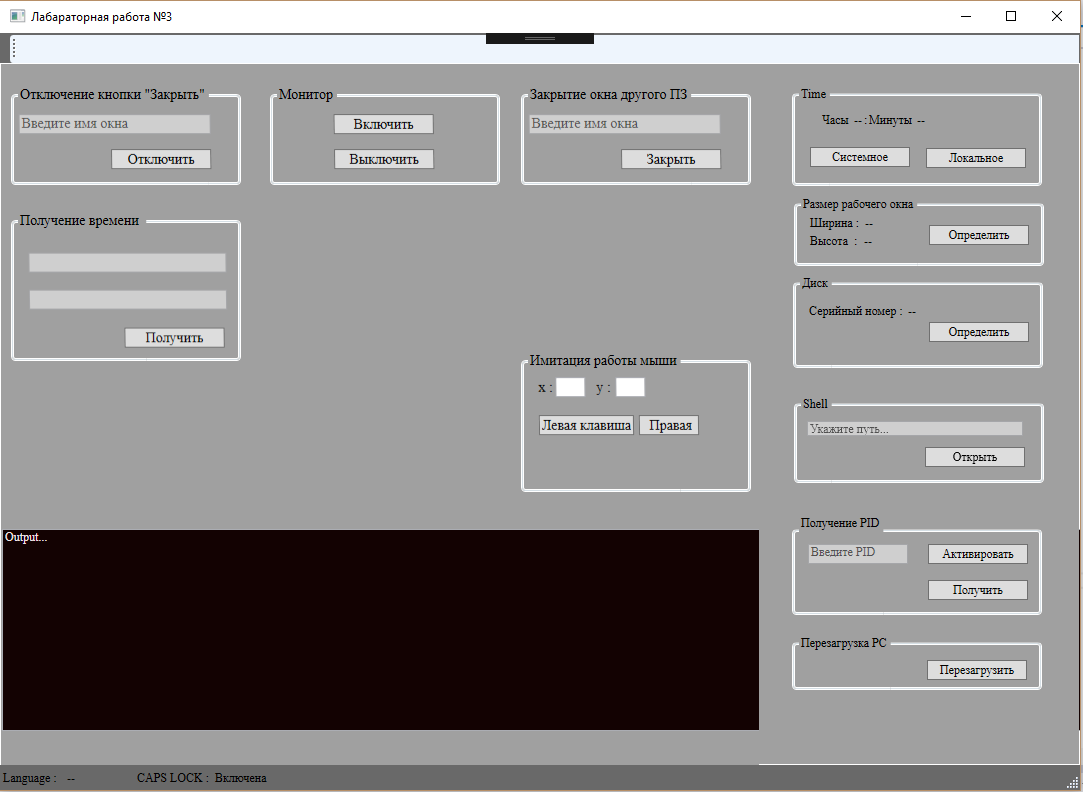
5. Програмно виключити/включити монітор (WM\_SYSCOMMAND).

6. Імітація подій руху та натискання миші (mouse\_event).

7. Програмно закрити іншу програму (WM\_QUIT).

8. Одержати дескриптор вікна іншого ПЗ та зробити його активним

(EnumWindowsProc, BringWindowToTop).



using System;

using System.Diagnostics;

using System.Runtime.InteropServices;

using System.Text;

using System.Windows;

using System.Windows.Input;

using WpfApp1.Tasks;

namespace WpfApp1

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public object InputLanguage { get; private set; }

public MainWindow()

{

InitializeComponent( );

int CAPSLOCK = 20;

int result = (int)GetKeyState(CAPSLOCK);

if (result == 0x8000)

{

txtKey.Text = "Pressed";

}

else

if (result == 1)

{

txtKey.Text = "Включена";

}

else

txtKey.Text = "Выключена";

RegisterHotKey(Process.GetCurrentProcess( ).MainWindowHandle,

0, 0, (int)Key.PrintScreen);

