DONG HUA UNIVERSITY

硕士学位论文

MASTER DISSERTATION

目题文论：[illegible]

业专修论：[illegible]

名姓体导：[illegible]

间期成完论：[illegible]

期日交论：[illegible]

硕士学位论文

学校编号： 10255
学号： 201398

基于Windows 9x/NT操作系统的磁盘备份与恢复的研究与实现

硕士研究生: 张宗伟

导师: 邢传鼎（教授）

专业: 计算机应用技术

答辩日期: 2004年3月

东华大学信息学院计算机系

Master's Degree Thesis

School Code: 10255
Student ID: 201398

Research and Implementation of Disk Backup and Restore Operations On Windows 9x/NT Operating Systems

Candidate: Zhang Zongwei

Supervisor: Xing Chuanding (prof.)

Major: Computer Science

Date: March 2004

Information College, Donghua University, P. R. China

附件一：

东华大学学位论文原创性声明

本人郑重声明：我恪守学术道德，崇尚严谨学风。所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已明确注明和引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品及成果的内容。论文为本人亲自撰写，我对所写的内容负责，并完全意识到本声明的法律结果由本人承担。

学位论文作者签名：张宗伟

日期：2004 年 3 月 4 日

附件二：

东华大学学位论文版权使用授权书

学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅或借阅。本人授权东华大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密 □，在 ____ 年解密后适用本版权书。

本学位论文属于

不保密 ☑。

论文作者：张宗伟

论文指导老师：

中文摘要

在DOS系统环境下，经常会使用GHOST软件对磁盘进行数据备份，从而当系统崩溃的时候，或是克隆分区安装新系统的时候，能够快速恢复源磁盘分区。在磁盘数据备份恢复的领域中，GHOST软件已经具有了相当高的知名度。本论文就是根据实际在公司中参与完成的项目为背景，以著名磁盘备份恢复软件GHOST为原型，独立检索资料撰写完成的。实际完成后的软件可以实现在Windows 9x操作系统以上的版本中，对所有Windows可识别支持的FAT16、FAT32、NTFS文件系统进行磁盘备份和恢复操作，而且对GHOST软件已经实现的功能进行了合理有益的补充，在数据安全性、完整性方面做出了有益的增强。因为实用程序是基于Windows多任务、保护模式的操作系统上，所以作成的应用软件不仅具有Windows的界面友好性，而且它的备份速度比纯DOS下的GHOST软件要快，特别是在Windows NT内核的操作系统下面，备份恢复速度的提高是非常明显的。

整篇论文以循序渐进的方式，由浅入深。首先概要地介绍了磁盘的发展历史，磁盘的物理存储结构和磁盘读写的寻址模式。然后，简单介绍了本文重点研究的FAT16、FAT32和NTFS文件系统功能和优缺点，给出了一个基本的介绍。接着，对磁盘的分区结构进行了详解，结合了在众多操作系统中（包括Windows）被广泛使用的分区应用程序FDISK程序的使用，对其划分后的多组分区数据表进行了实验研究，总结归纳了FDISK分区后的分区表数据规则，为日后的分区表改写作了理论准备。再接着，具体讨论了各种文件系统类型和相关结构，对FAT16、FAT32文件系统作出了详尽细致的分析。特别对于由于商业原因微软没有公布内部结构的NTFS分区，论文结合实际数据详细分析了这个神秘分区。把在实际磁盘备份恢复操作中，被巧妙使用到的FAT文件系统下的FAT表结构部分和NTFS文件系统下的MFT表、文件属性等等文件系统结构特点都作了着重说明。在了解了分区表和各种文件系统结构，得到了具体的备份恢复解决方案后，文章具体结合目前在国内被PC广泛使用的WINDOWS 9x和WINDOWS NT两个不同内核的操作系统，讨论分析了其在磁盘读写方面的异同性，引出了在Windows 9x操作系统下独有的中断技术、THUNK技术、Windows 2000操作系统下的统一硬件抽象层。这些技术在读写磁盘时都得到了具体而有效的运用。

论文在其它一些虽非主体功能实现，但在实际应用中也不容忽视的重要环节，也用了专门的章节作了详细论述。主要包括恢复后分区的正常启动引导，盘符号的正确识别，数据压缩检验原理应用等等方面。

为了能更好的实现磁盘备份恢复操作，论文列示了很多在研究过程中通过实验积累的测试数据，力求能用详实的数据，对这些数据进行的综合分析，来获得较佳的读写数据内存块大小，较快的数据读写速度，最正确的读写分区表数据的方法，从而提高应用软件的整体性能。

本文所介绍的技术已经被运用在上海易仁信息技术有限公司的相关磁盘备份恢复软件中。该软件支持所有基于Windows 9x和Windows NT内核的操作系统，支持IDE硬盘，USB活动硬盘。

关键词：簇、扇区、寻址模式、**FAT**、**NTFS**、文件类型、分区表、**Thunk**、**int13**、标准设备读写、**CRC32**、**LZO**

ABSTRACT

Under the DOS Operating System, the software GHOST is often used to backup and restore the data in the disk. At the crash time or cloning partition to reinstall a new system time, it can restore the source partition very fast. So GHOST has gotten a very high award in the restorage field. According to the real project, this paper takes the famous disk utility software GHOST as the original module. All the correlative knowledge is collected individually. The software can backup and restore all the Windows recognized file system volume. Including FAT16, FAT32, NTFS file system under the Windows Operating System which Windows 9x core kernel based on, or above. In addition, the software makes some reasonable improvement compared with the GHOST, special in the data security and integrity. Because Windows Operating System is multi-task Operating system using the protected mode, the software can backup or restore the data in a faster speed than GHOST. In the Windows Operating System based on NT core kernel, the speed is much faster specially.

The paper illuminates the topic step by step. Firstly, it makes the brief introduce about the disk development history, disk physical storage structure, and the disk I/O access mode in a simple way. Secondly, it talks about some kinds of file system type. Such as the FAT16, FAT32 and NTFS file systems, to give a beginning introduce. Thirdly, it discusses the disk partition table carefully, with the famous program fdisk's application, and concludes the fdisk program writing partition table method. Fourthly, it makes much more words in the certain file systems and correlative structures, analysing the FAT16, FAT32 file system in detail. Special in NTFS, which is not published by Microsoft Company for the business reason, the paper will discover this secret file system with many test datas. Making the FAT table data structure under the FAT file system, the MFT table data structure and data property under the NTFS file system more clear. These structures will be used flexibly in the backup and restore operation. At the last, the paper will talk about the difference in disk I/O access method between Windows 98 and Windows 2000, the two different kernel Operating Systems used in the PC field frequently in the native country. In

the same time, the paper will discuss the interrupt technology, THUNK technology under the Windows 98 and the hardware independence under the Windows 2000. These technologies or character is used in accessing the disk with full efficiency.

After presenting the main functions and their realizing principium, the paper makes up the special charters to talk about some other important fields. Such as, booting successfully after restoring the partition data, recognizing the volume characters, compressing data and checking data integrity, etc.

To make the backuping and restoring operation better, the paper presents much test datas, and want to use the reliable data and the analyse on these data to get the best I/O memory size, the fastest I/O speed and the correct partition table modifying method. They will improve the software performance wholly.

The technology discussed by the paper has been used in the certain project of the Shanghai Easier Information Technolongy Limited Company about the disk backup and restore operation. This software supports all Windows Operating System which Windows 9x and Windows NT core kernel based on and supports the IDE HDD and USB Storage Disk.

Author: Zhang Zongwei (Computer Application)

Supervised by: Prof. Xing Chuan ding

Key Words: Cluster、Sector、Disk Access Mode、FAT、NTFS、File System Type、Disk Partition Table、Thunk、Int13 Extension、Standard Device Driver、CRC32、LZO

目录

第一章 绪论

1956年9月世界上第一块硬盘IBM 350 RAMAC诞生，当时的这块硬盘由50张24英寸大小的盘片组成，重量达50公斤，但容量却仅有5MB。由于在当时对数据存储处理的要求还不是很大，而且计算机的处理性能还很低，所以，对当时的数据存储而言，已经是足够满足需求了。可是，随着计算机的性能在新技术的推动下，日益完善提高，硬盘作为数据存储重要媒介的功能日益体现。对硬盘容量要求日益突出。作为现在PC的重要组成部分。硬盘技术发展经历了温彻斯特（Winchester）技术、薄膜磁头技术，直到不久前IBM的高密度存储技术“仙尘（Pixie Dust）”等等，都对提高硬盘的容量起到了强有力的推动作用。从而使硬盘能够在经历了40多年后，从最初的5M容量达到了现在的120G以上。使硬盘成为了计算机的大数据容量储存部件，越来越多的数据被存储到了硬盘上，其中不乏很多对企业乃至政府相当重要的数据，万一数据丢失或者泄漏，对政府企业将会造成无可估量的损失。对于后者，现在很多操作系统（如Windows NT，Linux，Unix等等）都做了相应的访问权限控制，可以在很大程度上保护系统数据不被泄漏。可是对于前者，由于一些误操作等人为原因和一些不可避免的自然灾害而造成的数据丢失，就要求能够常常对磁盘分区数据进行备份。从而能在系统遭到毁灭性损失时，可以恢复最新备份数据，把损失降低到最低。而且源分区是可引导的，在恢复后，要求此分区也是能够被引导启动的。本文就是研究在Windows操作系统下，磁盘分区的备份与恢复。并在一定程度下。进行了功能扩展。把很多GHOST软件功能实现在Windows操作系统，具有了比较好的界面友好性、扩展功能和卓越的速度性能。论文中研究分析了磁盘物理结构、逻辑结构、读写模式、分区表数据结构、Windows系统下文件系统类型和FAT、NTFS文件类型的结构、标准文件读写方式、中断技术、THUNK技术等等磁盘读写相关技术。在数据安全和压缩方面也对使用的LZO压缩算法和CRC32数据校验作出了简要说明。结合备份恢复中所要涉及到的知识点，本文根据软件搭建的平台在读写硬盘方式上的异同点，运用到了实际完成的软件，给出一个完整的解决方案。

全文共分为两个部分。1） 磁盘物理结构及数据组织方式。对磁盘发展历史、物理结构、数据存储访问模式、Windows下主要文件系统概述做出了简要的分析。使用FDISK磁盘分区程序，结合多组实际磁盘分区表数据，归纳总结了FDISK程序的算法，

从实际数据中进行归纳总结，解决了资料匮乏的难题，把FDISK读写分区表算法运用到实际编写的程序中。2）不同Windows平台下磁盘备份与恢复的实现与研究。从两个不同方面由浅及深，详细阐述了磁盘的结构，磁盘备份恢复操作，特别是对于微软从未公布过的NTFS文件系统结构，进行了详细的叙述，并把分析后的成果运用到了具体软件实现过程中。

第一部分 磁盘物理结构及数据组织方式

第二章　磁盘物理结构以及寻址模式概述

2．1　磁盘物理结构

硬盘的英文名称:Hard Disk Drive (简称HDD)，是一种在电脑系统中用来储存资料的周边设备。虽然目前市面上有许多可以存资料文档的周边设备，如CD-R、MO、ZIP等，但是硬盘的速度快、容量高、价格便宜的特性，使得它仍然是目前最经常使用的一种资料储存设备。硬盘是现在计算机上最常用的存储器。

它的工作原理与平时家中使用的录放影机、录音机等都是一样的，都是应用磁性物质的物理特性。所不同的是前者都是将磁性材料涂在磁带上，而后者是把磁性材料涂在硬盘的磁盘片上。

硬盘经过几年的市场竞争淘汰之后，目前只剩下了ATA(IDE)以及SCSI两种媒介的硬盘，前者普遍应用于个人电脑应用环境，后者则应用于较高端的服务器、工作站以及大型主机上。如果应用环境只是单机的个人电脑，IDE硬盘已经能满足应用与存储所需。

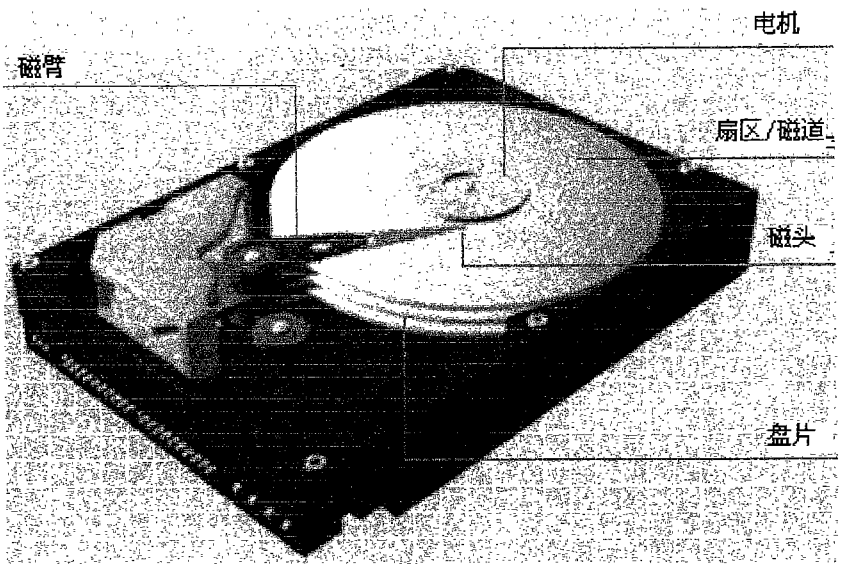
每次都会有新的存储装备被推出，一些专家或是分析家会大胆预测硬盘终将被这些新的储存装置所取代。然而十几年过去了，由于硬盘技术不断发展，甚至比CPU处理速度和软件发展速度还快。並且将价格压低到了平民普及化的水准，已使原来相当贵族化的硬盘转变成每台家用电脑必备的储存外设。

计算机具有高速分析处理数据的能力。而这些数据都被以文件的形式存储在硬盘里。在读取相应的文件时，必须要给出它相应的规则。这就产生了分区概念。分区从实质上说就是对硬盘的一种格式化。当创建分区时，就已经设置好相应分区表值参数，指定了硬盘主引导记录(即Master BootRecord，一般简称为MBR)和引导记录备份的存放位置，即Fdisk命令来实现。而对于文件系统以及其他操作系统管理硬盘所需要的信息则是通过之后的高级格式化，即Format命令来实现。

硬盘分区后，将会被划分为面、磁道和扇区。需要注意的是，这些只是个虚拟的概念，并不是真正在硬盘上划道子。先从面说起，硬盘一般是由一片或几片圆形薄膜叠加而成。每个圆形薄膜都有两个“面”(Side)，这两个面都是用来存储数据的。按照面的多少，依次称为0面、1面、2面……由于每个面都专有一个读写磁头，也常用0

头(head)、1头……称之。按照硬盘容量和规格的不同，硬盘面数（或头数）也不一定相同，少的只有2面，多的可达数十面。各面上磁道号相同的磁道合起来，称为一个柱面（cylinder）。

图2-1 硬盘物理结构



读写硬盘时，由于磁盘是旋转的，则连续写入的数据是排列在一个圆周上的。这样的圆周被称为一个磁道（Track）。如果读写磁头沿着圆形薄膜的半径方向移动一段距离，以后写入的数据又排列在另外一个磁道上。

根据硬盘规格的不同，磁道数可以从几百到数千不等；一个磁道上可以容纳数KB字节的数据，而主机读写时往往并不需要一次读写那么多，于是，磁道又被划分成若干段，每段称为一个扇区（Sector）。一个扇区一般存放512字节的数据。扇区也需要编号，同一磁道中的扇区，分别称为1扇区，2扇区...。这里需要注意的是，硬盘在划分扇区时，和一般的软盘有一定的区别。软盘的一个磁道中，扇区号依次编排，即2号与1号相邻，3号与2号相邻，以此类推。而在硬盘的一个磁道中，扇区号是按照某个间隔跳跃着编排的。举一个例子：在某个硬盘上，以实际存储位置而论，2号扇区并不是1号扇区后的第一个，而是第5个，3号扇区又是2号扇区后的第5个，以此类推。这个"5"就是一般常说的交叉因子。当然，这个交叉因子的设定并不是绝对的，每个种类的硬盘会根据自身的情况加以变化。选择适当的交叉因子，可使硬盘驱动器

