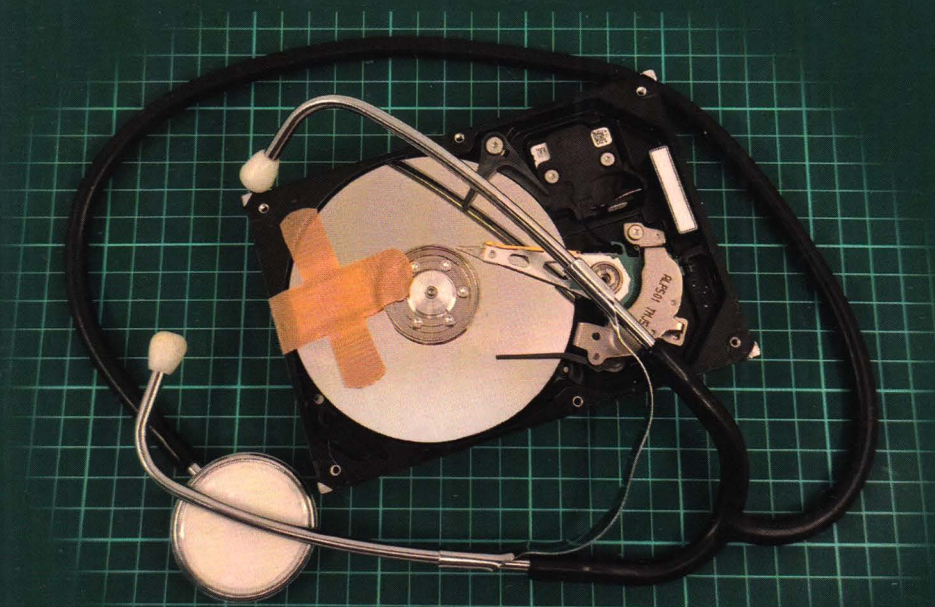
Крис Касперски
Валентин Холмогоров
Ксения Кирилова

2-е издание

ВОССТАНОВЛЕНИЕ ДАННЫХ

ПРАКТИЧЕСКОЕ РУКОВОДСТВО



- Автоматическое и ручное восстановление данных с жестких дисков и SSD
- Программы для восстановления удаленных файлов в Windows и Linux
- Приложения для резервного копирования и работы с резервными копиями
- Восстановление поврежденных носителей
- Ремонт жестких дисков
- Использование облачных хранилищ

bhv®

Крис Касперски
Валентин Холмогоров
Ксения Кирилова

ВОССТАНОВЛЕНИЕ ДАННЫХ

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2021

УДК 004.4
ББК 32.973.26-018.2
К28

Касперски, К.

К28 Восстановление данных. Практическое руководство / К. Касперски, В. А. Холмогоров, К. С. Кирилова. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2021. — 288 с.: ил.

ISBN 978-5-9775-6681-0

Книга представляет собой подробное пошаговое руководство по восстановлению поврежденных данных на жестких дисках, съемных носителях и твердотельных накопителях. Подробно рассмотрена структура популярных файловых систем: NTFS, ext3/ext4, UDF/UFS/FFS и др. Описаны автоматические методы восстановления данных для операционных систем Windows и Linux. Приведены способы ручного восстановления, используемые в случае, когда автоматическое восстановление невозможно. Даны рекомендации по ремонту жестких дисков.

Во втором издании уделено внимание работе с новыми приложениями для Windows 10 и Linux, с файловой системой ext4, твердотельными накопителями и флеш-памятью, рассмотрено использование облачных технологий. Книга сопровождается большим количеством полезных советов и исчерпывающим справочным материалом. На сайте издательства находятся цветные иллюстрации к книге и дополнительные материалы.

УДК 004.4
ББК 32.973.26-018.2

Группа подготовки издания:

Руководитель проекта	*Павел Шалин*
Зав. редакцией	*Людмила Гауль*
Компьютерная верстка	*Ольги Сергиенко*
Дизайн обложки	*Карины Соловьевой*

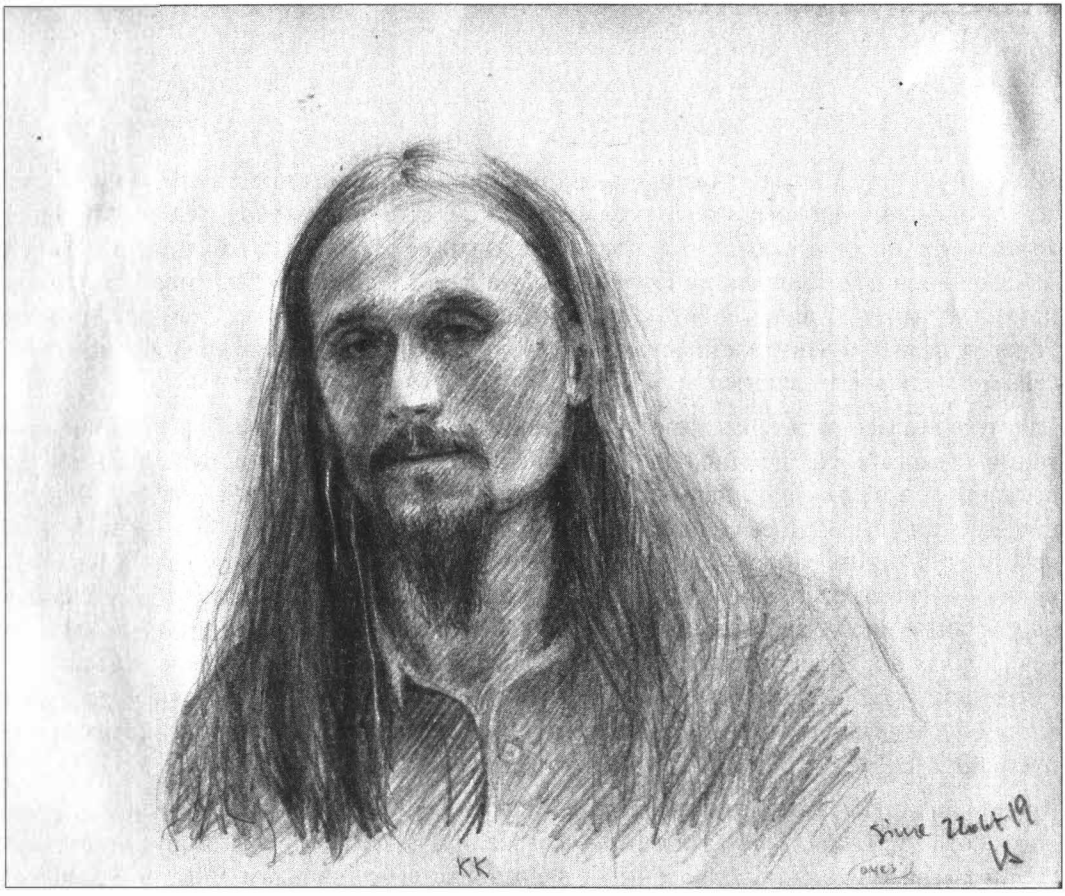
Подписано в печать 05.04.21.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 23,22.
Тираж 1000 экз. Заказ № 756.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-6681-0

Оглавление

Памяти Криса Касперски



(Автор портрета Ксения Кирилова)

Предисловие

Далекий 2008 год вошел в историю множеством запоминающихся событий. Он положил начало мировому финансовому кризису, с поста руководителя Кубы ушел казавшийся вечным Фидель Кастро, президентом России стал Дмитрий Медведев, начался военный конфликт в Грузии… И на фоне этой новостной повестки почти незамеченным осталось обстоятельство, навсегда изменившее ландшафт мира IT-технологий: в Интернете впервые начали распространяться вредоносные программы — шифровальщики.

Эти трояны (их также называют энкодерами) принято относить к категории программ-вымогателей. Проникнув на компьютер жертвы под видом какого-нибудь полезного приложения, например, проигрывателя Adobe Flash Player, энкодер шифрует файлы на дисках зараженного компьютера (а иногда — и в сетевых папках, доступных на запись), после чего требует у пользователя выкуп за их расшифровку. Хорошо, если шифровальщику удалось погубить лишь коллекцию скачанных из Интернета смешных картинок, но гораздо чаще зашифрованной оказывается вся деловая и личная переписка, рабочие документы, семейный фотоархив или бухгалтерские базы в программе 1С. Тут впору впасть в панику. Тем более требуемая вирусописателями сумма может колебаться от нескольких десятков до сотен тысяч рублей.

Вредоносное ПО, как и любые другие программы, пишут люди, а им свойственно допускать ошибки. Путем кропотливого анализа этих просчетов вирусописателей крупным антивирусным компаниям и энтузиастам-одиночкам иногда удавалось нащупать брешь в архитектуре энкодера и создать "противоядие", способное расшифровать поврежденные файлы. Но, по статистике, это получалось нечасто: примерно в 10% случаев. Разумеется, злоумышленники не могли не принять ответных мер. Они запутывали исходный код своих творений, зашифровывали и прятали ключи, применяли сразу несколько алгоритмов шифрования, с использованием которых последовательно "проходили" все содержимое диска по нескольку раз. В результате всех этих ухищрений расшифровать файлы порой не получалось даже у самих вирусописателей, причем после получения выкупа. Пострадавший пользователь оставался и без денег, и без ценной информации.

Показательным примером может служить эпидемия трояна nePetya, разразившаяся в июне 2017 года. Этот энкодер портил загрузочную запись раздела (VBR), запол-

няя соответствующий раздел диска мусорными данными. Затем он пытался зашифровать главную загрузочную запись (Master Boot Record, MBR) с использованием алгоритма XOR, а если это не удалось, портил ее. Наконец, он переходил к шифрованию хранящихся на дисках компьютера файлов с использованием AES-128-CBC, причем для каждого диска создавался собственный ключ, который тоже шифровался. Троян умел самостоятельно распространяться по сети с использованием уязвимости в протоколе SMB, в силу чего специалисты отнесли его к категории сетевых червей. От действия этого энкодера пострадали десятки тысяч компьютеров частных лиц, банков и крупных коммерческих организаций по всему миру. Поскольку адрес электронной почты, оставленный злоумышленниками для связи, оказался заблокированным, жертвы трояна не имели возможности связаться с ними для расшифровки файлов.

До случая с nePetya компьютерную общественность сотрясали эпидемии шифровальщиков WannaCry и других подобных троянов. Закономерным результатом заражения этими вредоносами чаще всего становилась безвозвратная потеря всей хранящейся на компьютере информации. Безусловно, существуют и другие причины разрушения данных — выход из строя жесткого диска, повреждение компьютера или сбой ОС, случайное форматирование раздела. Но надежность современных аппаратно-программных средств неуклонно растет, а беспечность пользователей, как показывает практика, — величина постоянная.

В 2019 году, когда пишутся эти строки, опасность распространения шифровальщиков по-прежнему велика. Никуда не делись и иные причины удаления данных. Как бы то ни было, в ряде случаев файлы еще можно спасти, если, конечно, знать, как это делается. Именно этой важной теме — защите и восстановлению ценной информации — была посвящена книга Криса Касперски "Восстановление данных. Практическое руководство", впервые опубликованная в нашем издательстве в 2006 году.

Трагическая гибель Криса Касперски в феврале 2017 года стала для всех нас настоящим шоком. Мы помнили Криса как тонкого, умнейшего человека, обладающего неподражаемым чувством юмора и энциклопедическим багажом знаний. Он был неутомимым исследователем, опытным аналитиком, признанным экспертом в сфере IT-технологий и великолепным рассказчиком, книгами и журнальными статьями которого зачитывались компьютерщики не только в России, но и в других странах мира. Невозможно представить, что он ушел и его больше нет с нами. Но след, который оставил в компьютерной индустрии Крис, еще долго будет востребован специалистами в сфере компьютерных технологий, а его публикации будут помогать людям в обретении новых знаний.

Эта книга основана на издании 2006 года, но в значительной степени дополнена, переработана и актуализирована. В ней рассматриваются современные средства восстановления данных для операционных систем семейства Linux и актуальной версии Windows 10, приведены рекомендации по выбору и ремонту жестких дисков и твердотельных накопителей. Даны советы по применению инструментов резервного копирования информации, в том числе с использованием облачных техноло-

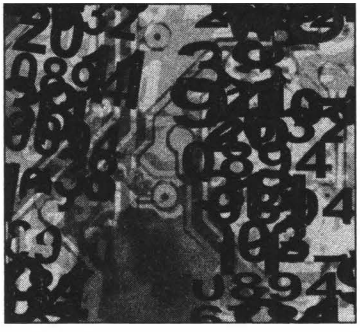
гий, которые еще не были распространены в 2006 году. Мы удалили из книги несколько устаревших разделов и дополнили ее самой актуальной информацией.

В результате проделанной работы получилось издание, которое, мы надеемся, будет полезно не только системным администраторам и специалистам по ремонту и обслуживанию ПК, но и простым пользователям, желающим уберечь свои файлы от утраты или повреждения. Ведь описанные здесь методы и технологии достаточно просты, чтобы ими мог воспользоваться каждый, кто умеет обращаться с компьютером. Желаем вам приятного чтения!

Благодарности

Пользуясь случаем, Ксения Кирилова выражает искреннюю благодарность своим родителям, без которых ее путь вряд ли сложился бы так, как сложился; Валентину Холмогорову за возможность приложить руку к этому проекту, а также CLO, ShəLMā, taueld и zhl0b за незаменимую помощь в процессе работы над книгой (порядок исключительно алфавитный!).

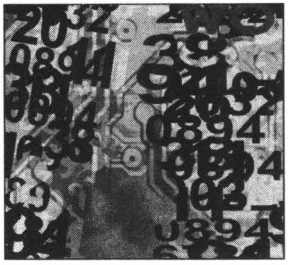
Валентин Холмогоров выражает искреннюю благодарность Владимиру Леонидовичу Лихачеву за помощь в подготовке этого издания, понимание, терпение и деятельное участие.



ЧАСТЬ I

Средства восстановления данных

ГЛАВА 1



Введение в восстановление данных

Современные операционные системы класса Windows NT и накопители с технологией в стиле S.M.A.R.T. поддерживают целый комплекс защитных мер по предотвращению непреднамеренной порчи данных. Тем не менее восстановление утраченных данных часто оказывается невозможным. Почему это происходит? Дело в том, что когда речь идет о *непреднамеренной* порче данных ключевое значение имеет именно эпитет "непреднамеренная". Главный виновник большинства разрушений — сам пользователь. Это именно он превращает свой компьютер в рассадник вирусов, это он бездумно устанавливает непроверенные программы откровенно безответственных производителей, манипулирует настройками, смысл и значение которых ему не до конца ясны. Иначе говоря, он пожинает плоды собственной некомпетентности и безответственности.

Существует ли возможность восстановления разрушенных данных? Без сомнения, она существует! Даже серьезные и масштабные разрушения бывают полностью или частично обратимы. Восстановление информации после катастрофического сбоя — всего лишь вопрос времени и квалификации (если квалификация недостаточна, то проблема восстановления превращается из технического вопроса в финансовый). Рядовой пользователь способен справиться лишь с наименее тяжкими повреждениями, не затрагивающими ключевых служебных структур. Но как раз такие разрушения и распространены шире всего. Именно поэтому описываемых здесь приемов восстановления данных, по крайней мере на первых порах, будет вполне достаточно.

Только не путайте знания и умение. Практические навыки следует приобретать не в "боевых" условиях, пытаясь восстановить действительно ценные или уникальные данные. Начинать тренировки нужно задолго до катастрофического сбоя. Экспериментировать рекомендуется с дисками, на которых нет никакой ценной информации, которую было бы жалко потерять. Как говорится, сапер ошибается всего лишь однажды. Восстановление данных не подчиняется четким и жестко заданным правилам, и правильная стратегия поведения заранее неизвестна. Существует много путей, но лишь один из них правилен, а остальные — фатальны. Восстановление данных — это искусство продвижения во мраке с завязанными глазами. Здесь

опыт, знания, интуиция и умение "чувствовать" компьютер сливаются воедино. Этому нельзя научиться. Это — талант, который у человека либо имеется, либо нет. Задача любой науки в первую очередь состоит в том, чтобы низвести мастерство до уровня технологии. Например, часто требуется свести интуитивно выполняемую операцию к более или менее постоянной последовательности действий, выполнить которую может практически каждый. То же самое можно сказать и об информационных технологиях. Кибернетика за последние годы совершила качественный скачок вперед, вложив в руки неквалифицированного пользователя мощнейший набор средств, но не объяснив при этом, как именно ими следует пользоваться. Отсюда и множество случаев безнадежного уничтожения данных, которые при квалифицированном подходе могли бы быть восстановлены.

Короче говоря, к битве против энтропии готовы? Тогда — банзай!

Разгребаем обломки

Если ваш винчестер издает странные звуки, операционная система не загружается, а на одном или нескольких логических дисках образовалась "каша", то самое лучшее, что вы можете сделать, — это немедленно выключить компьютер и передать его в руки профессионалов. Особенно это касается твердотельных накопителей, поскольку при отказе встроенной электроники без специального оборудования вернуть их к жизни практически нереально. Пытаясь "отремонтировать" данные самостоятельно, вы идете на огромный и ничем не оправданный риск. Этот риск особенно велик, если вы осуществляете восстановление не вручную, а с помощью различных автоматизированных утилит, слепо веря в их интеллектуальность.

С другой стороны, многие из тех, кто необоснованно претендует на звание специалиста, используют те же самые утилиты, что и вы. Отдавать винчестер на растерзание этим "специалистам" по меньшей мере неразумно. В таком случае вы рискуете потерять не только данные, но и деньги. Жители крупных городов практически всегда могут найти фирму, специализирующуюся на восстановлении данных и накопившую в этой области бесценный опыт и фирменные "ноу-хау". Выбирая одну из подобных фирм, услугами которой вы планируете воспользоваться, обратите особое внимание не только на техническую оснащенность компании, но и на квалификацию работающих там специалистов. Человек, профессионально занимающийся восстановлением данных, должен располагать особо чистой комнатой, прецизионным оборудованием для смены магнитных головок, не падать в обморок от вопросов, звучащих, например, так: "Что такое MFT и чем оно отличается от $MFT?" К таковым, в частности, относятся фирмы ACE (**http://www.datarec.ru**), ЕПОС (**http://www.epos.kiev.ua/**), DATARC (**http://www.datarc.ru**) и многие другие. Здесь работают специалисты, которым можно доверять! Поверьте, это не реклама, а констатация факта.

В глубинке дела обстоят значительно хуже. Массового рынка восстановления компьютерных носителей нет, отказы носят единичный характер, следовательно, нет и фирм, выбравших восстановление данных основным направлением своей деятельности. Повсюду процветают кустари и Левши-умельцы, среди которых, несомненно, встречаются и подлинные мастера своего дела. Тем не менее откровенных ламеров,

выдающих себя за профессионалов, намного больше. Если и вы оказались в такой ситуации, не унывайте. Вместо того чтобы жаловаться на судьбу, попробуйте обратиться к патриархам отечественного восстановления данных. Сделать это можно в том числе и через могущественную сеть Интернет.

Технологий удаленного восстановления данных, собственно говоря, всего две. В первом случае вы скачиваете программу восстановления, которая общается с удаленной программой. С ее помощью оператор может восстанавливать данные так, будто диски подключены к его компьютеру. Менее именитые компании могут использовать расшаривание экрана. Главный минус этого подхода состоит в том, что в течение всей процедуры восстановления вы будете вынуждены "висеть" в Интернете, наблюдая за тем, как оператор будет принимать/передавать данные, попутно пытаясь вернуть эту байтовую "мешанину" в исходный вид. Работа одной из таких программ, R-studio (демоверсию скачать можно отсюда: **http://www.r-studio.com/ru/Data_Recovery_Download.shtml**), показана на рис. 1.1.

Разумеется, в описанном случае речь идет только о логическом восстановлении разрушенных данных. Отремонтировать физически поврежденный винчестер через Интернет нереально. Тем не менее логические разрушения встречаются гораздо чаще физических повреждений.

И все же на случай физической неисправности диска некоторые компании предлагают жителям регионов свой сервис, при этом накопитель отправляется в лабораторию по почте или сдается в специальных пунктах приема. Разумеется, это своеобразный способ удаленного восстановления, но все же лучше, чем ничего!

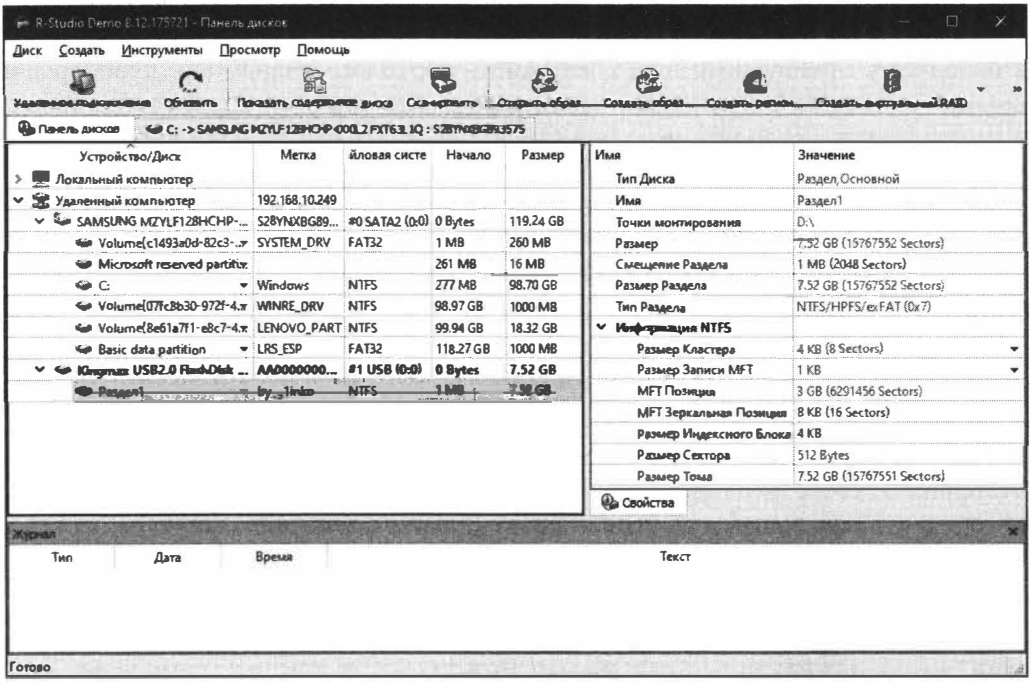


Рис. 1.1. Удаленное восстановление данных

Если же, несмотря ни на что, вы все-таки намерены восстанавливать диск самостоятельно, позвольте для начала дать вам несколько практических советов.

- Никогда не пытайтесь восстанавливать данные с физически неисправных накопителей! Если электрическая и/или механическая части диска вызывают у вас хотя бы тень сомнения, не пытайтесь восстанавливать данные, пока диск не будет полностью отремонтирован!
- Никогда не записывайте на восстанавливаемый диск никаких утилит, не пытайтесь загрузить с него Windows и уже тем более не запускайте `chkdsk` и средства дефрагментации!
- Не пытайтесь читать bad-сектора больше двух-трех раз на каждый непрерывный дефектный блок до тех пор, пока не будут прочитаны и сохранены все "здоровые" сектора!
- Занимайтесь восстановлением только в трезвом уме и здравой памяти. Иными словами, переждите, пока пройдет шоковое состояние от пережитого разрушения.

Физические повреждения

Жесткие диски — чрезвычайно надежные устройства, самостоятельно следящие за своей исправностью и автоматически переназначающие подозрительные сектора задолго до их полного разрушения. При бережном обращении и соблюдении всех рекомендаций производителя шансы столкнуться с физическим разрушением информации ничтожно малы (порядка 0,1–1% в зависимости от качества изготовления конкретного экземпляра). Тем не менее при нынешних масштабах производства ни одному бренду не удалось избежать "проколов". Например, субподрядчик Fujitsu — компания Cirrus Logic — однажды изменила химический состав подложки микросхем, в результате чего они стали впитывать влагу, через короткое время выводящую электронику из строя. Винчестеры от Samsung славятся своей чувствительностью к статическому электричеству, приводящему к "прострелу" микросхем кэш-памяти, после чего на диск пишется сплошной "мусор", необратимо разрушающий служебные структуры файловой системы.

Два примера поврежденных контроллеров жестких дисков, публикуемые с любезного разрешения владельцев сайта **http://www.hdd-info.ru**, приведены на рис. 1.2 и 1.3.

При отказе электроники плату обычно не ремонтируют, а заменяют целиком, приобретая "донора" точно такой же модели. При этом следует учитывать, что некоторые производители заносят калибровочные данные в микросхему ROM, которую следует аккуратно выпаять из неработающей платы и ввести в "донора". Если этого не сделать, то данные либо вообще не будут читаться, либо при первом же запуске винчестера окажутся необратимо испорченными (*более подробную информацию на эту тему можно найти в главах 2 и 5 данной книги*).

Никогда и ни при каких обстоятельствах не вскрывайте крышку гермоблока! Единственная пылинка, попавшая под головку винчестера, может стоить жизни вашему

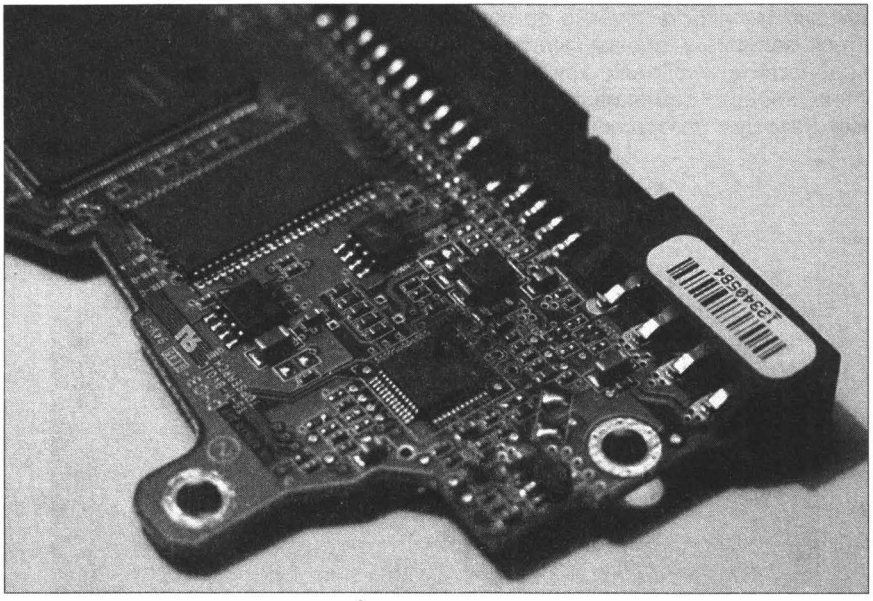


Рис. 1.2. "Сгоревший" контроллер жесткого диска
(публикуется с любезного разрешения владельцев сайта **http://www.hdd-info.ru**)

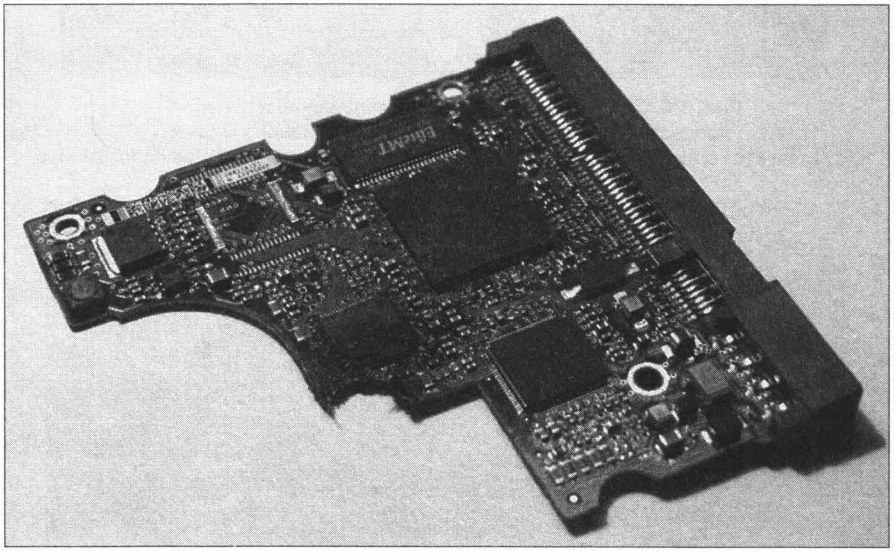


Рис. 1.3. Еще один неисправный контроллер
(публикуется с любезного разрешения владельцев сайта **http://www.hdd-info.ru**)

диску, а с ним и данным, ради восстановления которых и был затеян ремонт (рис. 1.4 и 1.5).

Примечание

Кстати, о головках. Среди обывателей ходит легенда о том, что они "залипают", и чтобы их "разлепить" якобы следует аккуратно стукнуть по винчестеру рессорой от трактора "Беларусь" или резко крутануть его вокруг своей оси, неизбежно выронив из рук.

Бóльшую нелепость сложно придумать! Когда пластины винчестера начинают вращаться, залипшие головки выдираются "с мясом", и "разлеплять" там уже нечего (если они действительно "залипали"). Подшипники (особенно гидродинамические), действительно, нередко заклинивают, да так, что вал невозможно провернуть даже пассатижами. Какое уж тут вращение в горизонтальном направлении...

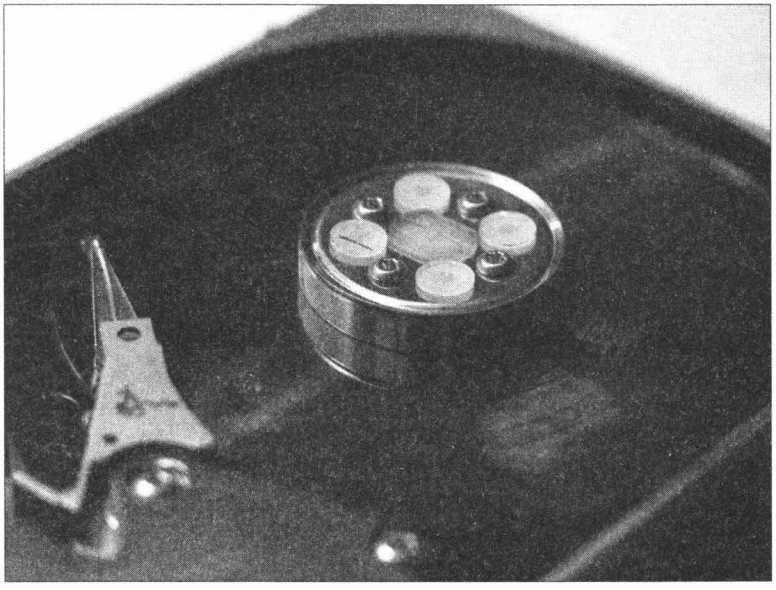


Рис. 1.4. Отпечатки пальцев на зеркальной поверхности — последствия вскрытия гермоблока в домашних условиях (публикуется с любезного разрешения владельцев сайта **http://www.hdd-info.ru**)

Рис. 1.5. Еще один "запиленный" диск, восстановить который, увы, уже невозможно (публикуется с любезного разрешения владельцев сайта **http://www.hdd-info.ru**)

Впрочем, до тотальных отказов дело обычно не доходит, и все повреждения обычно сводятся к появлению сбойных секторов. Обнаружив их, ни в коем случае не пытайтесь запускать диагностические утилиты, включая и утилиты от фирмы-производителя винчестера. По непонятной причине практически все они, встретив сбойный сектор, "мучают" его до победного конца, неизбежно распространяя зону воздействия дефекта как вглубь, так и вширь. Что еще хуже, такие утилиты могут изуродовать магнитную головку, цепляющуюся за неровности дефектной зоны. Иными словами, лекарство часто оказывается хуже, чем сама болезнь.

Каждый винчестер имеет специальный настроечный регистр, который, помимо всего прочего, задает и количество повторных попыток чтения, если с первой попытки сектор прочитать не удалось. Установите его либо в ноль (не делать повторов), либо в единицу, если ноль закреплен за значением количества повторов по умолчанию (как обстоят дела в конкретно взятом случае, поможет установить техническая документация, скачанная с сайта производителя). Длинное чтение секторов (long read) возвращает весь сектор целиком — пользовательские данные вместе с контрольной суммой. Различные модели жестких дисков имеют свои особенности реализации данной команды, которые, к сожалению, не всегда документированы. Часто требуемую информацию приходится по крупицам собирать в Интернете (как вариант — можно дизассемблировать прошивку, но это требует достаточно высокой квалификации).

Чаще всего сектор разрушается не целиком, а искажается лишь пара десятков байтов, расположенных наиболее неблагоприятным для корректирующих кодов образом. Согласитесь, что часть сектора — это намного лучше, чем совсем ничего.

Твердотельные же накопители, построенные на базе самой по себе довольно надежной флеш-памяти, выходят из строя, как правило, внезапно. Нет, они тоже честно "следят" за своим состоянием, стараясь распределять нагрузку равномерно (функция выравнивания износа, Wear Levelling) между всеми ячейками памяти и переназначая неисправные блоки. Но никто не застрахован от нежданных сюрпризов в работе контроллеров. Вот, к примеру, у дисков OCZ (ныне принадлежащей Toshiba) на базе контроллеров Sandforce была такая особенность, нареченная "sleep bug": при выходе из сна или гибернации компьютер мог зависнуть, после чего SSD "замыкался в себе" и переставал определяться BIOS и вообще как-либо.

В отличие от HDD, для новомодных SSD, не имеющих движущихся частей, механические повреждения совсем не так страшны даже для работающего диска. Физические неисправности могут заключаться только в нерабочей электрической схеме — если повезет, то это будет пробитый конденсатор в цепи питания, после замены которого диск вновь начнет работать. Если повезет не очень, то — отказавший контроллер или микросхемы памяти.

Поскольку в твердотельных дисках имеется несколько микросхем памяти (обычно 8 для форм-фактора 2,5", от 2 до 8 для M.2) и один файл может быть запросто разнесен по разным чипам, устройству нужно помнить, как они адресуются в соответствии с логическими адресами, выставляемыми протоколом SATA. Для этого данные о соответствии физических и логических адресов заносятся в таблицу трансляции (Translation Table), которая по мере износа ячеек может изменяться. Она хранится

либо в служебной области флеш-памяти, либо в выделенном участке памяти на плате. Без таблицы трансляции собрать линейный дамп из информации, разбросанной по неполному десятку чипов, будет крайне затруднительно (а последнее время все чаще SSD используют еще и аппаратное шифрование...), так что ее повреждение ведет к невозможности чтения хранящихся файлов без кропотливейшей операции по ее воссозданию.

Что касается ремонта твердотельных накопителей, то за редким исключением приходится выбирать — или восстановление диска, или восстановление хранящейся информации (о причинах такого грустного ультиматума будет рассказано в *главе 10*).

Логические разрушения

Ни одна из существующих файловых систем (например, FAT16/32 или HPFS) по своей надежности не выдерживает никакого сравнения с NTFS. Поэтому сосредоточим свое внимание исключительно на NTFS. Это очень надежная система и "уронить" ее можно только вместе со всем системным блоком, а для уничтожения данных потребуется тротил или нитроглицерин. Конечно, абсолютной защиты не существует, и катастрофы различной степени тяжести все же случаются.

Помимо официальной документации по NTFS имеется множество независимых хакерских источников. К их числу относятся исходные тексты драйверов NTFS, выдернутых из Linux, дизассемблерных листингов NTFS-утилит от Марка Руссиновича и т. д. Но все эти способы ненадежны, и драйверы NTFS от сторонних производителей плохо совместимы с новыми операционными системами, особенно с их локализованными версиями. К сожалению, никакой альтернативы вышеописанному подходу не существует.

Для восстановления винчестера, содержащего один или несколько разделов NTFS, подключите его "вторым" к компьютеру, на котором уже установлена операционная система Windows и все необходимое программное обеспечение. Затем запустите режим восстановления Windows, загрузив компьютер с дистрибутивного диска, после чего откройте Командную строку.

Непосредственно из командной строки можно запустить команду `chkdsk` *`логический диск`*. Если команда запущена с ключом `/p`, то будет проведена углубленная проверка с последующим внесением всех изменений. Запуск команды с ключом `/r` указывает на необходимость поиска и восстановления дефектных секторов. Пользоваться утилитой `chkdsk` категорически не рекомендуется. Однако если никаких других идей у вас нет, то на худой конец сойдет и `chkdsk`.

Если ни один из логических дисков не доступен (команда `C:` выдает ошибку, а `chkdsk` сообщает, что такого тома не существует), то, скорее всего, повреждена *таблица разделов* (partition table), находящаяся в *главной загрузочной записи* (Master Boot Record, MBR) или в *таблице GPT* (GUID Partition Table), пришедшей с новым стандартом UEFI. Для восстановления MBR были разработаны десятки утилит, но появились и программы, поддерживающие новый или даже оба формата

разметки разделов. Это, например, DMDE, которую можно найти по следующему адресу: **https://dmde.ru/download.html** (рис. 1.6). Консоль восстановления поддерживает команду `bootrec.exe /fixmbr`. Если верить названию этой команды, можно было бы предположить, что она должна лечить MBR. На самом деле никакого "лечения" она не осуществляет. Все, что делает эта команда, сводится лишь к записи в MBR системного загрузчика Windows, причем эта команда, доставшаяся современным версиям Windows в наследство от предыдущих поколений ОС, не всегда работает корректно. Чтобы восстановить загрузчик в Windows 10, можно воспользоваться командой `bcdboot C:\windows`, где вместо `C:` следует указать имя любого другого тома, в котором установлена Windows.

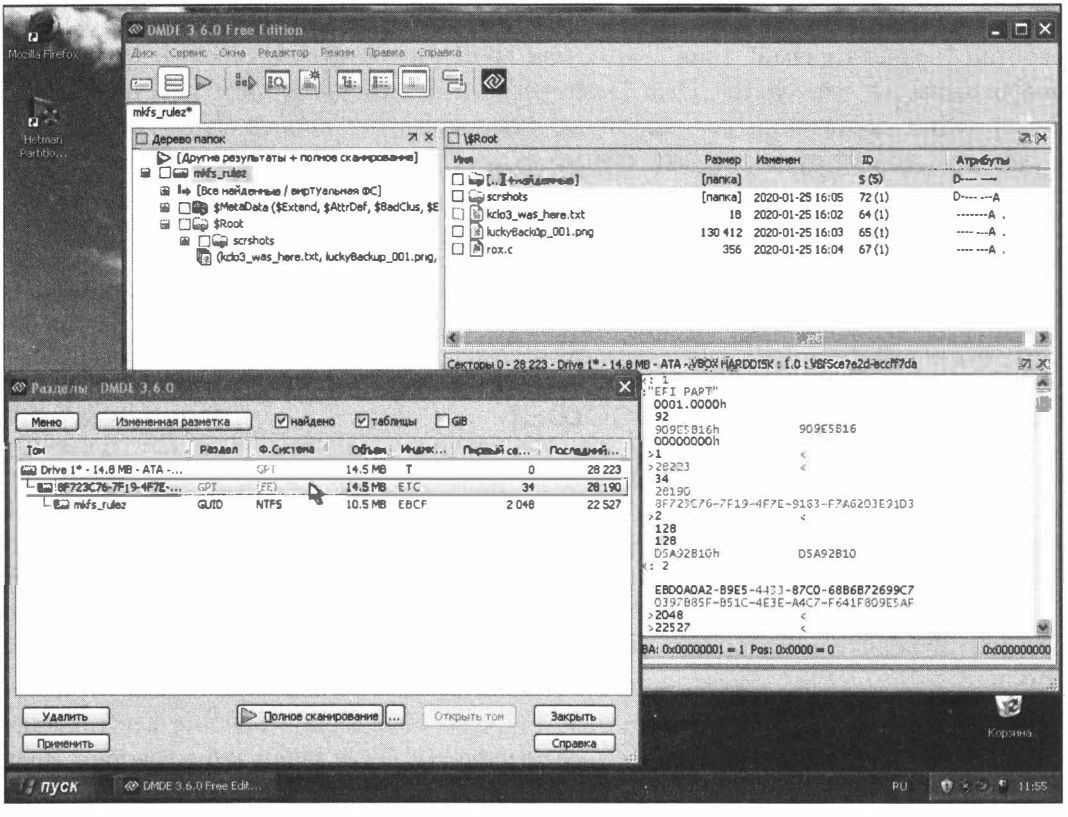


Рис. 1.6. DMDE работает с накопителями GPT даже на ОС, не поддерживающей новый формат разметки

Для восстановления загрузочного сектора в Командной строке можно использовать команду `bootrec.exe /fixboot`. Команда `FIXBOOT` (без параметров) "лечит" основной загрузочный сектор, а точнее, записывает в его начало стандартный boot-загрузчик. Воспользуйтесь ею, если операционная система не загружается, а на экране появляется сообщение `Missing operating system`.

Если корневой каталог не отображается или содержит бессвязный "мусор", значит, случилось самое страшное, что только могло произойти, — повреждена или унич-

тожена *главная файловая таблица* (Master File Table, MFT), описывающая размещение файлов на диске. Как правило, такое случается крайне редко. Благодаря поддержке механизма транзакций Windows автоматически выполняет откат, если операция обновления файловой системы завершилось неудачно. Однако когда драйвер NTFS начинает работать нестабильно (например, из-за конфликтов с другими драйверами или вследствие нарушения целостности кэш-буфера), транзакции уже не спасают и дисковая структура подвергается разрушениям. Первые четыре записи MFT хранятся в специальном резервном файле, на который указывает поле `Cluster to MFT mirr`, и могут быть элементарно восстановлены. А как же быть со всеми остальными? Ведь при современных объемах жестких дисков и астрономических количествах файлов число записей в MFT оказывается намного больше четырех. Можно ли восстановить и их, или они утеряны безвозвратно? Если диск не был обработан "врачевателями" типа `chkdsk` или NDD (который в хакерских кругах расшифровывается как Norton Disk Destroyer), то шансы на ручное восстановление информации достаточно велики. Более подробно этот вопрос будет рассмотрен в *главе 8*. Следует отметить, что там же будет приведено достаточно краткое описание данных методик, поскольку подробное и детальное изложение этого материала потребует сотен страниц убористого текста, к концу которых большинство "обычных" пользователей начнет откровенно скучать. Тем не менее никаких гарантий того, что после лечения диску не сделается еще хуже, у вас нет. Не очень-то утешительное заключение, но зато откровенное.

Как избежать катастрофы

От разрушения данных не застрахован никто. Однако если вы разработали правильную политику резервирования, то вся процедура восстановления сводится к замене негодного винчестера на новый и переписыванию данных с архивного накопителя. Восстанавливать данные на старый диск даже при логических разрушениях категорически недопустимо, т. к. если в процессе копирования выяснится, что резервный носитель испорчен, вы одновременно лишитесь как оригинала, так и копии.

Если же резервной копии нет или она устарела, немедленно бросьте все текущие дела, перейдите к *главе 3* и займитесь ее созданием! Ох, и зачем это я говорю? Ведь все равно не займетесь же. Если бы молодость знала, если бы старость могла. Если вы не намерены следовать этой рекомендации, то, по крайней мере, регулярно дефрагментируйте винчестер — дефрагментированные файлы восстанавливаются не в пример легче. Правда, если у вас твердотельный диск, то его дефрагментировать как раз не рекомендуется — при перестановке файлов расходуется ресурс флеш-памяти в SSD, не давая никакого существенного прироста производительности операций чтения. К примеру, Windows 10 при определении SSD сама отключает дефрагментацию для него (если только не активна защита системы).

Кстати говоря, из всех разделов чаще всего гибнет именно первый (т. е. диск C:). Поэтому категорически не рекомендуется хранить на нем что-либо ценное. Диск, разумеется, должен быть разбит на разделы. Но если вы не разбили его перед уста-

новкой операционной системы, не пытайтесь переразбивать его сейчас, т. к. риск потерять при этом все данные очень велик!

Внимательно следите за показаниями системы мониторинга S.M.A.R.T., для считывания которых разработано множество разнообразных утилит, например AIDA64 (рис. 1.7). Стремительный рост количества замещенных секторов (Reallocated Sector Count) не предвещает ничего хорошего, и такой диск рекомендуется срочно заменить. При этом следует учитывать именно *градиент*, а не абсолютное значение! Увеличение количества ошибок "сырого" чтения (Raw Read Error Rate) указывает на серьезные проблемы, и от такого диска лучше поскорее избавиться, скопировав все данные на другой. Рост численности характерных для HDD ошибок позиционирования (Seek Error Rate) и в особенности повторных раскруток шпинделя (Spin Retry Count) говорит о растущем рассогласовании в работе механической части, обычно заканчивающемся поломкой. С другой стороны, увеличение времени раскрутки шпинделя (Spin Up Time) — вполне нормальное явление, обусловленное конструктивными особенностями некоторых жестких дисков. Поэтому до тех пор, пока этот параметр не достигнет порогового значения, никаких поводов для волнений нет.

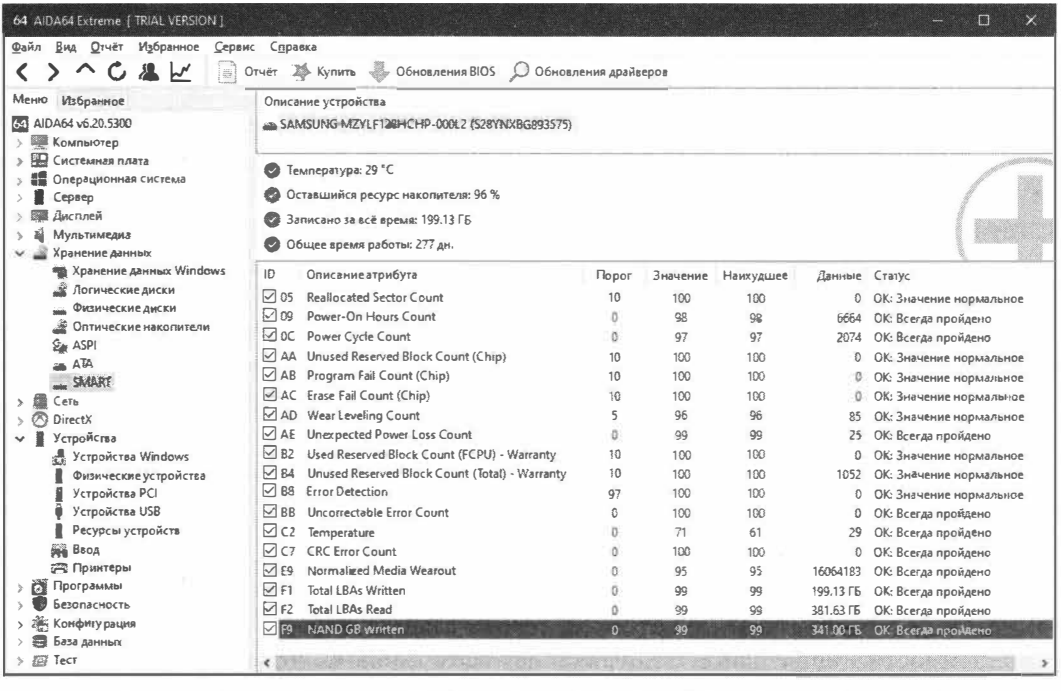


Рис. 1.7. Показания S.M.A.R.T. для SSD-накопителя, считанные утилитой AIDA64

Для SSD один из важнейших показателей S.M.A.R.T. — количество использованных резервных блоков (Used Reserved Block Count), прямо коррелирующее с числом переназначенных секторов. Оно явно говорит об износе основных ячеек памяти, и чем оно больше, тем ближе накопитель к заслуженной пенсии. Количество

пройденных циклов перезаписи, он же уровень износа (Wear Levelling Count), дает примерное представление о том, сколько раз перезаписывалась память носителя. Нормализованное значение уменьшается, начиная со 100, и когда оно доходит до нижней границы, производитель более не гарантирует успешную запись на него. Впрочем, это теоретически рассчитанная величина, так что устройство еще может и вполне успешно послужить и далее, но следует иметь в виду, что неизвестно, как долго.

Ошибки передачи данных по интерфейсу ATA (Ultra ATA CRC Error Rate) очень коварны. Если они превысят пределы корректирующей способности избыточных кодов, то файловая система разрушится. Проверьте, не перекручен ли интерфейсный кабель, и при необходимости укоротите его.

Внимательно следите за температурой, не допуская перегрева винчестера. Предельно допустимую температуру можно узнать из спецификации. Большинство современных винчестеров нормально выдерживают температуру окружающей среды до 50–60 °C (не путайте ее с показаниями встроенного термодатчика), не требуя дополнительного охлаждения. SSD в силу своего устройства не греются так сильно, как HDD, и поэтому чрезмерный нагрев твердотельного накопителя без видимой активности свидетельствует о неисправности аппаратной части. Заявленная производителями рабочая температура SSD обычно ограничивается значением в 70 °C, а температура хранения — до 85 °C.

Не разгоняйте процессор, PCI-шину и оперативную память! Все это может привести к необратимому разрушению служебных структур файловой системы, затраты на восстановление которой сведут на нет весь выигрыш, полученный от разгона. Кстати, винчестеры с большим кэшем лучше не приобретать, т. к., по слухам, они недостаточно надежны и имеют много нерешенных инженерных проблем. Если же вы счастливый обладатель твердотельного накопителя, то не забывайте, что они чувствительны к сбоям в подаче энергии. Если в SSD нет защиты от сбоев питания (Power Loss Protection), то при следующем включении он может не читаться или будут доступны не все данные, хранящиеся на нем. Также не забывайте, что со статическим электричеством при работе с SSD, как и с любыми платами, следует быть внимательным, иначе, если звезды не так сойдутся, можно остаться без данных и без диска.

Никогда не запускайте программы, в которых вы не уверены, и уж тем более не предоставляйте им права администратора! Всегда используйте минимум прав, требуемых для работы, входя в систему от имени администратора лишь при реальной необходимости. Запрещайте модификацию всех файлов, которые не собираетесь модифицировать в ближайшее время, отобрав этот атрибут у всех пользователей, от имени которых вы обычно регистрируетесь в системе.

Следуя всем перечисленным советам, вы многократно снижаете риск необратимой потери данных и значительно упрощаете процедуру их восстановления в случае, если они все-таки будут разрушены.

Windows и безопасность

Считается, что операционные системы семейства Windows NT, к которым относится и Windows 10, надежно защищают данные от разрушения. Во-первых, они блокируют прямой доступ к аппаратуре, во-вторых, обеспечивают возможность разграничения доступа, что позволяет обезопасить критичные файлы операционной системы от случайного удаления. Кроме того, в ОС Windows существуют механизмы теневого копирования, а современные версии BIOS спецификации UEFI используют протокол безопасной загрузки Secure Boot, который должен защитить систему от повреждения загрузочной записи и проникновения буткитов. Также в Windows, начиная с Vista, реализован механизм контроля учетных записей пользователей (User Account Control, UAC), призванный предотвратить запуск приложений с правами администратора, что должно защитить систему от действия вредоносных программ (для обхода этого механизма существует множество элементарных, но действенных способов). Казалось бы, безопасность ОС Windows благодаря этим инструментам должна быть на высоте.

И вы этому верите? Ха! 2017 год, по признанию аналитиков антивирусных компаний, можно было смело назвать годом троянов-шифровальщиков. Энкодеры распространялись преимущественно по каналам электронной почты, причем вирмейкеры рассылали не самого трояна, а маленький файл-загрузчик под видом ценного документа, который скачивал и запускал шифровальщика на компьютере жертвы. Появились и энкодеры-черви вроде нашумевшего WannaCry, умевшие распространяться по Сети самостоятельно, без участия пользователя. Тот же WannaCry, по данным англоязычной Википедии, инфицировал более 200 000 компьютеров в 150 странах мира, а нанесенный им ущерб эксперты оценивают в диапазоне от сотен миллионов до миллиардов долларов.

Летом того же года случилась эпидемия уже упоминавшегося NePetya (WannaCry-2), принесшего значительно меньший ущерб, но наделавшего немало шума. А в октябре начал распространяться энкодер BadRabbit, очень похожий на своих предшественников — он тоже обладал функционалом сетевого червя и умел распространяться самостоятельно.

2018 год выдался чуть более спокойным в плане распространения шифровальщиков. Так, в своем итоговом декабрьском обзоре вирусной активности компания "Доктор Веб" сообщила, что ежемесячно в их службу технической поддержки обращалось в среднем от 750 до 1000 пострадавших пользователей, файлы которых были зашифрованы подобными вредоносными программами. В совокупности получается порядка 10 000 пострадавших в год — и это только те пользователи, которые официально обратились за помощью.

Наверное, вы уже поняли, что все жертвы троянцев-шифровальщиков являются счастливыми обладателями компьютеров и ноутбуков под управлением ОС Windows. На сегодняшний день известно всего лишь два или три энкодера для Linux (один из них использовал старую уязвимость в прошивке сетевого дискового хранилища NAS, давно закрытую разработчиком), да и те не распространяются "в живой природе". Так что там говорилось о надежности этой операционной системы?

Конечно, причиной повреждения данных могут быть не только вредоносные программы. Но исторически сложилось так, что именно Windows в силу своей широкой распространенности является предметом пристального интереса вирмейкеров, и, как следствие, рассадником всевозможной компьютерной инфекции. К сожалению, успех в расшифровке поврежденных энкодерами файлов не гарантирован. Чаще гораздо эффективнее попытаться восстановить их иными способами, которые описаны на страницах этой книги.

Есть и еще один аспект безопасности, менее очевидный, но вместе с тем важный. Начиная с Windows Vista, корпорация Microsoft активно внедряет модель установки драйверов устройств, имеющих верифицированную цифровую подпись издателя, а 64-разрядные версии Windows вообще блокируют установку неподписанных драйверов. Казалось бы, это должно значительно усилить защищенность ОС и исключить ситуации, когда информация на диске повреждается из-за "кривого" драйвера. Как бы не так.

Например, многие пишущие приводы CD/DVD (и использующие их программы) исторически работали через Advanced SCSI Programming Interface (ASPI). А вы знаете, что драйвер ASPI позволяет любому приложению, независимо от уровня его полномочий, работать со всеми устройствами IDE/SCSI на низком уровне, например, стирая все сектора? В комплекте поставки Windows 10 (в частности, в 64-разрядной версии) уже отсутствуют базовые драйверы с поддержкой ASPI, но их до сих пор используют приложения для записи оптических дисков, в частности, Nero.

Вы решили ликвидировать эту брешь в системе безопасности и не пользуетесь драйвером ASPI? Можете ли вы теперь считать, что вы в безопасности? Нет, т. к. останется SCSI Pass Through Direct (SPTD), "намертво вживленный" в операционную систему (включая Windows 10) и позволяющий делать все то же самое, что и ASPI, пускай и с правами администратора. Неужели вы верите, что злоумышленнику будет так уж трудно их получить? Вы жестоко ошибаетесь! Чтение физического диска на секторном уровне по умолчанию доступно всем пользователям без исключения (и его не так-то просто запретить). В то же время на секторном уровне не существует понятия "привилегий" (access permissions), и файлы с конфиденциальной информацией (включая информацию по аутентификации) доступны всем пользователям (строго говоря, никаких "файлов" на секторном уровне не существует, но для хакеров это не преграда).

Что касается драйверов, то их полномочия ничем не ограничены, и они способны делать с системой все что угодно. Нередко драйверы содержат критические ошибки, которые могут привести к потере данных. Особенно этим грешат драйверы от кустарных разработчиков, наплевательски относящихся к культуре программирования. Можно ли запретить приложению устанавливать свои драйверы в системе? Ну, в общем-то, можно (для этого достаточно лишь быть зарегистрированным в системе с правами простого пользователя на момент установки приложения). Вот только что это нам даст? Без драйвера приложение работать не будет, а достойную альтернативу ему удается найти не всегда.

Диагностика и устранение повреждений накопителей

Наиболее распространенные повреждения жестких дисков и твердотельных накопителей, а также методы их диагностики и устранения кратко описаны в табл. 1.1.

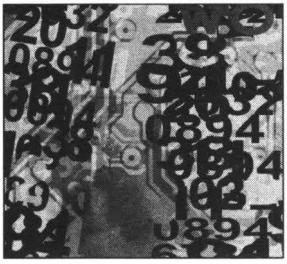
Таблица 1.1. *Диагностика и методы устранения повреждений накопителей*

Симптом		Диагноз	Устранение
Накопитель не опознается BIOS		Отказ электроники накопителя	Обратитесь в специализированную фирму по ремонту носителей или попытайтесь выполнить ремонт самостоятельно по методике, описанной в *главе 5* или *10*
Операционная система не загружается, BIOS выдает сообщения `Non-system disk`, `Missing operating system`, запускается UEFI Shell или нечто подобное	При загрузке с Live-дистрибутива логические диски не видны (команда `C:` возвращает сообщение об ошибке)	Повреждена таблица разделов или сигнатура `55h AAh` для MBR и `"EFI PART"` для GPT	Восстановите MBR (GPT) самостоятельно по методике, описанной в *главе 6* или с помощью специализированной утилиты
	При загрузке с Live-дистрибутива логические разделы видны и исправны (команды `C:` и `dir C:` работают)	Поврежден загрузочный сектор и/или MBR	Запустите консоль восстановления и дайте команды `FIXMBR` и `FIXBOOT`
	При загрузке с Live-дистрибутива логические разделы видны, но команда `dir C:` дает ошибку	Поврежден загрузочный сектор или MFT	Попробуйте восстановить boot-сектор вручную по методике, описанной в *главе 6*. Восстановите MFT с помощью резервной копии из `MFTMirr`
Операционная система начинает загружаться, но затем процесс загрузки "зависает" или прерывается с выводом сообщения об ошибке	При загрузке с Live-дистрибутива команда `dir C:` выполняется нормально, а `chkdsk` не находит ошибок	Возникла проблема с конфигурацией самой операционной системы	Переустановите операционную систему, предварительно скопировав все ценные файлы на другой носитель
	При загрузке с Live-дистрибутива команда `dir` в одном или нескольких подкаталогах выводит "мусор" или показывает не все файлы	Повреждена MFT или одна из ее дочерних структур	Запустите DiskExplorer и прочитайте все файлы из MFT напрямую, в обход индексов
	При загрузке с Live-дистрибутива некоторые файлы не читаются, при этом винчестер издает ритмичные скребущие звуки	Физические повреждения поверхности диска	Запустите утилиту восстановления жесткого диска, полученную от его производителя

Таблица 1.1 *(окончание)*

Симптом		Диагноз	Устранение
	Некоторые файлы содержат в себе фрагменты других файлов	На диске образовались пересекающиеся кластеры	Запустите `chkdsk`
	Свободное место на диске стабильно уменьшается без видимых причин	Некоторые кластеры оказались потерянными	Запустите `chkdsk`

ГЛАВА 2



Основные средства восстановления данных

Даже если у вас золотые руки и светлая голова, при восстановлении данных ни за что не обойтись без специализированных инструментов. А уж подходящий инструментарий сегодня отыскать несложно: как говорится, в Интернете есть все. Или почти все. Как минимум там можно найти программы, позволяющие автоматизировать восстановление поврежденных данных в различных ОС и в различных файловых системах, если, конечно, сам носитель информации все еще физически жив. Естественно, прежде чем что-то автоматизировать, необходимо разобраться в ситуации, а выполнить эту работу может только человек. Компьютеру доверять ее ни в коем случае нельзя — для этого он недостаточно интеллектуален.

Среди утилит, представленных на рынке, есть практически все необходимое. Естественно, большинство таких утилит распространяются на коммерческой основе, и за них приходится платить. К сожалению, многие из дорогостоящих инструментов не оправдывают возлагаемых на них ожиданий, и к выброшенным на ветер деньгам примешивается горечь от утраты данных. Работая над этой книгой, авторы протестировали множество разнообразных программных продуктов. В этой главе будут кратко описаны наиболее известные из них и даны рекомендации по их использованию.

Загрузочные носители для Windows

В составе операционной системы Windows 10 имеется набор инструментов для восстановления работоспособности ОС в случае сбоев, но они годятся исключительно для того, чтобы привести в порядок загрузочную запись или перезаписать утерянные системные файлы из резервной копии. То есть этот инструментарий создан, чтобы позволить Windows загрузиться, если штатная загрузка по какой-то причине невозможна. О восстановлении пользовательских файлов тут речи не идет.

Если сбой окажется не настолько серьезным, чтобы воспрепятствовать загрузке Windows, попытка "лечения" диска в многозадачной среде носит весьма непредсказуемый характер. Записывая что-либо на диск в обход драйвера файловой системы, вы сильно рискуете. Проиллюстрируем это утверждение на простом примере.

Представьте себе, что вы восстанавливаете ранее удаленный файл, обновляя святую святых файловой системы NTFS — главную файловую таблицу (MFT), а в это время система создает или удаляет другой файл, обращаясь при этом к тому же самому сектору, что и вы. Что произойдет в результате? Правильно — файл, а возможно, и весь дисковый том, окажется окончательно разрушенным. Кроме того, система блокирует активные исполняемые файлы и файлы данных, что делает невозможным их восстановление даже при наличии архивной копии. О борьбе с вирусами в таких условиях лучше вообще не упоминать. Многие вирусы, обосновавшись в системе, блокируют запуск антивирусных программ или умело скрываются от них, не позволяя себя удалить или обнаружить.

Для того чтобы получить доступ к разделам с NTFS, можно подключить восстанавливаемый диск "вторым" к системе с работоспособной Windows. Для этого даже не обязательно иметь два компьютера. Если у вас ноутбук или вам неохота вскрывать корпус ПК, можно воспользоваться кейсом для подключения жесткого диска в качестве внешнего накопителя — он позволяет присоединить диск к USB с помощью кабеля и наслаждаться жизнью.

Динамические диски, появившиеся в Windows 2000, хранят свои атрибуты на фиксированных участках диска и потому не привязаны к своей "родной" системе. С шифрованными файлами дела обстоят не в пример хуже. Ключ шифрования хранится глубоко в недрах пользовательского профиля, и на другой системе расшифровка файлов оказывается невозможной. Причем создание пользователя с таким же именем и паролем не решает проблемы, т. к. ключ шифрования генерируется системой случайным образом и не может быть воспроизведен. В таких ситуациях не остается никакого другого пути, кроме атаки по методу "грубой силы".

Некоторые типы разрушений файловой системы способны вызывать "зависание" оригинального драйвера NTFS или приводить к появлению синего экрана смерти (Blue Screen of Death, BSOD). Это создает серьезные проблемы, т. к. для восстановления диска требуется запустить средства восстановления (минимально необходимый инструментарий), а для их запуска необходимо загрузить Windows (а вот это как раз и невозможно!). В подобной ситуации можно подключить восстанавливаемый диск к системе Linux. Драйвер Linux игнорирует вспомогательные структуры файловой системы (например, файл транзакций) и потому успешно монтирует даже диски, содержащие сплошной "мусор".

Начиная с Windows 2000, Microsoft включила в операционную систему некоторое подобие загрузчика, способного стартовать с оптического диска или флешки и поддерживающего NTFS. Называется это средство консолью восстановления (Recovery Console). Это действительно консоль, ничего не "знающая" о GUI и способная запускать лишь консольные приложения (например, chkdsk.exe и другие утилиты подобного типа). Чтобы запустить Recovery Console в Windows 10, откройте Главное меню системы, нажмите и удерживайте клавишу <Shift>, щелкните мышью на кнопке выключения питания и выберите в открывшемся меню пункт **Перезагрузка** (рис. 2.1). После перезапуска система загрузится в режиме восстановления. Выберите пункт **Поиск и устранение неисправностей**, затем **Дополнительные параметры** и, наконец — **Командная строка**.

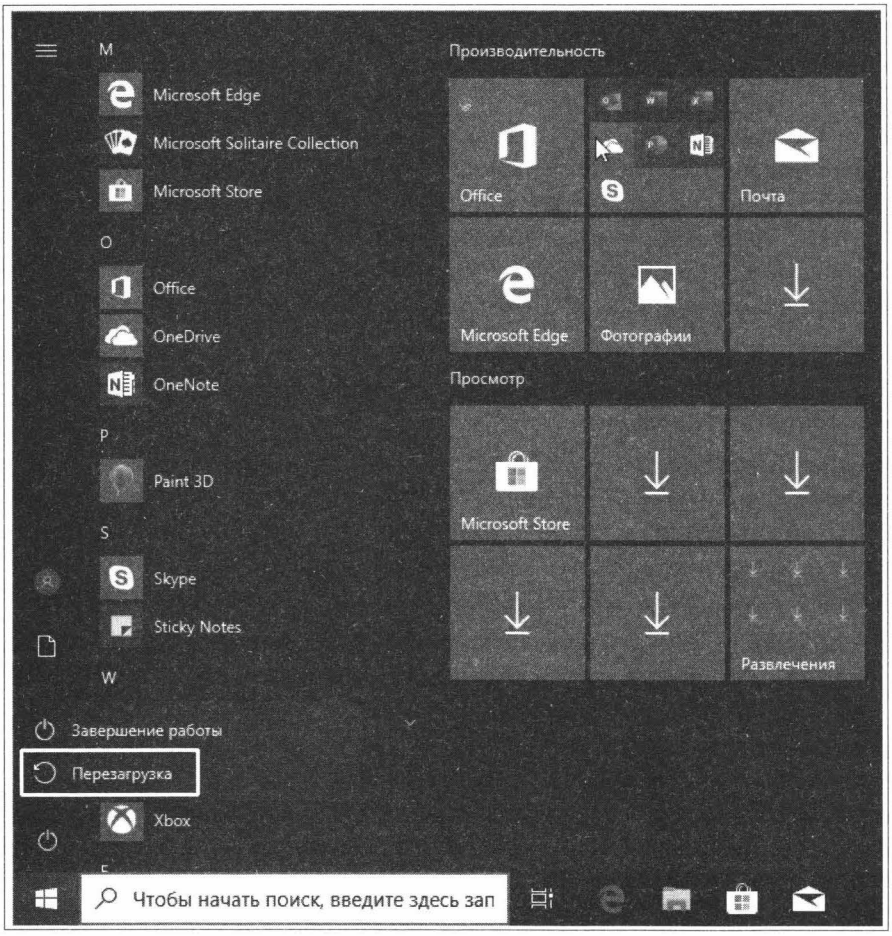


Рис. 2.1. Перезагрузка компьютера в Windows 10

Вам будет предложено выбрать учетную запись пользователя Windows и ввести пароль этой учетной записи (запустить консоль не удастся, если вы забыли пароль, или если поврежден системный реестр). После успешной регистрации запустится командный интерпретатор, теоретически позволяющий скопировать уцелевшие файлы на другой диск. Находясь в консоли восстановления, вы можете:

- запускать `chkdsk` (полезность этого "доктора" весьма сомнительна, т. к. он зачастую лишь усугубляет разрушения);
- создавать и удалять разделы на жестком диске;
- перезаписывать MBR и boot-сектор;
- форматировать логические диски;
- управлять службами и драйверами;
- удалять, копировать, переименовывать файлы и изменять их атрибуты (включая те, что блокируются при запуске системы);
- выполнять другие сервисные операции.

При желании вы можете запускать и свои собственные консольные приложения, а также приложения с поддержкой командной строки, написанные другими разработчиками.

Начиная с Windows XP, идея консоли восстановления получила дальнейшее развитие, вылившееся в *Windows PE (Windows Preinstall Environment)*. Это — слегка усеченная версия Windows, способная загружаться с флешки или оптического диска и запускать GUI-приложения. Фактически она полностью заменяет собой "второй" жесткий диск, и для восстановления системы с ее помощью не требуется никакого дополнительного оборудования!

Наиболее актуальную версию Windows PE можно скачать с сайта

https://docs.microsoft.com/en-us/windows-hardware/manufacture/desktop/download-winpe--windows-pe.

Эта урезанная версия операционной системы входит в состав Windows Assessment and Deployment Kit (ADK) — набора программ для специалистов по ремонту и сборке ПК, который также можно бесплатно загрузить с сайта Microsoft. С помощью этого инструмента можно создать загрузочный флеш-накопитель или загрузочный оптический диск: все инструкции имеются на вышеуказанном сайте.

Если по каким-либо причинам вы не имеете возможности получить официальную копию Windows PE, можно воспользоваться одной из "кустарных" сборок "реанимационных дисков", которые нетрудно отыскать на торрентах. Обычно такие диски включают в себя готовый набор утилит для диагностики аппаратных средств компьютера, резервного копирования и восстановления данных. Как правило, "реанимационные" диски распространяются в виде ISO-образов загрузочного CD или DVD. Вам останется только прожечь этот образ на болванку или записать на флешку с использованием любой подходящей утилиты для создания загрузочных флеш-накопителей.

Live Linux CD

За последнее время появилось множество дистрибутивов Linux, загружающихся прямо с CD-ROM или флеш-накопителя и не требующих установки на компьютер. Некоторые даже способны загружать живую систему непосредственно в оперативную память, что ускоряет ее работу, а также позволяет работать далее уже без загрузочного носителя (это дает возможность записывать восстанавливаемые данные в том же дисководе, с которого загрузились, а также убережет в случае действий, способных поломать еще и спасительную флешку). Нужно ли говорить, что это очень удобная штука для восстановления данных. Тем не менее далеко не все дистрибутивы пригодны для этой цели, и не все пригодные одинаково хороши, поэтому краткий обзор не помешает.

- **KNOPPIX** — с нашей точки зрения, это самый лучший из всех имеющихся дистрибутивов. Основан на GNU/Debian с легковесной оболочной LXDE. Содержит практически все: от дисковых утилит и компиляторов до офисных пакетов и мультимедийных приложений. Очень шустро работает, для запуска требует от

128 Мбайт оперативной памяти (для комфортной работы желательно, конечно, хотя бы 1 Гбайт RAM, потому как имеющееся ПО он распаковывает "на лету" в память). В 2007 году издательство O'Reilly переиздало шикарную книгу "KNOPPIX Hacks", содержащую главы, посвященные технике восстановления данных. KNOPPIX в версиях для DVD или CD можно загрузить через Интернет (например, отсюда: **http://www.knoppix.net/get.php**).

- **Frenzy 1.4** — дистрибутив, основанный на FreeBSD с небольшим количеством дисковых утилит, способных работать, в частности, с ext2fs/ext3fs, FAT и NTFS. Следует при этом отметить, что основной файловой системой самой BSD является UFS/FFS, и поддержка других систем в ней весьма ограниченна. Тем не менее для восстановительных работ данный дистрибутив, в отличие от немногих других живых систем на базе Berkeley UNIX, вполне пригоден, и всем поклонникам BSD его можно смело рекомендовать. Да и размер образа, надо сказать, невелик — всего около 100 Мбайт.
- **SystemRescueCD** — интересный дистрибутив на базе Arch Linux, созданный для восстановления систем и данных после сбоев. Обладает неплохим набором инструментов для работы с разнообразными файловыми системами, среди которых ext2/3/4, FAT и NTFS, ReiserFS, XFS. Занимает всего один диск, и в ходе опытов запустился в графическом режиме (с оболочкой LXDE) при 512 Мбайт оперативной памяти. Помимо восстановительных утилит, содержит пару вайперов, несколько браузеров, полдюжины текстовых редакторов на любой вкус и множество инструментов администратора.

Выбор носителей для копирования

Очевидно, что восстановленные данные нужно куда-то сохранить. На сегодняшний день наилучший выбор — это несколько флешек или внешний жесткий диск, который можно подключить к компьютеру через интерфейс USB. Основная проблема здесь заключается в том, что Windows PE не в состоянии монтировать разрушенные диски NTFS (на некоторых из них драйвер NTFS "зависает" или вызывает появление синего экрана смерти).

Таким образом, оптимальный инструмент для резервного копирования данных — внешний жесткий диск или твердотельный накопитель форм-фактора 3,5 или 2,5 дюйма. Для этих целей можно приобрести бывший в употреблении, но исправный жесткий диск, воспользовавшись каким-либо сервисом бесплатных объявлений, а внешний кейс или переходник SATA-USB для "ноутбучного" диска очень недорого можно купить в интернет-магазине AliExpress.

Дисковые редакторы

Времена, когда остатки побитых данных приходилось собирать с дисков вручную, давно прошли. Сейчас существует множество автоматизированных программ, позволяющих надежно восстанавливать информацию и файлы на поврежденных раз-

делах жестких дисков в полностью автоматическом режиме. Да и обработать диск объемом в несколько терабайт без средств автоматизации — задача практически невыполнимая.

Вместе с тем существует отдельная категория программ, которая называется дисковыми редакторами. Они позволяют работать с физической структурой диска, просматривая его содержимое по секторам. Иногда это необходимо, если автоматизированные утилиты не сумели справиться со своей задачей.

DiskExplorer от Runtime Software

https://www.runtime.org/diskexpl.htm

Это поистине великолепный дисковый редактор, самый лучший из всех, с которыми нам только доводилось работать. Фактически это аналог старого доброго Norton DiskEditor, популярного в конце 90-х дискового редактора для MS-DOS и Windows 9*x*, правда, в отличие от него, современная версия DiskExplorer прекрасно работает в Windows 10 (рис. 2.2).

С его помощью можно просматривать все основные структуры NTFS в естественном виде, монтировать виртуальные диски, работать с образами лазерных и жестких дисков, перемещаться по каталогам, восстанавливать удаленные файлы из лю-

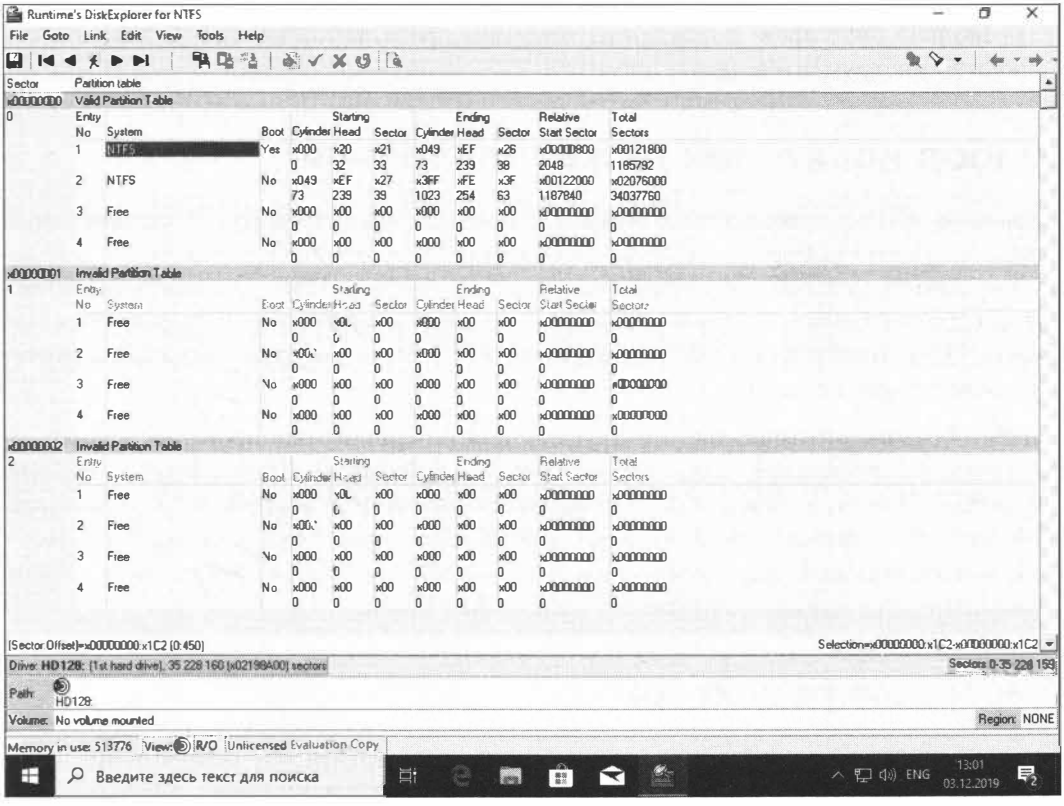


Рис. 2.2. Программа DiskExplorer

бой записи MFT, копировать файлы (и даже целые каталоги) с возможностью предварительного их просмотра в текстовом или шестнадцатеричном формате. И это еще далеко не все! Редактор предоставляет удобную систему навигации (приблизительно такую же, как в браузере или IDA PRO, включая поддержку гиперссылок), изобилие "горячих" клавиш, а также поддерживает историю переходов. Кроме того, он реализует мощную систему поиска с поддержкой основных структур (`INDEX`, `MFT`, `Partition`) и предоставляет возможность поиска ссылок на текущий сектор. С его помощью можно проводить удаленное восстановление диска с подключением по TCP/IP, локальной сети или по прямому кабельному соединению. Все числа выводятся в двух системах счисления — шестнадцатеричной и десятичной.

Короче говоря, это наш основной (и притом горячо любимый!) инструмент для исследования файловой системы и восстановления данных. Первое же знакомство с ним вызывает эйфорию, граничащую со щенячьим восторгом. Наконец-то мы получили то, о чем так долго мечтали. Естественно, за все хорошее надо платить. Полнофункциональная версия Runtime's Disk Explorer — это коммерческий продукт. Его демонстрационная версия, доступная для скачивания, лишена возможности записи на диск. Причем имеется несколько различных версий редактора: для NTFS, для файловых систем FAT, и еще одна — для ОС семейства Linux.

DM Disk Editor

https://dmde.ru/

DM Disk Editor (DMDE) — это дисковый редактор российского производства, автором которого является программист Дмитрий Сидоров. Программа имеет коммерческую лицензию, но с сайта разработчика можно бесплатно скачать демоверсию. DMDE поддерживает файловые системы NTFS/NTFS5, FAT12/16, FAT32, exFAT, Ext2/3/4, HFS+/HFSX, ReFS и работает во всех версиях Windows, начиная с 9*x* и заканчивая Windows 10 (рис. 2.3).

Интересная особенность программы заключается в том, что она не требует установки — достаточно скопировать содержимое архива на диск и запустить исполняемый файл. Помимо функций дискового редактора, DMDE имеет встроенный менеджер разделов, с помощью которого можно быстро отыскать и восстановить потерянный логический раздел на поврежденном диске. Для этого используются данные, извлекаемые из загрузочных секторов и суперблоков (superblock) разделов FAT, NTFS, exFAT, Ext2/3/4, HFS. Также менеджер позволяет копировать образы дисков, клонировать разделы и выполнять реконструкцию RAID-массивов.

Встроенные возможности программы дают возможность восстанавливать данные после сложных повреждений, удаления и перемещения файлов, форматирования, даже в случае, если информация была частично перезаписана.

В случаях, когда воспользоваться структурой файловой системы для восстановления информации невозможно, программа может выполнить восстановление файлов по сигнатурам (так называемый Raw-поиск). Дисковый редактор позволяет автоматически блокировать тома для разделов Windows, а также исследовать различные объекты структуры диска, такие как таблицы разделов MBR, GPT, загрузочные

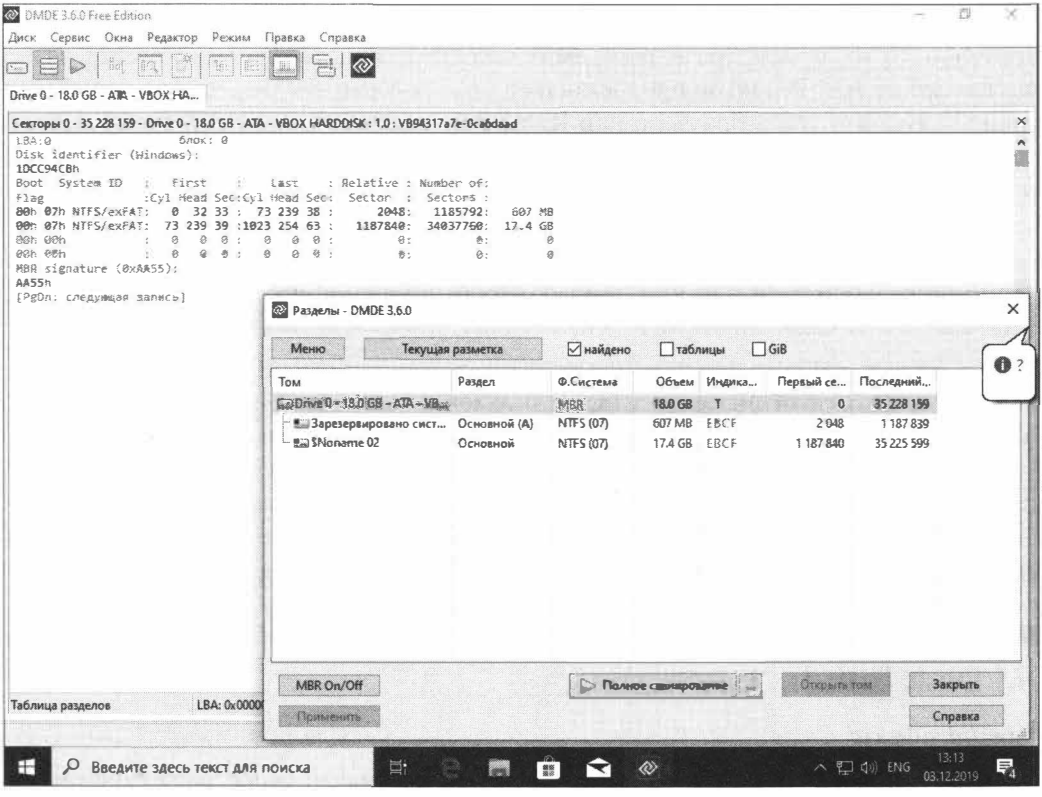


Рис. 2.3. DM Disk Editor в Windows 10

секторы, структуру папок в разделе. В интерфейсе DMDE предусмотрена возможность перехода между связанными элементами по щелчку мыши. Можно просматривать содержимое диска по кластерам.

С помощью других инструментов DMDE можно работать с RAID-массивами уровней 0–6, работать с образами дисков, а именно: осуществлять их клонирование, посекторное копирование, реверсивное копирование и развертывание, гибко управлять поведением программы при обнаружении в структуре диска бэд-секторов. К несомненным достоинствам DMDE можно отнести то, что программа умеет работать с файлами, имена которых содержат символы Unicode и национальных языков (в том числе кириллицы), работать с большими файлами и секторами большого размера, сжатыми файлами NTFS, выполнять трансляцию секторов. При этом DMDE не использует стандартный драйвер NTFS, благодаря чему программа может взаимодействовать с поврежденными дисками без опасения "завалить" систему в BSOD.

Sector Inspector

https://www.microsoft.com/en-us/download/details.aspx?id=19470

Данный инструмент входит в бесплатно распространяемый фирмой Microsoft пакет Windows Resource Kit. Он представляет собой пакетную утилиту для чтения и запи-

си отдельных секторов в файл. Sector Inspector (рис. 2.4) поддерживает режимы адресации LBA и CHS. При запуске без параметров он выводит декодированную таблицу разделов вместе с расширенными разделами и загрузочными секторами. Редактирование диска осуществляется путем правки предварительно сохраненного дампа сектора в любом подходящем hex-редакторе с последующей записью исправленной версии на диск. Естественно, это непроизводительно и неудобно, однако Sector Inspector — единственный известный нам редактор, поддерживающий работу из Recovery Console. Именно поэтому в некоторых случаях он бывает просто незаменим!

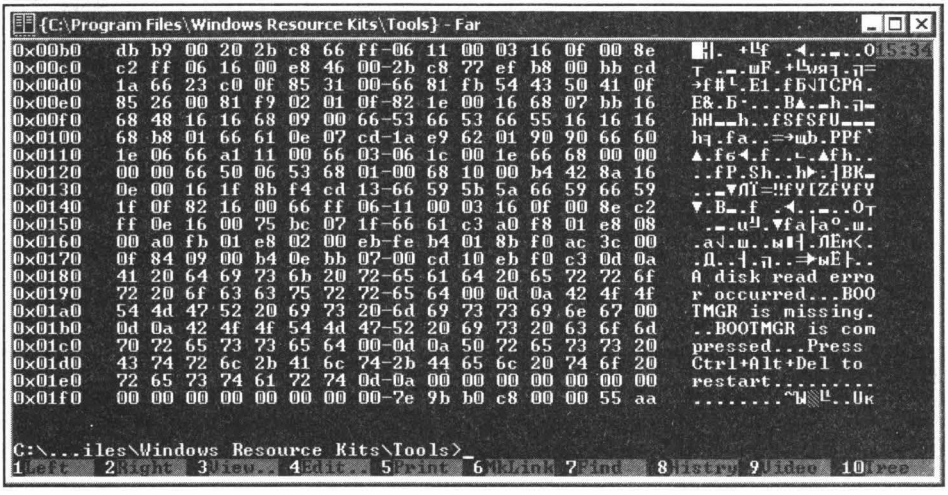


Рис. 2.4. Sector Inspector за работой

Linux Disk Editor

Программ, пригодных для восстановления данных, под Linux совсем немного. Фактически их намного меньше, чем под Windows NT, да и тем, что имеются, до совершенства далеко, как до Луны. Но ведь не разрабатывать же весь необходимый инструментарий самостоятельно?! Тем более что уже упомянутый DMDE, да и не он один, имеет версию под Linux. Будем использовать то, что дают, утешая себя тем, что программистам первых поколений, работавшим на "большом железе" (динозаврах машинной эры), приходилось намного сложнее.

Некогда в Linux для редактирования дисков применялась программа lde (Linux Disk Editor). Список поддерживаемых файловых систем ограничивается ext2fs, minix, xiafs и отчасти FAT. В современных реалиях этого едва достаточно, так что с учетом того, что последняя версия датируется маем 2005 и очень неохотно работает в нынешних системах, упомянутая программа представляет скорее исторический интерес. Сами возможности ее весьма неплохи — просмотр суперблока, файловых записей (inode) и каталогов в удобочитаемом виде, режим восстановления с ручным редактированием списка прямых/косвенных блоков и прочие прелести, упрощающие низкоуровневую работу с диском. К сожалению, проект застопорился, а полноцен-

ной замены на горизонте пока не видать. Исходные тексты, впрочем, по-прежнему доступны по адресу **https://lde.sourceforge.net**.

Шестнадцатеричные редакторы

UNIX — это вам не Windows! Без дисковых редакторов здесь, в принципе, можно и обойтись. Берем любой hex-редактор, открываем соответствующее дисковое устройство (например, `/dev/sdb1`) и редактируем его в свое удовольствие.

Программисты старшего поколения, должно быть, помнят, как во времена первой молодости MS-DOS, когда еще не существовало таких средств, как HIEW или QVIEW, правка исполняемых файлов на предмет "отлома" ненужного `7xh` обычно осуществлялась редактором DiskEdit. Иными словами, дисковый редактор использовался как hex-редактор. В UNIX, наоборот, hex-редакторы служат для редактирования диска.

Какой редактор выбрать? В общем-то, это дело вкуса (причем не только вашего, но еще и составителя дистрибутива). Одни предпочитают консольные редакторы hexedit или hexeditor (он же ncurses-hexedit) (рис. 2.5), другие тяготеют к графическим редакторам типа wxHexEditor (рис. 2.6).

Рис. 2.5. Внешний вид редакторов hexedit и hexeditor

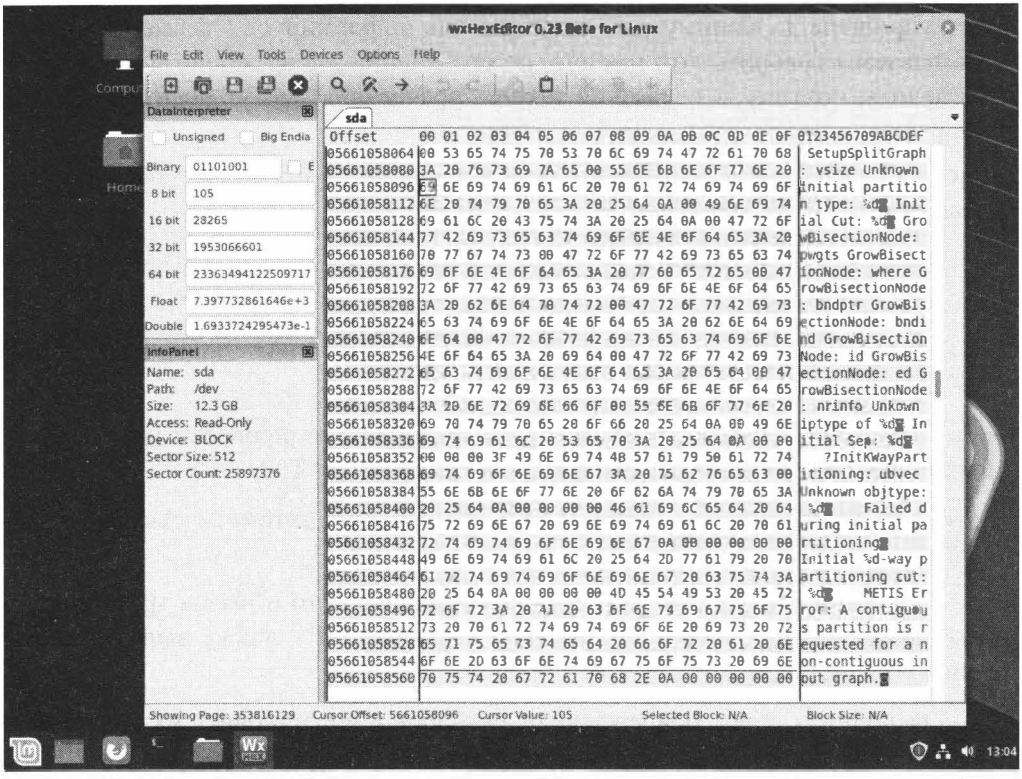


Рис. 2.6. Внешний вид редактора wxHexEditor

Автоматизированные дисковые утилиты

Более убогой утилиты, чем chkdsk (рис. 2.7) — стандартный дисковый "доктор", входящий в штатный комплект поставки Windows еще со времен Windows 95, — по-видимому, не придумать даже сценаристам из Голливуда. Система диагностики

```
CHKDSK is verifying files (stage 1 of 3)...
File verification completed.
CHKDSK is verifying indexes (stage 2 of 3)...
Index verification completed.
CHKDSK is verifying security descriptors (stage 3 of 3)...
Security descriptor verification completed.

  23551289 KB total disk space.
   4336872 KB in 7463 files.
      2168 KB in 484 indexes.
         0 KB in bad sectors.
     74733 KB in use by the system.
     65536 KB occupied by the log file.
  19137516 KB available on disk.

      4096 bytes in each allocation unit.
   5887822 total allocation units on disk.
   4784379 allocation units available on disk.
```

Рис. 2.7. Утилита chkdsk за работой

ошибок упрощена до минимума — доктор лишь информирует о факте их наличия, но отказывается говорить, что именно, по его мнению, повреждено и что он собирается лечить, поэтому последствия такого "врачевания" могут носить фатальный характер.

В Windows 10 эта консольная утилита также присутствует, ее синтаксис в общем виде выглядит следующим образом:

```
CHKDSK [том:[[путь]имя_файла]] [/F] [/V] [/R] [/X] [/I] [/C] [/L[:размер]]
```

Здесь

- `том` — определяет метку тома проверяемого диска, точку подключения, либо имя диска с двоеточием (например, C:);
- `путь, имя файла` — имя файла или группы файлов для проверки на фрагментацию. Используется только в файловой системе FAT/FAT32;
- `/F` — выполнение проверки на наличие ошибок и их автоматическое исправление;
- `/V` — в процессе проверки диска выводить полные пути и имена хранящихся на диске файлов. Для дисков, содержащих разделы NTFS, также выводятся сообщения об очистке;
- `/R` — выполнить поиск поврежденных секторов и восстановить их содержимое. Требует обязательного использования ключа /F;
- `/X` — в случае необходимости выполнить отключение тома перед его проверкой. После отключения все текущие дескрипторы для данного тома будут недействительны. Требует обязательного использования ключа /F;
- `/L:размер` — в ходе проверки изменить размер файла журнала до указанной величины (в килобайтах). Если значение не указано, выводится текущий размер файла. Используется только в файловой системе NTFS;
- `/I` — не проводить строгую проверку индексных элементов. Используется только в файловой системе NTFS;
- `/C` — не проводить проверку циклов внутри структуры папок. Используется только в файловой системе NTFS.

Однако просто так запустить `chkdsk` из интерфейса Windows, скорее всего, не получится: система "ругнется" на то, что диск занят другим процессом, или у вас недостаточно прав для запуска утилиты. Однако есть возможность запустить проверку диска при следующей загрузке системы, для чего в Windows 10 можно воспользоваться консольной командой `chkntfs`, которая имеет следующий синтаксис:

```
CHKNTFS том: [...]
CHKNTFS /D
CHKNTFS /T[:время]
CHKNTFS /X том: [...]
CHKNTFS /C том: [...]
```

Здесь

- том — определяет метку тома проверяемого диска, точку подключения либо имя диска с двоеточием (например, C:);
- /D — включается стандартный режим проверки дисков данной программой: проверка дисков осуществляется каждый раз при загрузке компьютера, в случае обнаружения ошибок запускается chkdsk;
- /T[:время] — позволяет изменить значение параметра AUTOCHK (в секундах), управляющего промежутком времени перед началом автоматической проверки дисков (в течение этого времени программа ведет посекундный обратный отсчет). Если временной промежуток не задан, демонстрируется текущее значение параметра AUTOCHK;
- /X — запрещает выполнять стандартную проверку дисков при загрузке. Данные об исключенных ранее из списка проверки дисках при этом утрачиваются;
- /C — запрашивает разрешение на проверку дисков при следующей загрузке компьютера. В случае обнаружения ошибок запускается программа chkdsk.

При вызове данной команды без каких-либо аргументов на экране отображается текущее состояние флага проверки данного диска.

Известны случаи, когда chkdsk "залечивал" до смерти полностью исправные разделы, поэтому обращаться за помощью к этой утилите нужно с осмотрительностью и осторожностью. Благо сейчас хватает более продвинутых программ для автоматической диагностики и "лечения" дисков, о которых мы подробнее поговорим далее в этой главе.

В мире UNIX проверка целостности файловой системы обычно осуществляется программой fsck (аналог chkdsk под Windows), представляющей собой консольную утилиту, практически лишенную пользовательского интерфейса и входящую в штатный комплект поставки любого дистрибутива. Как и любое другое полностью автоматизированное средство, она не только лечит, но, случается, что и калечит, так что пользоваться ею следует с большой осторожностью.

Victoria

По-латыни Victoria означает "победа", и в точности так же называется известнейшая компьютерная программа, предназначенная для "лечения" жестких дисков. Victoria разработана белорусским программистом Сергеем Олеговичем Казанским, написана на Ассемблере (благодаря чему имеет очень небольшой объем) и распространяется под свободной лицензией — ее можно бесплатно загрузить с сайта **http://hdd.by/victoria/**. Несколько версий Victoria были доработаны московским программистом Олегом Щербаковым: в программе были исправлены найденные ошибки, в частности баги при работе в 64-разрядных версиях Windows и "тормоза", возникавшие в процессе сканирования дисков объемом более терабайта. Программа полностью совместима со всеми актуальными версиями Windows, включая Windows 10 (рис. 2.8).

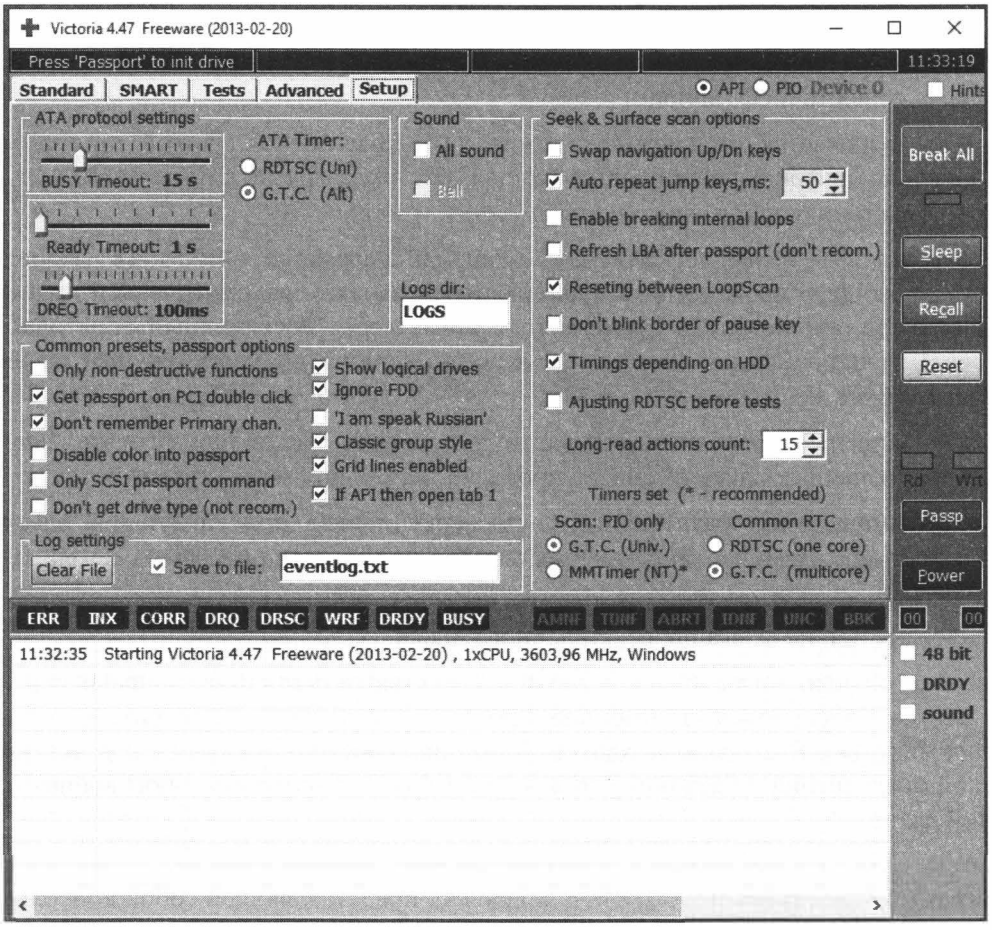


Рис. 2.8. Программа Victoria в Windows 10

Первые версии Victoria были написаны еще для MS-DOS, и эта система подходила для целей программы как нельзя лучше, поскольку в однозадачной ОС значительно проще обеспечить эксклюзивный доступ к диску. С появлением Windows Victoria обрела графический интерфейс и научилась использовать все возможности многозадачности. Однако, поскольку приложение работает с жесткими дисками на низком уровне, неопытные пользователи вместо восстановления данных рискуют и вовсе запороть винчестер, в связи с чем в современные версии приложения включено несколько видов защиты "от дурака".

