Разработка драйверов для 64-разрядной Windows

Любой разработчик драйверов сталкивается в процессе работы с несколькими неприятными аспектами. По сути, драйверы могут принести вред: малейшая ошибка в драйвере может вывести из строя всю систему. Модели программирования замысловаты и коварны, они требуют глубоких знаний программирования режима ядра и детального представления о внутренних элементах операционной системы. К счастью, Стивен Смит, руководитель группы разработчиков программного обеспечения в компании Compuware, знает, что может сыграть ведущую роль в построении драйверов к 64-разрядной Windows, использующих архитектуру AMD64. А вот и ответ: ряд инструментальных средств DriveStudio, выпущенных Compuware.

Я не побоюсь заявить, что разработка драйверов под Windows, отладка и тестирование – крайне напряженный процесс. Думаю, что уже достаточному количеству «счастливчиков» выпадала на долю каторга 4-х часового непрерывного перезапуска системы.

Еще одна проблема, с которой сталкиваются при создании драйверов, - это нехватка инструментария разработки, отладки и тестирования. То, что нужно – это средства, ускоряющие и облегчающие процесс. Compuware DriverStudio (http://www.compuware.com/products/driverstudio/default.htm) – одно из решений.

DriverStudio – набор инструментальных средств для создания драйверов и отладки, тестирования. Он охватывает все фазы цикла написания: создание кода, установку, отладку, тестирование, удаленный анализ и настройку. DriverStudio разработан для ускорения процесса создания структурированных, надежных, оптимизированных драйверов и решения самых сложных проблем, связанных с режимом ядра.

DriverStudio включает в себя все разнообразие средств разработки для ускорения процесса написания драйверов.

DriverWorks – еще одно такое средство, «гвоздем» которого являются библиотеки классов, интеграция Visual Studio .NET IDE, мастера генерирования кода C++, экономящие несколько недель работы, тысячи строк тестируемого, отлаживаемого исходного кода, соответствующего стандартам Microsoft Windows Hardware Quality Lab сертификации, примеры исходного кода для реальных устройств, утилиты, расширенная документация.

DriverStudio включает в себя мощнейшие системные отладчики. Вы наверняка знакомы с SoftICE, одних из самых известных отладчиков для семейства операционных систем Windows. DriverStudio 3.0 представляет Visual SoftICE, способный работать на двух машинах, кроссплатформенный член семьи системных отладчиков SoftICE, обладающий всеми удивительными классическими свойствами SoftICE. Он был дополнен мощным расширяемым графическим пользовательским интерфейсом, спроектированным для распределенной отладки кода на 32-х и 64-х битной платформах, включая и платформу AMD64.

1. Драйверы AMD64

Исходный код демонстрационного драйвера, поставляемого с DriverStudio, довольно легко перенести на платформу AMD64. Благодаря интеллектуальному дизайну процессора AMD и отладчику Visual SoftICE, процесс перенесения и отладки драйверов был сделан легким и простым.

Платформа AMD64 представляет собой новую задачу для разработчиков. Другие компании, специализирующиеся на микропроцессорах, выпустили 64-битные процессоры на основе архитектуры RISC или с полностью новыми наборами инструкций. Драйверы обычно создаются на C, и поэтому изменения на этом базовом уровне не слишком отражаются на судьбе большинства разработчиков (компилятору берет на себя вопросы адаптации к обновляемому набору инструкций процессора и оптимизации). До настоящего времени все было неплохо.

Однако, борьба за повышение производительности кода на новой платформе может стать настоящим кошмаром для разработчиков: это напрямую относится к тем, кто в процессе отладки имеет дело с языком ассемблера, или же вручную оптимизирует стандартные подпрограммы.

К счастью, AMD определила архитектуру, расширяющую набор команд x86, поддерживающую модель и подход к разработке программных средств, так же, как и обеспечивающую совместимость с существующими приложениями. Этот эволюционный подход позволяет разработчикам использовать привычный набор инструкций x86 и в нужный момент «мягко» перейти 64-битному программированию. Опять таки, компилятор отвечает за создание

правильного машинного кода. Благодаря этому можно без затруднений создавать и отлаживать драйверы для платформы AMD64. Безусловно, такой подход AMD существенно облегчает жизнь программистов.

Что можно придумать для исследования платформы AMD64 лучше, чем экспериментирование с мощным отладчиком.

Я использую систему Windows 2000 x86 в качестве машины для разработки. Немного позже я установлю тестовую (целевую) машину для инсталляции и отладки драйвера. Давайте сначала напишем драйвер. Не беспокойтесь, мы выполним это задание в кратчайшие сроки с минимальными затратами, в чем помогут нам DriverWorks и DriverWizard.

DriverWizard позволяет создавать все виды драйверов для Windows. Чтобы продемонстрировать достоинства платформы AMD64, создадим виртуальный Windows Driver Model драйвер. После запуска приложения DriverWizard, я просто ввожу в строке project name имя "MyDriver" (см. Рисунок 1).

