Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра «»

Лабораторная работа №4

по дисциплине «Операционные системы»

«Процессы»

Выполнил: студент гр..

Проверил:.

Тамбов, 20

***Цели и задачи****.*

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

Задание:

1. Для программы и библиотеки:

* in – дескриптор [0]
* имя [1]
* полное имя модуля [2]
* out – in\_0 out\_[1]\_[2]
* in\_1 out\_[0]\_[2]
* in\_2 out\_[0]\_[1]

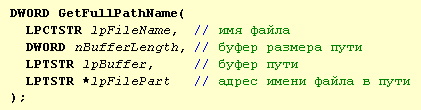
1. GetCurrentProcessId
   * вернет идентификатор текущего процесса

* GetCurrentProcess
  + вернет псевдодескриптор текущего процесса
  + DuplicateHandle
  + определит дескриптор текущего процесса
* OpenProcess
  + вернет копию дескриптора текущего процесса
* Закрыть
  + OpenProcess
  + DuplicateHandle

1. Вывести список всех процессов
   * потоков
   * модулей
   * их свойства в системе

***Решение задачи****.*

Используемые функции:

DWORD GetFullPathName( lpszFile, cchPath, lpszLongPath, plpszFileNamePart );  
DWORD GetShortPathName( lpszLongPath, lpszShortPath, cchBuffer );  
  
Функция GetFullPathName возвращает полное, включая путь, длинное имя файла, заданного параметром lpszFile. Если указанное имя не содержит пути, то предполагается, что файл находится в текущем каталоге. Функция GetShortPathName выполняет обратную задачу — она сообщает короткое имя файла, заданного его длинным именем.

**Параметры**

***lpFileName***

[in] Указатель на символьную строку с нулем в конце, которая определяет допустимое имя файла. Эта символьная строка может использовать или короткие ( формы 8.3 ) или длинные имена файла.

***nBufferLength***

[in] Размер буфера для приема строки с нулем в конце для  диска и пути, в **TCHARs** (символах).

***lpBuffer***

[out] Указатель на буфер, который получает символьную строку с нулем в конце для диска и пути.

***lpFilePart***

[out] Указатель на буфер, который получает адрес (в ***lpBuffer***) имени заключающего файла, как составляющей части в строке пути.

**Функция GetModuleHandle**

Функция **GetModuleHandle** извлекает дескриптор указанного модуля, если файл был отображен в адресном пространстве вызывающего процесса.

Чтобы избежать состояний гонки, описанных в разделе Замечаний, используйте функцию [**GetModuleHandleEx**](http://www.vsokovikov.narod.ru/New_MSDN_API/DLL/fn_getmodulehandleex.htm).

**Параметры**

***lpModuleName***

[in] Указатель на символьную строку с нулем в конце, которая содержит имя модули (или **.dll** или **.exe** файл). Если расширение имени файла опускается, в конец добавляется заданное по умолчанию библиотечное расширение **.dll**. Символьная строка имени файла может включать в себя конечный символ точки (**.**), который указывает, что имя модуля не имеет расширения. Строка не должна определять путь. Когда определяется путь, убедитесь, что используются обратные слэши (**\**), а не прямые слэши (**/**). Имя сравнивается (независимо от ситуации) с именами модулей в текущий момент отображаемыми в адресном пространстве вызывающего процесса.

Если этот параметр - NULL, **GetModuleHandle** возвращает дескриптор файла, используемый, чтобы создать вызывающий процесс (**.exe** файл).

**Возвращаемые значения**

Если функция завершается успешно, возвращаемое значение - дескриптор указанного модули.

Если функция завершается ошибкой, возвращаемое значение - NULL. Чтобы получить дополнительную информацию об ошибке, вызовите [**GetLastError**](http://www.vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm).

**Замечания**

Возвращенный дескриптор - не глобальный или наследуемый. Его нельзя дублировать или использовать другим процессом.

Если ***lpModuleName*** не включает в себя путь и имеется больше, чем один загруженный модуль с тем же самым базовым именем и расширением, функция извлекает дескриптор модуля, который был загружен сначала.

Функция **GetModuleHandle** возвращает дескриптор отображаемого модуля без увеличения на единицу ее итогового числа ссылок. Поэтому, используйте осторожно при передаче дескриптора функции [**FreeLibrary**](http://www.vsokovikov.narod.ru/New_MSDN_API/DLL/fn_freelibrary.htm), потому что такое выполнение может заставить модуль **DLL**   преждевременно прекратить отображение.

Эта функция должна использоваться тщательно в многопоточном приложении. Нет никакой гарантии, что дескриптор модуля остается допустимым в промежутке времени когда эта функция возвращает дескриптор и, когда он используется. Например, поток извлекает дескриптор модуля, но прежде, чем он использует этот дескриптор, второй поток освобождает модуль. Если система загружает другой модуль, он может  многократно использовать дескриптор модуля, который был недавно освобожден. Поэтому, первый поток получит дескриптор модуля другой, а не предназначенный.

