Large-Disk-HOWTO

Andries Brouwer, <mailto:aeb@cwi.nl>
Перевод: Russian Linux Documentation Project
v2.2j, 5 февраля 2000

Все о геометрии диска и лимите в 1024 цилиндра.

Содержание

1 Введение

1.1 Авторские права

1.2 О RLDP

Перевод данного документа был выполнен силами переводчиков входящих в проект *RLDP* (Russian Linux Documentation Project).
В его переводе участвовали:

- *Александр Михайлов*

- *Андрей Добровольский*
- *Евгений Балдин*
- *Волков Сергей*

2 Проблема

Допустим у вас есть диск с более чем 1024 цилиндров. Допустим так же, что ваша операционная система использует старый интерфейс INT13 BIOS для операций ввода-вывода на диск. В этом случае у вас есть проблема, этот интерфейс использует поле из 10 бит для номера цилиндра на который будет выполняться В/В, следовательно цилиндры начиная с 1024 будут недоступны.
К счастью, Linux не использует BIOS, значит проблемы нет.
Все хорошо если бы не две вещи:
(1) При загрузке вашей системы, Linux еще не запущен и не может избавить вас от проблемы с BIOS. Это усложняет работу LILO и подобных менеджеров загрузки.
(2) Когда вы используете несколько операционных систем на одном диске они должны понимать таблицу разбиения диска на разделы. Другими словами, если вы используете Linux и DOS на одном диске, то обе системы должны интерпретировать таблицу разделов одинаково. Это влияет и на ядро Linux и на `fdisk`.
Ниже все это описывается более детально. Отмечу, что я использовал ядро версии 2.0.8. Для других версий могут быть небольшие отличия.

3 Основное

У вас есть новый большой диск. Что делать? Итак, в отношении программ: используйте `fdisk` (или лучше `cfdisk`) для создания разделов, теперь `mke2fs` для создания файловой системы и затем `mount` to для подключения новой файловой системы в основную.
В большинстве случаев нет нужды читать этот документ поскольку сегодня *нет* проблем с большими дисками. В основном, проблемы возникают у людей которые думают, что проблемы должны быть и устанавливают менеджер диска или запускают `fdisk` в режиме эксперта, или указывают для LILO или в командной строке ядра геометрию диска явно.
Однако,несколько проблем все же остаются, обычно они относятся к : (i) древним аппаратным средствам, (ii) несколько операционных систем на одном диске одновременно (iii) загрузка.
Примечание:
Для больших SCSI дисков: Linux поддерживает их уже очень давно. Не требуется никаких специальных действий.
Для больших IDE дисков (более 8.4 GB): получите последнее стабильное ядро(2.0.34 или позднее). Обычно, это решит все проблемы, особенно, если вы были достаточно мудры, чтобы не включать в BIOS трансляцию диска типа LBA и ей подобных.
Для очень больших дисков IDE (более 33.8 GB): смотри 13.1 (проблемы с 34+ GB IDE дисками) ниже.
Если у LILO проблемы при загрузке, попробуйте определить `6.1 (linear)` в файле конфигурации `/etc/lilo.conf`.
Возможны проблемы с геометрией диска которые могут быть решены заданием явной геометрии диска в kernel/LILO/fdisk.
Если у вас старый `fdisk` и он предупреждает о 7 (overlapping) разделов: игнорируйте предупреждение или используйте `cfdisk` чтобы убедиться, что все в порядке.
Если вы думаете, что что-то неправильно с размером вашего диска, убедитесь, что вы не путаете двоичные и десятичные 4 (единицы) и понимаете, что свободное пространство которое сообщает `df` при пустом диске, на несколько процентов меньше чем размер раздела, так как существует еще служебная область.
Теперь, если вы все еще думаете, что есть проблемы или просто из любопытства - читайте дальше.

.

4 Единицы и размерности

Килобайт (кБ) это 1000 байт. Мегабайт (МБ) это 1000 кБ. Гигабайт (ГБ) это 1000 МБ. Терабайт (ТБ) это 1000 ГБ. Приведенные выше значения определяются системой Си *Система Си*. Несмотря на это , многие люди считают что 1 МБ=1024000 байт. (например дискета на 1.44 МБ), а некоторые что в 1МБ - 1048576 байт.
Здесь находиться спецификация текущего стандарта на размерности информации.Этот новый стандарт определяет новые обозначения для бинарных размерностей (1 кБ = 2^10 байта(1024) 1 МБ = 2^10 байта(1024) и т.д.) .Они обозначаются как Ки , Ми , Ги и Ти. Например дискета размером 1440 КиБ (1.47 МБ , 1.41 МиБ)
Большинство производителей дисков следуют стандарту Си , и используют десятичные единицы. В то-же самое время некоторые программы Линукс используют обозначения МБ , ГБ , для обозначения двоичных единиц или смешанных десятично-двоичных единиц.Так что если ваш новый диск внезапно оказался меньше чем вы ожидали , подсчитайте его размер в десятичных единицах.
Обсуждая терминологию для обозначения двоичных единиц можно также упомянуть , *Knuth* который выступил с альтернативным *предложением*, использовать обозначения ККБ, ММБ, ГГБ и называть их большой килобайт , большой мегабайт и т.д. Впрочем на русском это звучит не так уж и красиво :)
Вообщем главное что необходимо запомнить - это то , что в 1 мегабайте 1000000 байт , и если вы имеете ввиду другое значение то оно должно обозначаться другим образом.

4.1 Размер Сектора

В данном документе за стандартный сектор принят сектор размеров 512 байт. На практике это почти всегда так, хотя некоторый магнитооптические диски используют размер сектора 2048 байт , и все емкости приведенные здесь для них надо умножить на 4. (При использовании `fdisk` с такими дисками необходимо обладать как минимум версией 2.9i и указывать опцию '-b 2048')

4.2 Размер Диска

Диск с C цилиндрами , H головками и S секторами на дорожку , будет иметь C*H*S секторов , и сможет хранить C*H*S*512 байт. Например , если на диске указано C/H/S=4092/16/63 то данный диск имеет 4092*16*63=4124736 секторов, и может содержать 4124736*512=2111864832 байт (2.11 ГБ). Существует негласное соглашение среди производителей давать обозначение C/H/S=16383/16/63 дискам более 8.4 ГБ размером , т.к. эти диски нельзя читать используя C/H/S значения.

5 Доступ к Диску

Перед тем, как прочитать или записать что-либо на диск, надо определить позицию с которой мы хотим начать читать или писать информацию. Позицию можно определить например путем передачи номера сектора или блока диска. Если рассматриваемый диск является SCSI диском, то номер сектора переводится в команды SCSI, а далее сам диск находит нужную позицию. Если диск имеет интерфейс IDE и использует LBA, то происходит почти то же самое. Если же диск нам достался с доисторических времен (RLL, MFM или IDE, когда не знали LBA), в этом случае диску для поиска нужной позиции требуется три значения: цилиндр, считывающая головка, сектор.
Между последовательной нумерацией и 3D нотацией(под выражением 3D нотация подразумевается поиск нужного сектора по трем параметрам c,h,s) имеется следующее соответствие: для диска с C цилиндрами, H считывающими головками и S секторами/дорожками - позиция (c,h,s) в 3D или CHS

нотации соответствует позиции c*H*S + h*S + (s-1) в последовательной или LBA нотации. (Минус 1, это следствие того, что по традиции номер сектора в 3D нотации начинается с 1, а не с 0.)
Следовательно, чтобы получить доступ к очень старому не SCSI диску, надо знать его *геометрию*.