胸部
前翅
後翅（図鑑）
尾
体節

读写扇区的速度与硬盘的旋转速度相匹配，提高存储数据的速度。计算机对硬盘的读写，处于效率的考虑，是以扇区为基本单位的。即使计算机只需要硬盘上存储的某个字节，也必须一次把这个字节所在的扇区中的512字节全部读入内存，再使用所需的那个字节。不过，在上文中也提到，硬盘上面、磁道、扇区的划分表面上是看不到任何痕迹的，在这些由计算机存取的数据的前、后两端，都另有一些特定的数据，这些数据构成了扇区的界限标志，标志中含有扇区的编号和其他信息。计算机就凭借着这些标志来识别扇区[1]。

2．2 磁盘寻址模式

硬盘的寻址模式，通俗地说，就是主板BIOS通过什么方式，查找硬盘低级格式化划分出来的扇区的位置。硬盘是通过某种寻址模式找到系统所需要的资料的，寻址模式就好象日常生活中的地址，当要找地址的时候，一定要靠路名及门牌号码来寻找一样，而硬盘定址就是：将硬盘资料所在的位置加以定义的方式。

在老式硬盘中，由于每个磁道的扇区数相等，所以外道的记录密度要远低于内道，因此会浪费很多磁盘空间 (与软盘一样)。为了解决这一问题，进一步提高硬盘容量，人们改用等密度结构生产硬盘。也就是说，外圈磁道的扇区比内圈磁道多。采用这种结构后，硬盘不再具有实际的3D参数，寻址方式也改为线性寻址，即以扇区为单位进行寻址。为了与使用3D寻址的老软件兼容 (如使用BIOS Int13H接口的软件)，在硬盘控制器内部安装了一个地址翻译器，由它负责将老式3D参数翻译成新的线性参数。这也是为什么现在硬盘的3D参数可以有多种选择的原因(不同的工作模式，对应不同的3D参数，如LBA、LARGE、NORMAL)。

CHS（或称为Normal）模式： 适应容量≤504MB的硬盘。

LARGE（或称LRG）模式： 适应504MB≤容量≤8.4GB的硬盘。

LBA（Logical Block Addressing）模式： 适应容量≥504MB的硬盘，但BIOS需支持扩展INT13H，否则也只能适应≤8.4GB的硬盘。而对访问8.4GB以上的硬盘，必须使用扩展INT13H，在后面具体实现的章节中，将具体给出介绍及应用。

由于LARGE、LBA寻址模式采用了逻辑变换算法，比CHS复杂。但到目前为止大多数的资料、磁盘工具类软件中，采用的硬盘参数介绍和计算方法却还是按照相对而言比较简单的CHS寻址模式，因此，CHS寻址模式是硬盘寻址模式的基础，理解CHS寻址模式，对目前而言的硬盘使用和维护，还是很有用的。

一、 **CHS模式**

CHS寻址模式将硬盘划分为磁头（Heads）、柱面（Cylinder）、扇区（Sector）。是最早的IDE方式。在此方式下对硬盘访问时，BIOS和IDE控制器对参数不作任何转换。该模式支持的最大柱面数为1024,最大磁头数为16，最大扇区数为63，每扇区字节数为512。因此支持最大硬盘容量为：512×63×16×1024=528MB。在此模式下即使硬盘的实际物理容量更大，但可访问的硬盘空间也只能是528MB。

知道了磁头数、柱面数、扇区数，就可以很容易地确定数据保存硬盘的哪个位置。也很容易确定硬盘的容量，其计算公式是：

硬盘容量＝磁头数×柱面数×扇区数×512字节。

二、 LARGE模式

LARGE大硬盘模式。当硬盘的柱面超过1024而又不为LBA支持时可采用此种模式。LARGE模式采取的方法是把柱面数除以2,把磁头数乘以2,其结果总容量不变。例如，在NORMAL模式下柱面数为1220,磁头数为16，进入LARGE模式则柱面数为610,磁头数为32。这样从DOS操作系统柱面数小于1024，即可正常工作。LARGE寻址模式把柱面数除以整数倍、磁头数乘以整数倍而得到的逻辑磁头／柱面／扇区参数进行寻址，所以表示的已不是硬盘中的物理位置，而是逻辑位置。

三、 LBA模式

LBA（Logical Block Addressing）逻辑块寻址模式。这种模式所管理的硬盘空间突破了528MB的瓶颈,可达8.4GB。在LBA模式下，设置的柱面/磁头/扇区等参数并不是实际硬盘的物理参数。LBA寻址模式是直接以扇区为单位进行物理扇区寻址的，不再用柱面／磁头／扇区三种单位来进行寻址。但为了保持与CHS模式的兼容，通过逻辑变换算法，可以转换为磁头／柱面／扇区三种参数来表示，但表示的也和LARGE寻址模式一样，已不是硬盘中的物理位置，而是逻辑位置了。在LBA模式下，可设置的最大磁头数为255，其余参数与普通模式相同。由此可计算出可访问的硬盘容量为:512×63×255×1024=8.4GB 。目前基本上只有LBA有实际意义了。[2]

第三章　文件系统及Windows操作系统下主要文件系统详述

3．1　文件系统及当前的主要系统类型概述

文件系统是操作系统用于明确磁盘或分区上的文件的方法和数据结构；即在磁盘上组织文件的方法。也指用于存储文件的磁盘或分区，或文件系统种类。一个分区或磁盘能作为文件系统使用前，需要初始化，并将记录数据结构写到磁盘上。这个过程就叫建立文件系统。

Windows 3.x和MS-DOS一直使用的是文件分配表（FAT）统；Windows95使用的是扩展FAT文件系统；Windows NT文件系统则在继续支持16位文件系统的同时，还支持两种32位的文件系统——Windows NT文件系统（NTFS）和高性能文件系统（HPFS）。这几种文件系统各有优缺点，适合于不同的应用目的。

一、　文件分配表(FAT)系统

FAT文件系统1982年开始应用于MS-DOS中。FAT文件系统主要的优点就是它可以由多种操作系统访问，如MS-DOS、Windows3.x、Windows 95、Windows 98、Windows NT和OS/2等。而且对于ARC兼容计算机来说，它的主分区必格式化为FAT分区，这个分区的大小只需要能存放引导机器的文件就可以了，而不会用于存放数据和其他应用程序文件。遗憾的是FAT文件系统不支持长文件名。人们给文件命名时受8个字符名3个字符扩展名8.3命名规则限制。同时FAT文件系统无法支持系统高级容错特性，不具有内部安全特性等。 关于FAT16和FAT32的详述，在下几节中将具体给出。

二、　扩展文件分配表(VFAT)系统

在Windows95中，通过对FAT文件系统的扩展，长文件名问题得到了妥善解决，这也就是所谓的扩展FAT（FAT）文件系统。在Windows95中，文件名可长达255个字符，所以人们很容易通过名字来表现文件内容。但是为了同MS-DOS和Win16位程序兼容，它仍保留有扩展名。它同时也支持文件日期和时间属性，为每个文件保留了文件创建日期/时间、文件最近被修改的日期/时间和文件最近被打开的日期/时间这三个日期/时间戳。

三、 WindowsNT文件系统NTFS

支持WindowsNT的所有优点。这些优点中最重要的是WindowsNT的安全性。与NTFS文件系统相结合，能够指定谁能访问某一文件或目录和对它作什么操作。在创建一个文件时，可以通知WindowsNT，哪些用户可以读该文件，哪些用户可以修改该文件；另外，还可以指定谁可以列出一个目录的内容和谁可以在该目录下增加文件。即使用户知道文件的路径，仍可以禁止访问目录中的文件，只有NTFS分区中的文件才有这种称为任意访问控制的能力。

NTFS的第二个优点是它具有先进的容错能力。NTFS使用一种称为事务（transaction）登录的技术跟踪对磁盘的修改，因此，NTFS可以在几秒钟内恢复错误而不是HPFS的几分钟或几小时（取决于HPFS分区的大小）。

NTFS的第三个优点是其文件不易受到病毒和系统崩溃的侵袭，这种抗干扰直接源于WindowsNT操作系统的高度安全性能。即使在FAT和NTFS两种文件系统在一个磁盘中并存时，由于NTFS文件系统一般只能被WindowsNT识别，普通的病毒还是很难在NTFS文件系统中找到生存空间。对于大分区，NTFS比FAT和HPFS效率都高，FAT和HPFS比NTFS需要更多的空间来存储文件系统用于管理硬盘上文件和目录的信息。此外，由于NTFS文件系统支持长文件名，人们给文件命名时现也不需受8.3命名规则限制，从而可以给文件起一个反映其意义的文件名。NTFS支持向下兼容，甚至可以从新的长文件名中产生老式的短文件名。当文件写入可移动媒体（如软盘）时，它自动采用FAT文件命名规则。

实际上NTFS的主要弱点是它一般只能被WindowsNT所识别，在日前有些Linux的操作系统中，有人根据NTFS文件系统也给出了相应的驱动程序，使得某些Linux系统也能读写NTFS文件系统上的文件数据。NTFS文件系统可以存取FAT文件系统和HPFS文件系统的文件，但其文件却不能被FAT文件系统和HPFS文件系统所存取，兼容性不是特别好。但从网络安全性的角度来说，这种限制也是一种优点，它可以保证其他操作系统没有Windows的安全控制，其用户就不能对NTFS分区中的文件进行访问。

四、 高性能文件系统OS/2的高性能文件系统(HPFS)

主要克服了FAT文件系统不适合于高档操作系统这一缺点，HPFS支持长文件名，比FAT文件系统有更强的纠错能力。WindowsNT也支持HPFS，使得从OS/2到WindowsNT的过渡更为容易。HPFS和NTFS有包括长文件名在内的许多相同特性，但使用可靠性较差，也较低级。

上述的4种文件系统都被Windows所支持。总的来说，用户可以从文件系统的能力、文件系统的安全机制、使用的方便性以及相应的硬件环境等几个方面来综合考虑需安装的文件系统，单独选择其中一种文件系统或者它们中几种文件系统的组合，以使系统工作于最佳状态。本论文除了根据微软关于FAT文件系统的白皮书，详细阐述了关于FAT文件系统备份恢复的制作过程，而且还对由于商业原因，微软并未提供的NTFS文件系统内部结构做出了详细说明，并相应的给出了磁盘备份恢复的解决方案。

3．2 FAT16文件系统

FAT16文件系统使用了2个字节（16位比特）的大小的空间来表示每个簇配置文件的情形，故称之为FAT16。

FAT表由于受到先天的限制，因此每超过一定容量的分区之后，它所使用的簇（Cluster）大小就必须扩增，以适应更大的磁盘空间。所谓簇就是磁盘空间的配置单位，就象图书馆内一格一格的书架一样。每个要存到磁盘的文件都必须配置足够数量的簇，才能存放到磁盘中。FAT16各分区与簇大小的关系如下表：

表3-1 FAT16各分区与簇大小的关系 [3]

分区大小	FAT16簇大小
16MB-127MB	2KB
128MB-255MB	4KB
256MB-511MB	8KB
512MB-1023MB	16KB
1024MB-2047MB	32KB

如果在一个1000MB的分区中存放50KB的文件，由于该分区簇的大小为16KB，因此它要用到4个簇才行。而如果是一个1KB的文件，它也必须使用一个簇来存放。所以在使用磁盘时，无形中都会或多或少损失一些磁盘空间。[3]

由上可知，FAT16文件系统有两个最大的缺点：

1）磁盘分区最大只能到4GB。FAT16文件系统已不能适应当前这种大容量的硬盘，必须被迫分区成几个磁盘空间。

2)使用簇的大小不恰当。试想，如果一个只有1KB大小的文件放置在一个1000MB的磁盘分区中，根据表3-1显示，此时一个簇拥有了32个扇区合16K，所以它所占的空间并不是1KB，而是16KB，足足浪费了15KB！所以对于许多的小文件，譬如说：HTML文件，其大小几乎多为1KB、2KB，而制作一个网站往往用到数十个HTML文件。如果硬盘中有100个这种小文件的话，那浪费的磁盘空间可从700KB（511MB的分区），

到3.1MB(2047MB的分区)。 在这种情况下，把这些小文件压缩打包到一个文件，常常就会使一个占用几兆大小的小文件组，变成只占有几十K大小的压缩包，因为它整合了这些小文件，排除了一些无用的数据字节。

以上这两个问题会使得很多系统管理员在“分多大的分区，才能节省空间，同时又可使硬盘的使用更加方便有效”的抉择中犹豫不决。从理论上来说，在同一个FAT文件系统中，分区越大，那么它所对应的簇也会越大，从而由此可能产生的空间浪费也会越多。

3．3 FAT32文件系统

为了解决FAT16存在的问题，开发出FAT32系统。FAT32使用了4个字节（32位比特）的空间来表示每个簇配置文件的情形。利用FAT32所能使用的单个分区，最大可达到2TB（2048GB），而且各种大小的分区所能用到的簇的大小，也是恰如其分，上述两大优点，造就了硬盘使用上更有效率。现将分区与簇的大小汇整如下，可以仔细做个比较：

表3-2 FAT32、FAT16各分区与簇大小的关系 [3]

分区大小	FAT16簇大小	FAT32簇大小
16MB-32MB	2KB	不支持
32MB-127MB	2KB	512bytes
128MB-255MB	4KB	512bytes
255MB-259MB	4KB	512bytes
260MB-511MB	8KB	4KB
512MB-1023MB	16KB	4KB
1024MB-2047MB	32KB	4KB
2048MB-8GB	不支持	4KB
8GB-16GB	不支持	8KB
16GB-32GB	不支持	16KB
32GB以上	不支持	32KB

以当前硬件所能支持的情况下，如将8GB硬盘划分为单个分区的话，使用的簇的大小也只有4KB，比起以往的FAT16来说，真是节省了许多空间。

看到这些优点，也必须指明FAT32不足之处。大部分FAT32的缺点大部分都还只是局限于老版本的硬件或是软件环境。所以现在随着软硬件的不断升级，FAT32被广泛的应用，可以基本不用在意FAT32的局限性。

3．4 NTFS文件系统

NTFS使用了64位的簇索引，从而能够使NTFS文件系统能寻址到一个高达160亿GB的分区；然而Windows 2000把NTFS分区限制为只能寻址32位的簇，也就是128万亿字节（使用了64-KB的簇）。下表显示了NTFS分区的默认簇大小。（但当格式化NTFS分区的时候，可以更改默认簇的大小）

表3-3 NTFS分区的默认簇大小 [4]

分区大小	默认簇大小
512MB or less	512 bytes
513MB-1024MB (1GB)	1 KB
1025MB-2048MB (2GB)	2 KB
大于 2048MB	4 KB

NTFS包含了很多高级的特征，譬如说文件目录级安全，磁盘配额，文件压缩，基于目录的符号链接，和加密。其中最主要的就是可恢复能力。如果一个系统意外当机，FAT分区的元数据就只能是处于一个不连续的状态。 NTFS日志以一种事务的方式改变了分区元数据，以至于文件系统结构能被修复到一个连续的状态，避免了文件或者目录结构信息的丢失（文件数据可能会丢失）。

NTFS具有众多优点。主要就缘由其在此磁盘结构上独特的实现方法。可是微软出于商业原因，并没有公布NTFS文件系统的具体结构。本文将通过搜集的资料，和实际得到的NTFS扇区数据，逐步揭开NTFS文件系统的神秘面纱。在下一章中将具体介绍NTFS如何组织文件和目录，如何存储文件属性与数据的。

第四章　磁盘数据结构详述

4．1　分区表分析

所谓硬盘分区，实际上就是将硬盘的整体存储空间划分成相互独立的多个区域。从应用的角度来看，硬盘必须分区后才能使用，如果不对硬盘分区，则操作系统将不能识别硬盘。在本章中，将详细讨论分区表，并做了相当多的分区试验，掌握了丰富的实际分区数据，为实际项目中，用程序改写分区表做了充分准备。先给出一些准备知识。

