Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение   
высшего профессионального образования   
«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л. И. Сучкова

(подпись) (и.о., фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_2014 г.

дата

Отчет по лабораторной работе №7

по дисциплине Периферийные устройства

Студент группы ПИ-21 С. Ю. Искуснов

и.о., фамилия

Преподаватель к.т.н., профессор Л. И. Сучкова\_

должность, ученое звание и.о., фамилия

Барнаул 2014

Задание:

1. Изучить структуру драйвера для клавиатуры.

2. Разработать модификацию функций драйвера для выполнения фильтрации на основе примера из WDK.

Код программы:

/\*

\* COPYRIGHT: See COPYING in the top level directory

\* PROJECT: ReactOS Keyboard class driver

\* FILE: drivers/kbdclass/kbdclass.c

\* PURPOSE: Keyboard class driver

\*

\* PROGRAMMERS: Herve Poussineau (hpoussin@reactos.org)

\*/

#include "kbdclass.h"

#include <stdio.h>

#include <pseh/pseh2.h>

#include <kbdmou.h>

#include <debug.h>

static DRIVER\_UNLOAD DriverUnload;

static DRIVER\_DISPATCH ClassCreate;

static DRIVER\_DISPATCH ClassClose;

static DRIVER\_DISPATCH ClassCleanup;

static DRIVER\_DISPATCH ClassRead;

static DRIVER\_DISPATCH ClassDeviceControl;

static DRIVER\_DISPATCH IrpStub;

static DRIVER\_ADD\_DEVICE ClassAddDevice;

static DRIVER\_STARTIO ClassStartIo;

static DRIVER\_CANCEL ClassCancelRoutine;

static NTSTATUS HandleReadIrp(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp,

BOOLEAN IsInStartIo);

/\*

DriverUnload вызывается при выгрузке драйвера из системы

своеобразная точка выхода из программы

\*/

static VOID NTAPI

DriverUnload(IN PDRIVER\_OBJECT DriverObject)

{

// nothing to do here yet

}

/\*

Диспетчер ввода вывода посылает запрос IRP\_MJ\_CREATE, вызывается

данная функция, когда новый или существующий файл или устройство

были созданы. Обычно данный IRP посылается пользовательским

приложением при вызове функции CreateFile или иными функциями

в режиме ядра.

\*/

static NTSTATUS NTAPI

ClassCreate(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

TRACE\_(CLASS\_NAME, "IRP\_MJ\_CREATE\n");

if (!((PCOMMON\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension)->IsClassDO)

return ForwardIrpAndForget(DeviceObject, Irp);

/\* FIXME: open all associated Port devices \*/

Irp->IoStatus.Status = STATUS\_SUCCESS;

Irp->IoStatus.Information = 0;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_SUCCESS;

}

/\*

Обычно данный запрос следует после запроса очистки

\*/

static NTSTATUS NTAPI

ClassClose(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

TRACE\_(CLASS\_NAME, "IRP\_MJ\_CLOSE\n");

if (!((PCOMMON\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension)->IsClassDO)

return ForwardIrpAndForget(DeviceObject, Irp);

/\* FIXME: close all associated Port devices \*/

Irp->IoStatus.Status = STATUS\_SUCCESS;

Irp->IoStatus.Information = 0;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_SUCCESS;

}

/\*

Обычно следует за вызовом CloseHandle функции

\*/

static NTSTATUS NTAPI

ClassCleanup(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

TRACE\_(CLASS\_NAME, "IRP\_MJ\_CLEANUP\n");

if (!((PCOMMON\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension)->IsClassDO)

return ForwardIrpAndForget(DeviceObject, Irp);

/\* FIXME: cleanup all associated Port devices \*/

Irp->IoStatus.Status = STATUS\_SUCCESS;

Irp->IoStatus.Information = 0;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_SUCCESS;

}

/\*

Каждый драйвер устройства передающий данные от устройства

к системе должен применять процедуры DispatchRead or DispatchReadWrite

\*/

static NTSTATUS NTAPI

ClassRead(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

PCLASS\_DEVICE\_EXTENSION DeviceExtension = DeviceObject->DeviceExtension;

KIRQL OldIrql;

NTSTATUS Status;

TRACE\_(CLASS\_NAME, "IRP\_MJ\_READ\n");

ASSERT(DeviceExtension->Common.IsClassDO);

if (!((PCOMMON\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension)->IsClassDO)

return ForwardIrpAndForget(DeviceObject, Irp);

if (IoGetCurrentIrpStackLocation(Irp)->Parameters.Read.Length < sizeof(KEYBOARD\_INPUT\_DATA))

{

Irp->IoStatus.Status = STATUS\_BUFFER\_TOO\_SMALL;

Irp->IoStatus.Information = 0;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_BUFFER\_TOO\_SMALL;

}

KeAcquireSpinLock(&DeviceExtension->SpinLock, &OldIrql);

Status = HandleReadIrp(DeviceObject, Irp, FALSE);

KeReleaseSpinLock(&DeviceExtension->SpinLock, OldIrql);

return Status;

}

/\*

обработка запроса DispatchDeviceControl,

действия по управлению устройства

\*/

static NTSTATUS NTAPI

ClassDeviceControl(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

//PCLASS\_DEVICE\_EXTENSION DeviceExtension;

NTSTATUS Status = STATUS\_NOT\_SUPPORTED;

TRACE\_(CLASS\_NAME, "IRP\_MJ\_DEVICE\_CONTROL\n");

if (!((PCOMMON\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension)->IsClassDO)

return ForwardIrpAndForget(DeviceObject, Irp);

//DeviceExtension = (PCLASS\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension;

switch (IoGetCurrentIrpStackLocation(Irp)->Parameters.DeviceIoControl.IoControlCode)

{

case IOCTL\_KEYBOARD\_QUERY\_ATTRIBUTES:

case IOCTL\_KEYBOARD\_QUERY\_INDICATOR\_TRANSLATION:

case IOCTL\_KEYBOARD\_QUERY\_INDICATORS:

case IOCTL\_KEYBOARD\_QUERY\_TYPEMATIC:

{

/\* FIXME: Мы надеемся, что все устройства будут возвращать тот же результат.

