SoftICE

Руководство пользователя

Перевод руководства "Using SoftICE" фирмы Compuware Corp.

Windows NT™
Windows® 98
Windows® 95
Windows® 3.1
DOS

Оглавление

Некоторые книги предназначены для того, чтобы с ними ознакомиться,
другие для того, чтобы их проглотить,
и лишь немногие, чтобы их обдумывать и усваивать.
Сэр Френсис Бэкон

Some books are to be tasted, others to be swallowed,
and some few to be chewed and digested.
Sir Francis Bacon

Предисловие

Цель данного руководства

SoftICE является мощным отладчиком общего назначения, с помощью которого можно отлаживать практически любые типы кодов, включая исполняемые файлы, драйверы устройств, DDL, OCX, а также статические и динамические VxD. Данное руководство описывает, как установить и использовать SoftICE для загрузки и отладки программ под управлением Windows 95 и Windows NT. Так как многие программисты предпочитают учиться, непосредственно экспериментируя с программой, в данное руководство включен раздел, который посвятит Вас в основы отладки.

Для кого предназначено данное руководство

Это руководство предназначено для программистов, которые хотят использовать SoftICE для отладки программ для операционных систем Windows 95 и Windows NT.

Организация руководства

Руководство пользователя организовано следующим образом:

- Глава 1. "Знакомство с SoftICE"
  Кратко описывает компоненты пакета и их характеристики, и объясняет, как можно связаться с Центром технической поддержки фирмы NuMega.
- Глава 2. "Установка SoftICE"
  Перечисляет аппаратные и программные требования SoftICE и поясняет, как установить отладчик для операционных систем Windows 95 и Windows NT.
- Глава 3. "Наставник по SoftICE"
  Рассказывает об основах отладки программ, включая трассировку кода, просмотр содержимого локальных переменных и структур, установку различных контрольных точек и просмотр таблиц с символьной информацией.
- Глава 4. "Загрузка кода в SoftICE"
  Объясняет, как использовать утилиту Symbol Loader для загрузки в отладчик различных типов программ.
- Глава 5. "Интерфейс SoftICE"
  Описывает интерфейс SoftICE.
- Глава 6. "Использование SoftICE"
  Рассказывает о способах перехвата ошибок исполнения программ, контекстах адресов, использовании команд INT 0x41 . (точка) и переходах из кольца 3 в кольцо 0.

- Глава 7. "Использование прерываний"
  Объясняет, как устанавливать на исполняемые команды, на любые обращения к памяти, на прерывания, на чтение или запись в порты ввода/вывода.
- Глава 8. "Использование выражений"
  Описывает правила составления выражений при работе с контрольными точками.
- Глава 9. "Загрузка символов для системных компонентов"
  Рассказывает, как загружать экспортируемые .DLL и .EXE-файлами символы, и как использовать в SoftICE файлы с отладочной информацией для Windows NT.
- Глава 10. "Использование SoftICE с модемом"
  Описывает, как установить модемную связь для управления SoftICE с удаленного компьютера.
- Глава 11. "Настройка SoftICE"
  Объясняет, как использовать конфигурационный файл SoftICE для настройки окружения отладчика, предварительной загрузки отладочной информации и настройки удаленной отладки, как модифицировать клавиатуру для Ваших нужд и создавать макрокоманды, и описывает варианты решений некоторых возникающих проблем.
- Глава 12. "Исследование Windows NT"
  Краткий обзор строения операционной системы Windows NT.
- Приложение A "Сообщения об ошибках"
  Поясняет выдаваемые в процессе работы SoftICE сообщения.
- Приложение B. "Поддерживаемые видеоадаптеры"
  Здесь приведен список поддерживаемых отладчиком видеоадаптеров.
- Приложение C. "Устранение проблем SoftICE"
  Объясняет, каким образом можно решить некоторые возникающие в процессе работы проблемы.
- Словарь

Принятые обозначения

В данном руководстве приняты следующие обозначения:

Обозначение	Описание
Enter	Указывает, что Вам следует набрать текст и нажать клавишу RETURN или щелкнуть по кнопке OK.
курсив	Обозначает самую разную информацию, например, *имя-библиотеки.*
`такой текст`	Используется в инструкциях и в примерах программ, чтобы указать символы, которые Вам следует набрать на клавиатуре.
МАЛЫЕ ПРОПИСНЫЕ	Обозначают элементы пользовательского интерфейса, такие, например, как кнопки или меню.
ПРОПИСНЫЕ	Обозначают имена директорий и файлов, ключевые слова и акронимы.

Как пользоваться этим руководством

В следующей таблице предлагаются различные варианты использования данного руководства на основании Вашего опыта отладки приложений:

Опыт отладки	С чего начать
Опыт использования отладчиков отсутствует	Изучение главы 3 "*Наставник по SoftICE*"
Имеется опыт использования других отладчиков	Прочитать главу 4 "*Загрузка кода в SoftICE*" и главу 5 "*Управление SoftICE*"
Опыт использования ранних версий SoftICE	Прочитать главу 4 "*Загрузка кода в SoftICE*", а затем просмотреть главу 5 "*Управление SoftICE*" об использовании мыши и окон слежения и локальных переменных

Другая полезная документация

В дополнение к данному руководству компания NuMega выпускает следующую документацию по отладчику SoftICE:

- Справочник по командам SoftICE
  В справочнике в алфавитном порядке описаны все команды SoftICE. Дается полное описание синтаксиса команд и выводимая ими информация, а также приведены примеры, поясняющие их использование.
- Встроенная система подсказок
  Комплекс SoftICE оснащен контекстно-зависимой системой подсказки для программы Symbol Loader, а также строкой помощи для команд отладчика.
- Документация в электронном виде
  Как руководство пользователя, так и справочник по командам SoftICE доступны в электронном виде. Для их использования необходимо запустить программу Acrobat Reader и открыть файлы SI30UG.PDF (данное руководство) или SI30CR.PDF (Справочник по командам).

Для великого эксперимента чрезвычайно важна одна вещь — сам экспериментатор.
Уолтер Бейджхот

To a great experience one thing is essential, an experiencing nature.
Walter Bagehot

1. Знакомство с SoftICE

Общая характеристика

Приложение SoftICE выпускается как для операционной системы Windows 95, так и для Windows NT, и состоит из отладчика уровня ядра SoftICE и утилиты Symbol Loader. Собственно программа SoftICE (далее просто — SoftICE) представляет собой многоцелевой отладчик, который может быть использован для отладки практически любых типов программ, включая обработчики прерываний и драйверы ввода/вывода. Утилита Symbol Loader загружает отладочную информацию Вашего модуля для использования ее в SoftICE, помогает составить строку инициализации и позволяет сохранять протокол работы SoftICE в файл. В последующих разделах данной главы дается краткое описание обоих программ.

SoftICE

SoftICE сочетает в себе мощь аппаратного отладчика с легкостью использования символьного отладчика. Он предоставляет возможность установки прерываний, подобных аппаратным, которые сохраняются, несмотря на все манипуляции операционной системы с виртуальной памятью. SoftICE позволяет просматривать исходные тексты отлаживаемых Вами программ и обращаться к локальным и глобальным переменным по их символическим именам.

SoftICE обладает следующими привлекательными особенностями:

- Отладка на уровне исходных кодов 32-битных приложений (Win32), драйверов устройств Windows NT (как пользовательского уровня, так и уровня ядра), драйверов Windows 95, VxD, 16-битных Windows программ и DOS программ.
- Отладка практически любых типов кодов, включая обработчики прерываний и ядро операционных систем Windows NT и Windows 95.
- Установка прерываний на обращение к памяти как по чтению, так и по записи, на чтение/запись в порты ввода/вывода и на аппаратные прерывания.
- Установка прерываний на сообщения Windows.
- Задание условий возникновения прерываний и последовательности действий при их возникновении.
- Показывает промежуток времени, прошедший до момента возникновения прерывания, используя счетчик процессора Pentium.
- Отладка ядра операционной системы на одной машине.
- Выдача внутренней информации операционной системы, такой как:
  - ◊ полная информация о процессах и потоках;
  - ◊ распределение виртуальной памяти процесса;

- ◊ точки входа ядра;
- ◊ полная информация об объектах типа драйвер и устройство;
- ◊ параметры кучи Win32;
- ◊ кадры структурной обработки исключений;
- ◊ экспорт DLL.

- Позволяет использовать команду WHAT для идентификации имени или выражения, если они могут быть приведены к одному из известных отладчику типов.
- Автоматическое появление экрана SoftICE при возникновении необрабатываемого исключения.
- Использование модема для подключения SoftICE к удаленному компьютеру. Это позволяет изучить проблемы в работе удаленного компьютера, например, причины краха системы.
- Поддержка набора инструкция MMX.
- Создание пользовательских макрокоманд.

Как устроен отладчик SoftICE

SoftICE для Windows 95 состоит из 2 файлов виртуальных драйверов (.VxD), а SoftICE для Windows NT - из 2 драйверов устройств уровня ядра.

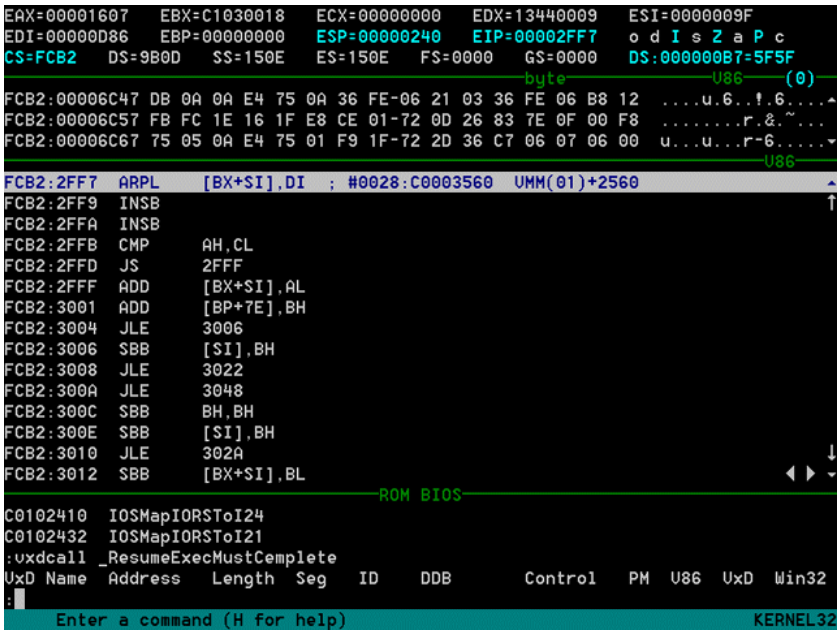
Windows 95 (VxD)	Windows NT (драйверы уровня ядра)	Описание
WINICE.EXE	NTICE.SYS	Обеспечивает собственно отладку
SIWVID.386	SIWVID.386	Обеспечивает поддержку видеосистемы Вашего компьютера

Замечание: SoftICE для Windows NT должен загружаться самой операционной системы, так как он встраивается как драйвер устройств. Следовательно, Вы не можете отлаживать такие составные части Windows NT, как фрагменты кодов загрузки драйверов DriverEntry, подпрограммы инициализации HAL и NTOSKRNL, и коды загрузчиков Windows NT или NTDETECT.

Интерфейс пользователя SoftICE

Вне зависимости от платформы SoftICE имеет общий интерфейс для отладки приложений. Пользовательский интерфейс разработан для обеспечения максимального удобства работы, но не в ущерб устойчивости системы. Для того, чтобы обеспечить возможность появления экрана SoftICE в любой момент без нарушения работы системы, отладчику необходимо непосредственно взаимодействовать с аппаратным обеспечением. Исходя из этого, SoftICE использует полноэкранный символьный режим вывода информации[†]:

[†] Начиная с версии 3.2, благодаря включению в пакет "универсального" видеодрайвера, SoftICE может работать в текущем видеорежиме без переключения в текстовый режим. (*Здесь и далее примечания переводчика.*)



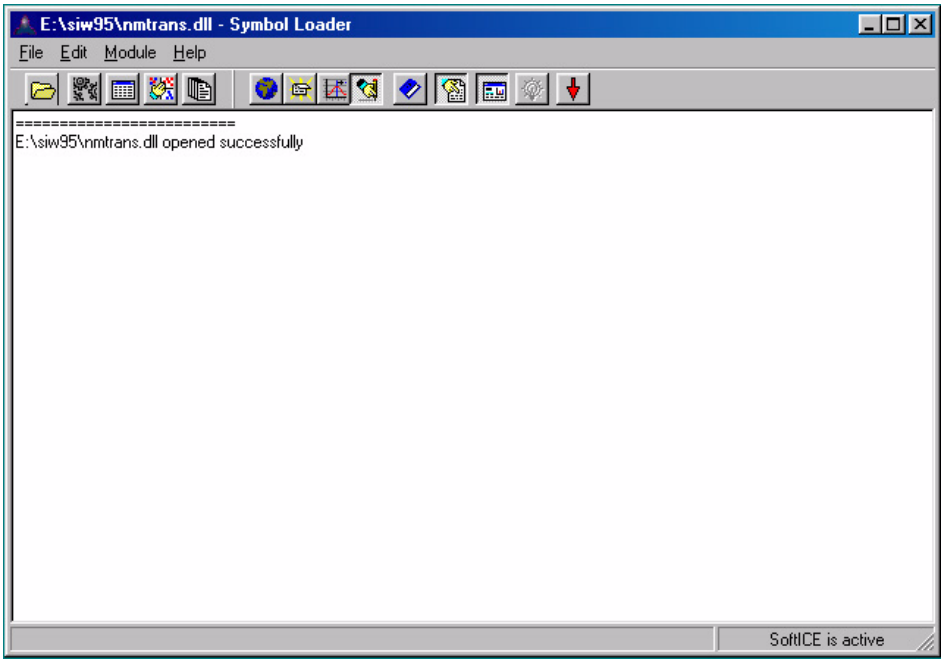
Подробно расположение информации на экране SoftICE описано в главе 5 "*Управление SoftICE*".

Symbol Loader

Утилита Symbol Loader (Символьный загрузчик) является программой, которая позволяет извлекать отладочную информацию из Ваших, .EXE-, .DLL- и .OCX-файлов, статических и динамических виртуальных драйверов (VxD), драйверов устройств и загружать ее в SoftICE. Утилита также позволяет:

- настраивать типы и количество загружаемой в отладчик информации в соответствии с Вашими потребностями;
- автоматически загружать приложение и устанавливать прерывание на точку его входа;
- сохранять результаты Вашей работы в файл.

Общий вид окна Symbol Loader показан на следующем рисунке:



Кроме графического режима работы, Symbol Loader поддерживает интерфейс командной строки, что позволяет использовать многие его свойства в окне DOS и

автоматизировать большинство его функций. Кроме того, вместе с SoftICE поставляется дополнительная утилита командной строки (NMSYM), с помощью которой можно загружать отладочную информацию в пакетном режиме.

Поддержка пользователей

По общим вопросам

Служба поддержки пользователей фирмы NuMega позволит Вам получить ответы на любые вопросы, касающиеся обновления программы, серийных номеров, и связанные с поставками программных продуктов. Обращайтесь в любые дни с понедельника по пятницу с 8^{30} до 5^{30} по восточно-американскому времени (EST) по телефонам:

- 888-283-9896 — для жителей США и Канады;
- +1 603 578 8103 — для жителей других стран.

По техническим вопросам

Центр технической поддержки фирмы NuMega может помочь Вам в разрешении любых вопросов, начиная с установки продукта, и кончая различными техническими проблемами.

Прежде чем обратиться на фирму, внимательно прочтите соответствующие разделы документации и ReadMe файлов.

С центром технической поддержки можно связаться следующими способами:

E-mail: Укажите серийный номер используемого Вами продукта, как можно подробнее опишите возникшие у Вас проблемы и направьте письмо по адресу Tech@numega.com.

WWW Наша база знаний доступна по адресу www.numega.com, раздел Support (Поддержка).

Телефон Поддержка по телефону является платной услугой (помощь в вопросах установки и запуска продуктов предоставляется бесплатно) и доступна с понедельника по пятницу с 8^{30} до 5^{30} по восточно-американскому времени (EST). При обращении в службу имейте наготове номер версии продукта и его серийный номер. Звоните по телефонам:
888 NUMEGA-S — для жителей США и Канады;
+1 603 578 8100 — для жителей других стран.

Факс Укажите серийный номер используемого Вами продукта, как можно подробнее опишите возникшие у Вас проблемы и направьте факс по номеру 603 578 8401.

Прежде чем обращать в службы поддержки фирмы NuMega, пожалуйста подготовьте следующую информацию:

- Номер версии и серийный номер продукта.
- Конфигурацию системы: версия операционной системы, параметры сети, количество оперативной памяти, переменные окружения и путь.
- Название и версию используемого Вами компилятора и линкера и параметры, с которыми они запускаются.
- Описание проблемы; параметры инициализации, сообщения об ошибках, дамп стека, содержание всех диагностических сообщений.
- Если проблема возникает постоянно, то опишите способ ее воспроизведения.

Любое великое дело должно иметь свое начало,
однако, лишь тщательное доведение его до логического конца
приносит желанную победу.
Сэр Френсис Дрейк

There must be a beginning of any great matter,
but the continuing unto the end until it be thoroughly finished
yields the true glory.
Sir Francis Drake

2. Установка SoftICE

Введение

В настоящей главе объясняется, как правильно установить SoftICE под управлением Windows 95 или Windows NT. Учтите, что программный продукт SoftICE для Windows 95 может быть установлен только под управлением операционной системы Windows 95, а SoftICE для Windows NT — только под управлением Windows NT.

Требования к аппаратному и программному обеспечению

В зависимости от используемой Вами операционной системы SoftICE предъявляет к аппаратному и программному обеспечению следующие требования:

Аппаратные или программные требования	Windows 95	Windows NT
Процессор	Процессор Intel x86 Хронометраж прерываний, прерывание по вводу/выводу в 0 кольце защиты и полная поддержка мыши требует наличия процессоров Intel Pentium или Intel Pentium Pro	Процессор Intel x86 Хронометраж прерываний, прерывание по вводу/выводу в 0 кольце защиты и полная поддержка мыши требует наличия процессоров Intel Pentium или Intel Pentium Pro SoftICE поддерживает работу однопроцессорных систем или одного процессора в мультипроцесорных системах
Операционная система	Windows 95	Windows NT 3.51 (вып. 1057) Windows NT 4.0 (вып. 1381)
Оперативная память (RAM)	Минимально 16 Мб Рекомендуется 32 Мб	Минимально 32 Мб Рекомендуется 64 Мб

Продолжение на следующей странице

Аппаратные или программные требования	Windows 95	Windows NT
Дисковое пространство	5 Мб (без установки программы Adobe Acrobat Reader) 7.5 Мб с установкой программы Adobe Acrobat Reader (для просмотра руководства в электронной форме)	5 Мб (без установки программы Adobe Acrobat Reader) 7.5 Мб с установкой программы Adobe Acrobat Reader (для просмотра руководства в электронной форме)
Мышь (не обязательно)	Последовательный или PS/2 порты (совместимость с Microsoft)	Последовательный или PS/2 порты (совместимость с Microsoft)
Видеосистема	Один из следующих вариантов: • один видеоадаптер и монитор • дополнительные монохромная видеокарта и монитор • удаленный компьютер, подключенный через последовательный порт • дополнительные VGA видеокарта и монитор	Один из следующих вариантов: • один видеоадаптер и монитор • дополнительные монохромная видеокарта и монитор • удаленный компьютер, подключенный через последовательный порт • дополнительные VGA видеокарта и монитор

Истинные потребности в оперативной памяти определяются количеством загружаемых одновременно символьных таблиц и файлов исходных кодов, так как всю эту информацию SoftICE держит в памяти, чтобы в процессе работы не обращаться к файловой системе.

Варианты вывода информации SoftICE на экран

В зависимости от типа отлаживаемого программного продукта и от Ваших привычек можно использовать любую из следующих конфигураций видеосистем:

- один видеоадаптер и один монитор;
- дополнительные монохромная видеокарта и монитор;
- удаленный компьютер, подключенный через последовательный порт;
- дополнительные VGA видеоадаптер и монитор.

Каждая из перечисленных выше конфигураций подробно описывается в следующих разделах.

Один видеоадаптер и монитор

Стандартная конфигурация компьютера с одним видеоадаптером и одним монитором используется, когда нет необходимости в дополнительной гибкости вывода отлаживаемой информации. В активизированном состоянии SoftICE использует монитор для вывода информации о текущем состоянии системы и приложения. Однако, в этом случае Вы лишены возможности одновременно видеть состояние Вашего приложения (*прим. перев.* — смотрите примечание на странице 10).

Дополнительные монохромная видеокарта и монитор

Использование дополнительного монохромного видеоадаптера типа MDA (Monochrome Display Adapter) или Hercules и подключенного к нему монохромного монитора позволит Вам одновременно видеть как отладочную информацию, так и состояние отлаживаемого Вами приложения. В такой конфигурации главный монитор используется приложением, а отладочная информация выводится на монохромный монитор. Однако, в такой ситуации размер окна SoftICE ограничен 25 строками.

Замечание: Большинство существующих видеоадаптеров может одновременно работать с монохромными картами, однако, имеются и исключения. За дополнительной информацией о возможностях Вашего видеоадаптера обращайтесь к соответствующей документации.

Применение дополнительных видеоадаптера и монитора особенно полезно в следующих ситуациях:

- Работа с видеоадаптерами, не поддерживаемыми отладчиком.
  Если SoftICE не поддерживает работу с Вашим видеоадаптером или Вы разрабатываете для него новый драйвер, то для вывода отладочной информации Вы можете использовать такую конфигурацию.
- Отладка видеодрайверов.
  Когда SoftICE берет на себя управление видеоадаптером, он может изменить содержимое некоторых его регистров. Использование дополнительной видеокарты позволит Вам избежать такой ситуации.

Удаленный компьютер, подключенный через последовательный порт

Подключение через последовательный порт удаленного компьютера также позволит Вам одновременно видеть как работу Вашего приложения, так и всю отладочную информацию. При этом основной компьютер используется приложением, а удаленный демонстрирует работу SoftICE.

Проще говоря, удаленный компьютер используется как неинтеллектуальный терминал для вывода информации и ввода команд с клавиатуры. Однако, в таких условиях SoftICE не поддерживает на удаленном компьютере работу мыши.

Замечание: Удаленный компьютер должен обеспечивать работу MS-DOS.

Описываемая конфигурация может быть полезна при следующих обстоятельствах:

- Работа с видеоадаптерами, не поддерживаемыми отладчиком.
  Если SoftICE не поддерживает работу с Вашим видеоадаптером или Вы разрабатываете для него новый драйвер, то для вывода отладочной информации Вы можете использовать такую конфигурацию. Эта возможность особенно полезна при работе с лаптопами.
- Отладка видеодрайверов.
  Когда SoftICE берет на себя управление видеоадаптером, он может изменить содержимое некоторых его регистров. Использование удаленного компьютера позволит Вам избежать такой ситуации.
- Отладка драйверов клавиатуры
  SoftICE использует тот же драйвер клавиатуры, что и приложение, поэтому нельзя исключить вероятность возникновения самых непредвиденных ситуаций. Применение удаленного компьютера позволит Вам избежать этого.

Дополнительные VGA видеокарта и монитор

Использование дополнительных VGA-адаптера и монитора обеспечивает Вам максимальную гибкость работы. Основной монитор используется для работы Вашего приложения, в то время как SoftICE показывает отладочную информацию дополнительном мониторе.

Внимание: Лишь немногие видеоадаптеры разработаны для поддержки мультидисплейных систем. Тестирование одновременной работы двух видеоадаптеров может испортить Ваш компьютер. Поэтому, если в документации на видеоадаптер нет указаний о такой возможности, лучше всего считать, что он ее не поддерживает.

Обратитесь к документации на Ваш видеоадаптер за дополнительной информацией, каким образом отключить на ней поддержку режима VGA.

Подготовка к установке

Прежде чем устанавливать пакет SoftICE, необходимо выяснить следующие вопросы:

1 Если Вы собираетесь устанавливать SoftICE под управлением Windows NT, то проверьте, обладаете ли Вы полномочиями администратора.

2 Определите, какую из приведенных ниже конфигураций видеосистемы Вы будете использовать для работы:

- ◊ Один видеоадаптер и один монитор
  Определите фирму-производителя видеоадаптера и его модель.
- ◊ Дополнительные видеокарта и монитор
  Определите производителя и модель основного видеоадаптера.
- ◊ Удаленный компьютер
  Определите производителя и модель видеоадаптера на основном компьютере.
- ◊ Дополнительный VGA видеоадаптер
  Проверьте, могут ли оба видеоадаптера работать одновременно.

Совет: Для определения производителя и модели видеоадаптера используйте окно контрольной панели.

3 Определите тип используемой Вами мыши (последовательный или PS/2). Если Вы используете мышь с последовательным интерфейсом, то определите порт ее подключения (COM1 или COM2).

4 Завершите все открытые программы.

Установка

Пакет SoftICE поставляется на одном компакт-диске. Приведенные ниже инструкции поясняют, как правильно установить пакет под управлением Windows 95 или Windows NT.

1 Поместите компакт-диск в дисковод и запустите программу установки (setup.exe).

2 Выберите пункт "INSTALL SOFTICE".

3 Введите в окне регистрации (Registration window) Ваше имя и название компании (Name и Company, соответственно), и серийный номер пакета (Serial number). Серийный номер указан в Вашей регистрационной карте.

4 В окне "Select Install Directory" выберите каталог, куда Вы собираетесь устанавливать пакет.

По умолчанию под управлением Windows 95 пакет устанавливается в директорию "C:\SIW95", а под Windows NT — в "C:\NTICE". Если выбранная Вами директория не существует, то мастер установки создаст ее самостоятельно.

5 В окне "Display Adapter Selection" (выбор видеоадаптера), выберите один из следующих вариантов:

- ◊ Для использования SoftICE в стандартной конфигурации с одним видеоадаптером и монитором выберите производителя (Manufacturer) и модель (Model) видеоадаптера. Если такого адаптера в списке нет, то выберите видеоадаптер с таким же набором микросхем, как указано в разделе "COMPATIBILITY" (совместимость). Если же и такого не существует, то выберите "STANDARD VGA (640X480 PIXELS)", а по окончании установки обратитесь к приложению B "*Поддерживаемые видеоадаптеры*".
- ◊ Для использования SoftICE с дополнительным монохромным монитором, выберите "DISPLAY SOFTICE ON ATTACHED MONOCHROME MONITOR" (Вывод SoftICE на дополнительном монохромном мониторе), а затем выберите производителя (Manufacturer) и модель (Model) основного видеоадаптера. Если такого адаптера в списке нет, то выберите видеоадаптер с таким же набором микросхем, как указано в разделе "COMPATIBILITY" (совместимость). Если же и такого не существует, то выберите "STANDARD VGA (640x480 pixels)".
- ◊ Для того, чтобы использовать SoftICE с удаленным компьютером, выберите производителя (Manufacturer) и модель (Model) основного видеоадаптера. Если такого адаптера в списке нет, то выберите видеоадаптер с таким же набором микросхем, как указано в разделе "COMPATIBILITY" (совместимость). Если же и такого не существует, то выберите "STANDARD VGA (640X480 PIXELS)".
- ◊ Чтобы использовать SoftICE с дополнительной VGA видеокартой, выберите "STANDARD VGA (640X480 PIXELS)". В этом случае SoftICE игнорирует основной видеоадаптер и использует дополнительную VGA видеокарту и монитор.

Замечание: Если после установки SoftICE необходимо сменить тип или модель видеоадаптера, пользуйтесь утилитой "DISPLAY ADAPTER SETUP"

6 Если параметры видеоадаптера, выбранного Вами в окне "Display Adapter Selection", совпадают с параметрами, установленными в Windows 95 или Windows NT, то нажмите кнопку "TEST" для их окончательной проверки. В следующей таблице приведены условия, когда необходимо проводить дополнительное тестирование, а когда нет.

Ситуация	**Тестировать**	**Не тестировать**
Для Windows и для SoftICE выбран "Standard VGA" видеоадаптер.	X	
Видеоадаптер для Windows совпадает либо по фирме-производителю и модели, либо по набору микросхем с выбранным для SoftICE.	X	
Для SoftICE выбран адаптер VGA, а в Windows установлен какой-либо иной видеоадаптер, не стандартный VGA.		X

Внимание: Сохраните любую незавершенную работу перед тестированием видеоадаптера. Если для SoftICE выбран несоответствующий видеоадаптер, то весьма вероятно, что по окончании тестирования экран не будет восстановлен должным образом. Кроме того возможно, хотя и мало вероятно, что тестирование приведет к краху всей системы.

Если тестирование пройдет успешно, то в течение примерно 5 секунд экран восстановит свое прежнее состояние. Если же установки SoftICE несовместимы с указанным Вами типом видеоадаптера, то экран будет черным, или будет иметь необычный вид. Если Вам не подойдет ни один из перечисленных видеоадаптеров, то выберите режим "STANDARD VGA (640X480 PIXELS)", а по окончанию установки обратитесь к приложению B "*Поддерживаемые видеоадаптеры*".

7 Если Вы устанавливаете SoftICE под управлением Windows NT, то выберите один из вариантов в окне "Startup Mode Selection window" для определения способа загрузки SoftICE.

Windows NT Startup Mode Option	**Описание**
BOOT	Такие загрузочные драйверы, как драйвер контроллера диска и некоторые системные драйверы являются критическими на этапе загрузки. Чтобы гарантировать загрузку инициализационных файлов или файлов с отладочной информацией, SoftICE всегда загружается в качестве последнего загрузочного драйвера. Отсюда следует, что Вы не сможете отлаживать подпрограммы инициализации (DriverEntry) других загрузочных драйверов.
SYSTEM	Системные драйверы загружаются после загрузочных. В данный момент система все еще находится в режиме "Синего окна".
AUTOMATIC	Автоматические драйверы загружаются контроллером сервисов (Service Controller Manager) на последнем этапе запуска системы. Основная часть системы к данному моменту уже загружена. Если Вы хотите, что SoftICE запускался каждый раз при старте Windows NT, но Вы не собираетесь отлаживать драйверы устройств ядра системы, то можете загружать его на данном этапе.
MANUAL	SoftICE не запускается автоматически при старте системы. Этот ражим наиболее безопасен и гибок, однако, он исключает возможность отладки драйверов устройств на этапе загрузки. За дополнительной информацией обращайтесь к разделу "*Загрузка SoftICE вручную*" главы 4 на странице 37.

Замечание: Для изменения варианта загрузки SoftICE используйте утилиту "Startup Mode Setup".

8 В окне "Mouse Selection" укажите имеющуюся у Вас мышь:

- ◊ Serial (connected to COM1) — Последовательная (подключена к COM1).
- ◊ Serial (connected to COM2) — Последовательная (подключена к COM2)
- ◊ PS/2 compatible — PS/2-совместимая
- ◊ None — мышь отсутствует

Замечание: Чтобы выбрать иной тип мыши после установки программы, воспользуйтесь утилитой "Mouse Setup"

9 Если Вы устанавливаете SoftICE под управлением Windows 95, то выберите один из следующих вариантов в окне "System Configuration".

Windows 95 Startup Mode Option	**Описание**
Let Setup modify AUTOEXEC.BAT (Модифицировать AUTOEXEC.BAT)	Данная опция позволяет добавить в конец AUTOEXEC.BAT команду C:\SIW95\WINICE.EXE. В случае необходимости C:\SIW95\ заменяется на имя директории, где в действительности установлен SoftICE.
Save the required changes in AUTOEXEC.ICE (Записать необходимые изменения в AUTOEXEC.ICE)	Данная опция позволяет Вам предварительно оценить возможные изменения прежде, чем они будут сделаны. Для этого мастер установки копирует AUTOECE.BAT в файл AUTOEXEC.ICE и добавляет команду WINICE.EXE. После того, как Вы одобрите предлагаемые изменения, файл AUTOEXEC.BAT будет удален, а вместо него будет записан (и соответствующим образом переименован) AUTOEXEC.ICE.
Do not make any changes (Не делать никаких изменений)	При выборе данной опции изменения в файле AUTOEXEC.BAT не производятся. Вам потребуется самостоятельно изменить конфигурацию системы для того, чтобы загрузить SoftICE до WIN.COM. Обратитесь к разделу "*Настройка загрузки SoftICE под управлением Windows 95*" данной главы на странице 20.

10В окне "Start Copying Files" (Начать копирование файлов) нажмите кнопку "NEXT".

11В окне "Setup Complete" (Установка закончена) нажмите кнопку "FINISH" для перезагрузки компьютера. Если Вы не хотите в данный момент перезагружать компьютер, нажмите "NO, I WILL RESTART MY COMPUTER LATER" (Нет, я перезагружу компьютер позднее), а затем кнопку "FINISH".

12Прочтите файл README для ознакомления с последними изменениями программного продукта.

13Выполните необходимые процедуры после установки программы.

Действия после установки программы

После установки SoftICE необходимо выполнить следующие действия:

1 Если Вы предполагаете запускать SoftICE под управлением Windows NT на одном процессоре в мультипроцессорной системе, отредактируйте файл BOOT.INI, чтобы добавить новый режим загрузки. За дополнительной информацией обратитесь к разделу "*Настройка BOOT.INI для поддержки одного процессора в мультипроцессорной системе*" на странице 20.

2 Если Вы работаете под управлением Windows 95 и выбрали режим "LET SETUP MODIFY AUTOEXEC.BAT", операционная система не передаст управление SoftICE при выключении компьютера. Следовательно, Вы не сможете сохранить протокол выполнения прерываний. Чтобы избежать этого, установите в файле MSDOS.SYS параметр BootGUI=0. Просмотрите раздел "*Справочника по командам SoftICE*", посвященный команде BH, а также раздел "*Настройка загрузки SoftICE под управлением Windows 95*" на странице 20.

3 Если Вы работаете под управлением Windows 95 и выбрали режим "DO NOT MAKE ANY CHANGES", сконфигурируйте Windows 95 таким образом, чтобы загружать SoftICE перед WIN.COM. За дополнительной информацией обратитесь к разделу "*Настройка загрузки SoftICE под управлением Windows 95*" на странице 20.

4 Если Вы используете удаленный компьютер для вывода отладочной информации, сконфигурируйте SoftICE, как это описано в разделе "*Подключение удаленного компьютера через последовательный порт*" на странице 21.

5 Если Вы работаете под управлением Windows 95 и количество физической памяти на Вашем компьютере превышает 32 Мб, установите правильное значение параметра "TOTAL RAM" в настройках SoftICE. Смотрите раздел "*Изменение общих установок*" на странице 121.

6 Заполните регистрационную карточку и карту "NuMega Upgrade Subscription Program" и перешлите их фирме NuMega.
NuMega Upgrade Subscription Program предоставляет Вам возможность автоматического обновления пакета. Если Вы покупаете NuMega Upgrade Subscription Program через посредников, эта карта позволит фирме узнать, куда пересылать необходимые обновления. Если Вы не покупали участие в программе, то данная карта подробно расскажет Вам о ней и предоставит возможность дальнейшего в ней участия.

Настройка BOOT.INI для поддержки одного процессора в мультипроцессорной системе

Если Вы хотите использовать SoftICE для Windows NT для работы с одним процессором в многопроцессорной системе, то отредактируйте файл BOOT.INI добавив новый режим загрузки:

1 Файл BOOT.INI является скрытым файлом. Для того, чтобы сделать его доступным, используйте команду ATTRIBUTE:

```
ATTRIBUTE BOOT.INI -S -H -R
```

2 Запустите какой-либо текстовый редактор, например Notepad, и откройте файл.

3 В качестве отправной точки для создания нового режима загрузки используйте уже имеющуюся запись, которая запускает Вашу версию Windows NT. Например, у Вас существует запись такого типа (детали могут различаться в зависимости от используемой Вами версии операционной системы):

```
multi(0)disk(0)rdisk(0)partition(1)\winnt="Windows NT
Workstation Version 3.51"
```

4 Скопируйте ее в секцию "Operating Systems" и добавьте в конце строки значение **/onecpu**

```
multi(0)disk(0)rdisk(0)partition(1)\winnt="Windows NT
Workstation Version 3.51" /onecpu
```

5 Измените название нового режима загрузки (запись, помещенную между двойными кавычками) для облегчения дальнейшей работы:

```
multi(0)disk(0)rdisk(0)partition(1)\winnt="Windows NT
Workstation Version 3.51 One CPU" /onecpu
```

6 Перезапустите компьютер.

Настройка загрузки SoftICE под управлением Windows 95

SoftICE является отладчиком уровня ядра, поэтому он должен быть загружен до запуска WIN.COM. По умолчанию загрузка Windows 95 производится автоматически, не предоставляя Вам возможности самому вызывать программу WIN.COM. Если Вы отлаживаете приложения, то просто добавьте WINICE.EXE в конце Вашего AUTOEXEC.BAT файла. Если же Вы отлаживаете статические VxD или иные драйверы, то для оптимизации загрузки используйте описываемую ниже процедуру.

1 Выполните одно из перечисленных ниже действий для предотвращения автоматической загрузки Windows 95 и для запуска только ядра DOS.
- В начале загрузки нажмите клавишу F8. В появившемся стартовом меню выберите пункт "COMMAND PROMPT ONLY". Эту процедуру приходится выполнять при каждом запуске компьютера.
- Добавьте в конце файла AUTOEXEC.BAT команду PAUSE и при остановке загрузки нажмите Ctrl-C для перехода в режим DOS.
- Создайте пустой файл с именем WIN.BAT. При запуске Windows 95 она исполнит этот файл вместо WIN.COM и перейдет в командный режим DOS.
- Модифицируйте скрытый файл Windows 95 MSDOS.SYS (который фактически является текстовым .INI-файлом) следующим образом:
  ◊ Выполните команду ATTRIB для удаления атрибутов "скрытый", "системный" и "только для чтения" у файла MSDOS.SYS.
  ◊ Замените параметр BootGUI=1 на BootGUI=0 (или добавьте его, если данная запись в файле отсутствует)[†] .

2 Выполните команду WINICE.EXE для загрузки SoftICE, который, в свою очередь, запустит Windows 95.

3 Выполните Вашу работу.

4 Если необходимо перезапустить компьютер, то выберите режим "SHUT DOWN THE COMPUTER" или "RESTART THE COMPUTER" (соответственно, "ВЫКЛЮЧИТЬ КОМПЬЮТЕР" или "ПЕРЕЗАГРУЗИТЬ КОМПЬЮТЕР" в русской версии Windows 95). Любой иной режим выключения нарушает работу SoftICE. После вывода заключительного сообщения о выключении компьютера SoftICE переводит видеоадаптер в текстовый режим 80×25 и выводит подсказку командной строки DOS шага 1. Для продолжения отладки повторите шаг 2.

Подключение удаленного компьютера через последовательный порт

В данном разделе описывается, каким образом использовать команду SERIAL и утилиту SERIAL.EXE для подключения удаленного компьютера через последовательный порт. Утилита SERIAL.EXE является MSDOS-программой, которая позволяет использовать удаленный компьютер в качестве неинтеллектуального терминала для вывода сообщений и ввода информации с клавиатуры.

Замечание: Вы можете также использовать утилиту SERIAL.EXE для подключения компьютера через модем. Более подробно об этом описано в главе 10 "*Использование SoftICE с модемом*" на странице 117.

Для настройки работы SoftICE с удаленным компьютером необходимо выполнить следующее:

1 Если Вы работаете с операционной системой Windows 95, то укажите, какой последовательный порт (COM1, COM2, COM3 или COM4) вы собираетесь использовать для соединения компьютеров. Если же Вы используете Windows NT, то перейдите к этапу 2, так как SoftICE под управлением Windows NT самостоятельно определяет используемый порт.

Для указания номера последовательного коммуникационного порта сделайте следующее:

◊ Запустите утилиту Symbol Loader

◊ Выберите из меню "EDIT" пункт "SOFTICE INITIALIZATION SETTINGS" (Начальные установки SoftICE)

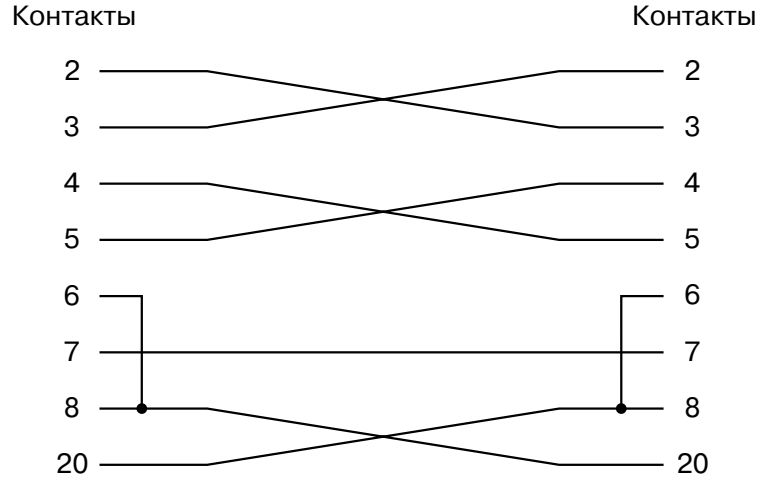
[†] После этого следует восстановить у файла MSDOS.SYS атрибуты "скрытый", "системный" и "только для чтения".

◊ Выберите необходимый последовательный порт.

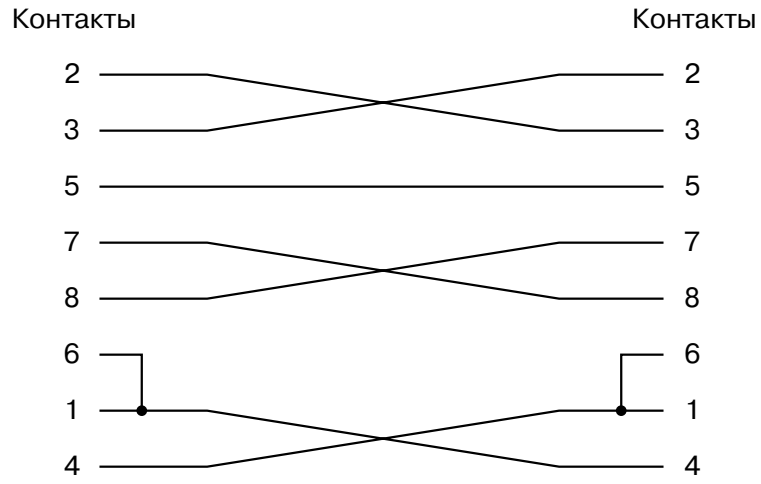
◊ Перезапустите компьютер.

За дополнительной информацией обращайтесь к разделу 11 главы "*Настройка удаленной отладки*" на странице 123.

2 Для соединения локального и удаленного компьютеров используйте нуль-модемный кабель. Распайка кабеля показана на следующих рисунках.



Распайка 25-контактного нуль-модемного кабеля



Распайка 9-контактного нуль-модемного кабеля

3 Скопируйте утилиту SERIAL.EXE из директории SoftICE на удаленный компьютер.

4 Запустите на удаленном компьютере утилиту SERIAL.EXE с параметрами:
`SERIAL.EXE [r] [com-port [baud-rate]]`,

например, `SERIAL.EXE 1 57000`

Параметр "r" используется при запуске утилиты в окне DOS под управлением Windows NT для отключения режима FIFO и установки в исходное состояние скорости обмена, стоповых битов и бита четности.

Параметр "`com-port`" указывает номер последовательного порта (от 1 до 4, по умолчанию 1). Если Вы указываете параметр "*`baud-rate`*" (скорость обмена), он должен совпадать со значением скорости обмена, заданным на локальном компьютере. Кроме того, при указании этого параметра необходимо задавать и "*`com-port`*". Если скорость обмена не указана, то два компьютера устанавливают ее автоматически.

После этого удаленный компьютер ожидает, пока локальный установит с ним связь.

5 Каждый раз при запуске SoftICE Вам необходимо выполнить на локальном компьютере команду:
`SERIAL [ON [com-port [baud-rate]] | OFF]`
например, `SERIAL ON 1 57000`
Параметр "ON" инициирует связь, тогда как параметр "OFF" завершает ее. Параметры "`com-port`" и "`baud-rate`" аналогичны описанным в шаге 2. За дополнительной информацией по команде SERIAL обращайтесь к "*Справочнику по командам SoftICE*".
Если связь установлена успешно, то на удаленном компьютере появляется окно SoftICE.

Совет: Чтобы не набирать эту команду каждый раз при запуске SoftICE, добавьте ее в инициализационную строку. Более подробную информацию смотрите в главе 11 "*Настройка SoftICE*" на странице 120.

Разрешение проблем с видеоадаптерами

SoftICE может быть использован для отладки практически любых частей системы, так как в процессе работы он не обращается к операционной системе. Чтобы это было возможным, SoftICE должен взаимодействовать с аппаратным обеспечением непосредственно. Хотя большинство устройств стандартизировано, однако, стандарт на режимы работы видеоадаптеров за пределами режимов стандартного VGA отсутствует. Следовательно, отладчику приходится обеспечивать собственную поддержку наиболее распространенных видеоадаптеров для безопасного переключения из графического режима в текстовый и обратно.

Вполне возможна ситуация, когда, несмотря на то, что мы поддерживаем используемый Вами видеоадаптер, тем не менее, у Вас он не работает. Это может происходить по нескольким причинам:

- Первая, и наиболее частая, причина заключается в том, что производитель внес в Вашу модель некоторые изменения. Например, он мог использовать модифицированный набор микросхем или применить другой RAMDAC.
- Вы работаете с разрешением и глубиной цветности, которые мы не тестировали.
- Используемый Вами драйвер видеоадаптера несколько отличается от того, который был использован нами, и он меняет настройку видеоадаптера таким образом, что это нарушает работу SoftICE.

Если используемый Вами видеоадаптер либо не поддерживается SoftICE, либо не работает должным образом, попробуйте следующее:

1 Проверьте все возможные варианты видеоадаптеров, совпадающие по модели и фирме-производителю или по набору микросхем.

2 С момента получения Вами пакета прошло некоторое время, в течение которого фирма NuMega могла дополнить поддержку пакетом Вашего видеоадаптера. Вы можете попробовать обновить Ваш пакет. Для этого:

◊ Переименуйте имеющийся у Вас видеодрайвер и сохраните его на всякий случай.
Под управлением Windows 95 файл siwvid.386 находится в директории установки SoftICE, а под управлением Windows NT файл siwvid.sys находится в подкаталоге \system32\drivers директории установки Windows NT.

◊ Загрузите свежий видеодрайвер (для Windows 95 это файл SIWVID.ZIP, а для Windows NT — SIWVIDNT.ZIP) с сервера FTP или с BBS фирмы NuMega:
FTP-сервер: ftp.numega.com, pub/tech/file name

BBS: (603)595-0386,file *имя-файла* (параметр "*имя-файла*" чувствителен к регистру, поэтому проследите, чтобы он был набран прописными символами)

- ◊ Распакуйте новый драйвер и поместите его в соответствующую директорию.
- ◊ Запустите утилиту Display Adapter Setup.

3 Пошлите письмо по электронной почте по адресу tech@numega.com, указав, какие проблемы вы имеете с видео. Добавьте также:

- ◊ Полное наименование производителя видеоадаптера и его модель и приблизительное время покупки.
- ◊ Разрешение и глубину цвета, которые Вы используете.
- ◊ Если известно, идентификационные номера использованного в видеоадаптере набора микросхем.
- ◊ Если известно, идентификационные номера использованного в видеоадаптере RAMDAC.

4 Попробуйте следующие варианты работы:

- ◊ Установите режим "STANDARD VGA (640X480 PIXELS)" как для SoftICE, так и для Windows. Ваш видеоадаптер будет в этом случае функционировать в режиме VGA, снимая таким образом все проблемы совместимости. Фирма NuMega рекомендует эту возможность в качестве решения на короткий срок.
- ◊ Используйте дополнительные монохромный видеоадаптер и монитор. Это популярное и относительно недорогое решение, которым пользуются многие разработчики.
- ◊ Используйте в качестве экрана для вывода информации SoftICE удаленный компьютер, подключенный через последовательный порт. Это решение наиболее подходит для лаптопов.
- ◊ Используйте дополнительные VGA видеокарту и монитор. Некоторые видеоадаптеры обеспечивают возможность одновременной работы нескольких видеокарт.

Более подробная информация по этим конфигурациям дана в разделе "*Варианты вывода информации SoftICE на экран*" странице 14.

Теперь мы обсудим борьбу за существование более детально.
Чарльз Дарвин

We will now discuss in a little more detail the struggle for existence.
Charles Darwin

3. Учебник SoftICE

Введение

Эта глава позволит Вам получить необходимый начальный опыт отладки приложений Windows, продемонстрирует основные этапы отладки приложений и драйверов. В течение этого урока вы узнаете, как:

- загрузить SoftICE;
- сформировать приложение;
- загрузить исходный код приложения и файлы с отладочной информацией;
- трассировать и пошагово выполнять исходные и машинные коды;
- просматривать локальные данные и структуры;
- устанавливать контрольные точки и точки останова;
- использовать команд SoftICE для исследования состояния приложения;
- работать с символическими именами и таблицами символов;
- изменять контрольные точки таким образом, чтобы они активизировались лишь при определенных условиях.