一、分区与逻辑磁盘的关系

每个逻辑磁盘实际上是一个独立的分区，也就是说一个独立的逻辑磁盘实际上与一个独立的分区相对应。

二、硬盘分区方式

硬盘分区后一般会存在三种类型的分区，即主分区、扩展分区和非DOS分区。其中主分区又称为主DOS分区（Primary DOS Partition），扩展分区又称为扩展DOS分区（Extended DOS Partition）。非DOS分区（Non-DOS Partition）是一种特殊的分区形式，它是将硬盘中的一块区域单独划分出来供另一个操作系统使用，如Windows NT、Linux和Unix等。对主分区的操作系统来讲，非DOS分区是一块被划分出去的存储空间。只有非DOS分区内的操作系统才能管理和使用这块存储区域，非DOS分区之外的系统一般不能对该分区内的数据进行访问。即象在DOS操作系统、Windows操作系统不能对Linux下的ext2、ext3文件系统分区进行访问一样。[5]

主分区是一个比较单纯的分区，通常位于硬盘前面的区域中，构成逻辑C磁盘。在主分区中，不允许再建立其他逻辑磁盘。

扩展分区的概念则比较复杂，也是造成分区和逻辑磁盘混淆的主要原因。由于微机操作系统在MBR（主引导记录）仅仅为分区表保留了64个字节的存储空间，而每个分区的参数占据16个字节，故主引导扇区中总计可以存储4个分区的数据。由于操作系统只允许存储4个分区的数据，所以如果全部都是主分区的话，则系统最多只允许存放4个主分区。对于具体的应用，4个分区往往并不能满足实际需求，而且在Windows 9x操作系统下，如果系统存在一个以上的主分区的话，也会产生异常。所以

为了在保证系统稳定的情况下，建立更多的逻辑磁盘供操作系统使用，系统引入了扩展分区的概念。

所谓扩展分区，严格地讲它并不是一个实际意义的分区，它仅仅是一个指向下一个分区的指针，这种指针结构将形成一个单向链表。这样在主引导扇区中除了主分区外，仅需要存储一个被称为扩展分区的分区数据，通过这个扩展分区的数据可以找到下一个分区（实际上也就是下一个逻辑磁盘）的起始位置，以此起始位置类推可以找到所有的分区。无论系统中建立多少个逻辑磁盘，在主引导扇区中通过一个扩展分区的参数就可以逐个找到每一个逻辑磁盘。

需要特别注意的是，其一：由于主分区之后的各个分区是通过一种单向链表的结构来实现链接的，因此，若单向链表发生问题，将导致逻辑磁盘的丢失；其二：由于分区表仅仅只有64个字节的存储空间，所以在一块硬盘中最多只能划分三个主分区加一个扩展分区或者是四个主分区,而且在windows 9x版本中，更是只能有一个主分区存在。

三、硬盘分区结构

为了方便操作系统在启动过程中访问硬盘的分区参数，分区的相关数据存储在一个被称为主引导扇区（MBR）的特殊存储空间中，也就是存储在硬盘的0磁头0柱面1扇区。下面简单介绍一下MBR（主引导扇区）

1．硬盘主引导扇区

每个硬盘主引导记录等于硬盘主引导记录（MBR）加上硬盘分区表（DPT）。

它的物理位置位于磁盘的0面0道1扇区（cylinder 0, side 0, sector 1），大小等于512字节。其中，MBR占用了446字节（从偏移0x0000到0x01BD），DPT（Data Partition Table）占用了64字节（从偏移0x01BE到0x01FD），结束标志符为2字节的（0x55、0xAA）

硬盘主引导扇区的功能主要是，MBR通过检查DPT分区信息引导系统跳转至相关可引导活动分区的DBR，再通过此分区的DBR信息，调用引导程序（譬如说：Windows 2000下的NTLDR），从而启动操作系统。

表4-1　MBR详细数据及偏移介绍 [6]

000H--08AH	MBR启动程序（寻找开机分区）

08BH--0D9H	MBR启动字符串
0DAH--1BCH	保留（"0"）
1BEH--1FDH	硬盘分区表
1FEH--1FFH	结束标志（55AA）

主引导扇区为分区数据保留了64个字节的存储空间，每个分区的数据占用16个字节的存储空间。硬盘分区数据由于受主引导扇区中只能存储四个分区数据的影响，因此采用的是一种混合的分区数据保存结构，这种混合的分区数据结构由以下三个部分组成：

1. 主分区的分区数据存储在主引导扇区中，并且通常是分区表数据中的第一项。考虑到应用的广泛性，如果此块磁盘中仅有一个主分区，则此主分区通常也是活动分区。

2. 根据用户需要，可以建立一个扩展分区。扩展分区的相关数据也存储在主引导扇区中。扩展分区实际上是一个单链指针，它指向系统下一个分区，也就是下一个逻辑磁盘的位置。

3. 根据是否已经建立了扩展分区，系统允许建立2～3个非DOS分区，用于建立其他操作系统的存储和管理区域。

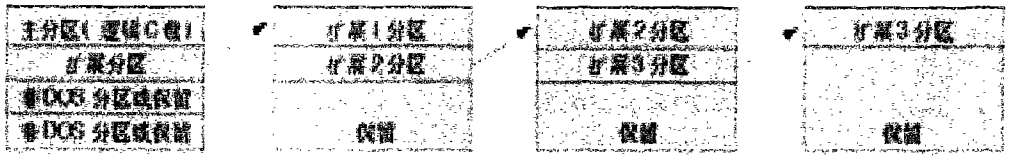
4. 主分区和扩展分区可以混合排放。在多主分区和扩展分区并存的情况下，顺序上并不一定是先主分区再扩展分区，或者是先扩展分区再主分区，可以交叉划分分配磁盘的空间。

5. 在扩展分区被划分后，可以暂时不划分逻辑磁盘。在扩展分区这个单链指针，指向的系统下一个分区中，也就是下一个逻辑磁盘的位置处的分区表处位置全部置零。具体可以见后面的解释说明。

根据分区的结构，我们知道在主引导扇区的分区表中存储上述三种类型分区形式的参数，这些参数包括分区的起始磁头、柱面、扇区和分区结束的磁头、柱面、扇区参数，以及分区的类型和是否为活动分区等对分区至关重要的数据。对于主分区和非DOS分区来讲，这些参数就是分区实际的参数。而对于扩展分区来讲，其扩展分区的起始物理地址实际上是另一个逻辑分区表数据的存储物理地址。根据扩展分区中所建立的逻辑磁盘的数量，将相应产生相同数量的逻辑分区数据。例如，假设一个物理磁

盘的扩展分区中建立了3个逻辑磁盘，则在主引导扇区中存储的分区表中可以找到第一个逻辑磁盘的起始分区地址；而在第一个逻辑磁盘的起始物理地址中将存储两项分区数据，一项是当前逻辑磁盘的分区数据，另一项是指向下一个逻辑磁盘的起始分区地址。也就是说，每一个逻辑磁盘实际上均有一个分区引导扇区，该分区引导扇区存储当前分区和下一个分区的引导扇区的物理地址。若分区引导扇区中仅存有当前分区的物理地址，没有下一个分区的物理地址，则表示当前分区是最后一个逻辑磁盘。所以，习惯称这种逐项给出下一个分区地址的指针链表为一个分区结构的链表。

图4-1：包括有4个逻辑磁盘的分区数据存储结构



根据图4-1所阐述的分区与逻辑磁盘结构关系，不难得出这样的结论：逻辑磁盘实际上就是一个独立的分区，只不过分区的参数不是存储在主引导扇区中，而是存储在各个逻辑分区所在柱面的0磁道1扇区（位于隐藏扇区区域）中。通过存储在主引导扇区中的扩展分区参数，按单向链表的方式可以逐个访问到逻辑磁盘的分区参数。

四、用实际的分区数据来理解磁盘参数的存储结构

1．活动分区主引导记录（DBR）

它的物理位置位于此磁盘活动分区的逻辑第1扇区，也就是位于此分区占用的当前柱面、第1磁道、第1扇区（cylinder 0, side 1, sector 1）。如果是FAT16文件系统，DBR大小等于512字节；但如果是FAT32的话，DBR大小便达到了3个扇区（1536个字节）。DBR包9了机器CMOS等信息（0000--0059），核对该信息并引导指定的系统文件，如NTLDR等；

表4-2　DBR详细数据及偏移介绍

000H--002H	3 BYTE的跳转指令(去启动程序，跳到03EH)
003H--03DH	BIOS参数区

03EH--19DH	DOS启动程序
19EH--1E5H	开机字符串
1E6H--1FDH	文件名(IO.SYS, MSDOS.SYS)
1FEH--1FFH	结束标志(55AA)

2．硬盘分区表（DPT）

表4-3 DPT详细数据及偏移介绍

偏移地址	字节数	含义分析。
0x01BE	1	分区类型：00表示非活动分区：80表示活动分区；其他为无效分区。
0x01BF 至 0x01C1	3	分区的起始地址（面/扇区/磁道），通常第一分区的起始地址开始于1面0道1扇区，因此这三个字节应为010100（a）。
0x01C2	1	分区的操作系统的类型；（b）。
0x01C3 至 0x01C5	3	分区的结束地址（面/扇区/磁道）。
0x01C6 至 0x01C9	4	该分区起始逻辑扇区。
0x01CA 至 0x01CD	4	该分区占用的总扇区数。

注a：注意分区的起始地址（面/扇区/磁道）和结束地址（面/扇/道）中字节分配：

00000000 01000001 00010101

~~~~~~~~ ==^^^^^^ ========

~ 面(磁头) 8 位

^ 扇区 6 位

= 磁道 10 位

注b：分区的操作系统类型(文件格式标志码)

1---DOS FAT12

4---DOS FAT16<32M

5---EXTEND

6---DOS FAT16>32M
~~~~~~~~

7---NTFS(OS/2)

11---DOS FAT32 UP TO 2047GB

12---类似于11，但是是用LBA的int13扩展模式访问磁盘的。

14---类似于6，但是是用LBA的int13扩展模式访问磁盘的。

15---类似于5，但是是用LBA的int13扩展模式访问磁盘的。

83---LINUX>64M

在此次实用软件中，因为编程的便利性，在原有的分区的操作系统类型基础上，又重新添加了几个自定义的分区类型。如下所示：

表4-4 程序中宏定义、相关值和含义

宏定义	字节值	相关含义
#define PART_UNKNOWN	0x00	未知文件类型
#define PART_DOS2_FAT	0x01	FAT12
#define PART_DOS3_FAT	0x04	FAT16。分区小于32MB
#define PART_EXTENDED	0x05	扩展 MS-DOS 分区
#define PART_DOS4_FAT	0x06	FAT16。分区大等于32MB
#define PART_DOS5_NTFS	0x07	NTFS
#define PART_DOS32	0x0B	FAT32. 分区最大达到2047GB
#define PART_DOS32X	0x0C	和FAT32(0Bh)一样，但使用了逻辑块的中断13扩展
#define PART_DOSX13	0x0E	和FAT16(06h)一样，但使用了逻辑块的中断13扩展extensions
#define PART_DOSX13X	0x0F	和扩展MS-DOS 分区(05h)一样，但使用了逻辑块的中断13扩展extensions
#define PART_DOS_UNUSE	0xCC	表示此分区未被使用
#define PART_DOS_UNFDISK	0xDD	表示此磁盘空间并未被fdisk程序分区过

#define PART_DOS_NOTFORMAT	0xEE	表示此磁盘空间已经被fdisk程序分区过，但尚未被格式化
#define PART_DOS_TOTUNUSES	0xFF	表示所有未被使用的磁盘空间

DPT 总共64字节（从0x01BE至0x01FD），如上所示每个分区占16个字节，所以可以表示四个分区，这也就是为什么一个磁盘的主分区和扩展分区之和总共只能有四个的原因。

扩展分区的信息位于以上所示的硬盘分区表（DPT）中，而逻辑驱动器的信息则位于扩展分区的起始扇区，即该分区的起始地址（面/扇区/磁道）所对应的扇区，该扇区中的信息与硬盘主引导扇区的区别是不包含MBR，而16字节的分区信息则表示的是逻辑驱动器的起始和结束地址等。

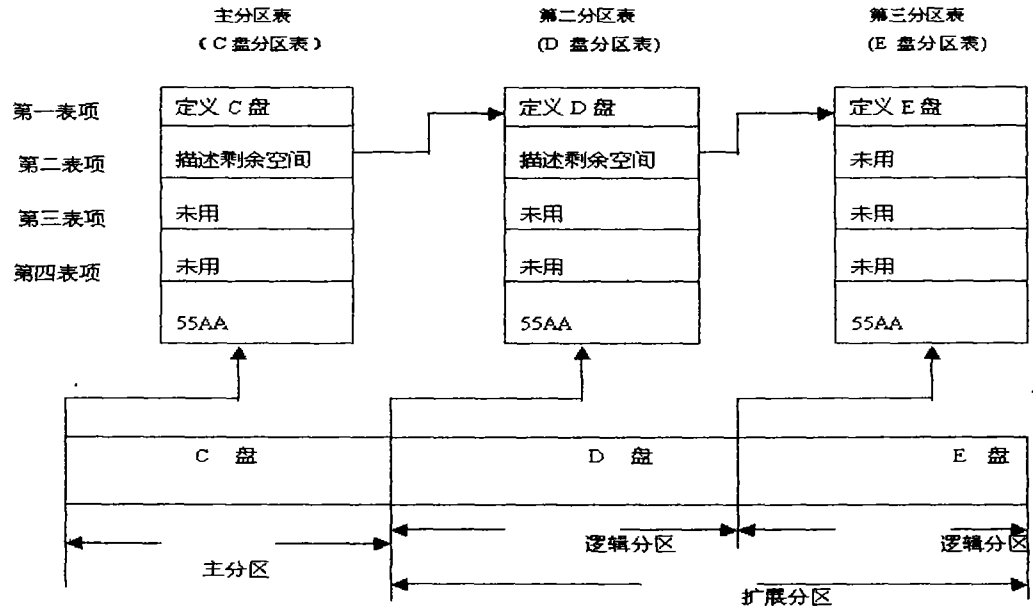
表4-5　一个硬盘的分区情况：

类型	道	面	扇	道	面	扇	起始扇区	结束扇区	总共扇区
主分区C	0	1	1	276	239	63	63	4,188,239	4,188,177
扩展区	277	0	1	554	239	63	4,188,240	8,391,599	4,203,360
逻辑区D	277	1	1	554	239	63	4,188,303	8,391,599	4,203,297

如果主分区表损坏，则可以通过手工查找扩展分区表中所包含的逻辑驱动器数据，在本例中就是D盘所对应的起始扇区数据4,188,303，然后将其起始扇(逻辑)减去63就是所对应的扩展分区的起始扇(逻辑)，将其起始地址（面/扇区/磁道）改为0面就是扩展分区的起始地址4,188,240。然后通过扩展分区就可以得到主分区C的信息，然后就可以使用FDISK/MBR命令和手工填写分区表恢复整个硬盘。

下图中给出了一个完整的主分区，扩展分区，逻辑分区的链表示意图。从图中可以很清楚地明白分区表的组织结构。

图4-2　分区表链示意图 [7]



下面再给出一组完整的实际分区表数据对上述分区数据结构进行具体说明。

图4-3 分区表数据

磁盘 0 磁头 0 柱面 1 扇区的分区数据

```
03B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 80 01   ................
03C0  01 00 06 FE 3F F3 3F 00-00 00 B5 CF 3B 00 00 00   ....?.?.....;...
03D0  01 F4 05 FE FF 0F F4 CF-3B 00 1C 5F 84 00 00 00   ........;.._....
03E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

磁盘 0 磁头 F4H(244)柱面 1 扇区的分区数据

```
03B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 01   ................
03C0  01 F4 06 FE 7F AA 3F 00-00 00 B8 DB 2C 00 00 00   ......?.....,...
03D0  41 AB 05 FE BF 61 F7 DB-2C 00 F7 DB 2C 00 00 00   A....a..,...,...
03E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

磁盘 0 磁头 1ABH(427)柱面 1 扇区的分区数据

```
03B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 01   ................
03C0  41 AB 06 FE BF 61 3F 00-00 00 B8 DB 2C 00 00 00   A....a?.....,...
03D0  81 62 05 FE FF 0F EE B7-59 00 2E A7 2A 00 00 00   .b......Y...*...
03E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

磁盘 0 磁头 762H(610)柱面 1 扇区的分区数据

```
03B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 01   ................
03C0  81 62 06 FE FF 0F 3F 00-00 00 EF A6 2A 00 00 00   .b....?.....*...
03D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
03E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

