турбо-перенос драйверов из Windows в LINUX/BSD

крис касперски ака мыщъх

главный недостаток UNIX-подобных систем — отсутствие нормальных драйверов под кучу "вкусного" оборудование, с которым Windows справляется без проблем. на самом деле, проблему можно решить и эта статья рассказывает как.

введение

Несколько лет назад ситуация с драйверами под LINUX и BSD была просто катастрофической. Поддерживалось лишь некоторое оборудование и железо для UNIX-машин приходилось закупать отдельно. Тогда LINUX еще не вышла из стадии "конструктора" для хакеров, а BSD в основном использовалась на серверах, все оборудование которых сводилось к сетевой карте и SCSI-контроллеру. Так что основой массе пользователей жаловаться не приходилось.

На домашних и "десктопных" компьютерах ситуация совсем иная. Тут и сканеры, и мультимедиа, и конечно же видео-карты, издавна славящееся отсутствием общих стандартов и пестротой know-how. Производители железа в основном ориентируются на Windows и крайне неохотно вкладывают деньги в другие системы, поскольку они приносят одни убытки. Разработка и тестирование драйверов — удовольствие не из дешевых, а парк UNIX-машин очень маленький (несколько процентов от общего числа, если не меньше) и к тому же очень разобщенный, следовательно рыночная доля каждой из осей становится еще меньше и "подобрать" ее к рукам могут только очень крупные производители такие, например, как nVIDIA, продающие миллионы видео-карт, среди которых и пара процентов становится вполне ощутимой величиной.

Часто приходится слышать о большой армии энтузиастов, дизассемблирующих Windows-драйвера (а откуда еще взять информацию? техническая документация ведь не всегда доступна) и переписывающих их под LINUX или BSD. Как говорится, "есть хакеры в русский селениях, в горящий компьютер войдут, драйвер напишут и Гейтсу хвост оторвут", но... этих энтузиастов не так уж и много. Разнотипного оборудования гораздо больше, тем более что сплошь и рядом приходится сталкиваться с ситуацией, что на компьютере энтузиаста драйвер работает, а на моем нет, хотя оборудования у нас вроде бы одинаковое. Драйвер мало написать, его еще нужно отладить, протестировать на большом количестве конфигурацией и сопровождать, иначе это будет не драйвер, а просто игрушка. Все это требует затрат времени и если драйвер создается не для коммерческой выгоды, а "под себя", то его надежность и совместимость оставляют желать лучшего.

Составители дистрибутивов проделывают огромную работу, собирая различные драйвера и закатывая их на диск, примерно так же как закатывают хамсу или кильку. Закинули невод, вытащили рыбку и кинули в банку. Пускай едят! Качество тестирования таких драйверов очень невелико и даже если в списке поддерживаемого оборудования значится такое-то железо, никаких гарантий, что оно заработает у нас нет. Может быть, подойдет драйвер от другой модели, а может быть ничего не подойдет вообще! А ведь без драйверов сидеть хреново. Ни каменный топор обтесать, ни огонь развести. Компьютер слегка греет, если развернуть его вентилятором вперед, но это как-то не то...

виртуальные машины

Прежде чем ставить LINUX/BSD задумаетесь — а зачем вам, собственно, все это нужно? Если просто хотите пощупать альтернативную систему, освоить средства разработки или компилировать исходные тексты, но наилучшим выбором будет виртуальная машина такая как, например, VM Ware. Fedora Core на ней, конечно жутко тормозит (на P-III 733 работать вообще невозможно), но Debian с KDE идет вполне нормально. Хочешь — разрабатывай программы, хочешь — читай man'ы. Еще и в игры типа Star Wars можно поиграть. Никаких драйверов в этом случае не потребуется, в смысле "никаких драйверов сверх того, что есть в любом нормальном дистрибутиве". Большинство разработчиков именно так и поступают. Как ни крути, а любой уважающий себя UNIX-программист вынужден держать на компьютере десяток осей различных пород, чтобы тестировать свои программы на совместимость. На "живом" компьютере переключения между ними происходят только через перезагрузку, что не

есть хорошо, а виртуальные машины переключаются как карусель (главное памяти побольше иметь!).

