РАЗРАБОТКА ДРАЙВЕРА ДЛЯ ПРЯМОГО ДОСТУПА К ПОРТАМ ДЛЯ MICROSOFT WINDOWS 8.1 x86-x64

А.Г. Черемнов, В.С. Аврамчук
Томский политехнический университет
Научный руководитель: В.С. Аврамчук
8xandr@gmail.com

Задача непосредственного управления устройством очень сложна и требует высокого уровня детализации, по этой причине в состав устройств входит отдельная вычислительная система, называемая контроллером [1]. В контроллере устройства для управления имеется некоторое число регистров. Пространство портов ввода-вывода формируется из совокупности всех регистров устройств [1]. В операционных системах Windows регистры большинства устройств помещаются в специальное адресное пространство портов ввода-вывода (I/O port space) [1], располагаемое физически в оперативной памяти и виртуально в адресном пространстве операционной системы, в котором каждый регистр имеет адрес порта. В режиме ядра имеется доступ к специальным командам ввода-вывода, позволяющим осуществлять запись и чтение данных в регистры [2].

Информация об устройствах, подключенных к главной магистрали материнской платы, в том числе шине PCI или PCI Express (шины PCI и PCI Express подключаются к главной магистрали посредством мостовых схем), также хранится в адресном пространстве портов ввода-вывода операционной системы [1]. Безусловно, для опроса устройств, подключенных компьютеру можно использовать специальные функции WinAPI, но скорость их выполнения достаточно медленная, а получаемая информация об устройствах существенно ограничена [3,4]. Задачи требующие перевод работы нескольких устройств в режим DMA или Ultra DMA, например, посекторное копирование информации с поврежденного жесткого диска на другой накопитель также требуют прямого доступа к портам устройств.

Начиная с операционной системы Windows Vista, любая возможность прямого доступа к портам устройств закрыта [3], возможно, с целью повышения безопасности и устойчивости операционной системы.

Единственной возможной альтернативой является написание специального драйвера, который будет исполняться в режиме ядра операционной системы, заниматься опросом портов по требованию некоторого процесса. Отдельно отметим, что процесс должен выполняться в режиме пользователя. Также необходимо обеспечить безопасный обмен данными между драйвером и процессом в пользовательском режиме, с реализаций удобного интерфейса, третьему приложению.

Функциональная схема взаимодействия между драйвером и процессом приведена на рисунке 1.

Рис. 1 Функциональная схема взаимодействия между драйвером и процессом

Под драйвером в данном случае понимается некоторый набор функций, вызываемых операционной системой при наступлении некоторых событий, приходящих от процесса, выполняемого в режиме пользователя.

В операционной системе Microsoft Windows 8.1 используется WDF (Windows Driver Foundation) драйверная модель [6].

User Mode Driver Framework (UMDF) и Kernel Mode Driver Framework (KMDF) являются основными элементами модели WDF. Поддержка современной программной объектно-ориентированной модели разработки драйверов для Microsoft Windows 8-8.1 обеспечивается наборами описанными в [5,6].

Существует огромное число типов драйверов, перечислим некоторые из них[5,7]:

- мини-драйвера;
- драйверы классов;
- фильтрующие драйвера;
- функциональные драйвера.

Драйвер прямого доступа к портам, был реализован как функциональный драйвер.

Поскольку, опрос портов может осуществляться несколькими сторонними процессами, существует проблема синхронизации работы драйвера. Возможным решением является использования ассемблерных ставок следующего вида:

```
__asm
{
    lock dec некоторая переменная,
уменьшаемая на единицу
}
```
или
```
__asm
{
    lock inc некоторая переменная,
увеличиваемая на единицу
}
```

Префикс lock позволяет безопасно выполнить команду, которая идёт за ним, так как блокирует процессы, пока выполняется команда [1].

