MBR своими руками

крис каперски ака мыщъх, no-email

сегодня мы напишем свой менеджер мультизагрузки. это такая штука, что сидит в загрузочном секторе и грузит любую из нескольких установленных операционных систем по нашему выбору. он познакомит нас с прерыванием INT 13h, таблицей разделом и кое-чем еще

введение

Стандартный загрузчик, устанавливаемый большинством осей по умолчанию, слишком примитивен, чтобы его воспринимать всерьез, а нестандартные загрузчики от независимых разработчиков обычно слишком неповоротливы, монстроузны и ненадежны. Вот и давайте напишем свой! Пока мы будет его писать, мы познаем дао и дзен ассемблера, научимся отлаживать программы без отладчика, и попробуем низкоуровневое железо винчестера на вкус.

начальная теоретическая подготовка

Загрузка системы начинается с того, что BIOS считывает первый сектор жесткого диска, размещает его в памяти по адресу 0000:7C00h и передает сюда управление. Программисты называют его Главным Загрузочным Сектором (Master Boot Record), или сокращенно MBR. В начале MBR расположен машинный код загрузчика, за ним идет Таблица Разделов (Partition Table), описывающая схему разбиения логических дисков. В конце загрузочного сектора находится сигнатура 55h AAh, говорящая BIOS'у о том, что это действительно MBR, а не что-то еще.

Загрузчик должен проанализировать Таблицу Разделов, найти предпочтительный логический диск, считать его первый сектор (он называется загрузочным — boot) и передать ему бразды правления. Вот минимум требований, предъявляемых к стандартному загрузчику, главный недостаток которого заключается в том, что на каждом логическом диске может быть установлена только одна операционная система, причем, она должна быть установлена непременно на Primary Master'e, в противном случае загрузчик ее просто "не увидит" и нам придется менять порядок загрузки в BIOS Setup, а это слишком хлопотно и утомительно. Наш загрузчик будет свободен от всех этих глупых ограничений, но прежде чем зарываться вглубь, окинем MBR беглым взглядом.

Воспользовавшись любым редактором диска (например, Microsoft Disk Probe из комплекта Resource Kit, прилагаемого к лицензионной Windows), считаем первый сектор физического диска. Он должен выглядеть приблизительно так:

```
0000000000:  33 C0 8E D0 BC 00 7C FB   50 07 50 1F FC BE 1B 7C  3 O   ¦√P•P▼ ' ←¦
0000000010:  BF 1B 06 50 57 B9 E5 01   F3 A4 CB BE BE 07 B1 04  ┐←♠PW╢x╝єдт╛╛•╓╢♦
0000000020:  38 2C 7C 09 75 15 83 C6   10 E2 F5 CD 18 8B 14 8B  8,¦○u§Г├►тї=↑Л¶Л
0000000030:  EE 83 C6 10 49 74 16 38   2C 74 F6 BE 10 07 4E AC  юГ├►It▪8,tЙ╛►•Nм
0000000040:  3C 00 74 FA BB 07 00 B4   0E CD 10 EB F2 89 46 25  < t·╗• ┤♂=►ыЄЙF%
0000000050:  96 8A 46 04 B4 06 3C 0E   74 11 B4 0B 3C 0C 74 05  ЦКF♦┤♠<♂t◄┤♂<♀t♣
0000000060:  3A C4 75 2B 40 C6 46 25   06 75 24 BB AA 55 50 B4  :─u+@╞F%♠u$╗кUP┤
0000000070:  41 CD 13 58 72 16 81 FB   55 AA 75 10 F6 C1 01 74  A=‼Xr▪Б√Uкu►Й┴╝t
0000000080:  0B 8A E0 88 56 24 C7 06   A1 06 EB 1E 88 66 04 BF  ♂КрИV$╟╞♠б♠ы▲Иf♦┐
0000000090:  0A 00 B8 01 02 8B DC 33   C9 83 FF 05 7F 03 8B 4E  ◙ ┐►☺Л▄З╔Г ♣?♥ЛN
00000000A0:  25 03 4E 02 CD 13 72 29   BE 56 07 81 3E FE 7D 55  %♥N☻=‼r)╛V•Б>▪}U
00000000B0:  AA 74 5A 83 EF 05 7F DA   85 F6 75 83 BE 2D 07 EB  кtZГя♣?╔ЕЙuГ╛-•ы
00000000C0:  8A 98 91 52 99 03 46 08   13 56 0A E8 12 00 5A EB  КШCRЩ♥F◘‼V◙ш↨ Zы
00000000D0:  D5 4F 74 E4 33 C0 CD 13   EB B8 00 00 81 29 58 16  ╒Otф3└═‼ы╕  Б)X▬
00000000E0:  56 33 F6 56 56 52 50 06   53 51 BE 10 00 56 8B F4  V3ЙVVRP♠SQ╛► VЛ╘
00000000F0:  50 52 B8 00 42 8A 56 24   CD 13 5A 58 8D 64 10 72  PR╕ BKV$=‼ZXHd►r
0000000100:  0A 40 75 01 42 80 C7 02   E2 F7 F8 5E C3 EB 74 8D  ◙@u☺BА╟☻тў°^├ыtH
0000000110:  A5 AF E0 A0 A2 A8 AB EC   AD A0 EF 20 E2 A0 A1 AB  еправильная табл
0000000120:  A8 E6 A0 20 E0 A0 A7 A4   A5 AB AE A2 00 8E E8 A8  ица разделов Оши
0000000130:  A1 AA A0 20 AF E0 A8 20   A7 A0 A3 E0 E3 A7 AA A5  бка при загрузке
0000000140:  20 AE AF A5 E0 A0 E6 A8   AE AD AD AE A9 20 E1 A8   операционной си
0000000150:  E1 E2 A5 AC EB 00 8E AF   A5 E0 A0 E6 A8 AE AD AD  стемы Операционн
0000000160:  A0 EF 20 E1 A8 E1 E2 A5   AC A0 20 AD A5 20 AD A0  ая система не на
0000000170:  A9 A4 A5 AD A0 00 00 00   00 00 00 00 00 00 00 00  йдена
0000000180:  00 00 00 8B FC 1E 57 8B   F5 CB 00 00 00 00 00 00     ЛЮ▲WЛї╥
0000000190:  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00000001A0:  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00000001B0:  00 00 00 00 00 00 00 00   18 F3 97 A1 00 00 80 01          ↑єЧб  А☺
00000001C0:  01 00 07 FE FF FF 3F 00   00 00 02 68 54 02 00 00  ☺ •▪  ?   ☻hT☻
00000001D0:  C1 FF 0F FE FF FF 41 68   54 02 80 7C FC 06 00 00  ┴ ☼▪  AhT☻А|№♠
00000001E0:  00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00000001F0:  00 00 00 00 00 00 00 00   00 00 00 00 00 00 55 AA                Uк
```