**Функция GetCurrentProcessId**

Функция **GetCurrentProcessId** извлекает идентификатор вызывающего процесса.

**Возвращаемые значения**

Величина возвращаемого значения - идентификатор вызывающего процесса.

**Замечания**

До тех пор, пока процесс не закончит работу, идентификатор процесса уникально идентифицирует его повсюду в системе.

**Функция GetCurrentProcess**

Функция **GetCurrentProcess** извлекает псевдодескриптор для текущего процесса.

**Возвращаемые значения**

Величина возвращаемого значения - псевдодескриптор текущего процесса.

**Замечания**

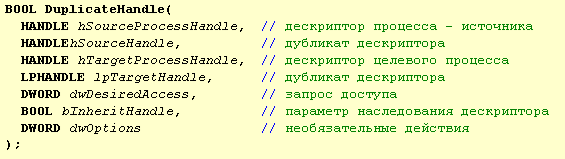
Псевдодескриптор - специальная константа обычно **(HANDLE)-1**, которая интерпретируется как дескриптор текущего процесса. Для совместимости с будущими операционными системами, самое лучшее, вызвать функцию **GetCurrentProcess,** вместо жестко запрограммированного этого постоянного значения. Вызывающий процесс может использовать псевдодескриптор, чтобы устанавливать свой собственный процесс всякий раз, когда требуется дескриптор процесса. Псевдодескрипторы дочерними процессами не наследуются.

Этот дескриптор имеет максимально возможный доступ к объекту процесса. Для систем, которые поддерживают дескрипторы безопасности, это - максимальный доступ, разрешенный дескриптором безопасности для вызывающего процесса. Для систем, которые не поддерживают дескрипторы безопасности, это - **PROCESS\_ALL\_ACCESS**. Для получения дополнительной информации, см. статью [**Защита процесса и права доступа**](http://www.vsokovikov.narod.ru/New_MSDN_API/Process_thread/security_access_process.htm).

Процесс может создавать "действительный" дескриптор для себя, который является допустимым применительно к другим процессы, или который может быть унаследован другими процессами, при помощи определения псевдодескриптора как исходного дескриптора при вызове функции [**DuplicateHandle**](http://www.vsokovikov.narod.ru/New_MSDN_API/Handles_objects/fn_duplicatehandle.htm). Процесс может использовать также и функцию [**OpenProcess**](http://www.vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_openprocess.htm), чтобы открыть действительный дескриптор для себя.

Когда псевдодескриптор больше не нужен, закрывать его нет необходимости. Вызов функции [**CloseHandle**](http://www.vsokovikov.narod.ru/New_MSDN_API/Handles_objects/fn_closehandle.htm) с псевдодескриптором не имеет никакого влияния. Если псевдодескриптор продублирован функцией **DuplicateHandle**, продублированный дескриптор должен быть закрыт.

**Функция DuplicateHandle**

Функция **DuplicateHandle** делает копию дескриптора объекта. Продублированный дескриптор относится к тому же самому объекту, что и исходный дескриптор. Поэтому, любое изменение объекта отражается через оба дескриптора. Например, дескриптор файла, зарегистрированный для текущего файла, является всегда одним и тем же для обоих дескрипторов.

**Параметры**

***hSourceProcessHandle***

[in] Дескриптор процесса, дескриптор которого дублируется.

**Windows NT /2000/XP:** дескриптор должен иметь право доступа **PROCESS\_DUP\_HANDLE**. Для получения дополнительной информации, см. статью Защита процесса и права доступа.

***hSourceHandle***

[in] Дескриптор, который дублируется. Это открытый дескриптор объекта, который является правильным применительно к исходному процессу. Перечень объектов, дескрипторы которых могут быть продублированы, см. нижеследующем разделе Замечаний.

***hTargetProcessHandle***

[in] Дескриптор процесса, который должен получить продублированный дескриптор. Дескриптор должен иметь право доступа **PROCESS\_DUP\_HANDLE**.

***lpTargetHandle***

[out] Указатель на переменную, которая получает значение продублированного дескриптора. Это значение дескриптора правильно применительно к целевому процессу.

Если параметр ***lpTargetHandle*** имеет значение ПУСТО (NULL), функция делает копию дескриптора, но не возвращает значение продублированного дескриптора вызывающей программе. Это поведение существует только для совместимости вниз с предшествующими версиями этой функции. Вы не должны использовать эту функцию, поскольку будете терять системные ресурсы до тех пор, пока не целевой процесс не закончит свою работу.

***dwDesiredAccess***

[in] Доступ, требуемый для нового дескриптора. Перечень флажков, которые могут быть установлены для каждого типа объекта, см. в нижеследующем разделе Замечаний.