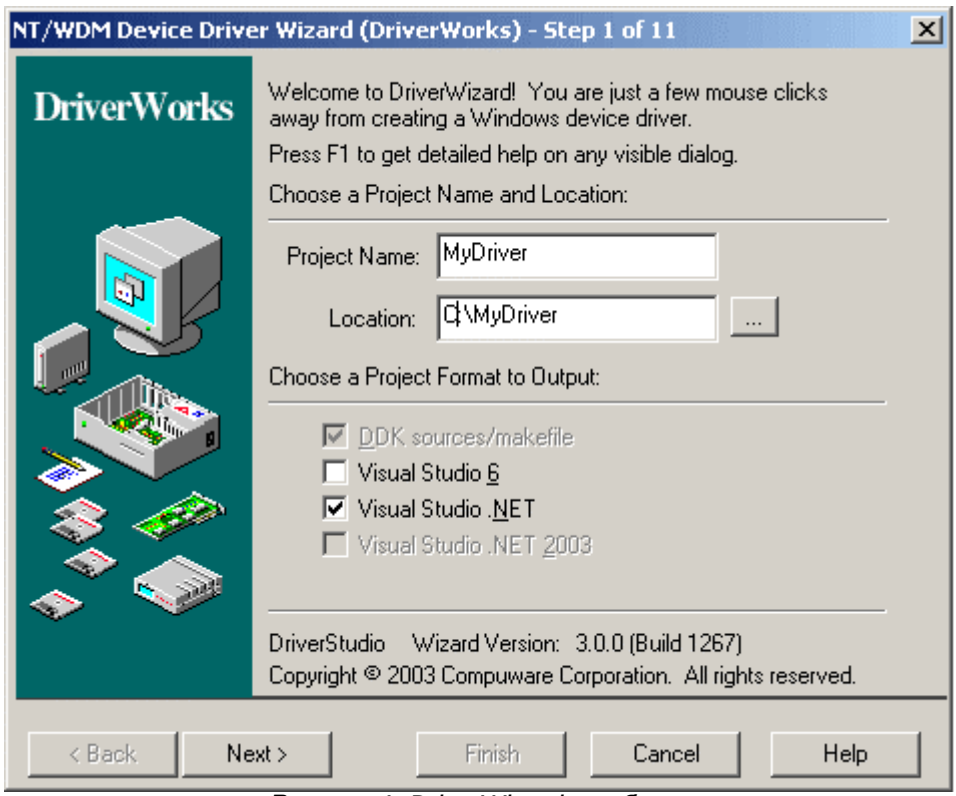

Рисунок 1. DriverWizard в работе

Теперь пару щелчков и мастер начинает создание всех необходимых файлов проекта, файлов исходных кодов и файлов инсталляции для драйверов и тестовых приложений. Затем следует открытие проектного файла Visual Studio и проверка кода (см. Рисунок 2).