Рисунок 1 внешний вид MBR, очень похоже на Матрицу не правда ли?

Первые 1BBh байт занимают код и данные загрузчика, среди которых отчетливо выделяются текстовые строки (кстати говоря, русифицировав сообщения загрузчика, Microsoft допустила грубейшую стратегическую ошибку, ведь никакого кириллического фонта в BIOS'е нет и русские символы выглядят бессмысленной абракадаброй).

По смещению 1BBh расположен четырехбайтовый идентификатор диска, принудительно назначаемый Windows при запуске Disk Manager'a. Коварство Microsoft не знает границ! Еще со времен первых IBM PC (тогда они назывались XT), загрузчик владел первыми 1BEh байтами MBR-сектора, и достаточно многие загрузчики (и вирусы!) использовали эти байты на всю катушку. Нетрудно сообразить, что произойдет, если внутрь загрузчика вдруг запишется идентификатор. Это убьет его! Поэтому, байты 1BBh – 1BEh лучше чего не трогать.

Со смещения 1BEh начинается Таблица Разделов, представляющая собой массив из четырех записей типа partition. Каждый partition описывает свой логический диск, что позволяет нам создавать до четырех разделов на каждом HDD. Динамические диски, впервые появившиеся в W2K, хранятся в Базе Менеджера Логических Дисков (Logical Disk Manager Database) и в таблице разделов присутствовать не обязаны.

В общем, устройство Главного Загрузочного Сектора выглядит так:

смещение	размер	назначение
000h	перемен.	код загрузчика
1BBh	4h	идентификатор диска
1BEh	10h	partition 1
1CEh	10h	partition 2
1DEh	10h	partition 3
1EEh	10h	partition 4
1FEh	0x2	признак таблицы разделов сигнатура 55h Aah

Таблица 1 устройство MBR

Таблица Разделов — это святая святых операционной системы. Каждая запись partition состоит из: ***адресов*** начала и конца раздела, ***типа*** раздела (NTFS, FAT16, FAT32…), количество секторов в разделе и флага "загружечности" раздела.

Все адреса задаются либо CHS (Cylinder-Head-Sector — Цилиндр-Головка-Сектор), либо LBA (Logical Block Address – Логический Адрес Блока) формате. Конкретный формат

определяется типом раздела (Boot ID), записанным в 04h байте. Количество существующих типов огромно и было бы слишком утомительно перечислять их здесь. В таблице 3 приведены лишь самые популярные из них.

В CHS-формате, 01h и 05h байты partition'a хранят номер первой и последней головки раздела (см. таблицу 2). Байты 02h и 06h хранят 5 младших бит начального/конечного сектора и по два старших бита номера цилиндра, а оставшиеся биты лежат в следующем байте. Получается довольно запутанная схема, да к тому же адресующая только первые 8 Гбайт дискового пространства (CHS адрес занимает три байта или 24 бита, что при длите сектора в 512 байт дает $512*2^{24}$ == 8.388.608 байт). Ха! Да жесткие диски преодолели этот барьер еще в прошлом веке! Это было достигнуто за счет введения LBA-адресации, последовательно нумерующей все сектора от 0 до многодетной матери. Начало раздела хранится в 32-битном поле relative offset (относительное смещение), содержащим смещение первого сектора раздела от начала partition или, попросту говоря, расстояние между концом partition и началом раздела. Конец раздела в явном нигде не хранится, вместо этого в специальном 32-битном поле partition size записывается количество секторов в разделе. Как нетрудно подсчитать, предельно допустимый размер одного раздела составляет ($512*2^{32}$ == 2.199.023.255.552 байт или 2.048 Гбайт), а совокупный объем всего диска вообще не ограничен! Так что, для сегодняшних нужд LBA-адресации вполне достаточно, а там уж мы что-нибудь придумаем.