根据上图提供的数据，观察第一组数据，磁盘0磁头0柱面1扇区的分区数据，可以发现主引导扇区中存储的分区表数据表示物理硬盘分成两个区，其中第1个分区为主分区（分区类型标志为06H），且为活动分区。而第2个分区为扩展分区（分区类型标志为05H），下一个分区表数据的存储地址为0磁头F4H柱面（即244柱面）1扇区。接着读入0磁头F4H柱面1扇区的数据，观察第二组数据，磁盘0磁头F4H柱面1扇区的分区数据，发现其分区表的数据结构与主引导扇区完全相同，但没有分区引导程序部分，除了分区表数据外，其余数据均为0。第2个分区引导扇区中存储的分区表数据表示该分区从1磁头F4H柱面1扇区开始，这是该分区的逻辑0扇区的位置，而分区标志（06H）表示对应的分区为主分区，下一个分区从0磁头1ABH（即427）柱面（柱面参数共10位二进制，其中第9和10位由前一个字节的最高两位提供）1扇区开始，为扩展分区。依次类推，可以通过第1个扩展分区的数据依顺序找到每一个分区或逻辑磁盘的数据。在0磁头262H（即610）柱面1扇区中存储的分区表数据表示该分区是最后一个分区。

根据图4-3所提供的硬盘实际存储的分区数据，不难得出这样的结论：该硬盘共有两个分区，其中主分区构成逻辑C磁盘，扩展分区构成D、E和F 3个逻辑磁盘。

至此，已经全面了解了硬盘分区及逻辑磁盘的构成原理，同时掌握了分区的数据存储结构。

五、总结

由于受硬盘主引导扇区为分区数据所保留的存储空间的限制，系统不得不使用扩展分区的链表结构来构成整个硬盘系统的分区结构，所以说逻辑盘和分区的概念是完全相同的。另外，扩展分区的链表结构由于是单向链表，只能从主引导扇区向后逐个查找到各个分区，不能逆向从最后一个分区查找前面的分区。因此，若因某种原因造成链表中断，则可能导致后面的分区或逻辑磁盘丢失。当然，存储在逻辑盘中的数据也就自然丢失了。

通过前面分析了分区的数据存储结构后，得出了一个规律，这个规律就是分区数据总是存储在某个柱面的0磁头1扇区的物理位置上，而每个逻辑盘的逻辑0扇区则为同一柱面1磁头的1扇区，分区开始位置与逻辑0扇区之间的所有扇区为隐藏扇区，一般间隔大小为62个扇区。所以，如果因为某种原因导致分区数据的丢失，通过相应查找每个柱面0磁头1扇区的数据，不难用手工方式恢复分区表数据。

4．2 研究FDISK程序修改分区表

在上一节中，论述了磁盘分区表的概念、结构、作用和特点。本节中将结合实际程序中的功能需求，研究fdisk程序，对磁盘上的分区进行动态划分。

如上节所说的，磁盘内的分区通过单向链表进行连接。分区类型可分为主分区和扩展分区。而且扩展分区中又因为需要突破分区数量的限制，可以继续被划分多个逻辑分区。Windows 98中，最多只能存在一个主分区和一个扩展分区；然而在Windows 2000中，就可以存在多达三个主分区和一个扩展分区（也可以是四个主分区）。在正式使用磁盘空间，安装系统程序前，磁盘一般都只是被低级格式化，划分了基本的柱面、磁道和扇区，确定了基本的数据的存储和寻址地址。然后，就需要使用分区划分程序，对磁盘进行分区操作。Fdisk程序就是这样一个被Dos、Windows和Unix等等操作系统广泛使用的分区程序，本节会根据磁盘可能存在的几种状态，研究fdisk程序，对磁盘进行动态划分。

4．2．1 实验说明

在以下实验过程中，所有的改写分区操作，都是在Windows 98和2000下分别调用不同的读写绝对扇区的函数或是方法实现的。表格中使用到的缩写字符所代表的含义如下表：

表4-6 DPT表中使用到的字段名

字段名	注释说明
BI	此分区是否是活动分区。根据DPT对应位置的值0x80和0x00来区别该分区是否可以引导
C	此分区的起始柱面数
H	此分区的起始磁头数
S	此分区的起始扇区数
SI	此分区的分区类型

EC	此分区的结束柱面数
EH	此分区的结束磁头数
ES	此分区的结束扇区数
NOS	此分区的总扇区数。（单位：扇区数）
RS	此分区的偏移数。（单位：扇区数）

情况一：

磁盘初始状态：只被低级格式化，但并没有被划分过分区。

目的状态：生成一个800兆字节大小的主分区C和一个2298兆字节大小的扩展分区。在扩展分区中又包含一个902兆字节大小的逻辑分区D，其后都是空余分区空间。

初始状态：

在磁盘0柱面0磁道1扇区的分区表（偏移0x01BE处）皆为0x00。

在运行了fdisk程序后，在磁盘0柱面0磁道1扇区的分区表（偏移0x01BE处）得到的数据如下表：

表4-7　主分区分区表数据

第一分区表信息：	
BI	0x80
C	0
H	1
S	1
SI	6
EC	101
EH	254
ES	63
NOS	1638567
RS	63
第二分区表信息：	
BI	0x00
C	102
H	0
S	1
SI	5
EC	138

EH	254
ES	127
NOS	4707045
RS	1638630

第一分区数据表示的是第一主分区分区信息，根据缩写字符代表的含义，可以获得以下信息。BI等于0x80，表示该分区是可以引导的，C/H/S等于0/1/1表示该主分区，始于0柱面1磁头1扇区；SI等于6，表示该分区是FAT16文件类型；EC/EH/ES等于101/254/63表示该主分区，结束于101柱面254磁头63扇区；NOS等于1638567，表示该主分区的空间大小（不包含隐藏扇区）即为1638567个扇区数。RS等于63，表示该主分区的逻辑分区空间起始于偏移位置为63个扇区的地方。

从表面上看，上面的分析都是正确的，其实，从某种意义来说，确实也是正确的。但实际上，这样的分析只在该磁盘寻址模式为CHS时才成立。因为EC/EH/ES（结束磁盘柱面/磁头/扇区数）都是只占用一个字节，所以，在CHS模式下，该磁盘能寻址到（255+1）*（255-1）* 255 = 16516096个扇区 = 7.87G。当磁盘大于7.87G的时候，CHS模式将无能为力，从而C/H/S和EC/EH/ES也不能正确表示它们在CHS模式中被设定的含义，于是在LARGE和LBA寻址模式中这些字段被赋予了新的含义。实际上，在LARGE和LBA模式下，系统无法根据传统的柱面、磁头和扇区这样的3D寻址方式，定位需要读写的扇区的位置。C/H/S和EC/EH/ES两组值只是存在理论上仅有的一点意义，并没有实际可用价值。只能通过NOS和RS这两个字段定位扇区位置。

在分析这些被重新定义的字段域前面，将给出一些准备知识，先讨论磁盘的分区的规则。在磁盘划分分区的时候，常常会发现，有些时候，磁盘分区的大小并不是恰好就是我们所指定的大小。会有几兆大小的出入。这是因为系统在给磁盘划分区间的时候，一般都以柱面为单位，一个柱面一个柱面的进行区间划分，譬如，当一个柱面包含了255个磁道、63个扇区的时候，此时，分区大小必将是 255 * 63 = 16065个扇区的单位。折合成以兆为单位，就是7.84兆。所以，系统在分区的时候，就会根据用户输入的分区大小，自动向上下最靠近的可分区大小靠拢。

在MBR（主引导扇区的分区记录中），RS这个字段表示的是绝对偏移扇区数。但是在其后将要介绍的扩展分区中的分区表纪录中，RS所表示的偏移扇区数就是相对偏移数，是相对于扩展分区的起始扇区数而言。

NS字段是相对于该分区的逻辑分区的大小。大小等于该分区实际占用的扇区数减去63个隐藏扇区数后的大小。在后面的例子中也将根据实例分析说明。

在给出了这些预备知识后，将讨论这些被重新定义的字段所表示的数值。当磁盘容量大于7.84G后，分区表中的柱面/磁头/扇区数在无法表示要读写的准确扇区位置时，将重新定义C/H/S和EC/EH/ES这六个字段域。

C在CHS寻址模式中表示的是分区的起始柱面数。在LBA和LARGE模式中，由于磁盘容量的原因，C就不再代表原来的表示含义，基本上可以认为它的数据已经不再具有任何的有效意义，只是为了保持与CHS寻址模式的一致性才被继续保留。它的取值规则是该分区的起始扇区绝对位置（单位：扇区数），除以每柱面可以容纳的扇区数后，所得的数值再对256求模的结果。象上例中，第一分区的起始扇区数为63，所以除以每柱面可容纳扇区数63*255=16065个扇区数后，得到的数值就是0(即使再对256求模，所得的数据依旧是0)。所以在第一个分区表，C的值即为0。在第二分区（即扩展分区）中，它的起始分区位置在（1638567 + 63 = ）1638630处，那么按照前面给出的公式，起始分区位置/每柱面扇区数 % 256。（1638630 / 16065） % 256 = 102。所以在第二分区处的C值即为102。

EC在LBA和LARGE寻址模式中的表示作用和C稍有差别，它表示的是该分区的结束柱面位置。EC的计算方式和C也很相像。只是公式中的起始分区位置改为了结束分区位置，然后再在得到的数据后面减去一个柱面，即结束分区位置/每柱面扇区数 % 256 - 1。在上例中，在第一个表中，第一主分区结束位置在1638630扇区，那么依照公式，1638630 / 16065 % 256 – 1 = 101；在第二个分区（即是扩展分区）处，它的分区结束位置为（4707045 + 1638630） % 256 – 1 = 395 % 256 – 1 = 138。

H表示的是起始磁头数，相对于前两个数据域，它的取值就相对简单，当分区表的分区类型为0x05或是0x0F时，它的数值为0；当分区类型为其它数值时，它的数值即为1。通过上例，可以很清楚看见这个规律。

EH表示的是结束磁头数，它的取值也相对简单。就等于该寻址模式下，每柱面所拥有的磁头数减一。譬如上表显示的磁盘，在LBA模式下，它的每柱面拥有255个磁头，那么，在分区表的EH数据域的值皆为254。

S表示的是起始扇区数，在LBA和LARGE模式下，它的取值就相对复杂一点。它的数据公式为：（起始分区位置 / 每柱面扇区数 / 256 * （每柱面磁头数 + 1））+

1。参照上例，在第一个分区表中，S值等于（0 / 16065 / 256 * （63 + 1））+ 1 = 1；在第二个分区表中，S值等于（1638630 / 16065 / 256 * （63 + 1））+ 1 = 1。[注]

ES表示的是结束扇区数，它的取值公式类似于S：（结束分区位置 / 每柱面扇区数 / 256 * （每柱面磁头数 + 1））+ 每柱面磁头数。参照上例，在第一个主分区表中，ES值就等于（1638630 / 16065 / 256 * （63 + 1））+ 63 = 63；在第二个分区表中，ES值就等于（（1638630+4707045）/ 16065 / 256 * （63 + 1））+ 63 = 127。[注]

在本例中，扩展分区并没有被完全填满，在整个4707045扇区（合2298兆）的扩展分区空间中，只有一个占用902兆大小的空间。在这种情况下，扩展分区偏移第1638630个扇区将会存在扩展分区表，指明第一个902兆的逻辑分区。具体数据见下表。

表4-8　第一逻辑分区分区表数据

第一分区表信息：	
BI	0x00
C	102
H	1
S	1
SI	6
EC	216
EH	254
ES	63
NOS	1847412
RS	63
第二分区表信息：	
BI	0
C	0
H	0
S	0
SI	0
EC	0
EH	0
ES	0
NOS	0
RS	0

[注]：此处“/”号，代表了整除。

可以看到在第一分区表中，各数据域的字段都严格按照本节中先前说明的含义进行赋值。在这里需要特别指出说明的是，第一分区表中的RS字段域的数值，是相对扩展分区起始扇区位置而言。此外，由于逻辑分区并没有完全占满扩展分区，所以在此表中， NOS加RS的总和小于扩展分区的总空间4707045个扇区大小。后面的分区自动空余保留，可以作为以后对扩展分区做进行进一步逻辑分区的空间。

更进一步，如果在此例的扩展分区中，存在多个逻辑分区，则只需要在第二个分区表的逻辑分区引导记录中，填写下一个分区的引导信息，包括偏移量（相对于扩展分区的起始扇区偏移值而言）和该分区总的扇区数量（一般情况下，包括62个隐藏扇区）。如果扩展分区中，两个逻辑分区在物理上不连续，也就是说，它们之间可能会有多个柱面相隔（由于分区分配空间都是每柱面扇区数的倍数）。那么，只要把前一引导扇区的第二个分区表的RS（偏移值）设置为下一个不连续分区的相对起始位置就可以继续把分区链接起来了。形成了一个连续的单链分区引导信息。如果在扩展分区中存在某个分区未被格式化或者是原先被分配后又被删除，也同样可以通过把前一引导扇区的第二个分区表的RS（偏移值）设置为下一个不连续分区的起始相对位置就可以继续把分区链接起来了。

4．2．2　更多的分区表范例

情况一

下例是来自一个活动USB硬盘，此硬盘分了一个主分区和一个扩展分区，扩展分区中包含有五个逻辑分区。

表4-9　分区表数据一

第一分区表:MBR,0柱面0磁道1扇区	
BI	0x80
C	0
H	1
S	1
SI	6
EC	101

第一逻辑分区表:102柱面0磁道1扇区	
BI	0x00
C	102
H	1
S	1
SI	6
EC	216

EH	254
ES	63
NOS	1638567
RS	63
BI	0x00
C	102
H	0
S	1
SI	5
EC	138
EH	254
ES	127
NOS	4707045
RS	1638630
主分区；800兆；FAT16	

EH	254
ES	63
NOS	1847412
RS	63
BI	0x00
C	217
H	0
S	1
SI	5
EC	11
EH	254
ES	127
NOS	819315
RS	1847475
第一逻辑分区；902兆；FAT16	

第二逻辑分区表:217柱面0磁道1扇区	
BI	0x00
C	217
H	1
S	1
SI	7
EC	11
EH	254
ES	127
NOS	819252
RS	63
BI	0x00
C	12
H	0
S	65
SI	5
EC	49
EH	254
ES	127
NOS	610470
RS	2666790
第二逻辑分区；298兆；NTFS	

第三逻辑分区表:304柱面0磁道1扇区	
BI	0x00
C	12
H	1
S	65
SI	7
EC	49
EH	254
ES	127
NOS	610407
RS	63
BI	0x00
C	50
H	0
S	65
SI	5
EC	94
EH	254
ES	127
NOS	722925
RS	3277260
第三逻辑分区；353兆；NTFS	

第四逻辑分区表:217柱面0磁道1扇区	
BI	0x00
C	50

第五逻辑分区表:304柱面0磁道1扇区	
BI	0x00
C	95

C	50
H	1
S	65
SI	7
EC	94
EH	254
ES	127
NOS	722862
RS	63
BI	0x00
C	95
H	0
S	65
SI	5
EC	138
EH	254
ES	127
NOS	706860
RS	4000185
第四逻辑分区；353兆；NTFS	

C	95
H	1
S	65
SI	6
EC	138
EH	254
ES	127
NOS	706797
RS	63
BI	0x00
C	0
H	0
S	0
SI	0
EC	0
EH	0
ES	0
NOS	0
RS	0
第五逻辑分区；345兆；FAT16	

情况二

下例是来自一个IDE硬盘，此硬盘分了二个主分区和一个扩展分区。这里将详细讨论MBR的可能情况。

表4-10 分区表数据二

第一分区表:MBR,0柱面0磁道1扇区	
BI	0x80
C	0
H	1
S	1
SI	7
EC	96
EH	254
ES	191
NOS	9783522
RS	63
BI	0x00
C	97
H	0

说明：

有两个主分区，一个扩展分区。第一主分区又是活动分区。扩展分区夹在两个主分区之间。在MBR的DPT（Data Partition Table）中，所有的RS都是磁盘的绝对偏移量。除了第一个分区表使用了62个隐藏扇区，从第一个逻辑分区开始存储实际分区数据，其他主分区都是直接从RS指向的绝对位置开始存储分

