УДК 004.454

ИССЛЕДОВАНИЕ ПРИМЕРА USB ДРАЙВЕРА WINDOWS NT

Евстратов Е.К, Теплинский С.В.
Донецкий национальный технический университет
E-mail: evgeniy.evstratov@gmail.com

Рассматриваются пример USB драйвера Windows NT с описанием основных действий, некоторые общие сведения о USB драйверах и краткая информация об их установке.

Введение

В пакет WinDDK входят разнообразные примеры драйверов, в том числе USB, которые могут сильно облегчить написание драйвера собственного устройства. В данной статье пойдет описание примера из «WinDDK\7600.16385.1\src\usb\osrusbfx2\kmdf\sys», в этом каталоге есть папки «step1», .. «step5», в которых по шагам наращивается пример драйвера, реализующего функции блочного чтения и записи.

Основные понятия

USB драйвер устанавливается для определенного устройства и автоматически загружается системой при подключении этого устройства. USB протокол предусматривает несколько операций для обмена данными, одна из которых выполняется при установлении связи для идентификации оборудования. Ключевые идентификаторы это код производителя (vendor id - VID) и код устройства (product id - PID), каждый по 16 бит. Также есть название устройства и остальные параметры, однако в Windows отображается название устройства определенное драйвером (точнее конфигурационным файлом при установке драйвера). Для передачи данных используются так называемые «конечные точки» (endpoint), которые могу быть следующих типов:

- управляющие передачи;
- передачи массивов данных (bulk transfers);
- передачи по прерываниям;
- изохронные передачи.

В данной статье будут рассмотрены только передачи массивов данных. Полный пример достаточно громоздкий и практически не содержит комментариев, здесь только описаны некоторые моменты.

Точка входа

На первом шаге (папка step1) в примере драйвер содержит только точку входа и функцию добавления устройства. В точке входа выполняется инициализация конфигурации WDF драйвера и передача ее в объект драйвера:

```
    WDF_DRIVER_CONFIG  config;
    WDF_DRIVER_CONFIG_INIT(&config,    //    инициализируем    стандартную
конфигурацию
    EvtDeviceAdd); // и регистрируем функцию добавления устройства
```

```
    status = WdfDriverCreate(DriverObject, // объект драйвера из параметров
DriverEntry
    RegistryPath, // параметр DriverEntry
    WDF _ NO _ OBJECT _ ATTRIBUTES,
    &config, // передаем конфигурацию
    WDF _ NO _ HANDLE);
```

1

Функция добавления устройства

Здесь, в «EvtDeviceAdd» регистрируется функция подготовки оборудования:

```
    WDF _ PNPPOWER _ EVENT _ CALLBACKS  pnpPowerCallbacks;
    WDF _ PNPPOWER _ EVENT _ CALLBACKS _ INIT(&pnpPowerCallbacks);
    //  стандартное  конфигурирование  структуры  WDF _ PNPPOWER _ EVENT _
CALLBACKS
    pnpPowerCallbacks.EvtDevicePrepareHardware = EvtDevicePrepareHardware;
    // заносим адрес функции в структуру
    WdfDeviceInitSetPnpPowerEventCallbacks(DeviceInit, // DeviceInit – из
параметров функции
    &pnpPowerCallbacks); // передаем структуру для инициализации
```

Далее подготавливаются атрибуты, и создается устройство:

```
    WDF _ OBJECT _ ATTRIBUTES  attributes;
    WDFDEVICE                                device;
    WDF _ OBJECT _ ATTRIBUTES _ INIT _ CONTEXT _ TYPE(&attributes,
    DEVICE _ CONTEXT); // передаем тип структуры контекста устройства
    status = WdfDeviceCreate(&DeviceInit, &attributes, &device); // создаем
WDF устройство
```

Далее создается интерфейс:

```
    DEFINE _ GUID(GUID _ DEVINTERFACE _ OSRUSBFX2, // Generated using guidgen.
exe
        0x573e8c73, 0xcb4, 0x4471, 0xa1, 0xbf, 0xfa, 0xb2, 0x6c, 0x31, 0xd3,
0x84);
    // {573E8C73-0CB4-4471-A1BF-FAB26C31D384}

    status = WdfDeviceCreateDeviceInterface(device, // объект устройства
                (LPGUID) &GUID _ DEVINTERFACE _ OSRUSBFX2, // серийный номер
                NULL);
```

Далее настраиваем очередь запросов:

```
    WDF _ IO _ QUEUE _ CONFIG   ioQueueConfig;
    WDF _ IO _ QUEUE _ CONFIG _ INIT _ DEFAULT _ QUEUE(&ioQueueConfig,
    WdfIoQueueDispatchParallel);  //  настройка  очереди  с  параллельной
отправкой
    ioQueueConfig.EvtIoRead = EvtIoRead; // функция чтения из устройства
    ioQueueConfig.EvtIoWrite = EvtIoWrite; // функция записи в устройство
    status = WdfIoQueueCreate(device, // объект устройства
    &ioQueueConfig, // конфигурация
    WDF _ NO _ OBJECT _ ATTRIBUTES,
    WDF _ NO _ HANDLE); // создаем очередь
```

