Глава 2. Начало работы с инструментами разработчика ядра

Это вторая часть, с первой частью можно ознакомиться здесь:https://ru-sfera.org/threads/windows-kernel-programming-glava-1-osnovnye-momenty-i-arxitektura-jadra.3943/

Тема для обсуждения на форуме:https://ru-sfera.org/threads/windows-kernel-programming-glava-2-nachalo-raboty-s-instrumentami-razrabotchika-jadra.3945/

В этой главе рассматриваются основы, которые необходимы для разработки драйверов.

В этой главе вы установите необходимые инструменты и напишете первый драйвер, который можно загружать и выгружать.

В этой главе:

• Установка инструментов.
• Создание проекта драйвера.
• Описание процедур DriverEntry и Unload.
• Разработка драйвера.
• Простая трассировка.

1)Установка инструментов

В старые времена (до 2012 года) процесс разработки и сборки драйверов включал использование особого инструмента для сборки из комплекта драйверов устройств (DDK).

Нужно-было качать специальный пакет компиляторов, далее в командной строке собирать драйвер, не было никакой интеграции в Visual Studio.

Были некоторые обходные пути, но ни один из них не был ни идеален, ни официально поддержан.

К счастью, начиная с Visual Studio 2012 и Windows Driver Kit 8, Microsoft начала официально поддерживать сборку драйверов с помощью Visual Studio (и msbuild), без необходимости использовать отдельный компилятор и инструменты сборки.

Чтобы начать разработку драйверов, необходимо установить следующие инструменты (в следующем порядке):

• Visual Studio 2017 или 2019 с последними обновлениями.

Убедитесь, что C ++ выбрана во время установки. Обратите внимание, что подойдет любой SKU, включая бесплатная версия Community.
• Windows 10 SDK.
• Windows 10 Driver Kit (WDK).
• Инструменты Sysinternals, необходимы для отладки драйверов, можно скачать бесплатно на http://www.sysinternals.com. Нажмите на Sysinternals Suite слева от этой веб-страницы и загрузите файл Sysinternals Suite. Распакуйте в любую папку, и инструменты готовы к работе.

Быстрый способ убедиться, что шаблоны WDK установлены правильно, - это открыть Visual Studio и выберите «Новый проект» и найдите проекты драйверов, например «Пустой драйвер WDM».

2)Создание проекта драйвера

При наличии вышеуказанных пакетов можно создать новый проект драйвера.

Шаблон, который вы будете использовать в этом разделе «Пустой драйвер WDM».

Рисунок 2-1 показывает, как выглядит диалог New Project для этого типа драйвера в Visual Studio 2017. На рисунке 2-2 показан тот же начальный мастер в Visual Studio 2019.