Эта программа благодаря широчайшему набору функций считается общепризнанным лидером среди приложений для восстановления информации. Прежде всего, в Victoria встроен мощный сканер поверхности HDD, позволяющий обнаружить сбойные сектора, ошибки интерфейса и "плавающие" дефекты. Программа умеет автоматически настраивать тайм-ауты и определять размеры блоков, благодаря чему имеет возможность проверять диски на максимально возможной для них скорости. Есть возможность прогнать быстрый тест поверхности, в противовес полному сканированию диска, которое может занять до 4 часов. В обычном режиме, если

скорость обращения к какому-либо сектору слишком велика, контроллер диска считает такой сектор сбойным и автоматически заменяет его в адресной таблице на "исправный" сектор из резервной области. На практике автоматика иногда подводит, поэтому Victoria может проделать такую операцию (она называется Remap) принудительно. Кроме того, в ранних коммерческих версиях Victoria присутствовала функция Restore — она позволяла прочитать данные из сбойного сектора и перезаписать их в исправный сектор, но на протяжении уже нескольких лет эти коммерческие версии продукта не поддерживаются разработчиком.

Victoria — одна из немногих автоматизированных программ, способных работать с внешними накопителями, которые подключаются к компьютеру через интерфейс USB. По заверениям разработчика, приложение поддерживает до 90% моделей USB-SATA мостов, и эта цифра непрерывно увеличивается.

Большинство современных жестких дисков и твердотельных SSD-накопителей поддерживают технологию S.M.A.R.T. (Self Monitoring, Analysis and Reporting Technology) — это специальный аппаратно-программный механизм, позволяющий мониторить состояние жесткого диска и сообщать пользователю информацию о его "здоровье". Использование S.M.A.R.T. дает возможность предсказать появление технических сбоев или ошибок в работе HDD и своевременно заменить готовое прийти в негодность устройство или скопировать с него данные, чтобы потом не пришлось кусать локти. Victoria умеет работать со S.M.A.R.T. и анализировать S.M.A.R.T.-данные, она выводит информацию о состоянии и значении каждого параметра и показывает текущее "самочувствие" винчестера (рис. 2.9). Уход какого-либо значения в "красную зону" — верный признак того, что с диском что-то не так. В 2018 году разработчик добавил в программу поддержку S.M.A.R.T. для твердотельных SSD-накопителей.

Известно, что накопители с поддержкой технологии S.M.A.R.T. на протяжении своего жизненного цикла ведут специальные журналы, в которые записывается информация о состоянии устройства, и из которых можно почерпнуть много полезных данных. Вместе с тем большинство программ не умеет работать с данными логами, и даже не могут получить к ним доступ. Victoria к этому "большинству" не относится: она не только способна прочитать содержимое SMART-журналов, но и демонстрирует их пользователю в удобной и понятной форме.

Ряд современных накопителей поддерживает еще одну любопытную функцию — Device Configuration Overlay (DCO), позволяющую отключать ненужные функции, включать их при необходимости или менять доступный объем диска. С помощью этой технологии можно адаптировать накопитель к каким-либо нестандартным условиям использования, например подключить его к устройству, не поддерживающему диски больше определенного объема. Victoria умеет работать с DCO — считывать параметры конфигурации, изменять ее и сохранять в накопителе.

Помимо стандартного режима, подразумевающего использование при работе с диском Windows API, Victoria может взаимодействовать с накопителем в режиме PIO, т. е. обращаться к устройству IDE/SATA напрямую через порты, в обход Windows и BIOS. При этом программа реализует свой собственный физический протокол взаимодействия с устройством, что позволяет ей полноценно работать с неисправ-

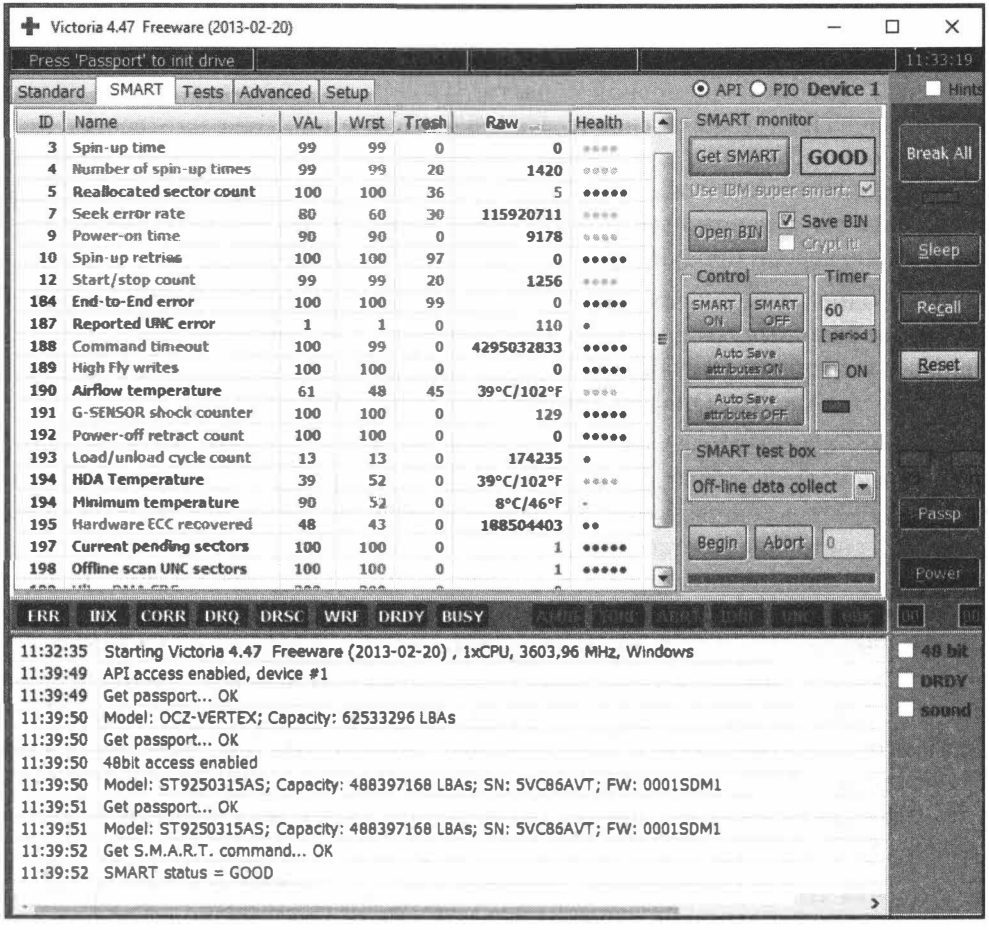


Рис. 2.9. Информация о S.M.A.R.T. в Victoria

ными накопителями и получать доступ к жесткому диску, который по всем признакам выглядит "мертвым". Однако скорость с диском в режиме PIO невелика, поскольку в этом случае не поддерживается DMA и некоторые функции чтения-записи. Обычно такой режим предназначен для реализации низкоуровневых операций с диском. Если режим PIO недоступен (обычно это случается в 64-разрядных версиях Windows), рекомендуется создать загрузочную флешку с 32-разрядной версией WinPE, скопировать на нее Victoria, подключить восстанавливаемый диск к шине SATA/PATA, затем в BIOS переключить режим работы SATA-контроллера в Native или IDE, после чего загрузиться в WinPE и запустить Victoria.

Одна из ключевых особенностей Victoria заключается в том, что в режиме PIO эта замечательная программа позволяет сбросить заданный пользователем в BIOS или с помощью сервисных утилит пароль доступа к диску или установленный производителем мастер-пароль. Основная проблема заключается в том, что при удалении утилиты, с помощью которой был запаролен диск, или при установке такого диска в другое устройство (прежде всего это касается ноутбуков) разблокировать диск не получится, даже если пользователь введет правильный пароль. Причина кроется

в том, что длина ATA-пароля всегда составляет 32 байта и недостающие символы утилиты добавляют "от себя", причем каждая из них — свои собственные. Некоторые символы (в частности, код 00h) вообще невозможно ввести с клавиатуры, а ряд ноутбуков еще и шифрует пароль перед пересылкой его значения на контроллер. Если ноутбук сгорит, а на диске остались ценные данные, вытащить их оттуда простой перестановкой накопителя в другой ноут (даже аналогичной модели) станет невозможно. Если вы столкнулись с подобной ситуацией, выйти из нее поможет Victoria.

Кроме всего прочего, Victoria может помочь в восстановлении дисков с нарушенной геометрией, поскольку в режиме PIO она умеет работать со специальной областью данных, где хранится информация о диске, включающая число блоков и объем накопителя — HPA (Host Protected Area).

Как и дисковые редакторы, речь о которых шла в предыдущем разделе, Victoria позволяет просматривать содержимое секторов диска и менять его. Эта программа подойдет даже для диагностики состояния физических интерфейсов, разъемов и проводов. Самым заметным недостатком бесплатной версии Victoria является, пожалуй, схема ее монетизации — современные версии приложения при запуске втихаря устанавливают в Windows кучу различных игр и ненужного софта, вследствие чего программа иногда детектируется антивирусами как неблагонадежная.

MHDD

MHDD — еще один общепризнанный инструмент для починки сбойных жестких дисков (рис. 2.10). Программа распространяется бесплатно, ее можно загрузить с сайта разработчика, расположенного по адресу **http://ihdd.ru/mhdd**. На этом же

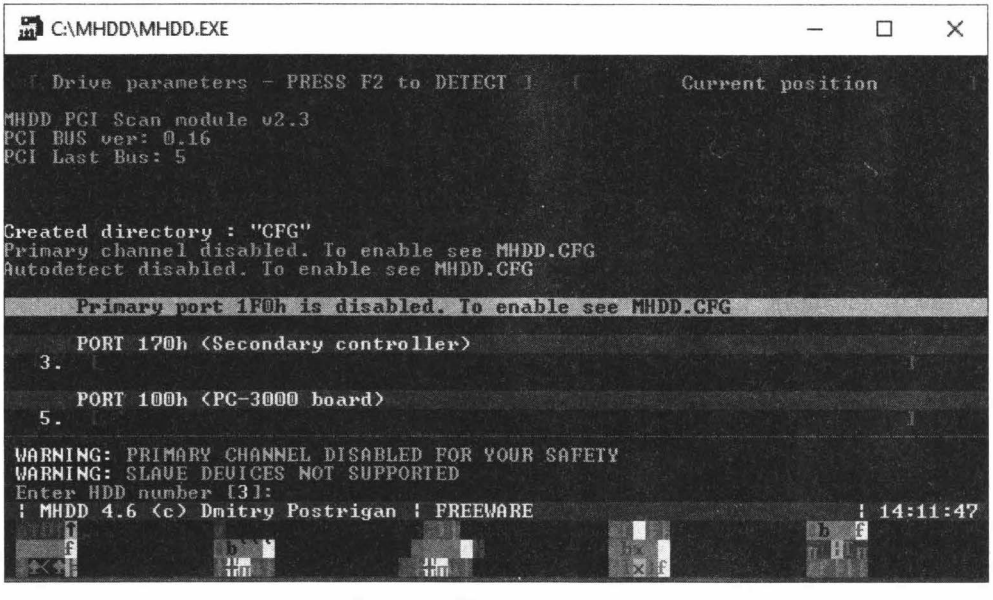


Рис. 2.10. Программа MHDD

сайте можно найти подробную документацию по данному продукту на русском языке.

Программа предназначена для работы с жестким диском на низком уровне, и ее первая версия увидела свет еще в 2000 году. Со временем возможности утилиты значительно выросли, и сейчас MHDD позволяет проделывать с диском практически все то же, что и Victoria, — читать содержимое произвольных секторов, работать со S.M.A.R.T, параметрами геометрии диска, паролями, и т. д. Единственное отличие заключается в том, что MHDD не может похвастаться столь же красивым и удобным графическим интерфейсом. Из ограничений следует отметить тот факт, что утилита не поддерживает интерфейс AHCI, поэтому режим работы жесткого диска придется настроить в BIOS. Также MHDD, в отличие от Victoria, по умолчанию не умеет работать с внешними USB-дисками, но этот дефект можно побороть с помощью драйвера USBASPI, эмулирующего режим работы USB-устройств через SCSI.

> ***Примечание***
>
> В наследство от DOS программе MHDD достался различный алгоритм работы с первичными и вторичными дисками (master/slave), подключенными к шине IDE. По умолчанию MHDD не умеет работать со slave-дисками, поэтому такой винчестер нужно подключить к компьютеру как master, либо немного поправить файл `CFG\ mhdd.cfg`, отредактировав параметр `PRIMARY_ENABLED=TRUE`.

Несмотря на то что утилита MHDD пережила несколько обновлений, программа заметно проигрывает по своим возможностям Vicroria, а угробить с ее помощью диска гораздо проще. Да и интерфейс в стиле MS-DOS выглядит в 2020 году более чем архаично.

GetDataBack

Еще одна утилита от создателей Disk Explorer. Она автоматизирована полностью и не предоставляет никаких возможностей ручной настройки, зато список поддерживаемых файловых систем не ограничивается лишь FAT и NTFS (рис. 2.11). Скачать программу можно с сайта **http://www.runtime.org/data-recovery-software.htm**.

Программа сканирует MFT и выводит все файлы, которые только удалось найти (включая удаленные), распределяя их по каталогам (при условии, что соответствующие индексы не повреждены). Зато она поддерживает удаленное восстановление, создание образов дисков, а также мощную систему поиска по файлам (дата/размер). Возможность поиска по содержимому, к сожалению, отсутствует, что не может не разочаровывать. Представьте себе, что вы хотите восстановить файл со своей диссертацией, ключевые слова которой вам известны, а вот в каких секторах они располагаются — неведомо. То же самое относится и к поиску файла записной книжки с телефоном приятеля. Тем не менее для большинства рядовых задач по восстановлению возможностей GetDataBack хватает с лихвой. Важно отметить, что GetDataBack не является доктором, она не "лечит" разделы, а всего лишь позволяет скопировать из них уцелевшие файлы.

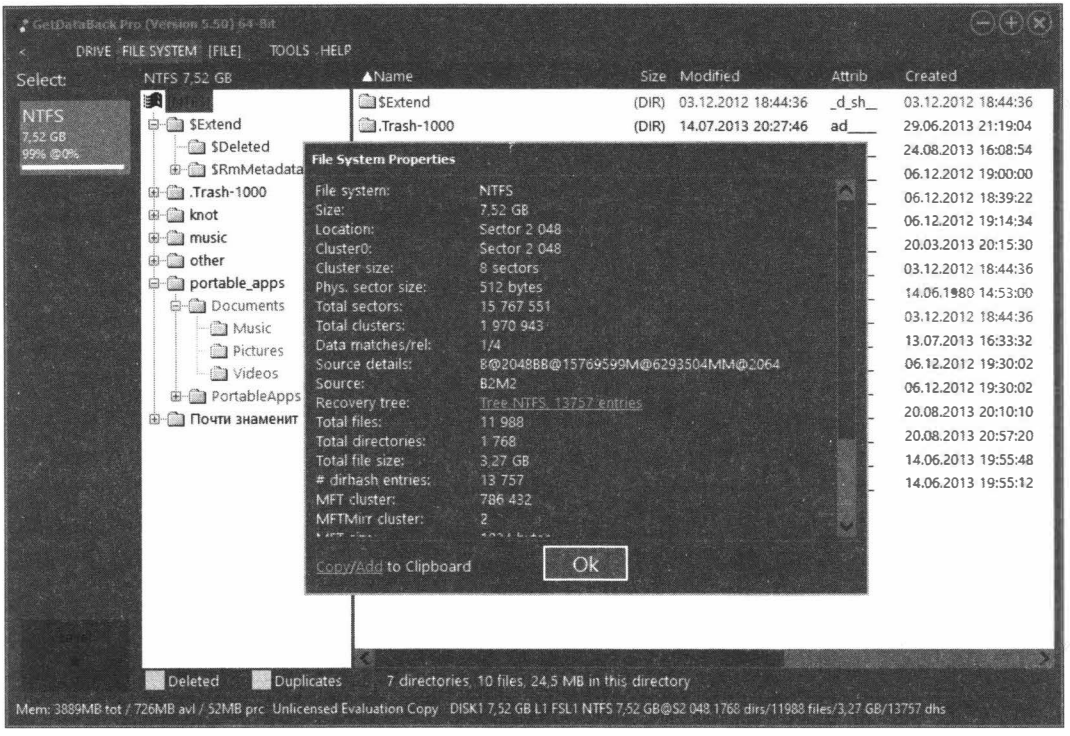


Рис. 2.11. Внешний вид GetDataBack

Easy Recovery

На первый взгляд, эта широко разрекламированная утилита от компании OnTrack Data Recovery (**http://www.ontrack.com**) кажется весьма многообещающей (рис. 2.12). Внешность, однако, обманчива.

Это коммерческий продукт, имеющий бесплатную триальную версию, загрузить которую, однако, не так-то просто. Для бесплатного скачивания Easy Recovery у вас попросят зарегистрироваться, указав ваше имя, адрес электронной почты, страну проживания, номер телефона и название компании, в которой вы работаете (все поля обязательны для заполнения), пообещав выслать ссылку на загрузку по e-mail. Ссылка, однако, приходит далеко не всегда, даже если все поля вы заполните правильно. Зато поток рекламных писем с предложением купить коммерческую версию продукта гарантирован.

OnTrack Easy Recovery — достаточно простой в использовании инструмент, основное предназначение которого — поиск в автоматическом режиме удаленных файлов. Поэтому мы не рекомендуем вам эту утилиту для практического применения, за исключением, возможно, тех случаев, когда требуется восстановить только что отформатированный том, на который еще не было записано ничего существенного.

Рис. 2.12. EasyRecovery

Stellar Windows Data Recovery

Компания Stellar (**http://www.stellarinfo.com**) предлагает утилиту, носящую гордое имя Windows Data Recovery и предназначенную для восстановления данных. Как заявлено разработчиком, она поддерживает практически все популярные файловые системы, которые только известны на сегодняшний день (включая UFS).

Демонстрационную копию можно скачать по следующему адресу: **http://download.stellarinfo.com/StellarDataRecovery.exe**. Обратите внимание на расширение файла — это исполняемый файл для Windows. Потребуется устанавливать в систему дополнительный винчестер с работоспособной Windows и инсталлировать Windows Data Recovery поверх нее.

Программа обладает стильным пользовательским интерфейсом (рис. 2.13) и прекрасно работает в Windows 10.

Приложение предназначено для поиска удаленных файлов и может восстановить их, если поверх этих файлов система ничего не успела записать. Никакими другими функциональными возможностями приложение не обладает.

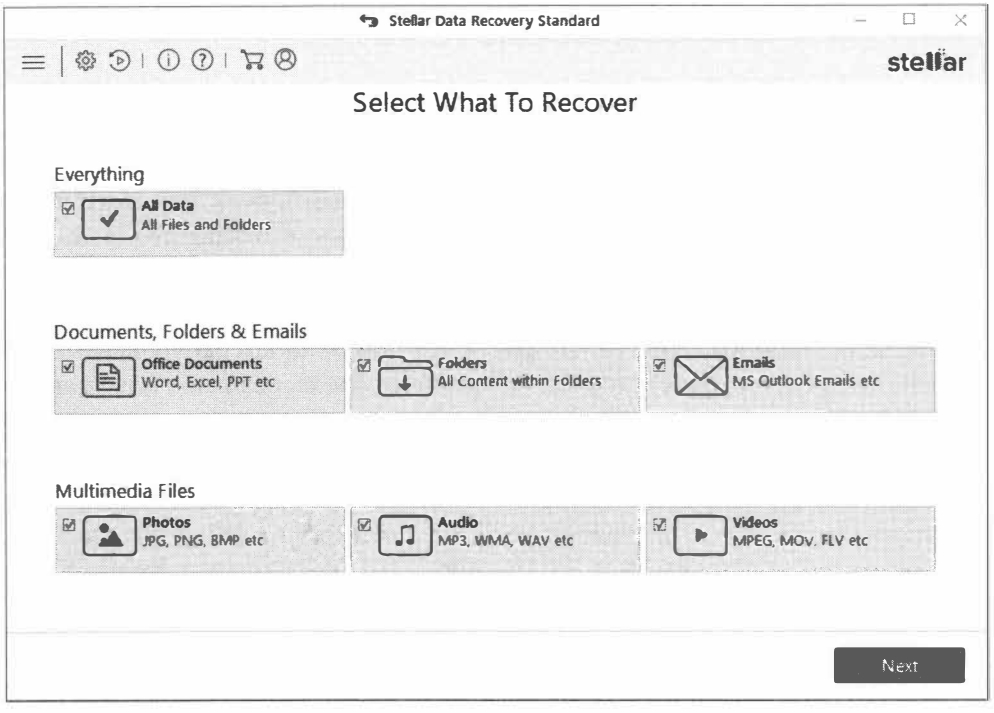


Рис. 2.13. Stellar Windows Data Recovery

The Sleuth Kit

Бесплатно распространяемый комплект утилит для ручного восстановления файловой системы, который можно найти по адресу **http://www.sleuthkit.org/**. Утилиты для работы с разделами жесткого диска можно скачать со странички **http://www.sleuthkit.org/sleuthkit/download.php**, там же выложены исходные коды. Все означенные программы работают исключительно в режиме командной строки, документация по их возможностям и использованию весьма скудная. Вместе с тем инструментарий регулярно обновляется: последняя версия The Sleuth Kit на момент написания этих строк датирована январем 2020 года.

Foremost

Это еще одна бесплатная утилита для восстановления удаленных файлов, основанная на формате их заголовков и на особенностях их структуры. Естественно, она работает только с теми файлами, строение которых ей известно. Кстати говоря, утилита взаимодействует с файловой системой не напрямую, а обрабатывает файлы, полученные командой `dd` или набором Sleuth Kit, благодаря чему она "поддерживает" все файловые системы. Последняя версия, датированная 2006 годом, находится на сервере **http://foremost.sourceforge.net/**.

Отладчики файловой системы

Отладчиками файловой системы называют утилиты, дающие доступ к святая святых файловой системы и позволяющие манипулировать ключевыми структурами данных по своему усмотрению. Чем они отличаются от простых редакторов? Редактор работает на более низком уровне — уровне блоков или секторов. Он, в принципе, может представлять некоторые структуры в наглядном виде, однако в их "физический" смысл никак не вникает.

Отладчик файловой системы работает через драйвер, поэтому испортить раздел с его помощью намного сложнее. Он реализует довольно высокоуровневые операции, такие как установка и снятие флага занятости блока, создание новой символьной ссылки, просмотр индексных дескрипторов с соответствующими им файлами и т. д. А вот "посекторного" hex-редактора отладчики файловой системы обычно не содержат, поэтому обе категории программ взаимно дополняют друг друга.

Большинство (если не все) дистрибутивов Linux включают в себя отладчик `debugfs`, поддерживающий ext2fs, ext3fs и ext4fs (рис. 2.14). Для отладки ReiserFS можно использовать утилиту `debugreiserfs`.

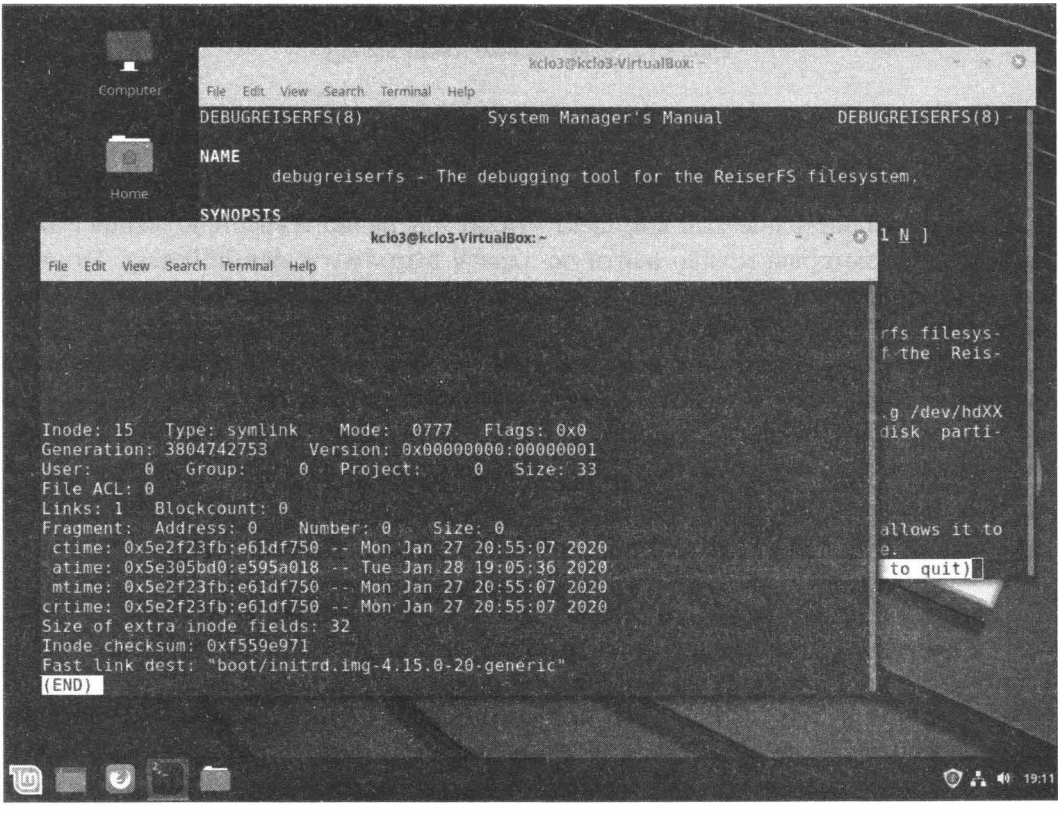


Рис. 2.14. Отладчик файловой системы `debugfs` за работой

Необходимое техническое оборудование

Непременным атрибутом серьезной фирмы была и остается "чистая комната", обеспечивающая класс чистоты 100. Это означает, что в одном кубическом футе воздуха не может содержаться более 100 пылинок размером 0,5 мкм каждая. За этими незатейливыми словами скрывается грандиозное инженерное сооружение стоимостью от 30 тыс. долларов. Конструкция типовой "чистой комнаты" показана на рис. 2.15. Менее серьезные ремонтники ограниваются "чистой камерой", что на порядок дешевле. Однако для кустарных мастеров даже это неподъемно дорого. Можно ли обойтись без чистой комнаты или соорудить ее самостоятельно?

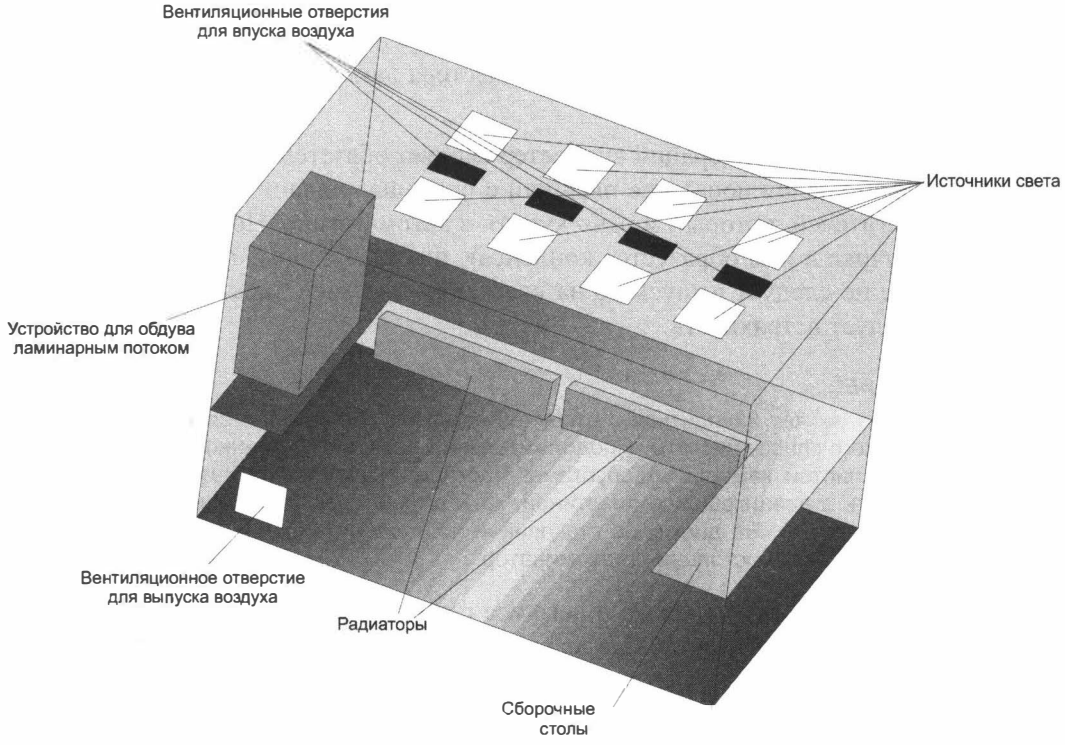


Рис. 2.15. Схематичное устройство типовой "чистой комнаты"

Вопреки распространенным слухам и опасениям — да, можно! Как минимум достаточно обыкновенной незапыленной комнаты с работающим кондиционером или даже без него. Кроме того, желательно обзавестись ионизатором (ионизатор вызывает слипание частичек пыли, и они вместо того, чтобы носиться по комнате, оседают на пол, откуда их удаляет нехитрая система вентиляции). Хороший ионизатор стоит достаточно дорого, но при желании его можно сконструировать и самостоятельно. Взять хотя бы ту же "Люстру Чижевского", схему которой легко найти в Интернете. Естественно, непосредственно перед проведением работ ионизатор нужно выключать.

Если вы занимаетесь ремонтом винчестеров более или менее постоянно, имеет смысл соорудить некоторое подобие чистой камеры. Для этого потребуется стеклянный аквариум, воздушный фильтр и компрессор, нагнетающий воздух внутрь аквариума и препятствующий попаданию пыли через открытую переднюю стенку. Передняя стенка при этом остается открытой! Аквариум ставится на бок, открытой стороной на себя. Сверху закрепляется стеклянная пластина, закрывающая до 2/3 поверхности, а внутрь устанавливается воздушный фильтр. Компрессор остается снаружи. Оставшаяся 1/3 закрывается другой пластинкой, после чего на несколько часов включается фильтр (точное время зависит от его пропускной способности и объема аквариума), а затем, перед началом работ, эта пластинка удаляется, предоставляя простор рукам. Невероятно дешево, но достаточно чисто. Во всяком случае, намного чище открытой жилой комнаты. Учитывая непродолжительное время вскрытия гермозоны, на пластины успевает осесть не так уж много пыли, и у вас есть все шансы считать с винчестера данные прежде, чем он окончательно прикажет долго жить.

После выполнения всех операций винчестер следует обязательно закрыть крышкой, предварительно удалив попавшие пылинки с помощью баллончика с воздухом для продувки двигателей, который можно купить в автомагазине. При длительном хранении баллончика в нем образуется конденсат, поэтому первые порции воздуха ни в коем случае не следует выпускать на восстанавливаемый диск. Кроме того, баллончик не следует встряхивать.

> ***Внимание!***
>
> **Накопитель может находиться с открытой крышкой только при условии обеспечения надлежащего класса чистоты. Продолжительная работа с "оголенной" гермозоной вне пределов чистой камеры недопустима! Частицы пыли, присутствующие в воздухе, сталкиваясь с вращающейся пластиной, за короткий срок уничтожают магнитное покрытие (рис. 2.16). На дисках со стеклянной подложкой (например, винчестерах типа DTLA) образуется настоящий "иллюминатор".**

Но ведь при вскрытии гермоблока в него все равно попадает пыль! Разве после закрытия крышки она исчезнет? На самом деле внутри гермоблока расположен

Рис. 2.16. Для жесткого диска каждая пылинка равносильна метеориту

фильтр рециркуляции, активно поглощающий попавшую пыль, в результате чего ее концентрация быстро уменьшается до приемлемых значений. Если же гермоблок остается открытым, то концентрация пыли остается постоянной. Еще одна причина состоит в том, что закрепленная крышка слегка деформирует гермоблок, поэтому без нее диск может читаться нестабильно, с многократными повторами. Таким образом, установка крышки жизненно важна. Запустив утилиту, выводящую скоростную кривую на экран, попеременно подтягивайте болты, добиваясь наиболее ровного графика чтения.

Как уже говорилось, часы жизни винчестера, вскрытого вне чистой комнаты, сочтены. При этом время, требующееся для вычитки данных, — велико. Поэтому жесткий диск лучше подключать к компьютеру напрямую и в первую очередь считывать только самые важные данные, установив счетчик повторов чтения на значение $3x$. То есть сначала читаем все, что читается само, и только затем — то, что читается с трудом.

Кроме наличия "чистой комнаты", еще одним козырем серьезных фирм являются аппаратно-программные комплексы. Наибольшую известность получил PC-3000 от ACE Lab (**http://www.acelab.ru**), имеющий на текущий момент вариации, ориентированные на работу с флеш-накопителями, SSD, и портативную версию, удобную для выездной работы с ноутбуком.

Что же представляют собой аппаратно-программные комплексы по восстановлению данных? С "аппаратной" точки зрения это обыкновенный контроллер IDE/SATA, поддерживающий режимы PIO и UDMA, оборудованный встроенным электронным ключом, позволяющим подключать и отключать жесткие диски "на лету", без выключения компьютера, что очень удобно. Однако того же эффекта можно достичь, если подсоединить жесткий диск к отдельному блоку питания, а перед его выключением подать ATA-команду `94h` (standby immediate). Портативный аппаратно-программный комплекс PC-3000 Portable III показан на рис. 2.17. PC-3000 Flash позволяет чи-

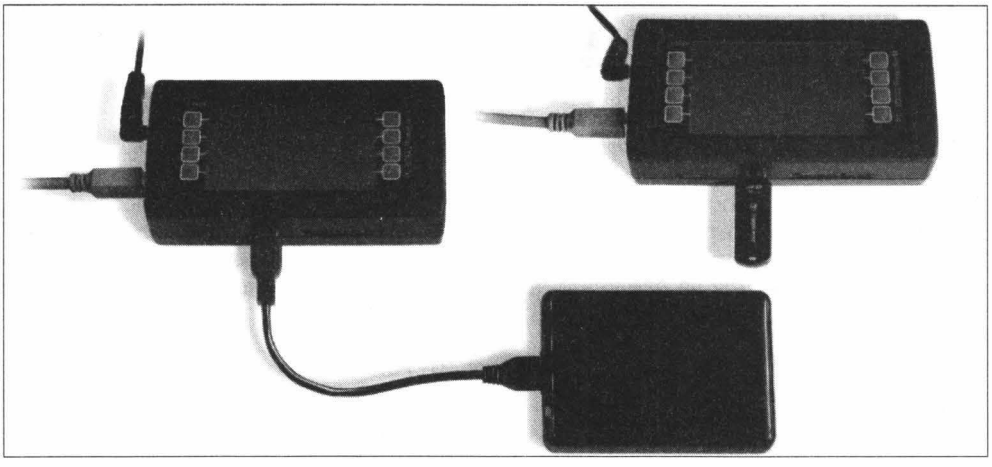


Рис. 2.17. Аппаратно-программный комплекс PC-3000 Portable III, поддерживающий USB-накопители (фото с сайта производителя **acelab.ru**)

тать содержимое памяти на основе NAND напрямую, что дает надежду на спасение данных, когда сам контроллер накопителя уже не спасти, а версия PC-3000 для SSD способна помочь при восстановлении таблицы трансляции твердотельных накопителей, использующих аппаратное шифрование.

Технологические команды, приоткрывающие дверь во внутренний мир накопителя, передаются либо по ATA-интерфейсу, либо через COM-терминал. Да-да! На многих моделях винчестеров имеется интегрированный COM-порт, подключившись к которому можно контролировать процесс инициализации и управлять приводом (правда, не на всех дисках он распаян, т. е. выведен на разъем). Обычного COM-порта, встроенного в компьютер, плюс пары переходников, которые любой радиолюбитель легко смастерит самостоятельно, для наших целей вполне достаточно.

Кроме того, к комплексу прилагается мощное программное обеспечение, в частности Data Extractor, отличительной чертой которого является способность автоматического восстановления транслятора (в *главе 5* мы об этом еще поговорим), а также продуманный механизм "вычитывания" информации. Если сектор прочитался, он заносится в базу. В дальнейшем такой сектор никогда не читается с диска повторно (если только пользователь не даст команду сделать это), а всегда берется из базы.

Большинство распространенных утилит ведут себя совсем не так. Они многократно перечитывают одни и те же сектора, особенно сектора, принадлежащие служебным областям диска, например FAT или MFT, или даже попросту завершают свою работу при встрече с bad-сектором. В случае логических разрушений это нормальный подход. Однако для восстановления физически поврежденных жестких дисков он непригоден. Тем не менее аппаратно-программный комплекс не панацея! Специалист, умеющий ремонтировать жесткие диски, при необходимости обойдется и без специализированного аппаратно-программного комплекса. С другой стороны, людям, не обладающим необходимыми знаниями и навыками, никакой комплекс ничем не поможет.

Из других инструментов нам в первую очередь понадобятся отвертки-звездочки. Как минимум номер 10, номер 9, для 2.5-дюймовых винчестеров нужны и более мелкие номера.

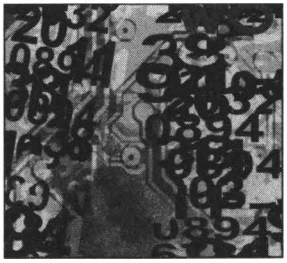
> ***Примечание***
>
> При отсутствии звездочек можно воспользоваться и обыкновенной плоской отверткой. В частности, звездочка-10 соответствует плоской-3. Под звездочку-9 отвертку придется затачивать самостоятельно. На практике, однако, пользоваться плоскими отвертками не рекомендуется — шлицы срываются, и потом их придется высверливать. Тем более что сейчас звездочки уже не проблема, и приобрести их можно в любом техническом магазине или на AliExpress. Короче, будем считать, что мы ничего вам не говорили.

Остальной инструментарий вполне стандартен. Пассатижи, плоскогубцы, пинцеты. Для перестановки "блинов" придется собрать специальный захват, устройство и приемы работы с которым можно найти на YouTube.

В процессе ремонта жестких дисков, флешек и твердотельных SSD-накопителей вам придется демонтировать микросхемы. Для этого нужен либо специальный (или

строительный) фен, либо паяльник плюс фантазия. При этом феном еще необходимо научиться пользоваться. Паяльная станция — это, конечно, не фен, но принципы работы с ней схожи. Если фена нет, то можно обойтись паяльником с расплющенным жалом, лезвием (для демонтажа планарных микросхем) и медицинской иглой со сточенным концом (для демонтажа элементов, установленных в отверстия со сквозной металлизацией). О методиках демонтажа микросхем с печатных плат снято множество роликов, которые без труда можно найти на YouTube, а весь необходимый инструментарий — приобрести в специальных магазинах электроники или на AliExpress. Здесь важно проявлять аккуратность и не спешить, т. к. повредить микросхему очень легко, а вот найти ей замену далеко не всегда представляется возможным.

ГЛАВА 3



Системы резервного копирования данных

Не зря говорят: предупрежден — значит вооружен. Предупредив потерю данных, вы будете бесконечно благодарны себе, если с вашими основными накопителями случится неприятность. Вам не придется в панике заниматься ликвидацией этой проблемы, не имея даже стопроцентной уверенности в успешности исхода. Хочется верить, что эта глава подвигнет вас заняться созданием резервных копий важных данных, если вы по какой-либо причине (особенно если это лень) до сих пор этого не сделали.

Основы резервирования информации

Существует такое правило: если у вас данные имеются в одном-единственном экземпляре, то можно считать, что у вас их нет (а если у вас один бэкап — у вас нет бэкапа). Создание резервных копий важных файлов должно войти в привычку и стать столь же естественным действием, как мытье рук перед едой! А храниться они должны независимо от машины, с которой снимались, но реализовать это, к сожалению, не всегда легко.

Представьте себе, что вы настроили копирование на внешний или сетевой диск по расписанию. Тут вдруг внезапно компьютер оказывается заражен шифровальщиком, и многие файлы на нем, соответственно, будут без выкупа не доступны. Но ведь у нас есть резервная копия! Правда, она находится на диске, подключенном к компьютеру, а потому с ненулевой вероятностью тоже была зашифрована... И что же теперь делать? А вот что. Помимо обычных ежедневных копий следует создавать также автономные "офлайновые" копии. И это действительно убережет ваши данные от участи стать жертвой энкодеров.

Другой важный вопрос — это выбор подходящего носителя для надежного хранения резервных копий. Ни в коем случае не храните копию на соседнем разделе того же винчестера или SSD! Иначе ваша копия "накроется" так же, как и оригинал, если у накопителя возникнут неисправности. Выбор носителя для сохранения запасных данных зависит от цели резервирования. А они при организации резервного копирования могут быть совершенно разные, например:

- создание архива. Это может быть архив фотографий, книг, фильмов, музыки, различных программ и вообще чего угодно;
- сохранение промежуточных результатов работ. Обидно случайно остаться без целой главы дипломной работы (а еще обиднее — без исправлений какой-нибудь из глав этой книги...), над которой вы корпели полдня, из-за перебоев с питанием или багов операционной системы (а Windows 10, надо сказать, успела прославиться и удалением пользовательских файлов после неудачных пакетов обновлений);
- клонирование устройства и снятие его образа;
- резервное копирование конфигураций и данных серверов. Это, кстати, не раз спасало крупные компании, предоставляющие облачные сервисы;
- синхронизация данных. Вообще говоря, это отдельная задача, но она тесно связана с резервным копированием.

Архивы все еще нередко хранят на оптических дисках, причем для этого используются не только уже давно всем привычные CD/DVD, но и намного более емкие диски Blu-Ray. Такой выбор можно понять: современные оптические диски имеют в целом достаточный объем для хранения множества фотографий и документов, а сами носители можно вполне компактно хранить в специальных коробках или банках. Главное, чтобы это происходило в темном месте без резких перепадов температур, ведь от воздействия света отражающий слой деградирует, и со временем оборудование не сможет различить 0 от 1 по уровню сигнала. Следует обращать внимание на состав красителя и отражающего слоя болванки: некоторые диски исправно читаются и спустя пару десятков лет после записи, а иные из отдельных партий не прочесть уже через пару лет из-за неудачного состава. Немало огорчений принесло коллекционерам раритетных музыкальных изданий "бронзовение" дисков (CD bronzing), происходящее из-за ошибок на этапе производства. Такая безрадостная судьба ожидала партию альбома Rolling Stones, изготовленную в 90-х. Подробнее проблемы оптических носителей рассматриваются в *главе 11*.

В более или менее серьезной организации резервного копирования в серверной инфраструктуре обычно участвуют сетевые хранилища или FTP-серверы. Но, как уже было сказано, при нападении трояна-шифровальщика это мало чем сможет помочь, поскольку вполне возможно, что он без проблем зашифрует и содержимое присоединенных сетевых папок. В крупных же компаниях для хранения "офлайновых" резервных копий могут использоваться... что бы вы думали? Магнитные ленты! Именно кассеты спасли данные электронной почты 0,02% пользователей Google (основываясь на оценке BBC, получается несколько десятков тысяч аккаунтов) в 2011 году, в то время как остальные копии в нескольких дата-центрах были испорчены из-за багов ПО.

Оптимальный вариант для простого пользователя — внешний жесткий диск (и лучше не один!). Можно, в общем-то, использовать и SSD, но это несколько расточительно по отношению к его ресурсам: твердотельные накопители создавались, чтобы ускорить загрузку программ и ОС, а не затем, чтобы их время от времени подключали и что-то на них скидывали. К тому же время хранения информации на

обесточенном SSD — все еще обсуждаемый вопрос без однозначного ответа (в среднем советуют подключать SSD хотя бы раз в 6 месяцев). Флешки же могут сгодиться для хранения и переноса промежуточных документов.

В любом случае хоть какая-то копия — в сотни раз лучше, чем никакой. А еще лучше иметь несколько бэкапов, причем хранящихся в разных местах. Если компьютер, с которого снималась копия, и сама копия хранятся в одном здании или даже помещении, то в случае пожара, затопления и прочих бедствий может быть уничтожено и то и другое. Но если администратор позаботился о том, чтобы сохранить еще одну копию в другом месте, сил на восстановление ему придется потратить намного меньше, не сделай он этого. Ведь это в некотором плане простейший способ распределенного хранения информации, а в вопросе сохранности распределенные хранилища куда надежнее централизованных. Некоторые компании предоставляют свои ресурсы как раз для надежного удаленного хранения резервных копий. RAID-массивы и сети хранения данных (СХД) рассматриваются в *главе 12*.

Когда настраивается схема автоматического резервного копирования по расписанию, важно решить, будет ли это каждый раз полный слепок содержимого или только так называемая дельта — то, что изменилось за время, прошедшее с момента последней полной копии. По объему сохраняемой информации чаще всего встречаются три варианта:

- полное копирование — когда создается полная копия всех выбранных данных;
- инкрементное копирование — когда в копию добавляются только файлы, измененные с момента любого последнего копирования;
- дифференциальное — добавление в архив только данных, изменившихся с момента сохранения последней полной копии.

Важный момент при этом — дедупликация. Если создается копия нескольких виртуальных машин или же несколько последовательных бэкапов одной и той же системы, где с высокой вероятностью будут одинаковые файлы, то избавление от дублей поможет сохранить многие гигабайты резервного хранилища. Дедупликация позволяет сохранять только уникальные фрагменты данных, избегая расточительной траты запасного пространства.

Другой аспект политики резервирования — ротация бэкапов. Иногда очень полезно хранить не только самый последний бэкап, но и несколько предшествующих: бывает так, что пользователь удалил какой-нибудь файл (как обязательно выяснилось впоследствии, непомерно важный), после чего был создан бэкап, не содержащий его. Схема ротации определяет, как удаляются старые бэкапы: как правило, по их количеству или по возрасту. Когда создается очередная копия, то обычно последняя становится предпоследней и т. д., а самая старая удаляется (подобно тому как работает ротация файлов логов). Важно решить, сколько копий одновременно хранить, и планировать резервное хранилище соответствующего объема.

Не нужно жалеть времени на создание резервной копии перед любыми действиями, в успехе которых нельзя быть абсолютно уверенным: обновление ПО, системных конфигураций, установка патчей и т. п. Проверка корректности резервной копии

(а заодно периодические "учения" по восстановлению) тоже важна, чтобы в ответственный момент не оказалось, что бэкап есть, но пользы от него нет. Если же говорить об устройстве для хранения бэкапа, то в зависимости от задачи это могут быть локальные носители (диски, флешки, магнитные ленты — стримеры), сетевые ресурсы или облачные сервисы.

Ручное резервное копирование и облака

Это, конечно же, самый простой способ создания резервной копии: скопировал нужные файлы на другой носитель или в облако, и дело сделано. Главная проблема здесь состоит в том, что иногда хочется понадеяться на авось и не делать копию, ведь ну сегодня-то точно ничего не произойдет! И по закону подлости с оригиналом обязательно что-нибудь случится именно в этот день.

Но не будем о грустном. Если нам позарез нужна копия архива фотографий и только его, то мы просто сохраним их на нужное устройство или в облако, не тратя времени на развертывание системы резервного копирования. Облачные сервисы вообще очень удобны для синхронизации данных между устройствами и позволяют легко обзавестись идентичными копиями сразу на всех устройствах.

Подобные способы часто подкупают своей простотой. Но это оказывается удобным лишь до поры до времени, пока не требуется ставить создание объемных бэкапов на поток. Поэтому самое время перейти к рассмотрению утилит, автоматизирующих процесс резервного копирования.

Системы резервного копирования для Windows

Необходимость заранее делать резервные копии самых ценных файлов — вещь самоочевидная, у любого пользователя это должно войти в привычку, как мыть руки перед едой или надевать шапку на морозе. Правда, любителей отморозить уши с годами меньше не становится, а потому компании, специализирующиеся на восстановлении данных, в большинстве своем не бедствуют. Поистине золотой век для них наступил с началом массового распространения троянов-энкодеров, когда число пострадавших, чьи файлы оказались зашифрованы вредоносными программами, исчислялось тысячами. Сейчас эпидемии шифровальщиков вроде бы понемногу пошли на спад, но актуальность связанных с потерей ценных данных проблем от этого не снижается. Любая техника, к сожалению, может сломаться. А некоторая — еще и внезапно.

Очевидно и то, что копировать информацию нужно на внешние накопители, независимые от основного устройства хранения данных. Если просто сохранить файлы в соседнем разделе жесткого диска, он с высокой долей вероятности "накроется" вместе с винчестером, если у того возникнут проблемы с аппаратной частью или логикой. Раньше для этих целей использовались оптические диски, но они, во-первых, недолговечны, во-вторых, ограничены по объему и максимальному числу итераций записи, в-третьих, сама скорость обращения к оптическому накопителю

очень невелика, в-четвертых, пишущие приводы скоро можно будет отыскать разве что в музее, в-пятых... Да и хватит, пожалуй. Уже перечисленных причин достаточно для того, чтобы ответственно заявить: оптика — прошлый век, хотя очень многие компании (да и простые пользователи) до сих пор хранят архивы своих документов на DVD.

Флешки из-за незначительных объемов и низкой скорости работы с шиной USB для резервного копирования тоже подходят не слишком хорошо. Поэтому оптимальный вариант — внешний жесткий диск либо SSD-накопитель, облачное хранилище или удаленный FTP-сервер. Еще одним вариантом является NAS — Network-Attached storage, так называемое сетевое хранилище. По сравнению с внешним винчестером оно выигрывает за счет того, что не требует непосредственного подключения к компьютеру: NAS — сам по себе мини-компьютер, основная задача которого заключается в организации обмена данными по сети и записи их на диск или RAID-массив. Прошивка NAS, как правило, базируется на каком-либо сильно урезанном варианте Linux.

Устройство тихо живет себе в локальной сети, играя роль сетевого файлообменника, на который можно закачать что угодно, чем и удобно — к нему можно обратиться в любой момент, например автоматически скопировать на него файлы по расписанию или во время простоя рабочего компьютера. В этом кроется и один из недостатков NAS: большинство троянов-энкодеров прекрасно умеет шифровать содержимое присоединенных к Windows сетевых папок, да и в прошивке самих хранилищ иногда обнаруживаются уязвимости, которыми пользуются злоумышленники. Так, например, в 2014 году троян под названием Trojan.Encoder.737 шифровал файлы, размещенные на сетевых хранилищах производства одной известной компании, и требовал за их восстановление 350 долларов. Проникал он туда, используя уязвимость в прошивке девайса. Разработчик быстро залатал "дыру", но некоторое количество пользователей, не успевших вовремя обновиться, пострадали от этой угрозы.

Итак, представим себе, что место, где мы собираемся хранить резервные копии, у нас есть: это внешний диск, NAS или облако. Но одним из главных препятствий на пути внедрения резервного копирования в повседневную практику является элементарная лень. Действительно, нужно ведь отвлечься от любимых игрушек, вспомнить, какие файлы поменялись с момента создания последней резервной копии, загрузить их в хранилище... Напрашивается простое решение: максимально автоматизировать этот процесс с помощью подходящего ПО. И если в macOS есть, например, встроенное приложение для резервного копирования Time Machine, то разработчики Windows такого удобного инструмента в составе ОС не предусмотрели, в силу чего счастливым пользователям этой платформы приходится искать продукты сторонних разработчиков.

Иными словами, нам нужно найти подходящую программу — по возможности бесплатную, поддерживающую работу по расписанию, имеющую возможность сохранять копии в сети или в облаке и желательно включающую максимально возможное число типов резервного копирования (лучше всего все три). Итак, что нам предлагает рынок?

Acronis True Image

Про продукцию компании Acronis слышали, наверное, все. Логотипы этой фирмы красуются на болиде Формулы-1 российского пилота Даниила Квята, да и в Интернете рекламы Acronis хватает. Собственно, эта фирма на протяжении всей своей истории специализируется на разработке софта для резервного копирования и потому достигла на этом поприще определенных успехов. Для домашних пользователей компания предлагает программу Acronis True Image, распространяющуюся как в виде лицензионной версии, рассчитанной на одну установку, так и по подписке.

В возможности базовой версии приложения входит копирование образа диска или выбранных файлов в указанное пользователем хранилище, восстановление данных из резервной копии, клонирование дисковых разделов и защита от шифровальщиков. Последняя предусматривает отслеживание запущенных в системе процессов и предупреждение пользователя при попытке шифрования файлов: в случае опасности пользователю будет предложено заблокировать потенциально опасный процесс. Покупателям подписки становится также доступно облачное хранилище на серверах Acronis объемом 250 Гбайт (или 1 Тбайт, в зависимости от того, насколько ты готов раскошелиться).

Сама по себе программа в версии от 2020 года может похвастаться понятным и удобным пользовательским интерфейсом (рис. 3.1): выбираешь локальный диск и

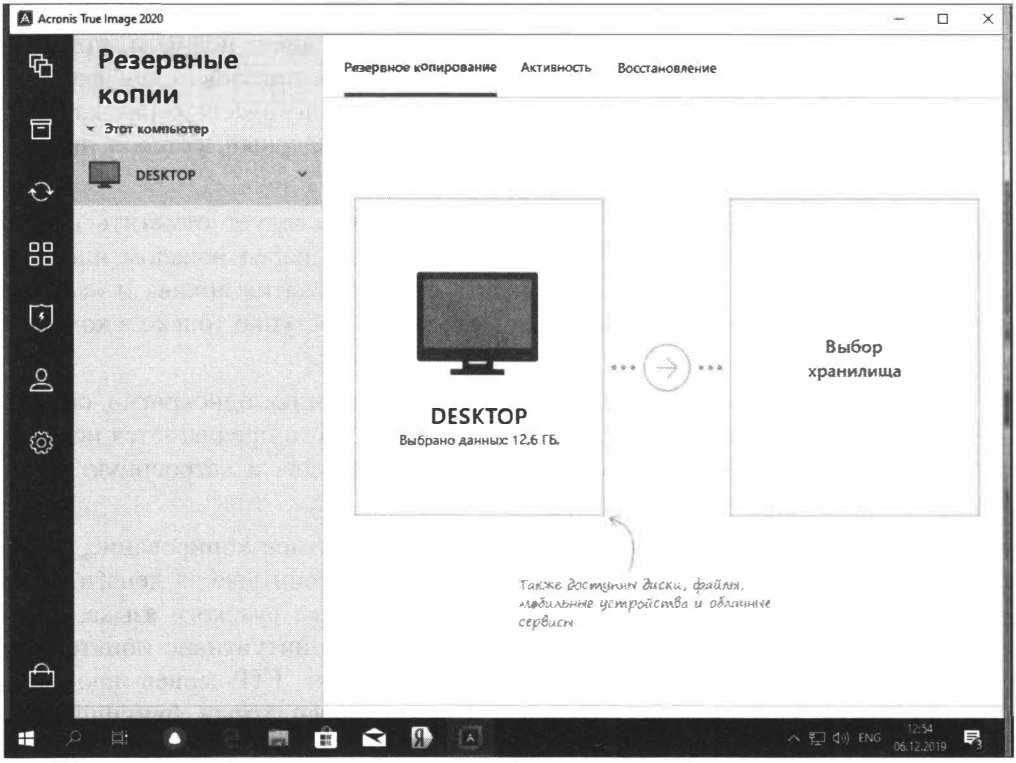


Рис. 3.1. Интерфейс программы Acronis

файлы, резервную копию которых требуется создать, показываешь, куда выгружать архив, причем в качестве места назначения можно выбрать подключенное локально устройство, удаленный FTP-сервер, NAS или любое облако, и... И, собственно, все. При желании резервную копию можно зашифровать, а в разделе **Активность –> Параметры** — задать расписание и схему резервного копирования выбранных объектов.

Поддерживаются все три возможные схемы копирования, также можно настроить исключения для объектов, которые копировать не нужно, и указать периодичность создания полной копии.

Удобством использования достоинства Acronis True Image, собственно, и ограничиваются, в то время как самый существенный недостаток программы — коммерческая лицензия. Если срок ее действия закончился, восстановить данные из резервной копии вы уже не сможете. Кроме того, если вы перестали оплачивать подписку, копии в облаке Acronis тоже через некоторое время будут удалены. Безопасен ли такой инструмент с точки зрения надежности хранения копий? Каждый волен дать собственный ответ на этот вопрос, а мы тем временем поищем бесплатную альтернативу этому приложению.

Aomei Backupper Standard

Это полностью бесплатная для домашнего использования программа с простым и понятным инферфейсом, которую можно скачать с сайта **https://www.ubackup.com/download.html**. Приложение позволяет создавать копии диска целиком, отдельных его разделов, только системных файлов или указанных пользователем файловых объектов. Доступен режим клонирования диска или его логического раздела. В качестве места расположения архива можно выбрать локальный, сетевой диск или облачное хранилище (рис. 3.2).

Из дополнительных опций Aomei Backupper Standard следует отметить возможность запуска указанного пользователем batch-скрипта перед началом или после окончания копирования, также можно задать степень сжатия архива и настроить разбиение его на тома. Шифрование резервной копии доступно только в коммерческой версии программы.

Можно настроить создание резервных копий по расписанию: однократно, ежедневно или в заданном временном интервале, причем для этого предлагается использовать как стандартный Планировщик заданий Windows, так и встроенную службу приложения.

Программа имеет в своем арсенале полное и инкрементное копирование, дифференциальный режим подключается в коммерческой версии уже за деньги. Кроме того, в Aomei Backupper Standard отсутствует поддержка русского языка, однако интерфейс программы настолько очевиден и прост, что интуитивно понятен даже неподготовленному пользователю. Загрузку файлов на FTP-сервер программа, к сожалению, не поддерживает. В целом базовых, бесплатных функций Aomei Backupper Standard вполне достаточно для того, чтобы заменить "жадный" до денег Acronis True Image на домашнем компьютере или ноутбуке.

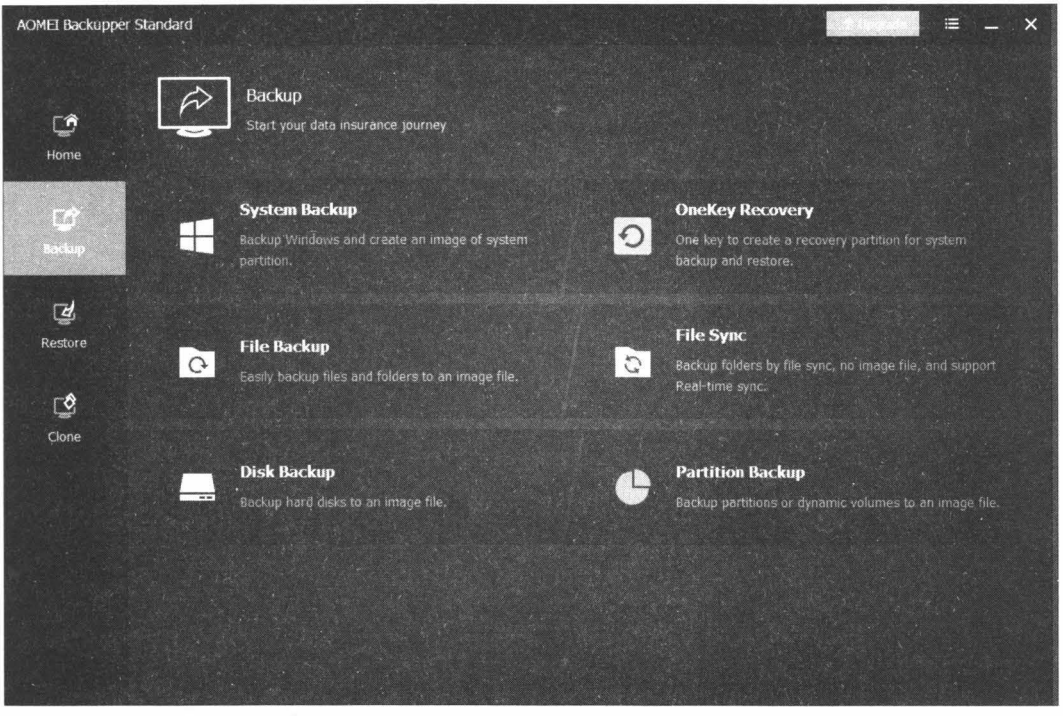


Рис. 3.2. Окно программы Aomei Backupper Standard

Paragon Backup & Recovery

Компания Paragon Software хорошо известна пользователям во всем мире своим замечательным приложением Partition Magic, предназначенным для гибкого управления логическими разделами жесткого диска. Но есть у этого разработчика и бесплатная программа для резервного копирования: Paragon Backup & Recovery Community Edition (рис. 3.3), которую можно скачать по адресу **https://www.paragon-software.com/free/br-free/**.

Программа довольно-таки проста в настройках и поддерживает все тот же набор базовых функций, что и ее основные конкуренты. Копировать можно диск целиком, указанный раздел или выбранный пользователем набор файлов и папок. Также можно выбрать для копирования определенные типы файлов на указанном диске: документы, музыку и видео, причем для каждого из этих типов можно указать необходимые расширения. Копии можно сохранять в локальной или сетевой папке, облачные хранилища и FTP в бесплатной версии приложения не поддерживаются. Не поддерживается и русский язык.

В бесплатной версии имеется возможность выполнять все три вида резервного копирования: полное, инкрементное и дифференциальное. Можно настроить расписание: запуск копирования ежедневно, еженедельно, ежемесячно или при старте системы. Однако шифрование резервных копий в этой редакции Paragon Backup & Recovery отсутствует.

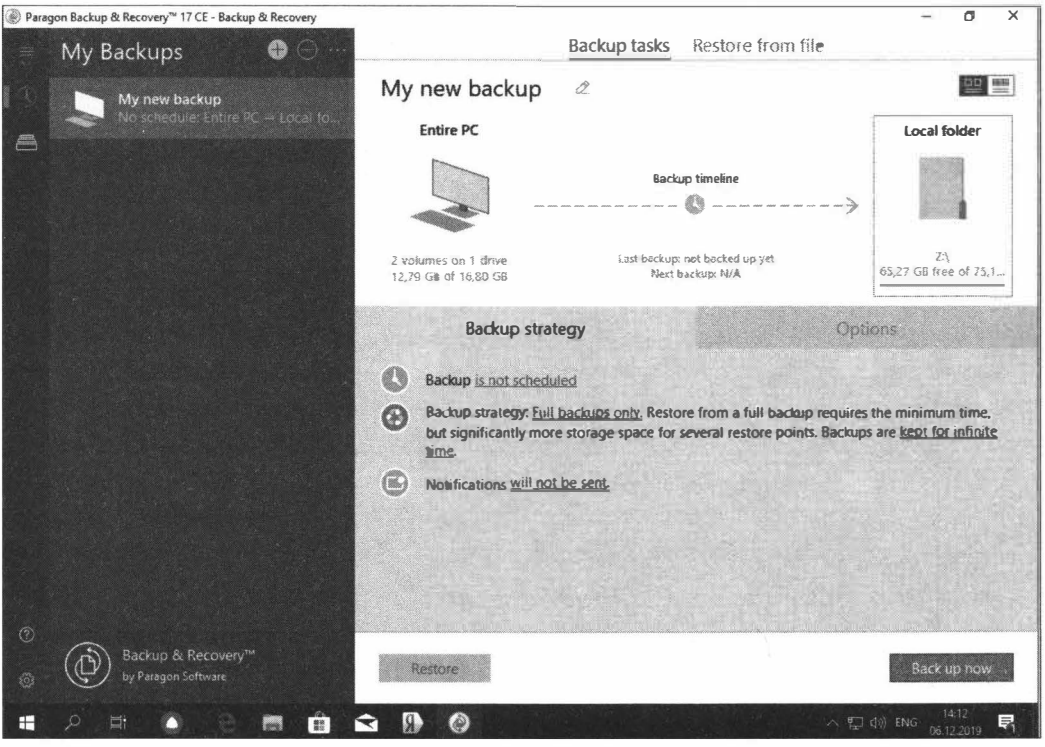


Рис. 3.3. Программа Paragon Backup & Recovery

Iperius Backup Free

Бесплатная версия программного комплекса Iperius Backup (рис. 3.4), предназначенного для создания резервных копий в среде Microsoft Windows, включая Windows 10 и Windows Server. Скачать программу можно со странички **https://www.iperiusbackup.ru/software-backup-free.aspx**.

Бесплатная версия Iperius Backup имеет ограниченный функционал: можно выбрать для копирования отдельные файлы и папки на локальном либо сетевом диске и скопировать архив опять же в локальную или сетевую папку. Копирование на FTP-сервер или в облако в бесплатной версии не поддерживается. Коммерческая редакция Iperius Backup также может копировать диск или раздел целиком, данные с FTP, виртуальные машины Hyper-V, данные Microsoft Exchange и различные СУБД, но эти возможности, опять же, доступны только за деньги.

В приложении имеется планировщик, с помощью которого можно настроить копирование в заданное время — ежедневно, еженедельно, по дням недели или по определенным датам. Имеется возможность настроить способ сжатия архива, ведения лог-файлов, возможность добавлять или исключать из бэкапа системные и скрытые файлы. Можно настроить отправку сообщения по электронной почте при завершении копирования, а также запуск произвольной программы или скрипта до или после создания резервной копии. Шифрование архива в бесплатной версии недоступно, зато программа оснащена русскоязычным интерфейсом, что уже неплохо.

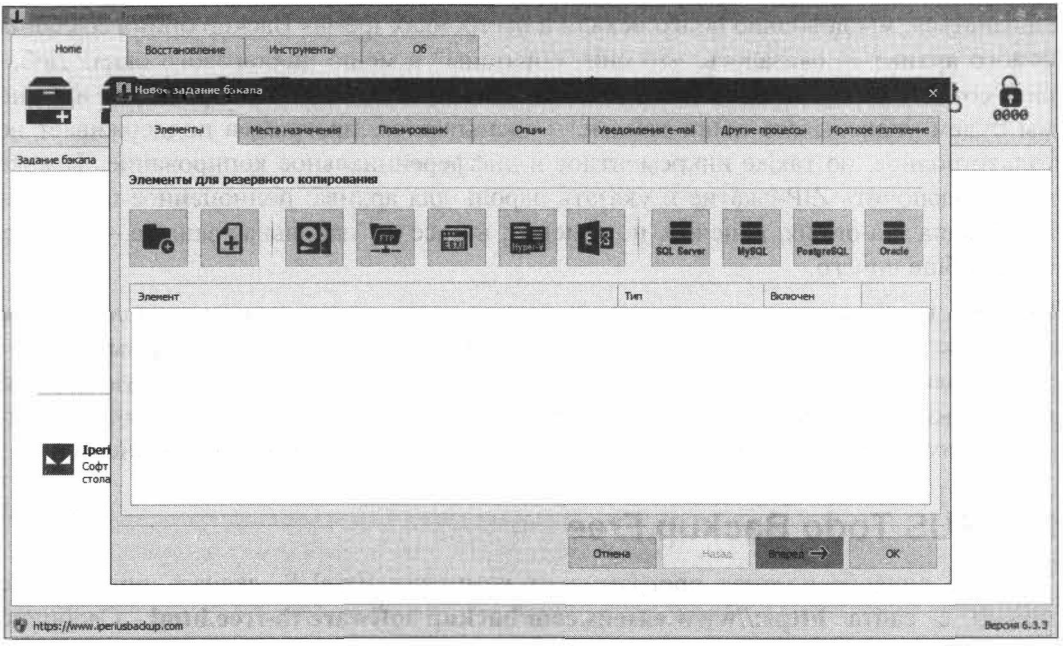


Рис. 3.4. Программа Iperius Backup Free

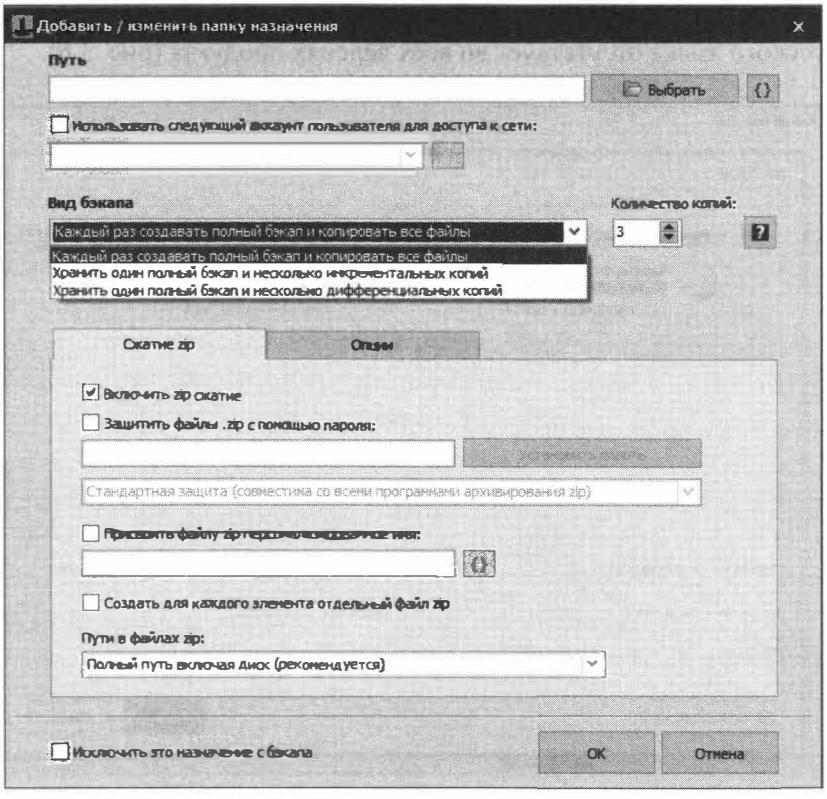


Рис. 3.5. Если порыться в настройках Iperius Backup Free, можно найти много интересного

Признаться, мы довольно долго искали в настройках Iperius Backup опции создания самого архива — оказалось, что они "спрятаны" в меню диалогового окна "Добавить/создать папку назначения" (рис. 3.5). Именно там можно выбрать, как именно мы будем упаковывать наши данные — оказывается, программа поддерживает не только полное, но также инкрементное и дифференциальное копирование. Там же можно включить ZIP-сжатие и указать пароль для архива: полноценное шифрование бэкапа эта опция, конечно, не заменит, но все же парольная защита — лучше, чем вообще ничего.

Вывод в целом можно сделать такой: возможности приложения выглядят вполне привлекательными, но все портит довольно-таки мудреный и запутанный интерфейс, половина кнопок в котором при нажатии предлагает купить полную версию программы. Но если потратить время, чтобы разобраться в ее бесплатных возможностях, их вполне достаточно для закрытия большинства базовых потребностей.

EASEUS Todo Backup Free

Это еще одна бесплатная программа от компании EaseUS, скачать приложение можно с сайта **https://www.easeus.com/backup-software/tb-free.html.** Свободно распространяемая версия также имеет ряд ограничений: не поддерживается отдельное копирование архивов электронной почты Outlook и создание расписания, основанного на событиях, — запуск можно настроить только по времени. Поддержка русского языка отсутствует во всех версиях продукта (рис. 3.6).

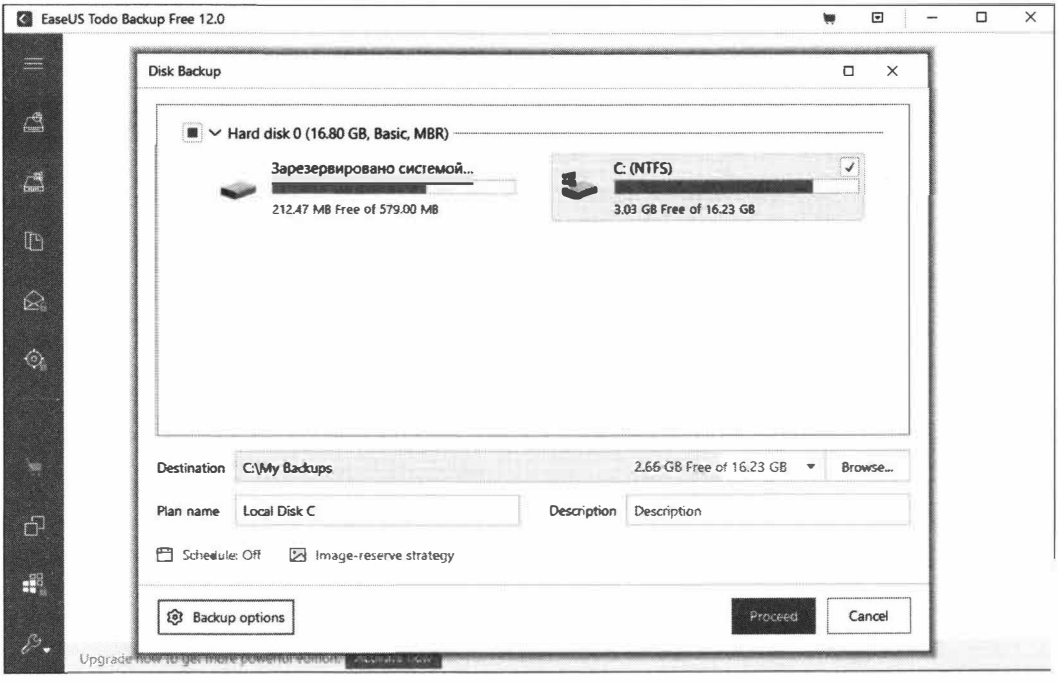


Рис. 3.6. Программа EaseUS Todo Backup Free

Что, безусловно, порадовало, так это возможность шифрования бэкапа с заданным паролем. Можно также включить посекторное копирование выбранного раздела — по-моему, эту опцию не поддерживает ни одна другая бесплатная программа.

Запуск резервного копирования настраивается по дням, неделям или месяцам. Созданный архив можно поместить на локальный или сетевой диск, в сетевое хранилище NAS или в облачное хранилище (если оно, опять же, подключено к компьютеру, как диск). Копирование на удаленный FTP-сервер доступно только в платной версии.

Отдельной функцией программы является System Backup — создание образа установленной на компьютере операционной системы. Todo Backup Free сама определяет тип ОС и позволяет сохранить ее для быстрого развертывания. А еще можно настроить копирование только отдельных файлов с указанных дисков. К сожалению, выбор типов файлов по маске или расширению бесплатным пользователям недоступен.

Кроме всего прочего, в этой версии продукта доступна функция клонирования диска или логического раздела в виде пошагового мастера, а в дополнительных настройках Todo Backup Free можно найти средства для проверки ранее созданного образа диска, а также для создания загрузочной флешки или CD с WinPE или Linux. К сожалению, мы не нашли в настройках программы возможности создавать инкрементную и дифференциальную копию — такое ощущение, что Todo Backup Free умеет работать только с полной копией. Ну и последнее: при беглом знакомстве с этим приложением создалось впечатление, что по сравнению с конкурентами оно заметно "тормозит". На мощном компьютере это, возможно, будет не слишком заметно, но на нетбуке с ограниченными аппаратными ресурсами Todo Backup Free может и вовсе "повесить" систему. Будьте осторожны.

Если не брать в расчет коммерческий Acronis True Image, из бесплатных продуктов наиболее интересными по ассортименту предлагаемых функций выглядят русскоязычный Iperius Backup Free и AOMEI Backupper Standard. Второй обладает более простым и интуитивно понятным интерфейсом, но не умеет шифровать или защищать паролем резервные копии. Если же вам необходимо шифрование, но при этом достаточно только простого копирования файлов без возможности создания инкрементных или дифференциальных копий, можно воспользоваться бесплатной программой EaseUS Todo Backup Free.

Резервное копирование в Linux/BSD

Утилиты, предназначенные для резервного копирования в Linux, отличаются от рассмотренных ранее тем, что немалая часть их — консольные. К ним, как правило, прилагается подробная документация, переписывать которую нет смысла, так что ограничимся описанием особенностей каждой из программ.

Самое что ни на есть хардкордное резервное копирование данных с носителей можно выполнить в Linux командой `dd`. И самое интересное, что можно примонтировать получившийся образ и работать с ним как с обычным разделом в режиме

чтения и записи. И никаких DAEMON Tools или Alcohol для эмулирования дисковода... Но если требуется организовать резервное копирование в сетях SOHO (Small office/home office — сеть малого офиса/домашняя сеть) по всем правилам, следует использовать предназначенные именно для этого утилиты. Итак, начинаем наш обзор с классики резервного копирования в Linux.

rsync

`rsync` (**https://rsync.samba.org**), появившаяся в 1996-м и даже портированная в 1999 году на NT, была разработана в качестве замены утилиты для создания удаленных копий `rcp` (remote copy). Имя ее произошло от словосочетания remote synchronization (удаленная синхронизация), для чего она и была предназначена.

Утилита включена во все популярные дистрибутивы Linux; существуют версии для систем BSD, macOS и Windows. Эта свободная программа с интерфейсом командной строки предоставляет широкие возможности синхронизации локальных и удаленных файлов с помощью собственного алгоритма, уменьшающего объем пересылаемого трафика. Она применяется для создания зеркал и резервных копий — полных, инкрементных и при желании дифференциальных. Для автоматизации можно настроить расписание с помощью задач `cron`. При передаче на удаленные хосты `rsync` умеет использовать разный транспорт, в том числе rsh, SSH и "сырые" сокеты с собственным протоколом `rsync`. Можно указывать, какие файлы не следует синхронизировать, перечислив их имена, размер и не только; можно ограничивать скорость передачи данных и использовать множество других настроек (рис. 3.7).

Программа `rsync` позволяет создать демон — своего рода удаленный сервер `rsync`, который также можно сконфигурировать совершенно по-разному в зависимости от

```
remnux@remnux: $ rsync -avz scrshots/ /media/hardcore/rsync/
sending incremental file list
./
2019-10-13-094445_1920x975_scrot.png
2019-11-22-113132_1920x975_scrot.png
2019-12-13-084332_1920x975_scrot.png

sent 703,547 bytes  received 76 bytes  469,082.00 bytes/sec
total size is 761,402  speedup is 1.08
remnux@remnux: $ rsync -avz scrshots /media/hardcore/rsync/
sending incremental file list
scrshots/
scrshots/2019-10-13-094445_1920x975_scrot.png
scrshots/2019-11-22-113132_1920x975_scrot.png
scrshots/2019-12-13-084332_1920x975_scrot.png

sent 703,564 bytes  received 77 bytes  1,407,282.00 bytes/sec
total size is 761,402  speedup is 1.08
remnux@remnux: $ ls /media/hardcore/rsync/
2019-10-13-094445_1920x975_scrot.png  2019-12-13-084332_1920x975_scrot.png
2019-11-22-113132_1920x975_scrot.png  scrshots
```

Рис. 3.7. Вот так один маленький слеш в конце пути-источника решает, создаст ли rsync папку в назначении

поставленной задачи. Он может принимать удаленные соединения анонимно, по паролю и без него, с шифрованием по SSH на заданном порте с использованием ключей, с различными конфигурациями для разных пользователей, создаваемых в рамках `rsync`... При всем этом для успешного общения двух узлов демон `rsync` необязателен.

В руководстве описано множество нюансов работы с этим мощным инструментом, таких как обработка символических ссылок и атрибутов файлов, опции удаления файлов при обновлении директорий, параметры пересылки данных и многое другое, что и в голову так просто не придет. Опции настройки демона описаны в `man rsyncd.conf`, а примеры использования программы также можно найти на сайте. В общем, в любой непонятной ситуации — RTFM!

luckyBackup

Это графическая утилита (**http://luckybackup.sourceforge.net**), построенная на основе `rsync`. LuckyBackup умеет делать резервные копии и выполнять синхронизацию, работать с удаленными хостами и экономить трафик, передавая лишь изменения в данных, гибко настраивается. Поддерживает русский интерфейс.

Для резервного копирования в luckyBackup создаются профили, внутри которых можно довольно тонко определить задачи:

- тип (бэкап или синхронизация) и описание задачи;
- правила копирования определенных директорий: настройка исключений по шаблонам, можно задать их самостоятельно или использовать имеющиеся; включений, которые должны быть обязательно скопированы;
- настройка использования удаленного узла в качестве источника или места назначения, с SSH или без;
- сохранение метаданных файлов, обработка ссылок и специальных файлов, в том числе по пользовательским шаблонам;
- указание команд оболочки, которые необходимо выполнить до или после исполнения задачи;
- можно просмотреть команду `rsync`, которая будет запущена для выполнения созданной задачи.

Задачи можно включать и отключать, а профили можно экспортировать (почему бы не забэкапить профиль программы бэкапа?). Контекстная справка при наведении курсора помогает не затеряться в расширенных опциях.

Back In Time

Это программа (**https://github.com/bit-team/backintime**) с набором функций, очень похожим на luckyBackup. Тоже поддерживает профили, но экспортировать их нельзя, и внутри них нет задач — здесь профиль и есть задача.

Back In Time может функционировать в консольном или GUI-исполнении. Предлагает обычные или шифрованные EncFS копии, которые могут храниться локально

или передаваться по SSH. Позволяет создавать бэкапы по дням и по часам, при включении компьютера и при подключении устройств, для чего использует `cron` или правила `udev`.

Можно настроить удаление старых бэкапов по их возрасту и количеству свободного места или индексных дескрипторов (inodes). Эти дескрипторы (о чем будет подробнее рассказано в *главе 9*) тоже могут закончиться, и тогда не получится создать файл, даже если есть место на диске. Вкладка Expert Options содержит набор расширенных опций для `rsync` (рис. 3.8). Бэкапы утилита сохраняет в виде архивов.

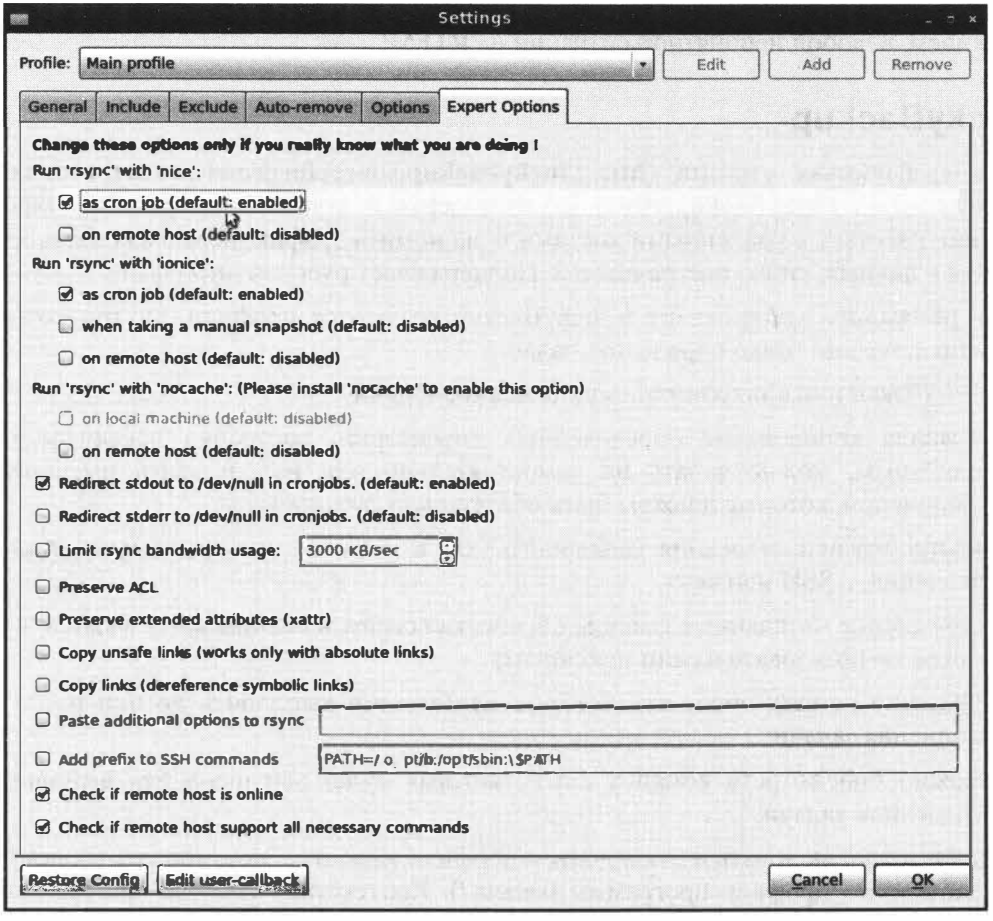


Рис. 3.8. Расширенная настройка профиля в Back In Time

duplicity

Это консольная утилита, построенная на базе `librsync` и поддерживающая полное и инкрементное копирование. Duplicity сохраняет данные в tar-архивах на удаленный или локальный файловый сервер и может использовать GnuPG для их подписи и шифрования. На сайте проекта (**http://duplicity.nongnu.org**) указаны поддерживае-

мые на текущий момент протоколы для работы с файловым сервером, среди которых: Dropbox, FTP, Google Drive, IMAP, MS Azure и Onedrive, rsync, SSH/SCP.

Duplicity предлагает следующие функции:

- работу с правами и атрибутами файлов, символьными ссылками и файлами устройств, но жесткие ссылки считает за обычные файлы;
- исключения и включения для создания бэкапа, в том числе с регулярными выражениями;
- сравнение содержимого директории с ее резервной копией;
- логирование и восстановление к определенной дате;
- симметричное и асимметричное шифрование резервной копии перед отправкой на сервер. Не доверяете облачному сервису? Этот вариант для вас!

Здесь приведены, естественно, далеко не все возможности duplicity, но эта утилита снабжена хорошей документацией с примерами, которая не оставит в беде. Кроме того, к ней есть графический фронтенд Déjà Dup (**https://wiki.gnome.org/Apps/DejaDup**) с ограниченным набором функций, но умеющий делать все самое необходимое (рис. 3.9). В системе он может называться Backups.

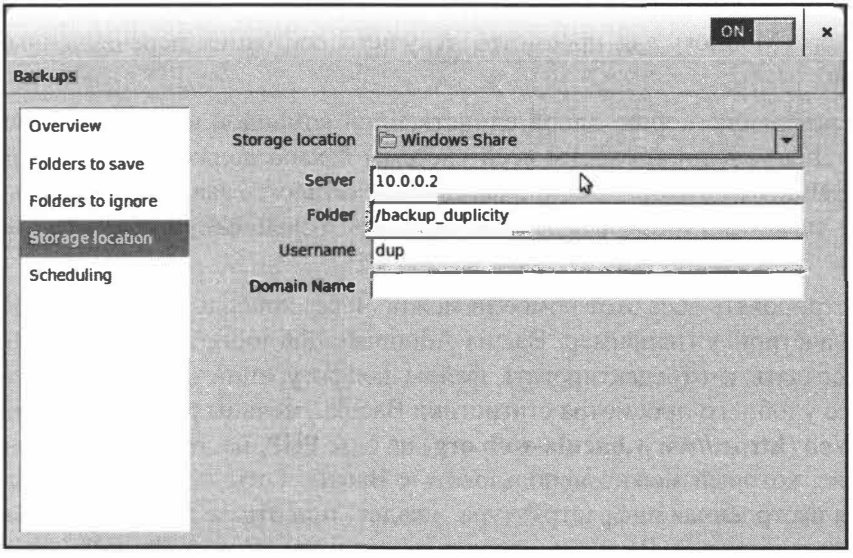


Рис. 3.9. Déjà Dup весьма лаконичен и прост

Bacula

Bacula по праву можно назвать "опенсорсным монстром для резервного копирования" (**https://www.bacula.org**). Его можно собрать из исходников, однако готовые пакеты последних версий с сайта разработчиков просто так не получить. Bacula обещает мощную функциональность — программа не в последнюю очередь ориентирована на работу в крупных компьютерных сетях и вообще на масштабирование;

она умеет делать все три вида бэкапов. Имеются версии для Windows (есть даже клиент под Win98!), macOS и Open/NetBSD.

Bacula придерживается клиент-серверной архитектуры для организации хранения резервных копий, т. е. подразумевает наличие отдельного сервера, на который клиентские машины "скидывают" свои бэкапы. Тем не менее можно настроить ее так, чтобы она работала целиком на одном компьютере.

Компоненты программы Bacula:

- директор (Director), который централизованно следит за всеми операциями резервирования и восстановления;
- консоль (Console), с помощью которой пользователь управляет директором. Может использовать командную строку или GUI;
- файловый демон (File) — клиент, который устанавливается на клиентской машине и обменивается данными с директором;
- демон хранилища (Storage), состоящий из программ, заведующих хранением и управлением файлами бэкапов;
- каталог (Catalog), в котором хранятся данные о резервных копиях. Для него Bacula использует MySQL, PostgreSQL или SQLite;
- монитор (Monitor) для просмотра текущего состояния перечисленных компонентов.

Сделать резервную копию одной-единственной командой в случае с Bacula не получится. Для успешной работы всей системы бэкапа должны быть настроены директор, файловый демон, демон хранилища и каталог. Такая модульность позволяет гибко ее конфигурировать и разворачивать отдельные компоненты на разных серверах.

Администрировать весь этот комбайн можно через консоль или какую-либо графическую надстройку (например, Bacula Administration tool), но первоначально необходимо создать и отредактировать файлы конфигураций основных компонентов. Для более удобного просмотра статистики Bacula умельцы разработали инструмент Bacula-Web (**https://www.bacula-web.org**) на базе PHP, но это не единственный веб-интерфейс, который можно использовать с Bacula. Есть, однако, у Bacula и минусы — вся настроенная инфраструктура "упадет" при отказе директора. В свое время по разным причинам у Bacula появились ответвления: Burp и Bareos.

rsnapshot

Утилита (**https://rsnapshot.org/**), также разработанная на основе `rsync`, умеет делать инкрементные "снапшоты" файловой системы (а еще поддерживает LVM) и способна сохранять их на локальной или удаленной машине по протоколам SSH и `rsync`. На самом деле она создает простые файловые копии. Умеет работать с включениями и исключениями так же, как `rsync`, запускать скрипты до и после выполнения копирования. Для автоматизации задачи предлагается использовать `cron`.

Перед запуском утилиты необходимо ее настроить в файле `/etc/rsnapshot.conf`. Все опции в нем хорошо прокомментированы, так что при знании английского хотя бы на уровне чтения конфигов проблем возникнуть не должно.

Если с момента прошлого бэкапа какие-то файлы не изменились, то `rsnapshot` в целевой директории просто создаст жесткие ссылки на такие файлы в предыдущем бэкапе. Кстати, она по умолчанию организует ротацию резервных копий согласно заданному в конфигурационном файле количеству одновременно хранящихся копий.

rdiff-backup

Проект (**https://rdiff-backup.net/**) вдохновлен `rsync` и позволяет создавать инкрементные файловые копии. Rdiff-backup умеет сохранять бэкапы на локальной машине или в сети по SSH, но тогда на второй машине тоже должен быть установлен rdiff-backup. Для поддержки атрибутов файлов необходима дополнительная питоновская библиотека, но указывается, что с ними все равно могут возникнуть проблемы.

Восстановление можно выполнить с помощью ключа `-r` с указанием нужного бэкапа по точному времени его создания, в днях или по номеру, считая от последнего бэкапа. По каждой сессии копирования утилита пишет файл статистики, из которого можно увидеть, например, сколько файлов появилось или было удалено с момента последнего копирования и сколько времени занял сам процесс.

Разработчик почему-то подчеркивает, что rdiff — это Reverse differential backup tool, т. е., в отличие от "обычного моющего средства", она может восстановить старый вариант файла, а не только самую последнюю версию, хотя вообще-то не одна только эта программа позволяет так делать. Еще во всех отзывах, которые удалось найти, отмечается, что rdiff-backup работает заметно медленнее в сравнении, например, с `rsync` и `rsnapshot`. Особенно когда из нескольких тысяч файлов были изменены пара сотен.

Backupninja

Это программа (**https://0xacab.org/riseuplabs/backupninja**) для централизованного управления утилитами резервного копирования, по сути, обертка над их функциональностью. Базируется на duplicity и rdiff-backup, т. е. с ее помощью можно создавать зашифрованные и подписанные копии и передавать их по SSH. Также Backupninja умеет делать tar-архивы со сжатием или без него, копии баз данных MySQL, PostgreSQL, OpenLDAP и subversion и записывать копии на диски. Работает с Linux-VServer. На странице проекта доступны исходники.

В комплекте с Backupninja поставляется ninjahelper — псевдографическая оболочка, в которой можно удобно и наглядно настраивать задания для резервного копирования, не тратя время и силы на изучение ключей разных утилит, хотя при наборе длинных путей программе явно не хватает автодополнения (рис. 3.10).

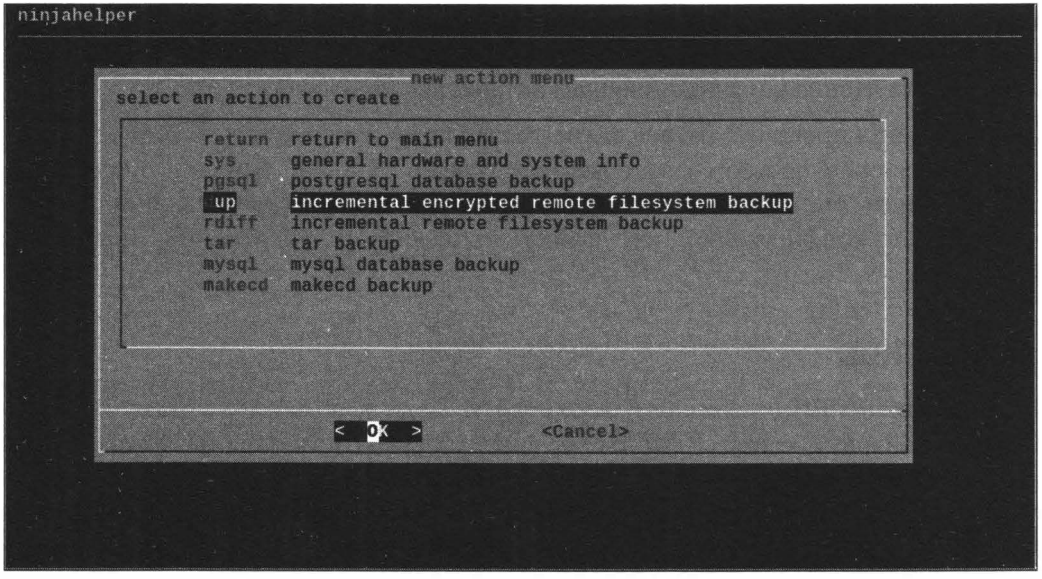


Рис. 3.10. Меню создания действия в ninjahelper

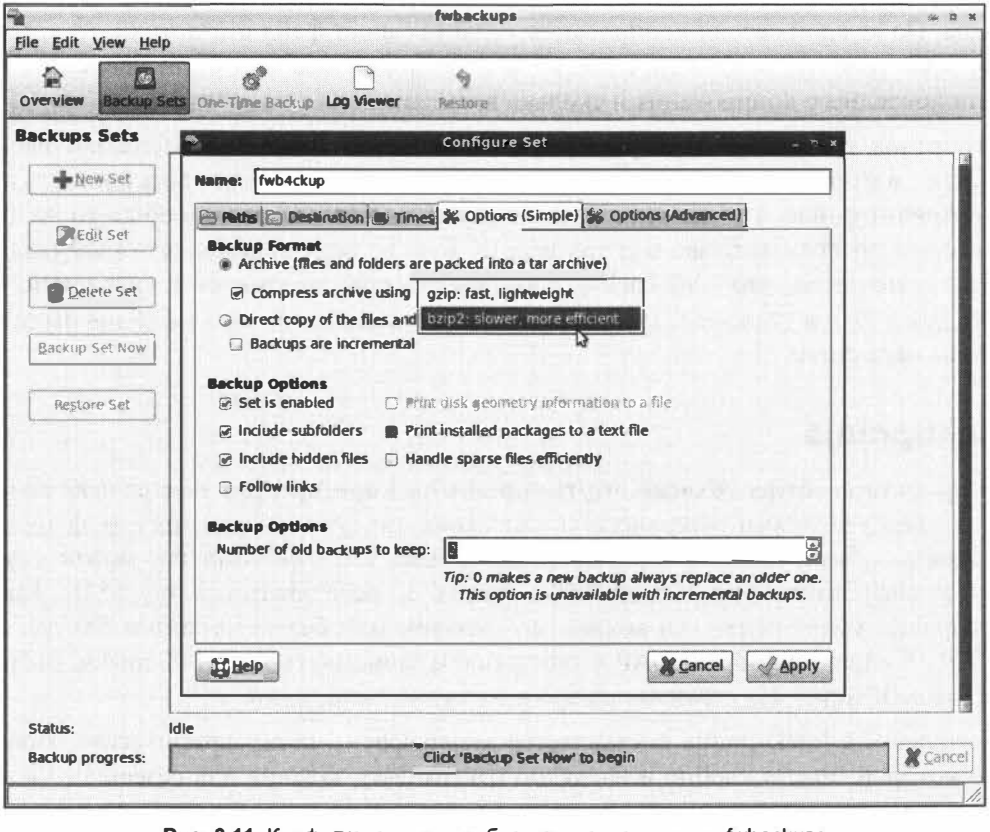


Рис. 3.11. Конфигурирование набора резервирования в fwbackups

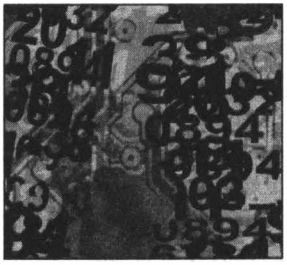
fwbackups

Графическая утилита (**https://diffingo.com/oss/fwbackups/**) с простым и понятным интерфейсом, в которой можно создавать "наборы резервирования" (Sets) — профили — и экспортировать их. Бэкап сохраняется в виде архива или файловой копии локально либо передается по SSH. Инкрементное копирование поддерживается только в случае файловых копий. Fwbackups умеет выполнять ротацию бэкапов и восстановление из них. В настройках расписания можно указать, в который час какого дня недели и месяца следует выполнять копирование, либо можно запустить его вручную. В расширенных опциях указываются команды, которые будут запущены до и после выполнения резервного копирования, а также исключения. Примечательно, что fwbackups позволяет выполнить разовую (One-Time Backup) копию без создания профиля (рис. 3.11).