H	0
S	129
SI	15
EC	31
EH	254
ES	63
NOS	97659135
RS	9783585
BI	0x00
C	41
H	0
S	129
SI	15
EC	128
EH	254
ES	63
NOS	9622935
RS	107603370

区数据。NOS代表了实际分区数据大小。另外，还需要注意一点，第三分区，即第二主分区与扩展分区间隔10个柱面的区间（160650个扇区），当中有一段未使用空间，这种形式的分区在Windows 9x中可能会引起系统盘符分配混乱。扩展分区中的RS，是针对扩展分区的起始位置而言。而且在纪录扩展分区中的逻辑分区时，都会要求在每个分区前面预留62个隐藏分区。关于其中字段的具体含义请参见前面几节中所阐述的。扩展分区中的逻辑分区被省略。

情况三

下例是来自一个IDE硬盘，此硬盘具有一个扩展分区。这里将详细讨论扩展分区中逻辑分区的可能存在的一些特殊情况。

表4-11 分区表数据三

第一逻辑分区表:14柱面0磁道1扇区	
BI	0x00
C	14
H	0
S	65
SI	5
EC	26
EH	254
ES	127
NOS	208845
RS	144585
BI	0x00
C	0
H	0
S	0

说明：

表中列出的是某硬盘扩展分区中的第一、二逻辑分区表的数据。第一逻辑表中只有第一个16位表数据有数据，观察SI值为5，说明其后还有一个分区，由于此分区表只有一个分区数据，且SI值为5（其实15也表示后面有分区），表明此分区是曾经被划分后又被删除的，且观察第一逻辑分区RS值，可以发现一般在第一个16位表数据，其值一般

S	0
SI	0
EC	0
EH	0
ES	0
NOS	0
RS	0

第二逻辑分区表:14柱面0磁道1扇区	
BI	0x00
C	14
H	1
S	65
SI	7
EC	26
EH	254
ES	127
NOS	208782
RS	63
BI	0x00
C	27
H	0
S	65
SI	5
EC	77
EH	254
ES	127
NOS	819315
RS	353430

都为63，这里为144585，是指明了下面一个分区的相对起始位置，为相对于扩展分区的起始位置。绝对物理位置等于144585+扩展分区的绝对起始值。看第二逻辑分区表，它的第一个16位分区数据和第一逻辑分区表很相似，除了SI值改为了一个实实在在的分区类型值7（表示NTFS），RS改为了正常的63（表示此分区逻辑第一扇区在物理第一扇区后63扇区）。看第二逻辑分区的第二个16位分区数据，其中SI值又等于5，说明后面还有分区，可能是实际已被分区格式化的分区，也可能只是一个被分区未格式化的分区。想这种分区虽然曾经被划分，但未被格式化的情况，在分区表中，也可以通过其他一种形式来表示。见下例

情况四

下例是来自一个IDE硬盘，此硬盘具有一个扩展分区。这里将详细讨论扩展分区中逻辑分区的一些特殊可能存在的情况，详见下图数据。

表4-12 分区表数据四

第I逻辑分区表:51柱面0磁道1扇区	
BI	0x00
C	51
H	1
S	1

说明：

表中例出的是某硬盘扩展分区中的第I、I+1个逻辑分区表的数据。第一

S	1
SI	6
EC	67
EH	254
ES	63
NOS	273042
RS	63
BI	0x00
C	68
H	0
S	1
SI	5
EC	84
EH	254
ES	63
NOS	273105
RS	819315

第I+1逻辑分区表:68柱面0磁道1扇区	
BI	0x00
C	68
H	1
S	1
SI	6
EC	84
EH	254
ES	63
NOS	273042
RS	63
BI	0x00
C	102
H	0
S	1
SI	5
EC	118
EH	254
ES	63
NOS	273105
RS	1365525

逻辑表中正常，根据它第二分区表所指示的RS值819315，表示下一个分区起始值在相对于扩展分区起始扇区819315个扇区的位置。观察第I+1个逻辑分区表的分区表记录。重点注意第二分区表中的RS和C值和第一分区表中的EC。第一分区表中EC值为84，第二分区表中C值为102，按照原先收集的数据，可以发现如果两个分区在物理上是连续的，第一分区表的EC值和第二分区表的C值差1，现在两个值差18，说明这两个分区当中有18个柱面的空闲分区。同样的情况，也可以根据第二分区表中的RS值1365525作出判断，由第I逻辑分区表第二分区表RS值加以NOS值，总计1092420，和1365525差273105个扇区。显然这两万多个扇区即夹在第I,I+1逻辑分区间，空闲的被删除的分区。

4．2．3 引用FDISK原理

根据对多组实测磁盘分区数据研究，完全掌握了FDISK程序在对磁盘进行分区时，写分区表信息方法。应用在自己开发的磁盘恢复与备份程序中，使程序能够动态改写磁盘分区表，能够把一个原先已经被备份的分区映象文件导入到另一个与源分区大小、文件类型不一致的磁盘空间中，甚至可以导入一个刚被低格后的新硬盘中。这一切都只需要经过改写分区就可以了。但在修改分区表的时候，也会有点限制，譬如说在以下几种情况，就不能正常恢复分区数据：

1）当用户要求导入磁盘后面剩余的空间，如果剩余的空间不能满足映象文件恢复的要求时。

2）当用户要求导入磁盘后面剩余的空间，扩展分区位于最后一个主分区之前时。在Windows 2000下，如果已经拥有了三个主分区；在Windows 9x下，如果已经拥有了一个主分区时，就不可以在此磁盘后面的空余空间中，生成一个新的主分区，即使磁盘空间足够能够恢复。相反，如果扩展分区位于主分区之后，当剩余分区足够的时候，就可以把映象文件恢复到剩余空间，并把这一段空间纳入到扩展分区中，同时MBR中扩展分区大小NOS值和原最后一个逻辑分区的分区表，可以继续指向一个新生成的分区。

3）如果导入一个已存在的分区。在一般情况下，如果目的分区小于源分区，则不允许导入恢复，但在某些比较特殊的情况下，即使源分区和目的分区的大小不同，甚至源分区大于目的分区的情况下，只要源分区和目的分区的文件类型一致，每簇对应的扇区数一致，也可以不需要更改分区信息。这主要是因为在FAT表中，簇是一个文件存储单位，如果一个簇对应八个扇区的话，那么每个文件对象最少需要八个扇区的空间进行存储，即使这个文件只有一个字节的数据。那么在每簇对应的扇区数相等的情况下，只要匹配FAT表长度(即FAT表所占用的FAT表长度)，只要目的分区可提供的FAT表扇区长度足够长，那么，在同一文件系统中，当然可以容纳源分区的全部数据，即使源分区的实际分区大于目的分区。

对分区表的理解和对FDISK程序的研究应用。使磁盘备份恢复程序，不再仅仅停留在源、目的分区必须同文件类型，同大小的情况下了。向实用性更迈进了一步。

4．2．4 读写内存块大小参数的讨论

在实现磁盘分区恢复与备份程序的时候，除了需要考虑分区的划分，扇区的读写正确，还需要着重考虑磁盘读写速度。很难想象一个耗时很长的备份恢复操作会具有什么实际存在的价值。

备份和恢复操作，都需要对磁盘的扇区进行读写。那么为了尽快完成用户所要求的操作，就要求尽可能减少读写硬盘次数，使用最合适尺寸的内存块。很显然，如果每次只读写一个扇区，一共读取N次和每次读写X个扇区，一共需要读写N/X次扇区的速度是不一样的（X远大于1）。分配读取磁盘的内存块越大，读写磁盘的次数就越小，则速度也可以显著地被提高。可是，还需要考虑一个不可忽视的事实，那就是如果读写内存块越大，很可能在这一块内存块中无用扇区数据也会越多，可能还需要程序根据FAT表中的信息来判断此数据块对应的簇是否为有效簇，从而进行备份恢复操作。所以不能只是一味地追求降低读写磁盘次数，而盲目地把读写磁盘的内存块大小最大化。所以在编写本商用程序前，笔者针对读写内存块最合适的大小尺寸与相应的内存块可能的数据利用率之间的关系，进行了一系列测试。下表将给出详细的测试数据，来证明得出最优的读写内存块尺寸参数。

实验一：对一次读取数据块大小和读磁盘次数之间关系的讨论

表4-13 不同读写方式与不同数据利用率下，耗时数据比较

数据利用率	操作方式（所有读取，都是指从磁盘中读取数据）	耗时(秒)
16/16	每个回合中，1次读取64K,然后16次写入对应字符串位置	64
	每回合中，16次读取4K，然后合成在1个对应字符串位置	113
15/16	每个回合中，1次读取64K,然后15次写入对应字符串位置	64
	每回合中，15次读取4K，然后合成在1个对应字符串位置	106
14/16	每个回合中，1次读取64K,然后14次写入对应字符串位置	64
	每回合中，14次读取4K，然后合成在1个对应字符串位置	99
13/16	每个回合中，1次读取64K,然后13次写入对应字符串位置	65
	每回合中，13次读取4K，然后合成在1个对应字符串位置	91
12/16	每个回合中，1次读取64K,然后12次写入对应字符串位置	64
	每回合中，12次读取4K，然后合成在1个对应字符串位置	84
11/16	每个回合中，1次读取64K,然后11次写入对应字符串位置	64

	每回合中，11次读取4K，然后合成在1个对应字符串位置	78
10/16	每个回合中，1次读取64K,然后10次写入对应字符串位置	65
	每回合中，10次读取4K，然后合成在1个对应字符串位置	70
09/16	每个回合中，1次读取64K，然后9次写入对应字符串位置	64
	每个回合中，9次读取4K，然后合成在1个对应字符串位置	63

结论：当数据利用率接近于9/16，两种方法效率相似。所以，当读取内存块的可用数据利用率小于等于9/16的时候，就按照实际有多少数据就读写多少扇区数的方法为宜。而当数据利用率大于9/16的时候，则可以使用读写指定大小内存块数据的方法，然后从已经读出的内存数据块中提取有用合适的数据。

实验二：对一次读取数据块的讨论

表4-14 指定内存块大小情况下，每次都进行1000次读写操作的耗时情况

内存块大小	耗时	备注
256K	253秒	舍弃了读写字符串的操作，时间依然和原先的类似，所以字符串的操作时间可以忽略不计
128K	128秒	同上
64K	64秒	同上
32K	34秒	同上
4K	8秒	同上

结论：当数据一次性读取量大的时候，只要数据量相等，就应该采用一次读取尽量多个数据量。把每次读取的最大数据量扩展到256k的时候，发现数据读取速度效率和内存块大小为64k的时候基本相似，考虑到可能会有大量无用的数据存在，使相应的内存块数据利用率大大降低的情况存在，认为采用64K大小的内存块是比较合适。

4．3 FAT分区结构、功能和解决方案

一、 概述

在讨论了磁盘的分区表结构和读写内存块大小参数的确定后，本节将着重介绍FAT16和FAT32分区的结构，并利用其中相关的结构特点，在进行相关备份恢复操作时，获得了便利。

FAT (File Allocation Table：文件分配表) 文件系统起源于上世纪七十年代末和80年代早期。当初是被MS-DOS操作系统支持，是少于500K的小文件系统，适合于软驱。现在已经被发展成为支持越来越大的存储媒质。当前存在有三种FAT文件系统：FAT12、FAT16、FAT32。这3个FAT子类型的基础区别和名字的由来在于磁盘上实际的FAT表结构实体占有的位数。在FAT12文件系统中每个FAT表占有12位，FAT16文件系统中每个FAT表占有16位，FAT32文件系统中每个FAT表占有32位，所以，相应的可寻址大小也不同了。致使不同的FAT文件系统可以容纳的分区大小也各不相同。

一个FAT文件系统卷通常有四个部分组成，分别依顺序排列在文件系统卷中：

0 - 被保留区

1 - FAT表区

2 - 根目录区（但不存在于FAT32卷中）

3 - 文件目录数据区

下面将顺序介绍各个部分的特点，及相关特征功能。在本节最后将给出FAT文件系统备份恢复的具体解决方案。[8]

二、被保留区

在一个FAT卷中的第一个重要数据结构就叫BPB（BIOS Parameter Block：BIOS参数块），它位于保留区。这个扇区有时也称为“引导扇区”或者是“被保留扇区”，第0个扇区。但是最重要的事实是位于每个分区卷的第一个扇区中。

BPB经历了几次修改，特别是在Windows 95 OEM Service第二个发布版本上（OSR2），在这个操作系统版本上，使被FAT16长期限制在2G左右的分区容量，又再次被扩大，超过了传统的2G容量。FAT32文件类型的BPB结构和FAT12/FAT16的BPB几乎很相似，但包含了BPB_TotSec32字段。根据FAT分区类型的不同，他们在BPB中的位移量也不同。（请看后面关于判别FAT文件类型的讨论），相关的一点是，

在FAT分区卷中的BPB区应该总是包含有所有新的BPB字段域，不管是FAT12/FAT16还是FAT32。这样，就能确信这种FAT文件类型的最大容量，以及确认所有的FAT文件系统都能正确识别和支持这个分区卷。这一切都是因为，BPB总是包含了所有现存的，和某文件系统相关的字段域。

注：在以下的叙述中，所有字段名起于BPB_的都是BPB的一部分。所有字段名以BS_起头的都是引导扇区的一部分，但不是BPB的真正一部分。以下的显示了FAT分区卷的0扇区，它包含了BPB。

表4-15 引导扇区和BPB结构一

名字	偏 移 (字节)	大 小 (字节)	说明
BS_jmpBoot	0	3	引导扇区的跳转结构。这个数据域有两种可能的形式：jmpBoot[0]=0xEB,jmpBoot[1]=0x??,jmpBoot[2]=0x90 和 jmpBoot[0]=0xE9,jmpBoot[1]=0x??,jmpBoot[2]=0x?? 0x??表示任何一个8位值都是允许的。这种形式是Intel x86无条件跳转结构，跳转到操作系统的引导代码。引导代码一般位于分区0扇区紧接BPB后的剩余空间和一些可能的其他扇区中。
BS_OEMName	3	8	“MSWIN4.1”。此数据域只是一个字符串名。
BPB_BytsPerSec	11	2	每扇区字节数。这个值只可以取以下的值：512，1024，2048或是4096。如果考虑需要和原先的磁盘工具具有最大的兼容性，就应该选用512。
BPB_SecPerClus	13	1	每分配单位的扇区数。这个值必须是2的幂次方，大于0。合法的值是1，2，4，8，16，32，64和128。
BPB_RsvdSecCnt	14	2	保留区中的保留扇区的数目。起始于分区的第一个扇区。这个数据域必须不为0。对于FAT16和FAT16分区，这个值只能为1。对于FAT32分区，这个值一般为32（也有可能为36）。现在有很多分区处理程序默认保留扇区的长度为1，不再做其他检查。微软操作系统将支持一些非0的值。
BPB_NumFATs	16	1	分区上FAT数据结构的数目。对FAT分区的任一类型，这个数据域的值都应该等于2。
BPB_RootEntCnt	17	2	对于FAT12和FAT16分区，这个数据域包含了根目录中32字节目录实体的数目。对于FAT32来

			说，这个数据域的值必须为0。对于FAT12和FAT16分区，这个值将总是指定一个被32乘的数值。
BPB_TotSec16	19	2	这个数据域是老式16位寻址分区上总扇区数目。这个数目包含分区中四个部分的扇区总数。这个数据域值可能为0；如果是0，那么BPB_TotSec32值必须非0。对于FAT32分区，这个数据域值必须为0。对于FAT12和FAT16分区，这个数据域包含了扇区的数目，同时，如果总扇区数小于0x10000，那么BPB_TotSec32是0。
BPB_Media	21	1	0xF8是固定不可移动媒质的标准值。对于可移动媒质而言。0xF0经常被使用。
BPB_FATSz16	22	2	这个数据域是FAT12/16中一个FAT表的扇区数目。在FAT32分区中，这个数据域必须是0，BPB_FATSz32包含了FAT32分区中FAT表扇区长度。
BPB_SecPerTrk	24	2	每簇的磁道数，用于中断13。
BPB_NumHeads	26	2	磁头数目，用于中断13。
BPB_HiddSec	28	4	包含这个FAT分区物理位置前的隐藏扇区。
BPB_TotSec32	32	4	这个数据域是基于32位寻址的分区上总扇区数。这个数目包含分区中四个区域的扇区总数。这个数据域可能是0；如果它是0，BPB_TotSec16必须是非零的。对于FAT32分区，这个数据域必须非零。对于FAT12/16分区，这个数据域包含了扇区总数目如果BPB_TotSec16等于0（总数大于或是等于0x10000）。