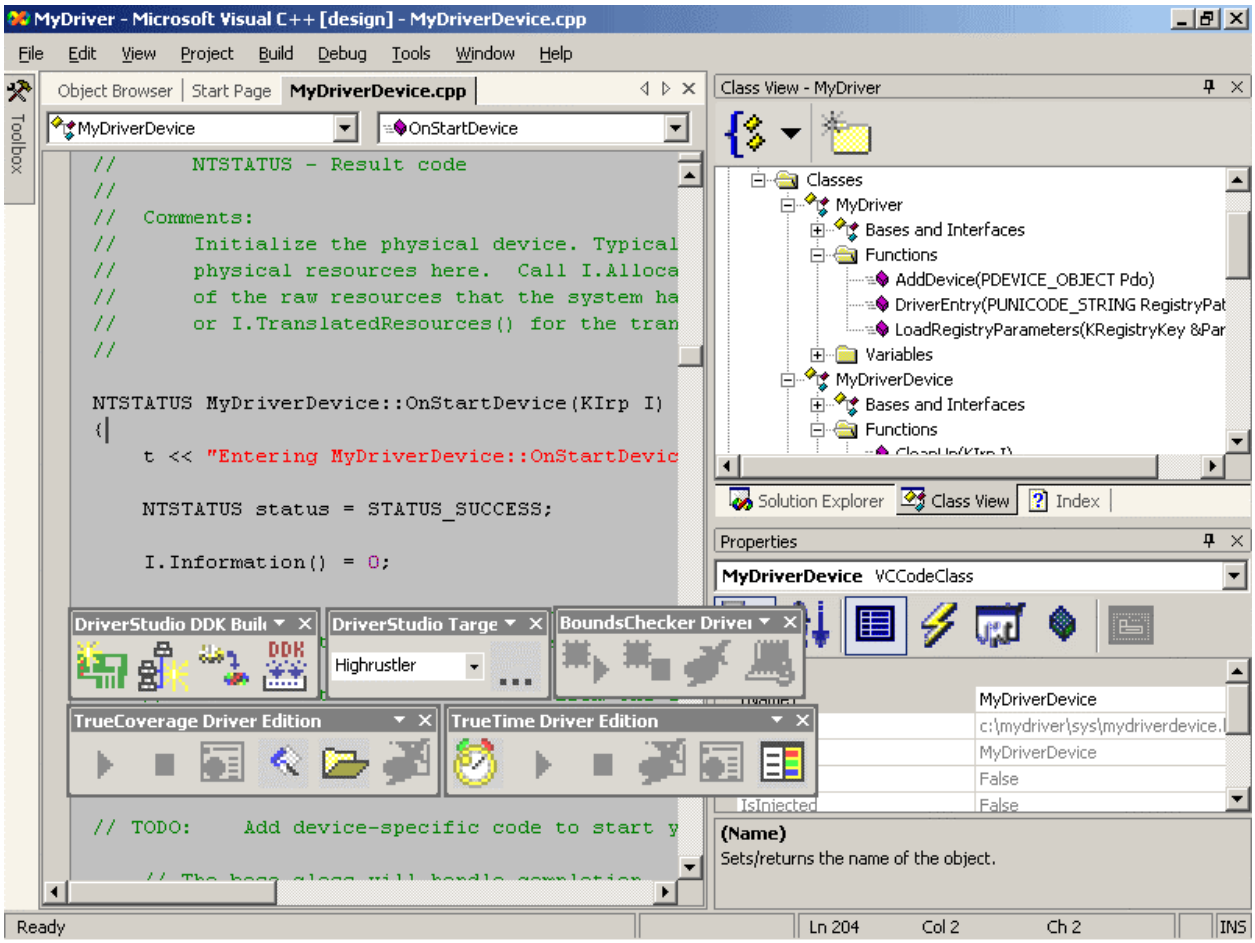

Рисунок 2. Окно Visual Studio с кодом, сгенерированным DriverWizard

Стоп! Что это такое? C++? Да, C ++ в драйвере! DriverWorks - это каркас C ++, значительно упрощающий написание драйверов. Код прекрасно работает на x86, Itanium и AMD64. Если к процессу привлечь Windows Server 2003 Driver Development Kit (DDK), то драйвер можно будет легко подогнать под платформу AMD64.

2. Установка целевой системы

Сейчас, когда написан и собран WDM-драйвер и тестовое приложение (что заняло у нас не более пяти минут), наступил момент установки целевой системы AMD64.

Чтобы установить целевую систему, рекомендуется установить связь между ней и хостом в той же самой сети (подсеть TCP/IP). Visual SoftICE поддерживает различные варианты подключений, включая Ethernet основанный на TCP/IP, USB, IEEE 1394 и последовательный интерфейс. Это действительно мощная и полезная особенность, поскольку с ее помощью обеспечивается легкий просмотр и отладка многочисленных целевых машин с одного хоста. Итак, с этого момента можно забыть о том, как фанаты своего дела молятся на последовательные кабели для осуществления отладки.

С хоста Visual SoftICE может подключиться к целевой машине (см. Рисунок 3)

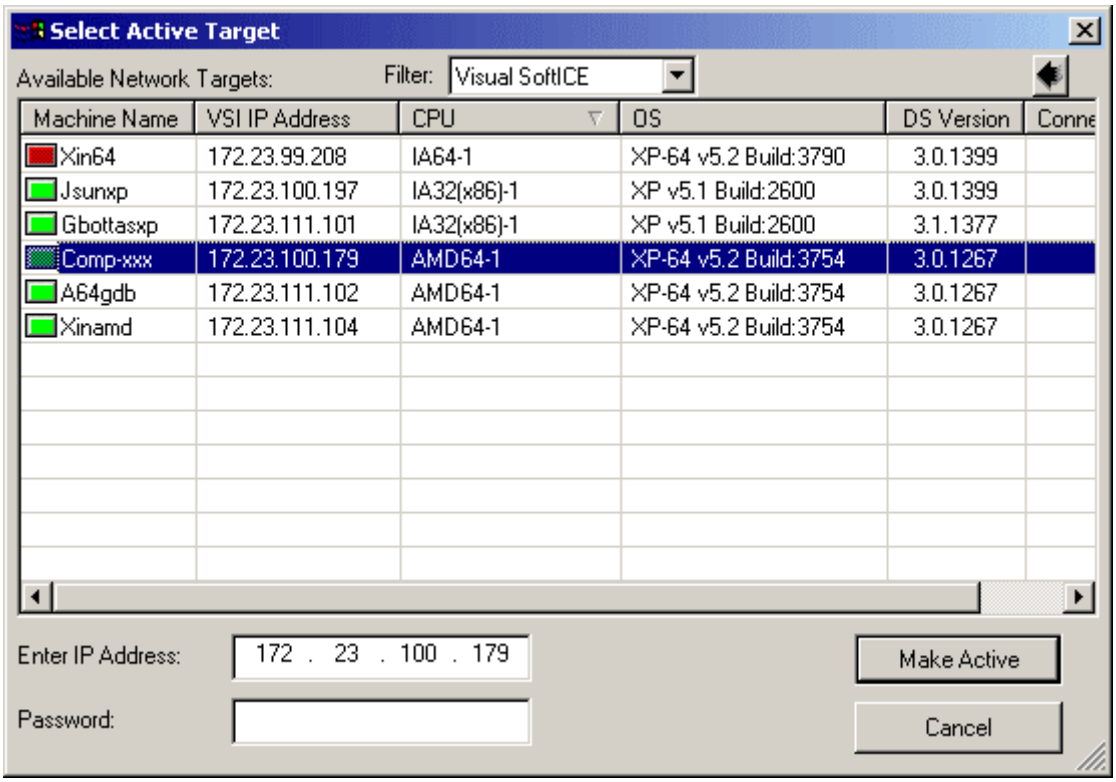

Рисунок 3. Список активных целей

Диалоговое окно "Target Browser" отображает машины, где установлен DriverStudio, и информацию о самой машине, включая имя, тип процессора, операционную систему, версию и состояние машины (рабочее, остановка, проверка на сбои, ожидание). Осуществить подключение к любой из показанных машин очень просто.