Каждый раздел в учебнике построен на основе предыдущего, так что их следует изучать по порядку.

В качестве примера в учебнике используется приложение GDIDEMO, предназначенное для демонстрации возможностей GDI операционных систем Windows. GDIDEMO размещен в каталоге \EXAMPLES\GDIDEMO Вашего CDROM'а, а также в каталоге \mstools\samples\win32\GDIDEMO. Если у Вас GDIDEMO находится на CDROM, то скопируйте его на жесткий диск.

Вы можете заменить пример другими, в том числе и Вашими своими собственными приложениями. Принципы отладки и функции SoftICE, использованные в данной главе, применимы в большинстве случаев.

Замечание: Примеры этого учебника основаны на работе под операционной системой Windows NT. При использовании Windows 95 могут быть незначительные различия.

Загрузка SoftICE

Если Вы запускаете SoftICE под Windows 95 или под Windows NT в режиме Boot, System или Automatic, то SoftICE автоматически запускается при каждом запуске или перезагрузке компьютера. В ручном (Manual) режиме под Windows NT SoftICE автоматически загружаться не будет.

Для загрузки SoftICE из Windows 95 введите команду WINICE. Для загрузки SoftICE под Windows NT выполните одно из перечисленных ниже действий:

- Выберите в меню пункт "START SOFTICE".

- Введите команду: NET START NTICE.

Замечание: Запустив SoftICE, Вы не сможете его выгрузить до полной перезагрузки компьютера.

Для проверки загрузки SoftICE нажмите комбинацию клавиш Ctrl-D. Должен появится экран SoftICE. Для возвращения в систему используйте команды X (eXit — Выход) или G (Go to — Перейти к) (функциональная клавиша F5).

Пример построения приложения GDIDEMO

Первый шаг при подготовке к отладке приложения Windows — это компиляция и компоновка его с отладочной информацией. Для этого в демонстрационном приложении GDIDEMO уже создан makefile.

Для построения программы выполните следующие действия:

1 Откройте окно DOS.

2 Перейдите в директорию, где находится исходный код приложения.

3 Выполните команду NMAKE:

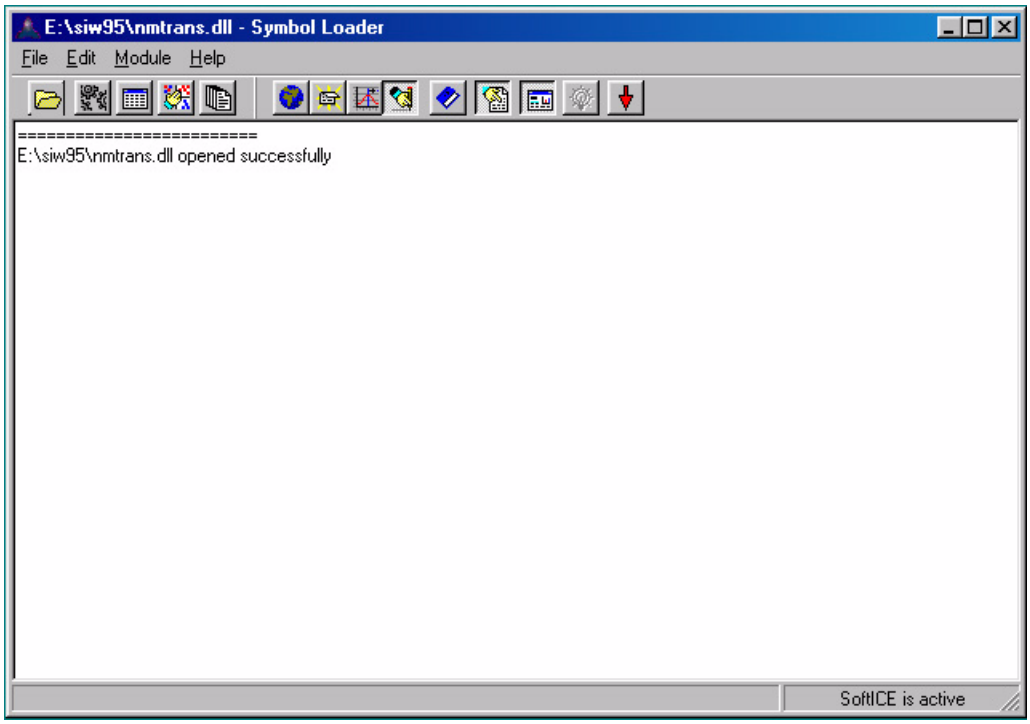
```
C:\MSTOOLS\SAMPLES\WIN32\GDIDEMO>NMAKE
```

Если GDIDEMO находится в другой директории, укажите правильный путь.

Загрузка приложения GDIDEMO

При загрузке приложения происходит создание символьного файла из отладочной информации и загрузка символьных и исходных файлов в SoftICE. Для загрузки приложения GDIDEMO выполните следующие действия:

1 Запустите утилиту Symbol Loader.
Появится окно символьного загрузчика.



2 Выберите "OPEN MODULE" из меню "FILE" или щелкните по кнопке "OPEN".
Появится диалог выбора файла.

3 Выберите файл GDIDEMO.EXE и нажмите клавишу "OPEN".

4 Выберите пункт "LOAD" из меню "MODULE" или щелкните по кнопке "LOAD" для загрузки GDIDEMO.

Символьный загрузчик транслирует отладочную информацию в символьный файл .NMS, загружает символьные и исходные файлы, запускает GDIDEMO, открывает экран SoftICE, и отображает исходный текст приложения из файла GDIDEMO.C

Управление экраном SoftICE

Экран SoftICE — это Ваш главный инструмент для просмотра и отладки кода. На экране может быть отображено до семи различных окон и одна справочная строка, что позволит вам просматривать и управлять различными аспектами сеанса отладки. По умолчанию, на экране отображено следующее:

- Local Window (Окно локальных переменных) — Отображает текущий кадр стека.
- Code Window (Окно кода) — Отображает исходный текст программы или соответствующие им ассемблерные инструкции.
- Command Window (Окно команд) — Предназначено для ввода команд пользователя и отображение информации.
- Help Line (Строка подсказки) — Предоставляет краткую информацию о командах SoftICE, а также показывает текущий контекст адреса.

1 Рассмотрим содержимое окна кода. Обратите внимание, что SoftICE отображает подпрограмму WinMain в 34 строке. По умолчанию, после загрузки вашего приложения SoftICE создает контрольную точку в первом основном модуле и останавливается на ней.

2 Чтобы увидеть все исходные файлы, которые загружены в SoftICE, введите команду FILE с параметром "*":

```
:FILE *
```

SoftICE покажет список исходных файлов GDIDEMO: draw.c, maze.c, xform.c, poly.c, wininfo.c, dialog.c, init.c, bounce.c, и gdidemo.c. Окно команд изменяет свой размер в зависимости от числа строк, используемых другими открытыми окнами, так что Вы можете и не увидеть весь список сразу. Для просмотра продолжения списка нажмите любую клавишу. (Изменение размеров окон описано в глава 5 "*Интерфейс SoftICE*" на странице 58.)

3 Практически в каждом окне SoftICE можно использовать прокрутку. Если у Вас подключена мышь, то Вы можете щелкнуть на стрелках прокрутки. Если нет, то не расстраивайтесь, SoftICE позволяет пролистывать окна с помощью различных комбинаций клавиш. В таблице приведены последовательности действий для листания окна кода.

Листание окна кода	**Комбинации клавиш**	**Работа мышью**
На предыдущую страницу	PageUp	Щелкните по верхней внутренней стрелке
На следующую страницу	PageDown	Щелкните по нижней внутренней стрелке
На предыдущую строку	Стрелка вверх	Щелкните по верхней внешней стрелке
На следующую строку	Стрелка вниз	Щелкните по нижней внешней стрелке
На один символ влево	Ctrl-Стрелка влево	Щелкните по левой стрелке
На один символ вправо	Ctrl-Стрелка вправо	Щелкните по правой стрелке

4 Ввод команды U с параметром EIP приводит к дизассемблированию текущей инструкции (инструкция, подсвеченная в данный момент).

```
:U EIP
```

Это же действие можно выполнить с помощью команды "." (точка):

```
:.
```

Трассировка и пошаговое выполнение кода

Следующие шаги покажут Вам, как использовать SoftICE для трассировки исходного кода:

1 Введите команду T (Trace — Трассировать) или нажмите клавишу F8.

```
:T
```

Происходит выполнение одной строки исходного кода. Следующая за ней строка окажется подсвеченной.

```
if(!hPrevInst)
```

2 Окно кода в настоящее время отображает исходный текст. Однако, это окно может также отображать дизассемблированный код или смешанный (исходный и дизассемблированный) код. Для просмотра смешанного кода используйте команду SRC (F3).

```
:SRC
```

Обратите внимание, что каждая строка исходного текста теперь сопровождается ассемблерными командами.

3 Нажмите F3 еще раз для получения "чистого" дизассемблерного кода, а затем еще раз для возврата на экран исходного текста.

4 Введите команду T (F8) для трассировки следующей инструкции.
Исполняйте эту команду до тех пор, пока не достигнете строки, выполняющей функцию RegisterAppClass.

Как было показано, команда T выполняет одну исходную инструкцию или команду ассемблера. Вы также можете использовать команду P (F10) для пошагового исполнения программы. Пошаговое исполнение отличается от трассировки тем, что при выполнении команды CALL, управление не будет возвращено до полного завершения вызванной функции.

Совет: Команда T не трассирует вызовы функций, если их исходный текст недоступен. Хороший пример этого — Win32 API. Для трассировки вызовов функций в случаях, когда недоступен исходный текст, используйте команду SRC (F3) для переключения в смешанный или ассемблерный режим.

Просмотр локальных данных

Окно локальных переменных предназначено для отображения текущего кадра стека. В нашем случае это окно содержит локальные данные для функции WinMain. Следующие шаги иллюстрируют, как можно использовать окно локальных переменных:

1 Выполните команду T, чтобы войти в функцию RegisterAppClass. Окно локальных переменных на данный момент пусто, потому что память под локальные переменные еще не выделена.
Код функции RegisterAppClass находится в исходном файле INIT.C. SoftICE отображает имя текущего исходного файла в верхнем левом углу окна кода.

2 Снова выполните команду T.
Окно локальных переменных содержит параметр hInstance, переданный в RegisterAppClass, и локальную структуру wndClass. Структура wndClass отмечена знаком "+". Это означает, что Вы можете развернуть структуру и просмотреть ее содержимое.
Замечание: Вы также можете разворачивать символьные строки и массивы.
3 Если у Вас установлен процессор типа Pentium и имеется мышь, то двойным щелчком по структуре WNDCLASSA Вы можете развернуть ее. Чтобы свернуть структуру wndClass, щелкните дважды по знаку "–".
4 Развернуть структуру можно и с помощью клавиатуры: нажмите Alt-L, чтобы переместить курсор в окно локальных переменных, стрелками вверх или вниз переместите подсвеченную строку на структуру и нажмите Enter. Чтобы свернуть структуру, нажмите Enter еще раз.

Установка точек прерывания

Этот раздел научит вас устанавливать два типа контрольных точек: одноразовые и постоянные.

Установка одноразовой контрольной точки

Ниже показано, как можно установить одноразовую контрольную точку. Она отличается тем, что удаляется сразу после того, как произойдет ее выполнение.

1 Чтобы вернуться в окно кода, необходимо щелкнуть по нему мышкой или нажать Alt-C.
Если Вы хотите вернуться обратно в окно команд, то снова нажмите Alt-C.
2 Используйте стрелку вниз на клавиатуре, стрелку прокрутки вниз или команду U для того, чтобы переместить курсор на 61 строку (первый вызов API функции RegisterClass). Если вы используете команду U, то командная строка должна выглядеть так:
```
:U .61
```
SoftICE разместит эту строку вверху окна кода.
3 Используйте команду HERE (функциональная клавиша F7), чтобы исполнить код до строки 61.
Команда HERE исполняет программу, начиная с текущей инструкции, до инструкции, на которую указывает курсор. Команда HERE устанавливает одноразовую точку прерывания на определенный адрес или строку исходного кода и продолжает исполнение до срабатывания этого прерывания. После срабатывания прерывания, SoftICE автоматически удаляет данную точку, так что второй раз она уже не сработает.
В результате окажется подсвеченной следующая строка:
```
if (!RegisterClass(&wndClass))
```

Замечание: Тоже самое вы можете сделать, используя команду G (Перейти к), задав номер строки, до которой необходимо исполнить программу:
```
:G .61
```

Установка постоянной точки прерывания

Следующая последовательность действий иллюстрирует установку второго типа контрольных точек: постоянной точки прерывания, которая не удаляется после срабатывания.

1 Найдите следующий вызов функции RegisterClass, он происходит в строке 74. Установите курсор на строку 74 и введите команду BPX (F9). Команда BPX устанавливает точку прерывания, вставляя в исходный код инструкцию INT 3.
2 Повторное нажатие клавиши F9 удалят контрольную точку.
Если у Вас в компьютере используются процессор Pentium и мышь, вы можете двойным щелчком на любой строке в окне кода устанавливать или удалять в ней контрольную точку.
3 Установите точку прерывания на 74 строке, и командой G или X (F5) продолжайте выполнения программы до тех пор, пока эта точка не сработает.
```
:G
```
Окно SoftICE появится после выполнения инструкции INT 3.
В отличие от команды HERE, ставящей одноразовую контрольную точку, команда BPX устанавливает постоянную точку прерывания, которая существует до тех пор, пока Вы ее не удалите.
4 Для просмотра информации о текущей точке прерывания используйте команду BL:
```
:BL
00) BPX #0137:00402442
```
Замечание: Адрес, который вы увидите, может не совпадать с данным здесь.
Команда BL показала Вам, что установлена одна контрольная точка на адрес 0x402442. Этот адрес соответствует 74 строке исходного текста в файле INIT.C.
5 Вы можете использовать вычислитель выражений SoftICE, чтобы перевести номер строки в адрес. Чтобы определить адрес 74 строки, используйте команду ?:
```
:? .74
void *= 0x00402442
```
6 Функция RegisterAppClass имеет относительно простую реализацию, поэтому нет необходимости трассировать каждую строку исходного кода. Используя команду P с параметром RET (функцинальная клавиша F12), можно вернуться в точку, откуда произошел вызов функции:
```
:P RET
```
Параметр RET в команде P заставляет SoftICE полностью выполнить вызванную процедуру до момента выхода из нее. SoftICE остановится в блоке WinMain на инструкции, расположенной после вызова функции RegisterAppClass, так как функция RegisterAppClass была вызвана из WinMain. Следующая строка в блоке WinMain окажется подсвеченной.
```
msg.wParam = 1;
```
7 Чтобы удалить все установленные контрольные точки, введите команду BC со звездочкой в качестве параметра:
```
:BC *
```

Использование информационных команд SoftICE

SoftICE предоставляет большое количество информационных команд, которые позволяют вам следить за состоянием приложения или системы. Данный раздел объясняет назначение двух команд: H (помощь) и CLASS.

1 Команды H и CLASS лучше всего работают, когда у Вас достаточно места для размещения выводимой ими информации, поэтому закройте окно локальных переменных командой WL, что приводит к увеличению размера окна команд.

2 Команда H предоставляет общие сведения обо всех командах SoftICE или детальную справку о конкретной команде. Для просмотра детальной справки о команде CLASS введите CLASS в качестве параметра команды H:

```
:H CLASS

Display window class information
CLASS [-x] [process | thread | module | class-name]
ex: CLASS USER
```

Первая строка справки предоставляет описание команды, вторая — полный ее синтаксис, включая все опции и/или параметры, а третья — пример использования команды.

3 Функция RegisterAppClass регистрирует шаблон класса окна, который используется приложением GDIDEMO для создания окна. Используйте команду CLASS, чтобы исследовать классы, зарегистрированные GDIDEMO.

```
:CLASS GDIDEMO

Class Name    Handle        Owner       Wndw Proc    Styles
----------------------Application Private----------------------
BOUNCEDEMO    A018A3B0      GDIDEMO     004015A4     00000003
DRAWDEMO      A018A318      GDIDEMO     00403CE4     00000003
MAZEDEMO      A018A280      GDIDEMO     00403A94     00000003
XFORMDEMO     A018A1E8      GDIDEMO     00403764     00000003
POLYDEMO      A018A150      GDIDEMO     00402F34     00000003
GDIDEMO       A018A0C0      GDIDEMO     004010B5     00000003
```

Замечание: *В* этом примере показаны только классы, которые зарегистрированы приложением GDIDEMO. Классы, зарегистрированные другими модулями Windows, например USER32, опущены.

Информация, выводимая командой CLASS — это сводная информация для каждого класса окна, зарегистрированного от имени процесса GDIDEMO. Она включает имя класса, адрес внутренней структуры данных WINCLASS, модуль, который этот класс зарегистрировал, адрес процедуры для оконного класса, заданной по умолчанию, и значение флагов стиля класса.

Замечание: Для получения более детального описания класса окна используйте команду CLASS с опцией –X, как показано ниже:

```
:CLASS -x
```

Использование идентификаторов и таблиц символов

Теперь, когда Вы достаточно знакомы с SoftICE, чтобы пошагово отлаживать, трассировать и создавать точки прерывания, настало время исследовать идентификаторы и таблицы. Когда Вы загружаете отладочную информацию приложения, SoftICE создает таблицу, которая содержит все идентификаторы, определенные для этого модуля.

1 Используйте команду TABLE, чтобы просмотреть все таблицы идентификаторов, загруженных на данный момент.

```
:TABLE

GDIDEMO [NM32]
964657 Bytes Of Symbol Memory Available
```

Активная в данный момент таблица идентификаторов отображается жирным шрифтом. Эта таблица используется для интерпретации идентификаторов имен. Сделать активной другую необходимую Вам в данный момент таблицу можно с помощью команды TABLE, указав в качестве параметра ее имя:

```
:TABLE GDIDEMO
```

2 Для вывода списка идентификаторов из текущей таблицы используется команда SYM. Если текущей окажется таблица GDIDEMO, то команда SYM выведет список, подобный приведенному ниже:

```
:SYM

.text(001B)
001B:00401000 WinMain
001B:004010B5 WndProc
001B:004011DB CreateProc
001B:00401270 CommandProc
001B:00401496 PaintProc
001B:004014D2 DestroyProc
001B:004014EA lRandom
001B:00401530 CreateBounceWindow
001B:004015A4 BounceProc
001B:004016A6 BounceCreateProc
001B:00401787 BounceCommandProc
001B:0040179C BouncePaintProc
```

Это список символических имен из раздела .text исполняемой программы. Раздел .text обычно используется для процедур и функций. Все идентификаторы, отображенные в этом примере, являются функциями GDIDEMO.

Установка условного прерывания

Одним из идентификаторов, определенных в GDIDEMO, является имя функции LockWindowInfo. Эта функция предназначена для получения значения указателя, которое является специфическим для каждого экземпляра окна.

Для изучения условных точек прерывания и контрольных точек в памяти Вами будут выполнены следующие шаги:

- вы установите контрольную точку типа BPX на функцию LockWindowInfo;
- отредактируете эту точку с использованием условного выражение, получив таким образом условную контрольную точку;
- установите контрольную точку на адрес в памяти для того, чтобы отлеживать доступ к фрагменту ключевой информации, как описано это в разделе *"Установка контрольной точки чтения/записи в память"* на странице 34.

Установка точки прерывания BPX

Перед установкой условной контрольной точки, Вам необходимо установить точку прерывания типа BPX на функцию LockWindowInfo.

1 Установите прерывание на функцию LockWindowInfo:

```
:BPX LockWindowInfo
```

Когда одному из окон GDIDEMO необходимо вывести что-либо в своей рабочей области, оно вызывает функцию LockWindowInfo. Каждый раз, когда вызывается эта функция, будет активизирован экран SoftICE. Окна в приложении GDIDEMO меняются непрерывно, поэтому эта точка прерывания срабатывает очень часто.

2 Проверьте факт установки контрольной точки командой BL.

3 Выйдите из SoftICE командой X или G.
Всякий раз, когда произойдет вызов функции LockWindowInfo, на мониторе будет появляться экран SoftICE.

Редактирование точки прерывания

Из прототипа функции LockWindowInfo, расположенного в строке 47, видно, что в функцию передается один параметр типа HWND, а из функции возвращается указатель не установленного типа. Параметр HWND — это дескриптор окна, которое пытается вывести информацию в рабочей области. В настоящий момент нам необходимо изменить существующую контрольную точку, добавив условие для выявления нужного нам значения HWND.

1 Прежде чем устанавливать условную контрольную точку, необходимо получить значение HWND для окна POLYDEMO. Команда HWND предоставляет информацию об окнах приложения, поэтому выполните команду HWND с параметром GDIDEMO:

```
:HWND GDIDEMO
```

Ниже показано, что Вы должны увидеть на экране при использовании операционной системы Windows NT. Если Вы работаете под Windows 95, информация будет немного отличаться.

```
Handle      Class          WinProc      TID      Module
07019C      GDIDEMO        004010B5     2D       GDIDEMO
100160      MDIClient      77E7F2F5     2D       GDIDEMO
09017E      BOUNCEDEMO     004015A4     2D       GDIDEMO
100172      POLYDEMO       00402F34     2D       GDIDEMO
11015C      DRAWDEMO       00403CE4     2D       GDIDEMO
```

Дескриптор окна POLYDEMO выделен подчеркиванием и жирным шрифтом. Именно это значение Вы и будете использовать при формировании условного выражения. Если в области вывода HWND информация об окне POLYDEMO отсутствует, то выйдите из SoftICE, используя команду G или X (F5), и повторяйте шаг 1 до тех пор, пока данное окно не будет создано.

Значение, использованное в нашем примере, скорее всего будет отличаться от того, которое увидите Вы. Для правильного выполнения данного упражнения Вам следует использовать фактическое значение HWND для Вашей системы.

Используя дескриптор окна POLYDEMO, Вы можете составить условное выражение для контроля вызова функции LockWindowInfo, проверяя совпадение передаваемого программе параметра с требуемым значением дескриптора. Когда будет вызвана функция LockWindowInfo с дескриптором окна POLYDEMO, появится экран SoftICE.

2 Так как на функцию LockWindowInfo контрольная точка уже установлена, то с помощью команды BPE (редактирование контрольной точки) ее можно изменить:

```
:BPE 0
```

Когда Вы выдаете команду BPE для изменения существующей контрольной точки, SoftICE помещает ее описание в командную строку:

```
:BPX LockWindowInfo
```

В конце командной строки появится курсор, чтобы вы смогли набрать условное выражение.

3 Не забудьте подставить значение дескриптора окна POLYDEMO, которое Вы нашли с помощью команды HWND вместо значения 100172, используемого в данном примере. Ваше условное выражение должно выглядеть следующим образом (условное выражение выделено жирным шрифтом):

```
:BPX LockWindowInfo IF ESP->4 == 100172
```

Замечание: Приложения Win32 передают параметры через стек в точке входа функции; первый параметр имеет положительное смещение 4 относитель-

но регистра ESP. Для вычислителя выражений SoftICE это записывается в виде: ESP ->4. ESP — это регистр указателя вершины стека, а оператор "->" заставляет левую часть выражения (в данном случае — ESP) принять смещение, указанное в правой его части (4). Для более полной информации о вычислителе выражений обратитесь к главе 8 "*Использование выражений*", а за сведениями относительно стека в условных выражениях обратитесь к разделу "*Условные прерывания*" на странице 91.

4 Командой BL проверьте, что точка прерывания и условное выражение установлены правильно.

5 Выйдите из SoftICE, используя команду G или X (F5).
Когда в следующий раз появится экран SoftICE, условное выражение будет иметь значение ИСТИНА.

Установка контрольной точки чтения/записи в память

Мы установили контрольную точку, а затем добавили условное выражение так, чтобы иметь возможность получить адрес структуры данных, уникальной для каждого экземпляра окна POLYDEMO. Это значение сохранено в дополнительных данных окна (extra data) и является глобальным дескриптором. Функция LockWindowInfo находит этот глобальный дескриптор и использует Win32 API LocalLock для трансляции его в указатель, который может быть использован для доступа к данным этого экземпляра окна.

1 Получите значение указателя для данных экземпляра окна, выполняя программу до строки 57.
```
:G .57
```

2 Функции Win32 API возвращают 32-разрядные значения в регистре EAX, так что Вы можете использовать команду BPMD с регистром EAX для установки контрольной точки в памяти на указатель данных экземпляра окна.
```
:BPMD EAX
```

Для контроля чтения и/или записи двойных слов (Dword) по линейному адресу команда BPMD использует аппаратные отладочные регистры процессоров фирмы Intel. В нашем случае, вы выполняете команду BPMD для перехвата доступа чтения и записи к первому двойному слову (Dword) данных экземпляра окна.

3 Используйте команду BL для того, чтобы проверить правильность установки контрольной точки. Вы должны увидеть примерно следующее:
```
:BL

00) BPX LockWindowInfo IF ((ESP->4)==0x100172)
01) BPMD #0023:001421F8 RW DR3
```

Контрольная точка с индексом 0 — это прерывание выполнения функции LockWindowInfo, а точка 1 — это контрольная точка BPMD на данные экземпляра окна.

4 Для отключения точки прерывания на LockWindowInfo выполните команду BD.
```
:BD 0
```

SoftICE имеет команды BC (удалить точку прерывания) и BD (отключить точку прерывания). Отключение полезно, когда Вы хотите повторно использовать эту же контрольную точку в следующем сеансе отладки. Если же Вы не собираетесь снова использовать какое-либо прерывание, имеет смысл его удалить.

5 Используйте команду BL для того, чтобы проверить отключение точки прерывания на LockWindowInfo. SoftICE отмечает блокированные прерывания звездочкой "*" после его индекса. Вы должны увидеть примерно следующее:
```
:BL

00) * BPX LockWindowInfo IF ((ESP->4)==0x100172)
01) BPMD #0023:001421F8 RW DR3
```
Замечание: Снова задействовать временно отключенную точку прерывания можно с помощью команды BE:
`:BE` *`индекс-точки-прерывания`*

6 Выйдите из SoftICE, используя команду G или X.
Когда окно POLYDEMO обратится к первому двойному слову (Dword) данных экземпляра окна, сработает прерывание и появится экран SoftICE.
Если экран SoftICE появится вследствие срабатывания контрольной точки в памяти, Вы окажетесь в функции PolyDrawBez или PolyRedraw. Обе функции обращаются к полю nBezTotal со смещением 0 данных экземпляра окна POLYDRAW.
Замечание: Архитектура процессоров фирмы Intel определяет точки прерывания в памяти как ловушки, что означает их срабатывание после обращения к памяти. Команда или исходная строка, подсвеченные в SoftICE в данный момент — это команда или исходная строка, которая только что обратилась к памяти.

7 Удалите контрольные точки, которые вы устанавливали в этом разделе, используя команду BC.
```
:BC *
```
Замечание: Вы можете использовать символ звездочка "*" в командах BC, BD и BE для удаления, отключения и включения всех контрольных точек одновременно.

8 Выйдите из SoftICE, используя команду G или X.
Операционная система завершит приложение.

Поздравляем, Вы закончили свой первый сеанс отладки. В этом сеансе Вы научились трассировать исходный код, просматривать локальные данные и структуры, устанавливать контрольные точки и добавлять к ним условия, устанавливать точки прерывания в памяти. SoftICE предоставляет гораздо больше возможностей. Команды ADDR, HEAP, LOCALS, QUERY, THREAD, TYPES, WATCH и WHAT являются одними из многих команд SoftICE, которые помогут Вам отлаживать приложения более просто и быстро. За полным описанием команд обращайтесь к "*Справочнику по командам SoftICE*".

Дьявол кроется в деталях.
Декарт

The devil is in the details.
Descartes

4. Загрузка кода в SoftICE

Концепции отладки

SoftICE позволяет прозводить отладку прикладных программ Windows и драйверов устройств на уровне исходного кода. Для этого SoftICE использует утилиту Symbol Loader (Символьный Загрузчик), транслирующую отладочную информацию из Вашего модуля в файл .NMS. Затем Symbol Loader может загрузить .NMS-файл и, по желанию, исходный код в SoftICE для их использования в сеансе отладки.

Момент загрузки .NMS зависит от того, когда исполняется отлаживаемый модуль — после инициализации операционной системы, или это драйвер устройства или статический VxD, который загружается перед инициализацией операционной системы. Если Вы разрабатываете драйвер устройства или VxD, SoftICE загрузит символы модуля и исходный код в момент собственного запуска. Если Вы отлаживаете модуль или компонент, который исполняется после загрузки операционной системы, с помощью Symbol Loader Вы можете загрузить символьную информацию тогда, когда она понадобится.

Эта глава объясняет, как использовать Symbol Loader, чтобы загрузить Ваш модуль в SoftICE. Здесь также рассказывается, как использовать загрузчик из командной строки DOS для автоматизации исполняемых им задач, и как пользоваться утилиту командной строки NMSYM для создания пакетов трансляции и загрузки символьной информации.

Замечание: Symbol Loader поддерживает только прикладные программы Windows. Для отладки прикладных программ MS-DOS используйте утилиты каталога UTIL16.

Подготовка к отладке прикладных программ

Ниже приведены общие принципы подготовки к отладке модулей и их компонентов, исполняемых после загрузки операционной системы. Это могут быть исполняемые .EXE-файлы, библиотеки DLL, динамические .VxD и .OCX - файлы. В последующих разделах перечисленные этапы объясняются более подробно.

1 Сформируйте модуль с отладочной информацией.
2 Загрузите SoftICE, если он еще не загружен.
3 Запустите Symbol Loader.
4 Щелкните по кнопке "OPEN", чтобы открыть отлаживаемый модуль.
5 Оттранслируйте отладочную информацию в файл .NMS и загрузите ее вместе с исходным кодом в SoftICE.

Подготовка к отладке драйверов устройств и VxD

Следующая последовательность шагов показывает, что необходимо выполнить для отладки драйверов устройств или статических VxD, которые загружаются прежде, чем операционная система полностью инициализируется. В последующих разделах перечисленные этапы объясняются более подробно.

1 Сформируйте прикладную программу с отладочной информацией.
2 Загрузите SoftICE, если он еще не загружен.
3 Запустите Symbol Loader.
4 Щелкните по кнопке "OPEN", чтобы открыть отлаживаемый модуль
5 Выберите пункт "PACKAGE SOURCE WITH SYMBOL TABLE" в режимах трансляции Symbol Loader. (См. раздел "*Изменение параметров настройки модуля*" на странице 40.)
6 Для создания нового .NMS файла щелкните по кнопке "TRANSLATE".
7 Измените параметры настройки SoftICE для загрузки отладочной информации VxD или драйвера устройства при запуске. (Подробности настройки смотрите в разделе "*Предварительная загрузка символьной информации и исходного кода*" на странице 121.)
8 Перезагрузите компьютер.

Запуск SoftICE вручную

SoftICE не запускается автоматически при следующих конфигурациях:

- Если Вы не запустили WINICE.EXE из AUTOEXEC.BAT перед стартом Windows 95.
- Если Вы установили ручной режим запуска SoftICE для Windows NT.

Если Вы используете такие конфигурации, то SoftICE необходимо загрузить вручную. Следующие разделы описывают, как это сделать под операционными системами Windows 95 и Windows NT.

Загрузка SoftICE для Windows 95

Чтобы загрузить SoftICE для Windows 95, запустите его из командной строки DOS. После инициализации SoftICE автоматически производит запуск Windows 95. Команды имеет следующий синтаксис:

Синтаксис команды

WINICE [/HST n][/TRA n][/SYM n][/M] [/LOAD [x] *имя-файла*]
[/EXP *имя-файла*] [*диск*:*путь*\WIN.COM [*WIN-командная-строка*]]

Необязательные параметры:

Параметр	Описание
/EXP *имя-файла*	Добавляет экспортируемые символы из DLL или приложений Windows, заданных параметром "*имя-файла*" данные в SoftICE. Это позволит Вам обращаться к этим данным с помощью символических имен
/HST n	Увеличивает размер буфера протокола вызова команд, где n — количество Кб (десятичное). По умолчанию n=8 Кб[†] .

Продолжение на следующей странице

[†] Если речь идет о параметре "HISTORY BUFFER SIZE" (размер буфера протокола), то его размер по умолчанию равен 256K (см. страницу 120).

Ключ	Описание
/LOAD *имя-файл* [x]	Загружает отладочную информацию и исходный код, где *файл* — полный путь и имя собранных вместе с отладочной информацией VxD, DOS TSR, загружаемых драйверов DOS, DOS программ, драйверов Windows, Windows DLL и программ для Windows. Если присутствует параметр "x", то исходный код загружаться не будет.
/M	Направляет вывод SoftICE на дополнительный монохромный монитор в обход VGA.
/SYM n	Резервирует n (десятичное) килобайт под таблицу символов. По умолчанию n = 0 Кб[†] .
/TRA n	Устанавливает размер буфера обратной трассировки равным n Кб (десятичное). По умолчанию n = 8 Kb.

Совет: Вы можете определять эти переключатели в строке инициализации. (Смотрите раздел "*Изменение начальных установок SoftICE*" на странице 119.)

Запуск SoftICE для Windows NT

Чтобы загрузить SoftICE для Windows NT, сделайте следующее:

- Выберите "START SOFTICE".
- Введите команду: NET START NTICE.

Замечание: Как только Вы загрузите SoftICE, Вы не сможете его дезактивировать до тех пор, пока не перегрузите компьютер.

Создание приложений с отладочной информацией

Ниже дается сводная таблица параметров компиляции и компоновки программ. Полную информацию, касающуюся формирования Вашей прикладной программы с отладочной информацией, смотрите в соответствующей документации.

Компилятор	Создание отладочной информации
Borland C++ 4.5 и 5.0	Для создания стандартной отладочной информации: Компиляция с ключом /v Компоновка с ключом /v
Delphi 2.0	Для создания стандартной отладочной информации компилируйте проект со следующими ключами: -V — для включения отладочной информации в исполняемый модуль. -$W+ — чтобы создать кадры стека. -$D+ — чтобы создать информацию отладки. -$L+ — чтобы создать локальные символы отладки. -$O — чтобы отключить оптимизацию

Продолжение на следующей странице

[†] Под отладочную информацию по умолчанию выделяется 1024 Кб в Windows 95 и 512 Кб в Windows NT (см. страницу 122).

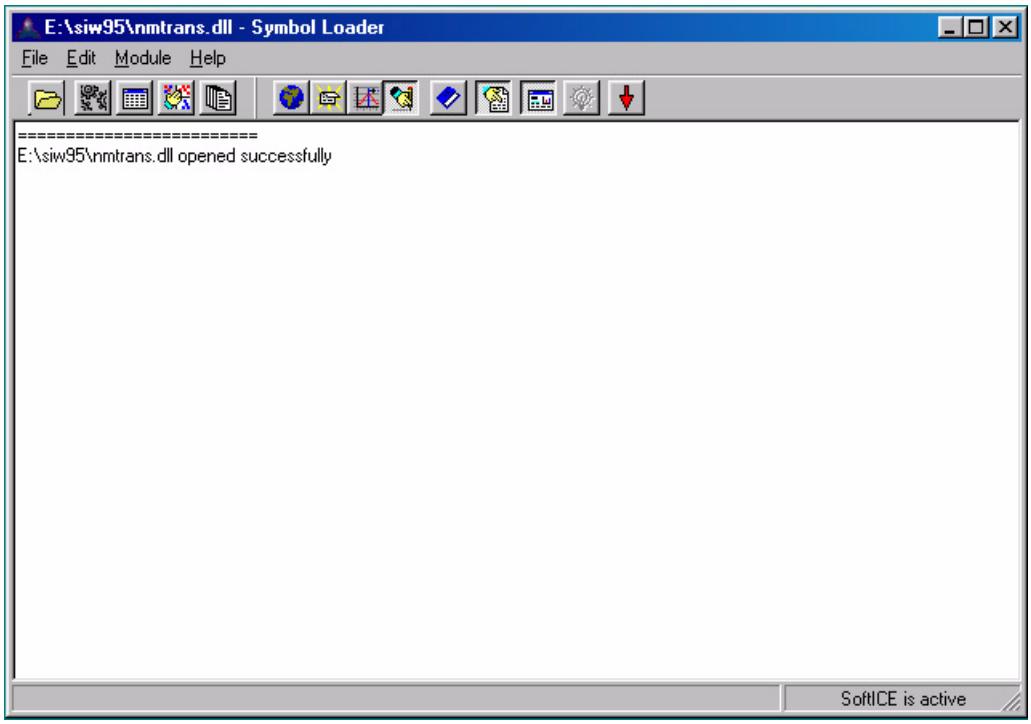
Компилятор	Создание отладочной информации
MASM 6.11	Для создания отладочной информации CodeView: Ассемблировать с ключами /Zi /COFF Используйте 32-разрядный компоновщик Microsoft (LINK.EXE) с ключами /DEBUG /DEBUGTYPE:CV /PDB:NONE
Microsoft Visual C++ 2.x, 4.0, 4.1, 4.2, и 5.0	Для создания базы данных отладочной информации программы (PDB): Компилируйте, используя опцию командной строки /Zi Используйте компоновщик Microsoft с ключами /DEBUG /DEBUGTYPE:CV *Замечание*: VxD требует создания отладочной информации PDB. Для создания отладочной информации CodeView: Компилируйте с отладочной информацией, совместимой с C7, используя опцию командной строки /Z7 Используйте компоновщик Microsoft с ключами /DEBUG /DEBUGTYPE:CV /PDB:NONE *Замечание*: Если Вы используете стандартную процедуру Windows NT DDK, то используйте следующие переменные среды: NTDEBUG=ntsd и NTDEBUGTYPE=windbg.
Symantec C++ 7.2	Компилятор фирмы Symantec создает отладочную информацию CodeView по умолчанию. Все, что требуется — это разрешить ее вывод: Компилируйте с ключом −g Также можно использовать −gh, −gf, и −gg Компонуйте проект с ключом /CO
Watcom C++ 10.5	Для создания отладочной информации CodeView: Компилируйте с ключами −hc −d3 Компонуйте проект с ключами DEBUG CODEVIEW OPTION CVPACK

Использование Symbol Loader для трансляции и загрузки файлов

Прежде чем SoftICE сможет отлаживать Вашу прикладную программу, .DLL или драйвер, Вы должны создать символьный файл для каждого отлаживаемого модулей, а затем загрузить их в SoftICE. Утилита Symbol Loader делает эту процедуру простой и быстрой. Эта программа позволяет Вам выбрать необходимый модуль и создать соответствующий символьный файл, после чего загрузит его в SoftICE вместе с исходным кодом и исполняемым файлом. По умолчанию загружаются все файлы, упомянутые в отладочной информации. Подробности того, как можно загружать только необходимые в конкретном сеансе исходные файлы смотрите в разделе " *Задание исходных файлов программы*" на странице 45.

Чтобы загрузить модуль с помощью утилиты Symbol Loader, сделайте следующее:

1 Запустите Symbol Loader.



2 Выберите пункт "OPEN MODULE" из меню "FILE", либо щелкните по кнопке "OPEN".

3 Выберите в диалоговом окне необходимый файл и нажмите кнопку "OPEN".

4 Если Вы открываете файл типа .SYM, Symbol Loader поинтересуется, 32-разрядный это файл или нет. Ответьте, соответственно YES или NO.
Из-за ограничений формата .SYM-файлов SoftICE не может самостоятельно определить разрядность .SYM-файлов (16- или 32-битный).

Совет: Как только Вы открываете файл, он добавляется к списку недавно открывавшихся файлов в нижней части меню "FILE". Используйте этот список для ускорения повторного открытия файла.

1 Выберите пункт "LOAD" из меню "MODULE", либо нажмите кнопку "LOAD" для загрузки открытого файла.
Symbol Loader транслирует отладочную информацию из Вашей прикладной программы в символьный файл .NMS, а затем загружает его вместе с исходным текстом программы в SoftICE. Если Вы загружаете исполняемый файл (.EXE), то SoftICE запускает программу и устанавливает точку прерывания в первом основном модуле (WinMain, Main, или DllMain), с которым он сталкивается.
Объем загружаемой информации зависит от параметров настройки режимов трансляции и отладки. Смотрите раздел "*Изменение параметров настройки модуля*" на странице 40 для получения более полной информации относительно параметров настройки режимов трансляции и отладки.

Изменение параметров настройки модуля

Утилита Symbol Loader имеет ряд параметров для управления трансляцией и загрузкой файлов. Их можно разделить на 3 группы:

- Общие — параметры командной строки и пути поиска исходных файлов.
- Параметры трансляции — определяют, какую комбинацию символов (открытая информация (Public), информация о типах, символы или символы и источник) транслирует Symbol Loader.

- Параметры отладки — определяют типы файлов (символьные или исполняемые), загружаемые утилитой в SoftICE, а также действия, исполняемые SoftICE во время загрузки.

Эти параметры устанавливаются для каждого модуля отдельно, то есть изменение индивидуальных установок сказывается только на текущем модуле. Когда Вы открываете другой модуль, Symbol Loader использует установки по умолчанию.

Чтобы изменить заданные по умолчанию параметры настройки модуля, сделайте следующее:

1 Откройте файл, если он еще не открыт.

Совет: Имя открытого в данный момент файла показано в заголовке окна Symbol Loader.

2 Выберите пункт "SETTINGS" из меню "MODULE".

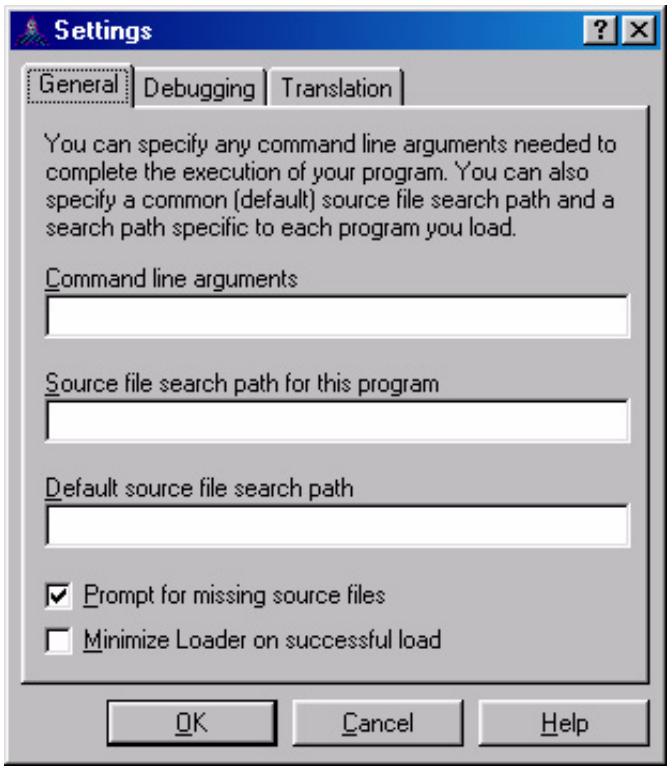
3 Щелкните на закладке, которая объединяет группу параметры настройки, которые Вы хотите изменить. (Более подробная информация о параметрах настройки дана в последующих разделах).

4 Когда все изменения сделаны, нажмите OK.

5 Загрузите файл, чтобы применить Ваши изменения.

Изменение Общих Параметров настройки (вкладка General)

Вкладка General позволяет установить параметры командной строки и определить путь для поиска исходного файла.



Общие параметры настройки описаны в следующих разделах.

Command line arguments (Параметры Командной строки)

Используйте "COMMAND LINE ARGUMENTS" для задания параметров командной строки, передаваемых Вашему приложению.

Source file search path for program (Путь для поиска исходного файла данной программы)

Параметр "SOURCE FILE SEARCH PATH FOR THIS PROGRAM" задает пути поиска файлов, связанных с данным приложением. Если Symbol Loader не может найти файлы по заданному пути, он использует "DEFAULT SOURCE FILE SEARCH PATH" (путь поиска файлов по умолчанию) для развернутого поиска.

Default source file search path (Путь поиска файлов по умолчанию)

Параметр "DEFAULT SOURCE FILE SEARCH PATH" используется для задания основного пути, используемого SoftICE для поиска файлов. Эта установка является глобальной.

Обратите внимание, что, если Вы задаете значение параметра SOURCE FILE SEARCH PATH FOR THIS PROGRAM для конкретной программы, то Symbol Loader использует сначала этот путь, и только потом путь, заданный по умолчанию.

Prompt for missing source files (Запрос при отсутствии исходного файла)

Если флаг "PROMPT FOR MISSING SOURCE FILES" установлен, то Symbol Loader обратится к Вам за помощью в случае, если исходный файл не будет найден. Этот параметр является глобальным и по умолчанию включен.

Minimize loader on successful load (Минимизировать загрузчик при успешной загрузке)

Используйте "MINIMIZE LOADER ON SUCCESSFUL LOAD" для автоматической минимизации Symbol Loader после загрузки .EXE-файла. Этот параметр является глобальным и по умолчанию включен.

Изменение параметров трансляции (вкладка Translation)

Параметры трансляции определяют, какая информация транслируется утилитой Symbol Loader при создании .NMS-файла, а также нужно ли сохранять исходный текст программы в символьном файле. Эти настройки определяют количество памяти, необходимой для отладки Вашей программы, и перечислены в порядке возрастания требуемого количества памяти.

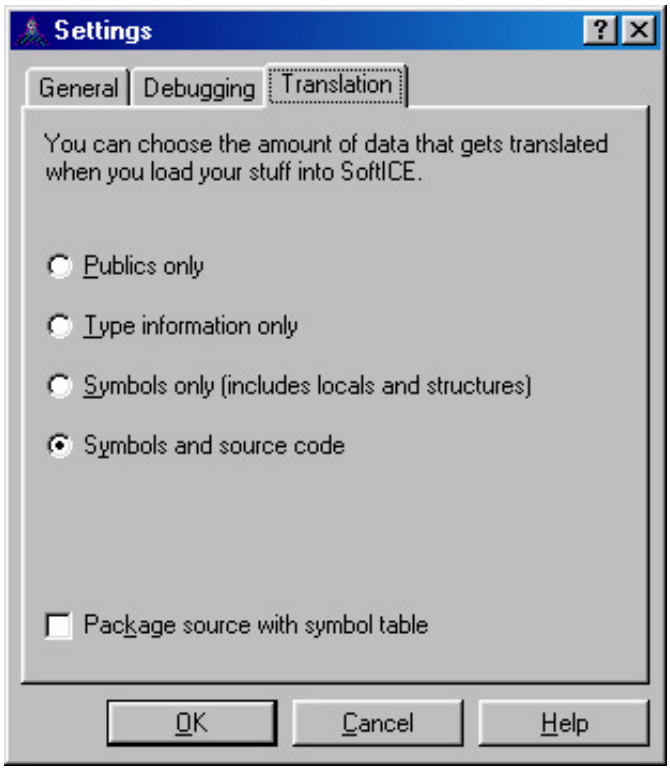
Следующие разделы описывают параметры настройки трансляции (изображение вкладки находится на странице 43).

Public only (Только общие)

В символьный файл транслируются только глобальные имена переменных и функций. Ни информация о типах, ни исходный код в этом случае в файл не включаются.

Type information only (Только информация о типах)

При этом значении параметра транслируется только информацию о типах. Используйте эту установку для добавлении информации о типах структур данных, которую Вы смогли обнаружить при исследовании программ.



Symbols only (Только Символы)

Установка этого значения параметра позволяет транслировать глобальные, статические и локальные имена символов в дополнение к информации об их типах. Исходный текст не включается.

Symbols and source code (Символы и исходный код)

При значении "SYMBOLS AND SOURCE CODE" в символьный файл помещается вся отладочная информация, включая исходный код и номера строк. Эта установка задействована по умолчанию.

