жизнь без BSOD или скрытые рычаги управления ядром Server 2003

крис касперски ака мыщъх, a.k.a. nezumi, a.k.a. souriz, a.k.a. elraton, no-email

голубой экран смерти — одна из самых больших неприятностей, которая только может случиться с сервером и хотелось бы, чтобы она происходила как можно реже, пускай даже ценой некоторого снижения производительности и увеличения потребления оперативной памяти. главным образом мы будем говорить о Server 2003 Standard Edition, хотя сказанное во многом справедливо и для остальных операционных систем линейки NT

введение

Голубой экран смерти вспыхивает всякий раз, когда ядро диагностирует критическую ошибку, которую не в состоянии корректно обработать (обращение по нулевому указателю, попытка освобождения уже освобожденной памяти и т. д.), передавая управление процедуре KeBugCheckEx, ответственную за "отрисовку" BSOD и сохранение дампа памяти (если администратор действительно хочет его сохранить). Некоторые отладчики уровня ядра (например, Soft-Ice) перехватывают вызов KeBugCheckEx, позволяя оператору "разрулить" ситуацию самостоятельно, вернув систему к жизни, однако, это требует достаточно высокой квалификации, так что на этом вопросе мы подробно останавливаться не будем.

При всем моем уважении к NT, следует сказать, что у нее очень тупое, кривое, и недописанное (!) ядро. Еще со времен NT 4.x (если не раньше) была предусмотрена возможность вызова call-back'ов (функций обратного вызова) из KeBugCheckEx, позволяющих, в частности, сбросить дисковые буфера, чтобы не угробить дисковый том, но… уже Server 2008 на дворе, а практическая реализация call-back'ов даже не обещается (хотя в NTFS драйвере все готовое для этого уже есть, странно не правда ли?!).

Другая нехорошая черта NT – нежелание разбираться с источниками критических ошибок. При возникновении исключения в загружаемом модуле ядра и Linux, и BSD просто выгружают данный модуль, продолжая нормальную работу системы, которая впадает в панику ("kernel panic" — аналог BSOD) только в действительно критических ситуациях. Казалось бы, у разработчиков NT имеются в наличии все необходимые ингредиенты: есть список модулей (т.е. драйверов), у драйверов есть процедура, ответственная за выгрузку драйвера (а даже если ее и нет, система имеет возможность выгружать драйвера в аварийном режиме), функция KeBugCheckEx в большинстве случаев определяет имя драйвера-виновника. Ну так что же мешает выгрузить его?! Ладно, не будем о грустном… лучше сразу перейдем к делу.

Основными источниками BSOD являются: дефекты железа, кривые драйвера и rootkit'ы. Некоторые API-функции прикладного уровня за счет ошибок проектирования так же могут приводить к голубым экранам (плюс ошибки в самом ядре системе), но их доля в общем зачете невелика. Итак, по большому счету у нас остается только железо и драйвера/rootkit'ы.

Что касается железо, то лучим средством борьбы будет приобретение качественных комплектующих, установка дополнительных радиаторов, прокладка аэродинамических кабелей, уменьшение тактовых частот/увеличение таймингов вкупе с увеличением питающего напряжения (большинство BIOS это позволяют). Программным путем аппаратные глюки никак не исправишь (исключение составляет блокировка использования битых ячеек памяти, но с учетом нынешних цен на память эти "извращения" уже неактуальны).

Короче, у нас остаются только драйвера/rootkit'ы. Ну rootkit'ы мы сразу удаляем (и журнал "хакер" неоднократно писал как), а вот с драйверами сложение. Хороших системных программистов очень мало, а фирм, разрабатывающих железо — много, вот и приходится нанимать "мальчиков по объявлению", клепающих драйвера в визуальных средах проектирования типа DriverStudio, а потом удивляющихся почему они падают. Исправление ошибок в драйверах — дело посильное (достаточно дружить с дизассемблером и отладчиком). Проще, конечно, скачать версию посвежее (в надежде, что там хотя бы часть ошибок исправлена), но лучше и надежнее переконфигурировать ядро операционной системы, сделав его менее чувствительным к ошибкам в драйверах, чем мы сейчас, собственно говоря, и займемся.