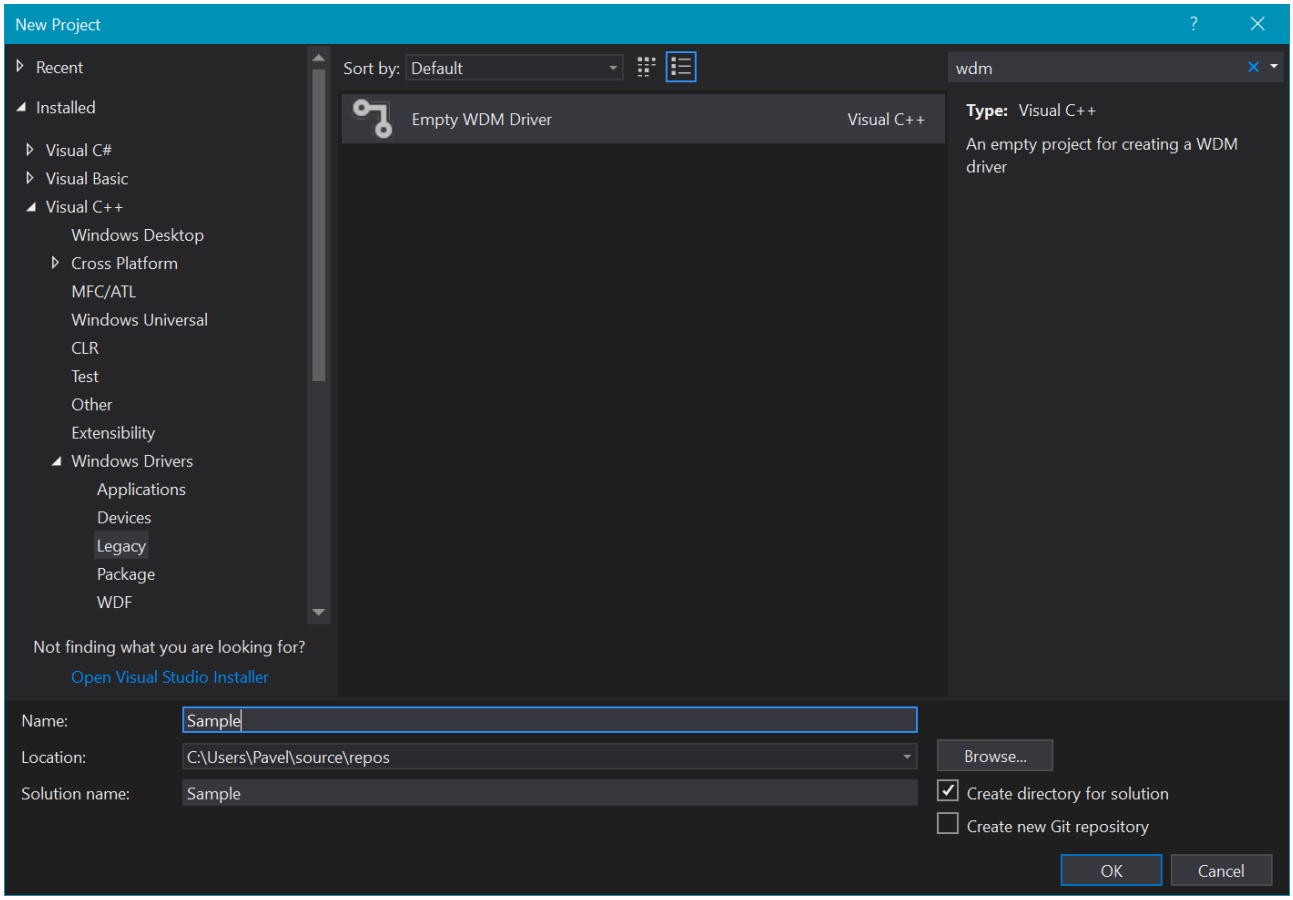

Рисунок 2-1.Новый проект драйвера в Visual Studio 2017

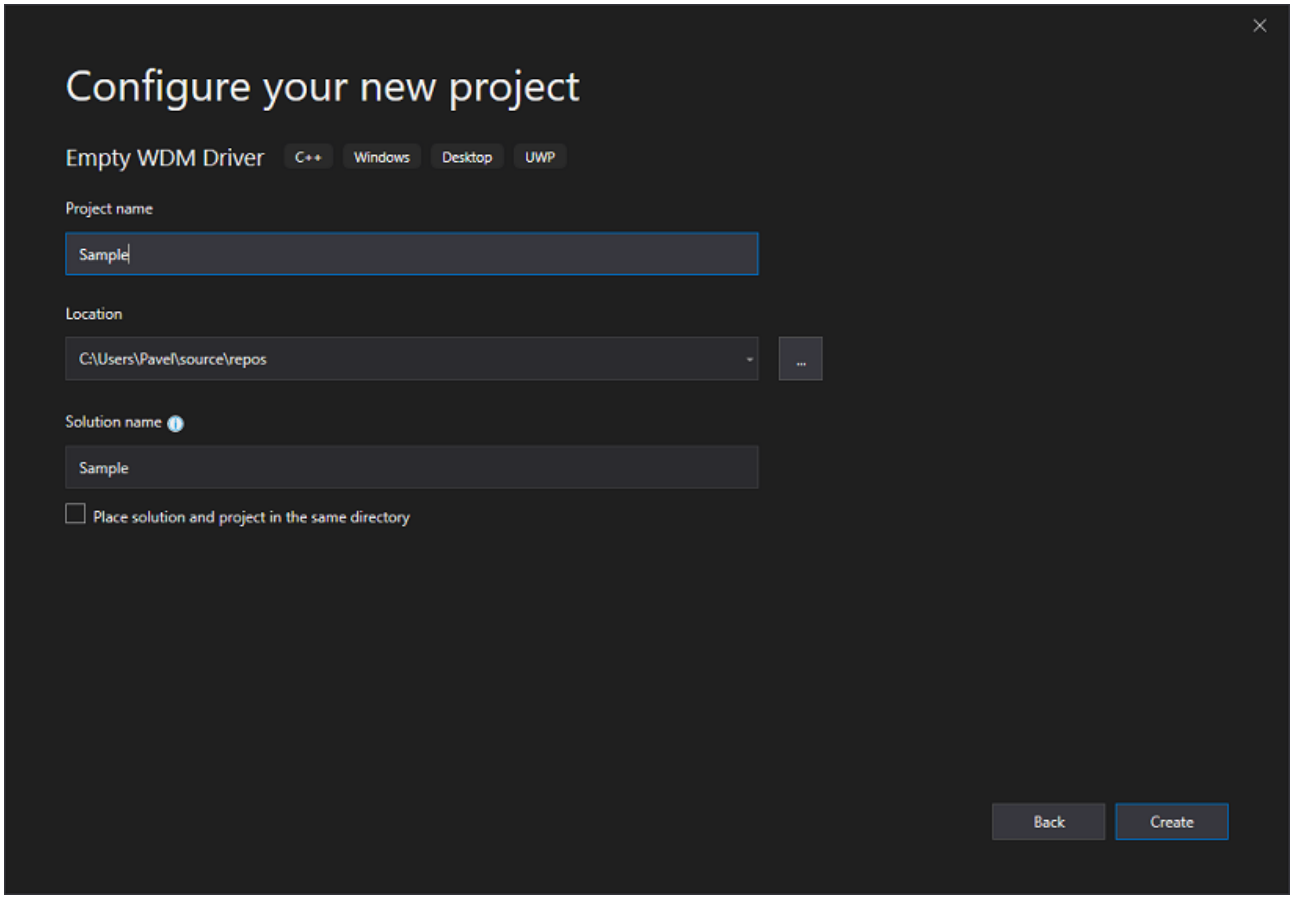

Рисунок 2-2.Новый проект драйвера в Visual Studio 2019

После создания проекта в обозревателе решений отображается один файл - Sample.inf. Вам не нужен этот файл в этом примере, так что просто удалите его.

Теперь пришло время добавить исходный файл. Щелкните правой кнопкой мыши узел «Исходные файлы» в обозревателе решений и выберите Добавить/Новый элемент ... из меню Файл.

Выберите исходный файл C ++ и назовите его Sample.cpp. Нажмите ОК, чтобы создать файл.

2)Функции DriverEntry и Unload

По умолчанию каждый драйвер имеет точку входа DriverEntry.

Это можно считать «main» драйвера, если сопоставить с классическим основным приложением пользовательского режима.

DriverEntry имеет следующий прототип, показанный здесь:

NTSTATUS
DriverEntry*(_In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING RegistryPath);*

_In_ аннотации являются частью языка аннотаций исходного кода (SAL). Эти аннотации прозрачны для компилятора, но предоставляют метаданные, полезные для читателей и инструментов статического анализа.

Мы постараемся максимально использовать их для улучшения ясности кода.

DriverEntry может просто вернуть успешный статус, например так:

```
NTSTATUS
DriverEntry(_In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING RegistryPath) {
return STATUS_SUCCESS;
}
```

Этот код еще не скомпилируется. Во-первых, вам нужно включить заголовок, который имеет определения для типов, присутствующих в DriverEntry.

Например так:

```
#include <ntddk.h>
```

Теперь код имеет больше шансов на компиляцию, но все равно потерпит неудачу.

Одна из причин в том, что по умолчанию компилятор настроен на обработку предупреждений как ошибок, а функция не использует заданные аргументы.