Package source with symbol table (Сохранение исходного файла вместе с таблицей символов)

Установка этого флага приводит к сохранению исходный текст Вашей программы вместе с отладочной информацией в файле .NMS, что может понадобится при следующих обстоятельствах:

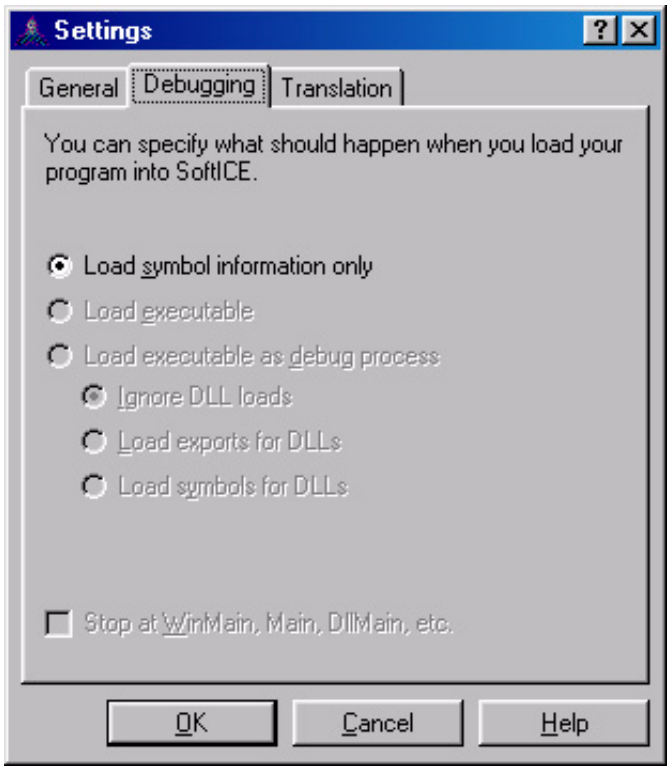
- Загрузка исходного текста на этапе начальной загрузки
  SoftICE не ищет кодовых файлов при начальной загрузке. Если Вам необходимо загрузить исходный текст для VxD или драйвера Windows NT, установите флаг "PACKAGE SOURCE WITH SYMBOLS TABLE". Затем измените параметры инициализации SoftICE так, чтобы загрузить отладочную информацию для VxD или драйвера устройства при запуске. Смотрите раздел "*Предварительная загрузка символической информации и исходного кода*" на странице 121.
- Отладка в системе, которая не имеет доступа к Вашим исходным файлам
  Если Вы хотите отлаживать Ваше приложение в системе, которая не имеет доступа к Вашим исходным файлам, установите при трансляции флаг "PACKAGE SOURCE WITH SYMBOLS TABLE и скопируйте файл" .NMS в эту систему.

Внимание: Если Вы устанавливаете флаг "PACKAGE SOURCE WITH SYMBOLS TABLE", исходный текст Вашей программы будет доступен любому, кто об-

ращается к таблице символов. Если Вы не хотите, чтобы другие имели доступ к исходному тексту, но Вы предоставляете .NMS файл вместе с приложением, то выключите этот флаг.

Изменение параметров отладки (вкладка Debugging)

Параметры отладки определяют, какой тип информации загружать, и нужно ли останавливаться в точке входа Вашего модуля.



Следующие разделы описывают настройки параметров отладки.

Load symbol information only (Загрузка только символьной информации)

"LOAD SYMBOL INFORMATION ONLY" позволяет загрузить только файл символов .NMS, но не загружает при этом исполняемый код. Исходные файлы загружаются в том случае, если Вы выбрали значение параметра трансляции "SYMBOLS AND SOURCE CODE". По умолчанию, Symbol Loader использует эту установку для файлов .DLL, .SYS, и .VxD типов.

Load executable (Загрузка исполняемого кода)

"LOAD EXECUTABLE" позволяет загрузить и Вашу исполняемую программу, и файл .NMS. Также загружаются соответствующие исходные файлы, если Вы выбрали значение параметра трансляции SYMBOLS AND SOURCE CODE. По умолчанию, Symbol Loader использует эту установку для файлов типа .EXE.

Stop at WinMain, Main, DLLMain, etc. (Останов в WinMain, Main, DllMain и т.д.)

Установка этого флага создает контрольную точку в первом основном модуле, с которым сталкивается SoftICE в процессе загрузки Вашего приложения.

Задание исходных файлов программы

По умолчанию производится загрузка всех исходных файлов приложения, которые упомянуты в отладочной информации. В зависимости от Ваших задач исходные коды всей программы могут Вам и не понадобиться. Кроме того, если количество исходных файлов велико, загрузка их всех непрактична.

Во избежание загромождения памяти ненужными исходными текстами, SoftICE позволяет Вам использовать .SRC-файл, который определяет, что именно необходимо загрузить для данного сеанса работы. Этот текстовый файл, содержащий список имен исходных файлов (по одному в строке) создается в том же каталоге, где размещено Ваше приложение; имя файла совпадает с именем этого приложения, но имеет расширение .SRC. Файл .SRC.

Пример: Если у Вас есть программа PROGRAM.EXE, Вы можете создать .SRC-файл с именем PROGRAM.SRC. Содержимое его будет выглядеть примерно так:

```
FILE1.C
FILE3.CPP
FILE4.C
```

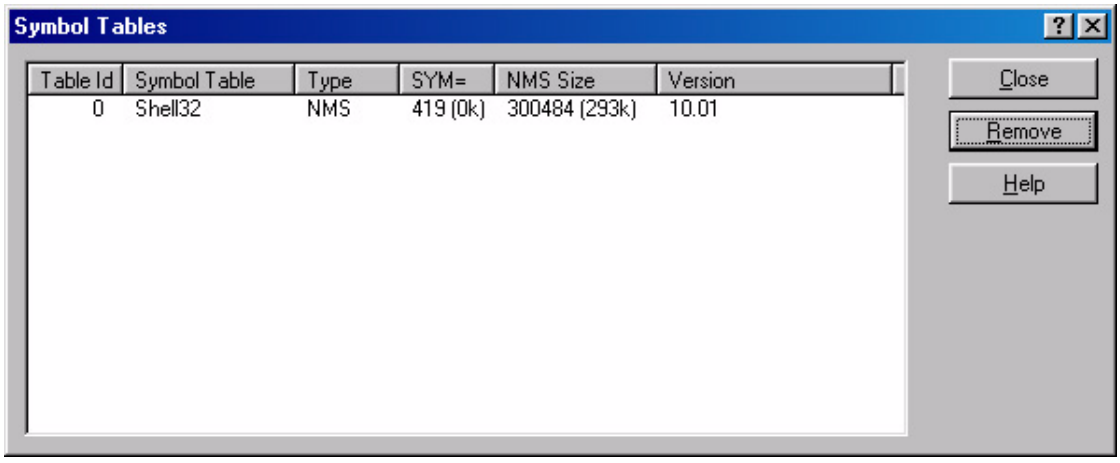
Несмотря на то, что FILE2.C — это нормальный исходный файл приложения, он не будет загружен, потому что его нет в .SRC файле, а загружены будут только файлы FILE1.C, FILE3.CPP и FILE4.C.

Удаление таблиц символов

Каждый раз, когда Вы транслируете приложение, утилита Symbol Loader создает .NMS-файл отладочной информации в форме таблицы символов. После загрузки модуля, Symbol Loader сохраняет таблицу в памяти до тех пор, пока Вы ее не удалите или не перезагрузите компьютер.

Чтобы удалить таблицу символов, сделайте следующее:

1 Выберите пункт "SYMBOL TABLES" в меню "EDIT".



2 Выберите в списке TABLE ID таблицы, которые Вы хотите удалить и нажмите "REMOVE" (Удалить).

Совет: Используйте заголовки таблиц для их сортировки по идентификатору, типу, .SYM=, размеру .NMS или номеру версии. Список всегда сортируется в порядке возрастания.

Запуск утилиты Symbol Loader из командной строки DOS

Утилита Symbol Loader (LOADER32.EXE) поддерживает интерфейс командной строки, который позволяет Вам использовать многие ее свойства из командной строки DOS без использования графического интерфейса. Таким образом, Вы можете автоматизировать многие часто встречающиеся задач.

Прежде чем запускать LOADER32.EXE из командной строки DOS, с помощью графического интерфейса утилиты установите пути поиска файлов и определите параметры трансляции и отладки для каждого из модулей, которые Вы планируете загружать. Symbol Loader сохраняет эти параметры настройки для каждого файла и использует их, когда Вы запускаете ее, чтобы загружать или транслировать файлы из командной строки DOS. Подробности настройки смотрите в разделе "*Изменение параметров настройки модуля*" на странице 40.

Для запуска LOADER32.EXE войдите в каталог, который содержит LOADER32.EXE, или укажите каталог SoftICE в путях поиска файлов.

Синтаксис команды

LOADER32.EXE имеет следующий синтаксис:
`LOADER32 [[`*параметр*`(ы)] `*имя-файла*`]`

Имя-файла задает файл, который Вы хотите транслировать или загрузить. Необязательные параметры описаны в следующей таблице.

Параметр	Описание
/EXPORTS	Загружает экспортируемую информацию
/LOAD	Транслирует модуль в .NMS-файл, если такой файл еще не создан, и загружает его в SoftICE. Если Вы предварительно установили для данного файла параметры трансляции и отладки, то LOADER32.EXE при работе будет использовать их, в противном случае используются установки по умолчанию для данного типа модуля.
/LOGFILE	Сохраняет протокол работы SoftICE в .LOG-файле.
/NOPROMT	Указывает LOADER32.EXE не выводить сообщение при невозможности найти файл с исходными кодами.
/PACKAGE	Сохраняет Ваш исходный код вместе с отладочной информацией в .NMS-файле.
/TRANSLATE	Транслирует модуль в .NMS-файл, используя настройки последнего сеанса трансляции, или, если таковых нет, стандартные настройки для данного типа модулей.

При формировании команды имейте в виду следующее:

- Параметры не являются обязательными. Если Вы запускаете утилиту без параметров, LOADER32.EXE загружает графический интерфейс Symbol Loader и открывает указанный файл.
- Укажите одновременно параметры /TRANSLATE и /LOAD, чтобы LOADER32.EXE оттранслировал модуль перед его загрузкой.
- Не используйте параметры /EXPORTS или /LOGFILE совместно с другими параметрами.

Замечание: Если Вы задаете какой-либо параметр, LOADER32.EXE не загружает графический интерфейс, если только не сталкивается с ошибкой. Если это произойдет, то LOADER32.EXE выведет ее в своем графическом окне.

Использование утилиты загрузчика символов командной строки

NMSYM — это утилита, которая позволяет Вам создать пакет для трансляции и загрузки отладочной информации для использования с SoftICE или другими программами, которые используют таблицы символов формата NM32™. NMSYM предоставляет ряд возможностей, аналогичных свойствам Symbol Loader (LOADER32.EXE) и выполняет следующие функции:

- Трансляция и загрузка отладочной информации для каждого модуля в отдельности.
- Загрузка и выгрузка таблиц символов и экспортируемых модулем данных.
- Сохранение содержимого протокола работы SoftICE в файле.
- Выдача информации о версии продукта и справку.

В следующей таблице перечислены параметры каждой из этих функций:

Функция	**Параметр NMSYM**
Трансляция и загрузка символьной информации для каждого модуля в отдельности.	/TRANSLATE или /TRANS /LOAD /SOURCE /ARGS /OUTPUT или /OUT /PROMPT
Загрузка и выгрузка групп таблиц символов и экспортов модуля.	/SYMLOAD или /SYM /EXPORTS или /EXP /UNLOAD
Сохранение содержимого буфера протокола работы SoftICE в файле.	/LOGFILE или /LOG
Выдача информации о версии продукта и справки.	/VERSION или /VER /HELP или /H

Синтаксис утилиты NMSYM

NMSYM.EXE имеет следующий синтаксис:
`NMSYM [`*`параметр`*`(ы)] <`*`имя-модуля`*`>`

- параметры перечисляются друг за другом через косую черту (/).
- *имя-модуля* является именем файла, который Вы хотите транслировать или загрузить.

Следующий пример показывает правильный вариант задания командной строки:
```
NMSYM /TRANSLATE C:\myproj\myproject.EXE
```

Использование модификаторов параметров и списка файлов

Многие параметры имеют собственные дополнительные модификаторы и возможность задания списка файлов, что позволяет конкретизировать режим работы параметра, а также выполнять операции над группами файлов.

Модификаторы параметров имеют следующий синтаксис:
`/параметр:<`*`модификатор-параметра`*`> [<, `*`модификатор-параметра`*`>]`

Параметр заканчивается двоеточием (:), после которого через запятую идет список модификаторов.

Следующий пример демонстрирует использование параметра /TRANSLATE с модификаторами SOURCE и PACKAGE для трансляции исходного кода и символов, и упаковки исходных текстов в .NMS-файл:

```
/TRANSLATE:SOURCE, PACKAGE
```

Списки файлов задаются следующим образом:

```
/параметр:<имя-файла | путь-поиска>[; <имя-файла | путь-поиска>]
```

В следующем примере параметр /SOURCE используется с тремя дополнительными путями поиска. NMSYM использует их для обнаружения исходных файлов в процессе трансляции и загрузки:

```
/SOURCE:c:\myproj\i386; c:\myproj\include; c:\msdev\include;
```

Использование NMSYM для трансляции отладочной информации

Основное назначение утилиты NMSYM состоит в извлечении из модуля сгенерированной транслятором отладочной информации и перевод ее в формат NM32, а затем сохранение этой информации в .NMS-файле. С этой целью используются следующие параметры командной строки NMSYM:

1 Параметр /TRANSLATION — для задания типа отладочной информации, которую Вы хотите получить.
2 Параметр /SOURCE определяет пути, по которым NMSYM будет искать исходные файлы.
3 Если Вы хотите самостоятельно определить имя .NMS-файла, то используйте параметр /OUTPUT.
4 Задайте имя модуля, который Вы собираетесь транслировать.

```
NMSYM /TRANSLATE C:\myproj\myproject.EXE
```

Следующие разделы описывают параметры, которые используются для управления трансляцией символьной информации каждого конкретного модуля.

/TRANSLATE:<*список-модификаторов*>

Параметр /TRANSLATE позволяет определить тип отладочной информации, которую Вы хотите транслировать в .NMS-файл, а также нужно ли заносить в него исходный код. Кроме того, имеется возможность выполнить повторную трансляцию, несмотря на то, что файл с отладочной информацией уже создан и еще не устарел.

Параметр /TRANSLATE имеет ряд дополнительных модификаторов, конкретизирующих режимы работы утилиты.

Модификаторы режимов трансляции отладочной информации

Следующая таблица содержит список необязательных модификаторов, которые определяют, какая отладочная информация будет транслироваться. Может быть задан только один модификатор. По умолчанию утилита NMSYM использует модификатор SOURCE.

Модификатор	Описание
PUBLICS	Включаются только общедоступные (public) (глобальные) символы. Статические функции и переменные исключаются. В результате использования этого режима создается аналог .MAP-файла. Таблицы символов имеют наименьший размер.
TYPEINFO	Включена только информация о типе. Символьная информация не включается. Используйте эту опцию, когда Вы хотите внести дополнительную информацию от типах данных без исходного кода или информации отладки.
SYMBOLS	Включает и символьную информацию, и информацию о типах. Исходный код и номера строк не добавляются. Размер таблиц символов относительно невелик.
SOURCE	Это режим трансляции по умолчанию. Включается вся отладочная информация и исходный код.

Замечание: Включение информации об исходном коде не означает непосредственного внесения исходных файлов. Заносятся только имена самих файлов с исходными текстами и номера строк.

Модификаторы упаковки исходного кода

Необязательный модификатор упаковки исходного кода определяет, будет ли NMSYM присоединять исходный код к .NMS-файлу или нет. По умолчанию NMSYM делает следующее:

- Для модулей драйвера устройства исходный код добавляется в .NMS-файл, так как он загружается прежде, чем операционная система будет полностью инициализирована.
- Для прикладных программ, которые исполняются после загрузки операционной системы, исходные тексты приложения в .NMS-файл не включаются.

Имеются следующие модификаторы режима включения исходных кодов в .NMS-файл:

Модификатор	Описание
PACKAGE	Включает исходные файлы с .NMS файлом.
NOPACKAGE	Не включает исходные файлы с .NMS файлом.

Замечание: Если Вы пакуете исходный код с .NMS файлом, то он будет доступен каждому, кто обращается к таблице символов.

Модификатор ALWAYS

По умолчанию, NMSYM не транслирует символьную информацию, если .NMS-файл уже создан, а приложение после этого не подвергалось изменениям. Модификатор ALWAYS позволяет заставить NMSYM провести трансляцию в любом случае.

Примеры использования параметра /TRANSLATE

Следующий пример задает имя модуля без параметра /TRANSLATON, что приводит к использованию при трансляции режимы в зависимости от типа модуля.

```
NMSYM myproj.exe
```

Замечание: Для приложений Win32 или библиотек DLL по умолчанию используется строка "/TRANSLATE:SOURCE, NOPACKAGE", а для модулей драйверов — "/TRANSLATE:SOURCE, PACKAGE".

Следующий пример показывает трансляцию отладочной информации для VxD. Используются модификатор SYMBOLS, чтобы исключить информацию, связанную с исходным кодом и модификатор /NOPACKAGE для запрещения включения исходного кода в .NMS-файл.

```
NMSYM /TRANSLATE:SYMBOLS, NOPACKAGE c:\myvxd.vxd
```

Следующий пример использует задает трансляцию в зависимости от типа модуля, а модификатор /ALWAYS заставляет утилиту выполнить ее в любом случае.

```
NMSYM /TRANSLATE:ALWAYS myproj.exe
```

/SOURCE:<*список-путей-поиска*>

Параметр /SOURCE позволяет определить пути, в пределах которых NMSYM должен искать исходный файл. Во время трансляции (только с модификатором PACKAGE) или загрузки модуля (/LOAD или /SYMLOAD) утилита NMSYM будет пытаться обнаружить все исходные файлы, заданные в таблице .NMS-файла. По умолчанию поиск будет проводиться в этих директориях.

Модификатор "*список-путей-поиска*" — это один или несколько отделенных друг от друга точкой с запятой (;) путей. Параметр /SOURCE может быть задан в одной командной строке несколько раз. Порядок инструкций /SOURCE и путей внутри каждого списка определяет последовательность поиска.

Примеры использования параметра /SOURCE

Следующий пример задает два пути поиска исходных файлов.

```
NMSYM /TRANSLATE:PACKAGE
  /SOURCE:c:\myproj\i386; c:\myproj\include; myproj.exe
```

Следующий пример задает два набора исходных путей.

```
NMSYM /TRANS:PACKAGE
  /SOURCE:c:\myproj\i386; c:\myproj\include;
  /SOURCE:c:\msdev\include; myproj.exe
```

Следующий пример задает основной путь к исходным файлам проекта и использует DOS-оператор '%' для извлечения пути из стандартной переменной окружения INCLUDE=. Список путей расширяется, чтобы включить c:\myproj\i386 и пути, перечисленный в переменной окружения INCLUDE=.

```
NMSYM /TRANS:PACKAGE /SOURCE:c:\myproj\i386;
  %INCLUDE% myproj.exe
```

Замечание: В случае, если исходный код не найден, параметр /PROMPT определяет, следует ли пропустить этот файл, или просить у Вас помощи для его поиска.

/OUTPUT:<*имя-файла*>

NMSYM составляет имя .NMS-файла из имени основного модуля, добавляя к нему стандартное для таблиц NM32 расширение .NMS. Образованный файл сохраняется в той же директории, где находится транслируемый модуль. Если Вам необходимо изменить получаемое по умолчанию имя или расположение .NMS-файла, то сделать это можно с помощью параметра /OUTPUT. Если Вы задаете имя, но не определяете путь для записи файла, то он будет сохранен вместе основным модулем.

Примеры использования параметра /OUTPUT

В следующем примере путь к .NMS файлу изменен на общий каталог для .NMS-файлов.

```
NMSYM /output:c:\ntice\symbols\myproj.nms
   c:\myproj\myproject.exe
```

/PROMPT

NMSYM — это командная утилита, специально разработанная для автоматизации трансляции и загрузки отладочной информации. Скорее всего, Вам не захочется получать сообщения об отсутствии исходных файлов, однако в некоторых случаях это могло бы пригодиться. Параметр /PROMPT позволяет NMSYM определить, следует ли ее просить у Вас помощи для обнаружения файлов исходных кодов при использовании параметров /TRANSLATE:PACKAGE, /LOAD или /SYMLOAD.

Использование NMSYM для загрузки модуля и отладочной информации

Подобно режиму трансляции функционирование параметра /LOAD в утилите NMSYM спроектировано таким образом, чтобы иметь возможность работать с каждым конкретным модулем отдельно. Любой модуль будет оттранслирован и загружен. Если Вы не собираетесь транслировать или загружать и исполнять модуль, то следует использовать параметр /SYMLOAD.

Пример использования NMSYM для трансляции, загрузки и исполнения модуля:

```
NMSYM /TRANS:PACKAGE /LOAD:EXECUTE myproj.exe
```

Следующий пример показывает альтернативные возможности загрузки группы транслированных отладочных файлов с использованием параметра /SYMLOAD:

```
NMSYM /SYMLOAD:ntdll.dll; ntoskrnl.nms; myproj.exe
```

В этом примере будут загружены три символьные таблицы, однако трансляции не произойдет, даже если соответствующие .NMS-файлы устарели. Также не произойдет и запуска файла MYPROJ.EXE для отладки.

/LOAD:<*список-модификаторов*>

Параметр /LOAD позволяет загружать файлы с отладочной информацией .NM32 в SoftICE, и, по желанию, запускать модуль для отладки.

Параметр имеет следующие модификаторы.

Модификаторы типов загрузки

Данные модификаторы используются для определения того, каким образом будут загружены модуль и его отладочная информация. Значение модификатора по умолчанию зависит от типа модуля: для исполняемых программ — это EXECUTE, для неисполняемых — SYMBOLS.

Модификатор	Описание
SYMBOLS	Будет загружена только отладочная информация, с помощью которой после загрузки модуля Вы можете устанавливать в нем контрольные точки.
EXECUTE	Загружается и отладочная информация, и исполняемая программа с возможностью ее дальнейшей отладки.

Модификаторы остановки после загрузки

Для управления режимом установки прерывания на входной точке модулей используется один из следующих модификаторов:

Модификатор	Описание
BREAK	Устанавливает команду прерывания на входной точке модуля (WinMain, DllMain, илиDriverEntry).
NOBREAK	Не устанавливает контрольную точку на входе модуля.

Наличие этого модификатора обусловлено зависимостью значения этого параметра по умолчанию от типа модуля: для прикладных программ оно равно "BREAK", а для всех остальных модулей — "NOBREAK".

Модификатор NOSOURCE

NOSOURCE запрещает загрузку файлов исходного кода, даже если в .NMS-файл включен список файлов исходных текстов или информация о номерах строк.

Примеры использования параметра /LOAD

В следующем примере NMSYM загрузит и (по умолчанию) запустит модуль MYPROJ.EXE; если таблица символов устарела, то будет выполнена трансляция для данного типа модуля:

```
NMSYM /LOAD myproj.exe
```

В следующем примере программа должна быть запущена, но контрольной точки на входе программы не устанавливается. В случае необходимости будет выполнен определенный для данного типа модуля вид трансляции.

```
NMSYM /LOAD:NOBREAK myproj.exe
```

В следующем примере будет загружена только символьная информация и явно определен тип трансляция — PUBLIC:

```
NMSYM /TRANS:PUBLIC /LOAD:SYMBOLS myproj.dll
```

/ARGS:<*аргументы-программы*>

Параметр /ARGS служит для передачи аргументов в исполняемый модуль. Эта опция полезна только при совместном использовании с параметром /LOAD:EXECUTE.

<*Аргументы-программы*> — это строка, задающая передаваемые программе аргументы. Если она содержит пробел, то вся строка должна быть заключена в двойные кавычки (").

Примеры использования параметра /ARGS

В следующем примере модуль MYPROJ.EXE будет загружен для отладки, и в него будет передан аргумент TEST.RTF.

```
NMSYM /LOAD:EXECUTE /args:test.rtf myproj.exe
```

В следующем примере командная строка немного усложнена, так что приходится заключать ее в двойные кавычки (''):

```
NMSYM /LOAD:EXECUTE "/ARGS:/PRINT /NOLOGO test.rtf" myproj.exe
```

Использование двойных кавычек предотвращает ошибку NMSYM при разборе командной строки из-за при наличии пробела в параметрах программы: /PRINT_/NOLOGO_test.rtf (пробел отмечен знаком подчеркивания).

Использование NMSYM для загрузки таблиц символов или экспортов

Кроме трансляции и загрузки утилита NMSYM также предоставляет возможность пакетной загрузки и выгрузки таблиц символов и экспортируемой информации. Это чрезвычайно полезно при загрузке "окружения" или связанного набора файлов с отладочной информацией. Например, если Вы запускаете SoftICE вручную, то можете применить NMSYM для получения возможностей, аналогичных использовании параметров настройки для загрузки символической и экспортируемой информации.

Например, Вы можете создать пакетный файл, подобный приведенному ниже, для управления загрузкой отладочной информации. Пакет содержит один дополнительный параметр, который определяет принадлежность загружаемых файлов — для отладки драйвера или прикладной программы (по умолчанию — прикладная программа). В обоих случаях загружается информация, экспортируемая стандартными модулями Windows.

```
net start ntice
ECHO OFF
if "%1" == "D" goto dodriver
if "%1" == "d" goto dodriver
REM *** Это набор для отладки прикладных программ ***
set SYMBOLS=ntdll.dll; shell32.dll; ole32.dll; win32k.sys
goto doload
:dodriver REM *** это набор для отладки драйверов ***
set SYMBOLS=hal.dll; ntoskrnl.exe;
:doload
NMSYM /SYMLOAD:%SYMBOLS% EXPORTS:kernel32.exe;user32.exe;gdi32.exe
```

Другое преимущество использования NMSYM в том, что для поиска .NMS-файлов и модулей не требуется указания полного пути поиска. Если Вы не определяете путь, а заданный модуль или .NMS-файл не будет найден в текущем каталоге, то поиск будет продолжен по дополнительным путям поиска.

/SYMLOAD:<*список-модулей*>

Параметр /SYMLOAD используется для загрузки отладочной информации в SoftICE. Символьные таблицы должны быть предварительно оттранслированы, так как в этом случае утилита трансляцию не выполняет.

Модификатор <*список-модулей*> задает необходимые .NMS-файлы или модули с возможностью явного указания их местонахождения. Если местонахождение модуля не определено, то NMSYM будет пытаться найти файл в текущем каталоге или по дополнительным путям поиска. Если Вы определяете абсолютный или относительный каталог модуля, затем никакого дополнительного поиска производится не будет.

Примеры использования параметра /SYMLOAD

В приведенном ниже примере параметр /SYMLOAD используется для загрузки таблицы символов, обычно используемых для отладки OLE-программ. Здесь пути не задаются, так что при необходимости будет произведен дополнительный поиск.

```
NMSYM /SYMLOAD:ole32.dll; oleaut32.dll; olecli32.dll
```

/EXPORTS:<*список-модулей*>

Параметр /EXPORTS используется для загрузки информации, экспортируемой одним или несколькими модулями, в SoftICE. Экспортируемая информация — это облегченная версия символьной информации об API модуля (обычно из библиотек DLL, но и исполняемые .EXE-файлы также могут экспортировать информацию).

Модификатор <*список-модулей*> задает необходимые модули возможностью явного указания их местонахождения. Если местонахождение модуля не определено, то NMSYM будет пытаться найти файл в текущем каталоге или по дополнительным путям поиска. Если Вы определяете абсолютный или относительный каталог модуля, затем никакого дополнительного поиска производится не будет.

Примеры использования параметра /EXPORTS

Следующий пример демонстрирует применение параметра /EXPORTS для загрузки экспортируемой информации из программ, обычно используемых при отладке OLE-программ. Здесь пути не задаются, так что при необходимости будет произведен дополнительный поиск.

```
NMSYM /EXPORTS:ole32.dll; oleaut32.dll; olecli32.dll
```

Использование NMSYM для выгрузки отладочной информации

NMSYM имеет параметр /UNLOAD, с помощью которого Вы можете программным путем удалять отладочную информацию для определенного набора таблиц символов и(или) экспорта. Это может использоваться для экономии оперативной памяти, занимаемой уже ненужными таблицами символов.

/UNLOAD:<*список-модулей*>

Модификатор <*список-модулей*> определяет таблицы символов или экспорта. Имя таблицы получается из имени основного модуля без пути или расширения. Для обеспечения гибкости и поддержки возможных в будущем изменений имен таблиц, следует задавать пути или расширения, необходимых для однозначного определения таблицы.

Примеры использования параметра /UNLOAD

Следующий пример выполняет действие, обратное описанному в разделах /SYMLOAD и /EXPORTS:

```
NMSYM /UNLOAD:ole32.dll; oleaut32.dll; olecli32.dll
```

SoftICE найдет таблицу, которая соответствует заданному имени модуля и удалит ее, если это окажется возможным, освободив тем самым занимаемую ею память.

Замечание: По умолчанию SoftICE пытается выгрузить таблицу символов. Если заданная таблица символов не существует, то SoftICE, пытается выгрузить экспортную таблицу с тем же именем.

Использование NMSYM для сохранения протокола работы в файл

NMSYM предоставляет возможность сохранить буфер протокола SoftICE в файле при использовании параметр /LOGFILE. Эта операция эквивалентна возможности утилиты Symbol Loader "SAVE SOFTICE HISTORY AS...". NMSYM поддерживает возможность присоединения новых данных к существующему файлу с помощью модификатора APPEND.

/LOGFILE:<*имя-файла*>[,*список-модификаторов*]

Модификатор "*имя-файла*" задает путь и имя файла, в котором будет сохранен буфер протокола. Если путь не задан, то будет использован текущий каталог.

Модификатор параметра /LOGFILE

Модификатор APPEND позволяет Вам добавить текущее содержание буфера протокола работы SoftICE к уже существующему файлу предыдущих протоколов. По умолчанию файл переписывается заново.

Примеры использования параметра /LOGFILE

Следующий пример создаст (или перепишет) файл MYPROJ.LOG с текущим содержанием буфера протокола SoftICE:

```
NMSYM /LOGFILE:myproj.log
```

Следующий пример объединит текущее содержание буфера протокола SoftICE с файлом MYPROJ.LOG (или создаст файл заново):

```
NMSYM /LOGFILE:myproj.log,APPEND
```

Внимание: NMSYM не запрашивает разрешения на перезапись существующего файла, а делает это автоматически.

Получение информации об утилите NMSYM

Для получения информации об утилите NMSYM используются параметры /VERSION и /HELP.

/VERSION

Параметр /VERSION выдает номера версий NMSYM, SoftICE, также как транслятора символьного обработчика. Для правильной работы SoftICE, LOADER32.EXE и NMSYM эти версии должны быть совместимы. Каждая из этих программ сверяет свой номер версии с другими для проверки возможности совместной работы.

/HELP

Параметр /HELP дает краткое описание синтаксиса командной строки, параметров и их модификаторов.

Для меня уже давно стало аксиомой,
что мелочи имеют первостепенное значение.
Сэр Артур Конан Дойл

It has long been an axiom of mine that the little things are
infinitely the most important.
Sir Arthur Conan Doyle

5. Интерфейс SoftICE

Введение

В данной главе описана структура экрана SoftICE и его окон, и управление ими.

Если Вы новичок в SoftICE, то внимательно прочтите эту главу, а затем обращайтесь к ней, как к справочнику. Если Вы уже знакомы с отладчиком, то прочитайте разделы, касающиеся окна локальных переменных и окна слежения, а также разделы, описывающие работу с мышью.

Вызов экрана SoftICE

После загрузки отладчика экран SoftICE автоматически появляется на мониторе в следующих случаях:

- На этапе загрузки SoftICE. По умолчанию инициализационная строка содержит команду X (Exit — Выход) с последующей точкой с запятой, поэтому на данном этапе экран исчезает сразу после возникновения. Подробности смотрите в разделе "*Изменение начальных установок SoftICE*" на странице 119.
- При нажатии клавиатурной комбинации вызова "Ctrl-D". С ее же помощью экран SoftICE может быть снова закрыт.
- При срабатывании контрольной точки.
- При перехвате системной ошибки.
- При крахе Windows NT и появлении "Синего экрана смерти".

При активизации SoftICE вся остальная активность Вашего компьютера прекращается, все прерывания запрещаются, а SoftICE обеспечивает доступ к клавиатуре и вывод на монитор собственными средствами, непосредственно обращаясь к аппаратурному обеспечению компьютера.

Совет: Комбинация вызова отладчика (Ctrl-D) может быть изменена с помощью команды ALTKEY.

Отключение SoftICE на этапе загрузки

Если Вы установили SoftICE под Windows NT в качестве загрузочного или системного драйвера, то Вы имеете возможность отключить его в момент старта системы, нажав клавишу ESC, когда в нижней части экрана появится следующее сообщение:

```
Press Esc to cancel loading SoftICE
(Нажмите Esc для прекращения загрузки SoftICE)
```

Если Вы установили SoftICE под Windows NT в качестве автоматического драйвера, то Вы можете отключить его, только изменив режим загрузки и перезагрузив компьютер. Если загрузка SoftICE вызовет сбой системы, выберите в загрузочном меню Windows NT пункт:

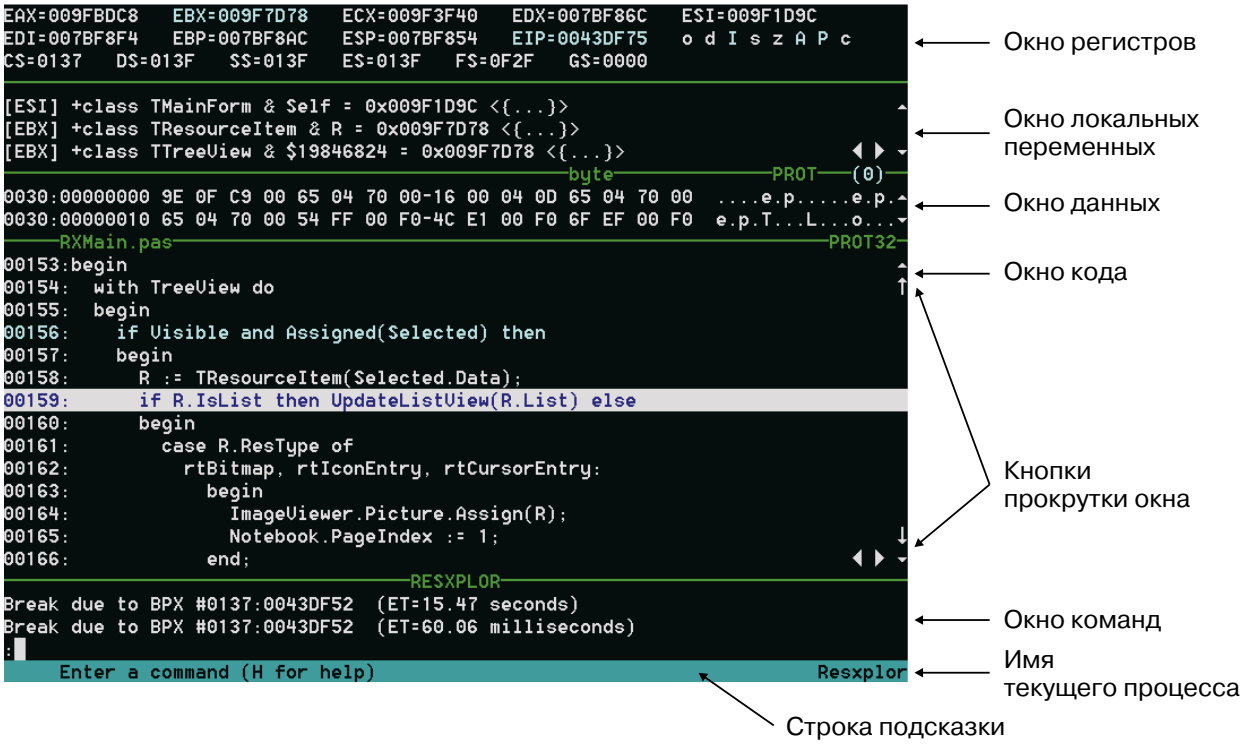
```
Last known good configuration
(Последняя работоспособная конфигурация)
```

Содержимое экрана SoftICE

Экран SoftICE — это Ваш основной инструмент при отладке приложения. Он подразделяется на 7 окон и строку подсказки, позволяющие контролировать различные стороны процесса отладки. В приведенной ниже таблице все окна перечислены в порядке убывания их значимости.

Окно SoftICE	Описание
Окно команд	Ввод команд и выдача сообщений
Окно кода	Вывод машинных инструкций и/или исходных кодов
Окно локальных переменных	Вывод содержимого текущего кадра стека
Окно слежения	Вывод значений переменных, указанных командой WATCH
Окно регистров	Вывод и изменение содержимого регистров и флагов
Окно данных	Отображение и изменение содержимого участка памяти
Окно стека сопроцессора	Вывод содержимого стека (регистров) сопроцессора или MMX-регистров
Строка подсказки	Краткая информация о командах SoftICE

По умолчанию SoftICE выводит на экран строку подсказки и окна команд, кода и локальных переменных. В зависимости от Ваших задач Вы можете открывать и закрывать и другие необходимые Вам окна. На следующем рисунке показан типичный экран отладчика.



Изменение размера экрана SoftICE

По умолчанию SoftICE выводит информацию на экран высотой в 25 строк, используя их для размещения всех своих окно. Если Вы используете цветной монитор (и соответствующую видеокарту), с помощью команды LINES Вы можете изменить размер экрана до 43, 50 или 60 строк[†] . Вывод на монохромные мониторы всегда ограничен 25 строками.

Пример: `LINES 60`

Управление окнами SoftICE

С окнами отладчика можно выполнять следующие действия:

- открывать и закрывать все окна, кроме окна команд;
- изменять размер окон кода, данных и локальных переменных;
- листать содержимое окон кода, команд, данных, локальных переменных и слежения.

Управление окнами может быть осуществлено с клавиатуры или с помощью мыши.

Открытие и закрытие окон

Чтобы открыть необходимое окно, используются команды, перечисленные в следующей таблице. Если окно уже открыто, то та же самая команда закрывает его. Чтобы закрыть окно с помощью мыши, протащите линию, ограничивающую его снизу, до верхней границы.

Команда	Окно
WC	Окно кода
WD	Окно данных
WF	Окно стека сопроцессора
WL	Окно локальных переменных
WR	Окно регистров
WW	Окно слежения

Изменение размеров окон

Для изменения размера окна переместите мышью нижнюю границу окна так, как Вам это необходимо. С клавиатуры это можно сделать с помощью тех же команд открытия/закрытия окна, добавив после них десятичное число, означающее высоту окна в строках.

Пример: `WD 7`

Обратите внимание, что окно команд автоматически увеличивается или уменьшается при изменении размеров других окно. (Других способов изменить размер окна команд нет.)

[†] С появлением "Универсального видеодрайвера", начиная с версии 3.2, появилась возможность менять размер экрана в пределах от 25 до 128 строк в высоту и от 80 до 160 символов в ширину.

Перемещение курсора из окна в окно

По умолчанию курсор размещается в окне команд. Для его перемещения щелкните мышью в нужном Вам окне. Если курсор находится в окнах команд или кода, то перечисленными ниже комбинациями клавиш его можно переместить в нужное окно (и вернуть назад).

Окно	Комбинация клавиш
Окно кода	Alt-C
Окно данных	Alt-D
Окно стека сопроцессора	Переместить курсор в это окно невозможно
Окно локальных переменных	Alt-L
Окно регистров	Alt-R
Окно слежения	Alt-W

Пролистывание содержимого окна

Вы можете листать содержимое окон кода, команд, данных, локальных переменных и слежения. Для окон стека сопроцессора и регистров это невозможно, так как их размер всегда равен 3 и 4 строкам, соответственно.

SoftICE позволяет листать содержимое как с клавиатуры, так и с помощью мыши. Все эти способы описаны в следующей таблице.

Замечание: Комбинации клавиш для некоторых окон могут меняться. Некоторые окна, например, не позволяют переходить к первой и последней строкам. Полное описание способов пролистывания информации в окнах дается в соответствующих разделах.

Размер и направление листания окна	Комбинация клавиш	Действие мышью
На одну страницу назад	PageUp	Щелкните по внутренней стрелке вверх
На одну страницу вперед	PageDown	Щелкните по внутренней стрелке вниз
На одну строку назад	Стрелка вверх	Щелкните по наружной стрелке вверх
На одну строку вперед	Стрелка вниз	Щелкните по наружной стрелке вниз
Перейти к первой строке кода	Home	Невозможно
Перейти к последней строке кода	End	Невозможно
На один символ влево	Стрелка влево	Щелкните по стрелке влево
На один символ вправо	Стрелка вправо	Щелкните по стрелке вправо

Копирование и вставка данных в окнах

Если у Вас в системе есть мышь, то Вы можете копировать и переносить данные между окнами, что очень удобно, например, для копирования адресов и данных в выражения. Эти действия выполняются следующим образом:

1 Выделите данные, которые Вы хотите копировать.
2 Нажмите правую кнопку мыши, чтобы появилось контекстное меню.
3 Выберите левой кнопкой мыши необходимую команду.

Команда	Описание
Copy	Копирует выделенные данные в буфер
Copy and Paste	Копирует выделенные данные и вставляет их в место, указываемое текущим положением курсора
Paste	Вставляет данные из буфера в место, указываемое текущим положением курсора

Ввод команд с помощью мыши

С помощью мыши можно ввести команды D, U и WHAT. (Подробное описание команд можно найти в "*Справочнике по командам SoftICE*".)

Для ввода этих команд с помощью мыши выполните следующие действия:

1 Выделите данные, к которым Вы хотите применить команду. Например, выделите выражение для выяснения его типа.
2 Нажмите правую кнопку мыши, чтобы появилось контекстное меню.
3 Выберите левой кнопкой мыши необходимую Вам команду. Сами команды кратко описаны в приведенной ниже таблице.

Команда в меню	Команда SoftICE	Описание
Display	D	Показать содержимое памяти по указанному адресу
Un-Assemble	U	Показать исходный либо дизассемблированный код по указанному адресу
What	WHAT	Определить, относится ли имя или выражение к известному типу
Previous	нет	Повторить последнюю команду

Получение помощи

SoftICE предоставляет 2 способа получения помощи во время отладки приложений: с использованием строки подсказки и по команде H.

Использование строки подсказки

В нижней части экрана SoftICE всегда имеется строка подсказки. Содержимое этой строки обновляется по мере того, как Вы вводите команды. В строке показывается информация следующих видов:

- До тех пор, пока вводимая команда не завершена, в строке перечисляются все возможные команды, которые начинаются с уже введенных Вами символов.

- Если введенная Вами строка совпадает с командой SoftICE, в строке подсказки появляется краткое ее описание.
- Если после команды Вы введете пробел, то Вам будет показан синтаксис данной команды.
- При редактировании содержимого окна регистров или данных в строке подсказки указываются допустимые команды для этого окна.

Использование команда H

Команда H выдает общие сведения по всем командам SoftICE или более детальную информацию по какой-либо конкретно указанной Вами команде. Для получения краткого описания всех команд необходимо выполнить команду H без параметров.

Для получения более детального описания какой-либо конкретной команды необходимо указать эту команду после H в качестве параметра. SoftICE покажет описание команды, синтаксис и пример ее использования.

Приведенный ниже пример показывает получение подсказки о команде BPINT:

```
:H BPINT

Breakpoint on interrupt
BPINT interrupt-number {IF expression] [DO bp-action]
ex: BPINT 50

(Установка контрольной точки на прерывание
BPINT номер-прерывания [IF выражение] [DO действие]
пример: BPINT 50)
```

Окно команд

Окно команд позволяет Вам вводить необходимые для управления работой отладчика команды и выводит информацию о текущем процессе отладки. Содержимое окна сохраняется в буфере протокола SoftICE.

Окно команд всегда открыто и имеет по крайней мере 2 строки в высоту. Хотя Вы и не можете непосредственно изменить размер окна, тем не менее его высота автоматически меняется при изменении размеров других окон.

Листание содержимого окна команд

Для листания содержимого окна команд используются следующие клавиатурные комбинации.

Функция	**Комбинация клавиш**
Перейти к предыдущей странице содержимого буфера протокола	PageUp
Перейти к следующей странице содержимого буфера протокола	PageDown
Перейти к предыдущей строке содержимого буфера протокола	Стрелка вверх
Перейти к предыдущей строке содержимого буфера протокола	Стрелка вниз

Ввод команд

Команды SoftICE можно вводить, когда курсор находится в окне команд или кода. Для ввода команды наберите ее на клавиатуре и нажмите ENTER.

Совет: По мере ввода команды в строке подсказки перечисляются все допустимые команды, которые начинаются с уже введенных символов. Если в этой строке показывается только одна команда, то нажатием пробела можно автоматически завершить ввод. SoftICE самостоятельно дополнит недостающие символы.

После выполнения команды выдаваемая ею информация появляется непосредственно после нее в окне команд. Если эта информация выводится в последней строке окна, то окно автоматически пролистывается. Если выдаваемая информация не умещается в окне, то появляется следующее сообщение:

```
Any Key To Continue, ESC To Cancel
(Нажмите любую клавишу для продолжения
 или ESC для прекращения вывода)
```

Для отключения этой подсказки можно ввести команду: SET PAUSE OFF

Синтаксис команд

Команды SoftICE имеют следующие общие правила построения:

- Все команды представляют собой нечувствительные к регистру строки длиной от 1 до 6 символов.
- Все параметры представлены ASCII-строками или выражениями.
- Адрес в SoftICE может быть представлен парой селектор:смещение или сегмент:смещение, либо одним смещением.
- В выражениях могут использоваться:
  - ◊ символы группировки — круглые скобки '(', ')'
  - ◊ числа в шестнадцатеричном или десятичном формате
  - ◊ адреса
  - ◊ номера строк
  - ◊ строковые литералы
  - ◊ идентификаторы
  - ◊ операторы
  - ◊ встроенные функции
  - ◊ регистры

Пример: Выражением является строка (1 + 2)*3

Любой команде, которая принимает в качестве параметра число или адрес, может быть передано выражение любой сложности. Для вычисления выражения может быть использована команда '?' (вопросительный знак). Кроме того, контрольные точки могут содержать условия, основанные на результате вычисления выражения, а именно, контрольная точка срабатывает только в том случае, когда результат имеет не нулевой значение (TRUE, Истина).

Использование функциональных клавиш

SoftICE присваивает некоторым функциональным клавишам (и их комбинациям) значения наиболее часто используемых команд. В следующей таблице приведен список таких назначений.

Клавиша	Команда	Функция
F1	H	Выдать подсказку
F2	WR	Открыть/закрыть окно регистров
F3	SRC	Переключиться между режимами исходных кодов, смешанных кодов или только ассемблерных команд
F4	RS	Показать экран отлаживаемого приложения
F5	X	Перейти
F6	EC	Перевести курсор в окно кодов или из него
F7	HERE	Выполнить приложение до команды, на которую указывает курсор
F8	T	Шаг трассировки (с заходом в функции)
F9	BPX	Установить контрольную точку в текущей строке
F10	P	Выполнить один шаг программы (без захода в функции)
F11	G @SS:EIP	Перейти в вызывающую функцию программу
F12	P RET	Выполнить функцию до выхода в вызывающую программу
Shift-F3	FORMAT	Изменить формат вывода информации в окне данных
Alt-F1	WR	Открыть/закрыть окно регистров
Alt-F2	WD	Открыть/закрыть окно данных
Alt-F3	WC	Открыть/закрыть окно кодов
Alt-F4	WW	Открыть/закрыть окно слежения
Alt-F5	CLS	Очистить содержимое окна команд
Alt-F11	`dd dataaddr->0`	Показать данные по адресу, размещенному в первом двойном слове окна данных
Alt-F12	`dd dataaddr->4`	Показать данные по адресу, размещенному во втором двойном слове окна данных

Вы можете самостоятельно присвоить функциональным клавишам значения необходимых Вам команд. Подробности смотрите в разделе "*Изменение назначений клавиатуре*" на странице 124.

Редактирование командных строк

Для редактирования командных строк используются описанные ниже клавиши:

Функция	Комбинация клавиш
Перевести курсор в начало командной строки	Home
Перевести курсор в конец командной строки	End
Переключение режимов вставки/замены. В режиме вставки вводимые символы вставляются на место курсора (мерцающий блок), сдвигая вправо остаток строки. В режиме замены вводимые символы заменяют символы, указываемые курсором	Insert

Продолжение на следующей странице

Функция	Комбинация клавиш
Удалить указываемый курсором символ со сдвигом остатка строки влево	Delete
Удалить предыдущий символ со сдвигом остатка строки влево	Bksp
Отменить использование вводимой командной строки	Esc
Перемещение курсора влево/вправо в пределах строки	<-, ->

Вызов предыдущих команд

SoftICE помнит последние 32 введенные Вами команды. Любую из них можно вызвать для повторного исполнения или для редактирования с последующим исполнением, как из окна команд, так и из окна кодов.

В следующей таблице приведены способы вызова предыдущих команд из окна команд.

Функция (курсор в окне команд)	Комбинация клавиш
Получить предыдущую команду из буфера протокола	Стрелка верх
Получить следующую команду из буфера протокола	Стрелка вниз

Замечание: Допускается использовать префиксы. Например, если Вы введете символ 'A', то нажатие стрелки вверх (или вниз) будет перемещать Вас только по командам, начинающихся с 'A'.

В следующей таблице приведены способы вызова предыдущих команд из окна кодов.