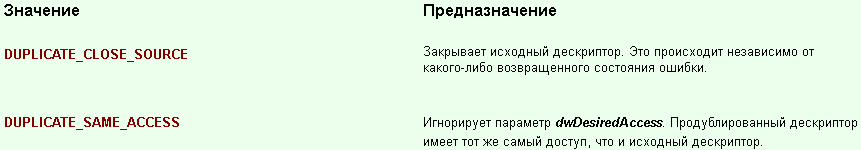
Этот параметр игнорируется, если параметр***dwOptions*** устанавливает флажок **DUPLICATE\_SAME\_ACCESS**. Иначе, флажки, которые могут быть установлены, зависят от типа объекта, дескриптор которого должен быть продублирован.

***bInheritHandle***

[in] Указывает, является ли дескриптор наследуемым. Если он - ИСТИНА (TRUE), продублированный дескриптор может быть унаследован новыми процессами, созданными целевым процессом. Если он - ЛОЖЬ (FALSE), новый дескриптор не может быть унаследован.

***dwOptions***

[in] Необязательные действия. Этот параметр может быть нулевой, или любой комбинацией следующих значений.



**Возвращаемые значения**

Если функция завершается успешно, величина возвращаемого значения - не ноль.

Если функция завершается с ошибкой, величина возвращаемого значения - ноль. Чтобы получать расширенные данные об ошибках, вызовите [**GetLastError**](http://www.vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm).

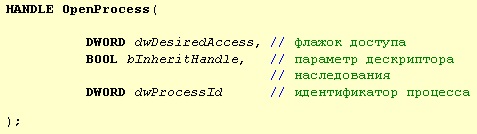
**Замечания**

Функция **DuplicateHandle** может вызываться или исходным процессом, или целевым процессом. Она может также быть вызвана, где исходный и целевой процесс - одно и то же. Например, процесс может использовать **DuplicateHandle**, чтобы создать ненаследуемый дубликат наследуемого дескриптора, или дескриптора с другими правами доступа, чем исходный дескриптор.

Процесс дублирования использует функцию [**GetCurrentProcess**](http://www.vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_getcurrentprocess.htm), чтобы получить свой собственный дескриптор. Чтобы получить другой дескриптор процесса, может быть необходимо использовать некоторую форму межпроцессорного взаимодействия (например, именованный канал или совместно используемую память), чтобы сообщить идентификатор процесса процессу дублирования. Этот идентификатор затем используется в функции [**OpenProcess**](http://www.vsokovikov.narod.ru/New_MSDN_API/Process_thread/fn_openprocess.htm), чтобы открыть дескриптор.

Если процесс, который вызывает функцию **DuplicateHandle** - не целевой процесс, процесс дублирования должен использовать межпроцессорную связь, чтобы передать значение продублированного дескриптора целевому процессу.

**Функция OpenProcess**

Функция **OpenProcess** открывает существующий объект процесса.

**Параметры**

***dwDesiredAccess***

[in] Устанавливает уровень доступа к объекту процесса. Этот параметр может состоять из одного нескольких [**прав доступа к процессу**](http://www.vsokovikov.narod.ru/New_MSDN_API/Process_thread/security_access_process.htm).

**Windows NT /200/XP:** Это право доступа проверяется у любого дескриптора безопасности для процесса.

***bInheritHandle***

[in] Если этот параметр является ИСТИНА (TRUE), дескриптор наследуем. Если этот параметр является ЛОЖЬ (FALSE), дескриптор не может наследоваться.

***dwProcessId***

[in] Идентификатор процесса, который открыт.

**Возвращаемые значения**

Если функция завершается успешно, величина возвращаемого значения - открытый дескриптор заданного процесса.

Если функция завершается с ошибкой, величина возвращаемого значения ПУСТО (NULL). Чтобы получить дополнительные данные об ошибке, вызовите [**GetLastError**](http://www.vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm).

**Замечания**

Дескриптор, возвращенный функцией **OpenProcess,** может быть использован в любой функции, которая требует дескриптора процесса, в такой, как [**функции ожидания**](http://www.vsokovikov.narod.ru/New_MSDN_API/Synchronization/wait_functions.htm) (**wait functions**), предоставляя соответствующие права доступа, которые были затребованы.

Когда Вы завершаете работать с дескриптором, убедитесь, что закрыли его, используя функцию [**CloseHandle**](http://www.vsokovikov.narod.ru/New_MSDN_API/Handles_objects/fn_closehandle.htm).

# Функция CloseHandle

Функция **CloseHandle** закрывает дескриптор открытого объекта.

**Параметры**

***hObject***

[in/out] Дескриптор открытого объекта.

**Возвращаемые значения**

Если функция завершается успешно, величина возвращаемого значения - не нуль.

Если функция завершается с ошибкой, величина возвращаемого значения - нуль. Чтобы получить дополнительные данные об ошибке, вызовите [**GetLastError**](http://www.vsokovikov.narod.ru/New_MSDN_API/Debbag_error/fn_getlasterror.htm).