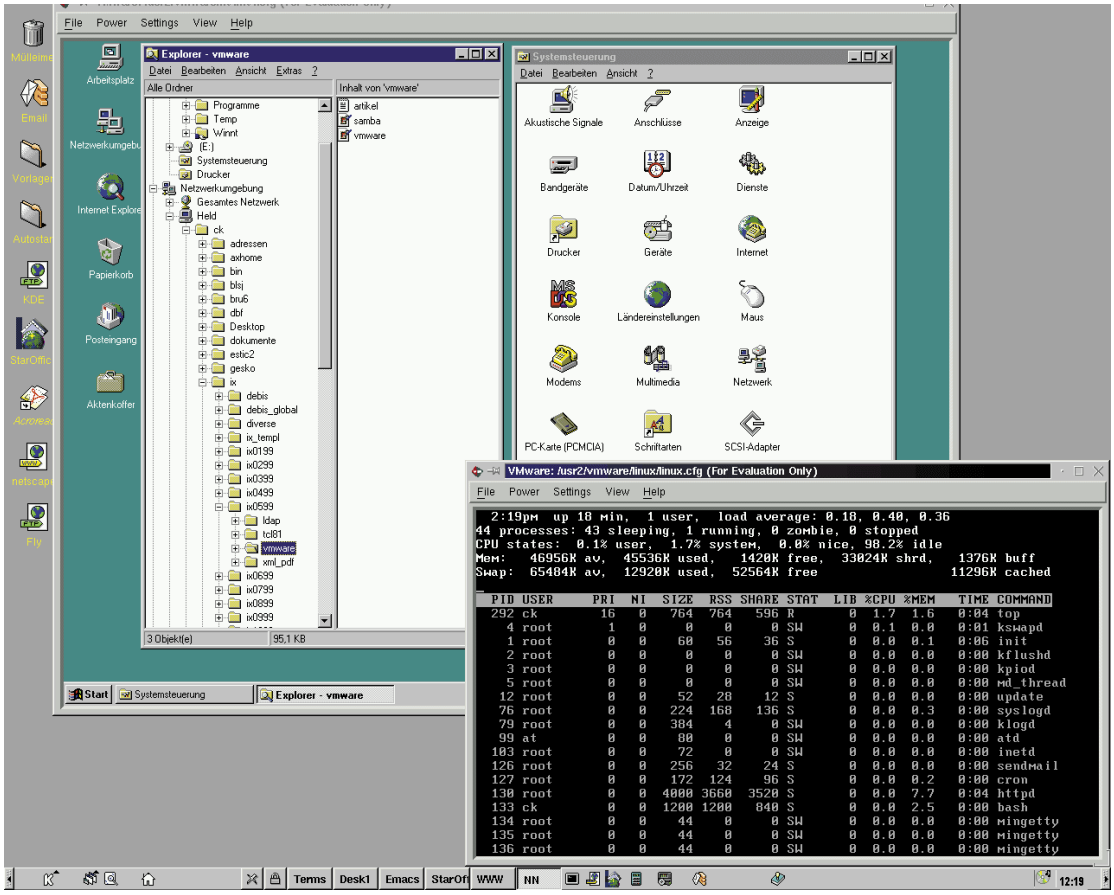

Рисунок 1 виртуальная Windows-машина, работающая под управлением LINUX

Можно поступить и наоборот. Установить LINUX/BSD как базовую систему, а Windows водрузить на виртуальную машину. Поскольку VMWare дает прямой доступ к COM/LPT/USB портам, то подключение сканера/принтера/цифровой камеры к вашей машине уже не станет проблемой. С ней будет работать Windows! Базовая UNIX-машина в этом случае получает в свое распоряжение все системные ресурсы и падения производительности уже не происходит, но появляются другие проблемы. Windows-приложения (например, игрушки) будут либо сильно тормозить, либо откажутся запускаться совсем, к тому же со всеми остальными типами устройств, например, интегрированной WLAN платой или видео-картой, Windows работать не сможет. А все потому, что VMWare представляет собой закрытый ящик, отгороженный от базовой операционной системы толстой стеной эмулятора. Вот если бы существовала возможность предоставить виртуальной машине полный доступ ко всему физическому оборудованию, вот тогда бы... Готовьтесь! Именно такой способ мы и собираемся описать!

