**2** **Взаимодействие с памятью ОС Linux**

**2.1 Вызов функций Windows API**

Программный код, который выполняется под управлением CLR (Common Language Runtime, т. е. общая среда выполнения языков), называется управляемым (managed) кодом. Программный код, выполняющийся вне среды выполнения CLR, называется неуправляемым (unmanaged) кодом. Примером неуправляемого программного кода служат функции Win32 API, компоненты COM, интерфейсы ActiveX. Несмотря на большое количество классов .NET Framework, содержащих множество методов, программисту все равно приходится иногда прибегать к неуправляемому коду. Надо сказать, что число вызовов неуправляемого кода уменьшается с выходом каждой новой версии .NET Framework.

Управляемый код .NET Framework может вызывать неуправляемую функцию из DLL (функцию Windows API) при помощи специального механизма Platform Invoke (сокр. P/Invoke). Для того чтобы обратиться к какой-нибудь неуправляемой библиотеке DLL, нужно преобразовать .NET-объекты в наборы struct, char\* и указателей на функции, как того требует язык C. Чтобы вызвать DLL-функцию из C#, сначала ее необходимо объявить. Для этого используется атрибут DllImport.

using System.Runtime.InteropServices;

public class Win32

{

[DllImport("User32.Dll")]

public static extern void SetWindowText(IntPtr hwnd,

String lpString);

}

Атрибут DllImport сообщает компилятору, где находится точка входа, что позволяет далее вызывать функцию из нужного места. Нужно всегда использовать тип IntPtr для HWND, HMENU и любых других описателей. Для LPCTSTR используется String, а сервисы взаимодействия (interop services) выполнят автоматический маршаллинг System.String в LPCTSTR до передачи в Windows. Компилятор ищет указанную выше функцию SetWindowText в файле User32.dll и перед ее вызовом автоматически преобразует строку в LPTSTR (TCHAR\*). Для каждого типа в C# определен свой тип, используемый при маршалинге по умолчанию (default marshaling type) . Для строк это LPTSTR.

**2.2 Функция WriteProcessMemory**

2.2.1 Основные параметры

Функция WriteProcessMemory пишет данные области памяти в заданном процессе. Вся область, чтобы записаться, должна быть доступна, или операция завершается ошибкой.

Синтаксис:

BOOL WriteProcessMemory(

HANDLE hProcess,

LPVOID lpBaseAddress,

LPCVOID lpBuffer,

SIZE\_T nSize,

SIZE\_T\* lpNumberOfBytesWritten

);

Параметры:

* hProcess;

[in] Дескриптор процесса, память которого должна модифицироваться. Дескриптор должен иметь доступ к процессу PROCESS\_VM\_WRITE и PROCESS\_VM\_OPERATION.

* lpBaseAddress;

[in] Указатель на базовый адрес в заданном процессе, в который должны быть записаны данные. Прежде, чем происходит какая-либо передача данных, система проверяет, что все данные в базовом адресе и в заданном размере памяти доступны для записи. Если дело обстоит так, то функция приступает к работе; в противном случае, функция завершается ошибкой.

* lpBuffer;

[in] Указатель на буфер, который содержит записываемые данные в адресном пространстве заданного процесса.

* nSize;

[in] Число байтов, записываемых в заданном процессе.

* lpNumberOfBytesWritten.

[out] Указатель на переменную, которая получает число байтов, переданное в указанный процесс. Этот параметр дополнительный. Если lpNumberOfBytesWritten - ПУСТО (NULL), этот параметр игнорируется.

2.2.2 Возвращаемые значения

Если функция завершается успешно, возвращаемое значение не нуль.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке, вызовите GetLastError. Функция завершится ошибкой, если затребованная операция записи пересекается с областью процесса, которая является недоступной.

2.2.3 Замечания

Следует отметить, что WriteProcessMemory копирует данные из указанного буфера в текущем процессе в область адреса заданного процесса. Любой процесс, который имеет дескриптор с доступом к процессу PROCESS\_VM\_WRITE и PROCESS\_VM\_OPERATION, чтобы записать в него, может вызвать функцию. Процесс, адресное пространство которого записывается - обычно, но не обязательно, отлаживается.

Вся область, в которую будет производится запись, должна быть доступна. Если это не так, функция завершается ошибкой, как отмечено выше.

**2.3 Функция ReadProcessMemory**

2.3.1 Основные параметры

Функция ReadProcessMemory читает данные из области памяти в заданном процессе. Вся область, которая читается, должна быть доступна, или операция завершается ошибкой.