Удаление обрабатывать предупреждения как ошибки не рекомендуется, так как некоторые предупреждения могут быть замаскированными ошибками.

Эти предупреждения могут быть устранены путем полного удаления имен аргументов (или комментирования их), что хорошо для файлов C ++.

Существует еще один классический способ решения этой проблемы, который заключается в использовании макроса UNREFERENCED_PARAMETER:

```
NTSTATUS
DriverEntry(_In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING
RegistryPath) {

UNREFERENCED_PARAMETER(DriverObject);
UNREFERENCED_PARAMETER(RegistryPath);
return STATUS_SUCCESS;

}
```

Вот объявление данного макроса:

```
#define UNREFERENCED_PARAMETER(P)          (P)
```

Сборка проекта теперь компилируется нормально, но вызывает ошибку компоновщика.

Функция DriverEntry должна-быть C-linkage, который не используется по умолчанию при компиляции C ++.

Вот финальная версия успешной сборки драйвера, состоящего только из функции DriverEntry:

```
extern "C"
NTSTATUS
DriverEntry(_In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING
RegistryPath) {

UNREFERENCED_PARAMETER(DriverObject);
UNREFERENCED_PARAMETER(RegistryPath);

return STATUS_SUCCESS;
}
```

В какой-то момент драйвер может быть выгружен.

В то время все, что сделано в функции DriverEntry должно быть отменено.

Невыполнение этого требования приводит к утечке, которую ядро не очистит до следующей перезагрузки.

Драйверы могут иметь процедуру выгрузки, которая автоматически вызывается перед выгрузкой драйвера из памяти.

Указатель на функцию, которая должна выполнится перед выгрузкой из памяти должна быть установлена с помощью элемента DriverUnload объекта драйвера:

DriverObject->DriverUnload = SampleUnload;

Процедура выгрузки принимает объект драйвера (тот же, что передан в DriverEntry).