Функция (курсор в окне кодов)	Комбинация клавиш
Получить предыдущую команду из буфера протокола	Shift-Стрелка верх
Получить следующую команду из буфера протокола	Shift-Стрелка вниз

Использование макрокоманд времени исполнения

Макрокоманды представляют собой разработанные пользователем команды, которые могут использоваться точно также, как и встроенные команды SoftICE. Определение, или тело макрокоманды, содержит последовательность вызовов других команд, в том числе и иные макрокоманды и аргументы командной строки.

Существует 2 способа создания макрокоманд. Можно создать макрокоманды времени исполнения, которые существуют до момента перезагрузки SoftICE, и постоянные макрокоманды, которые автоматически загружаются при старте отладчика. Создание и использование постоянных макрокоманд подробно описаны в разделе "*Работа с постоянными макрокомандами*" на странице 125.

В приведенной ниже таблице описаны способы работы с макрокомандами времени исполнения.

Действие	**Команда**
Создание или изменение макрокоманды	MACRO *имя-макроса* = "*команда1*; *команда2*; ..."
Удаление макрокоманды	MACRO *имя-макроса* *
Удаление всех макрокоманд	MACRO *
Редактирование макрокоманды	MACRO *имя-макроса*
Вывод списка всех макрокоманд	MACRO

Совет: Можно изменить постоянную макрокоманду только на время текущего сеанса отладки. При новом запуске SoftICE эта макрокоманда восстановит свое прежнее содержание.

Тело макрокоманды состоит из последовательности команд SoftICE или других макросов, разделенных точками с запятой. При этом последняя команда не обязательно должна заканчиваться точкой с запятой. На аргументы командной строки макрокоманды можно ссылать в любом месте тела макроса с помощью `%<номер-параметра>`, где *номер-параметра* — это цифры от 1 до 8.

Пример: Макрокоманда MACRO asm = "a %1" определяет псевдоним для команды A (команда ассемблера). Идентификатор "%1" заменяется на первый аргумент после имени макроса "asm" или просто удаляется, если аргумента нет.

Если требуется вставить в тело макрокоманды символ двойных кавычек (") или знак процента (%), то перед ними необходимо добавить символ обратной косой черты (\). Чтобы добавить собственно знак обратной косой черты, используется последовательность из двух таких символов (\\).

Замечание: Рекурсивный вызов макрокоманд возможен, однако, польза от этого приема сомнительна, так как программного способа завершить его работу не существует. Если макрокоманда вызывает саму себя в последней команде тела макроса (так называемый "хвостовой вызов"), то такая команда выполняется до тех пор, пока пользователь не нажмет 'ESC', чтобы прервать ее работу. Если рекурсивный вызов является не последней командой макроса, то такой вызов выполняется 32 раза (ограничение вложенности).

В следующей таблице приведены примеры допустимых макрокоманд.

Макрокоманда времени исполнения	**Пример вызова**
`MACRO Qexp = "addr explorer; Query %1"`	`Qexp` `Qexp 140000`
`MACRO lshot = "bpx %1 do \"bc bpindex\""`	`1shot eip` `1shot @esp`
`MACRO ddt = "dd thread"`	`ddt`
`MACRO ddp = "dd process"`	`ddp`
`MACRO thr = "thread %1 tid"`	`thr` `thr -x`
`MACRO dmyfile =` `"macro myfile = \"TABLE %1;file \%1\""`	`dmyfile mytable` `myfile myfile.c`

Сохранение содержимого буфера протокола окна команд в файл

Буфер протокола окна команд содержит всю информацию, которая выводилась в окне команд. Его сохранение в файле может оказаться полезным по следующим причинам:

- выводились большие массивы данных или содержимого регистров;
- выводились листинги дизассемблированных кодов;
- необходимо сохранить протоколы исполнения контрольных точек, создаваемые командой BPLOG;
- сохранение протоколов работы команды BMSG;
- сохранение отладочных сообщений приложения пользователя с помощью вызова функции OutputDebugString, а также отладочных сообщений программ ядра системы через вызовы функции KdPrint.

Подробности о размерах буфера протокола и способ его изменения описаны в разделе "*Размер буфера протокола*" на странице 120.

Для сохранения буфера протокола в файл необходимо выполнить следующие действия:

1 Удостоверьтесь, что необходимая Вам информация находится в окне команд, то есть была сохранена в буфере протокола работы.
Например, для сохранения содержимого больших массивов памяти отключите окно данных, чтобы быть заставить SoftICE выводить информацию в окно команд.
2 Запустите утилиту Symbol Loader.
3 В меню "FILE" выберите пункт "SAVE SOFTICE HISTORY AS..." или нажмите кнопку "SAVE SOFTICE HISTORY".
4 С помощью диалога сохранения задайте имя файла и каталог.

Связанные с окном команды

Приведенная ниже команда имеет непосредственное отношение к окну команд. Дополнительные подробности смотрите в "*Справочнике по командам SoftICE*".

Команда	Функция
SET [*переменная*] [ON | OFF] [*значение*]	Отображает или устанавливает значение параметров работы

Окно кодов

В окне кодов отображаются исходные тексты программы, дизассемблированный код, или и то, и другое одновременно (смешанный тип вывода). С его помощью Вы можете устанавливать контрольные точки. (Подробности установки контрольных точек описаны в главе 7 "*Использование прерываний*".)

Управление окном кодов

Для управления окном используются следующие команды.

Команда	Описание
WC	Открыть/закрыть окно кодов
WC [*количество-строк*]	Изменить размер окна кодов
Alt-C	Переместить курсор в окно кодов или из него

Листание содержимого окна кодов

Для листания содержимого окна кодов используются стрелки скроллинга (при работе с мышью), либо, если курсор находится в окне кодов, следующие комбинации клавиш.

Функция (курсор в окне кодов)	Комбинация клавиш
Перейти к предыдущей странице содержимого окна кодов	PageUp
Перейти к следующей странице содержимого окна кодов	PageDown
Прокрутить окно кодов на одну строку вверх	Стрелка вверх
Прокрутить окно кодов на одну строку вниз	Стрелка вниз
Перейти к первой строке файла исходных текстов	Ctrl-Home
Перейти к последней строке файла исходных текстов	Ctrl-End
Передвинуть содержимое окна кодов на один символ влево (только в режиме отображения исходных кодов)	Ctrl-Стрелка влево
Передвинуть содержимое окна кодов на один символ вправо (только в режиме отображения исходных кодов)	Ctrl-Стрелка вправо

Можно листать окно кодов, даже если курсор находится в окне команд. Для этого используются следующие комбинации клавиш.

Функция (курсор в окне команд)	Комбинация клавиш
Перейти к предыдущей странице содержимого окна кодов	Ctrl-PageUp
Перейти к следующей странице содержимого окна кодов	Ctrl-PageDown
Прокрутить окно кодов на одну строку вверх	Ctrl-Стрелка вверх
Прокрутить окно кодов на одну строку вниз	Ctrl-Стрелка вниз
Перейти к первой строке файла исходных текстов	Ctrl-Home
Перейти к последней строке файла исходных текстов	Ctrl-End
Передвинуть содержимое окна кодов на один символ влево (только в режиме отображения исходных кодов)	Ctrl-Стрелка влево
Передвинуть содержимое окна кодов на один символ вправо (только в режиме отображения исходных кодов)	Ctrl-Стрелка вправо

Отображение информации

Окно кода может работать в трех различных режимах.

Режим	Описание
Исходных текстов	Если исходный код приложения доступен, то он отображается в окне кодов
Смешанный	В этом режиме каждая строка исходного текста приложения сопровождается соответствующими ей машинными командами.
Кодовый	Отображаются только дизассемблированные команды программы

Переключение между режимами производится с помощью команды SRC (функциональная клавиша F3).

Использование смешанного и кодового режима

Каждая дизассемблированная инструкция в кодовом и смешанном режимах содержит следующие поля.

Поле	Описание
Адрес	Шестнадцатеричный адрес инструкции. Если для данного адреса известен глобальный идентификатор, то он отображается перед данной строкой.
Байты кода	Шестнадцатеричные байты, соответствующие данной инструкции. По умолчанию вывод этого поля отключен, так как эти значения, как правило, не нужны. Для включения их отображения используется команда SET CODE ON.
Инструкция	Ассемблерная мнемоника данной машинной команды. Если адресу, на который ссылается данная инструкция, соответствует известный идентификатор, то он выводится в строке вместо шестнадцатеричного значения адреса. Для того, чтобы отключить вывод идентификаторов и всегда отображать числовые значения адресов, используется команда SET SYMBOLS OFF.
Комментарий	Дополнительные комментарии дизассемблера.

Пример: Ниже показан пример вывода дизассемблированной инструкции:

```
00FD:00001DA1  56      PUSH ESI
```

Кроме этого, дизассемблер SoftICE предоставляет следующие комментарии:

- Команда INT 2E дополняется именем вызываемого сервиса ядра операционной системы и количеством параметров, которое он требует. Если Вы загрузили отладочную информацию для модуля NTOSKRNL, и ее таблица является текущей, то вместо адреса Вы увидите имя вызываемой программы.
- Если в инструкции используется непосредственный операнд, который совпадает со значением кода статуса Windows NT, то его имя будет показано в комментарии.
- Команда INT 21 комментируется именем DOS-функции.
- Команда INT 31 комментируется именем DPMI-функции.
- Так, где это имеет смысл, в качестве меток используются имена сервисов VxD.

Отображение дополнительной информации

Кроме исходных текстов и дизассемблированных команд в окне кодов отображается и некоторая дополнительная информация:

- При активизации экрана SoftICE текущая инструкция, адрес которой находится в регистре EIP, выделяется. Если ею является относительный переход, то в поле комментария дизассемблера содержится строка JUMP или NO JUMP ("Переход" или "Нет перехода", соответственно) в зависимости от того, будет он выполняться или нет. Кроме того, в строке JUMP имеется стрелка вниз или вверх, указывающая направление, куда — вперед или назад по коду будет выполнен этот переход, что помогает определить направление листания содержимого окна кодов, чтобы посмотреть цель перехода.

- Команда-цель инструкции JUMP всегда отмечается подсвеченной стрелкой (⇒), перекрывающей селекторную часть адреса.
- Если инструкция ссылается на содержимое в памяти, то в конце строки инструкции показывается эффективный адрес и содержимое по этому адресу. В случае, если на экране SoftICE открыто окно регистров, то эффективный адрес и значение по этому адресу отображаются в этом окне непосредственно под значениями регистра флагов.
- Если на какой-либо инструкции в окне кодов установлена контрольная точка, то это инструкция показывается жирным шрифтом.
- Разделительные линии выше и ниже окна кодов содержат дополнительную информацию о коде.

  В верхней разделительной линии размещаются:
  - ◊ Символическое имя + смещение
  - ◊ Имя файла исходного кода, если он отображается в окне
  - ◊ Тип сегмента:

V86	Код реального режима с адресацией сегмент:смещение
PROT16	Код 16-битного защищенного режима с адресацией селектор:смещение
PROT32	Код 32-битного защищенного режима с адресацией селектор:смещение

  В нижней разделительной линии размещаются:
  - ◊ Имя модуля Windows, имя секции и смещение, если это 32-битный модуль Windows. Например, `KERNEL32!.Text + 002f`
  - ◊ Имя модуля Windows и номер сегмента в скобках, если это 16-битный модуль Windows. Например, `Display (01)`
  - ◊ Имя владельца кодового сегмента, если это код режима V86. Например, `DOS`.

Ввод команд из окна кодов

Находясь в окне кодов Вы, тем не менее, можете вводить и исполнять команды. Как только Вы наберете первый символ команды, курсор автоматически переходит в окно команд. После того как набор команды завершен нажатием клавиши ENTER или ESC, курсор возвращается в окно кодов. Кроме того, находясь в окне кодов, Вы можете исполнять команды с помощью функциональных клавиш. Дополнительную информацию, касающуюся введения команд смотрите в разделе "*Окно команд*" на странице 61.

В следующей таблице приведены наиболее полезные команды.

Команда	Функция
. (точка)	Показать инструкцию по текущему адресу (EIP)
A *адрес*	Ассемблировать инструкцию непосредственно в память
BPX (F9)	Установить постоянную контрольную точку в строке, где находится курсор
FILE *имя-файла*	Выбрать файл с исходными кодами для отображения в окне. Параметр *имя-файла* может содержать неполное имя. Если Вы не знаете точного имени, то по команде FILE * можно просмотреть список всех загруженных файлов исходных текстов.
HERE (F7)	Установить одноразовую контрольную точку в текущей строке
SET	Показать или изменить переменные окружения SoftICE

Продолжение на следующей странице

Команда	Функция
SRC	Переключение между режимами отображения окна кодов: режимом кода, исходных текстов и смешанным режимом.
SS *строка*	Показать в окне кодов следующую инструкцию, содержащую заданную подстроку
TABS *позиции-табуляции*	Установить позиции табуляции для отображения исходного текста приложения.
U *адрес*	Дизассемблировать код по любому указанному адресу. Если в качестве адреса Вы укажите имя функции, то SoftICE переместит отображение окна кодов на указанную Вами функцию.

Полное описание этих команд приведено в "*Справочнике по командам SoftICE*".

Окно локальных переменных

В окне локальных переменных отображается текущий кадр стека. Вы можете видеть содержимое структур, массивов и символьных строк, размещенных в стеке.

Управление окном локальных переменных

Для управления окном локальных переменных используются следующие команды.

Команда	Описание
WL	Открыть/закрыть окно локальных переменных
WL [*количество-строк*]	Изменить размер окна локальных переменных
Alt-L	Переместить курсор в окно локальных переменных или из него

Листание содержимого окна локальных переменных

Для листания окна содержимого используются стрелки скроллинга (при работе с мышью), либо переместите курсор в окно командой Alt-L и примените необходимую комбинацию клавиш.

Функция	Комбинация клавиш
Перейти к предыдущей странице содержимого окна локальных переменных	PageUp
Перейти к следующей странице содержимого окна	PageDown
Прокрутить окно на одну строку вверх	Стрелка вверх
Прокрутить окно на одну строку вниз	Стрелка вниз
Перейти к первому объекту содержимого окна	Home
Перейти к последнему объекту содержимого окна	End
Сместить содержимое окна на один символ влево	Стрелка влево
Сместить содержимое окна кодов на один символ вправо	Стрелка вправо

Раскрытие и сжатие содержимого стека

Вы можете раскрывать структуры, массивы и символьные строки, отмеченные знаком "плюс" (+), для просмотра их содержимого. Для раскрытия и сжатия содержимого объекта сделайте следующее:

- Только на компьютерах с процессором класса Pentium или выше — Двойной щелчок на объекте.
- На всех компьютерах — Перейдите в окно локальных переменных (Alt-L), переместите курсор на необходимый объект и нажмите клавишу ENTER.

Связанные с окном команды

Приведенные ниже команды имеют непосредственное отношение к окну локальных переменных. Дополнительные подробности о них смотрите в "*Справочнике по командам SoftICE*".

Команда	Функция
LOCALS	Показать список переменных текущего кадра стека.
TYPES [*имя-типа*]	Список всех заданных типов в текущем контексте или информацию о типе, указанном в параметре.

Окно слежения

Окно слежения позволяет постоянно наблюдать за значениями выражений, введенными с помощью команды WATCH. Дополнительную информацию по этой команде можно получить в "*Справочнике по командам SoftICE*".

Управление окном слежения

Для управления окном используются следующие команды.

Команда	Описание
WW	Открыть/закрыть окно слежения
WW [*количество-строк*]	Изменить размер окна слежения
Alt-W	Переместить курсор в окно слежения или из него

Листание содержимого окна слежения

Для листания окна содержимого используются стрелки скроллинга (при работе с мышью), либо переместите курсор в окно командой Alt-W и примените необходимую комбинацию клавиш.

Функция	Комбинация клавиш
Перейти к предыдущей странице содержимого окна слежения	PageUp
Перейти к следующей странице содержимого окна	PageDown
Прокрутить окно на одну строку вверх	Стрелка вверх
Прокрутить окно на одну строку вниз	Стрелка вниз

Продолжение на следующей странице

Функция	Комбинация клавиш
Перейти к первому объекту окна слежения	Home
Перейти к последнему объекту окна слежения	End
Сместить содержимое окна на один символ влево	Стрелка влево
Сместить содержимое окна кодов на один символ вправо	Стрелка вправо

Создание выражений для отслеживания их значений

Выражение, за значением которого Вы хотите наблюдать, задается с помощью команды WATCH. В выражении могут использоваться как глобальные, так и локальные переменные, регистры и адреса.

Замечание: Для наблюдения за выражениями, содержащими локальные переменные Вы должны находиться в области их видимости.

Следующие примеры иллюстрируют применение команды WATCH.

Пример: Наблюдение за значением "ds:esi":
```
WATCH ds:esi
```

Пример: Наблюдение за значением, на которое указывает содержимое "ds:esi":
```
WATCH *ds:esi
```

Удаление отслеживаемых выражений

Удалить выражение, заданное в окне слежения можно с помощью мыши или с клавиатуры. С помощью мыши нужно выделить выражение, щелкнув по нему, а затем нажать "DELETE". При работе с клавиатурой необходимо перейти в окно с помощью клавиатурного сочетания Alt-W, перевести курсор на необходимое выражение и нажать "DELETE".

Отображение информации

Окно слежения содержит следующие поля (слева направо):

Поле строки	Описание
Выражение	Содержит собственно выражение, заданное командой WATCH. Оно вычисляется всякий раз при отображении окна
Определение типа	Тип выражения
Значение	Текущее значение наблюдаемого выражения

Развертывание и сжатие выражений

Вы можете развернуть выражения, помеченные значком "плюс" (+), чтобы просмотреть его содержимое. Развертывание и сжатие можно выполнить с помощью следующих действий:

- Только на компьютерах с процессором класса Pentium или выше — Двойной щелчок мыши на необходимом выражении.

- На всех компьютеров — Перейти в окно слежения командой Alt-W, найти необходимое выражение и нажать ENTER.

Связанные с окном команды

Приведенная ниже команда имеет непосредственное отношение к окну слежения. Дополнительные подробности о ней смотрите в "*Справочнике по командам SoftICE*".

Команда	Функция
WATCH [*выражение*]	Добавление выражений в окно слежения.

Окно регистров

Окно регистров демонстрирует текущие значения регистров, флагов и эффективный адрес, если таковой используется в команде. Окно может быть использовано для определения, какие регистры были модифицированы во время вызова процедуры или для изменения содержимого регистров или флагов.

Управление окном регистров

Для управления окном используются следующие команды.

Команда	Описание
WR	Открыть/закрыть окно регистров
Alt-R	Переместить курсор в окно регистров или из него

Если Вы не пользуетесь окном регистров, то закройте его, чтобы освободить дополнительное место для отображения других окон.

Отображение информации

Первые три строки окна показывают содержимое регистров, флаги и эффективный адрес, если последний используется в команде:

```
EAX   EBX   ECX   EDX   ESI
EDI   EBP   ESP   EIP               o d i s z a p c
CS    DS    SS    ES    FS    GS   эффективный-адрес
```

При выполнении команд T (Trace — трассировка), P (Step over — пошаговое исполнение) или G (Go to — перейти к) SoftICE подсвечивает измененные регистры. Это свойство особенно полезно для выявления вопроса, какие регистры изменяются во время вызова процедуры.

Во 2й строке окна находятся флаги процессора. Их описание приведено в следующей таблице.

Флаг	Описание	Флаг	Описание
o	Флаг переполнения	z	Флаг нуля
d	Флаг направления	a	Флаг дополнительного переноса
i	Флаг разрешения прерывания	p	Флаг четности
s	Флаг знака	c	Флаг переноса

Замечание: Строчный неподсвеченный символ означает, что флаг не установлен и имеет значение "0". Выделенный заглавный символ показывает, что флаг имеет значение "1". Например, `o d I s Z a p c`.

Если текущая команда обращается к адресу в памяти, то в третьей строке показывается эффективный адрес и значение по этому адресу. Эффективный адрес и значение могут использоваться в выражениях с помощью встроенных функций Eaddr и Evalue. Подробности смотрите в разделе "*Встроенные функции*" на странице 105.

Изменение содержимого регистров и значений флагов

Окно регистров может быть использовано для изменения содержимого регистров или флагов. Для перемещения курсора в окно следует щелкнуть мышью в пределах окна или нажать клавиатурную комбинацию Alt-C. Изменение содержимого окна может быть выполнено с помощью следующих клавиатурных комбинаций.

Функция редактирования	**Комбинация клавиш**
Перевести курсор к началу поля следующего регистра	Tab или Shift-стрелка вправо
Перевести курсор к началу поля предыдущего регистра	Shift-Tab или Shift-стрелка влево
Закончить редактирование и сохранить сделанные изменения	Enter
Закончить редактирование, не сохраняя изменения в текущем регистре. (Однако, все предыдущие изменения сохраняются.)	Esc
Переключение значения флага, когда курсор находится в одном из них	Insert
Перемещение курсора в пределах окна регистров	Стрелки

Связанные с окном команды

Приведенные ниже команды имеют непосредственное отношение к окну регистров. Дополнительные подробности о них смотрите в "*Справочнике по командам SoftICE*".

Команда	**Функция**
CPU	Показывает значения всех регистров процессора
G [=*стартовый-адрес*] [*адрес-останова*]	Начать исполнение (с указанного адреса по адрес останова)
P	Выполнить один шаг программы
T [=*стартовый-адрес*] [*счетчик*]	Трассирование команд (начиная с указанного адреса)

Окно данных

Окно данных позволяет Вам видеть и изменять содержимое памяти. Вы можете задать до 4 окон данных, показывающих различные участки памяти в разных форматах отображения. Однако, на экране одновременно может быть открыто только одно такое окно.

Управление окном данных

Для управления окном используются следующие команды.

Команда	Описание
WD	Открыть/закрыть окно данных
WD [*количество-строк*]	Изменить размер окна данных
Alt-D	Переместить курсор в окно данных или из него
DATA	Циклическое переключение окон данных
D [*адрес*]	Указать адрес для отображения в окне данных
FORMAT (или Shift-F3)	Выбрать формат отображения данных в окне

Циклическое переключение окон данных

Циклическое переключение окон данных осуществляется командой DATA или щелчком мыши по верхней границе окна данных. Окна переключаются в порядке от 0 до 3.

Листание содержимого окна данных

Для листания содержимого окна данных используются стрелки скроллинга (при работе с мышью), либо переместите курсор в окно командой Alt-D и примените необходимую комбинацию клавиш.

Функция	Комбинация клавиш
Перейти к предыдущей странице содержимого окна данных	PageUp
Перейти к следующей странице содержимого окна	PageDown
Прокрутить окно на одну строку вверх	Стрелка вверх
Прокрутить окно на одну строку вниз	Стрелка вниз

Отображение информации

Строка над окном данных имеет 4 поля, перечисленные в следующей таблице (слева направо).

Поле	Описание
Строка	Если окну данных командой DEX было присвоено выражение, то в строке показывается выражение в текстовом виде. В противном случае здесь показывается смещение начала отображаемой в окне области памяти относительно ближайшего символического имени; это может быть один из следующих типов строк: • символическое имя с последующим шестнадцатеричным смещением относительного этого имени, например, `MySYMBOL+00010` • имя модуля Windows с последующим типом, если сегмент данных является частью кучи Windows, например, `mouse.moduleDB` • имя владельца сегмента данных, если он является частью виртуальной DOS-машины • имя модуля Windows, имя секции и шестнадцатеричное смещение от этого имени, например, `KERNEL32!.text+001F`
Тип формата данных	Данные отображаются в виде байтов, слов, двойных слов или коротких, длинных или 10-байтных вещественных значений
Тип сегмента	Имеет значение V86 или PROT. Индикатор V86 обозначает данные реального режима (сегмент:смещение), а PROT указывает на данные защищенного режима с адресацией селектор:смещение
Номер окна	Номер окна данных от 0 до 4. На экране SoftICE одновременно может отображаться только одно окно данных.

Каждая строка окна данных отображает 16 байт области памяти в текущем формате в виде байт, слов, двойных слов или коротких или длинных вещественных значений. Если текущий формат задает отображение 10-байтных вещественных значений, то каждая строка содержит 20 байт. Если текущий формат задает отображение данных в шестнадцатеричном виде (байт, слово или двойное слово), то в правой части окна показываются значения данных в символьном виде.

Изменение адреса отображаемой памяти и формата данных

Для изменения формата отображения данных необходимо щелкнуть по полю формата, указанному в верхней границе окна, либо использовать команду FORMAT (клавиатурная комбинация Shift-F3). В любом случае происходит циклическое переключение между форматами отображения: байт, слово, двойное слово, короткий вещественный, длинный вещественный или 10-байтный вещественный формат.

Для изменения адреса отображаемой области данных укажите необходимый адрес в команде D. Например, следующая команда выводит содержимое области памяти, начиная с адреса ES:1000h:

```
:D es:1000
```

Совет: Команда D может быть одновременно использована и для указания формата отображения. Подробна команда D описана в "*Справочнике по командам SoftICE*".

Изменение содержимого памяти

Для изменения содержимого памяти переместите курсор в окно данных и введите символьное или шестнадцатеричное значение. При редактировании в окне данных могут быть использованы следующие клавиши.

Функция редактирования	Комбинация клавиш
Перемещение между числовым и текстовым (ASCII) полями окна	Tab
Перевести курсор к началу предыдущего поля данных (предыдущий байт, слово, двойное слово в шестнадцатеричном числовом поле или к предыдущему символу в текстовом поле)	Shift-Tab
Закончить редактирование и сохранить сделанные изменения	Enter
Закончить редактирование, не сохраняя изменения в текущем поле данных. (Однако, все предыдущие изменения сохраняются.)	Esc

Совет: Для изменения содержимого памяти может быть использована команда E.

Присвоение выражения

Любому окну с помощью команды DEX может быть присвоено выражение. При активизации SoftICE это выражение вычисляется, и область данных, начинающаяся с полученного адреса, отображается в этом окне. Например, в следующем примере команда заставляет окно 0 всегда при активизации SoftICE показывать содержимое стека:

```
DEX 0 SS:ESP
```

Связанные с окном команды

Приведенные ниже команды имеют непосредственное отношение к окну данных. Дополнительные подробности о них смотрите в "*Справочнике по командам SoftICE*".

Команда Функция	
D [*размер*] [*адрес*]	Отображение содержимого памяти
DEX [*номер-окна* [*выражение*]]	Отображение или присвоение выражения окну данных
E [*размер*] [*адрес*] [*список-данных*]	Редактирование содержимого памяти
S [-cu] [*адрес* L *длина список-данных*]	Поиск данных в памяти

Окно стека FPU

Окно стека FPU (Float Point Unit — блок плавающей запятой или математический сопроцессор) показывает текущее состояние стека регистров FPU или MMX (MMX или MultiMedia Extention означает расширение системы команд процессоров Pentium или Pentium Pro).

Для открытия и закрытия окна используется команда WF.

Отображение информации

Если данные в окне FPU отображаются в виде вопросительных знаков (?), то это означает, что математический сопроцессор отключен или отсутствует в системе. В Windows NT FPU подключается к потоку после исполнения первой FPU-инструкции.

Архитектура процессоров фирмы Intel предполагает совместное (но не одновременное) использование регистров блоками FPU и MMX. Для отображения содержимого стеков используются следующие форматы.

Команда	Функция	
WF F	Плавающая запятая	Только для FPU
WF B	Упакованные байты	Только для MMX
WF W	Упакованные слова	Только для MMX
WF D	Упакованные двойные слова	Только для MMX

При отображении стека в виде вещественных чисел регистры обозначаются от ST0 до ST7. При отображении данных в виде упакованных байтов, слов или двойных слов регистры обозначаются MM0-MM7. (Подробности о команде WF смотрите в "*Справочнике по командам SoftICE*".)

Совет: Для отображения содержимого регистров, статуса и управляющего слова используйте команду WF -D (с дефисом!).

Ошибки являются частью нашей жизни.
Их последствия — вот с чем необходимо считаться.
Никки Джиованни

Mistakes are a fact of life. It is the response to error that counts.
Nikki Giovanni

6. Использование SoftICE

Отладка нескольких программ одновременно

Утилита Symbol Loader позволяет загружать в одном сеансе несколько таблиц с отладочной информацией. Следовательно, Вы можете отлаживать сложные комплексы программного обеспечения, состоящие из таких разных компонентов как, например, приложения, библиотеки и драйверы.

Для того, чтобы просмотреть список всех загруженных таблиц и выбрать необходимую в данный момент, применяется команда TABLE. Когда программа в процессе своего исполнения достигает установленной Вами контрольной точки, применение команды TABLE в сочетании с несколькими первыми символами имени таблицы позволяет сменить текущую таблицу на ту, которая соответствует исполняемой в данный момент программе.

Если же Вы не знаете, какая таблица является текущей, то команда TABLE без параметров покажет список всех загруженных отладочных таблиц, а текущая таблица будет выделена цветом.

Возможно переключение на таблицу, которая не соответствует исполняемой в данный момент программе. Это может понадобиться, например, для установки с их помощью контрольных точек в еще незагруженной программе.

Перехват ошибок

SoftICE позволяет перехватывать ошибки для следующих типов программных кодов:

- 32-битный код защищенного режима кольца 3 (программы Win32)
- Драйверы кольца 0 (драйверы устройств режима ядра)
- 16-битный код защищенного режима кольца 3 (16-битные Windows программы)

SoftICE не перехватывает ошибки окон DOS. Это касается как программ виртуальных машин V86, так и расширителей DOS.

В следующих разделах поддержка перехвата ошибок описана подробнее.

32-битный код защищенного режима кольца 3 (программы Win32)

SoftICE перехватывает все необрабатываемые исключения, которые в обычных ситуациях приводят к выводу сообщения об ошибке. SoftICE автоматически возвращается к инструкции, вызвавшей ошибку, выводит на экран свое рабочее окно и показывает эту инструкцию и примерно следующее сообщение:

```
Break due to Unhandled Exception NTSTATUS=STATUS_ACCESS_VIOLATION
(Прерывание вследствие необрабатываемого исключения ...).
```

Поле NTSTATUS содержит сообщение об ошибке, соответствующее коду статуса. (Полный список статусных кодов находится в заголовочном файле NTSTATUS.H в Windows NT DDK.)

Если исполнение программы после перехвата ее отладчиком продолжить, то SoftICE проигнорирует эту ошибку и предоставит операционной системе выполнить обычную процедуру обработки ошибок, например, вывести окно диалога с сообщением об ошибке приложения.

Драйверы кольца 0 (драйверы устройств режима ядра)

SoftICE обрабатывает все исключения 0 кольца, которые приводят к вызову KeBugCheckEx. KeBugCheckEx — это программа, которая выводит в Windows NT "синий экран смерти".

Если KeBugCheckEx вызвана в результате ошибки обращения к странице, общей ошибки защиты, ошибки стека или недопустимого кода команды, то SoftICE пытается повторно выполнить ошибочную инструкцию. Если снова будет выдано сообщение об ошибке на той же самой инструкции, то либо произойдет рестарт системы, либо будет сделана попытка обойти данную ситуацию, изменяя содержимое EIP или меняя условия функционирования системы.

Если KeBugCheckEx вызвана по другим причинам, то инструкция повторена быть не может. В этом случае SoftICE активизируется на входе в KeBugCheckEx с сообщением примерно следующего содержания:

```
Break Due to KeBugCheckEx (Unhandled kernel mode exception) Error=1E
(KMODE_EXCEPTION_NOT_HANDLED) P1=8000003 P2=804042B1 P3=0 P4=FFFFFFFF
(Прерывание в результате KeBugCheckEx (необрабатываемое исключение ядра ...))
```

Поле ошибки содержит шестнадцатеричный код с кратким описанием. Определения кодов ошибок содержится в заголовочном файле BUGCODES.H в NT DDK.

Поля P1-P4 содержат параметры, передаваемые в программу KeBugCheckEx. Эти поля не имеют стандартных значений.

Если Вы попытаетесь продолжить работу в этой точке, то Windows NT продемонстрирует "синий экран" и зависнет. Если же Вы хотите контролировать ситуацию и после "синего экрана", то включите режим I3HERE (SET I3HERE ON); после появления "синего экрана" Windows NT выполнит инструкцию INT 3.

16-битный код защищенного режима кольца 3 (16-битные Windows программы)

SoftICE обрабатывает перехват ошибок 16-битных программ несколько иным способом, чем 32-битных. Обычно при возникновении ошибки 16-битного приложения Windows NT выводит диалоговое окно с описанием ошибки и предоставляет Вам выбор — CANCEL или CLOSE.

Если Вы выберите CANCEL, то команда, вызвавшая ошибку, будет повторена, а Windows NT выдаст отладочное сообщение для перехвата ошибочной инструкции. SoftICE использует его для своей активизации и выводит эту инструкцию на экран. Иными словами, SoftICE появляется после того, как Вы получите сообщение об ошибке и "проигнорируете" ее (то есть нажмете кнопку CANCEL), но не перед этим.

Если же Вы нажмете кнопку CLOSE, то Windows NT не будет делать попытку повторного исполнения инструкции, а SoftICE не будет активизирован. Следовательно, если Вы хотите отлаживать исключения, всегда нажимайте кнопку CANCEL.

Некоторые ошибки приводят к демонстрации целой серии диалоговых окон. В этом случае первое окно содержит выбор CLOSE или IGNORE. IGNORE заставляет

Windows NT пропустить инструкцию, вызвавшую ошибку, и продолжить выполнение программы. Если же нажать CLOSE, то Windows NT покажет следующее диалоговое окно.

Контекст адресов

Как Windows 95, так и Windows NT выделяют каждому процессу собственное адресное пространство в диапазоне адресов от 0 до 2 Гб. Кроме того, Windows 95 резервирует первые 4 Мб каждой виртуальной машины (для размещения DOS и ее драйверов). Область памяти от 2 до 4 Гб разделяется всеми процессами.

Выделенное процессу виртуальное адресное пространство называется адресным контекстом (или контекстом процесса). SoftICE выводит имя текущего процесса в правой части строки статуса в нижней части экрана. Учтите, что текущий контекст не обязательно совпадает с контекстом Вашего приложения, особенно если Вы самостоятельно активизировали SoftICE. Если это так, то прежде чем просматривать или модифицировать какие-либо данные, или устанавливать контрольные точки в Вашем приложении, выполните команду ADDR, чтобы переключиться на контекст приложения.

SoftICE автоматически переключает адресный контекст в следующих случаях:

- Если Вы используете команду TABLE для переключения 32-битных таблиц, то SoftICE меняет текущий контекст адресов на контекст модуля указанной в команде таблицы.
- Если Вы выполняете команду FILE для просмотра файла с исходными текстами из 32-битной таблицы, SoftICE выполняет аналогичное переключение контекста.
- Если Вы используете в выражении символические имена, то SoftICE также выполняет необходимое переключение контекста. Эта ситуация включает в себя и экспортируемые символы, загруженные с помощью утилиты Symbol Loader.

Если Вы меняете адресный контекст, то при просмотре кодов или данных, расположенных в адресном пространстве приложения (линейные адреса от 0x400000 до 0x7FFFFFFF для Windows 95 и от 0 до 0x7FFFFFFF для Windows NT), могут возникать недоразумения. Отображаемые коды или данные меняются, хотя значения селектор:смещение остаются неизменными. Это нормально. Линейный адрес остается тем же самым, однако используемые теперь системные таблицы страниц указывают на физическую память иного адресного контекста.

SoftICE не позволяет использовать в выражениях адресный контекст. Если Вы используете в выражении непосредственные значения адресов, то проверьте, установлен ли необходимый Вам адресный контекст. Например, команда D 137:401000 покажет содержимое памяти для текущего адресного контекста.

Внимание: Прежде чем устанавливать контрольные точки с использованием непосредственных значений адресов, проверьте правильность контекста, так как SoftICE использует его для трансляции линейных адресов в физические.

Использование команд '.' (точка) INT 0x41

Windows 95 предоставляет набор расширений, которые позволяют виртуальным драйверам VxD или 32-битным библиотекам DLL взаимодействовать с отладчиком уровня ядра. (Подробности смотри в файле DEBUGSYS.INC Windows 95 DDK.) API команд с точкой '.' позволяет получать специфическую VxD отладочную информацию или выдавать необходимые команды через стандартный интерфейс отладчика уровня ядра операционной системы. Хотя первоначально эти команды разрабатывались для отладчика WDEB386 фирмы Microsoft, SoftICE также поддерживает

большой набор команд с точкой, и Вы можете использовать их для доступа к VMM и VxD, а также добавлять любые собственные команды с точкой для Ваших VxD.

Внимание: Реализация всех '.'-команд встроена в VMM или в соответствующие VxD-драйверы. Они не являются частью SoftICE, и поэтому отладчик не может гарантировать корректность их работы. Кроме того, эти команды не обеспечивают проверку ошибок, что может привести к краху системы при неверном задании параметров. И, наконец, SoftICE не может определить, находится ли операционная система в состоянии, необходимом для правильного выполнения данной команды с точкой, что также может привести к краху системы, если команда была выдана в неподходящий момент.

SoftICE поддерживает следующие '.'-команды Windows 95:

- Зарегистрированные расширения для команд с точкой.
  Для получения списка зарегистрированных расширений используется следующая команда
  .?
- Debug-Query расширения.
  Чтобы вызвать обработчик этих команд с точкой наберите после точки имя VxD. Большинство обработчиков (если они существуют для данного VxD), выводят на экран меню. Например, перечисленные ниже VxD имеют обработчики команд с точкой как в отладочной, так и в коммерческой версии Windows 95:
  .VMM
  .VPICD
  .VXDLDR
  Чтобы узнать, существует ли обработчик команд с точкой для данного VxD, просто выдайте команду. Для некоторых VxD в отладочной версии DDK обеспечивается поддержка большего количества команд, чем в коммерческой.
- Встроенные '.'-расширения для VMM.
  VMM предоставляет большое количество команд с точкой как в отладочной, так и в коммерческой версии. Для получения полного списка воспользуйтесь командой
  ..?
  В коммерческой версии Windows 95 команда "..?" выдает следующий список расширений для команд с точкой:

. (точка)-команда	**Описание**
.R[#]	Показывает регистры текущего потока
.VM[#]	Показывает полное состояние виртуальной машины
.VC[#]	Показывает блок управления (control block) текущей виртуальной машины
.VH[#]	Показывает VMM linked list и, если возможно, список обработчиков
.VR[#]	Показывает регистры текущей виртуальной машины
.VS[#]	Показывает стек текущей виртуальной машины
.VL	Показывает список всех обработчиков виртуальной машины
.DS	Выводит дамп стека защищенного режима с метками
.VMM	Меню для получения информации о состоянии VMM
.<*имя-устройства*>	Показывает информацию о состоянии заданного устройства

Переходы из 3 кольца защиты к 0 кольцу — общие сведения

Много раз во время трассировки программ под управлением Windows 95 Вам приходилось достигать команд INT 30h или ARPL. Обе они используются для выполнения перехода из 3 кольца защиты в кольцо 0. Если Вы хотите выполнить такой переход, но при этом сэкономить силы и время, не трассируя большие объемы кода VMM, то с помощью команды G (Go — перейти) Вы можете запустить исполнение VMM с остановкой на адресе, который Вы обнаружите в окне дизассемблирования.

Windows 95 применяет следующие методы для перехода из 3 кольца защиты к кольцу 0.

- В режиме V86 Windows 95 использует команду ARPL, которая вызывает ошибку исполнения недопустимой команды. Обработчик такой ошибки передает управление соответствующему VxD. В Windows 95 используется единственная команда ARPL, которая задается различными сочетаниями сегмент:смещение в V86, чтобы указать адреса различных VxD. Например, если используется ARPL по адресу 0xFFFF:0, то Windows 95 применяет комбинации адресов 0xFFFF:0, 0xFFFE:10, 0xFFFD:20, 0xFFFC:30 и так далее.

  Ниже показан пример вывода команды ARPL в окне дизассемблера:

```
FDD2:220D ARPL DI,BP ; #0028:C0078CC9 IFSMgr(01)+0511
```

- В защищенном режиме для перехода Windows 95 использует программное прерывание с номером 30h. Сегмент 3Bh не содержит никакого другого кода кроме инструкций INT 30h, передающих управлением соответствующим VxD. Ниже приведены результаты дизассемблирования кода по адресу 3B:31A.

```
003B:031A   INT30      ; #0028:C008D4F4     VPICD(01)+0A98
003B:031C   INT30      ; #0028:C007F120     IOS(01)+0648
003B:031E   INT30      ; #0028:C02C37FC     VMOUSE(03))00F0
003B:0320   INT30      ; #0028:C02C37FC     VMOUSE(03))00F0
003B:0322   INT30      ; #0028:C023B022     BIOSXLAT(05)=0022
003B:0324   INT30      ; #0028:C230F98      BIOSXLAT(04)=0008
003B:0326   INT30      ; #0028:C023127C     BIOSXLAT(04)=02EC
```

Вам известны мои методы. Примените их.
Сэр Артур Конан Дойл

You know my methods. Apply them.
Sir Arthur Conan Doyle

7. Использование прерываний

Введение

При отладке приложений в SoftICE Вы можете устанавливать прерывания (или контрольные точки) на исполняемые команды, на чтение и запись в области памяти, на прерывания, на операции чтения или записи в порты ввода/вывода. SoftICE присваивает каждой контрольной точке номер от 0 до 0FFh, который используется для установки, удаления, включения/выключения соответствующего прерывания или для задания условий его возникновения.

Все прерывания в SoftICE являются постоянными, что означает, что отладчик отслеживает и поддерживает их до тех пор, пока Вы их самостоятельно не удалите или не отключите с помощью команд BC или BD, соответственно. После своего удаления контрольная точка может быть восстановлена с помощью команды BH, которая показывает протокол установки контрольных точек.

Одновременно Вы может установить до 256 прерываний. Однако, число контрольных точек на обращение к памяти (BPM) и к портам ввода/вывода в сумме не может быть больше 4 из-за архитектурных особенностей процессоров x86.

Если доступна отладочная информация, то прерывание может быть установлено с использованием имен функций. В режиме исходных кодов или смешанном режиме можно устанавливать контрольные точки методом "укажи и установи" на любой строке исходного текста. Особую ценность в данном случае представляет то, что Вы можете установить прерывания в модуле еще до его фактической загрузки.

Типы контрольных точек, поддерживаемых SoftICE

SoftICE, основываясь на особенностях архитектуры процессоров x86, обеспечивает поддержку множества разнообразных типов прерываний:

- Прерывания исполняемых команд. SoftICE заменяет существующие инструкции командами INT 3. Этот тип прерываний устанавливается командой BPX.
- Прерывания на обращение к памяти. SoftICE использует отладочные регистры для прерывания работы, когда читается (или исполняется) определенный байт, слово или двойное слово в памяти, или по этому адресу производится запись. Для установки такого типа прерываний используется команда BPM.
- Контрольные точки на прерываниях. SoftICE перехватывает прерывания, модифицируя IDT (таблицы дескрипторов прерываний). Для таких контрольных точек используется команда BPINT.
- Прерывания ввода/вывода. SoftICE использует расширения отладочных регистров процессоров Pentium и Pentium Pro, чтобы отслеживать инструкции IN и OUT по заданному номеру порты. Этот тип прерываний устанавливается командой BPIO.

- Прерывания на сообщения Windows. SoftICE отслеживает поступление в окно определенного сообщения (или сообщений из заданного диапазона). Этот тип прерываний не является фундаментальным, скорее это полезная надстройка над другими типами прерываний. Для установки контроля на поступление сообщения Windows применяется команда BMSG.

Дополнительные возможности контрольных точек

Любое из вышеперечисленных прерываний может быть задано со следующими дополнительными параметрами:

- Условное выражение [`IF` *выражение*]. Для возникновения прерывания выражение должно иметь в результате ненулевое значение (TRUE, Истина). Подробности смотрите в разделе "*Условные прерывания*" на странице 91.
- Действие при прерывании [`DO "`*команда1*`;` *команда2*`; ...`"]. При возникновении прерывания будет автоматически выполнена указанная последовательность команд. В качестве команд могут быть использованы и пользовательские макрокоманды. Подробности смотрите в разделе "*Задание действия при прерывании*" на странице 90.

Замечание: Полная информация по каждому типу прерываний приведена в "*Справочнике по командам SoftICE*".

Прерывание исполняемых команд

Контрольные точки на исполняемых командах перехватывают управление процессором при выполнении, например, вызова функции или строки на исходном языке программирования. Это наиболее часто используемый тип прерывания. SoftICE замещает исходную команду на инструкцию INT 3 и получает управление при ее выполнении.

Контрольная точка на исполняемой команде в SoftICE может быть установлена двумя способами: с помощью мыши или командой BPX. В следующих разделах подробно описаны оба эти способа.

Установка прерывания с помощью мыши

Если у Вас в компьютере используется процессор Pentium и подключена мышь, то с ее помощью Вы может устанавливать (и удалять) постоянные и одноразовые контрольные точки. Для установки прерывания необходимо выполнить двойное нажатие на строке программе, где Вы хотите его установить. После установки прерывания SoftICE выделит эту строку цветом. Повторный двойной щелчок на той же строке удаляет данную контрольную точку. Для установки одноразовой контрольной точки, необходимо щелкнуть по командной строке, где Вы хотите ее установить, и выполнить команду SoftICE HERE (функциональная клавиша F7).

Использование команды BPX для установки прерывания

Для установки прерывания исполнения программы можно применять команду BPX со следующими параметрами:

`BPX [`*адрес*`] [IF` *выражение*`] [DO "`*команда1*`;` *команда2*`; ..."]`

`IF` *выражение*	Подробности смотрите в разделе "Условные прерывания" на странице 91.
`DO "`*команда1*`;`*команда2*`;..."`	Подробности смотрите в разделе "*Задание действия при прерывании*" на странице 90.

Пример: Для установки прерывания на функцию WinMain используется команда:
```
BPX WinMain
```

Без каких-либо параметров команда может быть использована для установки контрольной точки на текущей строке исходного кода. Для этого перейдите в окно кодов (комбинация клавиш Alt-C), с помощью стрелок переместите курсор в строку, где необходимо установить .прерывание, и выполните команду BPX (или нажмите функциональную клавишу F9). Если Вы предпочитаете использовать мышь, то пролистайте кодовое окно до необходимого фрагмента, а затем двойным щелчком установите контрольную точку в нужной строке.

Прерывания на обращение к памяти

Прерывания на обращение к памяти основаны на использовании отладочных регистров, впервые введенных в процессорах серии 386. Этот тип прерываний чрезвычайно полезен для выяснения вопроса, когда и где происходит изменение программных переменных, а также для установки контрольных точек в исполняемом коде, размещенном в памяти "только для чтения". Одновременно может быть установлено не более 4 таких прерываний, так как процессоры x86 содержать только 4 необходимых отладочных регистра.

Команда BPM имеет следующий синтаксис:
```
BPM[B|W|D] адрес [R|W|RW|X] [отладочный-регистр]
                 [IF выражение] [DO "команда1;команда2;..."]
```

BPM и *BPMB*	Прерывание по обращению к байту по указанному адресу.
BPMW	Прерывание по обращению к слову.
BPMD	Прерывание по обращению к двойному слову.
R, W и *RW*	Прерывание по чтению, записи или по чтению и записи.
X	Прерывание по исполнению указанного адреса. Это более мощная команда, чем BPX, так как в этом случае в памяти не делается каких-либо изменений, что позволяет устанавливать прерывание в памяти "только для чтения" или по адресам, которые еще недоступны.
отладочный-регистр	Указывает, какой регистр необходимо использовать для работы команды. Как правило, в этом нет необходимости, так как SoftICE самостоятельно находит свободный регистр.
IF *выражение*	Подробности смотрите в разделе "*Условные прерывания" на странице* 91.
DO "*команда1*;*команда2*;..."	Подробности смотрите в разделе "*Задание действия при прерывании*" на странице 90.

Пример: Ниже приведена команда установки прерывания по записи значения "5" в переменную размером двойное слово с именем MyGlobalVariable.
```
BPMD MyGlobalVariable W IF MyGlobalVariable==5
```

Имейте в виду, что если обращение к указанному адресу происходит достаточно часто, то производительность системы может значительно снизится, даже несмотря на то, что выражение может иметь результатом FALSE (Ложь).

Контрольные точки на прерываниях

Контрольные точки на прерывании используются для перехвата прерывания с использованием IDT. Контрольная точка срабатывает только в том случае, когда прерывание передается через IDT.

Для установки контрольной точки используется команда BPINT:

```
BPINT номер-прерывания [IF выражение] [DO "команда1;команда2;..."]
```

номер-прерывания	Число в диапазоне от 0 до 255 (от 0 до 0FFh).
IF *выражение*	Подробности смотрите в разделе "*Условные прерывания*" на странице 91.
DO "*команда1*;*команда2*;..."	Подробности смотрите в разделе "*Задание действия при прерывании*" на странице 90.