Поскольку значения псевдо-дескриптора допускаются как правильные от функции **GetCurrentHandle**, они (или **-** (минус)**1**) могут задаваться как значение ***hObject***.

Эта функция породит исключительную ситуацию, если или значение дескриптора, которое не допустимо (или 0) или значение псевдо-дескриптора, задается как значение ***hObject***, а приложение запущено в среде отладчика. Это включает в себя закрытие дескриптора дважды и использование функции **CloseHandle** для дескриптора, возвращенного функцией [**FindFirstFile**](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_findfirstfile.htm).

Функция **CloseHandle** аннулирует заданный дескриптор объекта, уменьшает итоговое число дескрипторов объекта и выполняет проверку наличия объекта. После того, как последний дескриптор объект закрывается, объект удаляется из системы.

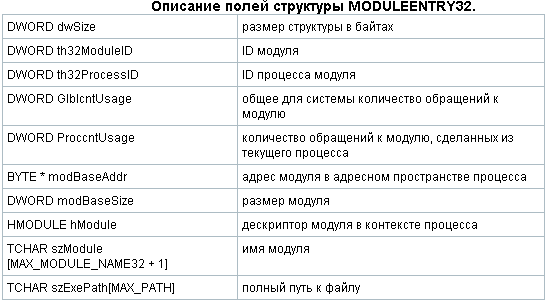
Закрытие дескриптора потока не завершает работу связанного потока. Чтобы удалить объект потока, Вы должны завершить работу потока, затем закрыть все дескрипторы потока.

Используйте функцию **CloseHandle**, чтобы закрыть дескрипторы, возвращенные при помощи вызова функции [**CreateFile**](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_createfile.htm). Используйте функцию [**FindClose**](http://www.vsokovikov.narod.ru/New_MSDN_API/Menage_files/fn_findclose.htm), чтобы закрыть дескрипторы, возвращенные при помощи вызова функции **FindFirstFile**.

**Структура THREADENTRY32**

Описание:   
TThreadEntry32 = record   
**dwSize: DWORD;   
cntUsage: DWORD;   
th32ThreadID: DWORD;   
th32OwnerProcessID: DWORD;   
tpBasePri: Longint;   
tpDeltaPri: Longint;   
dwFlags: DWORD;**end;   
  
Описывает блок списка, который перечисляет потоки, выполняющиеся в системе на момент снятия слепка.   
**dwSize**: Размер структуры в байтах. Заполнять так ThreadEntry32:= sizeof(TThreadEntry32)   
**cntUsage**: Счетчик ссылок на поток. Поток существует пока счетчик выше нуля. Поток завершается как только счетчик становиться равным нулю.   
**th32ThreadID**: Идентификатор потока. Этот идентификатор совместим с иднтификатором потока возвращаемым функцией CreateProcess.   
**th32OwnerProcessID**: Идентификатор процесса породившего поток. Содержание этого элемента может быть использовано элементами Win32 API.   
**tpBasePri**: Начальный приоритет назначенный потоку. См. THREAD\_PRIORITY\_IDLE, THREAD\_PRIORITY\_LOWEST, THREAD\_PRIORITY\_BELOW\_NORMAL, THREAD\_PRIORITY\_NORMAL, THREAD\_PRIORITY\_ABOVE\_NORMAL, THREAD\_PRIORITY\_HIGHEST, THREAD\_PRIORITY\_TIME\_CRITICAL   
**tpDeltaPri:** Изменение приритетного уровня потока. Это значение является всего лишь дельтой(разница) базового и измененного уровня потока.   
**dwFlags**: Зарезервированно.

**MODULEENTRY32**



Код программы:

using System;

using System.Runtime.InteropServices;

using System.Text;

namespace Eliza

{

**class ConnectDll**

{

#region Дескриптор-Имя-Путь

/// <summary>

/// get full path for run application

/// </summary>

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

[PreserveSig]//SysProperty\_defoult = true

public static extern uint **GetModuleFileName**([In]IntPtr hModule, [Out]StringBuilder lpFilename,

[In][MarshalAs(UnmanagedType.U4)]int nSize);

/// <summary>

/// GetFullPathName не ищёт файл на диске, а только составляет полное имя.по короткому

/// </summary>

[DllImport("kernel32.dll")]

public static extern uint **GetFullPathName**(string lpFileName, uint nBufferLength,

[Out] StringBuilder lpBuffer, out StringBuilder lpFilePart);

/// <summary>

/// Vозвращаемое значение - дескриптор указанного модуля.(или .dll или .exe файл)

/// </summary>

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

public static extern IntPtr **GetModuleHandle**(string lpModuleName);

#endregion

#region Открыть-ЗакрытьТекущийПроцесс

/// <summary>

/// Функция GetCurrentProcessId извлекает идентификатор вызывающего процесса.