Поскольку пример нашего драйвера ничего не делает с точки зрения распределения ресурсов в DriverEntry, то и в процедуре Unload ничего не нужно делать, поэтому мы можем пока оставить ее пустой:

```
void SampleUnload(_In_ PDRIVER_OBJECT DriverObject) {
UNREFERENCED_PARAMETER(DriverObject);
}
```

Вот полный код драйвера на данный момент:

```
#include <ntddk.h>

void SampleUnload(_In_ PDRIVER_OBJECT DriverObject) {
	UNREFERENCED_PARAMETER(DriverObject);
}

extern "C"
NTSTATUS
DriverEntry(_In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING
RegistryPath) {

UNREFERENCED_PARAMETER(RegistryPath);
DriverObject->DriverUnload = SampleUnload;

return STATUS_SUCCESS;
}
```

3)Развертывание и запуск драйвера

Теперь у нас есть успешно скомпилированный файл драйвера Sample.sys, давайте установим его в систему, а затем загрузим его.

Обычно лучше устанавливать и загружать драйвер на виртуальную машину, чтобы устранить риск сбоя вашей основной машины.

Установка программного драйвера, как и установка службы в пользовательском режиме, требует вызова сервисного API с правильными аргументами или с использованием существующих инструментов.

Один из известных инструментов для этого программа Sc.exe, это встроенный в Windows инструмент для управления сервисами. Мы будем использовать этот инструмент для установки и затем загрузим драйвер.

Обратите внимание, что установка и загрузка драйверов является привилегированной операцией, обычно разрешено только для администраторов. Откройте командное окно с повышенными правами и введите следующее (последняя часть должна быть указана для вашей системы, где находится файл SYS):

sc create sample type= kernel binPath= c:\dev\sample\x64\debug\sample.sys

Обратите внимание, что между типом и знаком равенства нет пробела, а между знаком равенства и kernel пробел. То же самое касается второй части.

Если все идет хорошо, результат должен указывать на успех. Чтобы проверить установку, вы можете открыть реестр редактор (regedit.exe) и найдите драйвер в **HKLM\System\CurrentControlSet\Services\Sample**.

Рисунок 2-3 показывает снимок экрана редактора реестра после предыдущей команды.