смещение					**разм.**	**назначение**
000	1BE	1CE	1DE	1EE	BYTE	флаг активного загрузочного раздела. (Boot Indicator) 80h – загрузочный раздел, 00h – не загрузочный
001	1BF	1CF	1DF	1EF	BYTE	стартовая головка раздела
002	1C0	1D0	1E0	1F0	BYTE	стартовый сектор раздела (биты 0 – 5) старшие биты стартового цилиндра (биты 6-7)
003	1C1	1D1	1E1	1F1	BYTE	младшие биты стартового цилиндра (биты 0-7)
004	1C2	1D2	1E2	1F2	BYTE	идентификатор системы (Boot ID), см. таблицу.3
005	1C3	1D3	1E3	1F3	BYTE	конечная головка раздела
006	1C4	1D4	1E4	1F4	BYTE	конечный сектор раздела (биты 0 – 5) старшие биты конечного цилиндра (биты 6-7)
007	1C5	1D5	1E5	1F5	BYTE	младшие биты конечного цилиндра (биты 0-7)
008	1C6	1D6	1E6	1F6	DWORD	смещение раздела относительно начала таблицы разделов в секторах
00C	1CA	1DA	1EA	1FA	DWORD	кол-во секторов раздела

Таблица 2 формат partition

Boot ID	тип раздела
00h	раздел свободен
0x01	FAT12 (менее чем 32.680 секторов в томе или 16 Мбайт), CHS
0x04	FAT16 (32,680-65,535 секторов или 16-33 Мбайт), CHS
0x05	**расширенный раздел (extended partition)**, CHS
0x06	BIGDOS FAT16 раздел (33 Мбайт – 4 Гбайт), CHS
0x07	NTFS-раздел, CHS
0x0B	FAT32 раздел, CHS
0x0C	FAT32 раздел с поддержкой расширенной BIOS INT 13h, LBA
0x0E	BIGDOS FAT16 раздел с поддержкой расширенной BIOS INT 13h, LBA
0x0F	**расширенный раздел с поддержкой расширенной BIOS int 13h**, LBA
0x42	динамический диск, LBA
0x86	legacy FT FAT16 раздел, CHS
0x87	legacy FT NTFS раздел, CHS
0x8B	Legacy FT volume formatted with FAT32, CHS
0x8C	Legacy FT volume using BIOS INT 13h extensions formatted with FAT32, LBA

Таблица 3 возможные значения Boot ID

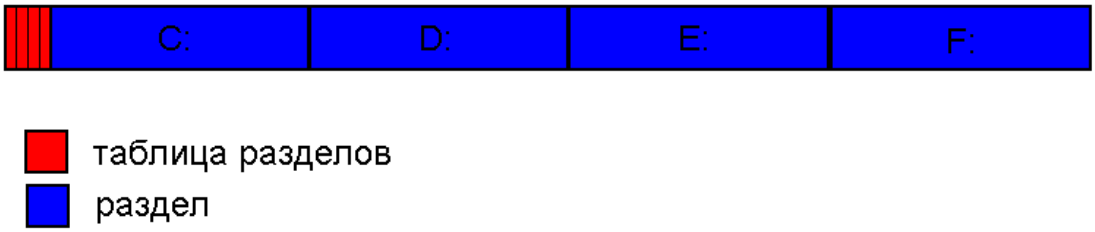

Рисунок 2 Основная Таблица Разделов, разбивающая винчестер на четыре логических диска

Четыре раздела partition обслуживают до четырех логических дисков, а больше уже никак. На большее в MBR-секторе просто не хватает места! Но ведь хорошо известно, что FDISK может разбивать винчестер хоть на 26 разделов. Как же ему это удается? А вот как! Поимо Основной Таблицы Разделом, хранящейся в MBR мы можем создавать любое количество Расширенных Таблиц Разделов (Extended Partition Table), разбросанных по всему диску (см. рис. 3)

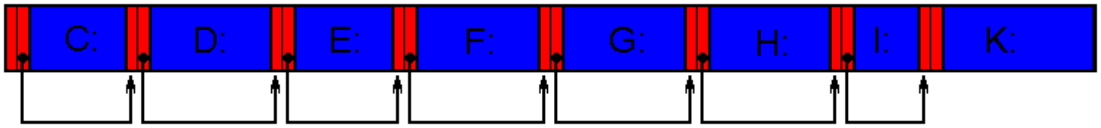

Рисунок 3 несколько Расширенных Таблиц Разделов, объединенных в одну цепочку, и разбивающие винчестер на любое количество логических дисков

Если partition имеет тип 05h или 0Fh, то она указывает совсем не на начало раздела, а на следующий MBR. Точнее, не совсем MBR, но нечто очень на него похожее. В нем присутствует полноценная Таблица Разделов с четырьмя входами: partition 1, partition 2, partition 3 и partition 4, каждая из которых указывает либо на логический диск, либо на новый MBR. Длина такой цепочки практически неограниченна и может превышать 26. Однако, назначить буквы всем последующим разделам уже не удаться и под Windows 9x они будут просто не видны. Windows NT поддерживает гибридный механизм наименования разделов — по буквам и по именем, поэтому ей эти ограничения не страшны.

Стандартный загрузчик позволяет запускать системы только из Основной Таблицы Разделов. Цепочку MBR'ов он не анализирует. В своем загрузчике мы исправим этот недостаток.