Для опроса портов из-под ядра операционной системы создается виртуальное устройство GiveIO, ожидающие некоторые команды от стороннего процесса (API). Это устройство использует специальные механизмы внутри ядра Windows, позволяющие получить доступ к системному пространству операционной системы.

После завершения работы устройство удаляется. В качестве примера приведён фрагмент кода, осуществляющий удаление виртуального устройства.

```
void GiveIoUnload(IN PDRIVER_OBJECT DriverObject)
    {
        UNICODE_STRING DeviceLinkUnicodeString;
        NTSTATUS ntStatus;

        KdPrint(("Entering GiveIoUnload"));

        RtlInitUnicodeString (&DeviceLinkUnicodeString,
L"\\DosDevices\\GiveIo");

        ntStatus = IoDeleteSymbolicLink (&DeviceLinkUnicodeString);

        if (NT_SUCCESS(ntStatus))
        {
            IoDeleteDevice (DriverObject->DeviceObject);
        }
        else
        {
            KdPrint(("ERROR: IoDeleteSymbolicLink"));
        }

        KdPrint(("Leaving GiveIoUnload"));
    }
```

Разница версий для 32-разрядных и 64-разрядным систем заключается лишь в некоторых разных системных вызовах операционной системы Windows, которые происходят внутри виртуального устройства GiveIO.

В процессе отладки использовались следующие инструменты:

- Static Driver Verifier (SDV);
- PREFast for Drivers (PFD).

PFD использовался для поверхностного анализа операций драйвера, проверялось наличие проблем переполнения буфера. SDV использовался для глубокой проверки исполнения кода. Этот инструмент позволяет отслеживать исполнение вызовов и функций через WDM.

Отдельно отметим, что PFD гораздо быстрее SDV, поскольку возможности SDV ограничены пределами одной функции.

Обработка прерываний стеком устройства осуществлена через IRP – пакеты. Функциональная схема обработки пакетов IRP стеком устройства приведена на рисунке 2 [5].

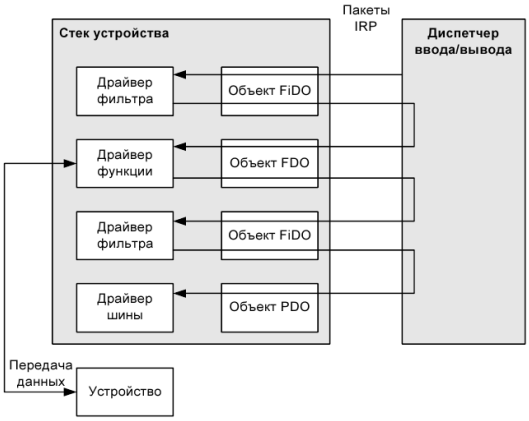

Рис. 2. Функциональная схема обработки пакетов IRP стеком устройства

Разработанный драйвер позволяет «напрямую» взаимодействовать с регистрами устройств.

Литература

1. Tanenbaum Andrew S. Structured Computer Organization. Sixth Edition. – University of Michigan, 2010. – p. 816.

2. Tanenbaum Andrew S. Modern Operating Systems. Third Edition. – University of Michigan, 2010. – p. 1120.

3. Microsoft Developer Network. MSDN Library. URL: http://msdn.microsoft.com/en-us/library/default.aspx (Дата обращения: 17.10.2014)

4. Джефри Рихтер. Windows. Создание эффективных Win32 приложений с учётом специфики 64-разрядной версии Windows. . – М.: Питер, 2000. – с. 583.

5. Penny Orwick, Guy Smith. Developing Drivers with the Windows Driver Foundation. – Microsoft, 2008. – p. 880.

6. Windows Driver Kit (WDK). URL: http://msdn.microsoft.com/en-US/library/windows/hardware/ff557573 (Дата обращения: 17.10.2014)

7. Комиссарова В. Программирование драйверов для Windows. – Спб.: БХВ-Петербург, 2007. – 256 с.