Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра «»

Лабораторная работа №2

по дисциплине «Операционные системы»

«Архитектура Windows»

Выполнил: студент гр. qwinmen

Проверил:.

Тамбов, 20

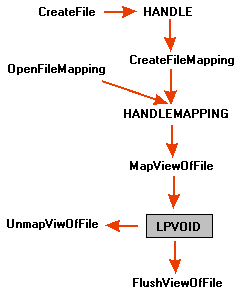
***Цели и задачи****.*

Получение практических навыков по программированию в Win32 API и изучение архитектуры операционной системы Windows.

***Решение задачи****.*

«Отображение файлов в память»

Самый главный вопрос - это зачем нужно отображать файлы на память? Ведь есть виртуальная память до 2 ГБ. Одна из причин это обеспечение работы с файлами большего размера. Второе преимущество при работе с большими файлами, что память не выделяется, а резервируется и при обращении к ней происходит подгрузка из файла. Этот механизм используется операционной системой, например, при работе с DLL. Общий набор функций по работе с отображаемыми файлами представлен на диаграмме.

Сначала файл открывается с помощью **CreateFile()**, и если открытие прошло успешно, то идентификатором файла можно воспользоваться для создания отображенного файла **CreateFileMapping()**, после которой мы получим идентификатор отраженного файла, на основе которого мы можем проецировать файл в память с помощью функции **MapViewOfFile()**. Эта функция даст нам указатель LPVOID, который используется для последующих операций, в том числе и по отмене проецированного файла **UnmapViewOfFile()** или принудительной записи на диск **FlushViewOfFile()**. В том случае, если с отображенным файлом будут работать несколько приложений одно приложение создает файл проецированный на память **CreateFileMaping()**, а остальные открывают эту проекцию **OpenFileMapping()**. 

С теорией разобрались, теперь практика. Для реализаций отображений средствами системных библиотек в C# сначала используется подгрузка DLL:

#region CreateFile

[DllImport("Kernel32.dll", SetLastError = true, CharSet = CharSet.Auto)]

public static extern SafeFileHandle CreateFile(string fileName,//имя файла

[MarshalAs(UnmanagedType.U4)] FileAccess fileAccess,//доступ r\w

[MarshalAs(UnmanagedType.U4)] FileShare fileShare,//общий доступ yes\no

IntPtr securityAttributes,//атрибуты безопасности файла

[MarshalAs(UnmanagedType.U4)] FileMode creationDisposition,//режим открыть\создать\перезаписать

[MarshalAs(UnmanagedType.U4)] FileAttributes flags,//атрибут архив\скрытый\дляЧтения

IntPtr template);//!= 0 копирует атрибуты с указаного шаблона файла

#endregion

#region GetFileSize

[DllImport("Kernel32.dll")]

public static extern uint GetFileSize(int hFile, int lpFileSizeHigh);

#endregion

#region CreateFileMapping

public enum FileMapProtection : uint

{

PageReadonly = 0x02,

PageReadWrite = 0x04,

PageWriteCopy = 0x08,

PageExecuteRead = 0x20,

PageExecuteReadWrite = 0x40,

SectionCommit = 0x8000000,

SectionImage = 0x1000000,

SectionNoCache = 0x10000000,

SectionReserve = 0x4000000,

}

[DllImport("Kernel32.dll", CharSet = CharSet.Auto)]

private static extern SafeFileHandle CreateFileMapping(

IntPtr hFile,//Дескриптор файла, из которого создается "проецируемый" объект

IntPtr lpFileMappingAttributes,//атрибут безопасности

FileMapProtection flProtect,//эт отображение доступно для r\w\rw\copy\

int dwMaximumSizeHigh,//размер для файлового отображения

int dwMaximumSizeLow,//размер для файлового отображения

[MarshalAs(UnmanagedType.LPStr)] string lpName);//имя обьекта файлового отображения

internal static SafeFileHandle CreateFileMapping(/\*FileStream\*/IntPtr File, FileMapProtection flProtect, Int64 ddMaxSize, string lpName)

{

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

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

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

}

#endregion

#region MapViewOfFile вернет указатель на отображенный файл, переданый в первом параметре

public enum FileMapAccess : uint

{

FileMapCopy = 0x0001,

FileMapWrite = 0x0002,

FileMapRead = 0x0004,

FileMapAllAccess = 0x001f,

fileMapExecute = 0x0020,

}

[DllImport("Kernel32", CharSet = CharSet.Auto, SetLastError = true)]

unsafe public static extern IntPtr/\*LPVOID\*/ MapViewOfFile(IntPtr hFileMapping, FileMapAccess dwDesiredAccess,