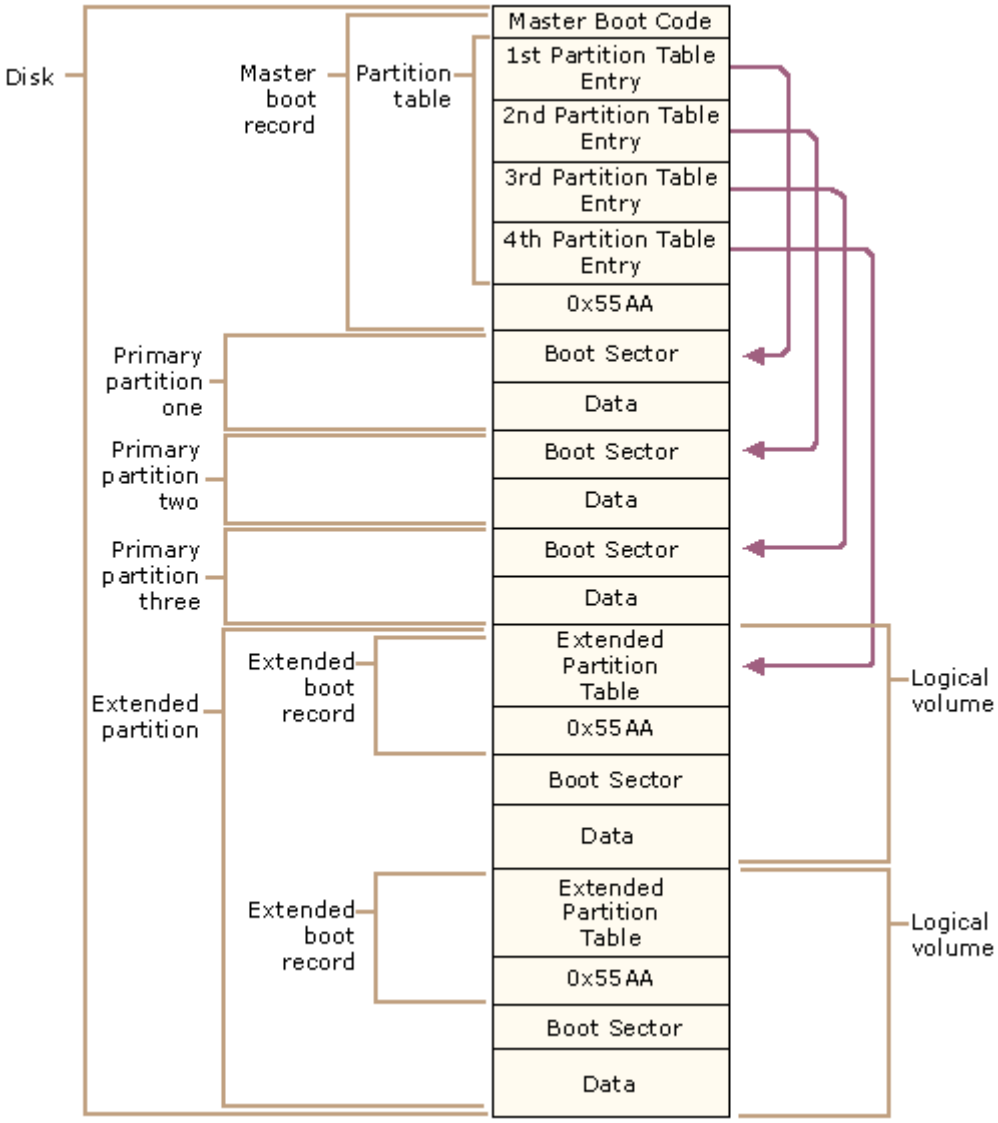

Рисунок 4 структурная схема типичной Расширенной Таблицы Разделов

интерфейс INT 13h

Управлять дисками можно как через порты ввода/вывода, так и через BIOS. Порты намного более могущественны и интересны, однако, BIOS программируется намного проще, к тому же она поддерживает большое количество разнокалиберных накопителей, абстрагируя нас от конструктивных особенностей каждой конкретной модели. Поэтому, мы будем действовать через нее, а точнее через интерфейс прерывания INT 13h.

Попробуем прочитать сектор с диска в CHS-mode. Естественно, действовать нужно из самого MBR или из "голой" MS-DOS, иначе у нас ничего не получится, ведь Windows NT блокирует прямой доступ к диску даже из режима "Эмуляции MS-DOS"!

Номер функции заносится в регистр AH. В случае чтения он равен двум. Регистр AL отвечает за количество обрабатываемых секторов. Поскольку, мы собираемся читать по одному сектору за раз, занесем сюда единицу. Регистр DH хранит номер головки, а DL – номер привода (80h – первый жесткий диск, 81h – второй и так далее). Пять младших битов регистра CL задают номер сектора, оставшиеся биты регистра CL и восемь битов регистра CH определяют номер цилиндра, который мы хотим прочитать. Регистровая пара ES:BX указывает на адрес буфера-приемника. Вот, собственного говоря, и все. После выполнения команды INT 13h считываемые данные окажутся в буфере, а если произойдет ошибка (например, головка споткнется о BAD-сектор) BIOS установит флаг переноса (carry flag) и мы будем вынуждены либо повторить попытку, либо вывести грустное сообщение на экран.

На ассемблерном языке это звучит так:

```
MOV SI, 1BEh           ; на первый partition
MOV AX, CS             ; настраиваем ES
MOV ES, AX             ;
MOV BX, buf            ; смещение буфера
…
read_all_partitions:
        MOV AX, 0201h          ; читать 1 сектор с диска
        MOV DL, 80h            ; читать с первого диска
        MOV DH, [SI+1]         ; стартовый номер головки
        MOV CX, [SI+2]         ; стартовый сектор с цилиндром
        INT 13h
        JC error               ; ошибка чтения

        ; обрабатываем считанный boot-сектор или extended partitions
        ; ==========================================================
        ;
        CMP byte [SI], 80h
        JZ LOAD_BOOT           ; это загрузочный раздел
                               ; передаем на него управление

        CMP byte [SI+4], 05h
        JZ LOAD_CHS_EXT        ; это Расширенная Таблица Разделов в CHS-формате

        CMP byte [SI+4], 0Fh
        JZ LOAD_LBA_EXT        ; это Расширенная Таблица Разделов в LBA-формате

        ADD SI, 10h            ; переходим на следующую partition
        CMP SI, 1EEh
        JNA read_all_partitions; читаем все партиции одну за другой
…
buf rb 512             ; буфер на 512 байт
```

Листинг 1 код, считывающий загрузочный сектор или Расширенную Таблицу Разделов

Запись сектора в CHS-режиме происходит практически точно так же, только регистр AH равен не 02h, а 03h. С LBA-режимом разобраться намного сложнее, но мы, как настоящие хакеры, его обязательно осилим. Вот только пива хлебнем.