5.1 Доступ к диску через BIOS и ограничение в 1024 цилиндра

Linux не использует при своей работе BIOS, но другие операционные системы могут использовать BIOS. Старые BIOS, которые были сделаны перед внедрением LBA, используют для доступа к диску функции INT13, которые используют 3D нотацию (c,h,s). (Точнее: `AH` - выбор операции, `CH` - младшие 8 бит номера цилиндра, `CL` - 7-6 биты соответствуют старшим битам номера цилиндра, 5-0 биты соответствуют номеру сектора, `DH` - номер считывающей головки и `DL` - номер диска (80h или 81h). Это объясняет схему построения таблицы разделов.)
В результате мы имеем номер в CHS нотации, укладывающийся в 3 байта: 10 бит на номер цилиндра, 8 бит на номер считывающей головки и 6 бит на номер сектора (от 1 до 63). Следствием этого является ограничение на количество цилиндров (от 0 до 1024), которое может поддерживать BIOS.
Когда появились IDE диски с поддержкой LBA, программное обеспечение для DOS и Windows не изменилось. И DOS, и Windows хотят знать геометрию диска, даже если она им не нужна для операция чтения/записи, им требуется это для обращения к BIOS. OS Linux требуется знание физической геометрии диска, если от нее требуется общаться с BIOS или с другими операционными системами, даже при работе с современными дисками.
Такое состояние дел продолжалось около 4ех лет, или около того, а затем на рынке появились диски, которые нельзя читать через функции INT13 (потому что $10+8+6=24$ бит, следовательно обращаться только к 8.5 ГБ - для большего не хватает адресного пространства). Был создан новый интерфейс для BIOS - так называемые расширенные INT13 (Extended INT13) функции, где DS:SI указывает на 16 байтовый Адресное Дисковое Пространство (Disk Address Packet), которое содержит 8-ми байтовый абсолютный номер блока.
Очень медленно Microsoft ползет в сторону использования расширенных INT13 функций. Вероятно, через несколько лет никому больше не понадобятся знания о физической геометрии диска.

5.2 История ограничений для BIOS и IDE

Спецификация (для IDE дисков) - ограничение в 137 ГБ

Не более чем 65536 цилиндров (нумерация 0-65535), 16 считывающих головок (нумерация 0-15), 255 секторов/дорожек (нумерация 1-255). Максимальная емкость: 267386880 секторов по 512 байт каждый, что составляет 136902082560 байт (137 ГБ). Сейчас (1999 год) это не является проблемой, но через несколько лет это станет проблемой.

BIOS Int 13 - ограничение в 8.5 ГБ

Не более чем 1024 цилиндров (нумерация 0-1024), 256 считывающих головок (нумерация 0-255), 63 сектора/дорожек (нумерация 1-63). Максимальная емкость: 8455716864 байт (8.5 ГБ). Сегодня это является серьезным ограничением на размер диска. А именно, DOS не в состоянии использовать большие диски.

Ограничение в 528 МБ

Если при обращению к диску использовать одни и те же значения c,h,s при обращении через Int 13 функции BIOS и через функции IDE интерфейса, то оба эти ограничения складываются так, что никто не может использовать больше чем 1024 цилиндра, 16 считывающих головок и 63 сектора/дорожки, что составляет 528482304 байт (528 МБ). Это и есть всем известная проблема ограничения в 504 Мб для DOS в комбинации со старыми BIOS. Это стало проблемой примерно в 1993 году. Для обхода этой проблемы использовались всевозможные трюки: железо (LBA), firmware (BIOS 'translation'), программное обеспечение (disk managers). Концепция трансляции ('translation') была изобретена в 1994 году: BIOS использует одну

геометрию диска, когда общается с диском и другую (фальшивую) при общении с DOS, и 'переводит' одну нотацию в другую.

Ограничение в 2.1 ГБ (Апрель 1996)

В некоторых старых BIOS под число цилиндров в CMOS RAM резервируется только 12 бит. Следовательно число цилиндров не может быть больше 4096, что дает максимально доступную емкость диска в 4095*16*63*512=2113413120 байт. Если диск имел большую емкость, то в процессе загрузки это приводило к 'зависанию' компьютера. Это привело к росту популярности дисков обладающих 4092/16/63 геометрией. До сих пор попадается довольно много больших дисков, которые с помощью переключателей можно заставить 'прикинуться' диском с подобной геометрией. Более подробную информацию можно найти по адресу *over2gb.htm*. *Более новые BIOS* в процессе загрузки не 'завешивают' компьютер, но видят диск меньшего размера. Например определяют, что диск размером в 2.5 ГБ имеет размер в 429 МБ.

Ограничение в 3.2 ГБ

В BIOS от Phoenix версий 4.03 и 4.04 был баг, который приводил к перезагрузке системы в случае наличии в компьютера диска с емкостью более 3277 МБ. Подробности ищите по адресу *over3gb.htm*.

Ограничение в 4.2 ГБ (Февраль 1997)

Простая BIOS трансляция (ECHS=Extended (расширенный) CHS, иногда зовется как 'Large disk support' или просто как 'Large') работает путем увеличения числа считывающих головок и уменьшения числа цилиндров, выдаваемых для использования DOS, до тех пор, пока число цилиндров не сравняется с 1024. На данный момент и DOS, и Windows 95 не поддерживает 256 считывающих головок. Обычно, когда диск сообщает о 16 считывающих головках, простой механизм трансляции позволяет получить доступ к диска с емкостью не более чем 8192*16*63*512=4227858432 байт (с фальшивой геометрией диска в 1024 цилиндров, 128 считывающих головок, 63 сектора/дорожки). Обратите внимание, что ECHS не меняет число секторов на дорожке, поэтому, если число секторов меньше 63, то ограничение становится еще более жестким. Для получения более подробной информации смотрите *over4gb.htm*.

Ограничение в 7.9 ГБ

Более продвинутые BIOS обходят, описанную выше проблему, путем приравнивание числа считывающих головок 15 ('Улучшеный ECHS'), поэтому фальшивая геометрия может состоять из 240 считывающих головок, что дает 1024*240*63*512=7927234560 байт.

Ограничение в 8.4 ГБ

Ну и наконец: если BIOS делает все возможное и невозможное для увеличения емкости диска - использует 256 считывающих головок и 63 сектора/дорожки ('assisted LBA' или просто 'LBA'), то это дает 1024*255*63*512=8422686720 байт, немного меньше чем 8.5 ГБ, так как геометрии с 256 считывающими головками следует избегать.