Резюме

Итак, теперь у вас достаточно знаний, чтобы приступить к созданию резервной копии, если у вас ее по-прежнему нет. В результате вам никогда не придется тратить время и средства на восстановление ваших данных с поврежденных основных носителей, если позаботиться о сохранности данных заранее.

ГЛАВА 4



Выбираем жесткий диск

Своему винчестеру мы доверяем самое дорогое, что у нас есть, — свои данные. Нам часто приходится отвечать на вопросы своих знакомых, сформулированные примерно так: какого производителя выбрать? Какой модели отдать предпочтение? Наш ответ таков: цена и другие параметры (за исключением, может быть, издаваемого шума) важны, но не критичны. Важнейший критерий — надежность. Выбранный вами диск не должен выйти из строя неожиданно. Разумеется, медленная деградация, сопровождающаяся посторонними скрежещущими звуками, и стремительное размножение bad-секторов не в счет, т. к. в данном случае любому и так понятно, что диск надо менять. Тот же самый вопрос возникает постоянно в связи с проблемой надежности диска, но ответа нет. У жестких дисков нет надежности. Вместо этого у них есть гарантийный талон. И это все! Даже не пытайтесь строить свои рассуждения на данных о сотнях тысяч часов наработки на отказ, приводимых в документации. Почему? Да потому, что эта информация берется фактически "с потолка", и производитель не несет за нее никакой ответственности.

Не бывает "хороших" и "плохих" производителей. С каждым брендом случались свои проколы. Независимо от производителя, из партии в тысячу дисков от одного до десяти винчестеров возвращаются задолго до истечения гарантийного срока, даже если они позиционируются как серверные модели. Все решает вероятность. Кому-то жить, а кому-то и умирать.

Правильнее было бы говорить о неудачных моделях. В качестве примера можно привести печально известную серию Fujitsu MPG, в которой использовалась микросхема Cirrus Logic с измененным составом подложки. С течением времени из-за этой подложки образовывались паразитные утечки, и практически все такие винчестеры вымерли в течение двух лет. Еще один пример — IBM DTLA (в просторечии называемый дятлом) с неудачной конструкцией разъема гермоблока, вызывающей периодическое исчезновение контакта и, как следствие, преждевременное прекращение операции записи. При этом, естественно, часть сектора оказывалась незаписанной. В результате на диске образуются виртуальные bad-сектора, на которых нет физических дефектов, однако контрольная сумма не совпадает. Такие сектора можно прочитать, но нельзя восстановить, т. к. запись данных сектора не была завер-

шена. Крис как-то упомянул, что у него было три таких диска. Один из них отказал в течение первых двух месяцев эксплуатации. Он был успешно отремонтирован, а затем заброшен на полку в качестве экспоната. При этом невозможно сосчитать, сколько дисков катастрофически отказало у наших знакомых!

А другому особо сбойному винчестеру даже посвятили страничку в английской Википедии! Это печально известный Seagate ST3000DM001, он же Barracuda 7200.14 (3 ТБ), лидер статистики Backblaze 2014 года по годовой интенсивности отказов. Для этой модели она составила аж 43,1%, тогда как обычно в редких случаях превышает 3–4%! Подозревается (сам производитель так и не дал разъяснений), что герметичность гермоблока была далека от идеала. Пыль попадала в него, из-за чего пластины начинали буквально "сыпаться", лавинообразно усугубляя ситуацию. В результате диски этой серии редко проживали больше одного-полутора лет. Оставшись без рабочего диска, умельцы соорудили из него "трехтерабайтный граммофон", подключив его звуковую катушку в качестве динамика (видео о "поющем" Seagate можно найти в Интернете). Немудрено, ведь что же еще оставалось с ним делать? У WD есть неудачная серия Green, где алгоритм, паркующий головки после 8 секунд бездействия, вместо сокращения энергопотребления приводит к их усиленному износу. В общем, как уже говорилось, в этой области все решает слепая вероятность. В качестве дополнительных факторов можно указать качество блока питания, отсутствие вибраций и т. д.

Сбор статистики об отказах жестких дисков — дело затруднительное. Абсолютное количество отказов само по себе еще ни о чем не говорит. При сборе статистики необходимо учесть распространенность данной модели, а также условия эксплуатации. Считается, что диски SAS надежнее, чем SATA. Однако эта картина наблюдается отчасти потому, что диски SATA устанавливаются в серверах и работают, практически никогда не выключаясь. Стоит учесть, что большинство отказов происходит как раз в момент включения/выключения.

На сайте компании Backblaze, занимающейся облачным хранением данных, приводится любопытная статистика отказов, зафиксированных за разные годы (**https://www.backblaze.com/blog/hard-drive-stats-for-2019/**), которую мы в сокращенном виде приводим далее. В табл. 4.1 приведена статистика по производителям, а в табл. 4.2 — по моделям.

Таблица 4.1. *Статистика отказов жестких дисков по производителям*

Производитель	Всего дисков (годовая интенсивность отказов)		
	2017	2018	2019
Western Digital	661 (1,09–2,21%)	383 (2,15%)	—
HGST (Hitachi)	23 221 (0–0,63%)	20 582 (0,36–1,58%)	48 228 (0,40–0,79%)
Seagate	59 230 (0,70–3,17%)	81 373 (0,33–2,13%)	89 782 (0,96–2,00%)
Toshiba	191 (0–0,70%)	1395 (0–3,03%)	3718 (0–0,65%)

Таблица 4.2. *Статистика отказов жестких дисков по моделям, используемым с 2013 по 2019 год и еще "живым" на момент составления отчета*

Модель	Количество зафиксированных отказов (годовая интенсивность отказов)
Seagate ST4000DM000 4 TB	3843 (2,67%)
Seagate ST12000NM0007 12 TB	1466 (2,57%)
Seagate ST8000NM0055 8 TB	439 (1,22%)
HGST HMS5C4040BLE640 4 TB	245 (0,46%)
HGST HMS5C4040ALE640 4 TB	165 (0,52%)
Seagate ST10000NM0086 10 TB	19 (0,69%)
Toshiba MG07ACA14TA 14 TB	16 (0,87%)
Toshiba MG04ABA400V 4 TB	5 (0,76%)
HGST HUH72121ALE600 12 TB	5 (0,56%)

В данной статистике важно не само абсолютное число отказов, а годовая интенсивность отказов (Annualized Failure Rate, AFR). Как видно из приведенных данных, у Seagate есть модели с показателем AFR 2,67% (всего за период с 2013 по 2019 год было 165 отказов из 2852 винчестеров), а есть с AFR, равной 0,69% (19 отказов из 1200 дисков). Toshiba показала как модели, ни разу не отказавшие за несколько лет, так и модель со вторым по величине AFR в статистике — 3,03%.

Как уже говорилось, время от времени у всех производителей встречаются неудачные модели. К тому же источник отказов зачастую располагается вне диска. Таким образом, вопрос о надежности правильнее ставить так: "Какой диск имеет наибольшие шансы на успешное восстановление?"

Крис в свое время обратился с этим вопросом к Сергею Яценко, работавшему тогда ведущим инженером в фирме ACE Lab. Через руки Сергея прошли тысячи дисков. На основании его ответов сформулированы приведенные далее краткие рекомендации по выбору наиболее "живучей" модели.

Список дисков, наиболее удачных с точки зрения восстановления, т. е. таких, которые проще восстанавливать, составлялся с учетом следующих факторов:

- удобство и простота подбора блока головок в случае проблем с ним;
- отсутствие самоповреждения записи;
- сравнительно небольшое число экстремально сложных узлов.

С учетом вышеперечисленных факторов в список лидеров включены следующие модели: Seagate, Samsung, Hitachi (HGST), Fujitsu и с некоторой натяжкой Toshiba. Не следует забывать, что увеличение объема диска ведет к усложнению его конструкции. Конкретного производителя порекомендовать сложно, поскольку технологии производства HDD в целом развиваются в одном направлении. Современные диски с объемами в терабайты используют до четырех и даже более "блинов", не-

смотря на то, что иные модели умещают 1 ТБ на одной-единственной пластине (так, например, сделано в Seagate Barracuda 7200.14). Большое количество пластин увеличивает нагрузку на ось, из-за чего спровоцировать заклинивание шпинделя на таких дисках несложно даже при слабых ударах (а уж если использовать такие диски в качестве вместительных внешних накопителей, которые постоянно перекладывают туда-сюда, то их дни в неосторожных руках сочтены с самого начала...).

> **Примечание**
>
> Наименования производителей перечислены в порядке увеличения проблематичности восстановления их дисков.

В списке, приведенном далее, перечислены диски, которые, может быть, и отказывают не намного чаще представителей из первого списка, но доставляют массу неприятностей при восстановлении. Этот список тоже упорядочен по мере нарастания проблематичности:

- Maxtor — эти диски "радуют" глючной записью и нестабильностью головок;
- WDC — для этих дисков крайне сложно подобрать исправные головки и в некоторых случаях восстановить функциональность служебной зоны. Кроме того, они имеют статический транслятор, что приводит к невозможности прочитать данные пользователя в случае разрушения модулей транслятора и таблицы дефектов в служебной зоне. Заодно в некоторых моделях Western Digital ось БМГ фиксируется крышкой гермоблока, и после вскрытия воссоздать ее изначальный наклон крайне затруднительно.

Такие производители, как Maxtor и WDC, со своими трудностями справляются, но с явной неохотой. Впрочем, ныне под брендом Maxtor Seagate Technology продает недорогие продукты, ранее известные под именем Samsung, а HGST, ранее Hitachi, в настоящий момент принадлежит Western Digital. Естественно, объективную оценку дать сложно, но ситуация, которую мы наблюдаем, выглядит именно так.

SCSI против ATA, SAS против SATA, NVMe против всех

Некоторые жесткие диски и оптические приводы поддерживают интерфейсы ATA или ATAPI (ATA packet interface), т. е. IDE. С другой стороны, многие стандарты (в числе которых iSCSI, ATAPI, USB, SAS и различные драйверы) "под капотом" используют команды SCSI. Появление интерфейса Serial ATA (SATA) изменило соотношение сил в этой области, и теперь IDE, как и SCSI, встречаются обычно в виде "экспонатов", иногда и работоспособных.

Ожесточенные "звездные войны" вокруг интерфейсов SCSI и ATA начались уже давно. Стандарт SCSI изначально проектировался с прицелом на рынок серверов, прочно на нем обосновался и сдавать свои позиции какому-то Serial ATA не собирался. Стандарт ATA, напротив, задумывался как максимально дешевое решение для однопользовательских маломощных машин. Но SCSI, разработанный в 80-х,

использует параллельный интерфейс, одна из главных проблем которого — рассинхронизация сигналов, идущих с большой частотой по разным проводам. Это основная помеха увеличению скорости передачи. И SATA, и USB, широко распространенные сейчас, являются последовательными интерфейсами. В результате и для SCSI появилась аналогичная замена — Serial Attached SCSI (SAS).

Вавилонская башня технологий

SCSI, ATA, ATAPI, IDE, SAS, NVMe... В этом ворохе аббревиатур даже матерому специалисту не так-то просто разобраться. Но мы все же попробуем!

Помимо системы команд SCSI, терминология этого стандарта была унаследована в нескольких последующих стандартах, поэтому нельзя просто обойти его стороной. Итак, аббревиатура SCSI расшифровывается как Small Computer System Interface (системный интерфейс малых компьютеров). Конструктивно SCSI представляет собой интеллектуальный контроллер, интегрированный непосредственно в само периферийное устройство и поддерживающий унифицированный набор управляющих команд, общий для всех устройств данного типа. По сути своей контроллер SCSI представляет собой мини-компьютер, по мощности сопоставимый с Intel 80486. Во времена становления SCSI это решение было отчаянно смелым и действительно являлось огромным шагом вперед. До появления стандарта SCSI всякое устройство имело свою собственную систему команд, ориентированную на выполнение элементарных операций (например, включить или выключить двигатель, прочитать индексную метку, переместить головку на следующую дорожку и т. д.). Это не только затрудняло программирование, но и требовало переделки контроллера даже при незначительных конструктивных изменениях периферийного устройства.

Устройства SCSI имеют единую схему логической адресации, независимую от физической геометрии устройства, и высокоуровневую систему команд (например, прочитать сектор или группу секторов, начать воспроизведение аудиодиска). Получив команду, устройство ставит ее в очередь и освобождает шину, а инициатор запроса (которым может быть как центральный процессор, так и другое устройство SCSI) переключается на решение другой задачи. Обработав запрос, устройство вновь повторяет захват шины и пересылает данные инициатору, уведомляя его об этом через механизм прерываний. Таким образом, шина эффективно используется несколькими устройствами, и время простоя центрального процессора сводится к минимуму. Максимальное количество устройств на шине SCSI различно и варьируется от одного электрического интерфейса к другому. В среднем к одной шине можно подключить от 7 до 15 устройств, не сильно проигрывая в скорости передачи данных.

Аббревиатура ATA расшифровывается как Advanced Technology Attachment (интерфейс подключения накопителей), и история его возникновения тесно связана с фирмой IBM и компьютерами типа IBM AT. Для преодоления ограничений, свойственных интерфейсу подключения накопителей, использовавшему модифицированную частотную модуляцию (Modified Frequency Modulation, MFM), применяв-

шемуся в IBM XT, компания поручила комитету T10 (**https://www.t10.org**) разработку нового индустриального стандарта. С этой задачей комитет справился на славу, отголоски которой дошли до наших дней, пускай и в сильно измененном виде. Впрочем, никаких революционных идей комитет не предложил, ограничившись интеграцией стандартного контроллера жесткого диска непосредственно с самим устройством, соединенным параллельным шлейфом с не менее стандартной шиной ISA. Так вот почему контроллеры ATA такие дешевые и простые! Фактически они состоят из микросхемы буферной памяти и дешифратора адреса. Разумеется, последующие контроллеры ATA существенно усложнились. Однако эти усложнения не настолько существенны, чтобы вызвать сильное подорожание.

Тем не менее даже первая версия стандарта обнаруживает много общих черт со SCSI. Это и интегрированный контроллер, и унифицированный набор команд (пусть и не такой богатый, как в SCSI), и возможность совместной работы нескольких устройств на шине. Но здесь нет ни "прозрачной" схемы адресации, ни механизма отложенного выполнения команд, ни, тем более, очереди запросов. При этом количество устройств на шине не превышает двух, причем в каждый момент времени может работать только одно устройство, а другое вынуждено ожидать освобождения шины, происходящего только после завершения цикла обмена. Передав команду на чтение сектора, процессор непрерывно опрашивает специальный порт, в котором устройство выставляет флаг готовности данных, пословно считываемый процессором через порт ввода/вывода. Впрочем, в однозадачных системах тех дней это не казалось дикостью, ведь переключиться на выполнение другой задачи процессор все равно не мог, поскольку задача была всего одна.

Между тем аппаратные мощности процессоров непрерывно росли, и на основе IBM PC начали возникать первые многозадачные системы. Как следствие, во второй ревизии стандарта, получившей кодовое наименование ATA-2, появилась поддержка режима DMA. Теперь, передав команду на чтение сектора, процессор мог спокойно переключаться на другую задачу, перекладывая заботу о дисковой подсистеме на контроллер ATA. В последующих ревизиях скорость передачи по физическому интерфейсу увеличилась до 100 Мбайт/с. Кроме того, появилась прозрачная логическая адресация (а вместе с ней и поддержка жестких дисков большого объема). Наконец, было введено расширение ATA, получившее называние ATAPI (ATA Packet Interface — пакетный интерфейс ATA), реализующее ту же самую схему обмена командными пакетами, что и SCSI. В ATA/ATAPI-7 была обеспечена поддержка режима Ultra-DMA. В дальнейших спецификациях был увеличен максимальный объем диска, разрядность логического адреса и скорость передачи данных. В ATA-8 были проведены внутренние доработки, в частности управление защищенной областью диска (Host Protected Area, HPA).

В операционных системах семейства Windows, начиная с версии Vista, предусмотрен особый драйвер ATA port driver (Ataport.sys), специально созданный для работы с устройствами IDE. Все остальные системные драйверы, работающие с портами накопителей, задействуют протокол SCSI (SRB, SCSI_REQUEST_BLOCK) для связи как с драйверами более высокого уровня, так и с драйверами портов. Ataport.sys использует SRB только для связи с драйверами более высокого уровня

(в основном в целях обеспечения совместимости), а для работы с портом применяются запросы IDE (IRB, IDE_REQUEST_BLOCK), имеющие собственную структуру. Драйверы Windows для контроллеров ATA/ATAPI и Serial ATA (SATA) также используют интерфейс минипорта Storport. Помимо драйверов ATA в составе операционной системы имеются стандартные драйверы для минипорта ATA и драйверы стандартных контроллеров ATA/IDE.

Последние версии ATA обеспечивают контроль целостности передачи по интерфейсному кабелю, значительно увеличивая его пропускную способность, и содержат некоторое подобие планировщика. Однако воспользоваться им все равно не удается, поскольку наличие второго устройства на шине многократно уменьшает скорость передачи данных. Для достижения адекватной производительности каждое устройство должно быть подключено к своему контроллеру, а таких контроллеров на подавляющем большинстве материнских плат всего два.

Интерфейс SATA (Serial ATA — последовательный ATA) представляет собой дальнейшее развитие интерфейса ATA (IDE), который после появления SATA был переименован в PATA (Parallel ATA). Теперь широкий шлейф заменен тонким кабелем, соединяющим единственное устройство со своим портом. Вместе с SATA пришел логический интерфейс расширенного хост-контроллера AHCI (Advanced Host Controller Interface), улучшающий производительность жестких дисков за счет уменьшения движений считывающих головок. Впрочем, накопители SATA по-прежнему могут работать и в режиме IDE, но, естественно, не в полную силу. Максимальная длина кабеля и скорость передачи существенно увеличены, однако на жизни большинства пользователей это никак не отражается, поскольку и прежняя длина кабеля в большинстве случаев была вполне достаточной. Что касается скорости передачи данных, то винчестеры не в полной мере использовали даже ту пропускную способность, которая была предусмотрена предыдущей ревизией ATA, но в настоящее время для лучшей работы с SSD она может достигать 6 Гбит/с.

Помимо этого, в спецификации SATA 3.1 появился коннектор mSATA, который может встретиться в ноутбуках и других компактных устройствах. Количество подключаемых устройств по-прежнему невелико (один SATA-порт — одно SATA-устройство, а таких портов на материнских платах раз-два и обчелся). Правда, с возникновением AHCI появилась возможность горячей замены дисков, но для домашних компьютеров она не столь уж критична, а вот на каких-нибудь китайских материнках может привести к тому, из пары портов SATA рабочим останется один.

> ***Примечание***
>
> Если же оставить технические подробности в стороне и взглянуть на SATA с этической точки зрения, то худшего интерфейса, вероятно, не существует в природе. Разработка SATA велась и ведется закрытым сообществом SATA-IO (Serial ATA International Organization — Международная организация Serial ATA). По этой причине и сам стандарт SATA является *закрытым* (см. **https://www.sata-io.org/ developers/ purchase-specification**). Таким образом, подробная техническая документация доступна только членам данного сообщества. В открытом доступе находится лишь устаревшая информация, а современные и актуальные ревизии доступны для бесплатного скачивания лишь членам SATA-IO.

AHCI поддерживает жизненно важную для корректной работы SSD команду TRIM. Тем не менее, чтобы SATA не стал узким местом при работе с SSD, у инженеров Intel появилась идея подключать накопители прямо к шине PCIe (современные шины позволяют работать на скоростях 8–32 Гбит/с!). Был разработан очередной стандарт — логический интерфейс NVM Express (NVMe). Он обладает собственным набором команд, оптимизированных специально под твердотельные носители и многоядерные процессоры.

Аббревиатура IDE расшифровывается как Integrated Device Electronic (интегрированное электронное устройство) и де-факто является синонимом ATA, хотя в "девичестве" обозначала не более чем интеграцию устройства с контроллером. Эта аббревиатура переродилась в торговую марку, практически полностью вытеснившую из употребления аббревиатуру ATA, пока устройства IDE сами не оказались вытесненными SATA.

SAS, он же Serial Attached SCSI (последовательный SCSI), явился на замену своего почтенного предшественника в уже далеком 2004 году. Ощущение некой "Санта-Барбары" в мире интерфейсов усугубляется тем, что SAS в одну сторону совместим с SATA и отличается от него лишь небольшим зазором в физическом интерфейсе, который позволяет подключать устройства SATA в разъем SAS, но не наоборот. Логика работы SAS при этом целиком восходит к SCSI, давнему сопернику ATA (IDE). Нацелен этот стандарт, как и некогда SCSI, на серверный сегмент. Это ощущается как минимум из-за поддерживаемых скоростей передачи данных — 12–24 Гбит/с. Также в SAS бóльшая, чем в SATA, глубина очереди, используется дуплексная передача данных, тогда как в SATA лишь полудуплексная, а также к одному порту можно подключить множество дисков.

Смертельная схватка

Основной недостаток интерфейсов ATA/SATA, который до сих пор не преодолен, — это ограниченное количество подключаемых устройств. Тем не менее SATA, да и SAS прочно заняли свои места, вытеснив IDE и SCSI. Один из немногих часто называемых недостатков SATA — недостаточная пропускная способность при работе с твердотельными накопителями, но его логический интерфейс разрабатывался для оптимальной работы с жесткими дисками, и этом качестве, вероятно, он послужит еще не один год. SAS более высокопроизводительный сам по себе, что проявляется и при работе с SSD. Впрочем, это еще не означает, что стандарт собирается в скором времени последовать за SCSI. С другой стороны, для подключения устройств SAS в домашних сетях требуется приобрести весьма дорогостоящий хост-контроллер, что ставит под сомнение целесообразность его применения там, где без него можно и обойтись.

Логический интерфейс NVMe в сравнении с AHCI во многом выигрывает, и нет особых сомнений, что SSD остановятся именно на нем. В частности, это много бóльшая глубина очереди (65 535 очередей по 65 535 команд против одной с 32 командами), более качественная поддержка прерываний и параллельных операций, что

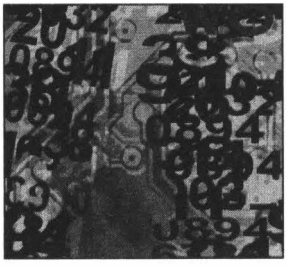
в целом обосновывает лучшую производительность. При этом, как уже было сказано, AHCI ориентирован на работу с винчестерами, где еще удерживает позиции.

Контроллеры RAID за прошедшее со времен первого издания время научились работать не только с SATA-накопителями, но также SAS и NVMe, причем некоторые умеют взаимодействовать сразу со всеми. Все это позволяет настроить RAID не только в серверных инфраструктурах, но и при необходимости в небольших сетях SOHO.

Резюме

Для домашнего использования (если только количество подключенных устройств не очень велико) лучше всего подойдут накопители SATA. То же самое относится и к серверам, обслуживающим локальные сети небольших организаций. Для высокопроизводительных рабочих станций и серверов с внушительными дисковыми массивами однозначно выбирают SAS или в случае SSD — NVMe, вместо ранее широко распространенного SCSI.

ГЛАВА 5



Ремонт жестких дисков

Прежде чем затрагивать логические разрушения, например непреднамеренное форматирование или удаление файлов (обсуждению которых в основном и посвящена наша книга), рассмотрим методику восстановления данных после аппаратных отказов жестких дисков. Сделать это абсолютно необходимо, поскольку выполнение таких операций — вопрос жизни и смерти вашего жесткого диска.

Введение

Объемы жестких дисков стремительно растут, а их надежность неуклонно падает. С одной стороны, поджимает плотность записи, с другой — конкуренция. Повсеместно применяются дешевые комплектующие и "сырые" технические решения, обкатывать которые приходится потребителям. Залог безопасности данных — ежедневное резервирование (тем более что современные съемные носители это позволяют), о чем мы поговорили в *главе 3*. Однако, несмотря на это, даже продвинутые специалисты часто пренебрегают этой рекомендацией, ведь все "и так работает"...

После отказа винчестера данные практически всегда можно восстановить, если действовать по плану. Если же плана нет, от "врачевания" лучше сразу же отказаться. Неумелые попытки только затрудняют процедуру восстановления, а иногда делают ее совершенно невозможной. Сотрудники сервис-центров настоятельно отговаривают пользователей от самостоятельного ремонта, а за ослушание карают либо удвоенной (утроенной) ценой, либо же отказываются от восстановления. И делают они это совсем не потому, что боятся, что клиент сможет обойтись без их помощи! Жесткие диски — очень сложные устройства. Это не радиоприемники или другая бытовая техника, которую часто можно отремонтировать и без специализированных знаний! У работников сервисного центра есть специализированное оборудование, накопленные знания и опыт. Через их руки прошли десятки тысяч винчестеров, флешек и твердотельных дисков, поэтому шансы на успешное восстановление данных здесь намного выше, чем у простого программиста или администратора, рыдающего над убитым диском. Это теоретически. Практически же... Цена за восстановление зачастую переходит все границы, причем никаких гарантий на благо-

приятный исход все равно нет (хотя не так мало компаний, которые в случае неудачи денег за свои старания тем не менее не берут). Известно немало ситуаций, когда "кустари-одиночки" бесплатно восстанавливали винчестеры, угробленные "специалистами" сервис-центров. А для жителей глубинки "центры" доступны весьма условно, и им приходится рассчитывать скорее на себя.

В этой главе будет идти речь о восстановлении данных. Полноценный ремонт винчестеров (за исключением редких случаев) или невозможен, или экономически нецелесообразен. Нашей задачей будет временное восстановление работоспособности жесткого диска, достаточное лишь для копирования самых ценных данных, в идеале — всего диска целиком. Мы рассмотрим исключительно общие вопросы ремонта жестких дисков, а в подробности пошаговой методики диагностики вдаваться не будем. Это — чрезвычайно обширная тема, заслуживающая отдельной книги и к тому же требующая индивидуального подхода к каждой конкретной модели диска. Заинтересованных читателей, желающих изучить данную тему углубленно, можно отослать к документации и полезным ссылкам, представленным на сайте ACE Lab (**https://www.acelab.ru/dep.pc/information.php**). Так что не будем повторяться. Основная цель данной главы — продемонстрировать, что небольшой ремонт жестких дисков возможен даже в домашних условиях.

Внутреннее устройство жесткого диска

Блок-схема типичного жесткого диска представлена на рис. 5.1. Конструктивно жесткий диск состоит из гермоблока и платы электроники. В гермоблоке расположены:

- шпиндельный двигатель, вращающий пакет из одного или нескольких магнитных дисков;

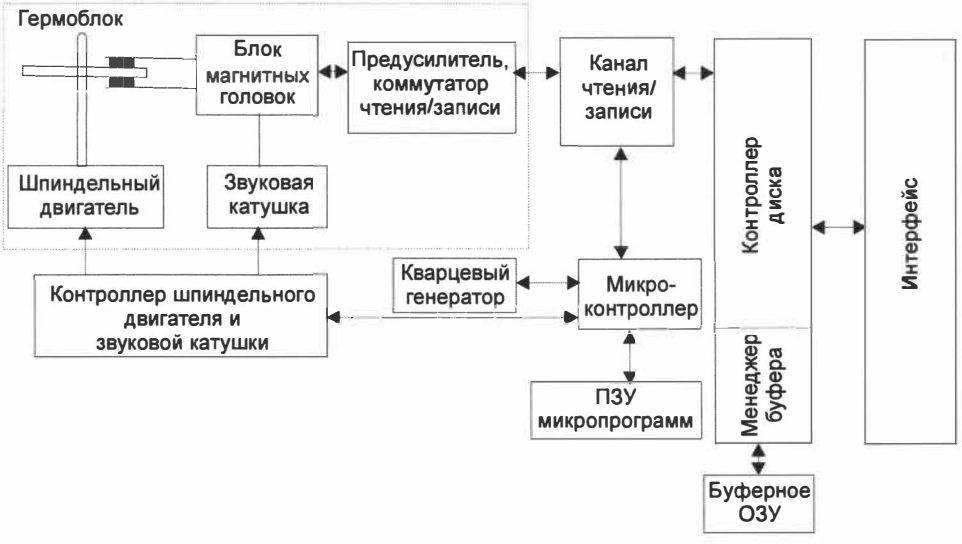


Рис. 5.1. Блок-схема типичного жесткого диска

- блок магнитных головок (БМГ), который ранее управлялся шаговым электродвигателем, а теперь работает под управлением устройства, известного как "звуковая катушка" (voice coil);
- предусилитель-коммутатор чтения/записи, смонтированный в микросхеме, расположенной либо непосредственно на БМГ, либо на отдельной плате рядом с ней. В последнем случае замена коммутатора возможна без съема БМГ, что существенно упрощает его ремонт.

Плата электроники содержит:

- контроллер шпиндельного двигателя и звуковой катушки, управляющий вращением пакета дисков и позиционированием головок;
- канал чтения/записи;
- микроконтроллер, являющийся, по сути, "сердцем" винчестера;
- контроллер диска, отвечающий за обслуживание интерфейса ATA.

Принципы ремонта жестких дисков

Древние жесткие диски стоили дорого, использовали целую россыпь микросхем с низкой степенью интеграции и серийные комплектующие, над которыми еще имело смысл подолгу "зависать" с осциллографом, выискивая неисправный элемент. Но затем степень интеграции начала стремительно нарастать, производители перешли на заказные чипы, а цены на винчестеры упали. Ремонтировать электронику стало не только сложно, но еще и нерентабельно.

Основным способом возвращения работоспособности жесткому диску стала замена платы контроллера целиком. Для этой цели берется диск идентичной модели (донор) и плата переставляется на гермоблок с восстанавливаемыми данными (акцептор). Исключение составляет мелкий ремонт, наподобие замены перегоревшего предохранителя или транзистора, который можно выполнить непосредственно на теле "пациента".

Возникает вопрос — если ремонтники уже давно ничего не ремонтируют, а только тасуют платы, зачем же к ним обращаться и платить деньги, когда эту операцию можно проделать и самостоятельно? Однако в данном случае проще сказать, чем реально сделать.

Во-первых, необходимо найти подходящего донора. У разных моделей винчестеров совместимость плат электроники существенно различна. Некоторые из них требуют совпадения всех цифр в номере модели, а некоторые соглашаются работать и с "родственным" контроллером. Есть и такие модели, которые могут не работать даже при полном совпадении всех букв и цифр, и тогда приходится перебирать одного донора за другим в надежде найти подходящий. Особенности поведения каждой модели можно почерпнуть из документации, прилагаемой к PC-3000 (см. *главу 2*), или найти в Интернете. Поиски доноров серьезно осложняются тем, что период производства большинства винчестеров намного меньше их среднего срока существования. Компьютерные магазины постоянно обновляют свой ассортимент,

и приобрести модель, аналогичную той, что вы купили несколько лет назад, скорее всего, не удастся. Остаются сервисы объявлений, радиорынки и фирмы, торгующие подержанными комплектующими, но и здесь выбор невелик.

> ***Внимание!***
> "Неродной" контроллер может повредить микросхему коммутатора/предусилителя, расположенную внутри гермоблока, и разрушить служебную информацию, что существенно затруднит дальнейший ремонт. Никогда не переставляйте платы, если у вас есть хотя бы тень сомнения в их совместимости!

Во-вторых, помимо электроники на плате контроллера имеется микросхема ПЗУ, в которой могут быть записаны индивидуальные настройки. В этом случае с чужой платой винчестер работать просто не будет! Тут есть два пути. Если акцептор еще подает признаки жизни, с него считывается оригинальная прошивка, которая затем записывается на плату донора. Если этот вариант не срабатывает, приходится перепаивать непосредственно само ПЗУ.

В-третьих, даже если винчестер "заведется" с чужой платой, последовательность нумерации секторов может быть нарушена, и файловая система превратится в "мусор". Если это случится, разгребать этот мусор придется вручную или с помощью специализированных программных комплексов. Лучшим среди этих комплексов является Data Extractor, входящий в комплект PC-3000, но также способный работать и отдельно от него со штатным контроллером SATA/IDE.

Вообще говоря, никаких экстраординарных способностей для ремонта не требуется, и он вполне по силам мастерам средней руки. Отказ электроники — это еще полбеды. Гораздо хуже, если испорчена часть служебной информации, записанной на магнитных пластинах (эта тема будет освещена более подробно далее в этой главе). Это может произойти по разным причинам, наиболее распространенными среди которых являются: ошибки в прошивке, сбои питания, отказ электроники, вибрация/удары, деформация гермоблока. При этом жесткий диск не инициализируется или выдает сообщение об ошибке в ответ на любую команду. Некоторые винчестеры автоматически переходят в технологический режим, предназначенный для записи служебной информации, которая может быть передана либо через стандартный интерфейс ATA, либо через COM-терминал.

В состав PC-3000 входит большая коллекция разнообразных служебных модулей для популярных моделей жестких дисков, а зарегистрированным пользователям предоставляется доступ к форуму технической поддержки — своего рода базе знаний, где можно найти практически все что угодно. Как вариант, можно воспользоваться специализированными утилитами, распространяемыми производителями винчестера, выбрав режим обновления прошивки. Важно отметить, что при этом обновляются далеко не все модули диска; более того, далеко не для всех моделей винчестеров такие утилиты существуют. К тому же этот способ восстановления бесполезен, если в служебной зоне имеются физические дефекты или если накопитель "зависает" еще на старте, отказываясь входить в технологический режим. На этот случай существует метод горячей замены (hot-swap). В этой процедуре также участвуют два накопителя — донор и акцептор, но трансплантация осуществляется

"на лету". Акцептор обесточивается, с него снимается плата электроники, обнажая гермоблок. Донор подключается к шлейфу, на него подается питание, затем, после завершения процесса инициализации и выдачи сигнала готовности, отдается команда ATA `Sleep` (`95h`), останавливающая шпиндельный двигатель. Все остальные узлы остаются под напряжением. Контроллер аккуратно свинчивается и переставляется на гермоблок акцептора. Затем ему подается любая команда для пробуждения (например, команда чтения сектора). Поскольку контроллер уже был проинициализирован, обращения к служебной зоне не происходит и с диска удается считать всю уцелевшую информацию.

> ***Примечание***
>
> При использовании штатного контроллера SATA/IDE необходимо заблаговременно отключить S.M.A.R.T. в настройках UEFI/BIOS, иначе винчестер будет осуществлять запись протокола S.M.A.R.T. в служебную зону.

Требования к совместимости плат электроники — те же самые, что и в случае простой перестановки контроллера. В принципе, нет необходимости переставлять плату донора на акцептор. Можно взять плату акцептора, проинициализировать ее на гермоблоке донора, а затем вернуть обратно. Такой способ даже более предпочтителен, поскольку в этом случае акцептор будет работать со "своим" ПЗУ.

Ряд неисправностей требует вскрытия гермоблока и ювелирного мастерства рук. Первое место по частоте отказов занимает выход из строя одной или нескольких магнитных головок (рис. 5.2). Причиной может быть и заводской брак, и пробой электроники, и механическое воздействие (например, удар). Если головка остается физически неповрежденной, то одна из поверхностей перестает читаться, и тогда

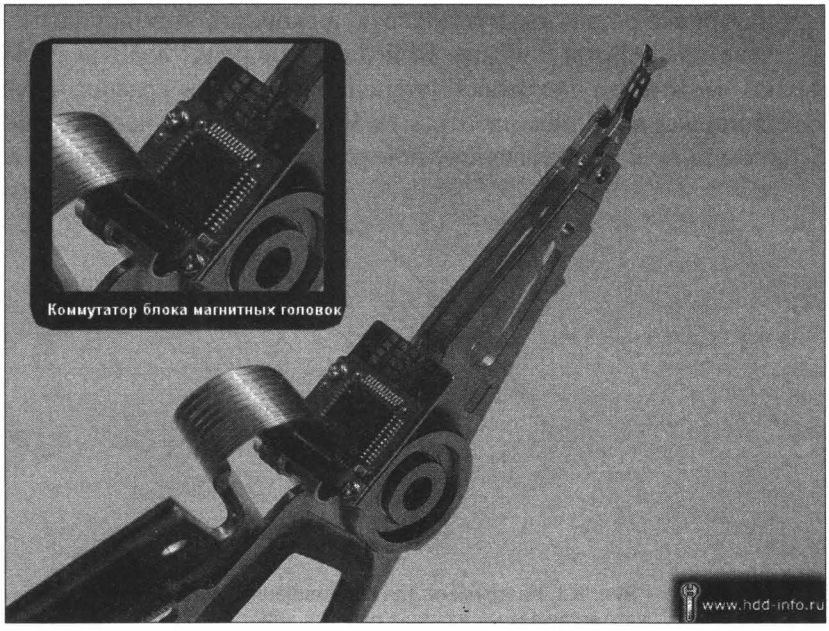


Рис. 5.2. Блок магнитных головок с микросхемой коммутатора/предусилителя

через каждые N секторов образуется bad-сектор, где N — количество головок. Некоторые модели имеют 6 головок, некоторые — только одну, тогда при ее отказе диск становится полностью неработоспособным и не может прочитать даже служебную зону. Но и при отказе одной из шести головок информация превращается в "труху". Все файлы, размер которых превышает 3 Кбайт (512×6), становятся "продырявленными". Что делать? Переставлять блок головок! Это очень сложная операция, и у начинающих мастеров в половине случаев она заканчивается фатально. Практиковаться на своем рабочем винчестере, который надо восстановить, категорически недопустимо! Сначала потренируйтесь на жестких дисках разной степени убитости, на которых нет ничего интересного.

Нам потребуется донор близкой модели. Точное совпадение всех цифр модели уже не обязательно, главное — чтобы БМГ был аналогичного типа. Некоторые диски паркуют головки за пределами внешней кромки магнитных пластин, некоторые — в специальной зоне близ центра шпинделя. Последний случай наиболее сложен. Ведь чтобы снять головки, их нужно протащить через всю поверхность, а контакт головки с поверхностью недопустим, иначе магнитное покрытие будет разрушено!

Вооружившись тонкой полоской выгнутого и обезжиренного пластика, аккуратно заводим ее под каждую головку так, чтобы пластик приподнимал головку над поверхностью, но сам ее не касался, и выводим головки за пределы внешний кромки. Чтобы головки не соприкасались и не царапали друг друга, между ними вставляется полоска полиэтилена, которую можно вырезать из антистатической упаковки жесткого диска (рис. 5.3). Заменяется только БМГ. "Родной" магнит звуковой катушки акцептора остается на месте. В зону парковки магнитные головки заводятся аналогичным образом, но в обратной последовательности. Остается лишь закрутить винт оси позиционера и надеть крышку на гермоблок. При включении винчестера практически наверняка раздастся жуткий звук, а скорость чтения упадает в десятки раз. Это — следствие работы с чужим БМГ на "неродных" адаптивах. Подтягивая винты крышки, можно до некоторой степени выровнять график чтения. Долго в таком состоянии жесткий диск работать не может, поэтому необходимо как можно скорее приступать к вычитыванию поверхности, начиная с наиболее ценных

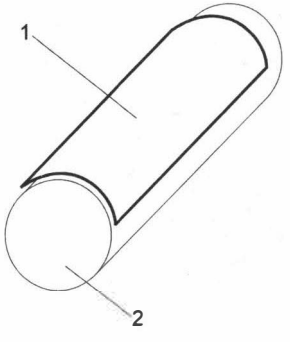


Рис. 5.3. Инструмент для перемещения БМГ, изготавливаемый из узкой полоски пластика (1), обжимаемого на разогретом металлическом стержне (2)

данных. Более подробную информацию на эту тему можно найти в статье Сергея Казанского "Как я переставлял блок головок на Fujitsu MPG3409AH, чтобы спасти информацию. (Записки сумасшедшего ремонтника)", которая сейчас, правда, доступна лишь в архиве: **https://web.archive.org/web/20050312172952/http://onehalf.pisem.net/stat/heads.html**.

Некоторые жесткие диски содержат только одну магнитную головку, в случае отказа которой выгоднее переставлять саму пластину. Множество видеоклипов о ремонте разнообразных неисправностей винчестеров можно найти на видеохостинге Youtube.

Также приходится сталкиваться с "залипанием" магнитных головок, в прямом смысле слова прилипших к поверхности за счет сил межмолекулярного притяжения. Некоторые источники рекомендуют в этом случае просто крутануть диск в горизонтальном направлении, но польза от этого действия очень сомнительна, а вот вред оно может нанести немалый и зачастую непоправимый (например, повредить подвески головки с последующим фрезерованием магнитной поверхности). В этом случае лучше разобрать гермоблок и аккуратно приподнять головки с помощью уже знакомого нам куска изогнутого пластика, вернув их в зону парковки. Более подробно об этом можно узнать в следующей статье: **https://r-hdd.blogspot.com/2012/05/blog-post.html**.

Кроме того, встречаются случаи повреждения коммутатора/предусилителя или обрыва гибкого шлейфа. Если коммутатор/предусилитель расположен непосредственно на БМГ (особенно в микросхеме бескорпусного исполнения), то БМГ заменяют целиком по вышеописанной методике.

Звуковая катушка, в силу своей конструктивной простоты, не отказывает практически никогда (там просто нечему ломаться), но вот выводные провода обломаться могут. К счастью, их легко припаять.

Шпиндельный двигатель очень надежен и перегорает/замыкает обмотки только в исключительных случаях. Однако заклинивание гидродинамического подшипника — вполне распространенное явление. Если это происходит, то подшипник приходится расклинивать по одной из методик, описанных на странице **http://www.mhdd.ru/hdd/klin-podshipnika-dvigatelja.html**.

Прошивка и адаптивы жесткого диска

Электроника диска — это только скелет. Без управляющих микропрограмм она работать не будет! Первые модели винчестеров хранили микропрограммы в ПЗУ, что создавало неудобства и накладывало определенные ограничения. Теперь же для этой цели используется сам жесткий диск! Разработчик резервирует некоторый объем дискового пространства и размещает в нем весь необходимый код и все требуемые данные. Эта информация организована в виде модулей (слабое подобие файловой системы) и управляется специализированной операционной системой. В ПЗУ остается лишь базовый код, своеобразный "фундамент" винчестера. Некоторые производители пошли еще дальше, убрав из ПЗУ все, кроме первичного загрузчика.

Само ПЗУ может быть расположено как внутри микроконтроллера, так и в отдельной микросхеме. Практически все винчестеры имеют микросхему FLASH-ROM, но не на всех моделях она распаяна. Если микросхема FLASH-ROM установлена, то микроконтроллер считывает прошивку из нее, если нет — обращается к своему внутреннему ПЗУ.

Часть модулей (и информации, находящейся в ПЗУ) одинакова для всей серии винчестеров. К ней в первую очередь относится совокупность управляющих микропрограмм. Эти модули полностью взаимозаменяемы, и один диск свободно может работать с модулем другого без каких-либо последствий.

Часть модулей (реже — информации из ПЗУ) готовится отдельно для каждой партии. Например, паспорт диска, описывающий его конфигурацию, указывает количество головок, физических секторов и цилиндров. В процессе инициализации микропроцессор опрашивает коммутатор и перечисляет головки. Если их количество не совпадает с указанным в паспорте, винчестер может "забастовать" и отказаться инициализироваться. Зачастую производители отключают некоторые головки из-за дефектов поверхности, неисправностей самых головок, или же по маркетинговым соображениям. Как следствие — образуются внешне очень похожие модели-близнецы, для которых непосредственная перестановка плат все же невозможна. В этом случае паспорт приходится корректировать, для чего опять-таки понадобится PC-3000. Однако, в принципе, подобрать донора с идентичным паспортом вполне можно и без коррекции.

Основным источником неприятностей при ремонте являются модули (и довольно часто информация, прошитая в ПЗУ), которые уникальны для каждого экземпляра винчестера и настраиваются строго индивидуально. В частности, каждый жесткий диск имеет как минимум два списка дефектов — первичный список, или P-list (Primary list), и растущий список, или G-list (Growing list). В P-list заносятся номера дефектных секторов, обнаруженные еще на стадии заводского тестирования, а G-list формируется самим жестким диском в процессе его эксплуатации. Если запись в сектор происходит с ошибкой, сбойный сектор переназначается другим сектором, взятым из резервной области. Некоторые жесткие диски поддерживают список "подозрительных секторов": если сектор начинает читаться не с первого раза, он замещается, а информация о замещении сохраняется либо в отдельном списке, либо в G-list.

Все эти процессы протекают скрытно от пользователя. Специальный модуль, называемый транслятором, переводит физические адреса в номера логических блоков или виртуальные номера CHS (цилиндр — головка — сектор), и внешне нумерация секторов не нарушается. Все работает нормально до тех пор, пока P- или G-списки не оказываются разрушенными или пока на гермоблок не устанавливается плата с чужими настройками. Если P/G-списки хранятся во FLASH-ROM (а часто так и бывает), то файловая система оказывается полностью неработоспособной, ведь трансляция адресов нарушена! При этом, хотя на секторном уровне все читается нормально, становится совершенно непонятно, какой сектор какому файлу принадлежит.

К счастью, восстановить транслятор довольно просто, поскольку практически все файловые структуры (да и сами файлы) имеют характерные последовательности байтов (сигнатуры). Для начала нужно очистить таблицы транслятора (сгенерировать пустые P/G-списки), в противном случае сектора, помеченные у донора как замещенные, не смогут быть прочитаны на акцепторе. Различные винчестеры имеют разное число замещенных секторов. В некоторых винчестерах замещенных секторов может не быть вообще, в то время как в других накопителях их количество может доходить до нескольких тысяч. Формат P/G-списков варьируется от одной модели к другой, и для работы с ним лучше всего применять PC-3000. В экстренных случаях, если в вашем распоряжении нет PC-3000, можно применить утилиты от производителей винчестера и дать команду ATA `unassign`.

Затем необходимо просканировать весь диск на предмет поиска характерных сигнатур и занести их "физические" адреса в список. Естественно, эти адреса не являются "физическими" в подлинном смысле этого слова. На самом деле они представляют собой логические адреса без переназначенных секторов.

На данном этапе, исследуя служебные структуры файловой системы (каталоги, MFT), мы определяем номера кластеров подчиненных структур. Переводим кластеры в сектора и создаем еще один список. В результате будет получено два списка, между которыми прослеживается четкая корреляция. Первый список как бы "растягивается" вдоль второго. Иными словами, каждый переназначенный сектор увеличивает расхождение между последующими "физическими" и логическими адресами на единицу. Проделав необходимые математические вычисления, можно рассчитать требуемую поправку и частично восстановить транслятор. Слово "частично" используется потому, что целевые адреса замещенных секторов остаются неизвестными, а это значит, что в восстанавливаемых данных образуются "дыры". Тем не менее бóльшая часть информации все же будет возвращена из небытия. Аппаратно-программный комплекс PC-3000 автоматически восстанавливает транслятор, используя довольно продвинутые алгоритмы, которые постоянно совершенствуются. Кстати, при желании утилиту для восстановления транслятора можно написать и самостоятельно, но для этого нужно быть настоящим профессионалом.

К сожалению, ни PC-3000, ни другие аппаратно-программные комплексы не всемогущи. Например, ни один из них не способен восстанавливать адаптивы. Адаптивы начали доминировать более десяти лет назад. До этого индивидуальные настройки диска сводились к высокоуровневым наслоениям, никак не препятствующим чтению информации на физическом уровне. Перестановка плат могла привести к невозможности работы с диском средствами операционной системы, но данные всегда можно было прочитать посекторно стандартными командами ATA или, на худой конец, на уровне физических адресов в технологическом режиме.

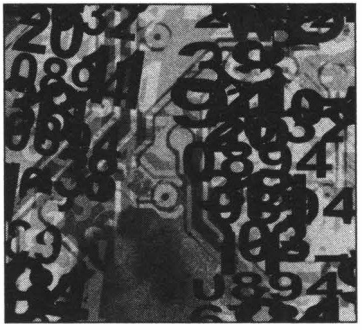
Но плотность информации неуклонно росла, нормативы допусков ужесточались, а это значит, что усложнялся и удорожался производственный цикл. В промышленных условиях невозможно изготовить два абсолютно одинаковых жестких диска. Справиться с неоднородностью магнитного покрытия, влекущего за собой непостоянство параметров сигнала головки в зависимости от угла поворота позиционера,

чрезвычайно сложно. Таким образом, производитель должен выбрать один из перечисленных далее путей.

1. Уменьшить плотность информации до такой степени, при которой рассогласованиями можно пренебречь. Однако в этом случае для достижения той же емкости придется устанавливать в диск больше пластин, что удорожает конструкцию и вызывает новые проблемы.
2. Улучшить качество производства. Это хороший вариант, но при современном уровне развития науки, технологий и экономики он настолько нереален, что даже не обсуждается.
3. Индивидуально калибровать каждый жесткий диск, записывая на него так называемые адаптивные настройки. Именно этот вариант и был выбран производителями, что и привело к появлению адаптивов.

Состав и формат адаптивов меняются от модели к модели. В грубом приближении в состав адаптивов входят следующие данные: ток записи, усиление канала, профиль эквалайзера, напряжение смещения для каждой головки, таблица коррекции параметров каждой головки для каждой зоны и т. д. и т. п. Без своих "родных" адаптивов жесткий диск просто не будет работать! Даже если произойдет чудо и "чужие" адаптивы все-таки подойдут (а чудес, как известно, не бывает), то информация будет считываться крайне медленно и с большим количеством ошибок. Подобрать адаптивы нереально, рассчитать их в "домашних" условиях — тоже. Но ведь как-то же эти адаптивы возникают? Чисто теоретически для заполнения таблицы адаптивов не нужно ничего, кроме самого винчестера, и некоторые модели жестких дисков даже содержат в прошивке специальную программу Self Scan, как раз и предназначенную для этих целей. Да, она действительно рассчитывает адаптивы, но... при этом уничтожает всю содержащуюся на жестком диске информацию, что делает ее непригодной для наших целей.

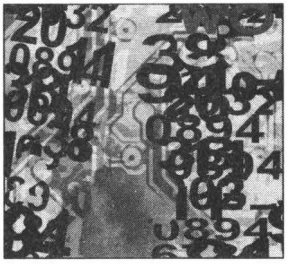
Адаптивы могут храниться как на самом диске в служебной зоне (и тогда смена плат проходит на ура, но не работает hot-swap), либо в микросхеме FLASH-ROM, которую перед заменой плат следует перепаять. Диски без адаптивов встречаются все реже и реже, можно сказать, что практически вообще не встречаются, хотя вероятность наткнуться на них в деле восстановления данных по-прежнему существует.



ЧАСТЬ II

Автоматическое и ручное восстановление данных с жестких дисков

ГЛАВА 6



Основные концепции восстановления данных

При всей своей надежности файловая система NTFS не застрахована от потрясений. Ошибки пользователя, вирусы, сбои питания, зависания ОС, дефекты поверхности накопителя, отказ электроники — любой из этих факторов может стать причиной повреждения, а то и разрушения файловой системы. С каждым днем человечество все сильнее и сильнее зависит от компьютеров, объемы жестких дисков стремительно растут, а с ними растет и ценность содержащихся на них данных, потеря которых зачастую невосполнима.

Спрос рождает предложение, и на рынке информационных услуг постоянно появляются фирмы, специализирующиеся на восстановлении данных. К сожалению, действительно квалифицированных специалистов можно встретить лишь в некоторых из них. Многие лишь создают видимость кипучей деятельности, выставляя астрономические счета при посредственном качестве восстановления. Но время кустарей уже ушло. Рабочая атмосфера изменилась. Структура NTFS хорошо документирована, разработан достойный инструментарий для восстановления информации. За минувшее время накопился огромный опыт по борьбе за спасение данных, частью которого мы и хотим поделиться с читателями.

Что делать в случае катастрофической потери данных

Прежде всего — не паникуйте! Заниматься восстановлением можно только на трезвую голову. Непродуманные, лихорадочные действия лишь усугубляют ваше и без того незавидное положение.

Не используйте непроверенные утилиты, если не уверены в их эффективности. Последствия такого "лечения" могут быть катастрофическими, а результаты "восстановления" — необратимыми. То же самое относится и к "специалистам", обитающим в фирмах непонятного происхождения и орудующим все теми же автоматизированными утилитами, которыми вы и сами можете воспользоваться. Некоторые пытаются создавать необходимый инструментарий самостоятельно. Чаще всего он

оказывается неработоспособным еще с рождения, но зато какая гордость для фирмы! Какое впечатляющее средство демонстрации собственной "крутизны"! Часто маркетологи этих фирм абсолютно необоснованно заявляют, что их разработка превосходит все имеющиеся утилиты, вместе взятые, как коммерческие, так и условно-бесплатные. Но поверьте, что хорошо известные и давно представленные на рынке утилиты тоже писали отнюдь не профаны.

Ничего не записывайте на восстанавливаемый диск и не позволяйте делать это остальным приложениям! Если вы случайно удалили файл с системного диска, ни в коем случае не выходите из Windows официально предписанным способом. Лучше нажмите кнопку RESET. Почему я даю такую "неправильную" рекомендацию? Она "некорректна" только на первый взгляд, а на самом деле это самый полезный совет, который только можно дать. Дело в том, что при штатном завершении сеанса система сохраняет на диске текущую конфигурацию, существенно увеличивая риск необратимого затирания удаленного файла.

Не пытайтесь "мучить" сбойные сектора многократными попытками чтения, т. к. это лишь расширяет дефектную область на соседние сектора и может даже привести к повреждению диска. Если винчестер издает подозрительные звуки вроде постукивания или скрежета, немедленно отключите питание компьютера (опять-таки, не позволяя системе ничего писать на диск), поскольку винчестер может доломаться окончательно в любой момент, и тогда ему уже никакой электронщик не поможет.

Диски RAID лучше восстанавливать на "родном" контроллере, т. к. разные контроллеры могут использовать различные схемы трансляции адресов. Если же контроллер отказал, его следует либо отремонтировать, либо заменить абсолютно идентичным. С дисками IDE и SATA в этом плане возникает гораздо меньше проблем, т. к. их контроллеры стандартизованы. Наконец, если данные восстановить так и не удалось — не расстраивайтесь. Во всех жизненных ситуациях надо видеть и хорошие стороны, даже когда ничего хорошего ожидать не приходится.

Основные сведения о структуре диска

Физически жесткий диск представляет собой запечатанный корпус, содержащий одну или несколько одно- или двусторонних пластин, насаженных на шпиндель. Чтение и запись данных осуществляются блоком магнитных головок, каждая из которых обслуживает одну из поверхностей пластины. Информация хранится на дорожках в форме концентрических колец, называемых *треками* (track). Треки, расположенные на равном расстоянии от центра всех пластин, образуют *цилиндр* (cylinder). Фрагмент трека, образованный радиальным делением, называется *сектором* (sector). В современных винчестерах количество секторов на трек не остается постоянным. Напротив, оно дискретно возрастает по мере удаления от центра пластины, таким образом, чтобы линейные размеры сектора оставались более или менее постоянными. Треки и головки нумеруются, начиная с нуля, а нумерация секторов начинается с единицы. Размер сектора для жестких дисков составляет 512 байт или 4096 байт (4 Кбайт). Для более эффективного использования дисково-

го пространства операционная система может объединять сектора в *кластеры* (cluster).

Твердотельные накопители SSD вместо металлических пластин использует перезаписываемые микросхемы памяти типа NAND или DRAM, за работу которых отвечает управляющий контроллер. Там нет физически движущихся и вращающихся частей, в силу чего такие накопители считаются более защищенными от неблагоприятных механических воздействий. Вместе с тем, если из строя выйдет механическая часть "традиционного" жесткого диска, есть шанс спасти информацию, установив уцелевшие "блины" в корпус диска-донора, или, наоборот, поменяв на винчестере сгоревшую плату управления — в зависимости от того, что именно сломалось. Если же сгорела микросхема памяти SSD, восстанавливать уже будет, скорее всего, нечего. SSD, в отличие от жестких дисков, работают беззвучно, обеспечивают высокую скорость обмена информацией — до 3800 мегабайт в секунду, обладают меньшим весом и более устойчивы к физическим воздействиям, например к падению. Вместе с тем они пока еще обладают меньшим ресурсом по числу циклов чтения-записи по сравнению с HDD.

С логической точки зрения структура твердотельных накопителей унаследована от обычных жестких дисков, что оправдано с точки зрения совместимости: компьютер воспринимает SSD как обычный винчестер и работает с ним так же. Вместе с тем Windows, начиная с версии 7, поддерживает специальный протокол NVMe (Non-Volatile Memory Express), ориентированный именно на работу с твердотельными дисками. Многие современные нетбуки и ноутбуки поставляются с SSD уже с завода и не позволяют установить обычный жесткий диск форм-фактора 2,5 дюйма. Постепенный переход от HDD к SSD обусловлен, прежде всего, скоростью работы последних — с повышением быстродействия процессоров и оперативной памяти медленные и неповоротливые винчестеры становятся "узким местом" компьютера, снижая скорость работы системы в целом. Решением проблемы и является переход на твердотельную технологию.

Первой схемой адресации секторов, доставшейся жестким дискам в наследство от дискет, стала так называемая *CHS-адресация*, представляющая собой сокращение от Cylinder/Head/Sector (Цилиндр/Головка/Сектор). Данная схема адресации возникла в силу экономических причин. Когда-то координаты адресуемого сектора непосредственно соответствовали физической действительности, что упрощало и удешевляло дисковый контроллер, не требуя от него никакого интеллектуального поведения. В такой схеме количество секторов в треке должно быть постоянным для всего диска, а в современных винчестерах это не так — именно поэтому уже на дисках объемом более 512 Мбайт схема CHS перестала соответствовать действительности. Параметры диска, сообщаемые устройством и напечатанные на этикетке, *всегда* виртуальны, и узнать реальное положение дел невозможно. Именно поэтому встроенные контроллеры жестких дисков обеспечивают преобразование виртуальных логических адресов в физические. Благодаря тому, что за работу логики диска отвечает интегрированный контроллер, они в наименьшей степени зависят от внешнего мира и могут свободно переноситься с компьютера на компьютер. Соот-

ветственно, логическую геометрию диска при желании можно изменить программными средствами, если в этом по каким-либо причинам возникла необходимость.

Современные контроллеры автоматически замещают плохие сектора, либо сохраняя эту информацию в своей энергонезависимой памяти, либо записывая ее в сектора инженерной зоны самого диска, а также выполняют переназначение секторов собственными средствами. Кроме того, технология SMART, о которой уже упоминалось ранее, позволяет заранее выявить потенциальные проблемы со структурой диска и вовремя предпринимать меры для минимизации последствий.

Сложнее всего обстоят дела с аппаратными реализациями RAID, схема трансляции адресов которых полностью определяется контроллером. Массивы уровня 1, известные как зеркальные наборы (mirror sets), чаще всего используют сквозную (pass-through) трансляцию. Поэтому их без особых проблем можно перенести на любой другой контроллер или даже подключить в обход него. Массивы остальных уровней, в особенности RAID 3/RAID 5, как правило, оказываются неработоспособными на контроллерах другого типа. Программные реализации RAID, монтируемые Windows, хранят информацию о своей геометрии в системном реестре и не могут быть непосредственно перенесены на другие системы. Переустановка Windows, как и ее крах, уничтожает программный RAID. К счастью, эта потеря обратима.

На сегодняшний день схема трансляции CHS признана устаревшей. Современные диски стандарта SATA поддерживают механизм логической адресации LBA (Linear Block Address), последовательно нумерующий все сектора от 0 до последнего сектора диска. В накопителях IDE режим адресации LBA появился, только начиная с ATA-3, но быстро завоевал всеобщее признание. Разрядность адресации определяется устройством. В настоящее время для задания адреса блока используется 48 бит (появилось в ATA-6), что ограничивает размер диска значением 131,072 Тбайт. Соответственно, повсеместное внедрение LBA полностью освобождает системный контроллер от необходимости учитывать физическую геометрию диска (количество цилиндров, сторон (головок), секторов на дорожке).

Один физический диск может быть разбит на несколько *логических*, каждый из которых последовательно нумеруется от первого до последнего сектора "сквозной" адресацией. На уровне файловой системы операционная система адресует диск *кластерами* (cluster). Каждый кластер образован непрерывной последовательностью секторов, количество которых равно степени двойки (1, 2, 4, 8, ...). Размер кластера задается на этапе форматирования диска и в дальнейшем уже не меняется. Основное назначение кластеров — уменьшение фрагментации файлов и уменьшение разрядности служебных файловых структур. Так, файловая система NTFS, оперирующая 64-битовыми величинами, использует типичную величину кластера в 4 Кбайт для томов до 16 Тбайт. Если размер тома составляет 16–32 ТБ, размер кластера может достигать 8 Гбайт, 32–64 ТБ — до 32 Гб, 64–128 ТБ — до 32 Гб, и т. д. В отличие от секторов, кластеры нумеруются, начиная с нуля.

Главная загрузочная запись

Первые жесткие диски имели небольшой размер и форматировались практически так же, как и дискеты. Однако их объемы стремительно росли, и MS-DOS была уже не в состоянии полностью их адресовать. Для преодоления этого ограничения был введен механизм *разделов* (partitions), позволяющий разбивать один физический диск на несколько логических. Каждый из логических дисков имеет собственную файловую систему и форматируется независимо от других. За счет чего это достигается?

В первом секторе физического диска (цилиндр 0/головка 0/сектор 1) хранится специальная структура данных — главная загрузочная запись (Master Boot Record, MBR). Она состоит из двух основных частей — первичного загрузчика (master boot code) и таблицы разделов (partition table), описывающей схему разбиения и геометрию каждого из логических дисков. Схематичное изображение жесткого диска, разбитого на разделы, представлено на рис. 6.1. В конце сектора по смещению `1FE` находится сигнатура `55h AAh`, по которой BIOS определяет признак "загрузочности" сектора. Даже если вы не хотите дробить свой винчестер на части и форматируете его как один диск, присутствие MBR обязательно.

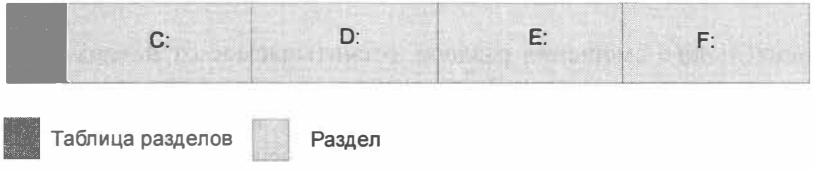


Рис. 6.1. Схематичное представление жесткого диска, разбитого на разделы

При старте компьютера BIOS выбирает загрузочный диск, который при желании можно указать в настройках, либо выбрать в меню, нажав в момент загрузки компьютера предусмотренную его спецификацией клавишу. Затем BIOS считывает первый сектор (цилиндр 0/головка 0/сектор 1) в память по адресу `0000h:7C00h`, проверяет наличие сигнатуры `55h AAh` в его конце и, если такая сигнатура действительно обнаруживается, передает управление по адресу `0000h:7C000h`. В противном случае анализируется следующее загрузочное устройство, а в случае его отсутствия выдается сообщение об ошибке.

Первичный загрузчик, получив управление, сканирует уже загруженную в память таблицу разделов, находит активный раздел (`Boot Indicator = 80h`), извлекает номер стартового сектора раздела, также называемого загрузочным сектором (boot-sector). Затем загрузчик перемещает свое тело по другому адресу, чтобы избежать затирания, загружает boot-сектор в память по адресу `0000h:7C00h`, убеждается в наличии сигнатуры `55h AAh` и передает управление на `0000h:7C00h`, в противном случае выдается сообщение об ошибке, и после нажатия на любую клавишу компьютер перезагружается. Ряд загрузчиков, выходящих за стандартные спецификации Microsoft, поддерживают несколько активных разделов, последовательно перебирая их один за другим.

Если первичный загрузчик поврежден, то BIOS не сможет запустить операционную систему с такого диска, однако при подключении его "вторым" (или при загрузке с внешнего накопителя) все логические диски будут доступны. Как минимум они должны быть "видимы". Для того чтобы это было именно так, необходимо соблюдение следующих двух условий:

1. Во-первых, файловая система соответствующего раздела должна быть известна загруженной операционной системе и не повреждена.
2. Во-вторых, загрузочный сектор должен быть в целости и сохранности (данный вопрос будет обсуждаться далее в этой главе).

Таблица разделов, которую анализирует первичный загрузчик (master boot code), а чуть позже — драйвер логических дисков операционной системы, состоит из четырех записей размером по `10h` каждая, расположенных по смещениям `1BEh`, `1CEh`, `1DEh` и `1EEh` байтов от начала диска соответственно. Каждая из них описывает свой логический раздел, задавая его стартовый и конечный сектора, записанные в формате CHS.

> ***Внимание!***
>
> **Даже если диск работает в режиме LBA, разделы в формате MBR все равно адресуются через CHS!**

Поле относительного смещения раздела, отсчитываемое от начала таблицы разделов, является вспомогательным. То же самое относится и к полю, содержащему значение общего количества секторов на диске, т. к. очевидно, что это значение может быть вычислено на основе стартового и конечного секторов. Одни операционные системы и загрузчики игнорируют вспомогательные поля, другие же их активно используют. Именно поэтому содержимое этих полей должно соответствовать действительности.

Поле идентификатора диска содержит уникальную 32-разрядную последовательность, помогающую операционной системе отличить один смонтированный диск от другого и автоматически копируемую в следующий ключ реестра: `HKLM\SYSTEM\MountedDevices`. Ранее содержимое данного поля было некритично, но, начиная с Windows 7, коллизии этих идентификаторов могут приводить к проблемам при загрузке Windows и работе с накопителями.

Поле `System ID` содержит идентификатор файловой системы, установленной на разделе. В случае NTFS этот идентификатор равен `07h`. За динамическими дисками, согласно фирменной спецификации, закреплен идентификатор `42h`. На практике это справедливо лишь для тех из них, которые были получены путем обновления (update) обычного (basic) раздела до динамического (dynamic) тома. Сведения об остальных динамических дисках в таблице разделов не хранятся, а содержатся в базе данных менеджера логических дисков (Logical Disk Manager, LDM). Загрузочный логический диск (независимо от того, динамический он или нет) в обязательном порядке должен присутствовать в таблице разделов, иначе BIOS не сможет его загрузить.

Четыре записи таблицы разделов позволяют иметь всего четыре логических диска (см. рис. 6.1). Этого явно недостаточно, но расширение таблицы разделов оказалось невозможным, т. к. последняя запись упирается в конец сектора. Разработчики сочли нежелательным задействовать следующий сектор, поскольку он активно используется многими вирусами и нестандартными драйверами. К тому же это все равно не позволяет устранить проблему радикально, а лишь оттягивает ее решение. Тогда инженеры нашли другое решение, предложив концепцию расширенных разделов (Extended partition). Если значение индикатора загрузки (`System ID`) некоторого раздела равно `05h` или `0Fh`, то такой раздел трактуется как "виртуальный физический диск", с собственной таблицей разделов, расположенной в его начале, на которую и указывает стартовый сектор расширенного раздела (рис. 6.2). Иначе говоря, таблица разделов получается вложенной, и уровень вложения ограничен разве что свободным дисковым пространством и объемом стековой памяти загрузчика (при условии, что он использует рекурсивный алгоритм сканирования). Таким образом, таблица разделов как бы "размазывается" вдоль винчестера (рис. 6.3).

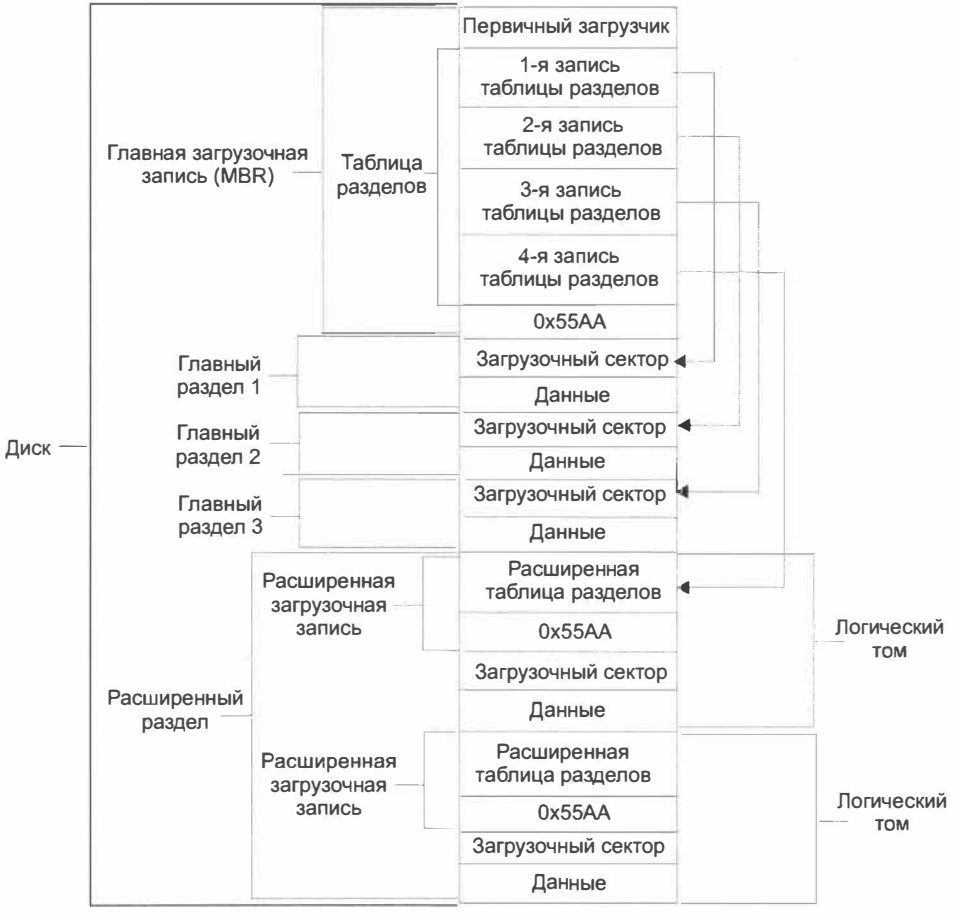


Рис. 6.2. Структурная схема типичного жесткого диска, содержащая главные (primary) и расширенные (extended) разделы

Некоторые утилиты для разбиения диска на разделы при создании логических дисков на расширенном разделе создают расширенную таблицу разделов с четырьмя записями: одна служит для описания логического раздела, вторая описывает еще один (следующий) логический раздел, а две не используются. Таким образом, получается "цепочка" таблиц разделов, в которой первая таблица разделов описывает от одного до трех основных (primary) разделов, а каждая следующая — соответствующий ей логический диск и положение следующей таблицы разделов (рис. 6.3). В результате при разбиении винчестера на четыре логических диска на нем образуются четыре таблицы разделов, хотя в данном случае можно было бы обойтись и одной.

Рис. 6.3. Расширенная таблица разделов

При установке актуальных версий Windows (начиная с Windows 7) система автоматически создает на диске специальный скрытый раздел System Reserved (Зарезервировано системой) объемом 300–600 Мбайт, который является одновременно основным и активным. В этом разделе хранится загрузчик ОС и данные, используемые загрузчиком. Увидеть этот скрытый раздел можно в оснастке **Управление дисками** окна **Управление компьютером** MMC, которое можно отыскать в блоке настроек **Администрирование** в **Панели управления** Windows (рис. 6.4). В этом же разделе находятся данные о шифровании диска, если используется BitLocker. Благодаря тому, что загрузчик теперь хранится в обособленном дисковом разделе, значительно упростилась задача установки на компьютере сразу нескольких версий Windows или нескольких различных операционных систем. Снизились и шансы случайно повредить загрузчик в ходе манипуляций с дисками. Однако если система устанавливается на компьютере, где уже присутствует максимально возможное количество основных разделов, то новый основной раздел создан не будет и загрузчик с конфигурационными файлами будет размещен в активном разделе.

Если таблица разделов повреждена, то логические диски, скорее всего, будут полностью недоступны — они не будут отображаться ни Проводником Windows (Windows Explorer), ни файловым менеджером FAR. Искажение таблицы разделов не приводит к немедленному изменению объема уже отформатированных томов, т. к. эта информация хранится в загрузочном секторе (boot sector). Однако при последующем переформатировании произойдет затирание данных из соседнего раздела или же текущий раздел окажется усеченным.

Воспользовавшись любым редактором диска, считаем первый сектор физического диска, отформатированного в MBR. В настоящее время, несмотря на широкое внедрение формата GPT, сменные носители нередко форматируются именно в MBR (и заодно с единственным разделом FAT16/32), что позволяет обеспечить наибольшую совместимость с разнообразными устройствами — от стационарных компьютеров до автомобильных магнитол и цифровых фоторамок. Сектор в формате MBR должен выглядеть приблизительно так, как показано на рис. 6.5.

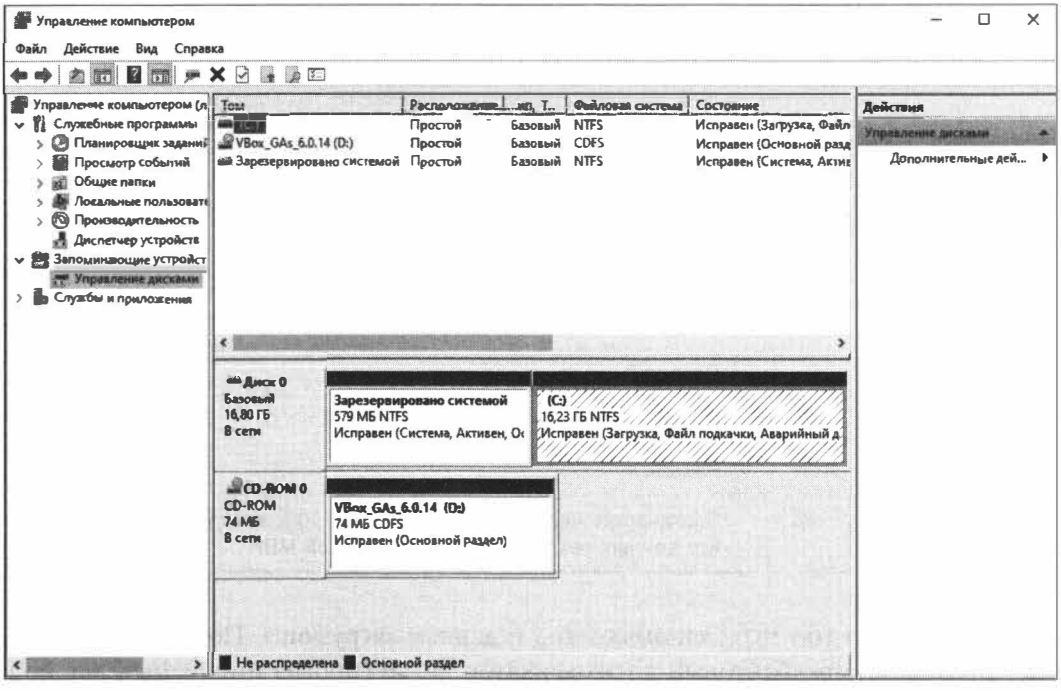


Рис. 6.4. Скрытый раздел "Зарезервировано системой", в котором хранится загрузчик Windows

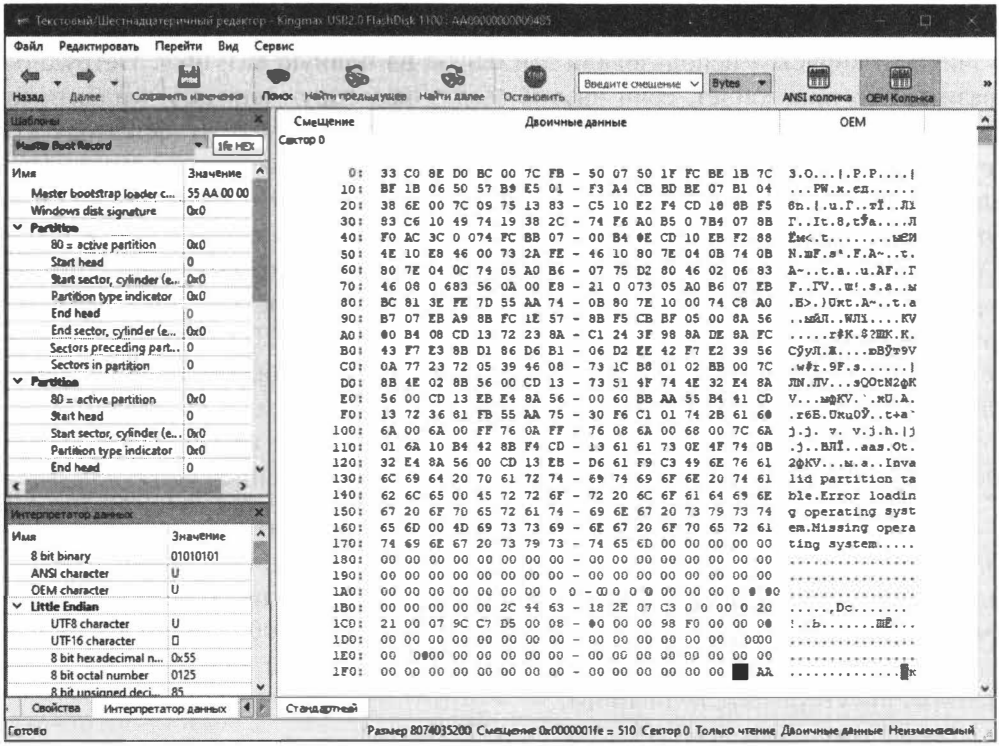


Рис. 6.5. Внешний вид MBR

Не правда ли, MBR выглядит как знаменитая Матрица? Ее формат кратко описан в табл. 6.1.

Таблица 6.1. *Формат MBR*

Смещение	Размер	Описание
0x000	перемен.	Код загрузчика
1x1B8	4h	Идентификатор диска
0x1BE	10h	Раздел 1
0x1CE	10h	Раздел 2
0x1DE	10h	Раздел 3
0x1EE	10h	Раздел 4
0x1FE	0x2	"Магическое число" — сигнатура 55h AAh, которое указывает, что данный сектор представляет собой MBR

Первые 440 байт (по 1B7h) занимают код и данные загрузчика. По смещению 1B8h расположен четырехбайтовый идентификатор диска (иногда именуемый как подпись диска — Disk Signature), принудительно назначаемый Windows. Коварство Microsoft не знает границ! Еще со времен первых IBM PC (тогда они назывались XT) загрузчик владел первыми 446 (1BEh) байтами MBR, и достаточно многие загрузчики (и вирусы!) использовали эти байты на полную катушку. Нетрудно сообразить, что произойдет, если внутрь загрузчика вдруг запишется идентификатор. Старые загрузчики это убьет! Тем не менее к этому идентификатору стали приспосабливаться остальные операционные системы и все последующие версии Windows, используя его аналогично GUID диска в GPT. Но, конечно же, это отступление от первоначальных правил не могло просто так сойти с рук, и в версиях Windows старше Vista коллизия идентификаторов дисков приводит к тому, что второй диск со значением Disk Signature, равным идентификатору уже подключенного носителя, незамедлительно отправится в режим "Не в сети". Более ранние же версии Windows просто переназначали неугодный идентификатор, так что пользователь не замечал возникающей проблемы. Ситуация с коллизией идентификаторов может произойти, например, при посекторном клонировании накопителей и попытке работать одновременно с оригиналом и копией.

Со смещения 1BEh начинается таблица разделов. Каждая из четырех записей этой таблицы описывает свой логический диск, что позволяет нам создавать до четырех разделов на накопителе. Формат записи таблицы разделов представлен в табл. 6.2. Динамические диски, впервые появившиеся в Windows 2000, хранятся в базе данных Менеджера Логических Дисков (Logical Disk Manager Database) и в таблице разделов присутствовать не обязаны.

Таблица 6.2. *Формат записи таблицы разделов*

Смещение					Размер	Описание
000	1BE	1CE	1DE	1EE	byte	Флаг активного загрузочного раздела (Boot Indicator). 80h — загрузочный раздел, 00h — незагрузочный раздел
001	1BF	1CF	1DF	1EF		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 (System ID), см. табл. 6.3
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		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	Количество секторов раздела

Таблица 6.3. *Возможные значения System ID*

System ID	Тип раздела
00h	Раздел свободен
0x01	Раздел FAT12 (менее чем 32 680 секторов в томе или 16 Мбайт)
0x04	Раздел FAT16 (32 680–65 535 секторов или 16–33 Мбайт)
0x05	Расширенный раздел (extended partition)
0x07	Раздел NTFS
0x0B	Раздел FAT32
0x42	Динамический диск
0x82	Раздел подкачки в Linux (Swap)
0x83	Раздел Linux (ext2/3/4)
0xEE	Защитный GPT-раздел
0xEF	Системный раздел UEFI

Техника восстановления главной загрузочной записи

В большинстве случаев в современных реалиях нет никакой необходимости восстанавливать поврежденную загрузочную запись вручную с использованием отладчиков или редакторов диска — с этой задачей превосходно справляются средства Windows 10, а также специальные инструменты, созданные независимыми разработчиками. Более того, при ручном восстановлении содержимого секторов диска гораздо больше шансов окончательно погубить загрузчик, чем восстановить его.

За 20 лет, минувших с момента выхода на рынок "пользовательской" версии операционной системы на платформе NT — Windows 2000, эта ОС проделала огромный эволюционный путь, значительно прибавив в надежности. Windows обзавелась целым арсеналом средств, позволяющих восстановить ее работоспособность в случае аварии или непредвиденных сбоев, не прибегая к "шаманским" методам, которые применялись в более ранних версиях.

Итак, если однажды вы включили свой компьютер или ноутбук и увидели на экране грозную надпись `No bootable device found. Insert boot disk and press any key` либо `An operating system wasn't found`, не спешите паниковать. Описанные далее действия должны помочь вам вернуть все как было.

> ***Примечание***
>
> Очень важно отличать сообщение BIOS от сообщений первичного загрузчика и загрузочного сектора. Зайдите в BIOS Setup и отключите все загрузочные устройства, оставив активным только съемный диск. А теперь перезагрузитесь и запомните, какое сообщение появится на экране. Это и будет "ругательством" BIOS.

Способ восстановления нормальной загрузки Windows достаточно прост. Для начала откройте настройки BIOS и проверьте, виден ли там жесткий диск. Если он перестал определяться, возможно, возникли аппаратные проблемы с самим устройством, разъемом или проводами, при помощи которых диск подключен к плате. Проверьте соединения, а еще лучше — отключите и подключите диск снова: возможно, после этого проблема исчезнет. Если этого не произошло, отключите все внешние накопители от USB-портов вашего компьютера и попробуйте загрузиться снова. Если и это не помогло, переходите к следующим действиям.

Настройте загрузку системы со съемного накопителя — флешки или оптического диска, после чего подключите этот накопитель к компьютеру (либо просто вставьте диск в привод). Дистрибутив Windows 10 должен быть той же разрядности, что и установленная операционная система, у которой "сломался" загрузчик. Если у вас нет такого диска, его образ можно скачать с сайта Microsoft по следующей ссылке: **https://www.microsoft.com/ru-ru/software-download**. После включения питания дождитесь появления на экране сообщения `Press any key to boot from CD or DVD...` и нажмите любую клавишу на клавиатуре компьютера.

После появления на экране окна приветствия программы установки Windows, щелкните мышью на надписи **Восстановление системы** в левом нижнем углу

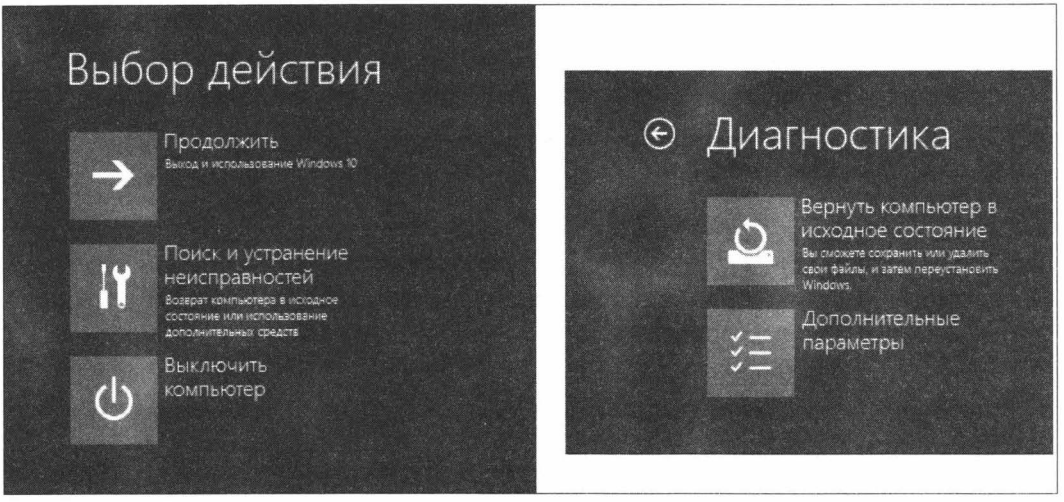


Рис. 6.6. Первый этап автоматического восстановления загрузки Windows

окна. Затем щелкните мышью на надписи **Поиск и устранение неисправностей**, затем — **Дополнительные параметры** (рис. 6.6).

В окне **Дополнительные параметры** нажмите на надпись **Восстановление при загрузке** (рис. 6.7). После этого будет автоматически выполнено восстановление загрузочной записи и устранение неполадок, мешающих загрузке Windows.

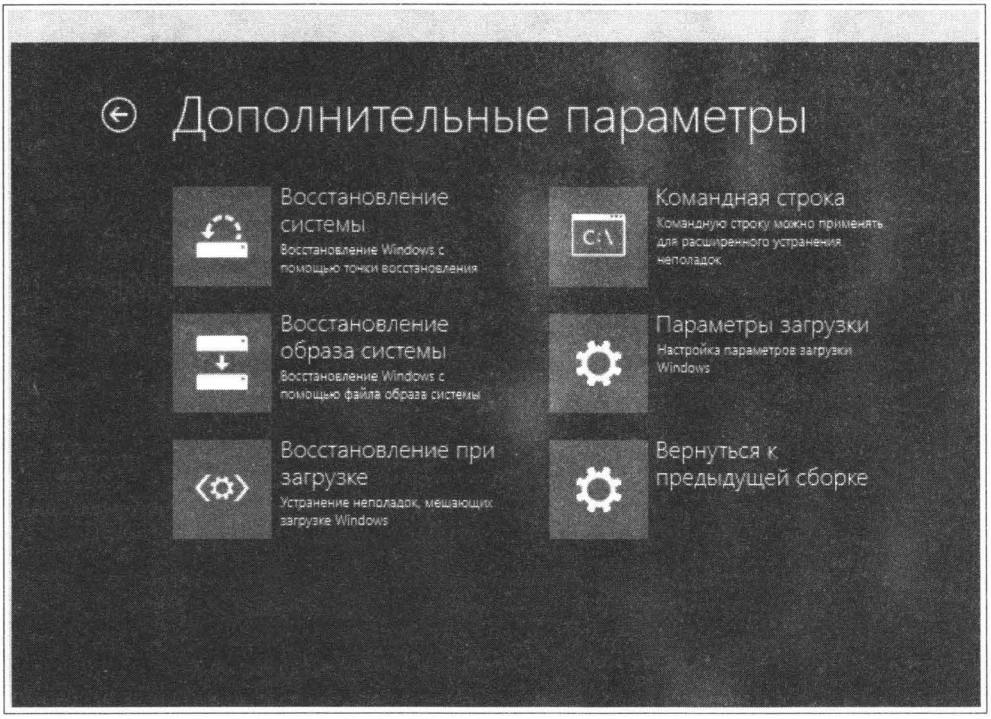


Рис. 6.7. Дополнительные параметры восстановления Windows

Назначение других вариантов, предложенных в этом окне, следующее:

- Восстановление системы — использование стандартной программы Восстановление системы и восстановление Windows с помощью точек восстановления;
- Восстановление образа системы — восстановление Windows из созданного ранее образа диска;
- Командная строка — переход в режим командной строки (потребуется выбрать учетную запись, от имени которой будет запущен командный интерпретатор);
- Параметры загрузки — выбор дополнительных параметров загрузки операционной системы и их настройка;
- Вернуться к предыдущей сборке — "откат" к предыдущей версии Windows, этот режим подходит для отказа от использования недавно установленных обновлений операционной системы.

Если восстановить загрузчик описанным выше способом не удалось, можно переходить к следующему методу. Снова загрузитесь с дистрибутивного диска, и при появлении на экране окна приветствия программы установки Windows нажмите сочетание клавиш <Shift>+<F10>. Откроется окно командной строки (рис. 6.8).

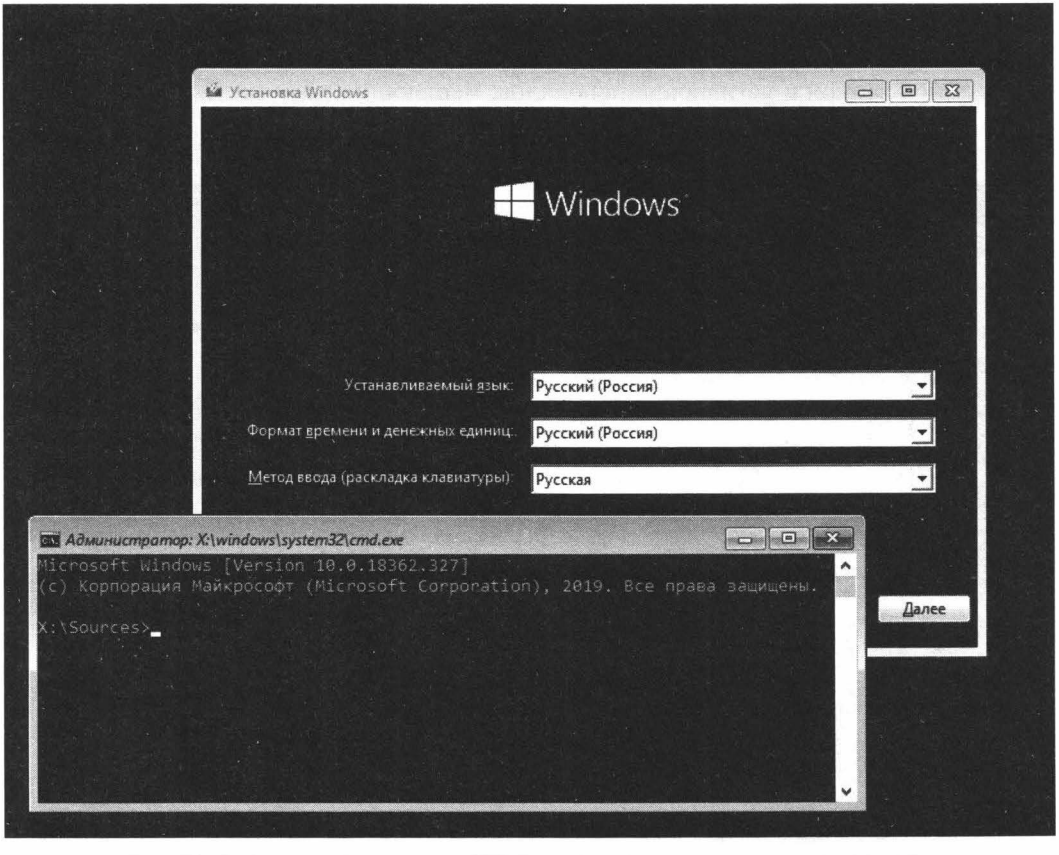


Рис. 6.8. Восстановление загрузки Windows с использованием командной строки

Введите в командной строке команду `bootrec /fixmbr`, нажмите <Enter> и перезагрузите машину. Скорее всего, Windows запустится в штатном режиме, но, возможно, загрузка завершится с ошибкой `0xc000000e`, либо система продемонстрирует вам синий экран с надписью *Your PC/Device needs to be repaired. A required device isn't connected or cant't be accessed. The application or operating system couldn't loaded because a required file is missing or contains errors. File: \Windows\system32\winload.exe*. Связано это с некорректной конфигурацией загрузчика.

Чтобы исправить эту ошибку, повторите предыдущие действия до открытия на экране окна командной строки. Введите команду `bootrec /rebuildbcd` и дождитесь, пока программа восстановит конфигурацию загрузочной записи (рис. 6.9).

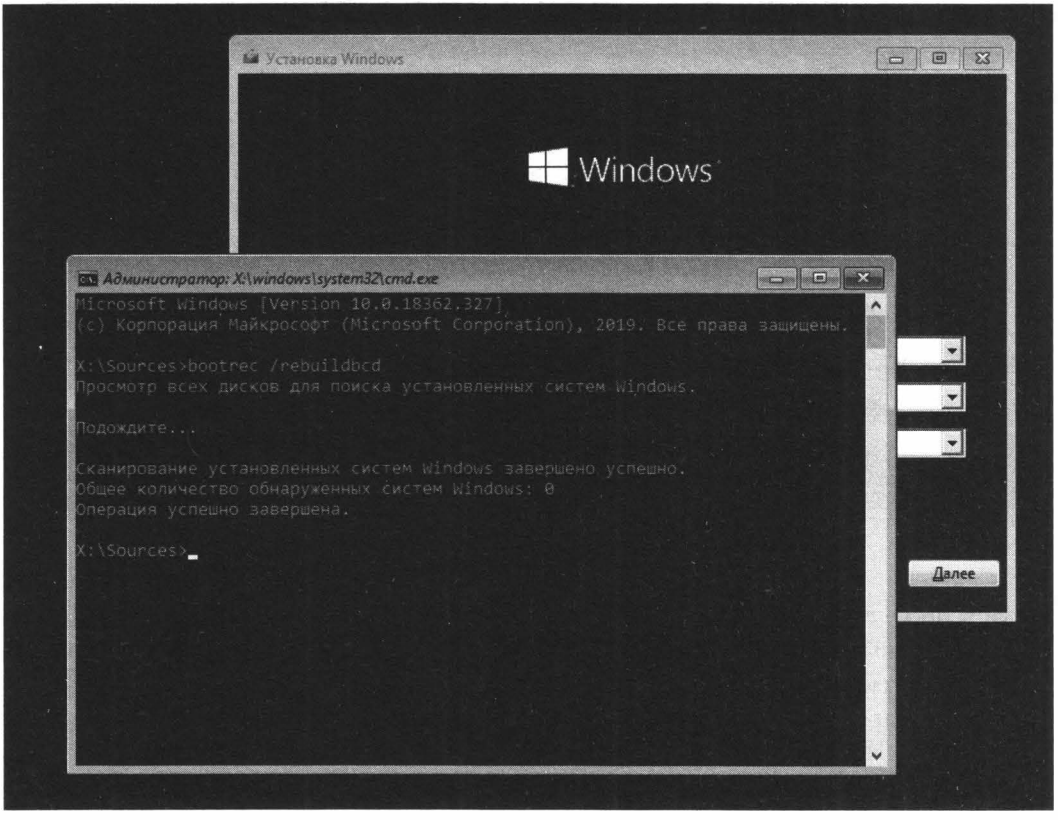


Рис 6.9. Исправляем некорректные данные загрузчика

Теперь нужно проверить, что у нас получилось. Введите в командной строке команду `bcdedit` (рис. 6.10).

В разделе Загрузка Windows должны быть корректно заполнены все строки, в них не должно содержаться пустых значений. Перезагрузите компьютер — теперь все должно быть в порядке.