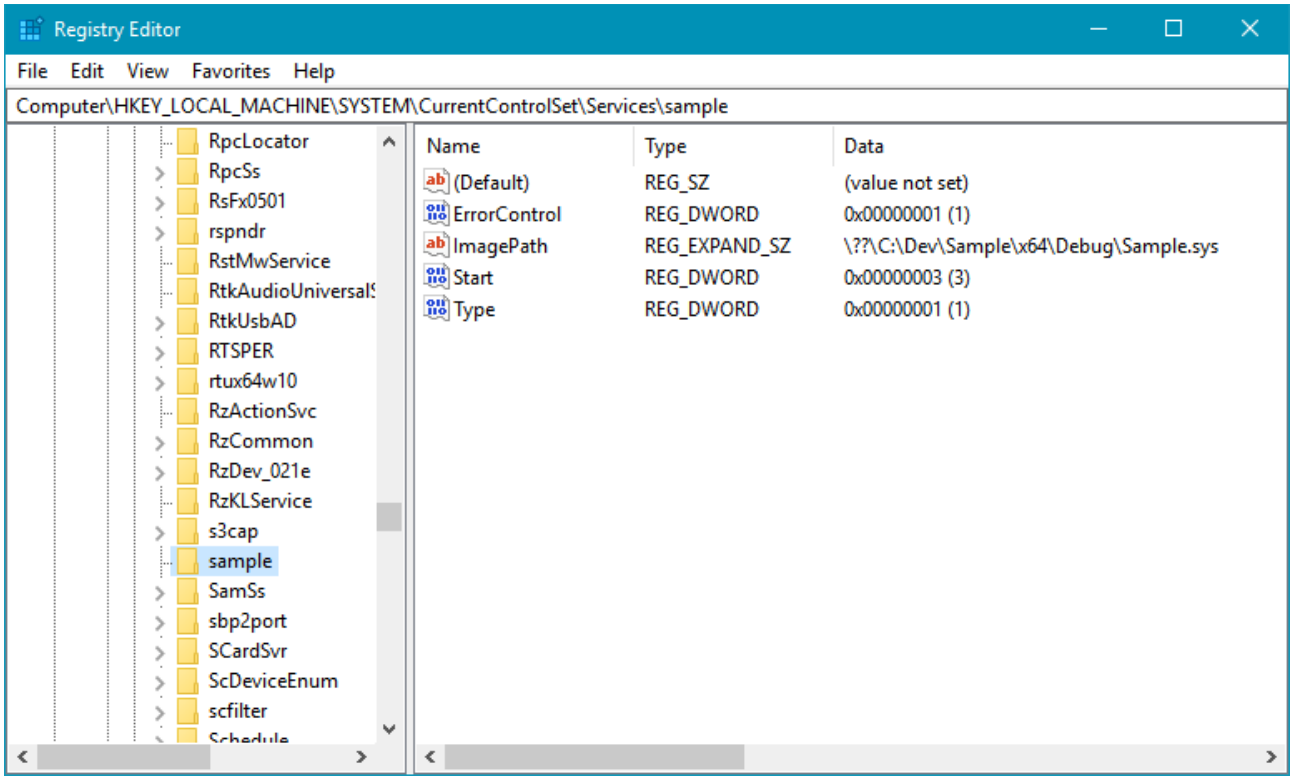

Рисунок 2-3. Снимок экрана редактора реестра

Чтобы загрузить драйвер, мы можем снова использовать инструмент Sc.exe, на этот раз с параметром запуска, который использует API-интерфейс StartService для загрузки драйвера (тот же API, который используется для загрузки служб).

Однако на 64 бит системные драйверы должны быть подписаны, и поэтому обычно следующая команда не будет работать:

sc start sample

Поскольку подписывать драйвер во время разработки неудобно (возможно, даже невозможно, если вы не имете соответствующий сертификат), лучший вариант - перевести систему в тестовый режим.

В этом режиме неподписанные драйверы могут быть загружены без проблем.

С командным окном с повышенными правами тестовый режим может быть включен следующим образом:

bcdedit /set testsigning on

К сожалению, эта команда требует перезагрузки для вступления в силу. После перезагрузки можно стартовать драйвер.

ВАЖНО:

Если вы тестируете в Windows 10 с включенной безопасной загрузкой, нельзя включить тестовый режим.

Это одно из свойств, защиты Secure Boot (также защищен от локальной отладки ядра).

Если вы не можете отключить безопасную загрузку через настройки BIOS, из-за ИТ-политики или по какой-то другой причине, ваш лучший вариант - это тестирование на виртуальной машине.

Существует еще один параметр, который вам может потребоваться указать, если вы собираетесь тестировать драйвер на Windows 10.

В этом случае вы должны установить целевую версию ОС в свойствах проекта.

Диалоговое окно, как показано на рисунке 2-4.

Обратите внимание, что я выбрал все конфигурации и все платформы, чтобы

при переключении конфигураций (Debug / Release) или платформ (x86 / x64 / ARM / ARM64), настройка сохранялась.