Ограничение в 33.8 ГБ (Август 1999)

Следующий барьер на пути повышение емкости приходится на 33.8 ГБ. Дело в том, что при 16 считывающих головках и 63 секторах/дорожках требуется число цилиндров большее, нежели 65535 - не вмещается в short (2 байта). Большинство BIOS на сегодня не в состоянии поддерживать подобные диски. (Для решения этой проблемы смотрите, например *Asus upgrades* для поиска новых прошивок для флэш памяти без этой проблемы.) Для решения этой проблемы ядру Linux версий ниже 2.2.14 / 2.3.21 требуется патч. Подробности смотрите в разделе 13.1 (Проблемы IDE с дисками емкостью 34+ ГБ).

Смотрите так же *Препятствия и ограничения*, и более детальное описание в статье *Ограничения для жестких дисков IDE*.

Жесткие диски емкостью более чем 8.4 ГБ сообщают, что их геометрия равна 16383/16/63. Это означает, что понятие геометрии изжило себя и что полный размер диска не может быть подсчитан через геометрию.

6 Загрузка

Когда система загружается , BIOS считывает сектор 0 ,более известный под названием MBR - Master Boot Record(Главная Загрузочная Запись) с первого диска (или флопи-дисковода или CD-ROM), и передает управление на код содержащийся в нём - обычно это программа загрузчик. Обычно такие мини-программы не имеют собственного драйвера диска и используют соответствующие сервисы , предоставляемые BIOS. Это означает , что ядро Линукс , может быть загружено только если оно помещается в пределах первых 1024 цилиндров.
Это проблемы решается весьма просто , достаточно разместить ядро (и сопутствующие файлы,используемые в процессе загрузки , такие как map файлы LILO), на разделе расположенном целиком в пределах первых 1024 цилиндров , к которым может получить доступ BIOS. Обычно это первый или второй раздел диска.
Итак : создайте небольшой раздел, скажем 10 МБ размером , так чтобы было достаточно места для всех ядер , убедитесь , что он целиком расположен в пределах первых 1024 цилиндров первого , либо второго диска. Смонтируйте его как `/boot` чтобы LILO держало там свои файлы.

6.1 LILO и опция 'linear' o

Другой аспект проблемы - это то что начальный загрузчик и BIOS должны иметь одинаковое представление о геометрии диска. LILO опрашивает ядро на предмет геометрии , но все больше авторов драйверов дисков , следуют дурной привычке - брать сведения о геометрии из таблицы разделов , вместо того , чтобы указать LILO , какую геометрию будет использовать BIOS. Поэтому , сведения о геометрии , предоставленные ядром , бесполезны. В таких случаях может помочь установка опции '`linear`'. Данная опция включает специальный режим LILO , в котором ему не требуется информация о геометрии диска во время установки загрузчика (оно хранит линейные адреса в maps файлах) , но производит преобразование линейных адресов во время загрузки. Существует только одна проблема связанная с использованием данной опции - LILO больше не знает о номерах цилиндров и не сможет предупредить вас , если часть ядра перекроет лимит в 1024 цилиндра , в итоге вы можете получить не загружаемую систему.

6.2 1024 цилиндра это не 1024 цилиндра

Тим Виллиамс пишет : 'Мой Линукс раздел располагался в пределах 1024 цилиндров и все равно не хочет работать.Сначала когда раздел располагался в пределах 1 Гб все работало нормально. Как такое может быть ? Это был SCSI диск с контроллером AHA2940UW, который имел следующие параметры : H=64, S=32 (каждый цилиндр 1 МиБ = 1.05 МБ) ,либо H=255, S=63 (цилиндры по 8.2 МБ), в зависимости от установок в BIOS и фирменном ПО. Без сомнения BIOS полагал ,что 1024 цилиндровый лимит - это 1 ГиБ (он использовал первые значения) , в то время как Линукс и LILO использовали последнии и LILO полагало что этот лимит находиться на 8.4 ГБ

7 Геометрия диска , разделы и 'overlap'

Если на вашем диске установлено несколько операционных систем, то каждая из них использует один или более разделов. Рассогласования между ними по поводу границ разделов могут привести к катастрофическим последствиям.
MBR содержит *таблицу разделов* описывающую расположение основных разделов. Данная таблица включает 4 описания , для 4х основных разделов , каждое выглядит таким образом

```
struct partition {
        char active;    /* 0x80: загрузочный, 0: не загрузочный */
        char begin[3];  /* CHS для первого сектора */
        char type;
        char end[3];    /* CHS для последнего сектора */
        int start;      /* 32 битный номер сектора (считая с 0) */
        int length;     /* 32 битное число , количество секторов */
};
```

(Где CHS означает Cylinder/Head/Sector).
Данная информация несколько избыточна : положение раздела на диске определяется как 24-битными полями `begin` и `end` , и 32 битными полями `start` и `length` .
Линукс использует только поля `start` и `length` , и поддерживает партиции содержащие не более 232 секторов , т.е. размер раздела ограничен 2 ТиБ. Это в 100 раз больше чем любой доступный сегодня диск , и вероятно это лимита хватит на следующие 5-8 лет. (Хотя разделы и могут быть очень больших размеров, но в тоже время нужно помнить что максимальный размер файла в ext2 системе 2 ГиБ)
DOS использует поля `begin` и `end` и использует сервисы BIOS (Int 13h) для доступа к диску , и поэтому не может использовать диски более 8.4 ГБ , даже с новым (преобразующим BIOS). (Разделы не могут быть более 2.1 ГБ , из за ограничений файловой системы FAT16).
Windows 95 поддерживает расширенный INT13 интерфейс ,и использует специальные типа разделов (c,e,f вместо b,6,5) чтобы обозначить что к партиции нужно обращаться таким образом.При использовании этих типов разделов , `begin` и `end` поля содержат "пустую"информацию , (1023/255/63). Windows 95 OSR2 поддерживает систему FAT32 (типы разделов b и c),которая позволяет использовать разделы почти 2 ТиБ размером.
Сообщения о перекрывании ('ovelapping') выдаваемые `fdisk` обычно обусловлены тем , что он воспринимает как не корректные поля `begin` и `end` для партиций большего размера. Проблема не поддается корректировке , т.к. разделы всегда будут восприниматься как перекрывающиеся , на дисках с более чем 1024 цилиндрами. Просто игнорируйте подобные сообщения , т.к. поля `start` и `length` все равно содержат корректную информацию. Следует быть осторожным при использовании fdisk , когда диск используется совместно с DOS, и использовать следующий формат вызова команд `cfdisk -Ps /dev/hdx` и `cfdisk -Pt /dev/hdx` для просмотра таблицы разделов диска `/dev/hdx`.

8 Преобразование и программы-загрузчики.