рычаги управления ядром

Ядро поддерживает множество рычагов управления. Ниже перечислены только основные из них:

boot.ini

Файл boot.ini, находящейся в корневом каталоге системного диска (которым обычно является диск "C:") принимает большое количество параметров, управляющих конфигурацией ядра и подобно описанных на MSDN: http://support.microsoft.com/kb/833721/ru (только документированные ключи) и в статье Марка Руссиновича "Boot.ini options reference": http://www.ingenieroguzman.com.ar/articulos/informatica/Boot-INI-Reference.htm;

глобальные флаги

Ядро NT поддерживает так называемые "глобальные флаги", управляющее его поведением и задаваемые с помощью утилиты gflags.exe, входящий в комплект поставки "Support Tools" и в NTDDK, который можно скачать бесплатно http://www.microsoft.com/whdc/devtools/ddk/default.mspx; (версия для Server 2003), для Server 2008 рекомендуется использовать более свежую версию http://www.microsoft.com/whdc/DevTools/default.mspx (бесплатна, но требует предварительной регистрации);

реестр

Ядро Server 2003 поддерживает свыше 128 ключей реестра, большинство из которых недокументированно и добыто путем дизассемблирования (см. таблицу 1). Мы не в состоянии рассказать обо всех ключах, однако, их описание (вместе с полным путем к реестру) легко найти в Google по имени ключа. Практически все ключи так или иначе описаны в различных источниках (форумах, блогах, исходных текстах ReactOS и т.д.)

ключи реестра, ответственные за конфигурирование ядра		
AdditionalCriticalWorkerThreads	AdditionalDelayedWorkerThreads	AdjustDpcThreshold
AllocationPreference	ClearPageFileAtShutdown	CountOperations
CriticalSectionTimeout	DeadlockRecursionDepthLimit	DeadlockSearchNodesLimit
DisablePagingExecutive	DpcQueueDepth	DPCTimeout
DynamicMemory	EnableTimerWatchdog	EnforceWriteProtection
FastSystemCallDisable	HeapDeCommitFreeBlockThreshold	HeapDeCommitTotalFreeThreshold
HeapSegmentCommit	HeapSegmentReserve	HighMemoryThreshold
IdealDpcRate	Idle0TimeCheck	IdleDefaultDemotePercent
IdleDefaultDemoteTime	IdleDefaultMinThrottle	IdleDefaultPromotePercent
IdleDefaultPromoteTime	IdleFrom0Delay	IdleFrom0IdlePercent
IdleThrottleCheckRate	IdleThrottleCheckTimeout	IdleTimeCheck
IdleTo0Percent	IoFailZeroAccessCreate	IoVerifierLevel
LargeIrpStackLocations	LargePageDrivers	LargePageMinimum
LowMemoryThreshold	MakeLowMemory	MapAllocationFragment
MaxTimeSeparationBeforeCorrect	MinimumDpcRate	MinimumStackCommitInBytes
ModifiedPageLife	NonPagedPoolMaximumPercent	NonPagedPoolQuota
NonPagedPoolSize	ObTraceNoDeregister	ObTracePoolTags
ObUnsecureGlobalNames	PagedPoolQuota	PagedPoolSize
PagingFileQuota	PerfCriticalFrequencyDelta	PerfCriticalTimeDelta
PerfDecreaseAbsoluteModifier	PerfDecreaseMinimumTime	PerfDecreasePercentModifier
PerfDecreaseTimeValue	PerfDegradeThrottleMinCapacity	PerfDegradeThrottleMinFrequency
PerfIncreaseAbsoluteModifier	PerfIncreaseMinimumTime	PerfIncreasePercentModifier
PerfIncreaseTimeValue	PerfMaxC3Frequency	PerfTimeDelta
PoCleanShutdownFlags	PoolTag	PoolTagBigTableSize
PoolTagOverruns	PoolTagSmallTableSize	PoolUsageMaximum
PowerPolicySimulate	PriorityControl	PriorityQuantumMatrix
ProductOptions	ProtectNonPagedPool	RegistryLazyFlushHiveCount
RegistryLazyFlushInterval	RegistryLogSizeLimit	RegistrySizeLimit
ResourceCheckFlags	ResourceTimeoutCount	SAppCompat
SecondLevelDataCache	SessionPoolSize	SessionViewSize
SnapUnloads	SpinlockTimeout	SystemPages