Функция подготовки оборудования

В этой функции нам нужно получить дескрипторы операций обмена, которые называются «pipe», они сохраняются в контекст устройства и используются при выполнении соответствующих операций. Получить их можно из интерфейса устройства, делается это следующим образом:

```
    if (pDeviceContext->UsbDevice == NULL) { // если USB устройство еще не
создано
        status = WdfUsbTargetDeviceCreate(Device, // объект устройства
        WDF _ NO _ OBJECT _ ATTRIBUTES,
        &pDeviceContext->UsbDevice); // получим USB устройство
    …
    }
    WDF _ USB _ DEVICE _ SELECT _ CONFIG _ PARAMS _ INIT _ SINGLE _ INTERFACE(
    &configParams); // инициализация конфигурации
    status = WdfUsbTargetDeviceSelectConfig(pDeviceContext->UsbDevice,
    WDF _ NO _ OBJECT _ ATTRIBUTES,
    &configParams); // получаем конфигурацию нашего устройства
    pDeviceContext->UsbInterface = configParams.Types.SingleInterface.
         ConfiguredUsbInterface;
    // получен интерфейс, теперь получаем дескрипторы обмена
    pDeviceContext->BulkReadPipe = WdfUsbInterfaceGetConfiguredPipe(
    pDeviceContext->UsbInterface, // передаем интерфейс
    BULK _ IN _ ENDPOINT _ INDEX, // определенный непосредственно в примере
    индекс операции
    NULL); // здесь может быть указатель на структуру типа
    WDF _ USB _ PIPE _ INFORMATION, в которую будет записана подробная
    информация об операции.
```

Получать дескрипторы лучше в цикле, начиная с индекса равного нулю, и пока функция не вернет NULL (либо получить реальное количество дескрипторов при помощи функции «WdfUsbInterfaceGetNumConfiguredPipes», в которую предается интерфейс), и определять тип операции по последнему параметру.

Функция чтения «EvtIoRead»

Получает в параметрах очередь, структуру запроса и еще отдельно запрашиваемое количество байт для чтения.

```
    WDFUSBPIPE  pipe;
    PDEVICE _ CONTEXT  pDeviceContext;
    WDFMEMORY  reqMemory;
    pDeviceContext = GetDeviceContext(
    WdfIoQueueGetDevice(Queue)); // получаем контекст устройства по его
очереди
    pipe = pDeviceContext->BulkReadPipe;
    status = WdfRequestRetrieveOutputMemory(Request, &reqMemory); //
запрашиваем память
```

Далее выполняем само чтение, оно может быть как синхронным, так и асинхронным с установлением функции, которая будет вызвана по завершению. В примере рассмотрен асинхронный способ:

```
    status = WdfUsbTargetPipeFormatRequestForRead(pipe, // дескриптор
```

```
операции
     Request, // дескриптор запроса
     reqMemory, // дескриптор памяти
     NULL); // переделываем запрос в асинхронный но не отправляем его
     WdfRequestSetCompletionRoutine(Request, // по завершению этого запроса
будет вызвана
     EvtRequestReadCompletionRoutine, // эта функция
     pipe);
     ret = WdfRequestSend(Request,
     WdfUsbTargetPipeGetIoTarget(pipe),
     WDF _ NO _ SEND _ OPTIONS); // отправляем асинхронный запрос
```

Если ret равен FALSE, то запрос отправить не удалось.

Функция завершения запроса чтения «EvtRequestReadCompletionRoutine»

В этой функции происходит проверка статуса выполнения операции и получение реального количества считанных байт, после чего запрос полностью завершается:

```
     status = CompletionParams->IoStatus.Status; // статус операции
     usbCompletionParams = CompletionParams->Parameters.Usb.Completion;
     bytesRead = usbCompletionParams->Parameters.PipeRead.Length; //
реальное количество
     считанных байт
     WdfRequestCompleteWithInformation(Request, status, bytesRead); //
завершение запроса
```

Функция записи выглядит похожим образом.

Пользовательская программа

В пользовательской программе устройство (драйвер) открывается по имени как файл, имя можно определить по серийному номеру, этот процесс достаточно громоздкий, мною был использован код из источника [1]. После получения имени устройство открывается с помощью функции CreateFile, обмен производиться с помощью ReadFile, WriteFile, они приведут к вызову функций драйвера EvtIoRead и EvtIoWrite соответственно.

Об инсталляции

При подключении устройства PNP менеджер запросит его VID и PID, если для такой пары драйвер не обнаружен то Windows предложит установить его, установить можно с указанного места на диске, в папке для инсталляции должны быть подготовлены несколько файлов: сам драйвер (файл *.sys), конфигурационный файл (*.inf), и установочная DLL. DLL можно взять из WinDDK: «WinDDK\7600.16385.1\redist\ wdf\x86\ WdfCoInstaller01009.dll», конфигурационный файл брался мною из источника [1], в прилагающемся архиве по адресу: «Chapter_6\driver\osrusbfx2.inf». Система проверит VID и PID на соответствие с заданными в конфигурационном файле и в случае несоответствия выдаст ошибку (они указаны по несколько раз), в этом же файле указаны имя драйвера и имя DLL, отвечающей за установку.

Выводы

В данной статье проведено описание основных частей примера USB драйвера из пакета WinDDK, в процессе написания статьи был реализован и успешно протестирован двусторонний обмен информации с микроконтроллером pic18f2550, имеющим аппаратную реализацию USB протокола.

Перечень источников

[1] Пишем драйвер для самодельного USB устройства. Интернет ресурс: http://habrahabr.ru/post/139593/

[2] Developing a WDF USB Kernel Mode Driver for the OSR USB FX2. Интернет ресурс: http://www.codeproject.com/Articles/13620/Developing-a-WDF-USB-Kernel-Mode-Driver-for-the-OS

[3] Пишем драйвер для USB девайса. Pipe 0: что такое usb?. Интернет ресурс: http://habrahabr.ru/post/92628/