Используя Visual SoftICE, можно легко и просто исследовать платформу AMD64 и Windows Server 2003. Visual SoftICE имеет мощный настраиваемый графический пользовательский интерфейс. Каждый из следующих режимов: дисассемблер, обозреватель событий, команды, стек, память, метки, процессы, регистры, точки останова представлен своим конфигурируемым плавающим окном (так называемые «окна с закладками»)

Выберите "Break" и машина остановит операцию выполнения. Посмотрите на окно дисассемблера, чтобы увидеть инструкции AMD64. Теперь обратим внимание на регистровое окно, где можно найти набор регистров в группах и для каждого из центральных процессоров. Для пошагового выполнения, используйте команду "Step".

А как же насчет драйвера, который был создан с помощью Driver Wizard? Скопировать его на целевой компьютер на поможет все тот же Visual SoftICE. Просто выполните команду "fput", и произойдет копирование MyDriver.sys, Test_MyDriver.exe и MyDriver.inf на целевой компьютер.

С помощью Visual SoftICE, установите на значке “загрузка” точку останова. Эта команда очень помогает, когда необходимо остановить работу машины при первой загрузке приложения или драйвера, ей требуется лишь имя исполняемого образа. В командном окне напишите "bpload mydriver.sys." В качестве альтернативы можно использовать диалоговое окно Breakpoint, чтобы установить, модифицировать или удалить любой тип точки останова (SoftICE и Visual SoftICE поддерживает широкий набор функциональности для работы с точеками останова).

После инсталляции драйвера на целевой машине уже установленная точка останова должна быть активизирована, и выполнение операции целевым компьютером будет приостановлено. Мастер выведет на экран текущую инструкцию, которая является подпрограммой драйвера DriverEntry - точкой входа в код драйвера (см. Рисунок 4).