ThreadDpcEnable	TimeLimitDpcMicroseconds	TimeLimitIsrMicroseconds
TrackLockedPages	TrackPtes	TSEnabled
VerifyDriverLevel	VerifyDrivers	Win32PrioritySeparation
ShutdownTime	SystemViewSize	XMMIZeroingEnable

Таблица 1 ключи реестра, поддерживающие ядром Server 2003 SP0 Enterprise Edition

наиболее популярные BSOD'ы и способы их устранения

Появление многоядерных процессоров поставило драйвера "раком", к чему большинство из них оказались совершенно не готово и голубые экраны начали вспыхивать с некоторой вероятностью, зависящей от частоты поступления прерываний от аппаратных устройств. Рассмотрим вполне типичную ситуацию: поток A выполняется на ядре I с IRQL=PASSIVE_LEVEL, в то время как поток B выполняется на ядре II с тем же IRQL. Устройство Device 1 посылал ядру I сигнал прерывания, которое ловит ядро оси, повышает IRQL процессорного ядра I до DIRQL и передает управление обработку прерывания устройства Device 1. Обработчик выполняет первичную обработку ситуации, и ставит отложенную процедуру DpcForIsr() в очередь для дальнейшей обработки, при этом функция добавляется в очередь того процессорного ядра, на котором запущен обработчик (в данном случае это ядро I).

Устройство Device 1 вновь генерирует прерывание, которое на этот раз посылается ядру II, поскольку ядру I еще не успело выйти из обработчика прерывания и понизить IRQL. Ось повышает IRQL ядра II до DIRQL и передает управление обработчику прерываний устройства Device 1, который ставит еще одну отложенную процедуру DpcForIsr() в очередь на выполнение на ядре II.

Наконец, обработчики прерываний на обоих ядрах завершаются, ось понижает IRQL и начинается выполнение отложенных процедур, стоящих в очередях ядра I и ядра II, в результате чего _одна_ и _та_ _же_ процедура DpcForIsr() на обоих ядрах исполняться _одновременно_, обрабатывания сразу два различных прерывания! Маленькая небрежность кодирования приводит к ошибкам синхронизации, приводящих к каскаду вторичных ошибок и способных генерировать добрую половину всех существующих экранов голубой смерти.

Выход? Использовать только одно ядро, игнорируя все остальные, для чего достаточно указать ключ /ONECPU или /NUMPROC=1 в файле boot.ini. Конечно, это снизит производительность, но… если кривой драйвер при интенсивном поступлении прерываний не справляется с синхронизацией и обрушивает систему в BSOD, то лучше поступиться производительностью, чем надежностью.