Если возникает программное прерывание, вызываемое инструкцией INT, то в окне кодов показывается эта команда. (SoftICE активизируется по достижению команды INT, на номер которой установлена контрольная точка, но до ее фактического исполнения.) В противном случае текущей командой будет первая инструкция обработчика прерывания. Список всех прерываний и их обработчики можно получить с помощью команды SoftICE IDT.

Пример: Следующая команда задает прерывание по обращению к программе ядра NtCreateProcess из пользовательского режима.

```
BPINT 2E IF EAX==1E
```

Замечание: Программа NtCreateProcess обычно вызывается из ZwCreateProcess в NTDLL.DLL, которая в свою очередь вызывается из CreateProcessW KERNEL32.DLL. В условном выражении значение 1Eh соответствует номеру сервиса NtCreateProcess. Для получения этого значения можно использовать команду NTCALL.

Команду BPINT можно использовать для перехвата программных прерываний, например, прерывания INT 21h, вызываемого программами Windows. Обратите внимание на то, что программные прерывания в режиме V86 не проходят через векторы IDT. В данном режиме инструкции INT вызывают генерацию общей ошибки защиты (GPF), которая передается через вектор 0Dh IDT. Обработчик GPF операционной системы определяет причину ошибки и передает управление обработчику, определенному для конкретного типа прерывания V86. Иногда обработка такого прерывания может продолжиться передачей управления обратно в виртуальную машина V86 с вызовом необходимого обработчика через таблицу векторов прерывания виртуального режима (Interrupt Vector Table — IVT). В некоторых случаях прерывание реального режима отображается (эмулируется) с помощью вектора прерываний реального режима.

В тех случаях, когда прерывание эмулируется, его можно перехватить, установив командой BPX контрольную точку на начало обработчика прерывания реального режима.

Пример: Для установки контрольной точки на обработчик прерывания реального режима INT 21h используется следующая команда:

```
BPX *($0:(21*4))
```

Прерывания на ввод/вывод

Прерывания на ввод/вывод позволяют отслеживать операции чтения или записи по заданному адресу порта. Прерывание перехватывает обращение к порту команд IN или OUT. SoftICE обрабатывает этот тип контрольных точек с использованием специального расширения отладочных регистров, введенного в процессоры Pentium. Следовательно, для установки такого прерывание необходимо наличие в компьютере

процессора класса Pentium или Pentium Pro. Одновременно на ввод/вывод может быть установлено до 4 контрольных точек. Прерывания могут быть обработаны как на уровне ядра (кольцо 0), так и в режиме пользователя (кольцо 3).

Замечание: SoftICE под управлением Windows 95 полагается на битовую карту разрешения ввода/вывода, которая ограничивает возможности перехвата операций ввода/вывода кольцом 3.

Использовать контрольные точки ввода/вывода для перехвата команд IN/OUT в программах MS-DOS невозможно. Эти команды перехватываются и, в последующем, эмулируются операционной системой и, следовательно, реальные обращения к портам ввода/вывода могут не полностью соответствовать исходным командам.

Команда BPIO имеет следующий синтаксис:

```
BPIO номер-порта [R|W|RW]
                 [IF выражение] [DO “команда1;команда2;…”]
```

R, *W* и *RW* — Прерывание по чтению из порта (команда IN), записи в порт (команда OUT) или по любой из них.

IF *выражение* — Подробности смотрите в разделе "*Условные прерывания*" на странице 91.

DO “*команда1*;*команда2*;…” — Подробности смотрите в разделе "*Задание действия при прерывании*" на странице 90.

Когда происходит прерывание по вводу/выводу, и активизируется SoftICE, текущей командой оказывается инструкция, следующая за командой IN или OUT, вызвавшей прерывание. В отличии от команды BPM в данном случае невозможно установить размер передаваемых данных; любое обращение к заданному порту приводит к генерации прерывания независимо от того, передается байт, слово или двойное слово. Кроме того, любые операции, перекрывающие указанный номер порта, также приводят к генерации прерывания. Например, если Вы установили контрольную точку на обращение к порту 2FFh, то передача слова через порт с номером 2FEh также приведет к активизации SoftICE.

Пример: Приведена установка прерывания на чтение из порта с номером 3FEh значения, у которого установлены два старших бита.

```
BPIO 3FE R IF (AL & C0)==C0
```

Проверка условия происходит после завершения операции ввода/вывода. Значение будет находиться в регистрах AL, AX или EAX, так как все операции ввода/вывода через порты за исключением строчных команд вводы/вывода (которые, однако, применяются редко) используют регистр EAX.

Прерывания на сообщения Windows

Контрольные точки на сообщениях Windows устанавливаются для перехвата определенного сообщения или сообщений из заданного диапазона, передаваемого процедуре окна. Хотя данный тип прерываний может быть установлен с использованием условной команды BPX, однако, задавать его командой BMGS значительно проще.

```
BMSG дескриптор-окна [L]
        [1-сообщение-диапазона [последнее-сообщение-диапазона]]
        [IF выражение] [DO “команда1;команда2;…”]
```

дескриптор-окна — Значение, возвращаемое при создании окна; для получения полного списка всех окон и их дескрипторов может быть использована команда HWND.

L	Параметр указывает, чтобы сообщение было просто отображено в окне команд без активизации SoftICE.
1-сообщение-диапазона	Единственное сообщение или нижнее значения диапазона сообщений Windows. Если диапазон сообщений не задается, то данный параметр указывает именно то сообщение, передачу которого необходимо зафиксировать. Как для первого, так и для последнего сообщения их значения могут быть заданы либо шестнадцатеричными числами, либо действительными именами сообщений, например, WM_QUIT.
последнее-сообщение	Старшее значение для диапазона сообщений.
IF *выражение*	Подробности смотрите в разделе "*Условные прерывания*" на странице 91.
DO "*команда1*;*команда2*;..."	Подробности смотрите в разделе "*Задание действия при прерывании*" на странице 90.

При задании сообщения (или диапазона сообщений) можно использовать их символические имена, например, WM_NCPAINT. С помощью команды WMSG (без параметров) можно получить список всех имен сообщений, который SoftICE понимает. Если сообщение (или диапазон сообщений) не заданы, то прерывания устанавливаются на все сообщений Windows.

Пример: Для установки прерывания на сообщения окна с дескриптором 1001E необходимо задать следующую команду.

```
BMSG 1001E WM_NCPAINT
```

SoftICE достаточно разумен, чтобы принимать во внимание адресный контекст процесса, который владеет данным окном, поэтому при использовании команды BMSG об этом можно не беспокоиться.

С помощью команды BPX и условных выражений можно сконструировать эквивалент команды BMSG. Используя команду HWND можно получить адрес процедуры окна, а затем командой BPX установить на нее контрольную точку (только для Win32).

```
BPX 5FEBDD12 IF (esp->8)==WM_NCPAINT
```

Внимание: Если при установке контрольной точки используются действительные адреса (а не символические имена), чрезвычайно важно указать правильный адресный контекст.

Понятие о контексте прерывания

Контекст прерывания состоит из адресного контекста, в котором была установлена данная контрольная точка, а также в каком модуле это прерывание срабатывает. Понятие контекста прерывания применимо к командам BPX и BPM, а также к контрольным точкам, установленным такими командами, как BMSG.

Для приложений Win32 контрольные точки, установленные в 2 верхних гигабайтах памяти, являются глобальными, и прерывания по ним происходят в любом контексте. Контрольные точки, установленные в 2 нижних гигабайтах памяти, являются контекстно-зависимыми, они срабатывают лишь при выполнении следующих условий:

- SoftICE активизируется лишь в том случае, если адресный контекст совпадает с контекстом, в котором данная контрольная точка была установлена.

- Если прерывание срабатывает в том же модуле, к котором оно было установлено, то SoftICE активизируется, игнорируя адресный контекст. Это означает, что контрольная точка, установленная в разделяемом модуле (например, KERNEL32.DLL), будет срабатывать при любом адресном контексте, в который данный модуль загружен, независимо от того, каким он был в момент ее установки.

  Исключения из этого правила возникают лишь тогда, когда другой процесс отображает данный модуль по иному базовому адресу, чем он был в момент установки контрольной точки. В этом случае прерывания не возникнет. Чтобы избежать этого, базируйте Ваши модули по стандартным "не конфликтным" адресам.

Прерывания, установленные в MS-DOS и 16-битных Windows программах, так же являются контекстно-зависимыми, что означает срабатывание прерывания лишь в том процессе NTVDM, в котором оно было установлено. Исключений не бывает даже для тех случаев, когда одна и та же программа запускается несколько раз.

Контекст прерывания более важен для контрольных точек, установленных командами BPM, чем BPX, так как они являются аппаратными прерывания по определенному виртуальному адресу. Механизм аппаратных прерываний ничего не знает об адресном пространстве, и возможны ситуации, когда он активизируется в несоответствующем участке кода или данных. Именно контекст прерываний позволяет отладчику SoftICE различать фальшивые и действительные срабатывания контрольных точек.

Виртуальные контрольные точки

Используя SoftICE, Вы можете устанавливать контрольные точки в модулях еще до их загрузки в память, так же как нет необходимости, чтобы при установке BPX-прерывания (инструкция INT 3) страница была загружена в физическую память. Такие контрольные точки называются виртуальными, фактически они устанавливаются автоматически в момент загрузки модуля или, когда страница окажется физически доступной. Виртуальные прерывания могут быть установлены лишь с помощью символических имен или по строкам в исходных кодах.

Задание действий при прерывании

Вы можете заставить SoftICE выполнить некоторую последовательность действий при срабатывании контрольной точки. Эта последовательность задается параметром "DO", который существует у всех типов прерываний:

`DO "`*`команда1`*`;`*`команда2`*`;...”`

Тело параметра представляет собой список команд SoftICE или макрокоманд, разделенных точкой с запятой. Последнюю команду завершать точкой с запятой необязательно.

Действие при прерывании тесно связано с понятием макрокоманды. Более подробную информацию о макрокомандах можно найти в разделе "*Работа с постоянными макрокомандами*" на странице 125. Фактически сама последовательность действий при прерывании является неименованной макрокомандой, которая не может получать аргументы командной строки. Подобно макрокомандам, сама эта последовательность может содержать и другие макрокоманды, ведь изначально они разрабатывались именно для задания сложной последовательности команд при возникновении прерывания.

Если в тело макрокоманды (или в последовательность действий при прерывании) необходимо вставить символ двойной кавычки (") или знак процента (%), то их

необходимо предварить обратной косой чертой (\). Сам же символ обратной косой черты задается последовательностью двух таких знаков (\\).

Если производится протоколирование контрольных точек (подробнее смотри описание встроенной функции BPLOG на странице 94), то заданная последовательность действий не выполняется.

Ниже приведены примеры задания наиболее употребительных действий:

```
BPX EIP DO “dd eax”
BPX EIP DO “data 1;dd eax”
BPMB dataaddr if (byte(*dataaddr)==1) do “? IRQL”
```

Условные прерывания

Условные прерывания представляют собой быстрый и удобный способ выявления особых условий или состояния операционный системы или отлаживаемого приложения. При задании условного выражения в контрольной точке SoftICE активизируется только тогда, когда это выражение имеет ненулевой результат (TRUE, Истина). Так как вычислитель SoftICE позволяет обрабатывать любые сложные выражения, Вы легко можете выявить требуемую Вам ситуацию.

Все команды прерывания SoftICE допускают использование условных выражений со следующим синтаксисом:

команда-прерывания [*параметры-прерывания*]
[IF *выражение*] [DO “*команды*”]

Ключевое слово "IF" (Если) сопровождается выражением, которое должно быть вычислено при достижении данной контрольной точки. Контрольная точка будет проигнорирована, если результат этого выражения окажется равен нулю (FALSE или Ложь). Когда же результатом выражения является TRUE (ненулевой результат), SoftICE активизируется и показывает причину возникновения прерывания, в данном случае команду, включающую в себя и условное выражение.

Приведенные ниже примеры демонстрируют условные выражения, используемые при отладке.

Замечание: Большинство примеров содержат системные переменные, значения которых могут различаться в зависимости от версии операционной системы.

- Отслеживание активизации потока:
  `bpx ntoskrnl!SwapContext IF (edi==0xFF8B4020)`
- Отслеживание выключение потока:
  `bpx ntoskrnl!SwapContext IF (esi==0xFF8B4020)`
- Отслеживание создания объекта CSRSS "HWND" (1 тип):
  `bpx winsrv!HMAllocObject IF (esp->c == 1)`
- Отслеживание уничтожения информационного объекта потока (6 тип CSRSS):
  `bpx winsrv!HMFreeObject+0x25 IF (byte(esi->8) == 6)`
- Отслеживание создание таблицы дескрипторов объектов процесса:
  `bpx ntoskrnl!ExAllocatePoolWithTag IF (esp->c == ‘Obtb’)`
- Отслеживание завершения потока (enum == 4):
  `bpmb _thread->29 IF (byte(_thread->29) == 4)`
- Отслеживание освобождения блока кучи (230CD8):
  `bpx ntddl!RtlFreeHeap IF (esp->c == 230CD8)`
- Отслеживание ситуации, когда процесс делает системный вызов:
  `bpint 2E if (process == _process)`

Во многих примерах используются функции потоков и процессов, встроенные в SoftICE. Эти функции предоставляют ссылки на активный поток или процесс операционной системы. В некоторых примерах имя функции предваряется символом подчеркивания "_". Этот специальный знак позволяет ссылаться на динамические переменные, такие как содержимое регистров или текущий поток или процесс как на константу. Следующие примеры должны помочь Вам уяснить эту возможность:

- Показанное ниже условное прерывание срабатывает, когда динамическое (в момент исполнения программы) содержимое регистра EAX равно его значению на момент установки прерывания.
  ```
  bpx eip IF (eax == _eax)
  ```
  Эта команда соответствует последовательности:
  ```
  ? EAX
  00010022
  bpx eip IF (eax == 10022)
  ```
- Условное прерывание из следующего примера сработает, когда идентификатор потока, исполняющегося в этот момент, совпадет с идентификатором текущего потока на момент установки прерывания.
  ```
  bpx eip IF (tid == _tid)
  ```
  Эта команда соответствует последовательности:
  ```
  ? tid
  8
  bpx eip IF (tid == 8)
  ```

Если Вы в выражении предварите имя какой-либо функции или регистра символом подчеркивания ("_"), то значение этой функции будет вычислено и останется постоянным на протяжении всего времени существования данного выражения.

Функции статистики условных прерываний

SoftICE предоставляет возможность наблюдать и управлять контрольными точками на основании того, сколько раз то или иное прерывание срабатывало или, наоборот, было пропущено. Для этого в условных выражениях используются следующие встроенные функции:

- BPCOUNT
- BPMISS
- BPTOTAL
- BPLOG
- BPINDEX

BPCOUNT

Функция BPCOUNT возвращает количество случаев, когда выражение в данном условном прерывании было равно TRUE. Каждый раз при попадании в контрольную точку вычисляется заданное в ней условное выражение (если оно существует), и, если его результатом будет TRUE, счетчик исполнений (BPCOUNT) увеличивается на 1. Если же условное выражение равно FALSE, то на 1 увеличивается счетчик "промахов" (BPMISS).

Пример: На пятое исполнение прерывания счетчик BPCOUNT окажется равным 5, поэтому условное выражение равно TRUE, и будет активизирован SoftICE.
```
bpx myaddr IF (bpcount==5)
```

Функцию BPCOUNT следует использовать только в крайней правой части составных условных выражений, чтобы правильно происходило изменение счетчика:

```
bpx myaddr if (eax==1) && (bpcount==5)
```

В соответствии с алгоритмом работы вычислителя SoftICE, если EAX не равно 1, то подвыражение 'BPCOUNT==5' вычисляться не будет. (Аналогичным образом вычисляются логические выражения в языке C.) А к тому моменту, когда потребуется определить результат 'BPCOUNT==5', вычисленная часть выражения окажется равной TRUE, счетчик BPCOUNT будет увеличен и, если окажется, что он стал равным 5, значение всего выражения станет TRUE, и произойдет активизировизация SoftICE. Если же BPCOUNT окажется неравным 5, общий результат будет FALSE, увеличится счетчик BPMISS, а SoftICE активизирован не будет.

Как только общий результат всего выражение оказывается равен TRUE, происходит активизация SoftICE, а оба счетчика (и BPCOUNT, и BPMISS) сбрасываются в 0.

Замечание: Не используйте проверку счетчика BPCOUNT в начале условного выражения, в противном случае изменения счетчика будут производиться неправильно.

```
bpx myaddr if (bpcount==5) && (eax==1)
```

BPMISS

Значением функции BPMISS является количество попаданий на контрольную точку, когда результат ее условного выражения оказывался равным FALSE.

Исполнение данной функции подобно функции BPCOUNT. Она используется для активизации SoftICE в тех случаях, когда в течение нескольких раз результатом условного выражения оказывается FALSE. При этом значение BPMISS всегда будет на 1 меньше, чем Вы ожидаете, так как оно не изменяется до окончания вычисления всего выражения. Чтобы скорректировать данную "задержку" обновления, следует применять оператор ">=" (больше или равно).

Пример: `bpx myaddr if (eax==43) || (bpmiss>=5)`

Вследствие алгоритма работы вычислителя SoftICE, если результат подвыражения 'EAX==43' равен TRUE, то и все условное выражение равно TRUE, и SoftICE будет активизирован. В противном случае счетчик BPMISS обновляется всякий раз, когда результат равен FALSE. После серии из 5 безуспешных попаданий в контрольную точку подряд, выражение станет равным TRUE, и появится окно SoftICE.

BPTOTAL

Значением функции BPTOTAL является общее количество выполнения данного прерывания.

Эта функция используется для определения состояния, в котором SoftICE будет активизирован. Значением функции является общее количество попаданий в данную контрольную точку за все время ее существования (смотри поле "Hits" (Попадание) в результатах, выводимых командой BSTAT). Этот счетчик никогда не сбрасывается.

Пример: В первые 50 исполнений контрольной точки, показанной ниже, результатом условного выражения будет FALSE, и SoftICE не активизируется. Во все последующие попадания будет происходить активизация отладчика, так как результат окажется равен TRUE.

```
bpx myaddr if (bptotal > 50)
```

Используя функцию BPTOTAL, можно составить выражение, соответствующее действию функции BPCOUNT. Для этого используется деление по модулю:

```
if (!(bptotal%COUNT))
```

Константа COUNT определяет частоту, с которой будет происходить срабатывание данного прерывания. Так, если COUNT, например, равен 4, то SoftICE будет активизироваться на каждое 4 попадание в данную контрольную точку.

BPLOG

Функция BPLOG используется для записи события прерывания в протокол SoftICE. Сам отладчик при этом не активизируется.

Замечание: Так как действия, заданные в команде контрольной точки, выполняются только при активизации SoftICE, то использовать команду "DO" совместно с функцией BPLOG смысла не имеет.

Значением функции BPLOG всегда является TRUE. Это заставляет SoftICE сделать запись в о данном прерывании в протоколе работы.

Пример: Каждый раз, когда происходит попадание в данную контрольную точку, и значение регистра EAX при этом равно 1, отладчик делает в протоколе запись об этом событии. Окно SoftICE при этом на экране монитора не появляется.

```
bpx myaddr if ((eax==1) && bplog)
```

BPINDEX

Функция BPINDEX используется для получения индекса контрольной точки для передачи его в параметр "DO".

Эта функция возвращает индекс контрольной точки, вызвавшей активизацию SoftICE. Это тот же самый индекс, который используется командами BL, BC, BD, BE, BPE, BPT и BSTAT. Вы можете использовать его в качестве параметра при задании этих функций в "DO"-последовательности.

Пример: Следующий пример демонстрирует использование команды BSTAT для выдачи статистики о контрольной точке:

```
bpx myaddr do “bstat bpindex”
```

Другой пример показывает контрольную точку, которая создает другую контрольную точку:

```
bpx myaddr
     do “t;bpx @esp if(tid==_tid) do \“bc bpindex\”;g”
```

Замечание: Функция BPINDEX предполагает, что она будет использована в задании последовательности действий, и вызывает ошибку, когда ее используют в условном выражении. Тем не менее, ее использование вне последовательности действий допустимо, однако, ее значение в данном случае не определено, и на него не следует полагаться.

Использование локальных переменных в условных выражениях

SoftICE разрешает Вам использовать имена локальных переменных в условных выражениях исполняемых контрольных точек (BPX или BPM X). В других типах контрольных точек (таких как BPIO или BPMD RW) локальные символы не распознаются, так они требуют определенной области действия. Эти типы прерываний не привязаны к определенному участку исполняемого кода, поэтому локальные переменных в них не имеют смысла.

Использование локальных переменных возможно, когда функция имеет пролог, где они создаются, и эпилог, где происходит их уничтожение. Локальные переменные доступны с момента завершения кода пролога и до начала исполнения кода эпилога. Использование символьных имен параметров функции во время исполнения пролога и эпилога также невозможно, так как в эти моменты происходит корректировка кадра стека.

Чтобы избежать этих ситуаций, устанавливайте прерывания на первую, либо на последнюю строки исходного текста тела функции. В следующем примере в качестве примера приводится функция "foobar".

Foobar Function

```
 1:DWORD foobar ( DWORD foo )
 2:{
 3: DWORD fooTmp=0;
 4:
 5: if(foo)
 6: {
 7: fooTmp=foo*2;
 8: }else{
 9: fooTmp=1;
10: }
11:
12: return fooTmp;
13:}
```

1 и 2 строки лежат вне тела функции. В этих строках происходит исполнение пролога. Если Вы будете использовать в этих точках имя локальной переменной, то получите сообщение об ошибке:

```
:BPX foobar if (foo==1)
error: Undefined Symbol (foo)
```

Ставьте контрольную точку на строке 3, где декларируется и инициируется локальная переменная "fooTmp":

```
:BPX .3 if (foo==0)
```

Строка 13 исходного текста содержит закрывающую скобку тела функции. Здесь также начинается исполнение кода эпилога, следовательно в данной точке и локальные переменные, и параметры находятся вне области видимости. Чтобы установить условное прерывание на окончание функции "foobar", используйте строку 12:

```
:BPX .12 if (fooTmp==1)
```

Замечание: Хотя использование локальных переменных в качестве параметра команды прерывания BPMD RW и не запрещено, этого следует избегать. Локальные переменные указываются относительно стека, и их абсолютные адреса меняются во время каждого вызова функции. После окончания первого исполнения функции, когда контрольная точка была установлена, адрес, к которому она оказалась привязана, больше не будет ссылаться на требуемую локальную переменную.

Использование стека в условных выражениях

Если Вы отлаживаете собственное приложение с полной отладочной информацией, то Вы можете обращаться к параметрам функций и локальным переменным через их символьные имена, как это описано в разделе "*Использование локальных переменных в условных выражениях*" на странице 94. Однако, если Вы проводите отладку без отладочной информации, например, если Вы транслировали модуль

только с общими (public) символами, или хотите изучать работу функции операционной системы, то Вам придется ссылаться на параметры и локальные переменные через указатели стека.

В данном разделе рассматривается адресация в 32-битной непрерывной (flat) модели памяти.

Параметры функции передаются через стек, следовательно, для обращения к ним Вам необходимы регистры ESP или EBP. Что именно Вы будете использовать, зависит от пролога функции, и того, в какой точке функции (относительно пролога) Вы собираетесь устанавливать прерывание.

Большинство 32-битных функций имеют следующий пролог:

```
PUSH EBP
MOV  EBP, ESP
SUB  ESP, size (размер-области-локальных-переменных)
```

Данный пролог размещает в стеке информацию следующим образом:

Вершина стека

	Параметр #n	ESP + (n*4) или EBP + (n*4) + 4	Устанавливаются вызывающей
	Параметр #2	ESP + 0x8 или EBP + 0xC + 4	программой
	Параметр #1	ESP + 4 или EBP + 8	
	Адрес возврата	<= ESP на входе в функцию	Пролог вызова
текущее значение EBP =>	Старое значение EBP	PUSH EBP; MOV EBP,ESP <= EBP - кадр стека	
	Локальные переменные + размер-1 Локальные переменные	<= ESP после пролога (SUB ESP, size (локальные переменные))	
текущее	Сохранение EBX	Сохранение регистров	Регистры,
значение	Сохранение ESI	компилятором "C"	сохраняемые
ESP =>	Сохранение EDI	<= ESP после сохранения регистров	компилятором

Дно стека

Для обращения к параметрам функции используйте регистры ESP или EBP. Учтите, что использование регистра EBP до выполнения команд пролога "PUSH EBP; MOV EBP,ESP" недопустимо. Кроме того, в случае создания области локальных переменных положение параметров функции относительно значения в регистре ESP потребует корректировки на величину этой области, а также на область, занятую сохраненными значениями регистров (если такое сохранение имело место).

Как правило, контрольные точки устанавливаются на адрес функции, например:

```
BPX IsWindow
```

В момент срабатывания такого прерывания, пролог еще не исполнен, и к параметром нужно обращаться с помощью регистра ESP. Регистр EBP в данной точке еще не установлен в должное значение.

Чтобы быть уверенным, что Вы используете ссылки на значения в стеке должным образом, следуйте следующим правилам.

Замечание: Приведенное ниже предполагает, что аргументы загружаются в стек справа-налево.

- Если Вы устанавливаете контрольную точку на точный адрес функции, например, BPX IsWindow, то для обращения к аргументам следует использовать ESP + (парам # * 4), где "парам #" — порядковый номер аргумента (от 1 до n).
- Если Вы устанавливаете контрольную точку в теле функции (после завершения пролога), то для обращения к аргументам используйте EBP + (парам # * 4) + 4, где "парам #" — порядковый номер аргумента (от 1 до n). Убедитесь, однако, что программа не использует регистр EBP для каких-либо иных целей, кроме адресации кадра стека.
- Функции, запрограммированные на ассемблере, или оптимизированные для исключения указателя кадра стека, могут потребовать использования регистра ESP, так как регистр EBP может быть некорректно установлен, или использован для других целей.

Замечание: Как только в стеке выделено пространство для локальных переменных, к ним можно адресоваться с помощью отрицательного смещения относительно EBP. Первая локальная переменная размещается по адресу EBP-4. Как правило данные имеют размер Dword, следовательно их смещение может быть легко вычислено по аналогии с аргументами. Например, 2 локальные переменные-указателя будут размещены в стеке по адресам EBP-4 и EBP-8.

Производительность

С условными прерываниями связан вопрос, касающийся скорости исполнения приложения. В большинстве случаев Вы не увидите никакого влияния от использования условных прерываний, или это влияние будет очень мало. Однако, если контрольная точка установлена на области данных, к которой происходит частое обращение, или в часто исполняемом участке кодов, то возможно некоторое снижение производительности системы. Это происходит вследствие того, что при попадании в контрольную точку будет производится вычисление условного выражения. Если же фрагмент кода выполняется сотни раз в секунду (как, например, функция ExAllocatePool или SwapContext), то любая контрольная точка с условным выражением или без него при такой частоте вызовов будет приводить к значительному снижению производительности.

Дублирование контрольных точек

После того, как Вы установите прерывание на какой-либо адрес, Вы не сможете поставить другую контрольную точку на тот же адрес. Однако, используя логические операторы (&& или || — "И" и "ИЛИ", соответственно), можно составить сложные условные выражения для проверки самых разных условий.

Затраченное время

SoftICE поддерживает использование команды чтения счетчика тактов (машинная инструкция RDTSC) на процессорах типа Pentium и Pentium Pro. В момент старта SoftICE показывает тактовую частоту компьютера, на котором он запущен. Каждый раз при появлении экрана SoftICE отладчик сообщает, сколько времени прошло с момента его последней активизации. Это время показывается в строке после причины прерывания и измеряется в секундах, мили- или микросекундах.

```
Break due to G (ET=23.99 microseconds)
Прерывание в результате команды "G" (затраченное время=23.99 мкс)
```

Счетчик тактов в процессорах Pentium чрезвычайно точен, однако, Вам следует учитывать 2 следующих обстоятельства.

1 Существует некоторая ошибка, связанная с активизацией SoftICE и последующим погружением его в "спячку". На компьютерах с тактовой частотой 100 МГц это занимает около 5 мкс. Эта величина может немного отличаться вследствие кеширования и изменения уровня привилегий.

2 Если перед активизацией SoftICE произойдет аппаратное прерывание, то время обработки этого прерывания будет учтено в показателе прошедшего времени. В момент активизации SoftICE все прерывания маскируются, поэтому аппаратные прерывания практически всегда обрабатываются сразу же после возобновления работы операционной системы.

Статистика прерываний

SoftICE собирает статистическую информацию о каждой контрольной точке, которая включает в себя:

- Общее число попаданий в контрольную точку, количество активизаций отладчика и пропусков контрольной точки, а также количество ошибок при вычислении условного выражения.
- Текущие значения счетчиков BPCOUNT и BPMISS.

Для получения этой информации используется команда BSTAT. Полная информация о команде BSTAT дана в "*Справочнике по командам SoftICE*".

Ссылка на контрольную точку в выражениях

Для ссылки на контрольную точку в выражениях в качестве ее имени используется комбинация префикса "BP" и ее индекса. Это допустимо для всех команд BPX и BPM. В этом случае в качестве значения SoftICE подставляет адрес данной контрольной точки.

Пример: Для дизассемблирования кода по адресу контрольной точки с индексом "0" может быть использована следующая команда:

```
U BP0
```

Управление контрольными точками

SoftICE предоставляет набор команд для манипулирования контрольными точками: выдача полного их списка, удаление, включение и выключение, а также просмотр протокола работы. Управление контрольными точками осуществляется по их номерам (индексам), которые представляют собой шестнадцатеричные значения от 0 до 0FFh. Индекс присваивается контрольной точке по мере их установки. В следующей таблице представлены команды управления контрольными точками.

Команда	Описание
BD	Выключить контрольную точку (без удаления)
BE	Включить контрольную точку
BL	Выдать список всех установленных контрольных точек
BPE	Редактировать контрольную точку
BPT	Использовать контрольную точку в качестве образца
BC	Удалить контрольную точку
BH	Показать протокола контрольных точек

Замечание: Полная информация о каждой команде дана в "*Справочнике по командам SoftICE*".

Использование встроенных контрольных точек

Иногда бывает полезно встраивать команды прерываний в программный код, а не устанавливать их с помощью SoftICE. Для работы с такими командами выполните следующее:

1 Поместите в необходимых участках программы команды INT 1 или INT 3.

2 Для активизации SoftICE на таких командах выполните:

- SET I1HERE ON для инструкций INT 1
- SET I3HERE ON для инструкций INT 3

Для меня уже давно стало аксиомой,
что мелочи имеют первостепенное значение.
Сэр Артур Конан Дойл

It has long been an axiom of mine that the little things are
infinitely the most important.
Sir Arthur Conan Doyle

8. Использование выражений

Выражения

SoftICE позволяет вместо постоянных значений использовать в командах и условных точках прерывания выражения. При этом обеспечиваются приоритет операций, поддержка стандартных для языка C арифметических, побитовых, логических операций и операций косвенной адресации, предопределенные макрокоманды для преобразования типов данных, а также доступ как к глобальным переменным собственно SoftICE, так и к глобальным переменным операционной системы.

SoftICE производит синтаксический анализ выражений и выполняет вычисления способом, сходным с работой трансляторов языков C и C++. Поэтому, если Вы уверенно чувствуете себя с любым из этих языков, то, следовательно, Вам уже знакома грамматика и синтаксис выражений SoftICE.

За исключением максимальной длины командной строки SoftICE (80 символов) не существует никаких иных ограничений на сложность выражений. Вы можете совмещать множество операторов, операндов и подвыражений, для составления выражений условных контрольных точек или иных различных вычислений.

Пример: Ниже показано составное выражение для выполнения прерывания в случае, когда первым параметром (ESP + 4), передаваемым функции IsWindow( ) является HWND со значениями 0x10022 или 0x1001E. Если любое из логических подвыражений имеет значение 'TRUE', то результатом всего условного выражения является 'TRUE', и прерывание выполняется:

```
BPX IsWindow if (esp->4 == 10022) || (esp->4 == 1001E)
```

Замечание: Выражение `esp->4` является сокращением для `*(esp+4)`

Операторы

SoftICE поддерживает следующие типы операторов:

Операторы адресации[†]	Примеры
->	ebp->8 (двойное слово, адресованное ebp + 8)
.	eax.1C (двойное слово, адресованное eax + 1c)
*	*eax (значение двойного слова по адресу eax)
@	@eax (значение двойного слова по адресу eax)

[†] Нагляднее было бы: ebp->8 результат — Dword
*(ebp->8) — Dword по адресу (ebp->8)

Математические операторы	**Примеры**
унарный +	+42 (десятичное)
унарный –	–42 (десятичное)
+	eax + 1
–	ebp – 4
*	ebx * 4
/	Symbol / 2
% деление по модулю	eax % 3
<< логический сдвиг влево	bl << 1 (результат: bl, сдвинутый влево на 1 бит)
>> логический сдвиг влево	eax >> 2 (результат: eax, сдвинутый вправо на 2 бита)

Побитовые операторы	**Примеры**
& побитовое AND	eax & F7
| побитовое OR	Symbol | 4
^ побитовое XOR	ebx ^ 0xFF
~ побитовое NOT	~dx

Логические операторы	**Примеры**
! логическое NOT	!eax
&& логическое AND	eax && ebx
|| логическое OR	eax || ebx
== равно	Symbol == 4
!= не равно	Symbol != al
<	ax < 7
>	bx > cx
<=	ebx <= Symbol
>=	Symbol >= Symbol

Специальные операторы	**Примеры**
. номер строки	.123 (значением является адрес 123 строки текущего файла с исходным текстом)
(,) группирующие скобки	(eax+3) * 4
, список аргументов	function(eax, ebx)
: оператор сегмента	es : ebx
имя-функции	word(Symbol)
# селектор защищенного режима	#es:ebx (адрес типа селектор:смещение в защищенном режиме)
$ сегмент реального режима	$es:di (адрес типа сегмент:смещение в реальном режиме)

Приоритет операторов

Приоритет операторов при вычислении выражений в SoftICE соответствует правилам языка C, дополненным несколькими специфическими для SoftICE операторами. Приоритет играет при вычислениях ключевую роль, поэтому то, как операторы располагаются в выражении, может оказать существенное влияние на его конечный результат. Для того, чтобы изменить последовательность выполнения операторов, необходимо использовать круглые скобки.

Пример: В предыдущих версиях SoftICE оператор сложения (+) и оператор умножения (*) имели одинаковые приоритеты, поэтому результатом выражения '3+4*5' было число 35. В новых версий оператору умножения присвоен более высокий приоритет, поэтому результатом выражения '3+4*5', эквивалентного по обычным правилам арифметики '3+(4*5)', является число 23. Для получения прежнего результата (35) необходимо воспользоваться скобками, чтобы заставить вычислитель сначала выполнить оператор сложения: '(3+4)*5'.

В следующей таблице перечислены все операторы в порядке уменьшения приоритета. Операторы, имеющие одинаковые приоритет, выполняются в соответствии правилами порядка.

Оператор	Порядок	Комментарий
(,), имя-функции		скобки, функции
->, .	слева - направо	адресация
:	слева - направо	селектор : смещение
#, $	справа - налево	переопределение селектора (сегмента)
*, @,	справа - налево	адресация
унарный +		по умолчанию - десятичное основание
унарный –		по умолчанию - десятичное основание
!,~		
.		номер строки
*, /, %	слева - направо	
+, -	слева - направо	
<<, >>	слева - направо	
<, <=, >, >=	слева - направо	
==, !=	слева - направо	
&	слева - направо	
^	слева - направо	
|	слева - направо	
&&	слева - направо	
||	слева - направо	
, (запятая)	слева - направо	список аргументов

Составные части выражений

SoftICE использует разнообразные операнды, такие, например, как идентификаторы и числа, которые можно свободно комбинировать с любыми операторами SoftICE. При этом акцент делается на обеспечении гибкости составления выражений, чтобы сделать его более естественным.

Совет: Результат любого выражения может быть получен с помощью команды '?' (вопросительный знак).

Числовые константы

SoftICE допускает использование следующих видов численных констант:

Ввод	Описание
Шестнадцатеричные	Шестнадцатеричная система является для SoftICE основной системой счисления при вводе и выводе значений. Допустимым набором символов для шестнадцатеричных чисел являются [0-9, A-F]. Шестнадцатеричному числу может предшествовать стандартный для языка C префикс '0x'. Допустимыми примерами шестнадцатеричных чисел являются: `FF, ABC, 0x123, 0xFFFF0000` Символьная форма ввода шестнадцатеричных чисел может, однако, конфликтовать с идентификаторами. Например, ABC. В этом случае использование префикса '0x' может уберечь от ошибочной интерпретации чисел в качестве символьных имен.

Десятичные	Для временного изменения основания системы счисления с шестнадцатеричной на десятичную SoftICE использует унарные операторы '+' и '-'. Это основано на том факте, что шестнадцатеричные числа со знаком, например, +FF или -ABC выглядят достаточно неестественно, хотя и являются математически допустимыми (равняясь, соответственно, 255 и -2748). Если Вы перед числом укажите унарные '+' или '-', то SoftICE попытается вычислить его как десятичное, и лишь затем, если это не удастся, как шестнадцатеричное.
	Следующие примеры демонстрируют, как использование унарных операторов '+' и '-' влияет на интерпретацию основания системы счисления:
	• `? +42`
	`0000002A 0000000042 "*"`
	• `? -42`
	`FFFFFFD6 4294967254 (-42) "яяяц"`
	• `? -1a`
	`FFFFFFE6 4294967270 (-26) "яяяж"`
	• `? +ff`
	`000000FF 0000000255 "я"`
	• `? +(12)`
	`00000012 0000000018 "."`
	Замечание: Оператор номера строки SoftICE (.) также изменяет основание счисления на десятичное. Оператор 'унарный +', за исключением изменения системы счисления, иного влияния на вычисление выражения не оказывает.

Символьные константы

SoftICE поддерживает использование стандартных для языка C символьные констант, таких как '\b', 'ABCD', или '\x23'. Основанием системы счисления для символьных констант, начинающихся с обратной косой черты '\', является десятичная система. Для того, чтобы задать шестнадцатеричную константу, необходимо использовать префикс 'x', например, '\x23'.

Регистры

SoftICE поддерживает стандартные имена набора регистров процессоров x86 фирмы Intel.

AH	CS	EBX	FL
AL	CX	ECX	FS
AX	DH	EDI	GS
BH	DI	EDX	IP
BL	DL	EFL	SI
BP	DS	EIP	SP
BX	DX	ES	SS
CH	EAX	ESI	
CL	EBP	ESP	

Совет: Для доступа к отдельным флагам флаговых регистров EFL и FL можно использовать встроенные функции SoftICE. Смотрите раздел "*Встроенные функции*" на странице 105.

Идентификаторы

Идентификаторы служат для символьного представления адресов или значений. Они могут встречаться во множестве форм, например в таблицах переменных, таблицах экспорта или во встроенных функциях.

Идентификаторы, за некоторыми исключениями, сделанными для некоторых языков, например, ассемблера и C++, строятся по правилам языка C. Допустимые имена начинаются с символов 'at' (@), подчеркивания (_) или букв латинского алфавита от A до Z и представляют собой непрерывную цепочку из букв (от A до Z), цифр (от 0 до 9) или других допустимых символов, таких как, например, подчеркивание (_).

Совет: Идентификаторы для языка C++ могут включать в себя оператор разрешения области видимости (::), когда идентификатор представляется в виде класс::член-класса.

Когда идентификаторы загружены в SoftICE, они размещаются в таблицах символов, причем идентификаторы разных исполняемых модулей размещены в разных таблицах, поэтому SoftICE не сможет обнаружить идентификатор, если он находится в неактивной в данный момент таблице. Смотрите команду TABLE в "*Справочнике по командам SoftICE*".

Совет: Это не относится к таблицам экспорта, так как SoftICE рассматривает все экспортируемые имена как однородное целое, в то время как таблицы идентификаторов используются независимо друг от друга.

Указатель таблицы идентификаторов может предшествовать самому символическому имени, что позволяет использовать идентификаторы из разных таблиц в одном выражении. Идентификаторы и имена их таблицы разделяются восклицательным знаком (!), например:

имя-таблицы!*идентификатор*

Встроенные функции

SoftICE предоставляет пользователю ряд собственных функций. Они обеспечивают доступ к статическим и динамическим переменным операционной системы или к собственным переменным SoftICE; некоторые из этих функций могут быть использованы для изменения значений или для преобразования типов данных.

Использование функций, не требующих аргументов, сходно с применением обычных идентификаторов из таблиц символов. Функции с аргументами по своему виду и поведению схожи с функциями языка C:

имя-функции(*список-аргументов*)

Замечание: В пределах одной таблицы символов или экспорта имена идентификаторов имеют более высокий приоритет по сравнению с именами функций и замещают их в выражениях.

В следующей таблице представлены функции, определенные в SoftICE:

Имя функции	Описание	Пример
Byte	Возвращает младший байт	`? Byte(0x1234) = 0x34`
Word	Возвращает младшее слово	`? Word(0x12345678) = 0x5678`
Dword	Возвращает двойное слово	`? Dword(0xFF) = 0x000000FF`
HiByte	Возвращает старший байт	`? HiByte(0x1234) = 0x12`
HiWord	Возвращает старшее слово	`? HiWord(0x12345678) = 0x1234`
Sword	Преобразует байт в слово со знаком	`? Sword(0x80) = 0xFF80`
Long	Преобразует байт или слово в длинное целое со знаком	`? Long(0xFF) = 0xFFFFFFFF` `? Long(0xFFFF) = 0xFFFFFFFF`
WSTR	Показать как строку в UNICODE	`? WSTR(eax)`
Flat	Преобразовать адрес на базе селектора в линейный адрес (непрерывная (flat) модель адресации).	`? Flat(fs:0) = 0xFFDFF000`
CFL	Флаг переноса	`? CFL =` *логический-тип*
PFL	Флаг четности	`? PFL =` *логический-тип*
AFL	Флаг вспомогательного переноса	`? AFL =` *логический-тип*
ZFL	Флаг нуля	`? ZFL =` *логический-тип*
SFL	Знаковый флаг	`? SFL =` *логический-тип*
OFL	Флаг переполнения	`? OFL =` *логический-тип*
RFL	Флаг возобновления	`? RFL =` *логический-тип*
TFL	Флаг трассировки	`? TFL =` *логический-тип*
DFL	Флаг направления	`? DFL =` *логический-тип*
IFL	Флаг разрешения прерываний	`? IFL =` *логический-тип*
NTFL	Флаг вложенной задачи	`? NTFL =` *логический-тип*

Продолжение на следующей странице

Имя функции	Описание	Пример
IOPL	Уровень привилегий ввода/вывода	`? IOPL = текущий уровень привилегий ввода/вывода`
VMFL	Флаг режима виртуального процессора	`? VMFL = логический-тип`
IRQL	Текущий IRQ для драйверов Windows NT	`? IRQL = беззнаковое-целое`
DataAddr	Возвращает начальный адрес блока данных, отображаемого в Окне Данных	`dd @dataaddr`
CodeAddr	Возвращает адрес первой инструкции, отображаемой в Окне Кода	`? codeaddr`
Eaddr	Возвращает эффективный адрес текущей инструкции, если таковой существует. Подробнее смотрите в разделе "*Функция Eaddr*" на странице 107	
Evalue	Возвращает значение по текущему эффективному адресу. Подробнее смотрите в разделе "*Функция Evalue*" на странице 108	
Process	Блок среды активного процесса операционной системы (KPEB — Kernel Process Environment Block)	`? process`
Thread	Блок среды активного потока операционной системы (KTEB — Kernel Thread Environment Block)	`? thread`
PID	Идентификатор активного процесса	`? pid == Test32Pid`
TID	Идентификатор активного потока	`? tid == Test32MainTid`
BPCount	Возвращает количество попаданий в контрольную точку, при которых значение условного выражения было равно TRUE. Более подробную информацию относительно BPxxx функций см. в разделе "*Функции статистики условных прерываний*" на странице 92	`bp <параметры-команды> IF bpcount==0x10`

Продолжение на следующей странице

Имя функции	Описание	Пример
BPTotal	Возвращает общее количество попаданий в контрольную точку	`bp <параметры-команды> IF bptotal>0x10`
BPMiss	Возвращает количество попаданий в контрольную точку, при которых условие возникновения прерывания не выполнялось и SoftICE не был активизирован	`bp <параметры-команды> IF bpmiss==0x20`
BPLog	Сохраняет в буфере информацию о попадании в контрольную точку без выполнения каких-либо дополнительных действий	`bp <параметры-команды> IF bplog`
BPIndex	Возвращает номер текущей контрольной точки в общем списке точек прерывания	`bp <параметры-команды> DO "bd bpindex"`

Функция Eaddr

Функция Eaddr возвращает эффективный адрес, если таковой существует, который использует текущая инструкция, указанная регистром EIP.

Замечание: Эффективный адрес, если он существует, а также значение по данному адресу высвечиваются в Окне Регистров непосредственно под значениями флагов.

Процессоры семейства x86 предоставляют различные способы адресации такие, как, например, 'регистр + смещение' или 'регистр + регистр'. Результат вычисления адреса памяти называется *эффективным адресом*. Об инструкции, обращающейся к памяти, говорят, что она использует эффективный адрес источника или адресата. Команды процессоров x86 никогда не используют эффективные адреса одновременно для обращения к источнику и приемнику.

Некоторые инструкции не используют эффективный адрес, так как они обращаются только к регистрам, или потому, что адресация производится специфическим для данного типа инструкций образом, как, например, у инструкций обращения к стеку PUSH и POP.

Пример: Для текущей инструкции:

```
MOV ECX,[ESP+4]
```

функция Eaddr возвратит значение равное [ESP]+4, то есть содержимое регистра ESP плюс 4.

Пример: Текущей инструкцией является:

```
ADD BYTE PTR [ESI+EBX+2],55
```

Функция Eaddr возвратит результат сложения содержимого регистров ESI, EBX и числа 2.

Функция Evalue

Функция Evalue возвращает значение по эффективному адресу текущей инструкции, если таковой для нее существует. При этом результат не обязательно соответствует 'Eaddr->0', так как функция Evalue чувствительна к размеру операнда. Она возвращает в качестве результата байт, слово или двойное слово.

Замечание: Эффективный адрес, если он существует, а также значение по данному адресу высвечивается в Окне Регистров непосредственно под значениями флагов.

Типы выражений

SoftICE использует очень простую систему, которая сводит все типы выражений к одному из следующих:

Тип	Пример
Литерный тип	1, 0x80000000, 'ABCD'
Регистровый тип	EAX, DS, ESP
Тип 'идентификатор'	PoolHitTag, IsWindow
Адресный тип	40:17, FS:18, &Symbol

Замечание: Функции, как таковые, типа не имеют, однако они возвращают результат одного из вышеперечисленных типов.

В большинстве случаев Вы можете игнорировать различия между типами, так как это важно только для SoftICE. Однако, в случае идентификаторов и адресного типа имеются некоторые семантические особенности (или ограничения), которые необходимо понимать.

Идентификаторы используются в качестве символических имен и размещаются в таблицах экспорта или символов. Как правило, этот тип данных представляет собой линейный адрес в сегменте кода или данных. Кроме того, этот тип символизирует содержимое памяти по этому линейному адресу. Это сходно с использованием переменных в программах на языке C, однако, вследствие того, что SoftICE — это отладчик, а не компилятор, имеются некоторые семантические различия. SoftICE определяет, что именно Вы имеете в виду — '*адрес*' или '*содержимое по адресу*' на основании того, как символ (переменная) используется в выражении. В общем случае, обращение SoftICE с идентификаторами достаточно естественно (в отличии от языка C); однако, если Вы не уверены в том, каким именно способом SoftICE будет их интерпретировать в конкретном случае, следует точно указать, что это:

`адрес (&Symbol)` **или** `содержимое по адресу (*Symbol)`

SoftICE обращается с идентификатором, как с адресным типом, если Вы используете его в выражении, где применение адресного типа допустимо, и имеет смысл. В противном случае, SoftICE автоматически использует '*содержимое*', извлекая его по тому адресу, который представляет данный идентификатор. Существует множество операций, в которых недопустимо или не имеет смысла использовать адресный тип, таких, например, как умножение или деление, поэтому большинство операторов обращаются с идентификаторами подобно языку C и автоматически извлекают содержимое по адресу, им указанному.

В следующей таблице показано, как SoftICE интерпретирует идентификаторы в выражениях.