Геометрия диск (с головками,цилиндрами и дорожками) это анархаизмы оставшиеся в наследство от времен MFM и RLL. В те дни эти величины отражали физические параметра диска. Сегодня с IDE и SCSI, никому не интересна настоящая , физическая геометрия диска. По настоящему, количество секторов на дорожку переменная величина, чем ближе к внешнему кругу диска , тем она больше, поэтому величина секторов/дорожка всегда не настоящая. В тоже время команда IDE - INITIALIZE DRIVE PARAMETERS (91h) служит для того чтобы сказать диску , сколько головок и секторов/ дорожку он должен иметь. Для современного диска совершенно нормальна ситуация когда в действительности он имеет 2 головки , сообщает BIOS , что их 15-16, а BIOS убеждает программы, что на самом деле головок 255.
Для пользователя удобнее всего представлять диск , как линейный массив секторов , пронумерованных по порядку, а задачу нахождения того , где расположен искомый сектор - оставить фирменному ПО. Такая линейная нумерация получила название LBA.
Таким образом на сегодняшний день концептуальная картина такова, DOS, или менеджер загрузки, сообщают BIOS значения c,h,s. BIOS преобразует эти значения в LBA , используя фальшивую геометрию диска ,которую использует пользователь. Если диск воспринимает LBA то эта величина используется для чтения/записи , иначе она преобразуется назад в c/h/s и используется.
Заметим что при использовании термина LBA могут возникнуть некоторые неточности, т.к. термин LBA - в оригинале означает 'Linear Block Addressing' (в противоположность CHS адресации). В BIOS

Setup он означает схему преобразования , также иногда называемую 'поддержка LBA'.См. '**??** ()'
Иногда тоже самое срабатывает , когда фирменная прошивка не поддерживает LBA , но BIOS знает о преобразовании. (В SETUP это часто обозначается как 'Large').Теперь BIOS будет предоставлять геометрию C/H/S операционной системе, и использовать C',H',S' при общении с контроллером диска. Обычно S=S',C=C'/N и H=H'*N,где N это наименьшая степень двойки, которая приблизит C' к 1024 (так чтобы были минимальные потери при переворачивание выражения C'=C/N). Это позволяет получить доступ к 7.8 ГиБ.
Третья опция в Setup - это обычно 'Normal', означающая что трансляция не производиться.
Если ваш BIOS не поддерживает режимы 'Large' или 'LBA',то существует несколько софтверных решений проблемы. Менеджеры Дисков,такие как OnTrack или EZ-Drive замещают процедуры BIOS для работы с дисками, своими собственными. Часто такие программы размещают свой код в MBR и последующих секторах (OnTrack называет этот код DDO: Dynamic Drive Overlay - Динамическое Замещение Диска),так что он загружается раньше любой операционной системы. Правда такой подход может создать проблемы при попытки загрузки с дискеты.
Достигаемый эффект примерно тот-же что и при преобразующим BIOS'ом. Отрицательная сторона использования менеджеров дисков - сложности возникающие при совмещении нескольких операционных систем на одном диске.
Линукс поддерживает OnTrack Disk Manager с версии 1.3.14, и EZ-Drive с версии 1.3.29. Более детально это описано далее.

9 Преобразования выполняемые ядром, для IDE дисков.

Если ядро Линукс обнаруживает наличие какого либо менеджера диска на IDE диске,он попытается разметить диск так же как и программа менеджер. Например OnTrack или EZ-Drive. В тоже время если геометрия была задана в явном виде в командной строке, то используется именно она. Например строка : '`hd=`*cyls*`,`*heads*`,`*secs*' может быстро избавить вас от совместимости с менеджером диска.
Переразметка производиться путем подбора,(последовательно пробуеться 4,8,16,32,64,128,25 головок, с H*C константой) пока C <= 1024 или H = 255.
Далее в тексте приводятся детали в следующем порядке, заголовки подсекций - это фразы появляющиеся в соответствующих загрузочных сообщениях. Здесь и во всем остальном документе,типы разделов указываются в смешанном - двоично-десятичном виде.

9.1 EZD

Наличие EZ-Drive определяется по типу первого основного раздела. Если EZ-Drive установлен,то тип будет равен 55. Геометрия переразмечаеться как описано выше,и таблица разделов в секторе 0 игнорируется,вместо неё считывается таблица разделов из сектора 1. Номера блоков диска не изменяются но операции записи обращенные к сектору 0 перенаправляются на сектор 1. Такое поведение можно изменить перекомпилировав ядро с : `#define FAKE_FDISK_FOR_EZDRIVE 0` в `ide.c`.

9.2 DM6:DDO

Менеджер диска OnTrack (на первом диске) определяется по типу первого главного раздела - 54. Геометрия переразмечаеться как описано выше и диск "сдвигается"на 63 сектора (так что 63 сектор становиться сектором 0). После этого новый MBR (с таблицей разделов) считывается из нового сектора 0. Сдвиг производиться только для того чтобы высвободить место для DD0, поэтому на других дисках сдвига нет.

9.3 DM6:AUX

Наличие менеджера диска OnTrack (на других дисках) определяется по типу первого главного раздела, он имеет тип 51 или 53. Геометрия перераспределяется как описано выше.

9.4 DM6:MBR

Наличие старых версий OnTrack Diskmanager определяется не по типу раздела,а о по специальной подписи.(Проверьте что смещение записанное в 2 и 3м байтах MBR не превышает 430 и слово найденное по этому смещению эквивалентно 0x55AA + следующий байт четный). Геометрия опять-же разпределяеться как описано выше.

9.5 PTBL

Также существует тест с помощью которого можно вычислить производиться ли преобразования анализируя значения `start` и `end` основных разделов: Если некий раздел имеет значения начального и конечного секторов 1 и 63 и 31,63,127 или 254 головки,тогда,т.к. обычно разделы оканчиваются на границе цилиндра, а интерфейс IDE использует не более 16 головок, то вероятно задействовано BIOS преобразование ,и геометрия преразмечена для использования 32,64,128,255 головок. Переразметка геометрии не производиться,если текущая геометрия уже имеет 63 сектора на дорожку и соответственное число головок. (обычно это означает,что переразметка уже была проведена)

10 Выводы

Что значит все выше сказанное? Для пользователей Линукс только одну вещь: необходимость быть уверенными в том, что LILO и fdisk используют правильную геометрию. Правильная геометрия для fdisk - та, которая используется ОС находящейся на том же диске, а для LILO - та которая позволяет успешно взаимодействовать с BIOS во время загрузки. (обычно эти две геометрии совпадают)
Как `fdisk` узнает о геометрии? Он спрашивает ядро,используя `HDIO_GETGEO` ioctl. Но пользователь может заставить `fdisk` использовать другую геометрию в интерактивном режиме или в командной строке.
Как LILO узнает о геометрии? Оно запрашивает ядро, используя `HDIO_GETGEO` ioctl. Но пользователь также может обойти эту геометрию, указав специальный параметр '`disk=`' в файле `/etc/lilo.conf` (см. lilo.conf(5)). Возможно также задание опции `linear` , которая заставит LILO хранить в его map файле LBA адреса, вместо CHS и будет определять геометрию при загрузке с помощью INT13(функция 8).
Как ядро узнает о геометрии? Во первых, возможно,что пользователь указал детальную геометрию в командной строке ядра '`hda=`*cyls*`,`*heads*`,`*secs*' (см. bootparam(7)),вручную или указав загрузчику необходимость передать эти параметры ядру. Например,для LILO нужно добавить строку '`append = "hda=`*cyls*`,`*heads*`,`*secs*`"`' в `/etc/lilo.conf` (см. lilo.conf(5)). Или же ядро может само догадаться о геометрии, опросив BIOS или "железо".
Начиная с версии ядра 2.1.79 возможно указание геометрии ядру с помощью файловой системы `/proc`. Например:

```
# sfdisk -g /dev/hdc
/dev/hdc: 4441 cylinders, 255 heads, 63 sectors/track
# cd /proc/ide/ide1/hdc
# echo bios_cyl:17418 bios_head:128 bios_sect:32 > settings
# sfdisk -g /dev/hdc
/dev/hdc: 17418 cylinders, 128 heads, 32 sectors/track
#
```

10.1 Вычисление параметров LILO.