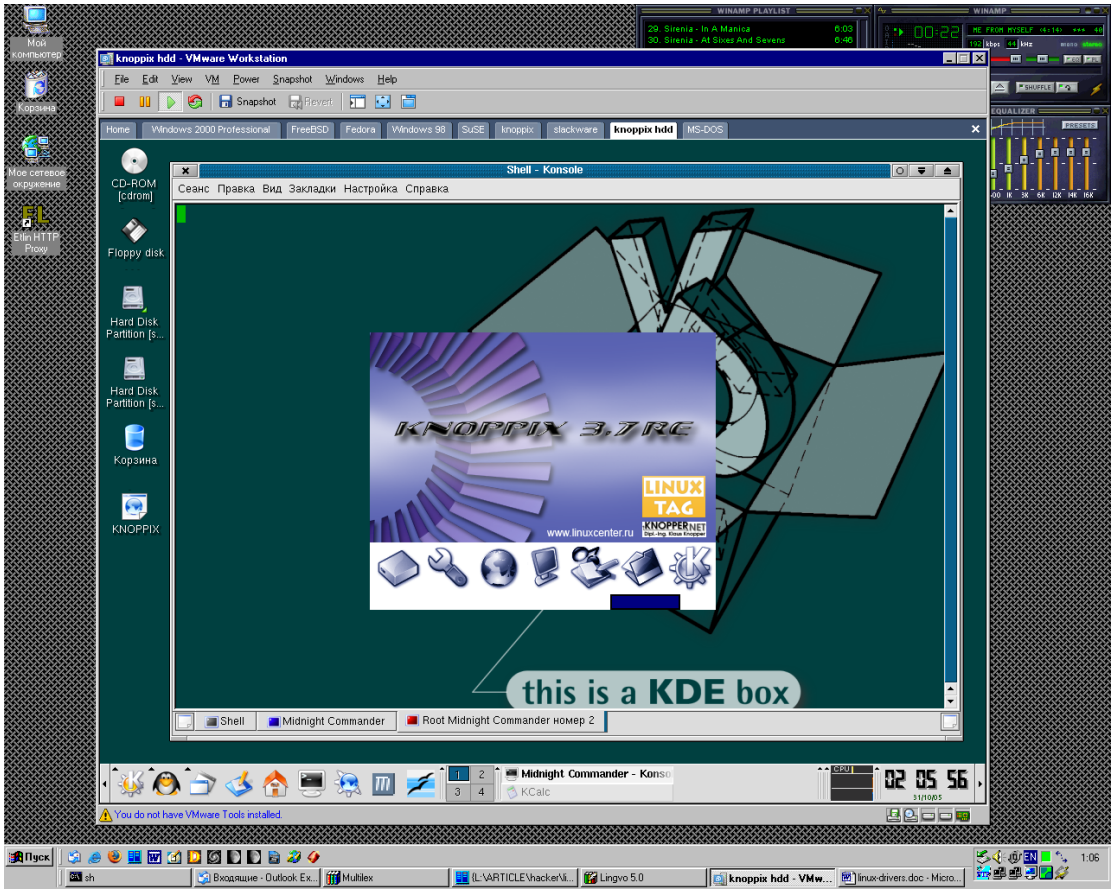

Рисунок 2 виртуальная LINUX-машина, работающая под управлением Windows

два в одном

Начнем с простого, но до сих пор никем не решенного вопроса, то есть уже давно решенного, конечно, но совсем не так. Известно, что поддержка NTFS-разделов представляет собой сплошную проблему. Драйвера, научившиеся писать на NTFS-раздел, появились совсем недавно, да и то лишь затем, чтобы покрасоваться на выставках. Для реальной работы они не пригодны, потому что не очень стабильно работают и несут на своем горбу кучу ограничений. Сжатые файлы, транзакции и куча других вещей все еще не поддерживается. К тому же, NTFS не стоит на месте и хоть и медленно, но совершенствуется, в смысле видоизменяется. Можно ли, хотя бы теоретически, написать 100% совместимый драйвер, "переваривающий" новые версии NTFS без участия программиста? Вопрос совсем не так глуп, каким кажется. Для чего нам горбатиться над своим собственным драйвером, когда под рукой есть уже готовый — NTFS.SYS. Если мы сумеем заставить его заработать под LINUX'ом, все проблемы решатся сами собой.