\* спросить только первого \*/

PLIST\_ENTRY Head = &((PCLASS\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension)->ListHead;

if (Head->Flink != Head)

{

/\* We have at least one device \*/

PPORT\_DEVICE\_EXTENSION DevExt = CONTAINING\_RECORD(Head->Flink, PORT\_DEVICE\_EXTENSION, ListEntry);

IoGetCurrentIrpStackLocation(Irp)->MajorFunction = IRP\_MJ\_INTERNAL\_DEVICE\_CONTROL;

IoSkipCurrentIrpStackLocation(Irp);

return IoCallDriver(DevExt->DeviceObject, Irp);

}

break;

}

case IOCTL\_KEYBOARD\_SET\_INDICATORS:

case IOCTL\_KEYBOARD\_SET\_TYPEMATIC: /\* not in MSDN, would seem logical \*/

{

/\* Отправка запроса на все связанные устройства \*/

PLIST\_ENTRY Head = &((PCLASS\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension)->ListHead;

PLIST\_ENTRY Entry = Head->Flink;

Status = STATUS\_SUCCESS;

while (Entry != Head)

{

PPORT\_DEVICE\_EXTENSION DevExt = CONTAINING\_RECORD(Entry, PORT\_DEVICE\_EXTENSION, ListEntry);

NTSTATUS IntermediateStatus;

IoGetCurrentIrpStackLocation(Irp)->MajorFunction = IRP\_MJ\_INTERNAL\_DEVICE\_CONTROL;

IntermediateStatus = ForwardIrpAndWait(DevExt->DeviceObject, Irp);

if (!NT\_SUCCESS(IntermediateStatus))

Status = IntermediateStatus;

Entry = Entry->Flink;

}

break;

}

default:

WARN\_(CLASS\_NAME, "IRP\_MJ\_DEVICE\_CONTROL / unknown I/O control code 0x%lx\n",

IoGetCurrentIrpStackLocation(Irp)->Parameters.DeviceIoControl.IoControlCode);

ASSERT(FALSE);

break;

}

Irp->IoStatus.Status = Status;

Irp->IoStatus.Information = 0;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return Status;

}

/\*

Данная подпрограмма вызывается в случае, если драйвер не поддреживает

функции запрошиваемые системой

\*/

static NTSTATUS NTAPI

IrpStub(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

NTSTATUS Status = STATUS\_NOT\_SUPPORTED;

if (!((PCOMMON\_DEVICE\_EXTENSION)DeviceObject->DeviceExtension)->IsClassDO)

{

/\* Forward some IRPs to lower device \*/

switch (IoGetCurrentIrpStackLocation(Irp)->MajorFunction)

{

case IRP\_MJ\_PNP:

case IRP\_MJ\_INTERNAL\_DEVICE\_CONTROL:

return ForwardIrpAndForget(DeviceObject, Irp);

default:

{

ERR\_(CLASS\_NAME, "Port DO stub for major function 0x%lx\n",

IoGetCurrentIrpStackLocation(Irp)->MajorFunction);

ASSERT(FALSE);

}

}

}

else

{

ERR\_(CLASS\_NAME, "Class DO stub for major function 0x%lx\n",

IoGetCurrentIrpStackLocation(Irp)->MajorFunction);

ASSERT(FALSE);

}

Irp->IoStatus.Status = Status;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return Status;

}

/\*

Эта функция лезет в реестр.

Для чего?

Возможно из-за поддержки только одного устройства

Из реестра узнают типы устройств, что подключено и работают только с одним

из них! - и тут ты прав Евгений!

\*/

static NTSTATUS

ReadRegistryEntries(

IN PUNICODE\_STRING RegistryPath,

IN PCLASS\_DRIVER\_EXTENSION DriverExtension)

{

UNICODE\_STRING ParametersRegistryKey;

RTL\_QUERY\_REGISTRY\_TABLE Parameters[4];

NTSTATUS Status;

/\* HACK: We don't support multiple devices with this disabled \*/

ULONG DefaultConnectMultiplePorts = 1;

ULONG DefaultDataQueueSize = 0x64;

PCWSTR DefaultDeviceBaseName = L"KeyboardClass";

ParametersRegistryKey.Length = 0;

ParametersRegistryKey.MaximumLength = RegistryPath->Length + sizeof(L"\\Parameters") + sizeof(UNICODE\_NULL);

ParametersRegistryKey.Buffer = ExAllocatePoolWithTag(PagedPool, ParametersRegistryKey.MaximumLength, CLASS\_TAG);

if (!ParametersRegistryKey.Buffer)

{

WARN\_(CLASS\_NAME, "ExAllocatePoolWithTag() failed\n");

return STATUS\_NO\_MEMORY;

}

RtlCopyUnicodeString(&ParametersRegistryKey, RegistryPath);

RtlAppendUnicodeToString(&ParametersRegistryKey, L"\\Parameters");

ParametersRegistryKey.Buffer[ParametersRegistryKey.Length / sizeof(WCHAR)] = UNICODE\_NULL;

RtlZeroMemory(Parameters, sizeof(Parameters));

Parameters[0].Flags = RTL\_QUERY\_REGISTRY\_DIRECT | RTL\_REGISTRY\_OPTIONAL;

Parameters[0].Name = L"ConnectMultiplePorts";

Parameters[0].EntryContext = &DriverExtension->ConnectMultiplePorts;

Parameters[0].DefaultType = REG\_DWORD;

Parameters[0].DefaultData = &DefaultConnectMultiplePorts;

Parameters[0].DefaultLength = sizeof(ULONG);

Parameters[1].Flags = RTL\_QUERY\_REGISTRY\_DIRECT | RTL\_REGISTRY\_OPTIONAL;

Parameters[1].Name = L"KeyboardDataQueueSize";

Parameters[1].EntryContext = &DriverExtension->DataQueueSize;

Parameters[1].DefaultType = REG\_DWORD;