Иногда бывает удобно указать какую либо конкретную геометрию указав 'hda=*cyls*,*heads*,*secs*' в командной строке ядра. Почти всегда необходимо указывать *secs*=63, для того чтобы указать *heads*. (Наиболее удачными значениями на сегодняшний день будут *heads*=16 и *heads*=255.) Что необходимо указать для *cyls*? Это должно быть число соответствующее полной емкости диска в секторах C*H*S. Например для диска с 71346240 секторами (36529274880 байт) C должно быть вычислено как 71346240/(255*63)=4441 (посчитать это можно например с помощью bc) , и заданы следующие параметры hdc=4441,255,63. Как можно узнать полную емкость диска? Например ,

```
# hdparm -g /dev/hdc | grep sectors
 geometry     = 4441/255/63, sectors = 71346240, start = 0
# hdparm -i /dev/hdc | grep LBAsects
 CurCHS=16383/16/63, CurSects=16514064, LBA=yes, LBAsects=71346240
```

дает два способа нахождения полного числа секторов - 71346240. Нахождение путем просмотра сообщений выдаваемых ядром при загрузке

```
# dmesg | grep hdc
...
hdc: Maxtor 93652U8, 34837MB w/2048kB Cache, CHS=70780/16/63
 hdc: [PTBL] [4441/255/63] hdc1 hdc2 hdc3! hdc4 < hdc5 > ...
```

говорит нам о (как минимум) 34837*2048=71346176 секторах и соотвественно о 70780*16*63=71346240 секторах. В данном случае вторая величина корректна, но в общем то обе могут быть округлены в меньшую сторону. Это хороший способ получить представление о размере диска когда hdparm не доступен. Но некогда не задавайте слишком большое значение для *cyls*! В случае с SCSI дисками, точное число секторов выдается ядром при загрузке.

```
SCSI device sda: hdwr sector= 512 bytes. Sectors= 17755792 [8669 MB] [8.7 GB]
```

(и MB, GB округляются, но никогда в меньшую сторону, и являются 'двоичными').

11 Детали

11.1 Детали IDE - семь геометрий

Драйвер IDE имеет пять источников информации о геометрии. Первый (G_user) указывается пользователем в командной строке. Второй (G_bios) - это таблица с фиксированными параметрами диска в BIOS (только для первого и второго дисков),которая считывается при запуске, до переключения в 32битный режим. Третий (G_phys) и четвертый (G_log) - возвращаются IDE контроллером, в ответ на команду IDENTIFY,это 'физическая' и 'текущая логическая' геометрии.
С другой стороны драйверу нужны две величины для определения геометрии - это G_fdisk,возвращаемая HDIO_GETGEO ioctl, и G_used,которая непосредственно используется при операциях ввода вывода.Оба этих параметра инициализируются для G_user если таковой указан,или для G_bios,когда информация предоставлена в CMOS, и G_phys в любом другом случае. Если использование G_log выглядит разумным,то G_used устанавливается для него. В случае же если G_phys выглядит оптимальным, то G_used устанавливается для G_phys. Примечание: термин 'выглядит оптимальным,разумным' - означает, что число головок лежит в пределах 1-16.
Говоря проще : параметры переданные в командной строке имеют больший приоритет, чем параметры полученные от BIOS,и будут определять геометрию которую видит fdisk,но если будет указана преобразованная геометрия (более 16 головок),то при операциях ввода/вывода она будет замещена геометрией полученной от команды IDENTIFY.
Заметит что G_bios достаточно ненадежен: для систем загружающихся с SCSI первый и второй диски, могут быть SCSI дисками,и геометрия которую BIOS сообщает для sda будет использована ядром для hda. Более того,диски которые не упомянуты в BIOS Setup,не будут видны BIOS. Это означает, что например на IDE системе,где hdb не упомянут в Setup, геометрия сообщаемая BIOS для первого и второго дисков, будет геометрий hda и hdc.

11.2 SCSI детали

Ситуация с SCSI немного отличается,т.к. все SCSI команды используют логические номера блоков(секторов),поэтому геометрия не зависит от физических процессов ввода/вывода. В тоже время формат таблицы разделов такой же что и для IDE, поэтому `fdisk` 'у приходиться изобретать некую геометрию,он также использует `HDIO_GETGEO`. Несмотря на это `fdisk` различает SCSI и IDE диски. Как можно видеть из описанного выше, различные драйвера изобретают различную геометрию. Что вызывает некий беспорядок и неорганизованность.
Если вы не используете DOS или ему подобные системы, то надо опасаться любых установок, связанных с расширенным преобразованием,просто используйте 64 головки,32 сектора на дорожку(для того чтобы у вас на 1 цилиндр приходился 1 МиБ),если это возможно,так чтобы при перестановки диска с одного контроллера на другой не возникало проблем. Некоторые (aha152x, pas16, ppa, qlogicfas, qlogicisp) драйвера SCSI дисков, так обеспокоены совместимостью с DOS ,что они не позволят Линукс использовать более 8 ГиБ. Это баг.
Что такое настоящая геометрия? Самый простой ответ - таковой не существует. Даже если таковая и была бы , вам не нужно бы было её знать и не в коем случае не рассказывать о ней `fdisk` или LILO или ядру. Это дело только SCSI контроллера и диска.
Но если вам это жизненно необходимо, вы можете спросить сам диск. Существует очень важная команда READ CAPACITY, которая сообщает полный размер диска, и команда MODE SENSE, которая на странице Rigid Disk Drive Geometry (страница 04) сообщает количество цилиндров и головок (т.е. параметры которые нельзя изменить), на странице Format (страница 03) дает количество байт в секторе и количество секторов на дорожку. Последнее число обычно зависит от notch, а количество секторов на дорожку обычно переменное - внешние треки имеют больше секторов,чем внутренние. Программа `scsiinfo` сообщает подобную информацию. Существует огромное количество деталей,которые никто,включая операционную систему,знать не хочет. Более того , т.к. мы концентрируемся на `fdisk` и LILO, то обычно получаемые величины вроде C/H/S=4476/27/171 - значения которые не могут быть использованы `fdisk`, т.к. в таблице разделов эти значения не помещаются в отведенные им поля.
А откуда-же берет данную информацию `HDIO_GETGEO`? Оно опрашивает SCSI контроллер,или делает сложный запрос. Некоторые драйвера думают что мы хотим знать настоящие значения, хотя на самом деле нам нужно знать лишь то, что будет использовать Fdisk в DOS или OS/2.
Заметит что `fdisk` для Линукс требуются H и S - количество головок и секторов на дорожку, для преобразования LBA номеров в c/h/s адреса,но значение C (количество цилиндров) не играет роли в данном преобразовании. Некоторые драйвера используют значения C/H/S = 1023/255/63 чтобы сигнализировать, что емкость диска как минимум 1023*255*63 секторов.Это плохо,т.к. не соотвествует реальным размерам, и ограничит пространство доступное пользователям большинства версий `fdisk` 8'ю ГиБ'ами.Это наиболее серьезная проблема сегодня.
В описании данном выше, M обозначает общую емкость диска, и C/H/S количество цилиндров,головок и секторов/дорожка. Задание H,S необходимо если мы расчитываем C как M/(H*S).
По умолчанию, H=64, S=32.