/// </summary>

[DllImport("kernel32.dll")]

public static extern uint **GetCurrentProcessId**();

/// <summary>

/// Функция GetCurrentProcess извлекает псевдодескриптор для текущего процесса.

/// </summary>

[DllImport("kernel32.dll")]

public static extern IntPtr **GetCurrentProcess**();

/// <summary>

/// Функция DuplicateHandle делает копию дескриптора объекта.

/// </summary>

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

[return: MarshalAs(UnmanagedType.Bool)]

public static extern bool **DuplicateHandle**(IntPtr hSourceProcessHandle,

IntPtr hSourceHandle, IntPtr hTargetProcessHandle, out IntPtr lpTargetHandle,

uint dwDesiredAccess, [MarshalAs(UnmanagedType.Bool)] bool bInheritHandle, DuplicateOptions dwOptions);

[Flags]

public enum **DuplicateOptions** : uint

{

DUPLICATE\_CLOSE\_SOURCE = (0x00000001),// Закрывает исходный дескриптор.

DUPLICATE\_SAME\_ACCESS = (0x00000002), //Игнорирует параметр dwDesiredAccess.

}

/// <summary>

/// величина возвращаемого значения - открытый дескриптор заданного процесса.

/// </summary>

[DllImport("kernel32.dll")]

public static extern IntPtr **OpenProcess**(ProcessAccessFlags dwDesiredAccess,

[MarshalAs(UnmanagedType.Bool)] bool bInheritHandle, uint dwProcessId);

[Flags]

public enum **ProcessAccessFlags** : uint

{

All = 0x001F0FFF,

Terminate = 0x00000001,

CreateThread = 0x00000002,

VMOperation = 0x00000008,

VMRead = 0x00000010,

VMWrite = 0x00000020,

DupHandle = 0x00000040,

SetInformation = 0x00000200,

QueryInformation = 0x00000400,

Synchronize = 0x00100000

}

/// <summary>

/// Функция CloseHandle закрывает дескриптор открытого объекта.

/// </summary>

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

[return: MarshalAs(UnmanagedType.Bool)]

public static extern bool **CloseHandle**(IntPtr hObject);

#endregion

#region Процесс-Поток-Модуль

/// <summary>

/// создает снимок-структуру указанного процесса

/// </summary>

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

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

[Flags]

public enum **SnapshotFlags** : uint

{

HeapList = 0x00000001,

Process = 0x00000002,/\*TH32CS\_SNAPPROCESS\*/

Thread = 0x00000004,

Module = 0x00000008,

Module32 = 0x00000010,

All = (HeapList | Process | Thread | Module),

Inherit = 0x80000000,

NoHeaps = 0x40000000

}

/// <summary>

/// Первый в снимке

/// </summary>

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

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

/// <summary>

/// Последующий в снимке

/// </summary>

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

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

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

public struct **PROCESSENTRY32**

{

const int MAX\_PATH = 260;

internal UInt32 dwSize;//размер структуры в байтах

internal UInt32 cntUsage;//число ссылок на процесс

internal UInt32 th32ProcessID;//является идентификатором процесса

internal IntPtr th32DefaultHeapID;//

internal UInt32 th32ModuleID;// .

internal UInt32 cntThreads;//число потоков, принадлежащих процессу.

internal UInt32 th32ParentProcessID;//идентификатор родительского по отношению к текущему процесса

internal Int32 pcPriClassBase;//cодержит базовый приоритет процесса.

internal UInt32 dwFlags;

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

internal string szExeFile;//cодержит имя файла, создавшего процесс

}

[DllImport("kernel32.dll")]

public static extern bool **Thread32First**(IntPtr hSnapshot, ref THREADENTRY32 lpte);

[DllImport("kernel32.dll")]

public static extern bool **Thread32Next**(IntPtr hSnapshot, out THREADENTRY32 lpte);

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

public struct **THREADENTRY32**

{

internal UInt32 dwSize;//Размер структуры в байтах

internal UInt32 cntUsage;//Счетчик ссылок на поток

internal UInt32 th32ThreadID;//Идентификатор потока

internal UInt32 th32OwnerProcessID;//Идентификатор процесса породившего поток

internal UInt32 tpBasePri;//Начальный приоритет назначенный потоку

internal UInt32 tpDeltaPri;//Изменение приритетного уровня потока

internal UInt32 dwFlags;//Зарезервированно

}

[DllImport("kernel32.dll")]

public static extern IntPtr **OpenThread**(ThreadAccess dwDesiredAccess, bool bInheritHandle,

uint dwThreadId);

[Flags]

public enum **ThreadAccess** : int

{

TERMINATE = (0x0001),

SUSPEND\_RESUME = (0x0002),

GET\_CONTEXT = (0x0008),

SET\_CONTEXT = (0x0010),

SET\_INFORMATION = (0x0020),

QUERY\_INFORMATION = (0x0040),

SET\_THREAD\_TOKEN = (0x0080),

IMPERSONATE = (0x0100),

DIRECT\_IMPERSONATION = (0x0200)