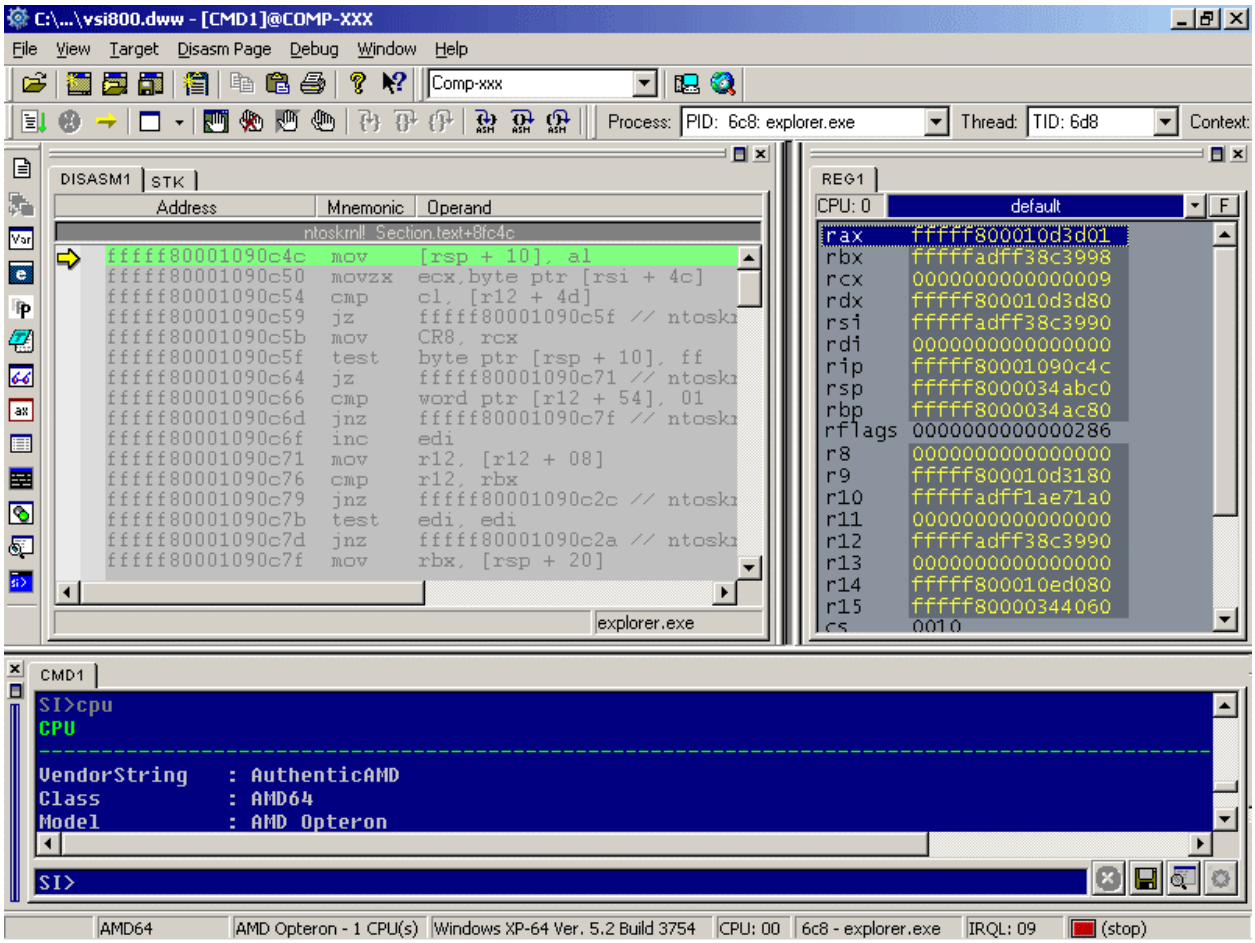

Рисунок 4. Окно отладчика поле активации точки останова

3. Точки останова и исходный код

Бесценной возможностью Visual SoftICE является отображение исходного кода для текущей инструкции после того, как точка останова уже была активизирована (см. Рисунок 5). Visual SoftICE автоматически загружает символ и исходный файл для драйвера. Подпрограмма DriverEntry находится внутри оболочки DriverWorks C++.

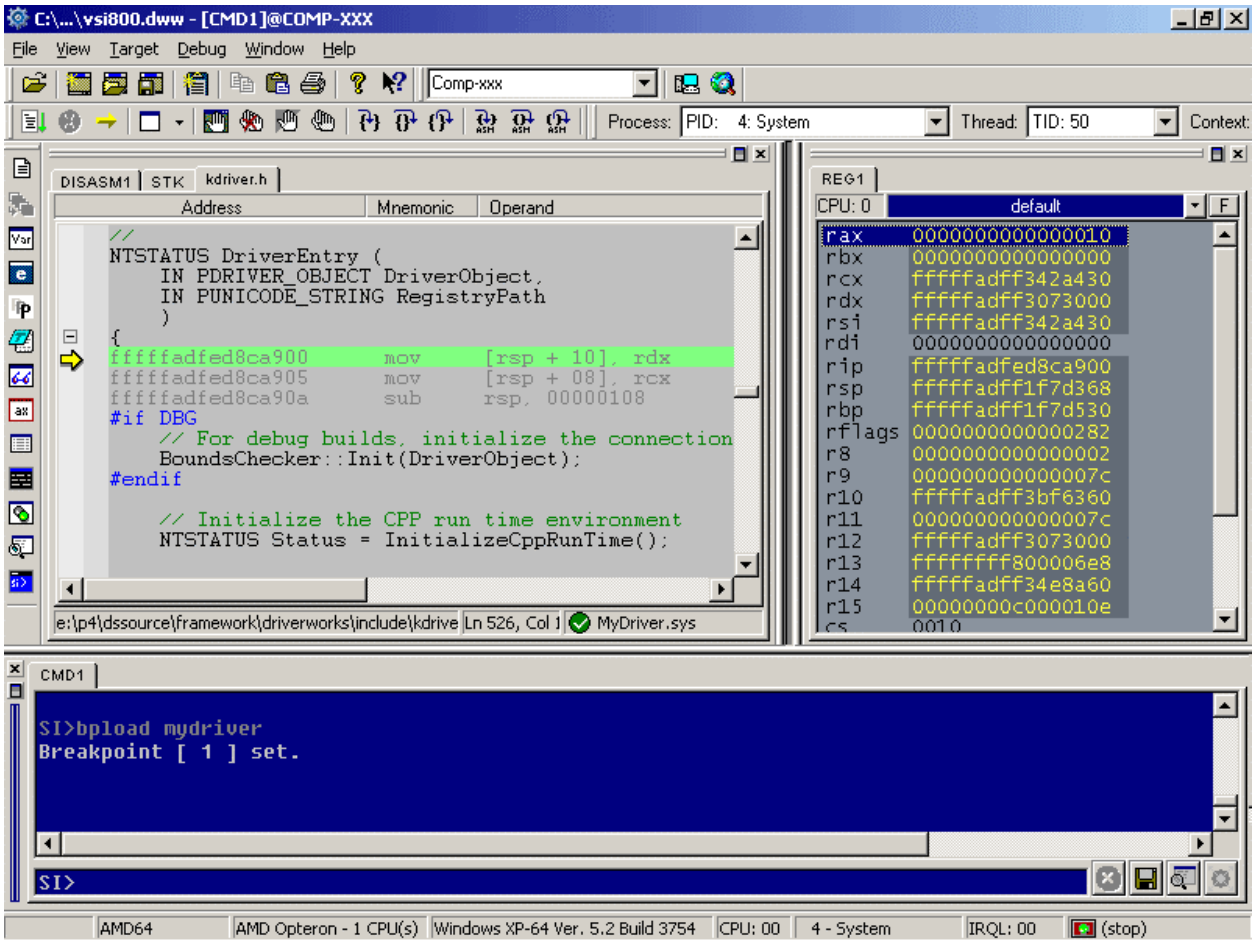

Рисунок 5. Пример отображения исходного кода в точке останова

Воспользовавшись окном Source, можно просто дисассемблировать отдельную строку или целый исходный файл, чтобы вывести на экран инструкции AMD64. Кроме этого, есть возможность установить, включить, отключить точки останова, исходный код или инструкции здесь же.