Пример	Точное выражение	Symbol интерпретируется как:
u Symbol	u &Symbol	адрес
db Symbol + 1	db &Symbol + 1	адрес
db Symbol + ds:8000	db *Symbol + ds:8000	содержимое
db Symbol + Symbol2	db &Symbol + *Symbol2	адрес
? Symbol - 1	? &Symbol - 1	адрес
? Symbol - ds:8000	? &Symbol - ds:8000	адрес
? Symbol - Symbol2	? *Symbol - *Symbol2	содержимое
? Symbol && 1	? *Symbol && 1	содержимое
? Symbol && ds:8000	? *Symbol && ds:8000	содержимое
? Symbol && Symbol2	? *Symbol && *Symbol2	содержимое
? Symbol <= 8000	? *Symbol <= 8000	содержимое
? Symbol != &Symbol2	? &Symbol != &Symbol2	адрес
? Symbol == Symbol2	? *Symbol == *Symbol2	содержимое
? Symbol : 8000	? *Symbol : 8000	содержимое
? -Symbol	? -*Symbol	содержимое
? !Symbol	? !*Symbol	содержимое
? Symbol->4	? *(&Symbol+4)	адрес

Следующие операторы не могут непосредственно использовать адресный тип:

Недопустимая форма выражений	Пример
адрес [*, /, %, <<, >>] любой тип	~~&Symbol * 4~~
адрес [+, &, |, ^] адрес	~~ds:80ff ^ &Symbol~~
любой тип [->, .] адрес	~~ebp->&Symbol2~~
адрес [:] любой тип	~~&Symbol : 8000~~
[-, ., &] адрес	~~- &Symbol~~, ~~.&Symbol (номер строки)~~
адрес - адрес *Замечание*: Это выражение недопустимо, если селекторы имеют различное содержимое и тип	23:8fff - 23:4ff0 (допустимо) ~~1b::0 - 23:0~~ (недопустимо)

Замечание: В отличие от идентификаторов SoftICE не может самостоятельно получать содержимое памяти с помощью адресного типа. Вам необходимо прямо указать это, используя один из операторов адресации.

Адресация содержимого

Существует тонкое различие между парой операторов (->) и (.) и операторами (*) и (@). Результатом оператора (->) или (.) является простое двойное слово, тогда как результатом (*) и (@) будет адрес.

Следующее выражение недопустимо, так как операция умножения для адреса невозможно:

```
? (*Symbol)*3
```

Если вы попытаетесь вычислить это выражение, то получите сообщение об ошибке: `Expecting value, not address` (Ожидается значение, а не адрес).

Однако, следующее выражение вполне допустимо, так как результатом `Symbol->0` является значение, а не адрес:

```
? (Symbol->0)*3
```

Это различие полезно, когда выполняются многочисленные переадресации в 16-битном коде, так как адреса содержат в себе информацию о сегменте и смещении.

Размеры операндов

SoftICE обращается со всеми операндами как с двойными словами (длинное беззнаковое целое). Это означает, что в случае необходимости Вы должны самостоятельно указывать размер операнда, используя операторы приведения типа или одну из функций преобразования, такую как, например, byte() или word().

Пример: Если Вы ссылаетесь на содержимое в области памяти, то SoftICE всегда возвращает значение в виде двойного слова. Чтобы правильно сравнить битовое значение в условном выражении, необходимо замаскировать 24 старших бита, оставив неизмененными младшие 8 (предположим, что Symbol представляет собой битовое значение):

```
BPX EIP IF (Symbol == 32)
```

Данное условие, скорее всего, не выполнится, так как SoftICE получит из памяти 32-битовое значение и сравнит его с двойным словом 32 или 0x00000032. А это не то, что Вам было нужно. Следующие выражения выполнят свою задачу правильно.

```
BPX EIP IF ((Symbol & FF) == 32)
```

или

```
BPX EIP IF (byte(Symbol)== 32)
```

Можно пользоваться любым из выше приведенных способов, они равноценны.

Приведение типов

SoftICE поддерживает следующие способы приведения типов:

- Способ приведения, подобный языку C++:
  Для приведения любого значения к желаемому типу используется следующая форма:
  ИмяТипа (Выражение)

Замечание: 'ИмяТипа' является регистр-зависимым, так как для поиска используется хеширование, а не линейный поиск.

- Адресация через члены структур или классов:
  ИмяТипа (выражение)->член
  После выполнения адресации тип выражения автоматически приводится к типу члена. Возможна множественная переадресация.
  ИмяТипа (выражение)->член->член->член
  При каждой последующей переадресации вычисляется значение очередного члена и применяется приведение типа, затем вычисляется следующий член, и так до тех пор, пока все выражение не будет вычислено.

- Получение адреса члена или типа:
  Можно использовать оператор & (адрес) для получения адреса структуры или члена структуры.
  &ИмяТипа (выражение)->член[->член[->член]]
  Это позволит Вам установить контрольную точку на член структуры с помощью оператора 'BPM'.
- Отображение типизированного выражения:
  Когда это возможно, команда '? (выражение)' выводит результат выражения в соответствии с его типом. Многие типы выражений, подобно регистрам, имеют свои основные типы.
  Для сложных типов происходит распространение типа члена класса или структуры. При этом распространяется тип только членов корневого уровня. Заметьте себе, что базовые или виртуальные базовые классы рассматриваются как корневые объекты.

Пример:
```
:? LPSTR (*(ebp-30))
char * = 0x009D000C
  <"C:\TOMSDEV\WINICE\NTICE"> char = 0x43 , 'C'
```

Пример:
```
:? STHashTable (a7bcb0)
class STHashTable = {...}
struct STHashNode ** pHashTable = 0x0089000C <{...}>
unsigned long bucketSize = 0x25
class GrowableArray * pHashEntries = 0x00A7BCC0
<{...}>
```

Пример:
```
:? STHashTable (a7bcb0)->pHashEntries
class GrowableArray * =0x00A7BCC0 <{...}>
unsigned char * arrayBase = 0x00790078 <"">
unsigned char * nextItem = 0x00790078 <"">
unsigned long memAvail = 0x1000
unsigned long elementSize = 0x10
```

- Отображение указателя на указатель:
  Типом указателя на указатель является значение, на которое ссылается последний:
  ```
  typedef LPSTR *LPLPSTR;
  ? LPLPSTR (eax)
  char **eax = 0x127894 <0x434000>
  ```
  где `0x127894` представляет собой значение указателя, а `<0x434000>` является значением, которое размещено данному (0x127894) адресу.
- Отображение строк в Unicode:
  Для отображения строк в Unicode используется оператор приведения WSTR:
  ```
  ? WSTR (eax)
  short *eax = <"Company Name">
  ```

Исчисление идентификаторов

Когда имеется информация о типе данных, использование команды ? (вычислить выражение) с идентификатором позволяет получить значение данного идентификатора, а не его адрес. Так, например, если MyVariable является переменной целого типа, содержащей значение 5, то

```
? MyVariable
int=0x5,"\0\0\0\x05"
```

Для получения адреса MyVariable, следует использовать:

```
? &MyVariable
```

Использование операторов адресации с идентификаторами

Когда Вы создаете файл, содержащий полную информацию о типах и идентификаторах, то SoftICE способен вычислять ссылки через символические имена, используя тип идентификатора. Вы также сможете получить адрес члена класса через его идентификатор.

```
typedef struct Test
{
  DWORD dword ;
  LPSTR lpstr ;
} Test ;
Test test={ 1, "test String" } ;
? test->dword
unsigned long dword=1
? test->lpstr
char * lpstr =0x123456 , <"Test String">
? &test
void * =0x123440
? &test->dword
void * =0x123440
? &test->lpstr
void * =0x123444
```

Того же самого можно достичь при использовании оператора приведения типа:

`Test(eax)->dword` **или** `Test(eax)->lpstr`

и

`&Test(eax)->dword` **или** `&Test(eax)->lpstr`

Отчетливо и зримо символы выделяются…
Джон Китс

Distinct, and visible; symbols devine…
John Keats

9. Загрузка символов для системных компонентов

Загрузка экспортируемых символов для DLL- и EXE-файлов

Экспортируемые символы являются одной из составных частей 16- и 32-битного формата исполняемых файлов в Windows, которые позволяют производить динамическое связывание между исполняемым модулем (как правило), который импортирует функции, и библиотекой .DLL, которая эти функции экспортирует.

В исполняемом файле имя в ASCII-формате и порядковый номер (ordinal number), или, иногда, только порядковый номер, ставятся в соответствие точкам входа в модуле. Вполне разумно загрузить экспортируемую информацию в отладчик в символьном виде, особенно, когда собственная отладочная информация приложения отсутствует. Как правило, таблицы экспорта присутствуют только в DLL, но иногда и EXE-файлы могут экспортировать функции; примером такого файла является NTOSKRNL.EXE.

Вы можете настроит установки SoftICE таким образом, чтобы загружать экспортируемые символы любых 16- и/или 32-битных .DLL и .EXE-файлов. Во время своего запуска SoftICE загружает также и таблицы экспорта указанных файлов, что делает идентификаторы доступными для использования в любых выражениях. Кроме того, все они автоматически показываются вместе с дизассеблированным кодом. Для получения дополнительной информации о загрузке экспортируемой информации при старте SoftICE смотрите раздел "*Предварительная загрузка экспортируемой информации*" на странице 123.

Если модуль еще не запущен в работу, то информация о 32-битном экспорте программы для сегмента отображается в виде порядкового номера «FE», а смещение является просто смещением относительно начала загружаемой части файла. После того, как модуль задействован в каком-либо процессе, появляются настоящие адреса в формате "селектор : смещение". В состав смещения теперь входит базовый адрес загрузки.

Когда 32-битный модуль исключается из всех функционирующих процессов, которые могли бы его использовать, все адреса снова превращаются в обычные пары "номер-сегмента:смещение".

Замечание: Когда .DLL задействована в двух процессах с различными виртуальными базовыми адресами, таблица экспорта использует базовый адрес первого процесса, задействовавшего данную DLL, однако эти адреса будут неверными для второго процесса. Вы можете избежать этого, указав необходимый адрес для этой DLL, или перебазировав ее.

Замечание: Для 16-битных программ загружается экспортируемая информация только из нерезидентных таблиц имен; однако, как правило, это вся или почти вся экспортируемая таким модулем информация.

Использование неименованных точек входа

Для 32-битного экспорта SoftICE показывает все экспортированные точки входа даже, если они не имеют ассоциированных с ними имен. Для 16-битного экспорта SoftICE показывает только имена. Если была экспортирована точка входа без имени, SoftICE создает для нее имя в виде:

`ORD_xxxx`

где *xxxx* - это порядковый номер.

Такие имена могут конфликтовать друг с другом, так как многие DLL могут иметь свои неименованные точки. Для того, чтобы быть уверенным, что используется требуемый идентификатор, следует добавить перед ним имя модуля с последующим восклицательным знаком.

Пример: Чтобы сослаться на первую точку входа KERNEL32, следует использовать выражение:

```
KERNEL32!ORD_0001
```

Нет необходимости вставлять после префикса ORD_ точное количество нулей, допустимым является как идентификатор ORD_0001, так и ORD_1. Следующее выражение эквивалентно предыдущему примеру:

```
KERNEL32!ORD_1
```

Использование экспортируемых имен в выражениях

SoftICE сначала просматривает все 32-битные таблицы экспорта и лишь затем ведет поиск в 16-битных. Это означает, что, если одно и то же имя присутствует в обоих типах таблиц, то будет использована 32-битная. Если же требуется изменить такую последовательность действий, то следует добавить перед именем экспортированного символа имя модуля с последующим восклицательным знаком.

Пример: Когда задается имя GlobalAlloc, SoftICE использует 32-битную таблицу экспорта модуля KERNEL32.DLL, а не 16-битное имя, экспортируемое из KRNL32.EXE. Вы сможете использовать 16-битную версию GlobalAlloc, задав полное имя экспортированного идентификатора:

```
KERNEL!GlobalAlloc
```

Кроме того, порядок поиска внутри каждого типа таблиц экспорта (32- и 16-битного) определяется той последовательностью, в которой эти таблицы загружались.

Динамическая загрузка 32-битного DLL-экспорта

SoftICE позволяет Вам загружать 32-битные таблицы экспорта без перезагрузки системы. Для того, чтобы загрузить 32-битные таблицы динамически, необходимо:

1 Запустить утилиту Symbol Loader.

2 Затем либо выбрать из меню "FILE" пункт "LOAD EXPORTS", либо нажать аналогичную кнопку на панели управления.
Появится окно загрузки экспорта (‘Load Exports').

3 Выбрать файлы, которые Вы хотите загрузить и нажать "OPEN".

Использование символьных файлов Windows NT (.DBG) с SoftICE

Microsoft предоставляет отладочную информацию для ключевых компонентов Windows NT. Она поставляется в виде .DBG-файлов, которые содержат отладочные данные в формате COFF для соответствующих .EXE и .DLL. .DBG-файлы можно найти на CD-ROM с Windows NT или загрузить их с соответствующими сервисными комплектами (service pack). Чтобы использовать .DBG-файлы с SoftICE, необходимо транслировать их с помощью утилиты Symbol Loader в формат .NMS и загрузить полученные файлы.

Использование символьных файлов Windows 95 (.SYM) с SoftICE

Windows 95 DDK включают в себя символьную информацию для некоторых системных модулей в виде .SYM-файлов. Используйте утилиты Symbol Loader или NMSYM для трансляции их в формат .NMS и загрузите полученные файлы в SoftICE.

Однако, если только я не ошибся, вот и наш клиент
Сэр Артур Конан Дойл

But here, unless I am mistaken, is our client.
Sir Arthur Conan Doyle

10. Использование SoftICE с модемом

Введение

Вы можете управлять SoftICE удаленно с помощью модема. Это чрезвычайно полезно для отладки ошибок, которые проявляются на удаленном компьютере, и которые невозможно воспроизвести иным путем.

Когда Вы работаете с SoftICE через модем, на локальном компьютере запущены и SoftICE, и отлаживаемое приложение. Удаленный компьютер при этом функционирует как неинтеллектуальный терминал, отображая выводимую отладчиком информацию, и воспринимая ввод с клавиатуры. В таком режиме SoftICE не поддерживает работу с мышью на удаленном компьютере.

Аппаратные требования

Модем, используемый для связи локального и удаленного компьютеров, должен отвечать следующим требованиям:

- Модем должен поддерживать стандартные AT-команды, такие, например, как ATZ и ATDT, а также возвращать стандартные коды типа RING и CONNECT.
- Модем должен поддерживать протокол коррекции ошибок, такой как V.42 или MNP5[†]. Это важно потому, что протокол, используемый SoftICE для связи, автоматическую коррекцию ошибок в себя не включает.

Установление соединения

При управлении SoftICE через модем процедура установки связи может быть запущена как с локального, так и с удаленного компьютера.

Для установления соединения, когда локальный компьютер звонит на удаленный, выполните следующие действия:

1 На удаленном компьютере запускается программа SERIAL.EXE

2 Пользователь на локальном компьютере в SoftCE вызывает команду DIAL. Устанавливается соединение, и удаленный компьютер оказывается под контролем SoftICE.

Если необходимо звонить с удаленного компьютера локальный, то выполните иную последовательность действий:

[†] Скорее всего, имеется в виду протокол MNP4, так как MNP5 является протоколом сжатия данных.

1 На локальном компьютере в программе SoftICE выполняется команда ANSWER.
2 С удаленного компьютера с помощью программы SERIAL.EXE звонят на локальный.
Устанавливается соединение, и удаленный компьютер оказывается под контролем SoftICE.

В следующем разделе описывается, как пользоваться программой SERIAL.EXE и командами DIAL и ANSWER. За более подробным описанием команд обращайтесь к "*Справочнику по командам SoftICE*".

Использование программы SERIAL.EXE

SERIAL.EXE является MS-DOS программой, которая выполняет для SoftICE роль неинтеллектуального терминала, воспроизводя выводимую отладчиком информацию, и воспринимая ввод с клавиатуры удаленного компьютера. Все, что выводится с помощью программы SERIAL.EXE на экран монитора, в точности соответствует тому, что Вы бы увидели, если бы просто запустили SoftICE на локальном компьютере.

Замечание: Вы можете использовать программу SERIAL.EXE при связи компьютеров как по модему, так и с помощью последовательного соединения. Если Вы хотите связать компьютеры с помощью последовательного соединения, то за дополнительной информацией обратитесь к разделу "*Подключение удаленного компьютера через последовательный порт*" на странице 21.

Программа SERIAL.EXE имеет следующие параметры командной строки:

`SERIAL.EXE [r] [`*`com-port`*`] [`*`baud-rate`*`] [I"`*`init-string`*`"] [P`*`номер`*`]`

r Опция "r" используется, когда Вы запускаете программу на удаленном компьютере в окне DOS под управлением Windows NT. Этот параметр отключает FIFO-буфер и автоматически устанавливает такие параметры, как скорость связи, стоповые биты и бит четности.

`com-port` Номер коммуникационного порта, 1-4; это обязательный параметр.

`baud-rate` Скорость, с которой SERIAL.EXE взаимодействует с модемом. Она не всегда совпадает со скоростью, которая сообщается командами DIAL и ANSWER программы SoftICE на локальном компьютере. Это также обязательный параметр.

I Используется для задания строки инициализации модема.

`"init-string"` Строка с командами, передаваемыми модему перед началом работы.

P Используется для задания телефонного номера.
Если задан параметр "P", то SERIAL набирает номер телефона, к которому подключен локальный компьютер, и на котором в программе SoftICE уже выполнена команда ANSWER.
Например: `SERIAL 1 57000 p1-603-555-1212.`
Если параметр не задан, то программа SERIAL работает в режиме ответа, ожидая входящего звонка. Например: `SERIAL 1 57000.`

`номер` Номер телефона, по которому необходимо звонить (когда SERIAL используется для исходящего звонка). Модему посылается командная строка в виде "ATDT*number*". Для звонка с применением импульсного набора номера используйте букву "P" в качестве первой цифры (работает не со всеми модемами).

После установления соединения пользователь на удаленном компьютере видит привычный экран SoftICE и может управлять работой отладчика.

Команда DIAL

Команда DIAL программы SoftICE используется для звонка на удаленный компьютер. На самом удаленном компьютере уже должна быть запущена программа SERIAL.EXE в режиме ответа.

Команда имеет следующие параметры:

```
DIAL [ON [com-port] [baud-rate] [I=init-string] [P=номер] | OFF]
```

ON Запуск набора номера.

OFF Окончание сеанса связи.

com-port Номер коммуникационного порта: 1-4 (по умолчанию 1 порт).

baud-rate Скорость, с которой SoftICE взаимодействует с модемом (по умолчанию равна 57000). Она не обязательно должна совпадать со скоростью, которая сообщается программой SERIAL.EXE на удаленном компьютере. Если этот параметр задан, то должен быть указан и номер коммуникационного порта.

I Используется для задания строки инициализации модема.

init-string Строка с командами, передаваемыми модему перед началом работы. Если параметр не задан, то используется значение строки последнего сеанса связи; если и оно не определено, то передается строка из начальных установок программы SoftICE.

P Используется для задания телефонного номера.
Если задан параметр "P", то SoftICE набирает номер телефона и пытается установить соединение с программой SERIAL.EXE, которая должна ожидать звонка в режиме ответа.

номер Номер телефона, по которому необходимо звонить. Модему посылается командная строка в виде "ATDT*number*". Для звонка с применением импульсного набора номера используйте букву "P" в качестве первой цифра (работает не со всеми модемами). Если параметр не задан, то используется номер телефона последнего сеанса связи или, если таковой не задавался вообще, номер из начальных установок программы SoftICE.
Например, `DIAL ON 1 57000 p=603-555-1212`.

Команда ANSWER

Команда ANSWER переводит SoftICE в режим ответа на входящий звонок, который производится с удаленного компьютера программой SERIAL.EXE.

Команда имеет следующие параметры:

```
ANSWER [ON [com-port] [baud-rate] [I=init-string] | OFF]
```

ON Включает режим ответа.

OFF Выключает режим ответа.

com-port Номер коммуникационного порта: 1-4 (по умолчанию 1 порт).

baud-rate Скорость, с которой SoftICE взаимодействует с модемом. По умолчанию она равна 57000. Она не обязательно должна совпадать со скоростью, которая сообщается программой SERIAL.EXE на удаленном компьютере. Если этот параметр задан, то должен быть указан и номер коммуникационного порта.

init-string Строка с командами, которые используются для инициализации модема. Если параметр не задан, то используется значение строки последнего сеанса связи; если и оно не определено, то передается строка из начальных установок программы SoftICE.
Например, `ANSWER ON 1 57000`.

В значительной степени формирование человеческой личности находится в ее собственных руках
Сэр Френсис Бэкон

Chiefly the mould of a man's fortune is in his own hands.
Sir Francis Bacon

11. Настройка SoftICE

Изменение начальных установок SoftICE

Программа SoftICE предоставляет пользователю широкие возможности для настройки режимов работы на этапе ее начальной загрузки. Параметры начальных установок можно разделить на следующие категории:

- Общие (General) — Множество полезных параметров, включая строку команд, которые автоматически выполняются при запуске SoftICE.
- Отладочная информация (Symbol) — Позволяют указать .NMS-файлы с отладочной информацией, которые загружаются при запуске отладчика.
- Экспортируемая информация (Exports) — Указывают .DLL- и .EXE-файлы, из которых при старте SoftICE будут загружены экспортируемые таблицы.
- Удаленная отладка (Remote Debugging) — Задание телефонного номера и строки инициализации модема для использования при удаленной отладке через коммуникационный порт.
- Переназначение клавиш (Keyboard Mappings) — Присваивают функциональным клавишам клавиатуры значения команд SoftICE.
- Определение макрокоманд (Marco Definitions) — Позволяют создавать собственные макрокоманды для использования при работе с SoftICE.
- Неисправности (Troubleshooting) — Предоставляют решения возможных проблем.

Для изменения начальных установок выполните следующие действия:

1 Запустите утилиту Symbol Loader.
2 Выберите пункт "SoftICE Initialization Settings..." (Начальные установки SoftICE) из меню Edit.
SoftICE обозначает диалог начальных установок следующим образом:
SoftICE Init Settings Dialog Box Here
(Окно диалога начальных установок SoftICE).
3 Выберите категорию параметров, которые Вы хотите изменить.
4 Измените необходимые параметры и нажмите OK.
В следующих разделах все параметры описаны подробно.
5 Перезапустите компьютер для запуска SoftICE с учетом сделанных Вами изменений.

Изменение общих установок (General)

На вкладке "GENERAL" могут быть модифицированы следующие параметры:

Initialization string (Инициализационная строка)

Строка инициализации содержит последовательность команд, которые выполняются при старте SoftICE. По умолчанию она содержит команду X (eXit — Выход), завершаемую точкой с запятой:

```
X;
```

В строку могут быть добавлены дополнительные команды, например, для изменения стандартной комбинации клавиш Ctrl-D, вызывающей экран SoftICE, для изменения размеров экрана отладчика или увеличения количество строк, им отображаемых, или для установления командой SERIAL соединения для удаленной отладки. Если Вы занимаетесь отладкой драйверов устройств, то, возможно, Вы захотите убрать команду X (или точку с запятой после нее) для предотвращения автоматического выхода из стадии инициализации.

Чтобы добавить команды в строку инициализации, впечатайте новые команды, разделенные точкой с запятой, перед командой X. Команды будут обрабатываться в той последовательности, в какой Вы их поместите. Так, помещение команд после команды X означает, что они не будут выполнены до тех пор, пока находитесь в окне SoftICE. Если Вы напечатаете команду без точки с запятой, то SoftICE только загрузит ее в окно команд, но не выполнит.

Пример: Следующая инициализационная строка переключает SoftICE в режим отображения экрана размером 50 строк, изменяет клавиатурную последовательность вызова на "Alt-Z", включает окно регистров и производит выход из стадии инициализации:

```
LINES 50; ALTKEY ALT Z; WR; X;
```

Замечание: Если Вы набираете строку, превосходящую по ширине поле ввода, то строка автоматически сдвигается влево, позволяя Вам видеть вводимую информацию.

History buffer size (Размер буфера протокола)

Параметр "HISTORY BUFFER SIZE" определяет размер буфера протокола работы SoftICE. По умолчанию он равен 256 Кб.

Протокол SoftICE содержит всю информацию, которая выводилась в окне команд. Поэтому сохранение протокола в файл полезно для записи больших объемов данных, дизассемблированных кодов, сохранения истории выполнения контрольных точек, выданной командой BPLOG, а также последовательности сообщений Windows, полученной по команде BMSG. Подробности смотрите в разделе "*Сохранение содержимого буфера протокола окна команд в файл*" на странице 66.

Trace buffer size (Размер буфера обратной трассировки) (только для Windows 95)

Этот параметр определяет размер буфера обратной трассировки. Буфер поддерживает обратную трассировку для команд BPR и BPRW. По умолчанию его размер 8 Кб.

Total RAM (Размер оперативной памяти) (только для Windows 95)

Этот параметр указывает количество физической памяти на Вашем компьютере. Устанавливайте его величину равной или большей, чем количество памяти на Вашем компьютере.

В результате особенностей архитектуры операционной системы SoftICE под управлением Windows 95 не может точно определить размер физической оперативной памяти. Чтобы правильно отобразить взаимосвязь между линейной и физической памятью, SoftICE использует значение, равное по умолчанию 128 Мб. Но, хотя это значение справедливо для большинства компьютеров, на которых занимаются созданием и отладкой программ, оно неверно работает на системах с большими размерами физической памяти. Это происходит вследствие того, что соответствующие структуры данных для страниц памяти выше 128 Мб не создаются.

Если на Вашем компьютере установлено меньше 128 Мб оперативной памяти, то Вы можете сэкономить некоторое количество памяти, устанавливая в этом параметре действительное значение, так как для отображения физической памяти будут созданы структуры меньшего размера.

Display diagnostic messages (Отображение диагностических сообщений)

Параметр "DISPLAY DIAGNOSTIC MESSAGES" определяет, будет ли SoftICE показывать в окне команд такую дополнительную информацию, как, например, загрузка и выгрузка модулей. По умолчанию этот параметр включен.

Trap NMI (Перехват немаскируемого прерывания)

Этот параметр определяет включен или нет перехват немаскируемого прерывания (Non-maskable interrupt - NMI). По умолчанию он включен. Генерирование немаскируемого прерывания позволяет Вам перейти в SoftICE даже, если запрещены все прерывания. Такие прерывания генерируются, например, системами энергопитания.

Lowercase disassembly (Вывод дизассемблерованного кода строчными буквами)

Параметр "LOWERCASE DISASSEMBLY" определяет будут ли использованы символы нижнего регистра для вывода дизассемблированных инструкций. По умолчанию параметр выключен.

Предварительная загрузка символической информации и исходного кода (Symbols)

Установки параметров инициализации символьной информации в сочетании с параметрами трансляции модулей (Module Translation) используются для предварительной загрузки отладочной информации и исходных кодов при старте SoftICE. Такая предварительная загрузка полезна, например, при отладке драйверов устройств.

Для предварительной загрузки символической информации или исходных кодов выполните следующее:

1 В установках "MODULE TRANSLATION" выберите пункт "SYMBOLS AND SOURCE CODE" или нажмите соответствующую кнопку на панели управления для загрузки исходных кодов в дополнение к отладочной информации.

2 Выберите пункт "PACKAGE SOURCE WITH SYMBOL TABLE".

3 В утилите Symbol Loader выберите пункт "TRANSLATE" из меню "MODULE" для того, чтобы оттранслировать файл с отладочной информацией в файл формата .NMS.
4 С помощью вкладки "SYMBOLS" добавьте созданный Вами .NMS-файл к списку символьных файлов. Что для этого надо сделать, описано в следующем разделе.

Добавление символьных файлов в список

На вкладке "SYMBOLS" начальных установок SoftICE:
1 Нажмите кнопку "ADD" (Добавить).
SoftICE откроет диалог, чтобы Вы нашли .NMS-файл, который содержит отладочную информацию и исходные коды, которые Вы хотите загрузить.
2 Выберите необходимые .NMS-файлы и нажмите OK.

Совет: Обычно .NMS-файл имеет то же имя, что и файл, из которого Вы его транслировали. Работая под управлением Windows 95, SoftICE не может выполнять предварительную загрузку файлов отладочной информации с длинными именами, так как начальная загрузка SoftICE выполняется в реальном режиме DOS. Если Ваш модуль имеет длинное имя, то после создания .NMS файла переименуйте его, присвоив имя из 8 символов с расширением .NMS, и при добавлении в список выбирайте переименованный файл.
Другой подход состоит в использовании вместо Symbol Loader утилиты командной строки NMSYM, которая позволяет задавать имя создаваемого в результате трансляции .NMS-файла.

Совет: Если вы выбираете режим " PACKAGE SOURCE WITH SYMBOL TABLE ", исходные тексты становятся частью .NMS файлов. В этом случае нет ограничения длины имени файлов даже при использовании Windows 95.

3 Всякий раз, когда вы изменяете исходные коды своей программы, транслируйте модуль заново, чтобы иметь свежую версию .NMS файла.

Удаление исходных файлов и файлов с символьной информацией из списка предварительной загрузки

Чтобы отменить предварительную загрузку отладочной информации, выделите соответствующие файлы в списке и нажмите кнопку "REMOVE" (Убрать).

Резервирование памяти под отладочную информацию

Пункт "SYMBOL BUFFER SIZE" определяет объем памяти (в Кб), резервируемой для хранения отладочной информации (например, номеров строк исходного кода). Под управлением Windows 95 эта область памяти также используется SoftICE в качестве буфера для хранения .NMS в процессе загрузки отладчика. По умолчанию SoftICE резервирует для буфера 1024 Кб памяти под Windows 95 и 512 Кб под Windows NT.

Как правило, 512 Кб достаточно как в том, так и в другом случае. Однако, иногда Вам может понадобиться увеличить размер буфера:

- Если Вы отлаживаете большие программы, используйте значение 1024 Кб или более.
- Если Вы работаете под Windows 95 и загружаете отладочную информацию при старте SoftICE, то подсчитайте общий размер всех используемых на этапе загрузки .NMS-файлов и установите величину буфера равной этому значению.

Чтобы определить количество памяти, доступной для хранения отладочной информации, используйте команду TABLE. Учтите, что основная часть этой информации находится в динамически выделяемой памяти.

Предварительная загрузка экспортируемой информации (Exports)

Начальные установки в вкладке "EXPORT" используются для назначения файлов, из которых SoftICE в момент старта сможет извлечь экспортируемую информацию. Экспортируемая информация может быть полезна, например, при работе с библиотекой DLL, когда отладочная информация вообще отсутствует.

Извлечение экспортируемой информации

Чтобы выбрать один или несколько файлов, из которых будет извлекаться экспортируемая информация сделайте следующее:

1 Нажмите кнопку "ADD" (Добавить).
SoftICE покажет диалог, чтобы Вы смогли найти необходимые файлы. Помните, что, если Вы подключены к сети, то Вы можете нажать кнопку "NETWORK" (Сеть), чтобы просмотреть содержимое доступных Вам сетевых устройств.
2 Выберите необходимые файлы и нажмите OK.
3 SoftICE поместит выбранные Вами файлы в список для экспорта.

Удаление файлов из списка экспорта

Чтобы удалить файл из списка экспорта, выберите требуемый файл и нажмите кнопку "REMOVE" (Убрать).

Настройка удаленной отладки (Remote Debugging)

Параметры удаленной отладки позволяют Вам определить тип последовательного соединения и задать строку инициализации модема и номер телефона, которые будут использоваться командами DIAL и ANSWER. (Вышеупомянутые параметры могут быть непосредственно заданы в этих командах.) За точным описанием команд управления Вашим модемом обращайтесь к соответствующей документации.

Telephone number (Номер телефона)

Параметр "TELEPHONE NUMBER" задает номер телефона, используемый командой DIAL (например, 421-555-1212).

Serial connection (Последовательное соединение) (только для Windows 95)

Если Вы запускаете SoftICE для отладки удаленной системы под управлением Windows 95, выберите на локальном компьютере коммуникационный порт (COM1, COM2, COM3 или COM4), который будет использован для последовательного соединения. После окончания отладки измените параметр на "NONE". По умолчанию параметр "SERIAL CONNECTION" установлен на "NONE".

Замечание: Если Вы используете SoftICE под управлением Windows NT, то отладчик автоматически определит параметры последовательного соединения.

DIAL initialization string (Строка инициализации модема командой DIAL)

Параметр "DIAL INITIALIZATION STRING" устанавливает строку инициализации модема при использовании команды DIAL, например ATX0.

ANSWER initialization string (Строка инициализации модема командой ANSWER)

Параметр "ANSWER INITIALIZATION STRING" устанавливает строку инициализации модема при использовании команды ANSWER, например ATX0.

Изменений назначений клавиатуре (Keyboard Mappings)

Раздел установок "KEYBOARD MAPPINGS" используется для назначения функциональным клавишам клавиатуры команд SoftICE. Команды SoftICE могут быть назначены любой из 12 функциональных клавиш или их комбинациям с клавишами SHIFT, CTRL или ALT.

Синтаксис команд

При модификации исходных или при создании новых назначений могут быть использованы любые допустимые команды SoftICE и символы вставки (^) и "точка с запятой" (;). Наличие символа вставки (^) перед началом команды указывает SoftICE выполнить ее, не помещая в окно команд. Символ "точка с запятой" (;) работает подобно клавише Enter, инициируя исполнение команды. В одной командной строке может находиться более одного символа "точка с запятой".

Изменение значений функциональных клавиш

Чтобы изменить команды SoftICE, присвоенные функциональной клавише, необходимо выполнить следующие действия:

1 Выберите функциональную клавишу, которую Вы хотите модифицировать, в списке назначений и нажмите кнопку "ADD" (Добавить).
Замечание: SoftICE использует следующие сокращения для функциональных клавиш и клавиш ALT, CTRL и SHIFT.

Клавиша	Сокращение	Пример
Функциональные	F	F1
Alt	A	AF1
Ctrl	C	CF1
Shift	S	SF1

2 Измените команду в поле "Command" и нажмите OK.

Присвоение значений незадействованным функциональным клавишам

Чтобы присвоить команду новой функциональной клавише (или комбинации клавиш), выполните следующие действия:

1 Определите функциональную клавишу (или комбинацию функциональной клавиши с SHIFT, CTRL или ALT), которой никаких команд SoftICE еще не присвоено.
2 Нажмите кнопку "ADD" (Добавить).
3 Выберите код функциональной клавиши из списка.
4 Выберите модификатор. Чтобы присвоить команду просто функциональной клавише, выберите "NONE". Для присвоении команды комбинации клавиш, выберите SHIFT, CTRL или ALT.
5 Наберите необходимую команду в поле "COMMAND" и нажмите OK.

Удаление назначения функциональной клавише

Чтобы удалить назначение команды функциональной клавише, выберите в списке необходимую комбинацию и нажмите "REMOVE" (Убрать).

Восстановление значений функциональных клавиш

В следующей таблице приведен список команд SoftICE, присваиваемых функциональным клавишам по умолчанию.

Клавиша	Команда	Клавиша	Команда
F1 =	H;	F11 =	^G @SS:ESP;
F2 =	^WR;	F12 =	^P RET;
F3 =	^SRC;	SF3 =	^FORMAT;
F4 =	^RS;	AF1 =	^WR;
F5 =	^X;	AF2 =	^WD;
F6 =	^EC;	AF3 =	^WC;
F7 =	^HERE;	AF4 =	^WW;
F8 =	^T;	AF5 =	CLS;
F9 =	^BPX;	AF11=	`dd dataaddr->0;`
F10 =	^P;	AF12=	`dd dataaddr->4;`

Вы можете модифицировать назначения отдельных клавиш или нажать кнопку "RESTORE DEFAULTS" (Восстановить исходные), чтобы восстановить всем модифицированным или удаленным функциональным клавишам их исходные значения. Однако, эта кнопка не удаляет команды, присвоенные новым функциональным клавишам.

Работа с постоянными макрокомандами (Macro Definitions)

Макрокоманды являются определяемыми пользователем командами, которые могут использоваться точно так же, как и встроенные команды. Определение (или тело) макрокоманды состоит из последовательности команд SoftICE. Набор допустимых команд включает в себя также пользовательские макросы и аргументы командной строки.

Существует 2 способа создания макрокоманд. Вы можете создать макрокоманды времени исполнения, которые существуют до момента перезапуска SoftICE, или создать постоянные макросы, которые сохраняются в файле начальных установок и автоматически загружаются каждый раз во время старта отладчика. В данном разделе описывается создание постоянных макрокоманд. Дополнительную информацию о создании макрокоманд исполнения времени смотрите в разделе "*Использование макрокоманд времени исполнения*" на странице 64.

Создание постоянных макрокоманд

Постоянные макрокоманды создаются следующим образом:

1 Нажмите "ADD" (Добавить).
  Появляется окно "Add Macro Definition" (Добавление определения макроса).
2 В поле "NAME" впишите имя макрокоманды.

Имя макроса может иметь длину от 3 до 8 символов и содержать любые буквы, цифры и знак подчеркивания (_). Имя обязательно должно содержать хотя бы одну букву. Имена макрокоманд не должны дублировать имена встроенных команд SoftICE.

3 Наберите содержимое макроса в поле "Definition" (Определение).

Определение (или тело) макроса состоит из последовательности команд SoftICE или других макросов, разделенных точками с запятой. При этом последняя команда не обязательно должна заканчиваться точкой с запятой. На аргументы командной строки макрокоманды можно ссылать в любом месте тела макроса с помощью %<*номер-параметр*>, где *номер-параметр* — это цифры от 1 до 8.

Пример: Макрокоманда MACRO asm = "a %1" определяет псевдоним для команды A (команда ассемблера). Идентификатор "%1" заменяется на первый аргумент после имени макроса "asm" или просто удаляется, если аргумента нет.

Если требуется вставить в тело макрокоманды символ двойных кавычек (") или знак процента (%), то перед ними необходимо добавить символ обратной косой черты (\). Чтобы добавить собственно знак обратной косой черты, используется последовательность из двух таких символов (\\).

Замечание: Рекурсивный вызов макрокоманд возможен, однако, польза от этого приема сомнительна, так как программного способа завершить его работу не существует. Если макрокоманда вызывает саму себя в последней команде тела макроса (так называемый "хвостовой вызов"), то такая команда выполняется до тех пор, пока пользователь не нажмет 'ESC', чтобы прервать ее работу. Если рекурсивный вызов не является последней командой макроса, то такой вызов выполняется 32 раза (ограничение вложенности).

4 Нажмите OK.

SoftICE поместит Вашу постоянную макрокоманду в список макроопределений.

Примеры макрокоманд

В следующей таблице приведены примеры допустимых макрокоманд.

Допустимое имя	Допустимое определение	Пример
`Qexp`	`addr explorer; Query %1`	`Qexp` `Qexp 140000`
`1shot`	`bpx %1 do \"bc bpindex\"`	`1shot eip` `1shot @esp`
`ddt`	`dd thread`	`ddt`
`ddp`	`dd process`	`ddp`
`thr`	`thread %1 tid`	`thr` `thr -x`
`dmyfile`	`macro myfile =` `\"TABLE %1;file \%1\"`	`dmyfile mytable` `myfile myfile.c`

В следующей таблице приведены примеры недопустимых макрокоманд.

Недопустимое имя или тело макроса	Пояснение
Имя: `DD` Тело: `dd dataaddr`	Макрокоманда использует имя встроенной команды SoftICE. Команды SoftICE не могут быть переопределены.
Имя: `AA` Тело: `addr %1`	Имя макрокоманды слишком короткое. Имя макроса должно содержать от 3 до 8 символов.
Имя: `tag` Тело: `? *(%2-4)`	Макрокоманда ссылается на параметр %2 без ссылки на параметр %1. Нельзя ссылаться на параметр %n + 1, не ссылаясь на параметр %n.

Запуск и остановка работы постоянных макрокоманд

Для того, чтобы исполнить постоянную макрокоманду, просто наберите ее имя. Чтобы остановить работу макроса, нажмите клавишу ESC.

Параметр Macro Limit

Параметр "MACRO LIMIT" определяет максимальное число макрокоманд и точек прерывания, которые Вы можете определить во время работы SoftICE. Это количество включает в себя как постоянные, так и макрокоманды времени исполнения. По умолчанию параметр установлен на минимальное значение — 32. Максимально допустимое значение равно 256.

Модификация постоянных макрокоманд

Чтобы модифицировать постоянную макрокоманду необходимо:

1 Выбрать макрокоманду, которую Вы хотите изменить и нажать "ADD".
2 В окне "ADD MACRO DEFINITION" (Добавление определения макроса) изменить имя и/или определение макрокоманды и затем нажать OK.

Удаление постоянной макрокоманды

Чтобы удалить постоянную макрокоманду, выберите соответствующий макрос и нажмите кнопку "REMOVE" (Удалить).

Параметры для устранения неисправностей (Troubleshooting)

Эти параметры позволяют Вам устранить некоторые проблемы при работе с SoftICE. Изменяйте их только в том случае, если Вы получили соответствующее указание от службы технической поддержки фирмы NuMega, или оказались в ситуации, описанной в данной документации. По умолчанию все параметры данного раздела установлены в состояние "Выключено".

Совет: Если Вы хотите вернуть все установки параметров данного раздела в исходное состояние, нажмите кнопку "RESTORE DEFAULTS" (Восстановить исходные значения). То же самое можно сделать, если Вы включили некоторые из параметров и теперь хотите все их выключить, не обращаясь к каждому из них по отдельности.

Disable mouse support (Отключить поддержку мыши)

Отметьте данный флажок, если у Вас возникли проблемы с использованием мыши при работе с SoftICE.

Disable Num Lock and Caps Lock programming (Отключить программирование клавиш Num Lock и Caps Lock)

Если при загрузке SoftICE у Вас блокируется клавиатура, или она ведет себя странно, то установите данный флажок. Если, однако, это не решит Вашей проблемы, но Вы работаете под Windows NT, попробуйте установить флаг "DO NOT PATCH KEYBOARD DRIVER".

Do not patch keyboard driver (Не использовать модификацию клавиатурного драйвера) (только для Windows NT)

Если при загрузке SoftICE у Вас блокируется клавиатура, или она ведет себя странно, установите данный флажок, чтобы не разрешать SoftICE модифицировать драйвер клавиатуры. В данном случае SoftICE будет использовать иной, обычно менее устойчивый способ управления клавиатурой. Если это не решит Вашей проблемы, попробуйте установить флаг "DISABLE NUM LOCK AND CAPS LOCK PROGRAMMING".

Disable mapping of non-present pages (Отключить отображение незагруженных страниц)

SoftICE пытается осуществить поиск страниц в физической памяти даже, если данная страница помечена как отсутствующая. Отметьте данный флажок, чтобы отключить эту функцию.

Disable Pentium support (Отключить поддержку процессоров "Pentium")

SoftICE автоматически определяет используете Вы процессор класса Pentium или нет. Если же Вы используете новый тип процессора, с которым SoftICE не знаком, а отладчик ошибочно определяет его как процессор Pentium, то установите данный флаг.

Disable thread-specific stepping (Отключение потоко-зависимой трассировки)

Команда P (steP over — пошаговая трассировка) является потоко-зависимой. Работа отлаживаемой программы будет прервана SoftICE только в том случае, если будет активным тот поток, в котором была выполнена команда P. Заметьте, что, как правило, Вам как раз и необходимо оказаться именно в том потоке, который Вы отлаживаете. Чтобы выключить такую зависимость, установите данный флаг.

— А король-то голый! — кричал маленький ребенок.
Ганс Христиан Андерсен

But the Emperor has nothing on at all! cried a little child.
Hans Christian Andersen

12. Исследование Windows NT

Обзор

Не вызывает никакого сомнения, что операционная система Windows NT — это великолепное произведение разработчиков и программистов. Трудно представить себе какую-либо иную столь же сложную систему, способную решать все поставленные перед нею задачи, включая и три самых непростых: переносимость, устойчивость и расширяемость, не поступаясь при этом ни интерфейсом, ни производительностью. Каким-то чудом разработчики фирмы Microsoft ухитрились обеспечить четкое взаимодействие всех компонентов, подобное работе механизма хорошо отлаженных часов. И, если Вы собираетесь разрабатывать приложения для Windows NT, Вам необходимо понимать то, что лежит в основе Вашей программы — саму операционную систему Windows NT. Знания, которые Вы приобретете, потратив время на изучение особенностей ее строения, обогатят как Вас, так и Ваши приложения.

В этой главе дается краткий обзор наиболее важных и интересных особенностей Windows NT. Особое внимание уделяется слабо или вообще недокументированным сторонам функционирования системы.

Ресурсы для квалифицированных разработчиков

Фирма Microsoft предоставляет некоторые дополнительные ресурсы для разработки и отладки приложений, среди которых отладочная версия системы, Windows NT DDK и .DBG-файлы. В следующих разделах все эти три ресурса обсуждаются подробнее.

Отладочная версия системы

Если Вы не используете в своей работе отладочную версию Windows NT, то значит Вы не получаете того значительного количества дополнительной информации и поддержки, которые она предоставляет, и которая отсутствует в коммерческой версии операционной системы. Вам будут доступны многочисленные отладочные сообщения, специальные флаги, используемые компонентами ядра и позволяющие проследить за работой системы, а также достаточно строгая проверка большинства API вызовов. Размеры и строение структур данных, а также организация системных вызовов в отладочной и коммерческой версиях практически идентичны. Это позволяет Вам изучить работу приложения, используя более информативные сообщения отладочной версии, а замет успешно завершить работу под управлением коммерческой версии Windows NT. Следовательно, если Вы хотите разрабатывать как можно более устойчивые приложения и драйверы, используйте отладочную версию.

Windows NT DDK

Windows NT DDK (Driver Development Kit — средства для разработки драйверов) содержит заголовочные файлы, примеры программ, систему подсказок, а также специальные программы, позволяющие исследовать различные составные части ядра системы. Самым полезным из них является файл NTDDK.H. Несмотря на то, что значительная часть информации в нем отсутствует, тем не менее его стоит изучить. Кроме основных структур данных, необходимых для разработки драйверов, в нем описаны и многие другие (одни подробно, другие кратко, а некоторые и не описаны вовсе). Здесь приведены прототипы многих API вызовов и перечислители, полезные как для исследования, так и для разработки. Имеется множество полезных комментариев об особенностях строения системы, ее возможностях и ограничениях. Большинство остальных заголовочных файлов касаются редко используемых сторон работы системы, из них наиболее полезны NTDEF.H, BUGCODES.H и NTSTATUS.H.

Windows NT DDK также включает в себя несколько полезных утилит. Например, POOLMON.EXE позволяет наблюдать за использованием системной памяти, OBJDIR.EXE предоставляет информацию об иерархии объектов в системе, и конкретно о каждом из них. Программа SoftICE для Windows NT позволяет получить аналогичные сведения с помощью команд **OBJDIR**, **DEVICE** и **DRIVER**. Утилита DRIVERS.EXE, подобно команде **MOD** SoftICE, перечисляет все драйверы и выдает их основные параметры. Некоторые версии Windows NT DDK включают расширенную версию стандартной утилиты PSTAT.EXE. PSTAT — это консольное приложение, которое позволяет получить информацию о процессах и потоках в системе. Среди включаемых в состав Win 32 SDK и Visual C++ утилит следует упомянуть две: PVIEW и SPY++. Обе предоставляют информацию о процессах и потоках системы; SPY++, кроме того, позволяет получать значения HWND и CLASS.

В Windows NT DDK включены HELP-файлы и справочный материал для разработки драйверов, а также примеры программ. Это примеры очень полезны при создании реальных драйверов. Найдите что-либо похожее на Вашу разработку, скомпилируйте образец и изучите его с помощью SoftICE.

.DBG файлы

Microsoft предоставляет .DBG-файлы всех исполняемых файлов как для отладочной, так и для коммерческой версий Windows NT. Сюда входят системные компоненты ядра, драйверы устройств, DLL системы Win32, приложения контрольной панели и даже игры. .DBG-файлы содержат основную отладочную информацию, подобную PUBLIC-разделам .MAP-файлов. Каждый API-вызов или глобальная переменная, как экспортируемая, так и импортируемая, имеют свое определение (например, имя, секцию и смещение). Информация о структурах данных или локальных переменных в них отсутствует, однако описание каждого API делает отладку системных вызовов более легкой.

Совет: .DBG-файлы являются наиболее важным инструментом в Вашей отладочной и исследовательской работе. Определите необходимые Вам ключевые компоненты, найдите соответствующие им .DBG-файлы, и с помощью программы Symbol Loader создайте .NMS-файлы, которые можно будет загрузить в SoftICE.

Независимо от специфики Вашей работы используйте отладочную информацию для ключевых компонентов системы. Наиболее важные их них отмечены в таблице жирным шрифтом.

