**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ **«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. В. Г. ШУХОВА»**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Операционные системы**

**Тема: Архитектура памяти Windows**

Выполнил: ст. группы ВТ-31

Подкопаев Антон Валерьевич

Проверил: ст. пр. ПОВТиАС

Михелев Владимир Михайлович

**Белгород 2019**

**Цель работы:** Получение практических навыков по использованию Win32 API для исследования памяти Windows.

**Типы памяти**

На рисунке ниже представлена взаимосвязь виртуального адресного пространства процесса с физической и внешней памятью.

Виртуальное адресное пространство процесса

Виртуальная страница 1

Виртуальная страница 2

Виртуальная страница 3

Виртуальная страница 4

Физическая страница А

Физическая страница В

Файл

подкачки

EXE или DLL

(загрузочный

модуль)

Файл

данных

Физическая память

Диск

4 Кб

4 Кб

4 Кб

4 Кб

**Физическая память**

Физическая память (*physical memory*) - это реальные микросхемы *RAM*, установленные в компьютере. Каждый байт физической памяти имеет физический адрес (*physical address*), который представляет собой число от нуля до числа на единицу меньшего, чем количество байтов физической памяти. Например, ПК с установлен­ными 64 Мб RAM, имеет физические адреса &Н00000000-&Н04000000 в шестнадцатеричной системе счисления, что в десятичной системе будет 0-67 108 863.

Физическая память (в отличие от файла подкачки и виртуальной памяти) является исполняемой *(executable*), то есть памятью, из которой можно читать и в которую центральный процессор может посредством системы команд записывать данные.

**Виртуальная память**

Виртуальная память (*virtual memory*) - это просто набор чисел, о которых говорят как о виртуальных адресах. Программист может использовать виртуаль­ные адреса, но Windows не способна по этим адресам непосредственно обращать­ся к данным, поскольку такой адрес не является адресом реального физического запоминающего устройства, как в случае физических адресов и адресов файла подкачки. Для того чтобы код с виртуальными адресами можно было выполнить, такие адреса должны быть отображены на физические адреса, по которым дей­ствительно могут храниться коды и данные. Эту операцию выполняет диспетчер виртуальной памяти (*Virtual Memory Manager - VMM*). Операционная система Windows обозначает некоторые области виртуальной памяти как области, к которым можно обратиться из программ пользовательского режима. Все остальные области указываются как зарезервированные. Какие области памяти доступны, а какие зарезервированы, зависит от версии операционной системы (Windows 9x или Windows NT).

**Страничные блоки памяти**

Как известно, наименьший адресуемый блок памяти - байт. Однако самым маленьким блоком памяти, которым оперирует Windows VMM, является страница *(page*) памяти, называемая также страничным блоком (*page frame*) памяти. На компьютерах с процессорами Intel объем страничного блока равен 4 Кб.

**Память файла подкачки**

Страничный файл (*page file*), который называется также файлом подкачки (*swap file*) в Windows находится на жестком диске. Он используется для хранения данных и программ точно так же, как и физическая память, но его объем обычно превышает объем физической памяти. Windows использует файл подкач­ки (или файлы, их может быть несколько) для хранения информации, которая не помещается в RAM, производя, если нужно, обмен страниц между файлом подкачки и RAM.

Таким образом, диапазон виртуальных адресов скорее согласуется с адресами в файле подкачки, чем с адресами физической памяти. Когда такое согласование достигается, говорят, что виртуальные адреса спроецированы (*backed*) на файл подкачки, или являются проецируемыми на файл подкачки (*pagefile-backed*).

Набор виртуальных адресов может проецироваться на физическую намять, файл подкачки или любой файл.

**Файлы, отображаемые в память**

Многим объектам ядра при их создании может быть присвоено имя. Облас­тью действия имени является вся система. Это означает, что любой другой про­цесс может получить доступ к объекту по его имени (если считать, конечно, что другому процессу это имя известно). Например, последний параметр функции

HAHDLE CreateFileMapping(

HANDLE hFile, // Дескриптор отображаемого файла.

LPSECUKITY\_ATTRIBUTES IpFileMappingAttributes,

// Необязательные атрибуты защиты.

DWORD flProtect, // Защита отображаемого объекта.

DWORD dwMaxiniumSizeHigh, // Старшие 32 бита размера объекта.

DWORD dwMaximumSizeLow, // Младшие 32 бита размера объекта.

LPCTSTR IpHame // Имя объекта отображения файла.

) ;

может использоваться для задания имени отображения файла.

