Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

**МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ**

Кафедра кібербезпеки та програмного забезпечення

**Звіт**

з виконаної лабораторної роботи № 3

Дисципліна: Системне програмування

на тему

«Взаємодія між процесами IPC (Частина 1)»

Виконав :

студент академічної групи КІ-15

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

Перевірив :

Викладач

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

Кропивницкий- 2018

**Мета:** Отримати практичні навики в використанні функцій Win API. Навчитися організовувати коректну взаємодію між процесами застосовуючи механізми:

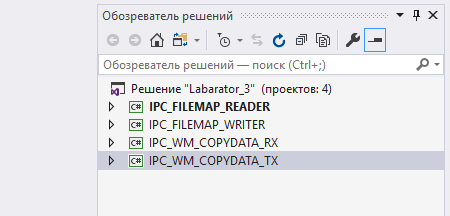
– Повідомлення WM\_COPYDATA;

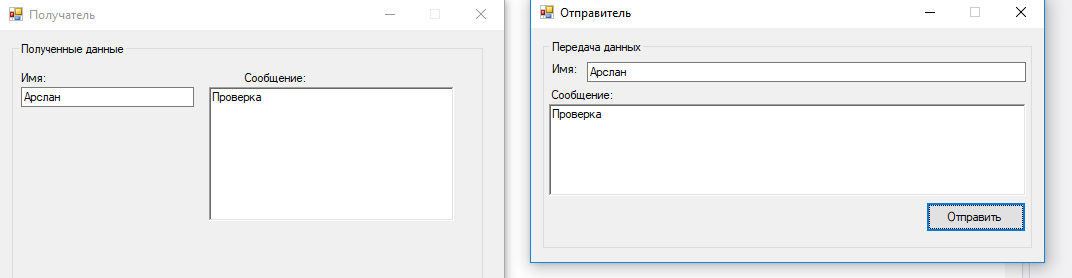
– Відображення файлу у пам’ять.

**Завдання:**

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

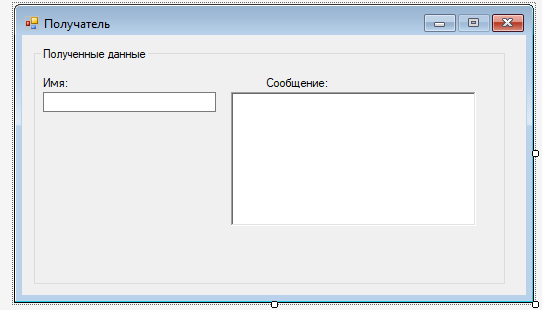
Тобто провести перетворення програм (трансляцію), представлених на мові програмування Object Pascal на мову С++ чи С#.





namespace Labarator\_3

{

 public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

#region Константы

const int WM\_COPYDATA = 0x004A;

#endregion

#region Структура

/// <summary>

/// Содержит данные, которые должны быть переданы другому приложению сообщением

/// </summary>

struct COPYDATASTRUCT

{

public IntPtr dwData;

public int cbData;

public IntPtr lpData;

}

#endregion

#region Обработчик

/// <summary>

/// Обработчик окна WinApi

/// </summary>

/// <param name="m"></param>

protected override void WndProc(ref Message m)

{

if (m.Msg == WM\_COPYDATA)

{

var copydata = (COPYDATASTRUCT)Marshal.PtrToStructure(m.LParam, typeof(COPYDATASTRUCT));

string msg = Marshal.PtrToStringAnsi(copydata.lpData, copydata.cbData);

textBox1.Text = msg.Split('|')[0];

richTextBox1.Text = msg.Split('|')[1];

}

base.WndProc(ref m);

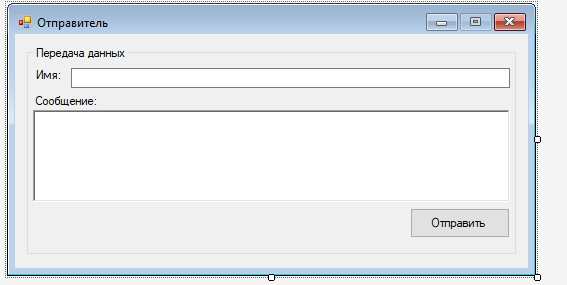
}

#endregion

}

}

namespace Labarator\_3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

#region Константы

const int WM\_COPYDATA = 0x004A;