Вот только не надо кричать, что я ни хрена не разбираюсь в программировании и что на уровне ядра LINUX/BSD отличается от Windows так же, как слонопотам от крокодила. Да, различий очень много, но что-то общее между ними все-таки есть. И Windows, и LINUX, и BSD работают на x86-процессорах в защищенном режиме, используют страничную организацию виртуальной памяти, и взаимодействуют с оборудованием в строго установленном порядке (через иерархию физических и виртуальных шин). Высокоуровневые драйвера такие, например, как NTFS.SYS вообще не касаются оборудования и содержат минимум системно-зависимого кода. Почему же тогда драйвер от одной системы не работает в другой? Главным образом, потому что интерфейс между осью и драйвером в каждом случае различен и еще потому, что драйвер использует библиотеку функций, экспортируемых системой и эти функции у каждой системы свои.

Перенести Windows-драйвер в LINUX/BSD вполне реально! Для этого даже не потребуется его исходный код. Достаточно лишь написать тонкий и несложный "переходник" между драйвером и операционной системой, принимающий запросы и транслирующий их по

всем правилами "этикета", а так же перетащить библиотеку функций, необходимых драйверу для работы. О, да! — для этого необходимо уметь программировать! Для простых смертных пользователей такой рецепт совершенно не годится, но тут уже ничего не попишешь. Тем не менее, перенести готовый драйвер намного проще, чем переписать его с нуля. Как минимум не потребуется проводить кропотливую работу по дизассемблированию оригинального кода, заменяющую собой поиск технической документации (которая либо совсем отсутствует, либо отдается только под подписку о неразглашении, зачастую запрещающее открытое распространение исходных текстов). К тому же, при выходе новых версий Windows-драйвера, процедура обновления LINUX/BSD порта существенно обновляется, просто скопировал поверх старого файла и все. Однако, все это теория. Перейдем к деталям.

Ядерная модель Windows NT и всех производных от нее операционных систем (как-то Windows 2000, XP, 2003, Longhorn) достаточно проста. С "внешним" миром ядро связывает Диспетчер Системных Сервисов, "подключенный" к NTDLL.DLL, которая находится уже за "скорлупой" ядра и исполняется в пользовательском режиме. Диспетчер системных сервисов, реализованный в NTOSKRNL.EXE, опирается на Вызываемые Интерфейсы Ядра, часть которых реализована внутри самого NTOSKRNL.EXE, а часть — во внешних драйверах, к числу которых в частности принадлежит диспетчер электропитания. Определенный класс драйверов, называемый Драйверами Устройств и файловой системы, находится в своеобразной "скорлупе" и взаимодействует с диспетчером системных вызовов через диспетчер ввода-вывода реализованный опять-таки в NTOSKRNL.EXE!

