Глава 5: Отладка

Тема на форуме:https://ru-sfera.org/threads/windows-kernel-programming-glava-5-izuchenie-otladchikov.3983/

Как и в любом программном обеспечении, драйверы ядра обычно имеют ошибки.

Отладка драйверов, в отличие от пользовательского режима, более сложный процесс.

Отладка драйвера по сути отладка всей системы, не просто конкретного процесса или процессов. Это требует другого мышления.

В этой главе будет обсуждаться отладка ядра с использованием отладчика WinDbg.

В этой главе:

• Инструменты отладки для Windows.

• Введение в WinDbg.

• Отладка ядра.

• Полная отладка ядра.

• Мануалл по отладке драйверов ядра.

1) Инструменты отладки для Windows

Пакет средств отладки для Windows содержит набор отладчиков, инструментов и документации, сосредоточим внимание на отладчиках в пакете.

Этот пакет может быть установлен как часть Windows SDK или WDK, но никакой реальной «установки» нет.

Установка просто копирует файлы, но не трогает реестр, то есть пакет зависит только от собственных модулей и библиотеки DLL Windows.

Это позволяет легко копировать весь каталог в любой другой каталог, включая на съемный носитель.

Пакет содержит четыре отладчика: Cdb.exe, Ntsd.Exe, Kd.exe и WinDbg.exe.

Вот краткое изложение основных функциональных возможностей каждого отладчика:

• Cdb и Ntsd - пользовательский режим, консольные отладчики. Это означает, что они могут быть прикреплены к процессу, как и любой другой отладчик пользовательского режима.

Оба имеют консольный интерфейс - введите команду, получите ответ и повторить.

Единственная разница между ними заключается в том, что если запустить консольные окна, Cdb использует ту же консоль, тогда как Ntsd открывает новое консольное окно.

В остальном они идентичны.

• Kd - отладчик ядра с консольным пользовательским интерфейсом. Может подключаться к локальному ядру (Локальная отладка ядра, описанна в следующем разделе) или на другую машину.

• WinDbg - единственный отладчик с графическим интерфейсом пользователя. Может работать в режиме пользователя или в режиме отладки ядра, в зависимости от выбора, выполненного из его меню или аргументов командной строки, с которыми он был запущен.

Недавняя альтернатива классического WinDbg - Windbg Preview, доступный в магазине Microsoft.

Это римейк классического отладчика с гораздо лучшим пользовательским интерфейсом и возможностями.

Может быть установлен на Windows 10 версии 1607 или более поздней.

С функциональной точки зрения, это похоже на классический WinDbg. Но он проще в использовании из-за современного, удобного пользовательского интерфейса.

Все команды, которые мы увидим позже в этой главе должны работать одинаково с любым отладчиком.

Хотя эти отладчики могут отличаться друг от друга, на самом деле отладчики в режиме пользователя по сути те же, что и отладчики ядра.

Все они основаны на одном отладчике.

Движок реализован в виде DLL (DbgEng.Dll).

Различные отладчики могут использовать расширения DLL, которые обеспечивают большую часть функций отладчиков.

Механизм отладчика достаточно документирован в документации по средствам отладки для Windows, и так что можно написать новые отладчики, которые используют тот же движок.

Другие инструменты, которые являются частью пакета, включают (частичный список):

• Gflags.exe - инструмент Global Flags, который позволяет устанавливать некоторые флаги ядра и флаги образов.

• ADPlus.exe - создать файл дампа сбоя или зависания процесса.

• Kill.exe - простой инструмент для завершения процесса (ов) на основе идентификатора процесса, имени или шаблона.

• Dumpchk.exe - инструмент для общей проверки файлов дампа.

• TList.exe - список запущенных процессов в системе с различными параметрами.

• Umdh.exe - анализирует распределение кучи в процессах пользовательского режима.

• UsbView.exe - отображает иерархическое представление USB-устройств и концентраторов.

2) Введение в WinDbg

В этом разделе описываются основы WinDbg, но имейте в виду, что все по сути то же самое для консольных отладчиков, за исключением окон графического интерфейса.

WinDbg построен вокруг команд.

Пользователь вводит команду, а отладчик отвечает текстом, описывающим результаты команды.

С GUI некоторые из этих результатов изображены в выделенных окнах, таких как локальные пользователи, стек, потоки и т. д.

WinDbg поддерживает три типа команд:

• Внутренние команды - эти команды встроены в отладчик и работают на

отлаживаемой цели.

• Метакоманды - эти команды начинаются с точки (.) И работают при отладке самого процессв, а не непосредственно на отлаживаемой цели.

• Bang (расширение) команды - эти команды начинаются с восклицательного знака (!). Обеспечивают большую часть возможности отладчика.

По умолчанию отладчик загружает набор предопределенных расширений DLL, но может и больше загружаться из каталога отладчика или из других источников.

Написание расширенных DLL-файлов возможно и полностью задокументировано в документации отладчика. На самом деле, многие такие DLL были созданы и могут быть загружены из соответствующего источника. Эти библиотеки предоставляют новые команды, которые улучшают отладку, часто нацеленные на конкретные сценарии.

3)Уроки. Основы отладки в пользовательском режиме

Если у вас есть опыт работы с WinDbg, вы можете смело пропустить этот раздел.

Этот учебник направлен на получение базового понимания WinDbg и как использовать его для пользовательского режима отладки.

Отладка ядра описана в следующем разделе.

Как правило, есть два способа инициировать отладку в пользовательском режиме - запустить исполняемый файл и прикрепить к нему, или присоединить к уже существующему процессу.

Мы будем использовать последний подход в этом уроке, но за исключением этого первого шага, все остальные операции идентичны.

• Запустите Блокнот.

• Запустите WinDbg (либо Preview, либо классический. На следующих снимках экрана).

• Выберите File / Attach To Process и найдите процесс Notepad в списке (см. Рисунок 5-1). затем нажмите «Прикрепить». Вы должны увидеть результат, похожий на рисунок 5-2.

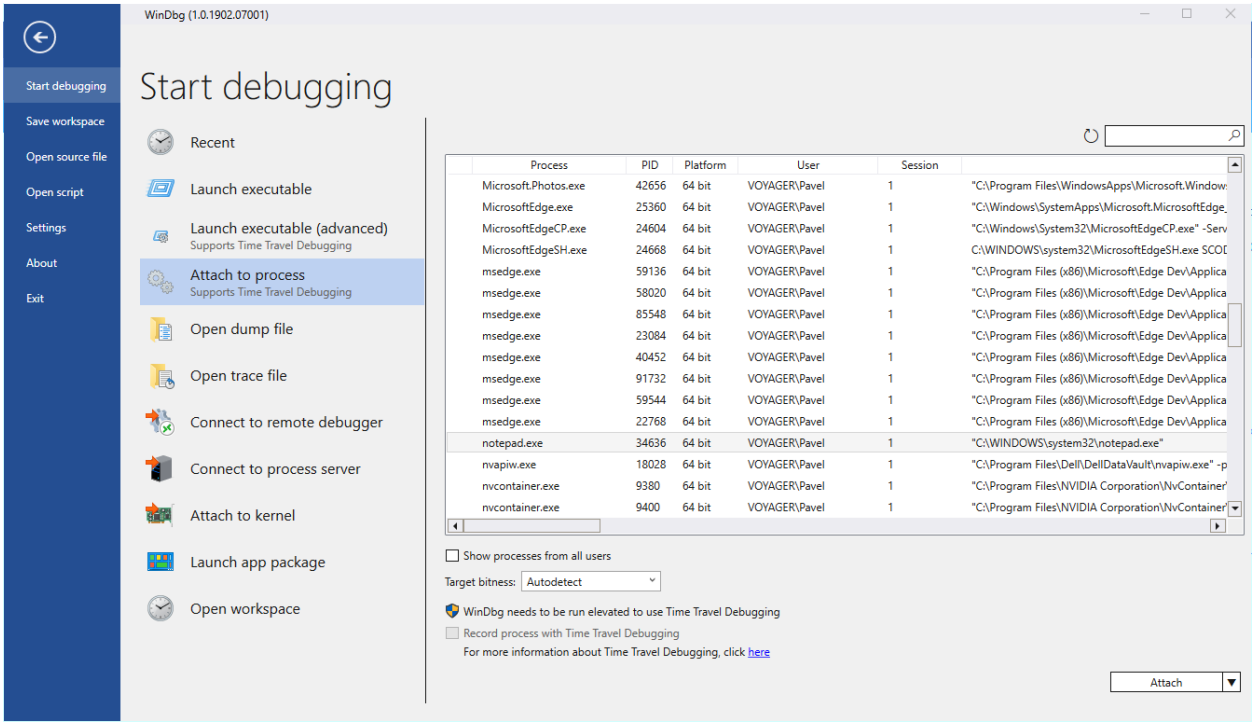


Рисунок 5-1: Присоединение к процессу с WinDbg

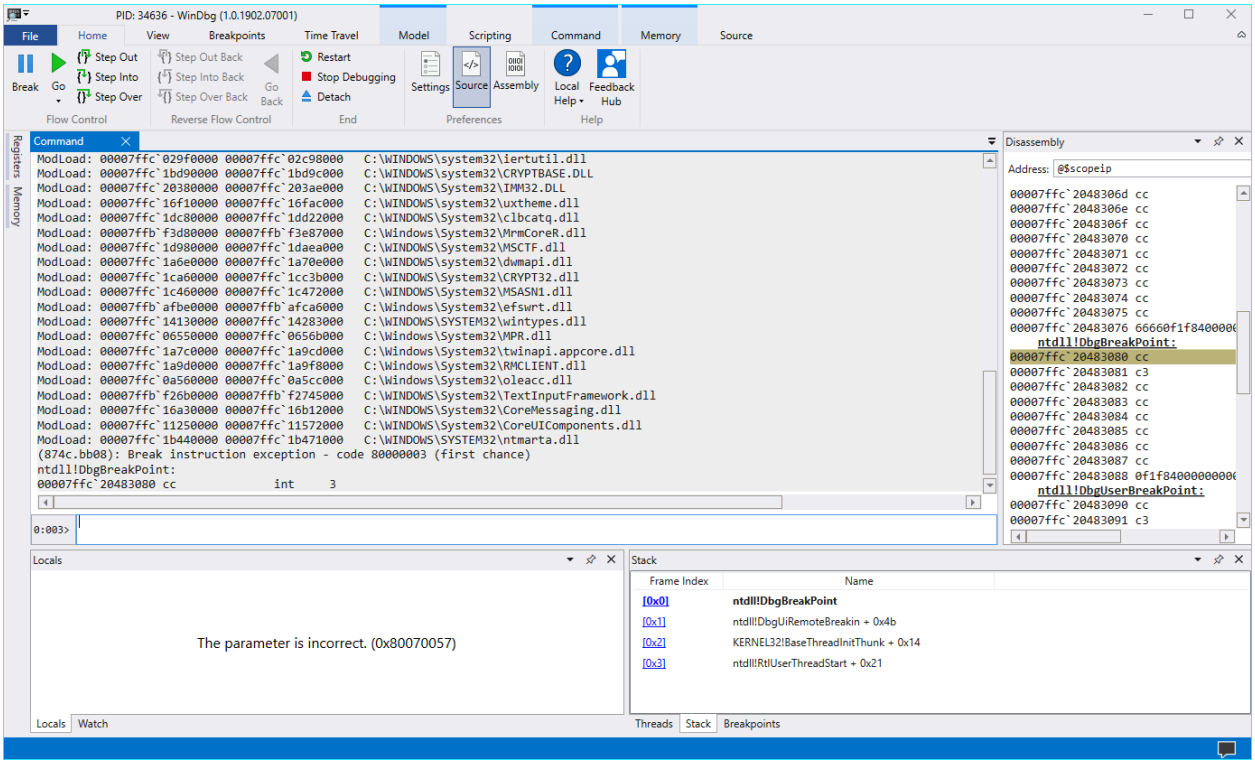


Рисунок 5-2: Первый вид после присоединения процесса

Окно команд является основным интересующим окном - оно всегда должно быть открыто. Это окно показывает различные ответы на команд. Как правило, большая часть сеанса отладки проходит на взаимодействие с этим окном.