Ситуация, когда диск, ранее размеченный в GPT, переформатируют в MBR (сам формат GPT будет рассмотрен в одном из следующих разделов), тоже может при-

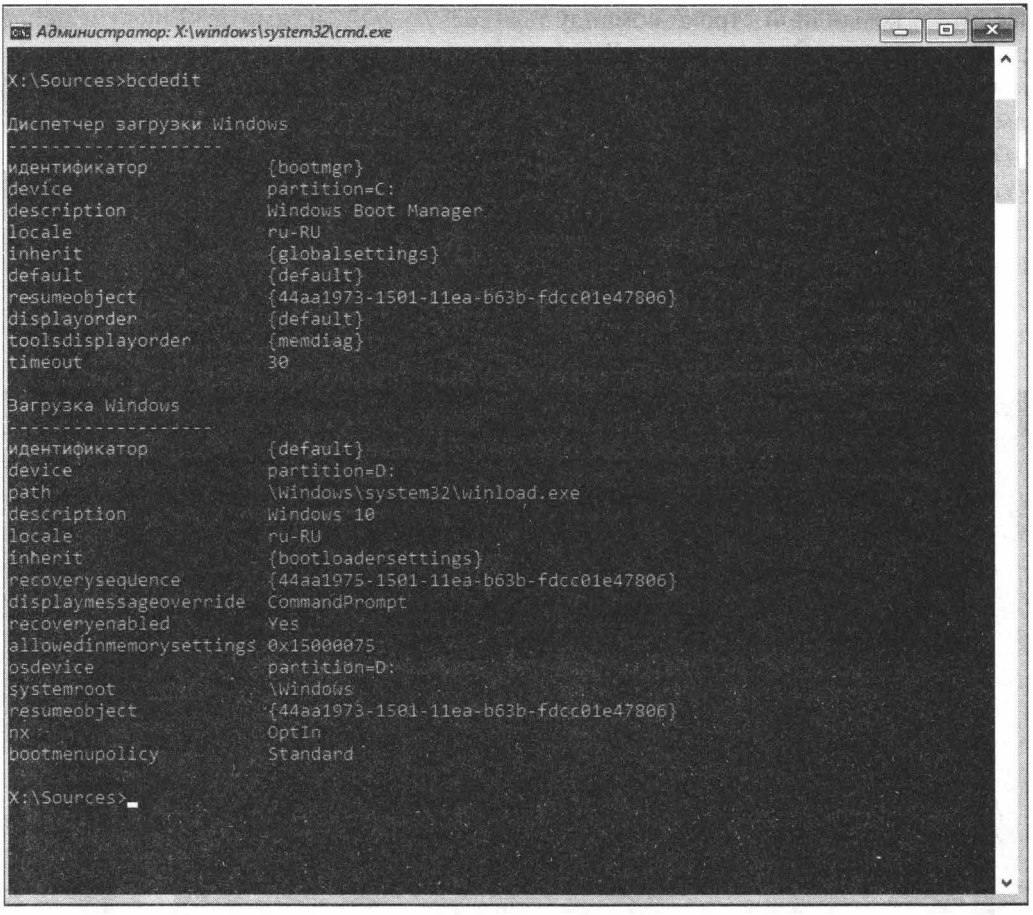


Рис. 6.10. Вывод команды `bcdedit`

вести к проблемам. В Windows утилита **Управление дисками** предлагает без лишних хлопот выполнить операцию "Преобразовать в MBR-диск". Однако если при этом преобразовании вдруг останутся нетронутыми служебные структуры GPT, то ПО может определить такой носитель как проблемный и откажется с ним нормально работать. И немудрено, ведь на нем находятся вполне валидные записи и MBR, и GPT; и оно не знает, чему из этого верить. К примеру, при попытке перечитать таблицу разделов такого неоднозначного "двойного" диска командой `partprobe` в Linux появляется следующее сообщение: `Warning: /dev/sdc contains GPT signatures, indicating that it has a GPT table. However, it does not have a valid fake msdos partition table, as it should. Perhaps it was corrupted -- possibly by a program that doesn't understand GPT partition tables. Or perhaps you deleted the GPT table, and are now using an msdos partition table. Is this a GPT partition table?` Чтобы дальше использовать носитель в стиле старой разметки, потребуется стереть таблицы GPT. Можно было бы просто обнулить первые 34 логических блока (LBA), но это не избавит нас от запасных таблиц, расположенных в конце диска, из которых легко и непринужденно восстановятся первичные. Тут на помощь придет утилита gdisk, позволяющая затереть как раз все, что

нужно, ни больше, ни меньше. Пример такой "зачистки" показан на рис. 6.11. Ее можно выполнить как до, так и после создания корректной загрузочной записи, но во избежание проблем лучше, безусловно, обо всем позаботиться заранее. Предупреждение о пересечении раздела и таблицы для нас неважно: gdisk при работе с носителями создает в памяти структуры GPT для дальнейшей работы, даже если на самом диске их нет, но он не записывает их на диск, пока не выполнить команду w.

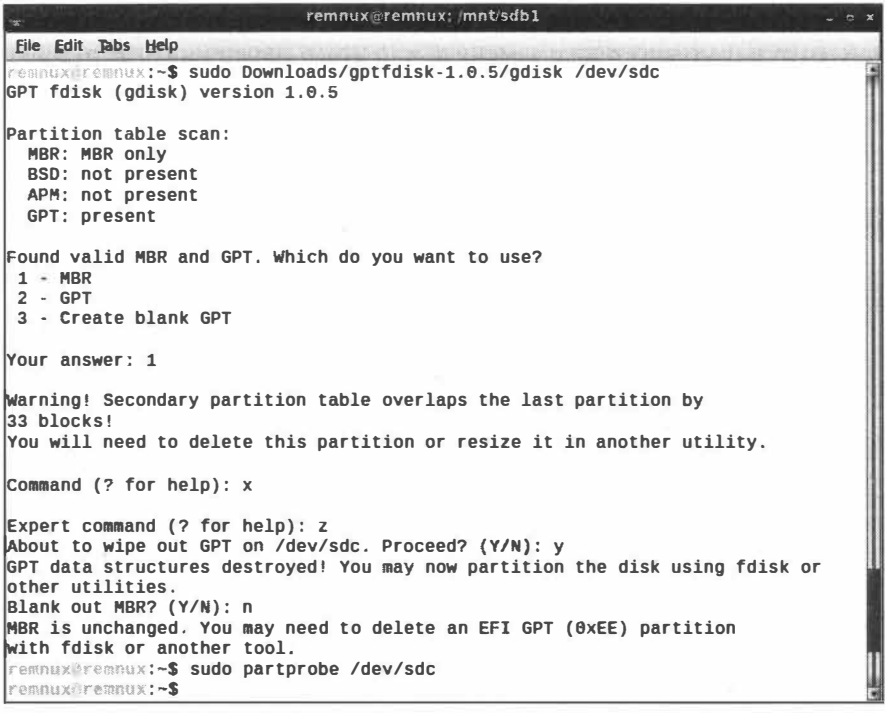


Рис. 6.11. Вычищаем структуры GPT с MBR-диска при помощи gdisk

В ОС семейства Linux доступна замечательная утилита TestDisk, способная помочь в восстановлении главной загрузочной записи и ее таблицы разделов (хотя, по правде сказать, утилита эта кросс-платформенная). Она позволяет перезаписать код первичного загрузчика и искать утраченные разделы, а также восстанавливать их после переформатирования в другую файловую систему.

Для поиска разделов нужно указать в меню восстанавливаемый носитель и его тип разметки (тип `Intel`, как ни странно, в данной программе означает именно MBR-разметку, а ведь и GPT непосредственно связана с Intel). Далее, после выбора пункта Analyse начнется поиск разделов, которые будут появляться по мере обнаружения. Поиск можно прервать, если все нужные разделы появились в списке. Если же вы считаете, что найдены не все имеющиеся разделы, можно попробовать выполнить более тщательный поиск, выбрав Deeper Search.

Для каждого найденного раздела TestDisk предлагает просмотреть список файлов на нем, нажав <P>, чтобы убедиться в правильности результата, поэтому иногда

TestDisk находит и сознательно удаленные пользователем разделы, к настоящему моменту содержащие только мусор. Внизу отображается размер выбранного раздела (рис. 6.12). После того как все искомые разделы отобразились в списке, запись полученной таблицы разделов на диск осуществляется выбором пункта **Write**. Восстановить код первичного загрузчика можно в главном меню, выбрав **MRB code**. Код, записываемый утилитой, пробует загрузиться с первого раздела диска. Если возникает проблема, то отобразится "приглашение" вида `1234F:`. При нажатии одной из этих клавиш на клавиатуре загрузчик пытается загрузиться из загрузочного сектора 1–4-го разделов или флоппи-привода соответственно. При желании после успешной загрузки можно также заменить код этого первичного загрузчика на тот, который вам больше нравится. TestDisk по умолчанию присутствует во множестве дистрибутивов Linux и в Live CD, предназначенных для восстановления систем. Программа доступна по адресу **https://www.cgsecurity.org/wiki/TestDisk_Download**.

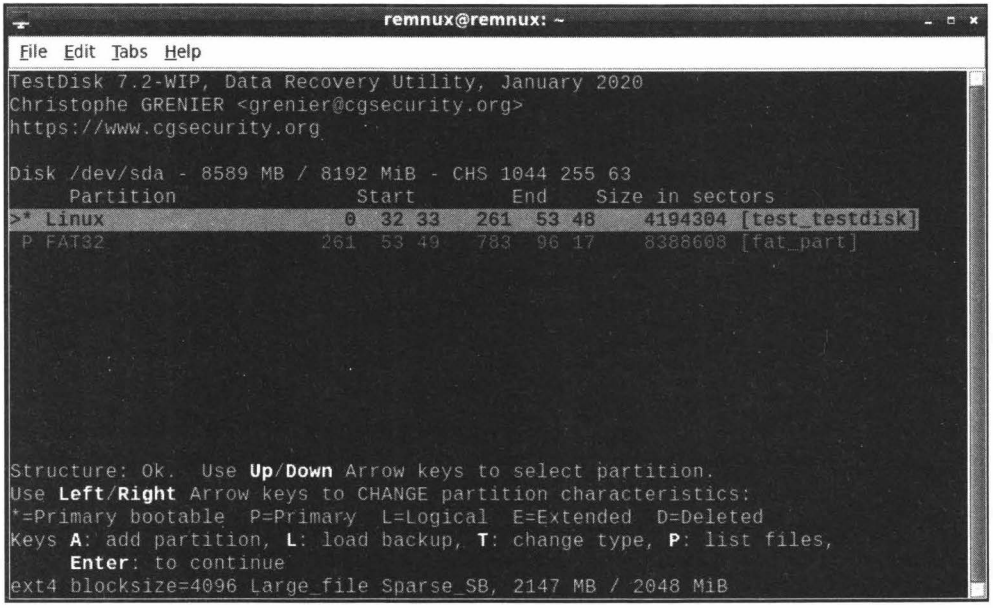


Рис. 6.12. TestDisk обнаружил два раздела на диске, не содержащем валидную MBR

Проблемы с загрузкой могут принести и нестандартные загрузчики. Некоторые из них изменяют схему трансляции адресов жесткого диска, поэтому со штатным загрузчиком такой диск окажется неработоспособным. Попробуйте переустановить загрузчик с дистрибутивных дисков — быть может, это поможет. В противном случае ничего не остается, как писать свой собственный загрузчик, определять текущую геометрию диска и соответствующим образом транслировать секторные адреса. Это довольно сложная задача, требующая серьезной подготовки, и здесь ее лучше не обсуждать.

Если загрузчик выводит сообщение `Invalid partition table`, то это еще не значит, что таблица разделов действительно повреждена. Вполне возможно, что на самом деле таблица разделов цела, но просто ни один из основных разделов не назначен

активным. Такое случается при использовании нестандартных загрузчиков, загружающих операционную систему из расширенного раздела. После выполнения команды `FDISK /MBR` или при установке операционной системы, автоматически заменяющей первичный загрузчик своим собственным, этот новый загрузчик не обнаружит в пределах досягаемости ни одного активного раздела и, что вполне естественно, сигнализирует вам об этом. Для решения проблемы либо восстановите прежний загрузчик, либо установите операционную систему на первичный раздел и, запустив FDISK, сделайте его активным.

Загрузитесь с "живого" образа и проверьте, видимы ли ваши логические диски. Если да, то смело переходите к следующему пункту, в противном случае соберитесь с духом и приготовьтесь немного поработать руками и головой.

Восстановление основного раздела, созданного с помощью FDISK или Disk Manager, в большинстве случаев осуществляется элементарно, а остальные, как правило, восстанавливать и не требуется, поскольку именно MBR гибнет чаще всего, а расширенные разделы, рассредоточенные по всему диску, погибают разве что при явном удалении разделов средствами FDISK или Disk Manager.

Адрес стартового сектора первого логического диска всегда равен 0/1/1 (Cylinder/Head/Sector), относительный (Relative) сектор — количеству головок жесткого диска, уменьшенному на единицу.

> ***Рекомендация***
>
> Сведения о геометрии диска можно почерпнуть из любого дискового редактора — например, Sector Inspector.

Конечный сектор определить несколько сложнее. Если загрузочный сектор цел (более подробно этот вопрос будет обсуждаться далее в этой главе), то узнать количество секторов в разделе (total sectors) можно на основании значения поля `BootRecord.NumberSectors`, увеличив его значение на единицу. Тогда номер конечного цилиндра будет равен `LastCyl = TotalSectors/(Heads*SecPerTrack)`, где `Heads` — количество головок на физическом диске, а `SecPerTrack` — количество секторов на трек. Номер конечной головки равен `LastHead = (Total Sector – (LastCyl*Heads SecPerTrack))/SecPerTrack`, а номер конечного сектора равен `LastSec = (Total Sector – (LastCyl*Heads SecPerTrack)) % SecPerTrack`. Пропишите полученные значения в MBR и проверьте, не находится ли за вычисленным концом раздела следующий раздел? Это должна быть либо расширенная таблица разделов, либо загрузочный сектор. Если это так, создайте еще одну запись в таблице разделов, заполнив ее соответствующим образом.

А теперь зададимся вопросом: можно ли восстановить таблицу разделов, если загрузочный сектор отсутствует и восстановить его не представляется возможным, а автоматизированные утилиты не помогли? Это вполне реалистичная задача. Необходимо лишь найти загрузочные сектора или расширенные таблицы разделов, принадлежащие последующим разделам. В этом вам поможет контекстный поиск. Ищите сектора, содержащие сигнатуру `55h AAh` в конце. Отличить загрузочный сектор от расширенной таблицы разделов очень просто. В загрузочном секторе по

смещению три байта от его начала расположен идентификатор производителя (OEM ID), например NTFS, MSWIN4.1 и т. д. Размер текущего раздела будет на один сектор меньше. Теперь, зная размер и геометрию диска, можно рассчитать и конечный цилиндр/головку/сектор.

Имейте в виду, что Windows хранит копию загрузочного сектора, которая, в зависимости от версии, может быть расположена либо в середине раздела, либо в его конце. Другие копии могут находиться в архивных файлах и файле подкачки. Как отличить копию сектора от оригинала? Элементарно, Ватсон! Если это подлинник, то вслед за ним пойдут служебные структуры файловой системы (в частности, для NTFS это будет MFT, каждая запись которой начинается с легко узнаваемой строки FILE*). К счастью, служебные структуры файловой системы обычно располагаются на более или менее предсказуемом смещении относительно начала раздела, и, отталкиваясь от их "географического" расположения, мы можем установить размеры каждого из логических дисков, даже если все-все-все загрузочные сектора и расширенные таблицы разделов уничтожены.

Что произойдет, если границы разделов окажутся определенными неверно? Если мы переборщим, увеличив размер раздела сверх необходимого, все будет нормально работать, поскольку карта свободного пространства хранится в специальной структуре (у NTFS это файл $bitmap, а у FAT13/32 — непосредственно сама FAT) и "запредельные" сектора будут добавлены только после переформатирования раздела. Если все, что нам требуется, — это скопировать данные с восстанавливаемого диска на другой носитель, то возиться с подгонкой параметров таблицы разделов не нужно! Распахните ее на весь физический диск — и дело с концом!

Естественно, такой способ восстановления подходит только для первого раздела диска, а для всех последующих нам потребуется определить стартовый сектор. Это определение должно быть очень точным, поскольку все структуры файловой системы адресуются от начала логического диска и ошибка в один-единственный сектор сделает весь этот тонкий механизм полностью неработоспособным. К счастью, некоторые из структур ссылаются сами на себя, давая нам ключ к разгадке. В частности, файлы $mft/$mftmirr содержат номер своего первого кластера. Стоит нам найти первую запись FILE*, как мы узнаем, на каком именно секторе мы сейчас находимся (конечно, при условии, что сумеем определить количество секторов на кластер, но это уже другая тема, которая более подробно будет обсуждаться далее в этой главе).

Проблема нулевой дорожки

Главная загрузочная запись жестко держит за собой первый сектор диска, и если он вдруг окажется разрушенным, работа с таким диском станет невозможной. Ранее проблема решалась посекторным копированием винчестера, переносом данных на здоровый жесткий диск с идентичной геометрией и последующим восстановлением MBR.

Сейчас ситуация изменилась. Современные винчестеры поддерживают возможность принудительного замещения плохих секторов из резервного фонда (а некото-

рые делают это автоматически), поэтому проблема нулевой дорожки, преследующая нас еще со времен гибких дисков и 8-разрядных машин, уже перестала существовать.

Как правило, замещением плохо читаемых секторов жесткие диски занимаются самостоятельно, но их также можно переназначить по старинке специальными утилитами, предоставляемыми производителем конкретной модели винчестера. Чаще всего они распространяются бесплатно и могут быть свободно найдены в Сети.

В завершение рассказа о главной загрузочной записи в качестве дополнительного чтения мы рекомендуем вам несколько весьма интересных источников. Вот они:

- *MBR and OS Boot Records* — масса интересного материала по MBR (на английском языке): **http://thestarman.pcministry.com/asm/mbr/index.html**.
- *Ralf Brown Interrupt List* (рис. 6.13) — знаменитый "Список прерываний" (Interrupt List) Ральфа Брауна (Ralf Brown), описывающий все прерывания, включая недокументированные (на английском языке): **http://www.pobox.com/~ralf**.

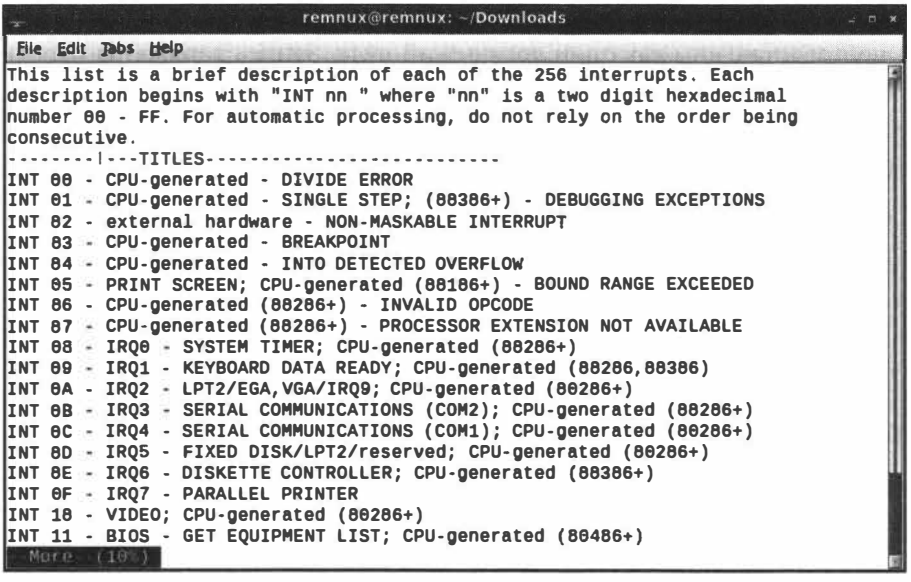


Рис. 6.13. Просмотр легендарного "Списка прерываний" Ральфа Брауна

- OpenBIOS — проект "Открытого BIOS", распространяемого в исходных текстах. Помогает понять некоторые неочевидные моменты обработки системного загрузчика: **https://www.openbios.org/OpenBIOS**.

Динамические диски

Динамические диски, впервые появившиеся в Windows 2000 и доступные в современных "профессиональных" версиях этой ОС, представляют собой все тот же программный RAID, призванный преодолеть ограничения стандартных механизмов

разбиения диска на разделы. Он учитывает ошибки своего прямого предшественника — программных реализаций RAID в Windows NT, хранящих конфигурационную информацию в системном реестре. Этот недостаток препятствовал переносу RAID с компьютера на компьютер и делал эти массивы чрезвычайно чувствительными к повреждениям реестра.

Типы динамических дисков, поддерживаемые Windows

Начиная с Windows 2000, операционные системы этого семейства поддерживают следующие типы динамических дисков:

- *Простые* (simple) — практически ничем не отличаются от обычных разделов, за исключением того, при переразбиении диска отпадает необходимость в перезагрузке. Данный тип является базовым для всех остальных динамических дисков.
  - Избыточность данных — отсутствует;
  - Эффективность — низкая.
- *Составные* (spanned) — состоят из одного или нескольких простых дисков, находящихся в различных разделах или даже на физически различных устройствах, но представлены как один логический диск. Запись данных на простые диски осуществляется последовательно (т. е. составной диск представляет собой классический линейный RAID).
  - Избыточность данных — отсутствует;
  - Эффективность — низкая.
- *Чередующиеся* (striped) — чередующиеся динамические диски аналогичны составным, но данные записываются параллельно на все простые диски. При условии, что простые диски расположены на различных каналах контроллера, это значительно увеличивает скорость обмена данными. Иными словами, чередующиеся динамические диски представляют собой классическую реализацию RAID уровня 0.
  - Избыточность данных — отсутствует;
  - Эффективность — средняя.
- *Зеркальные* (mirrored) — зеркальные динамические тома представляют собой два простых диска, расположенных на разных устройствах. Данные дублируются на оба носителя (RAID уровня 1).
  - Избыточность данных — средняя;
  - Эффективность — средняя.
- *Чередующиеся с контролем четности* (striped with parity) — соответствуют RAID уровня 5. Такие тома состоят из трех или большего количества дисков. Фактически они представляют собой чередующиеся тома, на которых реализован контроль ошибок. Запись данных ведется на два диска, в два блока, а на третий диск и в третий блок записывается код коррекции ошибок (Error Correction

Code, ECC), с помощью которого содержимое отказавшего блока можно восстановить по информации любого из блоков.

- Избыточность данных — высокая;
- Эффективность — высокая.

❒ *Зеркальные с чередованием* (mirrored striped) — эти тома соответствуют RAID 1+0.

- Избыточность данных — средняя;
- Эффективность — высокая.

По умолчанию Windows создает базовые диски (basic volumes). Однако любой базовый диск в любой момент времени может быть обновлен до динамического, и это не потребует даже перезагрузки системы.

> ***Примечание***
>
> Терминологические соответствия для обычных (basic) и динамических (dynamic) дисков приведены в табл. 6.4

Таблица 6.4. *Терминологические соответствия динамических и обычных дисков*

Базовые разделы (Basic disks)	Динамические разделы (Dynamic disks)
Основной раздел (Primary partition)	Простой том (Simple volume)
Системный и загрузочный разделы (System and boot partitions)	Системный и загрузочный тома (System and boot volumes)
Активный раздел (Active partition)	Активный том (Active volume)
Расширенный раздел (Extended partition)	Том и свободное пространство (Volume and unallocated space)
Логический диск (Logical drive)	Простой том (Simple volume)
Набор томов (Volume set)	Составной том (Spanned volume)
Чередующийся набор (Stripe set)	Чередующийся том (Stripe set)

Динамические диски могут использовать схемы разделов MBR или GPT, как и базовые разделы. Однако динамические диски, созданные путем обновления основных разделов, остаются в таблице разделов, при этом для них значение поля `System ID` меняется на `42h`. Если эта информация будет удалена, то система откажется подключать такой динамический диск. Между прочим, операционная система Windows может быть установлена только на такой динамический диск, который был получен путем обновления базового, т. к. BIOS может загружать систему лишь с тех разделов, которые перечислены в таблице разделов. При этом динамические диски, созданные "на лету", в таблицу разделов как раз и не попадают.

Схема разбиения динамических дисков содержится в базе данных менеджера логических дисков (Logical Disk Manager Database, LDM). Это протоколируемая (journaled) база данных, поддерживающая транзакции и устойчивая к сбоям. Если в процессе манипуляции с томами вдруг происходит сбой питания, то при после-

дующем включении компьютера будет выполнен откат (rollback) в предыдущее состояние. При переносе винчестера, содержащего один или несколько динамических дисков, на другой компьютер они автоматически распознаются и монтируются, как обыкновенные диски.

База данных LDM хранится в последнем мегабайте жесткого диска, а для дисков, полученных путем обновления базового раздела до динамического, — в последнем мегабайте этого раздела. В разделах GPT эта база данных содержится в зарезервированном (скрытом) разделе размером 1 Мбайт. Как уже говорилось, идентификатор загрузки `System ID` соответствующей записи таблицы разделов принимает значение `42h`. Так происходит потому, что при стандартном разбиении винчестера в его конце просто не остается свободного места, и операционной системе приходится сохранять эту информацию непосредственно на самом обновляемом диске (естественно, для этого на нем необходимо иметь по меньшей мере 1 Мбайт свободного пространства).

Сразу же за таблицей разделов по адресу `0/0/2` расположен приватный заголовок `PRIVHEAD`, содержащий в себе ссылки на основные структуры LDM (рис. 6.14). Если `PRIVHEAD` погибнет, Windows не сможет обнаружить и смонтировать динамические диски. К сожалению, гибнет он удручающе часто. Подавляющее большинство загрузочных вирусов и дисковых менеджеров считают сектор `0/0/2` свободным и используют его для хранения своего тела, необратимо затирая прежнее содержимое. Осознавая значимость заголовка `PRIVHEAD`, разработчики из Microsoft сохранили его в двух копиях, одна из которых хранится в конце LDM, а другая — в последнем секторе физического диска. Благодаря такой избыточности `PRIVHEAD` практически никогда не приходится восстанавливать вручную.

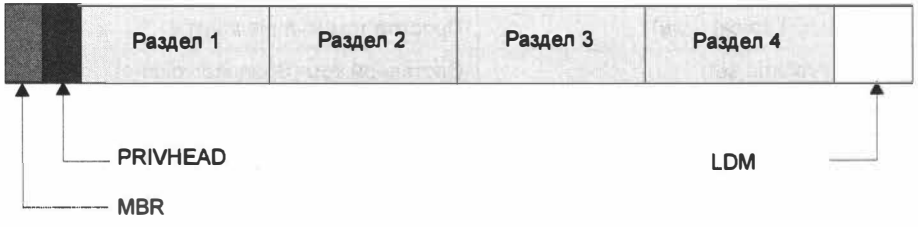


Рис. 6.14. База данных LDM и ее дислокация на дисках, использующих схему MBR

Внутреннее устройство базы данных LDM не документировано. При этом даже поверхностный взгляд на ее структуру сразу же дает понятие о ее мощи и сложности (рис. 6.15). На самом верху иерархии расположено оглавление базы — структура `TOCBLOCK` (Table Of Content Block), состоящая из двух секций `config` и `log` (причем, вероятно, в будущем их список будет расширен). Секция `config` содержит информацию о текущем разбиении диска на динамические тома, а `log` хранит журнал изменений схемы разбиения. Это очень мощное средство в борьбе с энтропией! Если удалить один или несколько динамических разделов, то информация о старом разбиении сохранится в журнале, что позволит с легкостью восстановить утерянные тома! Будучи очень важной структурой, оглавление диска защищено от случайного

разрушения тремя резервными копиями, одна из которых вплотную примыкает к оригинальной версии TOCBLOCK, расположенной в начале базы LDM, а две другие находятся в конце диска между копиями PRIVHEAD.

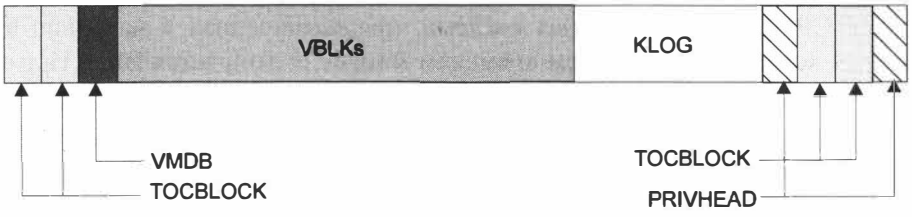


Рис. 6.15. Внутренняя структура LDM

Внутренне секция config состоит из заголовка (VMDB) и одного или нескольких 128-байтовых структур, называемых VBLKs, каждая из которых описывает соответствующий ей том, контейнер, раздел, диск или группу дисков. Заголовок VMDB не имеет копии и нигде не дублируется. Однако все его изменения протоколируются в журнале (KLOG) и потому могут быть восстановлены.

Для просмотра базы данных LDM и архивирования ее содержимого можно воспользоваться утилитой LDM-dump Марка Руссиновича, бесплатная копия которой доступна на сайте **https://docs.microsoft.com/en-us/sysinternals/downloads/ldmdump**. Как вариант, можно зарезервировать последний мегабайт физического диска, а также последние мегабайты всех разделов, для которых значение поля System ID равно 42h. Сделать это можно с помощью любого дискового редактора (например, Sector Inspector). Эту информацию рекомендуется хранить на надежном носителе. Помимо этого, не забудьте также создать и резервную копию структуры TOCBLOCK.

При восстановлении удаленных динамических дисков необходимо учитывать два фактора:

1. Журнал изменений на интерфейсном уровне недоступен, и выполнить откат штатными средствами операционной системы невозможно.
2. Загрузочные сектора удаляемых дисков автоматически очищаются, и восстанавливать их приходится вручную.
3. Если размер и тип удаленного динамического диска вам известны (на дисках NTFS их можно извлечь из копии загрузочного сектора), просто зайдите в Менеджер Управления Дисками (Disk Manager) и воссоздайте его заново, от предложения отформатировать раздел любезно откажитесь и восстановите очищенный загрузочный сектор по методике, описанной в следующем разделе.

Как видно, Microsoft хорошо позаботилась о своих пользователях и тщательно проработала структуру динамических дисков.

В стиле таблицы разделов GPT

Главная загрузочная запись неразрывно связана с базовой программой любого компьютера — с BIOS, которая с помощью MBR определяет дальнейшую последовательность загрузки машины. Идея BIOS, однако, восходит к 1975 году, когда разрабатывалась CP/M — операционная система, предназначенная изначально аж для 8-разрядных компьютеров! По тем временам многие ограничения BIOS (и формата MBR, в частности) таковыми, вероятно, не казались, но поддержка 16-разрядного реального режима (real mode) процессора, в котором работает BIOS, показалась Intel абсурдной в мире, где уже полным ходом используются 64-разрядные архитектуры. Так появился проект Intel Boot Initiative, опубликовавший в 1999 году стандарт EFI (Extensible Firmware Interface — расширяемый микропрограммный интерфейс). Он предназначался в качестве замены BIOS изначально для систем на базе процессоров Itanium (IA-64). В 2005 году этот проект превратился в Unified EFI Forum, который занимается дальнейшим развитием UEFI.

Все те многочисленные перемены, которые несет UEFI, нас, в общем-то, в текущем контексте не особо интересуют. Нам важно, что UEFI BIOS верно и не очень медленно вытесняет, да и уже практически вытеснил, традиционный BIOS: Linux мог (худо-бедно, но мог) загружаться с UEFI, как только появился подходящий загрузчик — elilo в начале 2000-х, в Apple на UEFI перешли в 2006 году, а в Windows поддержка UEFI начала появляться в Vista и стала полноценной в Windows 7. Ну и самое для нас главное: вместе с этим стандартом на смену MBR пришел новый формат разметки разделов — таблица разделов GUID (GUID Partition Table, GPT), — никоим образом не совместимый со своим предшественником. А это значит, что потребуются совершенно иные методы работы с GPT, в том числе и для ее восстановления.

Как это устроено?

Раз UEFI стремится заменить BIOS, старую как мир PC, то и изменения должны быть значительными, чтобы не возникло сомнений, за чем будущее. Так и есть — эти два механизма настолько разные, что новый стандарт даже не всегда обеспечивает совместимость со старыми ОС! Зато с приходом UEFI нам стали доступны невиданные прежде возможности:

- преодолено ограничение на размер диска в 2 ТиБ благодаря тому, что адреса логических блоков (LBA) стали 64-битовыми вместо 32-битовых в MBR (не путать с разрядностью адресов, используемых в протоколе ATA/ATAPI!);
- количество разделов не ограничено, и это без ухищрений в виде расширенного раздела и логических томов (на самом деле, конечно, в нашем мире нет ничего бесконечного, и разрядность переменной, содержащей количество разделов, позволяет указать лишь до 4 млрд разделов);
- для большей надежности применяются контрольные суммы CRC32 для служебных таблиц, а сами структуры GPT дублируются в конце диска;

- для идентификации раздела используется его GUID и может присутствовать человекочитаемое имя.

Итак, что же поменялось для пользователя? Теперь мы можем использовать носители объемом 8 ЗиБ (8×2^{70} байт) — а во столько оценивалось в конце 2015 года общее количество оцифрованной информации! При должной поддержке нового стандарта со стороны ПО больше не грозят проблемы с "заворотом" данных на накопителях объемом свыше 2 ТиБ, когда данные, записываемые по адресу, превышающему возможности LBA32, пишутся в начало диска, затирая имеющуюся там информацию. Стало намного проще создавать куда больше четырех разделов (таблицы разделов GPT, создаваемые по умолчанию, поддерживают до 128 записей). За счет чего же это все достигается?

В первом приближении различия между MBR и GPT не столь радикальны — и там и там есть служебная информация, описывающая сам носитель, и таблица разделов, описывающая собственно его разметку. На этом этапе разница заключается в том, что в UEFI используется служебный раздел EFI (EFI System Partition, ESP) размером, как правило, около 250 Мбайт, в котором находятся драйверы и приложения UEFI, не поместившиеся в прошивку на материнской плате. Запасная таблица GPT (Backup GPT Header) помещается в последний логический блок диска, а перед ней располагается запасная таблица разделов (Backup Partition Entry Array). Примерный вид типичного диска в GPT-разметке показан на рис. 6.16.

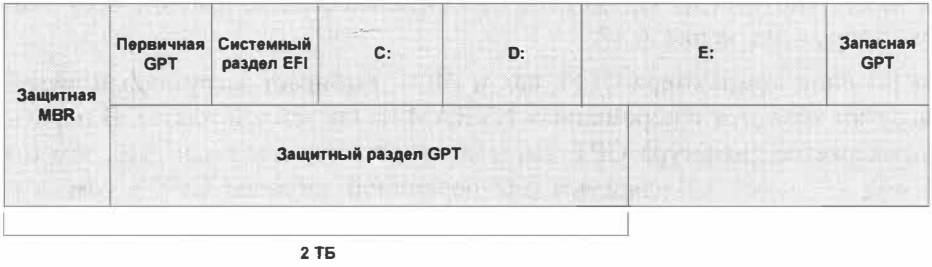


Рис. 6.16. Общий вид накопителя, отформатированного в GPT, с точки зрения GPT и MBR

Главная загрузочная запись целиком помещалась в один сектор размером 512 байт, а все ее элементы, включая таблицу разделов, определялись величиной смещения от его начала. Исключение составляют сами разделы (в том числе расширенные), которые адресуются или через CHS, или LBA. В случае же GPT в один сектор помещается лишь заголовок GPT, а адресация таблицы разделов и всего остального происходит только через LBA.

Первый логический блок на носителе — LBA0 — содержит так называемую защитную загрузочную запись (Protective MBR) на случай, если старые программы, слыхом не слыхавшие о GPT, сочтут диск нерабочим и решат его исправить. Также благодаря защитному MBR вместо действительной разметки старое ПО будет видеть один большой раздел типа `EEh`, именуемый защитным разделом GPT (GPT protective partition). Размер такого раздела, естественно, ограничен значением в 2 ТиБ даже на бóльших носителях, в силу ограничений MBR.

Следом, в LBA1, начинается первичная таблица GPT. В ней хранится заголовок GPT (GPT header) с общей информацией о носителе: сигнатурой и версией GPT, GUID диска, контрольными суммами служебных таблиц и прочими полезными вещами. За этим заголовком находится массив записей о разделах (Partition entry array), в котором описывается каждый имеющийся на носителе раздел: GUID типа раздела, GUID раздела, адреса логических блоков его начала и конца, атрибуты и имя. После таблицы разделов, начиная с блока `FirstUsableLBA`, могут располагаться непосредственно разделы.

Помимо всего этого, в первичной таблице хранится адрес логического блока запасной таблицы GPT. Она отличается от первой только значениями полей `MyLBA` и `AlternativeLBA`, которые здесь поменяны местами, и вследствие этого значениями CRC32. Заголовок запасной GPT находится в последнем блоке диска, перед ним расположена запасная таблица разделов, так что поле `PartitionEntryLBA` также имеет другое значение.

> ***Внимание!***
>
> Перенос образа накопителя на носитель большего объема потребует переопределения запасной GPT, ведь она должна находиться в последнем LBA накопителя.

Для хранения дополнительных программ UEFI, как уже было сказано, обычно присутствует небольшой системный раздел ESP. Как правило, он располагается первым и имеет тип FAT32. Примерное относительное расположение всех этих элементов изображено на рис. 6.17.

При включении компьютера UEFI, как и BIOS, выбирает загрузочный диск (порядок загрузки хранится в переменных NVRAM на системной плате). В первую очередь проверяется сигнатура GPT `45h 46h 49h 20h 50h 41h 52h 54h`, что в ASCII имеет вид `EFI PART`, находящаяся для первичной таблицы GPT в самом начале LBA1, а также то, указывает ли `MyLBA` на проверяемую таблицу. Проверяются CRC32 заголовка GPT и следующей за ним таблицы разделов. Все то же самое проделывается для запасной GPT. Если одна из таблиц распознана как поврежденная, то она восстанавливается из второй; если же так случилось, что обе таблицы содержат ошибки, то устройство определяется как не имеющее GPT-разметки. Тогда, в зависимости от реализации UEFI, либо появится сообщение об отсутствии загрузочного носителя, либо запустится UEFI Shell.

UEFI не зависит от первичного загрузчика MBR и не ищет его код в первом секторе. Вместо этого он ищет вспомогательные исполняемые файлы на разделе ESP, а затем передает управление файлу (в общем случае) `\efi\boot\boot<архитектура>.efi` на "загрузочном" разделе для загрузки ОС.

С помощью дискового или hex-редактора взглянем на первые сектора накопителя, размеченного в GPT. Их примерный вид показан на рис. 6.18. Здесь видна и запись о защитном GPT-разделе, расположившаяся в защитной MBR, и заголовок GPT. Формат его полей описан в табл. 6.5 (смещения указаны от начала таблицы, т. е. от начала LBA 1).

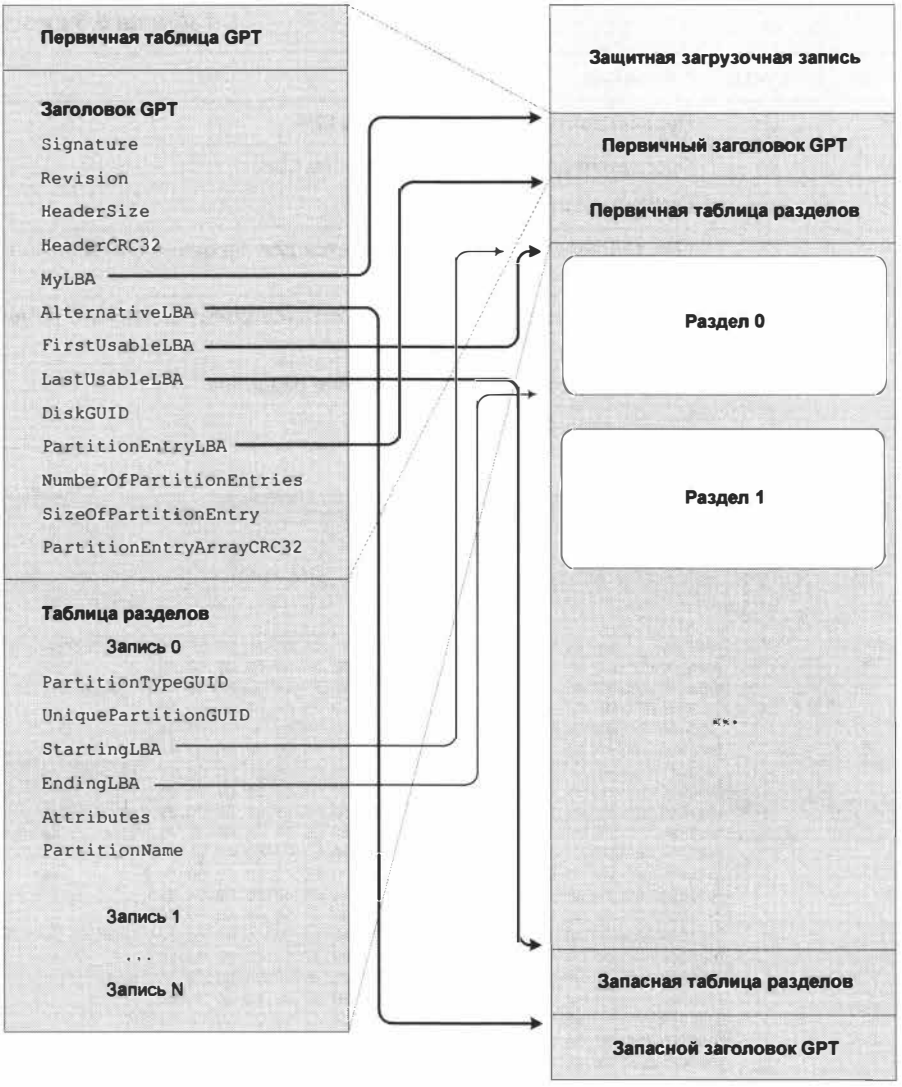


Рис. 6.17. Схематичное представление основных структур GPT

Таблица 6.5. *Формат заголовка GPT*

Смещение	Размер	Описание
0x00	8h	Сигнатура `EFI PART`, указывающая, что далее идет заголовок GPT
0x08	4h	Ревизия GPT
0x0C	4h	Размер заголовка GPT
0x10	4h	CRC32 заголовка GPT
0x18	8h	LBA текущего заголовка GPT
0x20	8h	LBA копии заголовка GPT

Таблица 6.5 (окончание)

Смещение	Размер	Описание
0x28	8h	Первый доступный для разделов LBA
0x30	8h	Последний доступный для разделов LBA
0x38	10h	GUID накопителя
0x48	8h	LBA таблицы разделов, различается для первичного и запасного заголовков GPT
0x50	4h	Количество записей о разделах, которое может содержать таблица разделов
0x54	4h	Размер записи о разделе в таблице разделов
0x58	4h	CRC32 таблицы разделов

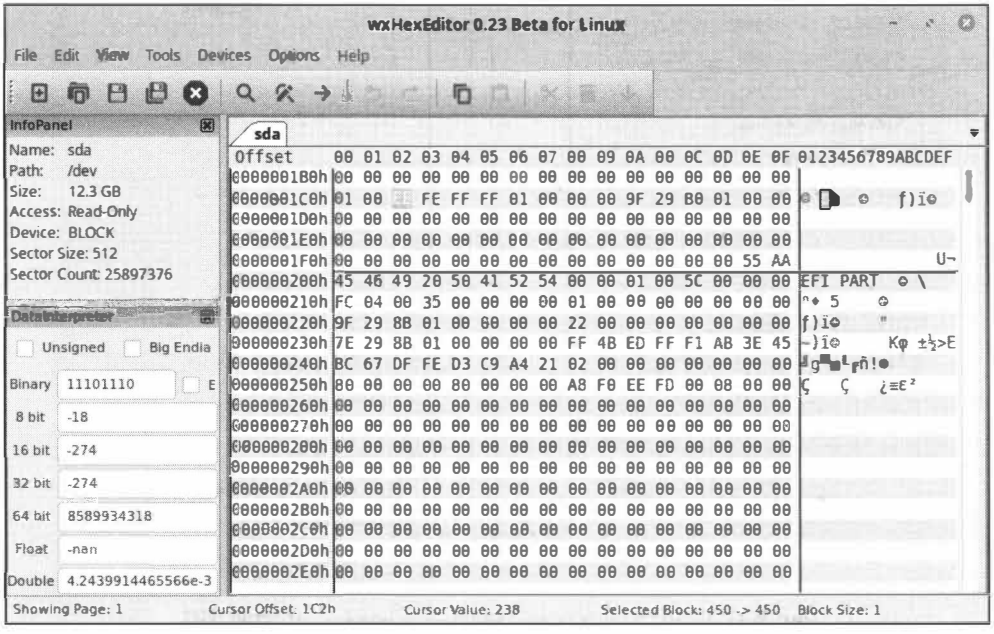


Рис. 6.18. Внешний вид GPT

Более наглядно просмотреть структуры GPT, включая защитную MBR, можно, воспользовавшись, к примеру, DMDE — она показывает в удобочитаемом виде заголовок GPT и таблицу разделов (рис. 6.19).

Реализации UEFI, как и традиционной BIOS, могут различаться от производителя к производителю. Одни поддерживают графический интерфейс с мышью и имеют целый ворох путаных настроек, другие внешне едва отличаются от древних псевдографических BIOS и крайне мало что позволяют изменять. Эта же вариативность верна и в плане поддержки загрузки старых ОС: без модуля поддержки обратной совместимости CSM (Compatibility Support Module) загрузить систему, не умеющую работать с UEFI и GPT, не получится; однако существует мнение, что в ско-

ром времени и этот модуль перестанет быть нужным, как и сама старая добрая BIOS, так что не на всяком оборудовании удастся запустить старые ОС (во всяком случае, без изысканных заклинаний). Впрочем, проблемы с совместимостью идут бок о бок с UEFI с самого его появления, и тут уж не привыкать.

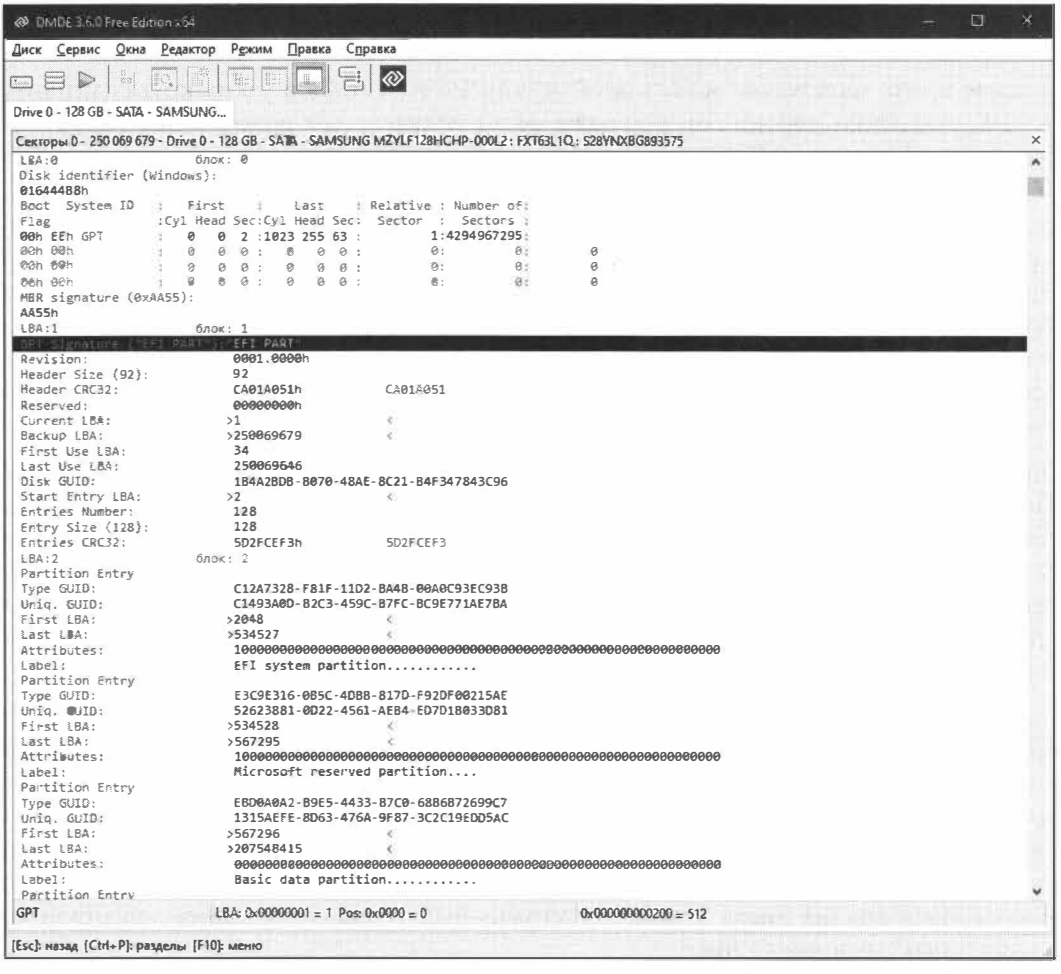


Рис. 6.19. Просмотр содержимого заголовка и таблицы разделов GPT в утилите DMDE

Восстановление разделов в формате GPT

Как бы то ни было, с приходом UEFI всем пришлось мириться, а значит, и учиться работать с новым форматом разделов. Операционные системы все лучше устанавливались на "железо" с новой прошивкой; программ, понимающих GPT, становилось все больше. Что осталось неизменным, так это возможность потерять свои данные.

В случае с GPT, надо отдать должное, получить поврежденную таблицу разделов стало сложнее — она, как и другая служебная информация GPT, дублируется

в конце диска и при наличии ошибок восстанавливается из копии, что значительно уменьшает вероятность случайно потерять разделы. Но реализации новых стандартов не лишены ошибок, иной раз настолько серьезных, что позволяют обойти защитные механизмы UEFI для установки вредоноса в такую область, что не спасет ни переустановка операционной системы, ни смена накопителя. А кто знает, на что способны такие вирусы... Так что же все-таки делать, если и с GPT приключилось несчастье?

Прежде всего, на помощь может прийти широко известная в узких кругах утилита DMDE, последние версии которой даже могут похвастаться полноценным графическим интерфейсом. DMDE объединяет в себе функции дискового редактора и менеджера разделов. Утилита умеет реконструировать структуру файлов и папок даже в случае сложных повреждений, но самое главное — она может работать с разделами GUID (GPT), причем варианты восстановления предлагаются в автоматическом режиме. Не менее важно, что эта утилита имеет версии не только под Windows, но также Linux и macOS. И хотя сами таблицы на диске DMDE не восстановит, но она позволит спасти данные с неисправных GPT-разделов даже на операционных системах, не имеющих никакого представления о GPT, как уже было показано в *главе 1*.

Другой инструмент, заслуживающий внимания, — это также упоминавшийся ранее TestDisk. Он может оказаться полезным для восстановления не только служебных таблиц, но и удаленных данных. Консольный интерфейс и функции экспертного режима (например, ручная настройка геометрии диска), несмотря на весьма информативные комментарии программы, требуют от пользователя более глубокого понимания происходящего, чем при использовании автоматизированных графических утилит, иначе велик шанс сделать тяжелую ситуацию еще хуже!

Скажем, у нас повреждена запасная таблица разделов, а первичная вообще куда-то подевалась — восстанавливать ее неоткуда, и остается лишь реконструировать. Для восстановления разделов в TestDisk необходимо выбрать устройство и тип разделов. Программа пытается определить его самостоятельно, но в нашем случае, когда не совпадают CRC обеих таблиц (первичной и запасной GPT), она считает, что это не формат GPT; впрочем, переубедить ее, выбрав нужный пункт, несложно. Далее выполняется анализ диска в поисках утраченных разделов и запись информации о найденных разделах на диск.

При переносе данных на больший носитель (клонировании или организации RAID-массивов с GPT) необходимо позаботиться о переносе запасных структур GPT в самый конец этого носителя, иначе возможна некорректная работа с ним, хотя Linux монтирует их разделы без особых возражений. GParted при обнаружении такого носителя сразу предлагает исправить эту проблему (рис. 6.20). Также могут помочь TestDisk, который исправит ошибки GPT при записи таблицы обнаруженных разделов, и gdisk (рис. 6.21) из пакета утилит GPT fdisk.

Много интересной информации об этом формате разметки разделов и UEFI можно найти, помимо официальной документации от UEFI Forum, на сайте Родерика Смита, автора набора утилит GPT fdisk для Linux, расположенного по адресу **https://rodsbooks.com/gdisk/index.html** (на английском). Там, например, описывается

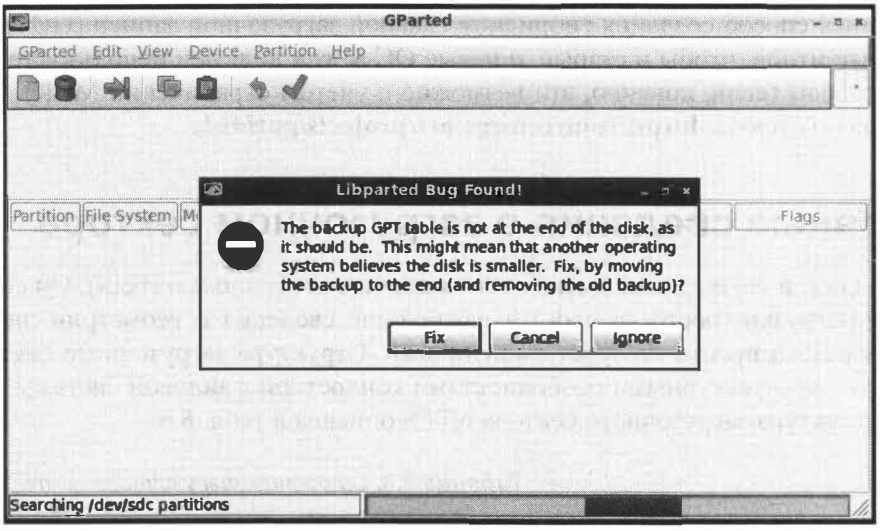


Рис. 6.20. GParted при обнаружении проблемных GPT-структур в ходе запуска

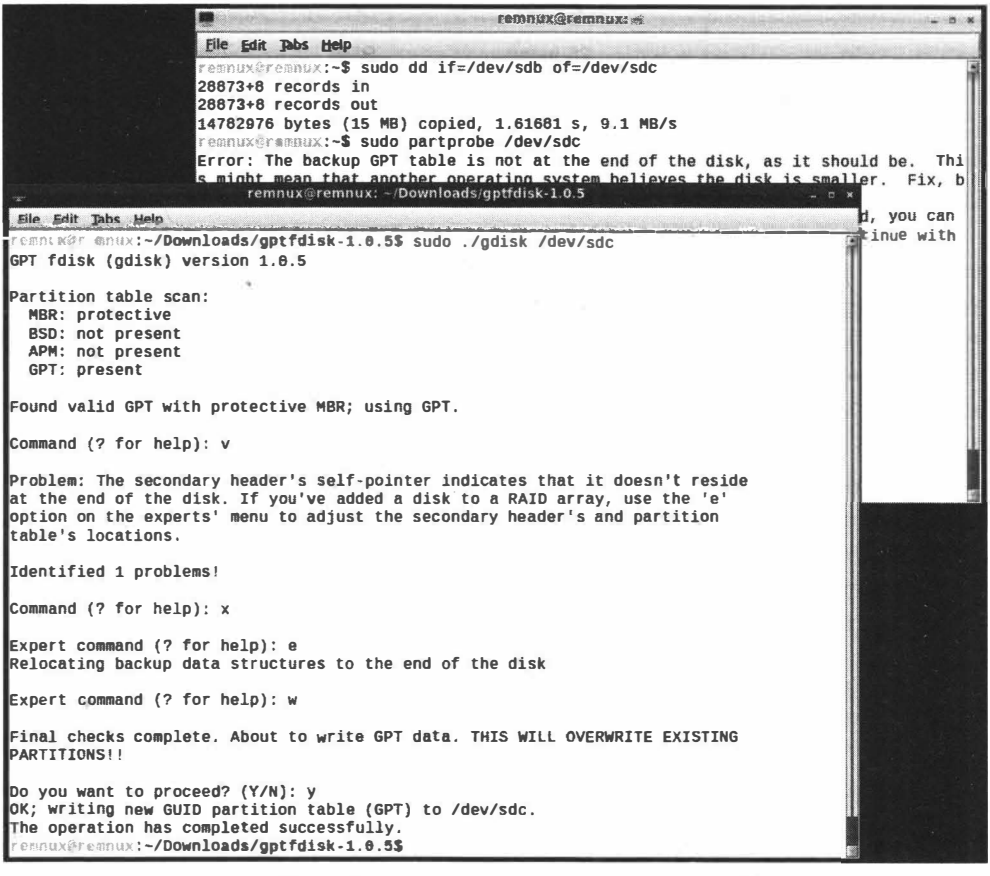


Рис. 6.21. gdisk за исправлением запасных структур GPT

интересный способ создания гибридной главной загрузочной записи (Hybrid MBR) вместо защитной, чтобы и старые, и новые ОС могли видеть и использовать одни и те же разделы (если, конечно, это возможно с учетом ограничений MBR). Скачать gdisk можно отсюда: **https://sourceforge.net/projects/gptfdisk/**.

Основные сведения о загрузочном секторе

Первый сектор логического диска носит название *загрузочного* (boot). Он содержит код самозагрузки (bootstrap code) и важнейшие сведения о геометрии диска, без которых раздел просто не будет смонтирован. Структура загрузочного сектора определяется архитектурными особенностями конкретной файловой системы. В частности, структура загрузочного сектора NTFS описана в табл. 6.6.

Таблица 6.6. *Строение загрузочного сектора NTFS*

Смещение	Размер	Описание
0x00	3 bytes	Инструкция перехода
0x03	8 байт	OEM ID — идентификатор
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap Code
0x01FE	WORD	55 AA

В начале всякого сектора расположена трехбайтовая машинная команда перехода на код самозагрузки (обычно она имеет вид `EB 52 90`, хотя возможны и различные вариации). Так происходит потому, что при загрузке boot-сектора в память управление передается на его первый байт, а код самозагрузки по туманным историческим соображениям был отодвинут в конец сектора (для NTFS верхняя граница составляет `54h` байтов), вот и приходится прыгать блохой!

С третьего по одиннадцатый байты (считая от нуля) хранится идентификатор производителя, определяющий тип и версию используемой файловой системы (например, `MSWIN4.0`/`MSWIN4.1` для FAT32 и `NTFS` для NTFS). Если это поле окажется искаженным, то драйвер не сможет смонтировать диск. В ряде случаев драйвер может даже счесть такой диск неотформатированным.

> ***Примечание***
>
> С дисками, отформатированными для использования FAT, Windows будет работать даже в том случае, если поле `OEM ID` запорчено. В отношении NTFS это не так.

Следом за идентификатором расположен 25-байтовый блок параметров BIOS (BIOS Parameter Block, BPB), хранящий сведения о геометрии диска (количество цилиндров, головок, секторов, размер сектора, количество секторов в кластере и т. д.). Если эта информация окажется утерянной или искаженной, то нормальное

функционирование драйвера файловой системы не гарантировано. Причем, в отличие от информации о числе цилиндров/головок/секторов, которая дублирует информацию, содержащуюся в MBR, а при ее утере элементарно восстанавливается описанным выше способом, размер кластера определить не так-то просто! Позже мы обсудим этот вопрос более подробно, пока же вполне достаточно сослаться на табл. 6.7, в которой указаны размеры кластера томов NTFS, по умолчанию выбираемые штатной утилитой форматирования.

Таблица 6.7. *Размеры кластеров, по умолчанию выбираемые штатной утилитой форматирования в Windows*

Размер тома	Размер кластера
7–512 МБ	4 КБ
512 МБ–1 ГБ	4 КБ
1–2 ГБ	4 КБ
2 ГБ–2 ТБ	4 КБ
2–16 ТБ	4 КБ
16–32 ТБ	8 ГБ
32–64 ТБ	16 ГБ
64–128 ТБ	32 ГБ
128–256 ТБ	64 ГБ

Чем больше размер кластера, тем слабее фрагментация и выше предельно адресуемый объем дискового пространства. Однако потери от грануляции с увеличением размера кластера тоже растут. Впрочем, размеры кластеров редко задаются вручную.

К блоку параметров BIOS вплотную примыкает его продолжение — расширенный блок параметров BIOS (extended BPB), хранящий номер первого кластера MFT, ее размер в кластерах, номер кластера с зеркалом MFT, а также некоторую другую служебную информацию. В отличие от FAT16/FAT32, MFT может располагаться в любом месте диска (для борьбы с bad-секторами это актуально). При нормальном развитии событий MFT располагается практически в самом начале диска (где-то в районе четвертого кластера), и если только она не была перемещена, то ее легко найти глобальным поиском (искать следует строку `FILE*` по смещению `0` от начала сектора). При разрушении или некорректном заполнении расширенного блока параметров BIOS драйвер файловой системы отказывается монтировать раздел, объявляя его неотформатированным.

Следом за расширенным блоком параметров BIOS идет код самозагрузки (Bootstrap Code), который ищет на диске загрузчик операционной системы, загружает его в память и передает ему управление. Если код самозагрузки отсутствует, загрузка операционной системы становится невозможной, однако при подключении восстанавливаемого диска вторым раздел должен быть прекрасно виден. Повреждение кода самозагрузки вызывает перезагрузку компьютера или его "зависание".

Наконец, завершает загрузочный сектор уже известная нам сигнатура 55h AAh, без которой он ни за что не будет признан загрузочным. Подробная информация обо всех полях загрузочного сектора NTFS приведена в табл. 6.8.

Таблица 6.8. *Значения полей загрузочного сектора NTFS*

Смещение	Размер	Описание
0x00	3 байта	Инструкция перехода
0x03	8 байт	OEM ID
0x0B	WORD	Количество байтов на сектор (для жестких дисков всегда 512)
0x0D	BYTE	Количество секторов на кластер
0x0E	WORD	Количество зарезервированных секторов, всегда равно 0
0x10	3 байта	Не используется NTFS и всегда должно быть равно 0
0x13	WORD	Не используется NTFS и всегда должно быть равно 0
0x15	BYTE	Дескриптор носителя (media descriptor) — для жестких дисков всегда равен 0xF8
0x16	WORD	Не используется NTFS и всегда должно быть равно 0
0x18	WORD	Количество секторов на дорожку
0x1A	WORD	Количество головок
0x1C	DWORD	Количество скрытых секторов
0x20	DWORD	Не используется NTFS и всегда должно быть равно 0
0x24	DWORD	Не используется NTFS и всегда должно быть равно 0
0x28	8 байт	Общее количество секторов (total sector)
0x30	8 байт	Логический номер кластера, с которого начинается MFT
0x38	8 байт	Логический номер кластера, с которого начинается зеркало MTF
0x40	DWORD	Количество кластеров на сегмент (File Record Segment)
0x44	DWORD	Количество кластеров на блок индексов (index block)
0x48	8 байт	Серийный номер тома
0x50	DWORD	Контрольная сумма (0 — не подсчитывать).
0x54	426 байт	Код самозагрузки (Bootstrap Code)
0x01FE	WORD	Сигнатура 55 AA

Техника восстановления загрузочного сектора

Осознавая значимость загрузочного сектора, операционная система Windows NT при форматировании диска создает его зеркальную копию (однако делает она это только на разделах NTFS). Для различных версий Windows расположение резервной копии загрузочного сектора различно. Так, ОС Windows, начиная с версии 2000,

хранит резервную копию загрузочной записи раздела в последнем секторе раздела. Если таблица разделов уцелела, то для восстановления загрузочного сектора достаточно просто перейти в начало следующего раздела и отступить на сектор назад.

Если таблица разделов разрушена, то найти резервную копию загрузочного сектора можно глобальным поиском (ищите строку `NTFS` по смещению `3` от начала сектора). Поскольку положение копии фиксировано и отсчитывается от начала логического диска, мы можем с абсолютной уверенностью определить границы раздела. Предположим, что копия загрузочного сектора найдена в секторе `1289724`, а поле `NumberSectors` содержит значение `12289661`. Тогда номер конечного сектора раздела равен `1289724`, а номер стартового сектора можно вычислить следующим образом: `1289724 – 12289661 = 63`. Поскольку загрузочный сектор расположен на расстоянии одной головки от таблицы разделов, что соответствует значению `SectorPerTrack`, мы сможем восстановить и ее.

Если резервных копий загрузочного сектора нет, его придется реконструировать заново. К счастью, это совсем не так сложно, как может показаться на первый взгляд. В поле идентификатора производителя заносится строка `NTFS` (обратите внимание, что на конце этой строки должны быть четыре пробела). Поля, задающие количество секторов на дорожке и количество головок, заполняются исходя из текущей геометрии диска. Количество скрытых секторов (т. е. количество секторов, расположенных между началом раздела и загрузочным разделом) равно количеству головок. Общее количество секторов в разделе вычисляется на основании его размера (если точный размер не известен, берите значение с запасом).

Количество секторов в кластере определить сложнее. Это особенно справедливо, если при форматировании диска было задано количество секторов на кластер, отличное от значения по умолчанию. Но ситуация вовсе не безнадежна. Последовательно сканируя файловые записи в MFT, найдите файл с предопределенной и заранее известной сигнатурой. Пусть, для определенности, это будет файл NTOSKRNL.EXE. Откройте его аутентичную копию в hex-редакторе, найдите уникальную последовательность, гарантированно не встречающуюся ни в каких других файлах и расположенную в пределах первых 512 байтов от его начала, после чего найдите эту сигнатуру глобальным поиском по всему диску. Начальный номер кластера вам известен (он содержится в MFT), логический номер сектора известен тоже (его нашел дисковый редактор). Теперь остается лишь соотнести эти две величины между собой. Естественно, если дисковый редактор найдет удаленную копию NTOSKRNL.EXE (или на диске будут присутствовать несколько файлов NTOSKRNL.EXE), данный метод даст осечку, поэтому полученный результат необходимо уточнить, проведя аналогичные исследования с использованием других файлов.

Логический номер первого кластера MFT равен первому кластеру, в начале которого встретилась строка `FILE*` (конечно, при том условии, что MFT не была перемещена). По умолчанию Windows выделяет под MFT 12,5% от емкости раздела, помещая ее зеркальную копию в середину. Кроме того, ссылка на "зеркало" присутствует и в самой MFT. Если же MFT разрушена, переместитесь в середину диска,

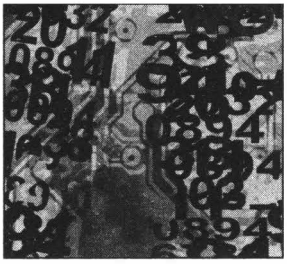
немного отступите назад и повторите глобальный поиск строки `FILE*` (только, смотрите, не вылетите в соседний раздел!). Первое же найденное вхождение с высокой степенью вероятности и будет зеркальной копией MFT.

Количество кластеров на сегмент обычно равно `F6h`, а количество кластеров на блок индексов — `01h`. Других значений нам встречать не доводилось. Серийный номер тома может быть любым — он ни на что не влияет.

Для восстановления отсутствующего (искаженного) кода самозагрузки можно воспользоваться стандартными средствами восстановления загрузчика Windows, описанными ранее в этой главе.

Как видите, в восстановлении загрузочного сектора нет ничего мифического, и для устранения большинства типов разрушений супервысокой квалификации не требуется. Если же какие-то из утверждений, приведенных в этой главе, вам не понятны, перечитайте ее, сидя за компьютером с запущенным дисковым редактором. До сих пор мы говорили о достаточно простых и хорошо известных вещах. Теперь, как следует раскачавшись и освоившись с основными понятиями, мы можем отправляться в самые "дебри" NTFS, восстановлению структур которой посвящена следующая глава.

ГЛАВА 7



Файловая система NTFS — взгляд изнутри

В этой главе будут рассмотрены основные структуры файловой системы NTFS и ее основополагающие концепции — главная файловая таблица (MFT), файловые записи, последовательности обновления, атрибуты или потоки (streams), а также отрезки (data runs).

Без полноценного понимания этих концепций невозможно более или менее осмысленно работать с дисковыми редакторами или заниматься восстановлением данных.

Введение

Файловую систему NTFS принято описывать как сложную реляционную базу данных, обескураживающую грандиозностью своего архитектурного замысла не одно поколение начинающих исследователей. NTFS похожа на огромный, окутанный мраком лабиринт, в котором очень легко заблудиться. К счастью, хакеры давно разобрались с основными структурами данных. Тем не менее от магистральных коридоров лабиринта, ярко освещенных светом настенных факелов (и подсознательно ассоциируемых с хорошо исследованными структурами данных), отходит большое количество ответвлений, которые освещены значительно хуже (если освещены вообще). Они хранят большое количество опасных ловушек, соответствующих особым случаям обработки структур данных, которые на первый взгляд кажутся знакомыми.

Основная задача, которая стоит перед нами в процессе восстановления данных — вернуть разрушенный том в состояние, пригодное для восприятия операционной системой или, по крайней мере, извлечь из такого тома все ценные файлы. Для этого совершенно не обязательно вникать в структуру журналов транзакций, дескрипторов безопасности и двоичных деревьев. Иначе говоря, нам потребуется разобраться лишь с устройством главной файловой таблицы — MFT, а также нескольких дочерних подструктур.

Версии NTFS

Служебные структуры файловой системы NTFS не остаются постоянными, а слегка меняются от одной версии Windows NT к другой (см. табл. 7.1). К счастью, с момента появления на свет ОС Windows XP структура NTFS практически не менялась, что значительно упрощает работу с программными средствами восстановления данных, даже выпущенными много лет назад.

Таблица 7.1. *Соответствие версий NTFS и Windows*

Версия NTFS	Операционная система	Условное обозначение
1.2	Windows NT	NT
3.0	Windows 2000	W2K
3.1	Windows XP — Windows 10	XP

В подавляющем большинстве случаев (особенно если ваш компьютер или ноутбук сошел с конвейера после 2001 года) используемой версией NTFS будет 3.1.

> ***Совет***
>
> Как быстро узнать тип текущего раздела — FAT или NTFS? Это очень просто — достаточно попробовать создать в его корневом каталоге файл `$mft` — если он будет создан успешно, то это FAT. Если создать файл с таким именем в корневом каталоге диска невозможно, значит, этот диск отформатирован для использования NTFS. Чтобы быстро определить версию NTFS, попробуйте создать в корневом каталоге диска файл `$Extend`. Если такой файл будет создан успешно, то версия файловой системы 3.0 или выше.

Взгляд на NTFS с высоты птичьего полета

Основным структурным элементом всякой файловой системы является *том* (volume), в случае с FAT совпадающий с *разделом* (partition), о котором мы говорили в прошлой главе. NTFS поддерживает тома, состоящие из нескольких разделов (рис. 7.1). Подробнее схему отображения томов на разделы мы обсудим в следующей главе, а пока будем для простоты считать, что том представляет собой отформатированный раздел (т. е. раздел, содержащий служебные структуры файловой системы).

Большинство файловых систем трактуют *том* как совокупность файлов, свободного дискового пространства и служебных структур файловой системы, но в NTFS *все* служебные структуры представлены файлами, которые (как это и положено файлам) могут находиться в *любом* месте тома, при необходимости фрагментируя себя на несколько частей.

Основным служебным файлом является главная файловая таблица *$MFT* (Master File Table) — своеобразная база данных, хранящая информацию обо всех файлах тома — их именах, атрибутах, способе и порядке размещения на диске. Каталог также является файлом особого типа, со списком принадлежащих ему файлов и

вложенных подкаталогов. Важно подчеркнуть, что в MFT присутствуют *все* файлы, находящиеся во *всех* подкаталогах тома, поэтому для восстановления диска наличия файла `$MFT` будет вполне достаточно.

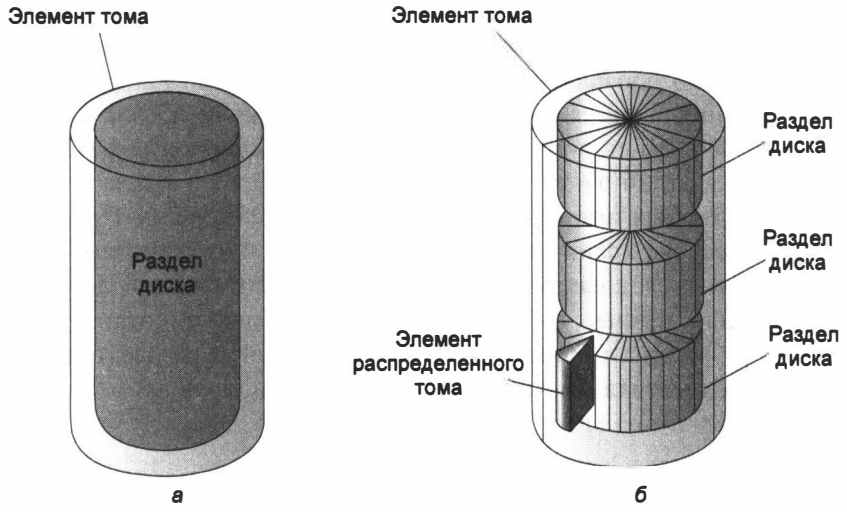


Рис. 7.1. Обычный (а) и распределенный (б) тома

Остальные служебные файлы, называемые *метафайлами* (metafiles) или *метаданными* (metadata), всегда имеют имена, начинающиеся со знака доллара (`$`), и носят сугубо вспомогательный характер, интересный только самой файловой системе. К ним в первую очередь относится: `$LogFile` — файл транзакций, `$Bitmap` — карта свободного/занятого пространства, `$BadClust` — перечень "плохих" кластеров и т. д. Более подробная информация о них будет приведена далее в этой главе. Текущие версии Windows блокируют доступ к служебным файлам с прикладного уровня (даже с правами администратора!), и всякая попытка открытия или создания такого файла в корневом каталоге обречена на неудачу.

Классическое определение, данное в учебниках информатики, отождествляет файл с именованной записью на диске. Большинство файловых систем добавляет к этому понятие *атрибута* (attribute) — некоторой вспомогательной характеристики, описывающей время создания, права доступа и т. д. В NTFS имя файла, данные файла и его атрибуты полностью уравнены в правах. Иначе говоря, всякий файл NTFS представляет собой совокупность атрибутов, каждый из которых хранится как отдельный *поток* байтов. Поэтому во избежание путаницы атрибуты, хранящие данные файла, часто называют *потоками* (streams).

Каждый атрибут состоит из *тела* (body) и *заголовка* (header). Атрибуты подразделяются на *резидентные* (resident) и *нерезидентные* (non-resident). Резидентные атрибуты хранятся непосредственно в `$MFT`, что существенно уменьшает грануляцию дискового пространства и сокращает время доступа. Нерезидентные атрибуты хранят в `$MFT` лишь свой заголовок, описывающий порядок размещения атрибута на диске.

Назначение атрибута определяется его *типом* (type), представляющим собой четырехбайтовое шестнадцатеричное значение. При желании атрибуту можно дать еще и *имя* (name), состоящее из символов, входящих в соответствующее пространство имен (namespace). Подавляющее большинство файлов имеет по меньшей мере три атрибута, к числу которых относится стандартная информация о файле (время создания, модификации, последнего доступа, права доступа), которая хранится в атрибуте типа `10h`, условно обозначаемом `$STANDARD_INFORMATION`. Ранние версии Windows NT позволяли обращаться к атрибутам по их условным обозначениям, но, начиная с Windows 2000, мы лишены этой возможности. Полное имя файла (не путать с путем!) хранится в атрибуте типа `30h` (`$FILE_NAME`). Если у файла есть одно или несколько альтернативных имен (например, имя MS-DOS), таких атрибутов может быть несколько. Здесь же хранится *ссылка* (file reference) на родительский каталог, позволяющая разобраться, к какому каталогу принадлежит данный файл или подкаталог. По умолчанию данные файла хранятся в безымянном атрибуте типа `80h` (`$DATA`). Однако при желании прикладные программы могут создавать дополнительные потоки данных, отделяя имя атрибута от имени файла знаком двоеточия (например: `ECHO xxx > file:attr1; ECHO yyy > file:attr2; more < file:attr1; more < file:attr2`).

Изначально в NTFS была заложена способность индексации любых атрибутов, значительно сокращающая время поиска файла по заданному списку критериев (например, по времени последнего доступа). Индексы хранятся в виде двоичных деревьев, поэтому среднее время выполнения запроса оценивается как $O(\lg n)$[1]. На практике в большинстве драйверов NTFS реализована индексация лишь по имени файла. Как уже говорилось ранее, каталог представляет собой файл особого типа — файл *индексов*. В отличие от FAT, где файл каталога является единственным источником данных об организации файлов, в NTFS файл каталога служит лишь для ускорения доступа к содержимому каталога. Он не обязательный, т. к. ссылка на родительский каталог всякого файла всегда присутствует в атрибуте его имени (`$FILE_NAME`).

Каждый атрибут может быть зашифрован, разрежен или сжат. Техника работы с такими атрибутами выходит далеко за рамки первичного знакомства с организацией файловой системы и будет описана позднее. Пока же рассмотрим углубленно фундамент файловой системы NTFS — структуру `$MFT`.

Главная файловая таблица

В процессе форматирования логического раздела в его начале создается так называемая *зона MFT* (рис. 7.2). По умолчанию она занимает 12,5% от емкости тома (а не 12%, как утверждается во многих публикациях), хотя, в зависимости от значения параметра `NtfsMftZoneReservation`, она может составлять 25, 37 или 50%.

---

[1] Подробнее об этой формуле см. **https://ru.wikipedia.org/wiki/%C2%ABO%C2%BB_%D0%B1%D0%BE%D0%BB%D1%8C%D1%88%D0%BE%D0%B5_%D0%B8_%C2%ABo%C2%BB_%D0%BC%D0%B0%D0%BB%D0%BE%D0%B5**.

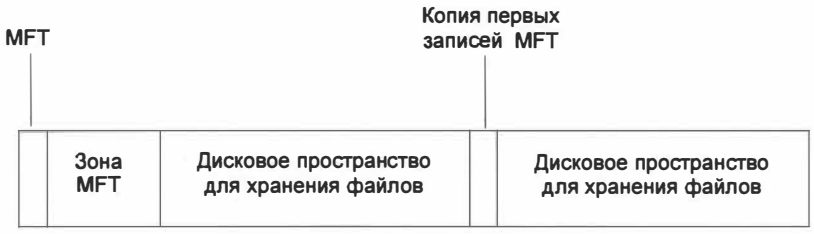


Рис. 7.2. Структура тома, отформатированного под NTFS

В этой области расположен файл `$MFT`, изначально занимающий порядка 64 секторов и растущий от начала зоны MFT к ее концу по мере создания новых пользовательских файлов и каталогов. Чем больше файлов содержится на томе, тем больше размер MFT. Приблизительный размер файла MFT можно оценить по следующей формуле: `sizeof(FILE Record)×N_Files`, где `sizeof(FILE Record)` обычно составляет 1 Кбайт, а `N_Files` — полное количество файлов и подкаталогов раздела, включая недавно удаленные.

Для предотвращения фрагментации файла `$MFT` зона MFT удерживается зарезервированной вплоть до полного исчерпания свободного пространства тома, затем незадействованный "хвост" зоны MFT усекается в два раза, освобождая место для пользовательских файлов. Этот процесс может повторяться многократно, вплоть до полной отдачи всего зарезервированного пространства. Решение красивое, хотя и не новое. Многие из файловых систем 1980-х годов позволяли резервировать заданное дисковое пространство в "хвосте" активных файлов, сокращая их фрагментацию (причем любых файлов, а не только служебных). Например, такая способность была у DOS 3.0, разработанной для персональных компьютеров типа "Агат". Может быть, кто-то из вас помнит такую машину?

Когда файл `$MFT` достигает границ зоны MFT, в ходе своего последующего роста он неизбежно фрагментируется, вызывая обвальное падение производительности файловой системы. При этом стоит заметить, что подавляющее большинство дефрагментаторов файл `$MFT` не обрабатывают! А ведь API дефрагментации, встроенный в штатный драйвер NTFS, обеспечивает такую возможность!

При необходимости файл `$MFT` может быть перемещен в любую часть диска, и тогда в начале тома его уже не окажется. Стартовый адрес файла `$MFT` хранится в загрузочном секторе по смещению `30h` байтов от его начала *(см. описание структуры загрузочного сектора в главе 6)*. В подавляющем большинстве случаев этот адрес ссылается на четвертый кластер.

Файл `$MFT` представляет собой массив записей типа `FILE Record` (в терминологии UNIX они называются *inodes*), каждая из которых описывает соответствующий ей файл или подкаталог. На практике один файл или подкаталог полностью описывается единственной записью типа `FILE Record`, хотя в теории этих записей может потребоваться и несколько.

Для ссылки на одну файловую запись из другой используется ее порядковый номер (он же индекс) в файле `$MFT`, отсчитываемый от нуля. *Файловая ссылка* (file

reference) состоит из двух частей (см. табл. 7.2) — 48-битового *индекса* и 16-битового *номера последовательности* (sequence number).

Таблица 7.2. *Структура файловой ссылки*

Смещение	Размер (байт)	Описание
00h	6	Индекс файловой записи (FILE record number), отсчитываемый от нуля
06h	2	Номер последовательности (sequence number)

При удалении файла или каталога соответствующая ему файловая последовательность помечается как неиспользуемая. При создании новых файлов записи, помеченные как неиспользуемые, могут задействоваться вновь, при этом счетчик номера последовательности, хранящийся внутри файловой записи, увеличивается на единицу. Этот механизм позволяет отслеживать "мертвые" ссылки на уже удаленные файлы. Номер последовательности внутри файловой ссылки в этом случае будет отличаться от номера последовательности соответствующей файловой записи.

Первые 12 записей в MFT всегда занимают служебные метафайлы: `$MFT` (собственно, сам файл `$MFT`), `$MFTMirr` (зеркало `$MFT`), `$LogFile` (файл транзакций), `$Volume` (сведения о дисковом томе), `$AttrDef` (определения атрибутов), `'.'` (корневой каталог), `$Bitmap` (карта свободного пространства), `$Boot` (системный загрузчик), `$BadClus` (перечень "плохих" кластеров) и т. д. Более подробно эти записи описаны в табл. 7.11.

Первые четыре записи настолько важны, что продублированы в специальном файле `$MFTMirr`, находящемся примерно в середине тома (точный адрес этого файла хранится в загрузочном секторе по смещению `38h` байтов от его начала). Вопреки своему названию, файл `$MFTMirr` — это отнюдь не "зеркало" всего файла `$MFT`, а всего лишь резервная копия первых четырех его элементов.

Записи с 12-й по 15-ю помечены как используемые, в то время как в действительности они пусты. Как несложно догадаться, они зарезервированы для использования в будущем. Записи с 16-й по 23-ю не задействованы и честно помечены как неиспользуемые.

Начиная с записи 24, располагаются пользовательские файлы и каталоги. Четыре метафайла, появившихся в Windows 2000, — `$ObjId`, `$Quota`, `$Reparse` и `$UsnJrnl` — могут располагаться в любой записи, номер которой равен 24 или больше (не забудьте, что нумерация файловых записей начинается с нуля).

Вот и вся теоретическая информация, необходимая на первых порах. Теперь можно приступать к практическому знакомству с NTFS. Для начала запустим утилиту DiskExplorer от Runtime Software, не забывая о том, что она требует прав администратора. В меню **File** найдем пункт **Drive** и в появившемся диалоговом окне выберем логический диск, который требует редактирования. Затем из меню **Goto** выберем пункт **Mft**, заставляя DiskExplorer перейти к MFT, автоматически меняя режим отображения на наиболее естественный (рис. 7.3). Как вариант, можно нажать кла-

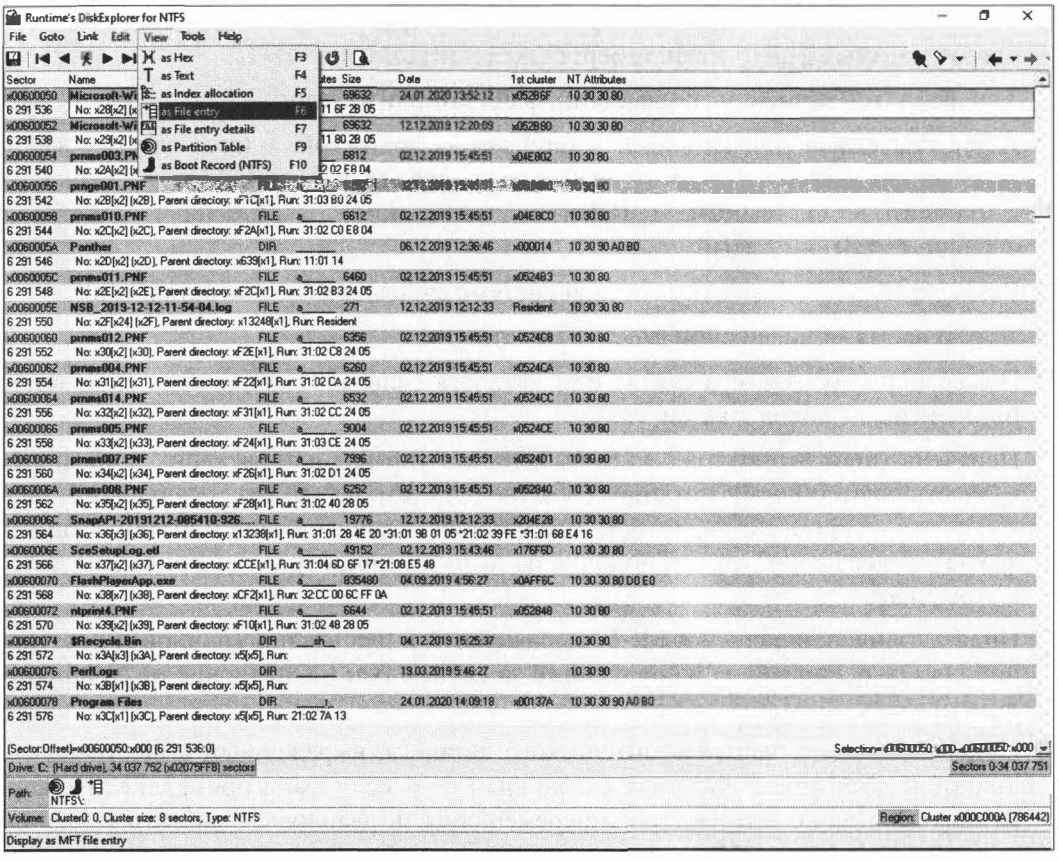


Рис. 7.3. Утилита DiskExplorer отображает главную файловую запись в естественном формате

вишу <F6> (**View as File Entry**) и пропустить несколько первых секторов нажатием клавиши <Page Down>.

Для каждого из файлов DiskExplorer сообщает следующее:

- Номер сектора, к которому принадлежит данная файловая запись. Обратите внимание, что номера секторов монотонно увеличиваются на 2, подтверждая тот факт, что размер одной файловой записи равен 1 Кбайт, хотя на практике можно столкнуться и с другими значениями. Для удобства информация отображается сразу в двух системах счисления — шестнадцатеричной и десятичной.
- Основное имя файла/каталога (т. е. имя файла из заголовка файловой записи). Стоит напомнить, что некоторые файлы имеют несколько альтернативных имен, более подробная информация о которых будет приведена далее в данной главе. Если имя файла или каталога зачеркнуто, это означает, что он был удален, но соответствующая ему файловая запись все еще цела. Чтобы извлечь файл с диска (не важно, удаленный или нет), подведите к нему курсор и нажмите клавиатурную комбинацию <Ctrl>+<T> для просмотра его содержимого в шестнадцатеричном виде или <Ctrl>+<S> для сохранения файла на диск. То же самое можно сделать и через контекстное меню, выбрав подпункт **Recovery**. При нажатии клавиатурной

комбинации <Ctrl>+<C> в буфер обмена копируется последовательность кластеров, занятых файлом, например: `DISKEXPL:K:1034240-1034240`.

- Тип файловой записи, указывающий, файл это или каталог.
- Атрибуты файла или каталога: `a` (archive) — архивный, `r` (read-only) — защищенный от записи, т. е. доступный только для чтения, `h` (hidden) — скрытый, `s` (system) — системный, `l` (label) — метка тома, `d` (directory) — каталог, `c` (compressed) — сжатый.
- Размер файла в байтах в десятичной системе счисления (не для каталогов!).
- Дату и время модификации файла или каталога.
- Номер первого кластера файла или каталога (или `resident` — для полностью резидентных файлов и каталогов).
- Перечень типов атрибутов NTFS, имеющихся у файла или каталога, записанных в шестнадцатеричной нотации (обычно эта строка имеет следующий вид: `10 30 80` — атрибут стандартной информации, атрибут имени и атрибут данных файла). Более подробная информация по данному вопросу будет приведена далее в этой главе.
- Индекс файловой записи в MFT, выраженный в шестнадцатеричной и десятичной системах счисления и следующий за словом **No:** (сокращение от Number — номер).
- Индекс файловой записи родительского каталога, выраженный в шестнадцатеричной и десятичной системах счисления (`5h` — если файл принадлежит к корневому каталогу). Для быстрого перемещения по файловым записям выберите в меню **Goto** пункт **Mft no** и введите требуемый индекс в шестнадцатеричной или десятичной нотации.
- Для нерезидентных файлов или каталогов — перечень кластеров, занятых файлом в закодированном виде (а зря — могли бы и декодировать). Схема кодирования кластеров подробно описана далее в данной главе.

Прежде чем продолжать чтение, попробуйте поэкспериментировать с файлами MFT (особенно фрагментированными). Посмотрите, как создаются и удаляются записи MFT. Лучше всего это делать на диске, содержащем небольшое количество файлов и каталогов. Чтобы не форматировать логический диск, создайте виртуальный.

Файловые записи

Благодаря наличию утилиты DiskExplorer от Runtime Software с файловыми записями практически никогда не приходится работать вручную. Тем не менее знание их структуры нам не помешает.

Структурно файловая запись состоит из *заголовка* (header) и одного или нескольких *атрибутов* (attributes) произвольной длины, завершаемых *маркером конца* (end marker) — четырехбайтовым шестнадцатеричным значением `FFFFFFFFh` (см. листинг 7.1). Несмотря на то что количество атрибутов и их длина меняются от одной

файловой записи к другой, размер самой структуры FILE Record строго фиксирован. В большинстве случаев он равен 1 Кбайт (это значение хранится в файле $boot, причем первый байт файловой записи всегда совпадает с началом сектора).

> ***Внимание!***
> Не следует путать файл $boot с загрузочным сектором (boot sector).

Если реальная длина атрибутов меньше размеров файловой записи, то ее "хвост" просто не используется. Если же атрибуты не умещаются в отведенное им пространство, создается дополнительная файловая запись (extra FILE Record), ссылающаяся на свою предшественницу.

Листинг 7.1. Структура файловой записи

```
FILE Record
        Header                  ; Заголовок
        Attribute 1             ; Атрибут 1
        Attribute 2             ; Атрибут 2
        ...                     ; ...
        Attribute N             ; Атрибут N
End Marker (FFFFFFFFh)          ; Маркер конца
```

Первые четыре байта заголовка заняты "магической последовательностью" FILE, сигнализирующей о том, что мы имеем дело с файловой записью типа FILE Record. При восстановлении сильно фрагментированного файла $MFT это обстоятельство играет решающую роль, поскольку позволяет отличить сектора, принадлежащие MFT, от всех остальных секторов.

Следом за сигнатурой идет 16-разрядный указатель, содержащий смещение *последовательности обновления* (update sequence). Под "указателем" здесь и до конца раздела подразумевается смещение от начала сектора, отсчитываемое от нуля и выраженное в байтах. В Windows NT и Windows 2000 это поле всегда равно 002Ah, поэтому для поиска файловых записей можно использовать сигнатуру FILE*\x00, что уменьшает вероятность ложных срабатываний. В Windows XP и более новых версиях последовательность обновления хранится по смещению 002Dh, и поэтому сигнатура приобретает следующий вид: FILE-\x00.

Размер заголовка также варьируется от одной операционной системы к другой и в явном виде нигде не хранится. Вместо этого в заголовке присутствует указатель на первый атрибут, содержащий его смещение в байтах относительно начала файловой записи и расположенный по смещению 14h байтов от начала сектора. Смещения последующих атрибутов (если они есть) определяются путем сложения размеров всех предыдущих атрибутов (размер каждого из атрибутов содержится в его заголовке) со смещением первого атрибута. За концом последнего атрибута находится маркер конца — значение FFFFFFFFh.

Длина файловой записи хранится в двух полях. Тридцатидвухразрядное поле *реального размера* (real size), находящееся по смещению 18h байтов от начала сек-

тора, содержит совокупный размер заголовка, всех его атрибутов и маркера конца, округленный по 8-байтовой границе. Тридцатидвухразрядное поле *выделенного размера* (allocated size), находящееся по смещению 1Ch байтов от начала сектора, содержит действительный размер файловой записи в байтах, округленный по размеру сектора. Документация утверждает, что выделенный размер должен быть кратен размеру кластера, но на практике это не так. Например, на моей машине длина поля выделенного размера равна четверти кластера.

16-разрядное поле флагов, находящееся по смещению 16h байтов от начала сектора, в подавляющем большинстве случаев принимает одно из следующих трех значений: 00h — данная файловая запись не используется или ассоциированный с ней файл или каталог удален, 01h — файловая запись используется и описывает файл, 02h — файловая запись используется и описывает каталог.

64-разрядное поле, находящееся по смещению 20h байтов от начала сектора, содержит индекс базовой файловой записи. Для первой файловой записи это поле всегда равно нулю, а для всех последующих, расширенных записей — индексу первой файловой записи. Расширенные файловые записи могут находиться в любых областях MFT, не обязательно расположенных рядом с основной записью. Следовательно, необходим какой-то механизм, обеспечивающий быстрый поиск расширенных файловых записей, принадлежащих данному файлу (просматривать всю MFT было бы слишком нерационально). Этот механизм существует и основан на ведении списков атрибутов ($ATTRIBUTE_LIST). Список атрибутов представляет собой специальный атрибут, добавляемый к первой файловой записи и содержащий индексы расширенных записей. Формат списка атрибутов будет подробно описан далее в этой главе.

Основные поля заголовка файловой записи описаны в табл. 7.3. Остальные поля заголовка файловой записи не столь важны и поэтому здесь не рассматриваются.

Таблица 7.3. *Структура заголовка файловой записи (FILE Record)*

Смещение	Размер (байт)	ОС	Описание
00h	4	Любая	Сигнатура FILE
04h	2	Любая	Смещение номера последовательности обновления (update sequence number)
06h	2	Любая	Размер (в словах) номера последовательности обновления и массива обновления (Update Sequence Number & Array), условно S
08h	8	Любая	Номер последовательности файла транзакций ($LogFile Sequence Number или LSN)
10h	2	Любая	Номер последовательности (sequence number)
12h	2	Любая	Счетчик жестких ссылок (hard link)
14h	2	Любая	Смещение первого атрибута

Таблица 7.3 *(окончание)*

Смещение	Размер (байт)	ОС	Описание	
16h	2	Любая	Флаги	
			Значение	**Описание**
			0x00	Файловая запись не используется
			0x01	Файловая запись используется и описывает файл
			0x02	Файловая запись используется и описывает каталог
			0x04	За справками обращайтесь к Биллу Гейтсу — вероятно, только он это знает
			0x08	За справками обращайтесь к Биллу Гейтсу — вероятно, только он это знает
18h	4	Любая	Реальный размер (real size) файловой записи	
1Ch	4	Любая	Выделенный размер (allocated size) файловой записи	
20h	8	Любая	Ссылка (file reference) на базовую файловую запись (base FILE record) или нуль, если данная файловая запись является базовой	
28h	2	Любая	Идентификатор следующего атрибута (next attribute ID)	
2Ah	2	Windows XP и позже	Используется для выравнивания	
2Ch	4	Windows XP и позже	Индекс данной файловой записи (number of this MFT record)	
	2	Любая	Номер последовательности обновления (update sequence number)	
	2S-2	Любая	Массив последовательности обновления (update sequence array)	

Последовательность обновления

Будучи очень важными компонентами файловой системы, `$MFT`, `INDEX` и `$LogFile` нуждаются в механизме контроля целостности своего содержимого. Традиционно для этого служат коды обнаружения и коррекции ошибок (ECC/EDC codes). Однако на тот момент, когда проектировалась NTFS, процессоры были не настолько быстрыми, как теперь, и расчет корректирующих кодов занимал значительное время, существенно снижающее производительность файловой системы. Именно поэтому от использования корректирующих кодов пришлось отказаться. Вместо них разработчики NTFS применили так называемые *последовательности обновления* (update sequences), также называемые *fix-ups*.

В конец каждого из секторов, образующих файловую запись (`INDEX Record`, `RCRD Record` или `RSTR Record`), записывается специальный 16-байтовый *номер последова-*

тельности обновления (update sequence number), дублируемый в заголовке файловой записи. При каждой операции чтения два последних байта сектора сверяются с соответствующим полем заголовка и, если драйвер NTFS обнаруживает расхождение, данная файловая запись считается недействительной.

Основное назначение последовательностей обновления — защита от "обрыва записи". Если в процессе записи сектора на диск исчезнет питающее напряжение, может случиться так, что часть файловой записи будет записана успешно, а другая часть сохранит прежнее содержимое (файловая запись, как мы помним, обычно состоит из двух секторов). После восстановления питания драйвер файловой системы не может уверенно определить, была ли файловая запись сохранена целиком. Вот тут-то последовательности обновления и выручают! При каждой перезаписи сектора последовательность обновления увеличивается на единицу. Потому, если произошел обрыв записи, значение последовательности обновления, находящейся в заголовке файловой записи, не совпадет с последовательностью обновления, расположенной в конце сектора.

Оригинальное содержимое, расположенное "под" последовательностью обновления, хранится в специальном *массиве обновления* (update sequence array), находящемся в заголовке файловой записи непосредственно за концом смещения последовательности обновления (update sequence number). Для восстановления файловой записи в исходный вид необходимо извлечь из заголовка указатель на смещение последовательности обновления (он хранится по смещению `04h` байтов от начала заголовка) и сверить лежащее по этому адресу 16-байтовое значение с последним словом каждого из секторов, слагающих файловую запись (`INDEX Record`, `RCRD Record` или `RSTR Record`). Если они не совпадут, значит, соответствующая структура данных повреждена. Использовать такие структуры следует очень осторожно (на первых порах лучше вообще отказаться от этого).

По смещению `006h` от начала сектора находится 16-разрядное поле, хранящее совокупный размер номера последовательности обновления вместе с массивом последовательности обновления (`sizeof(update sequence number) + sizeif(update sequence array)`), выраженный в словах (не в байтах!). Так как размер номера последовательности обновления всегда равен одному слову, то размер массива последовательности обновления, выраженный в байтах, должен вычисляться следующим образом: `(update sequence number & update sequence array - 1)*2`. Таким образом, смещение массива оригинального содержимого равно: `(offset to update sequence number) + 2`. В Windows NT и Windows 2000 номер последовательности обновления всегда располагается по смещению `2Ah` от начала заголовка файловой записи или индексного заголовка, а поле `update sequence array` — по смещению `2Ch`. В Windows XP и более новых операционных системах эти значения располагаются по смещениям `2Dh` и `2Fh` соответственно.

Первое слово массива последовательности обновления соответствует последнему слову первого сектора файловой записи или индексной записи. Второе — последнему слову второго сектора и т. д. Для восстановления сектора в исходный вид необходимо вернуть все элементы массива последовательности обновления на их законные места (естественно, модифицируется не сам сектор, а его копия в памяти).