Parameters[1].DefaultData = &DefaultDataQueueSize;

Parameters[1].DefaultLength = sizeof(ULONG);

Parameters[2].Flags = RTL\_QUERY\_REGISTRY\_DIRECT | RTL\_REGISTRY\_OPTIONAL;

Parameters[2].Name = L"KeyboardDeviceBaseName";

Parameters[2].EntryContext = &DriverExtension->DeviceBaseName;

Parameters[2].DefaultType = REG\_SZ;

Parameters[2].DefaultData = (PVOID)DefaultDeviceBaseName;

Parameters[2].DefaultLength = 0;

Status = RtlQueryRegistryValues(

RTL\_REGISTRY\_ABSOLUTE,

ParametersRegistryKey.Buffer,

Parameters,

NULL,

NULL);

if (NT\_SUCCESS(Status))

{

/\* Check values \*/

if (DriverExtension->ConnectMultiplePorts != 0

&& DriverExtension->ConnectMultiplePorts != 1)

{

DriverExtension->ConnectMultiplePorts = DefaultConnectMultiplePorts;

}

if (DriverExtension->DataQueueSize == 0)

{

DriverExtension->DataQueueSize = DefaultDataQueueSize;

}

}

else if (Status == STATUS\_OBJECT\_NAME\_NOT\_FOUND)

{

/\* Registry path doesn't exist. Set defaults \*/

DriverExtension->ConnectMultiplePorts = DefaultConnectMultiplePorts;

DriverExtension->DataQueueSize = DefaultDataQueueSize;

if (RtlCreateUnicodeString(&DriverExtension->DeviceBaseName, DefaultDeviceBaseName))

Status = STATUS\_SUCCESS;

else

Status = STATUS\_NO\_MEMORY;

}

ExFreePoolWithTag(ParametersRegistryKey.Buffer, CLASS\_TAG);

return Status;

}

/\*

Создаёт объект устройство

Создаёт для него символьную ссылку

\*/

static NTSTATUS

CreateClassDeviceObject(

IN PDRIVER\_OBJECT DriverObject,

OUT PDEVICE\_OBJECT \*ClassDO OPTIONAL)

{

PCLASS\_DRIVER\_EXTENSION DriverExtension;

ULONG DeviceId = 0;

ULONG PrefixLength;

UNICODE\_STRING DeviceNameU;

PWSTR DeviceIdW = NULL; /\* Pointer into DeviceNameU.Buffer \*/

PDEVICE\_OBJECT Fdo;

PCLASS\_DEVICE\_EXTENSION DeviceExtension;

NTSTATUS Status;

TRACE\_(CLASS\_NAME, "CreateClassDeviceObject(0x%p)\n", DriverObject);

/\* Create new device object \*/

DriverExtension = IoGetDriverObjectExtension(DriverObject, DriverObject);

DeviceNameU.Length = 0;

DeviceNameU.MaximumLength =

wcslen(L"\\Device\\") \* sizeof(WCHAR) /\* "\Device\" \*/

+ DriverExtension->DeviceBaseName.Length /\* "KeyboardClass" \*/

+ 4 \* sizeof(WCHAR) /\* Id between 0 and 9999 \*/

+ sizeof(UNICODE\_NULL); /\* Final NULL char \*/

DeviceNameU.Buffer = ExAllocatePoolWithTag(PagedPool, DeviceNameU.MaximumLength, CLASS\_TAG);

if (!DeviceNameU.Buffer)

{

WARN\_(CLASS\_NAME, "ExAllocatePoolWithTag() failed\n");

return STATUS\_NO\_MEMORY;

}

Status = RtlAppendUnicodeToString(&DeviceNameU, L"\\Device\\");

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "RtlAppendUnicodeToString() failed with status 0x%08lx\n", Status);

goto cleanup;

}

Status = RtlAppendUnicodeStringToString(&DeviceNameU, &DriverExtension->DeviceBaseName);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "RtlAppendUnicodeStringToString() failed with status 0x%08lx\n", Status);

goto cleanup;

}

PrefixLength = DeviceNameU.MaximumLength - 4 \* sizeof(WCHAR) - sizeof(UNICODE\_NULL);

DeviceIdW = &DeviceNameU.Buffer[PrefixLength / sizeof(WCHAR)];

while (DeviceId < 9999)

{

DeviceNameU.Length = (USHORT)(PrefixLength + swprintf(DeviceIdW, L"%lu", DeviceId) \* sizeof(WCHAR));

Status = IoCreateDevice(

DriverObject,

sizeof(CLASS\_DEVICE\_EXTENSION),

&DeviceNameU,

FILE\_DEVICE\_KEYBOARD,

FILE\_DEVICE\_SECURE\_OPEN,

FALSE,

&Fdo);

if (NT\_SUCCESS(Status))

goto cleanup;

else if (Status != STATUS\_OBJECT\_NAME\_COLLISION)

{

WARN\_(CLASS\_NAME, "IoCreateDevice() failed with status 0x%08lx\n", Status);

goto cleanup;

}

DeviceId++;

}

WARN\_(CLASS\_NAME, "Too many devices starting with '\\Device\\%wZ'\n", &DriverExtension->DeviceBaseName);

Status = STATUS\_TOO\_MANY\_NAMES;

cleanup:

if (!NT\_SUCCESS(Status))

{

ExFreePoolWithTag(DeviceNameU.Buffer, CLASS\_TAG);

return Status;

}

DeviceExtension = (PCLASS\_DEVICE\_EXTENSION)Fdo->DeviceExtension;

RtlZeroMemory(DeviceExtension, sizeof(CLASS\_DEVICE\_EXTENSION));

DeviceExtension->Common.IsClassDO = TRUE;

DeviceExtension->DriverExtension = DriverExtension;

InitializeListHead(&DeviceExtension->ListHead);

KeInitializeSpinLock(&DeviceExtension->ListSpinLock);

KeInitializeSpinLock(&DeviceExtension->SpinLock);

DeviceExtension->InputCount = 0;

DeviceExtension->PortData = ExAllocatePoolWithTag(NonPagedPool, DeviceExtension->DriverExtension->DataQueueSize \* sizeof(KEYBOARD\_INPUT\_DATA), CLASS\_TAG);

if (!DeviceExtension->PortData)

{

ExFreePoolWithTag(DeviceNameU.Buffer, CLASS\_TAG);

return STATUS\_NO\_MEMORY;

}

DeviceExtension->DeviceName = DeviceNameU.Buffer;

Fdo->Flags |= DO\_POWER\_PAGABLE;

Fdo->Flags |= DO\_BUFFERED\_IO; /\* FIXME: Why is it needed for 1st stage setup? \*/

Fdo->Flags &= ~DO\_DEVICE\_INITIALIZING;

/\* Add entry entry to HKEY\_LOCAL\_MACHINE\HARDWARE\DEVICEMAP\[DeviceBaseName] \*/

RtlWriteRegistryValue(

RTL\_REGISTRY\_DEVICEMAP,

DriverExtension->DeviceBaseName.Buffer,

DeviceExtension->DeviceName,

REG\_SZ,

DriverExtension->RegistryPath.Buffer,

DriverExtension->RegistryPath.MaximumLength);

if (ClassDO)

\*ClassDO = Fdo;

return STATUS\_SUCCESS;

}

/\*

Буфер наполнен, пересылка из буфера в системный буфер или

\*/

static NTSTATUS

FillEntries(

IN PDEVICE\_OBJECT ClassDeviceObject,

IN PIRP Irp,

IN PKEYBOARD\_INPUT\_DATA DataStart,

IN SIZE\_T NumberOfEntries)

{

NTSTATUS Status = STATUS\_SUCCESS;

if (ClassDeviceObject->Flags & DO\_BUFFERED\_IO)

{

//Конкретно здесь пересылка байтов в системный буфер, т.е. устройство поддерживает буфер. в/в

RtlCopyMemory(

Irp->AssociatedIrp.SystemBuffer,

DataStart,

NumberOfEntries \* sizeof(KEYBOARD\_INPUT\_DATA));

}

else if (ClassDeviceObject->Flags & DO\_DIRECT\_IO)

{

//сначала получить адрес куда пересылать, затем переслать

PVOID DestAddress = MmGetSystemAddressForMdlSafe(Irp->MdlAddress, NormalPagePriority);

if (DestAddress)

{

RtlCopyMemory(

DestAddress,

DataStart,

NumberOfEntries \* sizeof(KEYBOARD\_INPUT\_DATA));

}

else

Status = STATUS\_UNSUCCESSFUL;

}

else

{

\_SEH2\_TRY

{

//пересылка байт в системный буфер

RtlCopyMemory(

Irp->UserBuffer,

DataStart,

NumberOfEntries \* sizeof(KEYBOARD\_INPUT\_DATA));

}

\_SEH2\_EXCEPT(EXCEPTION\_EXECUTE\_HANDLER)

{

Status = \_SEH2\_GetExceptionCode();

}

\_SEH2\_END;

}

return Status;

}

/\*

функция пересылает данные с очереди порта в очередь класса драйвера

\*/

static BOOLEAN NTAPI

ClassCallback(

IN PDEVICE\_OBJECT ClassDeviceObject,

IN OUT PKEYBOARD\_INPUT\_DATA DataStart,

IN PKEYBOARD\_INPUT\_DATA DataEnd,

IN OUT PULONG ConsumedCount)

{

PCLASS\_DEVICE\_EXTENSION ClassDeviceExtension = ClassDeviceObject->DeviceExtension;

KIRQL OldIrql;

SIZE\_T InputCount = DataEnd - DataStart;

SIZE\_T ReadSize;

TRACE\_(CLASS\_NAME, "ClassCallback()\n");

ASSERT(ClassDeviceExtension->Common.IsClassDO);

KeAcquireSpinLock(&ClassDeviceExtension->SpinLock, &OldIrql);

if (InputCount > 0)

{

//если данных больше чем очередь драйвера, то конец очереди обрезается

if (ClassDeviceExtension->InputCount + InputCount > ClassDeviceExtension->DriverExtension->DataQueueSize)

{

/\*

\* We're exceeding the buffer, and data will be thrown away...

\* FIXME: What could we do, as we are at DISPATCH\_LEVEL?

\*/

ReadSize = ClassDeviceExtension->DriverExtension->DataQueueSize - ClassDeviceExtension->InputCount;

}

else

ReadSize = InputCount;

/\*

\* Move the input data from the port data queue to our class data

\* queue.

\*/

//сама переселыка данных в порт данных

RtlCopyMemory(

&ClassDeviceExtension->PortData[ClassDeviceExtension->InputCount],

(PCHAR)DataStart,

sizeof(KEYBOARD\_INPUT\_DATA) \* ReadSize);

/\* Move the counter up \*/

//увеличить счётчик считанных данных

ClassDeviceExtension->InputCount += ReadSize;

(\*ConsumedCount) += (ULONG)ReadSize;

/\* Complete pending IRP (if any) \*/

if (ClassDeviceExtension->PendingIrp)

HandleReadIrp(ClassDeviceObject, ClassDeviceExtension->PendingIrp, FALSE);

}

KeReleaseSpinLock(&ClassDeviceExtension->SpinLock, OldIrql);

TRACE\_(CLASS\_NAME, "Leaving ClassCallback()\n");

return TRUE;

}

/\* Send IOCTL\_INTERNAL\_\*\_CONNECT to port \*/

static NTSTATUS

ConnectPortDriver(

IN PDEVICE\_OBJECT PortDO,

IN PDEVICE\_OBJECT ClassDO)

{

KEVENT Event;

PIRP Irp;

IO\_STATUS\_BLOCK IoStatus;

CONNECT\_DATA ConnectData;

NTSTATUS Status;

TRACE\_(CLASS\_NAME, "Connecting PortDO %p to ClassDO %p\n", PortDO, ClassDO);

KeInitializeEvent(&Event, NotificationEvent, FALSE);

ConnectData.ClassDeviceObject = ClassDO;

ConnectData.ClassService = ClassCallback;

Irp = IoBuildDeviceIoControlRequest(

IOCTL\_INTERNAL\_KEYBOARD\_CONNECT,

PortDO,

&ConnectData, sizeof(CONNECT\_DATA),

NULL, 0,

TRUE, &Event, &IoStatus);

if (!Irp)

return STATUS\_INSUFFICIENT\_RESOURCES;

Status = IoCallDriver(PortDO, Irp);

if (Status == STATUS\_PENDING)

KeWaitForSingleObject(&Event, Suspended, KernelMode, FALSE, NULL);

else

IoStatus.Status = Status;

if (NT\_SUCCESS(IoStatus.Status))

{

ObReferenceObject(PortDO);

ExInterlockedInsertTailList(

&((PCLASS\_DEVICE\_EXTENSION)ClassDO->DeviceExtension)->ListHead,

&((PPORT\_DEVICE\_EXTENSION)PortDO->DeviceExtension)->ListEntry,

&((PCLASS\_DEVICE\_EXTENSION)ClassDO->DeviceExtension)->ListSpinLock);

if (ClassDO->StackSize <= PortDO->StackSize)

{

/\* Increase the stack size, in case we have to

\* forward some IRPs to the port device object

\*/

ClassDO->StackSize = PortDO->StackSize + 1;

}

}

return IoStatus.Status;

}

/\* Send IOCTL\_INTERNAL\_\*\_DISCONNECT to port + destroy the Port DO \*/

static VOID

DestroyPortDriver(

IN PDEVICE\_OBJECT PortDO)

{

PPORT\_DEVICE\_EXTENSION DeviceExtension;

PCLASS\_DEVICE\_EXTENSION ClassDeviceExtension;

PCLASS\_DRIVER\_EXTENSION DriverExtension;

KEVENT Event;

PIRP Irp;

IO\_STATUS\_BLOCK IoStatus;

KIRQL OldIrql;

NTSTATUS Status;

TRACE\_(CLASS\_NAME, "Destroying PortDO %p\n", PortDO);

DeviceExtension = (PPORT\_DEVICE\_EXTENSION)PortDO->DeviceExtension;

ClassDeviceExtension = DeviceExtension->ClassDO->DeviceExtension;

DriverExtension = IoGetDriverObjectExtension(PortDO->DriverObject, PortDO->DriverObject);

/\* Send IOCTL\_INTERNAL\_\*\_DISCONNECT \*/

KeInitializeEvent(&Event, NotificationEvent, FALSE);

Irp = IoBuildDeviceIoControlRequest(

IOCTL\_INTERNAL\_KEYBOARD\_DISCONNECT,

PortDO,

NULL, 0,

NULL, 0,

TRUE, &Event, &IoStatus);

if (Irp)

{

Status = IoCallDriver(PortDO, Irp);

if (Status == STATUS\_PENDING)

KeWaitForSingleObject(&Event, Suspended, KernelMode, FALSE, NULL);

}

/\* Remove from ClassDeviceExtension->ListHead list \*/

KeAcquireSpinLock(&ClassDeviceExtension->ListSpinLock, &OldIrql);

RemoveEntryList(&DeviceExtension->ListEntry);

KeReleaseSpinLock(&ClassDeviceExtension->ListSpinLock, OldIrql);

/\* Remove entry from HKEY\_LOCAL\_MACHINE\HARDWARE\DEVICEMAP\[DeviceBaseName] \*/

RtlDeleteRegistryValue(

RTL\_REGISTRY\_DEVICEMAP,

DriverExtension->DeviceBaseName.Buffer,

ClassDeviceExtension->DeviceName);

if (DeviceExtension->LowerDevice)

IoDetachDevice(DeviceExtension->LowerDevice);

ObDereferenceObject(PortDO);

if (!DriverExtension->ConnectMultiplePorts && DeviceExtension->ClassDO)

{

ExFreePoolWithTag(ClassDeviceExtension->PortData, CLASS\_TAG);

ExFreePoolWithTag((PVOID)ClassDeviceExtension->DeviceName, CLASS\_TAG);

IoDeleteDevice(DeviceExtension->ClassDO);

}

IoDeleteDevice(PortDO);

}

/\*

Добавление устройства

Процедура обрабатывает запрос на добавление устройства, и само добавление

\*/

static NTSTATUS NTAPI

ClassAddDevice(

IN PDRIVER\_OBJECT DriverObject,

IN PDEVICE\_OBJECT Pdo)

{

PCLASS\_DRIVER\_EXTENSION DriverExtension;

PDEVICE\_OBJECT Fdo = NULL;

PPORT\_DEVICE\_EXTENSION DeviceExtension = NULL;

NTSTATUS Status;

TRACE\_(CLASS\_NAME, "ClassAddDevice called. Pdo = 0x%p\n", Pdo);

DriverExtension = IoGetDriverObjectExtension(DriverObject, DriverObject);

if (Pdo == NULL)

/\* We may get a NULL Pdo at the first call as we're a legacy driver. Ignore it \*/

return STATUS\_SUCCESS;

/\* Create new device object \*/

Status = IoCreateDevice(

DriverObject,

sizeof(PORT\_DEVICE\_EXTENSION),

NULL,

Pdo->DeviceType,

Pdo->Characteristics & FILE\_DEVICE\_SECURE\_OPEN ? FILE\_DEVICE\_SECURE\_OPEN : 0,

FALSE,

&Fdo);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "IoCreateDevice() failed with status 0x%08lx\n", Status);

goto cleanup;

}

IoSetStartIoAttributes(Fdo, TRUE, TRUE);

DeviceExtension = (PPORT\_DEVICE\_EXTENSION)Fdo->DeviceExtension;

RtlZeroMemory(DeviceExtension, sizeof(PORT\_DEVICE\_EXTENSION));

DeviceExtension->Common.IsClassDO = FALSE;

DeviceExtension->DeviceObject = Fdo;

DeviceExtension->PnpState = dsStopped;

Status = IoAttachDeviceToDeviceStackSafe(Fdo, Pdo, &DeviceExtension->LowerDevice);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "IoAttachDeviceToDeviceStackSafe() failed with status 0x%08lx\n", Status);

goto cleanup;

}

if (DeviceExtension->LowerDevice->Flags & DO\_POWER\_PAGABLE)

Fdo->Flags |= DO\_POWER\_PAGABLE;

if (DeviceExtension->LowerDevice->Flags & DO\_BUFFERED\_IO)

Fdo->Flags |= DO\_BUFFERED\_IO;

if (DeviceExtension->LowerDevice->Flags & DO\_DIRECT\_IO)

Fdo->Flags |= DO\_DIRECT\_IO;

if (DriverExtension->ConnectMultiplePorts)

DeviceExtension->ClassDO = DriverExtension->MainClassDeviceObject;

else

{

/\* We need a new class device object for this Fdo \*/

Status = CreateClassDeviceObject(

DriverObject,

&DeviceExtension->ClassDO);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "CreateClassDeviceObject() failed with status 0x%08lx\n", Status);

goto cleanup;

}

}

Status = ConnectPortDriver(Fdo, DeviceExtension->ClassDO);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "ConnectPortDriver() failed with status 0x%08lx\n", Status);

goto cleanup;

}

Fdo->Flags &= ~DO\_DEVICE\_INITIALIZING;

/\* Register interface ; ignore the error (if any) as having

\* a registred interface is not so important... \*/

Status = IoRegisterDeviceInterface(

Pdo,

&GUID\_DEVINTERFACE\_KEYBOARD,

NULL,

&DeviceExtension->InterfaceName);

if (!NT\_SUCCESS(Status))

DeviceExtension->InterfaceName.Length = 0;

return STATUS\_SUCCESS;

cleanup:

if (Fdo)

DestroyPortDriver(Fdo);

return Status;

}

/\*

Если IRP запрос находящийся в очереди отменяется, диспетчер в/в выполняет удаление запрос

из очереди и уведомляет о этом драйвер, по средствам вызова данной процедуры

\*/

static VOID NTAPI

ClassCancelRoutine(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

PCLASS\_DEVICE\_EXTENSION ClassDeviceExtension = DeviceObject->DeviceExtension;

KIRQL OldIrql;

BOOLEAN wasQueued = FALSE;

TRACE\_(CLASS\_NAME, "ClassCancelRoutine(DeviceObject %p, Irp %p)\n", DeviceObject, Irp);

ASSERT(ClassDeviceExtension->Common.IsClassDO);

IoReleaseCancelSpinLock(Irp->CancelIrql);

KeAcquireSpinLock(&ClassDeviceExtension->SpinLock, &OldIrql);

if (ClassDeviceExtension->PendingIrp == Irp)

{

ClassDeviceExtension->PendingIrp = NULL;

wasQueued = TRUE;

}

KeReleaseSpinLock(&ClassDeviceExtension->SpinLock, OldIrql);

if (wasQueued)

{

Irp->IoStatus.Status = STATUS\_CANCELLED;

Irp->IoStatus.Information = 0;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

}

else

{

DPRINT1("Cancelled IRP is not pending. Race condition?\n");

}

}

/\*

Помещает покет IRP в очередь и сообщает о этом диспетчеру в/в

+ пакет, который полностью обработан помечаестся как обработанный, и

инициализируется обработка следующего пакета из очереди.(комент на англ язе от автора)

\*/

static NTSTATUS

HandleReadIrp(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp,

BOOLEAN IsInStartIo)

{

PCLASS\_DEVICE\_EXTENSION DeviceExtension = DeviceObject->DeviceExtension;

NTSTATUS Status;

KIRQL OldIrql;

TRACE\_(CLASS\_NAME, "HandleReadIrp(DeviceObject %p, Irp %p)\n", DeviceObject, Irp);

ASSERT(DeviceExtension->Common.IsClassDO);

if (DeviceExtension->InputCount > 0)

{

SIZE\_T NumberOfEntries;

NumberOfEntries = MIN(

DeviceExtension->InputCount,

IoGetCurrentIrpStackLocation(Irp)->Parameters.Read.Length / sizeof(KEYBOARD\_INPUT\_DATA));

Status = FillEntries(

DeviceObject,

Irp,

DeviceExtension->PortData,

NumberOfEntries);

if (NT\_SUCCESS(Status))

{

if (DeviceExtension->InputCount > NumberOfEntries)

{

RtlMoveMemory(

&DeviceExtension->PortData[0],

&DeviceExtension->PortData[NumberOfEntries],

(DeviceExtension->InputCount - NumberOfEntries) \* sizeof(KEYBOARD\_INPUT\_DATA));

}

DeviceExtension->InputCount -= NumberOfEntries;

Irp->IoStatus.Information = NumberOfEntries \* sizeof(KEYBOARD\_INPUT\_DATA);

}

/\* Go to next packet and complete this request \*/

Irp->IoStatus.Status = Status;

(VOID)IoSetCancelRoutine(Irp, NULL);

IoCompleteRequest(Irp, IO\_KEYBOARD\_INCREMENT);

DeviceExtension->PendingIrp = NULL;

}

else

{

IoAcquireCancelSpinLock(&OldIrql);

if (Irp->Cancel)

{

DeviceExtension->PendingIrp = NULL;

Status = STATUS\_CANCELLED;

}

else

{

//конкретно здесь пакет попадает в очередь

IoMarkIrpPending(Irp);

DeviceExtension->PendingIrp = Irp;

(VOID)IoSetCancelRoutine(Irp, ClassCancelRoutine);

Status = STATUS\_PENDING;

}

IoReleaseCancelSpinLock(OldIrql);

}

return Status;

}

/\*

Обработка запроса сервиса PNP

Драйверу посылается уведомление об обнаружение устройства,

данная процедура создаёт объект устройство, описывающий

добавленное устройство

Или же при отсоединение устройства прорабатывает

\*/

static NTSTATUS NTAPI

ClassPnp(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

PPORT\_DEVICE\_EXTENSION DeviceExtension = DeviceObject->DeviceExtension;

PIO\_STACK\_LOCATION IrpSp = IoGetCurrentIrpStackLocation(Irp);

OBJECT\_ATTRIBUTES ObjectAttributes;

IO\_STATUS\_BLOCK Iosb;

NTSTATUS Status;

switch (IrpSp->MinorFunction)

{

//включение устроства

case IRP\_MN\_START\_DEVICE:

Status = ForwardIrpAndWait(DeviceObject, Irp);

if (NT\_SUCCESS(Status))

{

InitializeObjectAttributes(&ObjectAttributes,

&DeviceExtension->InterfaceName,

OBJ\_CASE\_INSENSITIVE | OBJ\_KERNEL\_HANDLE,

NULL,

NULL);

//здесь устройство открывается, подобно файлу, на чтение

Status = ZwOpenFile(&DeviceExtension->FileHandle,

FILE\_READ\_DATA,

&ObjectAttributes,

&Iosb,

0,

0);

if (!NT\_SUCCESS(Status))

DeviceExtension->FileHandle = NULL;

}

else

DeviceExtension->FileHandle = NULL;

Irp->IoStatus.Status = Status;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return Status;

//остановка устройства

case IRP\_MN\_STOP\_DEVICE:

if (DeviceExtension->FileHandle)

{

//закрывается файл-устройство

ZwClose(DeviceExtension->FileHandle);

DeviceExtension->FileHandle = NULL;

}

Status = STATUS\_SUCCESS;

break;

//удаление устройства, подобно как остановка

case IRP\_MN\_REMOVE\_DEVICE:

if (DeviceExtension->FileHandle)

{

ZwClose(DeviceExtension->FileHandle);

DeviceExtension->FileHandle = NULL;

}

IoSkipCurrentIrpStackLocation(Irp);

Status = IoCallDriver(DeviceExtension->LowerDevice, Irp);

DestroyPortDriver(DeviceObject);

return Status;

default:

Status = Irp->IoStatus.Status;

break;

}

Irp->IoStatus.Status = Status;

if (NT\_SUCCESS(Status) || Status == STATUS\_NOT\_SUPPORTED)

{

IoSkipCurrentIrpStackLocation(Irp);

return IoCallDriver(DeviceExtension->LowerDevice, Irp);

}

else

{

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return Status;

}

}

/\*

драйвер умеет создавать очереди запросов при помощи системных средства

если драйвер когда-то раньше отложил выполнение запроса, то после того

как диспетчер в/в вызывает IoStartPacket вызывается данная процедура

\*/

static VOID NTAPI

ClassStartIo(

IN PDEVICE\_OBJECT DeviceObject,

IN PIRP Irp)

{

PCLASS\_DEVICE\_EXTENSION DeviceExtension = DeviceObject->DeviceExtension;

KIRQL OldIrql;

TRACE\_(CLASS\_NAME, "ClassStartIo(DeviceObject %p, Irp %p)\n", DeviceObject, Irp);

ASSERT(DeviceExtension->Common.IsClassDO);

KeAcquireSpinLock(&DeviceExtension->SpinLock, &OldIrql);

HandleReadIrp(DeviceObject, Irp, TRUE);

KeReleaseSpinLock(&DeviceExtension->SpinLock, OldIrql);

}

/\*

Данная функция ищет подобный драйвер, который имеется в системе

и однако регистрирует наш драйвер в системе

\*/

static VOID NTAPI

SearchForLegacyDrivers(

IN PDRIVER\_OBJECT DriverObject,

IN PVOID Context, /\* PCLASS\_DRIVER\_EXTENSION \*/

IN ULONG Count)

{

UNICODE\_STRING DeviceMapKeyU = RTL\_CONSTANT\_STRING(L"\\REGISTRY\\MACHINE\\HARDWARE\\DEVICEMAP");

PCLASS\_DRIVER\_EXTENSION DriverExtension;

UNICODE\_STRING PortBaseName = { 0, 0, NULL };

PKEY\_VALUE\_BASIC\_INFORMATION KeyValueInformation = NULL;

OBJECT\_ATTRIBUTES ObjectAttributes;

HANDLE hDeviceMapKey = (HANDLE)-1;

HANDLE hPortKey = (HANDLE)-1;

ULONG Index = 0;

ULONG Size, ResultLength;

NTSTATUS Status;

TRACE\_(CLASS\_NAME, "SearchForLegacyDrivers(%p %p %lu)\n",

DriverObject, Context, Count);

if (Count != 1)

return;

DriverExtension = (PCLASS\_DRIVER\_EXTENSION)Context;

/\* Create port base name, by replacing Class by Port at the end of the class base name \*/

//

Status = DuplicateUnicodeString(

RTL\_DUPLICATE\_UNICODE\_STRING\_NULL\_TERMINATE,

&DriverExtension->DeviceBaseName,

&PortBaseName);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "DuplicateUnicodeString() failed with status 0x%08lx\n", Status);

goto cleanup;

}

PortBaseName.Length -= (sizeof(L"Class") - sizeof(UNICODE\_NULL));

RtlAppendUnicodeToString(&PortBaseName, L"Port");

/\* Allocate memory \*/

Size = sizeof(KEY\_VALUE\_BASIC\_INFORMATION) + MAX\_PATH;

KeyValueInformation = ExAllocatePoolWithTag(PagedPool, Size, CLASS\_TAG);

if (!KeyValueInformation)

{

WARN\_(CLASS\_NAME, "ExAllocatePoolWithTag() failed\n");

Status = STATUS\_NO\_MEMORY;

goto cleanup;

}

/\* Open HKEY\_LOCAL\_MACHINE\HARDWARE\DEVICEMAP \*/

InitializeObjectAttributes(&ObjectAttributes, &DeviceMapKeyU, OBJ\_KERNEL\_HANDLE | OBJ\_CASE\_INSENSITIVE, NULL, NULL);

Status = ZwOpenKey(&hDeviceMapKey, 0, &ObjectAttributes);

if (Status == STATUS\_OBJECT\_NAME\_NOT\_FOUND)

{

INFO\_(CLASS\_NAME, "HKLM\\HARDWARE\\DEVICEMAP is non-existent\n");

Status = STATUS\_SUCCESS;

goto cleanup;

}

else if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "ZwOpenKey() failed with status 0x%08lx\n", Status);

goto cleanup;

}

/\* Open sub key \*/

InitializeObjectAttributes(&ObjectAttributes, &PortBaseName, OBJ\_KERNEL\_HANDLE | OBJ\_CASE\_INSENSITIVE, hDeviceMapKey, NULL);

Status = ZwOpenKey(&hPortKey, KEY\_QUERY\_VALUE, &ObjectAttributes);

if (Status == STATUS\_OBJECT\_NAME\_NOT\_FOUND)

{

INFO\_(CLASS\_NAME, "HKLM\\HARDWARE\\DEVICEMAP\\%wZ is non-existent\n", &PortBaseName);

Status = STATUS\_SUCCESS;

goto cleanup;

}

else if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "ZwOpenKey() failed with status 0x%08lx\n", Status);

goto cleanup;

}

/\* Read each value name \*/

while (ZwEnumerateValueKey(hPortKey, Index++, KeyValueBasicInformation, KeyValueInformation, Size, &ResultLength) == STATUS\_SUCCESS)

{

UNICODE\_STRING PortName;

PDEVICE\_OBJECT PortDeviceObject = NULL;

PFILE\_OBJECT FileObject = NULL;

PortName.Length = PortName.MaximumLength = (USHORT)KeyValueInformation->NameLength;

PortName.Buffer = KeyValueInformation->Name;

/\* Open the device object pointer \*/

Status = IoGetDeviceObjectPointer(&PortName, FILE\_READ\_ATTRIBUTES, &FileObject, &PortDeviceObject);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "IoGetDeviceObjectPointer(%wZ) failed with status 0x%08lx\n", &PortName, Status);

continue;

}

INFO\_(CLASS\_NAME, "Legacy driver found\n");

//Добавление устройства, вызов процедуры добавления устройства

Status = ClassAddDevice(DriverObject, PortDeviceObject);

if (!NT\_SUCCESS(Status))

{

/\* FIXME: Log the error \*/

WARN\_(CLASS\_NAME, "ClassAddDevice() failed with status 0x%08lx\n", Status);

}

ObDereferenceObject(FileObject);

}

cleanup:

if (KeyValueInformation != NULL)

ExFreePoolWithTag(KeyValueInformation, CLASS\_TAG);

if (hDeviceMapKey != (HANDLE)-1)

ZwClose(hDeviceMapKey);

if (hPortKey != (HANDLE)-1)

ZwClose(hPortKey);

}

/\*

\* Standard DriverEntry method.

\*/

/\*

DriverEntry регистрирует драйвер в системе

\*/

NTSTATUS NTAPI

DriverEntry(

IN PDRIVER\_OBJECT DriverObject,

IN PUNICODE\_STRING RegistryPath)

{

PCLASS\_DRIVER\_EXTENSION DriverExtension;

ULONG i;

NTSTATUS Status;

//выделение объекта "драйвер"

Status = IoAllocateDriverObjectExtension(

DriverObject,

DriverObject,

sizeof(CLASS\_DRIVER\_EXTENSION),

(PVOID\*)&DriverExtension);

//если неуспешно вернуть ошибку

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "IoAllocateDriverObjectExtension() failed with status 0x%08lx\n", Status);

return Status;

}

RtlZeroMemory(DriverExtension, sizeof(CLASS\_DRIVER\_EXTENSION));

Status = DuplicateUnicodeString(

RTL\_DUPLICATE\_UNICODE\_STRING\_NULL\_TERMINATE,

RegistryPath,

&DriverExtension->RegistryPath);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "DuplicateUnicodeString() failed with status 0x%08lx\n", Status);

return Status;

}

Status = ReadRegistryEntries(RegistryPath, DriverExtension);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "ReadRegistryEntries() failed with status 0x%08lx\n", Status);

return Status;

}

if (DriverExtension->ConnectMultiplePorts == 1)

{

Status = CreateClassDeviceObject(

DriverObject,

&DriverExtension->MainClassDeviceObject);

if (!NT\_SUCCESS(Status))

{

WARN\_(CLASS\_NAME, "CreateClassDeviceObject() failed with status 0x%08lx\n", Status);

return Status;

}

}

DriverObject->DriverExtension->AddDevice = ClassAddDevice;//процедура на добавление устройства

DriverObject->DriverUnload = DriverUnload; //данная функция вызовется при выгрузке драйвера

//если всё в порядке, то в объект "драйвер" записываются callback функций

for (i = 0; i <= IRP\_MJ\_MAXIMUM\_FUNCTION; i++)

DriverObject->MajorFunction[i] = IrpStub;

DriverObject->MajorFunction[IRP\_MJ\_CREATE] = ClassCreate;

DriverObject->MajorFunction[IRP\_MJ\_CLOSE] = ClassClose;

DriverObject->MajorFunction[IRP\_MJ\_CLEANUP] = ClassCleanup;

DriverObject->MajorFunction[IRP\_MJ\_READ] = ClassRead;

DriverObject->MajorFunction[IRP\_MJ\_PNP] = ClassPnp;

DriverObject->MajorFunction[IRP\_MJ\_DEVICE\_CONTROL] = ClassDeviceControl;

DriverObject->MajorFunction[IRP\_MJ\_INTERNAL\_DEVICE\_CONTROL] = ForwardIrpAndForget;

DriverObject->DriverStartIo = ClassStartIo;

/\* We will detect the legacy devices later \*/

/\*

Регистрация драйвера отложена, выполняется процедурой SearchForLegacyDrivers

та процедура ищет драйвер "старого образца" и регистрирует новый драйвер

\*/

IoRegisterDriverReinitialization(

DriverObject,

SearchForLegacyDrivers,

DriverExtension);

return STATUS\_SUCCESS;

}