}

public enum ModifierKey : uint

{

MOD\_NULL = 0x0000,

MOD\_ALT = 0x0001,

MOD\_CONTROL = 0x0002,

MOD\_SHIFT = 0x0004,

MOD\_WIN = 0x0008,

}

public enum HotKey

{

PrintScreen,

ALT\_PrintScreen,

CONTROL\_PrintScreen

}

const uint MOUSEEVENTF\_ABSOLUTE = 0x8000;

const uint MOUSEEVENTF\_LEFTDOWN = 0x0002;

const uint MOUSEEVENTF\_LEFTUP = 0x0004;

const uint MOUSEEVENTF\_MIDDLEDOWN = 0x0020;

const uint MOUSEEVENTF\_MIDDLEUP = 0x0040;

const uint MOUSEEVENTF\_MOVE = 0x0001;

const uint MOUSEEVENTF\_RIGHTDOWN = 0x0008;

const uint MOUSEEVENTF\_RIGHTUP = 0x0010;

const uint MOUSEEVENTF\_XDOWN = 0x0080;

const uint MOUSEEVENTF\_XUP = 0x0100;

const uint MOUSEEVENTF\_WHEEL = 0x0800;

const uint MOUSEEVENTF\_HWHEEL = 0x01000;

private int mouse\_coord\_x = 100;

private int mouse\_coord\_y = 100;

[DllImport("user32", SetLastError = true)]

static extern bool RegisterHotKey(IntPtr hWnd, int id,ModifierKey fsModifiers,int vk );

[DllImport("user32")]

static extern short GetKeyState( int key );

[DllImport("Kernel32.dll")]

static extern void GetSystemTime(SystemTime time);

[DllImport("Kernel32.dll")]

static extern void GetLocalTime( SystemTime time );

[DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]

[return: MarshalAs(UnmanagedType.Bool)]

private static extern bool SystemParametersInfo(

int uiAction,

int uiParam,

ref RECT pvParam,

int fWinIni );

[DllImport("Kernel32.dll")]

static extern bool GetVolumeInformation(

string path,

StringBuilder buffer,

int size,

out uint serialNumber,

out uint complength,

out uint flags,

StringBuilder fileBuffer,

int nFileNamesize

);

[DllImport("shell32.dll")]

static extern IntPtr ShellExecute(

IntPtr hwd,

string operation,

string file,

string parametrs,

string directory,

int command

);

[DllImport("user32.dll")]

static extern void mouse\_event( uint dwFlags, int dx, int dy, int dwData, int dwExtraInfo );

[DllImport("user32.dll")]

static extern IntPtr SendMessage( IntPtr hWnd, UInt32 msg, IntPtr wParam);

[DllImport("user32.dll")]

static extern bool DrawMenuBar( IntPtr hWnd );

[DllImport("user32.dll")]

static extern bool RemoveMenu( IntPtr hMenu, uint uPosition, uint uFlags );

[DllImport("user32.dll")]

static extern uint GetMenuItemCount( IntPtr hMenu );

[DllImport("user32.dll", SetLastError = true)]

static extern IntPtr FindWindow( string lpszClass, string lpszWindow );

[DllImport("user32.dll")]

static extern IntPtr GetSystemMenu( IntPtr hWnd, bool bRevert );

[DllImport("user32")]

static extern bool GetKeyboardLayoutName( StringBuilder str );

[DllImport("user32.dll")]

static extern bool EnumWindows( EnumWindowsProc lpEnumFunc, IntPtr lParam );

[DllImport("user32.dll", SetLastError = true)]

static extern bool BringWindowToTop( IntPtr hWnd );

private delegate bool EnumWindowsProc( IntPtr hWnd, IntPtr lParam );

[DllImport("user32.dll", SetLastError = true, CharSet = CharSet.Auto)]

static extern int GetWindowThreadProcessId( IntPtr hWnd, out int id );

[DllImport("user32")]

static extern bool ExitWindowsEx( uint flag, uint reason );

[DllImport("user32.dll")]

static extern IntPtr SendMessage( IntPtr hWnd, uint Msg, int wParam, int lParam );

private bool EnumWindowsProcCallBack( IntPtr hWnd, IntPtr lParam )