NTOSKRNL.EXE	Ядро Windows NT. Основная часть операционной системы реализована именно здесь.
HAL.DLL	Уровень аппаратной абстракции (Hardware Abstraction Layer). Важные примитивы для NTOSKRNL.
NTDLL.DLL	Основа реализации Win32 API и других функций, традиционно приписываемая KERNEL. Здесь также реализован интерфейс между Системой и Пользователем. Сюда переадресуются вызовы из KERNEL32.DLL.
CSRSS.EXE	Сервер подсистемы Win32. Большинство системных вызовов проходят через этот процесс.
WINSRV.DLL	В версии Windows NT 3.51 — ядро реализации функций USER и GDI. Единственный процесс, загруженный в контекст CSRSS.
WIN32K.SYS	Драйвер системных устройств, замещающий WINSRV.DLL и минимизирующий межпроцессовые взаимодействия между приложениями и CSRSS. В некоторых версиях операционной системы может быть отсутствовать.
USER32.DLL	Основа реализации функций USER. Содержит, главным образом, переадресацию в WINSRV.DLL (через LPC к CSRSS). В последних версиях сюда перенесена реализация большего числа функций для уменьшения количества переключений контекста.
KERNEL32.DLL	Реализация некоторых традиционных функций KERNEL, однако в основном содержит переадресацию в NTDLL.DLL.

Другие источники информации

Следующие источники предоставляют обширную информацию по разработке драйверов и приложений для Windows NT.

- *Microsoft Developers Network* (MSDN)
  MSDN выходит ежеквартально на CD-ROM и содержит много полезной информации, а также статьи по всем аспектам программирования для семейства операционных систем фирмы Microsoft. Это одно из немногих мест, где Вы можете найти подробности по написанию драйверов устройств для Windows NT.
- *Inside Windows NT* - Helen Custer, Microsoft Press[†]
  В книге "*Inside Windows NT*" сначала рассматривается общее строение операционной системы, а затем в деталях обсуждается каждая достаточно важная подсистема и приводится множество диаграмм для иллюстрации внутренних структур данных, правил и алгоритмов работы. Несмотря на то, что содержание этой книги может показаться достаточно абстрактным с точки зрения разработки приложений, после ее прочтения взаимосвязи систем Windows NT становятся более понятными. В настоящее время это наиболее ценный источник информации по строению и функционированию Windows NT. И еще большую пользу Вы сможете извлечь, если параллельно будете исследовать внутреннее устройство операционной системы с помощью SoftICE.

[†] Имеется русский перевод: Хелен Кастер "Основы Windows NT и NTFS". — Microsoft Press "Русская редакция", 1996 г.

- *Advanced Windows*, 2nd Edition - Jeffery Richter, Microsoft Press[†]
  Книга "*Advanced Windows*" является ценным пособием для программистов, разрабатывающих приложения для систем Win32. Автор подробно обсуждает процессы, потоки, управление памятью и синхронизацию объектов. Приводятся образцы кодов и утилиты.

Внутри ядра Windows NT

Чтобы получить общее представление о работе Windows NT, давайте взглянем на нее с разных точек зрения. Это позволит понять ограничения и допущения, сделанные при ее разработке.

Данный раздел касается наиболее важного компонента операционной системы — его ядра. Здесь рассказывается, как Windows NT заполняет основные структуры ядра, такие как IDT и TSS, и каким образом можно использовать команды SoftICE, чтобы оценить работу системы. Будет показано распределение системных областей памяти, описаны важные структуры данных и роль, которую они играют в работе операционной системы.

Большая часть этого раздела строится на деталях организации двух следующих модулей:

- HAL.DLL — Уровень аппаратной абстракции (Hardware Abstraction Layer).
  Роль этого модуля состоит в изоляции в пределах одного модуля аппаратную зависимость системы настолько, насколько это только возможно, что делает код Windows NT легко переносимым на другие платформы. Однако, с целью повышения производительности аппаратно-зависимые коды присутствуют и в других частях ядра.
  Главным образом, HAL отвечает за работу с устройствами на самом низком уровне: за программирование контроллера прерывание, обслуживание портов ввода/вывода и взаимодействие процессоров в мультипроцессорных системах. Многие программы HAL предназначены для работы с конкретным типом шины (PCI, EISA, ISA), с различными шинными адаптерами. HAL также отвечает за первичную обработку ошибок и управление прерываниями.
- NTOSKRNL.EXE — собственно ядро системы.
  Основная часть операционной системы Windows NT реализована в ядре или Windows NT Executive. Именно на нем базируется работа всех остальных подсистем, такой, например, как Win32. Ядро выполняет множество различных функций, а именно:
  ◊ управление памятью;
  ◊ управление объектами;
  ◊ создание процессов и потоков и управление их работой;
  ◊ обеспечение протокола LPC (Local Procedure Call — местный вызов процедуры);
  ◊ обеспечение системы безопасности;
  ◊ обработка исключений;
  ◊ функционирование виртуальных DOS-машин (VDM);
  ◊ примитивы синхронизации — семафоры и взаимоисключения;
  ◊ библиотека времени исполнения;
  ◊ файловая система;
  ◊ подсистема ввода/вывода.

[†] В 1998 г. на русском языке вышло 3-е издание книги: Джеффри Рихтер "Windows для профессионалов". — Microsoft Press "Русская редакция".

Управление процессорами Intel

При запуске операционной системы в защищенном режиме необходимо заполнить основные структуры процессора, установить параметры работы и выделить адресное пространство, которое она будет использовать. Инициализация системы выполняется модулями NTLDR, NTDETECT, NTOSKRNL и HAL. Применение перечисленных в приведенной ниже таблице команд SoftICE поможет Вам понять, каким образом Windows NT использует архитектуру процессоров x86 для обеспечения безопасной и устойчивой работы.

Команда	Описание
IDT	Показывает содержимое таблицы дескрипторов прерывания (Interrupt Descriptor Table)
TSS	Показывает содержимое сегмента состояния задачи (Task State Segment)
GDT	Показывает содержимое глобальной таблицы дескрипторов (Global Descriptor Table)
LDT	Показывает содержимое локальной таблицы дескрипторов (Local Descriptor Table)

Замечание: Подробное описание каждой из этих команд дается в "*Справочнике по командам SoftICE*".

IDT (Interrupt Descriptor Table — таблица дескрипторов прерываний)

Windows NT создает таблицу IDT для 256 векторов прерываний и размещает ее в системной области адресного пространства. Первые 48 векторов обычно используются ядром для перехвата исключений, однако некоторые вектора используются для системных нужд или приложений. Для получения содержимого таблицы дескрипторов прерываний Windows NT в SoftICE следует воспользоваться командой IDT.

Прерывание	Назначение
2h	Немаскируемое прерывание. Установленный здесь вентиль задачи имеет чистый набор регистров, таблиц страниц и стек уровня 0. Это позволяет операционной системе продолжить работу и успеть вывести "Синий экран смерти".
8h	Двойная ошибка. Установленный здесь вентиль задачи имеет чистый набор регистров, таблиц страниц и стек уровня 0. Это позволяет операционной системе продолжить работу и успеть вывести "Синий экран смерти".
21h	Перехват 21h прерывания MS-DOS. Используется только виртуальной DOS-машиной или подсистемой "Windows on windows".
2Ah	Сервис для получения текущего значения системных часов.
2Bh,2Ch	Сервис прямого переключения цепочек.
2Dh	Отладка.
2Eh	Выполнить системный сервис. Используется при передаче управления в Windows NT с пользовательского на системный уровень. За дополнительной информацией обращайтесь к описанию команды NTCALL в "*Справочнике по командам SoftICE*".

Продолжение на следующей странице

Прерывание	Назначение
30h-37h	Первичный контроллер прерываний (IRQ0-IRQ7). 30h - прерывание от системных часов (IRQ0).
38h-3Fh	Вторичный контроллер прерываний (IRQ8-IRQ15).

Векторы прерываний 30h-3Fh назначены первичному и вторичному контроллерам прерываний, следовательно аппаратные прерывания IRQ0-IRQ15 тоже проходят через IDT. Как правило, эти прерывания не перехватываются, а система передает их в стандартные программы-заглушки. Если требуется перехват этих аппаратных прерываний, то необходимо подключить специальные драйверы, а после работы отключить их.

Стандартные программы-заглушки имеют имена KiUnexpectedInterrupt#, где # соответствует номеру прерывания. Чтобы определить, какой вектор присвоен данной заглушке следует прибавить 30h к интересуемому номеру. Например KiUnexpectedInterrupt2 подключен к вектору 32h (30h + 2) таблицы прерываний IDT.

Прерывания виртуальной DOS-машины, которая включает в себя и подсистему WOW (Windows on Windows), непосредственно через IDT не передаются, а эмулируются генерацией общей ошибки защиты (GPF), которую анализирует специальная программа в составе NTOSKRNL. В большинстве случает прерывания возвращается обратно в VDM для дальнейшей обработки. Прерывание 21h MS-DOS обрабатывается особым образом — для него существует нормальный дескриптор в IDT. Это было сделано либо по соображениям производительности, либо для совместимости программ (а, возможно, и по обоим причинам).

Драйверы могут подключать и отключать обработчики прерываний по мере необходимости, используя системные вызовы IoConnectInterrrupt и IoDisconnectInterrupt. Эти вызовы создают специальные переходники, содержащие данные и код, расположенные в невыгружаемой области памяти (Non-Pageable Pool), чтобы обеспечить одновременное использование одного прерывания несколькими драйверами.

TSS (Task State Segment — Сегмент состояния задачи)

TSS предназначен для сохранения состояния процессора при переключении задач или контекста. По соображениям производительности Windows NT не использует эту архитектурную особенность процессора, а поддерживает единственный TSS, общий для всех процессов. Как было отмечено в предыдущем разделе, касающемся IDT, в Windows NT существуют и другие TSS, однако они используются только в исключительных случаях, чтобы обезопасить систему от перезарузки раньше, чем она сможет соответствующим образом это исключение обработать. Для получения содержимого текущего сегмента состояния задачи в SoftICE используется команда TSS.

TSS имеет специальное поле, в котором содержится смещение начала карты ввода/вывода относительно начала сегмента. Карта ввода/вывода определяет, к каким портам, если таковые вообще существуют, возможен доступ для программы, исполняемой в 3 кольце защиты. При работе виртуальной DOS-машины под управлением Windows NT 3.51 TSS содержит действительное значение смещения начала карты ввода/вывода, которая перехватывает обращения к портам для последующей эмуляции их операционной системой. Во время работы приложений Win32 TSS содержит смещение, указывающее за пределы сегмента, что позволяет операционной системе перехватывать все обращения к портам ввода/вывода.

В составе структуры TSS лишь одно поле представляет реальный интерес — адрес стека для нулевого кольца защиты. Этот стек используется при переходе с пользовательского уровня (кольцо 3) на системный.

GDT (Global Descriptor Table — глобальная таблица дескрипторов)

Windows NT использует непрерывную (flat) 32-битную модель адресации. Хотя применение селекторов все еще необходимо, использует она их крайне ограниченно. Большинство приложений Win32 и драйверов вообще не знают об их существовании. Ниже приведен фрагмент вывода команды GDT, который демонстрирует, какие селекторы содержит глобальная таблица.

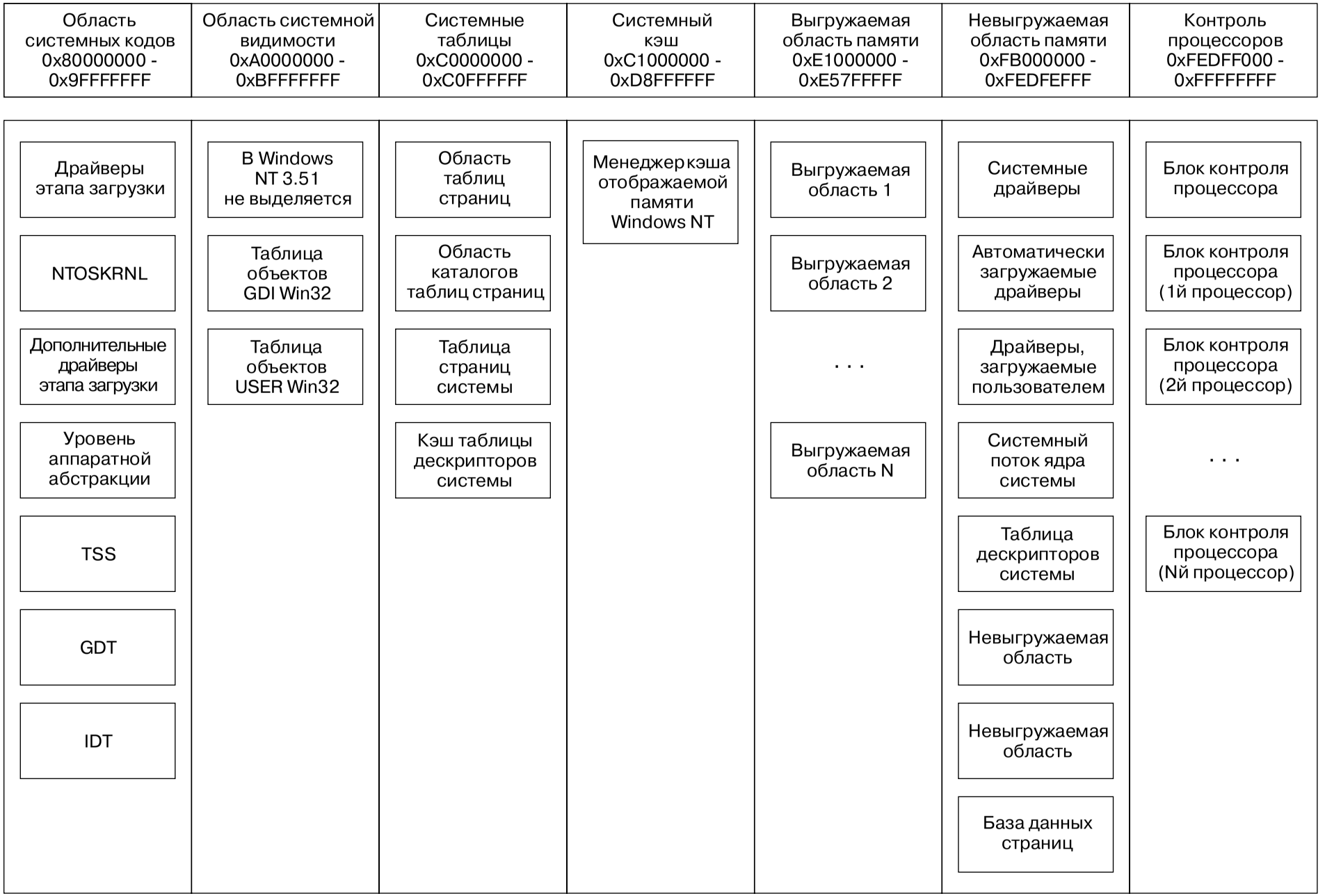
```
GDTbase=80036000                Limit=03FF
0008    Code32   Base=00000000   Lim=FFFFFFFF   DPL=0   P   RE
0010    Data32   Base=00000000   Lim=FFFFFFFF   DPL=0   P   RW
001B    Code32   Base=00000000   Lim=FFFFFFFF   DPL=3   P   RE
0023    Data32   Base=00000000   Lim=FFFFFFFF   DPL=3   P   RW
0028    TSS32    Base=8000B000   Lim=000020AB   DPL=0   P   B
0030    Data32   Base=FFDFF000   Lim=00001FFF   DPL=0   P   RW
003B    Data32   Base=7FFDE000   Lim=00000FFF   DPL=3   P   RW
0043    Data16   Base=00000400   Lim=0000FFFF   DPL=3   P   RW
0048    LDT      Base=E156C000   Lim=0000FFEF   DPL=0   P
0050    TSS32    Base=80143FE0   Lim=00000068   DPL=0   P
0058    TSS32    Base=80144048   Lim=00000068   DPL=0   P
```

Обратите внимание, что первые 4 селектора адресуют весь 4 Гб диапазон памяти. Эти селекторы используются драйверами и приложениями Win32. Первые два селектора имеют DPL, равный нулю, и используются драйверами и системными компонентами для обращения к системным кодам, данным и стекам. Селекторы 1Bh и 23h предназначены для приложений Win32 и адресуют коды, данные и стеки пользовательского уровня; фактически это константы, и Windows NT часто обращается к ним, используя их абсолютные значения.

Селектор 30h адресует область контроля процессора ядра (Kernel Processor Control Region) и всегда имеет на базовый адрес 0xFFDFF000. При выполнении системного кода этот селектор хранится в сегментном регистре FS. Кроме множества других функций, эта область содержит по смещению 0 текущий кадр обработки исключений ядра.

Аналогично, селектор 3Bh является селектором пользовательского уровня, который адресует текущий блок переменных окружения потока (User Thread Environment Block - UTEB). Этот селектор сохраняется в регистре FS при исполнении кода пользовательского уровня и содержит текущий кадр обработки исключений пользовательского уровня по смещению 0. Базовый адрес этого селектора изменяется в зависимости от того, какой поток выполняется в настоящий момент. Когда происходит переключение потоков, базовый адрес этого селектора в GDT изменяется, чтобы указать на новый UTEB.

Селектор 48h является селектором LDT и используется только процессами VDM. Приложения Win32 и драйверы не используют LDT. Во время исполнения процесса Win32 регистр LDT процессора Intel содержит NULL. В этом случае команда LDT SoftICE выдаст сообщение об ошибке "No LDT" (LDT отсутствует). При выполнении VDM- или WOW-процесса действительное значение регистра LDT устанавливается из этого селектора GDT. Во время переключения контекста процесса содержимое LDT восстанавливается в этот селектор из блока окружения процесса ядра (Kernel Process Environment Block - KPEB), для установки правильных значений базового адреса и предела.


Область системных кодов 0x80000000 - 0x9FFFFFFF
Область системной видимости 0xA0000000 - 0xBFFFFFFF
Системные таблицы 0xC0000000 - 0xC0FFFFFF
Системный кэш 0xC1000000 - 0xD8FFFFFF
Выгружаемая область памяти 0xE1000000 - 0xE57FFFFF
Невыгружаемая область памяти 0xFB000000 - 0xFEDFEFFF
Контроль процессоров 0xFEDFF000 - 0xFFFFFFFF
Драйверы этапа загрузки
NTOSKRNL
Дополнительные драйверы этапа загрузки
Уровень аппаратной абстракции
TSS
GDT
IDT
В Windows NT 3.51 не выделяется
Таблица объектов GDI Win32
Таблица объектов USER Win32
Область таблиц страниц
Область каталогов таблиц страниц
Таблица страниц системы
Кэш таблицы дескрипторов системы
Менеджер кэша отображаемой памяти Windows NT
Выгружаемая область 1
Выгружаемая область 2
. . .
Выгружаемая область N
Системные драйверы
Автоматически загружаемые драйверы
Драйверы, загружаемые пользователем
Системный поток ядра системы
Таблица дескрипторов системы
Невыгружаемая область
Невыгружаемая область
База данных страниц
Блок контроля процессора
Блок контроля процессора (1й процессор)
Блок контроля процессора (2й процессор)
. . .
Блок контроля процессора (Nй процессор)

LDT (Local Descriptor Table — локальная таблица дескрипторов)

Под управлением Windows NT локальные таблицы дескрипторов используются только в виртуальных DOS-машинах (VDM — Virtual DOS Machines) и создаются для каждой структуры данных процесса. 16-битная WOW-подсистема исполняется в рамках NTVDM процесса и также имеет LDT. Подобно Windows 3.1 LDT для WOW содержит селекторы для каждого 16-битного исполняемого кода защищенного режима и для сегментов данных каждого загруженного 16-битного приложения или DLL. Также она содержит селекторы каждой базы данных задач, базы данных модулей, каждой локальной кучи и блока памяти, выделенного в глобальных областях, и для всех USER и GDI-объектов, требующих создания селектора. Так как при работе WOW количество необходимых селекторов может оказаться достаточно большим, создается LDT максимального размера, основная масса элементов в которой зарезервирована для дальнейшего использования. Эти селекторы выделяются по мере необходимости. При работе обычных (не-WOW) VDM размер LDT значительно меньше.

Распределение памяти в Windows NT

Windows NT резервирует старшие 2 Гб адресного пространства для системного использования. В область с адресами 0x80000000 - 0xFFFFFFFF загружаются драйверы устройств, системные переменные, таблицы и структуры данных потоков и процессов. Хотя точную карту распределения адресов системной памяти составить невозможно, однако можно в целом охарактеризовать различные области, выделяемые для определенных нужд. На следующей диаграмме представлены общие принципы размещения системных областей. Помните, однако, что здесь приведены лишь приблизительные диапазоны адресов, так как большинство областей могут находиться в любом месте адресного пространства.

На диаграмме на странице 136 представлена общая схема распределения системной памяти Windows NT.

- Область системных кодов
  В эту область записываются драйверы этапа загрузки, NTOSKRNL и компоненты уровня аппаратной абстракции (HAL). Остальные драйверы размещаются в невыгружаемой области системных адресов, находящейся в верхней части памяти. С помощью команд MOD и MAP32 SoftICE можно получить адреса и протяженность блока загрузочных драйверов. В этой же области находятся такие структуры как TSS, IDT и GDT.
  Замечание: LDT создаются в выгружаемой области.
- Область системной видимости
  Область системной видимости упоминается в Windows NT версии 3.51 но, по-видимому, никогда в ней не используется. В последующих версиях Windows NT в этой области размещаются глобальные таблицы объектов GDI и USER. С помощью команды OBJTAB можно получить информацию о таблице объектов USER.
- Область системных таблиц
  Эта область памяти содержит таблицы страниц процесса и соответствующие структуры данных. Это одна из немногих областей системной памяти, которая фактически не является глобальной, так как каждый процесс имеет свои уникальные таблицы страниц. Когда происходит переключение контекста процесса, физический адрес каталога таблиц страниц извлекается из блока переменных окружения процесса ядра (Kernel Process Environment Block - KPEB) и загружается в регистр CR3. Это приводит к переадресации соответствующих блоков памяти на нужную область. Несмотря на то, что линейные

адреса остаются теми же самыми, используемые физические адреса принадлежат именно данному процессу. В терминологии SoftICE каталогу таблиц страниц соответствует контекст адреса. Когда Вы пользуетесь командой ADDR для установления соответствующего процессу контекста, Вы фактически загружаете информацию каталога таблиц страниц данного процесса.

Чтобы выполнять отображение линейной памяти в физическую, Windows NT резервирует в линейной системной области блок размером 4 Мб. Эти 4 Мб используются для создания каталога таблиц страниц и полного набора таблиц страниц. Необходимость выделения именно 4 Мб можно проверить, исходя из того, что существует только один каталог таблиц страниц, содержащий 1024 дескриптора. Чтобы отобразить 4 Гб линейного адресного пространства, каждая таблица страниц должна содержать информацию о 4 Мб линейного адресного пространства (4 Гб/1024). С другой стороны каждая таблица страниц содержит информацию о множестве физических страниц, которые в Windows NT имеют размер 4 Кб, поэтому, умножая 1024 на 4096 (размер страницы физической памяти), получаем запрашиваемые 4 Мб. Следовательно, чтобы операционная система могла использовать виртуальную память и физические страницы по 4 Кб, необходимо выделить 4 Мб для отображения всего адресного пространства. Как Windows NT, так и Windows 95 применяют простой и эффективный метод, выделяя для этой цели непрерывный блок линейной памяти.

В данной схеме каталог таблиц страниц фактически выполняет две функции. Помимо того, что он выполняет свою прямую функцию, он еще является и таблицей страниц, представляющей область размеров 4 Мб с адресами 0xC0000000 - 0xC03FFFFF, поэтому дескриптор, отображающий эту область должен указывать сам на себя. Если выполнить команду PAGE SoftICE, то в верхней части листинга вывода будет показан физический адрес, в который дескриптор отображает блок с адресами 0xC0000000 - 0xC03FFFFF. Если же выполнить команду ADDR, чтобы получить содержимое регистра CR3 (физический адрес каталога таблиц страниц) и передать его в качестве параметра команде PHYS, то будут показаны все линейные адреса, которые отображаются на физический адрес каталога таблиц страниц. Одним из этих адресов является адрес 0xC0300000.

Следующие примеры иллюстрируют описанные выше взаимосвязи. Наиболее важные значения выделены жирным шрифтом.

◊ Применение команды ADDR для получения *физического* адреса каталога таблиц страниц (содержимое регистра CR3).

```
:addr
 CR3         LDT Base:Limit   KPEB Addr    PID      Name
 00030000                     FF116020     0002     System
 0115A000                     FF0AAA80     0051     RpcSs
 0073B000                     FF083020     004E     nddeagnt
 00653000    E13BB000:0C3F    FF080020     0061     ntvdm
 00AEE000                     FF07A600     0069     Explorer
 01084000                     FF06ECA0     0077     FINDFAST
 010E9000                     FF06CDE0     007B     MSOFFICE
*01F6E000                     FF088C60     006A     WINWORD
 01E0A000                     FF09CCA0     008B     4NT
 017D3000    E1541000:018F    FF09C560     006D     ntvdm
 00030000                     80140BA0     0000     Idle
```

◊ Использование физического адреса в качестве параметра команды PHYS для получения всех линейных адресов, отображаемых в данный физический адрес (одна физическая страница может быть использована для отображения более чем одного линейного адреса, и один линейный адрес может быть отображен в разные физические страницы).

```
:phys 1F6E000
C0300000
```

◊ Использование линейного адреса (0xC0300000) с командой PAGE, чтобы проверить соответствие физической страницы данному линейному адресу.

```
:page C0300000
Linear Physical Attributes
C0300000 01F6E000 P D A S RW
```

◊ Использование команды PAGE без параметров, чтобы увидеть отображение всего блока линейных адресов. Это полезно для получения физического адреса каталога таблиц страниц и проверки того факта, что таблица страниц операционной системы отображается в линейный адрес 0xC0000000 (приводится сокращенный вывод).

```
:page
Page Directory    Physical=01F6E000
Physical          Attributes          Linear Address Range
01358000          P   A S RW          A0000000 - A03FFFFF
017F0000          P   A S RW          A0400000 - A07FFFFF
01727000          P   A S RW          A0800000 - A0BFFFFF
01F6E000          P   A S RW          C0000000 - C03FFFFF
0066F000          P   A S RW          C0400000 - C07FFFFF
00041000          P   A S RW          C0C00000 - C0FFFFFF
00042000          P   A S RW          C1000000 - C13FFFFF
```

Дескрипторы системной таблицы страниц (Page Table Entries) и виртуальные дескрипторы

Сокращение PTE, которое появлялось во многих местах диаграммы распределения системной памяти, обозначает Page Table Entry — Дескриптор Таблицы Страниц. PTE является одним из 1024 дескрипторов, содержащихся в таблице страниц. Каждый такой дескриптор описывает одну страницу памяти, включая ее физический адрес и атрибуты. Так как Windows NT может выполняться не только на процессорах фирма Intel, и так как операционной системе может понадобиться защищать страницы памяти на таком уровне, который данным процессором не поддерживается, Windows NT создает так называемые виртуальные PTE или виртуальные дескрипторы. По свой структуре они подобны дескрипторам фирмы Intel, однако включают в себя и некоторые дополнительные атрибуты. Перегружая значения атрибутов, операционная система может контролировать состояние "ошибка страницы" и затем проверить дополнительные атрибуты соответствующего виртуального дескриптора, чтобы определить причину его возникновения. Манипулирование виртуальными дескрипторами и трансляция их значений в действительные дескрипторы процессора выполняется в NTOSKRNL. Обратите внимание, что операционная система постоянно проверяет соответствие значений виртуальных дескрипторов физическим. Это позволяет не допустить прямое вмешательство в содержимое дескрипторов таблиц страниц со стороны приложений или драйверов устройств.

- Область выгружаемой памяти (Paged Pool Area)

Выгружаемая системная память представляет собой область, где функция ntoskrnl!ExAllocatePool (и другие связанные с ней функции) размещает блоки, которые при необходимости могут быть выгружены на диск. Она представляет собой прямую противоположность невыгружаемой системной памяти. Блоки, размещенные невыгружаемой области, постоянно находятся в памяти и предназначены для таких структур, как обработчики прерываний, которые требуют высокой производительности или гарантии постоянной доступности для использования.

Операционная система активно использует данную область, так как именно здесь создается большинство системных объектов. Отметьте для себя, что начальный адрес области, ее размер и количество страниц определяется динамически во время инициализации. Приведенные ранее значения адресов являются лишь приблизительными. Для того, что узнать точные значения, загрузите отладочную информацию для NTOSKRNL и проверьте соответствующие переменные, которые описывают конфигурацию выгружаемой области. (Некоторые из них можно просмотреть, выдав команду SYM с параметром "MmPaged*".)
Несмотря на то, что существует только одна область выгружаемой памяти, количество размещенных в ней блоков, определяемое во время инициализации системы, может быть велико. С одной стороны, такая работа должна выполняться чрезвычайно аккуратно, но, с другой стороны, так как, размещение блоков происходит достаточно часто, наличие единственной структуры данных, владеть которой в момент выделения или возвращения памяти может только один поток, создает в системе узкое место. Чтобы избежать потери производительности, в данной области образовано несколько регионов, каждая с собственными данными, с собственной структурой описания и собственным семафором для синхронизации потоков. Это позволяет нескольким потокам одновременно производить операции с памятью. Если вы решите реализовать аналогичную технику в своем приложении, то учтите, что накладные расходы в данном случае чрезвычайно малы, и в системе достаточно наличия 4-5 регионов. Однако, прежде чем реализовывать эту схему, следует точно определить наличие узкого места, чтобы не заниматься решением несуществующей проблемы.

- Невыгружаемая системная область (NonPaged Pool Area)
  Эта область линейных адресов предназначена для системных компонентов и структур данных, которые должны находиться в памяти постоянно. Сюда включаются драйверы, стеки потоков ядра, 2 региона для выделения блоков памяти и база данных страниц. В противоречие тому, что было сказано ранее о том, что страницы данной области не могут быть выгружены из памяти, иногда это все-таки возможно. В частности, это касается стеков потоков ядра и пространства адресов процесса, которые часто могут отсутствовать.
  Данная область во многом сходна с областью выгружаемой памяти, за исключением того, что размещенные здесь объекты не могут быть выгружены ни при каких условиях. Выделяемая здесь память используется для размещения ключевых структур системы, таких как процесс ядра и блоки окружения потоков. Имеется также еще один регион, используемый для размещения объектов, которые *должны быть всегда доступны*. Для этого во время инициализации системы NTOSKRNL резервирует небольшое количество физической памяти. Размер блока выделяемой здесь памяти должен быть меньше одной страницы (4 Кб). Если запрос на выделение памяти в этом регионе не может быть выполнен, или если запрашивается более 4 Кб, система выводит "синий экран".
- Область контроля процессора (Processor Control Region)
  В самой верхней части системных адресов находится область контроля процессора. Здесь Windows NT размещает блоки контроля процессора (Processor Control Block, PCRB) — структуры данных для каждого процессора в системе, и глобальную структуру — область контроля процессора (Processor Control Region), отражающую текущее состояние системы. В последней содержится такая ключевая информация, как выполняемый в настоящий момент поток ядра, текущий уровень запроса прерывания, текущий кадр обработки структурных исключений, базовые адреса IDT, TSS и GDT, и указатели стеков цепочек ядра системы. Небольшая часть структур PCR и PCRB документирована в файле NTDDK.H.

В большинстве случаев авторам драйверов устройств необходимо знать текущее прерывание в системе, на котором они будут исполняться. Несмотря на то, что его можно узнать, проанализировав содержимое области контроля процессора по смещению 24h, значительно проще получить ее с помощью команды IRQL:

```
? IRQL

00000002h
```

Наиболее полезной информацией из блока контроля процессора является указатель текущего потока ядра. Он находится в блоке контроля процессора по смещению 04h, однако чаще всего на него ссылаются через PCR по смещению 124h. Это возможно по той причине, что блок процессора вложен в PCR по смещению 0x120. Код, необходимый для получения указателя текущей цепочки, выглядит следующим образом:

```
mov reg,FS:[124]
```

Вспомните, что при работе в системном режиме, регистр FS содержит селектор из GDT, чей базовый адрес указывает на начало области контроля процессора. SoftICE позволяет получить эту информацию более простым способом, используя встроенные функции *thread* или *tid*:

```
? thread

FF088E90h

? tid

71h
```

Более подробную информацию о текущем потоке можно получить, используя следующие команды:

```
:thread tid

TID     Krnl TEB    StackBtm   StkTop     StackPtr    User TEB    Process(Id)
0071    FF0889E0    FC42A000   FC430000   FC42FE5C    7FFDE000    WINWORD(6A)
:thread thread

TID     Krnl TEB    StackBtm   StkTop     StackPtr    User TEB    Process(Id)
0071    FF0889E0    FC42A000   FC430000   FC42FE5C    7FFDE000    WINWORD(6A)
```

Текущий процесс ни в PCR, ни в PCRB не сохраняется. Windows NT получает указатель текущего процесса через текущий поток. Пример кода для получения указателя текущего процесса приведен ниже:

```
mov eax, FS:[124]   ; получить текущий поток (KTEB)
mov esi, [eax+40h] ; получить указатель процесса потока (KPEB)
```

Подсистема Win32

Внутри CSRSS†

Процесс CSRSS (Client Server Runtime Subsystem — подсистема "клиент-сервер" времени исполнения) подсистемы Win32 интегрирует в себе Win32 API. Этот API выполняет различные функции, включая и те, которые традиционно приписываются таким компонентам Windows, как KERNEL, USER и GDI. Хотя эти стандартные модули присутствуют в Windows NT 3.51 (и в меньшей степени в последующих вер-

† Все описанное ниже относится, главным образом, к Windows NT 3.51. В v.4.0 эта подсистема претерпела значительные изменения.

сиях) в виде 32-битных DLL, основная часть функций на самом деле реализована в WINSRV.DLL. Вызовы функций, традиционно ассоциирующиеся с одним из этих стандартных компонентов Windows, фактически выполнены в них в виде переадресации к другим модулям, например NTDLL.DLL или для выполнения обслуживания используют межпроцессовые взаимодействия с CSRSS.

Большинство API вызовов USER и GDI проходят через соответствующий 32-битый модуль в адресной пространство процесса. Там они переводятся в форму сообщений местных процедурных вызовов (Local Procedure Call — LPC) и переправляются в CSRSS для дальнейшего выполнения. Легко представить себе, насколько этот механизм, хотя и значительно более оптимизированный для данных условий, чем механизм удаленных процедурных вызовов (Remote Procedure Call — RPC), может снизить производительность системы по сравнению с простым вывозом функций. Каждый раз, когда Вы вызываете функцию IsWindows из USER.DLL, этот вызов должен упаковываться в LPC и пересылаться как системное сообщение в CSRSS. Для того, чтобы CSRSS смог обработать это сообщение, должно произойти переключение контекста, должен быть определен необходимый сервис, проверена допустимость параметров, и лишь затем может быть выполнено требуемое действие. После его завершения на стороне CSRSS необходимо составить LPC-ответ клиенту (Вашему приложению), что включает в себя новое переключение контекста и распаковку LPC-ответа. Уф-ф! И все это необходимо проделать лишь для того, чтобы определить, описывает ли переданный дескриптор (handle) реально существующее окно или нет.

При проектировании последующих версий Windows NT разработчики фирмы Microsoft постаралась, на сколько это оказалось возможным, избежать подобных накладных расходов. Во-первых, они перенесли большую часть исполняемых кодов из WINSRV.DLL в модули USER32 и GDI32, которые работают непосредственно в адресном пространстве Вашего приложения. Это позволяет запускать наиболее часто используемые процедуры с помощью обычных вызовов, не прибегая к LPC. Во-вторых, чтобы исключить переключение контекста в CSRSS для вызова процедур в WINSRV.DLL, новый системный драйвер WIN32K.SYS обеспечивает более быстрое выполнение процедур из модулей USER и GDI с помощью простого перехода из пользовательского режима в системный. Использование WIN32K.SYS в качестве драйвера, предоставляющего сервис на прикладном уровне, позволяет Windows NT поддерживать высокий уровень инкапсуляции и обеспечивает большую устойчивость, формируя значительно более эффективную псевдо-"клиент-серверную" архитектуру.

Хотя CSRSS выполняется как отдельный процесс, тем не менее он оказывает большое воздействие на адресное пространство любого приложения Win32. Если для Вашего процесса выполнить команду HEAP32, то Вы обнаружите по крайней мере 2 кучи, которые он не создавал, но которые ему принадлежат. Первая создается при инициализации процесса и является кучей по умолчанию. Вторая создается непосредственно CSRSS. Могут существовать и другие не создававшиеся приложением кучи, но тем не менее находящиеся в его адресном пространстве. Как правило, они расположены в верхней части пользовательского адресного пространства, и их можно обнаружить, выполнив команду QUERY, и, при этом, они не выявляются командой HEAP32. Причина такого поведения достаточно проста: для каждого пользовательского процесса создается список куч, а команда HEAP32 использует этот список для их обнаружения параметров. Если куча не создавалась Вашим процессом или его потомками, то она и не появится в этом списке. Команда QUERY сканирует адресное пространство приложения, используя работу для идентификации областей памяти команды WHAT. Если последняя находит область, чей базовый адрес совпадает с адресом кучи, занесенной в список, то он и обозначается как куча. Если же WHAT не в состоянии идентифицировать область таким способом, то исследует дескриптор, что позволяет определить область как кучу или ее сегмент.

Кучи, существующие в адресном пространстве процесса, но не перечисленные в его списке, были отображены в данное адресное пространство другим процессом. В большинстве случает этим занимается подсистема CSRSS. Во время инициализации она создает кучу по стандартному базовому адресу. Во время запуска нового процесса эта куча переотображается в его адресное пространство по тому же самому базовому адресу, что теоретически позволяет обоим процессам использовать одну кучу. На практике же существуют разные обстоятельства, которые могут препятствовать такой работе — одним из них является синхронизация. Заметьте себе, что в последующих версиях Windows NT в адресное пространство процесса может быть отображено более одной кучи, причем они могут отображаться в различных процессах по разным базовым адресам. Команда QUERY помечает это обстоятельство в своем листинге. Кроме того, новые версии операционной системы используют кучи, созданные в системном адресном пространстве, но, иногда, отображаемые в адресное пространство пользователя. Windows NT позволяет создавать кучи в системном адресном пространстве, используя API, экспортируемое NTOSKRNL. Эти вызовы сходны с аналогичным API, экспортируемым NTDLL.DLL.

Объекты USER и GDI

В Windows NT 3.51 процесс CSRSS подсистемы Win32 обеспечивает большую часть традиционных функций подсистемы USER. API и структуры данных модуля WINSRV.DLL управляют классами окон и их структурами, а также множеством других типов данных USER.

В Windows NT 3.51 существуют следующие типы объектов USER (в скобках указаны их идентификаторы).

FREE (0)	Дескриптор объекта не используется (или недействителен).
HWND (1)	Окна.
MENU (2)	Меню окна.
ICON/CURSOR (3)	Иконка или курсор окна.
DEFERWINDOWPOS (4)	Объект, возвращаемый BeginDeferWindowPosition API.
HOOK (5)	Переадресатор перехвата сообщений.
THREADINFO (6)	Данные экземпляра потока клиента CSRSS.
QUEUE (7)	Очередь сообщений окна.
CPD (8)	Call Procedure Data thunk.
ACCELERATOR (9)	Таблица акселераторов.
WINDOW STATION (0xA)	
DESKTOP (0xB)	Объект "Рабочий стол".
DDEOBJECT (0xC)	Объект DDE, например, строки.

В последующих версиях Windows NT добавлены (или переопределены) следующие объекты USER.

DESKTOP (---)	Этот тип объектов удален и теперь принадлежит объекту ядра, который управляется менеджером объектов ядра (Kernel Object Manager).
QUEUE (---)	Это тип объектов удален.
WINDOW STATION (0xD)	Изменен идентификатор (ID) типа объекта. Также существует как объект ядра.
DDEOBJECT (0xA)	Изменен идентификатор (ID) типа объекта.
KEYBOARD LAYOUT (0xE)	Новый тип объекта. Описывает раскладку клавиатуры.
CLIPBOARD FORMAT (7)	Новый тип объекта. Зарегистрированные форматы Clipboard.

Кроме поддержки структур данных объектов USER и GDI для каждого процесса, CSRSS ведет главную таблицу дескрипторов для всех процессов. Объекты USER и GDI разделены в две разные таблицы, имеющих одинаковое строение, но различные функции управления. Все они имеют префикс "HM", к которому для функций GDI добавляется символ "G". Так, HMAllocObject создает объекты USER, а HMGAlloc — это программа API, создающая объекты GDI.

Управление дескрипторами USER и GDI спроектировано достаточно просто и является хорошим примером того, как следует реализовывать базовое управление абстрактными типами объектов. Например, этот API использует простую, но стабильную технику для создания уникальных дескрипторов и поддержки счетчиков обращений. При этом система спроектирована так, чтобы исключить возможность непосредственного манипулирования объектами из любых внешних по отношению к менеджеру приложений, включая USER32 и CSRSS. Такое исключение любых внешних воздействий, включая и воздействия со стороны системы, позволяет гарантировать правильность работы менеджера дескрипторов, обеспечивает точность счетчиков и семафоров обращения в объектам.

Главные таблицы объектов, управляемые менеджером дескрипторов, представляют собой массивы с изменяемыми размерами, но с фиксированным размером элемента. В следующей таблице показаны поля таблицы объектов. (Обратите внимание, что элементами таблицы являются только колонки, обозначенные **жирным** шрифтом; колонки, озаглавленные *курсивом*, включены только для пояснений).

Элемент	**Указатель объекта (DWORD)**	**Владелец (DWORD)**	**Тип (BYTE)**	**Флаги (BYTE)**	**Счетчик экземпляров (WORD)**	*Значение дескриптора*
0	NULL	NULL	FREE (0)	00	0001	00010000
1	HEAP *	HEAP *	DESKTOP (0C)	00	0001	00010001
2	HEAP *	HEAP *	HWND (04)	01	0003	00030002

Поле "Указатель объекта" указывает на действительные данные объекта. Обычно оно указывает на одну из куч CSRSS или на выгружаемую область. Поле "Тип" полностью соответствует своему названию. Счетчик экземпляров используется при создании уникальных дескрипторов, а поле "Флаги" — для определения специальных условий, например, таких как блокирование объекта потока для исключительного использования.

Как создаются значения дескрипторов

Изначально все счетчики экземпляров объектов установлены в 1. Когда выделяется новая позиция в таблице, то счетчик экземпляров комбинируется с индексом выделяемой позиции, чтобы создать уникальное значение дескриптора. Когда производится ссылка на объект, часть дескриптора, содержащая номер позиции, извлекается и используется в качестве индекса. Для проверки подлинности дескриптора из таблицы получают значение счетчика экземпляров данного объекта и сравнивают с проверяемым дескриптором. Если значения не совпадают, то дескриптор недействителен. Следующий пример иллюстрирует вышесказанное:

Создание дескриптора объекта:

```
ДескрипторОбъекта = ИндексПозиции + (СчетчикЭкземпляров << 16);
```

Проверка подлинности дескриптора:

```
ТаблицаОбъектов [LOWORD(Дескриптор)].СчетчикЭкземпляров == HIWORD(Дескриптор);
```

Когда объект удаляется, то все поля дескриптора устанавливаются в ноль, а счетчик экземпляров увеличивается на единицу. Таким образом, при повторном использовании позиции в таблице дескриптор для нового объекта будет уникальным.

Замечание: Собственно тип объекта не является часть дескриптора. Это означает, что приложение не может определить его непосредственно, для этого необходимо получить доступ к позиции в таблице объектов.

Такая техника создания уникальных дескрипторов достаточно проста и эффективна, и делает проверку их подлинности тривиальной. Представьте себе, что процесс создает окно и получает его дескриптор. Во время исполнения программы процесс уничтожает окно, однако значение дескриптора остается доступным. Если процесс попытается использовать дескриптор после уничтожения окна, то такой дескриптор будет признан недействительным, а объект, на который он указывает, будет иметь тип FREE. Это состояние перехватывается, и программа не сможет использовать дескриптор. В то же самое время другой процесс может создает новый объект, и вполне вероятно, что для него будет заново выделена позиция, освобожденная при уничтожении окна. Если при этом первый процесс попытается использовать недействительный дескриптор, то значения счетчиков экземпляров уже не будут совпадать, и проверка снова закончится неудачей.

Таблицы объектов не являются специфичными для какого-либо процесса, поэтому значения дескрипторов для объектов USER и GDI уникальны не только для данного процесса. Дескрипторы HWND уникальны для всей подсистемы Win32. Ни один процесс никогда не может создать HWND со значением, которое совпадало бы с дескриптором другого процесса.

Таблица объектов USER

Используя команду OBJTAB, можно вывести на экран все содержимое таблицы объектов USER. Эта команда достаточно гибкая и позволяет задавать в качестве параметра дескриптор или индекс позиции в таблице. Также она позволяет просмотреть объекты определенного типа, используя сокращения, принятые для имен типов объектов. Чтобы получить список имен типов объектов, с которыми может иметь дело команда OBJTAB, укажите в командной строке параметр -H.

Поле указателя каждой позиции в таблице ссылаться на данные объекта. Все объекты имеют заголовок, поддерживаемый менеджером объектов, который содержит значение дескриптора объекта и счетчик ссылок потока. Большинство типов объектов также содержат указатель на объект "рабочий стол" и/или на своего предка.

Следующий пример демонстрирует содержимое позиции таблицы объектов, касающейся дескрипторов окна, а также содержимое памяти заголовка, поддерживаемого менеджером. Ключевая информация выделена в листинге жирным шрифтом.

1 Задаем команду OBJTAB, чтобы найти дескриптор произвольного окна и получить указатель объекта. В данном промере значение дескриптора равно 0x1000C, а поле "Owner" (владелец) содержит адрес 0xE12E7008:

```
:objtab hwnd

Object        Type     Id     Handle      Owner       Flags
E12E9EA8      Hwnd     01     0001001C    E12E7008    00
```

2 Выводя содержимое блока 20h байт по полученному адресу объекта, обнаруживаем следующее:

```
:dd e12e9ea8 l 20

0010:E12E9EA8  0001001C 00000006 00000000 FF0E45D8
0010:E12E9EB8  00000000 E12E7008 00000000 00000000
```

Величина 0x1001C по смещению 0 — это значение дескриптора объекта, поле по смещению 4, содержащее значение 6 — счетчик ссылок на объект, а значение по смещению 0Ch (0xFF0E45D8) — это указатель на объект "Рабочий стол".

3 Проверим подлинность дескриптора с помощью команды WHAT:

```
:what ff0e45d8

The value FF0E45D8 is (a) Kernel Desktop object (handle=0068)
for winlogon(21)
( Значение FF0E45D8 принадлежит объекту "Рабочий стол"
  (дескриптор=0068) для winlogon(21) )
```

Поле по смещению 14h имеет то же самое значение (0xE12E7008), которое размещается в поле "Owner" (владелец) в таблице объекта.

4 Выводя содержимое блока 20h байт по адресу владельца, обнаруживаем следующее:

```
:dd e12e7008 l 20

0010:E12E7008  0001001B 00000000 00000000 E12E9C34
0010:E12E7018  E17DB714 00000000 00000000 00000000
```

5 Величина 0x0001001B по смещению 0 очень похожа на дескриптор объекта, это и есть объект "информация потока". Команда OBJTAB со значением 0x1001B в качестве параметра показывает именно этот тип объекта:

```
:objtab 1001b

Object      Type           Id    Handle      Owner       Flags
E12E7008    Thread Info    06    0001001B    00000000    00
```

Наблюдение за созданием объектов USER

Если Вы разрабатывает большое количество приложений для Win32, то функция HMAllocObject является удобным средством для наблюдения за созданием различных типов объектов, например, окон. Для этого случая можно записать макрокоманду, которая устанавливает прерывание, перехватывающее создание объектов:

```
:MACRO obx = "bpx winsrv!HMAllocObject if (esp->c == %1)"
```

Функция HMAllocObject API реализована в составе WINSRV.DLL, а тип создаваемого объекта передается в третьем параметре (`Dword ptr esp [0Ch]`). В приведенной макрокоманде на тип объекта ссылается запись вида "`esp->c`", что эквивалентно ссылке в форме `*(esp+c)`. Часть условного выражения "%1" представляет собой подстановку аргумента; во время исполнения макрокоманды OBX вместо него подставляется задаваемый аргумент, например:

```
:OBX 1  —>  bpx winsrv!HMAllocObject if (esp->c == 1)
```

После установки данного прерывания, оно будет перехватывать все вызовы HMAllocObject, которые требуют создания объекта типа "Окно".

Адресное пространство процесса

Адресное пространство для процесса пользовательского режима отображается в младшие 2 Гб с линейными адресами 0x00000000 - 0x7FFFFFFF. Верхние 2 Гб зарезервированы для ядра операционной системы и драйверов устройств.

В общем случае в адресном пространстве приложения Win32 для предопределенных целей выделяются следующие области линейной памяти.