}

[DllImport("kernel32.dll")]

public static extern bool **Module32First**([In]IntPtr hSnapshot, ref MODULEENTRY32 lpme);

[DllImport("kernel32.dll")]

public static extern bool **Module32Next**([In]IntPtr hSnapshot, ref MODULEENTRY32 lpme);

public struct **MODULEENTRY32**

{//http://pastebin.com/BzD1jdmH

private const int MAX\_PATH = 255;

internal uint dwSize; //размер структуры в байтах

internal uint th32ModuleID;//ID модуля

internal uint th32ProcessID;//ID процесса модуля

internal uint GlblcntUsage;//общее для системы количество обращений к модулю

internal uint ProccntUsage;//количество обращений к модулю, сделанных из текущего процесса

internal IntPtr modBaseAddr;//адрес модуля в адресном пространстве процесса

internal uint modBaseSize;//размер модуля

internal IntPtr hModule;//дескриптор модуля в контексте процесса

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = MAX\_PATH + 1)]

internal string szModule;//имя модуля

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = MAX\_PATH+5)]

internal string szExePath;//полный путь к файлу

}

#endregion

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ФОРМА\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

using System;

using System.Diagnostics;

using System.Drawing;

using System.Runtime.InteropServices;

using System.Text;

using System.Windows.Forms;

namespace Eliza

{

public partial class Form1 : Form

{

public **Form1**()

{

InitializeComponent();

**ДескрипторИмяПуть**();

**ПроцессПотокМодуль**();

}

private void **ДескрипторИмяПуть**()

{

#region Дескриптор-Имя-Путь

StringBuilder fileName = new StringBuilder(255);

//get full path for run application

ConnectDll.**GetModuleFileName**(IntPtr.Zero, fileName, fileName.Capacity);

AddToListViewer("Полный путь до экзешника", fileName.ToString());

int bufsz = 1;// We'll grow this as necessary

string sRel =

**@"Eliza.pdb";**// Note relative path

AddToListViewer("Восстановить путь до", sRel);

StringBuilder sbFull = new StringBuilder(bufsz); // Full path will go here

StringBuilder sbFile = new StringBuilder(bufsz); // Filename will go here

uint u = ConnectDll.**GetFullPathName**(sRel, (uint)bufsz, sbFull, out sbFile); // 1st call: Get necessary bufsz

if (u > bufsz) // 'u' should be >1

{

bufsz = (int)u + 10; // Required size plus a few

sbFull = new StringBuilder(bufsz); // Re-create objects w/ proper size

sbFile = new StringBuilder(bufsz); // "

u = ConnectDll.**GetFullPathName**(sRel, (uint)bufsz, sbFull, out sbFile); // Try again, this should succeed

// 'sbFull' should now contain "c:\windows\system32\desktop.ini"

// and 'sbFile' should contain "desktop.ini"

}

AddToListViewer("Имя файла восстановлено", sbFile.ToString());

AddToListViewer("Полный путь до " + sbFile, sbFull.ToString());

string module = **"User32.dll";**

IntPtr ModKernel32 = ConnectDll.**GetModuleHandle**(module);

AddToListViewer("Дескриптор модуля " + module, ModKernel32.ToString());

#endregion

}

private void **ОткрытьЗакрытьТекущийПроцесс**()

{

#region Открыть-ЗакрытьТекущийПроцесс

uint MyID = ConnectDll.**GetCurrentProcessId**();

AddToListViewer("ID текущего процесса ", MyID.ToString());

IntPtr hproc = ConnectDll.**GetCurrentProcess**();

AddToListViewer("Текущий псевдо дескриптор ", hproc.ToString());

IntPtr duplicat;

bool state = ConnectDll.**DuplicateHandle**(hproc, hproc, hproc, out duplicat, 0, false, ConnectDll.DuplicateOptions.DUPLICATE\_SAME\_ACCESS);

AddToListViewer(state + " Копия дескриптора " + hproc, duplicat.ToString());

IntPtr handlPtr = ConnectDll.**OpenProcess**(ConnectDll.ProcessAccessFlags.DupHandle, true, MyID);

AddToListViewer("Открытый дескриптор от " + MyID, handlPtr.ToString());

bool ste = ConnectDll.**CloseHandle**(duplicat);

AddToListViewer("Закрыт дескриптор " + duplicat, ste.ToString());

bool steB = ConnectDll.**CloseHandle**(handlPtr);

AddToListViewer("Закрыт дескриптор " + handlPtr, steB.ToString());

#endregion

}

private void **ПроцессПотокМодуль**()

{

#region Процесс-Поток-Модуль

Process pr = **CurrentParentProcess**;

AddToListViewer("Родительский ID", pr.Id.ToString(), true);

AddToListViewer("Родительский Name", pr.ProcessName, true);