Процесс теперь приостановлен - мы находимся в точке останова, вызванной отладчиком.

• Первая команда, которую мы будем использовать, это shows, которая показывает информацию обо всех потоках в отлаженном процессе:

0:003> ~

0 Id:874c.18068 Suspend: 1 Teb: 00000001`2229d000 Unfrozen

1 Id:874c.46ac Suspend: 1 Teb: 00000001`222a5000 Unfrozen

2 Id:874c.152cc Suspend: 1 Teb: 00000001`222a7000 Unfrozen

3 Id:874c.bb08 Suspend: 1 Teb: 00000001`222ab000 Unfrozen

Точное количество потоков, которые вы увидите, может отличаться от показанного здесь.

Одна вещь, которая очень важна - это наличие правильных символов. Microsoft предоставляет общественности сервер символов, который может использоваться Microsoft для поиска символов для большинства модулей. Это важно в любой низкоуровневой отладке.

• Для быстрой установки символов введите команду .symfix.

• Лучший подход - настроить символы один раз и сделать их доступными для дальнейшей отладки.

Для этого добавьте системную переменную среды с именем _NT_SYMBOL_PATH и установите её:

*SRV*c:\Symbols*http://msdl.microsoft.com/download/symbols*

Средняя часть (между звездочками) - это локальный путь для кэширования символов на вашем локальном компьютере.

Вы можно выбрать любой путь, который вам нравится. Как только эта переменная окружения установлена, следующие вызовы отладчика автоматически найдет символы и при необходимости загрузит их с сервера символов Microsoft.

• Чтобы убедиться, что у вас есть правильные символы, введите команду lm (загруженные модули):

```
0:003> lm
start             end                 module name
00007ff7`53820000 00007ff7`53863000   notepad    (deferred)
00007ffb`afbe0000 00007ffb`afca6000   efswrt     (deferred)

(truncated)

00007ffc`1db00000 00007ffc`1dba8000   shcore     (deferred)
00007ffc`1dbb0000 00007ffc`1dc74000   OLEAUT32   (deferred)
00007ffc`1dc80000 00007ffc`1dd22000   clbcatq    (deferred)
00007ffc`1dd30000 00007ffc`1de57000   COMDLG32   (deferred)
00007ffc`1de60000 00007ffc`1f350000   SHELL32    (deferred)
00007ffc`1f500000 00007ffc`1f622000   RPCRT4     (deferred)
00007ffc`1f630000 00007ffc`1f6e3000   KERNEL32   (pdb symbols)          c:\symbols\k\
ernel32.pdb\3B92DED9912D874A2BD08735BC0199A31\kernel32.pdb
00007ffc`1f700000 00007ffc`1f729000   GDI32      (deferred)
00007ffc`1f790000 00007ffc`1f7e2000   SHLWAPI    (deferred)
00007ffc`1f8d0000 00007ffc`1f96e000   sechost    (deferred)
00007ffc`1f970000 00007ffc`1fc9c000   combase    (deferred)
00007ffc`1fca0000 00007ffc`1fd3e000   msvcrt     (deferred)
00007ffc`1fe50000 00007ffc`1fef3000   ADVAPI32   (deferred)
00007ffc`20380000 00007ffc`203ae000   IMM32      (deferred)
00007ffc`203e0000 00007ffc`205cd000   ntdll      (pdb symbols)          c:\symbols\n\
tdll.pdb\E7EEB80BFAA91532B88FF026DC6B9F341\ntdll.pdb
```

Список модулей показывает все модули (DLL и EXE), загруженные в отлаженный процесс на этом время.

Вы можете увидеть начальные и конечные виртуальные адреса, в которых загружен каждый модуль.

Наименование модуля позволяет увидеть статус символа этого модуля (в скобках).

Возможные значения включают в себя:

• deferred - символы для этого модуля никогда не были нужны в этом сеансе отладки и так что не загружаются в это время. Они будут загружены при необходимости.

• символы pdb - это означает, что были загружены правильные публичные символы. Локальный путь PDB.

• экспортировать символы - для этой DLL доступны только экспортированные символы. Это обычно означает, что там нет символов для этого модуля или они не были найдены.

• нет символов - пытались найти символы этого модуля, но ничего не было найдено, даже не экспортируемые символы (такие модули не имеют экспортированных символов, как в случае исполняемых файлов и файлов драйверов).

Вы можете принудительно загрузить символы модуля, используя команду *.reload /f modulename.dll*.

Это обеспечит окончательное подтверждение наличия символов для этого модуля.

Пути к символам также можно настроить в диалоговом окне настроек отладчика.

Откройте меню «Файл / Настройки» и найдите «Настройки отладки». Затем вы можете добавить больше путей для поиска символа. Это полезно при отладке собственного кода, искать можно в ваших каталогах, где могут быть найдены соответствующие файлы PDB (см. рисунок 5-3).

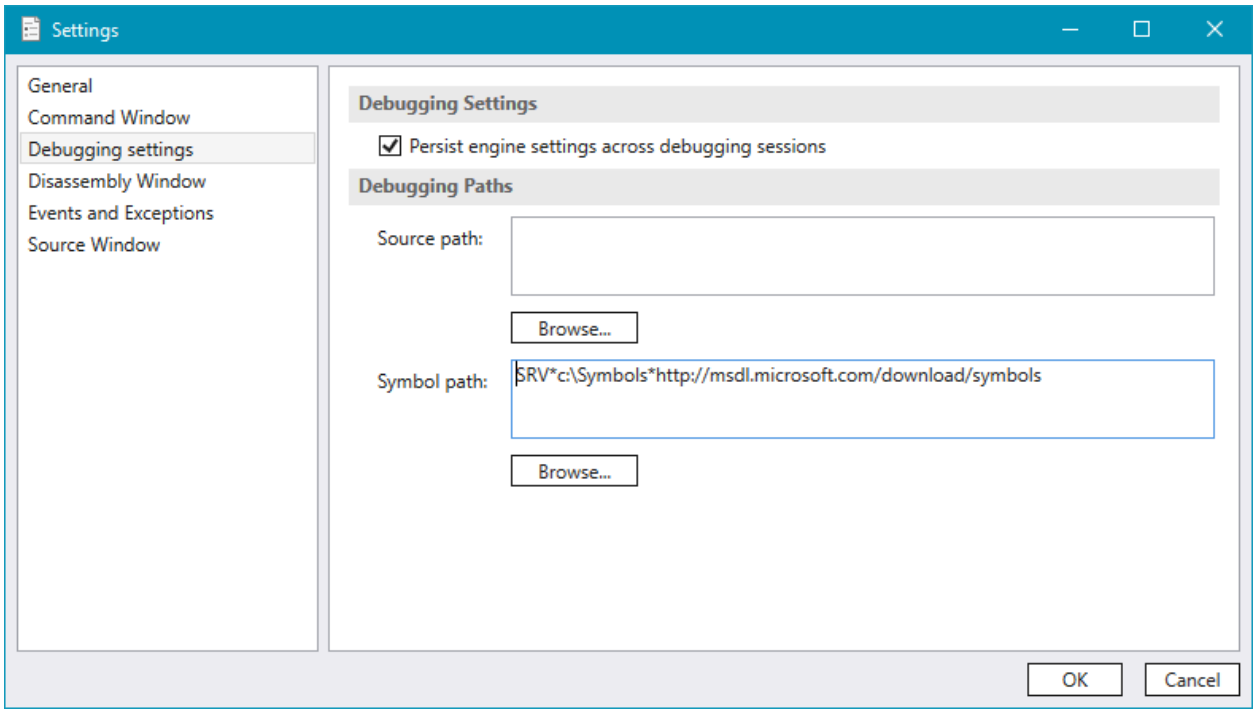


Рисунок 5-3: Конфигурация символов и исходных путей

Вернитесь к списку потоков - обратите внимание, что один из потоков имеет точку перед своими данными. Это текущий поток. Это означает, что любая выданная команда, которая включает поток, где поток не указан, будет работать на этом потоке.

«Текущий поток» также показан в приглашении (*0:003>*) - число справа от двоеточия является текущим индексом потока (3 в этом примере).

• Введите команду k, которая показывает трассировку стека текущего потока:

```
0:003> k
 # Child-SP          RetAddr           Call Site
00 00000001`224ffbd8 00007ffc`204aef5b ntdll!DbgBreakPoint
01 00000001`224ffbe0 00007ffc`1f647974 ntdll!DbgUiRemoteBreakin+0x4b
02 00000001`224ffc10 00007ffc`2044a271 KERNEL32!BaseThreadInitThunk+0x14
03 00000001`224ffc40 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

Вы можете увидеть список вызовов этого потока (конечно, только в режиме пользователя).

Вершиной стека является функция DbgBreakPoint, расположенная в модуле ntdll.dll.

Общий формат адресов с символами имеет «modulename!functionname+offset» (Пример:*ntdll!DbgUiRemoteBreakin+0x4b*). Смещение не является обязательным и может быть нулевым, если это именно начало этой функции.

Также обратите внимание, что имя модуля не имеет расширения.

В приведенном выше выводе DbgBreakpoint был вызван DbgUiRemoteBreakIn, который был вызван BaseThreadInitThunk и так далее.

Этот поток, кстати, был введен отладчиком для принудительного проникновения в цель.

• Чтобы переключиться на другой поток, используйте следующую команду: ns, где n - индекс потока.

Давайте переключимся на поток 0 и затем отобразим его стек вызовов:

```
0:003> ~0s
win32u!NtUserGetMessage+0x14:
00007ffc`1c4b1164 c3              ret
0:000> k
 # Child-SP          RetAddr           Call Site
00 00000001`2247f998 00007ffc`1d802fbd win32u!NtUserGetMessage+0x14
01 00000001`2247f9a0 00007ff7`5382449f USER32!GetMessageW+0x2d
02 00000001`2247fa00 00007ff7`5383ae07 notepad!WinMain+0x267
03 00000001`2247fb00 00007ffc`1f647974 notepad!__mainCRTStartup+0x19f
04 00000001`2247fbc0 00007ffc`2044a271 KERNEL32!BaseThreadInitThunk+0x14
05 00000001`2247fbf0 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

Это основной (первый) поток Блокнота. В верхней части стека показан поток, ожидающий сообщений пользовательского интерфейса.

• Альтернативный способ показать стек вызовов другого потока без переключения на него - это использовать тильду и номер потока перед фактической командой.

Следующий вывод для потока 1:

```
0:000> ~1k
 # Child-SP          RetAddr           Call Site
00 00000001`2267f4c8 00007ffc`204301f4 ntdll!NtWaitForWorkViaWorkerFactory+0x14
01 00000001`2267f4d0 00007ffc`1f647974 ntdll!TppWorkerThread+0x274
02 00000001`2267f7c0 00007ffc`2044a271 KERNEL32!BaseThreadInitThunk+0x14
03 00000001`2267f7f0 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

• Let's go back to the list of threads:

```
.  0  Id: 874c.18068 Suspend: 1 Teb: 00000001`2229d000 Unfrozen
   1  Id: 874c.46ac Suspend: 1 Teb: 00000001`222a5000 Unfrozen
   2  Id: 874c.152cc Suspend: 1 Teb: 00000001`222a7000 Unfrozen
#  3  Id: 874c.bb08 Suspend: 1 Teb: 00000001`222ab000 Unfrozen
```

Обратите внимание, что точка переместилась в нить 0 (текущая нить), показывая хеш-знак (#) в нити 3.

Поток, помеченный как (#), вызвал последнюю точку останова (которая в нашем случае была наша начальное присоединение отладчика).

Основная информация для потока, предоставленная командой is, показана на рисунке 5-4.

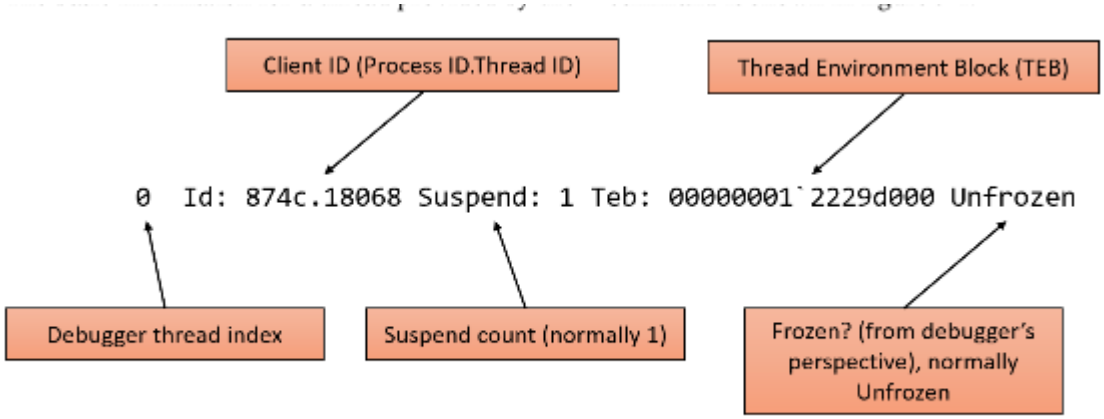


Figure 5-4: Thread information for the ~ command

Большинство чисел, сообщаемых WinDbg, являются шестнадцатеричными по умолчанию.

Чтобы преобразовать значение в десятичное, можно использовать «?».

• Введите следующее, чтобы получить десятичный идентификатор процесса (затем вы можете сравнить его с указанным PID в диспетчере задач):

0:000> ? 874c

Evaluate expression: 34636 = 00000000`0000874c

• Вы можете выражать десятичные числа с префиксом 0n, чтобы вы могли получить и обратный результат:

0:000> ? 0n34636

Evaluate expressio*n: 34636 = 00000000`0000874c*

• Вы можете проверить TEB потока с помощью команды «!teb».

```
0:000> !teb
TEB at 000000012229d000
    ExceptionList:        0000000000000000
    StackBase:            0000000122480000
    StackLimit:           000000012246f000
    SubSystemTib:         0000000000000000
    FiberData:            0000000000001e00
    ArbitraryUserPointer: 0000000000000000
    Self:                 000000012229d000
    EnvironmentPointer:   0000000000000000
    ClientId:             000000000000874c . 0000000000018068
    RpcHandle:            0000000000000000
    Tls Storage:          000001c93676c940
    PEB Address:          000000012229c000
    LastErrorValue:       0
    LastStatusValue:      8000001a
    Count Owned Locks:    0
    HardErrorMode:        0
0:000> !teb 00000001`222a5000
TEB at 00000001222a5000
    ExceptionList:        0000000000000000
    StackBase:            0000000122680000
    StackLimit:           000000012266f000
    SubSystemTib:         0000000000000000
    FiberData:            0000000000001e00
    ArbitraryUserPointer: 0000000000000000
    Self:                 00000001222a5000
    EnvironmentPointer:   0000000000000000
```

Некоторые данные, показанные командой! Teb, относительно известны:

•StackBase и StackLimit - основа стека пользовательского режима и ограничение для потока.

•ClientId - идентификаторы процессов и потоков.

•LastErrorValue - последний код ошибки Win32 (GetLastError).

•TlsStorage - Массив локального хранилища потока (TLS) для этого потока (полное объяснение TLS выходит за рамки этой книги).

• Адрес PEB - адрес блока среды процесса (PEB), который можно просмотреть с помощью «!Peb».

• Команда !Teb (и аналогичные команды) показывает части реальной структуры, в данном случае _TEB. Вы всегда можете посмотреть на реальную структуру, используя «dt» (тип отображения):

```
0:000> dt ntdll!_teb
   +0x000 NtTib            : _NT_TIB
   +0x038 EnvironmentPointer : Ptr64 Void
   +0x040 ClientId         : _CLIENT_ID
   +0x050 ActiveRpcHandle  : Ptr64 Void
   +0x058 ThreadLocalStoragePointer : Ptr64 Void
   +0x060 ProcessEnvironmentBlock : Ptr64 _PEB

   (truncated)

   +0x1808 LockCount        : Uint4B
   +0x180c WowTebOffset     : Int4B
   +0x1810 ResourceRetValue : Ptr64 Void
   +0x1818 ReservedForWdf   : Ptr64 Void
   +0x1820 ReservedForCrt   : Uint8B
   +0x1828 EffectiveContainerId : _GUID
```

Обратите внимание, что WinDbg не учитывает регистр символов. Также обратите внимание на название структуры, начинаются с подчеркивания; именно так, все структуры определены в Windows (пользовательский режим и режим ядра).

Использование имени typedef (без подчеркивания) может работать или не работать, поэтому всегда рекомендуется использовать подчеркивание.

Как узнать, какой модуль определяет структуру, которую вы хотите просмотреть? Если структура задокументирована, модуль будет указан в документации по структуре.

Вы также можете попробовать указать структуру без имени модуля, заставив отладчик искать.

Как правило, вы «знаете», где структура определяется с опытом, а иногда по контексту.

• Если вы прикрепите адрес к предыдущей команде, вы можете получить фактические значения данных:

```
0:000> dt ntdll!_teb 00000001`2229d000
   +0x000 NtTib            : _NT_TIB
   +0x038 EnvironmentPointer : (null)
   +0x040 ClientId         : _CLIENT_ID
   +0x050 ActiveRpcHandle  : (null)
   +0x058 ThreadLocalStoragePointer : 0x000001c9`3676c940 Void
   +0x060 ProcessEnvironmentBlock : 0x00000001`2229c000 _PEB
   +0x068 LastErrorValue   : 0

    (truncated)

   +0x1808 LockCount       : 0
   +0x180c WowTebOffset    : 0n0
   +0x1810 ResourceRetValue : 0x000001c9`3677fd00 Void
   +0x1818 ReservedForWdf  : (null)
   +0x1820 ReservedForCrt  : 0
   +0x1828 EffectiveContainerId : _GUID {00000000-0000-0000-0000-000000000000}
```

Каждый элемент отображается со смещением от начала структуры, его именем и значением.

Простые значения отображаются напрямую, а структурные значения (такие как NtTib выше) обычно отображаются с гиперссылкой. Нажав на эту гиперссылку, вы получите подробную информацию о структуре.

• Нажмите на член NtTib выше, чтобы показать детали этого члена данных:

```
0:000> dx -r1 (*((ntdll!_NT_TIB *)0x12229d000))
(*((ntdll!_NT_TIB *)0x12229d000))                 [Type: _NT_TIB]
    [+0x000] ExceptionList    : 0x0 [Type: _EXCEPTION_REGISTRATION_RECORD *]
    [+0x008] StackBase        : 0x122480000 [Type: void *]
    [+0x010] StackLimit       : 0x12246f000 [Type: void *]
    [+0x018] SubSystemTib     : 0x0 [Type: void *]
    [+0x020] FiberData        : 0x1e00 [Type: void *]
    [+0x020] Version          : 0x1e00 [Type: unsigned long]
    [+0x028] ArbitraryUserPointer : 0x0 [Type: void *]
    [+0x030] Self             : 0x12229d000 [Type: _NT_TIB *]
```

Отладчик использует более новую команду dx для просмотра данных.

Если вы не видите гиперссылки, возможно, вы используете очень старую WinDbg, где язык разметки отладчика (DML) не включен по умолчанию. Вы можете включить его с помощью команды .prefer_dml 1.

Теперь давайте обратим наше внимание на контрольные точки. Давайте установим точку останова, когда файл будет открыт блокнотом.

• Введите следующую команду, чтобы установить точку останова в функции API CreateFile:

0: 000> bp kernel32! Createfilew

Обратите внимание, что имя функции на самом деле CreateFileW, так как нет функции с именем CreateFile.

• Вы можете перечислить существующие точки останова с помощью команды bl:

```
0:000> bl
  0 e Disable Clear  00007ffc`1f652300  0001 (0001)  0:**** KERNEL32!CreateFileW
```

Вы можете увидеть индекс точки останова (0), независимо от того, включен он или нет (e = включено, d = отключено) и Вы получаете гиперссылки, чтобы отключить (команда bd) и удалить (команда bc) точку останова.

Теперь давайте продолжим выполнение блокнота, пока не достигнет точки останова:

• Введите команду g или нажмите кнопку «Перейти» на панели инструментов, или нажмите F5:

Вы увидите, что отладчик показывает Busy в командной строке, а область команд показывает, что Debuggee, это означает, что вы не можете вводить команды до следующего перерыва.

• Блокнот теперь должен быть активным. Перейдите в меню «Файл» и выберите «Открыть».

Отладчик должен остановить выполнение блокнота:

```
Breakpoint 0 hit
KERNEL32!CreateFileW:
00007ffc`1f652300 ff25aa670500    jmp     qword ptr [KERNEL32!_imp_CreateFileW (0000\
7ffc`1f6a8ab0)] ds:00007ffc`1f6a8ab0={KERNELBASE!CreateFileW (00007ffc`1c75e260)}
```

• Мы достигли точки останова! Обратите внимание на поток, в котором это произошло. Давайте посмотрим, что вызов выглядит как стек (может потребоваться некоторое время, чтобы показать, нужно ли отладчику загружать символы:

```
0:002> k
 # Child-SP          RetAddr           Call Site
00 00000001`226fab08 00007ffc`061c8368 KERNEL32!CreateFileW
01 00000001`226fab10 00007ffc`061c5d4d mscoreei!RuntimeDesc::VerifyMainRuntimeModule\
+0x2c
02 00000001`226fab60 00007ffc`061c6068 mscoreei!FindRuntimesInInstallRoot+0x2fb
03 00000001`226fb3e0 00007ffc`061cb748 mscoreei!GetOrCreateSxSProcessInfo+0x94
04 00000001`226fb460 00007ffc`061cb62b mscoreei!CLRMetaHostPolicyImpl::GetRequestedR\
untimeHelper+0xfc
05 00000001`226fb740 00007ffc`061ed4e6 mscoreei!CLRMetaHostPolicyImpl::GetRequestedR\
untime+0x120

(truncated)

21 00000001`226fede0 00007ffc`1df025b2 SHELL32!CFSIconOverlayManager::LoadNonloadedO\
```

Что мы можем сделать на этом этапе? Вы можете задаться вопросом, какой файл открывается.

Мы можем получить эту информацию, основанной на соглашении о вызовах функции CreateFileW.

Так как это 64-битный процесс (а процессор Intel/AMD), соглашение о вызовах гласит, что первое целое число/указатель аргументы передаются в регистрах RCX, RDX, R8 и R9.

Поскольку имя файла в CreateFileW, это первый аргумент, соответствующий регистр — RCX.

• Отобразите значение регистра RCX с помощью команды r (вы получите другое значение):

0:002> r rcx

rcx=00000001226fabf8

• Мы можем просматривать память, указанную RCX, с помощью различных команд d (отображения).

```
0:002> db 00000001226fabf8
00000001`226fabf8  43 00 3a 00 5c 00 57 00-69 00 6e 00 64 00 6f 00  C.:.\.W.i.n.d.o.
00000001`226fac08  77 00 73 00 5c 00 4d 00-69 00 63 00 72 00 6f 00  w.s.\.M.i.c.r.o.
00000001`226fac18  73 00 6f 00 66 00 74 00-2e 00 4e 00 45 00 54 00  s.o.f.t...N.E.T.
00000001`226fac28  5c 00 46 00 72 00 61 00-6d 00 65 00 77 00 6f 00  \.F.r.a.m.e.w.o.
00000001`226fac38  72 00 6b 00 36 00 34 00-5c 00 5c 00 76 00 32 00  r.k.6.4.\.\.v.2.
00000001`226fac48  2e 00 30 00 2e 00 35 00-30 00 37 00 32 00 37 00  ..0...5.0.7.2.7.
00000001`226fac58  5c 00 63 00 6c 00 72 00-2e 00 64 00 6c 00 6c 00  \.c.l.r...d.l.l.
00000001`226fac68  00 00 76 1c fc 7f 00 00-00 00 00 00 00 00 00 00  ..v.............
```

Команда db показывает память в байтах и символы ASCII справа.

Теперь понятно каково имя файла, но поскольку строка является Unicode, это не очень удобно для просмотра.

• Используйте команду du для более удобного просмотра строки Unicode:

0:002> du 00000001226fabf8

00000001`226fabf8 "C:\Windows\Microsoft.NET\Framewo"

00000001`226fac38 "rk64\\v2.0.50727\clr.dll"

• Вы можете использовать значение регистра напрямую, добавив к его имени префикс @:

0:002> du @rcx

00000001`226fabf8

00000001`226fac38

"C:\Windows\Microsoft.NET\Framewo"

"rk64\\v2.0.50727\clr.dll"

Теперь давайте установим другую точку останова в нативном API, который вызывается CreateFileW — NtCreateFile:

```
0:002> bp ntdll!ntcreatefile
0:002> bl
   0 e Disable Clear  00007ffc`1f652300  0001 (0001)  0:**** KERNEL32!CreateFileW
   1 e Disable Clear  00007ffc`20480120  0001 (0001)  0:**** ntdll!NtCreateFile
```

Обратите внимание, что нативный API никогда не использует W или A - он всегда работает со строками Unicode.

• Продолжите выполнение с помощью команды g.

Breakpoint 1 hit

ntdll!NtCreateFile:

00007ffc`20480120 4c8bd1 mov r10,rcx

• Check the call stack again:

```
0:002> k
 # Child-SP          RetAddr           Call Site
00 00000001`226fa938 00007ffc`1c75e5d6 ntdll!NtCreateFile
01 00000001`226fa940 00007ffc`1c75e2c6 KERNELBASE!CreateFileInternal+0x2f6
02 00000001`226faab0 00007ffc`061c8368 KERNELBASE!CreateFileW+0x66
03 00000001`226fab10 00007ffc`061c5d4d mscoreei!RuntimeDesc::VerifyMainRuntimeModule\
+0x2c
04 00000001`226fab60 00007ffc`061c6068 mscoreei!FindRuntimesInInstallRoot+0x2fb
05 00000001`226fb3e0 00007ffc`061cb748 mscoreei!GetOrCreateSxSProcessInfo+0x94

(truncated)
```

•Перечислите следующие 8 инструкций, которые должны быть выполнены с помощью команды u (unassemble):

```
0:002> u
ntdll!NtCreateFile:
00007ffc`20480120 4c8bd1          mov     r10,rcx
00007ffc`20480123 b855000000      mov     eax,55h
00007ffc`20480128 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`\
7ffe0308)],1
00007ffc`20480130 7503            jne     ntdll!NtCreateFile+0x15 (00007ffc`20480135)
00007ffc`20480132 0f05            syscall
00007ffc`20480134 c3              ret
00007ffc`20480135 cd2e            int     2Eh
00007ffc`20480137 c3              ret
```

Обратите внимание, что значение 0x55 копируется в регистр EAX. Это сервисный номер системы для NtCreateFile, как описано в главе 1. Показанная инструкция syscall является той, которая вызывает переход к ядру, а затем выполнить саму системную службу NtCreateFile.

• Вы можете перейти к следующей инструкции с помощью команды p (шаг - нажмите F10 в качестве альтернативы). Вы может войти в функцию (в случае сборки это инструкция вызова) с помощью команды t (трассировка - нажмите F11 как альтернативу):

```
0:002> p
Breakpoint 1 hit
ntdll!NtCreateFile:
00007ffc`20480120 4c8bd1          mov     r10,rcx
0:002> p
ntdll!NtCreateFile+0x3:
00007ffc`20480123 b855000000      mov     eax,55h
0:002> p
ntdll!NtCreateFile+0x8:
00007ffc`20480128 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`\
7ffe0308)],1 ds:00000000`7ffe0308=00
0:002> p
ntdll!NtCreateFile+0x10:
00007ffc`20480130 7503            jne     ntdll!NtCreateFile+0x15 (00007ffc`20480135\
) [br=0]
0:002> p
ntdll!NtCreateFile+0x12:
00007ffc`20480132 0f05            syscall
```

• Вход в системный вызов невозможен, так как мы находимся в режиме пользователя.

0:002> p

ntdll!NtCreateFile+0x14:

00007ffc`20480134 c3 ret

• Возвращаемое значение функций в соглашении о вызовах x64 сохраняется в EAX или RAX.

Для системы, это NTSTATUS, поэтому EAX содержит возвращенный статус:

0:002> r eax

eax=c0000034

• У нас есть код ошибки. Мы можем получить подробности с помощью команды! Error:

0:002> !error @eax

Error code: (NTSTATUS) 0xc0000034 (3221225524) - Object Name not found.

• Отключите все точки останова и дайте Notepad продолжить работу в обычном режиме:

*0:002> bd **

0:002> g

Поскольку в настоящее время у нас нет точек останова, мы можем принудительно выполнить разрыв, нажав кнопку «Разрыв» на панель инструментов, или нажав Ctrl + Break на клавиатуре:

```
874c.16a54): Break instruction exception - code 80000003 (first chance)
ntdll!DbgBreakPoint:
00007ffc`20483080 cc              int     3
```

• Обратите внимание на номер потока в приглашении. Показать все текущие потоки:

```
0:022> ~
   0  Id: 874c.18068 Suspend: 1 Teb: 00000001`2229d000 Unfrozen
   1  Id: 874c.46ac Suspend: 1 Teb: 00000001`222a5000 Unfrozen
   2  Id: 874c.152cc Suspend: 1 Teb: 00000001`222a7000 Unfrozen
   3  Id: 874c.f7ec Suspend: 1 Teb: 00000001`222ad000 Unfrozen
   4  Id: 874c.145b4 Suspend: 1 Teb: 00000001`222af000 Unfrozen

(truncated)

  18  Id: 874c.f0c4 Suspend: 1 Teb: 00000001`222d1000 Unfrozen
  19  Id: 874c.17414 Suspend: 1 Teb: 00000001`222d3000 Unfrozen
  20  Id: 874c.c878 Suspend: 1 Teb: 00000001`222d5000 Unfrozen
  21  Id: 874c.d8c0 Suspend: 1 Teb: 00000001`222d7000 Unfrozen
. 22  Id: 874c.16a54 Suspend: 1 Teb: 00000001`222e1000 Unfrozen
  23  Id: 874c.10838 Suspend: 1 Teb: 00000001`222db000 Unfrozen
  24  Id: 874c.10cf0 Suspend: 1 Teb: 00000001`222dd000 Unfrozen
```

Много потоков, верно? Они были фактически созданы/вызваны общим открытым диалогом.

• Продолжайте исследовать отладчик так, как хотите!

• Если вы закроете Блокнот, вы достигнете точки останова при завершении процесса:

```
ntdll!NtTerminateProcess+0x14:
00007ffc`2047fc14 c3              ret
0:000> k
 # Child-SP          RetAddr           Call Site
00 00000001`2247f6a8 00007ffc`20446dd8 ntdll!NtTerminateProcess+0x14
01 00000001`2247f6b0 00007ffc`1f64d62a ntdll!RtlExitUserProcess+0xb8
02 00000001`2247f6e0 00007ffc`061cee58 KERNEL32!ExitProcessImplementation+0xa
03 00000001`2247f710 00007ffc`0644719e mscoreei!RuntimeDesc::ShutdownAllActiveRuntim\
es+0x287
04 00000001`2247fa00 00007ffc`1fcda291 mscoree!ShellShim_CorExitProcess+0x11e
05 00000001`2247fa30 00007ffc`1fcda2ad msvcrt!_crtCorExitProcess+0x4d
06 00000001`2247fa60 00007ffc`1fcda925 msvcrt!_crtExitProcess+0xd
07 00000001`2247fa90 00007ff7`5383ae1e msvcrt!doexit+0x171
08 00000001`2247fb00 00007ffc`1f647974 notepad!__mainCRTStartup+0x1b6
09 00000001`2247fbc0 00007ffc`2044a271 KERNEL32!BaseThreadInitThunk+0x14
0a 00000001`2247fbf0 00000000`00000000 ntdll!RtlUserThreadStart+0x21
```

• Вы можете использовать команду q для выхода из отладчика. Если процесс еще жив, он будет прерван.

Альтернативой является использование команды .detach для отключения от цели без ее уничтожения.

3) Отладка ядра

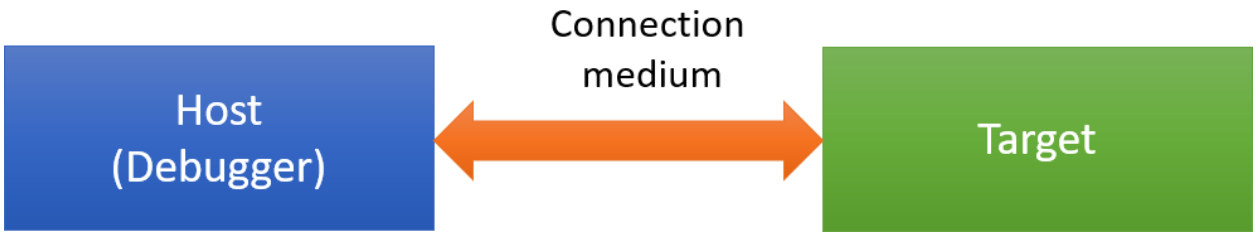
Отладка в пользовательском режиме включает в себя отладчик, присоединяющийся к процессу, устанавливающий точки останова и т.д. Отладка режима ядра с другой стороны, включает в себя управление всей машиной с помощью отладчика.

Это означает, что если точка останова установлена, то вся машина остановится.

Очевидно, что это не может быть достигнуто с помощью одной машины. В полном объеме отладка ядра, задействованы две машины: хост (где работает отладчик) и цель (То-что отлаживается).

Однако целью может быть виртуальная машина, размещенная на той же машине (хосте), где отладчик выполняется.

Рисунок 5-5 показывает хост и цель, подключенные через некоторую среду соединения.



Прежде чем мы перейдем к полной отладке ядра, мы рассмотрим его более простой родственник — локальная отладка адра.

Локальная отладка ядра

Локальная отладка ядра (LKD) позволяет просматривать системную память и другую системную информацию на местной машине.

Основное различие между локальной и полной отладкой ядра заключается в том, что с LKD нет никакого способа установить точки останова, что означает, что вы всегда смотрите на текущее состояние системы.

Это также означает, что все меняется, даже когда выполняются команды, поэтому некоторая информация может быть ненадежной. При полной отладке ядра команды можно вводить только во время, когда целевая система находится в точке останова, поэтому состояние системы не изменяется.

Чтобы настроить LKD, введите в командной строке с повышенными правами следующее и перезапустите систему:

bcdedit / debug on

После перезапуска системы запустите WinDbg с повышенными привилегиями. Выберите меню File/Attach To Kernel (WinDbg preview) или File/Kernel Debug (классический WinDbg). Выберите вкладку Local и нажмите ОК. Вы должны увидеть вывод, похожий на следующий:

```
Microsoft (R) Windows Debugger Version 10.0.18317.1001 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Connected to Windows 10 18362 x64 target at (Sun Apr 21 08:50:59.964 2019 (UTC + 3:0\
0)), ptr64 TRUE

************* Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       SRV*c:\Symbols*http://msdl.microsoft.\
com/download/symbols
Symbol search path is: c:\temp;SRV*c:\Symbols*http://msdl.microsoft.com/download/sym\
bols
Executable search path is:
Windows 10 Kernel Version 18362 MP (12 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 18362.1.amd64fre.19h1_release.190318-1202
Machine Name:
Kernel base = 0xfffff806`466b8000 PsLoadedModuleList = 0xfffff806`46afb2d0
Debug session time: Sun Apr 21 08:51:00.702 2019 (UTC + 3:00)
System Uptime: 0 days 11:33:37.265
```

Отладка локального ядра защищена безопасной загрузкой в Windows 10, Server 2016 и более поздних версиях.

Чтобы активировать LKD, вам необходимо отключить безопасную загрузку в настройках BIOS машины. Если по любой причине, это невозможно, есть альтернатива, использующая Sysinternals LiveKd инструмент.

Скопируйте LiveKd.exe в основной каталог Debugging Tools for Windows.

После запустите WinDbg с использованием LiveKd с помощью следующей команды: livekd -w.

Обратите внимание, что приглашение отображает lkd. Это означает, что локальная отладка ядра активна.

4)Мануал по отладке локального ядра

Если вы знакомы с командами отладки ядра, вы можете смело пропустить этот раздел.

• Вы можете отобразить основную информацию для всех процессов, запущенных в системе вместе с процессом 0 0 команда:

```
lkd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS ffff8d0e682a73c0
    SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 001ad002  ObjectTable: ffffe20712204b80  HandleCount: 9542.
    Image: System

PROCESS ffff8d0e6832e140
    SessionId: none  Cid: 0058    Peb: 00000000  ParentCid: 0004
    DirBase: 03188002  ObjectTable: ffffe2071220cac0  HandleCount:   0.
    Image: Secure System

PROCESS ffff8d0e683f1080
    SessionId: none  Cid: 0098    Peb: 00000000  ParentCid: 0004
    DirBase: 003e1002  ObjectTable: ffffe20712209480  HandleCount:   0.
    Image: Registry

PROCESS ffff8d0e83099080
    SessionId: none  Cid: 032c    Peb: 5aba7eb000  ParentCid: 0004
    DirBase: 15fa39002  ObjectTable: ffffe20712970080  HandleCount:  53.
    Image: smss.exe

(truncated)
```

Для каждого процесса отображается следующая информация:

• Адрес, присоединенный к тексту PROCESS, является адресом процесса (в пространстве ядра, конечно).

• SessionId - сессия, в котором выполняется процесс.

• Cid - (идентификатор клиента) уникальный идентификатор процесса.

• Peb - адрес блока среды процесса (PEB). Этот адрес находится в пространстве пользователя, естественно.

• ParentCid - (идентификатор родительского процесса) идентификатор родительского процесса. Обратите внимание, что возможно родительский процесс больше не существует, и этот идентификатор можно использовать повторно.

• DirBase - физический адрес (без младших 12 бит) каталога главной страницы для этого процесса, используется в качестве основы для виртуальной трансляции адресов.

На x64 это известно как Page Map Level 4, а на x86 это таблица указателей страниц (PDPT).

• ObjectTable - указатель на приватную таблицу дескрипторов процесса.

• HandleCount - количество дескрипторов в этом процессе.

• Image - имя исполняемого файла или имя специального процесса для тех, кто не связан с исполняемым файлом (примеры: защищенная система, система сжатие памяти).

Команда !Process принимает как минимум два аргумента. Первый указывает на интересующий процесс используя его адрес EPROCESS, где ноль означает «все или любой процесс».

Второй аргумент - это уровень необходимых деталей, где ноль означает наименьшее количество деталей (битовая маска).

Третий аргумент может быть добавлен для поиска конкретного исполняемого файла.

• Перечислить все процессы, на которых запущен csrss.exe:

```
lkd> !process 0 0 csrss.exe
PROCESS ffff8d0e83c020c0
    SessionId: 0  Cid: 038c    Peb: f599af6000  ParentCid: 0384
    DirBase: 844eaa002  ObjectTable: ffffe20712345480  HandleCount: 992.
    Image: csrss.exe

PROCESS ffff8d0e849df080
    SessionId: 1  Cid: 045c    Peb: e8a8c9c000  ParentCid: 0438
    DirBase: 17afc1002  ObjectTable: ffffe207186d93c0  HandleCount: 1146.
    Image: csrss.exe
```

• Перечислить больше информации для определенного процесса, указав его адрес и более высокий уровень детализации:

```
lkd> !process ffff8d0e849df080 1
PROCESS ffff8d0e849df080
    SessionId: 1  Cid: 045c    Peb: e8a8c9c000  ParentCid: 0438
    DirBase: 17afc1002  ObjectTable: ffffe207186d93c0  HandleCount: 1138.
    Image: csrss.exe
    VadRoot ffff8d0e999a4840 Vads 244 Clone 0 Private 670. Modified 48241. Locked 38\
106.
    DeviceMap ffffe20712213720
    Token                             ffffe207186f38f0
    ElapsedTime                       12:14:47.292
    UserTime                          00:00:00.000
    KernelTime                        00:00:03.468
    QuotaPoolUsage[PagedPool]         423704
    QuotaPoolUsage[NonPagedPool]      37752
    Working Set Sizes (now,min,max)  (1543, 50, 345) (6172KB, 200KB, 1380KB)
    PeakWorkingSetSize                10222
    VirtualSize                       2101434 Mb
    PeakVirtualSize                   2101467 Mb
    PageFaultCount                    841489
    MemoryPriority                    BACKGROUND
    BasePriority                      13
    CommitCharge                      1012
    Job                               ffff8d0e83da8080
```

Как видно из вышеприведенного вывода, отображается дополнительная информация о процессе.

Некоторые из этой информации является гиперссылкой, что облегчает дальнейшее изучение.

Job, частью которой является этот процесс является гиперссылкой.

• Нажмите на гиперссылку с адресом работы Job:

```
lkd> !job ffff8d0e83da8080
Job at ffff8d0e83da8080
  Basic Accounting Information
    TotalUserTime:             0x33db258
    TotalKernelTime:           0x5705d50
    TotalCycleTime:            0x73336f9ae
    ThisPeriodTotalUserTime:   0x33db258
    ThisPeriodTotalKernelTime: 0x5705d50
    TotalPageFaultCount:       0x8617c
    TotalProcesses:            0x3e
    ActiveProcesses:           0xd
    FreezeCount:               0

    BackgroundCount:           0
    TotalTerminatedProcesses:  0x0
    PeakJobMemoryUsed:         0x38fb5
    PeakProcessMemoryUsed:     0x29366
  Job Flags
    [wake notification allocated]
    [wake notification enabled]
    [timers virtualized]
  Limit Information (LimitFlags: 0x1800)
  Limit Information (EffectiveLimitFlags: 0x1800)
    JOB_OBJECT_LIMIT_BREAKAWAY_OK
    JOB_OBJECT_LIMIT_SILENT_BREAKAWAY_OK
```

Job - это объект, который содержит один или несколько процессов, для которых могут применяться различные ограничения.

Подробное обсуждение Job выходит за рамки этой книги. Дополнительную информацию можно найти в книгах по внутренним компонентам Windows.

• Как обычно, такая команда, как! Job, скрывает некоторую информацию, доступную в реальной структуре данных.

В данном случае это EJOB. Используйте команду dt nt! _Ejob с адресом задания, чтобы увидеть все детали.

• Также можно просмотреть PEB процесса, щелкнув его гиперссылку. Это похоже на !Peb команду, которая используется в пользовательском режиме, но здесь должен обязательно быть задан правильный контекст процесса.

Нажмите на гиперссылку Peb. Вы должны увидеть что-то вроде этого:

```
lkd> .process /p ffff8d0e849df080; !peb e8a8c9c000
Implicit process is now ffff8d0e`849df080
PEB at 000000e8a8c9c000
    InheritedAddressSpace:    No
    ReadImageFileExecOptions: No
    BeingDebugged:            No
    ImageBaseAddress:         00007ff62fc70000
    NtGlobalFlag:             4400
    NtGlobalFlag2:            0
    Ldr                       00007ffa0ecc53c0
    Ldr.Initialized:          Yes
    Ldr.InInitializationOrderModuleList: 000002021cc04dc0 . 000002021cc15f00
    Ldr.InLoadOrderModuleList:           000002021cc04f30 . 000002021cc15ee0
    Ldr.InMemoryOrderModuleList:         000002021cc04f40 . 000002021cc15ef0
                    Base TimeStamp                     Module



        7ff62fc70000 78facb67 Apr 27 01:06:31 2034 C:\WINDOWS\system32\csrss.exe
        7ffa0eb60000 a52b7c6a Oct 23 22:22:18 2057 C:\WINDOWS\SYSTEM32\ntdll.dll
        7ffa0ba10000 802fce16 Feb 24 11:29:58 2038 C:\WINDOWS\SYSTEM32\CSRSRV.dll
        7ffa0b9f0000 94c740f0 Feb 04 23:17:36 2049 C:\WINDOWS\system32\basesrv.D\
LL

(truncated)
```

Правильный контекст процесса задается мета-командой .process, а затем отображается PEB.

Это общая техника, которую вы должны использовать, чтобы показать информацию, которая находится в пространстве пользователя.

• Повторите команду! Process еще раз, но на этот раз без уровня детализации.

Будет показано больше информации для процесса:

```
        DeviceMap                 ffffe20712213720
        Owning Process            ffff8d0e849df080       Image:         csrss.exe
        Attached Process          N/A            Image:         N/A
        Wait Start TickCount      2856062        Ticks: 70 (0:00:00:01.093)
        Context Switch Count      6483           IdealProcessor: 8
        UserTime                  00:00:00.421
        KernelTime                00:00:00.437
        Win32 Start Address 0x00007ffa0ba15670
        Stack Init ffff83858295fb90 Current ffff83858295f340
        Base ffff838582960000 Limit ffff838582959000 Call 0000000000000000
        Priority 14 BasePriority 13 PriorityDecrement 0 IoPriority 2 PagePriority 5
GetContextState failed, 0x80004001
Unable to get current machine context, HRESULT 0x80004001
        Child-SP          RetAddr           Call Site
        ffff8385`8295f380 fffff806`466e98c2 nt!KiSwapContext+0x76
        ffff8385`8295f4c0 fffff806`466e8f54 nt!KiSwapThread+0x3f2
        ffff8385`8295f560 fffff806`466e86f5 nt!KiCommitThreadWait+0x144
        ffff8385`8295f600 fffff806`467d8c56 nt!KeWaitForSingleObject+0x255
        ffff8385`8295f6e0 fffff806`46d76c70 nt!AlpcpWaitForSingleObject+0x3e
        ffff8385`8295f720 fffff806`46d162cc nt!AlpcpCompleteDeferSignalRequestAndWai\
t+0x3c
        ffff8385`8295f760 fffff806`46d15321 nt!AlpcpReceiveMessagePort+0x3ac
        ffff8385`8295f7f0 fffff806`46d14e05 nt!AlpcpReceiveMessage+0x361
        ffff8385`8295f8d0 fffff806`46885e95 nt!NtAlpcSendWaitReceivePort+0x105
        ffff8385`8295f990 00007ffa`0ebfd194 nt!KiSystemServiceCopyEnd+0x25 (TrapFram\
e @ ffff8385`8295fa00)
        000000e8`a8e3f798 00007ffa`0ba15778 0x00007ffa`0ebfd194
        000000e8`a8e3f7a0 00000202`1cc85090 0x00007ffa`0ba15778
        000000e8`a8e3f7a8 00000000`00000000 0x00000202`1cc85090

        THREAD ffff8d0e84bbf140  Cid 045c.066c  Teb: 000000e8a8ca9000 Win32Thread: f\
fff8d0e865f4760 WAIT: (WrLpcReply) UserMode Non-Alertable
            ffff8d0e84bbf798  Semaphore Limit 0x1

(truncated)
```

Команда перечисляет все потоки в процессе. Каждый поток представлен своим адресом ETHREAD прилагается к тексту “THREAD”.

Стек вызовов также указан в списке - префикс модуля «nt» представляет ядро - нет необходимости использовать «настоящее» имя модуля ядра.

Одна из причин использования «nt» вместо явного указания имени модуля ядра состоит в том, что они различаются между 64 и 32-разрядными системами (ntoskrnl.exe на 64-разрядных и как минимум два варианта на 32 бита).

Символы пользовательского режима не загружаются по умолчанию, поэтому стеки потоков, которые охватывают пользовательский режим, показывают просто числовые адреса.

Вы можете явно загружать пользовательские символы с помощью .reload / user:

```
lkd> .reload /user
Loading User Symbols
.....................
lkd> !process ffff8d0e849df080
PROCESS ffff8d0e849df080
    SessionId: 1  Cid: 045c    Peb: e8a8c9c000  ParentCid: 0438
    DirBase: 17afc1002  ObjectTable: ffffe207186d93c0  HandleCount: 1149.
    Image: csrss.exe

(truncated)

        THREAD ffff8d0e849e0080  Cid 045c.046c  Teb: 000000e8a8ca3000 Win32Thread: f\
fff8d0e865f37c0 WAIT: (WrLpcReceive) UserMode Non-Alertable
            ffff8d0e849e06d8  Semaphore Limit 0x1
        Not impersonating
        DeviceMap                 ffffe20712213720
        Owning Process            ffff8d0e849df080       Image:         csrss.exe
        Attached Process          N/A            Image:         N/A
        Wait Start TickCount      2895071        Ticks: 135 (0:00:00:02.109)
        Context Switch Count      6684           IdealProcessor: 8
        UserTime                  00:00:00.437
        KernelTime                00:00:00.437
        Win32 Start Address CSRSRV!CsrApiRequestThread (0x00007ffa0ba15670)
        Stack Init ffff83858295fb90 Current ffff83858295f340
        Base ffff838582960000 Limit ffff838582959000 Call 0000000000000000
        Priority 14 BasePriority 13 PriorityDecrement 0 IoPriority 2 PagePriority 5
GetContextState failed, 0x80004001
Unable to get current machine context, HRESULT 0x80004001
        Child-SP          RetAddr           Call Site
        ffff8385`8295f380 fffff806`466e98c2 nt!KiSwapContext+0x76
        ffff8385`8295f4c0 fffff806`466e8f54 nt!KiSwapThread+0x3f2
        ffff8385`8295f560 fffff806`466e86f5 nt!KiCommitThreadWait+0x144
        ffff8385`8295f600 fffff806`467d8c56 nt!KeWaitForSingleObject+0x255
        ffff8385`8295f6e0 fffff806`46d76c70 nt!AlpcpWaitForSingleObject+0x3e


        ffff8385`8295f720 fffff806`46d162cc nt!AlpcpCompleteDeferSignalRequestAndWai\
t+0x3c
        ffff8385`8295f760 fffff806`46d15321 nt!AlpcpReceiveMessagePort+0x3ac
        ffff8385`8295f7f0 fffff806`46d14e05 nt!AlpcpReceiveMessage+0x361
        ffff8385`8295f8d0 fffff806`46885e95 nt!NtAlpcSendWaitReceivePort+0x105
        ffff8385`8295f990 00007ffa`0ebfd194 nt!KiSystemServiceCopyEnd+0x25 (TrapFram\
e @ ffff8385`8295fa00)
        000000e8`a8e3f798 00007ffa`0ba15778 ntdll!NtAlpcSendWaitReceivePort+0x14
        000000e8`a8e3f7a0 00007ffa`0ebcce7f CSRSRV!CsrApiRequestThread+0x108
        000000e8`a8e3fc30 00000000`00000000 ntdll!RtlUserThreadStart+0x2f

(truncated)
```

Информацию о потоке можно просмотреть отдельно с помощью команды! Thread и адреса потока.

Проверьте документацию отладчика, что-бы увидеть описания различных частей информации отображаемой этой командой.

Другие обычно полезные/интересные команды в режиме отладки ядра включают в себя:

•! Pcr - отображать область управления процессом (PCR) для процессора, указанного как дополнительный index(процессор 0 отображается по умолчанию, если индекс не указан).

•! Vm - отображать статистику памяти для системы и процессов.

•! Running - отображает информацию о потоках, запущенных на всех процессорах системы.

Мы рассмотрим более конкретные команды, полезные для отладки драйверов, в следующих главах.

5)Полная отладка ядра

Полная отладка ядра требует настройки на хосте и таргете. В этом разделе мы увидим, как настроить виртуальную машину как таргет для отладки ядра. Это рекомендуемое и наиболее удобная настройка для работы драйвера ядра (когда не разрабатываются драйверы устройств для оборудования).

Хорошо выполните шаги по настройке виртуальной машины Hyper-V (VM). Если вы используете другие технологии виртуализации (например, VMWare или VirtualBox), пожалуйста, обратитесь к документации этих программ.

Целевой и хост-компьютер должны обмениваться данными с использованием некоторого носителя подключения.

Есть несколько вариантов.

Лучший вариант - использовать сеть.

К сожалению, это требует хоста и таргета для запуска Windows 8 как минимум.

Поскольку Windows 7 по-прежнему является жизнеспособным таргетом, мы будем использовать другой вариант - COM (последовательный) порт.

Конечно, большинство машин больше не имеют последовательных портов, но в любом случае мы подключаемся к виртуальной машине, поэтому никаких реальных кабелей не требуется.

Все платформы виртуализации позволяют перенаправление виртуального последовательного порта на именованный канал на хосте; это конфигурация, которую мы будем использовать.

Конфигурирование таргета

Целевая виртуальная машина должна быть настроена для отладки ядра, аналогично локальной отладке ядра, но с добавленным носителем подключения установленой на виртуальный последовательный порт на этой машине.

Один из способов сделать это - использовать bcdedit в окне команд с повышенными правами:

bcdedit / debug on

bcdedit /dbgsettings serial debugport:1

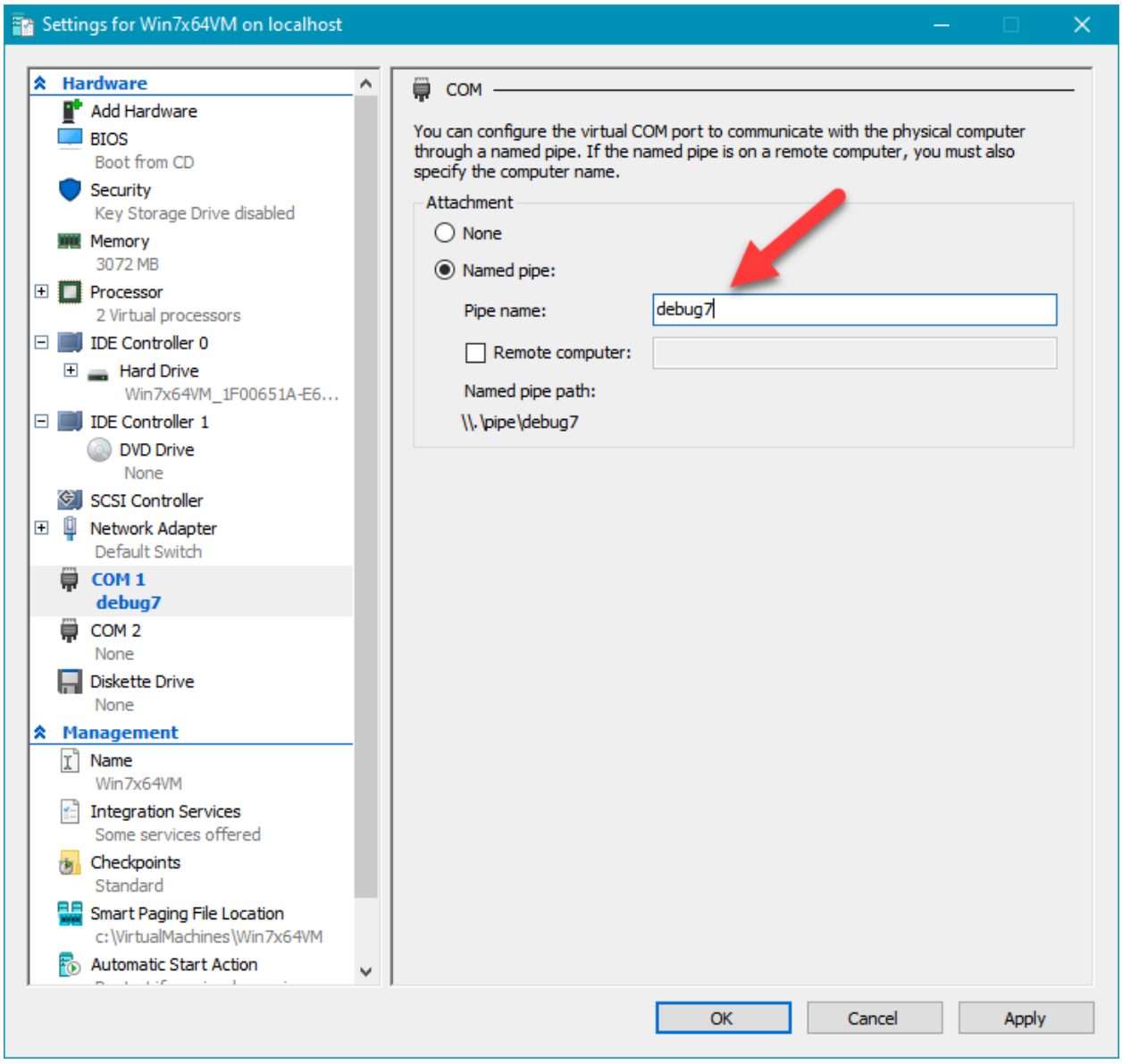
baudrate:115200

Измените номер порта отладки в соответствии с действительным виртуальным серийным номером (обычно 1).

Виртуальная машина должна быть перезапущена, чтобы эти конфигурации вступили в силу. Прежде чем сделать это, мы можем сопоставить последовательный порт для именованного канала. Вот процедура для виртуальных машин Hyper-V:

• Если виртуальная машина Hyper-V относится к поколению 1 (старше), в настройках виртуальной машины имеется простой пользовательский интерфейс для конфигурации. используйте опцию Add Hardware, чтобы добавить последовательный порт, если он не определен.

Затем настройте последовательный порт для сопоставления с именованным портом по вашему выбору. Рисунок 5-6 показывает это.



• Для виртуальных машин поколения 2 в настоящее время пользовательский интерфейс отсутствует. Чтобы настроить это, убедитесь, что виртуальная машина выключена (хотя не обязательно в самых последних версиях Windows 10) и откройте расширенный PowerShell

окно.

• Введите следующее, чтобы установить последовательный порт, сопоставленный с именованным каналом:

Set-VMComPort myvmname -Number 1 -Path \\.\pipe\debug

Измените имя виртуальной машины соответствующим образом и номер COM-порта, как было установлено внутри виртуальной машины ранее с помощью Bcdedit. Убедитесь, что путь *pipe* уникален.

Вы можете проверить, соответствуют ли настройки ожидаемым с помощью Get-VMComPort:

Get-VMComPort myvmname

VMName

---

myvmname

myvmname

Name Path

---- ----

COM 1 \\.\pipe\debug

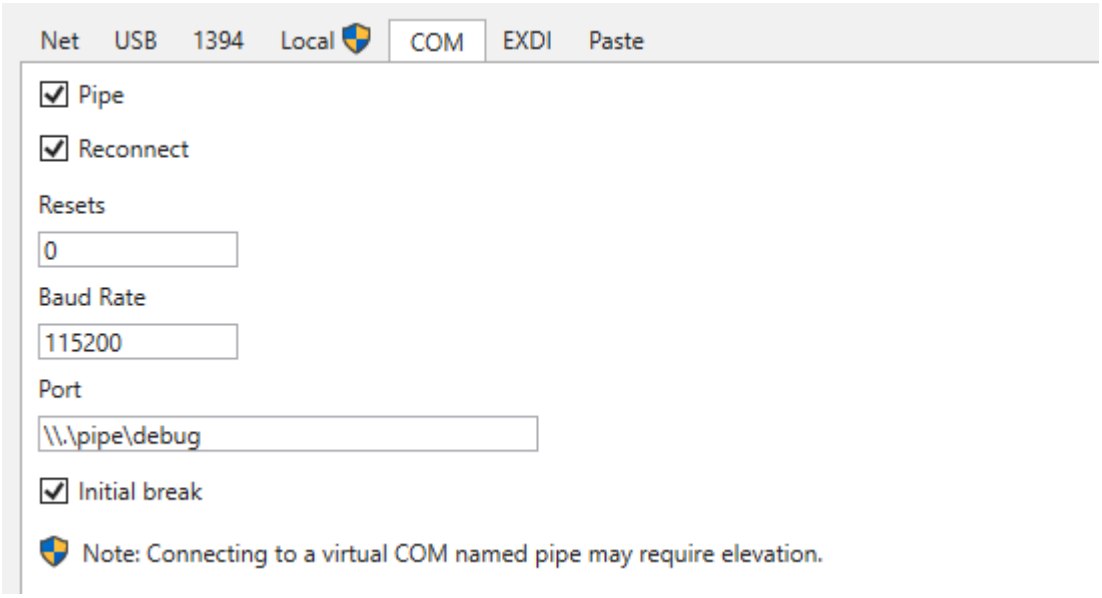
COM 2

Вы можете загрузить виртуальную машину - цель готова.

Конфигурирование хоста

Отладчик ядра должен быть настроен для подключения к виртуальной машине через тот же последовательный порт, сопоставленный с одноименным pipe выставленным на хосте.

• Запустите отладчик ядра и выберите File / Attach To Kernel. Перейдите на вкладку COM. Рисунок 5-7 показывает, как выглядят эти настройки.



• Нажмите на Отладчик должен прикрепиться к цели. Если это не так, нажмите кнопку панели инструментов Break.

Вот типичный вывод:

```
Microsoft (R) Windows Debugger Version 10.0.18317.1001 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Opened \\.\pipe\debug
Waiting to reconnect...
Connected to Windows 10 18362 x64 target at (Sun Apr 21 11:28:11.300 2019 (UTC + 3:0\
0)), ptr64 TRUE
Kernel Debugger connection established.  (Initial Breakpoint requested)

************ Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       SRV*c:\Symbols*http://msdl.microsoft.\
com/download/symbols
Symbol search path is: SRV*c:\Symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 10 Kernel Version 18362 MP (4 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 18362.1.amd64fre.19h1_release.190318-1202
Machine Name:
Kernel base = 0xfffff801`36a09000 PsLoadedModuleList = 0xfffff801`36e4c2d0
Debug session time: Sun Apr 21 11:28:09.669 2019 (UTC + 3:00)
System Uptime: 1 days 0:12:28.864
Break instruction exception - code 80000003 (first chance)
*******************************************************************************
*                                                                             *
*   You are seeing this message because you pressed either                    *
*       CTRL+C (if you run console kernel debugger) or,                       *
*       CTRL+BREAK (if you run GUI kernel debugger),                          *
*   on your debugger machine's keyboard.                                      *
*                                                                             *
*                   THIS IS NOT A BUG OR A SYSTEM CRASH                       *
*                                                                             *
* If you did not intend to break into the debugger, press the "g" key, then   *
* press the "Enter" key now.  This message might immediately reappear.  If it *
* does, press "g" and "Enter" again.                                          *
*                                                                             *
*******************************************************************************
nt!DbgBreakPointWithStatus:
fffff801`36bcd580 cc              int     3
```

Обратите внимание, что подсказка имеет индекс и слово kd. Индекс - текущий процессор, который вызвал изменения.

В этот момент целевая виртуальная машина полностью заморожена. Теперь вы можете отлаживать нормально.

6)Мануалл отладки драйверов

Как только хост и цель связаны, отладка может начаться. Мы будем использовать драйвер PriorityBooster, который мы разработанный в главе 4, чтобы продемонстрировать полную отладку ядра.

• Установите (но не загружайте) драйвер на цель, как это было сделано в главе 4. Убедитесь, что вы копируете файл PDB драйвера вместе с самим файлом SYS драйвера. Это упрощает получение правильных символов для драйвера.

• Давайте установим точку останова в DriverEntry. Мы не можем загрузить драйвер, потому что это может привести к DriverEntry и мы упустим шанс установить точку останова там.

Так как драйвер еще не загружен, мы можем использовать команду bu (неразрешенная точка останова), чтобы установить будущую точку останова.

Войдите цель, если она в данный момент запущена, и введите следующую команду:

0: kd> bu prioritybooster!driverentry

0: kd> bl

0 e Disable Clear u 0001 (0001) (prioritybooster!driverentry)

Точка останова еще не установлена, так как наш модуль еще не загружен.

• Выполните команду g, чтобы позволить цели продолжить, и загрузите драйвер с sc start booster

(при условии, что драйвер называется бустер). Если все идет хорошо, точка останова должна установится, и исходный файл должен загрузиться автоматически, показывая следующий вывод в командном окне:

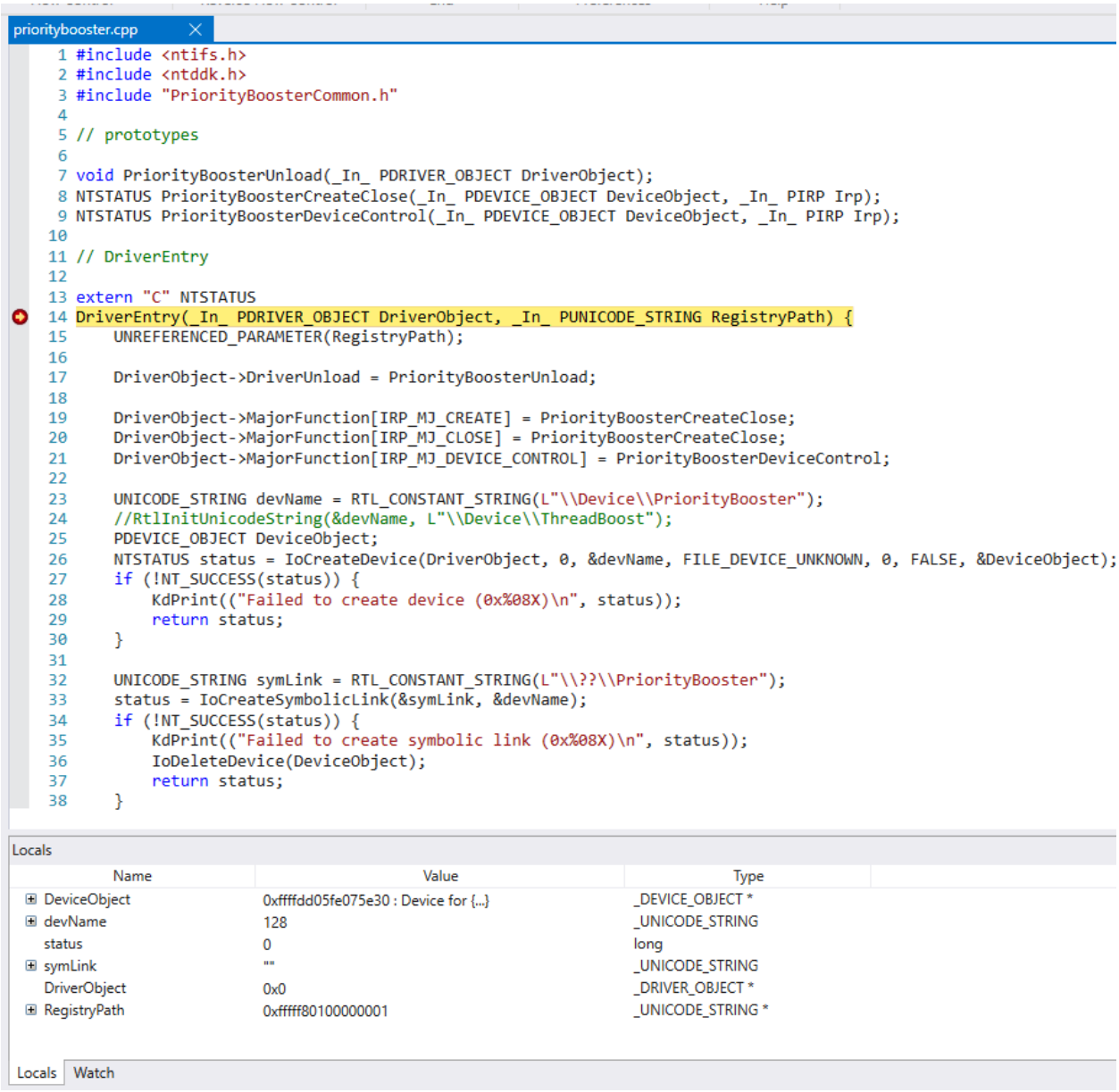
0: kd> g

Breakpoint 0 hit

PriorityBooster!DriverEntry:

fffff801`358211d0 4889542410 mov qword ptr [rsp+10h],rdx

Рисунок 5-8 показывает снимок экрана с исходным окном WinDbg Preview.



На этом этапе вы можете перешагнуть строки исходного текста, посмотреть переменные в окне Locals и даже добавить выражения в окне просмотра. Вы также можете изменить значения, используя окно Locals, как это делается с другими отладчиками.

Окно команд по-прежнему доступно, как всегда, но некоторые операции с пользовательским интерфейсом становятся проще.

Например, настройки точек останова можно выполнить с помощью обычной команды bp, но вы можете просто открыть исходный файл (если он еще не открыт), перейдите к строке, где вы хотите установить точку останова, и нажмите F9 или нажмите соответствующую кнопку на панели инструментов.

В любом случае команда bp будет выполнена в командном окне.

Окно Breakpoints может служить кратким обзором текущей точки останова.

Введите команду k, чтобы увидеть, как вызывается DriverEntry:

```
2: kd> k
 # Child-SP          RetAddr           Call Site
00 ffffad08`226df898 fffff801`35825020 PriorityBooster!DriverEntry [c:\dev\priorityb\
ooster\prioritybooster\prioritybooster.cpp @ 14]
01 ffffad08`226df8a0 fffff801`37111436 PriorityBooster!GsDriverEntry+0x20 [minkernel\
\tools\gs_support\kmodefastfail\gs_driverentry.c @ 47]
02 ffffad08`226df8d0 fffff801`37110e6e nt!IopLoadDriver+0x4c2
03 ffffad08`226dfab0 fffff801`36ab7835 nt!IopLoadUnloadDriver+0x4e
04 ffffad08`226dfaf0 fffff801`36b39925 nt!ExpWorkerThread+0x105
05 ffffad08`226dfb90 fffff801`36bccd5a nt!PspSystemThreadStartup+0x55
06 ffffad08`226dfbe0 00000000`00000000 nt!KiStartSystemThread+0x2a
```

Если точки останова, кажется, не в состоянии установить, это может быть проблема символов. Выполните команду .reload и посмотрите, решены ли проблемы.

Установка точек останова в пользовательском пространстве также возможна, но сначала выполните .reload / user.

Может случиться так, что точка останова должна срабатывать только тогда, когда конкретный процесс выполняет код.

Это можно сделать, добавив ключ / p к точке останова. В следующем примере точка останова устанавливается, только если это процесс explorer.exe:

```
2: kd> !process 0 0 explorer.exe
PROCESS ffffdd06042e4080
    SessionId: 2  Cid: 1df8    Peb: 00dee000  ParentCid: 1dd8
    DirBase: 1bf58a002  ObjectTable: ffff960a682133c0  HandleCount: 3504.
    Image: explorer.exe

2: kd> bp /p ffffdd06042e4080 prioritybooster!prioritybooosterdevicecontrol
2: kd> bl
     0 e Disable Clear  fffff801`358211d0  [c:\dev\prioritybooster\prioritybooster\p\
riorityb ooster.cpp @ 14]     0001 (0001) PriorityBooster!DriverEntry
     1 e Disable Clear  fffff801`35821040  [c:\dev\prioritybooster\prioritybooster\p\
riorityb ooster.cpp @ 63]     0001 (0001) PriorityBooster!PriorityBoosterDeviceContro\
l
     Match process data ffffdd06`042e4080
```

Давайте установим точку останова в месте, показанным на скриншете ниже:

```
62 _Use_decl_annotations_
63 NTSTATUS PriorityBoosterDeviceControl(PDEVICE_OBJECT, PIRP Irp) {
64     // get our IO_STACK_LOCATION
65     auto stack = IoGetCurrentIrpStackLocation(Irp);
66     auto status = STATUS_SUCCESS;
67
68     switch (stack->Parameters.DeviceIoControl.IoControlCode) {
69         case IOCTL_PRIORITY_BOOSTER_SET_PRIORITY:
70         {
71             // do the work
72             if (stack->Parameters.DeviceIoControl.InputBufferLength < sizeof(ThreadData))
73                 status = STATUS_BUFFER_TOO_SMALL;
74                 break;
75             }
```

• Запустите тестовое приложение с некоторым идентификатором потока и приоритетом:

Точка останова должна выполнится. Вы можете продолжить отладку в обычном режиме, используя комбинацию исходного кода.

Резюме

В этой главе мы рассмотрели основы отладки с помощью WinDbg. Это важный навык для разработки, так как программное обеспечение всех видов, включая драйверы ядра, могут содержать ошибки.

В следующей главе мы углубимся в некоторые механизмы ядра, с которыми нам необходимо ознакомиться, так как они часто возникают при разработке и отладке драйверов.