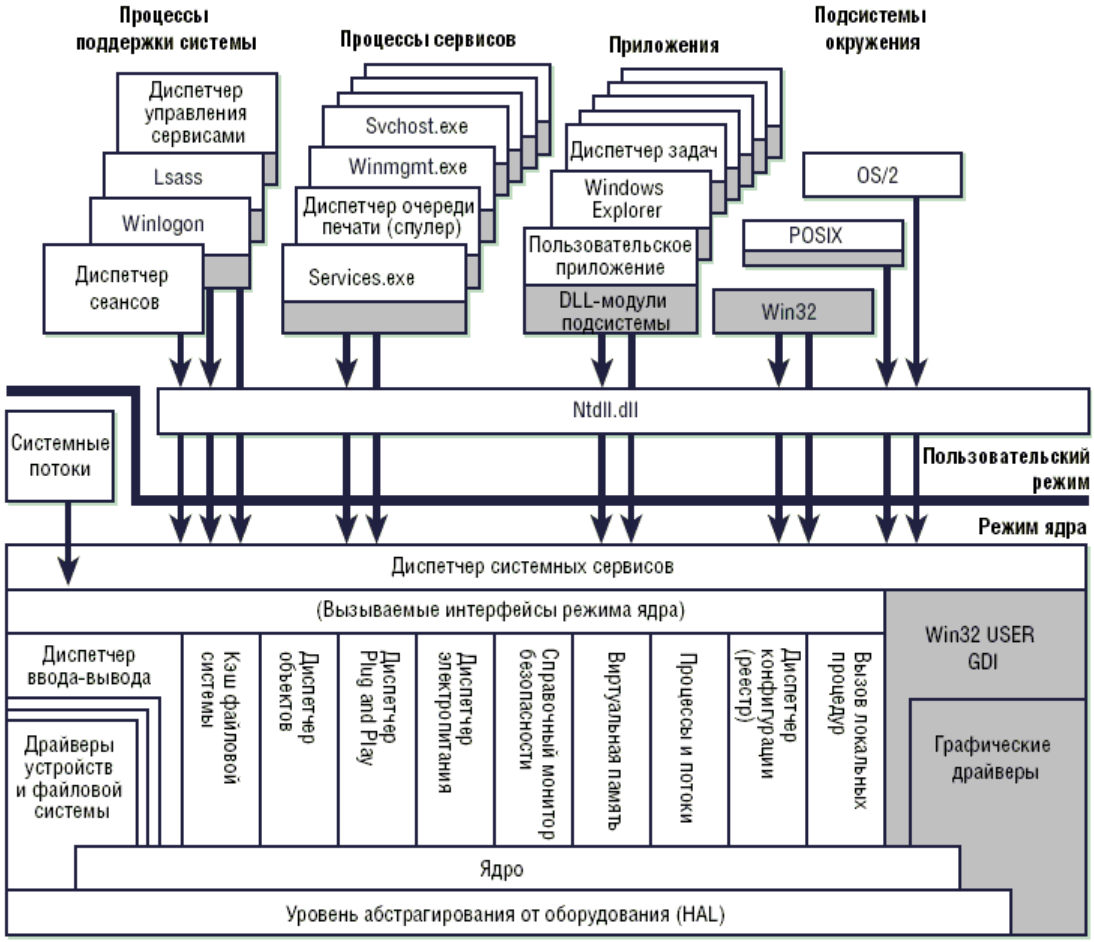

Рисунок 3 Windows NT изнутри

Ядро на котором как на фундаменте держаться все вышеупомянутые компоненты представляет собой просто совокупность низкоуровневых функций, сосредоточенных... правильно! В NTOSKRNL.EXE! Ниже находится только слой абстрагирования от оборудования или сокращенно HAL (Hardware Abstraction Level). Когда-то у Microsoft была идея разделить ядро на системно-зависимую и системно-независимую часть, чтобы упростить перенос Windows на другие платформы, но уже во времена NT 4.x все перемешалось и большая часть системно-

зависимых функций попала в NTOSKRNL.EXE, а ныне HAL медленно, но неотвратимо умирает. В нем осталось небольшое количество действительно низкоуровневых функций, непосредственно взаимодействующих с оборудованием. В частности, с портами и с DMA. Но в ядре LINUX/BSD есть свои функции для работы с DMA, так что тащить за собой HAL нам совершенно необязательно, тем более что драйвера взаимодействуют с DMA не напрямую, а через Plug-a-Pray менеджер, который находится в NTOSKRNL.EXE.

Что же касается портов ввода-вывода, то это вообще смех. Вот так, например, выглядит дизассемблерный текст функции READ_PORT_UCHAR, читающий из данного порта беззнаковый байт.

```
.text:80015A2C
.text:80015A2C                 public READ_PORT_UCHAR
.text:80015A2C READ_PORT_UCHAR proc near   ; CODE XREF: HalGetEnvironmentVariable+2C↑p
.text:80015A2C                                         ; HalSetEnvironmentVariable+3D↑p
.text:80015A2C
.text:80015A2C arg_0           = dword ptr  4
.text:80015A2C
.text:80015A2C                 xor     eax, eax
.text:80015A2E                 mov     edx, [esp+arg_0]
.text:80015A32                 in      al, dx
.text:80015A33                 retn    4
.text:80015A33 READ_PORT_UCHAR endp
```

Листинг 1 дизассемблерный листинг функции READ_PORT_UCHAR, выдернутой из HAL'a

Таким образом, если заставить NTOSKRNL.EXE работать в среде чужеродного ему LINUX'a (или BSD), мы получим возможность запускать любые NT-драйвера без какой-либо доработки их двоичного кода. Это не только упрощает задачу переноса, но и снимает проблему так называемых "авторских прав", от которых меня уже блевать тянет. У нас на них можно в принципе и забить, послав Била к черту, но не известно сработает ли это заклинание лет эдак через пять. Интеграция с Европой идет полным ходом, Третий Рим (известный под логотипом U.S.A) рвется в заснеженные леса России, всюду устанавливая свои порядки и законы... Любой обладатель лицензионной копии Windows (или другой программы) вправе вызывать готовый драйвер откуда угодно без каких бы то ни было разрешений и без выплаты дополнительного вознаграждения, но модифицировать двоичный код ему позволят едва ли.

Но мы ведь и не собираемся ничего модифицировать! Мы берем готовый NTOSKRNL.EXE и... Собственно, это все. Работы предстоит не так уж и много. Достаточно просто спроецировать его по адресам, указанным в заголовке PE-файла (а NTOSKRNL.EXE это обычный PE-Файл) и разобраться с таблицей экспорта, используемой драйверами. Короче говоря, мы должны реализовать свой собственный PE-загрузчик и заснуть его в загружаемый модуль ядра или в само ядро. Чтобы не мучаться, можно хлебнуть вина и содрать готовый загрузчик оттуда. Нет, это не спиртной напиток (хакеры предпочитают пиво), это эмулятор Windows'a такой. Он так и называется Wine (Windows Emulator).

Взаимодействие NTOSKRNL.EXE с LINUX/BSD ядром будет происходить через переходной код, эмулирующий HAL. Этот код мы будем должны написать сами, однако, ничего сложного в этом нет и объем работы предстоит минимальный, поскольку HAL содержит немного функций, да и те простые как самовар. Сложнее подружить диспетчер системных вызовов в внешним миром, то есть миром LINUX/BSD. Основная проблема в том, что интерфейс Диспетчера ни хрена не документирован и к тому же подвержен постоянным изменениям. В Windows 2000 он один, в Windows XP он другой, а потом Microsoft вновь придумает новую пакость и вся наша работа полетит к черту. Поэтому, приходится хитрить и тащить за собой не только NTOSKRNL.EXE, но еще и NTDLL.DLL. Некоторые могут спросить: а на фига, собствеено? Какое отношение NTDLL.DLL имеет к драйверам и ядру? Драйвера его не вызывают, да и сам NTDLL.DLL представляет собой всего лишь набор переходников к NTOSKRNL.EXE.

Так ведь тут-то и зарыта собака (да вы ее помните, ну там самая которая съела кусок мяса)! Интерфейс NTDLL.DLL худо бедно документирован и остается практически неизменным уже на протяжении многих лет, поэтому, его смело можно брать за основу. После этого остается "всего лишь" связать NTDLL.DLL с миром LINUX/BSD, то есть написать транслятор запросов к драйверам. Это не так-то просто сделать, поскольку писать придется достаточно много и работа отнимет не один день и даже не одну неделю, а с учетом отладки потребуется как минимум месяц. Но работа стоит того!!!

По крайней мере, в LINUX/BSD наладится нормальная работа с NTFS и некоторыми другими драйверами ввода-вывода. С видео-картами, правда, все значительно сложнее, поскольку они, как и следует из рис. 3, взаимодействуют отнюдь не с Диспетчером ввода-вывода (который находятся внутри NTOSKRNL.EXE), а с подсистемой win32. В Winnows 2000 она реализована в файле win2k.sys. Как обстоят дела в других системах — не знаю, да это и не важно. Драйвер win2k.sys — лишь малая часть того, что ему нужно для работы и просто так перетащить в LINUX/BSD его не получится. За ним неизбежно потянется все его окружение и написать столько "оберток" будет практически нереально. То есть, реально конечно, но сколько это потребует времени и сил? Переписать видео-драйвер гораздо проще, не говоря уже о том, что в этом случае он будет намного более производителен. Кстати говоря, компании NVIDIA и ATI в последнее время наладили выпуск LINUX/BSD драйверов под наиболее популярные чипсеты, так что проблема снимается сама собой.

Рисунок 4 видео-драйвера под LINUX x86 и x86-64, от ATI

Рисунок 5 видео-драйвера для LINUX x86, x86-64, IA64, FreeBSD x86 и Solaris x86/x64 от NVIDIA

готовый пример реализации или спроектировать — означает украсть

Конкретные переносы драйверов из мира Windows в LINUX/BSD мне неизвестны, однако, под MS-DOS кажется есть что-то похожее. Речь идет о проекте Марка Руссиновича "NTFS for MS-DOS" — известного хакера и исследования недр NT. Бесплатная версия (http://www.sysinternals.com/Utilities/NtfsDosProfessional.html) может только читать, а платную легко найти в Осле. Специальный мастер установки просит указать путь к системному каталогу Windows и создает две дискеты, на которые ожесточенно записывает что-то тяжелое. Давайте посмотрим, что именно:

```
30.10.2005  19:01               904 414 NTOSKRNL.gz
11.02.2002  09:39                89 472 ntfspro.exe
30.10.2005  19:00               314 665 NTFS.gz
30.10.2005  19:01                 1 403 C_866.gz
               4 файлов      1 309 954 байт
               0 папок         146 944 байт свободно
```

Рисунок 6 содержимое первой дискеты NTFS for MS-DOS

```
30.10.2005  19:03               212 681 AUTOCHK.gz
30.10.2005  19:04               219 099 NTDLL.gz
30.10.2005  19:04                 1 633 C_437.gz
30.10.2005  19:04                 1 467 C_1252.gz
30.10.2005  19:04                   746 L_INTL.gz
08.02.2002  10:45                56 748 ntfschk.exe
```

```
6 файлов          492 374 байт
0 папок           964 096 байт свободно
```

Рисунок 7 содержимое второй дискеты NTFS for MS-DOS

Начнем с первой дискеты (которая, кстати говоря, обычно бывает системной, поскольку NTFS for MS-DOS работает только из-под белого листа, то есть я хотел сказать "черного экрана", однако, для наглядности все системные файлы удаленны). Здесь находится только один исполняемый файл "ntfspro.exe", представляющий собой транслятор запросов слинкованный с расширением защищенного режима "WDOSX 0.96 DOS extender" от Michael Tippach (wuschel@geocities.com).

"NTFS.gz" – это "родной" NTFS.SYS драйвер, вытащенный из системного каталога Windows и для экономии места упакованный архиватором gzip. Для распаковки нам потребуется либо LINUX, либо pkzip для Windows/MS-DOS. Сравнив его с оригинальным файлом драйвера, мы не найдем никаких изменений! "NTOSKRNL.gz" – это ядро системы (NTOSKRNL.EXE), точно так же вытащенное и упакованное. Никаких изменений в нем нет.

На другой дискете находится "NTDLL.gz" (о происхождении которого догадаться нетрудно) и ntfschk.exe. Последний представляет собой полностью переписанный вариант штатной утили "chkdsk.exe", поскольку чтобы заставить консольное приложение заработать в MS-DOS пришлось бы эмулировать еще множество функций, что в планы Руссиновича очевидно не входило (тем не менее, легендарный хакер Юрий Харон все-таки создал расширитель, способный запускать Windows-приложения из под голого ДОС'а, без обращения к Windows вообще! Все умещается на одну дискетку, — красота! Сам расширитель можно скачать с http://www.doswin32.com. Для некоммерческого применения он бесплатен).

Еще на дискетах содержатся файлы C_866.gz, AUTOCHK.gz, C_437.gz, C_1252.gz, L_INTL.gz содержащие языковые страницы и прочую служебную мишуру, без которой можно в принципе и обойтись.

Суть в том, что ядро проекта "NTFS for MS-DOS" составляют три файла: NTOSKRNL.EXE, NTDLL.DLL и NTFS.SYS, которые помещаются в своеобразную скорлупу файла NTFSPRO.EXE, переводящего процессор в защищенный режим и транслирующего MS-DOS-запросы в "язык", понятный NTFS.SYS и наоборот. Как видите, это работает. Конечно, LINUX/BSD это совсем не чистая MS-DOS. Ядро по своему распределят прерывания и другие системные ресурсы, поэтому при написании "скорлупы-оболочки" возникает множество технических проблем, но все они решаемы. Пример аналогичного решения можно найти в другом проекте Марка Руссиновича "NTFS for Windows 9x". Здесь так же используется "скорлупа", создающая адекватное окружения для NTOSKRNL.EXE и транслятор запросов, но она уже работает совсем не в голой MS-DOS, с которой все и так ясно, а в агрессивной Windows 9x, которая отличается от NT ничуть не меньше, чем LINIX/BSD.

заключение

Так что, написать драйверную "скорлупу" для LINUX/BSD вполне реально и ничего фантастичного в этом нет. Ее достаточно создать лишь однажды, после чего в ней будет можно запускать различные драйвера. Почему бы нам, хакерам, не скооперироваться и не заняться этим? Например, создать новый проект на www.sourceforge.net, набрать группу и... оттянуться по полной программе. Ведь это действительно ХАКЕРСТВО, а не тупой бух и склад! Ну так чего же мы ждем?! Поехали!!!