{

StringBuilder str = new StringBuilder( );

int size;

GetWindowThreadProcessId(hWnd, out size);

txtConsole.Text += "PID" + Convert.ToString(size);

txtConsole.Text += Environment.NewLine;

return true;

}

/// <summary>

/// Обработчик клавиши определения системного времени

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnSysTime\_Click( object sender, RoutedEventArgs e )

{

SystemTime time = new SystemTime( );

GetSystemTime(time);

txtHour.Text = Convert.ToString(time.hour);

txtMinutes.Text = Convert.ToString(time.minute);

}

/// <summary>

/// Обработчик клавиши определение локального времени

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnLocalTime\_Click( object sender, RoutedEventArgs e )

{

SystemTime time = new SystemTime( );

GetLocalTime(time);

txtHour.Text = Convert.ToString(time.hour);

txtMinutes.Text = Convert.ToString(time.minute);

}

/// <summary>

/// Определить размеры окна

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnDefineSize\_Click( object sender, RoutedEventArgs e )

{

const Int32 SPI\_GETWORKAREA = 48;

RECT rect = new RECT( );

bool result = SystemParametersInfo(SPI\_GETWORKAREA,

0,

ref rect,

0);

if(result)

{

txtWidth.Text = Convert.ToString(rect.Right - rect.Left);

txtHeigth.Text = Convert.ToString(rect.Bottom - rect.Top);

}

}

/// <summary>

/// Определение серийный номер диска

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnDefineSerialNumber\_Click( object sender, RoutedEventArgs e )

{

StringBuilder buffer = new StringBuilder( );

uint serialNumber;

uint complength;

uint flag;

int \_size = 200;

StringBuilder fileBuffer = new StringBuilder( );

int size = 256;

if (GetVolumeInformation(null, buffer, \_size, out serialNumber, out complength, out flag, fileBuffer, size))

{

txtSerialNumber.Text = Convert.ToString(serialNumber);

}

}

/// <summary>

/// Shell

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnOpen\_Click( object sender, RoutedEventArgs e )

{

string path = txtPath.Text;

if (path != "" )

{

try

{

int result = (int)ShellExecute((IntPtr)0, "explore", path, null, null, 5);

if (result < 32)

{

MessageBox.Show("Не корректные параметры");

}

}

catch (Exception ex)

{

MessageBox.Show( ex.Message);

}

}

else

{

MessageBox.Show("Введите путь");

}

}

/// <summary>

/// Получение списка активных процессов в Windows

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnGetPid\_Click( object sender, RoutedEventArgs e )

{

if (EnumWindows(EnumWindowsProcCallBack, (IntPtr)0))

{

MessageBox.Show("Список процессов получен");

}

else

{

MessageBox.Show("Не удачное завершение получение PID");

}

}

/// <summary>

/// Активирование процесса

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnActivate\_Click( object sender, RoutedEventArgs e )

{

if (txtPid.Text != "")

{

try

{

int id = int.Parse(txtPid.Text);

BringWindowToTop((IntPtr)id);

MessageBox.Show( "Окно активировано");

}

catch (Exception ex)

{

MessageBox.Show( ex.Message);

}

}

else

{

MessageBox.Show("Введите PID");

}

}

/// <summary>

/// Перезагрузка PC

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnRestart\_Click( object sender, RoutedEventArgs e )

{

UInt32 reason = 0x00002000;

UInt32 command = 0x00000040;

try

{

bool result = ExitWindowsEx(command, reason);

}

catch (Exception ex)

{

MessageBox.Show( ex.Message);

}

}

/// <summary>

/// Определение раскладки клавиатуры

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void MainGrid\_KeyDown( object sender, System.Windows.Input.KeyEventArgs e )

{

StringBuilder str = new StringBuilder( );

if (GetKeyboardLayoutName(str))

{

int result = int.Parse(str.ToString( ));

if (result == 409)

{

txtLang.Text = "ENG";

}

else

{

txtLang.Text = "RUS";