#endregion

#region Сигнатура методов

/// <summary>

/// Найти дескриптор окна по имени

/// </summary>

[DllImport("user32.dll")]

static extern IntPtr FindWindow(string lpClassName, string lpWindowName);

/// <summary>

/// Передача сообщения

/// </summary>

[DllImport("user32.dll")]

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

#endregion

#region Структура

/// <summary>

/// Содержит данные, которые должны быть переданы другому приложению сообщением

/// </summary>

struct COPYDATASTRUCT

{

public IntPtr dwData;

public int cbData;

public IntPtr lpData;

}

#endregion

#region Обработчик

/// <summary>

/// Обработчик нажатия кнопки отпраки сообщения

/// </summary>

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

{

try

{

IntPtr hWnd = FindWindow(null, "Получатель");

if (hWnd != IntPtr.Zero)

{

string msg = new string((txtName.Text + "|" + rtxtChat.Text).ToCharArray());

IntPtr hMsg = Marshal.StringToHGlobalAnsi(msg);

COPYDATASTRUCT copydata = new COPYDATASTRUCT();

copydata.dwData = IntPtr.Zero;

copydata.cbData = msg.Length \* sizeof(char) + 1;

copydata.lpData = hMsg;

IntPtr hStruct = Marshal.AllocHGlobal(Marshal.SizeOf(copydata));

Marshal.StructureToPtr(copydata, hStruct, true);

SendMessage(hWnd, WM\_COPYDATA, IntPtr.Zero, hStruct);

Marshal.FreeHGlobal(hStruct);

Marshal.FreeHGlobal(hMsg);

}

else MessageBox.Show("Получатель не найден.");

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

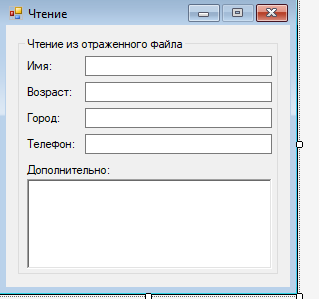
}

#endregion

}

}

namespace Labarator\_3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

#region Переменные

private IntPtr hMMF = IntPtr.Zero;

private FileStream fs;

public uint AllocationGranularity;

#endregion

#region Сигнатура методов

/// <summary>

/// Создает или открывает именованный или неназванный объект сопоставления файлов

/// </summary>

[DllImport("Kernel32.dll")]

private static extern IntPtr CreateFileMapping(IntPtr hFile, IntPtr lpAttributes, FileMapProtection flProtect, Int32 dwMaxSizeHi, Int32 dwMaxSizeLow, string lpName);

/// <summary>

/// Открывает именованный объект сопоставления файлов.

/// </summary>

[DllImport("Kernel32.dll")]

private static extern IntPtr OpenFileMapping(FileMapAccess DesiredAccess, bool bInheritHandle, string lpName);

/// <summary>

/// Отображает вид сопоставления файлов в адресное пространство вызывающего процесса.

/// </summary>

[DllImport("Kernel32.dll")]

private static extern IntPtr MapViewOfFile(IntPtr hFileMapping, FileMapAccess dwDesiredAccess, Int32 dwFileOffsetHigh, Int32 dwFileOffsetLow, Int32 dwNumberOfBytesToMap);

/// <summary>

/// Отключает сопоставленный вид файла из адресного пространства вызывающего

/// </summary>

[DllImport("Kernel32.dll")]

private static extern bool UnmapViewOfFile(IntPtr lpBaseAddress);

/// <summary>

/// Записывает на диск диапазон байтов внутри отображаемого вида файла.

/// </summary>

[DllImport("Kernel32.dll")]

private static extern bool FlushViewOfFile(IntPtr lpBaseAddress, Int32 dwNumberOfBytesToFlush);

/// <summary>

/// Получает информацию о текущей системе.