#endregion

}

private Process **CurrentParentProcess**

{

get

{

return **GetParentProcess**(**Process.GetCurrentProcess().Id**);

}

}

private Process **GetParentProcess**(int pid)

{

Process parentProc = null;

IntPtr handleToSnapshot = IntPtr.Zero;

try

{

ConnectDll.**PROCESSENTRY32** procEntry = new ConnectDll.**PROCESSENTRY32**();

procEntry.dwSize = (UInt32)Marshal.**SizeOf**(typeof(ConnectDll.PROCESSENTRY32));

handleToSnapshot = ConnectDll.**CreateToolhelp32Snapshot**(ConnectDll.**SnapshotFlags**.**Process**, 0);

AddToListViewer("Снимок Процессов Системы Колич.", handleToSnapshot.ToString(), true);

if (ConnectDll.**Process32First**(handleToSnapshot, ref procEntry))

{

do

{

if (pid == procEntry.**th32ProcessID**)

{

parentProc = **Process.GetProcessById((int)procEntry.th32ParentProcessID);**

break;

}

**ПоказатьСодержимоеСтруктуры**(procEntry);//проход по всем процессам снимка

} while (ConnectDll.**Process32Next**(handleToSnapshot, ref procEntry));

}

finally

{

// Must clean up the snapshot object!

ConnectDll.**CloseHandle**(handleToSnapshot);

}

return parentProc;

}

private void **ПоказатьСодержимоеСтруктуры**(ConnectDll.PROCESSENTRY32 entry)

{

AddToListViewer("Имя файла, создавшего процесс", entry.szExeFile, true, true);

AddToListViewer("ID процесса", entry.th32ProcessID.ToString(), true);

AddToListViewer("th32DefaultHeapID", entry.th32DefaultHeapID.ToString(), true);

AddToListViewer("th32ModuleID", entry.th32ModuleID.ToString(), true);

AddToListViewer("ID родительского по отношению к текущему процесса", entry.th32ParentProcessID.ToString(), true);

AddToListViewer("Число ссылок на процесс", entry.cntUsage.ToString(), true);

AddToListViewer("Число потоков, принадлежащих процессу", entry.cntThreads.ToString(), true);

AddToListViewer("Базовый приоритет процесса", entry.pcPriClassBase.ToString(), true);

AddToListViewer("dwFlags", entry.dwFlags.ToString(), true);

}

private IntPtr **GetParentThread**(int tid)

{

IntPtr hThread = new IntPtr();

IntPtr handleToSnapshot = IntPtr.Zero;

try

{

//threadEntry - описание очередного потока в текущем процессе

ConnectDll.**THREADENTRY32** threadEntry = new ConnectDll.**THREADENTRY32**();

threadEntry.dwSize = (UInt32)Marshal.**SizeOf**(**typeof**(ConnectDll.**THREADENTRY32**));

handleToSnapshot = ConnectDll.**CreateToolhelp32Snapshot**(ConnectDll.**SnapshotFlags**.**Thread**, 0);

AddToListViewer("Снимок потоков системы Количество", handleToSnapshot.ToString(), true);

if (ConnectDll.**Thread32First**(handleToSnapshot, ref threadEntry))

{

do

{// относится ли данный поток к нужному процессу

if (tid == threadEntry.**th32OwnerProcessID**)

{

// пытаемся получить описатель потока по его идентификатору

hThread = ConnectDll.**OpenThread**(ConnectDll.ThreadAccess.SUSPEND\_RESUME, false, threadEntry.th32ThreadID);

AddToListViewer("Oписатель потока идентификатора " + threadEntry.th32ThreadID, hThread.ToString(), true);

}

**ПоказатьСодержимоеПотока**(threadEntry);//проход по всем полям снимка

} while (ConnectDll.**Thread32Next**(handleToSnapshot, out threadEntry));

}

finally

{

// Must clean up the snapshot object!

ConnectDll.**CloseHandle**(handleToSnapshot);

}

return hThread;

}

private void **ПоказатьСодержимоеПотока**(ConnectDll.THREADENTRY32 threadEntry)

{

AddToListViewer("Идентификатор потока", threadEntry.th32ThreadID.ToString(), true, true);

AddToListViewer("Размер структуры в байтах", threadEntry.dwSize.ToString(),true);

AddToListViewer("Счетчик ссылок на поток", threadEntry.cntUsage.ToString(), true);

AddToListViewer("Идентификатор процесса породившего поток", threadEntry.th32OwnerProcessID.ToString(), true);

AddToListViewer("Начальный приоритет назначенный потоку", threadEntry.tpBasePri.ToString(), true);

AddToListViewer("Изменение приритетного уровня потока", threadEntry.tpDeltaPri.ToString(), true);

AddToListViewer("Зарезервированно", threadEntry.dwFlags.ToString(), true);

}

private IntPtr **GetParentModule**(int mid)

{

IntPtr parentMod = IntPtr.Zero;