aha1740, dtc, g_NCR5380, t128, wd7000:

H=64, S=32.

aha152x, pas16, ppa, qlogicfas, qlogicisp:

H=64, S=32 если C не > 1024, в таком случае H=255, S=63, C = min(1023, M/(H*S)). (Поэтому C урезается, и H*S*C не является усредненным значение емкости диска. Это вызывает недоумение у большинства версий `fdisk`.) Код `ppa.c` использует M+1 вместо M и сообщает что из за бага в `sd.c` M уменьшено на единицу.

advansys:

H=64, S=32 если C не > 1024 и '> 1 GB' опция в BIOS включена, в этом случае H=255, S=63.

aha1542:

Спрашивает контроллер какая из двух возможных схем преобразования используется,и использует либо H=255, S=63 либо H=64, S=32. В общем случае появляется следующее сообщение при загрузке. "aha1542.c: Использует расширенное преобразование BIOS".

aic7xxx:

H=64, S=32 если C не > 1024, и загрузочный параметр "extended"либо бит "extended"был установлен в SEEPROM или BIOS, в это случае H=255, S=63. В Линукс 2.0.36 расширенное преобразование всегда устанавливалось ,если не было найдено SEEPROM,но в Линукс 2.2.6 если SEEPROM не найден то расширенное преобразование устанавливается только если пользователь задал соответствующий загрузочный параметр. (если SEEPROM найден то загрузочный параметр игнорируется). Это значит, что setup,который работает с 2.0.36 может перестать загружаться с 2.2.6(и требовать указания параметра 'linear' для LILO,или параметра ядра 'aic7xxx=extended')

buslogic:

H=64, S=32 если C не >= 1024, и дополнительно,расширенное преобразование не было включено на контроллере ,в этом случае: если M < 222 то H=128, S=32; иначе H=255, S=63. Однако после выбора значений для C/H/S,считывается таблица разделов, и если для одной из трех возможностей (H,S) = (64,32), (128,32), (255,63) или значение H=H-1 найдено где либо,то используется пара (H,S),и выдается загрузочное сообщение "Adopting Geometry from Partition Table".

fdomain:

Находит информацию о геометрии в таблице параметров диска BIOS, или считывает таблицу разделов и использует H=endH+1, S=endS для первого раздела (если такая информация дана), или использует H=64, S=32 для M < 221 (1 ГиБ), H=128, S=63 для M < 63*217 (3.9 ГиБ) и H=255, S=63 иначе.

in2000:

Использует первое из (H,S) = (64,32), (64,63), (128,63), (255,63) которое сделает C равным < = 1024. В последнем случае, усекает C до 1023.

seagate:

Считывает C,H,S с дичка. (Кошмар!) Если C или S слишком велики, то устанавливает S=17, H=2 и удваивает H до тех пор пока C не станет <= 1024. Это означает,что H будет установлено в 0 если M > 128*1024*17 (1.1 ГиБ). Это баг.

ultrastor и u14_34f:

Одна из трех разметок ((H,S) = (16,63), (64,32), (64,63)) используется в зависимости от режима разметки контроллера.

Если драйвер не указывает геометрию, то мы возвращаемся назад к "отгадыванию"используя таблицу разделов, или используя полную емкость диска.
Взглянем на таблицу разделов. Т.к. обычно разделы оканчиваются на границе цилиндров, мы можем задать `end = (endC,endH,endS)` для любого раздела , просто укажите H = `endH+1` и S = `endS`. (обратно сектора отсчитываются от 1.) Если быть более точными , делается следующее. Если есть не пустой раздел , выбирается раздел с наибольшим `beginC`. Для этого раздела , смотрим на `end+1`, вычисляемое путем добавления `start` и `length` и предполагая что раздел заканчивается на границе цилиндров. Если оба значения совпадают, или если `endC` = 1023 и `start+length` целое произведение `(endH+1)*endS`, то предполагается что данный раздел расположен на границе цилиндров. , и используется H = `endH+1` и S = `endS`. Если это не срабатывает, потому - что нет разделов , или потому - что они имеют странные размеры, то используется только M - емкость

диска. Алгоритм: возьмем H = M/(62*1024) (округленное вверх), S = M/(1024*H) (округленное вверх), C = M/(H*S) (округленное вниз). Это имеет эффект умножения (C,H,S) с C максимум 1024 и S максимум 62.

12 IDE лимит в 8 ГиБ

Драйвер IDE Линукс берет информацию о геометрии и емкости диска (и множество другой информации) используя ATA IDENTIFY запрос. До недавнего времени драйвер не верил возвращаемому значению lba_capacity если оно было более чем на 10% больше чем емкость вычисленная как C*H*S. Хотя,по соглашению производителей,большие IDE диски (с более чем 16514064 секторами) возвращают C=16383, H=16, S=63, что соотвествует 16514064 секторам (7.8 ГБ) независимо от их реальных значений, но lba_capacity соотвествует настоящему размеру.
Последние версии ядер Линукс (2.0.34, 2.1.90) знают о этом и действуют правильно. Если у вас более старое ядро и вы не хотите производить апгрейд, то вы можете попытаться изменить процедуру `lba_capacity_is_ok` в `/usr/src/linux/drivers/block/ide.c` приведя её примерно к такому виду:

```
static int lba_capacity_is_ok (struct hd_driveid *id) {
        id->cyls = id->lba_capacity / (id->heads * id->sectors);
        return 1;
}
```

См. 2.1.90. (более хороший патч)

12.1 Сложности BIOS

Как было только что замечено большие диски возвращают геометрию: C=16383, H=16, S=63 в независимости от реального размера.(настоящее значение указывается в lba_capacity). Некоторые BIOS не знают о этом и преобразуют 16383/16/63 в нечто с меньшим числом цилиндров и большим головок, например 1024/255/63 или 1027/255/63. Поэтому правильно распознавать 16383/16/63 геометрию должно не только ядро , но и BIOS. Начиная с версии 2.2.2 это делается правильно (путем вычисления C = capacity/(H*S) на основе H и S полученных от BIOS). Обычно эта проблема решается путем установки типа диска Normal в BIOS setup (или еще лучше None (не всегда доступно)) Если это не возможно, из за необходимости загружаться с этого диска или использования его совместно с DOS/Windows, а апгрейд ядра до версии 2.2.2 невозможен , то нужно использовать загрузочные параметры ядра.
Если BIOS сообщает 16320/16/63, то это обычно делается в надежде получить 1024/255/63 после преобразования.