在偏移36字节处，FAT12/FAT16文件系统的BPB/引导扇区不同于FAT32文件系统的BPB/引导扇区。表4-16表示了FAT12和FAT16在引导扇区偏移36处的数据结构，而表4-17表示了FAT32在引导扇区偏移36处的数据结构。

表4-16 引导扇区和BPB结构二(FAT12和FAT16在引导扇区偏移36处的数据结构)

名字	偏移(字节)	大小(字节)	说明
BS_DrvNum	36	1	中断13的磁盘号（譬如说：0x80代表了第一个硬盘）。这个数据域支持MS-DOS的引导跳转，也被设置成了中断13的磁盘驱动号（0x00指软驱，0x80指硬盘）。需要注意的是，这个值实际上是由操作系统来指定的。
BS_Reserved1	37	1	保留（在Windows NT下使用）。未使用。

BS_BootSig	38	1	扩展引导符号（0x29）。这是一个象征符号字节，表明以下三个数据域在引导扇区中都被提供。
BS_VolID	39	4	卷序列号。
BS_VolLab	43	11	卷标。
BS_FileSysType	54	8	“FAT12 ”，“FAT16 ”，“FAT ”中的一个字符串值。需要注意的是，这个数据域并不是决定当前分区类型的判断依据。

表4-17 引导扇区和BPB结构三(FAT32文件系统在引导扇区偏移36处的数据结构)

名字	偏 移 (字节)	大 小 (字节)	说明
BPB_FATSz32	36	4	这个数据域表示了FAT32一个FAT表的扇区数。BPB_FATSz16值必须为0。(a)
BPB_ExtFlags	40	2	扩展标识符。(a)
BPB_FSVer	42	2	表示了一个FAT32的版本号。(a)
BPB_RootClus	44	4	这个数据域被用来设置根目录下的第一个簇号。通常是2（除非此簇中包含坏地扇区）。(a)
BPB_FSInfo	48	2	在FAT32保留区中FSINFO结构的扇区数目通常是1。(a)
BPB_BkBootSec	50	2	如果不等于零，就表示了引导记录的一份拷贝中，分区的保留区域的扇区数。(a)
BPB_Reserved	52	12	保留为了以后的扩展。格式FAT32分区的代码将总是把这个数据域所有的字节置为0。(a)
BS_DrvNum	64	1	中断13的磁盘号（譬如说：0x80代表了第一个硬盘）。这个数据域支持MS-DOS的引导跳转，也被设置成了中断13的磁盘驱动号（0x00指软驱，0x80指硬盘）。需要注意的是，这个值实际上是由操作系统来指定的。
BPB_Reserved1	65	1	保留（在Windows NT下使用）。格式化FAT分区的代码应该把这个字节设置为0。
BS_BootSig	66	1	扩展引导符号（0x29）。这是一个象征符号字节，表明以下三个数据域在引导扇区中都被提供。
BS_VolID	67	4	卷序列号。
BS_VolLab	71	11	卷标。
BS_FileSysType	82	8	总是设置为“FAT32 ”。

注**a**：表示此数据域只在FAT32媒质中被定义，不在FAT12和FAT16媒质中存在。

关于FAT分区卷还有其他一个重要知识点。如果把扇区看成是一个字节数组的话，那么扇区数组的第511（索引号为510）个单元的数值等于0x55，扇区数组第512（索引号为511）个单元的数值等于0xAA。

很多的FAT文档错误的指出0xAA、0x55占用了引到扇区的最后两个字节。实际上，当且仅当BPB_BytsPerSec的值为512时，这句话才成立。如果BytsPerSec大于512，这两个驻留字节的偏移位置并不会改变（虽然它也表明了引导扇区的最后包含了这两个结尾字节符）。

检验BPB_TotSec16/32的值。会发现BPB中TotSec（包括BPB_TotSec16或者BPB_TotSec32中任意一个值非零的）小于或是等于DskSz的值。因为BPB_TotSec16/32的值会稍小于DskSz的值。

这说明了磁盘被浪费了。但这并没有说明磁盘在某些方面被破坏了。然而，如果BPB_TotSec16/32大于DskSz的话，则说明这个分区卷被严重的破坏了，因为它超过了这个分区媒介的边界或是在磁盘上产生了数据交叠。如果把一个BPB_TotSec16/32值过大的媒介或是分区当作有效的话，可能会导致重大的数据流失。

三、 FAT表部分

紧接着的一个重要的数据结构就是FAT表本身。这个数据结构主要是定义了一个文件扩展的单链列表。需要注意的一点就是，一个FAT目录或是一个文件容器只是一个常规文件，它拥有一个指定的属性，表示了它是一个目录。关于这个目录唯一不同的是，文件的数据或是内容是一系列32字节长的FAT目录实体（详见下文）。从其他意义来讲，一个目录就象一个文件一样。FAT表通过簇号来映射卷内的数据区。第一个数据簇是2号簇。2号簇的第一个扇区（磁盘上的数据区域）通过用相应卷上的BPB区域来计算。具体算法如下，首先计算被根目录占用的扇区总数：

RootDirSectors = ((BPB_RootEntCnt * 32) + (BPB_BytsPerSec – 1)) / BPB_BytsPerSec; ---（公式 4.1.1）

在一个FAT32卷上BPB_RootEntCnt值总是0，所以在一个FAT32卷上，RootDirSectors值一直为0。上面公式中32是表示一个FAT目录实体的字节数。

数据区的起始位置，2号簇的第一个扇区位，通过以下的公司获得：

```
if(BPB_FATSz16 != 0)
    FATSz = BPB_FATSz16;
else
    FATSz = BPB_FATSz32; ---（公式 4.1.2）
FirstDataSector = BPB_ResvdSecCnt + (BPB_NumFATs * FATSz) +
```

RootDirSectors；---.（公式 4.1.3）

扇区号，是相对于当前卷的第一个扇区而言。每个卷都包含一个BPB（包含BPB的扇区是第0号扇区）。并不一定要直接映射到驱动磁盘上，因为卷的0扇区并不一定就是驱动磁盘的0扇区，这依赖于分区。假设任何一个有效数据簇N，这个簇的第一个扇区的扇区号（还是相对于FAT卷的0扇区而言）通过以下方法进行计算：

FirstSectorofCluster = ((N – 2) * BPB_SecPerClus) + FirstDataSector；---（公式 4.1.4）

因为BPB_SecPerClus被严格限制必须是2的幂次方（1，2，4，8，16，32，...，）。另外，“总簇数”，不同于“最大有效簇数”，因为第一个数据簇是2，而不是0或者是1。

接着，讨论被根目录占用的扇区数。

RootDirSectors = ((BPB_RootEntCnt * 32) + (BPB_BytsPerSec – 1)) / BPB_BytsPerSec；---（公式 4.1.5）

在FAT32卷中，BPB_RootEntCnt的值总是0；所以在FAT32卷中，RootDirSectors值总是0。

继续讨论在指定卷中的扇区总数：

```
if(BPB_FATSz16 != 0)
    FATSz = BPB_FATSz16;
else
    FATSz = BPB_FATSz32；---（公式 4.1.6）
if(BPB_TotSec16 != 0)
        TotSec = BPB_TotSec16;
else
    TotSec = BPB_TotSec32；---（公式 4.1.7）
```

DataSec = TotSec – (BPB_ResvdSecCnt + (BPB_NumFATs * FATSz) + RootDirSectors)；---（公式 4.1.8）

现在讨论总的簇数：

CountofClusters = DataSec / BPB_SecPerClus；---（公式 4.1.9）

请注意以下计算采用了四舍五入：

现在可以讨论FAT文件类型了。以下例子中，当用<，这并不表示<=。请注意，以下公式。

```
if(CountofClusters < 4085) {
    /* Volume is FAT12 */
    }
else if(CountofClusters < 65525) {
    /* Volume is FAT16 */
     }
else {
    /* Volume is FAT32 */
    }---（公式 4.1.10）
```

这个公式是唯一的一个决定FAT类型的方法。FAT12不会超过4084个簇。同样，FAT16不会少于4085个簇，不会多于65,524个簇；FAT32不会少于65,525个簇。如果创建了一个FAT卷，但破坏了这个法则，微软的操作系统将不会正确地处理这些错误的卷，因为操作系统将会把这个卷误认为另一中FAT文件系统。

一般不要在创建FAT卷类型的时候，把卷大小分配在4,085或者是65,525个簇数左右。至少要在分配的时候，远离这些FAT卷分界线16个簇。

CountofClusters 值等于起始于簇 2 后的总数据簇。最大有效簇号是CountofClusters+1，“包括了两个保留簇后的总簇数”是CountofClusters+2。

在讨论关于FAT文件类型的计算中还有更重要的一点就是：假设任何一个有效簇号N，那么这个簇N的起始位置的公式，对于FAT16和FAT32相对简单，就是：

```
if(BPB_FATSz16 != 0)
    FATSz = BPB_FATSz16;
else
    FATSz = BPB_FATSz32;
---（公式 4.1.11）
if(FATType == FAT16)
    FATOffset = N * 2;
else if (FATType == FAT32)
```

FATOffset = N * 4;

---（公式 4.1.12）

ThisFATSecNum = BPB_ResvdSecCnt + (FATOffset / BPB_BytsPerSec); ---（公式 4.1.13）

ThisFATEntOffset = REM(FATOffset / BPB_BytsPerSec); ---（公式 4.1.14）

REM(...)是一个求余的运算符。那意味着FATOffset被BPB_BytsPerSec除下来的余数。ThisFATSecNum是第N簇的第一扇区的扇区号（也就是此扇区在分区中的逻辑位置）。

FAT32文件系统中，一个32位FAT表实体并不真的是32位；它们只是28位值。例如，所有这些表示簇号的32位实体：0x10000000、0xF0000000，和0x00000000。所有这些都表示这个簇是空簇，因为当你读取簇号时，高四位被忽略了。如果32位空簇值当前是0x30000000，为了把这簇标记为坏簇，就需要把值0x0FFFFFF7保存。然后32位值将包含0x3FFFFFF7，因为当你写入0x0FFFFFF7坏簇标记的时候，必须要保存高四位。相应的在FAT16文件系统中，用16位FAT表实体表示，0x0000表示空簇；0xFFF7表示坏簇。其他的表示已用簇。

因为BPB_BytsPerSec值总是可以被2和4整除，所以不必担心一个FAT16或是FAT32的FAT表实体超过一个扇区的边界（但这个事实对FAT12不成立）。对FAT12而言，计算公式将会更加复杂，因为FAT12中FAT表每个实体12位，占用了1.5个字节。公式如下：

if (FATType == FAT12)

FATOffset = N + (N / 2); ---（公式 4.1.15）

/* Multiply by 1.5 without using floating point, the divide by 2 rounds DOWN */

ThisFATSecNum = BPB_ResvdSecCnt + (FATOffset / BPB_BytsPerSec); ---（公式 4.1.16）

ThisFATEntOffset = REM(FATOffset / BPB_BytsPerSec); ---（公式 4.1.17）

由于在现在的操作系统中，FAT12文件类型几乎已经不被应用，所以本节也只是一点而过。不作详细介绍。

有一点需要着重关注的就是，在实际编码，进行对磁盘操作进行数据读写后，如果发现资源管理器中空无一个文件，但是，在分区分配符处却显示已用几兆，几十兆，

甚至几百兆的空间的时候。常常会感到很困惑，其实，这完全是因为FAT表中簇信息被误写，写成其对应的在ROOT域中存在的有效文件，但实际上并未存在任何文件信息。所以便导致了资源管理器中空空荡荡，但系统指示，此分区中装有大量不存在的文件的奇怪现象。

还有就是，在Windows 2000等等系统中要求对磁盘FAT分区进行快速格式化时，只需要清空此分区的FAT表扇区值就可以了，但是如果执行普通格式化时，需要对此分区的，除第一个拥有分区信息扇区外的所有扇区清零，这也就是普通格式化远慢于快速格式化的原因，快速格式化不会清空数据区，所以在执行了快速格式化后，还有希望找回失去的文件数据，但是普通格式化后就永远无法复原原来的文件数据。

四、 根目录部分

一个FAT目录只是一个包含32字节数据结构的线性列表。唯一特殊、必须被介绍的就是根目录。对于FAT12和FAT16文件系统，根目录处于磁盘上，在一个指定的位置。它紧接在最后一个FAT表的后面。大小固定，由BPB_RootEntCnt值表示（参见前文中关于RootDirSectors的计算）。对于FAT12和FAT16文件系统，根目录的第一个扇区相对于FAT卷第一个扇区的扇区号公式如下：

FirstRootDirSecNum = BPB_ResvdSecCnt + (BPB_NumFATs * BPB_FATSz16); ---（公式 4.1.18）

对于FAT32，根目录的大小就可能有各种不同的尺寸，是一个簇链，就和一些其他的目录一样。FAT32卷中，根目录中第一个簇值由BPB中的BPB_RootClus值给出。不象其他目录，在任何一个FAT文件类型中的根目录本身并不包含任何日期和时间戳属性，也并没有一个文件名（除了隐含的文件名“\”），也不包含作为目录中最先的两个“.”和“..”目录文件。唯一的根目录的有别于其他方面的特点是，它只是FAT卷中唯一一个仅拥有一个ATTR_VOLUME_ID属性文件的目录。

在实际程序编制过程中，着重注意了FAT16文件系统中的ROOT位置，并对其进行了恢复和备份。而FAT32文件系统中，ROOT区被合并成DATA区，而且它的长度不定，所以就统一根据FAT表中的簇号信息，视其是否是有用簇，来进行备份与恢复。

五、 文件目录数据部分

磁盘上，分区中的实际文件数据全都处于数据部分，而且因为在FAT32中，根目录部分也位于数据部分，所以这个部分也称之为文件目录数据部分。

文件目录的第一个扇区相对于FAT卷第一个扇区的扇区号公式已经由公式4.1.3给出：

FirstDataSector = BPB_ResvdSecCnt + (BPB_NumFATs * FATSz) +RootDirSectors;

六、 解决方案

针对FAT文件系统结构特点，备份将依照FAT文件系统的四个部分逐个顺序地进行备份，即被保留部分，FAT表（一般是两个）部分，根目录部分和数据部分。

被保留部分一般比较小，FAT12/16一般都只占用1到2个扇区大小，FAT32也只占用32到36个扇区，所以就进行扇区直接拷贝至映像文件，不做删除。被保留部分偏移位置即从分区的逻辑第一个扇区起。

对于FAT表部分，由于第二个表只是第一个表的原版拷贝，是一个镜象。所以只备份第一个FAT表就可以了，但FAT表的长度是依照此分区所能够分配的数据簇决定的，由于FAT文件系统会根据分配分区的不同，自动分配每簇对应的扇区数，所以也可以说，在给定FAT文件系统类型后，不同分区大小的FAT表是不同的。反之，当分区相同的时候，在同一个FAT文件系统类型下，此FAT表大小是一致的。可是，虽说如此，但一个已经被文件填满的分区和一个只是刚刚被格式化后的分区的FAT表信息却是完全不一样的，对一个刚刚进行格式化的分区，其FAT表信息，除了最先的两个记录外，全部置0（没有坏簇的话）。那在备份的时候，就不需要把所有的FAT表信息全部做个备份，只需要备份足够有用的FAT表扇区信息就可以了，对后面的无用0字节扇区，完全可以不予理会。那样除了可以节省几百乃至几千个扇区的FAT表大小外，也使得后面的数据部分备份提高了效率，具体理由请见下面解释。FAT表部分在物理布局上就紧随被保留部分。

在完成FAT表部分数据备份后，就是Root部分。在FAT12/16文件系统中，Root部分长度一般固定为32个扇区；FAT32文件系统中，没有Root部分，而把FAT12/16中Root部分的数据整合进Data部分。由于这个区域最多32个扇区，所以在FAT12/16文件系统下，在生成映象文件的时候，就采取了原版拷贝的办法。Root区的具体偏移位置参考公式 4.1.18，大小参考公式 4.1.1。