/// </summary>

[DllImport("Kernel32.dll")]

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

#endregion

#region Структура и Перечисления

/// <summary>

/// Перечисление защиты

/// </summary>

public enum FileMapProtection : uint

{

PageReadonly = 0x02,

PageReadWrite = 0x04,

PageWriteCopy = 0x08,

PageExecuteRead = 0x20,

PageExecuteReadWrite = 0x40,

SectionCommit = 0x8000000,

SectionImage = 0x1000000,

SectionNoCache = 0x10000000,

SectionReserve = 0x4000000,

}

/// <summary>

/// Перечисление прав доступа

/// </summary>

public enum FileMapAccess : uint

{

FileMapCopy = 0x0001,

FileMapWrite = 0x0002,

FileMapRead = 0x0004,

FileMapAllAccess = 0x001f,

fileMapExecute = 0x0020,

}

/// <summary>

/// Информация о текущей системе

/// </summary>

public struct SYSTEM\_INFO

{

internal \_PROCESSOR\_INFO\_UNION uProcessorInfo;

public uint dwPageSize;

public IntPtr lpMinimumApplicationAddress;

public IntPtr lpMaximumApplicationAddress;

public IntPtr dwActiveProcessorMask;

public uint dwNumberOfProcessors;

public uint dwProcessorType;

public uint dwAllocationGranularity;

public ushort dwProcessorLevel;

public ushort dwProcessorRevision;

}

/// <summary>

///

/// </summary>

public struct \_PROCESSOR\_INFO\_UNION

{

internal uint dwOemId;

internal ushort wProcessorArchitecture;

internal ushort wReserved;

}

#endregion

#region Методы

/// <summary>

/// Создает или открывает именованный или неназванный объект сопоставления файлов для указанного файла

/// </summary>

public static IntPtr CreateFileMapping(System.IO.FileStream File, FileMapProtection flProtect, Int64 ddMaxSize, string lpName)

{

int Hi = (Int32)(ddMaxSize / Int32.MaxValue);

int Lo = (Int32)(ddMaxSize % Int32.MaxValue);

return CreateFileMapping(File.SafeFileHandle.DangerousGetHandle(), IntPtr.Zero, flProtect, Hi, Lo, lpName);

}

/// <summary>

/// Отображает вид сопоставления файлов в адресное пространство вызывающего процесса.

/// </summary>

public static IntPtr MapViewOfFile(IntPtr hFileMapping, FileMapAccess dwDesiredAccess, Int64 ddFileOffset, Int32 dwNumberOfBytesToMap)

{

int Hi = (Int32)(ddFileOffset / Int32.MaxValue);

int Lo = (Int32)(ddFileOffset % Int32.MaxValue);

return MapViewOfFile(hFileMapping, dwDesiredAccess, Hi, Lo, dwNumberOfBytesToMap);

}

/// <summary>

///

/// </summary>

public void CMemoryMappedFile(string FileName, string Name)

{ // создание или открытие отраженного файла

hMMF = OpenFileMapping(FileMapAccess.FileMapAllAccess, false, Name);

if (hMMF == IntPtr.Zero)

{

fs = File.Open(FileName, FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite);

hMMF = CreateFileMapping(fs, FileMapProtection.PageReadWrite, Int64.MaxValue, Name);

if (hMMF == IntPtr.Zero)

{

MessageBox.Show("Не удалось отразить в память файл.");

return;

}

}

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

GetSystemInfo(ref sysinfo);

AllocationGranularity = sysinfo.dwAllocationGranularity;

}

unsafe public int Read(byte[] Buffer, int BytesToRead, Int64 AtOffset)

{ // чтение из отраженного файла

IntPtr hMVF = IntPtr.Zero;

try

{

Int64 FileMapStart = (AtOffset / AllocationGranularity) \* AllocationGranularity;

Int64 MapViewSize = (AtOffset % AllocationGranularity) + AllocationGranularity;

Int64 iViewDelta = AtOffset - FileMapStart;

hMVF = MapViewOfFile(hMMF, FileMapAccess.FileMapRead, FileMapStart, (Int32)MapViewSize);

if (hMVF == IntPtr.Zero)

throw new Win32Exception();

byte\* p = (byte\*)hMVF.ToPointer() + iViewDelta;