Чтобы проиллюстрировать сказанное, рассмотрим пример, приведенный в листинге 7.2.

Листинг 7.2. Оригинальная файловая запись до восстановления

```
--> начало первого сектора FILE Record
00000000:  46 49 4C 45-2A 00 03 00-7C 77 1A 04-02 00 00 00  FILE* ♥ |w→♦●
00000010:  01 00 02 00-30 00 01 00-28 02 00 00-00 04 00 00  ☺ ● 0 ☺ (●    ♦
00000020:  00 00 00 00-00 00 00 00-06 00 06 00-00 00 47 11          ♠ ♠   G◄
...
000001F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 06 00                ♠
<-- конец первого сектора FILE Record
...
000003F0:  07 CC E1 0D-00 09 00 00-FF FF FF FF-82 79 06 00  •Ia♪ ○  yyyyBy♠
<-- конец второго сектора FILE Record
```

Сигнатура `FILE` указывает на начало файловой записи, следовательно, по смещению `04h` байтов будет расположен 16-разрядный указатель на номер последовательности обновления. В данном случае он равен `002Ah`. Очень хорошо! Переходим по смещению `002Ah` и видим, что здесь находится слово `0006h`. Перемещаемся в конец сектора и сверяем его с последними двумя байтами. Как и предполагалось, они совпадают. Повторяем ту же самую операцию со следующим сектором. Собственно говоря, количество секторов может и не равняться двум. Чтобы не гадать на кофейной гуще, необходимо извлечь 16-разрядное значение, расположенное по смещению `06h` от начала файловой записи (в данном случае оно равно `0003h`), и вычесть из него единицу. Действительно, получается два сектора.

Теперь нам необходимо найти массив последовательности обновления, хранящий оригинальное значение последнего слова каждого из секторов. Смещение массива обновления равно значению указателя на последовательность обновления, увеличенной на два, т. е. в данном случае `002Ah + 02h = 002Ch`. Извлекаем первое слово (в данном случае равное `00h 00h`) и записываем его в конец первого сектора. Извлекаем следующее слово (`47h 11h`) и записываем его в конец второго сектора.

В результате восстановленный сектор будет выглядеть, как показано в листинге 7.3.

Листинг 7.3. Восстановленная файловая запись

```
--> Начало первого сектора файловой записи
00000000:  46 49 4C 45-2A 00 03 00-7C 77 1A 04-02 00 00 00  FILE* ♥ |w→♦●
00000010:  01 00 02 00-30 00 01 00-28 02 00 00-00 04 00 00  ☺ ● 0 ☺ (●    ♦
00000020:  00 00 00 00-00 00 00 00-06 00 06 00-00 00 47 11          ♠ ♠   G◄
...
000001F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00                ♠
<-- Конец первого сектора файловой записи
...
000003F0:  07 CC E1 0D-00 09 00 00-FF FF FF FF-82 79 47 11  •Ia♪ ○  yyyyBy♠
<-- Конец второго сектора файловой записи
```

Внимание!

В листинге 7.3 `FILE Record`, `INDEX Record`, `RCRD Record` или `RSTR Record` искажены последовательностями обновления и в обязательном порядке должны быть восстановлены перед их использованием, в противном случае вместо актуальных данных вы получите мусор!

Атрибуты

Структурно всякий атрибут состоит из *атрибутного заголовка* (attribute header) и *тела атрибута* (attribute body). Заголовок атрибута всегда хранится в файловой записи, расположенной внутри MFT. Тела резидентных атрибутов хранятся там же. Нерезидентные атрибуты хранят свое тело вне MFT, в одном или нескольких кластерах, перечисленных в заголовке данного атрибута в специальном списке. Если 8-разрядное поле, расположенное по смещению `08h` байтов от начала атрибутного заголовка, равно нулю, то атрибут считается резидентным, а если единице — то атрибут нерезидентен. Любые другие значения недопустимы.

Первые 4 байта атрибутного заголовка определяют его тип. Тип атрибута, в свою очередь, определяет формат представления тела атрибута. В частности, тело атрибута данных (тип: `80h` — `$DATA`) представляет собой "сырую" последовательность байтов. Тело атрибута стандартной информации (тип: `10h` — `$STANDARD_INFORMATION`) описывает время его создания, права доступа и т. д. Более подробно эта тема будет рассмотрена далее.

Следующие 4 байта заголовка содержат длину атрибута, выражаемую в байтах. Длина нерезидентного атрибута равна сумме длин его тела и заголовка, а длина резидентного атрибута равна длине его заголовка. Если к смещению атрибута добавить его длину, мы получим указатель на следующий атрибут (или маркер конца, если текущий атрибут — последний в цепочке).

Длина тела резидентных атрибутов, выраженная в байтах, хранится в 32-разрядном поле, расположенном по смещению `10h` байтов от начала атрибутного заголовка. 16-разрядное поле, следующее за его концом, хранит смещение резидентного тела, отсчитываемое от начала атрибутного заголовка. С нерезидентными атрибутами в этом плане все намного сложнее, и для хранения длины их тела используется множество полей. *Реальный размер тела атрибута* (real size of attribute), выраженный в байтах, хранится в 64-разрядном поле, находящемся по смещению `30h` байтов от начала атрибутного заголовка. Следующее за ним 64-разрядное поле хранит *инициализированный размер потока* (initialized data size of the stream), выраженный в байтах. Судя по всему, инициализированный размер потока всегда равен реальному размеру тела атрибута. 64-разрядное поле, расположенное по смещению `28h` байтов от начала атрибутного заголовка, хранит *выделенный размер* (allocated size of attribute), выраженный в байтах и равный реальному размеру тела атрибута, округленному до размера кластера (в большую сторону).

Два 64-разрядных поля, расположенные по смещениям `10h` и `18h` байтов от начала атрибутного заголовка, задают первый (starting VCN) и последний (last VCN) номера виртуального кластера, принадлежащего телу нерезидентного атрибута. Вирту-

альные кластеры представляют собой логические номера кластеров, не зависящие от своего физического расположения на диске. В подавляющем большинстве случаев номер первого кластера тела нерезидентного атрибута равен нулю, а последний — числу кластеров, занятых телом атрибута, уменьшенному на единицу. 16-разрядное поле, расположенное по смещению 20h от начала атрибутного заголовка, содержит указатель на массив Data Runs, находящийся внутри этого заголовка и описывающий логический порядок размещения нерезидентного тела атрибута на диске.

Каждый атрибут имеет свой собственный *идентификатор* (attribute ID), уникальный для данной файловой записи и хранящийся в 16-разрядном поле, расположенном по смещению 0Eh от начала атрибутного заголовка.

Если атрибут имеет *имя* (attribute Name), то 16-разрядное поле, расположенное по смещению 0Ah байтов от атрибутного заголовка, содержит указатель на него. Для безымянных атрибутов оно равно нулю (большинство атрибутов имен не имеют). Имя атрибута хранится в атрибутном заголовке в формате UNICODE, а его длина определяется 8-разрядным полем, расположенным по смещению 09h байтов от начала атрибутного заголовка.

Если тело атрибута сжато, зашифровано или разрежено, 16-разрядное поле флагов, расположенное по смещению 0Ch байтов от начала атрибутного заголовка, не равно нулю.

Основные поля резидентных и нерезидентных атрибутов кратко описаны в табл. 7.4 и 7.5. Остальные поля не играют существенной роли и поэтому здесь не рассматриваются.

Таблица 7.4. *Структура резидентного атрибута*

Смещение	Размер (байт)	Значение	Описание	
00h	4		Тип атрибута (например, 0x10, 0x60, 0xB0)	
04h	4		Длина атрибута, включая этот заголовок	
08h	1	00h	Флаг нерезидентности (non-resident flag)	
09h	1	N	Длина имени атрибута (нуль, если атрибут безымянный)	
0Ah	2	18h	Смещение имени (нуль, если атрибут безымянный)	
0Ch	2	00h	Флаги	
			Значение	**Описание**
			0001h	Сжатый атрибут (compressed)
			4000h	Зашифрованный атрибут (encrypted)
			8000h	Разреженный атрибут (sparse)
0Eh	2		Идентификатор атрибута (attribute ID)	
10h	4	L	Длина тела атрибута, без заголовка	
14h	2	2N+18h	Смещение тела атрибута	

***Таблица 7.4** (окончание)*

Смещение	Размер (байт)	Значение	Описание
16h	1		Индексный флаг
17h	1	00h	Используется для выравнивания
18h	2N	UNICODE	Имя атрибута (если есть)
2N+18h	L		Тело атрибута

***Таблица 7.5.** Структура нерезидентного атрибута*

Смещение	Размер (байт)	Значение	Описание	
00h	4		Тип атрибута (например, 0x20, 0x80)	
04h	4		Длина атрибута, включая этот заголовок	
08h	1	01h	Флаг нерезидентности (non-resident flag)	
09h	1	N	Длина имени атрибута (нуль, если атрибут безымянный)	
0Ah	2	40h	Смещение имени (нуль, если атрибут безымянный)	
0Ch	2		Флаги	
			Значение	**Описание**
			0001h	Сжатый атрибут (compressed)
			4000h	Зашифрованный атрибут (encrypted)
			8000h	Разреженный атрибут (sparse)
0Eh	2		Идентификатор атрибута (attribute ID)	
10h	8		Начальный виртуальный кластер (starting VCN)	
18h	8		Конечный виртуальный кластер (last VCN)	
20h	2	2N+40h	Смещение списка отрезков (data runs)	
22h	2		Размер блока сжатия (compression unit size), округленный до 4 байтов в бóльшую сторону	
24h	4	00h	Используется для выравнивания	
28h	8		Выделенный размер (allocated size), округленный до размера кластера	
30h	8		Реальный размер (real size)	
38h	8		Инициализированный размер потока (initialized data size of the stream)	
40h	2N	UNICODE	Имя атрибута (если есть)	
2N+40h	..		Список отрезков (data runs)	

Типы атрибутов

NTFS поддерживает большее количество предопределенных типов атрибутов, перечисленных в табл. 7.6. Тип атрибута определяет его назначение и формат представления тела. Полное описание всех атрибутов заняло бы не одну главу, а целую книгу, поэтому здесь приводятся лишь наиболее "ходовые" из них.

Таблица 7.6. *Основные типы атрибутов*

Значение	ОС	Условное обозначение	Описание
010h	Любая	$STANDARD_INFORMATION	Стандартная информация о файле (время, права доступа)
020h	Любая	$ATTRIBUTE_LIST	Список атрибутов
030h	Любая	$FILE_NAME	Полное имя файла
040h	Windows NT	$VOLUME_VERSION	Версия тома
040h	Windows 2000 и позже	$OBJECT_ID	Глобально уникальный идентификатор (GUID) и прочие ID
050h	Любая	$SECURITY_DESCRIPTOR	Дескриптор безопасности и списки прав доступа (ACL)
060h	Любая	$VOLUME_NAME	Имя тома
070h	Любая	$VOLUME_INFORMATION	Информация о томе
080h	Любая	$DATA	Основные данные файла
090h	Любая	$INDEX_ROOT	Корень индексов
0A0h	Любая	$INDEX_ALLOCATION	Ветви (sub-nodes) индекса
0B0h	Любая	$BITMAP	Карта свободного пространства
0C0h	Windows NT	$SYMBOLIC_LINK	Символическая ссылка
0C0h	Windows 2000 и позже	$REPARSE_POINT	Для сторонних производителей
0D0h	Любая	$EA_INFORMATION	Расширенные атрибуты для HPFS
0E0 h	Любая	$EA	Расширенные атрибуты для HPFS
0F0h	Windows NT	$PROPERTY_SET	Устарело и ныне не используется
100h	Windows 2000 и позже	$LOGGED_UTILITY_STREAM	Используется шифрующей файловой системой (EFS)

$STANDARD_INFORMATION

Атрибут стандартной информации описывает время создания/изменения/ последнего доступа к файлу и права доступа, а также некоторую другую вспомогательную информацию (например, квоты). Структура атрибута стандартной информации кратко описана в табл. 7.7.

Таблица 7.7. *Структура атрибута* `$STANDARD_INFORMATION`

Смещение	Размер (байт)	ОС	Описание	
~ ~		Любая	Стандартный атрибутный заголовок (standard attribute header)	
00h	8	Любая	C — время создания (creation) файла	
08h	8	Любая	A — время изменения (altered) файла	
10h	8	Любая	M — время изменения файловой записи (MFT changed)	
18h	8	Любая	R — время последнего чтения (read) файла	
20h	4	Любая	Права доступа MS-DOS (MS-DOS file permissions)	
			Значение	**Описание**
			0001h	Только на чтение (read-only)
			0002h	Скрытый (hidden)
			0004h	Системный (system)
			0020h	Архивный (archive)
			0040h	Устройство (device)
			0080h	Обычный (normal)
			0100h	Временный (temporary)
			0200h	Разреженный (sparse) файл
			0400h	Точка передачи (reparse point)
			0800h	Сжатый (compressed)
			1000h	Офлайновый (offline)
			2000h	Неиндексируемый (not content indexed)
			4000h	Зашифрованный (encrypted)
24h	4	Любая	Старшее двойное слово номера версии (maximum number of versions)	
28h	4	Любая	Младшее двойное слово номера версии (version number)	
2Ch	4	Любая	Идентификатор класса (class ID)	
30h	4	Windows 2000 и позже	Идентификатор владельца (owner ID)	
34h	4	Windows 2000 и позже	Идентификатор безопасности (security ID)	
38h	8	Windows 2000 и позже	Количество квотируемых байтов (quota charged)	
40h	8	Windows 2000 и позже	Номер последней последовательности обновления (update sequence number USN)	

$ATTRIBUTE_LIST

Атрибут списка атрибутов (прямо каламбур) используется в тех случаях, когда все атрибуты файла не умещаются в базовой файловой записи и файловая система вынуждена располагать их в расширенных файловых записях. Индексы расширенных файловых записей содержатся в атрибуте списка атрибутов, помещаемом в базовую файловую запись.

При каких обстоятельствах атрибуты не умещаются в одной файловой записи? Это может произойти в следующих случаях:

- файл содержит много альтернативных имен или жестких ссылок;
- файл сильно фрагментирован;
- файл содержит очень сложный дескриптор безопасности;
- файл имеет очень много потоков данных (т. е. атрибутов типа `$DATA`).

Структура атрибута списка атрибутов приведена в табл. 7.8.

***Таблица 7.8.** Структура атрибута `$ATTRIBUTE_LIST`*

Смещение	Размер (байт)	Описание
~ ~		Стандартный атрибутный заголовок (standard attribute header)
00h	4	Тип (type) атрибута (см. табл. 7.6)
04h	2	Длина записи (record length)
06h	1	Длина имени (name length) или нуль, если нет, условно — N
07h	1	Смещение имени (offset to name) или нуль, если нет
08h	8	Начальный виртуальный кластер (starting VCN)
10h	8	Ссылка на базовую/расширенную файловую запись
18h	2	Идентификатор атрибута (attribute ID)
1Ah	2N	Если N > 0, то имя в формате UNICODE

$FILE_NAME

Атрибут полного имени файла хранит имя файла в соответствующем пространстве имен. Таких атрибутов у файла может быть и несколько (например, имя win32 и имя MS-DOS). Здесь же хранятся и жесткие ссылки (hard link), если они есть.

Структура атрибута полного имени приведена в табл. 7.9.

***Таблица 7.9.** Структура атрибута `$FILE_NAME`*

Смещение	Размер (байт)	Описание
~ ~		Стандартный атрибутный заголовок (standard attribute header)
00h	8	Ссылка (file reference) на материнский каталог

Таблица 7.9 (окончание)

Смещение	Размер (байт)	Описание
08h	8	C — время создания (creation) файла
10h	8	A — время последнего изменения (altered) файла
18h	8	M — время последнего изменения файловой записи (MFT changed)
20h	8	R — время последнего чтения (read) файла
28h	8	Выделенный размер (allocated size) файла
30h	8	Реальный размер (real size) файла
38h	4	Флаг (см. табл. 7.7)
3Ch	4	Используется HPFS
40h	1	Длина имени в символах — L
41h	1	Пространство имен файла (filename namespace)
42h	2L	Имя файла в формате UNICODE без завершающего нуля

Списки отрезков

Тела нерезидентных атрибутов хранятся на диске в одной или нескольких кластерных цепочках, называемых *отрезками* (runs). Отрезком называется последовательность смежных кластеров, характеризующаяся номером начального кластера и длиной. Совокупность отрезков называется *списком* (run-list или data run).

Внутренний формат представления списков не то чтобы сложен, но простым его тоже не назовешь. Для экономии места длина отрезка и номер начального кластера хранятся в полях переменной длины. Если размер отрезка умещается в один байт (т. е. его значение не превышает 255), то он займет один байт. По аналогии если размер отрезка требует для своего представления двойного слова, то он займет двойное слово.

Сами же поля размеров хранятся в 4-битовых ячейках, называемых *нибблами* (nibble) или *полубайтами*. Шестнадцатеричная система счисления позволяет легко переводить байты в нибблы и наоборот. Младший ниббл равен (X & 15), а старший — (X / 16). Иначе говоря, младший ниббл соответствует младшему шестнадцатеричному разряду байта, а старший — старшему. Например, 69h состоит из двух нибблов, причем младший равен 9h, а старший — 6h.

Список отрезков представляет собой массив структур, каждая из которых описывает характеристики "своего" отрезка. Структура элемента списка отрезков показана в табл. 7.10. В конце списка находится завершающий нуль. Первый байт структуры состоит из двух нибблов: младший задает длину поля начального кластера отрезка (условно обозначаемого буквой F), а старший — число кластеров в отрезке (L). Затем идет поле длины отрезка. В зависимости от значения L оно может занимать от

одного до восьми байтов (поля большей длины недопустимы). Первый байт поля стартового кластера файла расположен по смещению 1 + L байтов от начала структуры (что соответствует 2+2*L нибблам).

Таблица 7.10. *Структура одного элемента списка отрезков*

Смещение в нибблах	Размер в нибблах	Описание
0	1	Размер поля длины (L)
1	1	Размер поля начального кластера (S)
2	2*L	Количество кластеров в отрезке
2+2*L	2*S	Номер начального кластера отрезка

Покажем, как с этим работать на практике. Предположим, что мы имеем следующий список отрезков, соответствующий нормальному нефрагментированному файлу (что может быть проще!): 21 18 34 56 00. Попробуем его декодировать?

Начнем с первого байта — 21h. Младший полубайт (01h) описывает размер поля длины отрезка, старший (02h) — размер поля начального кластера. Следующие несколько байтов представляют поле длины отрезка, размер которого в данном случае равен одному байту — 18h. Два других байта (34h 56h) задают номер начального кластера отрезка. Нулевой байт на конце сигнализирует о том, что это последний отрезок в файле. Таким образом, наш файл состоит из одного-единственного отрезка, начинающегося с кластера 5634h и заканчивающегося кластером 5634h + 18h = 564Ch.

Рассмотрим более сложный пример фрагментированного файла со следующим списком отрезков: 31 38 73 25 34 32 14 01 E5 11 02 31 42 AA 00 03 00. Извлекаем первый байт — 31h. Один байт приходится на поле длины, и три байта — на поле начального кластера. Таким образом, первый отрезок (run 1) начинается с кластера 342573h и продолжается вплоть до кластера 342573h + 38 = 3425ABh. Чтобы найти смещение следующего отрезка в списке, мы складываем размер обоих полей с их начальным смещением: 3 + 1 = 4. Отсчитываем четыре байта от начала списка отрезков и переходим к декодированию следующего отрезка: 32h — два байта на поле длины отрезка (равное в данном случае 0114h) и три байта на поле номера начального кластера (0211E5h). Следовательно, второй отрезок (run 2) начинается с кластера 0211E5h и продолжается вплоть до кластера 0211E5h + 114h = 212F9h. Третий отрезок (run 3): 31h — один байт на поле длины и три байта на поле начального кластера, равные 42h и 0300AAh соответственно. Поэтому третий отрезок (run 3) начинается с кластера 0300AAh и продолжается вплоть до кластера 0300AAh + 42h = 300ECh. Завершающий нуль на конце списка отрезков сигнализирует о том, что это последний отрезок в файле.

Таким образом, подопытный файл состоит из трех отрезков, разбросанных по диску в следующем "живописном" порядке: 342573h – 3425ABh; 0211E5h – 212F9h; 0300AAh – 300ECh. Остается только прочитать его с диска! Нет ничего проще!

Начиная с версии 3.0, NTFS поддерживает разреженные (sparse) атрибуты, т. е. такие атрибуты, которые не записывают на диск кластеры, содержащие одни нули. При этом поле номера начального кластера отрезка может быть равным нулю, что означает, что данному отрезку не выделен никакой кластер. Поле длины содержит количество кластеров, заполненных нулями. Их не нужно считывать с диска. Вы должны самостоятельно изготовить их в памяти. Между прочим, далеко не все дисковые доктора знают о существовании разреженных атрибутов (если атрибут разрежен, его флаг равен `8000h`) и интерпретируют нулевую длину поля номера начального кластера весьма странным образом. Последствия такого "лечения" обычно оказываются весьма печальными.

Пространства имен

NTFS изначально проектировалась как файловая система, не зависящая от платформы, способная работать с большим количеством различных подсистем, в том числе: win32, MS-DOS, POSIX. Так как каждая из перечисленных подсистем налагает собственные ограничения на набор символов, допустимых для использования в имени файла, NTFS вынуждена поддерживать несколько независимых пространств имен (name spaces).

POSIX

Допустимы все символы UNICODE (с учетом регистра), за исключением символа нуля (`NULL`), обратной косой черты (`\`) и знака двоеточия (`:`). Последнее из перечисленных ограничений, кстати говоря, не есть ограничение POSIX. Напротив, это внутреннее ограничение файловой системы NTFS, использующей этот символ для доступа к именованным атрибутам. Максимально допустимая длина имени составляет 255 символов.

Win32

Доступны все символы UNICODE (без учета регистра), за исключением следующего набора: кавычки (`"`), звездочка (`*`), косая черта (`/`), двоеточие (`:`), знак "меньше" (`<`), знак "больше" (`>`), вопросительный знак (`?`), обратная косая черта (`\`), а также символ конвейера (`|`). Кроме того, имя файла не может заканчиваться точкой или пробелом. Максимально допустимая длина имени составляет 255 символов.

MS-DOS

Доступны все символы пространства имен win32 (без учета регистра), за исключением следующих: знак плюс (`+`), запятая (`,`), точка (`.`), точка с запятой (`;`), знак равенства (`=`). Длина имени файла не должна превышать восьми символов, за которыми следует необязательное расширение имени файла, имеющее длину от одного до трех символов.

Назначение служебных файлов

NTFS содержит большое количество служебных файлов (метафайлов) строго определенного формата. Важнейший из метафайлов, $MFT, мы только что рассмотрели. Остальные метафайлы играют вспомогательную роль. Для восстановления данных детально знать их структуру необязательно. Тем не менее если они окажутся искажены, то штатный драйвер файловой системы не сможет работать с таким томом, поэтому иметь некоторые представления о назначении каждого из них все же необходимо.

Краткие сведения о назначении важнейших метафайлов приведены в табл. 7.11. К сожалению, в пределах одной главы нет возможности подробно рассмотреть структуру всех существующих метафайлов, поэтому заинтересованным читателям рекомендуется искать эту информацию в документации.

Таблица 7.11. *Назначение основных метафайлов NTFS*

Inode	Имя файла	ОС	Описание
0	$MFT	Любая	Главная файловая таблица (Master File Table, MFT)
1	$MFTMirr	Любая	Резервная копия первых четырех элементов MFT
2	$LogFile	Любая	Журнал транзакций (transactional logging file)
3	$Volume	Любая	Серийный номер, время создания, флаг несброшенного кэша (dirty flag) тома
4	$AttrDef	Любая	Определение атрибутов
5	. (точка)	Любая	Корневой каталог (root directory) тома
6	$Bitmap	Любая	Карта свободного/занятого пространства
7	$Boot	Любая	Загрузочная запись (boot record) тома
8	$BadClus	Любая	Список плохих кластеров (bad clusters) тома
9	$Quota	Windows NT	Информация о квотах (quota information)
9	$Secure	Windows 2000 и позже	Использованные дескрипторы безопасности (security descriptors)
10	$UpCase	Любая	Таблица заглавных символов (uppercase characters) для трансляции имен
11	$Extend	Windows 2000 и позже	Каталоги: $ObjId, $Quota, $Reparse, $UsnJrnl
12–15	не используется	Любая	Помечены как использованные, но в действительности пустые
16–23	не используется	Любая	Помечены как неиспользуемые
Любой	$ObjId	Windows 2000 и позже	Уникальные идентификаторы каждого файла
Любой	$Quota	Windows 2000 и позже	Информация о квотах (quota information)

Таблица 7.11 (окончание)

Inode	Имя файла	ОС	Описание
Любой	`$Reparse`	Windows 2000 и позже	Информация о точке передачи (reparse point)
Любой	`$UsnJrnl`	Windows 2000 и позже	Журнал шифрованной файловой системы (journaling of encryption)
> 24	Пользовательский файл	Любая	Обычные файлы
> 24	Пользовательский каталог	Любая	Обычные каталоги

Практический пример

Рассказ о файловой системе NTFS был бы неполным без практической иллюстрации техники разбора файловой записи вручную. До сих пор мы витали в облаках теоретической абстракции. Пора спускаться на грешную землю.

Воспользовавшись любым дисковым редактором, попробуем декодировать одну файловую запись вручную. Найдем сектор, содержащий сигнатуру `FILE` в его начале (не обязательно брать первый встретившийся сектор). Он может выглядеть, например, как в листинге 7.4.

Листинг 7.4. Ручное декодирование файловой записи (разные атрибуты выделены шрифтом разной насыщенности)

```
         :  00 01 02 03 04 05 06 07 | 08 09 0A 0B 0C 0D 0E 0F
00000000:  46 49 4C 45 2A 00 03 00 | 60 79 1A 04 02 00 00 00   FILE* ♥ `y→♦●
00000010:  01 00 01 00 30 00 01 00 | 50 01 00 00 00 04 00 00   ☺ ☺ 0 ☺ P☺   ♦
00000020:  00 00 00 00 00 00 00 00 | 04 00 03 00 00 00 00 00         ♦ ♥

00000030:  10 00 00 00 60 00 00 00 | 00 00 00 00 00 00 00 00   ►  `
00000040:  48 00 00 00 18 00 00 00 | B0 D5 C9 2F C6 0B C4 01   H  ↑   ▒╒╔/╞♂─☺
00000050:  E0 5A B3 7B A9 FA C3 01 | 90 90 F1 2F C6 0B C4 01   pZ|{й·├☺PPё/╞♂─☺
00000060:  50 7F BC FE C8 0B C4 01 | 20 00 00 00 00 00 00 00   P⌂╝■╚♂─☺
00000070:  00 00 00 00 00 00 00 00 | 00 00 00 00 05 01 00 00            ♣☺
00000080:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00

00000090:  30 00 00 00 70 00 00 00 | 00 00 00 00 00 00 02 00   0   p           ●
000000A0:  54 00 00 00 18 00 01 00 | DB 1A 01 00 00 00 01 00   T   ↑ ☺ ▒─☺   ☺
000000B0:  B0 D5 C9 2F C6 0B C4 01 | B0 D5 C9 2F C6 0B C4 01   ▒╒╔/╞♂─☺▒╒╔/╞♂─☺
000000C0:  B0 D5 C9 2F C6 0B C4 01 | B0 D5 C9 2F C6 0B C4 01   ▒╒╔/╞♂─☺▒╒╔/╞♂─☺
000000D0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00
000000E0:  20 00 00 00 00 00 00 00 | 09 03 49 00 6C 00 66 00            o♥I l f
000000F0:  61 00 6B 00 2E 00 64 00 | 62 00 78 00 00 00 00 00   a k . d b x
```

```
00000100:  80 00 00 00 48 00 00 00 | 01 00 00 00 00 00 03 00   A   H   ☺       ♥
00000110:  00 00 00 00 00 00 00 00 | ED 04 00 00 00 00 00 00             э♦
00000120:  40 00 00 00 00 00 00 00 | 00 E0 4E 00 00 00 00 00   @         рN
00000130:  F0 D1 4E 00 00 00 00 00 | F0 D1 4E 00 00 00 00 00   Ёв╤N      Ёв╤N
00000140:  32 EE 04 D9 91 00 00 81 | FF FF FF FF 82 79 47 11   2ю♦┘С  Б     ByG◄
000001F0:  00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 03 00                  ♥
         :  00 01 02 03 04 05 06 07 | 08 09 0A 0B 0C 0D 0F 0F
```

Первым делом необходимо восстановить оригинальное содержимое последовательности обновления. По смещению `04h` от начала сектора лежит 16-разрядный указатель на нее, равный в данном случае `2Ah` (на NTFS разных версий это значение различно, это не меняет его сути). А что у нас находится по смещению `2Ah`? Это два байта `03 00`. Данная последовательность представляет собой номер последовательности обновления. Сверяем его с содержимым двух последних байтов этого и следующего секторов (смещения `1FEh` и `3FEh` соответственно). Они равны! Следовательно, данная файловая запись цела (по крайней мере на первый взгляд), и можно переходить к операции ее восстановления. По смещению `2Ch` расположен массив, содержащий оригинальные значения последовательности обновления. Количество элементов в нем равно содержимому 16-разрядного поля, расположенному по смещению `06h` от начала сектора и уменьшенному на единицу (в данном случае имеем `03h – 01h = 02h`). Извлекаем два слова, начиная со смещения `2Ch` (в данном случае они равны `00 00` и `00 00`), и записываем их в конец первого и последнего секторов.

Теперь нам необходимо выяснить, используется ли данная файловая запись или же ассоциированный с ней файл или каталог был удален. 16-разрядное поле, расположенное по смещению `16h`, содержит значение `01h`. Следовательно, перед нами файл, а не каталог, и этот файл еще не удален. Но является ли эта файловая запись базовой для данного файла или мы имеем дело с ее продолжением? 64-разрядное поле, расположенное по смещению `20h`, равно нулю, следовательно, данная файловая запись — базовая.

Очень хорошо, теперь переходим к исследованию атрибутов. 16-разрядное поле, находящееся по смещению `14h`, равно `30h`, следовательно, заголовок первого атрибута начинается со смещения `30h` от начала сектора.

Первое двойное слово атрибута равно `10h`, значит, перед нами атрибут типа `$STANDARD_INFORMATION`. 32-разрядное поле длины атрибута, находящееся по смещению `04h` и равное в нашем случае `60h` байтов, позволяет нам вычислить смещение следующего атрибута в списке: `30h` (смещение нашего атрибута) + `60h` (его длина) = `90h` (смещение следующего атрибута). Первое двойное слово следующего атрибута равно `30h`, значит, это атрибут типа `$NAME`, и следующее 32-разрядное поле хранит его длину, равную в данном случае `70h`. Сложив длину атрибута с его смещением, мы получим смещение следующего атрибута — `90h + 70h = 100h`. Первое двойное слово третьего атрибута равно `80h`, следовательно, это атрибут типа `$DATA`, хранящий основные данные файла. Складываем его смещение с длиной —

100h + 32h = 132h. И вот здесь мы наткнулись на частокол FFFFFFh, сигнализирующий о том, что атрибут $DATA последний в списке.

Теперь, разбив файловую запись на атрибуты, можно приступить к исследованию каждого из атрибутов в отдельности. Начнем с разбора имени. 8-разрядное поле, находящееся по смещению 08h от начала атрибутного заголовка (и по смещению 98h от начала сектора), содержит флаг нерезидентности. В данном случае этот флаг равен нулю. Это значит, что атрибут резидентный и его тело хранится непосредственно в самой файловой записи, что уже хорошо. 16-разрядное поле, расположенное по смещению 0Ch от начала атрибутного заголовка (и по смещению 9Ch от начала сектора), равно нулю, следовательно, тело атрибута не сжато и не зашифровано. Таким образом, можно приступать к разбору тела атрибута. 32-разрядное поле, расположенное по смещению 10h от начала атрибутного заголовка (и по смещению A0h от начала сектора), содержит длину атрибутного тела, равную в данном случае 54h байтов. 16-разрядное поле, расположенное по смещению 14h от начала атрибутного заголовка и по смещению A4h от начала сектора, хранит смещение атрибутного тела, равное в данном случае 18h. Следовательно, тело атрибута $FILE_NAME располагается по смещению A8h от начала сектора.

Формат атрибута типа $FILE_NAME описан в табл. 7.9. Первые восемь байтов содержат ссылку на родительский каталог этого файла, равную в данном случае 11ADBh:01 (индекс — 11ADBh, номер последовательности — 01h). Следующие 32 байта содержат данные о времени создания, изменения и времени последнего доступа к файлу. По смещению 28h от начала тела атрибута и D0h от начала сектора лежит 64-разрядное поле выделенного размера, а за ним — 64-разрядное поле реального размера. Оба равны нулю, значит, за размером файла следует обращаться к атрибутам типа $DATA.

Длина имени файла содержится в 8-разрядном поле, находящемся по смещению 40h байтов от начала тела атрибута и по смещению E8h от начала сектора. В данном случае оно равно 09h. Само же имя начинается со смещения 42h от начала тела атрибута и со смещения EAh от начала сектора. И здесь находится имя файла **Ilfak.dbx**.

Переходим к атрибуту основных данных файла, пропустив атрибут стандартной информации, который не содержит решительно ничего интересного. 8-разрядный флаг нерезидентности, расположенный по смещению 08h от начала атрибутного заголовка и по смещению 108h от начала сектора, равен 01h, следовательно, атрибут нерезидентный. 16-разрядный флаг, расположенный по смещению 0Ch от начала атрибутного заголовка и по смещению 10Ch от начала сектора, равен нулю, значит, атрибут не сжат и не зашифрован. 8-разрядное поле, расположенное по смещению 09h от начала атрибутного заголовка и по смещению 109h от начала сектора, равно нулю — атрибут безымянный. Реальная длина тела атрибута (в байтах) содержится в 64-разрядном поле, расположенном по смещению 30h от начала атрибутного заголовка и по смещению 130h от начала сектора. В данном случае она равна 4ED1F0h (5 165 552). Два 64-разрядных поля, расположенных по смещениям 10h/110h и 18h/118h байтов от начала атрибутного заголовка/сектора соответственно, содержат

начальный и конечный номер виртуального кластера нерезидентного тела. В данном случае они равны 0000h и 4EDh соответственно.

Остается лишь декодировать список отрезков, адрес которого хранится в 16-разрядном поле, находящемся по смещению 20h от начала атрибутного заголовка и 120h от начала сектора. В данном случае поле равно 40h, что соответствует смещению от начала сектора в 140h. Сам же список отрезков выглядит так: 32 EE 04 D9 91 00 00. Ага! Два байта занимают поле длины (равное в данном случае 04EEh кластерам) и три — поле начального кластера (0091h). Завершающий нуль на конце говорит о том, что этот отрезок последний в списке отрезков.

Подытожим полученную информацию. Файл называется Ilfak.dbx, он начинается с кластера 0091h и продолжается вплоть до кластера 57Fh, при реальной длине файла в 5 165 552 байта. Это все, что надо! Теперь остается только скопировать файл на резервный носитель.

Возможные опасности NTFS

Сейчас мы немного отвлечемся и поговорим о... компьютерных вирусах, обитающих внутри NTFS и активно использующих ее расширения в своих личных целях. Поскольку эта файловая система чрезвычайно широко распространена (благодаря популярности самой Windows), вирусописатели тоже уделяют ей пристальное внимание. Результатом стало появление целого "зоопарка" вредоносных программ, использующих возможности NTFS для своего запуска, работы и собственно деструктивной деятельности. Давайте кратко рассмотрим основные виды таких вредоносов и изучим принципы их действия.

Буткиты

Буткиты (от *англ.* boot — "загрузка" и kit — "инструмент") или так называемые "загрузочные вирусы" — это вирусы или трояны, способные заражать загрузочную запись на диске компьютера, благодаря чему они запускаются либо раньше операционной системы, либо одновременно с ней, но в любом случае перед загрузкой основных средств антивирусной защиты. Из этого логически вытекает основная сложность борьбы с буткитами — поскольку они стартуют еще на раннем этапе загрузки компьютера, буткиты перехватывают некоторые функции управления операционной системой и, как следствие, могут парализовать запуск и нормальную работу антивирусных программ, а также блокировать попытки "вылечить" инфицированное устройство. При неудачном удалении такой угрозы может произойти повреждение логической структуры диска, из-за чего система и вовсе перестанет загружаться и сложное электронное устройство "превратится в кирпич". Этим они и опасны.

Именно борьба с буткитами как явлением и декларируется в качестве одной из причин появления технологии UEFI. Ведь контроль над процессом загрузки, казалось бы, должен обеспечить полную безопасность операционной системы. Тем не

менее универсального средства защиты из UEFI все-таки не вышло: создатели вредоносного ПО приспособились и к этому новшеству, используя в основном обнаруженные в различных версиях EFI программные уязвимости.

Существуют различные разновидности буткитов: одни заражают главную загрузочную запись диска (Master Boot Record, MBR), другие — загрузочную запись тома (Volume Boot Record, VBR). В общем случае алгоритм действия буткита таков: запустившись на инфицированном компьютере, он размещает свою копию в одной из свободных логических областей диска, а затем модифицирует существующую загрузочную запись, внедряя в нее собственный код, который получает управление при запуске операционной системы и загружает в оперативную память основное тело буткита. По окончании этого процесса буткит передает управление дальнейшей загрузкой оригинальной загрузочной записи, позволяя ОС стартовать в штатном режиме. На момент завершения загрузки операционной системы вредоносная программа уже находится в памяти и может выполнять различные деструктивные действия, например перехватывать те или иные системные функции и предотвращать запуск антивирусных программ, а также блокировать пользователю доступ к сайтам их разработчиков.

Особая опасность буткитов заключается еще и в том, что, запускаясь вместе с операционной системой, эти вредоносные программы могут получить в ней максимальные привилегии (например, полномочия администратора), даже если текущий сеанс открыт пользователем с ограниченными системными правами. Таким образом, буткит имеет на зараженном компьютере поистине неограниченные возможности для реализации всяческих вредоносных функций, включая полный доступ к файловой системе, компонентам ОС, памяти и драйверам.

MBR-локеры

Этот вид троянов появился на свет и получил распространение в середине 2010-х годов, как дальнейшая эволюция винлокеров — вредоносов, которые после загрузки Windows блокировали нормальную работу системы, демонстрируя на весь экран баннер с требованием заплатить выкуп за разблокировку. И если обычные винлокеры просто подменяли какой-то из запускающихся в процессе загрузки ОС системных компонентов, например файл *winlogon.exe* (в результате чего такого трояна было несложно удалить), то MBR-локеры меняли загрузочную запись Windows, значительно усложняя процедуру лечения.

Обычно заражение такого рода вредоносом происходит следующим образом. Запустившись на компьютере, MBR-локер пытается выполнить инжект в какое-то из работающих приложений с высокими системными привилегиями, чаще всего — в процесс explorer.exe. Далее, в контексте этого процесса троян модифицирует загрузочную запись Windows таким образом, чтобы после начального этапа загрузки управление передавалось на исполняемый файл вредоносной программы, а оригинальная MBR копируется куда-либо в другое место, например в конец диска. После этого троян пытается завершить работу Windows. После перезагрузки вместо привычного интерфейса ОС пользователь видит на экране примерно такую картинку, которая показана на рис. 7.4.

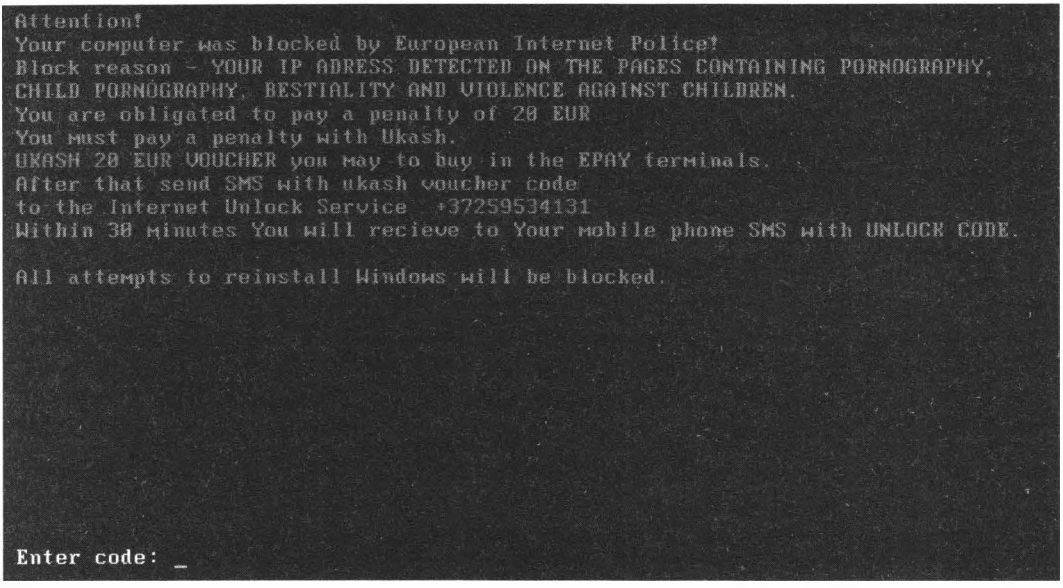


Рис. 7.4. Типичное окно MBR-локера

Очевидно, что в этом случае даже удаление самого исполняемого файла вредоносной программы не приведет к полному излечению системы — ведь для возвращения ее в работоспособное состояние придется восстанавливать загрузочную запись.

Более изощренные версии MBR-локеров троянцев уничтожают таблицу разделов, предварительно сделав копию первого сектора для последующего восстановления, в случае если жертва все-таки заплатит выкуп. Данные, необходимые им для работы, — текст требования и шрифты — они могут сохранять не в инфицированной MBR, а в случайных секторах диска, что также затрудняет лечение. Помимо этого, копия оригинальной загрузочной записи может быть зашифрована трояном, что сделает невозможным ее восстановление в исходном виде.

Энкодеры-шифровальщики

Троянцы-шифровальщики, или, как их еще называют, *энкодеры*, — одна из наиболее опасных компьютерных угроз. Именно их распространение делает крайне актуальным использование современных технологий резервного копирования и восстановления утерянных данных.

Такие троянцы также относятся к условной категории *программ-вымогателей*. Энкодеры впервые появились в 2009 году, и на текущий момент специалистам компьютерной безопасности известно уже порядка 2500 их разновидностей. Запустившись на инфицированном компьютере, энкодеры зашифровывают хранящиеся на дисках пользовательские файлы с применением различных криптостойких алгоритмов, после чего требуют у жертвы выкуп за их расшифровку (рис. 7.5). Отказавшись выплачивать требуемое злоумышленниками денежное вознаграждение, пользователь рискует в одночасье потерять все свои данные.

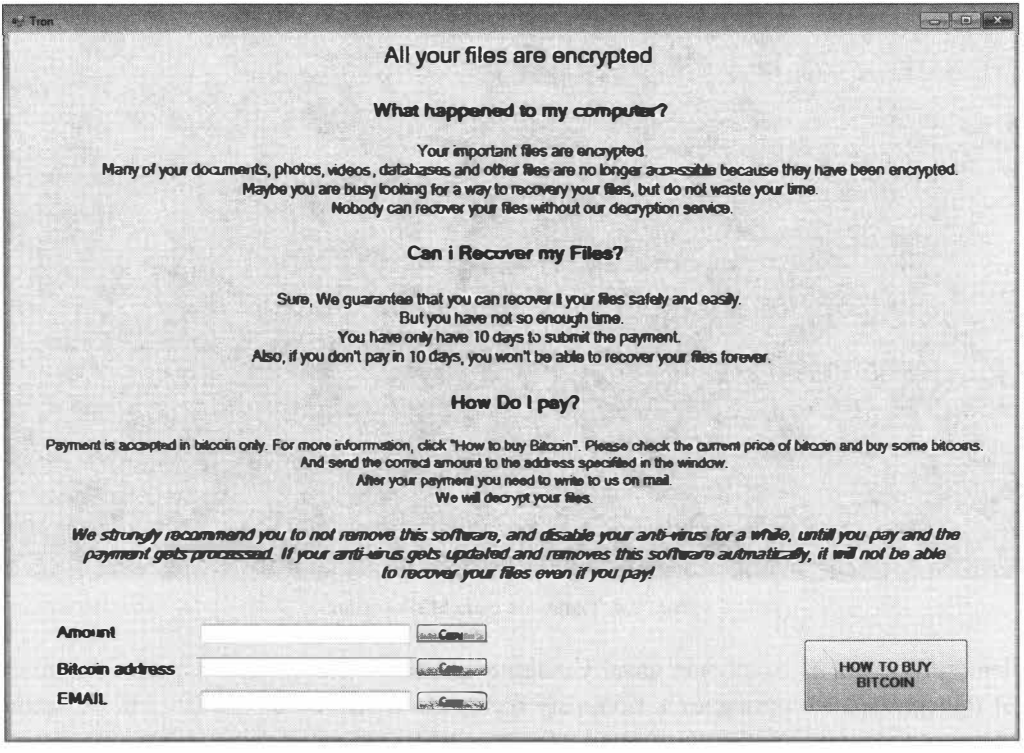


Рис. 7.5. Типичное сообщение с требованием выкупа, демонстрируемое трояном-шифровальщиком

Аппетиты у злоумышленников могут быть различными: вымогаемый ими выкуп может составлять эквивалент и нескольких десятков, и нескольких тысяч долларов. К сожалению, в некоторых случаях восстановить поврежденные данные бывает практически невозможно, даже несмотря на обещания злоумышленников. Причем оплата требуемого киберпреступниками вознаграждения не дает никакой гарантии того, что поврежденные троянцем-энкодером файлы будут когда-либо расшифрованы.

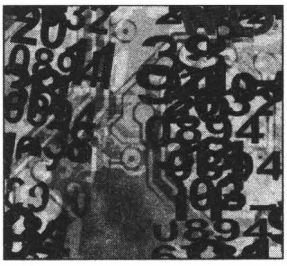
Вредоносные программы данного семейства применяют около двух десятков различных алгоритмов шифрования пользовательских файлов. Существуют определенные модификации троянцев — даже несколько поочередно, с целью затруднить последующую расшифровку информации. Широкому распространению подобных программ способствует то обстоятельство, что злоумышленники нередко выставляют на продажу на подпольных форумах исходные коды троянцев-шифровальщиков, помимо этого существуют специальные программы-конструкторы, с помощью которых вирусописатели могут создавать новые версии троянов, даже не обладая знаниями в области криптографии и программирования. Еще один аргумент, свидетельствующий об опасности таких приложений, заключается в том, что некоторые версии подобных вредоносных программ содержат ошибки, поэтому расшифровать пострадавшие от их действия файлы иногда не в состоянии даже сами создатели троянца. После успешного запуска на компьютере жертвы энкодер в общем случае выполняет перечисленную далее последовательность действий:

- автоматически генерирует по определенному алгоритму ключ шифрования либо получает его с принадлежащего злоумышленникам удаленного сервера;
- осуществляет поиск на дисках зараженного компьютера файлов, удовлетворяющих заданным вирусописателями критериям;
- шифрует все файлы, удовлетворяющие условиям;
- создает и сохраняет на диске документы с перечислением действий для последующей расшифровки файлов и условиями выкупа.

После того как файлы оказываются зашифрованными, троянцы-энкодеры, в зависимости от версии, могут изменить фон рабочего стола атакованного компьютера на графическое изображение с указанием дальнейших инструкций для своей жертвы. В целях конспирации некоторые модификации шифровальщиков размещают свои управляющие серверы в анонимной сети TOR, что значительно затрудняет их идентификацию и последующую расшифровку данных.

На сегодняшний день наиболее эффективный метод противодействия троянцам-энкодерам — использование современного антивирусного ПО, обладающего механизмами *превентивной защиты*, и своевременное *резервное копирование* всей актуальной информации на независимые носители, хранящиеся отдельно от основного компьютера пользователя (в качестве таковых могут в том числе выступать отключаемые облачные хранилища).

ГЛАВА 8



Восстановление ошибочно удаленных файлов на разделах NTFS

Надежность NTFS — это одно, а ошибочно удаленные файлы — совсем другое. Файловая система, даже такая мощная, как NTFS, бессильна защитить пользователя от себя самого. Но вот предусмотреть "откат" последних выполненных действий она вполне может, тем более что транзакции и ведение их журнала в NTFS уже реализованы. До совершенства остается всего лишь один шаг. Однако, к сожалению, Microsoft не торопится сделать этот шаг, возможно, оставляя эти усовершенствования как задел для будущих версий. "Защита" от непреднамеренного удаления реализована исключительно на интерфейсном уровне, а это не только неудобно, но и ненадежно.

Прекрасно, если удаленный файл сохранился в "Корзине", но что делать, если там его не окажется? Эта глава рассказывает о методах восстановления файлов, в том числе и файлов с отсутствующей файловой записью, которые приходится собирать буквально по кластерам.

Пакет *FILE_DISPOSITION_INFORMATION*

`IRP_MJ_SET_INFORMATION/FILE_DISPOSITION_INFORMATION` — это пакеты, посылаемые драйверу при удалении файла (имейте это в виду при дизассемблировании). Чтобы уметь восстанавливать удаленные файлы, необходимо отчетливо представлять, что происходит в процессе удаления файла с раздела NTFS. Вот последовательность выполняемых при этом действий.

1. Корректируется файл `/$MFT:$BITMAP`, каждый бит которого определяет "занятость" соответствующей файловой записи (FILE Record) в MFT (значение 0 говорит о том, что запись не используется).
2. Корректируется файл `/$BITMAP`, каждый бит которого определяет "занятость" соответствующего кластера (значение 0 говорит о том, что кластер не используется).

3. Файловые записи, соответствующие файлу, помечаются как удаленные (поле FLAG, находящееся по смещению 16h от начала файловой записи, сбрасывается в нуль).
4. Ссылка на файл удаляется из двоичного дерева индексов. Технические подробности этого процесса здесь не рассматриваются, поскольку восстановить таблицу индексов вручную сможет только "гуру". Кроме того, в таком восстановлении нет необходимости. Ведь в NTFS индексы играют вспомогательную роль, и гораздо проще переиндексировать каталог заново, чем восстанавливать сбалансированное двоичное дерево (*B*tree*).
5. Обновляется атрибут $STANDART_INFORMATION каталога, в котором хранится удаляемый файл (время последнего доступа и т. д.).
6. В файле /$LogFile обновляется поле Sequence Number (изменения, происходящие в журнале транзакций, здесь не рассматриваются).
7. Поля Update Sequence Number следующих файловых записей увеличиваются на единицу: сам удаляемый файл, текущий каталог, /$MFT, /$MFT:$BITMAP, /$BITMAP, /$BOOT, /$TRACKING.LOG.

Каталоги удаляются практически так же, как и файлы. В этом нет ничего удивительного, т. к. с точки зрения файловой системы каталог тоже представляет файл особого вида, содержащий внутри себя двоичное дерево индексов (B*tree).

Ни в том ни в другом случае физического удаления файла не происходит, и он может быть легко восстановлен. Легкое и быстрое восстановление возможно до тех пор, пока не будет затерта файловая запись (FILE Record), принадлежащая этому файлу и хранящая его резидентное тело или список отрезков (run-list) нерезидентного содержимого. Утрата файловой записи крайне неприятна, поскольку в этом случае файл придется собирать по кластерам. При этом стоит заметить, что чем сильнее был фрагментирован удаленный файл, тем сложнее будет эта задача. К счастью, в отличие от FAT, NTFS не затирает первого символа имени файла, что значительно упрощает восстановление.

Автоматическое восстановление удаленных файлов

Утилиты, восстанавливающие удаленные файлы, не входят в стандартный комплект поставки Windows. Разумеется, это явный недостаток — вспомните, ведь в MS-DOS такая утилита была. Следовательно, эти средства приходится приобретать отдельно. Одна из автоматических утилит для восстановления удаленных файлов — GetDataBack (рис. 8.1). Опасаясь разрушить файловую систему окончательно, большинство таких утилит избегает прямой записи на диск. Вместо этого пользователю предлагается считать удаленный файл и переписать его в другое место, но только не на сам восстанавливаемый раздел. Не слишком-то удачное решение. А если на остальных дисках свободного места нет или восстанавливаемый диск имеет всего лишь один логический раздел? Предположим, вам необходимо восстановить базу данных объемом в несколько гигабайт. Можно, конечно, подключить

второй винчестер, скопировать ее туда, а затем обратно. Однако подумайте, сколько же это займет времени, не говоря уже о том, что сервер лучше не выключать, а "горячую" замену поддерживают далеко не все жесткие диски! Другой недостаток подобных утилит — слишком медленная работа. Вместо того чтобы найти один-единственный файл, имя которого нам известно, они проводят полномасштабные "маневры", сканируя весь раздел целиком. При работе с большими дисками на это уходит от одного до нескольких часов, причем это время фактически тратится впустую.

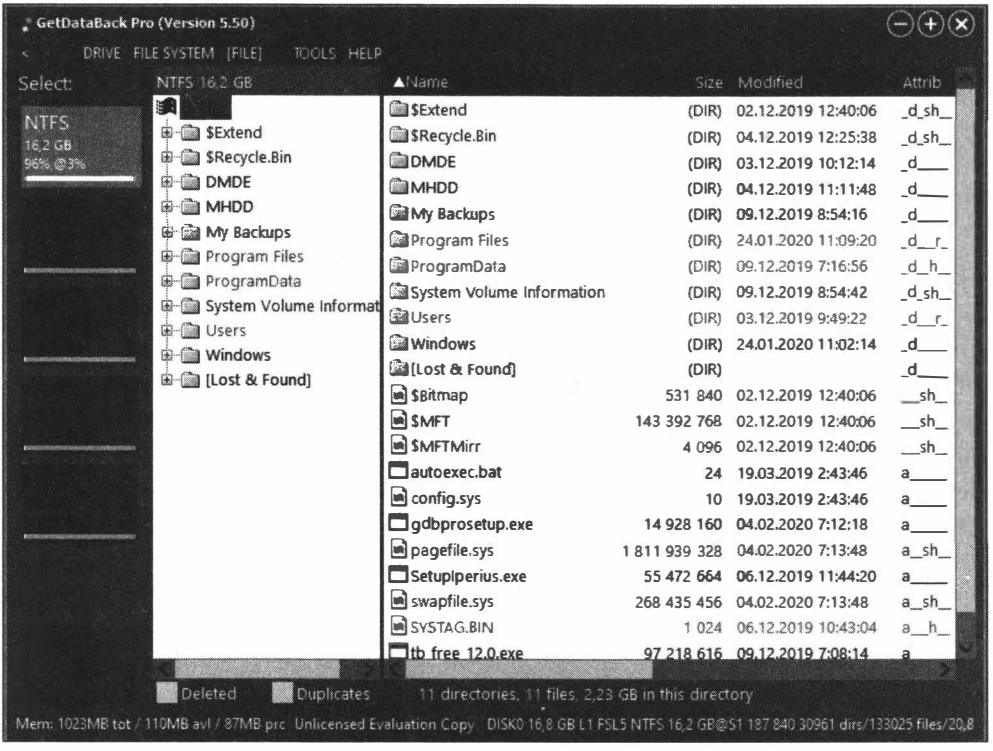


Рис. 8.1. Утилита GetDataBack за восстановлением удаленных файлов

С другой стороны, утилиты, вносящие изменения непосредственно в структуру NTFS, рискуют серьезно повредить дисковый том, после чего ему не помогут даже профессионалы. Настоящие хакеры не доверяют никакому коду, кроме созданного лично ими, особенно если исходные тексты недоступны, а документация туманна и двусмысленна. Различные версии NTFS отличаются друг от друга. Последние радикальные изменения произошли в Windows XP (NTFS версии 3.1) — массив последовательностей обновления (Update Sequence Number-n-Array) переместился на шесть байтов вперед, а его место было отдано под выравнивание и поле номера текущей файловой записи (Number of this MFT Record). С тех пор формат файловой системы не претерпел каких-либо архитектурных изменений.

Наконец, возможна и такая ситуация, когда утилит восстановления просто не окажется под рукой в тот момент, когда вам срочно потребуется восстановить какой-

нибудь ценный файл. Законов Мэрфи еще никто не отменял! В этом случае вам придется рассчитывать только на свои силы.

Ручное восстановление ошибочно удаленных файлов

Начнем с простейшего. Файл только что удален, и принадлежащая ему файловая запись еще не затерта. Как найти его на диске? Существуют два способа — "теоретический" и "практический". Теоретический метод исключительно надежен, но требует дополнительных операций, выполнения которых можно избежать, приняв ряд практических допущений.

Теоретически грамотный и правильный подход состоит в следующем. Извлекаем из загрузочного сектора указатель на MFT, извлекаем из нее первую запись (она описывает `$MFT`), находим атрибут `$DATA` (`80h`), декодируем список отрезков (data runs) и последовательно читаем все записи в MFT, анализируя содержимое атрибута `$FILE_NAME` (`30h`) — имя файла. Обратите внимание, что таких атрибутов у файла может быть несколько. Этот же атрибут хранит ссылку на родительский каталог. Поэтому если несколько одноименных файлов были удалены из различных каталогов, то необходимо выяснить, какой именно из них должен быть восстановлен.

Практический подход выглядит следующим образом. В девяти случаях из десяти файл `$MFT` не фрагментирован и располагается в самом начале диска. Имена файлов хранятся по смещению `EAh` от начала сектора, в начале которого расположена сигнатура `FILE0`. Поэтому можно просто запустить любой дисковый редактор, ввести имя восстанавливаемого файла в кодировке UNICODE и дать редактору указание искать его по смещению `F0h` от начала сектора.

Когда же искомая строка будет найдена, необходимо проверить, находится ли в начале сектора сигнатура `FILE0`. Если такой сигнатуры в начале сектора нет, следует продолжить поиск. Двухбайтовое поле по смещению `16h` от начала сектора содержит флаги записи: `00h` — запись не используется или была удалена, `01h` — запись используется, `02h` — запись используется и описывает каталог. Встречаются и другие значения, например `04h`, `08h`. Эти значения не документированы. Возможно, что именно вы сможете пролить свет на этот вопрос?

Исправляем `00h` на `01h`, записываем изменения и... Ничего не выходит?! А что же вы хотели? Ведь, помимо этого, необходимо выполнить еще несколько действий. Во-первых, следует сообщить файлу `/$MFT:$BITMAP`, что данная запись MFT вновь используется. Во-вторых, необходимо исключить из файла `/$BITMAP` номера кластеров, принадлежащие восстанавливаемому файлу. Наконец, необходимо перестроить двоичное дерево индексов, хранящее содержимое каталога. Первые два пункта не представляют серьезной проблемы, но вот с последней задачей придется повозиться. Хотя ее можно существенно упростить, просто запустив chkdsk с ключом `/F`. Утилита chkdsk самостоятельно найдет "потерянный" файл и внесет все необходимые изменения в файловую систему. От вас потребуется только установить флаг

по смещению 16h в единицу, а все остальное сделает chkdsk. После этих нехитрых манипуляций восстановленный файл окажется в своем родном каталоге.

> ***Примечание***
>
> В Windows 10 утилиту chkdsk можно запустить из командной строки (**cmd.exe**) или из консоли PowerShell. Помните, что выполнение chkdsk требует наличия системных привилегий, поэтому консоль следует запускать от имени Администратора. Если вы пытаетесь выполнить проверку системного раздела, на котором располагается текущая версия ОС, программа не сможет проверить диск, т. к. том окажется заблокированным. В этом случае вам будет предложено выполнить проверку при следующей загрузке Windows.

Восстанавливаем руины

Рассмотрим более сложный случай восстановления, а именно — ситуацию, когда файловая запись уже затерта. Если удаленный файл был резидентным (хранил свое тело в MFT), то восстанавливать уже нечего. Даже если на ранее занимаемом им месте создан нерезидентный файл (а файловая запись нерезидентного файла заканчивается там, где начинается резидентное тело), операционная система заботливо заполнит оставшийся "хвост" нулями, и для восстановления оригинального содержимого придется прибегнуть к дорогостоящему оборудованию, читающему поверхность жесткого диска на физическом уровне.

С нерезидентными файлами, хранящими свое тело вне MFT, ситуация обстоит не так плачевно, хотя и здесь проблем тоже хватает. Порядок размещения файла на диске хранится в списке отрезков (run-list) внутри файловой записи в MFT. При этом, поскольку файловая запись уже затерта, возможен лишь контекстный поиск по содержимому. Запускаем дисковый редактор, вводим последовательность, заведомо содержащуюся в удаленном файле, но не встречающуюся во всех остальных, и даем редактору команду начать поиск. Для ускорения поиска можно искать только в свободном дисковом пространстве (за это отвечает файл /$BITMAP).

Восстановление нефрагментированных файлов осуществляется элементарно. Достаточно просто выделить группу секторов и записать ее содержимое на диск.

> ***Внимание!***
>
> Повторим еще раз — ни в коем случае не записывайте файлы на сам восстанавливаемый том!

Единственная проблема заключается в определении оригинальной длины. Некоторые типы файлов допускают присутствие "мусора" в своем хвосте. В этом случае можно следовать правилу, гласящему, что перебор лучше недобора. Однако это справедливо не для всех файлов! Если конец файла не удается определить визуально (например, pdf-файлы завершаются сигнатурой %%EOF), проанализируйте заголовок файла. Как правило, наряду с прочей полезной информацией там присутствует и размер файла. В данном случае все зависит от структуры конкретного файла, и универсальных рекомендаций дать невозможно.

Если восстанавливаемый файл фрагментирован, то ситуация осложняется. По правде говоря, она практически безнадежна, поскольку, чтобы собрать разрозненные цепочки кластеров воедино, необходимо хорошо знать содержимое удаленного файла. В этом смысле NTFS восстанавливается намного хуже, чем FAT. Последовательность фрагментов файла, хранящаяся в FAT в виде однонаправленного списка, очень живуча. Если список не поврежден, достаточно лишь найти его первый элемент (а сделать это проще простого, поскольку он будет указывать на заголовок файла с вполне предсказуемым содержимым). Даже если список "разрубить" на несколько частей, они продолжат жить собственной жизнью, и останется лишь подобрать комбинацию, в которой их необходимо склеить воедино. Список гибнет лишь при полном затирании FAT, что случается, прямо скажем, не часто. В NTFS же порядок фрагментов файла хранится в крохотных списках отрезков, и их гибель — обычное дело, после чего мы остаемся один на один с миллионом беспорядочно разбросанных кластеров. С текстовыми файлами еще можно работать, но что делать, если файл представлял собой электронную таблицу, графическое изображение или архив? Без знания стратегии выделения дискового пространства восстановить такой файл невозможно. Порядок, в котором драйвер файловой системы находит подходящие свободные фрагменты, не предопределен. Он варьируется в зависимости от множества обстоятельств, однако некоторые закономерности в нем все же присутствуют.

В ходе анализа списков отрезков сильно фрагментированных дисков нам удалось установить следующие закономерности. Сначала заполняются самые большие "дыры", причем заполнение происходит в направлении от конца зоны MFT к концу диска. Затем драйвер файловой системы возвращается назад и начинает заполнять "дыры" поменьше. Так продолжается до тех пор, пока файл не оказывается на диске целиком. В последнюю очередь заполняются "дыры" размером в один кластер. Просматривая карту диска, представленную файлом `/$BITMAP`, мы можем в точности восстановить порядок размещения фрагментов удаленного файла, наскоро собрав их воедино. Во всяком случае, теоретически такая возможность существует. На практике же на этом пути нас ждут коварные препятствия. Дело в том, что с момента создания восстанавливаемого файла карта свободного дискового пространства могла радикально измениться. Всякая операция удаления файлов высвобождает одну или несколько "дыр", хаотично перемешивающихся с "дырами" восстанавливаемого файла. Как этому противостоять? Сканируем MFT в поисках записей, помеченных как удаленные, но еще не затертых. Декодируем списки отрезков и вычеркиваем соответствующие им фрагменты из списка кандидатов на восстановление. Это существенно сужает круг поиска, хотя количество комбинаций, в которые можно собрать фрагментированный файл, по-прежнему остается велико. Но это не самое главное.

Самое "интересное" начинается, когда на диск одновременно записываются несколько файлов (например, скачиваемых из Интернета) или когда размер некоего файла постепенно увеличивается (это происходит с документами Word при наборе текста) и одновременно с этим на диск записываются другие файлы. Когда к существующему файлу дописывается крошечная порция данных, файловая система на-

ходит наименьшую "дыру", затем следующую наименьшую "дыру" и т. д., вплоть до тех пор, пока маленькие "дыры" не исчерпаются. Когда это происходит, наступает черед "дыр" большего размера. В результате файл сильно фрагментируется. Кроме того, файл заполняется не от больших "дыр" к меньшим, а наоборот (т. е. происходит инверсия стратегии размещения). Таким образом, маленькие фрагменты одного файла перемешиваются с маленькими фрагментами других файлов.

Хуже всего поддаются восстановлению документы, созданные в Microsoft Office. Так происходит, потому что приложение создает большое количество резервных копий редактируемого файла как в текущем каталоге, так и в каталоге `%TEMP%`. Разобраться с тем, какой фрагмент какому файлу принадлежит, очень нелегко!

Проще всего восстанавливать ZIP-архивы. Для этого вам даже не потребуется запускать дисковый редактор. Откройте временный файл на запись, сделайте `seek` на размер свободного дискового пространства, закройте файл. А теперь обработайте его утилитой **pkzipfix.exe** (или воспользуйтесь утилитой ZIP Repair (**http://www.ziprepair.com**)). В "исправленном" файле волшебным образом появятся все уцелевшие ZIP-архивы! Внутренняя структура ZIP-архива такова, что pkzipfix легко распознает даже переупорядоченные блоки, поэтому высокая степень фрагментации ему не помеха.

Дефрагментация тоже происходит интересно. Стандартный API дефрагментации в силу малопонятных ограничений оперирует не единичными кластерами, а блоками! Минимальный размер блока составляет 16 кластеров, причем начало блока должно быть кратно 16 кластерам в файле! Количество мелких "дыр" после дефрагментации только возрастает, а непрерывных областей свободного пространства практически совсем не остается.

Не забывайте, что перемещать что-либо внутрь зоны MFT тоже нельзя. Наверняка вы знакомы с системным сообщением примерно следующего вида:

```
На томе C: свободно 17%, но только 5% доступно для использования
дефрагментатора диска. Для эффективной работы дефрагментатор требует
по крайней мере 15% доступного свободного места.
```

"Недоступное" для дефрагментатора пространство находится внутри зоны MFT, потому что, как вы помните, при форматировании диска для хранения файла `$MFT` резервируется 12,5% от емкости тома. Затем, по мере исчерпания дискового пространства, файл `$MFT` усекается наполовину, а освободившееся за счет этого дисковое пространство отдается для хранения пользовательских файлов. Иначе говоря, для гарантированной работы дефрагментатора ему нужно `10% + 15% = 25%` свободного дискового пространства. Не слишком ли высока плата за дефрагментацию? Если же у вас свободно свыше 25%, настоятельно рекомендуется создать на диске временный файл и выполнить `seek`, чтобы заполнить все более или менее крупные "дыры", что не позволит дефрагментатору их изуродовать. Разумеется, после дефрагментации этот файл нужно удалить. К счастью, на сжатые файлы ограничение на минимальный размер блока в 16 кластеров не распространяется, поэтому мелкие файлы очень выгодно держать в сжатом состоянии, т. к. это существенно уменьшает фрагментацию тома. Чаще дефрагментируйте свой диск! Это не только увеличит

быстродействие, но и упростит восстановление удаленных файлов с затертой файловой записью.

Методики изучения механизма фрагментации

Существуют по меньшей мере две методики исследования стратегии выделения дискового пространства: *статическая* и *динамическая*. При использовании статической методики мы просто запускаем дисковый редактор (предпочтение следует отдать DiskExplorer от Runtime Software) и анализируем списки отрезков уже существующих файлов, записанных в различное время и различными способами. Например, можно скопировать файл из одного каталога в другой или попеременно увеличивать размер нескольких файлов — стратегии выделения свободного пространства в обоих случаях будут различны (рис. 8.2). Статический подход полезен тем, что дает бесценный статистический результат для всего тома в целом. Однако этот метод позволяет определить лишь окончательный результат, но не путь, которым этот результат был достигнут.

Утилита Diskmon.exe, разработанная Марком Руссиновичем (Mark Russinovich) и доступная для свободного скачивания на сайте Microsoft (**https://docs.microsoft.com/en-us/sysinternals/downloads/diskmon**), позволяет заглянуть в "святая святых" фай-

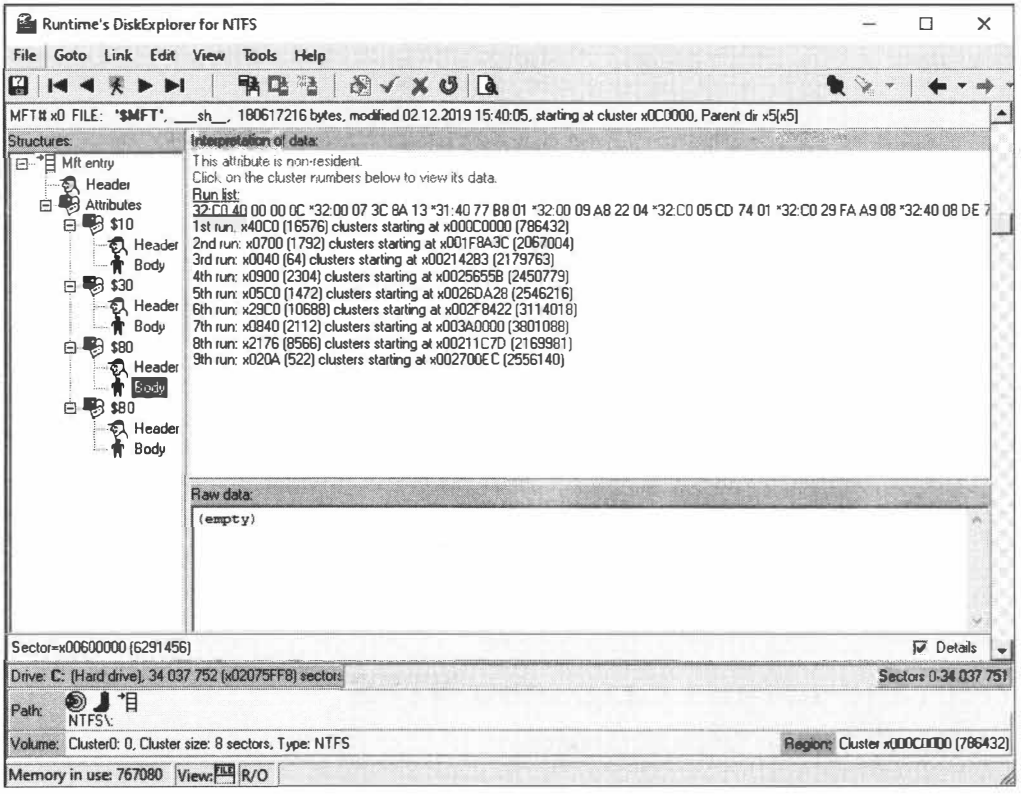
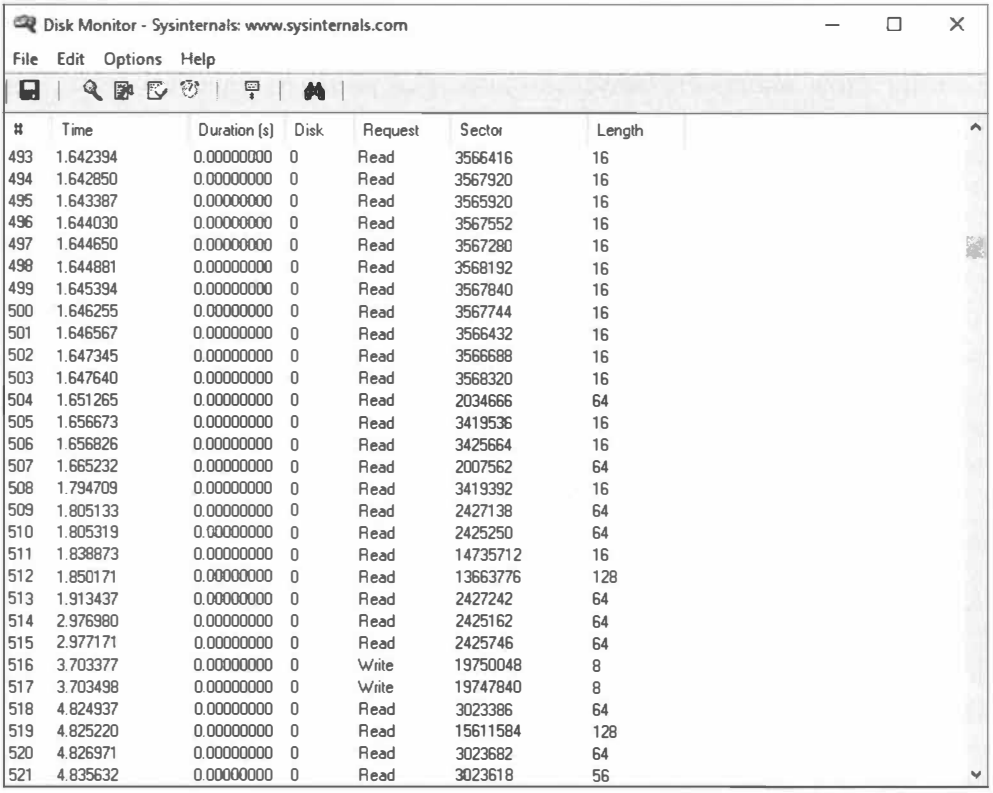


Рис. 8.2. Статический анализ стратегии выделения дискового пространства, выполняемый при помощи DiskExplorer от Runtime Software

ловой системы и увидеть, как именно она выделяет дисковое пространство для хранения файлов (рис. 8.3). Особенно интересно запускать Diskmon.exe параллельно с дефрагментатором или утилитой chkdsk, т. к. в этом случае все тайное сразу становится явным.



#	Time	Duration (s)	Disk	Request	Sector	Length
493	1.642394	0.00000000	0	Read	3566416	16
494	1.642850	0.00000000	0	Read	3567920	16
495	1.643387	0.00000000	0	Read	3565920	16
496	1.644030	0.00000000	0	Read	3567552	16
497	1.644650	0.00000000	0	Read	3567280	16
498	1.644881	0.00000000	0	Read	3568192	16
499	1.645394	0.00000000	0	Read	3567840	16
500	1.646255	0.00000000	0	Read	3567744	16
501	1.646567	0.00000000	0	Read	3566432	16
502	1.647345	0.00000000	0	Read	3566688	16
503	1.647640	0.00000000	0	Read	3568320	16
504	1.651265	0.00000000	0	Read	2034666	64
505	1.656673	0.00000000	0	Read	3419536	16
506	1.656826	0.00000000	0	Read	3425664	16
507	1.665232	0.00000000	0	Read	2007562	64
508	1.794709	0.00000000	0	Read	3419392	16
509	1.805133	0.00000000	0	Read	2427138	64
510	1.805319	0.00000000	0	Read	2425250	64
511	1.838873	0.00000000	0	Read	14735712	16
512	1.850171	0.00000000	0	Read	13663776	128
513	1.913437	0.00000000	0	Read	2427242	64
514	2.976980	0.00000000	0	Read	2425162	64
515	2.977171	0.00000000	0	Read	2425746	64
516	3.703377	0.00000000	0	Write	19750048	8
517	3.703498	0.00000000	0	Write	19747840	8
518	4.824937	0.00000000	0	Read	3023386	64
519	4.825220	0.00000000	0	Read	15611584	128
520	4.826971	0.00000000	0	Read	3023682	64
521	4.835632	0.00000000	0	Read	3023618	56

Рис. 8.3. Динамический анализ стратегии выделения дискового пространства, выполняемый при помощи Diskmon

Совет

Если, несмотря на все усилия, восстановить удаленный файл так и не удается, попробуйте отыскать его резервную копию. Многие приложения создают такие копии, но не все афишируют их присутствие. Не стоит забывать и о файле подкачки, временных файлах, дампе памяти и других источниках, которые могут хранить фрагменты восстанавливаемого файла. Если вам повезет, можно даже найти там весь файл целиком. Даже если эти источники информации тоже уже были удалены, возможно, принадлежащие им файловые записи еще не затерты, и тогда восстановление не займет много времени.

Восстановление разделов NTFS после форматирования

Представьте себе, что случилось самое страшное: вы потеряли весь раздел NTFS целиком. Возможно, вы случайно отформатировали его или пережили разруши-

тельный дисковый сбой. Где-то там остались миллиарды байтов бесценных данных, теперь уже недоступных операционной системе. Как вернуть информацию к жизни? До сих пор мы рассматривали лишь незначительные дисковые сбои и легкие разрушения данных наподобие ошибочно удаленных файлов. Теперь настал черед рассмотреть восстановление после тяжелых повреждений, при которых прежнее содержимое тома становится недоступно операционной системе. Причиной этого может быть, например, непредумышленное форматирование или искажение главной файловой таблицы. Но не падайте духом! Из любых переделок NTFS выходит с минимальными потерями, и во всех этих случаях возможно полное восстановление данных, если к делу подойти правильно.

Проще всего начать с форматирования. Для экспериментов нам потребуется утилита format.com, входящая в стандартный комплект поставки Windows, а также дисковый раздел, не содержащий никакой ценной информации.

Совет

Лучше всего проводить эксперименты на виртуальной машине — VirtualBox или VMWare, эмулирующей жесткий диск и ускоряющей процедуру форматирования в сотни раз (рис. 8.4). Ведь время — это не только деньги, но и бесцельно прожитые годы, проведенные за созерцанием индикатора прогресса.

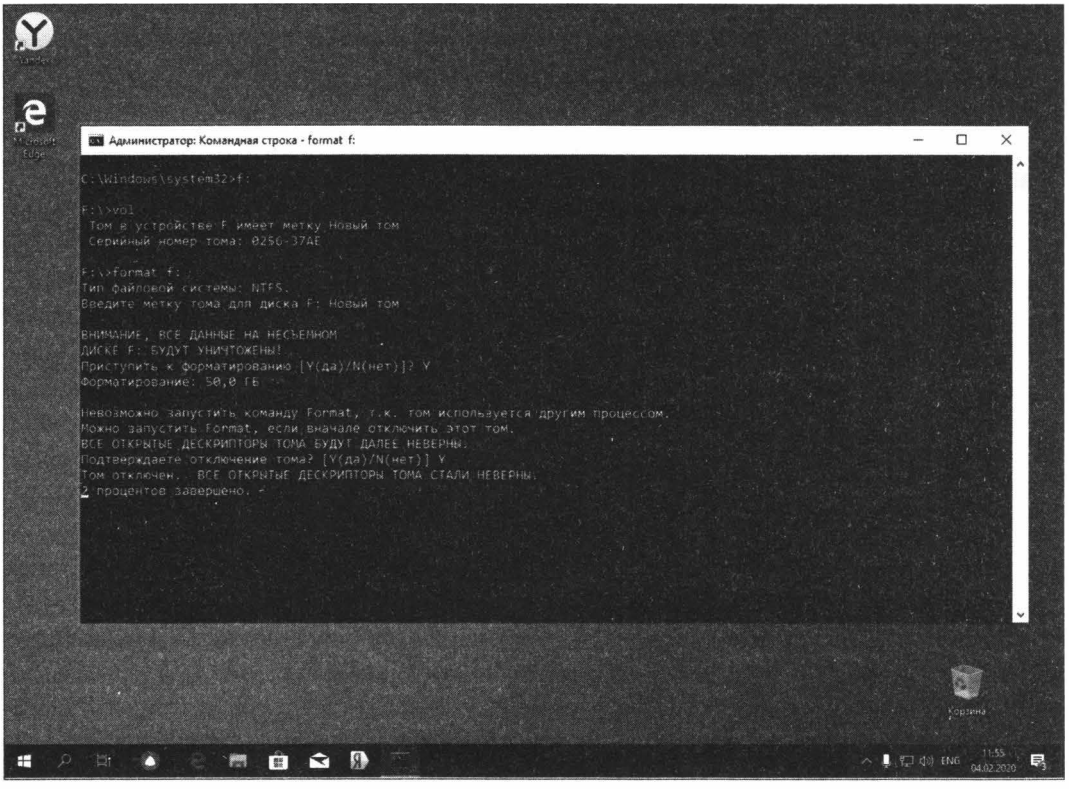


Рис. 8.4. Форматирование виртуального диска в среде VirtualBox

Примечание

Команда `format` в Windows 10 работает с одной интересной особенностью, которую вполне можно назвать багом: она не позволяет отформатировать том, если пользователь не указал его метку, выдавая ошибку: "*указана недопустимая метка диска*". Чтобы исправить эту ошибку, сделайте следующее. Перейдите в Командной строке на диск, который вы хотите отформатировать, и наберите команду `vol`. В окне Командной строки отобразится информация об имени метки и серийном номере тома. Затем, набрав команду `format` с именем диска, по запросу программы format.exe введите полученную на предыдущем шаге метку тома. Если том в настоящее время используется, согласитесь с предложением операционной системы размонтировать его.

"Живой" винчестер лучше не трогать, во всяком случае, до тех пор, пока вы не научитесь его восстанавливать.

Действия, выполняемые при форматировании

Форматирование диска — это сложная многоступенчатая операция, намного более сложная и намного более многоступенчатая, чем это может показаться на первый взгляд. Свои исследования мы начнем с изучения тома NTFS, непредумышленно переформатированного под NTFS. Техника восстановления томов NTFS, переформатированных под FAT32, будет рассмотрена отдельно.

При выполнении команды `format X: /U /FS:NTFS` в файловой системе диска `X:` происходят следующие изменения (форматирование диска утилитой GUI, вызываемой из контекстного меню "проводника", осуществляется по аналогичной схеме):

1. Формируется загрузочный сектор NTFS.
2. Генерируется новый серийный номер тома, который затем записывается в загрузочный сектор по смещению `48h` байтов от его начала.
3. Рассчитывается новая контрольная сумма загрузочного сектора, которая затем записывается по смещению `50h` от его начала (более подробная информация была приведена в *главе 6*).
4. Создается новый файл `$MFT`, содержащий сведения обо всех файлах на диске. Как правило, он записывается поверх старого файла `$MFT`. Исключения из этого правила бывают, но они крайне редки. Обычно они происходят, если прежний файл `$MFT` был заблаговременно перемещен дефрагментатором или при переформатировании был назначен новый размер кластера. Во всех остальных случаях первые 24 файловые записи (FILE Record) погибают безвозвратно. Эти записи содержат непосредственно сам файл `$MFT`, `$MFTMirr`, корневой каталог, `/$LogFile` — файл транзакций, `/$BITMAP` — карту свободного пространства, `/$Secure` — дескрипторы безопасности, а также ряд других служебных файлов.
5. Инициализируется файл `$MFT:$DATA` — назначаются новая длина файла (инициализируются `$MFT:$30.AllocatedSize`, `$MFT:$30.RealSize`, `$MFT:$80.AllocatedSize`, `$MFT:$80.RealSize`, `$MFT:$80.CompressionSize`, `$MFT:$80.InitializedSize` и `$MFT:$80.LastVCN`), дата и время создания и последней модификации (инициализируются `$MFT:$10.FileCreationTime`, `$MFT:$10.FileAlertedTime`, `$MFT:$10.FileReadTime`, `$MFT:$30.FileCreationTime`, `$MFT:$30.FileAlertedTime`, `$MFT:$30.MFTChangeTime`

и `$MFT:$30.FileReadTime`) и, самое главное, создается новый список отрезков (data-runs), необратимо затирающий старый. Это значит, что собирать фрагментированный файл `$MFT` нам придется по частям.

6. Создается новый файл `/$MFT:$BITMAP`, отвечающий за занятость файловых записей в MFT. При этом все старые записи помечаются как свободные, однако их фактического удаления не происходит (поле `FileRecord.flags` остается нетронутым), благодаря чему процедура восстановления заметно упрощается. Чаще всего `$MFT:$BITMAP` располагается на том же самом месте, что и старый (т. е. между загрузочным сектором и MFT), забивая прежнее содержимое нулями, однако с помощью утилиты chkdsk его можно восстановить.
7. Создается новый файл `/$BITMAP`, отвечающий за распределение дискового пространства (свободные и занятые кластеры). Этот файл также записывается поверх прежнего файла `/$BITMAP`, который тем не менее может быть восстановлен с помощью chkdsk.
8. Создается новый файл журнала транзакций — `/$LogFile`.
9. В заголовок файловой записи `$MFT` заносится новый LSN (LogFile Sequence Number).
10. `$MFT` назначается новый номер последовательности обновления (Update Sequence Number).
11. Создается новое зеркало `$MFTMirr`, необратимо затирающее старое (в текущих версиях файловых систем оно расположено в середине раздела NTFS).
12. Создаются новые `/$Volume`, `/$AttrDef` и другие служебные файлы, играющие сугубо вспомогательную роль и легко восстанавливаемые утилитой chkdsk. Следует отметить, что хотя `/$Volume` и присутствует в зеркальной копии MFT, его ценность явно преувеличена.
13. Осуществляется проверка целостности поверхности диска, и все обнаруженные "плохие" кластеры заносятся в файл `/$BadClus`.
14. Формируется новый корневой каталог.
15. Если до форматирования тома на нем присутствовал файл `/System Volume Information`, то он обновляется, в противном случае новый файл `/System Volume Information` создается только после перезагрузки.

На самом деле процесс форматирования протекает намного сложнее. Тем не менее для восстановления данных с непреднамеренно переформатированных разделов приведенной здесь информации вполне достаточно. Углубленное обсуждение этих технических деталей требуется только программисту, разрабатывающему собственную нестандартную утилиту форматирования. Заинтересованные читатели могут самостоятельно дизассемблировать утилиту format.com (рекомендуется делать это с помощью IDA Pro).

> ***Совет***
>
> Утилита format.com содержит лишь высокоуровневую надстройку, опирающуюся на библиотеки ifsutil.dll, untfs.dll и непосредственно на сам драйвер файловой системы. Так что

дизассемблировать придется много. Чтобы упросить себе работу, можно пронаблюдать за процессом форматирования с помощью "шпионских" средств, например утилит Марка Руссиновича Process Monitor (procmon.exe), Diskmon.exe, бесплатные копии которых можно скачать с сайта **https://docs.microsoft.com/en-us/sysinternals/**. Кроме того, не забывайте о точках останова на основные функции native API, такие как `NtFsControlFile`, `NtDeviceIoControlFile` и т. д.

Автоматическое восстановление диска после форматирования

Форматирование не уничтожает файловые записи пользовательских файлов, и их можно полностью восстановить. Существует множество утилит для восстановления данных, например R-Studio, EasyRecovery, GetDataBack и т. д. Тем не менее прямых наследников утилиты unformat среди них не наблюдается. Утилита unformat.exe восстанавливала весь том целиком, а все перечисленные современные средства всего лишь извлекают отдельные уцелевшие файлы и каталоги, переписывая их на новый носитель. Вот здесь мы сталкиваемся с рядом проблем. Во-первых, это выбор носителя, на который будут извлекаться восстанавливаемые данные. Запись на оптические накопители отпадает сразу же, т. к. количество носителей, потребное для сохранения содержимого жесткого диска объемом в терабайты, неприемлемо велико, да и сама технология пишущих оптических приводов сегодня фактически ушла в прошлое. Наконец, ни одна из известных мне утилит автоматического восстановления данных не позволяет "разрезать" большие файлы на несколько маленьких. Если в вашем распоряжении есть локальная сеть, можно перегнать данные по ней. Однако самый простой вариант — подключение внешнего жесткого диска, карты памяти или флеш-накопителя. Для извлечения пары сотен особо ценных файлов такая методика вполне подходит.

Продемонстрируем технику автоматического восстановления данных на примере утилиты R-Studio от компании R-Tools Technology (**https://www.r-studio.com**). Это довольно мощный и в то же время простой в управлении инструмент, на который можно положиться. После запуска утилиты на экране появится окно **Панель дисков**, где перечислены все физические устройства и логические разделы. Найдите среди них тот, который требуется восстановить, и, нажав правую кнопку мыши, выберите опцию **Сканировать**.

Программа предложит указать начальный сектор для сканирования (поле **Старт**), который по умолчанию равен нулю. Это значение следует оставить без изменений. Размер сканируемой области (поле **Размер**) по умолчанию развертывается на весь раздел. Это гарантирует, что сканер обнаружит все уцелевшие файловые записи, хотя сам поиск займет значительное время. Можно ли ускорить этот процесс? Давайте возьмем ручку и подсчитаем. Предположим, что восстанавливаемый раздел содержит сто тысяч файлов. Типичный размер файловой записи составляет 1 Кбайт. При условии, что файл `$MFT` не фрагментирован, достаточно просканировать всего около 100 Мбайт от начала раздела. Если эта величина (размер пространства, зарезервированного под MFT) не превышает 10% от полной емкости тома и диск никогда не заполнялся более чем на 90%, то, скорее всего, все так и есть.

В противном случае файл $MFT фрагментирован и "живописно" разбросан по всему диску. Впрочем, в случае ошибки мы ничем не рискуем. Вводим значение N Кбайт, где N — предполагаемое количество файлов (каталог также считается файлом), и выполняем сканирование. Если один или несколько файлов останутся необнаруженными, возвращаемся к настройкам по умолчанию и повторяем процедуру сканирования вновь (если количество имеющихся файлов заранее неизвестно, следует указать значение, равное 10% от емкости тома). В поле **Файловая система** выбираем файловую систему NTFS (можно оставить настройки по умолчанию, файловую систему диска программа определяет корректно). Затем нажмите кнопку **Сканирование**, и сканирование начнется (рис. 8.5).

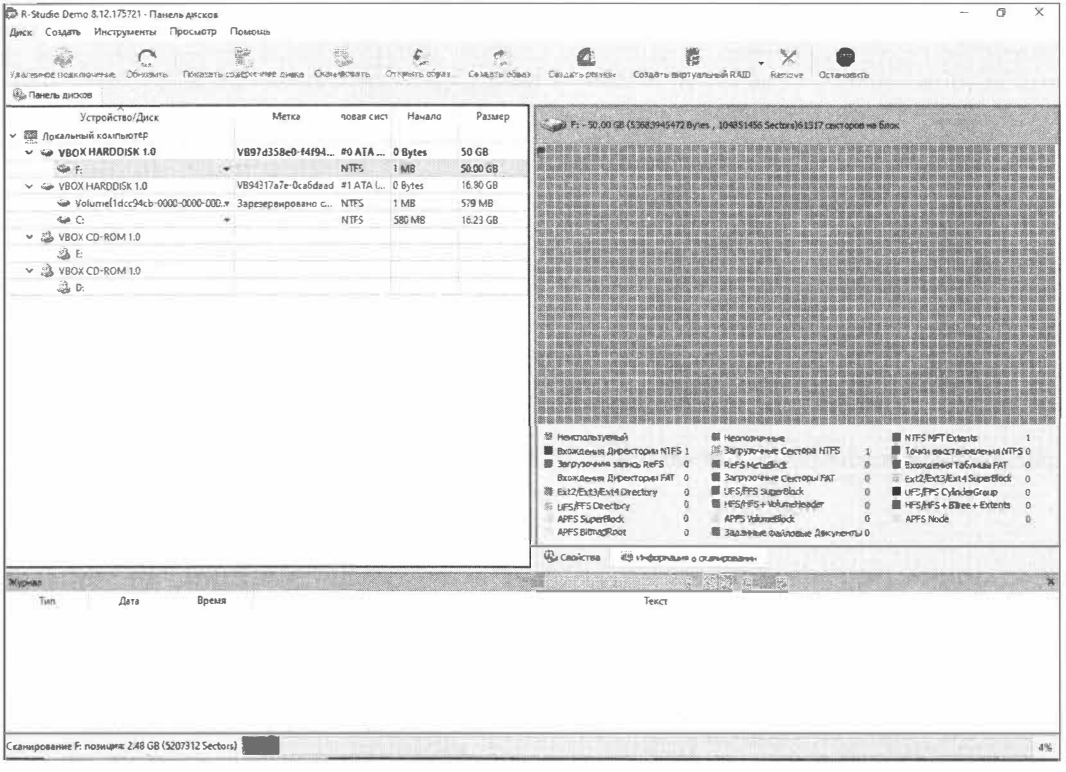


Рис. 8.5. R-Studio осуществляет поиск уцелевших файловых записей

В процессе сканирования будут найдены все уцелевшие файлы (как удаленные, так и нет). Кроме того, будет восстановлена структура каталогов, включая и корневой каталог (рис. 8.6). Постойте! Как же так? Ведь, как вы помните, при форматировании корневой каталог был уничтожен и сформирован заново! Но ничего удивительного в этом нет. Просто файловая система NTFS еще раз доказала свою живучесть — уничтожить ее можно, скорее всего, только динамитом. В отличие от FAT, в NTFS каталоги являются лишь вспомогательной структурой данных, проиндексированной для ускорения отображения их содержимого. Всякая файловая запись, независимо от своего происхождения, содержит ссылку на родительский каталог,

представляющую собой номер записи в MFT. А запись корневого каталога всегда располагается по одному и тому же адресу!

Удаленные файловые записи могут ссылаться на уже уничтоженные каталоги. R-Studio собирает их в папку \$\$\$Folder*XXX*, где *XXX* — порядковый номер каталога. Поэтому иерархия подкаталогов в большинстве случаев успешно восстанавливается.

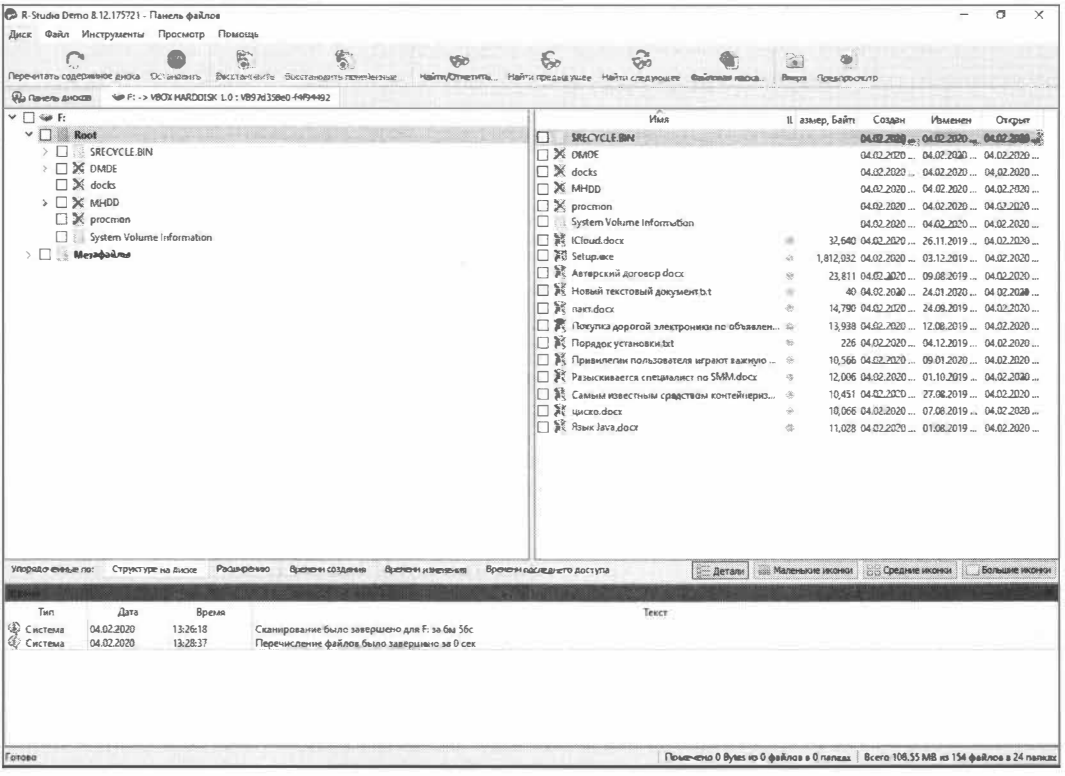


Рис. 8.6. Восстановленная структура каталогов

Просмотр виртуального дерева обнаруженных файлов осуществляется нажатием кнопки <F5> или с помощью соответствующей команды контекстного меню. Выбрав файл (или даже целый каталог с большим количеством вложенных подкаталогов), нажмите клавишу <F2>. При желании можно выполнить предварительный просмотр или редактирование (выбрав из контекстного меню пункт **Показать содержимое диска**). Это достаточно мощный инструмент, отображающий содержимое восстанавливаемого файла со всеми его атрибутами, списками отрезков и т. д. в удобочитаемом формате. При желании можно восстановить все файлы за одну операцию (**Восстановить все**) или выбрать восстановление по маске. Хваленая утилита EasyRecovery от Ontrack (рис. 8.7), вопреки своему названию, простотой управления отнюдь не отличается и имеет довольно специфические особенности поведения. С настройками по умолчанию всех файлов на отформатированном разделе эта утилита не увидит. Чтобы заставить ее работать, необходимо приобрести

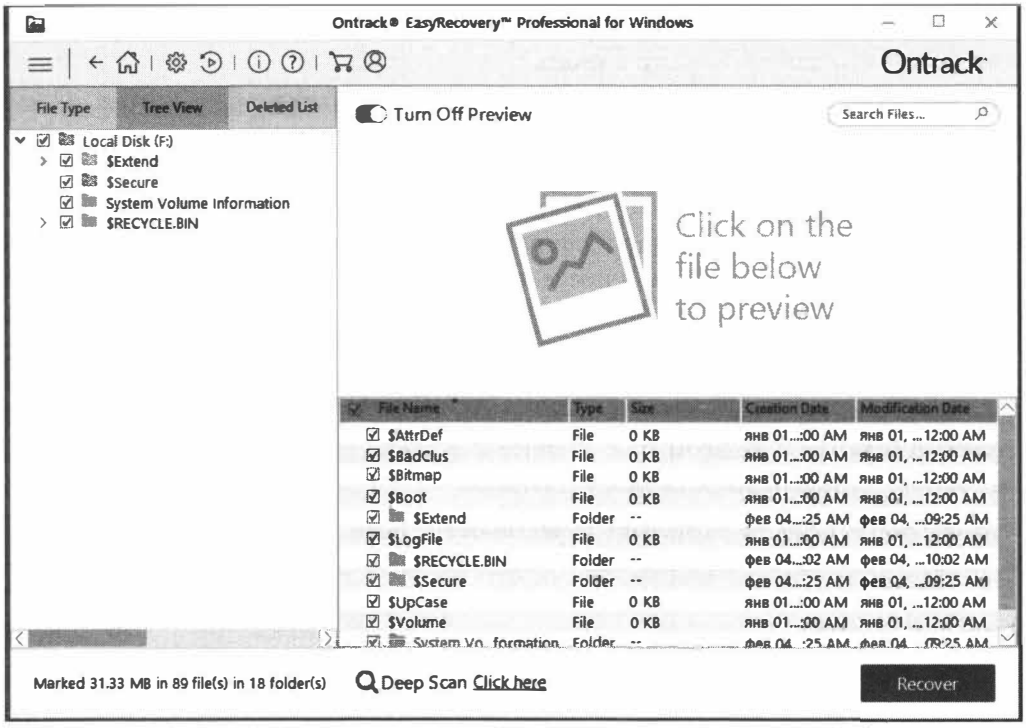


Рис. 8.7. Красивый интерфейс EasyRecovery еще не говорит о высоком качестве восстановления данных

коммерческую версию приложения. Однако и в этом случае качество восстановления будет оставлять желать лучшего.