Совсем несложно установить еще несколько точек останова, используя мощные возможности VSI и его сильнейшую символьную поддержку. Воспользуйтесь командой Sym для отображения символов.

```
Sym *::Create
```

Символы можно использовать, чтобы быстро поместить точку останова по правильному адресу. Для установки точки останова на запросе Create, используйте команду Bpx.

```
Bpx MyDriverDevice::Create
```

Воспользуйтесь командой 'Go', меню или же панелью инструментов для запуска программы на целевой машине до активизации точки останова. Чтобы осуществить активизацию, необходимо, чтобы тестовое приложение вызвало тестовый драйвер. Можно было бы сразу перейти к целевой системе и непосредственно запустить приложение, но зачем эти хлопоты, когда можно его запустить прямо из отладчика. Примените команду "Exec" для запуска тестового приложения на целевой машине.

```
Exec c:\Test_MyDriver.exe
```

Сейчас точка останова активизирована, и выполняемый исходный код текущего драйвера выводится на экран.

Давайте проанализируем, что нами было проделано.

При открытии стековой страницы (или же при использовании команды stack), можно просмотреть полную информацию вызовов из приложения, через кольцо операционной системы, в ядро и в наш тестовый драйвер. Кроме этого, можно просматривать стек, память, регистры и состояние любого процесса или образа в системе.

Я надеюсь, этот пример поможет вам глубоко прочувствовать стадии написания драйвера на платформе AMD64 и то, как правильный выбор инструментального средства может значительно облегчить процесс. Теперь давайте посмотрим, что происходит внутри драйверов Windows, работающих на платформе AMD64. Compuware DriverStudio и Visual SoftICE могут предоставить подробное описание всех внутренних компонентов.

4. Детали платформы AMD64

Итак, несколько слов о платформе x64. Сразу оговоримся, что все обсуждаемые моменты наглядно видны в Visual SoftICE, на примере одного из приложений, включенных в комплект DriverStudio.

Сейчас полезно было бы рассмотреть некоторые свойства Visual SoftICE's не только для того, чтобы разобраться, как работают эти средства, но и еще и представить себе внутренние составляющие архитектуры AMD64 и их использование в Windows.

64-битовые версии операционных систем Microsoft Windows XP и Windows 2000 Server были существенно доработаны для того, чтобы воспользоваться всеми преимуществами этой новой архитектуры. В рамках SoftICE мы попытались просто расширить существующие свойства, раскрыть возможности новой платформы, и как можно больше отнести к представлению общих данных.

Но сначала поговорим об аппаратных средствах.

4.1 Адреса

Разумеется, что в рамках 64-битовой платформы адреса могут иметь различные типы, длину и кодирование. По причине незначительного отличия архитектур между собой (некоторые работают с 48-ю битами, некоторые – с 64-мя битами), Visual SoftICE обеспечивает согласованное отображение данных и позволяет пользователю форматировать 16 знаков в 64-битовых адресах подходящим способом. (Например, воспользуйтесь командой SET ADDRESSFORMAT).

4.2 Регистры

Некоторые архитектуры опираются на большое количество новых регистров, в то время, как отдельные реализации лишь расширяют коллекцию хорошо знакомых нам регистров по архитектуре x86. С помощью Visual SoftICE можно увидеть последовательный механизм отображения и сбора для регистров, называемых «регистровыми группами». Каждая поддерживаемая платформенная архитектура (x86, IA64, AMD64), обладает собственными уникальными группами, а действия (чтение и запись) могут быть применены как к группам, так и к отдельным регистрам.

Команда RG обеспечивает возможность просмотра доступных групп регистров. Страница регистров обеспечивает легкий доступ для просмотра любой комбинации групп, отдельных регистров или регистровых полей.

Как команда R, так и страница регистров позволяют просматривать значения хранимые в регистрах процессоров многопроцессорных машинах.

Команда MSR определяет версию процессора. Она успешно работает и на процессорах AMD64.

В прошлом изготовители процессоров задокументировали один набор имен для регистров, в то время, как поставщики операционных систем обычно присваивают им свои названия. Более того, разработчики могут использовать другие имена для тех же самых физических регистров в силу сложившейся практики. Visual SoftICE поддерживает разные подходы к именованию регистров и обеспечивает имена "HW," "OS" и "Common", где это возможно (см. команду SET REGNAME).

4.3 Инструкции

Каждый процессор в обязательном порядке имеет собственный набор инструкций. Visual SoftICE обеспечивает полное дисассемлирование и подстрочное ассемблирование для всех поддерживаемых платформ, включая AMD64.