}

}

if (e.Key != Key.Up)

{

mouse\_coord\_y += 5;

mouse\_event(MOUSEEVENTF\_MOVE, mouse\_coord\_x, mouse\_coord\_y, 0, 0);

}

if (e.Key == Key.Down)

{

mouse\_coord\_y -= 5;

mouse\_event(MOUSEEVENTF\_MOVE, mouse\_coord\_x, mouse\_coord\_y, 0, 0);

}

if (e.Key == Key.Left)

{

mouse\_coord\_x -= 5;

mouse\_event(MOUSEEVENTF\_MOVE, mouse\_coord\_x, mouse\_coord\_y, 0, 0);

}

if (e.Key == Key.Right)

{

mouse\_coord\_y += 5;

mouse\_event(MOUSEEVENTF\_MOVE, mouse\_coord\_x, mouse\_coord\_y, 0, 0);

}

}

/// <summary>

/// Отключение меню окна

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void brnSwitchOff\_Click( object sender, RoutedEventArgs e )

{

IntPtr hWnd = FindWindow(null, txtClassName.Text);

UInt32 MF\_BYPOSITION = 0x00000400;

UInt32 MF\_DISABLED = 0x00000002;

if (hWnd == IntPtr.Zero)

{

MessageBox.Show("Окно не найдено");

return;

}

IntPtr hMenu = GetSystemMenu(hWnd, false);

uint n;

if (hMenu != IntPtr.Zero)

{

if ((n = GetMenuItemCount(hMenu)) > 0)

{

RemoveMenu(hMenu, n - 1, MF\_BYPOSITION | MF\_DISABLED);

RemoveMenu(hMenu, n - 2, MF\_BYPOSITION | MF\_DISABLED);

DrawMenuBar(hWnd);

MessageBox.Show("Команда выполнена");

}

}

else

{

MessageBox.Show("Команда не выполнена");

}

}

/// <summary>

/// Включение монитора

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void brnMonitorOn\_Click( object sender, RoutedEventArgs e )

{

IntPtr windowHandle = Process.GetCurrentProcess( ).MainWindowHandle;

const int SC\_MONITORPOWER = 0xF170;

const int WM\_STSCOMMAND = 0x0112;

const int MONITOR\_ON = -1;

SendMessage(windowHandle, WM\_STSCOMMAND, SC\_MONITORPOWER, MONITOR\_ON);

}

/// <summary>

/// Выключение монитора

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnMonitorOff\_Click( object sender, RoutedEventArgs e )

{

IntPtr windowHandle = Process.GetCurrentProcess( ).MainWindowHandle;

const int SC\_MONITORPOWER = 0xF170;

const int WM\_STSCOMMAND = 0x0112;

const int MONITOR\_OFF = 2;

SendMessage(windowHandle, WM\_STSCOMMAND, SC\_MONITORPOWER, MONITOR\_OFF);

}

/// <summary>

/// Закрытие окна другого пз

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void btnCloseWindow\_Click( object sender, RoutedEventArgs e )

{

IntPtr windowHandle = Process.GetCurrentProcess( ).MainWindowHandle;

const int WM\_CLOSE = 0x0010;

SendMessage(windowHandle, WM\_CLOSE,0 , 0);

}

private void btnGetOtherWinTime\_Click( object sender, RoutedEventArgs e )

{

IntPtr windowHandle = Process.GetCurrentProcess( ).MainWindowHandle;

const int WM\_TIMECHANGE = 0x001E;

SendMessage(windowHandle, WM\_TIMECHANGE, 0, 0);

}

}

}

StructLayout(LayoutKind.Sequential)]

public struct RECT

{

public Int32 Left { get; set; }

public Int32 Top { get; set; }

public Int32 Right{ get; set; }

public Int32 Bottom{ get; set; }

}

[StructLayout(LayoutKind.Sequential)]

public class SystemTime

{

public ushort year;

public ushort month;

public ushort weekday;

public ushort day;

public ushort hour;

public ushort minute;

public ushort second;

public ushort millisecond;

}