Int32 dwFileOffsetHigh, Int32 dwFileOffsetLow,

Int32 dwNumberOfBytesToMap);

#endregion

[DllImport("Kernel32.dll")]

unsafe public static extern void CopyMemory(void\* Куда, void\* Откуда, int length);

unsafe public static Char[] CloneCharArrayMemcpy(Char[] a)

{

Char[] ret = new Char[a.Length];

fixed (Char\* src = a, dest = ret)

{

CopyMemory(dest, src, a.Length\*sizeof (Char));

}

return (ret);

}

Далее, эти обьявления используем:

unsafe

{

#region 1) Создать текстовый файл и получить дескриптор

FileStream fs;

SafeFileHandle handle = clAPI.CreateFile("newFileATOM.txt", FileAccess.Write, 0,

IntPtr.Zero, FileMode.OpenOrCreate, 0, IntPtr.Zero);

if (handle.IsInvalid)

Marshal.ThrowExceptionForHR(Marshal.GetHRForLastWin32Error());

else fs = new FileStream(handle,FileAccess.ReadWrite);

int val = (int)handle.DangerousGetHandle();

txtBoxAdd("[Дескриптор newFileATOM.txt \*int] ", val.ToString());

#endregion

#region 2) Создасть объект File на базе созданного в предыдущем пункте файла, используя АРI-функцию CreateFile. Вывести значение дескриптора объекта File.

uint size = clAPI.GetFileSize(val, 0);

txtBoxAdd("[Размер newFileATOM.txt \*byte] ", size.ToString());

#endregion

#region 3) Используя дескриптор объекта File-mapping, а также API-функцию MapViewOfFile отобразить части файла в память. Данная функция назначает область виртуальной памяти, выделяемой этому файлу. Базовый адрес выделенной области памяти является дескриптором представления этой области в виде отображения файла.

fs = File.Open("nnewFileATOM.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite);

SafeFileHandle myFileMapping = clAPI.CreateFileMapping(fs.SafeFileHandle.DangerousGetHandle(),

clAPI.FileMapProtection.PageReadWrite,

(int) size, "FileMapping.bin");

if (myFileMapping.IsInvalid) Marshal.ThrowExceptionForHR(Marshal.GetHRForLastWin32Error());

txtBoxAdd("[Дескриптор FileMapping.bin \*int] ", myFileMapping.DangerousGetHandle().ToString());

IntPtr myFileMapView = clAPI.MapViewOfFile(myFileMapping.DangerousGetHandle(),

clAPI.FileMapAccess.FileMapWrite, 0, 0, 0);

if (myFileMapView.ToInt32() == 0) Marshal.ThrowExceptionForHR(Marshal.GetHRForLastWin32Error());

txtBoxAdd("[Указатель на отображенный FileMapping.bin \*int] ", myFileMapView.ToString());

#endregion

#region 4) Используя базовый адрес и функцию CopyMemory прочитайте ин-формацию из отображаемого файла. Согласно варианту измените текстовый файл и запишите информацию в этот же файл

txtBoxAdd("[ViewOfFile] ", "изменить регистр содержимого");

//копировать IntPtr в char[]

char[] managedArray2 = new char[size];

Marshal.Copy(myFileMapView, managedArray2, 0, (int)size);

fixed (char\* p = &managedArray2[0])

{

clAPI.CopyMemory((IntPtr)p, myFileMapView, size);

for (int i = 0; i < size - 1; i++)

{

char c = p[i];

if (Char.IsUpper(c))

p[i] = Char.ToLower(c);

else if (Char.IsLower(c))

p[i] = Char.ToUpper(c);

txtBoxAdd("\*\*",p[i].ToString());

}

clAPI.CopyMemory(myFileMapView, (IntPtr)p, size);

}

txtBoxAdd("[Done] ", "Ok");

#endregion

}

Итогом станет создание на диске файла с именем newFileATOM.txt. В этот файл вносим любой текст, запускаем программу на выполнение, получаем все содержимое файла в памяти и работаем непосредственно с памятью, после внесения всех изменений, производим обратное копирование из памяти в файл на диске, тем самым экономится время обработки.