Предположим, создан объект File-mapping с именем MyFMO. Другой про­цесс может вызнать функцию OpenFileMapping с этим именем в качестве пос­леднего аргумента. Функция вернет зависимый от процесса дескриптор объекта для использования во втором процессе. В виде альтернативы второй процесс мо­жет вызвать функцию GreateFileMapping, используя имя объекта в качестве ее последнего аргумента. Система определит, что объект File-mapping с таким име­нем уже существует и просто вернет его дескриптор. Здесь могут возникнуть про­блемы, поскольку процесс считает, что он создает новый объект, тогда как в действительности он получает дескриптор существующего объекта. Программист должен сразу проверить возвращенное функцией CreateFileMapping значение, чтобы правильно сориентироваться в ситуации.

**Совместно используемая физическая память**

О физической памяти говорят, что она совместно используется (shared), если она отображается на виртуальное адресное пространство нескольких процессов, хотя виртуальные адреса в каждом процессе могут отличаться. Следующим рисунок иллюстрирует это утверждение.

Если файл, такой как DLL, находится в совместно используемой физической памяти, то о нем можно говорить как о совместно используемом.

Одно из преимуществ файлов, отображаемых в память, заключается в том, что их легко использовать совместно. Присвоение имени объекту «отображение файла» делает возможным совместное использование файла несколькими процессами. В этом случае его содержимое отображено на совместно используемую физическую память (см. рис. ниже). Возможно также совместное пользование содержимого файла подкачки с помощью механизма отображения файла.

Физическая память

Совместно используемая физическая память

Виртуальное адресное пространство процесса 1

Виртуальное адресное пространство процесса 2

Физическая память

Совместно используемая физическая память

Файл подкачки загрузочный модуль или файл данных

Виртуальное адресное пространство процесса 1

Виртуальное адресное пространство процесса 2

MainWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using static WpfApp17.winapi;

namespace WpfApp17

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public static class BinaryExt

{

public static string ToBinary(this int number, int bitsLength = 32)

{

return NumberToBinary(number, bitsLength);

}

public static string NumberToBinary(int number, int bitsLength = 32)

{

string result = Convert.ToString(number, 2).PadLeft(bitsLength, '0');

return result;

}

public static int FromBinaryToInt(this string binary)

{

return BinaryToInt(binary);

}

public static int BinaryToInt(string binary)

{

return Convert.ToInt32(binary, 2);

}

}

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

SYSTEM\_INFO iNFO;

GetSystemInfo(out iNFO);

infolabel1.Content = iNFO.ProcessorArchitecture;

infolabel2.Content = iNFO.PageSize;

infolabel3.Content = iNFO.MinimumApplicationAddress.ToString("x") + "h";

infolabel4.Content = iNFO.MaximumApplicationAddress.ToString("x") + "h";

infolabel5.Content = BinaryExt.ToBinary(iNFO.ActiveProcessorMask.ToInt32()).ToString();

infolabel6.Content = iNFO.NumberOfProcessors;

infolabel7.Content = iNFO.ProcessorType;

infolabel8.Content = iNFO.AllocationGranularity;

infolabel9.Content = iNFO.ProcessorLevel;

infolabel10.Content = iNFO.ProcessorRevision;

task2();

//chartvalue = 0;

}

int chartvalue = 0;

public async void task2()

{

await Task.Run(() =>

{

while (true)

{

MEMORYSTATUSEX mEMORYSTATUSEX = new MEMORYSTATUSEX();

GlobalMemoryStatusEx(mEMORYSTATUSEX);

chartvalue += 10;

Dispatcher.Invoke(() =>

{

textBlock\_Copy19.Text = "TotalPhys: " + mEMORYSTATUSEX.TotalPhys.ToString();

textBlock\_Copy20.Text = "AvailPhys: " + mEMORYSTATUSEX.AvailPhys.ToString();

textBlock\_Copy21.Text = "TotalPageFile: " + mEMORYSTATUSEX.TotalPageFile.ToString();

textBlock\_Copy22.Text = "AvailPageFile: " + mEMORYSTATUSEX.AvailPageFile.ToString();

textBlock\_Copy23.Text = "TotalVirtual: " + mEMORYSTATUSEX.TotalVirtual.ToString();

textBlock\_Copy24.Text = "AvailVirtual: " + mEMORYSTATUSEX.AvailVirtual.ToString();

textBlock\_Copy25.Text = "AvailExtendedVirtual: " + mEMORYSTATUSEX.AvailExtendedVirtual;

Resources["GaugeValue1"] = Math.Round(Math.Abs((Double)mEMORYSTATUSEX.AvailPhys / mEMORYSTATUSEX.TotalPhys \* 100 - 100), 2);

