Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

**МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ**

Кафедра кібербезпеки та програмного забезпечення

**Звіт**

з виконаної лабораторної роботи № 2

Дисципліна: Системне програмування

на тему

«Механізм керування пам’яттю»

Виконав :

студент академічної групи КІ-15

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

Перевірив :

Викладач

КонстантиноваЛ.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

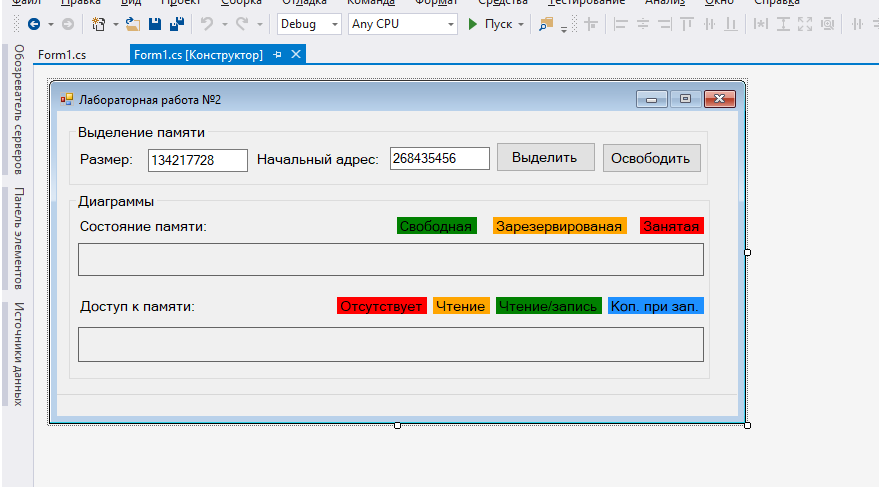
Кропивницкий- 2018

**Мета:** Отримати практичні навики використання функцій Win API.

**Завдання:**

Використовуючи наявну електронну документацію повністю повторити функціонал програм наведених в прикладах на мові С++ чи С#.

Тобто провести перетворення програм (трансляцію), представлених на мові програмування Object Pascal на мову С++ чи С#.





namespace Labarator\_2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

hProcess = GetCurrentProcess();

MemoryScan();

GetSystemInfo(ref sysinfo);

}

#region Константы

const int MEM\_COMMIT = 0x1000;

const int MEM\_FREE = 0x10000;

const int MEM\_RESERVE = 0x2000;

const int PAGE\_READWRITE = 0x04;

const int MEM\_RELEASE = 0x8000;

const int MEM\_DECOMMIT = 0x4000;

const int PAGE\_NOACCESS = 0x01;

const int PAGE\_READONLY = 0x02;

const int PAGE\_WRITECOPY = 0x08;

#endregion

#region Структуры

/// <summary>

/// Структура информации текущей системы

/// </summary>

public struct SYSTEM\_INFO

{

public ushort processorArchitecture;

ushort reserved;

public uint pageSize;

public IntPtr minimumApplicationAddress;

public IntPtr maximumApplicationAddress;

public IntPtr activeProcessorMask;

public uint numberOfProcessors;

public uint processorType;

public uint allocationGranularity;

public ushort processorLevel;

public ushort processorRevision;

}

public struct MEMORY\_BASIC\_INFORMATION

{

public IntPtr BaseAddress;

public IntPtr AllocationBase;

public int AllocationProtect;

public uint RegionSize;

public int State;

public int Protect;

public uint Type;

}

#endregion

#region Переменные

private IntPtr hProcess;

private IntPtr mem;

private SYSTEM\_INFO sysinfo = new SYSTEM\_INFO();

#endregion

#region Сигнатура методов

/// <summary>

///

/// </summary>

[DllImport("kernel32.dll")]

private static extern IntPtr GetCurrentProcess();

/// <summary>

///

/// </summary>

[DllImport("kernel32.dll")]

private static extern void GetSystemInfo(ref SYSTEM\_INFO lpSystemInfo);

/// <summary>

///

/// </summary>

[DllImport("kernel32.dll")]

private static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, int flAllocationType, int flProtect);

/// <summary>

///

/// </summary>

[DllImport("kernel32.dll")]

private static extern bool VirtualFreeEx(IntPtr hProcess, IntPtr lpAddress, int dwSize, int dwFreeType);

/// <summary>

///

/// </summary>

[DllImport("kernel32.dll")]

private static extern int VirtualQueryEx(IntPtr hProcess, IntPtr lpAddress, out MEMORY\_BASIC\_INFORMATION lpBuffer, int dwLength);

#endregion

#region Методу

/// <summary>

/// Сканирование памяти и отрисовка

/// </summary>

private void MemoryScan()

{

Bitmap bmp1 = new Bitmap(pictureBox1.Width, pictureBox1.Height);

Bitmap bmp2 = new Bitmap(pictureBox2.Width, pictureBox2.Height);

Graphics g1 = Graphics.FromImage(bmp1);

Graphics g2 = Graphics.FromImage(bmp2);

Pen Red = new Pen(Color.Red, 25);

Pen Green = new Pen(Color.Green, 25);

Pen Orange = new Pen(Color.Orange, 25);