Линейный адрес	**Назначение области**
Линейный адрес	Назначение области
0x00000000 - 0x0000FFFF	Защищенная область. Используется для перехвата операций записи с нулевым (NULL) указателем.
0x00010000	Стандартный адрес загрузки процесса Win32.
0x70000000 - 0x78000000	Стандартная область загрузки DLL подсистемы Win32.
0x7FFB0000 - 0x7FFD3FFF	Кодовые таблицы ANSI и OEM. Таблица(цы) трансляции Unicode.
0x7FFDE000 - 0x7FFDEFFF	Главный блок переменных окружения потока (thread environment block) пользовательского режима.
0x7FFDF000 - 0x7FFDFFFF	Блок переменных окружения процесса (process environment block - UPEB) пользовательского режима.
0x7FFE0000 - 0x7FFE0FFF	Область очереди сообщений.
0x7FFFF000 - 0x7FFFFFFF	Защищенная область (используется аналогично области 0x00000 - 0x0FFFF).

В Windows NT младшие и старшие 64 Кб адресного пространства пользовательского режима зарезервированы и никогда не используются для отображения физической памяти. Младшие 64 Кб используются для перехвата операций записи с использованием нулевых указателей (NULL).

По умолчанию адрес загрузки процессов под управлением Windows NT равен 0x10000. Однако, процессы часто изменяют базовый адрес загрузки. Так, например, процессы, разработанные для операционной системы Windows 95 имеют загрузочный адрес, равный 0x400000. Для установки требуемого загрузочного адреса .DLL или .EXE-файлов следует использовать соответствующие параметры компоновщика или утилиту REBASE.

Область линейных адресов, начинающаяся от 0x70000000, это область, куда загружаются модули подсистемы Win32; информацию о загруженных сюда модулях можно получить, используя команды MOD, MAP32 или QUERY.

Блок окружения процесса пользователя всегда отображается на адрес 0x7FFDF000, в то время как главный блок окружения потока пользователя размещается на одну страницу ниже по адресу 0x7FFDE000. По мере создания других потоков, они размещаются по границе страницы самого высокого еще неиспользуемого адреса.

Следующие пример использования команды THREAD показывает, каким образом каждый последующий поток размещается страницей ниже относительно предыдущего:

```
:thread winword

TID     Krnl TEB    StackBtm    StkTop      StackPtr    User TEB    Process(Id)
006B    FFA7FDA0    FEAD7000    FEADB000    FEADAE64    7FFDE000    WINWORD(83)
007C    FF0A0AE0    FEC2A000    FEC2D000    FEC2CE18    7FFDD000    WINWORD(83)
009C    FF04E4E0    FC8F9000    FC8FC000    FC8FBE18    7FFDC000    WINWORD(83)
```

Для получения дополнительной информации об адресном пространстве процесса пользователя используйте команду QUERY. Эта команду сообщает о зарезервированных и/или переданных в пользование линейных областей. Для идентификации содержимого области она использует команду WHAT. С ее помощью мы может посмотреть кучи процесса, модули, отображаемые в память файлы, а также стеки потоков и их блоки окружения.

Функции управления кучами

Строение кучи

Любое пользовательское приложение прямо или косвенно использует функции управления кучами, которые экспортируются KERNEL32 и NTDDL. Кучи предназначены для управления большими областями линейной памяти и для выделения в этих областях меньших по размеру блоков. Ядро реализации этих функций находится в NTDLL, однако некоторые функции, как, например, HeapCreate и HeapValidate, экспортируются из KERNEL32. Так как в модуле KERNEL32 не содержится кода для некоторых функций (например, HeapFree), то загрузчик исправляет указатели их вызова на код, расположенный в NTDLL.

Замечание: Методика замены экспортирования функций из одного модуля на функции, экспортируемые другим модулем, называется 'snapping' (перенаправление).

Хотя с точки зрения приложения управлять кучами относительно несложно, однако, реализация API и структур данных не столь проста. Управление кучей ушло достаточно далеко от стандартных функций языка "C" malloc() и free(), особенно в том, что касается выделения больших раздробленных областей линейной памяти, используемых для дальнейшего распределения меньших блоков, и объединения смежных блоков свободной памяти. Функции кучи выполняют быстрый выбор оптимального по размерам блока для удовлетворения запроса, обеспечивают безопасное взаимодействие потоков при операциях выделения памяти, а также предоставляют большое количество информации, касающееся параметров кучи, и поддерживают функции отладки.

Главная структура данных кучи достаточно велика, она занимает около 1400 байт в обычной версии и примерно в 2 раза больше в отладочной. И это не считая дополнительных структур данных, участвующих в управлении областями линейных адресов. Большая часть этой структуры занята 128 узлами двунаправленного списка свободных блоков. Для каждого значения меньше 1 Кб создается свой двунаправленный список, в котором хранятся блоки одного размера. Блоки больше 1 Кб хранятся в одном общем сортированном двунаправленном списке. Это позволяет очень быстро выполнять запрос на выделение оптимального по размеру блока памяти. Вот Вам наглядный пример решения компромисса в пользу большей скорости обработки запроса и в ущерб объему структур данных.

Для того, чтобы понять особенности строения и реализации функций управления кучами, очень важно уяснить, что куча Win32 не обязана быть представлена единым непрерывным блоком линейной памяти. Для куч с изменяемыми размерами с помощью функции VirtualAlloc может быть выделено множество линейных областей, которые в общем случае не обязательно будут непрерывными. Для отслеживания всех составляющих кучу областей линейной памяти создаются специальные структуры, называемые сегментами кучи. Другой важной особенностью организации API куч является двухстадийность процесса резервирования и выделения виртуальной памяти, который обеспечивается функцией VirlualAlloc и связанными с ней другими функциями. Отслеживание того, какая память лишь зарезервирована, а какая уже распределена, осуществляется с помощью структур данных, известных как таблицы нераспределенных областей (Uncommitted Range Tables — UCR).

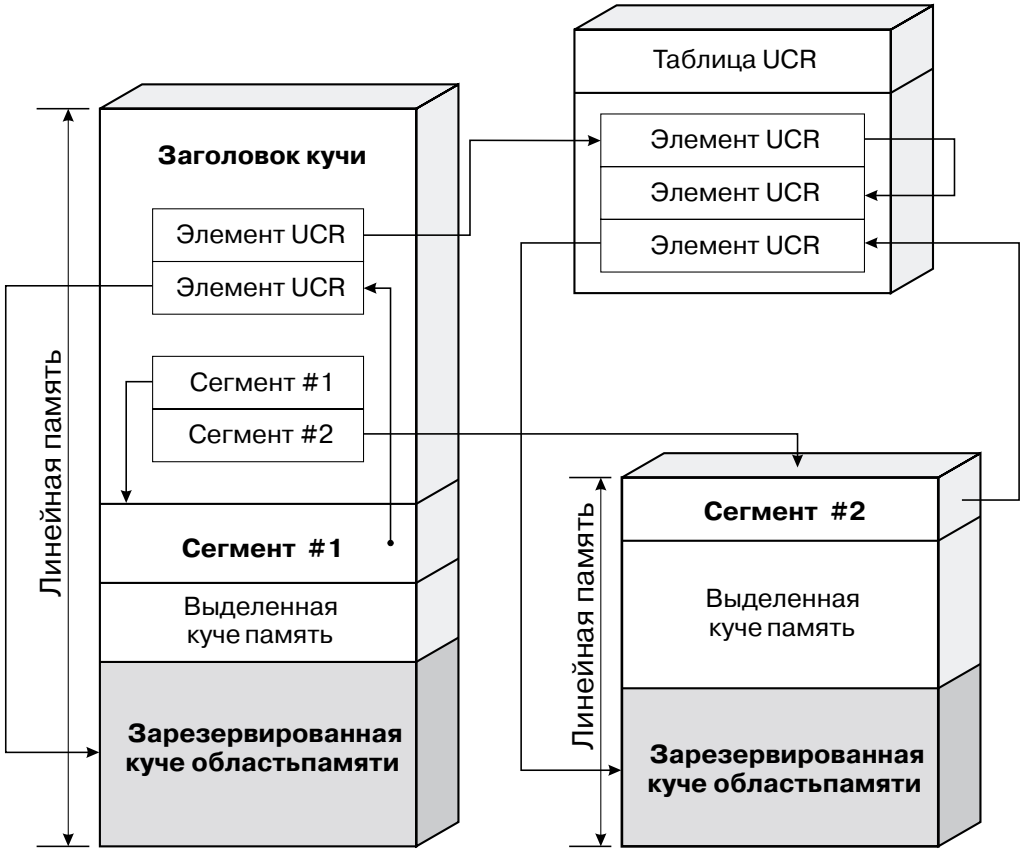
Когда функция Ntdll!RtlCreateHeap() создает и инициализирует кучу, она выделяет начальный виртуальный регион памяти для ее размещения, и строит соответствующие структуры данных. В первых 4 Кб (размер одной страницы) создается главная структура данных, и непосредственно после нее помещается 1й сегмент кучи. Этот сегмент инициализируется таким образом, чтобы управлять первоначальной областью виртуальной памяти, выделенной куче. Вся область памяти после него доступна для дальнейшего выделения через функцию HeapAlloc(). Если в 1м сегменте резервируется область любого размера, то создается элемент таблицы UCR для отслеживания оставшейся нераспределенной памяти.

Замечание: Функция Kernel32!HeapAlloc() перенаправляет свое действие функции Ntdll!RtlAllocateHeap.

Кроме упоминавшихся выше 128 списков свободной памяти главная структура кучи содержит 8 элементов таблицы UCR, что достаточно для небольших куч, однако, по мере необходимости может быть создано столько UCR, сколько потребуется. Здесь же находится таблица из 16 указателей на сегменты кучи, поэтому больше 16 сегментов куча никогда не может иметь. Если куче требуется синхронизация потоков, то заголовок кучи в конце главной структуры перед 1м сегментом дополняется структурой данных критической секции.

Приведенная ниже диаграмма в общих чертах иллюстрирует строение типичной кучи и размещение наиболее важных ее частей относительно друг друга.

На левой части диаграммы показана выделенная куче область виртуальной памяти. В начале выделенной области размещен заголовок кучи, сопровождаемый Сегментом №1. Адрес этой структуры содержится в первом указателе таблицы указателей сегментов. Остальная распределенная, но еще не затребованная память находится сразу после 1го сегмента и помечена как свободная. После нее находится область, которая зарезервирована для дальнейшего использования. Если поступает запрос на выделение большей области памяти, чем в настоящий момент распределено данной куче, то для удовлетворения запроса ей самой выделяется дополнительный блок.



Сегмент №1 содержит область виртуальной памяти, первоначально выделенной куче для использования. Начальный адрес в сегменте равен базовому адресу кучи, а конец области соответствует концу распределенной памяти. Заштрихованная часть столбца диаграммы представляет зарезервированную область адресов, память для которой еще не выделена, поэтому заголовок использует свободный элемент UCR для отслеживания этой области. Когда необходимо выделить память для удовлетворения поступившего запроса, проверяются все элементы UCR данного сегмента в поисках достаточного по размерам региона для ее размещения. Для увеличения производительности системы сегменты кучи отслеживают наибольшие доступные регионы UCR и общее число нераспределенных страниц в пределах виртуальной памяти каждого сегмента.

В правой части диаграммы показана вторая область выделенной виртуальной памяти, которая управляется с помощью Сегмента кучи №2. Дополнительные сегменты кучи создаются в тех случаях, когда размер запрашиваемой памяти превосходит размер наибольшего нераспределенного региона, имеющегося в пределах существующего сегмента. Однако, это возможно только в тех случаях, когда размер требуемой памяти меньше предопределенного для кучи порога Vmthreshold, иначе участок кучи распределяется непосредственно функцией VirtualAlloc, а новый сегмент не создается.

Как уже упоминалось, количество элементов UCR в заголовке кучи невелико. С иллюстративной целью на диаграмме показана дополнительная таблица элементов UCR, созданная специально для увеличения числа доступных UCR-элементов. В общем же случае необходимость в создании дополнительной таблицы возникает достаточно редко, и она является признаком существования большого количества сегментов или того, что эти сегменты сильно фрагментированы.

Фрагментация виртуальной памяти может произойти, когда функции управления кучей начинают возвращать распределенную память в процессе слияния свободных блоков. Термином возврат (decommitting) обозначается процесс перевода распределенной памяти в состояние зарезервированной или нераспределенной. Когда блок свободной памяти начинает превышать размер одной физической страницы (4 Кб), такая страница становится кандидатом на возвращение. Если при этом превышается пороговое значение, то менеджер кучи начинает процесс возвращения свободных страниц. В том случает, когда эти страницы не составляют непрерывного блока с существующим регионом зарезервированной памяти, для управления новым блоком зарезервированной памяти необходимо использование нового элемента UCR.

В следующем примере показано исследование кучи по умолчанию с помощью команды HEAP32.

1 Команда HEAP32 с параметром "-S" выдает информацию о сегменте для кучи по умолчанию:

```
                                                  номер сегмента
:heap32 -s 140000                                 /
Base      Id  Cmmt/Psnt/Rsvd    Segments  Flags      Process
00140000  01  001C/0018/00E4           1  00000002   Explorer
    01    00140000-00240000   001C/0018/00E4     E4000
          ↖диапазон памяти сегмента кучи      максимальный UCR ↗
```

2 Применение параметра "-X" позволяет получить дополнительную информацию о куче по умолчанию:

```
:heap32 -x 140000
 Extended Heap Summary for heap 00140000 in Explorer
Heap Base:    140000   Heap Id:       1   Process:    Explorer
Total Free:     6238   Alignment:     8   Log Mask:   10000
Seg Reserve:  100000   Seg Commit: 2000
Committed:       112k  Present:      96k  Reserved:   912k
Flags: GROWABLE
DeCommit:   1000     Total DeC:     10000     VM Alloc: 7F000
  исходный размер сегмента кучи                    порог кучи
                                размер области для распределения
```

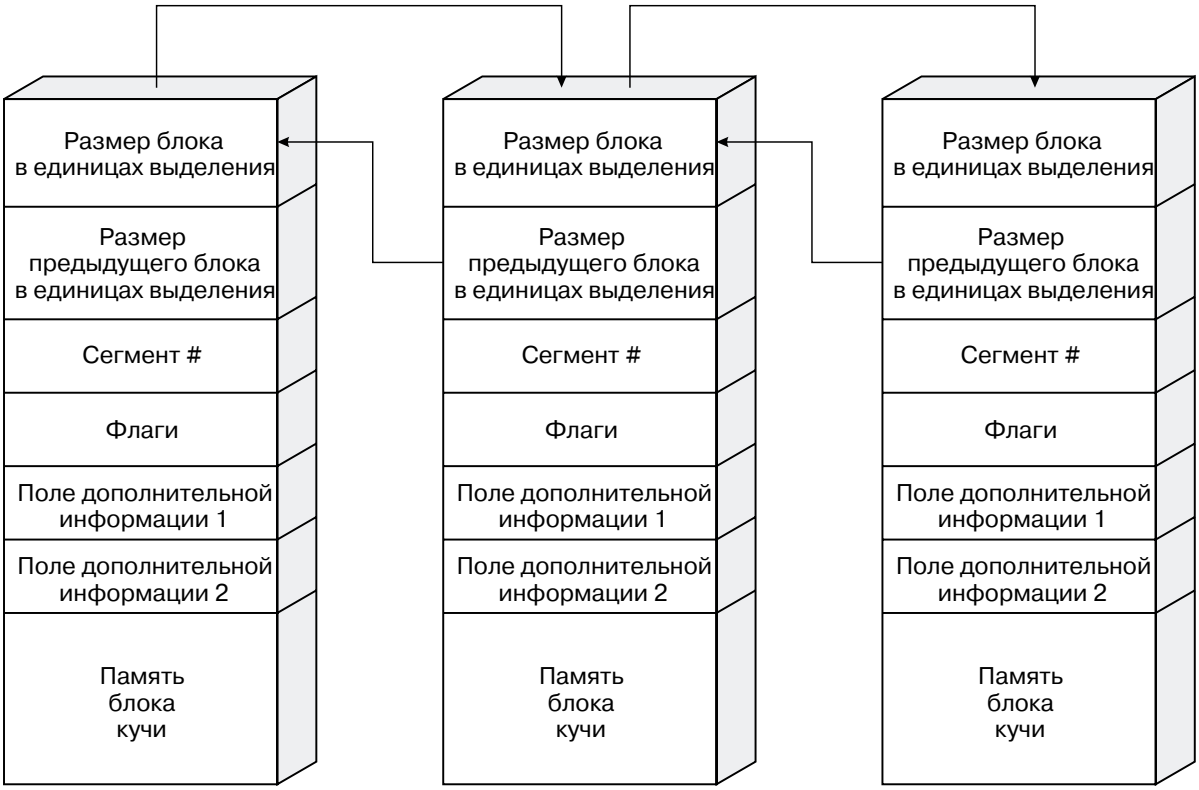
3 Параметр "-B" команды HEAP32 выдает адреса блоков памяти в куче по умолчанию:

```
:heap32 -b 140000
     Base          Type         Size       Seg#    Flags
     00140000      HEAP         580        01
     00140580      SEGMENT      38         01
     001405B8      ALLOC        30         01
```

В приведенном выше листинге можно увидеть, что заголовок кучи сопровождается Сегментом кучи №1, а первый выделенный куче блок памяти размещается сразу после структуры данных Сегмента.

Управление блоками кучи

Как уже упоминалось в предыдущем разделе, функции управления кучами используют API виртуальной памяти Win32 для размещения больших областей линейной памяти и пользуются Сегментами кучи для обслуживания выделенных и зарезервированных областей. На базе всех этих функций строится механизм, который и управляет выделением и возвращением блоков памяти в Вашем приложении.



Для управления памятью для каждого блока создаются заголовки. На приведенной диаграмме показано, каким образом менеджер куч управляет блоками *непрерывной* памяти. Кроме того, менеджер кучи отслеживает блоки свободной памяти с помощью двунаправленного списка, однако, указатели на предыдущий и последующий узлы в заголовке блока не сохраняются. Вместо этого с данной целью используются первые 2 двойных слова в памяти блока.

Как показано на диаграмме, каждый блок хранит в заголовке свой размер, а также размер предыдущего блока, выраженные количеством единичных областей выделения. Используя эти два размера, менеджер способен последовательно просмотреть все блоки кучи.

Память в куче выделяется участками, называемыми единичными областями выделения, и имеющих размер 8 байт. При необходимости величина запрашиваемой памяти округляется вверх, чтобы быть кратной этому размеру. Это означает, что размеры всех выделенных блоков кратны 8 байтам, и что размер любого блока может быть выражен в количестве единичных областей с помощью округления вверх и деления на 8.

Например, если процесс требует выделения 32 байт, то число единиц выделения составляет 32 / 8 = 4. Если же был запрос на 34 байта, то размер будет округлен вверх до числа, кратного 8, а размер выделенной области составит 40 байт или 5 единиц выделения. Однако, процесс, запросивший память, ничего не знает о единицах выделения и обращается с выделенной ему памятью, как будто она составляет ровно 34 байта.

При использовании в качестве единицы выделения 8 байт размер большинства типичных запросов на выделение памяти может быть записан в одном слове, что, однако, ограничивает размер блока кучи максимальным значением короткого беззнакового целого или 0xFFFF единицами выделения. Другими словами, теоретический предел размера блока составляет 0xFFFF * 8 или 524280 байт, что и документировано в описании функции HeapAlloc. Означает ли это, что программа не может разместить в куче блок больше 512 Кб? И да, и нет. Блок размером более 512 Кб размещен быть не может, однако, ничто не запрещает программе выделить область линейной памяти с помощью функции VirtualAlloc. Именно эту функцию и вызывает менеджер кучи в тех случаях, когда размер запрошенной для выделения области памяти превышает величину VMThreshold. Переменная VMThreshold хранится в заголовке кучи и по умолчанию равна 520192 байт (0xFE00 единиц выделения). Когда менеджер кучи размещает большой блок памяти с помощью функции VirtualAlloc, получающаяся в результате структура называется "Виртуально размещенный блок" (Virtually Allocated Block — VAB).

Менеджер куч переходит по списку непрерывных блоков от одного к другому, переводя размер блока в единицах выделения в байты и прибавляя его к базовому адресу блока. Адрес предыдущего блока вычисляется аналогичным образом, вычитая размер предыдущего блока из базового адреса текущего блока. Эти операции выполняются при слиянии свободных смежных блоков, при выделении блока меньшего размера из большого свободного блока, а также, когда производится проверка кучи или элемента кучи.

Размер блока в единицах выделения важен для управления списком свободных блоков, сортируемых по размеру, и информация о которых хранится в массиве 128 двунаправленных списков внутри заголовка кучи. Свободные блоки от 1 до 127 единиц выделения хранятся в списках, соответствующих их размеру. То есть все свободные блоки длиной, например, 32 единицы хранятся в Heap->FreeLists[32]. Так как существование блоков с нулевым размером невозможно, то в списке с индексом "0" хранятся все блоки больше 127 единиц; в списке они отсортированы в порядке возрастания. Так как основная часть выделяемых блоков памяти имеют размер меньше 128 единиц (1024 байт или 1 Кб), то такая организация обеспечивает быстрое

выделение оптимального по размером блока. Блоки больше 1 Кб выделяются реже, поэтому линейный поиск в единственном списке свободных блоков не оказывает на общую производительность системы большого отрицательного влияния.

Поле флагов в заголовке блока кучи содержит специальные атрибуты блока. Один бит обозначает свободен блок или занят. Другой используется, если это виртуально размещенный блок (VAB). Третий ставится в последнем блоке выделенной куче памяти; такой блок используется в качестве маркера и обозначает, что после него блоков нет. Применение этого флага позволяет проверить достоверность адреса блока кучи значительно быстрее, чем проходить по всей цепочке сегментов UCR. Еще один флаг маркирует блок для специальной "хвостовой" проверки. Во время отладки процесса менеджер кучи маркирует блок соответствующим флагом. Когда блок освобождается или свободный блок выделяется заново, то менеджер кучи может проверить, не было ли попыток писать информацию за пределами этого блока.

Поля дополнительной информации в заголовке блока используются по разному в зависимости от того, выделен ли этот блок или свободен. В выделенном блоке первое поле хранит количество дополнительных байтов, которые выделяются для выравнивания блока по заданной границе или для дополнения до обязательного размера выделения. Второе поле представляет собой псевдо-таг. Однако, таги заголовка и псевдо-таги выходят за пределы данного обсуждения.

В свободном блоке дополнительные поля содержат байтовые и битовые маски, которые обеспечивают доступ к битовому массиву списков свободных блоков в заголовке кучи. Этот битовый массив обеспечивает быстрый поиск свободного блока во время выделения памяти небольшого размера. Каждый бит в поле представляет собой один из 127 списков свободных блоков, и, если он установлен, то в данном списке имеется по крайней мере один элемент. Нулевое значение бита означает, что свободных элементов такого размера нет, и для удовлетворения запроса необходимо выделение большего блока. В первом дополнительном поле содержится байтовый индекс этого битового массива. Второе дополнительное информационное поле содержит инвертированную маску позиции бита в битовом массиве. Необходимо, однако, отметить, что все вышеописанное относится только к версии Windows 3.51. В последующих версиях битовый массив списков свободных блоков все еще присутствует, однако, ни байтовый индекс, ни битовая маска не хранятся.

Структура самого блока памяти также зависит от его статуса. У выделенного блока вся память используется запросившим его приложением. В свободном блоке первые два двойных слова (1 единичная область выделения) хранит указатели двунаправленного списка на предыдущий и следующий свободные блоки. Если процесс, требующий выделения блока памяти, находится в состоянии отладки, то выделенный блок содержит маркер "концевой проверки". Свободные блоки маркируются специальным тагом, который позволяет определить ситуацию, когда делается попытка писать в блок с помощью недействительного указателя, или когда процесс продолжает использовать блок после его освобождения.

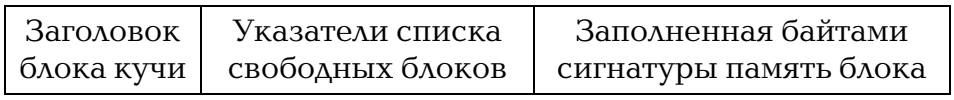
На следующей диаграмме представлено общая структура выделенного блока кучи.

Заголовок блока кучи	Память блока	Хвостовой маркер	Байты дополнения

На данной диаграмме часть блока, обозначенная "*байты дополнения*" представляет собой память, необходимую для дополнения размера до величины, кратной единице выделения. Эта память не должна использоваться запросившим ее процессом, однако, менеджер кучи не обеспечивает никакой непосредственной защиты от записи в эту область. Хвостовая метка размещается непосредственно после основной части блока. Если приложение производит запись за пределы блока, то

этот маркер оказывается разрушенным, а менеджер кучи сигнализирует об этом отладчику с помощью отладочного сообщения и прерывания INT 3. При этом возможна запись в область байтов дополнения без разрушения хвостового маркера. Такая ситуация не обрабатывается. Функции управления кучами предоставляют возможность инициализировать выделяемую память нулями. Если во время отладки такая возможность не используется, то менеджер кучи заполняет выделяемый блок памяти специальной сигнатурой, что может быть использовано для проверки, был ли инициализирован блок памяти Вашей программой.

На следующей диаграмме представлено общая структура свободного блока.



При освобождении блока в режиме отладки менеджер кучи записывает в память блока специальную сигнатуру. При последующем выделении проверяет, не была ли она изменена. Если это произошло, то менеджер выдает отладочное сообщение и генерирует прерывание INT 3, которое однако, в большинстве случаев отладчиками игнорируется, так как было не ими установлено. В качестве дополнительного замечания можно отметить, что размещение указателей списка узлов свободных блоков в начале блока достаточно спорно, так как, если программа продолжает использовать освобожденный блок памяти, она с большей вероятностью перепишет данные в начале блока, чем в конце. Эти указатели чрезвычайно важны для перемещения по куче, поэтому недействительный указатель вызывает ситуацию исключения, однако, когда это происходит, то бывает достаточно сложно восстановить исходный список свободных блоков.

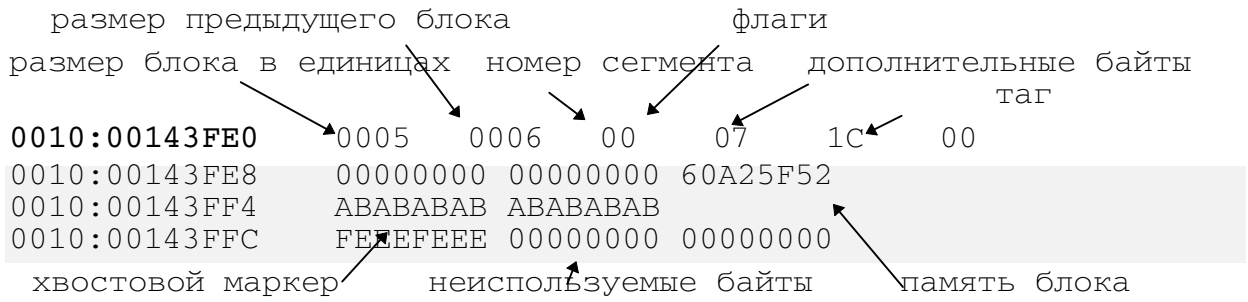
Следующие примеры демонстрируют использование команды HEAP32 с целью наблюдения и отладки элементов кучи.

В первом примере команда HEAP32 использована для прохождения по всем элементам кучи с базовым адресом 0x140000. Параметр "-B" выдает базовый адрес и размеры в таком виде, в котором их получает менеджер кучи. В противном случае эта информация показывается так, как ее получает приложение, запросившее выделение памяти. Листинг сокращен для наглядности, а информация о 2 блоках, выделенных жирным шрифтом, используется в дальнейших примерах.

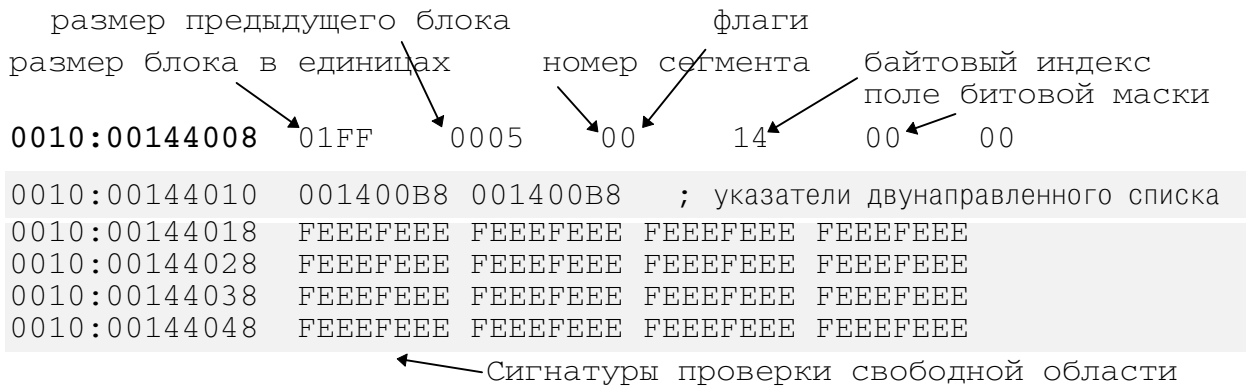
```
:HEAP32 -b 140000
Base        Type        Size      Seg#      Flags
00140000    HEAP        580       01
00140580    SEGMENT     38        01        TAGGED | BUSYTAIL
001405B8    ALLOC       40        01
. . .
00143FE0    ALLOC       28        01        TAGGED | BUSYTAIL
00144008    FREE        FF8       01        FREECHECK | SENTINEL
```

Для исследования содержимого выделенного и свободного блоков памяти в следующих примерах приведены снимки памяти этих блоков.

Поля заголовка блока по адресу 0x143FE0 идентифицируются следующим образом. Этот блок имеет размер 5 единиц выделения (40 байт), и из них 0x1C байт заняты служебной информацией: заголовок блока (1 единица), "хвостовой маркер" (1 единица), 1 элемент выравнивания (1 двойное слово) и одна дополнительная единица, оставшаяся от предыдущего выделения.



Непосредственно вслед за выделенным блоком по адресу 0x144008 размещается свободный блок. Он имеет размер 0x1FF единиц выделения, а размер предыдущего блока равен 5 единицам. Для свободных блоков размером 1 Кб и более (больше 80h единиц) байтовый индекс и поле битовой маски не используются и заполнены нулями. Флаг этого блока говорит также, что этот блок последний в куче (бит 4, или 0x10). Сразу после заголовка находится область, в которой помещены указатели двунаправленного списка для отслеживания свободных блоков. Оба указателя в данном блоке имеют значение 0x1400B8. Всю остальную часть памяти после указателей менеджер кучи заполнил специальной сигнатурой, которая проверяется во время последующего выделения блока, выполнения операций слияния свободных блоков или проверки целостности кучи.



Все человеческое знание проистекает из опыта.
Джон Локк

No man's knowledge here can go beyond his experience.
John Locke

Приложение A. Сообщения об ошибках

All break registers used, use in RAM only
(Все отладочные регистры задействованы, можно установить прерывание только в RAM)

Вы пытаетесь установить контрольную точку с помощью команды BPX в постоянной памяти (ROM), а все отладочные регистры уже задействованы. В оперативной памяти команда BPX еще может быть использована, так как в этом случае используется отладочное прерывание INT 3. Для решения проблемы необходимо удалить одну из контрольных точек типа BPM.

Attach to serial device has FAILED
(Неудачное подключение к последовательному устройству)

Неудачная попытка установления связи. Причинами могут быть неправильное указание последовательного порта, повреждение последовательного кабеля, или на удаленной машине не была запущена программа SERIAL.EXE.

BPM breakpoint limit exceeded
(Превышен предел количества BPM прерываний)

Из-за архитектурных ограничений процессоров x86 можно установить только 4 контрольных точки типа BPM. Чтобы установить новую контрольную точку такого типа, необходимо удалить одну из старых.

BPMD address must be on DWord boundary
(Адрес в команде BPMD должен указывать на границу двойного слова)

Адрес, заданный в команде BPMD, не указывает на границу двойного слова. Адрес на границе двойного слова должен иметь "0" в двух последних наименее значащих битах.

BPMW address must be on Word boundary
(Адрес в команде BPMW должен указывать на границу слова)

Адрес, заданный в команде BPMW, не указывает на границу слова. Адрес на границе слова должен иметь "0" в последнем наименее значащем бите.

Breakpoints not allowed within SoftICE
(Прерывания внутри SoftICE не разрешены)

Вы не можете установить контрольную точку внутри кода SoftICE.

Cannot interrupt to a less privileged level
Невозможно выполнить прерывание на менее привилегированном уровне

Вы не можете использовать команду GENINT для перехода с более привилегированного уровня на менее привилегированный. Это ограничение архитектуры процессоров x86.

Debug register is already being used
(Отладочный регистр уже используется)

Отладочный регистр, указанный в команде BPM, уже используется другой контрольной точкой.

Duplicate breakpoint
(Дублирование контрольной точки)

Контрольная точка такого типа по указанному адресу уже существует.

Expecting value, not address
(Должно быть задано значение, а не адрес)

Вычислитель выражений разделяет операнды на значения и их адреса. Адреса выражаются через селектор/сегмент и смещение, даже если используется непрерывная (flat) модель памяти. Некоторые операторы (например, * или /) могут использовать в качестве операндов только значения, и попытка использования в них адресов приводит к выдаче данного сообщения. В некоторых случаях использование операторов переадресации приводит к появлению адресов. Подробности смотрите в разделе "*Операторы*" на странице 100.)

Expression?? What expression?
(Выражение?? Что вычислять?)

Вычислитель не находит в заданной команде объекта для вычислений. Обратите внимание, что в более старых версиях SoftICE команда "?" могла быть использована для вызова подсказки. В текущей версии это уже не так — для вызова подсказки используйте команду H (F1).

Int 0D fault in SoftICE at address XXXXX offset XXXXX Fault Code=XXXX
(Ошибка прерывания Int 0D в коде SoftICE по адресу XXXXX
смещение XXXXX Код ошибки=XXXX)

(или)

Int 0E Fault in SoftICE at address XXXXX offset XXXXX Fault Code=XXXX (Ошибка прерывания Int 0E в коде SoftICE по адресу XXXXX смещение XXXXX Код ошибки=XXXX)

Эти 2 сообщения сигнализируют о внутренних ошибках SoftICE. Программа SoftICE вызвала либо общую ошибку защиты (0Dh) или отказ страницы (0Eh). Смещение — это смещение кода, исполнение которого вызвало эту ошибку. Пожалуйста запишите выданное Вам сообщение и перешлите его нам. В сообщении также показывается содержимое регистров, не забудьте записать их значения.

Invalid Debug register (Неверный отладочный регистр)

Номер отладочного регистра, задаваемого в команде BPM, больше 3. Могут быть использованы только регистры DR0, DR1, DR2, DR3.

No code at this line number (В строке с задаваемым номером исполняемый код отсутствует)

Строка с задаваемым в команде номером не имеет исполняемого кода.

No current source file (Файл с исходным текстом отсутствует)

Вы ввели команду SS, однако файл с исходным текстом программы на экране отсутствует.

No embedded INT 1 or INT 3 (Нет вложенных прерываний INT 1 или INT 3)

Команда ZAP не может найти вложенных прерываний INT 1 или INT 3. Данная команда находит только команды INT 1 или INT 3, расположенные перед текущим исполняемым адресом CS:EIP.

No files found (Файл не найден)

Для текущей таблицы отладочной информации не загружены исходные файлы.

No LDT (LDT отсутствует)

Это сообщение выдается, когда Вы используете команды 16-битной подсистемы Windows (HEAP, LHEAP, LDT или TASK), а текущий контекст не соответствует процессу NTVDM (виртуальная DOS машина NT).

No Local Heap (Локальная куча отсутствует)

В команде LHEAP задан селектор, не соответствующий локальной куче.

No more Watch variables allowed
(Больше переменных командой Watch добавлять нельзя)

Максимальное число устанавливаемых командой Watch переменных равно 8.

No search in progress
(Поиск не проводится)

Вы задаете команду S без параметров без предварительного поиска. Вы должны сначала выполнить команду S с адресом и списком данных в качестве параметров. Последующий поиск этих же данных может быть выполнен командой S без параметров.

NO_SIZE
(Нет размера операнда)

Во время выполнения команда A ассемблер не может определить, что Вы используете — байт, слово или двойное слово.

No symbol table
(Таблица с отладочной информацией отсутствует)

Вы выдаете команду SYM, SS или FILE, а символьная информация отсутствует.

No TSS
(TSS отсутствует)

Вы выдаете команду TSS в тот момент, когда в системе нет действительного сегмента состояния задачи.

Only valid in source mode
(Можно использовать только в режиме исходных кодов)

Вы не можете использовать команду SS в смешанном режиме или режиме кодов.

Page not present
(Страница отсутствует)

Задаваемый в команде адрес помечен в таблице страниц как отсутствующий. SoftICE, пытаясь получить информацию, обратился к памяти, которая расположена на странице, отсутствующей в физической памяти.

Parameter is wrong size
(Параметр имеет неправильный размер)

Один из введенных Вами параметров команды имеет неправильный размер. Например, Вы используете команду EB или BPMB с параметром размером в слово вместо байтового значения.

Pattern not found
(Образец не найден)

Команда S не нашла соответствия задаваемому для поиска образцу.

Press ‘C’ to continue, and ‘R’ to return to SoftICE
(Нажмите 'C' для продолжения или 'R' для возврата в SoftICE)

Экран SoftICE был активизирован в результате ошибки исполнения (06h, 0Ch, 0Dh, 0Eh). Нажмите клавишу 'R' для передачи управления в отладчик или клавишу 'C' для передачи управления в обработчик исключений Windows.

SoftICE is not active
(SoftICE не активен)

Это сообщение появляется в окне подсказки на монохромном или удаленном дисплее, когда SoftICE больше не работает.

Specified name not found
(Указанное имя не найдено)

Вы выдали команду TABLE с недействительным именем таблицы. Задайте команду TABLE без параметров для получения списка допустимых имен таблиц.

Symbol not defined (mysymbol)
(Символ "mysymbol" не определен)

Вы ссылаетесь на несуществующий символ. Используйте команду SYM для получения символьных имен для текущей таблицы отладочной информации.

Удивительно, что среди миллионов лиц не встретишь двух одинаковых.
Сэр Томас Браун

It is the common wonder of all men, how among so many millions of faces, there should be none alike.
Sir Thomas Browne

Приложение B. Поддерживаемые видеоадаптеры

В приведенной ниже таблице перечислены поддерживаемые SoftICE видеоадаптеры. включая и самые современные. Однако, NuMega регулярно добавляет поддержку новых видеоадаптеров для расширения возможностей SoftICE. Вы всегда можете скачать файлы для поддержки новых устройств с FTP фирмы NuMega или с различных BBS. Для получения дополнительной информации о загрузке файлов поддержки обратитесь к разделу "*Разрешение проблем с видеоадаптерами*" на странице 23.

Standard Display Adapter (VGA)	Actix GraphicsEngine 32I VL	Actix GraphicsEngine 32VL Plus
Actix GraphicsEngine 64	Actix GraphicsEngine Ultra 64	Actix GraphicsEngine Ultra Plus
Actix GraphicsEngine Ultra VL Plus	Actix ProSTAR	Actix ProSTAR 64
ATI 8514-Ultra	ATI Graphics Pro Turbo	ATI Graphics Pro Turbo PCI
ATI Graphics Ultra	ATI Graphics Ultra Pro	ATI Graphics Ultra Pro EISA
ATI Graphics Ultra Pro PCI	ATI Graphics Vantage	ATI Graphics Wonder
ATI Graphics Xpression	ATI 3d Xpression PCI	ATI VGA Wonder
ATI Video Xpression PCI	ATI WinTurbo	Boca SuperVGA
Boca SuperX	Boca Voyager	Cardinal VIDEOcolor
Cardinal VIDEOspectrum	Chips & Technologies 64310 PCI	Chips & Technologies 65545 PCI
Chips & Technologies 65548 PCI	Chips & Technologies Accelerator	Chips & Technologies Super VGA
Cirrus Logic	Cirrus Logic 5420	Cirrus Logic 5430 PCI
Cirrus Logic New	Cirrus Logic PCI	Cirrus Logic RevC
Cirrus Logic 7542 PCI	Cirrus Logic 7543 PCI	Compaq Qvision 2000
DEC PC76H-EA	DEC PC76H-EB	DEC PC76H-EC
DEC PCXAG-AJ	DEC PCXAG-AK	DEC PCXAG-AN
DFI WG-1000	DFI WG-1000VL Plus	DFI WG-1000VL/4 Plus

DFI WG-3000P	DFI WG-5000	DFI WG-6000VL
Diamond Edge 3D 2200XL	Diamond Edge 3D 3200XL	Diamond Edge 3D 3400XL
Diamond SpeedStar	Diamond SpeedStar 24	Diamond SpeedStar 24X
Diamond SpeedStar 64	Diamond SpeedStar Pro	Diamond SpeedStar Pro SE
Diamond Stealth 3D 2000	Diamond Stealth 24	Diamond Stealth 32
Diamond Stealth 64 2001	Diamond Stealth 64 (S3 964)	Diamond Stealth 64 (S3 968)
Diamond Stealth 64 Video	Diamond Stealth Pro	Diamond Stealth SE
Diamond Viper OAK	Diamond Viper PCI	Diamond Viper VLB
Diamond Stealth VRAM ELSA	WINNER 1000AVI ELSA	WINNER 1000PRO
ELSA WINNER 1000Trio ELSA	WINNER 1000 VL ELSA	WINNER 1280
ELSA WINNER 2000PRO ELSA	WINNER 2000 VL ELSA	WINNER/2-1280
Genoa Digital Video Wizard 1000	Genoa Phantom 32I	Genoa Phantom 64
Genoa WindowsVGA 24 Turbo	Genoa WindowsVGA 64 Turbo	Hercules Dynamite
Hercules Dynamite Pro	Hercules Graphite 64	Hercules Graphite Terminator 64
Hercules Graphite Terminator Pro	IBM 8514	IBM ThinkPad 755CX
IBM Think Pad 365XD	Matrox MGA Impression Lite	Matrox MGA Impression Plus
Matrox MGA Impression Plus 220	Matrox MGA Ultima Plus	Matrox MGA Ultima Plus 200
Matrox MGA Millennium	Number Nine GXE	Number Nine GXE64
Number Nine GXE64 Pro	Number Nine 9FX Vision 330	Number Nine 9FX Motion 531
Number Nine 9FX Motion 771	Number Nine FlashPoint 32	Number Nine FlashPoint 64
Number Nine Imagine 128	Number Nine Reality 332	Nvidia NVI Media Controller
Oak Technology 087	Oak Technology Super VGA	Orchid Fahrenheit 1280 Plus
Orchid Fahrenheit Pro 64	Orchid Fahrenheit VA	Orchid Kelvin 64
Orchid Kelvin EZ	Orchid ProDesigner II	Paradise Accelerator Ports O'Call
Paradise Accelerator VL Plus	Paradise Bahamas	Paradise Barbados 64
Paradise Super VGA	S3 805	S3 911/924
S3 928 PCI	S3 Trio32/64 PCI	S3 ViRGE PCI
S3 Vision864/964 PCI	S3 Vision868/968 PCI	Spider 32 VLB
Spider 32Plus VLB	Spider 64	Spider Tarantula 64
STB Ergo MCX	STB Horizon	STB Horizon Plus

STB LightSpeed	STB MVP-2X	STB MVP-4X
STB Nitro	STB Pegasus	STB PowerGraph Pro
STB PowerGraph VL-24	Trident 9420 PCI	Trident Cyber 93XX
Trident Super VGA	Tseng Labs	Tseng Labs ET4000
Tseng Labs ET4000/W32	Tseng Labs ET6000	Video Logic 928Movie
Video Seven VRAM/VRAM II/1024i	Western Digital	Western Digital (512K)
Weitek Power 9000	Weitek Power 9100	

Чем бы ни было творчество, оно — часть решения проблемы.
Брайан Олдис

Whatever creativity is, it is in part a solution to a problem.
Brian Aldiss

Приложение C. Устранение проблем SoftICE

Если у Вас возникла одна из перечисленных ниже проблем, попробуйте применить предлагаемое решение. Если это не поможет, обратитесь в Центр технической поддержки фирмы NuMega.

Проблема	Решение
Экран SoftICE черный или нечитаемый.	Возможно, при инициализации SoftICE Вы установили неверный тип видеоадаптера, или SoftICE не поддерживает Ваш видеоадаптер. Обратитесь к приложению B: "Поддерживаемые видеоадаптеры".
Компьютер зависает, когда Вы запускаете SoftICE, и при этом в нем не используется процессор Pentium или Pentium-Pro.	SoftICE ошибочно считает, что в Вашем компьютер используется процессор типа Pentium. Измените начальные установки SoftICE, чтобы отключить поддержку процессоров Pentium. Смотрите раздел "*Установка параметров для устранения неисправностей*" на странице 127.
Процессор зависает, когда вы запускаете SoftICE для Windows 95.	SoftICE не поддерживает режим "RESTART THE COMPUTER IN MS-DOS MODE?" ("Перезагрузить в режиме MS-DOS"?). Если после выбора этого режима вызвать SoftICE, то компьютер зависает. Измените параметр BootGUI=1 на BootGUI=0 в скрытом файле Windows 95 MSDOS.SYS, затем выбирайте пункт "SHUT DOWN THE COMPUTER" ("Выключить компьютер") для выхода в SoftICE.
Вы испытываете трудности с установлением модемной связи.	Модем возвращает код, который SoftICE не может обработать. SoftICE ожидает получить коды OK, COMNECT и RING. Замените команду ATX0 строке инициализации модема.
Мышь беспорядочно скачет по экрану SoftICE.	Нажмите Ctrl-M.

Проблема	Решение
Только для Windows NT: указатель мыши беспорядочно дергается по экрану SoftICE.	Перемещение указателя мыши во время появления экрана SoftICE может нарушить синхронизацию Windows NT и аппаратуры мыши. Переключитесь в полноэкранный режим DOS.
Клавиатура блокируется или ведет себя непредсказуемо во время загрузки SoftICE.	Отключите программирование клавиш "NumLock" и "CapsLock". Если это не помогло, но Вы используете Windows NT, то отключите модификацию драйвера клавиатуры. Смотрите раздел "*Установка параметров для устранения неисправностей*" на странице 127.
Windows 95 зависает, когда пытается сканировать последовательные порты.	Если Вы поместите команду SERIAL в строку инициализации, то SoftICE пытается установить связь до окончания полного запуска Windows 95. В время запуска Windows 95 соединение может нарушиться. Отключите выбранный Вами последовательный порт в Control Panel -> System Properties -> Device Manager (Панель управления -> Система -> Устройства).

Словарь

Interrupt Descriptor Table (IDT) (Таблица дескрипторов прерывания)

Таблица, содержащая дескрипторы прерывания/исключения и указываемая регистром IDTR. Для просмотра этой таблицы используется команда IDT.

MAP file (MAP-файл)

Генерируемый компилятором файл, содержащий отладочную информацию, включающую глобальные символы и, обычно, номера строк исходного текста программы.

MMX (MultiMedia eXtension)

Мультимедийное расширение набора команд процессоров Intel Pentium и Pentium Pro.

Object (Объект)

Представляет собой любой аппаратный или программный ресурс, который может быть разделяемым. Секции иногда также могут выступать в качестве объектов. Смотрите пункт "*Section" (Секция)*.

One-shot breakpoint (Одноразовая точка прерывания)

Контрольная точка, прерывание в которой происходит только один раз. Затем оно отключается и повторно может произойти лишь по другой причине.

Ordinal form (Порядковая форма представления)

Когда символьная таблица не размещена в памяти, то говорят, что она находится в порядковом виде; в этом состоянии все селекторы представляют собой шестнадцатеричные номера секций или сегментов.

Point-and-shoot breakpoint (Точка прерывания "укажи и поставь")

Точка прерывания, устанавливаемая с помощью указания курсором места в коде и выполнением команды BPX или HERE.

Relocate (Размещение)

Приведение адресов в программном коде в соответствие с фактическим адресом загрузки программы.

Section (Секция)

В файле PE-формата блок кода или данных, имеющий определенные атрибуты. Каждая секция имеет свое имя и порядковый номер.

Sticky breakpoint (Постоянная точка прерывания)

Точка прерывания, которая существует до тех пор, пока Вы ее не удалите явным образом. Она сохраняется даже при выгрузке отлаживаемого приложения и его последующей загрузке.

SYM file (.SYM-файл)

Файл, содержащий отладочную информацию, включая глобальные символы и информацию о номерах строк. .SYM-файл обычно транслируется из .MAP-файла.

Symbol table (Таблица символов)

Внутреннее представление в SoftICE отладочной информации, ассоциированной с соответствующим модулем, например, имена функций и номера строк.

Virtual breakpoint (Виртуальное прерывание)

Прерывание, которое может быть установлено на функцию (или переменную) или на номер строки, которые еще не загружены в память.

Перевод выполнен на сайте
http://dore.on.ru

Перевод: ..., Lucifer, Sergey R.
Снимки экрана SoftICE: Григорий Тренин.
Редактирование и верстка: Sergey R.

Июнь 1999 г.