Resources["GaugeValue2"] = Math.Round(Math.Abs((Double)mEMORYSTATUSEX.AvailVirtual / mEMORYSTATUSEX.TotalVirtual \* 100 - 100), 2);

Resources["GaugeValue3"] = Math.Round(Math.Abs((Double)mEMORYSTATUSEX.AvailPageFile / mEMORYSTATUSEX.TotalPageFile \* 100 - 100), 2);

Resources["GaugeValue4"] = Math.Round((double)mEMORYSTATUSEX.MemoryLoad, 2);

});

Thread.Sleep(1000);

}

});

}

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

{

//chartvalue += 5;

}

private void Window\_Initialized(object sender, EventArgs e)

{

}

public class mapelement

{

public string \_ModuleName;

public string \_BaseAddress;

public string \_EndAddress;

public string \_Size;

public string \_State;

public string \_Protection;

public string \_MemoryType;

public string ModuleName

{

get { return \_ModuleName; }

set { \_ModuleName = value; }

}

public string BaseAddress

{

get { return \_BaseAddress; }

set { \_BaseAddress = value; }

}

public string EndAddress

{

get { return \_EndAddress; }

set { \_EndAddress = value; }

}

public string Size

{

get { return \_Size; }

set { \_Size = value; }

}

public string State

{

get { return \_State; }

set { \_State = value; }

}

public string Protection

{

get { return \_Protection; }

set { \_Protection = value; }

}

public string MemoryType

{

get { return \_MemoryType; }

set { \_MemoryType = value; }

}

}

public ObservableCollection<mapelement> processmap { get; set; }

private void textBox\_TextChanged(object sender, TextChangedEventArgs e)

{

processmap = new ObservableCollection<mapelement>();

bool finded = false;

listblock.Items.Clear();

listblock.Items.Add(new ListBoxItem() { Content = String.Format("Начало страниц - конец страницы | размер | состояние | атрибут защиты | тип физической памяти") });

//try

//{

IntPtr hSnapshot = CreateToolhelp32Snapshot(SnapshotFlags.Process, 0);

if (hSnapshot.ToInt32() != 0)

{

PROCESSENTRY32 pe32 = new PROCESSENTRY32();

pe32.dwSize = (uint)Marshal.SizeOf(pe32);

if (Process32First(hSnapshot, ref pe32))

{

do

{

if (pe32.szExeFile.Contains(textBox.Text))

{

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

GetNativeSystemInfo(ref sysInfo);

IntPtr hProcess = OpenProcess(ProcessAccessFlags.All, false, (int)pe32.th32ProcessID);

if ((int)pe32.th32ProcessID != 0)

{

label2.Content = String.Format("(PID = {0})", (int)pe32.th32ProcessID);

}

else

{

label2.Content = "";

}

long MaxAddress = 0x7fffffff;

//long address = (long)System.Diagnostics.Process.GetProcessesByName("Adguard\_6.X\_Setup\_Online\_Ml\_Rus")[0].MainModule.BaseAddress;

//if ((long)System.Diagnostics.Process.GetProcessesByName("uTorrent.exe").Length == 0) break;

//long address = (long)System.Diagnostics.Process.GetProcessesByName("uTorrent.exe")[0].MainModule.BaseAddress;

long address = 0;

MEMORY\_BASIC\_INFORMATION m = new MEMORY\_BASIC\_INFORMATION();//GetCurrentProcess, System.Diagnostics.Process.GetProcessesByName("Adguard\_6.X\_Setup\_Online\_Ml\_Rus")[0].Handle

do

{

int result = VirtualQueryEx(hProcess, (IntPtr)address, out m, (uint)Marshal.SizeOf(m));//System.Diagnostics.Process.GetCurrentProcess().Handle

if (result == 0) break;

if ((uint)m.BaseAddress == 0x6a200000)

{

;

}

StringBuilder Name = new StringBuilder(255);

//GetModuleFileName(m.BaseAddress, Name, Name.Capacity);

GetModuleFileNameEx(hProcess, m.BaseAddress, Name, (uint)Name.Capacity);

//string convertedUtf8 = Encoding.UTF8.GetString(Encoding.Unicode.GetBytes(Name.ToString()));

string Statetext = "", ProtectText = "", MemoryTypeText = "";

//eltext += String.Format(" 0x{0:x8}-0x{1:x8} : {2:d8} ", (uint)m.BaseAddress, ((uint)m.BaseAddress + (uint)m.RegionSize - 1), (uint)m.RegionSize); uTorrent

if (Name.ToString().Length > 5 && Name.ToString().Contains("\\") && (uint)address != 0) listblock.Items.Add(new ListBoxItem() { Content = String.Format("({0})", Name.ToString().Split("\\".ToCharArray()).Last()) });