关于Data部分的文件数据备份，是按照FAT表中，对应的簇信息进行备份的。在Data部分中的数据，都是以簇为单位进行组织的。DATA部分内的数据是否有用，将会根据相应FAT表中的簇号值进行判断。以FAT32为例：0x00000000表示空簇；

0x0FFFFFF7表示坏簇；其他值就为已用簇，其对应的值显示的是该簇后面对应簇的起始扇区相对位置；0x0FFFFFFF表示当前文件对应簇结束；同理，相应地，FAT16中，0x0000表示空簇；0xFFF7表示坏簇；其他信息表示有用簇；0xFFFF表示当前文件对应簇结束。备份的时候，将会遍历FAT表中的簇号值，如果显示是已用簇，就通过公式4.1.19定位到Data区中的数据扇区，进行备份。在备份FAT表时，只备份了相应非空簇的扇区数据。所以在进行FAT表检索的时候，就避免了很多不必要的检索过程，当FAT表很长（即分区比较大），而分区上数据很少的情况下，大大提高了备份性能，缩短了时间，减少了生成的映象文件大小。

此外，在FAT表部分和Data部分的备份过程中，可能需要对上百乃至上千万个扇区进行读写备份，这就需要尽量减少读写磁盘的次数。在第4．2．4节读写内存块大小参数的讨论中已经给出了最适合的读写内存块大小64K，所以在备份和恢复的过程中，都会尽量把需要读写的数据整合在一个64K字节大小的内存块中（除了最后一次可能会小于64K），加快了读写的速度。提高了程序的整体性能。

最后，为了使备份的数据能够快速准确的恢复到目的磁盘（分区），在正式对磁盘进行备份以前，在映象文件的第一个扇区里，添加了很多源分区的信息。包括分区四个部分的扇区大小，可使用FAT表的扇区长度，每簇对应的扇区数，分区文件类型等等。也包含了验证信息，譬如说，映象文件密码信息，当映象文件生成的时候设置了密码，则在恢复分区的时候，也必须输入相同密码，否则将无法对分区数据进行恢复，而且在对密码保存的时候，使用了加密算法，即使使用两进制读写文件，打开了映象文件也无法获得密码信息，提高了映象文件的安全性。而且由于程序生成的文件也是以img结尾的，为了和一些其他软件生成的文件区别。在此额外扇区的头信息处，加入了一些标准识别字符，用于在恢复时，判断此映象文件（.img文件）是否是由本程序生成的，如果不是，则不允许进行恢复。当为了生成指定大小映象文件，而使用了分割选项，或是当前生成文件大小会大于4G（FAT表最大文件不能大于4G），则每一个被生成的文件都在头文件处设置了序号，只有先选第一个文件（其后的文件按照顺序会自动被恢复），才能进行文件恢复。保证了程序的安全可靠性。使程序可以识别一些其他软件生成的以img为后缀的文件，提高了软件的鲁棒性。

4．4 NTFS分区结构、功能和解决方案

一、概述

在讨论了FAT16和FAT32分区的结构以及相应的解决方案以后，本节将详细讨论NTFS的结构，利用其中相关的结构特点，详述NTFS的备份与恢复操作，提出解决放案。

NTFS文件系统格式是伴随着Windows NT操作系统的产生而产生的。NTFS使用了64位的簇索引，从而能够使NTFS文件系统能寻址到一个高达160亿GB的分区；然而Windows 2000把NTFS分区限制为只能寻址32位的簇，也就是128万亿字节（使用了64-KB大小的簇）。[9]

NTFS包含了很多高级的特征，譬如说文件目录级安全，磁盘配额，文件压缩，基于目录的符号链接，和加密。其中最主要的就是可恢复能力。如果一个系统意外当机，FAT分区的元数据就只能是处于一个不连续的状态。 NTFS日志以一种事务的方式改变了分区元数据，以至于文件系统结构能被修复到一个连续的状态，避免了文件或者目录结构信息的丢失（文件数据可能会丢失）。

NTFS具有众多优点。主要就缘由其在此磁盘结构上独特的实现方法。接下来将具体介绍NTFS如何组织文件和目录，如何存储文件属性与数据的。

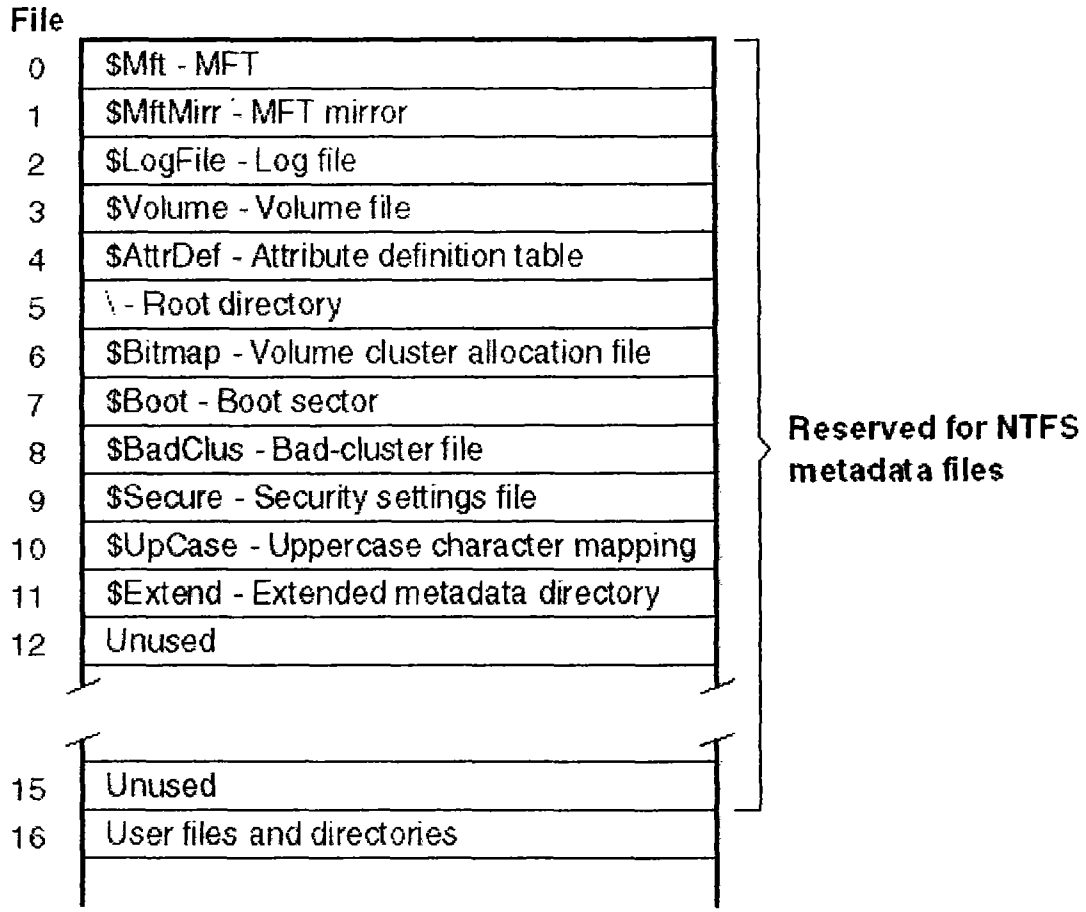
二、**MFT**（主文件表）

NTFS文件系统中，所有的数据，甚至包括所有的元数据（包括用于文件定位和恢复的数据结构、引导程序数据以及记录整个卷分配状况的位图等）都是保存在文件中的。这对于文件系统而言，很不寻常。但是将所有数据都存放在文件中可使得文件系统很容易进行数据定位和处理。而且，每个单独的文件可以通过安全描述符来保护。另外，如果磁盘的一个部分损坏，NTFS可以重新定位元数据，从而防止磁盘访问失效。

主文件表（MFT）是NTFS分区结构的核心。MFT是由一组文件记录数组组成。每个文件记录大小固定为1KB，和每个簇大小无关。逻辑上，MFT包含分区上每个文件的记录，也包含MFT本身的记录。此外，每个NTFS分区都包含一串包含用来执行文件系统结构的元文件。这些元文件都有一个起于“$”符号的名字，虽然这个符号

是隐藏的。例如，MFT的文件名是$MFT。在这些文件后面就是普通用户文件和目录。元数据文件如下图所显示：

图4-4 MFT中NTFS元文件的文件记录



通常，每个MFT记录都对应一个不同的文件。如果一个文件有很长的属性，或者是高度被分割的（一个文件放在多个不连续的簇），就需要不止一个记录。在这种情况下，MFT第一个记录将存储其他记录的位置，第一个记录也称之为基础文件记录。下表简要描述了NTFS文件系统MFT的元数据文件。

表4-18 MFT的元数据文件 [9]

1：$Mft	MFT本身
2：$MftMirr	MFT镜像。备份MFT文件内容

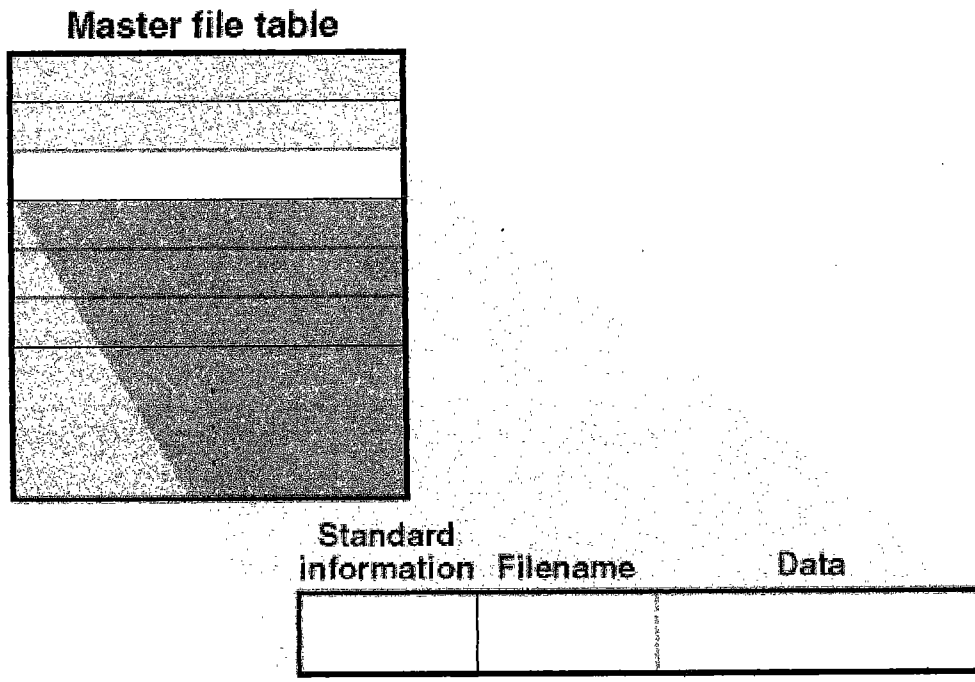
3：$LogFile	日志文件。主要用于恢复NTFS和保障NTFS的安全性。
4：$Volume	卷文件。标明卷名，被格式化的卷的NTFS版本和一个标注该磁盘是否损坏的标志位（用来判断是否需要用chkdsk来修复）
5：$AttrDef	属性定义表。卷中所支持的所有文件属性，并指出是否可以被索引和恢复等。
6：$\	根目录。保存了存放于该卷根目录下的所有文件和目录的索引。在访问了一个文件后，就保留该文件的MFT引用，第二次就能够直接对该文件进行访问。
7：$Bitmap	位图文件。NTFS卷的分配状态都存放在位图文件中，其中每一位代表卷中的一个簇，标识该簇是空闲的还是已经分配的。
8：$Boot	引导文件。存放着Windows的引导程序代码。必须位于特定的磁盘位置才能正确引导系统。实在Format时创建的。体现出NTFS把分区中所有的事务都视为文件的原则。此文件享受NTFS的安全保护，但也还是可以通过普通的I/O操作予以修改。
9：$BadClus	坏簇文件。记录了磁盘上该卷所有损坏的簇号，防止系统对其进行分配使用。
10：$Secure	安全文件。存储了整卷的安全性描述符数据库。NTFS文件和目录都有各自的安全描述符，为了节省空间，NTFS将具有相同描述符的文件和目录存放在一个公共文件中。
11：$UpCase	大写文件。包含了一个大小写字符转换表。
12：$Extended	扩展元数据目录。
13--16	预留。尚未使用。
〉16	其他文件和目录。

MFT的前16个元数据文件相当重要，为了防止数据的丢失，NTFS系统在该卷文件存储部分的正中央对其进行了备份。NTFS把磁盘分成了两个部分，其中大约12%分配给MFT，以满足其不断增长的文件数量。为了保持MFT元文件的连续性，MFT对这12%的空间享有独占权。余下的88%的空间被分配用来存储文件。而剩余磁盘空间则包含了所有的物理剩余空间（包括了MFT剩余空间）。MFT空间的使用机制可以概括为，当文件耗尽了存储空间，Windows操作系统会简单的减少MFT空间，并分配给文件存储。当有剩余空间的话，这些空间又会重新划分给MFT。虽然系统尽力保持MFT空间的专用性。但有时也不得不作出牺牲。尽管MFT碎片有时无法忍受，但却无法阻止。

三、文件记录

NTFS将文件作为属性/属性值集合来处理，这一点与其他文件系统不一样。文件数据就是未命名属性的值，其他文件属性包括文件名、文件拥有者、文件时间标记等。

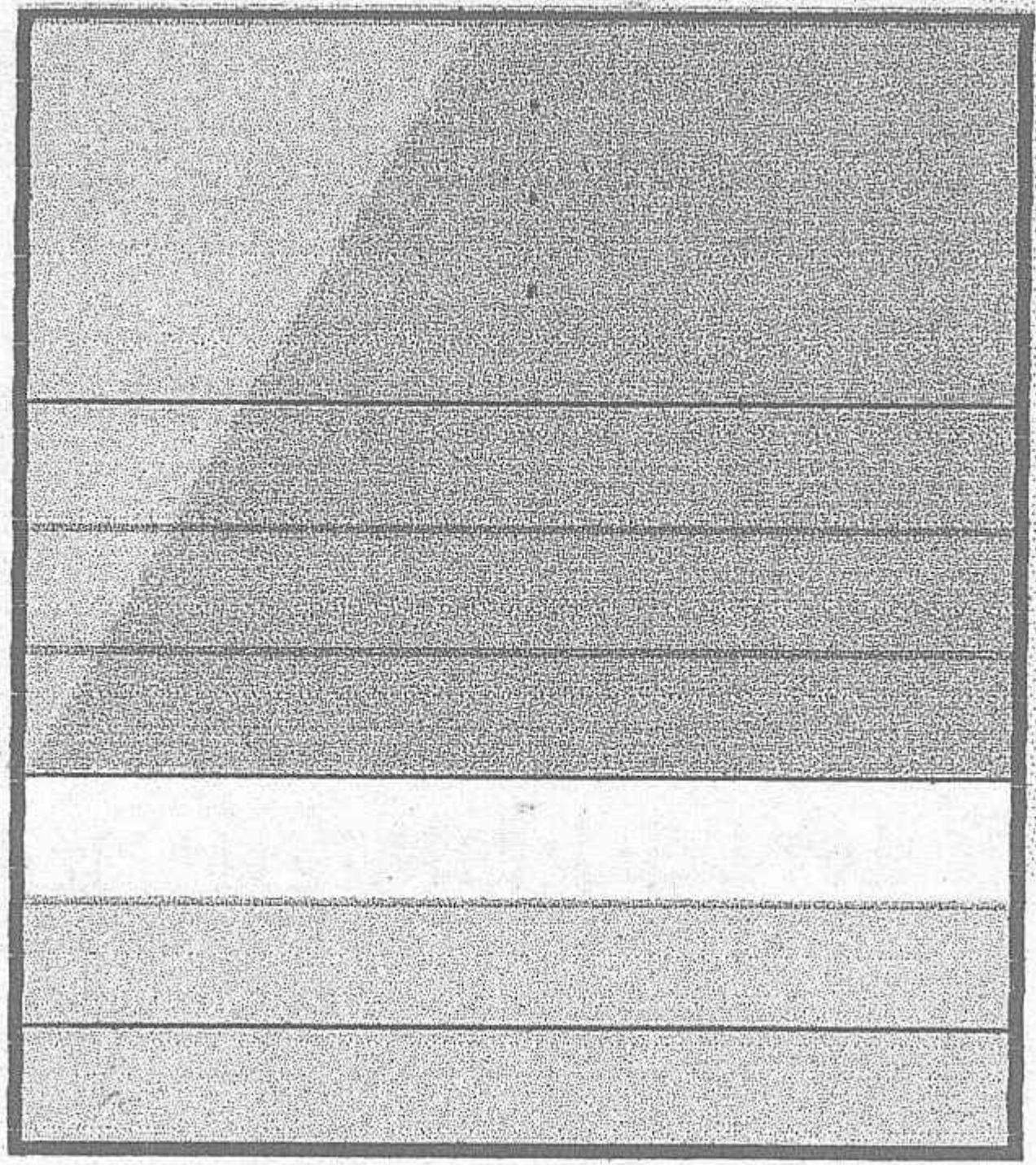
图4-5　小文件的MFT记录 [4]