12.2 Джамперы указывающие количество головок.

Многие диска имеют перемычки которые позволяют выбирать между 15 и 16 головками. Установки по умолчанию дают вам 16 - головочную геометрию. Иногда обе версии геометрии адресуют одинаковое количество секторов,иногда 15 головок уменьшают количество адресуемых секторов. Для такой настройки могут быть обьективные причины: : Petri Kaukasoina пишет: 'Диск на 10.1 ГиБ - IBM Deskstar 16 GP (model IBM-DTTA-351010) был настроен с помощью перемычек для использования 16 головок по умолчанию, но мой старый компьютер (с AMI BIOS) не загружался и я был вынужден переключить его на использование 15 головок. hdparm -i говорит RawCHS=16383/15/63 и LBAsects=19807200. Я использую 20960/15/63, чтобы получить полную емкость.' Если вы хотите подробнее узнать о настройке с помощью джамперов, то вам вероятно будет полезен следующий адрес: *http://www.storage.ibm.com/techsup/hddtech/hddtech.htm*.

12.3 Джамперы которые урезают полную емкость.

Многие диски имеют джамперы,которые заставляют диск выглядеть меньше, чем он есть на самом деле. Довольно глупая опция,не правда ли,врядли кто либо из пользователей Линукс захочет ей воспользоваться. Некоторые BIOS зависают при наличии больших дисков. Единственное решение в таком случае - сделать диск невидимым для BIOS.Но это возможно лишь в том случае если данный диск не является загрузочным в вашей системе.
Первым серьезным лимитом был лимит в 4096 цилиндра limit (16 головок и 63 сектора/дорожка, 2.11 ГБ). Например, Fujitsu MPB3032ATU 3.24 GB диск имеет геометрию по умолчанию 6704/15/63, но может быть установлен режим (перемычками) 4092/16/63, и выдается LBAcapacity 4124736 сектора, так чтобы операционная система не могла догадаться что в реальности диск больше. В таком случае (с BIOS который "падает"услышав о том какого размера диск по настоящему, так что необходима установка джампера) необходима установка загрузочных параметров ядра, чтобы указать Линукс истинный размер диска.
Это неудачное решение. Большинство дисков могут быть настроены так, чтобы выглядеть как 2 Гб диск, они сообщают урезанную геометрию, но возвращают полную LBAcapacity. Такие диски будут работать под Линукс правильно(с полной емкостью) независимо от установок джамперов.
Более "свежий"лимит это 13.1 (33.8 Гб лимит). Ядра Линукс ниже 2.3.21 требуют патча , чтобы работать с дисками такого обьема. Некоторые диски больше данного обьема, могут быть настроены, чтобы выглядеть как 33.8 Гб диск.
Например,диск IBM Deskstar 37.5 GB (DPTA-353750) может быть настроен так, чтобы казаться диском емкости 33.8 Гб, и затем он сообщает геометрию 16383/16/63 как любой другой большой диск, но LBAcapacity 66055248 (соответствующая 65531/16/63, или 4111/255/63)). Такие диски при настройке для режима 33.8 Гб, требуют указания загрузочных параметров для работы с полным обьемом под Линукс. Также вы можете взглянуть сюда: *the BIOS 33.8 GB limit*.

13 Лимит в 65535 цилиндров

Функция `HDIO_GETGEO` ioctl возвращает количество цилиндров в поле длиной 16 бит. Это означает, что если у вас более 65535 цилиндров, то эта цифра будет урезана, и (для обычного случая настройки SCSI с 1 МиБ цилиндрами) 80 ГиБ диск будет выглядеть как диск на 16 ГиБ. Если вы распознали эту проблему, то вы можете легко от неё избавиться.

13.1 Проблемы IDE с 34+ ГБ-ми дисками

Диски больше чем 33.8 ГБ не будут работать с ядрами младше 2.3.21. Детали этой проблемы приведены ниже. Предположим, что вы приобрели новый диск - IBM-DPTA-373420, емкостью 66835440 сектора (34.2 ГБ). Ядра более поздних версий , чем 2.3.21, будут сообщать что размер диска равен 769*16*63 = 775152 секторов (0.4 ГБ), что, согласитесь не совсем приятно. Задание параметров в командной строке(например:hdc=4160,255,63) не поможет,они будут игнорироваться. Что происходит? Процедура idedisk_setup() получает сведения о геометрии от диска (16383/16/63) и записывает эти данные поверх данных полученных от пользователя в командной строке,поэтому данные пользователя используются только для BIOS геометрии. Процедура current_capacity() или idedisk_capacity() пересчитывает количество цилиндров как 66835440/(16*63)=66305, но , так как поле имеет длину всего 16 бит,это число превращается в 769. Т.к lba_capacity_is_ok() разрушила id->cyls, каждый следующий её вызов будет возвращать неверные данные, поэтому емкость диска получается равна 769*16*63. Для нескольких версий ядер доступен патч. Патч для 2.0.38 можно взять на *ftp.kernel.org*. Патч для 2.2.12 на : *www.uwsg.indiana.edu*. Ядра 2.2.14pre поддерживают такие диски. В серии ядер 2.3.* , поддержка таких дисков есть в версиях начиная с 2.3.21. Решить проблему можно также путем настройки диска12.3 (с помощью перемычек) на емкость 33.8 ГБ. Во многих случаях может помочь 5.2 (Обновление BIOS) ,это необходимо если вы хотите загружаться с диска.

14 Расширенные и логические разделы

Ниже можно видеть структуру MBR (сектор 0): код загрузчика, затем 4 входа таблицы разделов по 16 байт каждый, затем специальная "подпись"AA55. Элементы таблицы разделов с типом 5 или F или 85 (в шестнадцатиричной системе) имеют специальное значение: они описывают *extended* (расширенные) разделы : куски диска, которые разбиты на несколько *логических* разделов.Т.е. расширенные разделы сами по себе не используются они могут лишь хранить логические разделы. Важно только положение первого сектора расширенного раздела. Этот первый сектор хранит таблицу разделов с четырьмя входами: один используется для логического раздела, другой для еще одного расширенного раздела,а два не используются. Таким образом можно получить цепочку из таблиц разделов, где первая описывает три основных раздела,а каждая следующая один логический раздел и положение следующей таблицы.
Очень важно понимать это: Когда люди делают что то глупое во время разбиения диска, их волнует вопрос - сохранились ли их данные ? Обычно данные остаются на своем месте, исключения составляют сектора, которые перекрываются расширенными таблицами разделов - они теряются на всегда.
Программа sfdisk показывает всю цепочку:

```
# sfdisk -l -x /dev/hda

Disk /dev/hda: 16 heads, 63 sectors, 33483 cylinders
Units = cylinders of 516096 bytes, blocks of 1024 bytes, counting from 0

   Device Boot Start     End   #cyls   #blocks   Id  System
/dev/hda1          0+    101     102-    51376+  83  Linux
/dev/hda2        102    2133    2032   1024128   83  Linux
/dev/hda3       2134   33482   31349  15799896    5  Extended
/dev/hda4          0       -       0         0    0  Empty

/dev/hda5       2134+   6197    4064-  2048224+  83  Linux
    -           6198   10261    4064   2048256    5  Extended
    -           2134    2133       0         0    0  Empty
    -           2134    2133       0         0    0  Empty

/dev/hda6       6198+  10261    4064-  2048224+  83  Linux
    -          10262   16357    6096   3072384    5  Extended
    -           6198    6197       0         0    0  Empty
    -           6198    6197       0         0    0  Empty
...
/dev/hda10     30581+  33482    2902-  1462576+  83  Linux
    -          30581   30580       0         0    0  Empty
    -          30581   30580       0         0    0  Empty
    -          30581   30580       0         0    0  Empty

#
```