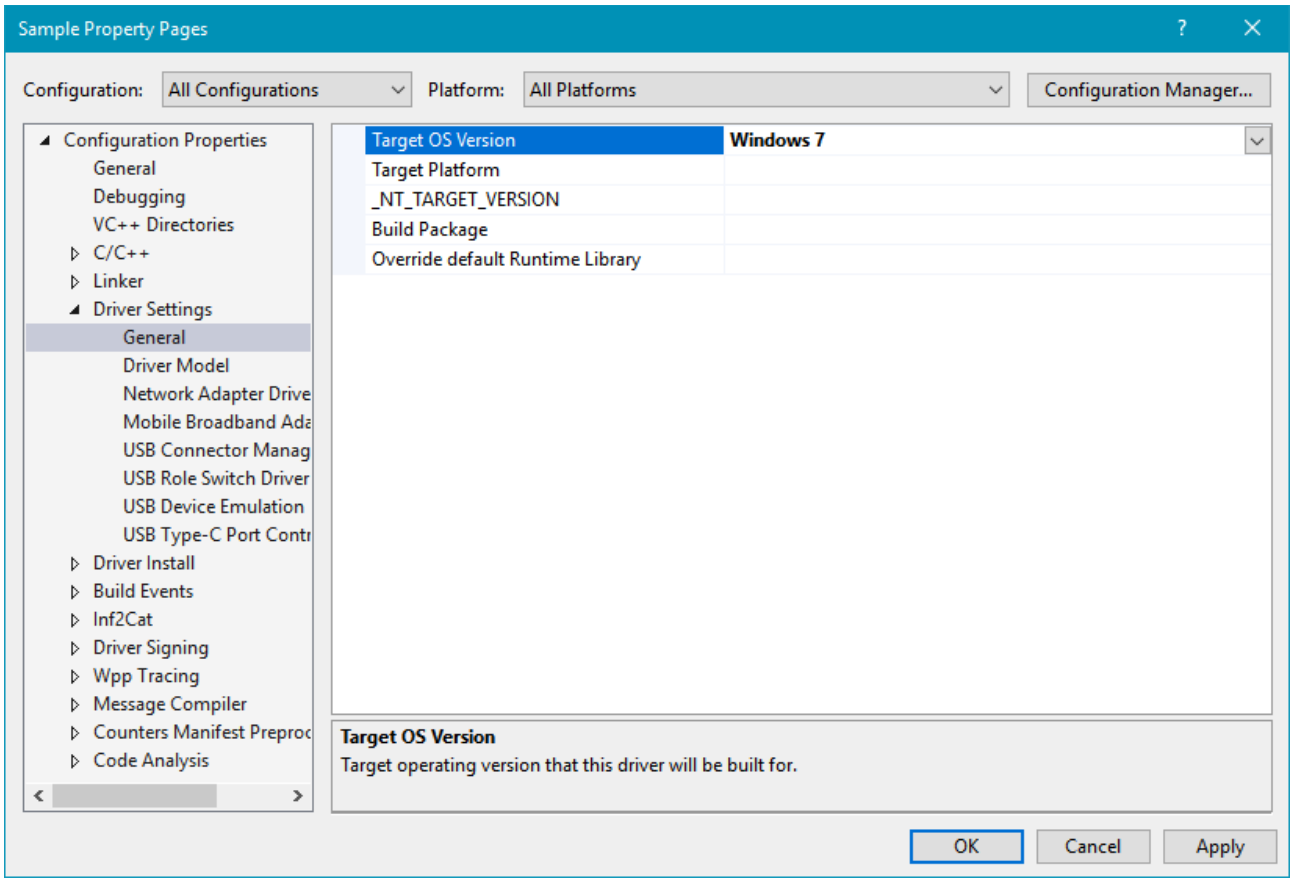

Рисунок 2-4. Настройки целевой системы, под которую будет собираться драйвер

Когда тестовый режим включен и драйвер загружен, вы должны увидеть следующее (sc state sample):

```
SERVICE_NAME: sample
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 0
        FLAGS              :
```

Это означает, что все хорошо, и драйвер загружен. Чтобы это проверить, мы можем открыть Process Explorer и найти Sample.sys в процессе System:

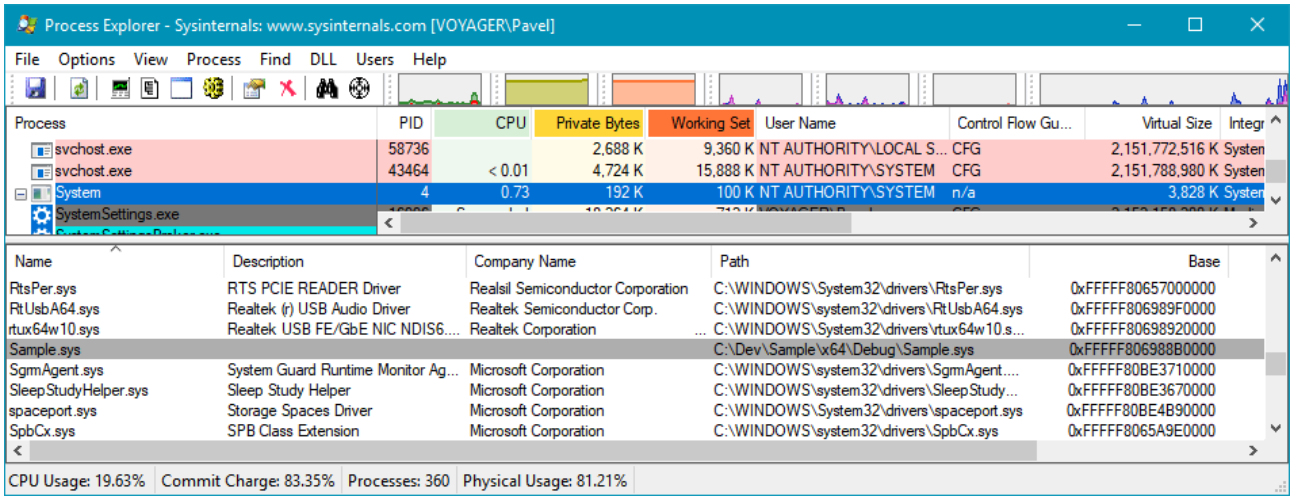

Рисунок 2-5. Скриншет Process Explorer и поиск драйвера Sample

Что-бы выгрузить драйвер, достаточно ввести команду:*sc stop sample.*

Выгрузка драйвера вызывает вызов процедуры Unload, которая в этом драйвере ничего не делает.

Вы можете убедиться, что драйвер действительно выгружен, снова взглянув на Process Explorer.