UnmanagedMemoryStream ums = new UnmanagedMemoryStream(p, MapViewSize, MapViewSize, FileAccess.Read);

byte[] ba = new byte[BytesToRead];

return ums.Read(Buffer, 0, BytesToRead);

}

finally

{

if (hMVF != IntPtr.Zero)

UnmapViewOfFile(hMVF);

}

}

#endregion

#region Обработчик

/// <summary>

/// Обработчик таймера - раз в секунду чтение файла

/// </summary>

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

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

private void timer1\_Tick(object sender, EventArgs e)

{

CMemoryMappedFile("exchange.buf", "lab3");

byte[] buff = new byte[1024];

Read(buff, 1024, 0);

string msg = Encoding.Default.GetString(buff);

if(msg != null)

{

try

{

txtName.Text = msg.Split('|')[0];

txtAge.Text = msg.Split('|')[1];

txtCity.Text = msg.Split('|')[2];

txtPhone.Text = msg.Split('|')[3];

txtAuxialiry.Text = msg.Split('|')[4];

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

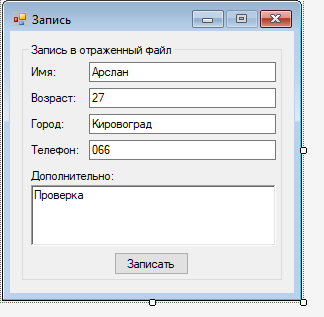
}

}

#endregion

}

}

namespace Labarator\_3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

#region Переменные

private IntPtr hMMF = IntPtr.Zero;

private FileStream fs;

private uint AllocationGranularity;

#endregion

#region Сигнатура методов

/// <summary>

/// Создает или открывает именованный или неназванный объект сопоставления файлов для указанного файла

/// </summary>

[DllImport("Kernel32.dll")]

private static extern IntPtr CreateFileMapping(IntPtr hFile, IntPtr lpAttributes, FileMapProtection flProtect, Int32 dwMaxSizeHi, Int32 dwMaxSizeLow, string lpName);

/// <summary>

/// Открывает именованный объект сопоставления файлов.

/// </summary>

[DllImport("Kernel32.dll")]

private static extern IntPtr OpenFileMapping(FileMapAccess DesiredAccess, bool bInheritHandle, string lpName);

/// <summary>

/// Отображает вид сопоставления файлов в адресное пространство вызывающего процесса.

/// </summary>

[DllImport("Kernel32.dll")]

private static extern IntPtr MapViewOfFile(IntPtr hFileMapping, FileMapAccess dwDesiredAccess, Int32 dwFileOffsetHigh, Int32 dwFileOffsetLow, Int32 dwNumberOfBytesToMap);

/// <summary>

/// Отключает сопоставленный вид файла из адресного пространства вызывающего процесса.

/// </summary>

[DllImport("Kernel32.dll")]

private static extern bool UnmapViewOfFile(IntPtr lpBaseAddress);

/// <summary>

/// Записывает на диск диапазон байтов внутри отображаемого вида файла.

/// </summary>

[DllImport("Kernel32.dll")]

private static extern bool FlushViewOfFile(IntPtr lpBaseAddress, Int32 dwNumberOfBytesToFlush);

/// <summary>

/// Получает информацию о текущей системе.

/// </summary>

[DllImport("Kernel32.dll")]

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

#endregion

#region Структура

/// <summary>

///

/// </summary>

public enum FileMapProtection : uint

{

PageReadonly = 0x02,

PageReadWrite = 0x04,

PageWriteCopy = 0x08,

PageExecuteRead = 0x20,

PageExecuteReadWrite = 0x40,

SectionCommit = 0x8000000,

SectionImage = 0x1000000,

SectionNoCache = 0x10000000,

SectionReserve = 0x4000000,

}

/// <summary>

///

/// </summary>

public enum FileMapAccess : uint

{

FileMapCopy = 0x0001,

FileMapWrite = 0x0002,

FileMapRead = 0x0004,

FileMapAllAccess = 0x001f,

fileMapExecute = 0x0020,

}

/// <summary>

///

/// </summary>

public struct SYSTEM\_INFO

{

internal \_PROCESSOR\_INFO\_UNION uProcessorInfo;

public uint dwPageSize;

public IntPtr lpMinimumApplicationAddress;

public IntPtr lpMaximumApplicationAddress;

public IntPtr dwActiveProcessorMask;

public uint dwNumberOfProcessors;

public uint dwProcessorType;

public uint dwAllocationGranularity;

public ushort dwProcessorLevel;

public ushort dwProcessorRevision;