Pen Blue = new Pen(Color.DodgerBlue, 25);

ulong address = 0, MaxAddress = 0x7FFFFFFF;

double scale = pictureBox1.Width / (double)(MaxAddress + 1);

int size = Marshal.SizeOf(typeof(MEMORY\_BASIC\_INFORMATION));

int result, start, end;

do

{

MEMORY\_BASIC\_INFORMATION m;

result = VirtualQueryEx(System.Diagnostics.Process.GetCurrentProcess().Handle, (IntPtr)address, out m, (int)Marshal.SizeOf(typeof(MEMORY\_BASIC\_INFORMATION)));

start = (int)((ulong)m.BaseAddress \* scale);

end = (int)(((ulong)m.BaseAddress + (ulong)m.RegionSize - 1) \* scale);

switch (m.State)

{

case MEM\_COMMIT:

g1.DrawRectangle(Red, start, 0, end, 25);

break;

case MEM\_FREE:

g1.DrawRectangle(Green, start, 0, end, 25);

break;

case MEM\_RESERVE:

g1.DrawRectangle(Orange, start, 0, end, 25);

break;

}

switch (m.Protect)

{

case PAGE\_NOACCESS:

g2.DrawRectangle(Red, start, 0, end, 25);

break;

case PAGE\_READONLY:

g2.DrawRectangle(Orange, start, 0, end, 25);

break;

case PAGE\_READWRITE:

g2.DrawRectangle(Green, start, 0, end, 25);

break;

case PAGE\_WRITECOPY:

g2.DrawRectangle(Blue, start, 0, end, 25);

break;

}

if (address == (ulong)((long)m.BaseAddress + (long)m.RegionSize))

break;

address = (ulong)((long)m.BaseAddress + (long)m.RegionSize);

pictureBox1.Image = bmp1;

pictureBox2.Image = bmp2;

} while (address <= MaxAddress);

}

#endregion

#region Обработчики

/// <summary>

/// Обработчик кнопку выделения памяти

/// </summary>

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

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

private void btnAllocate\_Click(object sender, EventArgs e)

{

try

{

UInt32 address = UInt32.Parse(txtAddress.Text);

UInt32 size = UInt32.Parse(txtSize.Text);

mem = VirtualAllocEx(hProcess, new IntPtr(address), size, MEM\_RESERVE | MEM\_COMMIT, PAGE\_READWRITE);

if (mem != IntPtr.Zero)

{

MessageBox.Show("Память выделена.");

txtAddress.Text = mem.ToString();

}

else

{

MessageBox.Show("Ошибка при выделении памяти.");

}

MemoryScan();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Обработчик кнопку освобождения памяти

/// </summary>

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

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

private void btnFree\_Click(object sender, EventArgs e)

{

try

{

UInt32 address = UInt32.Parse(txtAddress.Text);

UInt32 size = UInt32.Parse(txtSize.Text);

if (VirtualFreeEx(hProcess, new IntPtr(address), 0, MEM\_RELEASE))

{

MessageBox.Show("Память освобождена.");

}

else

{

MessageBox.Show("Ошибка при освобождении памяти.");

}

MemoryScan();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Обработчик потери фокуса мыши

/// </summary>

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

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

private void pictureBox\_MouseLeave(object sender, EventArgs e)

{

try

{

statusStrip1.Items[0].Text = "Размер страницы памяти: " + sysinfo.pageSize.ToString() + " байт";

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Обработчик перемещения мыши

/// </summary>

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

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

private void pictureBox\_MouseMove(object sender, MouseEventArgs e)

{

try

{

//вывод текущего адреса и информации в строке состояния

if (e.X < 0 || e.X > pictureBox1.Width)

return;

Color pixel = ((Bitmap)((PictureBox)sender).Image).GetPixel(e.X, 1);

if ((string)((PictureBox)sender).Tag == "status")

{ // если указатель мыши находится над pictureBox1

statusStrip1.Items[0].Text = "Состояние памяти по адресу " + (0x7fffffff / pictureBox1.Width \* e.X) + ": ";

switch (pixel.Name)

{

case "ff008000": statusStrip1.Items[0].Text += "свободная"; break;

case "ffffa500": statusStrip1.Items[0].Text += "зарезервированная"; break;

case "ffff0000": statusStrip1.Items[0].Text += "занятая"; break;

}

}

else

{ // pictureBox2

statusStrip1.Items[0].Text = "Доступ к памяти по адресу " + (0x7fffffff / pictureBox1.Width \* e.X) + ": ";

switch (pixel.Name)

{

case "ff008000": statusStrip1.Items[0].Text += "чтение/запись"; break;

case "ffffa500": statusStrip1.Items[0].Text += "чтение"; break;

case "ffff0000": statusStrip1.Items[0].Text += "отсутствует"; break;

case "ff1e90ff": statusStrip1.Items[0].Text += "копирование при записи"; break;

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Обработчик ввода размера памяти

/// </summary>

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

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

private void txtSizeKeyPress(object sender, KeyPressEventArgs e)

{

if ((e.KeyChar <= 47 || e.KeyChar >= 58) && e.KeyChar != 8)

{

e.Handled = true;

}

}

/// <summary>

/// Обработчик ввода адреса

/// </summary>

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

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

private void txtAddressKeyPress(object sender, KeyPressEventArgs e)

{

if ((e.KeyChar <= 47 || e.KeyChar >= 58) && e.KeyChar != 8)

{

e.Handled = true;

}

}

#endregion

}

}