Чтение сектора осуществляется функцией 42h (AH = 42h). В регистр DL, как и прежде, заносится номер привода, а вот регистровая пара DS:SI указывает на адресный пакет (disk address packet), представляющий собой продвинутую структуру следующего формата:

смещение	тип	назначение
00h	BYTE	размер пакета 10h или 18h
01h	BYTE	зарезервировано и должно быть равно нулю
02h	WORD	сколько секторов читать
04h	DWORD	32-разрядный адрес буфера-приемника в формате seg:offs
08h	QWORD	стартовый номер сектора для чтения
10h	QWORD	64-разряный плоский адрес буфера приемника (используется только если 32-разряный адрес равен FFFF:FFFF)

Таблица 4 адресный пакет, используемый для чтения/записи секторов в режиме LBA

Код, читающий сектор в LBA-режиме, в общем случае выглядит так:

```
MOV DI, 1BEh                   ; на первый partition
MOV AX, CS                     ; настраиваем…
MOV buf_seg                    ;              …сегмент
MOV EAX, [DI+08h]              ; смещение partition относительно начала раздела
ADD EAX, EDI                   ; EDI должен содержать номер сектора текущего MBR
MOV [X_SEC]                    ;
…
read_all_partitions:
        MOV AH, 42h            ; читать сектор в LBA-режиме
        MOV DL, 80h            ; читать с первого диска
        MOV SI, dap            ; смещение адресного пакета
        INT 13h
        JC error               ; ошибка чтения

…
dap:
```

```
packet_size     db 10h          ; размер пакета 10h байт
reserved        db 00h          ; заначка для будущих расширений
N_SEC           dw 01h          ; читаем один сектор
buf_seg         dw 00h          ; сюда будет занесен сегмент буфера-приемника
buf_off         dw buf          ; смещение буфера-приемника
X_SEC           dd 0            ; сюда будет занесен номер сектора для чтения
                dd 0            ; реально неиспользуемый хвост 64-битного адреса

buf rb 512                      ; буфер на 512 байт
```

Листинг 2 чтение сектора с диска в LBA-режиме

Запись осуществляется аналогично, только регистр AH содержит не 42h, а 43h. Регистр AL определяет режим: если бит 0 равен 1, BIOS выполняет не запись, а ее эмуляцию. Бит 2, будучи взведенным задействует запись с проверкой. Если AL равен 0, выполняется обыкновенная запись по умолчанию.

Теперь, освоившись с дисковыми прерываниями, перейдем к обсуждению остальных аспектов программирования.

как программируют загрузчики

Лучше всего загрузчики программируются на FASM. С точки зрения ассемблера загрузчик представляет собой обыкновенный двоичный файл, предельно допустимый объем которого составляет 1BBh (443) байт. Немного? Но не будет спешить с выводами. Всякий раздел всегда начинается с начала цилиндра, а это значит, что между концом MBR и началом раздела имеется по меньшей мере sector per track свободных секторов. Практически все современные винчестеры имеют по 64 секторов в треке, что дает нам: 443 + 63*512 == 32.699 байт или ~32 Кбайт. Да в этот объем даже графический интерфейс с мышью и голой красавицей на обоях уместить можно. Но мы не будем! Настоящие хакеры работают в текстовом режиме с командной строкой, а красавиц лучше трахать, чем смотреть.

Как уже говорилось, BIOS загружает MBR по адресу 7C00h, поэтому в начале ассемблерного кода должна стоять директива ORG 7C00h, а еще USE16 — ведь загрузчик выполняется в 16-разряном реальном режиме. Позже, при желании он может перейти в защищенный режим, но это будет уже потом. Не будет лезть в такие дебри.

Обнаружив загрузочный раздел (а обнаружить это можно по флагу 80h, находящемуся по смещению от начала partition), загрузчик должен считать первый сектор этого раздела, разместив его в памяти по адресу 0000:7C000h, то есть аккурат поверх своего тела. А вот это уже нехорошо! И чтобы не вызвать крах системы, загрузчик должен заблаговременно перенести свою тушу в другое место, что обычно осуществляется командой MOVSB. Копироваться можно в любое место памяти — от 0080:0067h до 9FE00h. Память, расположенную ниже 0080:0067h лучше не трогать, т. к. здесь находятся вектора прерываний и системные переменные BIOS'a, а от A000h и выше начинается область отображения ПЗУ, так что предельно доступный адрес равен A000h — 200h (размер сектора) == 9FE00h.

Что еще? Ах да! Трогать DL регистр ни в коем случае нельзя, поскольку в нем передается номер загрузочного привода. Некоторые загрузчики содержат ошибку, всегда загружаясь с первого жесткого диска. Стыдно не знать, что BIOS уже лет десять как позволяют менять порядок загрузки и потому загрузочным может быть любой привод.