Во время разбиения диска могут быть созданы плохие таблицы разделов. Многие ядра зацикливаются если какой либо расширенный раздел указывает сам на себя,или на более ранний раздел в цепочке. Также возможно иметь ссылки на два расширенных раздела в одной таблице, т.е. она раздваивается.. (Это например случается когда fdisk не распознает каждую из 5, F, 85 как расширенный раздел, и создает 5 следом за F.) Не одна из стандартных fdisk програм не может справиться с такой ситуацией,вам потребуется очень много времени и сил чтобы пофиксить эту проблему. Ядро Линукс воспринимает такое раздвоение нормально. То есть вы можете иметь две цепи логических разделов. Иногда это полезно - например можно использовать тип 5 (виден DOS) и тип 85(невиден DOS),так что DOS fdisk не "упадет"из за логических разделов за пределами 1024 цилиндра

15 Решение проблемы

Многие люди думают что у них есть проблемы, хотя на самом деле их нет. Или они думают что проблемы связаны с геометрией диска, хотя на самом деле она не причем. Все описанное в этом документе выглядит достаточно сложным, хотя работа с геометрией диска обычно предельно проста,обычно не нужно ничего делать вообще либо задать LILO параметр linear, если загрузка идет лишь до LI. Внимательно изучайте сообщения выдаваемые при загрузке и помните чем больше вы изменяете геометрию(устанавливаете число головок и цилиндров и т.д.) тем меньше вероятность что все это будет правильно работать. Проще говоря чаще всего все бывает установлено правильно по умолчанию.
И помните Линукс не использует геометрию диска практически не где, поэтому проблемы возникающие во время работы не могут быть вызваны геометрией.Геометрия используется только LILO и fdisk. Если разные операционные системы не воспринимают таблицу разделов, то тогда это возможно проблема с геометрией. Во всех других случаях геометрия НЕ используется. Например если у вас проблемы с mount, то геометрия здесь не причем.

15.1 Проблема: Мой IDE диск получает неправильную геометрию, когда я загружаюсь с SCSI..

Такая ошибка вполне вероятна. Ядро Линукс опрашивает BIOS о hd0 и hd1 (Диски номер 80H и 81H) и применяет эту геометрию к hda и hdb. Но на системе, которая загружается с SCSI,первые два диска,вполне могут быть SCSI дисками, и поэтому пятый диск (первый IDE диск), получает геометрию sda. Эта проблема легко решается путем задания загрузочных параметров. 'hda=C,H,S' с необходимыми C, H и S, во время загрузки или в /etc/lilo.conf.

15.2 Не проблема: Идентичные диски имеют различную геометрию?

'Я имею два одинаковых диска 10 GB IBM . Но, fdisk показывает для них различные размеры.

```
# fdisk -l /dev/hdb
Disk /dev/hdb: 255 heads, 63 sectors, 1232 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot  Start      End   Blocks   Id  System
/dev/hdb1           1     1232  9896008+  83  Linux native
# fdisk -l /dev/hdd
Disk /dev/hdd: 16 heads, 63 sectors, 19650 cylinders
Units = cylinders of 1008 * 512 bytes

   Device Boot  Start      End   Blocks   Id  System
/dev/hdd1           1    19650  9903568+  83  Linux native
```

Как такое может быть?'
В чем же дело ? Первый из всех этих дисков действительно 10 Гиг размером: hdb имеет размер 255*63*1232*512 = 10133544960, а hdd имеет размер 16*63*19650*512 = 10141286400, т.е. все правильно , и ядро видит оба диска как 10.1 ГБ. Почему же есть разница в размерах ? Ядро получает информацию о первых двух дисках у BIOS, а BIOS переразметил hdb на 255 головок (и 16*19650/255=1232 цилиндров). Округление этой цифры в меньшую сторону стоит вам около 8 МБ.
Если вы захотите переразместить hdd таким же образом, задайте такие загрузочные параметры :'hdd=1232,255,63'.

15.3 Не проблема: fdisk видит гораздо больше места,чем df?

fdisk показывает количество блоков на диске. Если же у вас на диске создана файловая система, то она будет занимать какое то место на диске,обычно около 4%, размера файловой системы.Или

даже больше если при создании вы запросили много inodes. Например:

```
# sfdisk -s /dev/hda9
4095976
# mke2fs -i 1024 /dev/hda9
mke2fs 1.12, 9-Jul-98 for EXT2 FS 0.5b, 95/08/09
...
204798 blocks (5.00%) reserved for the super user
...
# mount /dev/hda9 /somewhere
# df /somewhere
Filesystem         1024-blocks  Used Available Capacity Mounted on
/dev/hda9            3574475      13  3369664      0%   /mnt
# df -i /somewhere
Filesystem           Inodes   IUsed   IFree  %IUsed Mounted on
/dev/hda9            4096000      11 4095989     0%  /mnt
#
```

У нас есть раздел с 4095976 блоками, создаем файловую систему ext2 на нем,монтируем его и обнаруживаем что у нас только 3574475 блока - 521501 блоков (12%) было использовано для нужд файловой системы. Заметим, что разница между имеющимися 3574475 блоками и доступными пользователю 3369664 - это 13 использованных блоков + 204798 блоков зарезервированных для root. Позднее эту цифру можно изменить с помощью tune2fs. Размер блока '-i 1024' выгоден только для раздела содержащего очень много маленьких файлов. По умолчанию это будет выглядеть так:

```
# mke2fs /dev/hda9
# mount /dev/hda9 /somewhere
# df /somewhere
Filesystem         1024-blocks  Used Available Capacity Mounted on
/dev/hda9            3958475      13  3753664      0%   /mnt
# df -i /somewhere
Filesystem           Inodes   IUsed   IFree  %IUsed Mounted on
/dev/hda9            1024000      11 1023989     0%  /mnt
#
```

Теперь только 137501 блока (3.3%) используются для inodes, т.е. высвободилось 384 МБ свободного места. (Каждый inode занимает 128 байт.) С другой стороны данная файловая система может содержать только 1024000 файлов (более чем достаточно), против 4096000 (слишком много) бывших возможными ранее.