//eltext += " | ";

if (m.State == 0x1000) Statetext += "MEM\_COMMIT ";

if (m.State == 0x10000) Statetext += "MEM\_FREE ";

if (m.State == 0x2000) Statetext += "MEM\_RESERVE ";

if (m.Type == 0x1000000) MemoryTypeText += "MEM\_IMAGE ";

if (m.Type == 0x40000) MemoryTypeText += "MEM\_MAPPED ";

if (m.Type == 0x20000) MemoryTypeText += "MEM\_PRIVATE ";

//eltext += "[ Protect : ";

if (m.Protect == 0x10) ProtectText += "PAGE\_EXECUTE ";

if (m.Protect == 0x20) ProtectText += "PAGE\_EXECUTE\_READ ";

if (m.Protect == 0x40) ProtectText += "PAGE\_EXECUTE\_READWRITE ";

if (m.Protect == 0x80) ProtectText += "PAGE\_EXECUTE\_WRITECOPY ";

if (m.Protect == 0x01) ProtectText += "PAGE\_NOACCESS ";

if (m.Protect == 0x02) ProtectText += "PAGE\_READONLY ";

if (m.Protect == 0x04) ProtectText += "PAGE\_READWRITE ";

if (m.Protect == 0x08) ProtectText += "PAGE\_WRITECOPY ";

if (m.Protect == 0x40000000) ProtectText += "PAGE\_TARGETS\_INVALID ";

if (m.Protect == 0x40000000) ProtectText += "PAGE\_TARGETS\_NO\_UPDATE ";

if (m.Protect == 0x100) ProtectText += "PAGE\_GUARD ";

if (m.Protect == 0x200) ProtectText += "PAGE\_NOCACHE ";

if (m.Protect == 0x400) ProtectText += "PAGE\_WRITECOMBINE ";

//eltext += "]";

//string a = m.GetType().Name;

//listblock.Items.Add(new ListBoxItem() { Content = eltext });

string ModuleName = "";

if (Name.ToString().Length > 5 && Name.ToString().Contains("\\") && (uint)address != 0 && Name.ToString().Split("\\".ToCharArray()).Last().IndexOf(".")>1) ModuleName = String.Format("({0})", Name.ToString().Split("\\".ToCharArray()).Last());

processmap.Add(new mapelement() { ModuleName = ModuleName, BaseAddress = String.Format("0x{0:x8}", (uint)m.BaseAddress), EndAddress = String.Format("0x{0:x8}", ((uint)m.BaseAddress + (uint)m.RegionSize - 1)), Size = String.Format("{0:d8}", (uint)m.RegionSize), State = Statetext, Protection = ProtectText, MemoryType = MemoryTypeText });

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

//} while (address < (long)(System.Diagnostics.Process.GetProcessesByName("Adguard\_6.X\_Setup\_Online\_Ml\_Rus")[0].MainModule.BaseAddress.ToInt32()+sysInfo..ToInt32()));

} while (true);//(address < 0x7FFE7000);

GetLastError();

finded = true;

if (processmap != null)

{

dataGrid.ItemsSource = processmap;

}

else

{

dataGrid.Items.Clear();

}

break;

}

//ListBoxItem lbi = new ListBoxItem();

//lbi.Content = "pid : " + pe32.th32ProcessID + " | " + pe32.szExeFile;

//lbi.Tag = pe32.th32ProcessID;

//processeslst.Items.Add(lbi);

//printf("pid %d %s\n", pe32.th32ProcessID, pe32.szExeFile);

} while (Process32Next(hSnapshot, ref pe32));

}

//CloseHandle(hSnapshot);

}

//}

//catch (Exception)

//{

//throw;

//}

if (!finded) label2.Content = "";

}

}

}

winapi.cs

using System;

using System.Runtime.InteropServices;

using System.Text;

namespace WpfApp17

{

static class winapi

{

[Flags]

public enum ProcessAccessFlags : uint

{

All = 0x001F0FFF,

Terminate = 0x00000001,

CreateThread = 0x00000002,

VirtualMemoryOperation = 0x00000008,

VirtualMemoryRead = 0x00000010,

VirtualMemoryWrite = 0x00000020,

DuplicateHandle = 0x00000040,

CreateProcess = 0x000000080,

SetQuota = 0x00000100,

SetInformation = 0x00000200,

QueryInformation = 0x00000400,

QueryLimitedInformation = 0x00001000,

Synchronize = 0x00100000

}

[Flags]

public enum SnapshotFlags : uint

{

HeapList = 0x00000001,

Process = 0x00000002,

Thread = 0x00000004,

Module = 0x00000008,

Module32 = 0x00000010,

Inherit = 0x80000000,

All = 0x0000001F,

NoHeaps = 0x40000000

}

[StructLayout(LayoutKind.Sequential)]

public struct MEMORY\_BASIC\_INFORMATION

{

public IntPtr BaseAddress;

public IntPtr AllocationBase;

public uint AllocationProtect;

public IntPtr RegionSize;

public uint State;

public uint Protect;

public uint Type;