Кстати говоря, FASM – единственный известный мне ассемблер, "переваривающий" команду дальнего вызова JMP 0000:7C000h напрямую. Все остальные ассемблеры заставляют извращаться приблизительно так: PUSH offset_of_target/PUSH segment_of_target/RETF. Здесь мы заталкиваем в стек сегмент и смещение целевого адреса и выполняем далекий RETF, переносящий нас на нужное место. Еще можно воспользоваться самомодифицирующимся кодом, собрав команду JMP FAR "вручную" или просто расположить целевой адрес в одном сегменте с исходным адресом (например, 0000:7C000h → 0000:7E000h), но это все муторно и утомительно.

В общем, скелет нашего загрузчика будет выглядеть так:

```
use16
ORG 7C00h
CLD                     ; копируем слева направо (в сторону увеличения адресов)
MOV SI,7C00h            ; откуда копировать
MOV DI,7E00h            ; куда копировать
MOV CX,200h             ; длина сектора
REP MOVSB               ; копируем
```

```
; // выбираем раздел, который мы хотим загрузить,
; // считываем его в память по адресу 0000:7C000h
; // см. листинг 1, 2

JMP 0000:7C000h        ; передаем управление на boot-сектор
```

Листинг 3 скелет простейшего загрузчика на FASM'е

инсталляция нашего загрузчика в MBR

Под старушкой MS-DOS записать свой загрузчик в MBR было просто — достаточно дернуть прерывание INT 13h, функция 03h (запись сектора). Но под Windows NT этот прием уже не работает и приходится прибегать к услугам функции `CreateFile`. Если вместо имени открываемого фала указать название устройства, например, "`\\.\PHYSICALDRIVE0`" (первый физический диск), мы сможем свободно читать и записывать его сектора вызовами `ReadFile` и `WriteFile` соответственно. При этом, флаг `dwCreationDisposition` должен быть установлен в значение `OPEN_EXISTING`, а `dwShareMode` — в значение `FILE_SHARE_WRITE`. Еще потребуют права root'a или в терминологии Windows – администратора, иначе ничего не получится.

Законченный пример вызова `CreateFile` выглядит так:

```
XOR EAX,EAX
PUSH EAX                                    ; hTemplateFile
PUSH dword FILE_ATTRIBUTE_NORMAL            ; dwFlagsAndAttributes
PUSH dword OPEN_EXISTING                    ; dwCreationDisposition
PUSH EAX                                    ; lpSecurityAttributes
PUSH dword FILE_SHARE_WRITE                 ; dwShareMode
PUSH dword (GENERIC_WRITE OR GENERIC_READ)  ; dwDesiredAccess
PUSH DEVICE_NAME                            ; имя устройства
CALL CreateFile                             ; открываем устройство
INC EAX
TEST EAX,EAX
JZ error
DEC EAX
…
DEVICE_NAME DB "\\.\PHYSICALDRIVE0",0
BUF RB 512                                  ; буфер
```

Листинг 4 открытие непосредственного доступа к жесткому диску под Windows NT

Открыв физический диск и убедившись в успешности этой операции, мы должны прочитать оригинальный MBR-сектор в буфер, перезаписать первые 1BBh байт, ни в коем случае не трогая Таблицу Разделов и сигнатуру 55h AAh (мы ведь не хотим, чтобы диск перестал загружаться, верно?). Остается записать обновленный MBR на место и закрыть дескриптор устройства. Все! После перезагрузки все изменения вступят в силу, а может быть и не вступят… Загрузчик жестоко мстит за малейшие ошибки проектирования и чтобы не потерять содержимое своих разделов, для начала лучше попрактиковаться на VM Ware или любом другом эмуляторе PC.

Под Windows 9x, кстати говоря, трюк с `CreateFile` не работает. Но там можно воспользоваться симуляцией прерываний из DMPI или обратится к ASPI-драйверу. Оба способа подробно описаны в моей книге "Техника защиты компакт-дисков от копирования", фрагмент которой прилагается к журналу. И хотя в ней речь идет о CD, а не о HDD, жесткие диски программируются аналогичным способом.

содержимое комакт-диска

На прилагаемом к журналу компакт-диске расположены исходные коды нескольких загрузчиков, которые будет полезно изучить, прежде чем писать свой собственный. Все они распространяются по лицензии GPL или BSD, то есть без ограничений.

- ge2000.asm:
  - тщательно откоментированный Stealth-вирус, подменяющий системный загрузчик своим собственным. не опасен и может быть использован в учебных целях;
- mbr.asm:

- предельно простой, но полнофункциональный загрузчик с поддержкой разделов свыше 8 Гбайт, но увы — без комментариев;

- boot.asm:
  - отличный менеджер мулти-загрузки с подробными комментариями, переходит в защищенный режим, может грузится с дискеты, компакт-диска, zip'a, винчестера и т.д., поддерживает разделы свыше 8 Гбайт, показывает индикатор загрузки и делает множество других полезных вещей, которые не помешает изучить;
- cd-hack.SCSI.zip:
  - фрагмент книги "Техника защиты компакт-дисков от копирования", описывающий различные способы низкоуровневыми управления приводам с прикладного уровня под Windows, главным образом относится к оптическим накопителям, но во многом подходит и для жестких;

>>> врезка: отладка загрузчика

Отлаживать код загрузчиков невероятно трудно. Загрузчик получает управление задолго до запуска операционной системы, когда никакие отладчики еще не работают. Несколько лет назад это представляло огромную проблему и при разработке навороченных загрузчиков приходилось либо встраивать в них интегрированный мини-отладчик, либо выискивать ошибки руками, головой и карандашом. С появлением эмуляторов все изменилось. Достаточно запустить BOCHS и отлаживать загрузчик как и любую другую программу!