Ручное восстановление жесткого диска после форматирования

Нашей целью будет ручное восстановление всего отформатированного раздела без использования дополнительных носителей информации и дорогостоящих утилит от сторонних производителей. Все, что для этого потребуется, — любой редактор диска (предпочтительнее всего, конечно же, DiskExplorer от Runtime Software) в комбинации со встроенной утилитой chkdsk.

В процессе форматирования происходит необратимое разрушение большого количества ключевых структур данных, восстанавливать которые вручную было бы слишком затруднительно. К счастью, для извлечения особо ценных файлов этого и не требуется! Идея состоит в том, чтобы вернуть разделу потерянные файловые записи, а все остальные восстановительные операции поручить утилите chkdsk.

Дизассемблирование показывает, что единственной структурой данных, без которой не может работать chkdsk, является атрибут `$DATA` файла `$MFT`. А раз так, все, что требуется сделать, сводится к воссозданию прежнего файла `$MFT:$DATA` и его размещению поверх старых файловых записей. В простейшем случае, если файл

`$MFT:$DATA` не фрагментирован, это достигается так называемым спекулятивным увеличением его длины. Как это сделать?

Запускаем DiskExplorer, переходим в начало MFT (**Goto | Mft**), выделяем файл `$MFT`, находим атрибут `$DATA` (`80h`) и увеличиваем поля `Allocated Size`, `Real Size` и `Compressed Size` на требуемую величину, параллельно с этим корректируя список отрезков (рис. 8.8). Поле `Last VCN` трогать не нужно, т. к. оно будет исправлено утилитой chkdsk. Как определить длину нефрагментированного файла MFT? Она равна разнице номеров первого и последнего секторов, в начале которых присутствует сигнатура `FILE`, умноженной на 512 байтов (исключая сектора, принадлежащие `$MFTMirr`). Известные нам дисковые редакторы не поддерживают поиска последнего вхождения, поэтому соответствующую утилиту приходится писать самостоятельно. К счастью, точную длину MFT определять совершенно не обязательно, и вполне допустимо взять ее с запасом, т. к. лишнее все равно отсеет chkdsk. Действуйте по принципу — лучше перебрать, чем недобрать.

Утилита DiskExplorer не позволяет редактировать поля в естественном режиме отображения, заставляя нас переключаться в HEX-mode и искать смещения всех значений

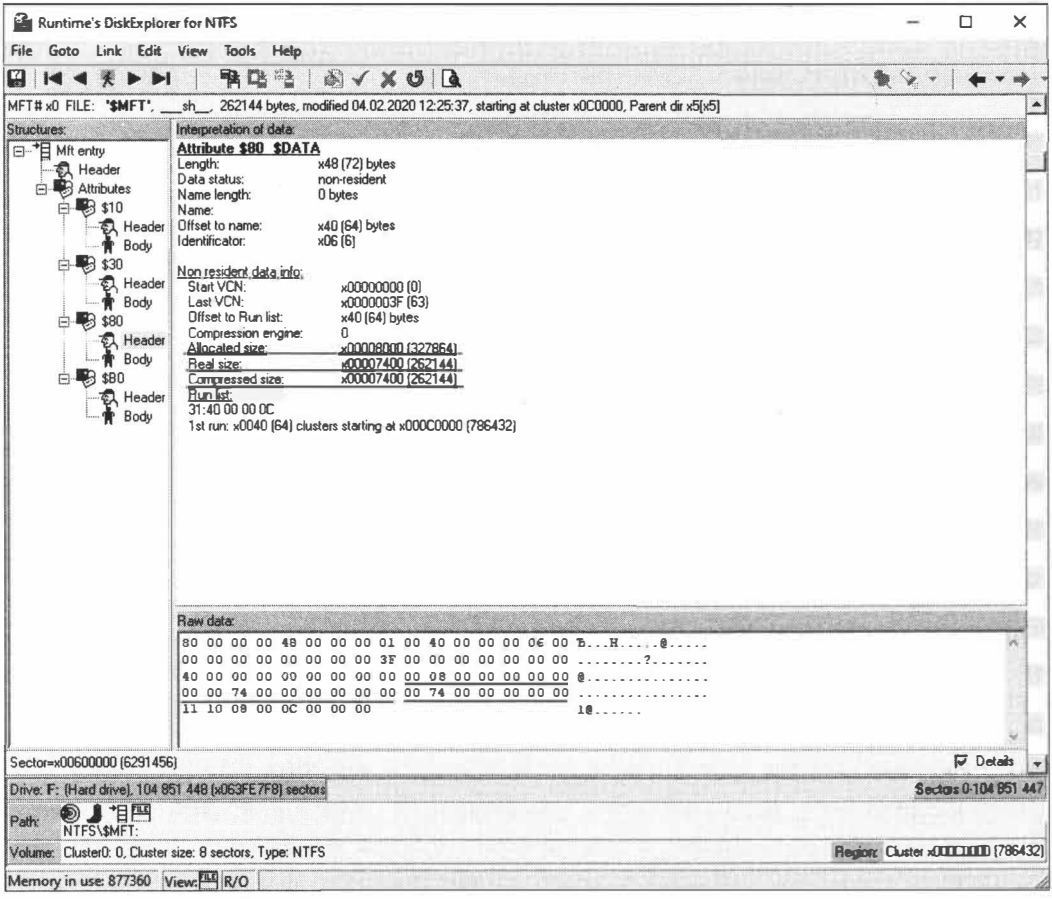


Рис. 8.8. Ручное восстановление MFT. Подчеркнуты поля, подлежащие изменению

самостоятельно. Найти заголовок атрибута $DATA очень просто — в его начале расположена последовательность 80 00 00 00 xx 00 00 00 01. Поле Real Size во всех версиях NTFS располагается по смещению 30h относительно заголовка, а поля Allocated Size и Initialized Size соответственно по смещениям 28h и 38h байтов, причем значение Allocated Size должно быть кратно размеру кластера. Убедитесь, что при переформатировании диска размер кластера не изменился, в противном случае у вас ничего не получится. Как восстановить исходный размер кластера? Да очень просто — набраться мужества и переформатировать восстанавливаемый диск с ключом /A:x, где x — размер кластера. А как его определить? Возьмем любой файл с известным содержимым и проанализируем его список отрезков. Запускаем контекстный поиск по всему диску, находим файл, запоминаем (записываем на бумажке) его стартовый сектор, после чего открываем закрепленную за ним файловую запись, декодируем список отрезков и вычисляем номер первого кластера. Делим номер сектора на номер кластера и получаем искомую величину.

Теперь необходимо сгенерировать новый список отрезков. В общем виде он будет выглядеть так: 13 XX XX XX YY 00, где XX XX XX — трехбайтовое значение размера $MFT в кластерах, а YY — стартовый кластер. Стартовый кластер обязательно должен указывать на первый кластер MFT, в противном случае chkdsk не сможет работать. Если новый список отрезков длиннее нынешнего (скорее всего, именно так и будет), то необходимо скорректировать длину атрибутного заголовка (она расположена по смещению 04h от его начала). Проделав эту нехитрую операцию, запустим chkdsk с ключом /F и блаженно откинемся на спинку кресла, созерцая, как возрождаются наши милые папки и файлы. Единственное, что не восстанавливается, — так это дескрипторы безопасности. Всем файлам и папкам будут назначены права доступа по умолчанию. Во всех остальных отношениях с отремонтированным таким образом диском вполне можно будет работать, не опасаясь, что он рухнет окончательно. Файлы, ссылающиеся на несуществующие каталоги, складываются в папку Found.xxx. Это — "долгожители", пережившие несколько циклов переформатирования, в буквальном смысле вытащенные из небытия.

Сложнее восстановить том, MFT которого сильно фрагментирована. Прежний список отрезков при форматировании был уничтожен, зеркальная копия также пострадала. Ничего другого не остается, как собирать все фрагменты руками. К счастью, на практике это оказывается не так сложно, как может показаться на первый взгляд. В отличие от всех остальных файлов диска, файл $MFT имеет замечательную сигнатуру FILE, присутствующую в начале каждой файловой записи. Поэтому все, что нам требуется сделать, сводится к следующим операциям. Последовательно сканируя раздел от первого кластера до последнего, выпишите начало и конец каждого из фрагментов, принадлежащих MFT. Затем из этой цепочки необходимо исключить файл $MFTMirr. Его легко узнать, т. к. он расположен в середине раздела и содержит копии файловых записей $MFT, $MFTMirr, $LogFile и $Volume, причем $MFTMirr ссылается на себя. В рассматриваемом примере наш список выглядит так: 08h — 333h, 669h — 966h, 1013 — 3210h. В грубом приближении ему будет соответствовать следующий список отрезков: 12 2B 03 08 22 23 03 69 96 22 FD 21 13 10 00.

"В грубом приближении" сказано потому, что мы не знаем, в какой последовательности располагались эти отрезки в файле (порядок расположения фрагментов на диске далеко не всегда совпадает с порядком отрезков в списке отрезков). Что произойдет, если порядок сборки файла `$MFT` окажется нарушен? Внутри MFT все файловые записи ссылаются друг на друга по своим порядковым номерам, представляющим собой индексы массива. Эти ссылки необходимы для восстановления структуры каталогов, организации жестких ссылок (hard links) и еще некоторых служебных структур. Ссылки на родительский каталог дублируются в индексах и восстанавливаются элементарно. Жесткие ссылки теряются безвозвратно (единственный способ восстановить их заключается в повторении попытки сборки файла `$MFT` в другом порядке). Однако они практически нигде и никак не используются, так что их потеря не столь уж существенна. По-настоящему туго приходится сильно фрагментированным файлам, занимающим несколько файловых записей, раскиданных по разным фрагментам `$MFT`. Здесь выручает только перестановка фрагментов. К счастью, количество комбинаций обычно бывает невелико и процедура восстановления занимает совсем немного времени. Хорошая новость — начиная с NTFS версии 3.1, в MFT номера файловых записей хранятся в явном виде (четырехбайтовое поле, расположенное по смещению `2Ch` от начала файловой записи), что делает задачу восстановления тривиальной.

Восстановление после тяжелых повреждений

В результате сбоя содержимое дискового тома может стать недоступным операционной системе. При попытке чтения оглавления такого тома будут выдаваться различные сообщения об ошибках (рис. 8.9). Chkdsk сообщает о повреждении MFT и прекращает работу.

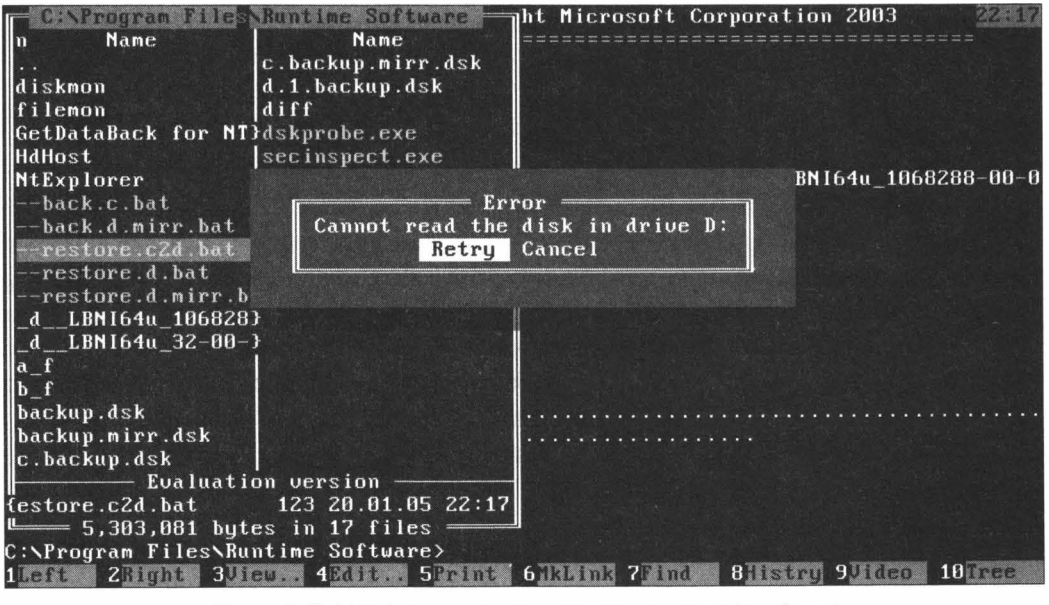


Рис. 8.9. Безуспешная попытка прочитать поврежденный том

Не паникуйте! Попробуйте запустить DiskExplorer и посмотрите, что он покажет. Маловероятно, чтобы содержимое всего тома было утеряно целиком. Если хотя бы часть файловых записей уцелела, то R-Studio, GetDataBack или EasyRecovery их обязательно восстановит!

Анализ показывает, что основной причиной, по которой chkdsk отказывается проверять том, обычно становится порча файловой записи, описывающей `$MFT`. Если в процессе обновления `$MFT` внезапно отключить питание, то такой исход вполне вероятен, особенно на жестких дисках с емким аппаратным кэшем. Такие диски не успевают завершить сохранение секторов, потребляя энергию, накопленную в конденсаторах, а вот их младшие собратья с этим справляются. То же самое происходит при неудачном перемещении файла `$MFT` или физическом разрушении первого сектора MFT. Зеркальная копия `$MFT` во всех этих случаях остается цела, однако chkdsk по каким-то таинственным причинам не хочет ею пользоваться, и вы должны восстановить ее самостоятельно. Просто скопируйте первый сектор `$MFTMirr` в первый сектор `$MFT`! Поклонники утилиты Sector Inspector (**https://www.microsoft.com/en-us/download/details.aspx?id=19470**) могут воспользоваться для этого командным файлом, приведенным в листинге 8.1.

Листинг 8.1. Командный файл для ручного восстановления $MFT из $MFTMirr, XXX — номер сектора $MFTMirr, YYY — номер сектора $MFT

```
SECINSPECT.EXE -backup     d:     backup.dsk     XXX     1
SECINSPECT.EXE -restore    d:     backup.dsk     YYY     CONFIRM
```

Теперь можно смело запускать chkdsk. Если же chkdsk по-прежнему не работает, это может происходить по одной из следующих причин.

- Повреждение загрузочного сектора. Решить проблему можно, восстановив загрузочный сектор, используя методику, рассмотренную в *главе 6*.
- Несовпадение списка отрезков файла `$MFT:$DATA` с истинным началом MFT. Чтобы решить эту проблему, найдите сектор с файловой записью `$MFT` и исправьте список отрезков. Вопросы, связанные с кодированием и декодированием списка отрезков, были изложены в *главе 7*, а методика исправления списка отрезков — ранее в этой главе.
- Несовпадение размера кластера, указанного в загрузочном секторе с фактическим размером кластера. Определение истинного размера кластера и методика восстановления были рассмотрены ранее в этой главе.

Если же сбой был настолько серьезен, что вместе с `$MFT` пострадало и зеркало, задача сводится к восстановлению отформатированного диска. При тяжелых разрушениях файловой структуры, когда на диске образуется настоящий кавардак, восстановление тома полезно начинать не с чего-нибудь, а именно с его форматирования. Нет, это не первоапрельская шутка! Утилита format.exe формирует заведомо исправные ключевые структуры, а подключение файловых записей — не проблема. Главное — сохраните список отрезков файла `$MFT:$DATA`, если, конечно, он еще уцелел. Все остальное — дело техники!

Восстановление тома NTFS после форматирования под FAT32

При переформатировании диска операционные системы семейства Windows NT (к которым относится и Windows 10) никогда не изменяют тип файловой системы, если только им не дать такое указание явно. Поэтому непреднамеренное переформатирование раздела NTFS под FAT16/32 крайне маловероятно. Однако большинство флеш-накопителей по умолчанию используют файловую систему FAT32. Если флешка была по каким-то причинам отформатирована в NTFS, на нее были записаны ценные данные, а потом в какой-то момент произошло ее повторное форматирование, велика вероятность, что тип файловой системы тоже изменился.

Стратегия восстановления данных во всем похожа на восстановление тома NTFS, случайно переформатированного под NTFS. Единственное отличие состоит в том, что при создании таблицы размещения файлов (file allocation table) первые несколько тысяч файловых записей в MFT затираются безвозвратно. Впрочем, даже их можно попытаться собрать вручную, действуя по методике, описанной ранее в данной главе.

Файлы с уцелевшей файловой записью легко восстанавливаются с помощью утилит R-Studio, GetDataBack или EasyRecovery. Для ручного восстановления всего тома его необходимо заново отформатировать под NTFS, предварительно определив количество секторов в кластере. Далее действуем по плану — увеличиваем размер `$MFT`, запускаем chkdsk и собираем все, что только можно собрать. Поскольку число файлов, хранящихся на современных дисках, зачастую исчисляется многими миллионами, потеря первой тысячи из них не так уж и страшна (если только по закону подлости они не окажутся самыми ценными из всего, что хранилось на диске).

Источники угрозы

Почему погибают дисковые разделы? Далее приводится список наиболее распространенных причин, отсортированный в порядке убывания их "популярности".

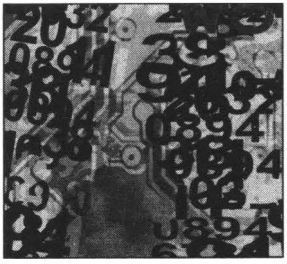
- Ошибки оператора, вирусы, троянские программы.
- Отключение питания или "зависание" системы во время интенсивных дисковых операций, сопровождаемых обновлением MFT (например, удаление/добавление файлов или каталогов).
- Некорректное поведение различных дисковых утилит.
- Физические дефекты оперативной памяти, приводящие к нарушению целостности дискового кэша и, следовательно, к порче самого диска.
- Некорректное поведение привилегированных драйверов, случайно или преднамеренно "залетающих" внутрь служебных структур драйвера NTFS.

Полезные советы

Чтобы предотвратить разрушение тома и упростить задачу восстановления данных, рекомендуется заблаговременно выполнить следующий комплекс мероприятий.

- Переместите `$MFT` подальше от начала раздела. Первые сектора раздела — это, как показала практика, самое небезопасное место. Во-первых, сюда стремятся попасть практически все вирусы. Невозможность прямого доступа к диску под управлением операционных систем из семейства Windows NT — это всего лишь миф. Если вам требуется аргументированное опровержение этого мифа, прочтите описание функции `CreateFile` и инструкцию к драйверу ASPI32. Во-вторых, если вы вдруг запустите какую-либо затирающую утилиту, первым погибнет именно `$MFT`, без которого весь дисковый том — это просто груда мусора. Можно привести и множество других, самых различных причин! Поэтому просто переместите `$MFT`, тем более что это совсем не сложно. Достаточно взять любой дефрагментатор, распространяющийся в исходных текстах, и слегка доработать его. Разумеется, резервная копия при этом всегда должна находиться под рукой.
- Не допускайте фрагментации файла `$MFT`! Не создавайте на диске огромного количества мелких файлов и не заполняйте его более чем на 90%. Стандартный дефрагментатор, входящий в комплект штатной поставки Windows, не позволяет дефрагментировать `$MFT`. Для этой цели приходится прибегать к сторонним средствам, лучшим из которых, на наш взгляд, является O&O Defrag от одноименной компании (**http://www.oo-software.com**). Это действительно профессиональный дефрагментатор.
- Периодически создавайте резервную копию файловой записи `$MFT`. Для этого достаточно сохранить один-единственный сектор — первый сектор MFT, номер которого содержится в загрузочном секторе. Только не забывайте его периодически обновлять, ведь при добавлении новых файлов и каталогов MFT планомерно расширяется и старые списки отрезков становятся все менее и менее актуальными.

ГЛАВА 9



Восстановление данных под Linux/BSD

Некогда главным недостатком UNIX-подобных систем было отсутствие нормальных драйверов под множество разнообразного оборудования, с которым Windows справляется без проблем. Полтора-два десятка 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 операционная система Linux еще не вышла из стадии "конструктора" для хакеров, а BSD в основном использовалась на серверах, все оборудование которых сводилось к сетевой карте и контроллеру SCSI. Поэтому основной массе пользователей жаловаться не приходилось.

На домашних и офисных компьютерах ситуация совсем иная. Тут и сканеры-принтеры, и мультимедиа, и, конечно же, видеокарты, издавна славящиеся отсутствием общих стандартов. 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. Прибрать соответствующий рыночный сегмент к рукам могут только очень крупные производители, такие как, например, nVIDIA, продающие миллионы видеокарт, среди которых и пара процентов становится вполне ощутимой величиной.

Иногда приходится слышать об армии энтузиастов, дизассемблирующих драйверы Windows и переписывающих их под Linux или BSD. Но, к сожалению, этих энтузиастов не так уж и много. Разнотипного оборудования гораздо больше, тем более что сплошь и рядом приходится сталкиваться с ситуацией, когда на компьютере энтузиаста драйвер работает, а на компьютере пользователя — нет, хотя аппаратные конфигурации этих компьютеров практически идентичны. Драйвер мало написать, его еще нужно отладить, протестировать на большом количестве конфигураций и сопровождать, иначе это будет не драйвер, а просто игрушка. Все это требует затрат времени, и если драйвер создается не для коммерческой выгоды, а для личных нужд, то его надежность и совместимость оставляют желать лучшего. Тем не менее бывает и так, что техника, официальные драйверы к которой последний раз в лучшем случае прикладывались на CD для Windows XP, совершенно без проблем

работает под Linux, в отличие от последних версий Windows, где без бубна точно не обойтись (или можно поставить виртуальную машину со старой ОС и дать ей прямой доступ к оборудованию через соответствующий порт, где это оборудование еще работало).

За последние годы ситуация с поддержкой разнообразного оборудования в UNIX-системах значительно улучшилась. То же самое касается поддержки различных стандартов, среди которых и "неродные" для UNIX файловые системы. Составители дистрибутивов проделывают огромную работу, собирая различные драйверы и включая их в свой комплект.

Виртуальные машины

Прежде чем ставить Linux/BSD, задумайтесь — а зачем вам, собственно, все это нужно? Если просто хотите протестировать альтернативную систему, освоить средства разработки или компилировать исходные тексты, то наилучшим выбором будет виртуальная машина, например, VirtualBox или VMware (рис. 9.1). Хочешь — разрабатывай программы, хочешь — читай руководства (man pages). Еще и в игры типа Star Wars можно поиграть. Никаких драйверов сверх того, что есть в любом "правильном" дистрибутиве, для этого не потребуется. Большинство разработчиков именно так и поступают. Как ни крути, а любой уважающий себя UNIX-программист вынужден держать на компьютере десяток различных операционных систем, чтобы тестировать свои программы на совместимость. На "живом" компьютере переключения между ними происходят только путем перезагрузки, что не слишком удобно. Виртуальные машины при этом можно переключать в любом порядке и с любой скоростью (главное — это иметь как можно больше памяти!).

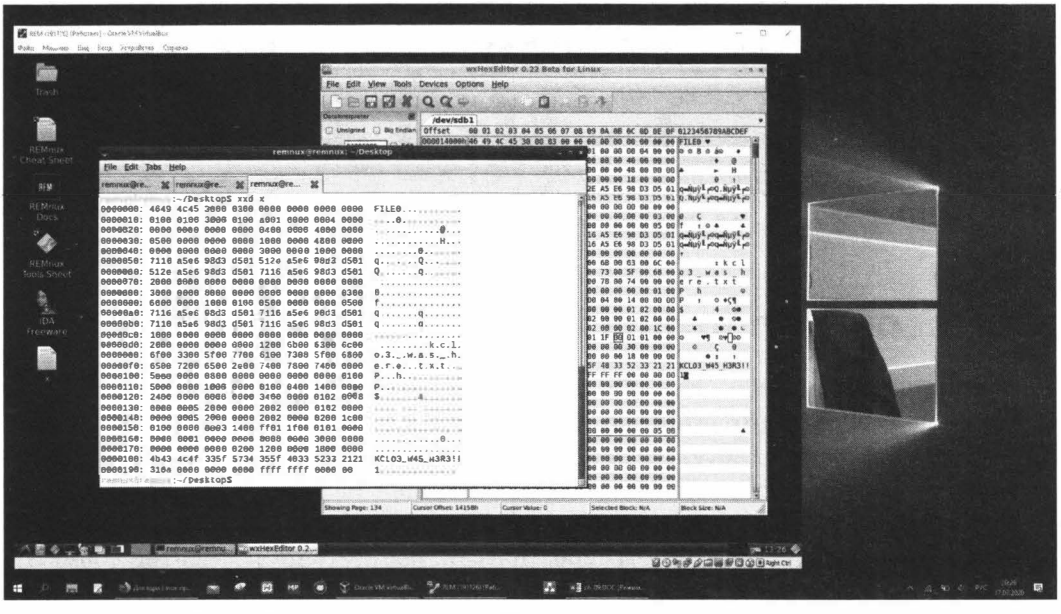


Рис. 9.1. Виртуальная машина Linux, работающая в среде Windows

Можно поступить и наоборот. Установить Linux/BSD как базовую систему, а Windows водрузить на виртуальную машину (рис. 9.2). Так как VMware дает прямой доступ к портам COM, LPT и USB, то подключение сканера, принтера или цифровой камеры к вашей машине перестанет быть проблемой. С этим оборудованием будет работать Windows! Базовая машина UNIX в этом случае получает в свое распоряжение все системные ресурсы, и падения производительности уже не происходит, но появляются другие проблемы. Приложения Windows (например, игры) будут либо тормозить, либо откажутся запускаться совсем, к тому же со всеми остальными типами устройств, например интегрированной платой WLAN или видеокартой, Windows работать не сможет. А все потому, что VMware представляет собой "черный 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...

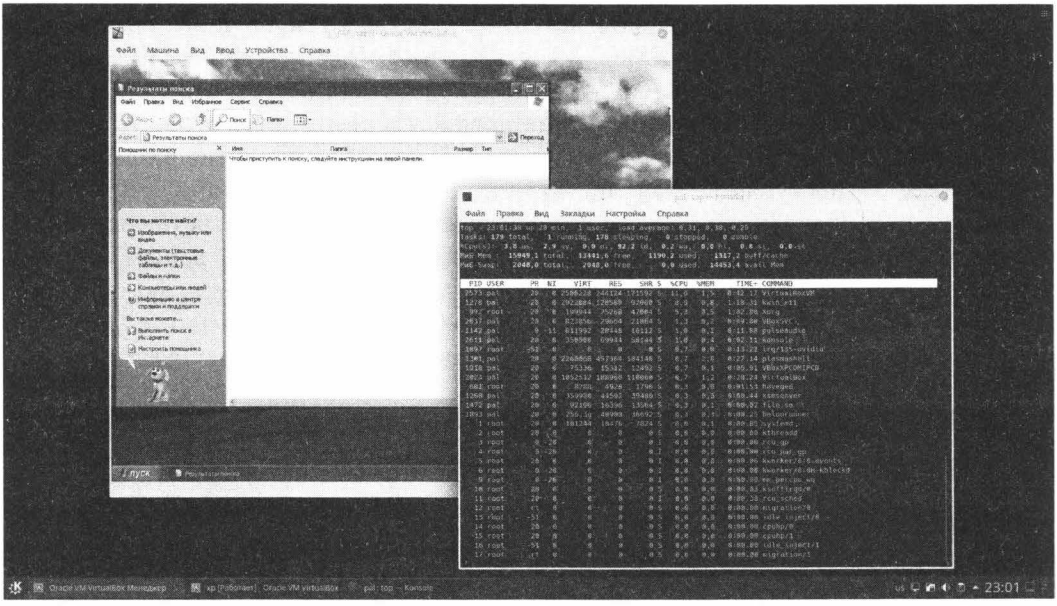


Рис. 9.2. Виртуальная машина Windows в среде Linux

Драйверы Windows в Linux/BSD

Начнем с простого, но до сих пор никем не решенного вопроса. Разумеется, на самом-то деле вопрос этот, конечно, уже давно решен, но совсем не так, как следовало бы. Поддержка разделов NTFS в Linux/BSD представляла собой сплошную проблему. Драйверы, способные осуществлять запись на разделы NTFS, появились совсем не сразу, не говоря уже о сжатых файлах, транзакциях и множестве других возможностей. Можно ли, хотя бы теоретически, написать стопроцентно совместимый драйвер, корректно работающий со всеми новыми версиями NTFS без участия программиста? Этот вопрос совсем не так глуп, как кажется. Для чего тратить силы

и время на собственный драйвер, когда под рукой есть уже готовый — NTFS.SYS. Если заставить его заработать под Linux, то все проблемы решатся сами собой. К примеру, программа NDISwrapper решает этот вопрос, но, к сожалению, лишь для драйверов Wi-Fi.

Несмотря на то, что на уровне ядра между Linux/BSD и Windows существует большое количество различий, но и кое-что общее между ними все-таки имеется. И Windows, и Linux, и BSD работают на процессорах семейства *x*86 в защищенном режиме, используют страничную организацию виртуальной памяти и, наконец, все они взаимодействуют с оборудованием в строго установленном порядке (через иерархию физических и виртуальных шин). Высокоуровневые драйверы, такие, например, как NTFS.SYS, вообще не работают с оборудованием напрямую и содержат минимум системно-зависимого кода. Почему же тогда драйвер от одной системы не работает в другой? А потому, что интерфейс между ОС и драйвером в каждом случае различен, а также потому, что драйвер использует библиотеку функций, экспортируемых системой, и эти функции у каждой системы свои.

Перенести драйвер Windows 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 также перенести библиотеку функций, необходимых драйверу для работы. Конечно, для этого необходимо уметь программировать! Для простых пользователей такой рецепт совершенно не годится, но тут уж ничего не поделаешь. Тем не менее перенести готовый драйвер намного проще, чем переписать его с нуля. Нам не потребуется кропотливая работа по дизассемблированию оригинального кода, заменяющая собой поиск технической документации (которая либо совсем отсутствует, либо отдается только под подписку о неразглашении, зачастую запрещающую открытое распространение исходных текстов). Наконец, при выходе новых версий драйвера Windows процедура его переноса в Linux/BSD проста до тривиальности — достаточно скопировать новый файл поверх старого файла. Однако все это лишь сухая теория. Перейдем к деталям.

Модель ядра Windows NT и всех производных от нее операционных систем достаточно проста (рис. 9.3). С "внешним" миром ядро связывает *диспетчер системных сервисов*, "подключенный" к NTDLL.DLL, которая находится уже за "скорлупой" ядра и исполняется в режиме пользователя. Диспетчер системных сервисов, реализованный в NTOSKRNL.EXE, опирается на *вызываемые интерфейсы ядра*, часть которых реализована в самом файле NTOSKRNL.EXE, а часть — во внешних драйверах, к числу которых, в частности, принадлежит *диспетчер питания*. 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!

Ядро, на котором, как на фундаменте, держатся все вышеупомянутые компоненты, представляет собой просто совокупность низкоуровневых функций, сосредоточенных в NTOSKRNL.EXE. Ниже находится только *уровень аппаратных абстракций* (Hardware Abstraction Layer, HAL). Когда-то у Microsoft была идея разделить ядро

на системно-зависимую и системно-независимую части, чтобы упростить перенос Windows на другие платформы. Однако уже во времена Windows NT 4 все перемешалось и большая часть системно-зависимых функций попала в NTOSKRNL.EXE. На сегодняшний день ситуация такова, что HAL медленно, но неотвратимо умирает. В нем осталось небольшое количество действительно низкоуровневых функций, непосредственно взаимодействующих с оборудованием, например 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 драйверы взаимодействуют с DMA не напрямую, а через *диспетчер Plug and Play*, который находится в NTOSKRNL.EXE.

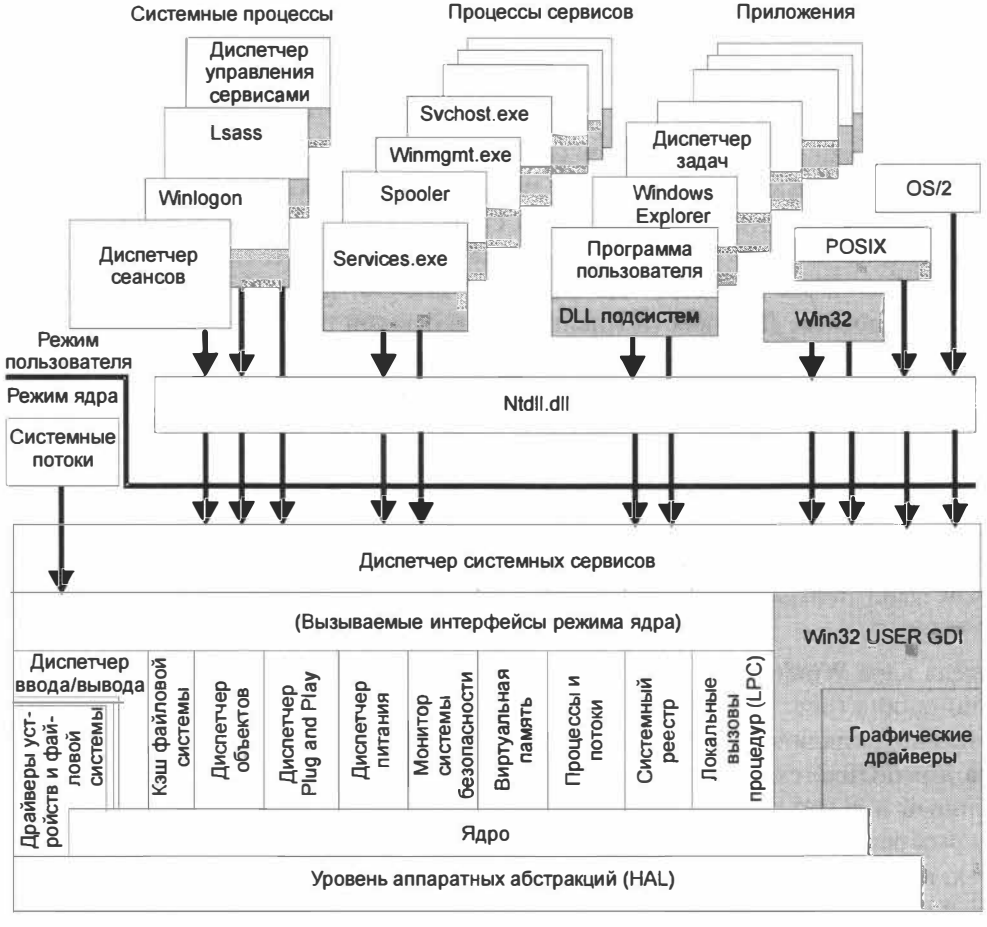


Рис. 9.3. Архитектура систем из семейства Windows NT

Иначе говоря, если заставить NTOSKRNL.EXE работать в чужеродной среде Linux или BSD, мы получим возможность запускать любые драйверы Windows NT без какой-либо доработки их двоичного кода. Это не только упрощает задачу переноса, но и снимает проблему авторских прав. 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 вот модифицировать двоичный код ему позволят едва ли.

Но мы ведь и не собираемся ничего модифицировать! Мы берем готовый NTOSKRNL.EXE. 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 загрузчик PE и включить его в загружаемый модуль ядра или в само ядро. Чтобы не мучиться, можно взять готовый загрузчик Wine (Windows Emulator).

Взаимодействие NTOSKRNL.EXE с ядром Linux/BSD будет происходить через код, эмулирующий HAL. Этот код мы должны написать сами, однако ничего сложного здесь нет, и объем работы предстоит минимальный, поскольку HAL содержит совсем немного простых функций. Сложнее подружить диспетчер системных вызовов с внешним миром, т. е. с миром Linux/BSD. Основная проблема в том, что интерфейс диспетчера системных вызовов не документирован, и к тому же подвержен постоянным изменениям. Поэтому приходится хитрить и тащить за собой не только NTOSKRNL.EXE, но еще и NTDLL.DLL. Некоторые могут спросить: а зачем? Какое отношение NTDLL.DLL имеет к драйверам и ядру? Драйверы его не вызывают, да и сам NTDLL.DLL представляет собой всего лишь набор переходников к NTOSKRNL.EXE.

Дело в том, что по традиции интерфейс NTDLL.DLL худо-бедно документирован. Кроме того, он 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, т. е. 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. С учетом отладки потребуется как минимум месяц. Но работа стоит того!

В результате в Linux/BSD наладится нормальная работа с NTFS и некоторыми другими драйверами ввода-вывода. Определенные сложности могут возникнуть разве что с видеокартами. Кстати говоря, компании nVIDIA (рис. 9.4) и AMD (рис. 9.5) выпускают проприетарные драйверы Linux под наиболее популярные чипсеты, так что проблема снимается сама собой.

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 можно будет запускать различные драйверы. Справедливости ради заметим, что подобный проект существовал и в свое время он один реализовывал операцию записи на NTFS-разделы. Имя ему Captive **http://www.jankratochvil.net/project/captive/**), но он заброшен с тех пор, как развивается NTFS-3G. Почему бы нам, хакерам, не скооперироваться и не продолжить это дело? Например, создать новый проект на **https://www.sourceforge.net**, набрать группу и "оттянуться" по полной программе.

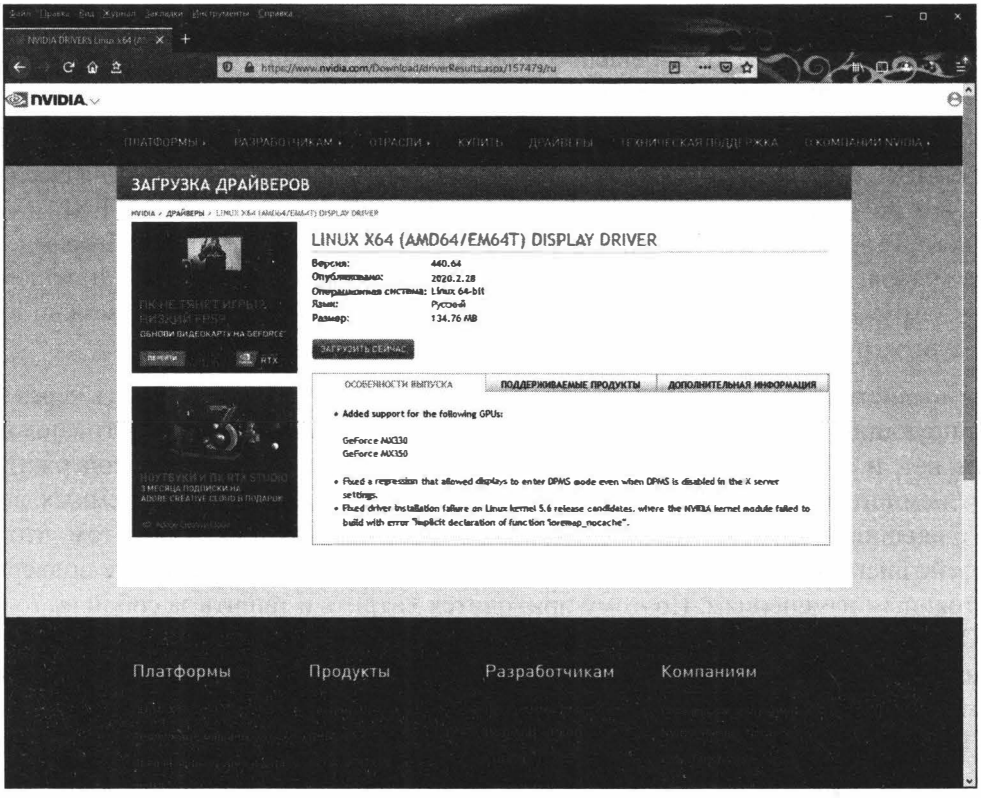


Рис. 9.4. Видеодрайверы для Linux x86_64 от nVIDIA

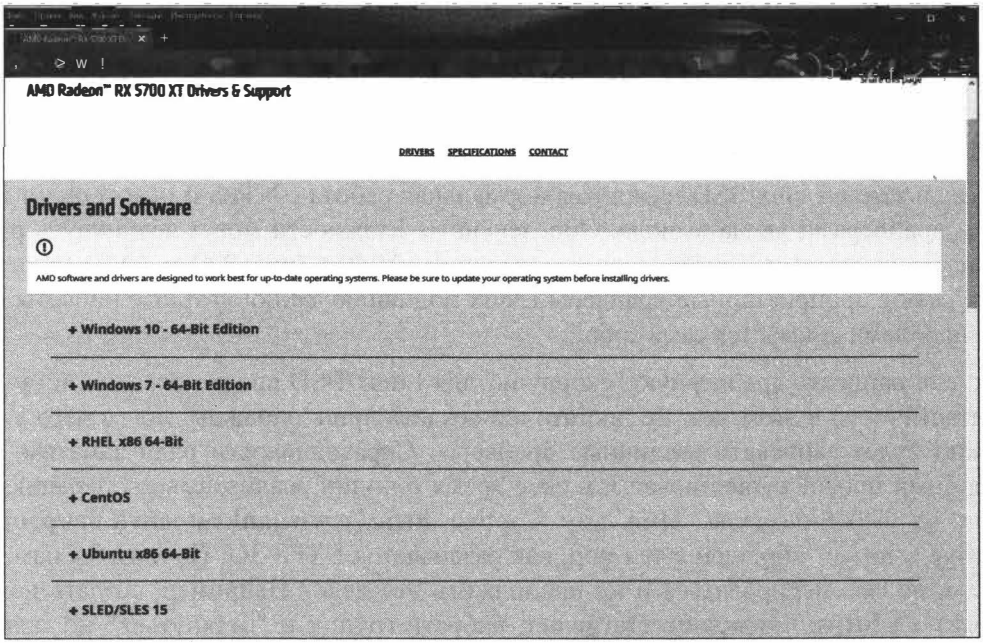


Рис. 9.5. Драйверы для Linux x86_64 Red Hat, CentOS, Ubuntu и SUSE от AMD (ATI)

Восстановление удаленных файлов под файловыми системами ext3fs/ext4fs

Каждый из нас хотя бы однажды удалял ценный файл, а то и весь корневой каталог целиком! Резервной копии нет (а должна бы быть!), времени на поиски утилит для восстановления — тоже. Как быть? К счастью, все необходимое уже включено в ваш любимый дистрибутив и всегда находится под рукой. Если говорить кратко, то это утилиты debugfs с командами `lsdel`, `stat`, `cat`, `dump`, а иной раз даже `grep`. Если требуется чуть более подробная информация — читайте этот материал, рассказывающий о восстановлении данных на разделах ext3fs и отчасти на разделах ext4fs.

Ошибочное удаление файлов в *NIX — это достаточно распространенное явление, наверное, даже более частое, чем в мире Microsoft. Под Windows большинство файловых операций осуществляется вручную с помощью Проводника или других интерактивных средств типа FAR или Windows Commander. Интерактивные среды есть и в Linux (KDE, GNOME, XFCE...), но немалая часть фанатов Linux — поклонники командной строки. Командная же строка — это регулярные выражения и скрипты, т. е. автоматизированные средства управления — мощные, удобные, и, при неправильном использовании, разрушительные. Малейшая небрежность — и можете навсегда попрощаться со своими файлами!

Перефразируя Булгакова, можно сказать: мало того, что файл смертен, так он еще и внезапно смертен! Беда никогда не предупреждает о своем приходе, и администратору приходится быть постоянно начеку. Несколько секунд назад все было хорошо: цвела весна, винчестер оживленно стрекотал всеми своими головками, администратор отхлебывал кофе из черной кружки с надписью root, как вдруг сотни гигабайтов ценнейших данных внезапно разлетелись на мелкие осколки. Все силы брошены на разгребание завалов и спасение всех, кого еще можно спасти.

Доступность исходных текстов драйвера файловой системы значительно упрощает исследование ее внутренней структуры, которая, кстати говоря, очень проста. Поэтому восстановление данных на разделах ext2/3/4 — задача тривиальная. Файловые системы UFS и FFS, работающие под FreeBSD, устроены намного сложнее, к тому же достаточно скудно документированы. А ведь FreeBSD занимает далеко не последнее место в мире UNIX-совместимых операционных систем, и разрушения данных даже в масштабах небольшого городка происходят сплошь и рядом. К счастью, в подавляющем большинстве случаев информацию можно полностью восстановить.

Подготовка к восстановлению

В первую очередь обязательно размонтируйте дисковый раздел или, на худой конец, перемонтируйте его в режим "только на чтение". Лечение активных разделов зачастую только увеличивает масштабы разрушений. Если восстанавливаемые файлы находятся на основном системном разделе, у нас два пути — загрузиться с LiveCD или подключить восстанавливаемый жесткий диск на Linux-машину вторым.

Чтобы случайно что-нибудь не испортить, никогда не редактируйте диск напрямую. Работайте с его копией! Копию можно создать командой `cp /dev/sdb1 my_dump`, где `sdb1` — имя устройства, а *my_dump* — имя файла-дампа. Файл-дамп можно разместить на любом свободном разделе или скопировать на другую машину по сети. Все дисковые утилиты не заметят подвоха и будут работать с ним как с "настоящим" разделом. При необходимости его даже можно смонтировать на файловую систему: `mount my_dump mount_point -o loop`, чтобы убедиться, что восстановление прошло успешно. Команда `cp my_dump /dev/sdb1` копирует восстановленный файл-дамп обратно в раздел, хотя делать это совсем необязательно. Проще (и безопаснее) копировать только восстанавливаемые файлы.

Восстановление удаленных файлов под ext3fs

Файловая система ext3fs все еще иногда встречается на клонах Linux, поэтому рассмотрим ее первой. Концепции, которые она исповедует, во многом схожи с NTFS, так что культурного шока при переходе с NTFS на ext3fs вы не испытаете. Подробное описание структуры хранения данных ищите в соответствующей документации, а также в книге Эндрю Таненбаума "Operating Systems: Design and Implementation". Исходные тексты дисковых утилит также не помешают.

Структура файловой системы

Третья расширенная файловая система (Third extended file system, ext3) во многом напоминает свою предшественницу, ext2 (настолько, что из современных ядер убран драйвер ext2, и для работы с этой ФС в ядре используется драйвер ext3), но отличается поддержкой журналирования (в терминологии NTFS — транзакций). В отличие от ext2fs, она намного бережнее относится к массиву каталогов.

В начале диска расположен загрузочный сектор, который на незагрузочных разделах может быть пустым. За ним по смещению 1024 байта от начала первого сектора лежит суперблок (super-block), содержащий ключевую информацию о структуре файловой системы. В FAT и NTFS эта информация хранится непосредственно в загрузочном секторе. В первую очередь нас будет интересовать 32-разрядное поле `s_log_block_size`, расположенное по смещению `18h` байтов от начала суперблока. Здесь хранится размер одного *блока* (block) или, в терминологии MS-DOS/Windows, кластера, выраженный в виде указателя позиции, на которую нужно сдвинуть число `200h`. В естественных единицах это будет звучать так: `block_size = 200h << s_log_block_size` (байт). То есть если `s_log_block_size` равен нулю, размер одного блока составляет `400h` байтов или два стандартных сектора. Структура дискового тома, отформатированного под ext3fs, показана в листинге 9.1. Подобную информацию можно увидеть в выводе команды `fsstat`.

Листинг 9.1. Структура дискового тома, размеченного под ext3fs

```
  смещение      размер                          описание
------- ------- -----------
      0           1 boot record                 ; Загрузочный сектор
```

```
        -- block group 0 --                    ; Группа блоков 0
(1024 bytes)       1 superblock                ; Суперблок
      2            1 group descriptors         ; Дескриптор группы
      3            1 block bitmap              ; Карта свободных блоков
      4            1 inode bitmap              ; Карта свободных inode
      5          214 inode table               ; Массив inode
                                               ; (сведения о файлах)
    219         7974 data blocks               ; Блоки данных
                                               ; (файлы, каталоги)
        -- block group 1 --                    ; Группа блоков 1
   8193            1 superblock backup         ; Копия суперблока
   8194            1 group descriptors backup  ; Копия дескриптора группы
   8195            1 block bitmap              ; Карта свободных блоков
   8196            1 inode bitmap              ; Карта свободных inode
   8197          214 inode table               ; Массив inode
                                               ; (сведения о файлах)
   8408         7974 data blocks               ; Блоки данных
                                               ; (файлы, каталоги)
        -- block group 2 --                    ; Группа блоков 2
  16385            1 block bitmap              ; Карта свободных блоков
  16386            1 inode bitmap              ; Карта свободных inode
  16387          214 inode table               ; Массив inode
                                               ; (сведения о файлах)
  16601         3879 data blocks               ; Блоки данных
                                               ; (файлы, каталоги)
```

Вслед за суперблоком идут *дескрипторы групп* (group descriptors) и *карты свободного пространства* (block bitmap/inode bitmap), которые не имеют большого практического значения для наших целей. Что же касается таблицы inode, расположенной за ними, то она заслуживает более подробного рассмотрения. В ext3fs (как и многих других файловых системах из мира UNIX) так называемый *индексный дескриптор* (inode) играет ту же самую роль, что и файловая запись в NTFS. Здесь сосредоточена вся информация о файле, кроме его имени: тип файла (обычный файл, каталог, символьная ссылка и т. д.), его логический и физический размер, схема размещения на диске, время создания, модификации, последнего доступа или удаления, права доступа, а также ссылки на файл. Формат представления inode в ext3 описан в листинге 9.2.

Листинг 9.2. Формат представления inode

```
смещение   размер          описание
-------- -------- -----------
       0        2 i_mode          ; Формат представления
       2        2 i_uid           ; Uid пользователя
       4        4 i_size          ; Размер файла в байтах
       8        4 i_atime         ; Время последнего доступа к файлу
```

```
 12       4 i_ctime          ; Время создания файла
 16       4 i_mtime          ; Время модификации файла
 20       4 i_dtime          ; Время удаления файла
 24       2 i_gid            ; Gid группы
 26       2 i_links_count    ; Количество ссылок на файл (0 - файл удален)
 28       4 i_blocks         ; Количество блоков, принадлежащих файлу
 32       4 i_flags          ; Различные флаги
 36       4 i_osd1           ; Значение, зависящее от ОС
 40 12 x 4 i_block           ; Ссылки на первые 12 блоков файла
 88       4 i_iblock         ; 1x INDIRECT BLOCK
 92       4 i_2iblock        ; 2x INDIRECT BLOCK
 96       4 i_3iblock        ; 3x INDIRECT BLOCK
100       4 i_generation     ; Поколение файла (используется NFS)
104       4 i_file_acl       ; Внешние атрибуты
108       4 i_dir_acl        ; higer size
112       4 i_faddr          ; Положение последнего фрагмента
116      12 i_osd2           ; Структура, зависящая от ОС
```

Первые 12 блоков, занимаемых файлом, называются непосредственными блоками (для наглядности они выделены полужирным шрифтом). Они хранятся в массиве `DIRECT BLOCKS` непосредственно в теле inode. Каждый элемент массива представляет собой 32-битовый номер блока. При среднем значении `BLOCK_SIZE`, равном 4 Кбайт, непосредственные блоки могут адресовать до `4×12 = 48` Кбайт данных. Если файл превышает этот размер, создаются один или несколько блоков косвенной адресации (`INDIRECT BLOCK`). Первый блок косвенной адресации (`1x INDIRECT BLOCK` или просто `INDIRECT BLOCK`) хранит ссылки на другие непосредственные блоки. Адрес этого блока хранится в поле `i_indirect_block` в inode. Как легко вычислить, он адресует порядка `BLOCK_SIZE/sizeof(DWORD) * BLOCK_SIZE = 4096/4 *4` Мбайт данных. Если этого окажется недостаточно, создается косвенный блок двойной косвенной адресации (`2x INDIRECT BLOCK` или `DOUBLE INDIRECT BLOCK`), хранящий указатели на косвенные блоки, что позволяет адресовать `(BLOCK_SIZE/sizeof(DWORD))*2* BLOCK_SIZE =4096/4 * 4096 = 4` Гбайт данных. Если же и этого все равно недостаточно, создается блок тройной косвенной адресации (`3x INDIRECT BLOCK` или `TRIPLE INDIRECT BLOCK`), содержащий ссылки на блоки двойной косвенной адресации. Иерархия непосредственных блоков и блоков косвенной адресации показана на рис. 9.6 (блок тройной косвенной адресации не показан).

По сравнению с NTFS такая схема хранения информации о размещении устроена намного проще. Вместе с тем она и гораздо "прожорливее". С другой стороны, ее несомненное достоинство по сравнению с NTFS состоит в том, что поскольку все ссылки хранятся в неупакованном виде, для каждого блока файла можно быстро найти соответствующий ему косвенный блок, даже если inode полностью разрушен!

Имя файла, как уже сказано, в inode не хранится. Ищите его внутри каталогов, представляющих собой массив записей, формат которого представлен в листинге 9.3.

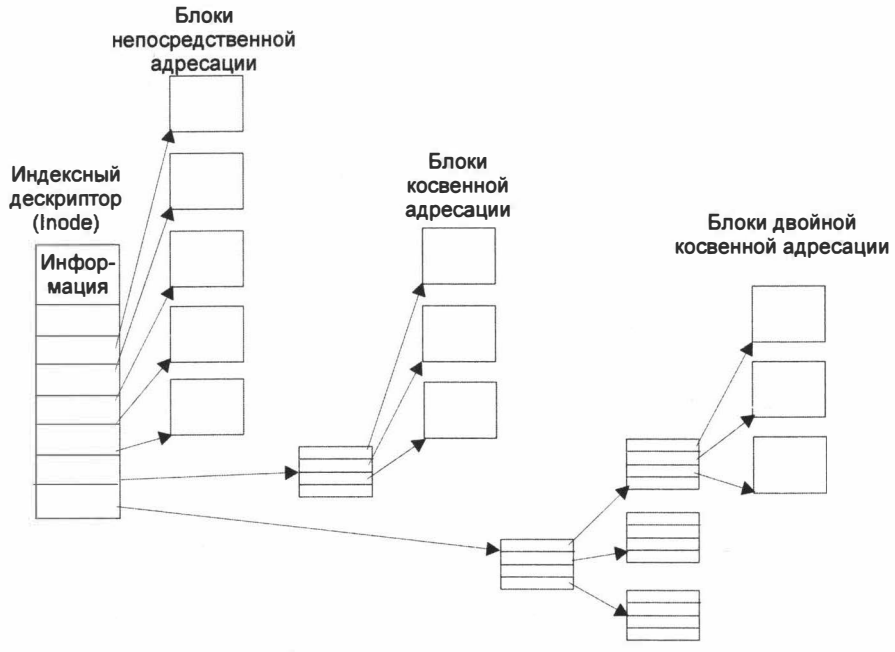


Рис. 9.6. Порядок размещения файла на диске — иерархия непосредственных и косвенных блоков (блок косвенной адресации третьего порядка не показан)

Листинг 9.3. Формат представления массива каталогов

```
смещение    размер            описание
------- ------- -----------
      0       4 inode         ; Ссылка на inode
      4       2 rec_len       ; Длина данной записи
      6       1 name_len      ; Длина имени файла
      7       1 file_type     ; Тип файла
      8     ... name          ; Имя файла
```

При удалении файла операционная система находит соответствующую запись в каталоге, обнуляет поле `inode` и увеличивает размер предшествующей записи (поле `rec_len`) на величину удаляемой записи. Таким образом, предшествующая запись "поглощает" удаленную. Хотя имя файла в течение некоторого времени остается нетронутым, ссылка на соответствующий ему индексный дескриптор (inode) оказывается уничтоженной. Это создает проблему, т. к. теперь придется разбираться, какому файлу принадлежит обнаруженное имя.

Отдельно стоит поговорить о журнале файловой системы. Он гарантирует целостность файловой системы в случае непредвиденных сбоев. При этом важно понимать, что целостность файловой системы совсем не означает сохранность файлов! Тем не менее наличие журнала играет важную роль при восстановлении данных в ext3/4, поскольку информация в нем зачастую помогает восстанавливать взаимо-

связи элементов каталогов, inode и содержимого файлов (если она, конечно, еще не была затерта новыми событиями файловой системы). Вот одна из причин, почему важно как можно скорее отмонтировать раздел со случайно удаленными файлами!

Журнал файловой системы ext3 (да и ext4) может работать в трех режимах, и от выбора будет зависеть как надежность, так и производительность:

- *обратной записи* — в журнал вносится только общая информация об операциях (метаданные), причем асинхронно по отношению к изменению в самих данных;
- *упорядочивания* — в журнал также вносятся только метаданные, но перед записью изменений в файле на диск;
- *полного журналирования* — в журнал записываются и метаданные, и изменения в самом файле. Этот вариант, соответственно, самый "прожорливый", но только он может обеспечить целостность данных. Два предыдущих лишь ускоряют выявление ошибок ФС при проверке утилитой fsck и позволяют восстановить целостность файловой системы, но не содержимого хранящихся файлов.

В самом индексном дескрипторе при удалении файла тоже происходят большие изменения. Число ссылок (`i_links_count`) обнуляется и обновляется поле последнего удаления (`i_dtime`). Все блоки, принадлежащие файлу, в карте свободного пространства (`block bitmap`) помечаются как неиспользуемые, после чего данный inode также освобождается (обновляется `inode bitmap`).

Техника восстановления удаленных файлов

В ext3fs в общем случае полное восстановление файлов невозможно, даже если эти файлы были только что удалены. В этом отношении данная файловая система проигрывает как FAT, так и NTFS. Как минимум нередко теряется связь имен файлов с их содержимым. При удалении небольшого количества хорошо известных файлов эта проблема остается решаемой. Однако ситуация серьезно осложняется, если вы удалили несколько служебных подкаталогов, в которые никогда ранее не заглядывали.

> ***Внимание!***
>
> **При непреднамеренном удалении важных файлов незамедлительное отмонтирование раздела или хотя бы его перемонтирование в режим "только для чтения" командой `mount -o remount,ro /dev/partition` может сыграть решающую роль при спасении ваших данных!**

Достаточно часто индексные дескрипторы назначаются в том же порядке, в котором создаются записи в таблице каталогов. Благодаря наличию расширений имен файлов (.c, .gz, .mpg и т. д.) количество возможных комбинаций соответствий обычно оказывается сравнительно небольшим. Тем не менее восстановить удаленный корневой каталог в автоматическом режиме никому не удастся (а вот NTFS с этим справляется без труда).

В целом стратегия восстановления выглядит приблизительно так: сканируем таблицу индексных дескрипторов (inode table) и отбираем все записи, у которых поле `i_links_count` равно нулю. Сортируем их по дате удаления, чтобы файлы, удален-

ные последними, оказались в верхних позициях списка. Как вариант, если вы помните примерное время удаления файла, можно просто наложить фильтр. Если соответствующие индексные дескрипторы еще не затерты вновь создаваемыми файлами, извлекаем список прямых/косвенных блоков и записываем их в дамп, корректируя его размер с учетом "логического" размера файла, за который отвечает поле `i_size`.

В ext3 при удалении файла ссылка на inode уже не уничтожается, что упрощает автоматическое восстановление оригинальных имен. Тем не менее поводов для радости у нас нет, поскольку в ext3fs перед удалением файла список принадлежащих ему блоков тщательно вычищается. Нефрагментированные файлы с более или менее осмысленным содержимым (например, исходные тексты программ) еще можно собрать по частям, но и времени на это потребуется немало. К счастью, блоки косвенной адресации не очищаются, а это значит, что мы теряем лишь первые `12 * BLOCK_SIZE` байт каждого файла. На типичном разделе объемом около 10 Гбайт значение `BLOCK_SIZE` обычно равно 4 или 8 Кбайт, т. е. реальные потери составляют менее 100 Кбайт. По современным понятиям, это сущие пустяки! Конечно, без этих 100 Кбайт большинство файлов просто не запустятся, однако найти на диске двенадцать недостающих блоков — вполне реальная задача. Если повезет, они окажутся расположенными в одном или двух непрерывных отрезках, но такое везение не гарантируется. Тем не менее непрерывные отрезки из 6–12 блоков достаточно часто встречаются даже на сильно фрагментированных разделах.

Как мы будем действовать? Необходимо найти хотя бы один блок, гарантированно принадлежащий файлу и при этом расположенный за границей в 100 Кбайт от его начала. Это может быть текстовая строка, информация об авторских правах разработчика или любая другая характерная информация! Короче говоря, нам нужен номер блока. Пусть для определенности он будет равен `0x1234`. Записываем его в обратном порядке так, чтобы младший байт располагался по меньшему адресу, и выполняем поиск `34h 12h 00h 00h` — именно это число будет присутствовать в косвенном блоке. Отличить косвенный блок от всех остальных блоков (например, блоков, принадлежащих файлам данных) очень легко — он представляет собой массив 32-битовых номеров блоков, более или менее монотонно возрастающих. Блоки с двойной и тройной косвенной адресацией отыскиваются по аналогичной схеме.

Проблема состоит в том, что одни и те же блоки в разное время могли принадлежать различным файлам, а это значит, что они могли принадлежать и различным косвенным блокам. Как разобраться, какой из найденных блоков является искомым? Да очень просто! Если хотя бы одна из ссылок косвенного блока указывает на уже занятый блок, данный косвенный блок принадлежит давно удаленному файлу и, следовательно, не представляет для нас интереса.

По правде говоря, debugfs обеспечивает лишь ограниченную поддержку ext3fs. В частности, команда `lsdel` всегда показывает отсутствие удаленных файлов, даже если удалить весь раздел. По этой причине вопрос выбора файловой системы от-

нюдь не так прост, каким его пытаются представить некоторые руководства по Linux для начинающих. Преимущества ext3fs на рабочих станциях и домашних компьютерах далеко не бесспорны и совсем не очевидны. Поддержка транзакций реально требуется лишь серверам, да и то не всем, а вот невозможность восстановления ошибочно удаленных файлов зачастую приносит бóльшие убытки, чем устойчивость файловой системы к внезапным отказам питания.

Восстановление с помощью отладчика файловой системы debugfs

Загружаем в отладчик редактируемый раздел или его копию. Сделать это можно с помощью команд `debugfs /dev/sdb1` или `debugfs my_dump` соответственно. Если мы планируем осуществлять запись на диск, то необходимо указать ключ `-w`: `debugfs -w my_dump` или `debugfs -w /dev/sdb1`.

Чтобы просмотреть список удаленных файлов, подаем команду `lsdel` (или `lsdel` *t_sec*, где *t_sec* — количество секунд, истекших с момента удаления файла). На экране появится список удаленных файлов (разумеется, без имен). Файлы, удаленные более *t_sec* секунд назад (если эта опция задана), в данный список не попадут.

Команда `cat <N>` выводит содержимое текстового файла на терминал. Здесь *<N>* — номер inode, заключенный в угловые скобки. При выводе двоичных файлов разобрать смысл отображаемой на экране информации практически невозможно. Такие файлы должны сбрасываться в дамп командой `dump <N>` *new_file_name*, где *new_file_name* — новое имя файла (с путем), под которым он будет записан в файловую систему, из-под которой был запущен отладчик debugfs. Файловая система восстанавливаемого раздела при этом остается неприкосновенной. Иными словами, команда должна даваться без ключа `--w`.

При желании можно восстановить файл непосредственно на самой восстанавливаемой файловой системе (что особенно удобно при восстановлении больших файлов). В этом нам поможет команда `undel <N>` *undel_file_name*, где *undel_file_name* — имя, которое будет присвоено файлу после восстановления.

> ***Внимание!***
>
> **Выполняя такую операцию, помните, что команда `undel` крайне "агрессивна" и деструктивна по своей природе. Она затирает первую свободную запись в таблице каталогов, делая восстановление оригинальных имен невозможным.**

Команда `stat <N>` отображает содержимое inode в удобочитаемом виде, а команда `mi <N>` позволяет редактировать его по своему усмотрению. Для ручного восстановления файла (откровенно говоря, этого не пожелаешь и врагу) мы должны установить счетчик ссылок (`link count`) на единицу, а время удаления (`deletion time`), наоборот, сбросить в нуль. Затем отдать команду `seti <N>`, помечающую данный inode как используемый, и для каждого из блоков файла выполнить команду `setb` *X*, где *X* — номер блока.

> ***Совет***
>
> **Перечень блоков, занимаемых файлом, можно просмотреть с помощью команды `stat`.**

Остается только дать файлу имя, что осуществляется путем создания ссылки на каталог (directory link). Сделать это можно с помощью команды `ln <N> undel_file_name`, где *undel_file_name* — имя, которое будет дано файлу после восстановления (при необходимости имя восстанавливаемого файла указывается с полным путем).

> ***Внимание!***
> **Создание ссылок на каталог необратимо затирает оригинальные имена удаленных файлов.**

После присвоения имени восстановленному файлу полезно задать команду `dirty`, чтобы файловая система была автоматически проверена при следующей загрузке. Как вариант, можно выйти из отладчика и вручную запустить команду `fsck` с ключом `-f`, форсирующим проверку.

Теперь перейдем к восстановлению оригинального имени. Рассмотрим простейший случай, когда каталог, содержащий удаленный файл (также называемый родительским каталогом), все еще цел. В этом случае следует подать команду `stat dir_name` и запомнить номер inode, возвращенный этой командой. Затем нужно задать отладчику команду `dump <INODE> dir_file`, где *INODE* — номер сообщенного индексного дескриптора, а *dir_file* — имя файла "родной" файловой системы, в которую будет записан дамп. Полученный дамп следует загрузить в шестнадцатеричный редактор и просмотреть его содержимое в "сыром" виде. Все имена будут там. При желании можно написать утилиту-фильтр, выводящую только удаленные имена. На Perl это не займет и десяти минут.

А как быть, если родительский каталог тоже удален? В этом случае и он будет помечен как удаленный! Выводим список удаленных индексных дескрипторов, выбираем из этого списка каталоги, формируем перечень принадлежащих им блоков и сохраняем дамп в файл, который затем следует просмотреть вручную или с помощью утилиты-фильтра. Как уже отмечалось, порядок расположения файлов в inode очень часто совпадает с порядком расположения файлов в каталоге, поэтому восстановление оригинальных имен в четырех случаях из пяти проходит на ура.

При тяжких разрушениях, когда восстанавливаемый файл приходится собирать по кусочкам, помогает команда `dump_unused`, выводящая на терминал все неиспользуемые блоки. Имеет смысл перенаправить вывод в файл, запустить hexedit и покопаться в этой "куче хлама" — так, по крайней мере, проще, чем искать по всему диску. На дисках, заполненных более чем на три четверти, этот трюк сокращает массу времени.

Таким образом, можно сделать вывод, что отладчик debugfs в значительной мере автоматизирует восстановление удаленных файлов, однако восстановить файл с разрушенным inode он не способен.

Восстановление удаленных файлов с помощью утилиты R-Studio

Утилита R-Studio поддерживает не только NTFS, но и файловые системы ext2/ext3/ext4. Имеются версии R-Studio под ОС Linux, а также бесплатный продукт

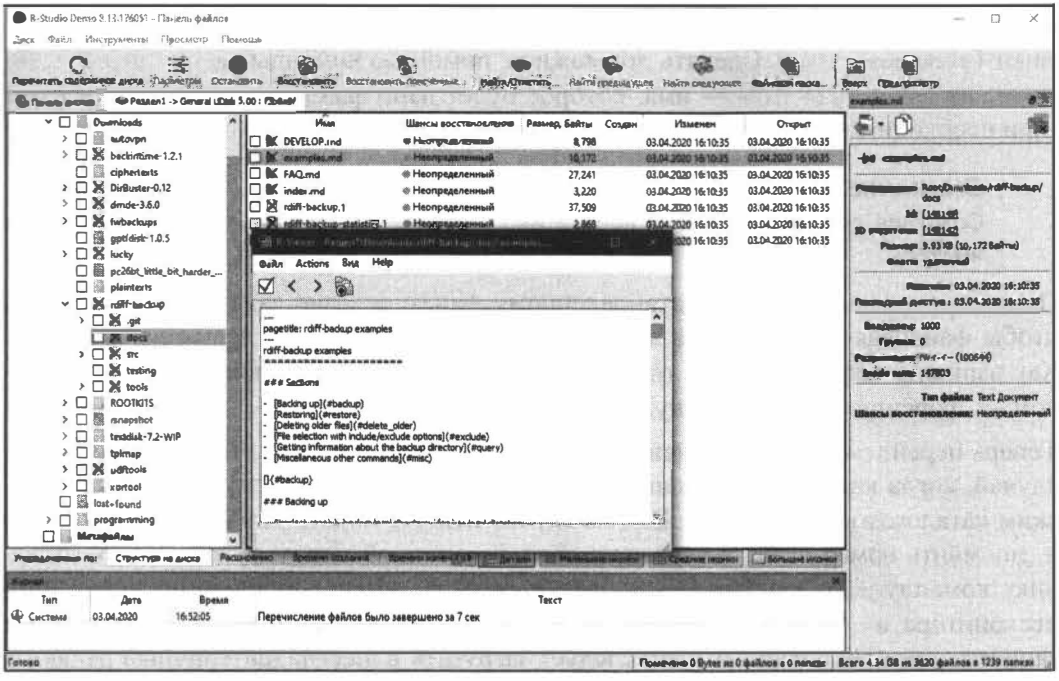


Рис. 9.7. Утилита R-Studio восстанавливает удаленные файлы на разделе ext3fs. Предпросмотр содержимого файлов и его метаданные выглядят правдоподобно

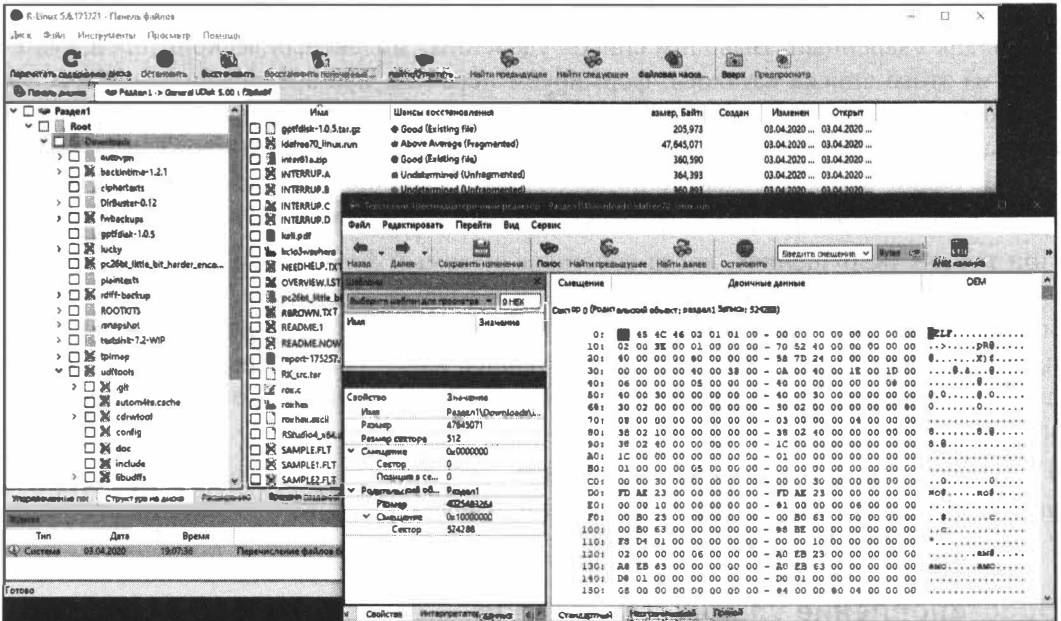


Рис. 9.8. Утилита R-Linux также небезуспешно восстанавливает удаленные файлы на разделе ext4fs

R-linux, предназначенный для восстановления исключительно ext2/3/4fs. С точки зрения обычных пользователей, это хорошее средство для автоматического восстановления удаленных файлов. Утилита предоставляет интуитивно понятный интерфейс, проста в управлении и в благоприятных ситуациях позволяет восстанавливать удаленные файлы несколькими щелчками мыши. К недостаткам R-Studio можно отнести следующие:

- отсутствие гарантий на восстановление файлов (чего, в общем-то, в случае ext2/3/4fs никто и не сможет гарантировать);
- отсутствие поддержки ручного восстановления.

Несмотря на это, со времени предыдущего издания этой книги R-Studio проделала долгий путь. В настоящее время при благоприятном стечении обстоятельств и своевременном начале восстановительных операций эта утилита может дать весьма обнадеживающие результаты (рис. 9.7 и 9.8).

Восстановление удаленных файлов на разделах ext4fs

Потолок разделов ext3 составляет 16 ТБ. И если обычным пользователям это ограничение до лампочки, то в корпоративных сегментах оно приносит неудобства. К тому же использование битовых карт на больших разделах (начиная примерно с 1 ТБ) заметно нагружает ЦП. В результате в ядре версии 2.6.19 появилась ext4fs. Она не так хорошо совместима с ext3, как та совместима с ext2, хотя ext3-раздел можно примонтировать как ext4 и наоборот (но тогда теряется смысл нововведений в последней ФС). Более того, разработчики предусмотрели возможность перехода с ext3 на ext4 без переформатирования раздела.

Одно из главных улучшений четвертой расширенной файловой системы — использование экстентов вместо карты свободных/занятых блоков, благодаря чему она намного эффективнее управляется с большими файлами, что улучшает масштабируемость на разделы больших объемов. В то же время механизм экстентов позволяет уменьшать фрагментацию файлов, копируя фрагментированные части в непрерывные экстенты.

Также в ext4 появился механизм контрольных сумм журнала, гарантирующий, что в файловую систему будут внесены корректные изменения. Еще ext4 может работать вообще без журнала, что немного улучшает ее производительность, но неизбежно снижает надежность файловой системы. Тем не менее данный режим можно назвать более предпочтительным для устройств на основе флеш-памяти, поскольку очень частые изменения файла журнала расходуют ресурс ячеек памяти.

В общем, казалось бы, с такими новшествами наши данные должны зажить новой жизнью, не зная горестей! Но они по-прежнему могут пропасть вследствие многочисленных причин. Что же на это скажет нам ext4?

Структура блоков ext4 не сильно отличается от имеющейся в ext3. Немного бóльшие изменения внесены в структуру индексных дескрипторов. Во-первых, она приобрела новые поля, стала вдвое объемнее и занимает минимум 256 байт. Это позволяет ей хранить, например, метки времени с наносекундной точностью

(раньше была точность до секунды), счетчик изменений файла (он позволяет понять клиенту NFS, изменился ли файл на стороне сервера), версию inode, а также ее контрольную сумму. Сами же номера inode стали 48-битовыми, благодаря чему поддерживаются разделы до 1 Эбайт при блоке в 4 КБайт. Подробно ознакомиться со структурой inode в ext4 можно в документации по адресу **https://www.kernel.org/doc/html/latest/filesystems/ext4/dynamic.html**. А мы заметим, что вместо карты блоков в inode ext4 по смещению `0x28` расположилось дерево экстентов `i_block[EXT4_N_BLOCKS=15]`. Экстенты определяют непрерывный участок из нескольких расположенных друг за другом блоков. Один экстент может адресовать до 128 МБайт при блоке в 4 Кбайт. Всего в одном индексном дескрипторе может храниться 4 экстента, а если их не хватает, то используется дерево экстентов, напоминающее схему косвенной адресации блоков в ext3 (рис. 9.9)

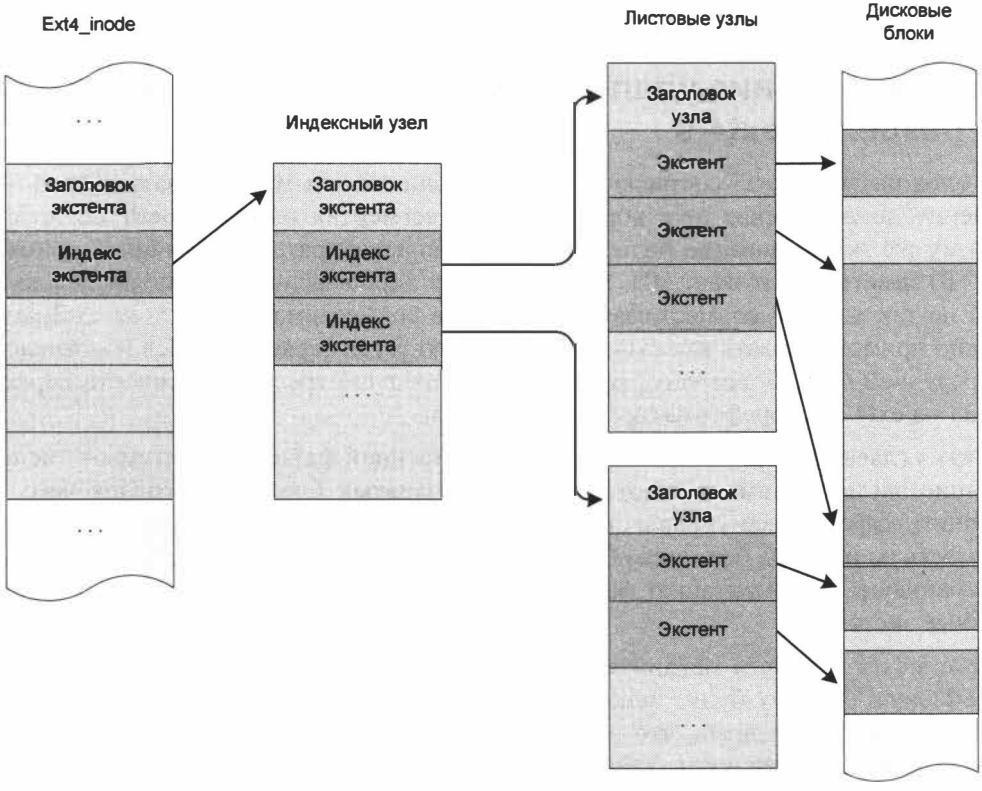


Рис. 9.9. Схема дерева экстентов в ext4

Пример структуры служебных данных на ext4-разделе приведен в листинге 9.4.

Листинг 9.4. Вывод команды fsstat в ext4fs (в сокращенном виде)

```
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext4
  < . . . >
```

```
Source OS: Linux
Dynamic Structure
Compat Features: Journal, Ext Attributes, Resize Inode, Dir Index
InCompat Features: Filetype, Needs Recovery, Extents, Flexible Block Groups,
Read Only Compat Features: Sparse Super, Large File, Huge File,
Extra Inode Size

Journal ID: 00
Journal Inode: 8

METADATA INFORMATION
--------------------------------------------
Inode Range: 1 - 1572865
Root Directory: 2
Free Inodes: 1313879
Inode Size: 256
Orphan Inodes: 1456398, 1456397, 1456396, 1456395, 1456394,

CONTENT INFORMATION
--------------------------------------------
Block Groups Per Flex Group: 16
Block Range: 0 - 6291199
Block Size: 4096
Free Blocks: 4295049

BLOCK GROUP INFORMATION
--------------------------------------------
Number of Block Groups: 192
Inodes per group: 8192
Blocks per group: 32768

Group: 0:
  Block Group Flags: [INODE_ZEROED]
  Inode Range: 1 - 8192
  Block Range: 0 - 32767
  Layout:
    Super Block: 0 - 0
    Group Descriptor Table: 1 - 2
    Group Descriptor Growth Blocks: 3 - 1024
    Data bitmap: 1025 - 1025
    Inode bitmap: 1041 - 1041
    Inode Table: 1057 - 1568
    Data Blocks: 9249 - 32767
  Free Inodes: 6469 (78%)
  Free Blocks: 23462 (71%)
  Total Directories: 53
  Stored Checksum: 0x09DC
```

```
Group: 1:
  Block Group Flags: [INODE_UNINIT, INODE_ZEROED]
  Inode Range: 8193 - 16384
  Block Range: 32768 - 65535
  Layout:
    Super Block: 32768 - 32768
    Group Descriptor Table: 32769 - 32770
    Group Descriptor Growth Blocks: 32771 - 33792
    Data bitmap: 1026 - 1026
    Inode bitmap: 1042 - 1042
    Inode Table: 1569 - 2080
    Data Blocks: 33793 - 65535
  Free Inodes: 8192 (100%)
  Free Blocks: 742 (2%)
  Total Directories: 0
  Stored Checksum: 0x24E7

Group: 2:
  Block Group Flags: [INODE_UNINIT, INODE_ZEROED]
  Inode Range: 16385 - 24576
  Block Range: 65536 - 98303
  Layout:
    Data bitmap: 1027 - 1027
    Inode bitmap: 1043 - 1043
    Inode Table: 2081 - 2592
    Data Blocks: 65536 - 98303
  Free Inodes: 8192 (100%)
  Free Blocks: 16423 (50%)
  Total Directories: 0
  Stored Checksum: 0xA60C
  < . . . >
```

По сути, процесс восстановления файлов (вернее, проблемы, препятствующие этому), тот же, что и в ext3. Однако появилась утилита с замечательным названием ext4magic (которая умеет работать и с разделами ext3). Она уже несколько лет не обновлялась, но и структуры ФС по большей части сформировались уже давно и изменяются незначительно. Эта программа в отдельных случаях способна спасать не только файлы, но и их названия вместе со всеми атрибутами! Важную роль при восстановлении играет состояние журнала, которое позволяет реконструировать взаимосвязь файлов с описывающей их служебной информацией — элементами каталогов и индексными дескрипторами. Естественно, чем меньше времени и изменений в файловой системе произошло с момента удаления файлов, тем больше вероятность их успешного восстановления (рис. 9.10). После случайного удаления файла следует как можно быстрее отмонтировать ФС, по возможности создать образ восстанавливаемого раздела и работать уже с ним. Что ж, в этом непростом деле без магии никак! Эта волшебная утилита доступна по адресу **https://sourceforge.net/projects/ext4magic/**.

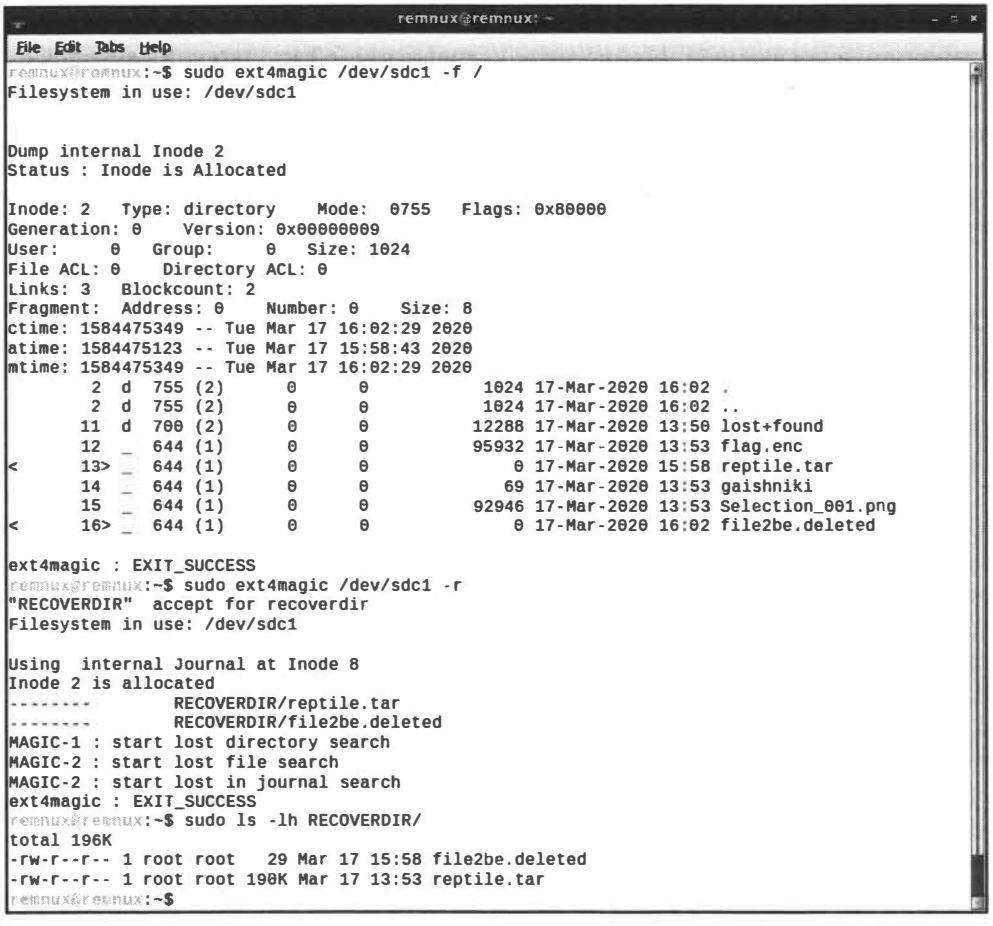


Рис. 9.10. Восстановление недавно удаленных файлов с помощью ext4magic

Довольно много интересной информации об ext4 можно отыскать в соответствующей документации по адресу:

https://www.kernel.org/doc/html/latest/filesystems/ext4/.

Рекомендуемые источники

- *"Design and Implementation of the Second Extended File system"* — подробное описание файловой системы ext2fs от разработчиков данного проекта (на английском языке). Полезно для понимания базовых концепций семейства ФС ext2/3/4. Это руководство доступно по адресу: **http://e2fsprogs.sourceforge.net/ext2intro.html**. Там же (**http://e2fsprogs.sourceforge.net**) доступен пакет утилит для работы с этими файловыми системами, исходные тексты которых достойны изучения.
- *"Не новое, но хорошо доработанное старое: взгляд на ext4"* — статья Андрея Пешеходова в журнале "Системный администратор" (2010. № 1–2 (86–87)), достаточно подробно описывающая логику работы ext4 и ее преимущества перед ext3.

- "*ext4 Data Structures and Algorithms*" — раздел документации, описывающий структуру и логику ext4: **https://www.kernel.org/doc/html/latest/filesystems/ext4/**.
- Описание нюансов восстановления файлов можно найти на страницах проектов, посвященных восстановлению удаленных файлов на разделах ext2/3/4: уже упомянутая утилита ext4magic (**https://sourceforge.net/projects/ext4magic/**), giis (**http://www.giis.co.in/**), extundelete (**http://extundelete.sourceforge.net/**) и ext3grep (**https://code.google.com/archive/p/ext3grep/**).

Восстановление удаленных файлов на разделах UFS

Файловая система UNIX (UNIX File System, UFS) — это основная файловая система для систем BSD, устанавливаемая по умолчанию. Многие коммерческие варианты UNIX также используют либо UFS в чистом виде, либо одну из файловых систем, созданных на ее основе и очень на нее похожих. В отличие от ext2/3/4, хорошо документированной и изученной вдоль и поперек, UFS в доступной литературе описана крайне поверхностно. Единственным источником информации становятся исходные тексты, в которых не так-то просто разобраться! Существует множество утилит, восстанавливающих уничтоженные данные (или, во всяком случае, пытающихся это делать), но на практике все они оказываются неработоспособными. Это, в общем-то, и неудивительно, поскольку автоматическое восстановление удаленных файлов под UFS невозможно в принципе. Тем не менее файлы, удаленные с разделов UFS, вполне возможно восстановить вручную, если, конечно, знать, как это делается.

Исторический обзор

UFS ведет свою историю от файловой системы s5. Это самая первая файловая система, написанная для UNIX в далеком 1974 году. Файловая система s5 была крайне простой и неповоротливой (по некоторым данным, ее производительность составляла от 2 до 5% от "сырой" производительности "голого" диска). Тем не менее понятия суперблока (super-block), файловых записей (inodes) и блоков данных (blocks) в ней уже существовали.

В процессе работы над дистрибутивом 4.2 BSD, вышедшим в 1983 году, оригинальная файловая система была несколько усовершенствована. Были добавлены длинные имена файлов и каталогов, символические ссылки и т. д. Так родилась UFS.

В 4.3 BSD, увидевшей свет уже в 1984 году, улучшения носили намного более радикальный, можно даже сказать — революционный, характер. Появились концепции фрагментов (fragments) и групп цилиндров (cylinder groups). Быстродействие файловой системы существенно возросло, что и позволило разработчикам назвать ее быстрой файловой системой (Fast File System, FFS).

Все последующие версии линейки 4.*x* BSD прошли под знаменем FFS, но в 5.*x* BSD файловая система вновь изменилась. Для поддержки дисков большого объема ши-

рину всех адресных полей пришлось удвоить: 32-битовая нумерация фрагментов уступила место 64-битовой. Были внесены и другие, менее существенные усовершенствования.

Таким образом, на практике мы имеем дело с тремя различными файловыми системами, несовместимыми друг с другом на уровне базовых структур данных. Тем не менее некоторые источники склонны рассматривать FFS как надстройку над UFS. Например, документ "Little UFS2 FAQ" (**https://lists.freebsd.org/pipermail/freebsd-current/2003-April/001444.html**) утверждает, что UFS и UFS2 определяют раскладку данных на диске, причем FFS реализована как надстройка над UFS/UFS2 и отвечает за структуру каталогов и оптимизацию операций доступа к диску. Однако если заглянуть в исходные тексты файловой системы, можно обнаружить два подкаталога — /ufs и /ffs. В /ffs находится определение суперблока (базовой структуры, отвечающей за раскладку данных), а в /ufs — определения inode и структуры каталогов, что опровергает данный тезис, с точки зрения которого все должно быть с точностью "до наоборот". Надо заметить, что кроме уже названных файловых систем в отдельных дистрибутивах могут использоваться свои собственные — например, ZFS, изначально разработанная для ОС Solaris, и HAMMER/HAMMER2 в DragonFlyBSD.

Чтобы не увязнуть в болоте терминологических тонкостей, под UFS мы будем понимать основную файловую систему 4.5 BSD, а под UFS2 — основную файловую систему 5.*x* BSD.

Структура UFS

В целом UFS очень похожа на ext2/3/4fs — те же inode, блоки данных, файлы, каталоги. Тем не менее есть между этими файловыми системами и различия. В ext2fs имеется только одна группа индексных дескрипторов (inodes) и только одна группа блоков данных для всего раздела. В отличие от ext2/3/4fs, UFS делит раздел на несколько зон одинакового размера, называемых группами цилиндров. Каждая зона имеет свою группу индексных дескрипторов и свою группу блоков данных, независимых ото всех остальных зон. Иначе говоря, индексные дескрипторы описывают блоки данных той и только той зоны, к которой они принадлежат. Это повышает быстродействие файловой системы, т. к. головка жесткого диска совершает более короткие перемещения. Кроме того, такая организация упрощает процедуру восстановления при значительном разрушении данных, поскольку, как показывает практика, обычно повреждается только первая группа индексных дескрипторов. Случаи, когда гибнут все группы, встречаются крайне редко. Образно выражаясь, для того, чтобы умышленно этого добиться, диск потребуется положить под гидравлический пресс.

Наглядное сравнение файловых систем s5 и UFS приведено на рис. 9.11. В UFS каждый блок разбит на несколько фрагментов фиксированного размера, предотвращающих потерю свободного пространства в "хвостах" файлов. В результате наличие блоков большого размера уже не кажется расточительной идеей, напротив, это увеличивает производительность и уменьшает фрагментацию. Если файл использует более одного фрагмента в двух несмежных блоках, то он автоматически

перемещается на новое место, в наименее фрагментированный регион свободного пространства. Таким образом, фрагментация в UFS очень мала или же совсем отсутствует, что существенно облегчает восстановление удаленных файлов и разрушенных данных.

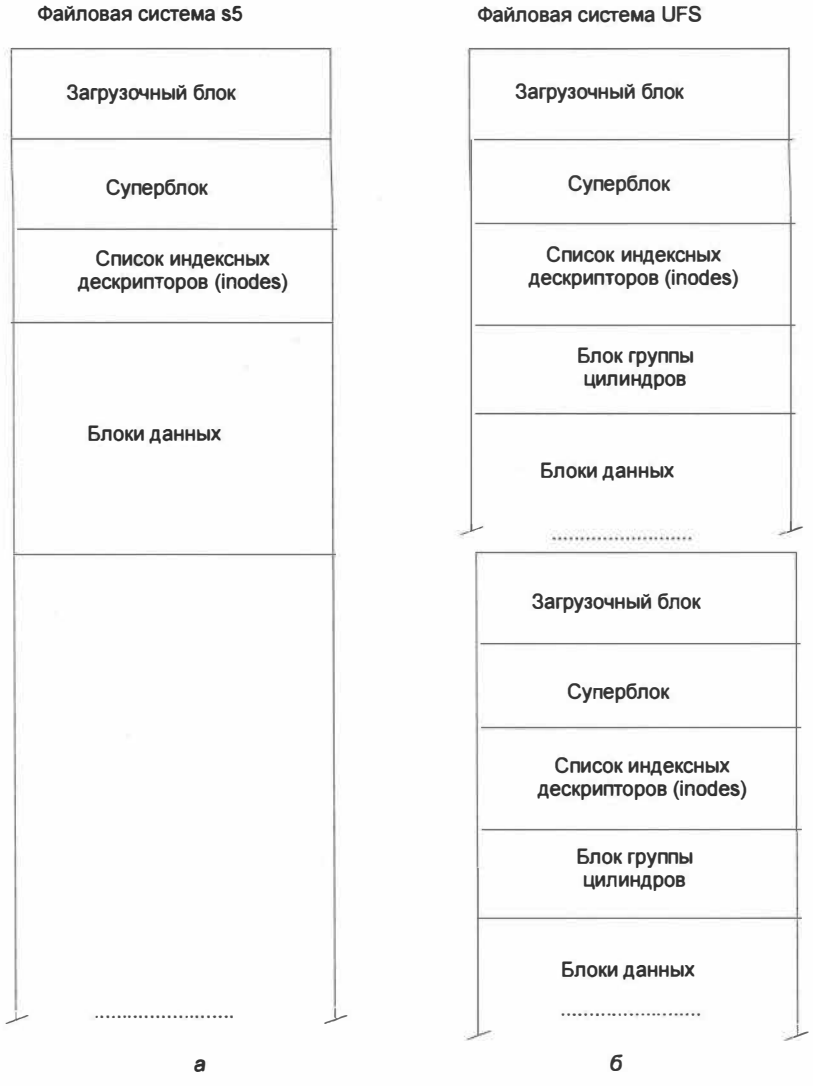


Рис. 9.11. Структура файловых систем s5/ext2fs (*а*) и UFS (*б*)

Адресация ведется либо по физическому смещению, измеряемому в байтах и отсчитываемому от начала группы цилиндров (реже — от начала раздела UFS), либо по номеру фрагмента, отсчитываемому от тех же самых точек. Допустим, размер блока составляет 16 Кбайт, разбитых на 8 фрагментов. Тогда 69-й сектор будет

иметь смещение `512 × 69 = 35328` байтов, или `1024 × (16/8)/512 × 69 = 276` фрагментов.

В начале раздела расположен загрузочный сектор, затем следует суперблок, за которым находится одна или несколько групп цилиндров (рис. 9.12). Для перестраховки копия суперблока дублируется в каждой группе. Загрузочный сектор не дублируется, но по соображениям унификации и единообразия под него просто выделяется место, таким образом, относительная адресация блоков в каждой группе остается неизменной.

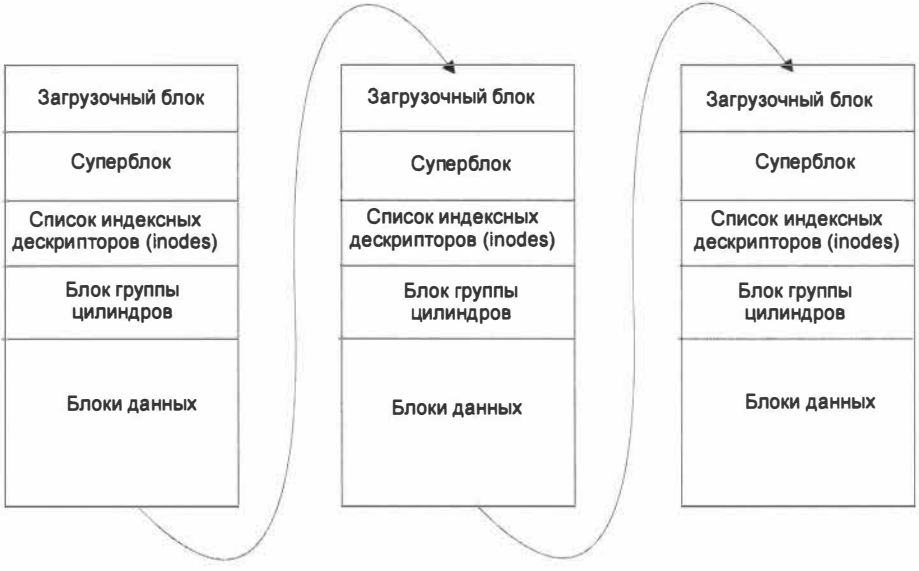


Рис. 9.12. Последовательно расположенные группы цилиндров

В UFS суперблок располагается по смещению 8192 байта от начала раздела, что соответствует 16-му сектору. В UFS2 он "переехал" на 65 536 байтов (128 секторов) от начала, освобождая место для дисковой метки и первичного загрузчика операционной системы, а для действительно больших (в исходных текстах они обозначены как "piggy") систем предусмотрена возможность перемещения суперблока по адресу 262 144 байта (целых 512 секторов).