4)Простая трассировка

Как мы можем точно знать, что процедуры DriverEntry и Unload действительно выполнены ?

Давайте добавим базовое отслеживание этих функций. Драйверы могут использовать макрос KdPrint для вывода текста в стиле printf, который можно просмотреть с помощью отладчика ядра и других инструментов.

KdPrint - это макрос, который компилируется только в Debug,создает и вызывает базовый API ядра DbgPrint.

Пример обновленного драйвера, который использует отладочные выводы:

```
void SampleUnload(_In_ PDRIVER_OBJECT DriverObject) {

UNREFERENCED_PARAMETER(DriverObject);
KdPrint(("Sample driver Unload called\n"));

}

extern "C"
NTSTATUS
DriverEntry(_In_ PDRIVER_OBJECT DriverObject, _In_ PUNICODE_STRING
RegistryPath) {

UNREFERENCED_PARAMETER(RegistryPath);
DriverObject->DriverUnload = SampleUnload;
KdPrint(("Sample driver initialized successfully\n"));

return STATUS_SUCCESS;

}
```

Обратите внимание на двойные скобки при использовании KdPrint. Это необходимо, потому что KdPrint является макросом, но по-видимому, принимает любое количество аргументов, а-ля printf.

Поскольку макросы не могут получить переменное число аргументов, трюк компилятора используется для вызова реальной функции DbgPrint.

Теперь мы должны снова загрузить драйвер и увидеть эти сообщения.

Мы будем использовать отладчик ядра в главе 4, но сейчас мы будем использовать полезный инструмент Sysinternals с именем DebugView.

Перед запуском DebugView вам нужно сделать некоторые приготовления.

Во-первых, начиная с Windows Vista, вывод DbgPrint фактически не генерируется, если в реестре не указано определенное значение.

Вам нужно добавить ключ с именем ***Debug Print Filter*** в ***HKLM\SYSTEM\CurrentControlSet\Control\Session***

Ключ обычно не существует. В этом новом ключе добавьте значение DWORD с именем DEFAULT (это не значение по умолчанию, которое существует в любом ключе) и установите его значение равным 8

Рисунок 2-6 показывает настройку в RegEdit.

К сожалению, вам придется перезагрузить систему чтобы этот параметр вступил в силу.