}

/// <summary>

///

/// </summary>

public struct \_PROCESSOR\_INFO\_UNION

{

internal uint dwOemId;

internal ushort wProcessorArchitecture;

internal ushort wReserved;

}

#endregion

#region Методы

/// <summary>

///

/// </summary>

public static IntPtr CreateFileMapping(System.IO.FileStream File, FileMapProtection flProtect, Int64 ddMaxSize, string lpName)

{

int Hi = (Int32)(ddMaxSize / Int32.MaxValue);

int Lo = (Int32)(ddMaxSize % Int32.MaxValue);

return CreateFileMapping(File.SafeFileHandle.DangerousGetHandle(), IntPtr.Zero, flProtect, Hi, Lo, lpName);

}

/// <summary>

///

/// </summary>

public static IntPtr MapViewOfFile(IntPtr hFileMapping, FileMapAccess dwDesiredAccess, Int64 ddFileOffset, Int32 dwNumberOfBytesToMap)

{

int Hi = (Int32)(ddFileOffset / Int32.MaxValue);

int Lo = (Int32)(ddFileOffset % Int32.MaxValue);

return MapViewOfFile(hFileMapping, dwDesiredAccess, Hi, Lo, dwNumberOfBytesToMap);

}

/// <summary>

/// открытие или создание отраженного файла

/// </summary>

public void CMemoryMappedFile(string FileName, string Name)

{

hMMF = OpenFileMapping(FileMapAccess.FileMapAllAccess, false, Name);

if (hMMF == IntPtr.Zero)

{

fs = File.Open(FileName, FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite);

hMMF = CreateFileMapping(fs, FileMapProtection.PageReadWrite, Int64.MaxValue, Name);

if (hMMF == IntPtr.Zero)

{

MessageBox.Show("Не удалось отразить в память файл.");

return;

}

}

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

GetSystemInfo(ref sysinfo);

AllocationGranularity = sysinfo.dwAllocationGranularity;

}

/// <summary>

/// Запись

/// </summary>

unsafe public void Write(byte[] Buffer, int BytesToWrite, Int64 AtOffset)

{ // запись в отраженный файл

IntPtr hMVF = IntPtr.Zero;

try

{

Int64 FileMapStart = (AtOffset / AllocationGranularity) \* AllocationGranularity;

Int64 MapViewSize = (AtOffset % AllocationGranularity) + AllocationGranularity;

Int64 iViewDelta = AtOffset - FileMapStart;

hMVF = MapViewOfFile(hMMF, FileMapAccess.FileMapWrite, FileMapStart, (Int32)MapViewSize);

if (hMVF == IntPtr.Zero) throw new Win32Exception();

byte\* p = (byte\*)hMVF.ToPointer() + iViewDelta;

UnmanagedMemoryStream ums = new UnmanagedMemoryStream(p, MapViewSize, MapViewSize, FileAccess.Write);

ums.Write(Buffer, 0, BytesToWrite);

FlushViewOfFile(hMVF, (Int32)MapViewSize);

}

finally

{

if (hMVF != IntPtr.Zero) UnmapViewOfFile(hMVF);

}

}

#endregion

#region Обработчик

/// <summary>

/// Обработчик нажатия кнопки Отправить

/// </summary>

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

{ // запись в отраженный файл

CMemoryMappedFile("exchange.buf", "lab3");

string msg = txtName.Text + "|" + txtAge.Text + "|" + txtCity.Text

+ "|" + txtPhone.Text + "|" + txtAuxiliary.Text;

Write(Encoding.Default.GetBytes(msg), msg.Length, 0);

}

#endregion

}

}