Синтаксис:

BOOL ReadProcessMemory(

HANDLE hProcess,

LPCVOID lpBaseAddress,

LPVOID lpBuffer,

SIZE\_T nSize,

SIZE\_T\* lpNumberOfBytesRead

);

Параметры:

* hProcess;

[in] Дескриптор процесса, память которого читается. Дескриптор должен иметь доступ к процессу PROCESS\_VM\_READ.

* lpBaseAddress;

[in] Указатель на базовый адрес в заданном процессе из которого делается чтение. Прежде, чем происходит какая-либо передача данных, система проверяет, все ли данные в базовом адресе и памяти в заданном размере доступны для чтения. Если дело обстоит так, функции приступает к работе; в противном случае функция завершается ошибкой.

* lpBuffer;

[out] Указатель на буфер, который получает содержание из адресного пространства заданного процесса.

* nSize;

[in] Число байтов, которое читается из заданного процесса.

* lpNumberOfBytesRead.

[out] Указатель на переменную, которая получает число байтов, переданное в указанный буфер. Если lpNumberOfBytesRead - ПУСТО (NULL), этот параметр игнорируется.

2.3.2 Возвращаемые значения

Если функция завершается успешно, возвращаемое значение не нуль.

Если функция завершается ошибкой, возвращаемое значение равняется нулю. Чтобы получить дополнительную информацию об ошибке, вызовите GetLastError.

Функция завершается ошибкой, если затребованная операция чтения пересекается с областью процесса, которая является недоступной.

2.3.3 Замечания

Функция ReadProcessMemory копирует данные в заданном адресе, в диапазоне от адресного пространства указанного процесса в заданный определенный буфер текущего процесса. Любой процесс, который имеет дескриптор с доступом PROCESS\_VM\_READ, может вызвать функцию. Процесс, адресное пространство которого читается - обычно, но не обязательно, отлаживается.

Вся область, которая читается должна быть доступна. Если это не так, функция завершается ошибкой как отмечено выше.

**2.4 Чтение и запись памяти процесса**

Чтобы прочитать значение из адреса памяти, нужно импортировать 2 функции в C #: OpenProcess() и ReadProcessMemory() из kernel32.dll*.*

[DllImport("kernel32.dll")]

public static extern IntPtr OpenProcess(int dwDesiredAccess, bool bInheritHandle, int dwProcessId);

[DllImport("kernel32.dll")]

public static extern bool ReadProcessMemory(int hProcess,

int lpBaseAddress, byte[] lpBuffer, int dwSize, ref int lpNumberOfBytesRead);

Когда процесс открыт, нужно указать разрешение на чтение из памяти:

const int PROCESS\_WM\_READ = 0x0010;

Чтение данных из памяти процесса, в данном примере notepad.exe:

using System;

using System.Diagnostics;

using System.Runtime.InteropServices;

using System.Text;

public class MemoryRead

{

const int PROCESS\_WM\_READ = 0x0010;

[DllImport("kernel32.dll")]

public static extern IntPtr OpenProcess(int dwDesiredAccess, bool bInheritHandle, int dwProcessId);

[DllImport("kernel32.dll")]

public static extern bool ReadProcessMemory(int hProcess,

int lpBaseAddress, byte[] lpBuffer, int dwSize, ref int lpNumberOfBytesRead);

public static void Main()

{

Process process = Process.GetProcessesByName("notepad")[0];

IntPtr processHandle = OpenProcess(PROCESS\_WM\_READ, false, process.Id);

int bytesRead = 0;

byte[] buffer = new byte[24]; //'Hello World!' takes 12\*2 bytes because of Unicode

// 0x0046A3B8 is the address where I found the string, replace it with what you found

ReadProcessMemory((int)processHandle, 0x0046A3B8, buffer, buffer.Length, ref bytesRead);

Console.WriteLine(Encoding.Unicode.GetString(buffer) +

" (" + bytesRead.ToString() + "bytes)");

Console.ReadLine();

}

}

**2.5 Запись в память процесса**

Понадобится 2 функции: OpenProcess() и WriteProcessMemory().