Среди прочей информации суперблок содержит:

- `cblkno` — смещение первой группы блока цилиндров, измеряемое во фрагментах, отсчитываемых от начала раздела;
- `fs_iblkno` — смещение первого inode в первой группе цилиндров (фрагменты от начала раздела);
- `fs_dblkno` — смещение первого блока данных в первой группе цилиндров (фрагменты от начала раздела);
- `fs_ncg` — количество групп цилиндров;

- `fs_bsize` — размер одного блока в байтах;
- `fs_fsize` — размер одного фрагмента в байтах;
- `fs_frag` — количество фрагментов в блоке;
- `fs_fpg` — размер каждой группы цилиндров, выраженный в блоках (также может быть найден через `fs_cgsize`).

Для перевода смещений, выраженных во фрагментах, в номера секторов служит следующая формула: `sec_n(fragment_offset) = fragment_offset*(fs_bsize/fs_frag/512)`, или ее более короткая разновидность: `sec_n(fragment_offset) = fragment_offset* fs_fsize/512`.

Структура суперблока определена в файле /src/ufs/ffs/fs.h и в упрощенном виде выглядит так, как показано в листинге 9.5.

Листинг 9.5. Формат суперблока (второстепенные поля опущены)

```
struct fs {
/* 0x00 */ int32_t fs_firstfield;  /* Связный список файловых систем */
/* 0x04 */ int32_t fs_unused_1;    /* для внутренних суперблоков */
/* 0x08 */ ufs_daddr_t fs_sblkno;  /* Адрес суперблока в файловой системе (ФС)
*/
/* 0x0C */ ufs_daddr_t fs_cblkno;  /* Смещение блока цилиндров в ФС */
/* 0x10 */ ufs_daddr_t fs_iblkno;  /* Смещение блоков inode в ФС */
/* 0x14 */ ufs_daddr_t fs_dblkno;  /* Смещение 1-го блока данных после
                                      группы цилиндров */
/* 0x18 */ int32_t fs_cgoffset;    /* Смещение группы цилиндров */
/* 0x1C */ int32_t fs_cgmask;      /* Используется в calc mod fs_ntrak */
/* 0x20 */ time_t  fs_time;        /* Время последней записи */
/* 0x24 */ int32_t fs_size;        /* Количество блоков в ФС */
/* 0x28 */ int32_t fs_dsize;       /* Количество блоков данных в ФС */
/* 0x2C */ int32_t fs_ncg;         /* Количество групп цилиндров */
/* 0x30 */ int32_t fs_bsize;       /* Размер базовых блоков в ФС */
/* 0x34 */ int32_t fs_fsize;       /* Размер фрагментов блоков в ФС */
/* 0x38 */ int32_t fs_frag;        /* Количество фрагментов в блоке в ФС */

/* Параметры конфигурации */
/* 0x3C */ int32_t fs_minfree;       /* Мин. процент свободных блоков */
/* 0x40 */ int32_t fs_rotdelay;      /* Мин. задержка (мс) для оптимального
                                        следующего блока */
/* 0x44 */ int32_t fs_rps;           /* Обороты диска в минуту */

/* Размеры, определяемые кол-вом ГЦ и их размерами */
/* 0x98 */ ufs_daddr_t fs_csaddr;    /* Адрес блока информации ГЦ */
/* 0x9C */ int32_t fs_cssize;        /* Размер блока информации ГЦ */
/* 0xA0 */ int32_t fs_cgsize;        /* Размер группы цилиндров */
```

```
/* Поля, которые могут быть вычислены на основании остальных */
/* 0xB4 */ int32_t fs_cpg;          /* Кол-во цилиндров в группе */
/* 0xB8 */ int32_t fs_ipg;          /* Кол-во Inode на группу */
/* 0xBC */ int32_t fs_fpg;          /* Кол-во блоков в группе * fs_frag */

/* Поля, очищаемые при монтировании */
/* 0xD0 */ int8_t  fs_fmod;             /* Флаг модификации суперблока */
/* 0xD1 */ int8_t  fs_clean;            /* Флаг "чистой" (clean) ФС */
/* 0xD2 */ int8_t  fs_ronly;            /* Флаг защиты от записи */
/* 0xD3 */ int8_t  fs_flags;            /* См. поле fs_ flags */
/* 0xD4 */ u_char  fs_fsmnt[MAXMNTLEN]; /* Путь монтирования ФС */
};
```

За концом суперблока, на некотором отдалении от него, находится первая группа цилиндров (ГЦ). В начале каждой группы расположена служебная структура `cg`, представляющая собой описатель группы цилиндров и содержащая магическую последовательность `55h 02h 09h`, по которой все уцелевшие группы можно найти даже при полностью испорченном суперблоке. Штатным образом стартовые адреса всех последующих групп вычисляются путем умножения номера группы на ее размер, содержащийся в поле `fs_cgsize`.

Другие важные параметры:

- `cg_cgx` — порядковый номер группы, отсчитываемый от нуля;
- `cg_old_niblk` — количество inode в данной группе;
- `cg_ndblk` — число блоков данных в данной группе;
- `csum` — количество свободных inode и блоков данных в данной группе;
- `cg_iusedoff` — смещение карты занятых inode, отсчитываемое от начала данной группы (в байтах);
- `cg_freeoff` — смещение карты свободного пространства (байты от начала группы).

Структура `cg` определена в файле /src/ufs/ffs/fs.h и выглядит следующим образом — листинг 9.6.

Листинг 9.6. Структура описателя группы цилиндров

```
#define CG_MAGIC      0x090255
#define MAXFRAG       8
struct cg {
/* 0x00 */ int32_t  cg_firstfield;     /* Связный список групп цилиндров */
/* 0x04 */ int32_t  cg_magic;          /* Магическая последовательность */
/* 0x08 */ int32_t  cg_old_time;       /* Время последней записи */
/* 0x0C */ int32_t  cg_cgx;            /* Мы находимся в ГЦ номер cgx */
/* 0x10 */ int16_t  cg_old_ncyl;       /* Кол-во цилиндров в этой ГЦ */
/* 0x12 */ int16_t  cg_old_niblk;      /* Кол-во блоков inode в этой ГЦ */
/* 0x14 */ int32_t  cg_ndblk;          /* Кол-во блоков данных в этой ГЦ */
```

```
/* 0x18 */ struct  csum cg_cs;          /* Краткое описание цилиндра */
/* 0x28 */ int32_t  cg_rotor;           /* Положение посл. исп. блока */
/* 0x2C */ int32_t  cg_frotor;          /* Положение посл. исп. фрагмента */
/* 0x30 */ int32_t  cg_irotor;          /* Положение посл. исп. inode */
/* 0x34 */ int32_t  cg_frsum[MAXFRAG]; /* Счетчик доступных фрагментов */
/* 0x54 */ int32_t  cg_old_btotoff;     /* (int32) блоков на цилиндр */
/* 0x58 */ int32_t  cg_old_boff;        /* (u_int16) своб. позиций блоков */
/* 0x5C */ int32_t  cg_iusedoff;        /* (u_int8) карта исп. inode */
/* 0x60 */ int32_t  cg_freeoff;         /* (u_int8) карта своб. блоков */
/* 0x64 */ int32_t  cg_nextfreeoff;     /* (u_int8) след. своб. блок */
/* 0x68 */ int32_t  cg_clustersumoff;   /* (u_int32) счетчик своб. кластеров */
/* 0x6C */ int32_t  cg_clusteroff;      /* (u_int8) карта своб. кластеров */
/* 0x70 */ int32_t  cg_nclusterblks;    /* Кол-во кластеров в этой ГЦ */
/* 0x74 */ int32_t  cg_niblk;           /* Кол-во блоков inode в этой ГЦ */
/* 0x78 */ int32_t  cg_initediblk;      /* Посл. инициализированный inode */
/* 0x7C */ int32_t  cg_sparecon32[3];   /* Зарезервировано */
/* 0x00 */ ufs_time_t cg_time;          /* Время последней записи */
/* 0x00 */ int64_t  cg_sparecon64[3];   /* Зарезервировано */
/* 0x00 */ u_int8_t cg_space[1];        /* Место для карт ГЦ */
                                        /* реально больше */
```

Между описателем группы цилиндров и группой inode расположены карта занятых inode и карта свободного дискового пространства, представляющие собой обыкновенные битовые поля, точно такие же, как и в NTFS. При восстановлении удаленных файлов без этих карт обойтись невозможно. Они существенно сужают круг поиска, что особенно хорошо заметно на дисках, заполненных более чем наполовину.

За картами следует массив inode, смещение которого содержится в поле `cg_iusedoff` (адрес первой группы inode продублирован в суперблоке). По сути, в UFS структура inode ничем не отличается от ext2fs, только расположение полей другое. К тому же имеется лишь один блок косвенной адресации вместо трех, но это уже детали, не имеющие большого практического значения. Рассмотрим назначение фундаментальных полей, к числу которых принадлежат:

- `di_nlink` — количество ссылок на файл (0 означает "удален");
- `di_size` — размер файла в байтах;
- `di_atime/di_atimensec` — время последнего доступа к файлу;
- `di_mtime/di_mtimensec` — время последней модификации;
- `di_ctime/di_ctimensec` — время последнего изменения inode;
- `di_db` — адреса первых 12 блоков данных файла, отсчитываемые во фрагментах от начала группы цилиндров;
- `di_ib` — адрес блоков косвенной адресации (фрагменты от начала группы).

Сама структура inode определена в файле /src/ufs/ufs/dinode.h. Для UFS1 эта структура выглядит, как показано в листинге 9.7 и на рис. 9.13.

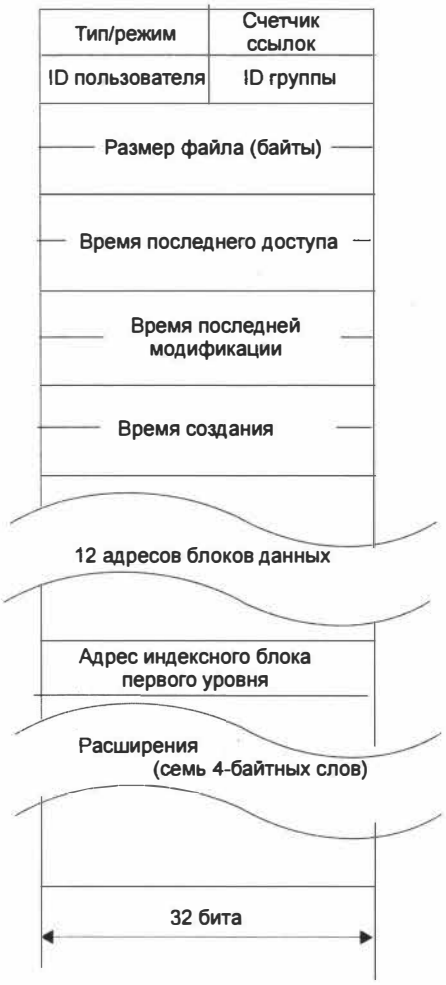


Рис. 9.13. Схематичное изображение inode

Листинг 9.7. Структура inode в UFS1

```
struct dinode {
/* 0x00 */   u_int16_t      di_mode;        /* 0: IFMT, права доступа; */
                                            /* см. ниже */
/* 0x02 */   int16_t         di_nlink;      /* 2: Счетчик ссылок */
/* 0x04 */   union {
               u_int16_t     oldids[2];     /* 4: Ffs: старые ID */
                                            /*    пользователя и группы */
               int32_t       inumber;       /* 4: Lfs: номер inode */
                   } di_u;
/* 0x08 */     u_int64_t     di_size;       /* 8:  Счетчик байтов файла */
/* 0x10 */     int32_t       di_atime;      /* 16: Время последнего доступа */
/* 0x14 */     int32_t       di_atimensec;  /* 20: Время последнего доступа */
```

```
/* 0x18 */      int32_t      di_mtime;      /* 24: Время последней */
                                            /*     модификации */
/* 0x1C */      int32_t      di_mtimensec;  /* 28: Время последней */
                                            /*     модификации */
/* 0x20 */      int32_t      di_ctime;      /* 32: Время последнего */
                                            /*     изменения inode */
/* 0x24 */      int32_t      di_ctimensec;  /* 36: Время последнего */
                                            /*     изменения inode */
/* 0x28 */      ufs_daddr_t  di_db[NDADDR]; /* 40: Непоср. дисковые блоки */
/* 0x58 */      ufs_daddr_t  di_ib[NIADDR]; /* 88: Косв. дисковые блоки */
/* 0x64 */      u_int32_t    di_flags;      /* 100: Флаги статуса (chflags) */
/* 0x68 */      int32_t      di_blocks;     /* 104: Факт. занятые блоки */
/* 0x6C */      int32_t      di_gen;        /* 108: Номер генерации */
/* 0x70 */      u_int32_t    di_uid;        /* 112: Владелец файла */
/* 0x74 */      u_int32_t    di_gid;        /* 116: Группа файла */
/* 0x78 */      int32_t      di_spare[2];   /* 120: Зарезервировано */
};
```

В UFS2 формат inode был существенно изменен — появилось множество новых полей, удвоилась ширина адресных полей (листинг 9.8). Что это означает для нас в практическом плане? Смещения всех полей изменились, только и всего, а общий принцип работы с индексными дескрипторами остался прежним.

Листинг 9.8. Структура inode в USF2

```
struct ufs2_dinode {
/* 0x00 */ u_int16_t    di_mode;        /*   0: IFMT, права доступа; */
                                        /*      см. ниже */
/* 0x02 */ int16_t      di_nlink;       /*   2: Счетчик ссылок */
/* 0x04 */ u_int32_t    di_uid;         /*   4: Владелец файла */
/* 0x08 */ u_int32_t    di_gid;         /*   8: Группа файла */
/* 0x0C */ u_int32_t    di_blksize;     /*  12: Размер блока Inode */
/* 0x10 */ u_int64_t    di_size;        /*  16: Счетчик байтов файла */
/* 0x18 */ u_int64_t    di_blocks;      /*  24: Практически занятые байты */
/* 0x20 */ ufs_time_t   di_atime;       /*  32: Время последнего доступа */
/* 0x28 */ ufs_time_t   di_mtime;       /*  40: Время последней */
                                        /*      модификации */
/* 0x30 */ ufs_time_t   di_ctime;       /*  48: Время последнего */
                                        /*      изменения inode */
/* 0x38 */ ufs_time_t   di_birthtime;   /*  56: Время создания Inode */
/* 0x40 */ int32_t      di_mtimensec;   /*  64: Время последней */
                                        /*      модификации */
/* 0x44 */ int32_t      di_atimensec;   /*  68: Время последнего доступа */
/* 0x48 */ int32_t      di_ctimensec;   /*  72: Время последнего доступа */
/* 0x4C */ int32_t      di_birthnsec;   /*  76: Время создания Inode */
/* 0x50 */ int32_t      di_gen;         /*  80: Номер генерации */
/* 0x54 */ u_int32_t    di_kernflags;   /*  84: Флаги ядра */
/* 0x58 */ u_int32_t    di_flags;       /*  88: Флаги статуса (chflags) */
```

```
/* 0x5C */ int32_t      di_extsize;      /*  92: Блок внешних атрибутов */
/* 0x60 */ ufs2_daddr_t di_extb[NXADDR]; /*  96: Блок внешних атрибутов */
/* 0x70 */ ufs2_daddr_t di_db[NDADDR];   /* 112: Непоср. дисковые блоки */
/* 0xD0 */ ufs2_daddr_t di_ib[NIADDR];   /* 208: Косв. дисковые блоки */
/* 0xE8 */ int64_t      di_spare[3];     /* 232: Зарезервировано */
 };
```

Имена файлов хранятся в каталогах (рис. 9.14). В индексных дескрипторах их нет. С точки зрения UFS каталоги являются файлами особого типа и могут храниться по любому адресу, принадлежащему группе цилиндров. Файловая система UFS поддерживает несколько типов хеширования каталогов, однако на структуре хранения имен это никак не отражается. Имена хранятся в блоках, называемых `DIRBLKSIZ`, в структурах типа `direct`, выровненных по 4-байтовой границе.

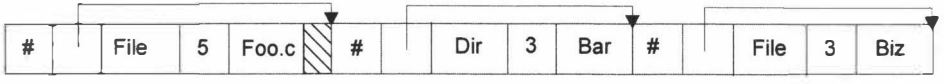


Рис. 9.14. Хранение имен файлов и каталогов

Структура `direct` определена в файле /src/ufs/ufs/dir.h (листинг 9.9) и содержит следующее: номер inode, описывающий данный файл, тип файла, его имя, а также длину самой структуры `direct`, используемую для нахождения следующей структуры этого типа в блоке.

Листинг 9.9. Структура `direct`, отвечающая за хранение имен файлов и каталогов

```
struct  direct {
/* 0x00 */ u_int32_t d_ino;              /* Номер inode данной записи */
/* 0x04 */ u_int16_t d_reclen;           /* Длина данной записи */
/* 0x06 */ u_int8_t  d_type;             /* Тип файла, см. ниже */
/* 0x07 */ u_int8_t  d_namlen;           /* Длина строки в d_name */
/* 0x08 */ char      d_name[MAXNAMLEN + 1]; /*Имя с длиной <=MAXNAMLEN */
};
```

На этом описание файловой системы UFS можно считать законченным. Для ручного восстановления данных приведенной информации вполне достаточно.

На развалинах империи

При удалении файла на разделе UFS происходят следующие события (они перечислены в порядке расположения соответствующих структур в разделе и могут не совпадать с порядком их возникновения).

- В суперблоке обновляется поле `fs_time` (время последнего доступа к разделу).
- В суперблоке обновляется структура `fs_cstotal` (количество свободных inode и блоков данных в разделе).

- В группе цилиндров обновляются карты занятых inode и блоков данных. Inode и все блоки данных удаляемого файла помечаются как освобожденные.
- В inode родительского каталога обновляются поля времени последнего доступа и времени последней модификации.
- В inode родительского каталога обновляется поле времени последнего изменения inode.
- В inode удаляемого файла обнуляются поля `di_mode` (IFMT, права доступа), `di_nlink` (количество ссылок на файл) и `di_size` (размер файла).
- В inode удаляемого файла затираются нулями поля `di_db` (массив указателей на 12 первых блоков файла) и `di_ib` (указатель на блок косвенной адресации).
- В inode удаляемого файла обновляются поля времени последней модификации и последнего изменения inode, время последнего доступа при этом остается неизменным.
- В inode удаляемого файла обновляется поле `di_spare`. В исходных текстах оно помечено как зарезервированное, но просмотр дампа показывает, что это не так. Судя по всему, здесь хранится нечто вроде последовательности обновления (update sequence), используемой для контроля целостности inode. Однако это только наше предположение.
- В каталоге удаленного файла размер предшествующей структуры `direct` увеличивается на значение `d_reclen`, в результате чего она как бы "поглощает" имя удаляемого файла. Однако физического затирания имени не происходит. Во всяком случае, оно затирается не сразу, а лишь в тот момент, когда в этом возникнет реальная необходимость.

Средства восстановления файлов

Обнаружив, что один или несколько файлов были непреднамеренно удалены, немедленно демонтируйте раздел и запустите дисковый редактор, работающий на секторном уровне. При наличии достаточного количества свободного дискового пространства можно скопировать раздел в файл и "натравить" на него любой hex-редактор. Как вариант, можно открыть непосредственно само устройство раздела (например, `/dev/ad0s1a`). Можно загрузить компьютер с загрузочного CD с Windows PE и воспользоваться любым подходящим Windows-редактором. Наконец, можно запустить KNOPPIX или любой дистрибутив Live Linux, ориентированный на восстановление данных.

В общем, выбор средств восстановления достаточно широк.

Техника восстановления удаленных файлов

Начнем наше обсуждение на грустной ноте. Поскольку при удалении файла ссылки на 12 первых блоков и 3 блока косвенной адресации необратимо затираются, автоматическое восстановление данных принципиально невозможно. Найти удаленный файл можно только по его содержимому. Искать, естественно, необходимо в сво-

бодном пространстве. Вот тут-то нам и пригодятся карты, расположенные за концом описателя группы цилиндров.

Если нам повезет и файл окажется нефрагментированным (а на разделах UFS, как уже отмечалось, фрагментация обычно отсутствует или крайне невелика), остальное будет делом техники. Достаточно выделить группу секторов и записать ее на диск. Здесь, как и во всех ранее описанных случаях, следует помнить, что запись ни в коем случае не должна вестись на сам восстанавливаемый раздел! Например, файл можно передать на соседний компьютер по сети. К сожалению, поле длины файла безжалостно затирается при его удалении и реальный размер приходится определять "на глазок". В реальности эта задача намного проще, чем кажется на первый взгляд. Неиспользуемый "хвост" последнего фрагмента всегда забивается нулями, что дает хороший ориентир. Проблема заключается в том, что некоторые типы файлов содержат в своем конце определенное количество нулей, при отсечении которых их работоспособность нарушается, поэтому тут приходится экспериментировать.

А что делать, если файл фрагментирован? Первые 13 блоков (именно блоков, а не фрагментов!) придется собирать руками. В идеальном случае это будет один непрерывный регион. Хуже, если первый фрагмент расположен в "чужом" блоке (т. е. блоке, частично занятом другим файлом), а оставшиеся 12 блоков находятся в одном или нескольких регионах. На практике, однако, достаточно трудно представить себе ситуацию, в которой первые 13 блоков были бы сильно фрагментированы, ведь UFS поддерживает фоновую дефрагментацию. Такое может произойти только при масштабной перегруппировке большого количества файлов, что в реальной жизни практически никогда не встречается. Поэтому будем считать, что 13-й блок файла найден. В массив непосредственной адресации он уже не помещается (там содержатся только 12 блоков), и ссылка на него, как и на все последующие блоки файла, должна содержаться в блоках косвенной адресации. Как вы помните, блоки косвенной адресации при удалении файла помечаются как свободные, но не затираются сразу же. Затирание происходит только по мере реальной необходимости, и это существенно упрощает нашу задачу.

Как найти этот блок на диске? Вычисляем смещение 13-го блока файла от начала группы цилиндров, переводим его во фрагменты, записываем получившееся число в обратном порядке (так, чтобы младшие байты располагались по меньшим адресам) и, наконец, осуществляем контекстный поиск по свободному пространству.

Отличить блок косвенной адресации от всех остальных типов данных очень легко — он представляет собой массив указателей на блоки, а в конце идут нули. Остается только извлечь эти блоки с диска и записать их в файл, обрезая его по нужной длине.

> ***Внимание!***
>
> Если вы нашли несколько "кандидатов" на роль блоков косвенной адресации, это означает, что 13-й блок удаленного файла в разное время принадлежал различным файлам (а так, скорее всего, и будет). Не все косвенные блоки были затерты, поэтому принадлежавшие им ссылки остались неизменными.

Как отличить "наш" блок от "чужих"? Если хотя бы одна из ссылок указывает на уже занятый блок данных (что легко определить по карте), такой блок можно сразу откинуть. Оставшиеся блоки перебирают вручную до получения работоспособной копии файла. Имя файла (если оно еще не затерто) можно извлечь из каталога. Естественно, при восстановлении нескольких файлов невозможно однозначно определить, какое из имен какому файлу принадлежит. Тем не менее это все же лучше, чем совсем ничего. Каталоги восстанавливаются точно так же, как и обыкновенные файлы, хотя, по правде говоря, в них кроме имен файлов восстанавливать больше нечего.

Описанный метод восстановления данных не свободен от ряда ограничений. В частности, при удалении большого количества сильно фрагментированных двоичных файлов он, скорее всего, не сработает. Вы только убьете свое время, но вряд ли найдете среди "обломков" файловой системы хоть что-то полезное. Тем не менее если у вас отсутствует резервная копия, то другого выхода просто нет, так что данная методика все-таки не совсем бесполезна.

Оптимизация производительности файловой системы

В отличие от Windows, Linux поддерживает целый спектр файловых систем различного типа и назначения: minix, ext2/3/4, ReiserFS/Reiser4, XFS, JFS, UFS, FFS... Какую файловую систему выбрать? Как правильно ее настроить? Стандартный выбор, предлагаемый составителями дистрибутива по умолчанию, не всегда оптимален. Как правило, быстродействие системы можно значительно улучшить.

Жесткий диск — надежное и быстрое устройство. Но процессор еще быстрее! И дисковая подсистема, несмотря на все усилия инженеров, по-прежнему остается слабейшим звеном, сдерживающим быстродействие всего компьютера в целом. А ведь объемы обрабатываемых данных все растут и растут.

Большинство материнских плат, выпущенных после 2000 года, несут на своем борту интегрированный RAID-контроллер, поддерживающий режимы RAID-0 ("stripe" mode — режим чередования, при котором данные пишутся на несколько жестких дисков сразу) и RAID-1 ("mirror" mode — зеркальный режим, при котором жесткие диски дублируют друг друга). Режим чередования значительно повышает производительность — два диска работают приблизительно в 1,5 раза быстрее, а четыре — примерно в 3,5 раза быстрее, чем один.

Обладатели ядер современных версий могут использовать программные реализации RAID (software RAID), практически не уступающие по скорости аппаратным реализациям. Стоит, правда, отметить, что программные RAID несколько повышают нагрузку на процессор. Более подробную информацию по данному вопросу можно найти здесь: **https://raid.wiki.kernel.org/index.php/Linux_Raid**.

Проблема развертывания программных RAID в том, что типичная материнская плата имеет всего два-четыре порта SATA. При этом помимо самих дисков иногда требуется еще как минимум один оптический привод! Для достижения наивысшей

скорости приходится приобретать материнскую плату, оснащенную бóльшим количеством портов SATA. Что поделаешь — оптимизация требует жертв! Результаты тестирования производительности различных вариантов реализации программных RAID, с которыми более подробно можно ознакомиться на страничке **https://interface31.ru/tech_it/2010/04/byudzhetnyj-raid-testiruem-proizvoditel-nost.html**, приведены на рис. 9.15.

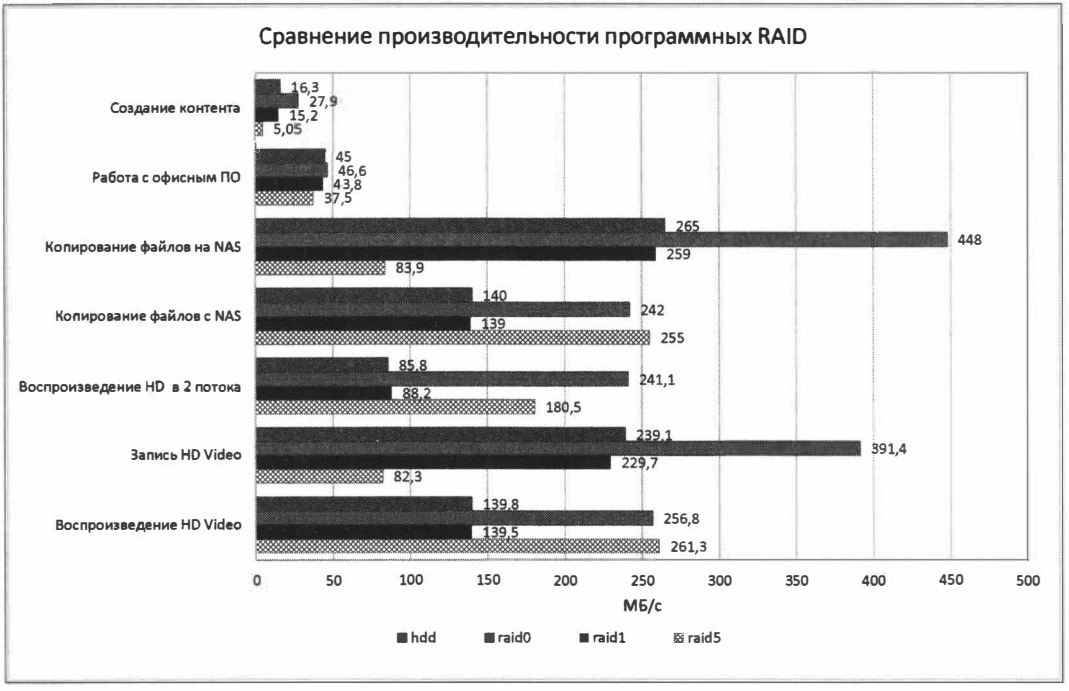


Рис. 9.15. Результаты тестов производительности разных реализаций RAID

Ряд встроенных RAID-контроллеров позволяет работать сразу с четырьмя, а иногда даже больше, SATA-накопителями. Например, такое поддерживает южный мост Intel ICH7 и новее, а AMD SB750, SB950 и некоторые другие модели этой линейки предлагают сразу шесть SATA-портов. При этом приходится покупать мощный блок питания и ставить специальные фильтры на разветвитель, чтобы подавлять помехи, к которым жесткие диски весьма чувствительны. Но выигрыш в скорости стоит того, особенно если компьютер используется для занятий видеомонтажом или обработки изображений полиграфического качества. Но с такими потребностями лучше сразу обратится к дискам SAS. Мы же остановимся на SATA как на самом демократичном и дешевом интерфейсе.

Настройка производительности с помощью утилиты hdparm

Для достижения наивысшей производительности каждый жесткий диск, установленный в систему, должен быть настроен в соответствии со своим предназначени-

ем. Стандартные настройки, принимаемые ядром по умолчанию, ориентированы на абстрактного среднестатистического пользователя и редко совпадают с конкретными требованиями. Учет преобладающего типа запросов к дисковой подсистеме значительно повышает быстродействие (в некоторых случаях — чуть ли не на порядок), хотя это оружие работает и в обратном направлении. Бестолковая настройка сваливает производительность в глубокую яму, из которой, впрочем, всегда можно выбраться, восстановив настройки по умолчанию.

Всем этим ведает консольная утилита hdparm, входящая в комплект штатной поставки большинства (если не всех) дистрибутивов Linux и требующая для своей работы полномочий администратора (root). Если вдруг этой утилиты в составе вашего дистрибутива не окажется, взять ее можно здесь: **https://sourceforge.net/projects/hdparm/**. Формат ее вызова следующий:

```
hdparm опция1 опция2 ... опцияN /dev/жесткий_диск
```

Жестким дискам с интерфейсом IDE обычно присваиваются имена `hda` (первый жесткий диск), `hdb` (второй жесткий диск), `hdc` и т. д. Диски SATA соответственно именуются `sda`, `sdb`, `sdc`. Строго говоря, hdparm настраивает параметры не одного лишь жесткого диска или SSD, но и его контроллера и отчасти драйвера. Рассмотрим несколько практических примеров.

Ключ `-a` устанавливает количество секторов *опережающего чтения* (read-ahead), которые будут автоматически прочитаны контроллером в надежде на то, что они все-таки пригодятся пользователю. По умолчанию ядро читает 8 секторов (4 Кбайт). При последовательном чтении больших слабо фрагментированных файлов это значение рекомендуется увеличить в несколько раз, а при хаотичном доступе, работе с мелкими или сильно фрагментированными файлами — уменьшить до 1–2 секторов.

Ключ `-m` указывает количество секторов, обрабатываемых приводом за одну операцию обмена (так называемый multiple sector I/O или block mode). В зависимости от конструктивных особенностей жесткого диска он может обрабатывать от 2 до 64 (и больше) секторов за операцию. Конкретное значение можно узнать с помощью ключа `-i` (оно находится в графе `MaxMultSect`). В целом скорость обработки данных прямо пропорциональна количеству секторов, однако некоторые приводы (например, WD Caviar) при больших значениях `-m` начинают жутко "тормозить". Выяснить практическое положение дел помогает ключ `-t`, измеряющий пропускную способность дисковой подсистемы в режиме чтения.

> ***Внимание!***
>
> Запредельные значения `-m` могут привести к повреждению данных. Рисковать, экспериментируя с этим параметром, без необходимости не рекомендуется!

Ключ `-M` отвечает за настройку шумовых характеристик накопителя (Automatic Acoustic Management, AAM). Значение 128 соответствует наиболее тихому режиму, 254 — наиболее быстрому. Промежуточные значения в общем случае не определены (некоторые накопители их поддерживают, некоторые нет). Следует сказать, что значение 128 не только уменьшает шум, но и способствует меньшему износу нако-

пителя, однако падение производительности может быть очень и очень значительным, поэтому трудно посоветовать, какое именно значение следует выбрать.

Ключ `-c` управляет режимом передачи данных. Параметр 0 — 16-битовая передача, 1 — 32-битовая передача, 3 — 32-битовая передача со специальным синхросигналом. По умолчанию ядро использует параметр 3 (возможно, не для всех ядер) как наиболее надежный, но и менее производительный, чем 1. Большинство современных чипсетов вполне нормально работают с параметром 1, так что излишняя осторожность тут ни к чему.

Узнать информацию об устройстве, в том числе список поддерживаемых им режимов, можно с помощью ключа `-i` или чуть более подробно и точно с ключом `-I` (рис. 9.16). По умолчанию ядро выбирает не слишком агрессивные режимы передачи данных, оставляя солидный запас производительности "за спиной".

> ***Внимание!***
>
> Не забывайте обязательно зарезервировать содержимое томов перед началом экспериментов с `hdparm`!

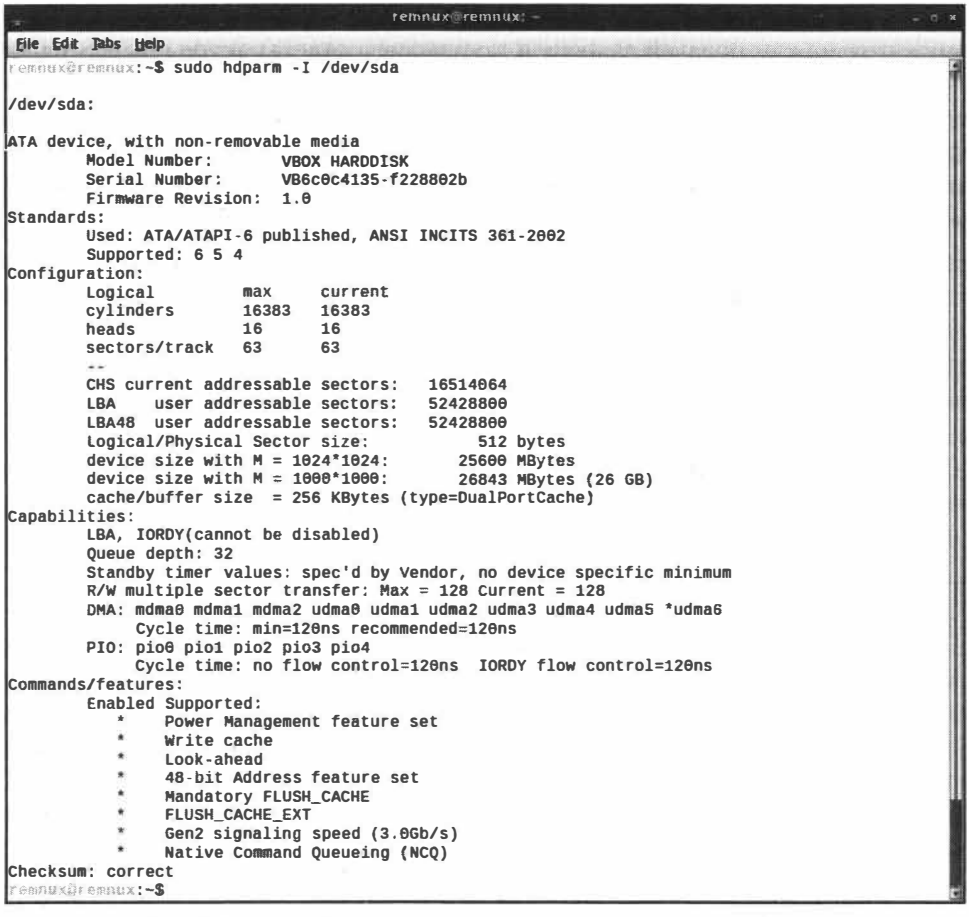


Рис. 9.16. Подробная информация о носителе, полученная hdparm

Для сохранения установок необходимо подать команду `hdparm -k 1 /dev/hxx`, в противном случае они будут утеряны при первом же сбросе контроллера SATA/IDE или перезапуске машины.

Выбор файловой системы

Существуют два типа файловых систем — журналируемые (journaling) и нет. К первым относятся ext3/4fs, ReiserFS/Reiser4, XFS, а ко вторым — minix, ext2fs и UFS. Журналируемые файловые системы намного легче переносят "зависание" системы и отключение питания во время интенсивных дисковых операций, автоматически возвращая файловую систему в стабильное состояние. Однако от других типов разрушений (отказ контроллера, дефекты поверхности, вирусное нашествие) журналирование уже никак не спасает, а вот производительность падает изрядно.

Для домашних компьютеров и большинства рабочих станций журналирование не нужно, хотя в большинстве современных дистрибутивах по умолчанию используется ext4 с включенным журналированием или ReiserFS. По тестам ReiserFS в среднем вдвое быстрее, чем ext3fs, что особенно хорошо заметно на небольших файлах. В реальности же часто все бывает наоборот. Высокая латентность ReiserFS (промежуток между подачей запроса и получением ответа) вкупе с агрессивной загрузкой процессора приводит к заметному отставанию от ext3fs, что бывает особенно заметно на малых файлах (да-да, на тех самых, на которых нам обещали выигрыш!). Подробнее с тестами производительности разных файловых систем Linux можно ознакомиться здесь: **https://www.phoronix.com/scan.php?page=article&item=linux_2638_large&num=1**.

Журналирование можно значительно ускорить, если разместить журнал на отдельном носителе. Такой журнал называется внешним (external). Подключить его можно командной строкой следующего вида: `tune2fs -J device=`*`external_journal`* (где *`external_journal`* — имя раздела соответствующего устройства), причем внешний журнал должен быть предварительно создан командой `mke2fs -O journal_dev external_journal`. Команда `tune2fs -J size=`*`journal_size`* управляет размером журнала. Чем меньше размер журнала, тем ниже производительность. Предельно допустимый размер составляет 102 400 блоков, или ~25 Мбайт (точное значение зависит от размера блока, о котором мы еще поговорим).

По умолчанию ext3 и ext4 журналируют только метаданные (т. е. служебные данные файла, например такие, как inode), записывая их на диск только после того, как будет обновлен журнал. Для увеличения быстродействия можно задействовать "разупорядоченный" режим, в котором метаданные записываются одновременно с обновлением журнала, что соответствует команде: `mount /dev/hdx /data -o data=writeback`. Естественно, надежность файловой системы при этом снижается. При желании можно журналировать все данные (команда `mount /dev/hdx /data -o data= journal`), после чего никакие "зависания" или сбои питания нам будут не страшны, правда, о производительности придется забыть.

При создании новой файловой системы важно выбрать правильный размер блока (в терминологии MS-DOS/Windows — кластера). На ext3fs и ext4fs это осуществля-

ется командой `mke2fs -b block-size`, на XFS — `mkfs.xfs -b size=block-size` и на UFS — `newfs -b block-size`. Чем больше блок, тем ниже фрагментация, но и выше дисковые потери за счет грануляции дискового пространства. Некоторые файловые системы (например, UFS) поддерживают фрагменты (fragments) — порции данных внутри блоков, позволяющие задействовать свободное пространство в "хвостах" блоков, благодаря чему использование блоков большого размера уже не приводит ни к каким потерям. Файловая система ReiserFS, в отличие от остальных, не "нарезает диск на ломтики" фиксированного размера, а динамически выделяет требуемый блок данных, "забивая" диск файлами под завязку. В среднем это на 6% увеличивает доступный объем, однако приводит к чрезмерной фрагментации, "съедающей" всю производительность. Рекомендуется использовать максимально доступный размер блока (4 Кбайт для ext2fs и ext3fs, 16 Кбайт для UFS и 64 Кбайт для XFS, файловые системы ReiserFS и JFS не поддерживают этой опции) и задействовать максимальное количество фрагментов на блок (в UFS — 8).

Другая важная опция определяет режим хеширования каталогов. Для ускорения работы с каталогами, содержащими большое количество файлов и подкаталогов, каталог должен быть организован в виде двоичного дерева. В ext3fs и ext4fs это осуществляется командой `mke2fs -O dir_index`, а в ReiserFS — `mkreiserfs -h` *`HASH`*, где *`HASH`* — один из следующих типов хеш-таблицы: `r5`, `rupasov` или `tea`. По умолчанию выбирается тип `r5`, который наилучшим образом подходит для большинства файловых операций. Тем не менее некоторые приложения (например, Squid Web Proxy-сервер) настоятельно рекомендуют использовать `rupasov`-хеш, в противном случае за быстродействие никто не ручается. С другой стороны, `r5` и `rupasov` очень медленно работают с каталогами, содержащими по нескольку миллионов файлов, и здесь лучше подходит `tea`, а на каталогах из нескольких десятков файлов все три алгоритма хеширования проигрывают стандартному нехешируемому `plain`-алгоритму. К сожалению, опция хеширования носит глобальный характер — нельзя одни каталоги хешировать, а другие — нет.

Файловая система XFS — одна из тех, что позволяют задавать размер inode вручную. Обычно в inode хранятся служебные данные файла (атрибуты, порядок размещения блоков на диске), но если файл целиком помещается в inode, то система сохраняет его именно там! Дополнительное дисковое пространство уже не выделяется, что избавляет головку винчестера от лишних перемещений, в результате чего время доступа к файлу существенно сокращается. Точно так же поступают ReiserFS, NTFS и некоторые другие файловые системы, однако размер inode они менять не в состоянии, а жаль! Если мы планируем работать с большим количеством небольших файлов, размер inode желательно увеличить, что положительно скажется как на производительности, так и на доступном дисковом пространстве. При работе с большими файлами размер inode лучше, наоборот, сократить, в противном случае потери дискового пространства будут довольно значительными. Выбор предпочтительного размера inode осуществляется командой следующего вида: `mkfs.xfs -i size=value`. Минимальный размер составляет 512 байтов, максимальный — 2048.

Внимание!

Windows предоставляет минимум рычагов управления для настройки дисковой подсистемы, и угробить свои данные под ее управлением довольно затруднительно. Linux же позволяет "крутить" и настраивать все и вся! Как следствие — малейшая оплошность приводит к катастрофическим разрушениям. И винить в этом некого — нечего было браться за штурвал, не выучив руководство, как правило, написанное на английском языке. Но даже хорошо составленное руководство не поможет определить, какие именно режимы поддерживаются вашим оборудованием, а какие — нет. Вполне может оказаться и так, что у вас кабель перекручен или разъем барахлит, а на высокоскоростных режимах это сразу же скажется! Настройка дисковой подсистемы на максимальную производительность — это огромный риск! Никогда не экспериментируйте, не зарезервировав всех данных!

Фрагментация

В процессе работы с диском его фрагментация неизбежно увеличивается. Больше всего от этого страдают ext2fs/ext3fs и ReiserFS. Ext4fs уже использует экстенты и внутренние механизмы, уменьшающие фрагментацию файлов. На UFS и XFS за счет поддержи блоков большого размера падение производительности уже не так заметно. Утверждение, что файловые системы Linux якобы не подвержены фрагментации — нелепый миф, который легко опровергнет любой опытный пользователь.

При последовательной записи на диск нескольких файлов система их размещает один за другим, так что первый файл "упирается" во второй. Свободного места для дальнейшего роста уже нет (короткий "хвост" в конце блока не считается), и система вынуждена выделять блоки где-то за концом следующего файла. Если же их там нет, свободные блоки ищутся в начале диска. В результате этого файл оказывается "размазанным" по поверхности диска. Рассмотрим еще один сценарий. Представьте себе, что вы записали пять файлов по 100 блоков каждый, а затем удалили первый, третий и пятый файлы. Таким образом, вы освободите 300 блоков в трех фрагментах. При записи 300-блочного файла система сначала попытается отыскать непрерывный участок свободного пространства подходящего размера, но если его не окажется, будет вынуждена "размазывать" файл по поверхности. Чтобы исправить ситуацию, необходимо собрать все свободные блоки, объединив их в один непрерывный фрагмент, т. е. дефрагментировать раздел.

С нашей точки зрения, из бесплатных дефрагментаторов лучшим является стандартный e4defrag, входящий в штатный комплект поставки большинства дистрибутивов Linux. Он уменьшает фрагментацию экстентов, делая их более "протяженными". Интересное отличие его от стандартного дефрагментатора Windows в том, что с помощью e4defrag можно дефрагментировать не только весь том, но и лишь отдельные файлы и папки на нем.

Фирма O&O Software в начале 2000-х наряду с одноименным дефрагментатором для Windows NT выпустила замечательный консольный дефрагментатор для Linux. К сожалению или нет, за рамки первой версии он не вырос, и его поддержка прекратилась аж в далеком 2006-м. В настоящее время существует проект udefrag

(**https://jp-andre.pagesperso-orange.fr/advanced-ntfs-3g.html**) — Linux-версия утилиты UltraDefrag, работающая с драйвером NTFS-3G. В общем, надежных дефрагментаторов для неродных файловых систем Linux вы, скорее всего, не найдете.

В общем, регулярная дефрагментация — это хороший способ противостоять растущему падению производительности файловой системы (а также заметно упрощает процесс восстановления данных на "полетевших" винчестерах, если, конечно, до этого дойдет. Но ведь вы-то сделали резервные копии?). Исследований в этом направлении не так много, как хотелось бы. "*Journaling-Filesystem Fragmentation Project*" — одна из исследовательских работ по фрагментации файловых систем и ее влиянию на производительность (на английском языке). К сожалению, кроме как в архиве ее уже не найти: **https://web.archive.org/web/20110517213612/http://www.informatik.uni-frankfurt.de/~loizides/reiserfs/agesystem.html**. Архив соответствующей рассылки можно найти здесь: **https://www.mail-archive.com/loizides@isg.de/**.

Обновлять или не обновлять

Некоторые приложения, в частности, уже упомянутый Squid Web Proxy-сервер, требуют особой настройки файловой системы. Для увеличения быстродействия рекомендуется отключить обновления времени последнего доступа к файлу с помощью команды `mount -o noatime`. Наибольший прирост производительности наблюдается на UFS, которая, в отличие от подавляющего большинства остальных файловых систем, не откладывает обновление inode в долгий ящик (lazy write), а делает это сразу же после его изменения (write through). На ext3/4fs в силу ее журналирующей природы обновление `atime` вносит столь незначительный вклад в общее быстродействие, что никакой разницы просто нет. Впрочем, подобные рекомендации могут встречаться, призванные увеличить срок жизни SSD-накопителей.

Проблема "хвостов"

По умолчанию ReiserFS сохраняет короткие файлы (и файловые хвосты) на листьях двоичных деревьев. В целом это многократно повышает производительность, особенно если свободное дисковое пространство далеко от исчерпания (рис. 9.17 и 9.18). Тем не менее при работе с некоторыми приложениями хвосты лучше отключить. При работе с огромным количеством малых файлов, которые постепенно растут, системе приходится перестраивать множество структур данных, "гоняя" растущие хвосты между блоками и деревьями, в результате чего производительность становится недопустимо низкой. Команда `mount -o notail` отключает "паковку" хвостов и коротких файлов. Повторное монтирование с настройками по умолчанию вновь активизирует эту опцию, но при этом уже "упакованные/распакованные" хвосты останутся на своем месте вплоть до модификации "своего" файла.

> ***Внимание!***
>
> Помните, что `mke2fs` — это деструктивная команда, разрушающая всю файловую систему целиком! Грубо говоря, это format.com под Linux.

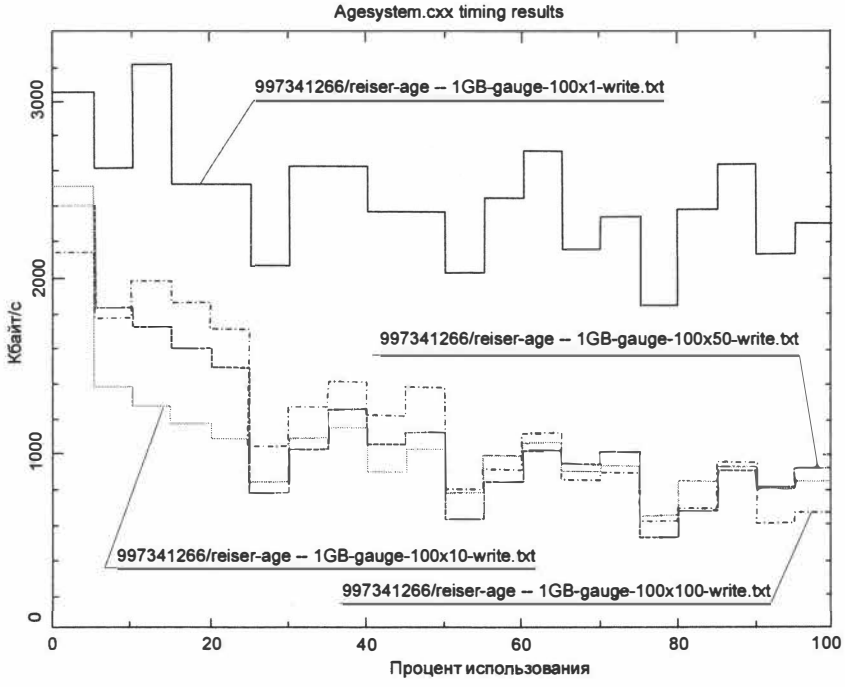


Рис. 9.17. Производительность файловой системы ReiserFS на операциях записи в зависимости от объема свободного пространства (паковка хвостов включена)

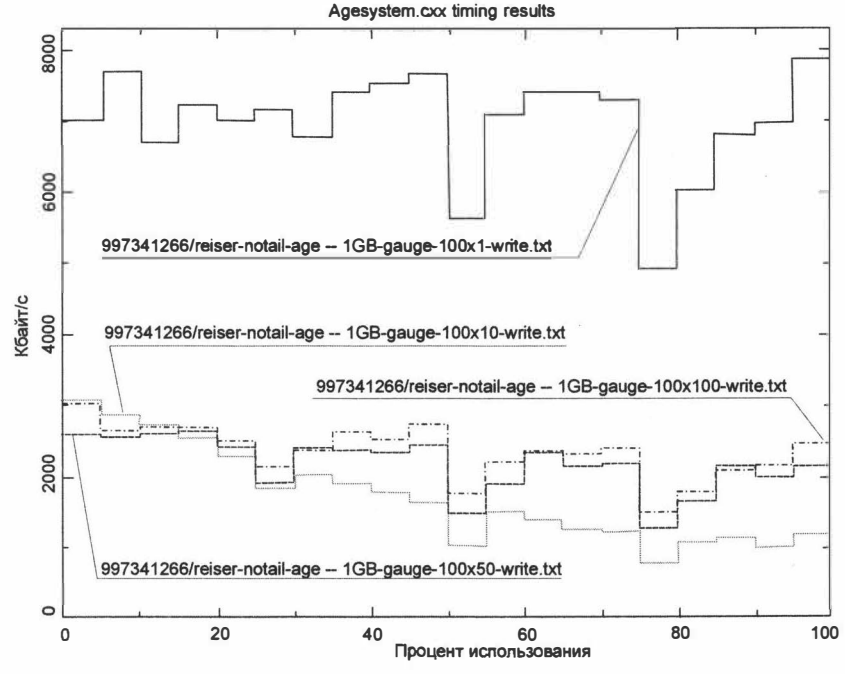


Рис. 9.18. Производительность файловой системы ReiserFS на операциях записи в зависимости от объема свободного пространства (паковка хвостов выключена)

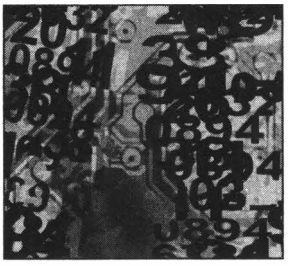
Полезные ссылки

- Справочник по созданию программных RAID'ов под Linux (на английском языке): **https://raid.wiki.kernel.org/index.php/Linux_Raid**.
- *"Тонкая настройка IDE дисков в Linux с помощью hdparm"* — отличная статья на русском языке. Доступна здесь: **http://www.opennet.ru/base/sys/htparm_tune.txt.html**.
- *"JFS for Linux"* — домашняя страничка проекта JFS. Содержит исходные тексты, документацию, технологию и т. д. (на английском языке): **http://jfs.sourceforge.net/**.
- *"Reiser4 file system for Linux OS"* — домашняя страничка проекта Reiser4 под Linux (на английском языке): **https://sourceforge.net/projects/reiser4/**.
- *Reiser4 wiki* — документация ядра, посвященная файловой системе reiser4 (на английском языке): **https://reiser4.wiki.kernel.org/index.php/Main_Page**.
- *"Работа с дисками и файловыми системами в FreeBSD"* — отличный faq на русском языке: **http://www3.opennet.ru/base/sys/freebsd_fs_mount.txt.html**.
- *"Understanding Filesystem Performance for Data Mining Applications"* — сравнение производительности различных файловых систем под Linux с советами по их "тонкой" настройке (на английском языке): **http://www.cs.rpi.edu/~szymansk/papers/hpdm03.pdf**.
- "*Real World Benchmarks Of The EXT4 File-System* " — еще одна статья по сравнению производительности файловых систем под Linux (на английском языке): **https://www.phoronix.com/scan.php?page=article&item=ext4_benchmarks**.
- *"Performance of Large Journaling File Systems"* — журналируемые файловые системы и их производительность (на английском языке): **http://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/journalingfilesystem/pubjournal.pdf**.
- *"Как устроена файловая система reiser4"* — название статьи говорит само за себя: **http://samag.ru/archive/article/653**.
- *"Analysis and Evolution of Journaling File Systems"* — еще одно сравнение журналируемых ФС (по большей части ext3fs и ReiserFS, на английском языке) **https://www.usenix.org/legacy/publications/library/proceedings/usenix05/tech/general/full_papers/prabhakaran/prabhakaran_html/main.html**.
- *"Оптимизация файловой системы ext3-ext4"* — отличная статья про оптимизацию файловых систем под Linux (на английском языке): **https://wiki.colobridge.net/полезное/советы/оптимизация_файловой_системы_ext3-ext4**.
- *"Ext2/3/4 file system utilities"* — исходные тексты пакета утилит e2fsprogs, где также можно найти код стандартного дефрагментатора ext4 (в ветке misc/e4defrag.c): **https://github.com/tytso/e2fsprogs**.

ЧАСТЬ III

Восстановление поврежденных носителей резервных копий

ГЛАВА 10



Восстановление данных с носителей остальных типов

Резервная копия — это последний рубеж обороны против беспощадной энтропии, но иногда случается так, что гибнет и она. Существует множество фирм, занимающихся восстановлением данных за деньги, но далеко не всегда они их восстанавливают.

В этой главе будет приведена обзорная информация по восстановлению данных с различных носителей, традиционно использующихся для хранения резервных копий.

Кого трогает чужое горе? К этим людям уж точно нельзя причислить специалистов из сервисных центров. Они просто делают свою работу, т. е. зарабатывают деньги с наименьшими телодвижениями. А по-другому и не получится. Рынок! Если принимать близко к сердцу чужие проблемы, то через месяц работы можно слечь с инфарктом. Бесспорно, у специалистов есть опыт, оборудование и все прочие составляющие, необходимые для успешного восстановления данных. Неквалифицированные попытки "самолечения" в девяти случаях из десяти заканчиваются полным провалом и необратимым уничтожением тех данных, которые еще можно было бы спасти. Тем не менее обращение к специалистам далеко не всегда оправдано. Это особенно справедливо, если речь идет о конфиденциальной информации.

В некоторых случаях данные можно восстановить и самостоятельно. В основном мы будем далее говорить о физическом разрушении носителей резервных копий (царапины, дефекты поверхности), не касаясь вопросов восстановления ошибочно удаленных файлов или непреднамеренного форматирования раздела.

Оптические носители

Начнем с восстановления носителей CD/DVD. Хотя в последнее время их долговечность вызывает все больше вопросов, они по-прежнему иногда используются для хранения архивов. Производители наперебой уверяют потребителей в исключительной надежности своей продукции. Но при этом диски мрут как мухи, зачастую выдерживая всего лишь один сезон. Сотрудники тестовой лаборатории датского отделения журнала PC-Active провели свое собственное расследование. Отобрав

несколько "брендовых" разновидностей, они исследовали процессы деградации в активном слое и получили шокирующие результаты. На рис. 10.1 изображены фотографии лазерного диска, полученные с помощью специального оборудования. Слева представлен диск сразу после прожига, справа — тот же самый диск спустя 20 месяцев. Белый цвет обозначает идеальные сектора, светло-серый — сектора, в процессе чтения которых изредка возникают ошибки чтения, и, наконец, более темные оттенки соответствуют секторам, имеющим серьезные повреждения. Несмотря на то что внешне такой диск читается вполне нормально, поскольку корректирующие коды Рида — Соломона делают свое дело, с каждым днем он будет читаться все хуже.

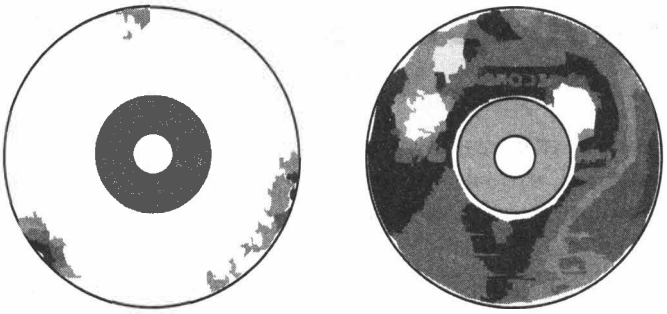


Рис. 10.1. Деградация активного слоя носителей CD-R с течением времени

Чтобы избежать неприятных сюрпризов, свой архив на оптических носителях следует тестировать, по крайней мере, раз в шесть месяцев. Для этого подойдет любая программа (например, NERO CD-DVD Speed или ее реинкарнация Opti Drive Control). Если такой программы под рукой нет, то качество носителя можно приблизительно оценить по звуку, издаваемому приводом. Если диск читается на полной скорости без характерных повторов и сброса оборотов — с ним все в порядке. В противном случае данные необходимо как можно скорее переписать на свежий носитель.

А что делать, если вы спохватились уже после того, как диск перестал читаться? Самое простое — затормозить привод до скорости 4x–8x (если, конечно, он это позволяет) и повторить попытку еще раз. Существует множество "тормозящих" утилит, например, разработанная Крисом Касперски программа xCD, которую можно найти в электронном архиве, размещенном на сайте издательства БХВ. К сожалению, не все приводы поддерживают программное изменение скорости и далеко не все читают "проблемные" диски, так что тут придется поэкспериментировать. Попробуйте прочитать диск у приятелей или зайдите в ближайшую фирму и попросите продавца "протестировать" привод перед его "покупкой". Впрочем, шансы на успешный исход дела не так уж и велики. И что тогда?

Практика показывает, что существует всего две основные причины, по которым диски перестают читаться: царапины и дегенеративные процессы в активном слое. Ну с царапинами мы еще разберемся, а что делать с активным слоем, контрастность которого необратимо снижается со временем? Проблему можно решить, повысив мощность лазерного излучателя. Прием варварский, но других путей, по-

видимому, просто нет. Лазеру, конечно, приходится туго, и долго в таком режиме он не проработает. Однако прежде чем окончательно отказать, он может успеть кое-что прочитать. Приводы прошлого поколения содержали подстроечные резисторы, регулируемые любой отверткой, но сейчас их заменила электроника. Яркость лазерного луча настраивается автоматически, и чтобы ее изменить, необходимо пропатчить прошивку (а это не каждому по плечу). Как вариант — можно найти на плате постоянный резистор, ведущий к излучателю, и припаять параллельно ему еще один, уменьшая эффективное сопротивление в 1,5–2 раза. Естественно, к этой мере следует прибегать только в тех случаях, когда на диске оказались действительно важные данные, стоимость которых сопоставима с ценой привода.

Теперь о царапинах. Даже глубокие борозды — это еще не приговор. Некоторые источники рекомендуют отполировать диск зубным порошком (который сейчас трудно, но все же можно найти в продаже) или специальной шлифовальной пастой типа ГОИ. Все это правильно, и такая методика отлично работает, но тут есть два маленьких "но". Во-первых, с первой попытки отполировать диск не удастся. Тут навык необходим! А чтобы его получить, требуется затратить уйму времени, которое не у всех есть. Во-вторых, глубокие царапины просто так не зашлифуешь, а ведь именно они — источник всех бед! Нет, мы пойдем другим путем. Возьмем зеленку (ту самую, что продают в аптеках) и аккуратно закрасим царапины зубочисткой или остро заточенной спичкой. Это предотвратит рассеивание света, а для лазерного луча зеленка прозрачна!

Хуже, если диск раскололся на несколько частей. Можно ли спасти хотя бы часть данных? Некоторые фирмы, специализирующиеся на восстановлении, используют электронные микроскопы, фотографирующие спиральную дорожку. Далее они проводят компьютерную обработку собранной информации, буквально по байтам восстанавливая утраченные файлы. Это довольно кропотливое и весьма дорогостоящее занятие, которое по карману только крупным компаниям, потерявшим судьбоносные данные. В домашних условиях обычно используется двусторонний строительный скотч и пустая болванка, к которой приклеиваются обломки диска, после чего эта конструкция аккуратно вставляется в привод, работающий на скорости 1х–2х. Конечно, для чтения потребуется специальное программное обеспечение (которое, в частности, можно найти в электронном архиве к книге, размещенном на сайте издательства БХВ) и прочие ухищрения, но тем не менее какая-то часть информации все же прочитается. Попробуем рассчитать, какая же именно. Размер одного сектора составляет ~15 мм, для позиционирования головки привод должен декодировать субканальную информацию, для чего ему необходимо прочитать не менее 11 секторов. Следовательно, данная технология позволяет читать обломки с длинной дуги от ~17 см. Для внешней кромки это составляет чуть меньше половины лазерного диска, т. е. если диск разломать напополам, мы сможем прочесть лишь ту часть информации, что была записана на самом краю. Не слишком-то воодушевляющая перспектива, но это все-таки лучше, чем совсем ничего.

И еще один совет напоследок. Достаточно часто диск перестает читаться из-за неисправности привода. Качество современных приводов уже не то, что было лет двадцать назад, и лазеры погибают сплошь и рядом. Внешне это проявляется в том,

что привод становится все более и более привередливым, отказываясь "переваривать" диски, которые еще вчера нормально читались. Если не читается диск на внешнем приводе, подключаемом к компьютеру по USB, сперва убедитесь в качестве используемого провода — бывает, что неисправность исключительно в нем. В общем, столкнувшись с такой проблемой, не спешите винить диск и не стремитесь протирать его всем, что только подвернется под руку. Во-первых, прежде чем протирать любую оптическую поверхность, обязательно сдуйте пылинки, иначе вы неминуемо создадите новые царапины. Во-вторых, протирать диски следует влажными салфетками (например, предназначенными для чистки монитора), меняя их при каждом проходе. Сам проход нужно вести в радиальном направлении (от центра к краям), но ни в коем случае не вдоль окружности! Никакой мистики здесь нет. Просто корректирующие коды были изначально ориентированы на борьбу с радиальными царапинами. Концентрическим царапинам они, увы, противостоять не могут.

Магнитные ленты

Хотя большинство из нас едва ли столкнется с этим чудом техники и человеческой мысли, считая магнитные ленты безнадежно устаревшим носителем информации, на самом деле они по-прежнему живы-здоровы и используются крупными центрами обработки данных, компаниями и научными организациями, среди которых Google, CERN, оператор облачного сервиса Wazee Digital (ранее T3Media и Thought Equity), хранящий данные BBC, Paramount, Sony Pictures Entertainment, National Geographic, New York Times.... Почему в XXI веке все еще встречается технология, которой почти 70 лет? Ну хотя бы из-за низкой стоимости 1 Гбайт по сравнению с остальными носителями. Тем более в деле хранения огромных архивов и плотности записи им нет равных: относительно недавние разработки IBM позволяют хранить 201 ГБайт на одном квадратном дюйме магнитной ленты и примерно 330 ТБайт в картридже размером с ладонь! Посему устроим небольшой ликбез.

Картриджи для стримеров очень долговечны и крайне надежны. Обычно с ними не случается никаких проблем, но иногда лента все-таки рвется. Виновником может быть как "мстительный" стример, плохо сконструированный и собранный в подпольной фирме кустарным образом "из чего бог послал", так и сам человек. Очень многие из нас питают нездоровое влечение к магнитным лентам. Кто не пробовал их потеребить, поковырять ножиком или даже карандашом (конечно, из тех, кто видел их вживую)?

Хорошая новость! Порванную ленту можно склеить любым универсальным клеем. Хорошо подойдет например японский Super Glue, который сейчас продается в крошечных тюбиках на каждом углу. Вопреки распространенному мнению, потери информации при этом не происходит! Стримеры используют помехозащитные коды (разновидность циклических кодов Рида — Соломона) и безболезненно переносят значительные "выпадения" ленты, вплоть до 5 см (конкретные цифры варьируются от модели к модели).

Также приходится сталкиваться с заклиниванием картриджа. Обладатели кассетных магнитофонов знают, что это такое. Как с этим бороться? Чуть-чуть ослабляем крепежные болты (а большинство картриджей разборного типа), чтобы лента могла свободно вращаться, и несколько раз вручную перематываем ее туда и обратно. Перемотку нужно выполнять именно вручную, и стримеру это дело лучше не доверять. Затем затягиваем болты, и картридж возвращается в строй.

Дефектные стримеры при определенных обстоятельствах иногда мнут ленту, что уже значительно хуже. Измятая лента не прилегает к магнитной головке и читается с огромным количеством ошибок, с которыми корректирующие коды уже не справляются. Что тогда? К счастью, в отличие от кассетного магнитофона, в котором запись происходит перпендикулярно движению ленты, в стримере запись осуществляется под некоторым углом, отличным от 90 градусов. В результате этого влияние локальных дефектов значительно ослабляется. Чтобы прочитать измятую ленту, в девяти из десяти случаев достаточно увеличить ее прижим к головке (для этого подойдет небольшой кусочек поролона или другого упругого материала со скользким покрытием). Многократное вычитывание поврежденных участков дает неплохой результат, и значительная часть информации все же возвращается из небытия.

Некоторые люди пытаются разгладить ленту руками, ногтем или другим "инструментом". Этого делать нельзя!!! Лента вытягивается, и потому время ее чтения увеличивается, а стример рассчитан на строго определенную скорость, и изменение частоты сигнала создает дополнительную нагрузку на корректирующие коды, которым и без того приходится тяжело. Впрочем, это уже крайности, с которыми большинство пользователей стримеров никогда не встречается.

FLASH-память

Термин "FLASH-память" охватывает целый спектр устройств, на краю которого находятся мини-драйвы, по сути, представляющие собой миниатюрные жесткие диски. Естественно, к каждому типу устройств нужен индивидуальный подход, и принципы их восстановления различны.

USB FLASH-карты

Давайте рассмотрим тип FLASH-носителей, которые состоят из перепрограммируемой микросхемы энергонезависимой памяти и контроллера USB, т. к. он встречается чаще всего. Микросхема памяти довольно надежна и отказывает, прямо скажем, не очень часто, хотя для нее и характерен износ ячеек памяти. Что же касается контроллера USB, то он легко выводится из строя вездесущим статическим электричеством, да и вообще довольно-таки уязвим. Если только память и контроллер USB не интегрированы в единую микросхему, то контроллер легко перепаять. Достаточно купить еще один FLASH-носитель точно такой же или аналогичной модели. Естественно, для этого необходимо уметь держать паяльник в руках, иначе данные умрут окончательно. Современные микросхемы очень боятся перегрева, и стоит нам лишь чуть-чуть замешкаться, как они тут же гибнут бесповоротно.

Впрочем, аппаратные отказы FLASH-карт — это все-таки экзотика. Гораздо чаще приходится сталкиваться с логическими разрушениями, например, вроде ошибочно удаленных файлов или неполадок работы драйвера. Глюки — вот настоящая проблема. Из-за них погиб Mars Rover (**https://www.extremetech.com/56932-nasa-dos-glitch-nearly-killed-mars-rover**), да и вообще теряется огромное количество данных. Суть в том, что часть памяти зарезервирована под служебные нужды, но доступ к ней не заблокирован, поэтому программными средствами можно не только прочесть эту область, но и записать в нее все что угодно. Если драйвер по ошибке или злому умыслу затирает служебную область, доступ к FLASH-карте чаще всего становится невозможным. Мы не можем даже отформатировать ее, не говоря уже о том, чтобы считать данные. Во время, когда карты были дорогими, это становилось настоящим потрясением. Впрочем, всегда было можно найти устройство, которое игнорирует служебную область и работает с картой без нее, а это значит, что низкоуровневый доступ к FLASH-памяти все же работал! А раз так — можно считать все данные и самостоятельно декодировать их.

Восстановлением информации с FLASH-карт занимается множество утилит. Например, есть неплохая утилита PhotoRec (**https://www.cgsecurity.org/wiki/PhotoRec**) от создателя TestDisk, восстанавливающая файлы по их сигнатурам. И хотя она позиционируется как средство "спасения" цифровых фотографий, восстановление "обычных" данных проходит ничуть не хуже. Это свободный продукт, доступный всем желающим и под различные платформы.

Да здравствует FAT!

Раз уж мы говорим о флешках, стоит затронуть самую широко используемую для них файловую систему — FAT32. Когда вы покупаете FLASH-накопитель, будь то USB-флешка или SD-карта, скорее всего он окажется отформатирован в FAT32. Большинство многочисленных устройств, работающих с FLASH-картами (магнитолы, смартфоны, ТВ-приставки и плееры, цифровые рамки, камеры и квадрокоптеры...), обязательно поддерживают FAT32, чего об остальных файловых системах уже не скажешь.

В 1996 году FAT32 была представлена как наследник FAT16, которая, в свою очередь, продолжила дело FAT12 (а одним из создателей FAT12 в далеком 1977 году был и Билл Гейтс!). До FAT12 существовала 8-битовая таблица размещения файлов (8-bit FAT) для 8-дюймовых дискет и Microsoft BASIC-80 во времена господства 8080/8086 — вот с нее-то все и началось... Спустя столько времени FAT32 по-прежнему активно применяется, ведь никакая другая файловая система не поддерживается таким же широким спектром устройств и операционных систем. Имя этого семейства файловых систем — "таблица размещения файлов" (File Allocation Table) — полностью отражает их незамысловатое внутреннее устройство. Логическое продолжение FAT32 под названием exFAT (FAT64) не так широко распространено и является проприетарным.

Раздел FAT32 организован намного проще, чем NTFS. Под резервной областью понимается область тома от его начала до таблиц размещения файлов. В этой области

в загрузочном (нулевом) секторе тома располагается блок параметров BIOS со ссылкой на структуру FSInfo, которая, как правило, находится в логическом секторе 1. Загрузочный сектор завершается уже знакомой нам сигнатурой `55h AAh`. Далее идут две файловые таблицы — основная FAT1 и запасная FAT2, за которыми следует область данных (рис. 10.2).

Рис. 10.2. Структура тома FAT32

Структура FSInfo была введена в FAT32 для ускорения работы с разделом. Она хранит информацию о количестве свободных кластеров и номере первого свободного кластера в файловой таблице. Эта самая таблица определяет связи одного блока данных с другим. Кроме нее, в файловой системе необходим корневой каталог. В предыдущих версиях FAT его расположение на разделе было строго фиксировано, но в FAT32 он может, как любой другой каталог, находиться в любом месте области данных и иметь произвольный размер.

Итак, каждому кластеру раздела соответствует элемент таблицы FAT. В нем описано, свободен кластер или же занят под данные какого-либо файла. Если он занят, то указывается также адрес следующего кластера, содержащего элемент файла. Так образуется цепочка кластеров. Если кластер является последним в цепочке для конкретного файла, то вместо адреса следующего кластера в нем указывается признак конца файла (рис. 10.3). Первые два элемента таблицы зарезервированы под

Рис. 10.3. Пример кластерной цепочки в FAT32

служебную информацию. Числа 12 и 16 в названиях FAT12/16 означают разрядность адресов кластеров, но вот в FAT32 он, как это ни странно, на самом деле составляет 28 бит (хотя под адрес и выделено 32 бита, 4 старших бита игнорируются).

А как же понять, какому файлу какая кластерная цепочка соответствует? Для этого нам нужны каталоги, которые содержат необходимые сведения о входящих в них файлах. В FAT32 каталожные записи делятся на основные и вспомогательные. В основной 32-битовой записи находится краткое имя файла, размер в байтах (для каталогов равен нулю), дата и время создания, доступа и изменения, номер первого кластера файла и атрибуты. Вспомогательные записи используются для представления длинных имен файлов (до 255 символов Unicode), на основе которых автоматически генерируются краткие имена. Подкаталоги в FAT представляются как файлы. Все они, за исключением корневого каталога, содержат элементы "." и "..". Первый кластер корневого каталога указывается в загрузочном секторе FAT32. Обычно это кластер 2, поскольку кластеры 0 и 1 зарезервированы.

Первый байт краткого имени файла говорит о занятости элемента каталога. Если он равен `E5h`, то данный элемент свободен и может быть использован для создания нового файла. Значение `00h` свидетельствует о том, что этот элемент свободен и после него нет ни одного занятого элемента. Соответственно, при удалении файла или каталога в FAT32 первый байт его записи устанавливается в `E5h`. Найдя удаленную запись, попробуем приступить к восстановлению содержимого. Доподлинно восстановить цепочку кластеров удаленного файла получится далеко не всегда, но можно попытаться прочесть данные из начального кластера файла, который указан в его каталожной записи, читая при этом только из кластеров, отмеченных как свободные. Этим методом при благоприятном раскладе (если участки файла еще не были перезаписаны) можно восстанавливать даже некоторые фрагментированные файлы.

Твердотельные накопители

За последние годы в массы прочно вошло еще одно устройство на основе FLASH-памяти — твердотельные накопители (Solid State Drive, SSD). Можно было бы сказать, что SSD — это всего лишь большая флешка, да только их логическое устройство несколько сложнее, что не позволяет восстанавливать с них данные такими же способами, как с простых FLASH-карт.

SSD состоит из массива микросхем памяти и контроллера. При этом совсем не обязательно, что один файл будет записан целиком в одну микросхему. То, на сколько частей он окажется поделен и в какие микросхемы попадет, — как говорится, одному богу известно. Если точнее, накопитель старается равномерно распределить нагрузку по всем своим ячейкам и уравнивать их износ, чтобы продлить их жизнь. Поэтому, чтобы прочитать содержимое SSD, необходим *транслятор* (Flash Translation Layer, FTL), преобразующий логические адреса (LBA) в физические. Кроме этого, данные транслятора используются механизмом сборки "мусора" (подробнее узнать об устройстве транслятора можно в третьей части цикла статей "Coding for SSD", см. список ресурсов в конце раздела). Обычно таблица трансля-

ции находится в служебной области все той же FLASH-памяти. Для улучшения быстродействия и уменьшения износа ячеек (а таблица трансляции может меняться очень часто) рабочая копия таблицы при включении SSD помещается в DRAM-буфер. Существует также технология Host Memory Buffer (HMB), когда таблица трансляции хранится не в DRAM-буфере, а в ОЗУ хостовой машины. В любом случае, при повреждении этой таблицы восстановить данные без специализированного оборудования не представляется возможным — слишком уж трудоемко собирать линейный дамп из разрозненных "осколков" информации...

Тут возможны два пути. В любом случае придется воссоздавать виртуальный транслятор. Если диск относительно старый, без аппаратного шифрования, а механизмы трансляции в нем просты, то осуществляется выпаивание чипов памяти (при этом важно их не перегреть!), чтение микросхем памяти при помощи PC3000 и сборка образа накопителя. Ясно, что после спасения данных использовать накопитель по назначению уже вряд ли получится, но зато можно попробовать дать вторую жизнь микросхемам памяти и сделать из них FLASH-карты. Если же испорчен транслятор на более современном SSD, применяющем аппаратное шифрование и "навороченные" алгоритмы трансляции, восстановить с него данные можно только переводом накопителя в технологический режим и построением виртуального транслятора при помощи комплекса PC3000. Но, к сожалению, воссоздать транслятор получится лишь при условии, что алгоритм трансляции и шифрования данного SSD изучен, что само по себе является трудоемкой задачей. Стоить такая операция будет несколько десятков тысяч рублей. В общем, восстанавливать данные с неисправных винчестеров не в пример дешевле (не считая отдельных особо сложных случаев HDD). Более подробно ознакомиться с процессом и статистикой восстановления данных с SSD можно в записи блога ACELab по адресу **https://habr.com/ru/company/acelab/blog/256895/**.

Стоит заметить еще одну важную особенность твердотельных накопителей, отличающую их от привычных винчестеров. Подобно тому как в HDD информация пишется секторами по 512 байтов, в SSD запись отдельного байта также невозможна. Во-первых, минимальный элемент хранения данных в SSD — ячейка. В настоящий момент она может хранить от одного до четырех битов в зависимости от типа — SLC, MLC, TLC или QLC. Ячейки объединяются в страницы (обычно по 4 Кбайт), а те, в свою очередь, в блоки. То есть, когда нам нужно считать какой-либо байт, будет считана целая страница. Записываются же данные в SSD блоками, размер которых обычно в пределах 64–512 страниц. Если в одной странице нужно изменить один-единственный байт, то придется писать целый новый блок! В итоге фактически на носитель пишется куда больше информации, чем необходимо на первый взгляд. Это явление получило название *усиление записи* (Write Amplification). Ну а во-вторых, записывать данные в SSD можно только в чистый блок. Если в винчестерах биты просто пишутся поверх старой информации, то в SSD содержимое ячейки нужно электрически стереть, прежде чем его можно будет перезаписать. Когда накапливается много блоков, заполненных не целиком, начинает "проседать" производительность SSD (причем на некоторых контроллерах, среди которых печально известный SandForce-2281, необратимо), потому что блок нужно скопиро-

вать в кэш, стереть, дополнить его новой информацией и записать обратно в память. Вам встречались SSD странных объемов 120 или 240 Гбайт? На самом-то деле в них, как и положено, 128 и 256 Гбайт, но часть объема отдается под резервную область, не доступную пользователю. Благодаря этой области у накопителя всегда есть запас чистых блоков, упрощающих сборку "мусора" и выравнивание износа. Помимо этого, когда ресурс какого-либо блока оказывается исчерпан, на замену ему назначается блок из резервной области.

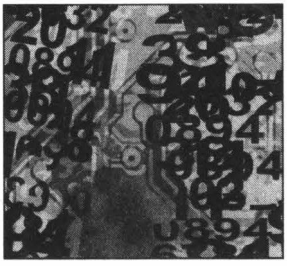
Чтобы еще лучше оптимизировать запись и удаление в SSD, в ATA появилась команда `TRIM` (в наборе команд NVMe аналогичная команда именуется `Deallocate`). С ее помощью операционная система сообщает накопителю, что файл удален, его данные больше не требуются и их можно окончательно стереть. Далее в игру вступает механизм сборки мусора, алгоритм которого разнится от модели к модели, но обычно он работает в фоне и не обязательно стирает данные сразу. В общем, командой `TRIM` дается зеленый свет безвозвратному удалению данных, потому что они физически стираются из FLASH-памяти, и тут уже никакая "остаточная намагниченность" их не вернет. Хотя в отдельных реализациях `TRIM` что-то и бывает возможно вычитать — подобнее об этом и не только можно узнать в приведенных далее источниках:

- "*Coding for SSDs*" — цикл статей, посвященных написанию оптимального кода для SSD. В нем описаны различные нюансы работы твердотельных накопителей и дано множество ссылок на научные исследования их устройства, надежности и быстродействия. Этот цикл статей доступен по адресу: **http://codecapsule.com/2014/02/12/coding-for-ssds-part-1-introduction-and-table-of-contents/**;
- "*Как свободное место на SSD влияет на его производительность и срок службы*" — статья наглядно разъясняет вопросы изменения производительности твердотельных накопителей по мере их заполнения. Доступна на страничке **http://www.outsidethebox.ms/14484/**;
- "*Жизнь после TRIM: как восстановить удаленные данные с накопителей SSD*" — статья, описывающая механизмы записи и удаления данных в SSD. Доступна по адресу: **https://blog.elcomsoft.com/ru/2018/12/zhizn-posle-trim-kak-vosstanovit-udalyonnyie-dannyie-s-nakopiteley-ssd/**.

Резюме

Мораль проста: чтобы никогда не заниматься восстановлением резервных копий (а это занятие не из приятных), всегда дублируйте все критические данные. Тогда при отказе одного из носителей вам не придется хвататься за сердце и "глушить" корвалол. Поверьте, время, затраченное на резервирование, не идет ни в какое сравнение с расходами на восстановление!

ГЛАВА 11



Восстановление лазерных дисков

В предыдущей главе проблема восстановления данных на оптических носителях уже упоминалась. Теперь рассмотрим эти вопросы подробнее.

Записываемые и перезаписываемые лазерные диски представляют собой идеальное средство для резервирования информации умеренных объемов (а всякий администратор обязательно должен заботиться о периодическом резервировании вверенной ему информации!). К сожалению, никакая работа без ошибок не обходится. Что поделаешь — человеку свойственно ошибаться — Errare humanum est, как говорили древние. Ошибочное удаление файлов с носителей CD-R/CD-RW, равно как и непредумышленная очистка последних, хотя бы однажды, да случается. Кстати, как показывает практика, с этим явлением приходится сталкиваться далеко не однажды, особенно если пользователи самостоятельно резервируют ту или иную информацию на CD/DVD.

И хотя в настоящее время существуют программы, предназначенные для восстановления информации с лазерных дисков, такие утилиты не были широко представлены на рынке, а их стоимость явно отпугнет рядового покупателя. Поэтому в подавляющем большинстве случаев восстановлением испорченных дисков приходится заниматься самостоятельно.

Восстановление удаленных файлов с CD-R/CD-RW

Заявляя о своей поддержке многосессионных дисков, операционные системы Windows (вплоть до Windows 10 включительно) "тактично" умалчивают о том, что поддерживают их лишь частично. Каждая сессия — это вполне самостоятельный том (в терминологии Windows — "логический диск"), имеющий собственную файловую систему и собственные файлы. Благодаря сквозной нумерации секторов лазерного диска файловая система одной сессии может ссылаться на файлы, физически расположенные в любой другой сессии. Для того чтобы с многосессионным диском можно было работать как с единым томом, файловая система последней

сессии должна включать в себя содержимое файловых систем всех предыдущих сессий. Если этого не сделать, то при просмотре диска штатными средствами Windows оглавления остальных сессий окажутся потерянными, поскольку Windows монтирует лишь последнюю сессию диска, а все прочие игнорирует. Программы "прожига" CD/DVD по умолчанию добавляют содержимое файловой системы предыдущей сессии к последующей, однако это еще не означает, что последняя сессия диска всегда содержит в себе все то, что имеется в предыдущих.

Рассмотрим, например, как осуществляется удаление файлов с CD/DVD-R. Нет, это не опечатка! Содержимое дисков CD-R, несмотря на физическую невозможность их перезаписи, в принципе все же уничтожаемо. Для имитации удаления файла программы записи на CD просто не включают ссылку на уничтожаемый файл в файловую систему последней сессии. Следует, правда, заметить, что эта возможность дарована далеко не всем программам. Например, Roxio Easy CD Creator может поступать таким образом, а некогда существовавший Stomp RecordNow — нет. И хотя "удаленный" файл все еще присутствует на диске, "отъедая" часть дискового пространства, при просмотре содержимого диска штатными средствами Windows он уже не отображается в каталоге. Если удалению одних файлов сопутствует запись других, то в любом случае приходится открывать новую сессию, а каждая вновь открываемая сессия требует для своего размещения определенного пространства. Какой же тогда смысл в удалении файлов с CD-R, если объем свободного пространства на диске при этом не увеличивается, а даже *уменьшается*? На самом же деле смысл этой операции (если его вообще можно назвать "смыслом") заключен исключительно в сокрытии "удаляемых" файлов от простых пользователей. Раз удаленные файлы не видны при просмотре содержимого диска штатными средствами, то неквалифицированному пользователю они формально недоступны.

> ***Примечание***
>
> Здесь уместно подчеркнуть, что "недоступны" эти файлы будут только для штатных средств операционной системы Windows. Например, компьютеры Macintosh позволяют монтировать любую сессию диска на отдельный том, благодаря чему при просмотре многосессионных дисков на Macintosh все якобы удаленные файлы сразу же "всплывают".

Аналогичным образом обстоят дела и при удалении информации с перезаписываемых носителей. Несмотря на теоретическую возможность физического уничтожения удаляемой информации подавляющее большинство записывающих программ поддерживают лишь функцию очистки всего диска целиком, но не могут выборочно удалять отдельные файлы. Так что все, сказанное выше о CD/DVD-R, в равной мере применимо и к CD/DVD-RW.

> ***Внимание!***
>
> Записывая на диск информацию, предназначенную для передачи постороннему лицу, ни в коем случае не используйте для этой цели носители, содержащие конфиденциальные данные. "Удаление" ранее записанных на лазерный диск данных на самом деле не уничтожает их!

Просматривая содержимое лазерного диска, полученного от приятеля (купленного на радиорынке, вытащенного из мусорной корзины), можно попытаться заглянуть внутрь предыдущих сессий для поиска скрытой информации. Как показывает практика, нередко там обнаруживается много интересного. Кроме того, вам может потребоваться восстановить ошибочно удаленный файл со своего собственного диска, а то и "воскресить" всю затертую сессию целиком. Стоит отметить, что некоторые программы записи на CD позволяют пользователю выбирать, следует ли при создании новой сессии добавлять в нее файловую систему предыдущей. При желании в новую сессию можно включить только новые файлы. Неверный выбор настроек приводит к утрате содержимого всех предыдущих сессий. К счастью, эта утрата обратима.

Для восстановления удаленных файлов можно воспользоваться любой программой, способной извлекать содержимое выбранной сессии диска и записывать его в образ ISO. Для определенности пусть это будет утилита PowerISO. Вставьте диск, подлежащий восстановлению, в привод, выберите пункт **Открыть CD/DVD/Blu-Ray привод** из меню **Файл**. На экране появится одноименное диалоговое окно (рис. 11.1).

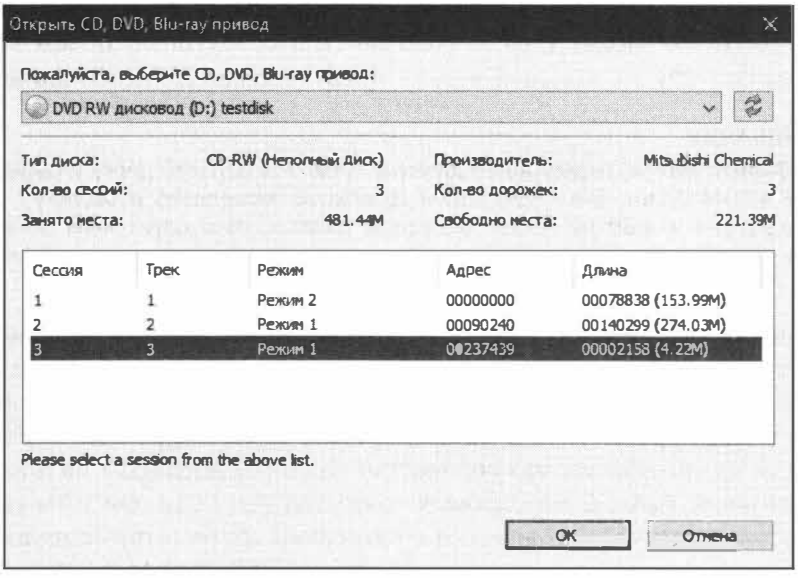


Рис. 11.1. Анализ содержимого диска для выявления удаленных файлов

Как мы и предполагали, в этом окне представлен перечень всех сессий, имеющихся на диске, с указанием номеров, стартовых адресов (в секторах) и длин (в мегабайтах). Давайте попробуем определить, имеются ли на диске скрытые файлы. Используя команду `dir`, выведем каталог диска и запомним суммарный размер всех файлов, которые только "видит" операционная система (листинг 11.1). Как следует из листинга 11.1, "коварная" Windows выводит содержимое одной лишь последней сессии диска. Что содержат все остальные — неизвестно. Во всяком случае — *пока* неизвестно.

Листинг 11.1. Вывод содержимого диска на экран

```
KPNC$G:\>dir
 Том в устройстве G имеет метку testdisk
 Серийный номер тома: 9B7D-008F
 Содержимое папки G:\

25.01.2020  12:08             2 169 Полезное.txt
07.02.2020  18:47             3 205 GRAND_PARTITION_THEFT_graph.drawio
07.03.2020  15:47             2 840 344 PLAN.jpg
07.03.2020 23:54        17 404 932 Prilyudie_v_do_mazhoryu_tsfts.wav
01.03.2020 15:08        28 495 512 bornagainbook.zip
10.03.2020 23:35               834 ch_04___stats
30.09.2019 17:44             4 548 defusing_LKM.zip
12.10.2019 00:14       258 210 726 publication_PDF.zip
17.12.2019 13:18           982 350 redeyesmen_backup_v.0.1.zip
               9 файлов    307 944 620 байт
               0 папок               0 байт свободно
```

Ага, совокупный объем девяти файлов, доступных для операционной системы, составляет всего 307 Мбайт (307 944 620 байт), а совокупный объем всех сессий диска — `153,99 + 274,03 + 4,22 = 432,24` Мбайт, что на 125 Мбайт длиннее!

> ***Примечание***
>
> Поле **Занято места**, содержащее длину записанной области диска и равное в данном случае 481,44 Мбайт, для наших целей абсолютно бесполезно, поскольку в записанную область диска входят не только полезные данные, но и служебные области каждой сессии. В результате этого полная емкость диска всегда меньше его эффективной емкости, даже если на нем нет никаких удаленных файлов.

В какой именно сессии содержатся удаленные файлы, сказать невозможно. Они могут присутствовать в любой из сессий или даже в нескольких сессиях сразу. Поэтому в общем случае все имеющиеся сессии нужно просматривать последовательно. Однако иногда удается найти более короткие пути. В простых случаях можно отталкиваться от того факта, что количество сессий, имеющихся на диске, на единицу больше числа файлов, выведенных командой `dir`. Если при этом размеры последних сессий практически совпадают с размерами соответствующих им файлов, а какая-то сессия диска не соответствует ни одному из видимых файлов, что же она тогда содержит? Давайте смонтируем эту сессию на отдельный дисковый том и посмотрим! К сожалению, штатные средства Windows не позволяют осуществлять такое монтирование непосредственно. Зато некоторые программы, включая PowerISO, позволяют нам не только просматривать содержимое разных сессий, но и отдельных файлов в них. Теперь найденные "удаленные" файлы можно извлечь простым перетаскиванием!

Возвращаясь к PowerISO (см. рис. 11.1), дважды щелкнем мышью по строке с сессией **1**. На экране немедленно появятся файлы, записанные в первую сессию (рис. 11.2).

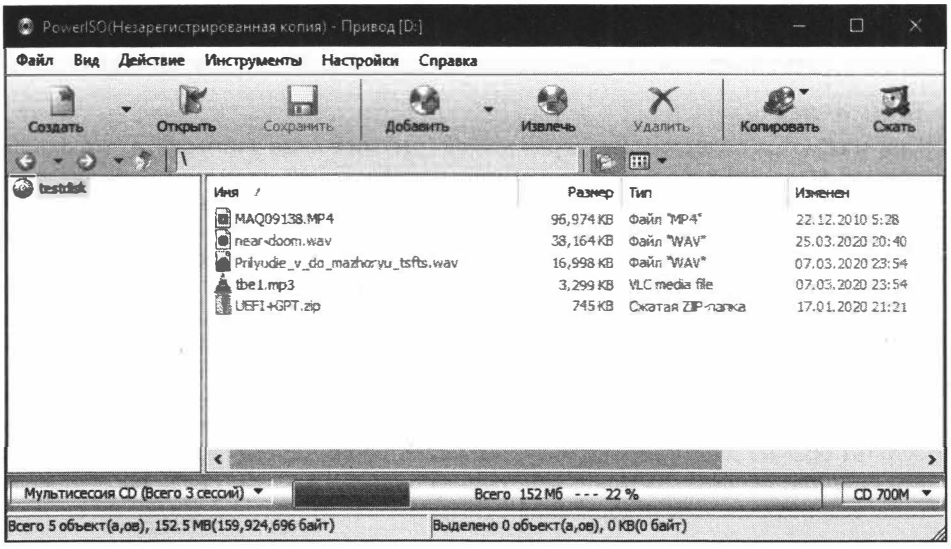


Рис. 11.2. Просмотр содержимого первой сессии в PowerISO. Четыре файла из пяти не выводятся командой `dir`

Содержимое диска, поделенное на сессии, при необходимости можно извлечь в виде образа. Для этого выберем на панели **Копировать — Создать файл образа CD/DVD/Blu-Ray** (рис. 11.3). Поле **Выходной файл**, как и следует из его названия, задает имя образа (по умолчанию "Track"), а **Формат образа** — формат. В случае многосессионного диска в качестве формата придется выбрать **BIN/CUE**, а отдельно сохранять сессии в образы ISO утилита PowerISO не позволяет. Затем созданный образ можно монтировать и читать в программах, поддерживающих этот формат, например DAEMON Tools.

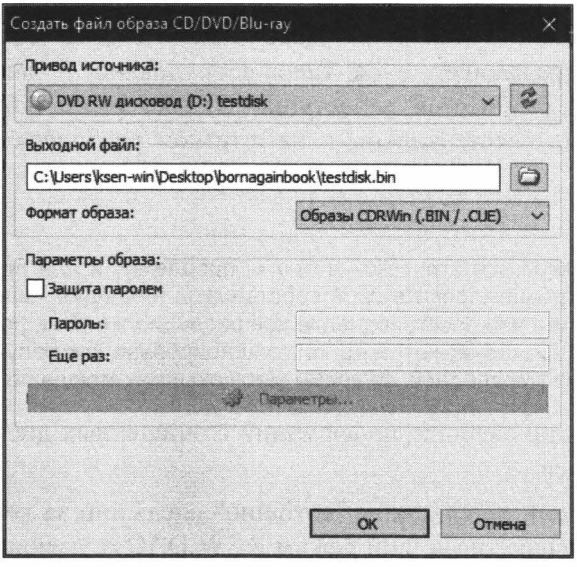


Рис. 11.3. Меню создания образа в PowerISO

Так или иначе, доступ к удаленным файлам будет получен, и вы сможете делать с ними все, что хотите.

> ***Внимание!***
>
> При просмотре содержимого "сграбленной" сессии всегда учитывайте, что файлы, физически принадлежащие другим сессиям, из данной сессии окажутся недоступными, в то время как ссылки на них здесь могут изобиловать. При обращении к реально несуществующему файлу будет выдаваться либо "мусор", либо сообщение об ошибке. Как альтернативный вариант — операционная система может просто "зависнуть". Если это произошло, нажмите кнопку выброса диска. Зависание тут же прекратится, и Windows радостно сообщит о неготовности устройства. Еще один факт, который обязательно нужно принять во внимание, состоит в том, что в силу сквозной адресации секторов каждая "сграбленная" сессия должна записываться на то же самое место диска, на котором она находилась ранее. В противном случае все ссылки на стартовые адреса файлов внутри этой сессии окажутся недействительными. Требуемый результат обычно достигается изменением стартового адреса первого трека. О том, как это сделать, рассказывается в следующем разделе данной главы, посвященном восстановлению очищенных носителей CD-RW.

Восстановление очищенных CD-RW

Существуют две принципиально разные методики очистки CD-RW: *быстрая* (quick) и *полная* (full). При быстрой очистке диска с него удаляется лишь область TOC (Table of Contents, таблица содержимого), в результате чего диск выглядит "пустым", хотя его основное содержимое остается нетронутым. Напротив, при полной очистке луч лазера "выжигает" всю поверхность диска целиком — от первого пита до последнего. Естественно, на это требуется время и полная очистка диска может растянуться на добрый десяток минут, в то время как быстрая спокойно укладывается в одну или две минуты.

Восстановление полностью очищенных дисков возможно только на специальном оборудовании, способном улавливать даже незначительные изменения отражательной способности отражающего слоя. Такое оборудование подавляющему большинству пользователей, разумеется, недоступно. Однако диски, подвергшиеся быстрой очистке, могут быть восстановлены и на штатном рекордере (правда, не на всех моделях).

> ***Внимание!***
>
> Мы не будем касаться этической стороны проблемы, и для простоты предположим, что вы хотите реанимировать свой собственный непредумышленно очищенный диск CD-RW. Отметим, что восстановление конфиденциальной информации с чужих перезаписываемых дисков может быть классифицировано как получение несанкционированного доступа к последней, со всеми вытекающими отсюда последствиями.

Для опытов по восстановлению информации с очищенных дисков CD-RW нам потребуется следующее.

- **Пишущий привод**, не слишком "дотошно" следящий за корректностью содержимого TOC, поддерживающий режим RAW DAO и умеющий читать содержимое `pre-gap` первого трека. Не все модели пишущих приводов подходят для этой

цели. Будьте готовы к тому, что вам придется перепробовать большое количество различного оборудования. Например, из двух имевшихся у нас рекордеров для восстановления очищенных дисков подошел лишь NEC, а PHILIPS на это, увы, не способен.