}

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]

public struct PROCESSENTRY32

{

const int MAX\_PATH = 260;

internal UInt32 dwSize;

internal UInt32 cntUsage;

internal UInt32 th32ProcessID;

internal IntPtr th32DefaultHeapID;

internal UInt32 th32ModuleID;

internal UInt32 cntThreads;

internal UInt32 th32ParentProcessID;

internal Int32 pcPriClassBase;

internal UInt32 dwFlags;

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = MAX\_PATH)]

internal string szExeFile;

}

public enum ProcessorArchitecture

{

X86 = 0,

Arm = 5,

Itanium = 6,

X64 = 9,

ARM64 = 12,

Unknown = 0xFFFF,

}

[StructLayout(LayoutKind.Sequential)]

public struct SYSTEM\_INFO

{

public ProcessorArchitecture ProcessorArchitecture; // WORD

public uint PageSize; // DWORD

public IntPtr MinimumApplicationAddress; // (long)void\*

public IntPtr MaximumApplicationAddress; // (long)void\*

public IntPtr ActiveProcessorMask; // DWORD\*

public uint NumberOfProcessors; // DWORD (WTF)

public uint ProcessorType; // DWORD

public uint AllocationGranularity; // DWORD

public ushort ProcessorLevel; // WORD

public ushort ProcessorRevision; // WORD

}

[DllImport("kernel32.dll", SetLastError = false)]

public static extern void GetSystemInfo(out SYSTEM\_INFO Info);

[StructLayout(LayoutKind.Sequential, CharSet = CharSet.Auto)]

public class MEMORYSTATUSEX

{

public uint Length;

public uint MemoryLoad;

public ulong TotalPhys;

public ulong AvailPhys;

public ulong TotalPageFile;

public ulong AvailPageFile;

public ulong TotalVirtual;

public ulong AvailVirtual;

public ulong AvailExtendedVirtual;

public MEMORYSTATUSEX()

{

this.Length = (uint)Marshal.SizeOf(typeof(MEMORYSTATUSEX));

}

}

[return: MarshalAs(UnmanagedType.Bool)]

[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError = true)]

public static extern bool GlobalMemoryStatusEx([In, Out] MEMORYSTATUSEX lpBuffer);

[DllImport("kernel32.dll")]

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

[DllImport("kernel32.dll", SetLastError = true)]

public static extern IntPtr OpenProcess(ProcessAccessFlags processAccess, bool bInheritHandle, int processId);

[DllImport("kernel32.dll", SetLastError = true)]

public static extern IntPtr CreateToolhelp32Snapshot(SnapshotFlags dwFlags, uint th32ProcessID);

[DllImport("kernel32", SetLastError = true, CharSet = CharSet.Auto)]

public static extern bool Process32First([In]IntPtr hSnapshot, ref PROCESSENTRY32 lppe);

[DllImport("kernel32", SetLastError = true, CharSet = CharSet.Auto)]

public static extern bool Process32Next([In]IntPtr hSnapshot, ref PROCESSENTRY32 lppe);

internal const ushort PROCESSOR\_ARCHITECTURE\_INTEL = 0;

internal const ushort PROCESSOR\_ARCHITECTURE\_IA64 = 6;

internal const ushort PROCESSOR\_ARCHITECTURE\_AMD64 = 9;

internal const ushort PROCESSOR\_ARCHITECTURE\_UNKNOWN = 0xFFFF;

[DllImport("kernel32.dll")]

public static extern void GetNativeSystemInfo(ref SYSTEM\_INFO lpSystemInfo);

[DllImport("kernel32.dll")]

public static extern uint GetLastError();

[DllImport("psapi.dll", CallingConvention = CallingConvention.StdCall, CharSet = CharSet.Unicode)]

public static extern uint GetModuleFileNameEx(IntPtr hProcess, IntPtr hModule, [Out] StringBuilder lpBaseName, uint nSize);

}

}

Изображение выглядит как внутренний, стена, монитор, снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана, монитор, внутренний, компьютер

Автоматически созданное описание