Когда процесс открыт, нужно указать разрешение:

const int PROCESS\_VM\_WRITE = 0x0020;

const int PROCESS\_VM\_OPERATION = 0x0008;

Запись данных в память процесса:

using System;

using System.Diagnostics;

using System.Runtime.InteropServices;

using System.Text;

public class MemoryRead

{

const int PROCESS\_VM\_WRITE = 0x0020;

const int PROCESS\_VM\_OPERATION = 0x0008;

[DllImport("kernel32.dll")]

public static extern IntPtr OpenProcess(int dwDesiredAccess,

bool bInheritHandle, int dwProcessId);

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

static extern bool WriteProcessMemory(int hProcess, int lpBaseAddress,

byte[] lpBuffer, int dwSize, ref int lpNumberOfBytesWritten);

public static void Main()

{

Process process = Process.GetProcessesByName("notepad")[0];

IntPtr processHandle = OpenProcess(0x1F0FFF, false, process.Id);

int bytesWritten = 0;

byte[] buffer = Encoding.Unicode.GetBytes("It works!\0");

// '\0' marks the end of string

// replace 0x0046A3B8 with your address

WriteProcessMemory((int)processHandle, 0x0046A3B8, buffer, buffer.Length, ref bytesWritten);

Console.ReadLine();

}

}

**Лабораторная работа №2 - Взаимодействие с памятью в ОС Linux**

Цель лабораторной работы **–** освоение навыков работы с адресным прос-транством процессов ОС Windows. Создание приложения с помощью Win32 API.

Краткие теоретические сведения

Java. Для работы с адресным пространством процесса можно использовать функции из библиотеки **libc**, такие как **mmap()**, **mprotect()**, **memcpy()** и другие.

Для записи данных в адресное пространство процесса Linux на языке Python можно воспользоваться модулем **ctypes**. Можно использовать системный вызов **write** или функцию **memcpy**.

Задание на лабораторную работу

Все варианты должны иметь графический интерфейс. Разработать приложение, которое позволяет чтение и запись данных из адресного пространства процесса. Пользователь самостоятельно указывается адресное пространство в программе. Для поиска нужного адреса рекомендуется использовать бесплатную программу scanmem или gameconqueror.

Четные номера зачетных книжек работают с блокнотом, нечетные с любой компьютерной игрой.

Дополнительно позволить запуск из приложения собственного терминала (аналог команд Linux). Добавить по 10 команд для работы с памятью ОС, дать им пояснение.

Добавить в приложение информацию, правильность которой необходимо подтвердить и объяснить:

**Вариант 1.** Процент используемой физической памяти.

**Вариант 2.** Процент используемой оперативной памяти.

**Вариант 3.** Размер файла подкачки в байтах.

**Вариант 4.** Размер рабочего множества страниц.

**Вариант 5.** Количество свободных байтов файла подкачки.

**Вариант 6.** Количество страничных ошибок.

**Вариант 7.** Значение размера используемой оперативной памяти.

**Вариант 8.** Пиковое значение размера используемой оперативной памяти.

**Вариант 9.** Объем оперативной памяти.

**Вариант 10.** Значение размера используемой оперативной памяти разработанным приложением.

**Вариант 11.** Количество используемой оперативной памяти текущим пользователем.

**Вариант 12.** Ошибок страницы физической памяти/сек: среднее ежесекундное число ошибок отсутствия страницы физической памяти.

**Вариант 13.** Объем виртуальной памяти, зарезервированной ОС для процесса.

**Вариант 14.** Объем физической памяти, используемой сейчас процессом.

**Вариант 15.** Объем используемой процессом физической памяти, которая может совместно использоваться другими процессами.

**Вариант 16.** Объем используемой процессом физической памяти, которая может использоваться другими процессами.

**Вариант 17.** Средняя скорость чтения процесса за последнюю минуту на HDD/SSD.

**Вариант 18.** Средняя скорость записи процесса за последнюю минуту на HDD/SSD.

**Вариант 19.** Средняя скорость доступа (чтения и записи) процесса за последнюю минуту на HDD/SSD.

**Вариант 20.** Получить значение регистра EAX в текущий момент времени.

Дополнительное задание

Добавить командный интерпретатор **Windows PowerShell/CMD** с соответствующим функционалом и аналогичными командами.

Контрольные вопросы

1. Что такое дескриптор?
2. Зачем нужен интерфейс прикладного программирования?
3. Вызов функций Windows API?
4. Функция WriteProcessMemory?
5. Параметры функции WriteProcessMemory?
6. Возвращаемые значения функции WriteProcessMemory?
7. Функция ReadProcessMemory?
8. Параметры функции ReadProcessMemory?
9. Возвращаемые значения функции ReadProcessMemory?
10. Чтение памяти процесса?
11. Запись в память процесса?
12. Командные интерпретаторы для работы с памятью?