4.4 Периферийные и дополнительные микросхемы

Доступная аппаратная поддержка может быть исследована в Visual SoftICE посредством команд CPU и PCI. Команда CPU выводит на экран данные PIC/APIC, в поддержку определенного CPU, а команда PCI предусматривает подсчет и идентификацию устройств на определенном типе шины.

(поддерживаются, также, некоторые команды ввода/вывода). Команда PCI может, также, выполнять специальное декодирование известных видов данных, чтение/запись пространства конфигурации устройства.

Пользователей AMD64 может заинтересовать исследование расположения набора микросхем HyperTransport на их 64-х разрядных AMD Opteron или AMD Athlon (владельцы машины SMP будут особо а этом заинтересованы).

```
SI>pci

Bus  Device  Function  VendorID  DeviceID  Name
-------------------------------------------------------------------------------
0    0       0         1022      7454      Host/PCI Bridge Device
0    1       0         1022      7455      PCI/PCI Bridge Device
0    6       0         1022      7460      PCI/PCI Bridge Device
0    7       0         1022      7468      PCI/ISA Bridge Device
0    7       1         1022      7469      IDE Mass Storage Controller
0    7       2         1022      746a      Serial Bus Controller
0    7       3         1022      746b      Bridge Device
0    7       5         1022      746d      Audio Multimedia Device
0    18      0         1022      1100      Host/PCI Bridge Device (HyperTransport)
0    18      1         1022      1101      Host/PCI Bridge Device (HyperTransport)
0    18      2         1022      1102      Host/PCI Bridge Device (HyperTransport)
0    18      3         1022      1103      Host/PCI Bridge Device (HyperTransport)
1    0       0         1002      5157      VGA PC Compatible Display Controller
2    0       0         1022      7464      USB (Universal Serial Bus) Serial Bus Controller
2    0       1         1022      7464      USB (Universal Serial Bus) Serial Bus Controller
2    5       0         10ec      8139      Ethernet Network Controller
2    7       0         10ec      8139      Ethernet Network Controller
```

Рисунок 6. Пример вывода команды SMP

Например, на Рисунке 6 показано, как можно провести идентификацию моста HyperTransport посредством PCI. Жирным выделены строки, описывающие интересующие нас устройства. После того, как мост идентифицирован, можно применить function 1, чтобы увидеть память HyperTransport, адресацию и ввод/вывод (см. Рисунок 7) .

```
SI>pci 0.18.1
------------------------------------------------------
Bus            : 0
Device         : 18
Function       : 1
VendorID       : 1022 : Advanced Micro Devices
DeviceID       : 1101
Name           : Host/PCI Bridge Device (HyperTransport)
Command        : 0 (0) :
Status         : 0 (0) :
Capabilities   :
Revision       : 0
CacheLineSize  : 0
LatencyTimer   : 0
HeaderType     : 80
BIST           : 0
BaseAddr0      : 0
BaseAddr1      : 0
BaseAddr2      : 0
BaseAddr3      : 0
BaseAddr4      : 0
BaseAddr5      : 0
CIS            : 0
SubVendorID    : 0
SubSystemID    : 0
RomBaseAddr    : 0
InterruptLine  : 0
InterruptPin   : 0
MinGrant       : 0
MaxLatency     : 0


HyperTransport DRAM Address Map

Region  Node  Starting Address  Ending Address    Interleave Enable  Interleave Select  Flags
-------------------------------------------------------------------------------------------
0       0     0000001101000000  0000000000000000  No Interleave      0                  W
1       0     0000000600000000  0000000080000000  No Interleave      0
2       0     0000000000000000  0000000000000000  No Interleave      0
3       0     0000000000000000  0000000000000000  No Interleave      0
4       0     0000000000000000  0000000000000000  No Interleave      0
5       0     0000000000000000  0000000000000000  No Interleave      0
6       0     0000000000000000  0000000000000000  No Interleave      0
7       0     0000000000000000  0000000000000000  No Interleave      0

HyperTransport Memory Mapped I/O Address Map

Region  Starting Address  Ending Address    Flags  HostBridge Node  HostBridge HT Link
-----------------------------------------------------------------------------------
0       0000000000000000  000000001f000000  RW     0                0
1       0000000020000000  0000000000000000         1                0
2       0000000020000000  0000000000000000         2                0
3       0000000020000000  0000000000000000         3                0
4       0000000020000000  0000000000000000         4                0
5       0000000020000000  0000000000000000         5                0
6       0000000020000000  0000000000000000         6                0
7       0000000020000000  0000000000000000         7                0

HyperTransport PCI I/O Address Map

Region  Starting Address  Ending Address    Flags  HostBridge Node  HostBridge HT Link
-----------------------------------------------------------------------------------
0       0000000000000e00  0000000000000e1f  RW     0                0
1       0000000000000d80  0000000000000dff  RW     0                0
2       0000000000000e20  0000000000000e3f  RW     0                0
3       0000000000000000  0000000000000000         0                0
4       0000000000000000  0000000000000000         0                0
5       0000000000000000  0000000000000000         0                0
6       0000000000000000  0000000000000000  RW     0                0
7       0000000000000200  0000000000000fe0  RW     0                0
```