- **Продвинутое ПО для записи CD/DVD**, позволяющее манипулировать служебными областями диска по своему усмотрению. Вы можете использовать Clone CD, Alcohol 120% или любую другую аналогичную утилиту по своему выбору. Однако весь последующий материал относится исключительно к Clone CD, и при переходе на другую программу вы можете столкнуться с теми или иными проблемами. Если вы не уверены, что сможете справиться с ними самостоятельно, используйте Clone CD. Затем, по мере приобретения профессиональных навыков и должного опыта, вы без труда восстановите диск любой из перечисленных ранее программ.
- **Средство для работы с диском на секторном уровне** — утилита, позволяющая прочесть любой заданный сектор (конечно, при условии, что он вообще читается приводом) и не пытающаяся пропустить те сектора, в которых по ее "самоуверенному мнению" ничего интересного все равно нет. Упомянутые ранее копировщики защищенных дисков для этой цели не подходят, т. к. отказываются читать "бесполезные" с их точки зрения сектора. Может быть, другие копировщики и ведут себя иначе — не знаем, не проверяли. Вместо этого необходимую для работы утилиту Крис в свое время написал самостоятельно.

Прежде чем начинать экспериментировать, давайте разберемся, почему после очистки диск перестает читаться. Вопрос не так уж глуп, как кажется, — ведь информация, необходимая для позиционирования головки и поиска конкретных секторов при быстрой очистке диска, остается нетронутой! Управляющие данные "размазаны" вдоль всей спиральной дорожки и для чтения диска на секторном уровне TOC, в общем-то, и не требуются. Да, отсутствие TOC значительно усложняет анализ геометрии диска, и для определения количества треков/сессий диска в общем случае привод должен прочитать весь этот диск целиком. Однако при восстановлении информации фактор времени играет второстепенную роль, и им можно полностью пренебречь.

Тем не менее при попытке чтения любого из секторов очищенного диска привод с неизменным упорством возвращает ошибку. Почему? Очень просто — это "защита" от чтения заведомо некорректной информации. Еще ни один из всех знакомых нам приводов не мог читать сектора за пределами области `Lead-Out` (собственно, на программном уровне содержимое областей `Lead-in`/`Lead-out` тоже недоступно). Тем не менее эта невозможность отнюдь не носит концептуального характера, и удаление из микропрограммы привода "лишних" проверок позволяет прочитать такой диск "на ура". Нет, не подумайте! Призывать вас к дизассемблированию прошивок мы не собираемся. Дело это сложное, трудоемкое, да к тому же еще и небезопасное. Неверно модифицированная прошивка может угробить привод без малейшей надежды на его восстановление. Нет, уж лучше мы пойдем другим путем!

Предлагаемая идея восстановления информации в общих чертах сводится к записи на диск фиктивного TOC, адреса областей `Lead-in` и `Lead-out` которого указывают

на первый и последний сектор диска соответственно. При этом стартовый адрес первого трека точно совпадает с концом области `pre-gap`, которая по стандарту должна занимать не менее 150 секторов (или 2 секунд в пересчете на абсолютные адреса). После такой нехитрой операции привод будет послушно читать оригинальное содержимое очищенного диска. Разумеется, произойдет это только при условии, что мы ухитримся настроить записывающую программу, чтобы она после записи фиктивного TOC никоим образом не пыталась интерпретировать подсунутые указатели на области `Lead-in`/`Lead-Out` как указание "выжечь" всю поверхность диска целиком.

Проверка показывает, что Clone CD вообще не записывает такое оглавление на диск, сообщая о несоответствии размеров диска и образа файла. Alcohol 120% выполняет это указание без лишних препирательств, но совсем не так, как требовалось. "Забив" весь восстанавливаемый диск непонятно откуда взятым "мусором", программа авторитетно сообщает, что в процессе записи произошли ошибки и, возможно, вам следует убедиться в исправности оборудования.

Хорошо, зайдем с другой стороны. Запишем на диск один реальный трек, занимающий минимально возможное количество секторов (по стандарту — 300, но некоторые приводы вполне удовлетворяются и меньшими значениями), но расширим его `pre-gap` с двух секунд на... весь диск! В результате мы потеряем лишь 300 последних секторов, зато получим доступ ко всему остальному содержимому. Учитывая, что на диске этих секторов насчитывается немногим более 300 тысяч, нетрудно подсчитать, что процент успешно восстановленной информации составляет по меньшей мере 99,999% емкости всего диска. Это при том условии, что исходный диск был заполнен целиком, что на практике встречается редко. Если же это вас не удовлетворяет, то разрабатывайте свои программы, корректно записывающие фиктивное оглавление, но ничего не делающие сверх того. В любом случае область `Lead-in` записывает сам привод, ну а без `Lead-out` при аккуратном обращении с диском, в принципе, можно и обойтись. Главное здесь — корректно работать с секторами, находящимися за пределами диска, иначе поведение привода станет трудно предсказуемым. Стоит, правда, отметить, что с восстановлением полностью забитых информацией дисков мы еще не сталкивались.

Процедура восстановления состоит из трех частей: подготовки исходного образа трека с нормальным `pre-gap`, увеличения `pre-gap` до размеров целого диска и записи исправленного образа на восстанавливаемый диск. Первые два этапа достаточно выполнить всего один раз, т. к. полученный образ (далее мы будем называть его "лечебным") может использоваться для всех дисков. Маленькое уточнение — для всех дисков *той же самой емкости*, что и восстанавливаемый, ведь по понятным соображениям вы не сможете корректно восстановить 23-минутный диск с помощью образа, предназначенного для 80-минутного диска, и соответственно наоборот.

Для начала возьмем чистый диск CD-RW. Здесь понятие "чистый" не означает "ни разу не записанный". Под "чистым" диском будем понимать носитель CD-RW, очищенный быстрой или полной очисткой. Кроме того, для этих целей также подойдет и носитель CD-R. Используя любую утилиту для штатного "прожига", за-

пишем на него один крошечный файл, "весящий" не более 500 Кбайт (более "тяжелый" файл просто не уместится в запланированные 300 секторов). Выполнять финализацию диска не нужно.

Запустим Clone CD (Alcohol 120%) и снимем образ диска. Спустя минуту-другую, на винчестере образуются два файла: filename.img и filename.ccd. Если вы дали Clone CD указание сохранять и субканальную информацию, то образуется третий файл — filename.sub. Поскольку субканальная информация в данном случае будет только мешать, опцию **Чтение субканалов из треков с данными** лучше всего отключить. Кроме того, можно просто удалить filename.sub с диска; также нам не нужен "Cue-Sheet", который Clone CD предлагает создавать для совместимости с другими программами, конкретно — с CDRWin.

Открыв файл filename.ccd любым текстовым редактором (например, "Блокнотом"), выполните в нем поиск по ключевым словам `Point=0xa2` и `Point=0x01`. Результаты поиска приведены в листинге 11.2.

Листинг 11.2. Оригинальный стартовый адрес Lead-Out (слева) и стартовый адрес первого трека диска (справа)

```
[Entry 2]                    [Entry 3]
Session=1                    Session=1
Point=0xa2                   Point=0x01
ADR=0x01                     ADR=0x01
Control=0x04                 Control=0x04
TrackNo=0                    TrackNo=0
AMin=0                       AMin=0
ASec=0                       ASec=0
AFrame=0                     AFrame=0
ALBA=-150                    ALBA=-150
Zero=0                       Zero=0
PMin=0                       PMin=0
PSec=29                      PSec=1
PFrame=33                    PFrame=0
PLBA=2058                    PLBA=0
```

Изменим поля `PMin:PSec:PFrame`, принадлежащие `point 0xa2`, так, чтобы они указывали на самый конец диска (`0xa2` — это и есть `Lead-Out`). Измененный `Lead-Out` может выглядеть, например, так: `74:30:00`. Адрес `Lead-Out` следует выбирать с тем расчетом, чтобы между ним и внешней кромкой диска остался по меньшей мере 30-секундный зазор. Еще лучше, если ширина `Lead-Out` составит примерно полторы минуты. Однако в этом случае будут неизбежно теряться последние треки восстанавливаемого диска (если, конечно, вам действительно требуется их восстановить).

К содержимому полей `PMin:PSec:PFrame`, принадлежащих `point 0x01` (стартовый адрес первого трека), необходимо добавить ту же самую величину, которую вы добавили к соответствующим полям `Lead-Out`. Отредактированный вариант может выглядеть, например, так: `74:01:42`. (`74:30:00 /* новый адрес Lead-out */ –`

`00:29:33 /* старый Lead-Out */ + 00:01:00 /* старый стартовый адрес первого трека */ = 74:01:42 /* новый стартовый адрес */`). Короче говоря, новая версия файла ccd должна выглядеть так, как показано в листинге 11.3.

Листинг 11.3. Ключевой фрагмент "реаниматора" 75-минутных CD-RW-дисков

```
[Entry 2]                    [Entry 3]
Session=1                    Session=1
...                            ...
PMin=74                      PMin=74
PSec=30                      PSec=01
PFrame=00                    PFrame=42
```

Вообще-то для приличия следовало бы скорректировать и поля `PLBA`. Адрес LBA связан с абсолютным адресом следующим соотношением: `LBA = ((Min*60) + Sec)*75 + Frame`, однако текущие версии работают исключительно с абсолютными адресами и игнорируют адреса LBA. Теперь все, что находится между концом области `Lead-in` и началом первого сектора, будет называться `pre-gap`. При "прожиге" диска область `pre-gap` останется нетронутой и позже может быть прочитана на секторном уровне, что нам как раз и требовалось. Сказать по чести, чрезмерное увеличение `pre-gap` первого трека — не самая лучшая идея, т. к. не все приводы способны читать такой "жирный" `pre-gap`. С точки зрения совместимости было бы лучше увеличивать `pre-gap` *второго* трека, однако при этом первый трек придется располагать в самом начале диска и его тело неизбежно затрет восстанавливаемые сектора. И хотя это не такая уж большая проблема, т. к. в первых секторах диска все равно ничего ценного нет, к подобной мере без особой необходимости все же лучше не прибегать. На крайний случай действуйте так: запишите на диск две сессии и вместо стартового адреса `point 0x01` меняйте стартовый адрес `point 0x02` (он будет находиться в разделе `session=2`).

Теперь наскоро очистим наш "подопытный" диск и до предела заполним его какими-нибудь файлами.

> ***Совет***
>
> Предпочтительнее всего использовать текстовые файлы, т. к. в этом случае будет сразу видно, что извлекается с восстановленного диска — "мусор" или полезная информация.

Записав файлы на диск, тут же выполним его быструю очистку. Убедившись, что диск действительно очищен и его содержимое уже недоступно, запустим Clone CD и запишем на очищенный диск только что созданный нами "лечебный" образ. Запись должна проводиться в режиме DAO, иначе ничего хорошего у вас не получится.

> ***Совет***
>
> Прежде чем восстанавливать сколько-нибудь ценный диск на еще неизвестном вам приводе, попробуйте провести эксперимент на диске, не содержащем ничего интересного.

Вот наконец мы держим в руках свежевосстановленный диск. Но действительно ли он восстановлен? А вот сейчас и убедимся! Вставляем "воскресшего из пепла" в привод NEC и с замиранием сердца пробуем прочитать один из наугад взятых секторов из середины диска (начальные сектора обычно содержат нули, потом — файловую систему, и их очень легко принять за бессмысленный "мусор"). О чудо!!! Оригинальное содержимое очищенного диска читается как ни в чем не бывало. Правда, при попытке прочесть оглавление диска средствами операционной системы привод может впасть в "задумчивость", граничащую с полным "зависанием" (ведь стартовый адрес первого трека расположен не в начале диска, а совсем в другом месте), но это все ерунда! Главное, что на секторном уровне диск все-таки доступен, пусть и не на всех приводах. Так, в частности, ASUS вообще отказывается читать такой диск, возвращая ошибку, а PHILIPS читает сплошной "мусор". К счастью, этот "мусор" можно восстановить, — достаточно на битовом уровне выполнить EFM-перекодировку с более "правильной" позиции. Поскольку возможных позиций всего 14, перебор обещает не затянуться на длительное время. Тем не менее лучше всего просто приобрести более качественный привод.

Остается лишь привести диск в состояние, пригодное для работы с ним средствами операционной системы. Последовательно читая все сектора диска один за другим, мы будем собирать их в один img-файл, для определенности именуемый recover.img. Сектора, которые не удалось прочитать даже с нескольких попыток, мы будем просто пропускать. Теперь скопируем "лечебный" ccd-файл в recover.ccd и вернем стартовый адрес первого трека на прежнее место. Запишем сформированный образ на новый диск. В результате, если все было сделано правильно, любой привод должен корректно читать созданный диск. Сеанс демонстрационного восстановления окончен, и мы, освоившись с этой технологией, можем приниматься за вещи куда более серьезные. Например, откроем собственную компанию по восстановлению очищенных дисков. Шутка! Хотя... почему бы и нет?

Хорошо, а как быть, если очищенный диск был многосессионным? Ведь описанные приемы рассчитаны на работу лишь с одной сессией! На самом деле можно восстановить и многосессионный диск. Это лишь чуть-чуть труднее. Но чтобы это сделать, мы должны предварительно познакомиться с остальными полями TOC.

Постойте, а что, если после очистки диска на него что-то писалось, — возможно ли тогда его восстановление, или нет? Разумеется, непосредственно затертые позиции утеряны безвозвратно, но остальную часть информации по-прежнему можно спасти. Если диск до очистки был многосессионным, то нам даже не придется возиться с восстановлением файловой системы, т. к. файловая система каждой последующей сессии дублирует предыдущую, за исключением удаленных файлов. Последняя сессия диска оказывается достаточно далеко от его начала, а потому и риск ее затирания минимален (если, конечно, спохватиться вовремя, а не тогда, когда весь диск перезаписан до отказа). Восстановление односессионных дисков с затертой файловой системой — намного более трудная, но все-таки разрешимая задача. Во-первых, этих файловых систем на типовом диске целых две: ISO-9660 и Joliet. Правда, в силу их близкого географического положения при затирании диска обе они обычно гибнут. Во-вторых, указанные файловые системы не поддерживают фрагментацию, и всякий файл, записанный на лазерный диск, представляет собой

единый информационный блок. Все, что нужно для его восстановления, — определить точку входа и длину. Точка входа в файл всегда совпадает с началом сектора, а подавляющее большинство типов файлов позволяют однозначно идентифицировать свой заголовок по уникальной сигнатуре (в частности, для zip-файлов характерна следующая последовательность: `50 4B 03 04`). Конец файла, правда, определяется уже не так однозначно, и единственная зацепка — структура самого восстанавливаемого файла. Впрочем, большинство приложений довольно лояльно относится к "мусору" в хвосте файла, и потому точность определения его длины с погрешностью в один сектор на практике оказывается вполне достаточной. Поскольку файлы располагаются на диске вплотную, без "зазоров", конечный сектор всякого файла надежно вычисляется путем вычитания единицы из стартового сектора следующего за ним файла.

Вообще говоря, техника восстановления лазерных дисков намного проще и незатейливее искусства врачевания их прямых коллег — дискет и жестких дисков. Правда, поговорку "семь раз отмерь — один раз отрежь" еще никто не отменял, и одна из неприятнейших особенностей работы с CD-RW как раз и состоит в том, что вы не можете гарантированно управлять процессом происходящей записи. Дискеты и жесткие диски в этом смысле полностью прозрачны — что вы пишете, то вы и получаете. Перезаписываемые же носители, напротив, представляют собой "черный ящик", и никогда нельзя быть уверенным в том, что конкретный привод будет правильно интерпретировать отдаваемые ему команды (увы, восстановление дисков CD-RW никак не вписывается в рамки Стандарта, а все нестандартные махинации могут интерпретироваться приводом неоднозначно). Единственное, что остается посоветовать: не пускайте все на самотек. Бесконечно экспериментируйте, экспериментируйте и еще раз экспериментируйте, накапливая бесценный опыт, который вам когда-нибудь может очень пригодиться.

Искажение размеров файлов

Еще (или, скорее, уже) во времена монохромных терминалов и первых магнитных дисков существовал некрасивый, но элементарно реализуемый защитный прием, препятствующий пофайловому копированию носителя. Внося определенные искажения в структуры файлов системы, разработчики "портили" дискету ровно настолько, чтобы работа с ней становилась возможной лишь при условии учета характера внесенных искажений. Защищенная программа, "знающая" об искажениях файловой структуры, работала с ней без проблем, но штатные утилиты операционной системы взаимодействовать с такими дисками не могли. Общедоступных "хакерских" средств копирования в те времена еще не существовало.

Несколько файлов зачастую ссылались на общие для всех кластеры, и тогда запись данных в один файл приводила к немедленному их появлению в другом, что защита могла так или иначе использовать. Естественно, после копирования файлов на новый диск пересекающиеся кластеры "разыменовывались" и хитрый способ неявной пересылки данных переставал работать. Вместе с ним переставала работать и сама защищенная программа, если, конечно, содержимое диска вообще удавалось

скопировать. Ведь копирование файлов с пересекающимися кластерами приводило к тому, что эти кластеры многократно дублировались в каждом копируемом файле, в результате чего их суммарный объем подчас увеличивался настолько, что емкости тогдашних носителей попросту не хватало для хранения таких объемов данных! Если же последний кластер файла "приклеивался" к его началу (файл таким образом попросту зацикливался), то объем и время его копирования попросту обращались в бесконечность. Конечно, дисковые доктора в то время уже существовали, но их использование не давало желаемого результата, потому что лечение файловой системы приводило к полной неработоспособности защиты. В случае с зацикливанием, например, если защита основывалась на том, что за концом файла следует его начало, то после обработки диска доктором осуществление этого приема становилось невозможным со всеми вытекающими последствиями.

Файловые системы лазерных дисков, конечно, совсем не те, что на гибких дисках, но общие принципы их искажений достаточно схожи. Увеличивая фиктивную длину защищаемых файлов на порядок-другой, разработчик защиты может довести их суммарный объем до нескольких сотен гигабайт, так что для копирования защищенного диска понадобится по меньшей мере пачка DVD или винчестер солидного объема. Защитный механизм, "помнящий" оригинальную длину всех файлов, сможет работать с ними без проблем, но все средства копирования файлов не поймут этого "юмора", и их поведение станет неадекватным.

В принципе, выход за границы файла ничем не чреват. Файловые системы лазерных дисков очень просты. Лазерные диски не поддерживают фрагментацию файлов, а потому и не нуждаются в FAT. Все файлы занимают непрерывный ряд секторов, и с каждым файлом связано только две важнейшие характеристики: *номер первого сектора* файла, заданный в формате LBA (Logical block address), и его *длина*, заданная в байтах. Остальные атрибуты, вроде имени файла и времени его создания — не в счет, мы сейчас говорим исключительно о секторах.

Увеличение длины файла приводит к "захвату" того или иного количества примыкающих к его "хвосту" секторов, и при условии, что номер последнего сектора, принадлежащего файлу, не превышает номера последнего сектора диска, копирование файла, в принципе, протекает нормально. Оговорка "в принципе, нормально", а не просто "нормально" здесь не случайна, т. к. в копируемый файл будут включены все файлы, встретившиеся на его пути. Если же в процессе своего копирования файл "выскакивает" за конец диска, то привод CD-ROM сигнализирует об ошибке и прекращает чтение. Штатные средства копирования, входящие в состав операционной системы, равно как и большинство оболочек сторонних производителей, автоматически удаляют "огрызок" недокопированного файла с диска, и в результате пользователь остается ни с чем.

Утилита ISO9660.DIR.EXE выгодно отличается тем, что позволяет копировать не только весь файл целиком, но и любую его часть! Но как мы узнаем, сколько именно байтов следует скопировать? Как определим, где идут полезные данные, а где начинается "послехвостовой мусор" (over-end garbage)? Будем исходить из того, что по Стандарту файлы на диске располагаются последовательно, т. е. за последним сектором одного файла непосредственно следует стартовый сектор следующего.

Поскольку стартовые сектора всех файлов нам известны, определение истинной длины всех файлов, за исключением последнего, не составит никакого труда! Так как длина одного сектора лазерного диска составляет 2048 байт, истинный размер всякого файла равен: `(стартовый адрес следующего файла – стартовый адрес самого этого файла) * 2048`. Все просто, не правда ли?

С помощью утилиты ISO9660.DIR.EXE нам и нашим друзьям удалось скопировать большое количество дисков с MP3, обладающих защитой данного типа.

Утилиты для восстановления информации с оптических дисков

Представим краткий обзор ряда программ, автоматизирующих процесс восстановления данных с лазерных дисков. Зачем это может понадобиться в XXI веке — уже другой вопрос. Архивы, offline-бэкапы, видео на Blu-Ray, музыка, разные раритетные (и не очень) издания, единственная оставшаяся запись с камеры утренника из двухтысячных... Словом, всякое бывает, а вот время не щадит никого.

CDRoller

Утилита позволяет восстанавливать данные с поврежденных или неправильно записанных дисков. Программа платная, но для скачивания доступна двухнедельная

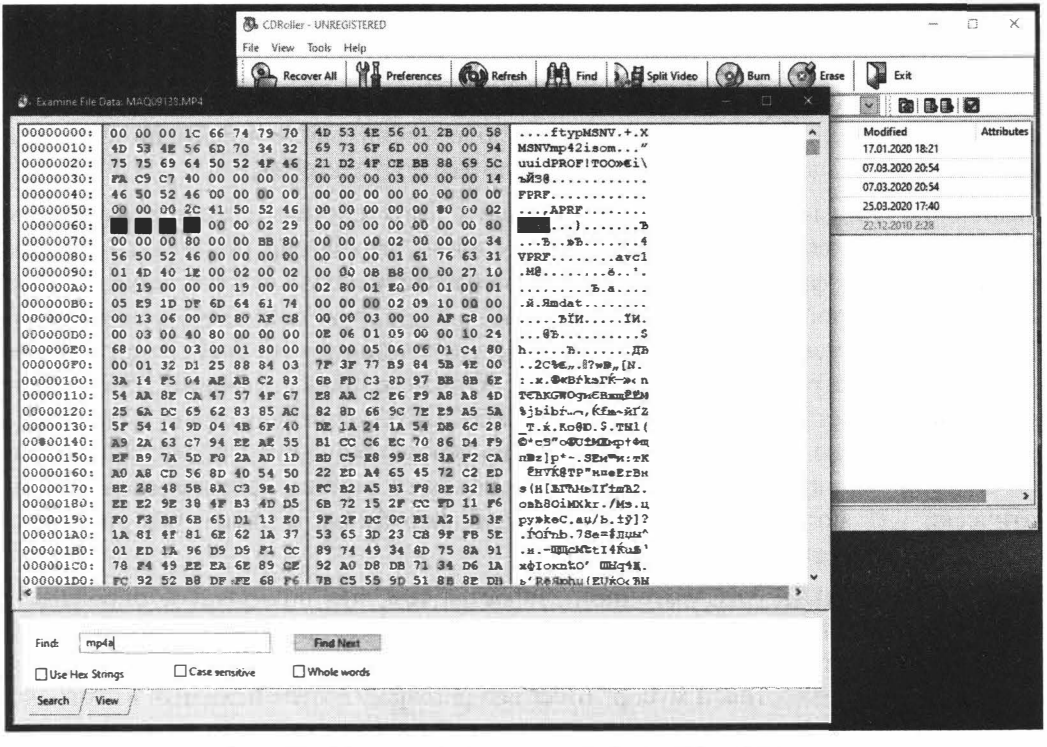


Рис. 11.4. Просмотр содержимого файла на CD в CDRoller

пробная версия. CDRoller умеет работать с CD, различными вариантами DVD, Blu-Ray; помимо оптических дисков может восстанавливать файлы еще и с HDD, SSD и карт памяти (чего уж мелочиться?). Поддерживает диски в форматах ISO9660, включая расширения Joliet и Rock Ridge, UDF и FAT32 (диски DVD-RAM и BD-RE). С полным списком ее богатых возможностей можно ознакомиться на домашней страничке по адресу: **https://www.cdroller.com**.

В CDRoller можно просмотреть каждый файл в шестнадцатеричном формате с поиском по его содержимому или даже по содержимому всей сессии (рис. 11.4). Кроме того, можно запустить тестирование диска (**Tools — Test Disc**), которое показывает информацию по каждой сессии и ее файлам — какой из них какой сессии на самом деле принадлежит.

DiskInternals CD & DVD Recovery 2.0

Эта утилита от DiskInternals тоже платная, но пробная версия умеет все самое необходимое, поможет, например, если диск был некорректно записан или требуется

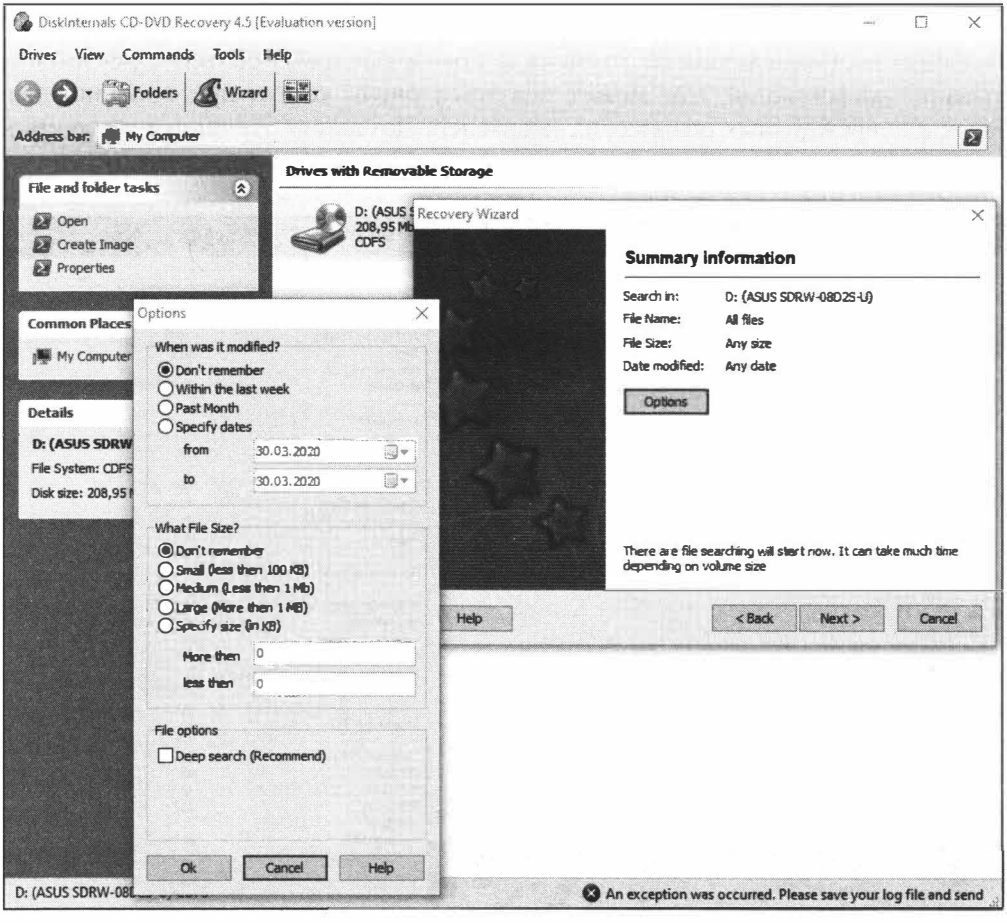


Рис. 11.5. Опции восстановления в DiskInternals CD & DVD Recovery

восстановить данные старых сессий. При запуске появляется помощник, ведущий за ручку через процесс восстановления файлов на диске. Программа поддерживает разные форматы CD/DVD с файловыми системами ISO9660, Joliet, UDF, а также созданные в InCD\DirectCD. А вот работа с Blu-Ray не заявлена. Интерфейс программы весьма прост, но есть возможность и самостоятельно выбирать определенные настройки (рис. 11.5). Получить DiskInternals CD & DVD Recovery можно по адресу **https://www.diskinternals.com/cd-dvd-recovery/**.

IsoBuster

Еще одна платная утилита, с заявленной поддержкой множества форматов CD, DVD и Blu-Ray, умеющая заодно восстанавливать данные с прочих переносных носителей и жестких дисков по сигнатурам и взаимодействующая с множеством файловых систем. Что касается оптических дисков, программе можно наглядно видеть, в какую сессию какие файлы входят, и восстановить необходимые, перетащив их в нужную папку. Также имеется возможность просмотреть карту блоков, проверить физические ошибки чтения и узнать много другой информации о записанном диске (рис. 11.6).

В IsoBuster доступно множество тонких настроек файловых систем и режимов вычитывания информации, что может оказаться очень кстати при восстановлении данных с поврежденных оптических носителей. Домашняя страничка продукта находится по адресу: **https://www.isobuster.com/isobuster.php**.

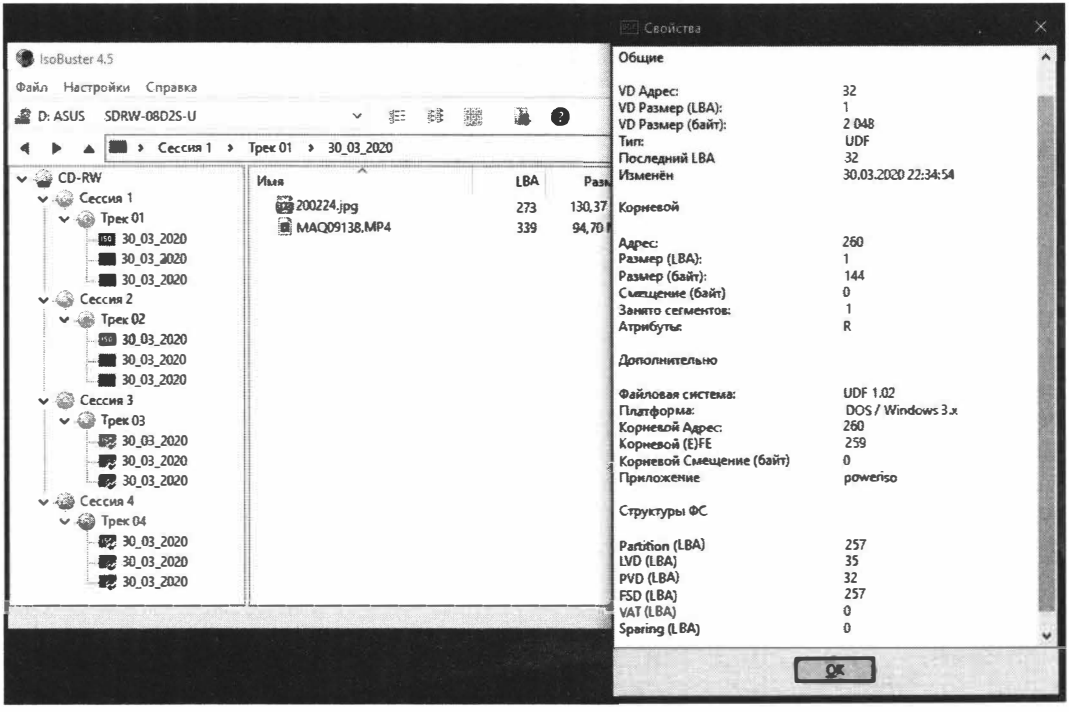


Рис. 11.6. Просмотр свойств одной из сессий в IsoBuster

Recovery ToolBox for CD Free

В отличие от предыдущих, это уже полностью бесплатная утилита. Несмотря на название, в ней заявлена поддержка DVD, HD-DVD и Blu-Ray. Интерфейс предельно прост: выбрал привод, диск, найденные утилитой файлы, которые нужно восстановить, — и готово. Надо сказать, утилита "обнаружила" файлы только из последней сессии без малейшего намека на остальные (рис. 11.7). А потому для восстановления удаленных файлов разных сессий она совершенно не подходит! Если же вы все-таки хотите попробовать эту программу или же просто посмотреть на нее, найти ее можно по адресу **https://recoverytoolbox.com/ru/cd.html**.

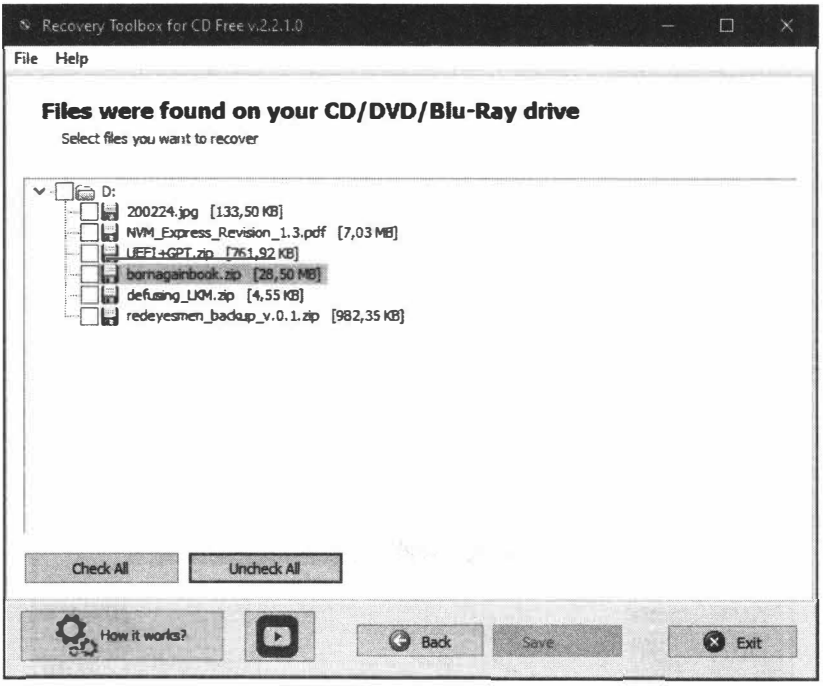


Рис. 11.7. Все, что смог найти Recovery ToolBox for CD Free, — то же самое видно и в обычном Проводнике Windows

CDCheck и Non-Stop Copy

Это две уже довольно старые утилиты. Они обе бесплатные, а раз уж бесплатных программ для восстановления дисков далеко не так много, то обратим внимание и на них.

CDCheck (**http://kvipu.com/CDCheck/download.php**) отобразил файлы только из последней сессии. Эта программа, скорее, предназначена для проверки и восстановления информации с физически нечитаемых оптических дисков (рис. 11.8). В силу своего возраста не имеет ни малейшего представления о дисках Blu-Ray.

Non-Stop Copy (**http://dsergeyev.ru/programs/nscopy/**), несмотря на свой возраст, все еще может оказаться весьма полезной. Просто иногда бывают программы,

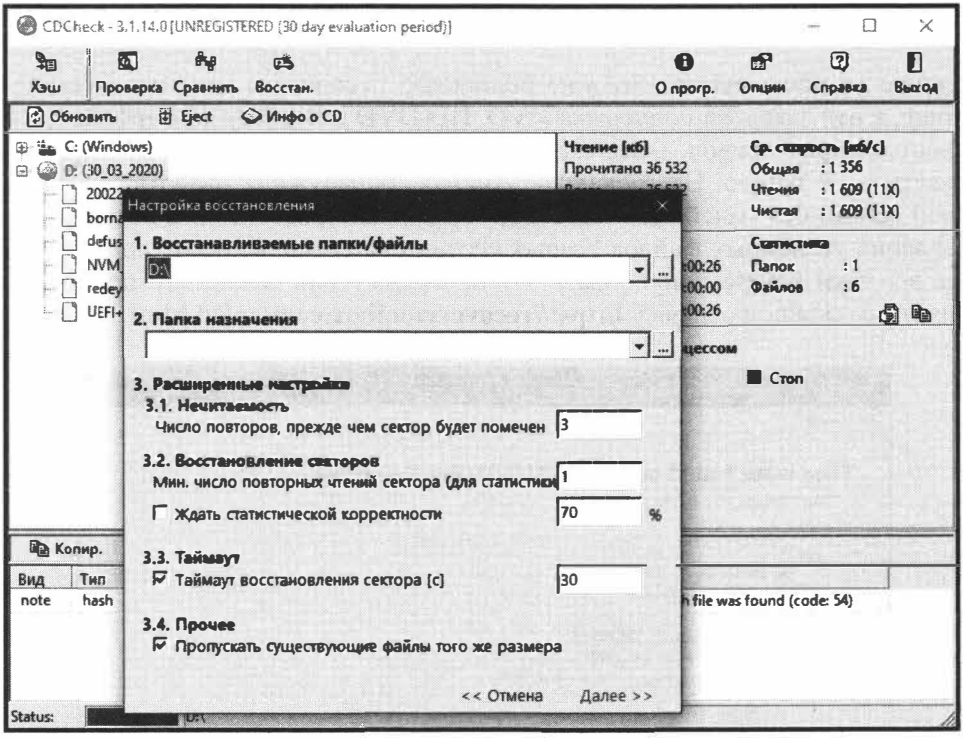


Рис. 11.8. Настройки восстановления в CDCheck

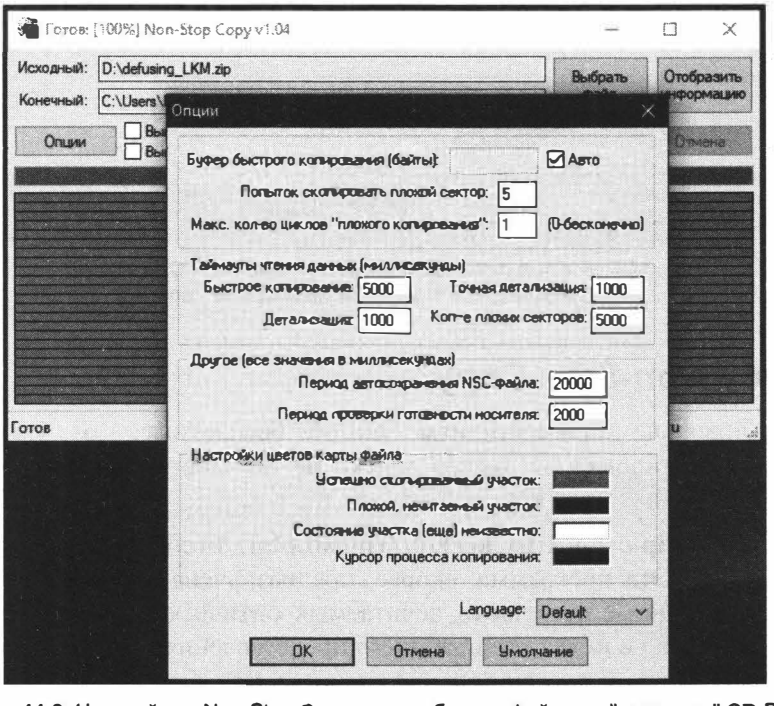


Рис. 11.9. Настройки в Non-Stop Copy и карта блоков файла на "здоровом" CD-RW

в которых нечего улучшать... Вообще говоря, Non-Stop Copy предназначена для восстановления файлов с любых носителей, имеющих физические дефекты, а не только оптических. Но поскольку программа не использует собственных низкоуровневых механизмов, то и восстанавливать она может лишь те файлы, которые "видит" операционная система. Зато она будет вычитывать их очень упорно, при этом бережно составляя наглядную карту блоков (рис. 11.9).

UDF — расплата за бездумность

Как-то раз, когда записывающие приводы только-только входили в моду, робко осваивая необъятные просторы российского рынка, в одной компьютерной фирме раздался звонок взбешенного покупателя. Состоялся следующий диалог:

Покупатель: "Мужики! Что за дела?! Какого черта вы мне подсунули неработающий рекордер!"

Продавец: "А какую программу вы используете для записи?"

Покупатель: "Нортон, естественно, и нечего держать меня за дурака!"

Сейчас этот анекдот уже не вызывает улыбки. Современные оптические носители поумнели настолько, что запись можно вести любыми средствами, хоть из Проводника Windows, хоть из FAR, хоть из того же Нортона. Однако это удобство дорого обходится — снижается надежность, уменьшается емкость, замедляется работа операционной системы и возникают прочие неприятности.

Как же непросто начинающему пользователю разобраться со всей этой кухней! Информация, почерпнутая с форумов, достаточно противоречива, а технические спецификации слишком сложны. Как быть? Что делать?

Механизм "прозрачной" записи на CD/DVD базируется на двух взаимодополняющих технологиях: *пакетной записи* (packet writing) и *динамической файловой системе* (dynamic file system), роль которой, как правило, играет UDF (Universal Disk Format — универсальный дисковый формат). Эти два понятия очень часто путают, хотя они стоят на разных ступенях иерархии.

Пакетная запись — это режим прожига, аппаратно поддерживаемый приводом. Помимо него существуют и другие режимы: SAO (Session At Once — сессия за раз), DAO (Disk At Once — диск за раз) и TAO (Track At Once — трек за раз). Не вдаваясь в технические подробности, отметим, что режим определяет размер порции данных, записываемых рекордером за один раз (т. е. без остановки лазера).

Самый "расточительный" из всех режимов, DAO, выжигает весь образ диска целиком от первого до последнего сектора и не допускает "дозаписи". Более экономичный режим SAO позволяет дописывать диск многократно, по одной сессии за раз, но при этом каждая сессия занимает по меньшей мере 15 Мбайт дискового пространства, что ощутимо бьет по карману. Потрековый режим TAO, "съедающий" всего лишь 300 Кбайт служебных данных на каждый трек, к сожалению, применим лишь к аудиодискам, т. к. ни одной существующей файловой системой он не поддерживается. К тому же все три режима не позволяют стирать ранее записанные

данные, поскольку они проектировались исключительно для однократно записываемых дисков CD-R. В лучшем случае обеспечивается лишь имитация стирания, осуществляемая путем удаления ссылок из каталога. При этом сами данные физически остаются нетронутыми, да и свободного места не прибавляется.

Всех этих недостатков лишен *пакетный режим*, сокращающий "аппетит" бюрократического аппарата до 14 Кбайт на пакет. При этом сама запись ведется блоками постоянного или переменного размера от 2 Кбайт до 2 Мбайт. Предельно допустимый размер пакетов определяется конструктивными особенностями привода и варьируется от одной модели к другой. Однако этот размер должен составлять не менее 64 Кбайт, иначе это будет неправильный привод, идущий вразрез со стандартом. Пакеты последовательно заполняют диск, двигаясь от его внешней кромки к центру. Спиральная дорожка должна быть непрерывна на всем своем протяжении. Природа оптических дисков такова, что информация "размазывается" вдоль спиральной дорожки, перемешивая биты различных секторов, что обеспечивает лучшую восстанавливающую способность в борьбе с радиальными царапинами и локальными дефектами, поэтому записать один-единственный сектор за раз невозможно в принципе! Нельзя записать пакет в середину диска, оставив за собой хотя бы один непрожженный сектор (рис. 11.10), но ранее записанные пакеты могут перезаписываться многократно, за счет чего, собственно говоря, и обеспечивается возможность удаления файлов.

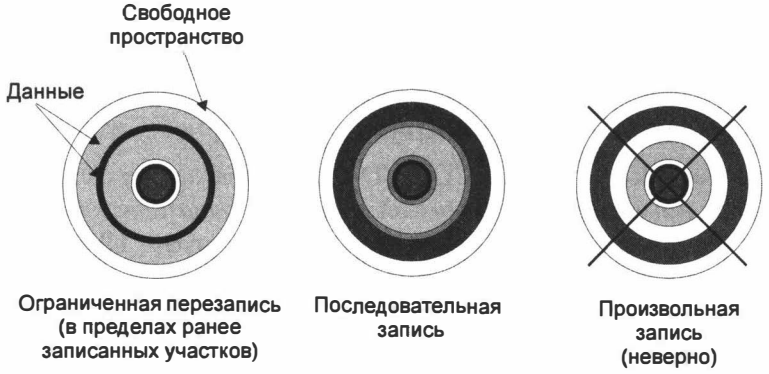


Рис. 11.10. Примеры инкрементной записи

Одного лишь механизма пакетной записи для осуществления задуманного явно недостаточно, и к нему еще требуется подобрать адекватную файловую систему. Стандартные файловые системы ISO-9660 и Joliet, разработанные специально для CD-ROM и ничего не знающие о фрагментации, при размещении файла на диске ожидают увидеть непрерывный блок свободного дискового пространства, который обнаруживается далеко не всегда.

Файловая система UDF — детище Optical Storage Technology Association — проектировалась с оглядкой на DVD и была далека от мыслей о мировом господстве. Однако разработка оказалась настолько удачной, что ее без труда удалось приспособить и к носителям CD-RW, с учетом всех особенностей их строения. UDF

оперирует не с физической, а с логической разметкой диска, и поэтому ей все равно, на каком носителе располагаться.

> ***Примечание***
>
> Таким образом, диск, записанный в формате UDF, не обязательно должен быть записан в пакетном режиме, равно как и наоборот — не всякий пакетный режим пользуется услугами файловой системы UDF. Возможность выборочной записи/удаления отдельных файлов на аппаратном уровне обеспечивается режимом пакетной записи, а на программном — специальной драйверной оснасткой. UDF лишь сокращает накладные расходы до разумного минимума, но не более того!

Существует несколько спецификаций UDF, самыми устойчивыми из которых являются следующие релизы:

- 1.02 описывает размещение данных (в том числе и видео) на DVD-ROM, поддерживает фрагментацию и ряд других полезных возможностей;
- 1.50 включает менеджер управления дефектами, препятствующий размещению данных на некачественных участках носителя, добавлена работа с CD-RW/CD-R;
- 2.00 поддерживает потоковые файлы, списки управления доступом, калибровку лазера и прочие второстепенные функции;
- 2.01 поддерживает файлы реального времени, гарантирующие сохранение заданной скорости считывания на всем протяжении диска;
- 2.50 может встретиться на HD-DVD и некоторых Blu-Ray, включает раздел метаданных для облегчения их группировки и восстановления данных;
- 2.60 поддерживает метод псевдоперезаписи на последовательно записанных дисках, используется в дисках Blu-Ray.

Windows XP уже поддерживала UDF 1.02, 1.50 и 2.01. Windows 10 по умолчанию предлагает форматирование диска в UDF 2.01, но можно выбрать другие версии (рис. 11.11). Если вам встретится сокращение LFS (Live File System), не пугайтесь — это не более чем наименование UDF 2.01 в Windows. Для работы с остальными операционными системами требуется установка соответствующего драйвера, точнее, даже драйверов, т. к. последующие спецификации не включают в свой состав предыдущие. Что же касается Linux-подобных операционных систем, то здесь поддержка UDF выглядит уже совсем не так печально, как лет 15 назад, когда пакет программ под названием udftools (**https://github.com/pali/udftools**) только-только "вставал на ноги"! Благодаря ему в современных Linux возможна работа с UDF 2.50 на чтение и по 2.01 — на запись. Ну а среди Berkeley UNIX только в современных (5.0) NetBSD заявлена поддержка всех существующих на данный момент версий UDF на чтение и запись, OpenBSD (4.7) — только на чтение. Mac OS X поддерживает чтение UDF 2.60 и запись UDF 2.50.

Компоненты, необходимые для работы с UDF

Для полноценной работы с дисками, размеченными в формате UDF, нам необходимо иметь следующее:

- рекордер, поддерживающий режим пакетной записи, причем поддерживающий его не кое-как (ради "галочки" в прайс-листе), а спроектированный и реализо-

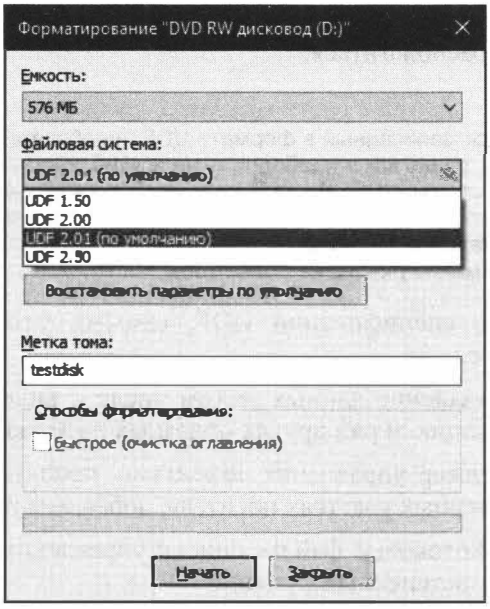


Рис. 11.11. Параметры форматирования оптического диска в Windows 10

ванный с учетом всей жесткости требований пакетного режима. Обычно тестовые лаборатории различных журналов приводят более или менее полную информацию о характере наиболее ходовых приводов, так что выбрать приличную модель не составит никакого труда;

- драйвер UDF, переводящий язык служебных структур UDF на язык операционной системы (UDF-reader);
- монитор UDF, перехватывающий все обращения к CD, а также обеспечивающий прозрачную запись и форматирование диска. Монитор обычно представляет собой иерархию драйверов, в которой можно выделить по меньшей мере два уровня — драйверы абстракции от конструктивных особенностей конкретного оборудования и драйверы, относящиеся непосредственно к самой операционной системе. Добавьте сюда еще программу-индикатор, отображающую состояние привода, и вы получите настоящий "коктейль" драйверов;
- если вы не планируете "прожигать" диски из FAR Manager, но хотите использовать режим пакетной записи для минимизации накладных расходов прожигателя, то без UDF-монитора можно обойтись, заменив его автономной программой записи, например Nero.

Первый опыт лучше всего приобретать, купив коробочный (retail) привод, в комплект поставки которого входит все необходимое программное обеспечение, автоматически устанавливаемое инсталлятором. В противном случае проблем совместимости вам не избежать. Естественно, гнаться за последними версиями драйверов совершенно не обязательно. Диски, размеченные в формате UDF v.2.*x*, в подавляющем большинстве случаев читаются и драйверами от UDF v.1.5, пускай и с ограниченными возможностями. Так, списков управления доступом вы не получите, а

при архивировании каталога Documents and Settings в многопользовательских системах без этого обойтись невозможно.

Windows 10 обеспечивает встроенную поддержку UDF и никаких дополнительных драйверов для пакетной записи не требует. Устанавливать дополнительное программное обеспечение не нужно, т. к. все оно нестабильное и неправильное. Хотя... вкусы бывают разные.

Технология Mount Rainier

Нашумевшая технология Mount Rainier в действительности является не более чем маркетинговой уткой, реально не сулящей ничего принципиально нового. Но обо всем по порядку. Что такое Mount Rainier? Это организация, названная в честь живописного национального парка (**http://www.nps.gov/mora**) и курирующая вопросы взаимодействия операционных систем с оптическими накопителями. В ее состав входят практически все крупные производители аппаратных средств и программного обеспечения: Philips, Microsoft, Compaq, Sony и т. д.

Mount Rainer Writer (сокращенно MRW), возносимый некоторыми журналистами чуть ли не до промышленного стандарта пакетной записи, в действительности представляет собой обычную программу для пакетной записи плюс UDF 2.0.1. От программного обеспечения требуется умение форматировать диск в фоновом режиме и корректно обрабатывать прерывание последнего по нажатию кнопки EJECT (после вставки диска форматирование будет продолжено).

Заявления о том, что технология Mount Rainer обеспечивает увеличение емкости и количества возможных циклов перезаписи дисков CD-RW, совместимость записанных носителей со всеми современными приводами и операционными системами, оптимизированную скорость передачи данных, дополнительную коррекцию ошибок приводами, не совсем соответствуют действительности. Увеличение циклов перезаписи за счет внедрения в файловую систему менеджера дефектов появилось еще в UDF v.1.5, которой нынче трудно кого-либо удивить. Совместимости со всеми операционными системами у Mount Rainer нет, и без соответствующего драйвера они не читаются. Конечно, это "неправильные" операционные системы, не поддерживающие MRW, а "правильность" — это прерогатива Windows, ради продвижения которой вся эта рекламная шумиха, собственно говоря, и затевалась.

Короче говоря, никакой необходимости в наличии логотипа Mount Rainer Compatible на коробке покупаемого привода нет! Живите спокойно! Ту же самую функциональность можно обеспечить и за меньшие деньги!

Регламент работ

При установке чистого диска CD-RW в привод UDF-монитор автоматически предлагает его отформатировать. Диски CD-R чаще всего игнорируются, и форматировать их приходится вручную. В зависимости от специфики драйверной оснастки эта операция осуществляется либо через стандартное контекстное меню проводника Windows, либо через интерфейс самой программы записи.

В зависимости от скорости и конструктивных особенностей привода форматирование может занять от двадцати минут до одного часа. В режиме Mount Rainer форматирование осуществляется в фоновом режиме, и возможность записи файлов доступна уже через несколько секунд после его начала. Эффективная емкость отформатированного диска составляет порядка 550 Мбайт, остальные мегабайты заняты служебными данными, так что если вы не обнаружите их на своем диске, не пугайтесь — все идет по плану. Структура пакета схематично показана на рис. 11.12. Как видите, значительный объем пространства отводится для служебной информации (run-in, run-out, pre-gap, post-gap и т. д.). В некоторых случаях эти накладные расходы могут быть весьма значительными. Попробуйте, например, скопировать в пакетном режиме полноценный фильм.

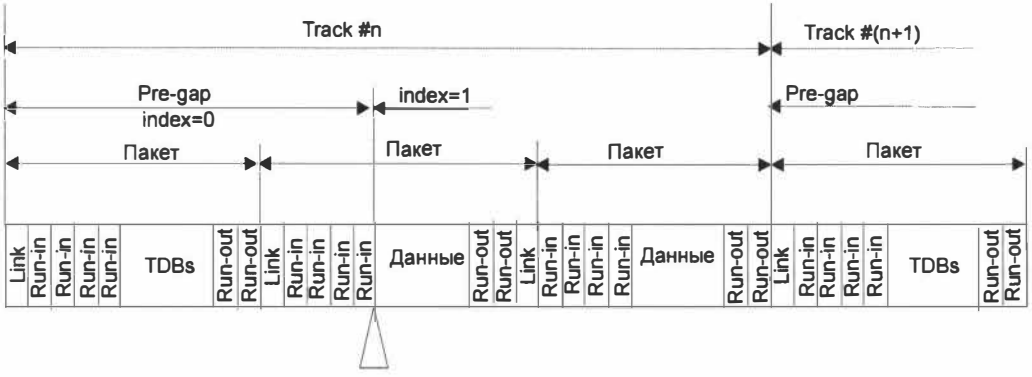


Рис. 11.12. Структура пакетов в режиме пакетной записи

Теперь, используя мышь или FAR Manager, попробуйте перетащить на CD-R/CD-RW диск несколько файлов, и они послушно скопируются, а свободный объем скачкообразно уменьшится на величину, существенно превышающую суммарный размер записываемых файлов. Что ж, пакетная технология берет свою мзду!

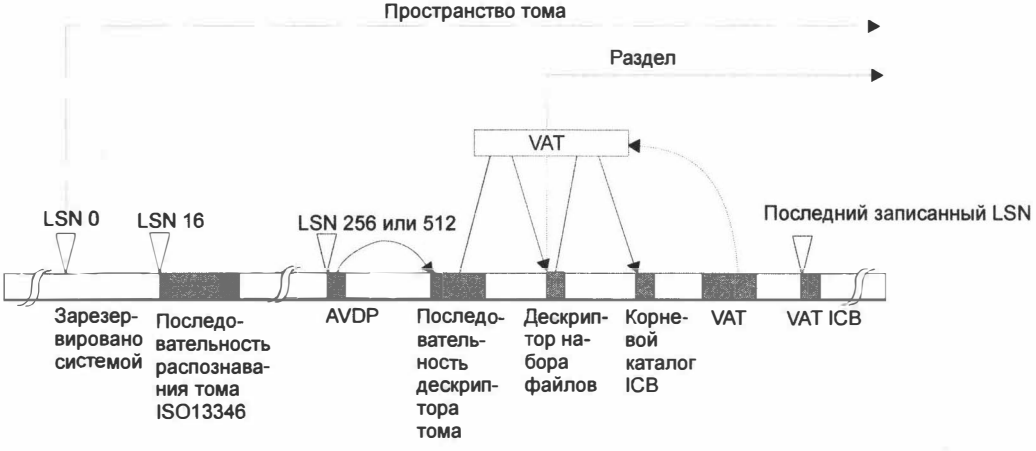
Будьте готовы к тому, что при попытке просмотра диска в системе без драйверов UDF диск либо не будет читаться совсем, либо, что более вероятно, обнаружит в своем каталоге один-единственный исполняемый файл, который вы туда не записывали. Успокойтесь! Это отнюдь не вирус, разрушивший все ваши файлы. Это — UDF-reader. Естественно, предназначенный для Windows и, естественно, требующий после установки перезагрузки (а под Windows — еще и прав администратора). При этом его еще не так-то просто удалить из системы. Подумайте — а захочет ли владелец того компьютера устанавливать на него что-то навязываемое? Он может взять да и отказаться работать с вашим диском!

Правда, при закрытии сессии на "родной" машине UDF-монитор обычно формирует файловую систему ISO 9660 — стандартную для всех операционных систем и читающуюся безо всяких драйверов, но дальнейшая запись файлов на этот диск уже становится невозможной вплоть до его очистки.

Информация к размышлению

Чтобы там ни говорили производители, пакетная технология намного менее надежна, чем классическая запись всей сессии целиком. Многократные зажигания/гашения лазера образуют прерывистую цепочку, концы которой плохо "склеиваются" друг с другом, и потому оптической головке стоит больших усилий не сбиться с дорожки. В момент зажигания лазера его характеристики довольно сильно "пляшут", что ухудшает качество прожига. В обычном режиме у привода есть время стабилизироваться, т. к. прожиг начинается с записи вводной области (lead-in), многократно дублирующей служебную информацию. По этой причине первые несколько секторов вводной области практически всегда оказываются дефектными. Что касается пакетной записи, то в этом режиме лазеру приходится включаться в работу "с места в карьер".

К тому же сектора, хранящие файловую систему, работают в необычайно интенсивном режиме, перезаписываясь при каждой операции копирования/удаления. Чтобы избежать множественной перезаписи одних и тех же секторов, были предприняты специальные меры. Например, виртуальная таблица расположения (Virtual Allocation Table, VAT), являющаяся одним из ключевых элементов UDF, позволяющая осуществлять инкрементную запись, может свободно мигрировать по всему диску, что дает возможность избежать множественной перезаписи одних и тех же секторов (рис. 11.13). Однако, как с этим ни борются, служебные структуры данных погибают раньше всего, иногда даже после ~100 циклов перезаписи. Диск перестает читаться безо всякой надежды на его восстановление (разумеется, мы говорим о непрофессионалах).



AVDP = Anchor volume descriptor pointer - указатель на якорный дескриптор тома
ICB = Information control block - блок контроля информации
LSN = Logical sector number - логический номер сектора
VAT = Virtual allocation table - виртуальная таблица расположения

Рис. 11.13. Структура файловой системы UDF, ключевым элементном которой является VAT, свободно мигрирующая по всему диску и потому предотвращающая многократную перезапись одних и тех же участков

Не забывайте и о механических повреждениях — диски UDF к ним относятся весьма щепетильно, и одна-единственная царапина может угробить все ваши файлы. Рекламируемый механизм управления дефектами здесь не срабатывает, т. к. он не устраняет ошибки, а лишь препятствует использованию сбойных секторов!

> ***Внимание!***
>
> Никогда и ни при каких обстоятельствах не записывайте в пакетном режиме действительно ценные файлы, которые вам было бы жаль потерять! Если вы все-таки делаете это, в обязательном порядке продублируйте их на несколько дисков. Перенос файлов между компьютерами — дело другое, и UDF тут практически незаменим.

Кстати, о надежности. Производители оптических накопителей склонны преувеличивать срок их службы, зачастую давая пожизненную гарантию. Но маловероятно, что кому-либо когда-либо удастся воспользоваться этой гарантией. Общеизвестно, что при попытке возврата дефектного диска на завод-изготовитель все компании отвечают неизменным отказом, ссылаясь на нарушение условий хранения диска. У вас помещение кондиционируется? Влажность и температура с какой точностью выдерживаются? Если вы честно ответите на такие вопросы, вы получите отказ в вашей просьбе. Реально можно сказать, что даже Verbatim спустя полтора-два года обнаруживает резкое ухудшение качества чтения за счет деградации активного слоя, поэтому хранить на оптических дисках свои архивы могут только самоубийцы.

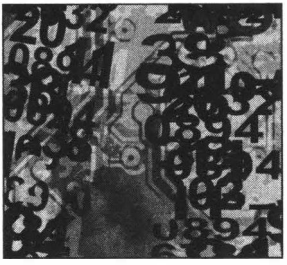
Какой привод выбрать

Для надежной работы в пакетном режиме и пишущий, и читающий приводы должны как минимум поддерживать режим MultiRead/MultiRead2, о чем свидетельствует одноименный логотип на его лицевой панели. Для соответствия этим стандартам CD-привод должен уметь читать CD-DA, CD-ROM, CD-R/RW, а привод DVD — DVD-ROM, DVD-Video, DVD-Audio и DVD-RAM вдобавок к перечисленным вариантам CD-накопителей. Разумеется, не всякому логотипу можно верить, но в текущее время все же наткнуться при покупке на привод, не поддерживающий пакетный режим, довольно трудно.

Важно задаться вопросом, чего вы ожидаете от привода, и уже в соответствии с этим подходить к его выбору. Надежной работы, высокой скорости чтения и записи, поддержки Blu-Ray или, может быть, конструкции, подходящей для "горячей замены", чтобы прочитать диски с некорректной TOC? Или, напротив, портативности? Как было замечено ранее в главе, в деле восстановления информации иной раз поможет не всякий привод, и это тоже следует учитывать.

И все-таки, несмотря на все свои многочисленные недостатки, технология пакетной записи и файловая система UDF вызывают восхищение. Это действительно прогрессивные технологии, активно подрывающие старые устои изнутри.

ГЛАВА 12



Распределенные хранилища информации

Лучший способ сохранить информацию — поделиться ею со своими друзьями. Это общеизвестный факт! В настоящей главе мы дадим рекомендации по автоматизации этого процесса и расскажем, как сделать так, чтобы компьютер самостоятельно "распределял" данные по локальной сети и не только.

Давным-давно, когда лазерных дисков и в помине не существовало, а дискеты "осыпались", как ржавчина с трубы, потеря всей информации была обычным делом. Вот и приходилось бегать по друзьям, копируя у них программы, которые они раньше копировали у вас. Исходные коды программ и офисные документы тоже часто отдавались на хранение приятелям в зашифрованном виде.

Естественно, при активном использовании такого подхода возникает то, что в ботанике называется перекрестным опылением, а у хакеров — вирусной эпидемией. Вирусы размножаются с ужасающей скоростью, как лесной пожар! Единожды попав в такой обменник, они прочно обосновываются в нем, да так, что их становится практически невозможно удалить, ведь перезаражение происходит многократно. Правда, сейчас можно упаковывать дистрибутивы любым архиватором с опцией "защиты от изменений" или использовать контрольные суммы. Стоит только сказать, что проделывать все это вручную — процесс длительный и утомительный. Двадцать первый век на дворе, как-никак! К тому же распределенное хранилище обычно получается слишком несбалансированным: какой-то файл (программа, музыка, клип) есть у всех, а какого-то нет ни у кого, и его потеря невосполнима.

Раньше приходилось бегать с дискетами и таскать винчестеры в сумке. Теперь же локальные и одноранговые сети, а также облачные хранилища позволяют обмениваться файлами, не выходя из дома. Так почему не воспользоваться преимуществами, которые несет прогресс?

Первые шаги

Распределенные хранилища информации можно создать в локальной сети, обеспечивающей скорость обмена данными хотя бы 10 Мбит/с. Осталось лишь подобрать программное обеспечение. Минималисты (к числу которых принадлежал и Крис)

могут ограничиться "общим доступом к файлам", встроенным в Windows. Начиная с Windows 2000, система поддерживает квотирование, т. е. позволяет ограничить предельный объем файлов для каждого пользователя. С квотированием уже не приходится бояться, что кто-нибудь войдет в раж и заполнит весь ваш диск целиком. Можно, например, ограничить объем общего хранилища значением 10 Гбайт и больше не беспокоиться. Остается только назначить права доступа так, чтобы все члены сети видели чужие файлы, но не могли их изменять или удалять. В Windows это совсем не сложно сделать! Достаточно открыть свойства папки и указать, что владелец имеет право выполнять любые действия, а остальные пользователи — только читать файлы.

Теперь каждый сможет зарезервировать самые ценные файлы, а то и весь диск целиком, на компьютерах своих соседей по сети! Образуется некое подобие файлообменной сети, к которой могут подключаться и другие пользователи. По действующему законодательству РФ любой потребитель может изготовить столько резервных копий, сколько ему необходимо, причем он не обязан предпринимать никаких дополнительных охранных мер, препятствующих распространению информации.

"Общий доступ" замечательно работает в сетях, насчитывающих до десяти узлов, но затем начинаются проблемы. Вы просто не можете вспомнить, на какой компьютер был зарезервирован тот или иной файл, равно как и то, какие файлы зарезервированы, а какие — нет. К тому же домашние компьютеры — это все-таки не выделенные файл-серверы, и они доступны не все время. Разбросанный по сотне узлов архив музыки/фильмов/софта практически неуязвим. Если все компьютеры отказали сразу, то это значит, что случилось что-то катастрофическое (например, землетрясение), и тут уже вам станет не до фильмов. Очевидный недостаток этого подхода состоит только в том, что для того, чтобы собрать все нужные файлы обратно на свой компьютер, потребуется длительное время. И все равно окажется, что самого нужного, как назло, не хватает, потому что некий особо ценный файл был зарезервирован в единственном экземпляре, который тоже оказался утерян. Чтобы застраховаться от таких непредвиденных случайностей, подойдем к делу творчески и воспользуемся e-Mule. Казалось бы, какая древность! Только вот количество скачиваний на страничке проекта (**https://sourceforge.net/projects/emule/**), идущее в своей категории следом за qBittorrent (а в рейтинге скачиваний за все время и вовсе занимает пятое место!), говорит о том, что он все еще вполне себе пользуется успехом и уверенно занимает свою нишу. Программа e-Mule — это клиент второй крупнейшей файлообменной сети e-Donkey, в которой можно найти все что угодно: от исходных кодов нужной программы до новейших блокбастеров, причем (если повезет, конечно) отдельные произведения на данный момент можно отыскать исключительно в этой сети. Система сама следит за целостностью файлов, показывает количество имеющихся источников и тянет со всех активных узлов сразу, равномерно распределяя нагрузку между ними. Мы можем разбивать пользователей на группы, ранжируя их по гибкой системе приоритетов, регламентировать входящий/исходящий трафик и т. д. В классическом e-Mule отсутствует возможность принудительной закачки, и все, что мы можем, — просто выложить файлы в общий каталог, дожидаясь, пока их кто-нибудь заберет. Это хорошо работает для обмена

музыкой, но для резервирования, увы, не подходит! Приходится договариваться каждый день (или хотя бы раз в неделю) просматривать содержимое общих папок всех членов сети (естественно, просмотр папок должен быть разрешен), находить новые файлы и качать их себе, если, конечно, это уже не сделал кто-то другой. Можно установить любой порог, скажем, забирать только те файлы, которые имеются менее чем у десяти источников (точная цифра зависит от размеров сети — чем больше сеть, тем выше порог). Очевидный недостаток — привязка к файлообменной сети и ее серверам, которые и без того загружены. К тому же мы не можем выборочно настраивать пропускную способность, и поэтому к нам будет ломиться толпа пользователей изо всех концов сети. Конечно, любой брандмауэр легко отсечет их, но это не решит всех проблем, самая главная из которых состоит в том, что наша маленькая приватная сеть становится видной извне.

Лучше использовать "равноправные" файлообменные сети, обходящиеся без выделенных узлов, т. е. работающие без сервера, например GNUTELLA (**https://sourceforge.net/projects/gtk-gnutella/**) или сети BitTorrent. Слегка доработав их под собственные нужды, мы получим отличное средство автоматизированного распределенного резервирования, после чего за сохранность наших данных можно будет не волноваться.

Сервер FTP или возрождение BBS

Файлообменные системы оправдывают себя только в больших сетях. В сетях среднего размера, насчитывающих несколько десятков узлов, они довольно обременительны. Поэтому для создания распределенного хранилища данных лучше всего использовать FTP-серверы.

Начнем с того, что даже в одноранговой сети не все узлы равноправны. Одни пользователи могут позволить себе держать компьютер включенным все дни и ночи напролет, другие — нет. Одни имеют емкие жесткие диски, мощный процессор и скоростной канал связи, а другие такими возможностями не обладают. Файлообменные системы уравнивают всех своих пользователей в правах, и 90% нагрузки ложится на плечи 10% клиентов. Скажите, а им это надо? Никто не захочет тянуть за собой остальных, ничего не получая взамен. В крупных сетях ситуация нормализуется за счет естественного притока новых меценатов — бескорыстных парней, стремящихся сделать что-то хорошее в жизни, ничего не ожидая взамен. Правда, со временем это стремление, как правило, проходит. Ведь как бывает? Помогаешь своим ближним, помогаешь, а они не только не поблагодарят, а еще и напакостят. Ладно, не будем о грустном, а лучше перейдем к сути дела.

Крупные узлы небольшой приватной сети могут поддерживать собственные FTP-сервера, открытые на закачку (upload) и на загрузку (download), которые будут доступны всем остальным пользователям. Это тоже распределенное хранилище, но, в отличие от описанных ранее, более надежное и быстрее работающее. Мы можем резервировать данные только на те серверы, которые оснащены источниками бесперебойного питания, отказоустойчивыми массивами RAID и прочими достижениями прогресса. Поскольку квоты на таких серверах, как правило, достаточно ве-

лики, никакой необходимости разбрасывать файлы по десяткам узлов уже нет. Достаточно продублировать файлы дважды или, на худой конец, трижды.

Остается лишь решить один маленький вопрос — с какой стати кто-то будет содержать FTP-сервер? Брать за это деньги нелепо, да и смысла нет. Не окупится. А на чистом энтузиазме далеко не уедешь. На самом деле, собственный FTP-сервер — это лучший способ раздобыть редкую музыку/фильмы/варез. Что резервируют пользователи? Самые ценные файлы, которые жалко потерять и которые они с большим трудом откопали в Сети (или купили за огромные деньги). И все это они добровольно несут нам, только успевай подставлять жесткий диск! Ну чем жизнь не малина? Давным-давно, когда Интернета еще не существовало, а софт считался общенародным достоянием (но собирать его приходилось буквально по битам), основными "малинниками" были электронные доски (BBS) или, проще говоря, компьютеры с модемом, принимающим входящие звонки и складирующим закачиваемые файлы. Сисоп (системный оператор) отбирал самые "вкусные" файлы, а остальные отправлял в мусорную корзину.

Так почему бы не возродить эту традицию, используя FTP-серверы для обмена файлами?

Массивы RAID

Что такое контроллер RAID, знает каждый (сейчас его можно купить в любом магазине). Упрощенно говоря, это такое устройство, которое позволяет писать сразу на несколько дисков одновременно. Если на диски пишутся разные данные — скорость обмена пропорционально возрастает. Если дублируются те же самые данные — возрастает надежность. Массивы RAID могут быть как программными, так и аппаратными, а сами образующие массив носители не обязаны быть сосредоточены на одном и том же компьютере. Чувствуете, куда я клоню?

С точки зрения программного RAID нет никакой разницы между диском, подключенным к локальному компьютеру через интерфейсы SAS или SATA, и диском, обменивающимся данными через сеть. Объединив несколько логических дисков в виртуальный массив RAID, мы получим отказоустойчивую систему — практичную и удобную. Мы можем использовать диски различной геометрии и даже различной емкости, причем никто не обязывает нас отводить под RAID-хранилище весь диск целиком! Достаточно выделить любую часть дискового пространства по выбору.

Как это можно реализовать на практике? Первое, что приходит в голову, — это использовать часть емкости жестких дисков под хранение избыточной информации (например, кодов Рида — Соломона, помогающих восстановить данные в случае аварии). Тогда при относительно небольших накладных расходах мы сможем восстановить любой из жестких дисков членов сети даже при полном его разрушении лишь за счет одной избыточной информации, распределенной между остальными компьютерами. Более надежного хранилища для ваших данных нельзя и придумать! Подобная схема давным-давно была мною реализована в локальных сетях нескольких фирм. Она доказала свою живучесть, гибкость и надежность. Необходимость

в постоянном ручном резервировании при этом отпадает, что для домашних пользователей более чем актуально!

Единственный минус программного RAID — его невысокая производительность. В частности, поставив программный RAID на сервер, обрабатывающий тысячи запросов ежесекундно и интенсивно модифицирующий большое количество файлов, мы не выиграем ничего. Однако ведь само понятие "производительность" относительно, и при достаточно быстром процессоре кодирование/декодирование информации вполне реально осуществлять и "на лету", безо всяких потерь в пропускной способности! Если операции чтения доминируют над операциями записи, то ставить программный RAID очень выгодно, поскольку контроль целостности считываемой информации осуществляется на "железном" уровне самим приводом и при систематическом кодировании (информационные слова — отдельно, байты четности — отдельно) декодеру Рида — Соломона нет никакой нужды как-то вмешиваться в этот процесс. Помощь его требуется лишь тогда, когда часть информации оказывается безнадежно разрушена, что случается, прямо скажем, не так уж часто. Так что, право же, не стоит перекармливать фирмы, специализирующие на выпуске аппаратных RAID, тем более что они все равно не уделят достаточного внимания домашним пользователям и малым предприятиям.

Варьируя размер блоков корректирующих кодов, мы получим лучшую или худшую защищенность при большей или меньшей избыточности информации. Действительно, пусть у нас есть N секторов на диске. Тогда, разбив их на блоки по 174 сектора в каждом и выделив 3 сектора для хранения контрольной суммы, мы сможем восстановить по меньшей мере N/174 секторов диска. Исходя из средней емкости диска в 1000 Гбайт (что соответствует 2 097 152 000 секторам), мы сможем восстановить до 12 052 590 секторов даже при их полном физическом разрушении, затратив всего лишь 2% дискового пространства для хранения контрольных сумм. Согласитесь, что винчестеры редко отказывают столь стремительно, что корректирующих способностей кодов Рида — Соломона оказывается недостаточно для восстановления информации. Разумеется, это справедливо только в тех случаях, если симптомы приближающейся катастрофы замечены своевременно и коэффициент чередования выбран правильно. Правильный выбор коэффициента чередования означает, что сектора, принадлежащие одной и той же пластине жесткого диска, должны обслуживаться разными корректирующими блоками, в противном случае при повреждении поверхности одной из пластин возникнет групповая ошибка, уже не исправимая данной программой.

А как быть, если погибнет весь жесткий диск целиком? Наиболее разумный выход — создать массив из нескольких дисков, хранящих полезную информацию вперемешку с корректирующими кодами. Главный минус такого подхода — его неэффективность на массивах, состоящих из небольшого количества жестких дисков. Разумный минимум: четыре информационных диска и один контрольный, тогда потеря любого из информационных дисков компенсируется оставшимся в живых контрольным. В случае потери контрольного диска его очень просто заменить на новый, с последующим пересчетом всех контрольных кодов. Правда, одновременный выход двух дисков из строя — это уже серьезно. Массив из пятнадцати

дисков, двенадцать из которых — информационные, а оставшиеся три — контрольные, намного более отказоустойчив и допускает одновременный крах двух *любых* дисков, а при благоприятном стечении обстоятельств — и трех.

Подробнее о кодах Рида — Соломона можно прочитать в книге Криса Касперски "Техника защиты CD от копирования". Исходные коды простейшего кодера/декодера, который подойдет для создания собственного драйвера RAID, можно найти в электронном архиве к этой книге, выложенном на сайте издательства БХВ.

Облачные хранилища

Конечно же, нельзя обойти стороной такое веяние современности, как облачные диски. Это поистине прогрессивное и безумно удобное изобретение, позволяющие иметь при себе необходимые файлы всюду, где только есть доступ к глобальной сети, не расходуя на их хранение внутреннюю память устройств. К тому же это очень простой способ синхронизации файлов между "зоопарком" устройств одного пользователя.

Ваши драгоценные файлы будут храниться под бдительным наблюдением профессионалов во внушительных центрах обработки данных в географически удаленном от вашего расположения месте! Что может быть лучше для их сохранности? Тем не менее основной недостаток облачных хранилищ следует из их главного преимущества и заключается в их полной зависимости от наличия доступа в Интернет. Впрочем, этот момент ничуть не умаляет достоинств облачных хранилищ.

Выбор таких хранилищ велик, причем он не ограничивается одними Яндекс.Диском, Dropbox, Google Drive да OneDrive. Есть и более узкоспециализированные платформы, например SoundCloud и в некотором смысле GitHub. Все эти платформы отличаются количеством бесплатно предоставляемого места, операционными системами, поддерживаемых приложениями-клиентами, интегрированными сервисами и оптимизацией под разные задачи. Некоторые облачные диски позволяют создавать и править документы, электронные таблицы и презентации, как говорится, не "отходя от кассы", а также просматривать содержимое файлов (в том числе медиа), не загружая их на компьютер. Рассмотрим далее возможности некоторых облачных сервисов.

Google Drive

Это один из самых функциональных на текущий момент облачных дисков (**https://drive.google.com**). После регистрации доступны бесплатные 15 Гбайт хранилища, при этом документы в "родном" формате (Google Документы) не занимают места для пользователя. Сервис по умолчанию присутствует на устройствах под управлением ОС Android. Google Drive обеспечивает глубокую интеграцию с остальными сервисами Google — документами, таблицами, формами и множеством других, а также позволяет подключать другие приложения для работы с различными файлами из браузера без их скачивания. В диске есть управление версиями файлов: версии хранятся до 30 дней либо пока их количество не превысит 100 штук; при этом

опцией "Никогда не удалять" можно уберечь необходимую версию файла от этой участи.

Приложение для доступа к Google Drive "Автозагрузка и синхронизация" существует для ОС Windows и macOS. Настроек в клиенте не так много (рис. 12.1). Самое главное, что он делает, — автоматически синхронизирует выбранные папки на компьютере с облаком и позволяет без лишних движений открывать общий доступ к файлам. Простым переносом необходимых файлов в проводнике можно отправить их прямиком в облако. При подключении USB-устройств программа предлагает сохранить в Google Drive данные с них.

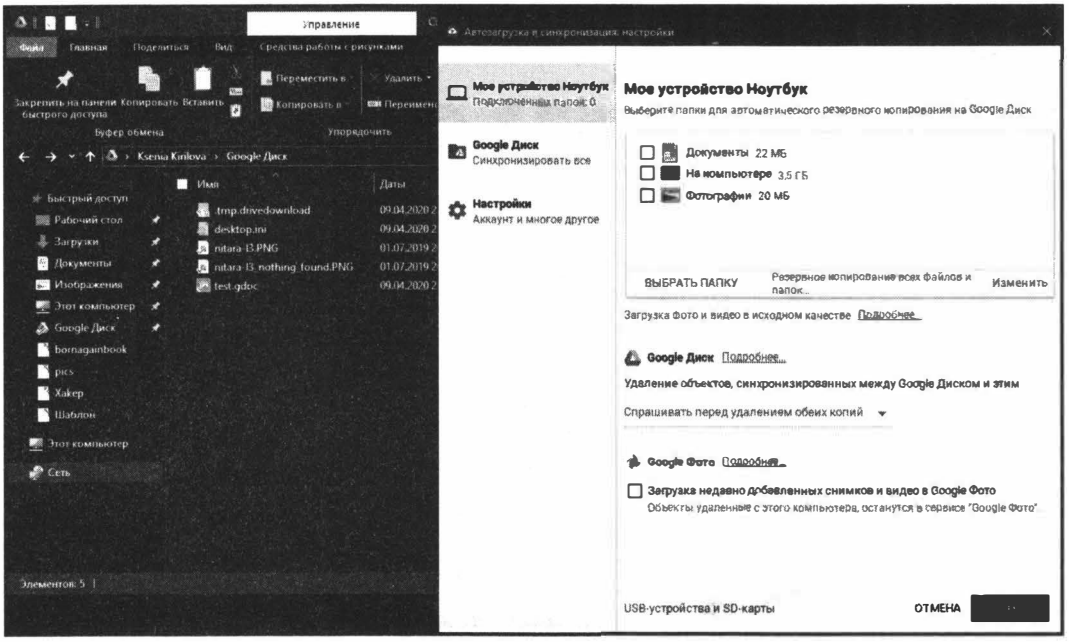


Рис. 12.1. Окно настроек приложения "Автозагрузка и синхронизация" и соответствующая папка в проводнике

Яндекс.Диск

Этот облачный сервис доступен по адресу **https://disk.yandex.ru**. После регистрации становится доступно 10 Гбайт, а после оплаты персональной подписки можно получить от 100 Гбайт до 3 Тбайт и некоторые дополнительные возможности, например просмотр истории изменений файлов до 90 дней (бесплатно доступны лишь версии файла за 14 дней). В диск интегрированы сервисы Office Online — прямо в браузере можно создавать и править электронные документы, хранящиеся на диске.

В клиенте для Windows можно настроить папки для автоматической синхронизации с облаком, создавать, редактировать и отправлять снимки экрана и заметки. Для удобства можно создать небольшую область на рабочем столе, при перетаски-

вании файлов в которую они будут загружаться в облако. Фотографии и видеоклипы при включенной автозагрузке в мобильном приложении (iOS, Android) не занимают места на Яндекс.Диске для пользователя. Помимо клиентов под Windows, macOS и мобильные платформы Яндекс предлагает консольный вариант для ОС Linux в виде пакетов deb и rpm (рис. 12.2) с возможностью использовать сторонние графические интерфейсы (например, yandex-disk-indicator из пакета YD-tools).

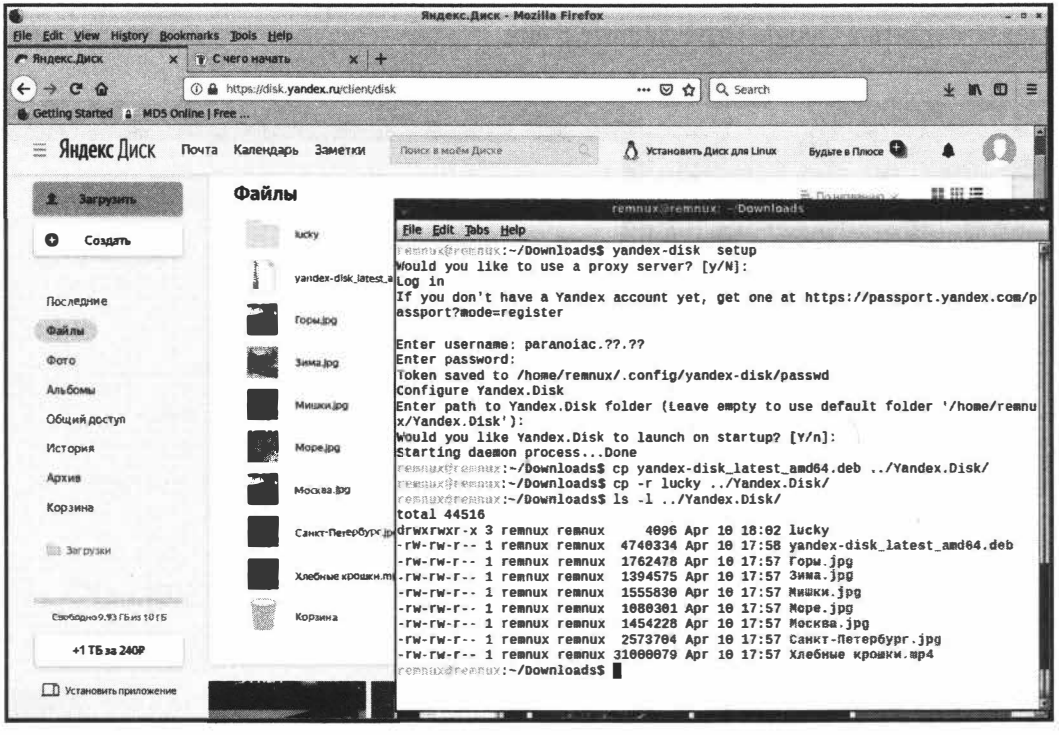


Рис. 12.2. Консольный клиент Яндекс.Диска для ОС Linux

Box

Это уже более серьезный вариант для коллективного хранения данных в облаке в корпоративных сетях (рис. 12.3). Сразу после регистрации доступны бесплатно 10 Гбайт при максимальном размере файла в 250 Мбайт. После покупки подписки к вашим услугам предоставляется от 100 Гбайт пространства и просмотр истории файлов. Для корпоративных клиентов предлагаются отдельные тарифные планы с разными лимитами и намного бóльшим набором возможностей, ознакомиться с которыми можно на сайте **https://www.box.com**. Сервис позволяет ставить пароли на папки и просматривать содержимое файлов некоторых форматов прямо в браузере. Существуют клиенты Box под Windows, macOS и различные мобильные платформы.

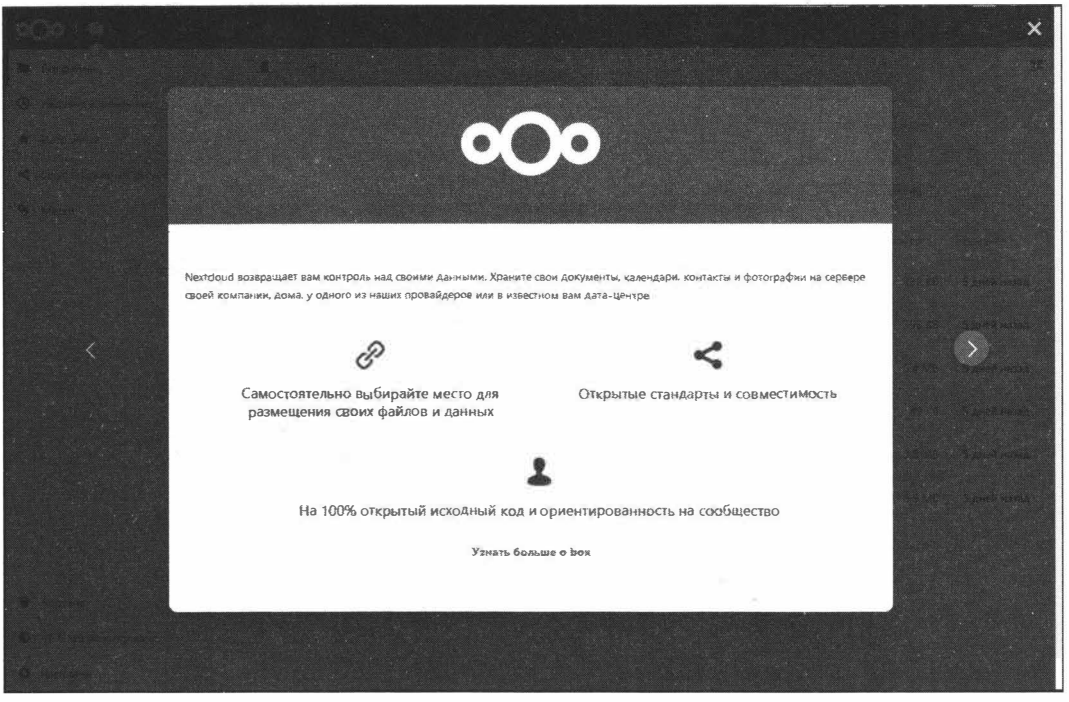


Рис. 12.3. Веб-интерфейс Box

Dropbox

Данный сервис (**https://www.dropbox.com/**) представляет собой не только облачный диск, но и платформу для совместной работы над различными задачами с соответствующими инструментами (например, функция Dropbox Badge для совместной работы с документами Microsoft Office). После регистрации становятся доступны бесплатно 2 Гбайт хранилища. Из интересных особенностей сервиса можно назвать функцию "запроса файлов", позволяющую любому загрузить в ваше хранилище Dropbox файлы по ссылке-запросу.

При первом запуске клиентского приложения предлагается выбрать режим Dropbox Basic, когда файлы целиком синхронизируются с облаком, или Dropbox Plus, где для экономии места на компьютере хранится только информация о файлах в облаке, а также доступно 2 Тбайт хранилища (рис. 12.4). В приложении можно настраивать импорт фотографий, прокси, ограничения на пропускную способность. Также при настройках по умолчанию при подключении съемных носителей выполняется импорт фото и видео с них в облако. Клиент для ОС Linux можно скачать в виде пакетов deb, rpm или исходных кодов.

rclone

Отдельного внимания заслуживает консольная утилита для работы с облаками rclone (рис. 12.5), суть которой разработчики емко описывают как "rsync для облач-

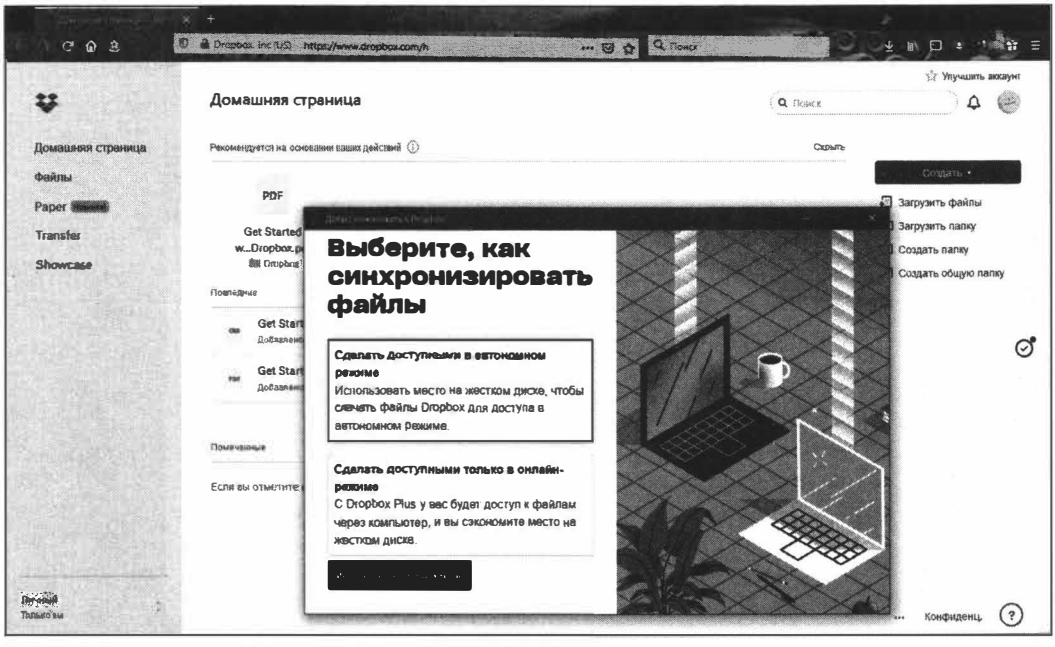


Рис. 12.4. Сайт и приложение Dropbox

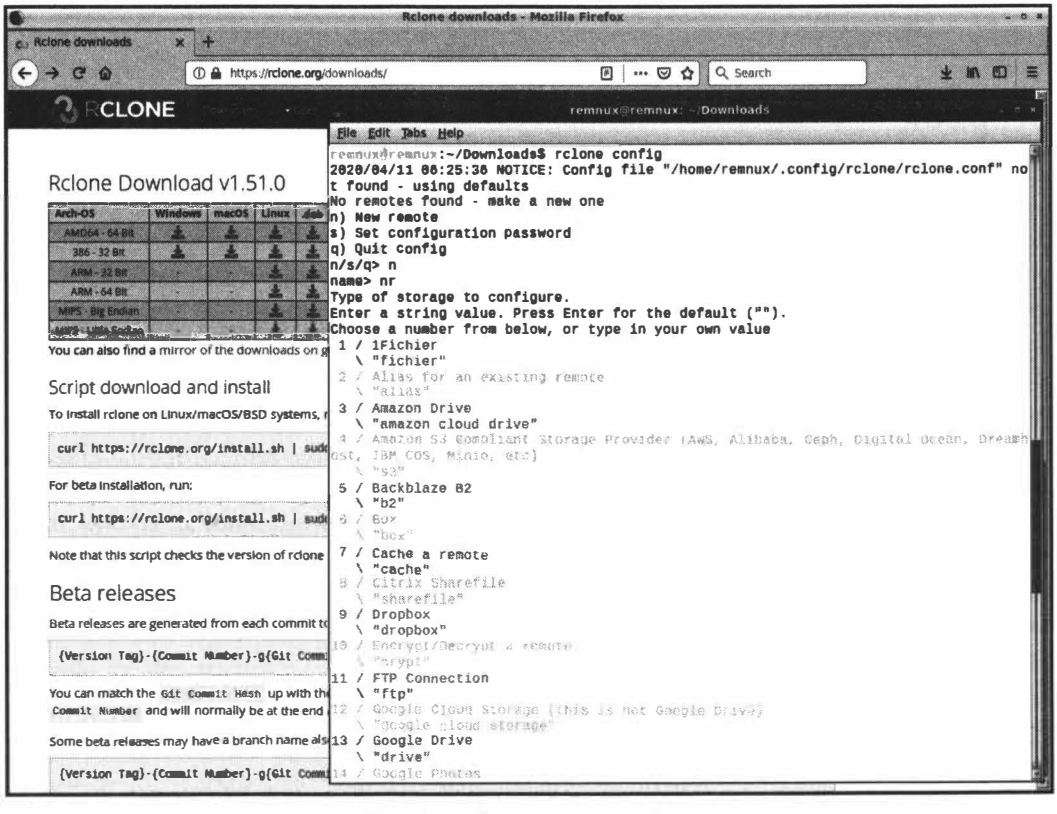


Рис. 12.5. Утилита rclone в Linux

ных хранилищ". Она существует в версиях под ОС Linux, Free/Net/OpenBSD, Solaris и даже Plan9, Windows, а также macOS. Все эти версии можно найти в разделе загрузок проекта по адресу: **https://rclone.org/downloads/**.

Программа rclone не привязана к определенному хранилищу и может работать с огромным количеством сервисов, среди которых, помимо уже рассмотренных ранее: OneDrive, Amazon S3, Backblaze, MEGA и работа по FTP/SFTP (полный список приводится на домашней страничке проекта). Другая примечательная особенность rclone — возможность переноса файлов из одного облачного хранилища в другое. Вместе с пакетом, а также на сайте поставляется подробная документация.

Заключение

Мы рассмотрели несколько типов распределенных систем синхронизации и резервирования данных. Не забывайте, что простая синхронизация может принести не только пользу: если синхронизируемые файлы окажутся зашифрованы на вашем компьютере, то именно в таком виде они попадут в резервное хранилище. Тем не менее теперь у вас точно есть все необходимое, чтобы организовать собственную полноценную систему резервного копирования и быть всегда начеку. Так что, держайте!

ПРИЛОЖЕНИЕ

Описание прилагаемого архива

Разрушение данных — это самое страшное, что только может случиться с вашим компьютером. В данном архиве собрано большое количество справочной информации, видеоклипы, иллюстрирующие процесс восстановления данных, а также иллюстрации и исходные коды утилит авторской разработки, предназначенных для восстановления данных.

Многие утилиты, собранные в этом архиве, предназначены для восстановления данных, а также для анализа и копирования защищенных CD. Обратите внимание на то, что обход защиты от копирования не является нарушением законодательства об авторских правах! Действующее законодательство многих стран явным образом разрешает создание резервных копий защищенных носителей. Например, корпорация PHILIPS, являющаяся одним из разработчиков технологии записи на CD, — активный противник всяческих отклонений от Стандарта и настаивает на том, что компакт-диски, защищенные от копирования за счет использования нестандартного формата, не должны маркироваться логотипом "Compact Disc".

Архив, прилагаемый к этой книге, можно скачать с сайта издательства по адресу: **ftp://ftp.bhv.ru/9785977566810.zip**, или со страницы книги на сайте **http://bhv.ru**. Архив содержит следующие материалы:

- FIGURES — цветные иллюстрации ко всем главам данной книги;
- LISTINGS — исходные коды всех примеров, приведенных в книге, которые вы можете свободно использовать по собственному усмотрению;
- SRC — исходный код и демонстрационные примеры, предназначенные для восстановления данных с нечитаемых CD, в том числе:
  - каталог RS.LIB — примеры работы с CD на секторном уровне;
  - каталог RS.SIMPLE — элементарные примеры, иллюстрирующие принципы действия кодов Рида — Соломона;
  - каталог SCSI.ALT — исходный код драйвера, демонстрирующий выполнение машинных команд `IN/OUT` с прикладного уровня;
  - каталог SCSI.LIB — ряд утилит, разработанных лично Крисом Касперски и предназначенных для работы с защищенными от копирования компакт-дисками;
- UTILITIES — набор небольших, но полезных утилит, предназначенных для посекторного копирования CD, на которых записаны файлы некорректной длины и с некорректными начальными секторами.

Предметный указатель

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Я

Информация на жестком диске, твердотельном накопителе или съемном носителе может погибнуть по разным причинам. Чаще всего уничтожение ценных файлов — следствие физического повреждения устройства, проникновения троянца-шифровальщика, программного сбоя или необдуманных действий пользователя. Но это не катастрофа! Легендарный хакер Крис Касперски подробно рассказывает в своей книге, как самостоятельно, без дорогостоящих услуг сервисных центров восстановить утраченные данные в Windows и Linux, как отремонтировать неисправный жесткий диск и избежать подобных проблем в будущем, грамотно настроив резервное копирование.

Крис Касперски — известный российский хакер, легендарный IT-журналист и писатель, автор множества бестселлеров, посвященных компьютерным технологиям. В прошлом — редактор рубрики «Взлом» популярного журнала «Хакер», бывший сотрудник антивирусной компании McAfee.

Валентин Холмогоров — российский IT-журналист, автор более 40 книг по компьютерным технологиям и информационной безопасности. Бывший сотрудник антивирусной компании «Доктор Веб», в настоящее время — редактор рубрики «Взлом» журнала «Хакер».

Ксения Кирилова — программист, исследователь безопасности Linux-систем, сотрудник одной из российских антивирусных компаний, постоянный автор журнала «Хакер».

Материалы на www.bhv.ru

Цветные иллюстрации и дополнительные материалы можно скачать по ссылке ftp://ftp.bhv.ru/9785977566810.zip, а также со страницы книги на сайте bhv.ru.

bhv®

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

ISBN 978-5-9775-6681-0