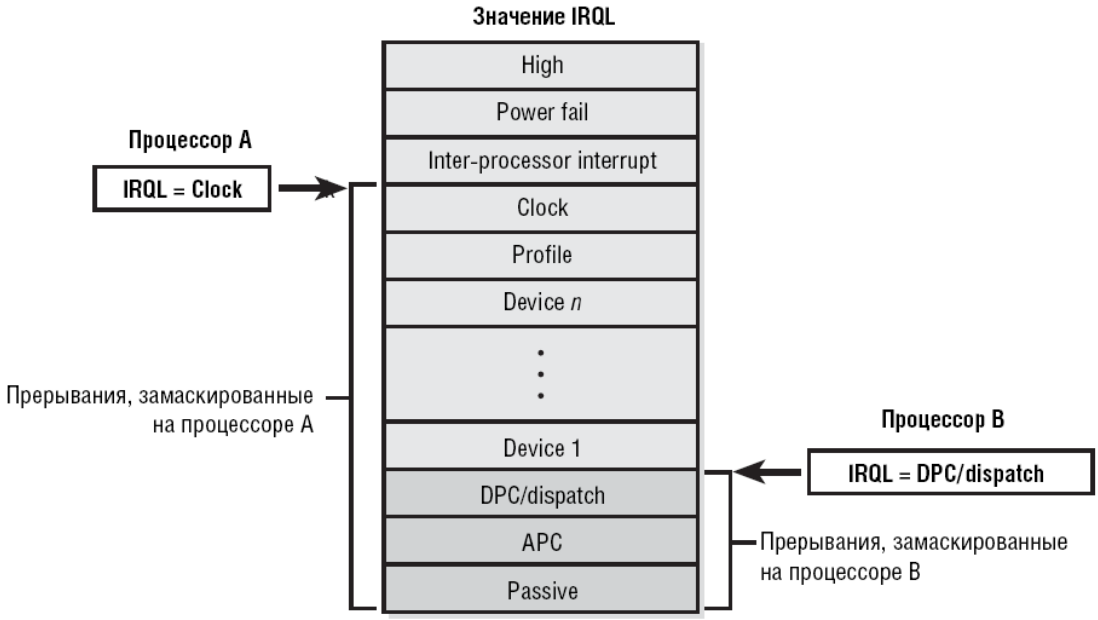

Рисунок 1 уровни привилегий, поддерживаемые ядром NT (хорошо заметно, что уровни прерывания от устройств, находятся выше уровня диспетчеризации, на котором работает подкачка страниц с диска)

А кому не знаком BSOD с противным названием IRQL_LESS_OR_EQUAL, выпрыгивающий в самый неподходящий момент? Что это такое и почему оно возникает? Операционные системы семейства NT используют особую систему приоритетов прерываний Interrupt Request Levels (или, сокращенно IRQ), оперирующую целыми числами от 0 до 31. Уровень 0 имеет минимальный приоритет, 31 — максимальный. Нормальное выполнение потока происходит на пассивным уровне (PASSIVE LEVEL == 0) и его может прерывать любое асинхронное событие, возникающее в системе. При этом ось повышает текущий IRQL до уровня возникшего прерывания и передает управление его ISR (Interrupt Service Routine — процедура обработки прерывания), причем, подкачка страниц с диска работает только на уровне 2, а прерывания, генерируемые устройствами, начинаются с уровня 3 и потому первичные обработчики прерываний не могут обращаться к памяти ядра, вытесняемой на диск, но… увы! Как показывает практика они к ней все-таки обращаются. Если запрещенная страница находится в памяти, то все ОК, но вот если она вытеснена на диск, возникает указанный BSOD.

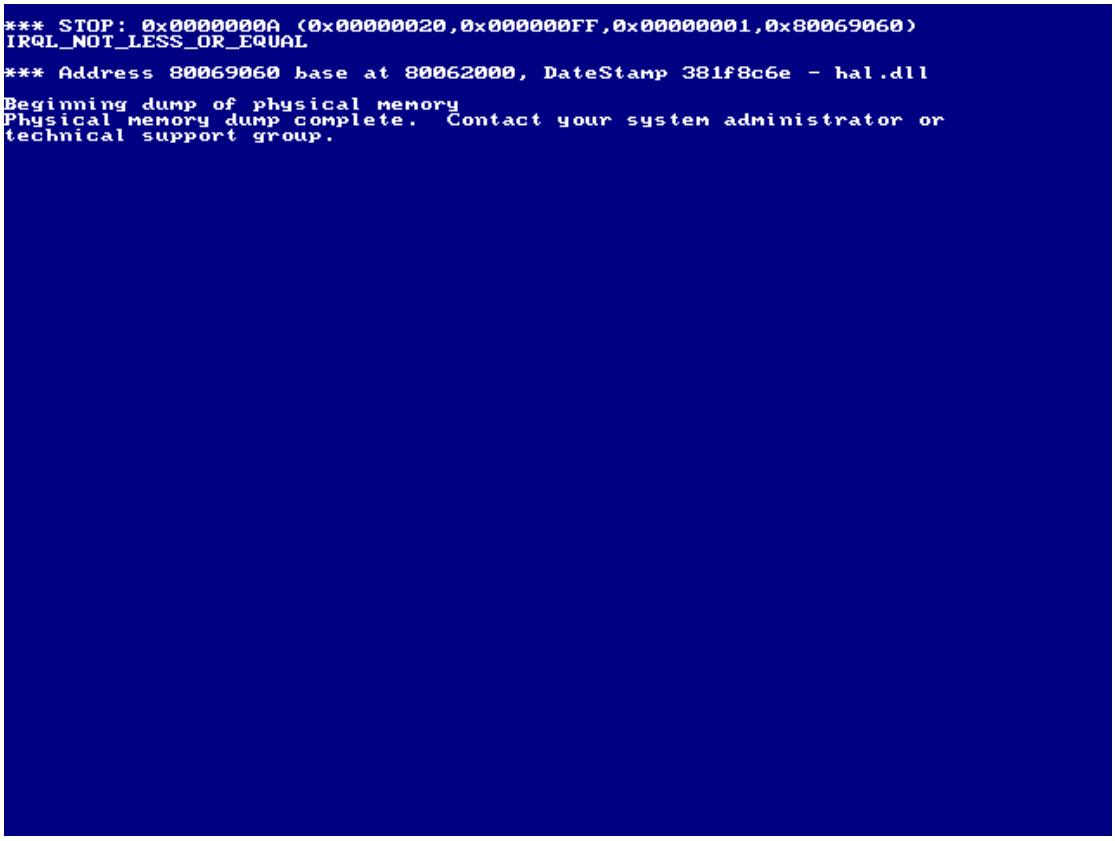

Рисунок 2 IRQL_LESS_OR_EQUAL — один из самых часто встречающихся экранов голубой смерти, возникающий при попытке обращения к странице, вытесненной на диск, на том уровне привилегий, на котором подкачка не функциональна

Как его предотвратить? Идея первая — увеличить количество оперативной памяти, чтобы шансы на вытеснение запрашиваемых страниц были минимальны. Решение второе — запретить своппинг ядра на диск, для чего необходимо установить параметр "DisablePagingExecutive" (типа DWORD) в значение 1 (он находится в следующей ветке реестра: HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\MemoryManagement), так же следует запустить gflag.exe с ключом dps, запрещающим вытеснение стека ядра на диск. Изменения вступают в силу после перезагрузки. И хотя потребности в памяти слегка увеличиваются (поскольку, ядро уже не может вытеснить бездействующие драйвера), общая надежность системы _значительно_ повышается, кроме того, система становится нечувствительной к определенным типам DoS атак, "съедающих" всю доступную память и вынуждающую ядро активно свопиться на диск, в надежде, что в системе обнаружиться хоть один кривой драйвер, вызывающий BSOD с пресловутым IRQL_LESS_OR_EQUAL.

Кстати говоря, в Server 2003 SP2 допущена досадная ошибка, связанная с некорректной реализацией SafeSEH и обрушивающая систему в BSOD при вызове такой безобидной функции как DbgPrint, использующейся в драйверах для отладочной печати. Подробнее об этом можно

прочитать в 16h выпуске "Exploits Review", опубликованном в журнале "Хакер", ну а для преодоления BSOD достаточно вызывать gflags.exe с ключом ddp.

Хочется отметить, что основными поставщиками BSOD являются драйвера видео- и звуковых карт, поэтому, удаляем драйвер звуковой карты (а зачем серверу звук?) а для форсирования VGA режима добавляем в boot.ini ключик /BASEVIDEO.

Еще одну проблему представляет собой поддержка расширений физических адресов (Physical Address Extensions или, сокращенно, PAE), теоретически позволяющей операционной системе использовать свыше 4х Гигабайт физической памяти, практически же Server 2003 Standard Edition этой возможности не поддерживает, вынуждая нас покупать Enterprise Edition, однако, при активном механизме DEP (Data Execution Prevention), включенном по умолчанию в Server 2003 SP1, система всегда стартует с поддержкой PAE, независимо от количества физической памяти, имеющейся на борту, что создает проблемы с некоторыми драйверами, поскольку, в режиме PAE на уровне ядра имеются некоторые тонкости работы с памятью, учитываемые далеко не всеми разработчиками. Очевидное решение — добавить ключ /NOPAE в boot.ini и забыть об этой проблеме раз и навсегда.

Остальные проблемы в BSOD'ами решаются экспериментальным путем посредством манипуляций с ключами реестра, перечисленными в таблице 1.

>>> врезка описание BSOD'ов

Подробное описание всех BSOD'ов вместе с причинами их возникновения и рекомендациями по их предотвращению можно найти как в MSDN, так и в DDK (однако, и то, и другое в первую очередь ориентированно на программистов, знающих ассемблер, умеющих работать с отладчиком и готовых часами разбирать дампы памяти для поиска ошибки в подопытном драйвере).

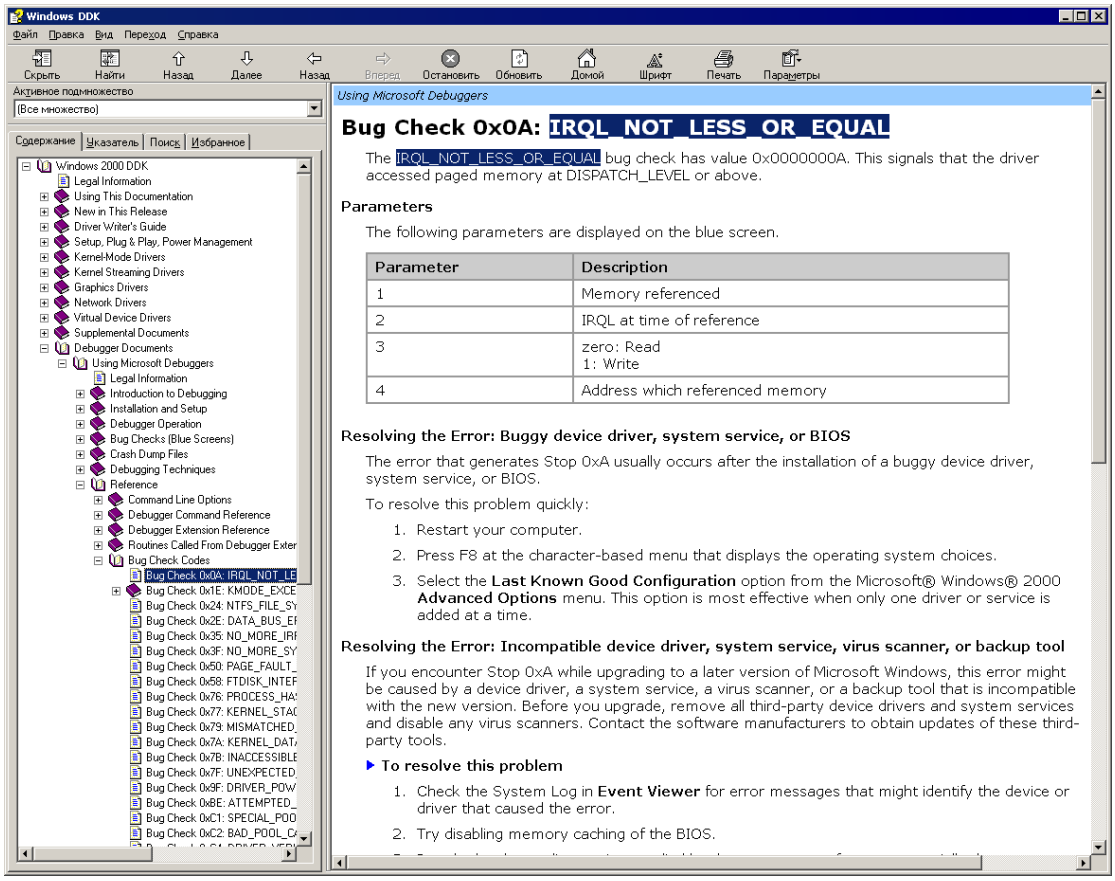

Рисунок 3 описание голубых экранов смерти в DDK

заключение

Настройку сервера удобнее осуществлять под виртуальной машиной типа VM Ware, выделяя гостевой операционной системе минимум памяти и направляя на нее шторм сетевых пакетов. При наличии "кривых" драйверов BSOD не заставит себя ждать и тут же появится на экране. К сожалению, VM Ware не дает прямого доступа к большинству компонентов материнской платы, а потому тестируются совсем не те драйвера, которые работают в реальных условиях, но даже такой примитивный тестовый стенд выявляет огромное количество ошибок, позволяя подобрать оптимальные ключи реестра и файла boot.ini, а окончательную настройку ядра можно выполнить уже на "живой" машине, естественно, создав резервную копию boot.ini и реестра с помощью любой утилиты резервирования, работающей _до_ загрузки операционной системы, поскольку, есть шанс, что после наших экспериментов, Server 2003 вообще не сможет загрузиться.

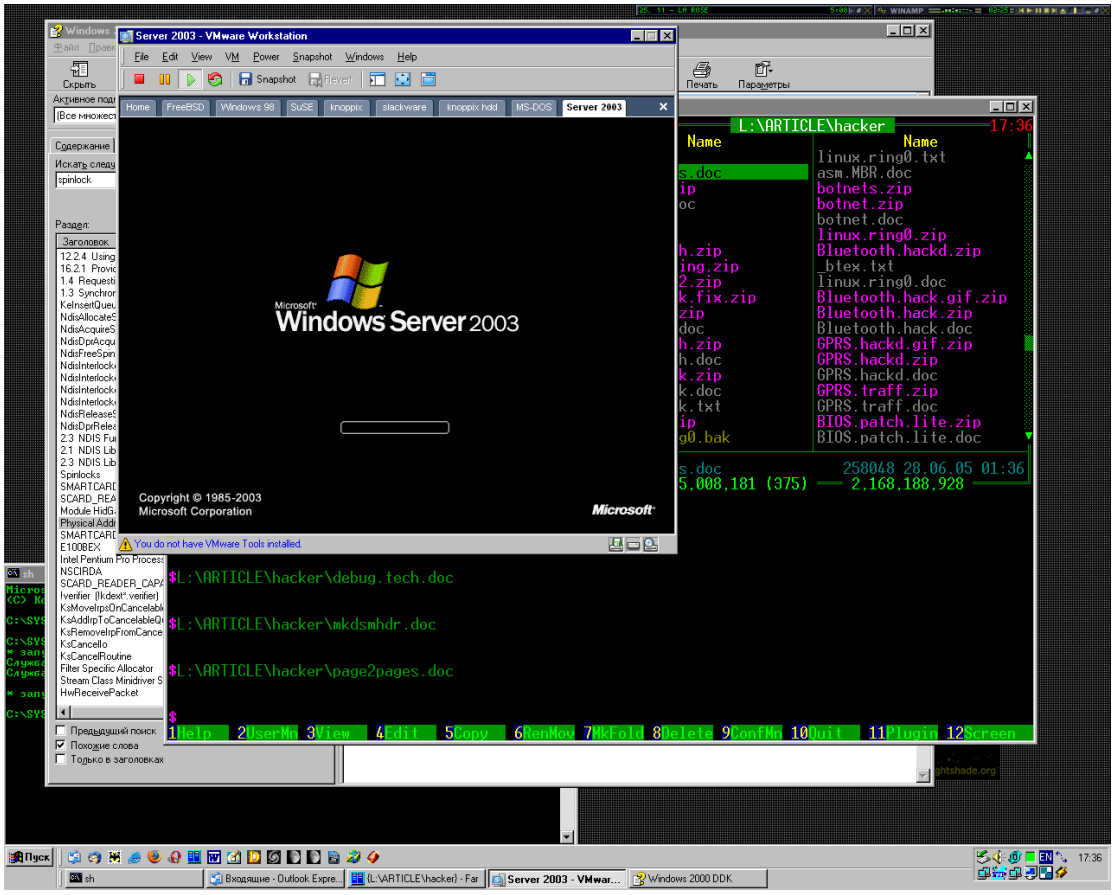

Рисунок 4 настройка Server 2003 под виртуальной машиной VM Ware