Рисунок 7. Детальная информация о HyperTransport мосте

4.5 Управление памятью

На различных платформах и архитектурах операционная система управляет большими участками адресуемой памяти по-разному. (Здесь учитываются и различия в версиях одной и той же операционной системы). Пользователей AMD64 может заинтересовать тот факт, что Windows Server использует 4-х уровневое табличное размещение страниц (four-level page-table layout). Это явление можно исследовать с помощью команды PAGE. Например, на Рисунке 8 представлены детали таблицы страниц.

```
SI>r
       rax 0000000000000001        rbx 0000000000000000        rcx fffffadff37a3lb0
       rdx 00000000000003c3        rsi fffffadff37204da        rdi fffffadffladf626
       rip fffffadfef8fa4d4        rsp fffff8000038aae8        rbp 0000000000000000
    rflags 0000000000000202         r8 fffffadfflaelale         r9 0000000000000032
       rl0 fffffadff37a3lb0        rll fffffadfflaela32        rl2 0000000000000001
       rl3 fffffadff3852010        rl4 0000000000000100        rl5 0000000000000010
        cs 0010                     ds 002b                     es 002b
        fs 0053                     gs 002b                     ss 0018

SI>page rip

Entry         Physical  Data      Physical Page(PPN)  Attributes
-------------------------------------------------------------------
PXE           0         2e00063   2e00000 (2e00)      P A RW S
-PDPE         0         3c73063   3c73000 (3c73)      P A RW S
--PDE         0         3df0163   3df0000 (3df0)      P A RW S
---PTE        19b564d4  19b56063  19b56000 (19b56)    P A D RW S
```

Рисунок 8. Дамп страницы памяти

4.6 Таблицы прерывания и дескрипторы

Операционная система управляет по-разному прерыванием на различных платформах. В некоторых архитектурах (таких как Intel IA64) вектора прерываний представлены на аппаратном уровне и не могут редактироваться ни чем другим, кроме как самым низким уровнем операционной системы, а именно, уровнем аппаратных абстракций (HAL), уровнем процессорной абстракции (PAL) или уровнем системной абстракции (SAL). Все остальное – это программные таблицы, поддерживаемые операционной системой в тесной связи с аппаратными средствами, и полностью доступные драйверам (x86 иAMD64). Visual SoftICE пытается абстрагировать их для всех 32-и 64-битных платформ посредством команды IT (также известной, как команда IDT), результат работы которой представлен на Рисунке 9.

```
SI>it

Interrupt Descriptor Table - CPU 0, Base Address: fffff80000383060, Limit: fff
Count: 256

Vector    Type                      IST  Present  DPL  SrvRoutine
-------------------------------------------------------------------------------------------
0         e; 64bit Interrupt Gate   0    yes      0    fffff800010915e0 (ntoskrnl!_Section.text+905e0)
1         e; 64bit Interrupt Gate   0    yes      3    fffff80001091690 (ntoskrnl!_Section.text+90690)
...
```

Рисунок 9. Результат работы команды IT

В равной степени различные аппаратные средства и операционная система обеспечивают 64-битовые версии глобальных и локальных дескрипторов, что можно проследить на этих платформах посредством команды GDT (см. Рисунок 10)

```
SI>gdt

Global Descriptor Table - Base Address: fffff80000383000, Limit: 5f
Count: 7

Selector  Type                                  Address           Limit     DPL  Granularity  Present
-----------------------------------------------------------------------------------------------------
10        64bit Code: Execute/Readable (accessed)  0              0         0    NA           P
18        Data: Read-Write (accessed)              0              0         0    Byte         P
23        Code: Execute/Readable                   0              ffffffff  3    Page         P
2b        Data: Read-Write (accessed)              0              ffffffff  3    Page         P
33        64bit Code: Execute/Readable (accessed)  0              0         3    NA           P
40        64bit TSS (busy)                         fffff80000384060  68     0    NA           P
53        Data: Read-Write (accessed)              ffdf6000       fff       3    Byte         P
```

Рисунок 10. Результат работы команды GDT

4.7 Эмуляция

Подобно эмуляции 16-битовых вычислений на 32-битовых платформе «Windows 16 on Windows 32» (WOW32), компания Microsoft решила обеспечить решение WOW64 для выполнения 32-битовых приложений в рамках 64-битовой операционной системы. На различных аппаратных архитектурах эта эмуляция реализована по-разному. Исследовать особенности эмуляции на сегодняшний день можно при помощи точек останова и просмотра стека (поддержка WOW64 в текущей альфа-версии Visual SoftICE для AMD64 реализована еще не полностью) Подробнее о WOW64 можно узнать на сайте Microsoft (http://msdn.microsoft.com/).

5. Заключение
Я бы посоветовал вам просмотреть полную документацию по архитектуре, предоставляемую AMD, и применить полный потенциал DriverStudio, чтобы быстро и продуктивно справиться с задачей, независимо от того, выполняется разработка для одной или нескольких платформах.

Статью Стивена Смита подготовила Ерофеева Вера