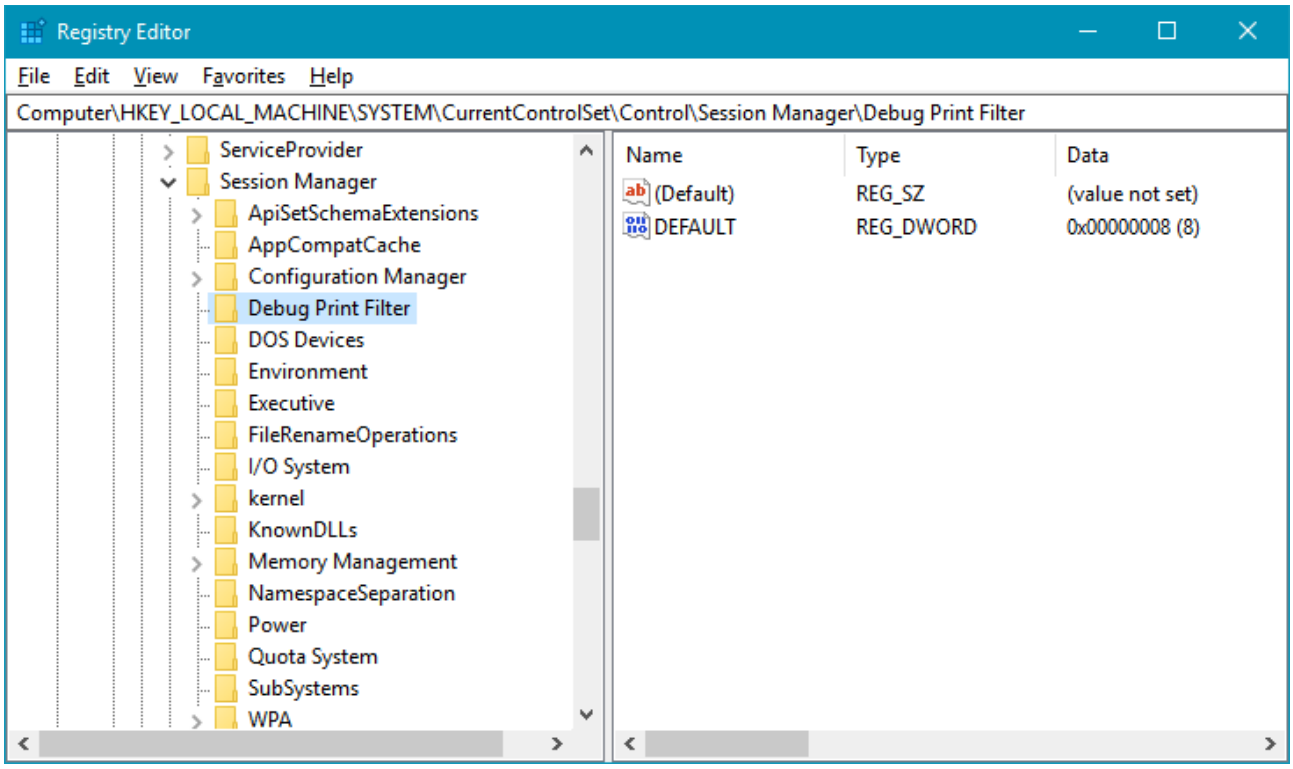

Рисунок 2-6. Включение отображение дебажного вывода в драйвере

После применения этого параметра запустите DebugView (DbgView.exe) с повышенными правами.

В меню «Параметры» убедитесь, что выбрано Capture Kernel (или нажмите Ctrl + K).

Соберите драйвер, если вы этого еще не сделали. Теперь вы можете снова загрузить драйвер.

Вы должны увидеть выходные данные в DebugView, как показано на рисунке 2-7.

Третья строка вывода получена из другого драйвера и не имеет ничего общего с нашим примером драйвера.

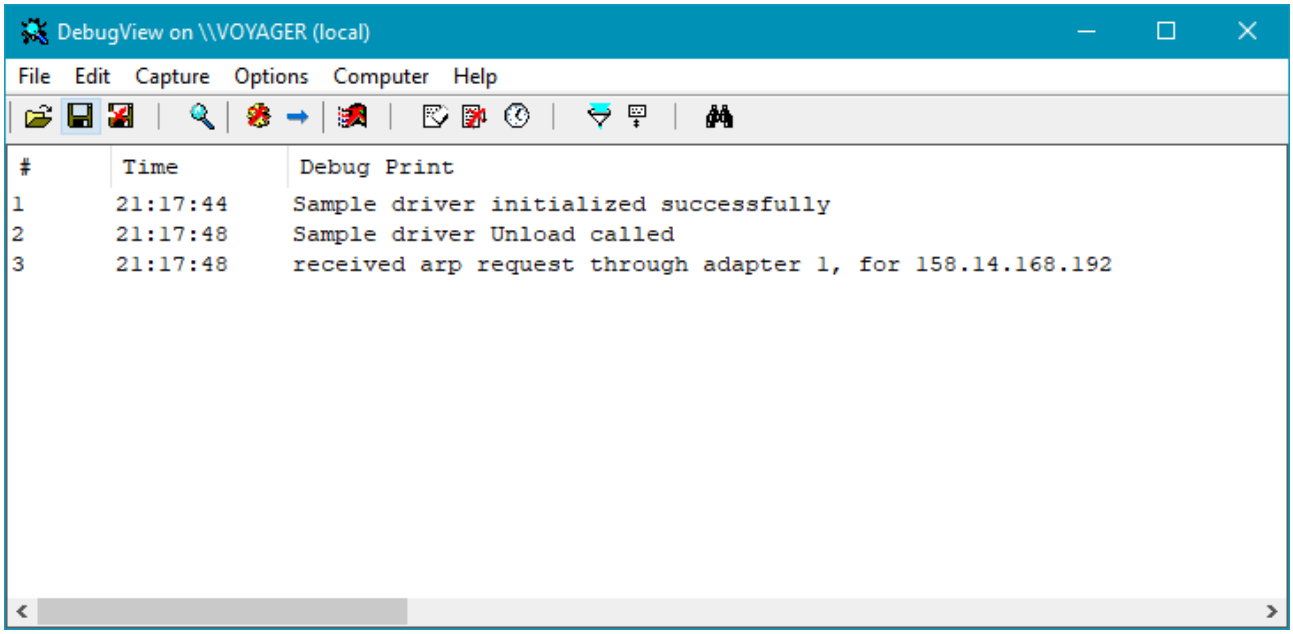

Рисунок 2-7. Вывод утилиты DebugView

Упражнения

Добавьте код в образец DriverEntry для вывода версии ОС Windows: мажорной, минорной и номер сборки. Используйте функцию RtlGetVersion для получения информации. Проверьте результаты с DebugView.

Резюме

Мы увидели инструменты, необходимые для разработки ядра, и написали очень маленький драйвер, что-бы проверить основные инструменты в работе.

В следующей главе мы рассмотрим основные API ядра, концепции и структуры.