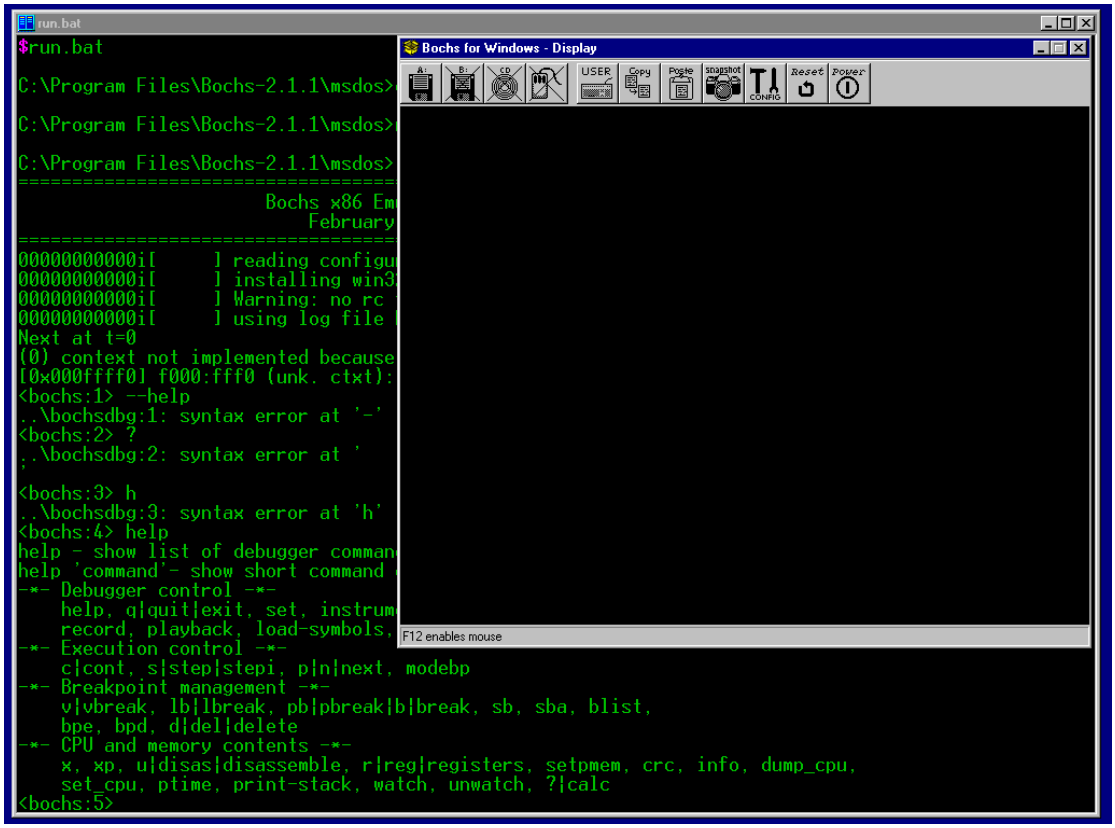

Рисунок 5 внешний вид эмулятора BOCHS, отлаживающего загрузочный сектор

заключение

Программирование загрузчиков — одна из тех немногих областей, в которых применение ассемблера действительно оправдано. Языки высокого уровня для этого слишком абстрагированы от оборудования и недостаточно гибки. Вот почему хакеры так любят возиться с загрузчиками, добавляя сюда множество новых фич таких например, как: автоматическая загрузка с CD-ROM или SCSI-винтов, противодействие вирусам, парольная защита с шифрованием данных и т.д. Здесь действительно есть где развернуться и показать себя. Но

читать о новых идеях скучно и неинтересно. Намного приятнее генерировать их самостоятельно. Так чего же мы сидим?!

>>> врезка: интересные ссылки

- **MBR and OS Boot Records**:
  - масса интересного материала по MBR (на английском языке) http://thestarman.narod.ru/asm/mbr/MBR_in_detail.htm;
- **BOCHS**:
  - отличный эмулятор со встроенным отладчиком, значительно облегчающий процесс "пуско-наладки" загрузочных секторов; бесплатен, распространяется с исходными текстами: http://bochs.sourceforge.net;
- **www.koders.com**:
  - отличный поисковик, нацеленный на поиск исходных кодов, по ключевому слову "MBR" выдает огромное количество загрузчиков на любой вкус;
- **Ralf Brown Interrupt List**:
  - знаменитый Interrupt List Ральфа Брауна, описывающий все прерывания, включая недокументированные (на английском языке)? http://www.pobox.com/~ralf;
- **OpenBIOS**:
  - проект "Открытого BIOS", распространяемого в исходных текстах, помогает понять некоторые неочевидные моменты обработки системного загрузчика: http://www.openbios.info/docs/index.html;