IntPtr handleToSnapshot = IntPtr.Zero;

try

{

ConnectDll.**MODULEENTRY32** moduleEntry = new ConnectDll.**MODULEENTRY32**();

moduleEntry.dwSize = (UInt32)Marshal.**SizeOf**(**typeof**(ConnectDll.**MODULEENTRY32**));

handleToSnapshot =

ConnectDll.**CreateToolhelp32Snapshot**(//http://pastebin.com/BzD1jdmH

ConnectDll.SnapshotFlags.Module | ConnectDll.SnapshotFlags.Module32, 0);

AddToListViewer("Снимок мoдули системы Количество", handleToSnapshot.ToString(), true);

if (ConnectDll.**Module32First**(handleToSnapshot, ref moduleEntry))

{

do

{

if (mid == moduleEntry.**th32ProcessID**)

{

parentMod = (IntPtr)Process.GetProcessById((int)moduleEntry.th32ProcessID).Id;

}

**ПоказатьСодержимоеМодуля**(moduleEntry);//проход по всем процессам снимка

} while (ConnectDll.**Module32Next**(handleToSnapshot, ref moduleEntry));

}

finally

{

// Must clean up the snapshot object!

ConnectDll.**CloseHandle**(handleToSnapshot);

}

return parentMod;

}

private void **ПоказатьСодержимоеМодуля**(ConnectDll.MODULEENTRY32 moduleEntry)

{

AddToListViewer("Имя модуля", moduleEntry.szModule, true, true);

AddToListViewer("Полный путь к файлу", moduleEntry.szExePath, true);

AddToListViewer("Размер структуры в байтах", moduleEntry.dwSize.ToString(), true);

AddToListViewer("ID модуля",moduleEntry.th32ModuleID.ToString(), true);

AddToListViewer("ID процесса модуля",moduleEntry.th32ProcessID.ToString(), true);

AddToListViewer("Oбщее для системы количество обращений к модулю", moduleEntry.GlblcntUsage.ToString(), true);

AddToListViewer("Kоличество обращений к модулю, сделанных из текущего процесса",

moduleEntry.ProccntUsage.ToString(), true);

AddToListViewer("Aдрес модуля в адресном пространстве процесса", moduleEntry.modBaseAddr.ToString(), true);

AddToListViewer("Pазмер модуля", moduleEntry.modBaseSize.ToString(), true);

AddToListViewer("Дескриптор модуля в контексте процесса", moduleEntry.hModule.ToString(), true);

}

private void AddToListViewer(string left, string right, params bool[] flag)

{

ListViewItem stroka1 = new ListViewItem(left);

stroka1.SubItems.Add(right);

if(flag.Length == 0)listView1.Items.Add(stroka1);

else

{

if (flag.Length != 1) stroka1.BackColor = Color.Red;

listView2.Items.Add(stroka1);

}

}

private void **Form1**\_**Resize**(object sender, EventArgs e)

{

columnHeader2.Width = groupBox1.Width - 220;

columnHeader4.Width = groupBox2.Width - 268;

groupBox2.Height = Form1.ActiveForm.Height - groupBox1.Height - 95;

}

private void **radioButton2**\_**CheckedChanged**(object sender, EventArgs e)

{

listView1.Items.Clear();

int k = Convert.ToInt32((sender as RadioButton).TabIndex);

switch (k)

{

case 1: //left cheked

if(radioButton1.Checked)

**ДескрипторИмяПуть**();

break;

case 2: //right cheked

if(radioButton2.Checked)

**ОткрытьЗакрытьТекущийПроцесс**();

break;

}

}

private void **radioButton3**\_**CheckedChanged**(object sender, EventArgs e)

{

listView2.Items.Clear();

int k = Convert.ToInt32((sender as RadioButton).TabIndex);

switch (k)

{

case 3: //left cheked

if (radioButton3.Checked)

**ПроцессПотокМодуль();**

break;

case 4: //center cheked

if (radioButton4.Checked)

**GetParentThread**(Process.GetCurrentProcess().Id);

break;

case 5: //right cheked

if (radioButton5.Checked)

**GetParentModule**(Process.GetCurrentProcess().Id);

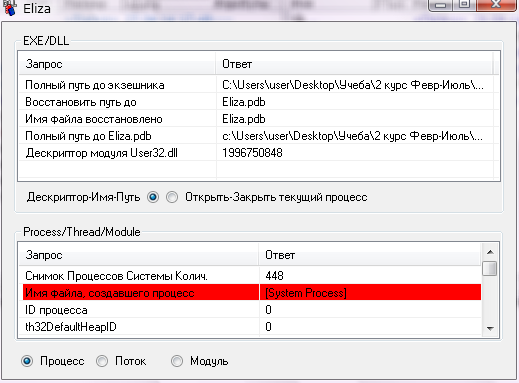
break;

}

}

}

}

Работа программы:

