Основы ядерной архитектуры ОС Windows

Часть 2

Структура драйвера

- Работа драйвера начинается с вызова функции DriverEntry, соответствующей начальной точке входа в исполняемый модуль драйвера, которая прописана в его PE-заголовке.
- Задача DriverEntry – инициализация работы драйвера. Если она завершается успешно – функция завершается с кодом STATUS_SUCCESS (=0), после чего драйвер остается в памяти ядра. При любом другом коде завершения драйвер выгружается из памяти.
- Работа драйвера после загрузки по сути определяется тем, какие callback-функции зарегистрированы драйвером при инициализации:
- Диспетчерские функции: массив точек входа DriverObject->MajorFunction[], вызываемых для обработки запросов, направляемых устройствам драйвера.
- Обработчики прерываний (ISR), отложенных вызовов (DPC).
- Подмена адресов функций обработки вызовов системных сервисов.
- Функции, определяемые типом программного интерфейса задействованного при инициализации (например NDIS).
- В ОС начиная с Vista – драйверы должны быть подписаны. Механизм проверки подписи может быть отключен при запуске ОС.

ограничения

- Проблема при попытке написания кода драйвера на C++: функция main() – не первая функция с которой начинает работать код (инициализация памяти, вызов конструкторов глобальных экземпляров классов).
- В коде драйвера – жесткие ограничения на библиотечные функции. Разрешенные к вызову – можно вызывать только функции ядра, некоторые типы драйверов имеют ограниченный набор функций, разрешенных к вызову, чтобы соответствовать некоторой спецификации – например NDIS.
- Для всех доступных для вызова ядерных функций документация фиксирует жесткие условия, в которых они могут быть вызваны (контекст потока – случайный или нет, уровень IRQL, с которого может быть осуществлен вызов). Кроме того – ограничения на доступ к памяти (не всегда можно работать с paged-памятью) в зависимости от условий работы.
- В ранних версиях ОС было ограничение (запрет) на работу с инструкциями FPU, возникающее вследствие несохранения контекста FPU при переключении user->kernel mode

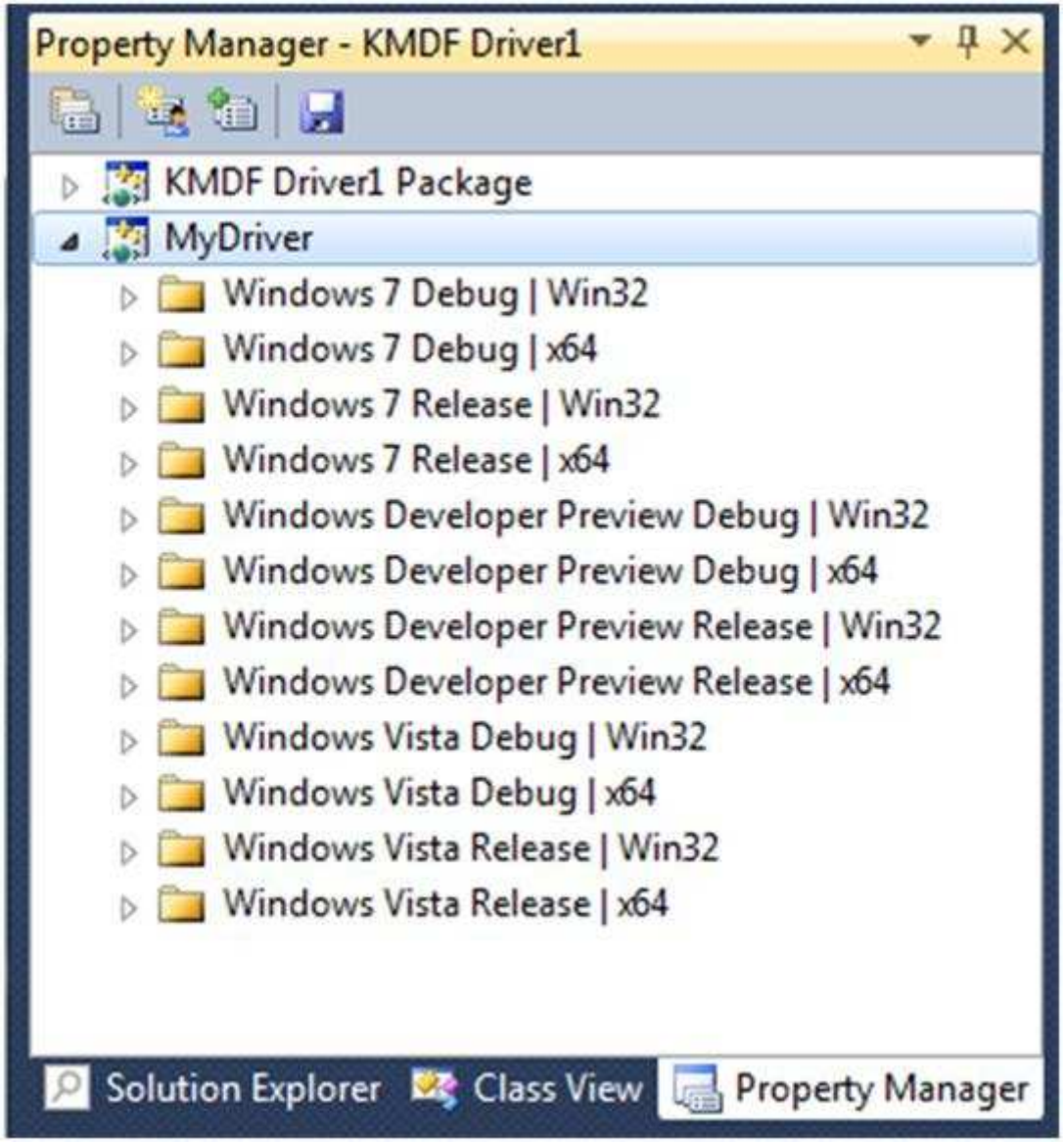
Среда разработки

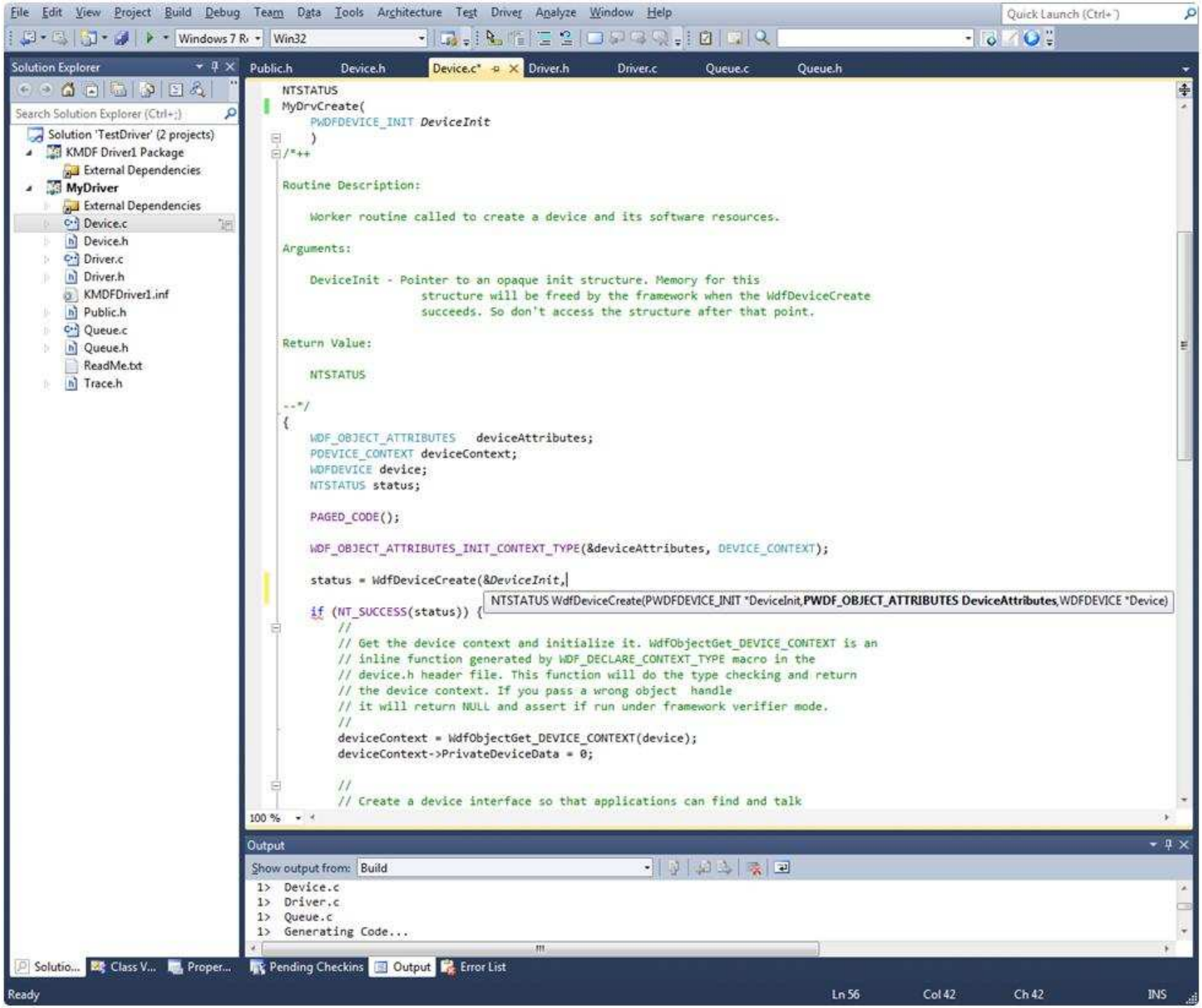
- Пакет DDK/WDK, поддерживает все ОС начиная с WinXP
- VisualStudio
- Для интеграции WDK с VisualStudio – сторонний продукт VisualDDK, без него – утилита build, работающая в режиме командной строки
- В пакете WDK для Windows 8 впервые официально заявлена возможность использования IDE (Integrated Development Environment) VisualStudio, но с рядом ограничений (в частности не поддерживается разработка драйверов для XP, предположительно поддерживает только WDF/KMDF-драйверы, полнофункциональная поддержка всех типов драйверов – по прежнему официально только через build)

Многоуровневая модель сокрытия знаний о функционировании ядра ОС

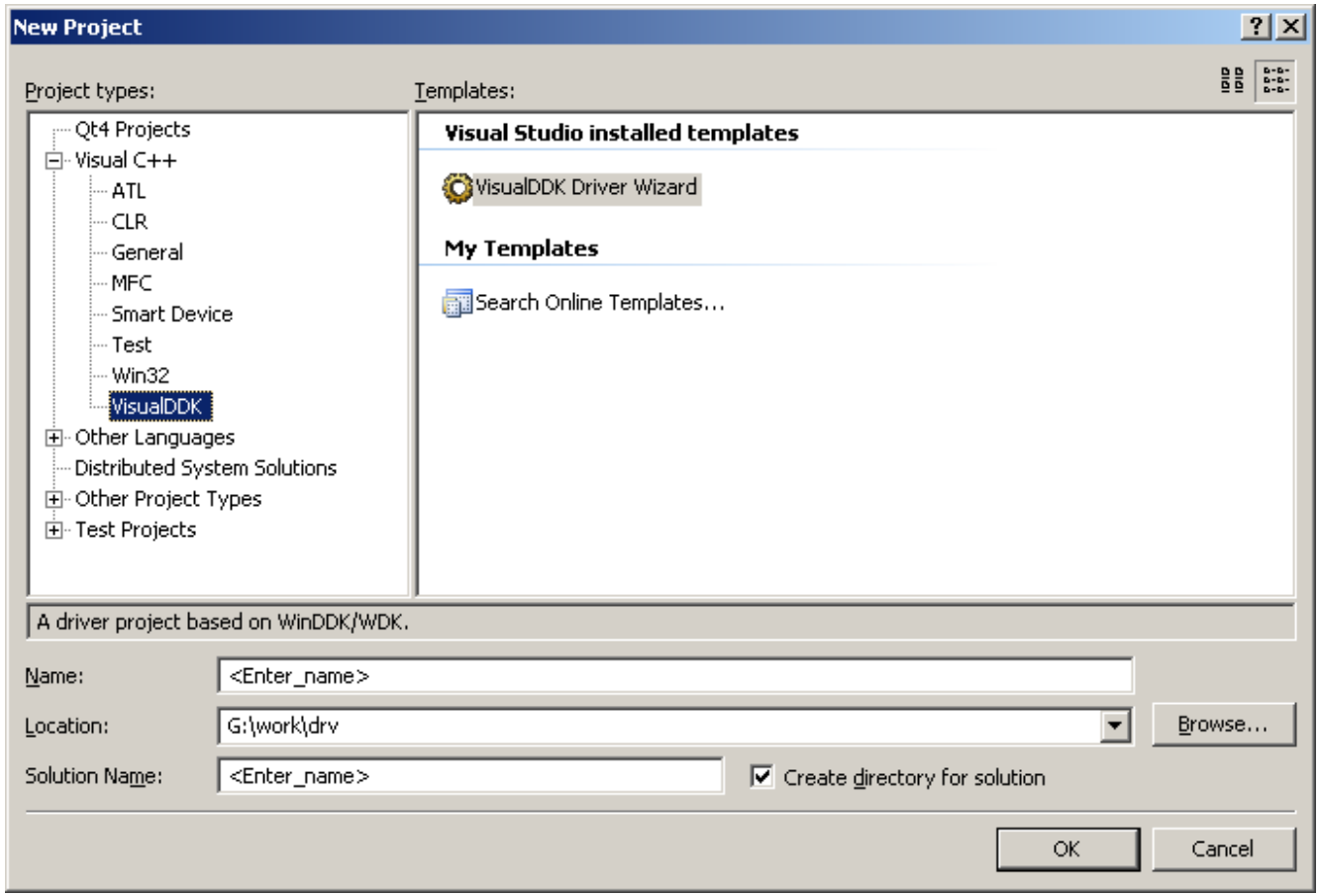
- WDF/KMDF – WDM – Legacy Drivers

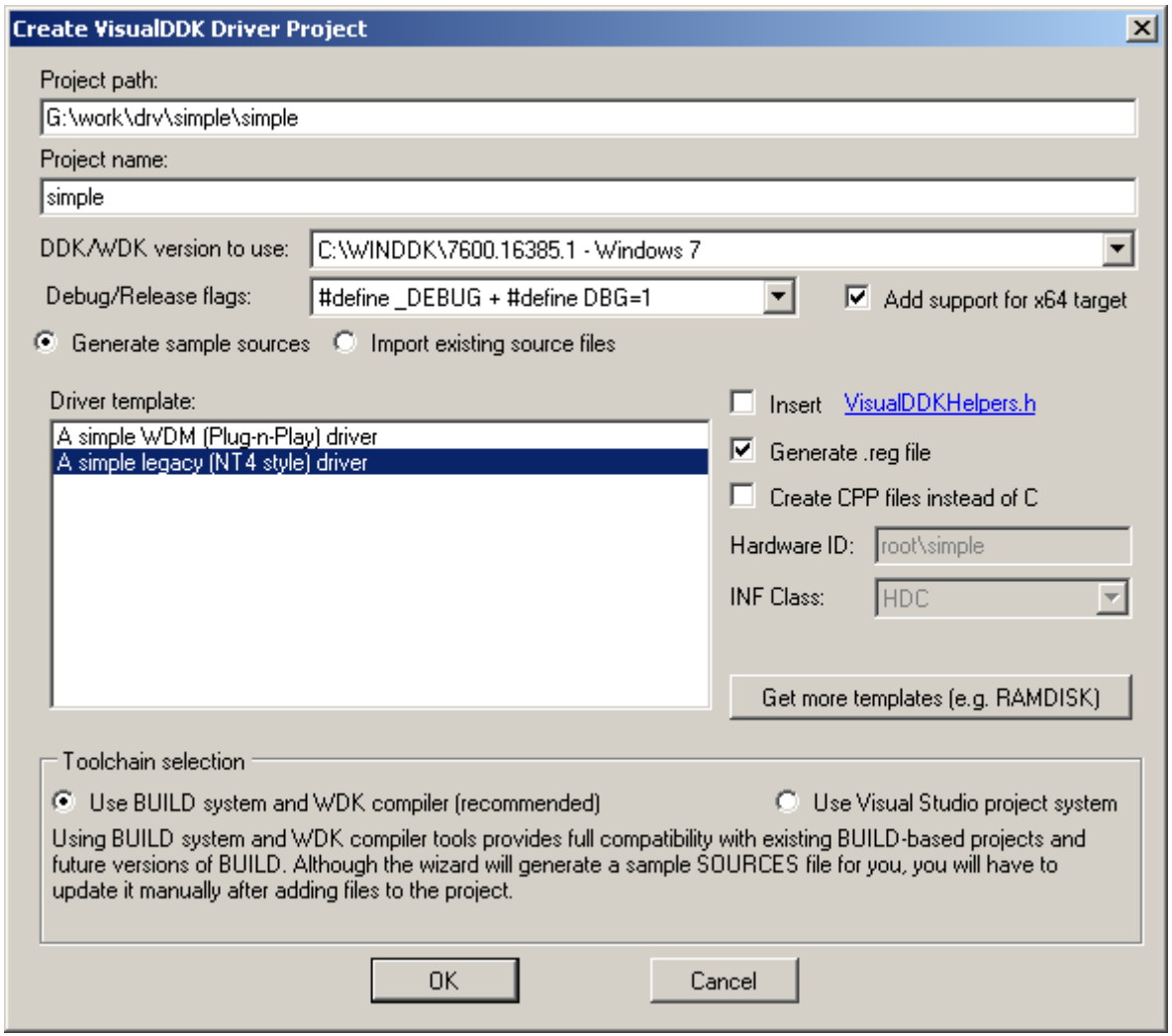
- Legacy Drivers – по сути способ написания ядерного драйвера «с нуля», опираясь только на знания архитектуры ОС. По сути архитектура таких драйверов не претерпела изменений с момента появления ОС Windows NT 3.51. Такой драйвер будет работать на всех современных Windows
- WDM – Windows Driver Model – модель драйверов, изначально разработанная как способ реализации универсальных драйверов для кардинально различающихся линеек ОС: Win9x и WinNT, путем предоставления драйверу для работы некоторого программного окружения, по сути соответствующего поведению ядра ОС линейки Windows NT . Данная архитектура стала стандартом для драйверов всех последующих ОС данной линейки
- WDF/KMDF – фреймворк, программная оболочка, скрывающая реальное устройство ядра ОС для облегчения разработки драйверов, но может стать «тормозом» при разработке «продвинутых» драйверов




File Edit View Project Build Debug Team Data Tools Architecture Test Driver Analyze Window Help
Quick Launch (Ctrl+`)
Windows 7 R
Win32
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'TestDriver' (2 projects)
KMDF Driver1 Package
External Dependencies
MyDriver
External Dependencies
Device.c
Device.h
Driver.c
Driver.h
KMDFDriver1.inf
Public.h
Queue.c
Queue.h
ReadMe.txt
Trace.h
Public.h
Device.h
Device.c*
Driver.h
Driver.c
Queue.c
Queue.h
NTSTATUS
MyDrvCreate(
PWDFDEVICE_INIT DeviceInit
)
/*++
Routine Description:
Worker routine called to create a device and its software resources.
Arguments:
DeviceInit - Pointer to an opaque init structure. Memory for this
structure will be freed by the framework when the WdfDeviceCreate
succeeds. So don't access the structure after that point.
Return Value:
NTSTATUS
--*/
{
WDF_OBJECT_ATTRIBUTES deviceAttributes;
PDEVICE_CONTEXT deviceContext;
WDFDEVICE device;
NTSTATUS status;
PAGED_CODE();
WDF_OBJECT_ATTRIBUTES_INIT_CONTEXT_TYPE(&deviceAttributes, DEVICE_CONTEXT);
status = WdfDeviceCreate(&DeviceInit,
NTSTATUS WdfDeviceCreate(PWDFDEVICE_INIT *DeviceInit,PWDF_OBJECT_ATTRIBUTES DeviceAttributes,WDFDEVICE *Device)
if (NT_SUCCESS(status)) {
//
// Get the device context and initialize it. WdfObjectGet_DEVICE_CONTEXT is an
// inline function generated by WDF_DECLARE_CONTEXT_TYPE macro in the
// device.h header file. This function will do the type checking and return
// the device context. If you pass a wrong object handle
// it will return NULL and assert if run under framework verifier mode.
//
deviceContext = WdfObjectGet_DEVICE_CONTEXT(device);
deviceContext->PrivateDeviceData = 0;
//
// Create a device interface so that applications can find and talk
100 %
Output
Show output from: Build
1> Device.c
1> Driver.c
1> Queue.c
1> Generating Code...
Solutio...
Class V...
Proper...
Pending Checkins
Output
Error List
Ready
Ln 56
Col 42
Ch 42
INS

Интеграции WDK с VisualStudio –VisualDDK

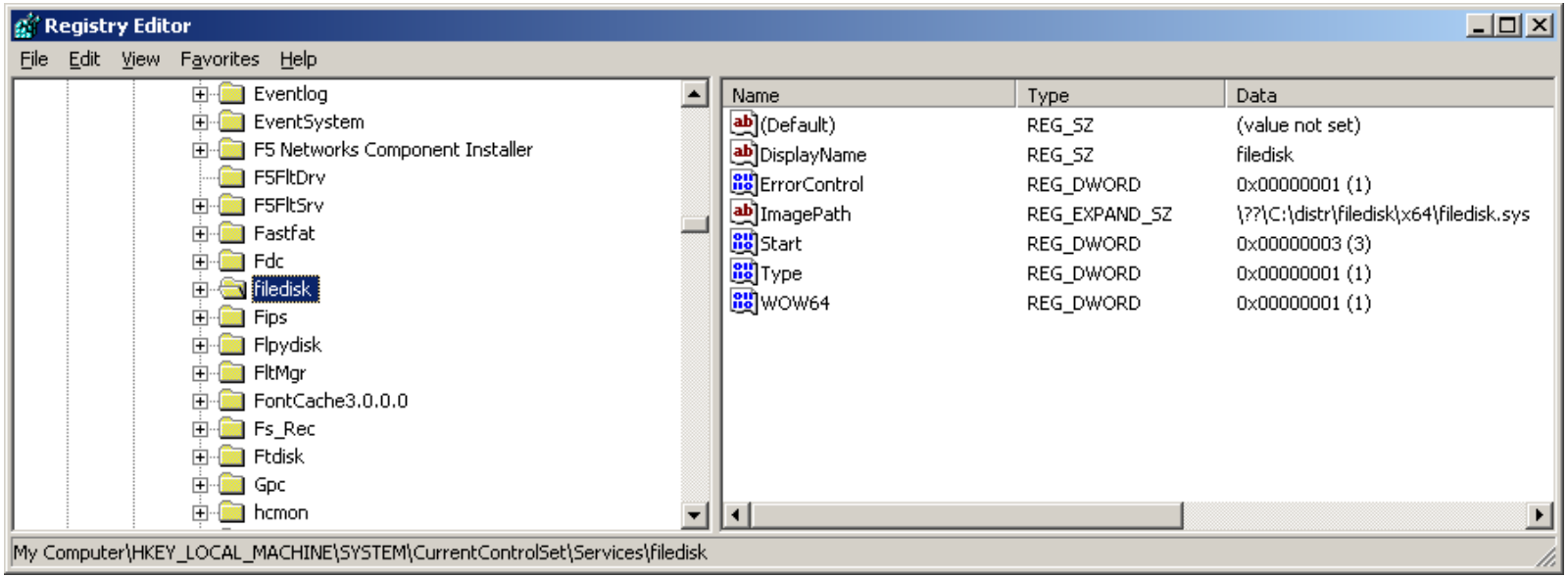




Код простейшего драйвера: ничего делать не умеет, после загрузки остается в памяти ядра и не может быть оттуда выгружен никаким другим способом кроме перезагрузки компьютера.

```
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0501
#endif

#include <ntddk.h>

NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING  RegistryPath)
{
            DbgPrint("simple: DriverEntry, regpath=%ws\n", RegistryPath->Buffer);
            return STATUS_SUCCESS;
}
```

Регистрация драйвера в реестре. Обратить внимание на формат пути в ImagePath



Зарегистрированный драйвер может быть запущен и остановлен с помощью консольных команд net start имя_сервиса / net stop имя_сервиса

Программно драйвер управляется с помощью WinAPI-функций диспетчера сервисов (SCM - ServiceControlManager): CreateService/DeleteService/StartService/ControlService

- Для обеспечения возможности выгрузки драйвера он должен реализовывать обработчик функции выгрузки `DriverObject->DriverUnload`
- Задача этой функции – очистка всех ресурсов, выделенных при работе драйвера

```
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0501
#endif

#include <ntddk.h>

void OnUnload(IN PDRIVER_OBJECT DriverObject)
{
            UNICODE_STRING Win32Device;
            DbgPrint("simple: OnUnload\n");
}

NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING  RegistryPath)
{
            DbgPrint("simple: DriverEntry, regpath=%ws\n", RegistryPath->Buffer);
            DriverObject->DriverUnload = OnUnload;
            return STATUS_SUCCESS;
}
```

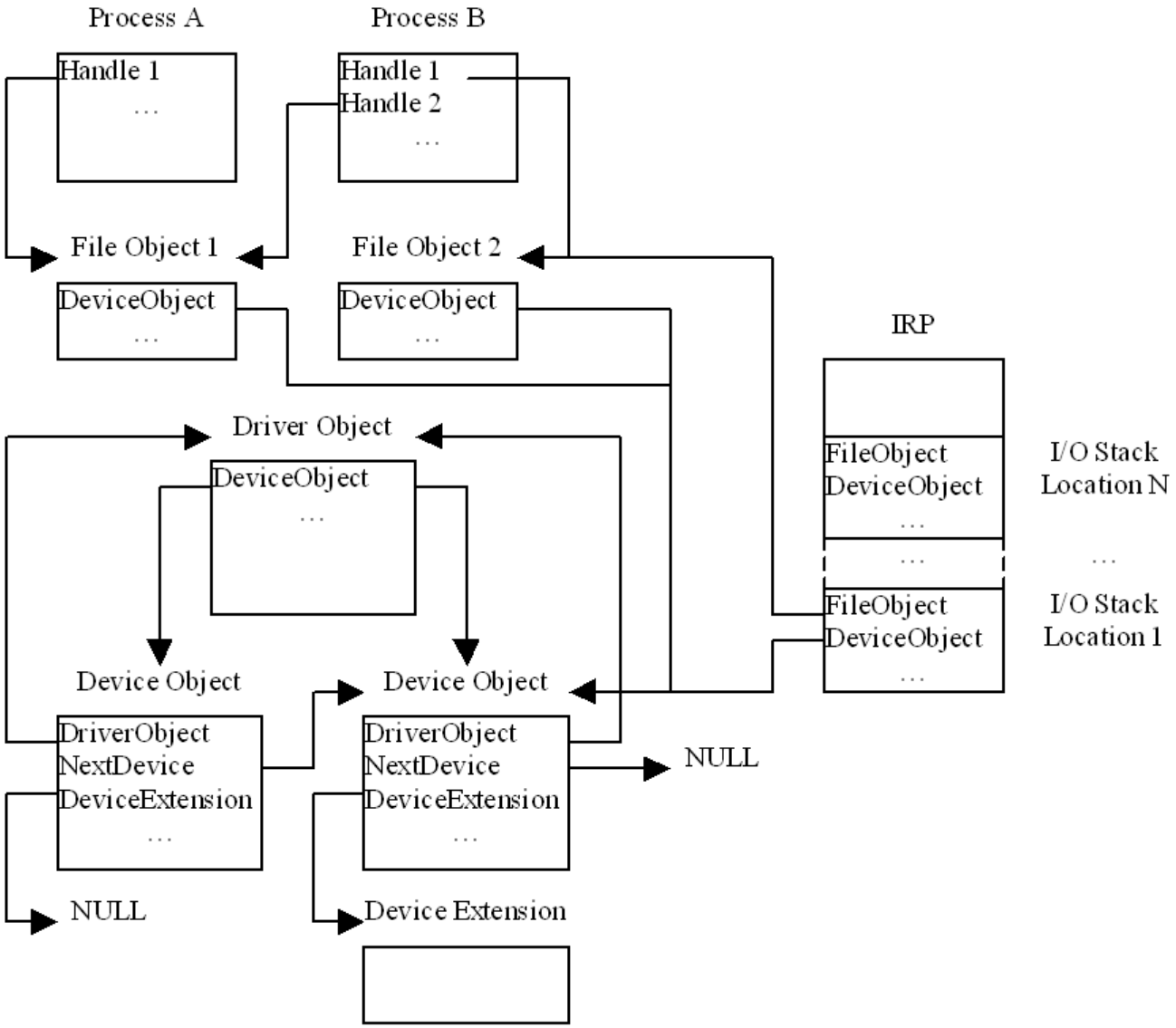
Взаимодействие с драйвером обычно осуществляется через создаваемые им устройства, которым направляются запросы в/в в формате IRP-пакетов

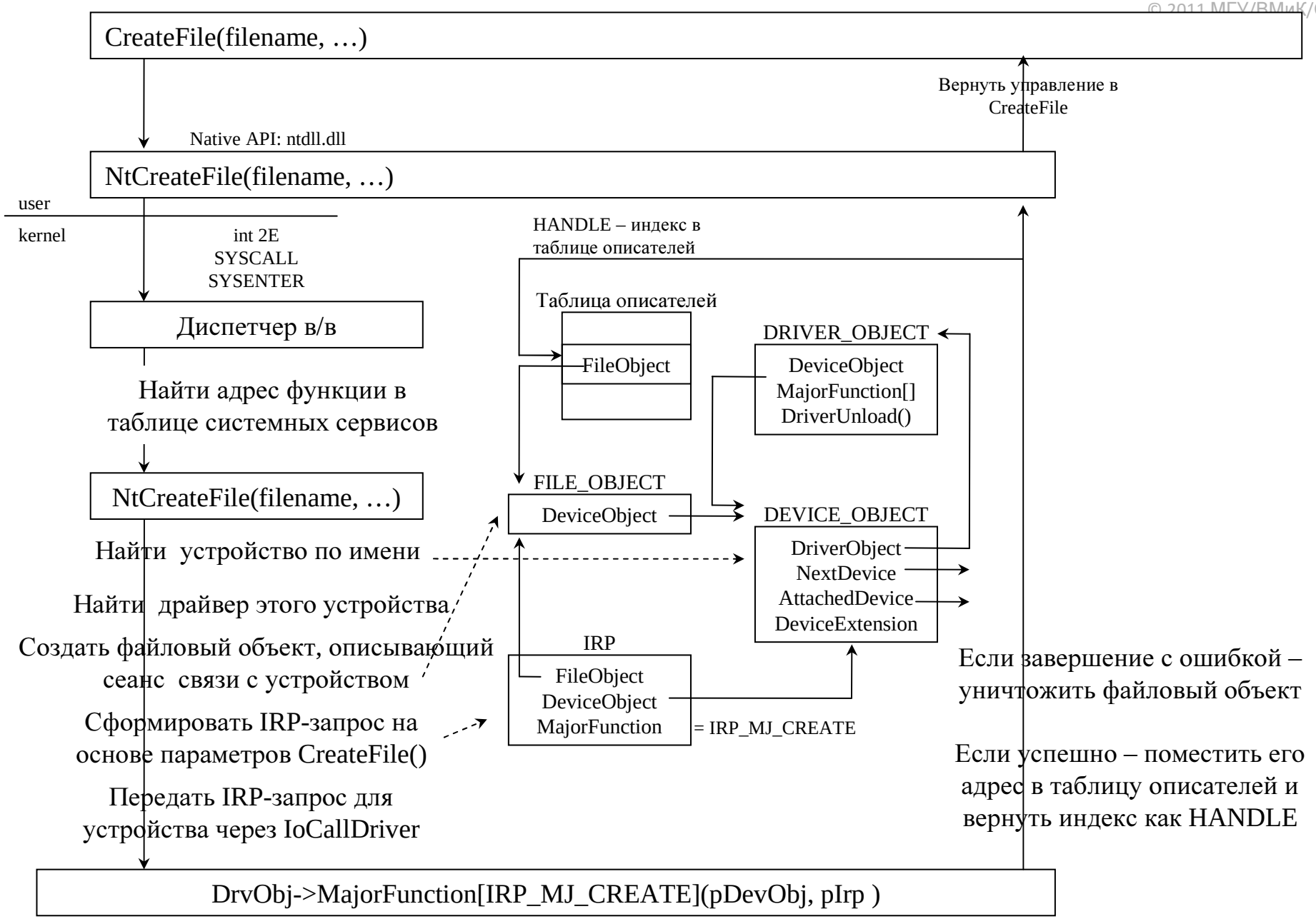
```
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0501
#endif
#include <ntddk.h>
void OnUnload(IN PDRIVER_OBJECT DriverObject)
{
                DbgPrint("simple: OnUnload\n");
                IoDeleteDevice(DriverObject->DeviceObject);
}
NTSTATUS OnCreate(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{
                DbgPrint("simple: OnCreate\n");
                Irp->IoStatus.Status = STATUS_SUCCESS;
                Irp->IoStatus.Information = 0;
                IoCompleteRequest(Irp, IO_NO_INCREMENT);
                return STATUS_SUCCESS;
}
NTSTATUS OnClose(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{
                DbgPrint("simple: OnClose\n");
                Irp->IoStatus.Status = STATUS_SUCCESS;
                Irp->IoStatus.Information = 0;
                IoCompleteRequest(Irp, IO_NO_INCREMENT);
                return STATUS_SUCCESS;
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING  RegistryPath)
{
                UNICODE_STRING DeviceName;
                PDEVICE_OBJECT DeviceObject = NULL;
                NTSTATUS status;
                DbgPrint("simple: DriverEntry, regpath=%ws\n", RegistryPath->Buffer);
                RtlInitUnicodeString(&DeviceName,L"\\Device\\simple0");
                DriverObject->MajorFunction[IRP_MJ_CREATE] = OnCreate;
                DriverObject->MajorFunction[IRP_MJ_CLOSE] = OnClose;
                DriverObject->DriverUnload = OnUnload;
                status = IoCreateDevice(DriverObject,
                                0,
                                &DeviceName,
                                FILE_DEVICE_UNKNOWN,
                                0,
                                FALSE,
                                &DeviceObject);
                if (!NT_SUCCESS(status))
                                return status;
                if (!DeviceObject)
                                return STATUS_UNEXPECTED_IO_ERROR;
                DeviceObject->Flags |= DO_DIRECT_IO;
                DeviceObject->AlignmentRequirement = FILE_WORD_ALIGNMENT;

                return STATUS_SUCCESS;
}
```

```
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0501
#endif
#include <ntddk.h>
void OnUnload(IN PDRIVER_OBJECT DriverObject)
{
        ✚       UNICODE_STRING Win32Device;
                DbgPrint("simple: OnUnload\n");
        ✚       RtlInitUnicodeString(&Win32Device,L"\\DosDevices\\simple0");
        ✚       IoDeleteSymbolicLink(&Win32Device);
                IoDeleteDevice(DriverObject->DeviceObject);
}
NTSTATUS OnCreate(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{
                DbgPrint("simple: OnCreate\n");
                Irp->IoStatus.Status = STATUS_SUCCESS;
                Irp->IoStatus.Information = 0;
                IoCompleteRequest(Irp, IO_NO_INCREMENT);
                return STATUS_SUCCESS;
}
NTSTATUS OnClose(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{
                DbgPrint("simple: OnClose\n");
                Irp->IoStatus.Status = STATUS_SUCCESS;
                Irp->IoStatus.Information = 0;
                IoCompleteRequest(Irp, IO_NO_INCREMENT);
                return STATUS_SUCCESS;
}
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject, IN PUNICODE_STRING  RegistryPath)
{
        ✚       UNICODE_STRING DeviceName,Win32Device;
                PDEVICE_OBJECT DeviceObject = NULL;
                NTSTATUS status;
                DbgPrint("simple: DriverEntry, regpath=%ws\n", RegistryPath->Buffer);
                RtlInitUnicodeString(&DeviceName,L"\\Device\\simple0");
        ✚       RtlInitUnicodeString(&Win32Device,L"\\DosDevices\\simple0");
                DriverObject->MajorFunction[IRP_MJ_CREATE] = OnCreate;
                DriverObject->MajorFunction[IRP_MJ_CLOSE] = OnClose;
                DriverObject->DriverUnload = OnUnload;
                status = IoCreateDevice(DriverObject,
                                0,
                                &DeviceName,
                                FILE_DEVICE_UNKNOWN,
                                0,
                                FALSE,
                                &DeviceObject);
                if (!NT_SUCCESS(status))
                                return status;
                if (!DeviceObject)
                                return STATUS_UNEXPECTED_IO_ERROR;
                DeviceObject->Flags |= DO_DIRECT_IO;
                DeviceObject->AlignmentRequirement = FILE_WORD_ALIGNMENT;
        ✚       IoCreateSymbolicLink(&Win32Device, &DeviceName);
                return STATUS_SUCCESS;
}
```

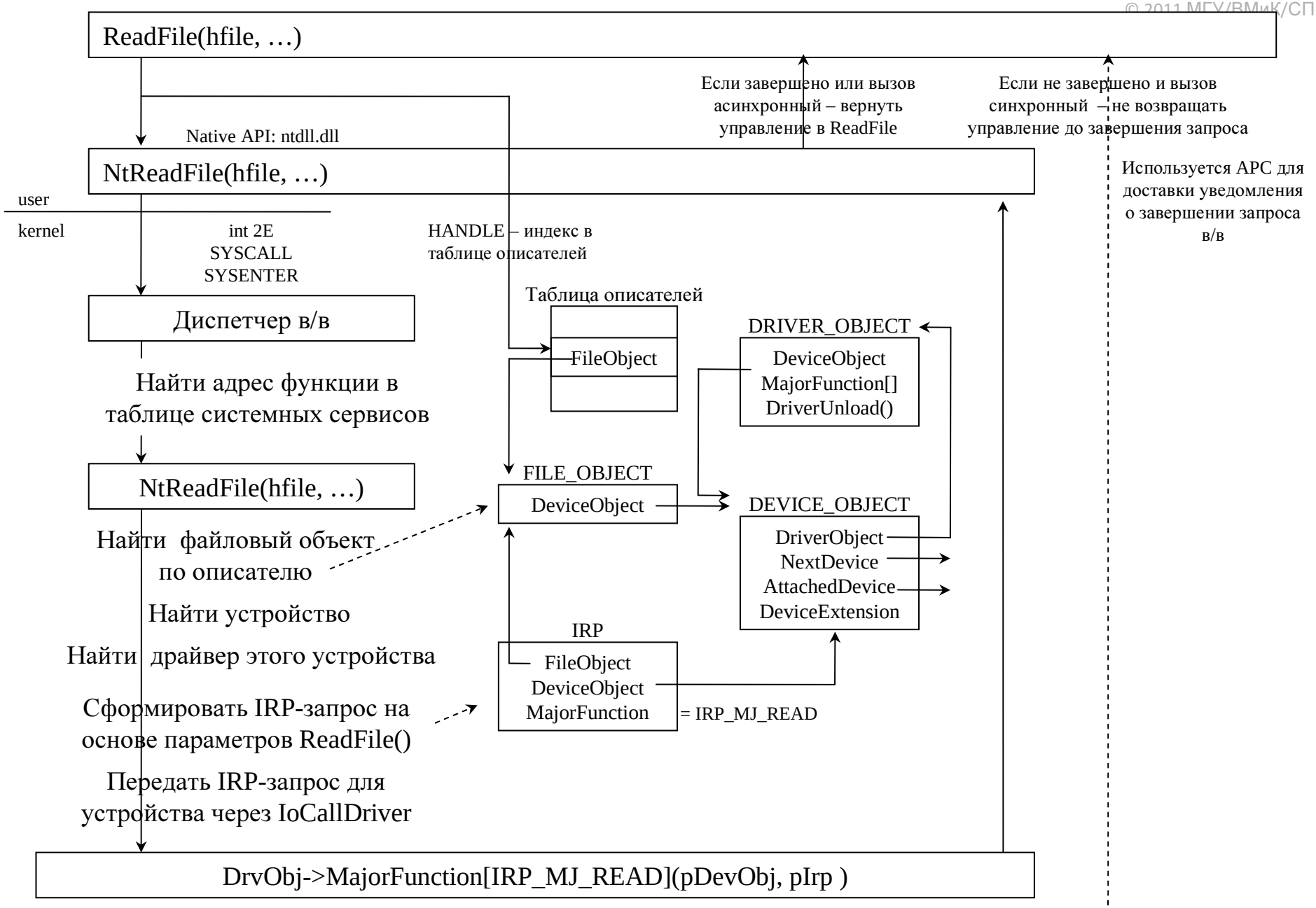
Чтобы к устройству мог обратиться код пользовательского режима, имя этого устройства должно стать видимым в специальной части пространства имен диспетчера объектов. Это достигается путем создания символической ссылки.

Взаимосвязь основных ядерных объектов при прохождении запроса в/в






```
pIrp->IoStatus.Status = STATUS_SUCCESS;
IoCompleteRequest(pIrp, 0);
return STATUS_SUCCESS;
```




```
pIrp->IoStatus.Status = STATUS_SUCCESS;
IoCompleteRequest(pIrp, 0);
return STATUS_SUCCESS;
```

```
//Отложить обработку: сохранить pIrp
IoMarkIrpPending(pIrp);
return STATUS_PENDING;
```

```
//где-то в случайном контексте
IoCompleteRequest(pIrp, 0);
```

Примеры анализа драйвера

- Использование дизассемблера IDA Pro
- Использование среды анализа TREX

Отладочные возможности процессора, особенности в Windows, средства отладки

- Пошаговая отладка: TraceFlag в регистре флагов

- Точки прерывания:
- Модификация отлаживаемого кода – запись инструкции int 3 (код 0xCC)
- Отладочные регистры DR0-DR3 (не более 4 точек прерывания на доступ к памяти r/w/e)

- И при пошаговой отладке, и при использовании отладочных регистров генерируется прерывание int 1

- Конку… …ы:
  прогр… …оля
  флаго… …тности
  кода (…

- Г…
  о…
  п…
  э…
  о…

- Д…
  о… …ь
  выявл… …ограмме
  обраб… …n Handling
  или V…

- Катас… …гладке -
  все су… …контроля
  време…

ИСХОДНЫЙ ТЕКСТ JOHOR CRACK-ME

```
#include <windows.h>

int count;
char str[]="0123456789ABCDEF!";

__declspec(naked) nezumi()
{
        __asm(
        ; //int 03       ; // for SoftICE
        xor eax, eax ; // eax := 0
        mov ebx, fs:[eax]          ; // old SEH
        ;//-[set TF bit]-;
        pushfd  ; // EFLAGS
        xor eax,[eax]    ; // <-- ACCESS VIOLATION
        popfd    ; // EFLAGS := 00244ED7h
        ;//-[TRACE-ZONE]-;
        mov eax, [eax]    ; // <-- ACCESS VIOLATION
        nop      ; // <-- INT 01
        ud2      ; // <-- ILLEGAL INSTRUCTION
        nop      ; // <-- INT 01
        nop      ; // <-- INT 01
        int 03 ; // <-- INT 01
        jmp end_of_line  ; // : -- to exit -->
        ;//-[seh handler]--;
        l1: mov eax, [esp + 04h] ; // *EXCEPTION_RECORD
        l2: mov edx, [esp + 0Ch] ; // EDX -> ContextRecord
        mov eax, [eax]    ; // EXCEPTION_CODE
        cmp eax, 0C000001Dh ; // ILLEGAL INSTRUCTION
        jz x2     ; // X-->
        cmp eax, 080000003h ; // INT 03
        jz x1     ; // -- skip 1 byte -->
        cmp eax, 0C0000005h ; // ACCESS VIOLATION
        jnz set_tf_bit   ; // -- don't skip -->
        x2:inc dword ptr [edx+0B8h] ; // skip one byte
        x1:inc dword ptr [edx+0B8h] ; // skip one byte
        set_tf_bit:      ; // <--X
        cmp dword ptr [edx + 0B8h], offset end_of_line
        jae end_of_handler
                ; // dont set TF-bit _outside_ trace-zone
        or dword ptr [edx+0C0h],100h
                ; // <---- set TF-bit _inside_ trace-zone
        end_of_handler:xor eax,eax
                        ; // EXCEPTION_CONTINUE_SEARCH
        inc [count]      ; // EXCEPTION COUNT
        ret              ; // end of the handler
        ;//-[exit]-;
        count = str[(count>0x10)?0x10:count];
        MessageBox(0,&count,"JohoR",MB_OK);
        ExitProcess(0);
}
```

- **Задача: найти ошибку в этом коде (исправить одну инструкцию)**

СЧАСТЛИВЫЙ ФИНАЛ

Так как же все-таки ломают эти программы? И какими отладчиками? В случае статического кода (к которому относится мой crack-me) проблема решается установкой точки останова за пределами «горячей» зоны, где происходит выброс исключений, с прогоном их на живом процессоре (без пошаговой трассировки). Если же нам жизненно необходимо подсмотреть значение некоторых регистров или ячеек памяти внутри «горячей» зоны — на них устанавливается аппаратная точка останова. Динамический код (упакованный, зашифрованный, самомодифицирующийся) заломать намного сложнее, поскольку нам реально необходимо прогнать его через пошаговый трассировщик, с которым и борется защита. Заметим, весьма эффективно борется. BOCHS (бесплатная виртуальная машина со встроенным отладчиком) — единственный разумный выбор, но, сколько грузится на нем Windows, лучше не говорить.

«Классические» отладчики уровня ОС фактически вытеснены отладочными возможностями симуляторов:

backend: VmWare
frontend: интерфейс GDB (IDA, VisualStudio, …)

QEMU, VirtualPC, BOCHS, AMD SimNow

Сам по себе дизассемблер или отладчик не автоматизирует решение задачи анализа потока данных, если под анализом понимать отслеживание зависимостей между инструкциями (по управлению и данным). Автоматизацией занимаются средства-«надстройки», например:

автоматизация поверх IDA:

Фирма zynamics (www.zynamics.com), средство BinNavi

Автоматизация поверх отладчиков - трассировка:

BitBlaze/TEMU

Valgrind + Avalanche

TREX