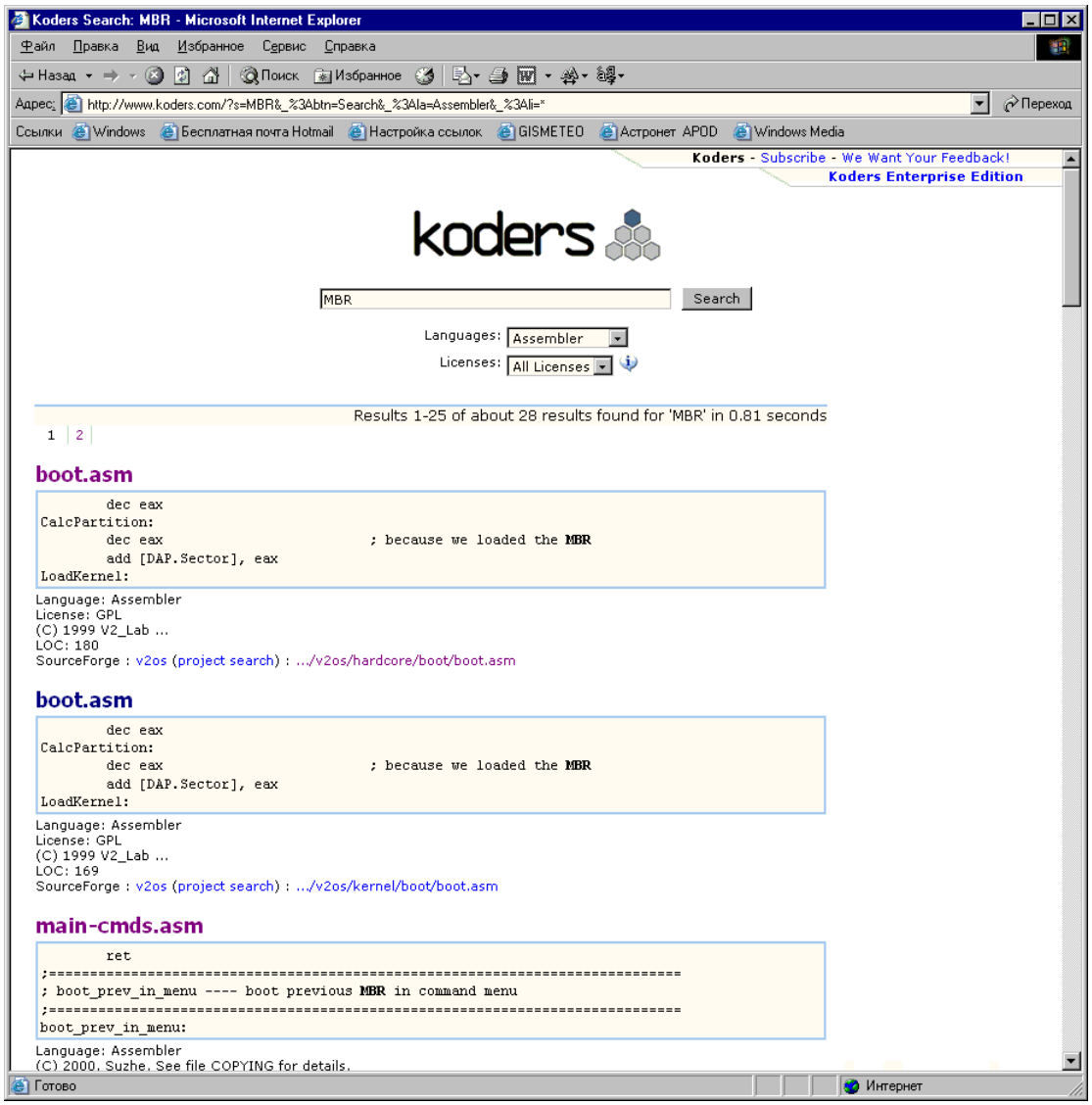

Рисунок 6 поиск исходных текстов MBR-загрузчиков на koders'e

```
D:\RALF\SEE.EXE
Files Help Category Topic Search
[▪]                    Interrupt Helper 1.0                    [↑]
c-1338          INT 13 U - QUICKCACHE II v4.20 - SET BUFFERED WRITES FOR SPECI
c-1339          INT 13 U - QUICKCACHE II v4.20 - SET READ BUFFER SIZE FOR SPEC
c-133A          INT 13 U - QUICKCACHE II v4.20 - SET WRITE BUFFER SIZE FOR SPE
c-133B          INT 13 U - QUICKCACHE II v4.20 - ENABLE/DISABLE ???
c-133C          INT 13 U - QUICKCACHE II v4.20 - ENABLE/DISABLE ???
c-133D          INT 13 U - QUICKCACHE II v4.20 - ENABLE/DISABLE CYLINDER FLUSH
c-133E          INT 13 U - QUICKCACHE II v4.20 - SET SINGLE-SECTOR BONUS
c-133F          INT 13 U - QUICKCACHE II v4.20 - SET BONUS THRESHOLD
c-1340          INT 13 U - QUICKCACHE II v4.20 - SET "sticky_max"
c-1341          INT 13 U - QUICKCACHE II v4.20 - SAVE/RESTORE ???
d-1341--BX55AA  INT 13 - IBM/MS INT 13 Extensions - INSTALLATION CHECK
d-1342          INT 13 - IBM/MS INT 13 Extensions - EXTENDED READ
N-134257DX1234  INT 13 U - Beame&Whiteside BWLPD - INSTALLATION CHECK
d-1343          INT 13 - IBM/MS INT 13 Extensions - EXTENDED WRITE
d-1344          INT 13 - IBM/MS INT 13 Extensions - VERIFY SECTORS
d-1345          INT 13 - IBM/MS INT 13 Extensions - LOCK/UNLOCK DRIVE
d-1346          INT 13 - IBM/MS INT 13 Extensions - EJECT MEDIA
d-1347          INT 13 - IBM/MS INT 13 Extensions - EXTENDED SEEK
d-1348          INT 13 - IBM/MS INT 13 Extensions - GET DRIVE PARAMETERS
d-1349          INT 13 - IBM/MS INT 13 Extensions - EXTENDED MEDIA CHANGE
d-134A          INT 13 - Bootable CD-ROM - INITIATE DISK EMULATION
d-134B00        INT 13 - Bootable CD-ROM - TERMINATE DISK EMULATION
d-134B01        INT 13 - Bootable CD-ROM - GET STATUS
```

Рисунок 7 просматриваем легендарный Interrupt List Ральфа Брауна,