每个属性由单个流（stream）组成。即简单的字符队列。严格地讲，NTFS并不对文件进行操作，而只是对属性流的读写。NTFS提供了对属性流的各种操作：创建、删除、读取（字节范围）以及写入（字节范围）。

表4-19 NTFS常用属性说明 [9]

属性值	属性名	属性描述
0x10	$STANDARD_INFORMATION	标准信息：包括了基本文件属性，如只读、存档；时间标记，如文件的创建时间和最近一次修改时间；有多少目录指向本文件（即它的硬链接数目）。Windows 2000加入了四个新数据域，用来引用配额、安全性、文件大小和日志信息。如同$AttrDef中定义的，这个属性最长72字节，最短48个字节。
0x20	$ATTRIBUTE_LIST	属性列表:当一个文件需要使用多个MFT记录时，这用来表示该文件的属性



		列表，因此，不经常使用。
0x30	$FILE_NAME	文件名：以Unicode字符表示。由于MS-DOS不能正确识别Win32子系统创建的文件名，当Win32子系统创建一个文件名时，NTFS会自动生成一个备用的MS-DOS文件名，所以一个文件可由多种文件名属性。
0x40	$VOLUME_VERSION（NT）	版本信息
	$OBJECT_ID（2K）	对象ID:具有64个字节的标识符，其中最低的16个字节对卷来说唯一。
0x50	$SECURITY_DESCRIPTOR	安全描述符：为向后兼容而保留的，用于保护文件以防止未授权访问
0x60	$VOLUME_NAME	卷名称或卷标识:仅存于$Volume元数据文件。
0x70	$VOLUME_INFORMATION,	卷信息:仅存于$Volume元数据文件；
0x80	$DATA	文件数据：是文件的内容。一个文件的长度就是未命名的数据流的长度。
0x90	$INDEX_ROOT	索引根:这是执行索引的B+树的根结点(譬如：一个目录)。这个文件的属性总是常驻的。
0xA0	$INDEX_ALLOCATION	索引分配：是一个索引的基础组件(譬如：一个目录)。这是一个执行索引（譬如：一个目录）的B+树的所有字节点的存储位置。这个属性总是不常驻的。
0xB0	$BITMAP	位图：是一串比特，每个都代表了一个实体（簇）的状态。
0xC0	$SYMBOLIC_LINK（NT）	符号链接。
	$REPARSE_POINT（2K）	重解析点：存储文件的重解析点数据（NTFS的软链接与装配点都包括这个属性）。
0xD0	$EA_INFORMATION	扩充属性信息：主要为与OS/2兼容，现已使用不多。
0xE0	$EA	扩充属性：主要为与OS/2兼容，现已使用不多。
0xF0	$PROPERTY_SET（NT）	属性设置
0x100	$LOGGED_UTILITY_STREAM（2K）	EFS加密属性：主要为了实现EFS（encrypted file system）而存储有关加密信息如解码钥匙、合法访问的用户列表。

四、文件目录的组织形式

NTFS文件的目录文件是通过B+树索引的，大大的减少了文件的索引深度。达到了索引时间的最小，最优化。下面将就简单介绍一下B+树结构，和使用的搜索方法。B+树的概念是递归的概念。

一棵顺序为m的B树的概念：[10]

1）根节点或者是叶子节点，或者至少拥有两个子节点的节点。

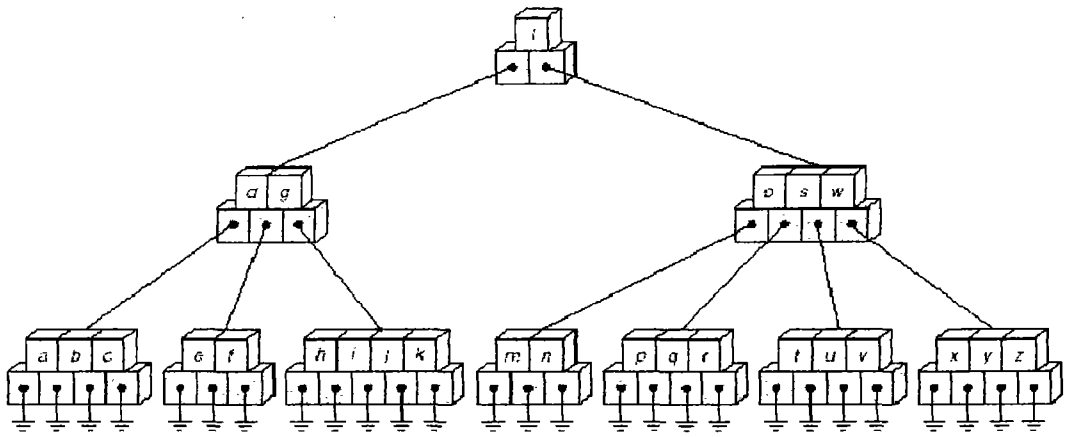
2）如果不是叶子节点，譬如说是根节点，则此根节点拥有m/2至m个子节点。

3）如果不是叶子节点，对于每个节点，关键字值数等于字节点数目减一。

4）树上所有的叶子都位于同一层上，所以此树高度总是平衡的。

5）每个非叶子节点的关键字都大于它的左子树包含的所有关键字，小于它的右子树包含的所有关键字。

图4-6　一棵顺序为4的B树



B+树是以B树为基础的。具体的B+树概念：

1）内部节点只存储关键字和指针*。（*在一些小型数据库中，使用B-树作为一个直接数据结构，把记录存储在节点的方法是很普遍的。）

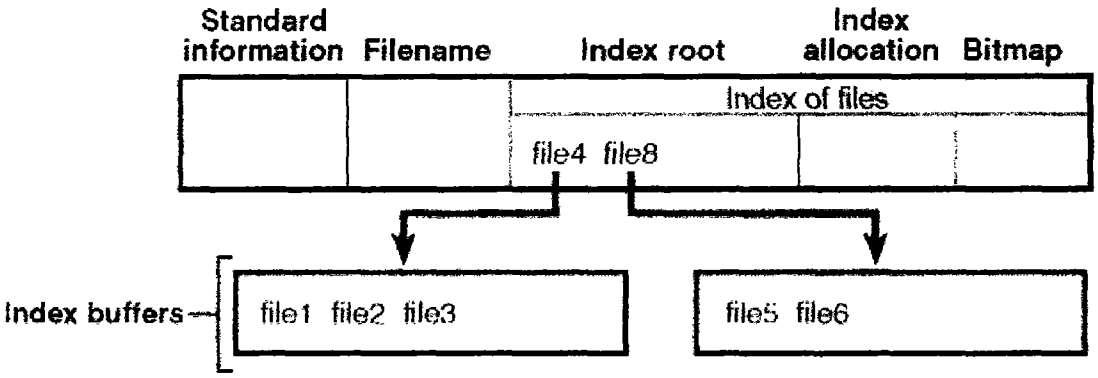
2）所有记录，或者是指向记录的指针存储在叶子节点中。

3）通常说，叶子可以简单地视为一个数据库文件索引的逻辑块，存储了关键字和偏移。在这种情况下，很多关键值在这棵树中将产生两次，一次在内部节点中来引导搜索，后一次在叶子中。

4）如果叶子只作为一个简单的索引，则能够把B+树所有叶子节点视为一个链表

结合NTFS文件目录组织结构，一个大的目录也可能拥有非常驻属性（或是部分属性），就如同下图显示。

图4-7　大目录MFT文件记录的非常驻文件名索引 [4]



在此图中，MFT文件目录并没有足够的空间容纳此目录下的所有文件索引。所以，只有部分文件索引被存在索引根目录属性中，剩余的一部分索引被存放在一个称之为index buffer的非常驻属性中。索引根，索引分配和位图属性被比较简单的表示了。标准信息和文件名属性总是常驻的。头信息和至少部分的索引根属性值对于目录而言也是常驻的。而Index root中的节点信息file4和file8也就是B+树中的内部节点。Index buffers第一个框中的中的file1、file2、file3都是小于file4的。同样的，第二个框中的file5和file6都是大于file4而小于file8的。就这样，NTFS形成了一个B+树的文件目录结构，便于程序进行快速搜索指定的文件目录。不过，备份恢复程序并不需要搜索指定的文件，而是需要遍历所有的目录和文件（包括系统保护的文件和目录）。具体实现中，使用了先序遍历所有目录和文件的方法，备份了所有遍历经过节点的对应文件数据，当然也可以使用中序或是后序遍历的方法。父目录与子目录、子文件间通过file reference进行关联，子文件和子目录中除了有一个file reference属性表示自己的文件参

考号，还有一个file reference to parent，可以看成是B+树中的指针的概念，只不过这个属性从叶子处指出了它的父节点。

五、解决方案

了解了 NTFS文件系统后，将具体分析研究文件系统的恢复与备份。在任何一个文件系统中，数据文件都是按照目录和文件形式排列的，只是在具体组织方式上不同，在FAT文件系统中，是通过ROOT域和Data域结合起来，实现文件数据的组织，具体占用了哪些簇，则是通过FAT表中的各个字节值表示的，如前一节所述，如果是FAT12的，FAT表中用1个半字节（12bit）来表示簇占用情况；如果是FAT16，那就用2个字节（16bit）表示；如果是FAT32的话，那就用4个字节（32bit）表示。而在NTFS文件系统中，相应的文件目录组织情况就不一样了。所有信息都是作为文件信息存储，包括目录。当当前目录中包含的文件或是子目录数目过多的时候，将会使用一个$Index Allocation属性对部分文件进行索引，并使用$Bitmap表示相应的簇使用情况，如上一节说叙述的。如果文件信息不多的时候，使用数据（常驻属性），就可以把文件信息全部放在此文件信息中。可是当文件信息过多的时候，就会另外分配一个称之为数据运行（Data Run）的空间，把文件中的信息存储，同时在源文件处，使用了一个Data Run属性，此属性的值指向对应文件数据所储存的逻辑簇的位置。具体请见下例：

表4-20 数据(非常驻)属性

80	00	00	00	48	00	00	00	01	00	00	00	00	00	07	00
00	00	00	00	00	00	00	00	09	00	00	00	00	00	00	00
40	00	00	00	00	00	00	00	00	50	00	00	00	00	00	00
20	49	00	00	00	00	00	00	20	49	00	00	00	00	00	00
31	0A	2C	F4	01	00	9D	81	FF	FF	FF	FF	82	79	03	00

在数据属性偏移0x40处，就是此非常驻文件的逻辑簇位置。31 0A 2C F4 01 00中，0x31表示1字节长度，3字节偏移量。具体长度就是0x0A（10）个簇；偏移位置（相对分区的逻辑第一簇而言）就是0x1F42C（128044）簇。而相应的其他数据表示的含义：

表4-21 NTFS数据属性(0x80)实例 [11]

偏移	大小	说明	属性值（十六进制值）
0x00	4	属性类型	80
0x04	4	长度（包括属性头）	48
0x08	1	非常驻符号	01
0x09	1	命名长度	00
0x0A	2	命名的偏移	00
0x0C	2	标志	00
0x0E	2	属性ID	07
0x10	8	起始VCN	00 00 00 00 00 00 00 00
0x18	8	结尾VCN	09 00 00 00 00 00 00 00
0x20	2	Data Run的偏移长度	40
0x22	2	压缩单元大小	00
0x24	4	填料	00
0x28	8	属性被分配的大小	00 50 00 00 00 00 00 00
0x30	8	属性的实际大小	20 49 00 00 00 00 00 00
0x38	8	初始化后流的大小	20 49 00 00 00 00 00 00
0x40	...	Data Run	31 0A 2C F4 01 00 9D 81

注：1）属性被分配的大小：0x5000（20480）个字节，合40扇区，10簇（此例情况为一簇对应四个扇区）。2）属性的实际大小/初始化后流的大小：0x4920（18720）个字节，合36.56125个扇区。

文件和文件所在的目录之间，通过相应的File Reference属性值进行联系，NTFS文件的目录文件是通过B+树索引的，大大的减少了文件的索引深度。达到了索引时间的最小，最优化。另外，NTFS文件系统除了引导信息，文件具体数据外，其他元数据文件，索引文件、日志文件重启信息的相应数据起始都有固定格式，分别是“FILE”、“INDX”、“ RSTR”。

针对NTFS文件系统的结构特点，在具体进行文件数据备份和恢复的时候，就先备份分区的前4K字节的引导记录和16个元数据文件。这点信息除了具体分区的数据大小、创建时间等等个别数据域值不同外，基本一致。然后根据根目录的信息，得到

根目录的File Reference，然后按照先序节点遍历的方法，依次遍历子目录信息和此子目录下的子目录或是文件信息，一一进行备份。如果还有文件分配出来的Data Run信息，也根据文件数据非常驻属性中此Data Run指示的数值，查找到此文件具体数据内容对应Data Run所位于的逻辑簇位置，进行备份，由于所有的位置都是具有相对位移的，而且每个文件数据占用的空间都是扇区的整数倍，一般为一个簇。所以在备份每个文件数据的时候，都会在其文件都上加上一个双字节数值，标示此文件数据在源分区的逻辑位置。用于在恢复的时候能够正确的按记录的偏移值恢复。

另外，由于在NTFS文件系统中，当一个文件被删除的时候，并不会简单的把此相关的文件记录从MFT中删除，因为那样的话，文件记录号将不会连续，所有的文件参考号都必须被更新。相应的，文件记录中一个有一个标志位可以指出，此文件不再被使用。然后，当一个文件再次被新建后，这个未被使用的文件记录可以再次被使用。但是它的顺序号被加了一。这种机制能够允许NTFS来检查此文件索引不再指向被删除的文件。具体可以见下图所示：

图4-8　文件记录头属性信息

```
0000bc00h: 46 49 4C 45 2A 00 03 00 F2 24 C0 01 00 00 00 00 ; FILE*...??....
0000bc10h: 02 00 01 00 30 00 01 00 48 01 00 00 00 04 00 00 ; ....0...H.......
0000bc20h: 00 00 00 00 00 00 00 00 06 00 02 00 00 00 00 00 ; ................
```

上图是一个NTFS文件信息的头信息部分，一般在每个文件数据的头信息偏移0x16处，在本例中就是0xBC16处的值就是表示此文件属性的标志。

表4-22　标志值及相关的含义

标志符值	相关含义
0x00	此文件信息表示的文件/目录已被删除。
0x01	此文件信息表示的文件正被使用。
0x02	此文件信息表示的是目录。
0x03	此文件信息表示的目录正被使用。

上例中，此标志值为0x01，即表示此文件信息表示的文件/目录正被使用。所以在搜索文件信息（包括目录信息，即所有以“FILE”起头的扇区数据）的时候，当遍历过的文件信息，对应的符号值为0x00的时候，就直接跳过。查找下一个文件信息。避免了对许多已经删除的文件或是子目录的遍历。这样即实现了NTFS文件的恢复与备份。

4．5 结论与将来的工作

一、 结论

根据对读写磁盘的速度比较，获得了最佳可读写的内存块大小。在以后的编程过程中，按照这个大小，进行磁盘读写，使读写速度最优化。根据前面的介绍，在编写代码的时候，除了在保留扇区区域，按照FAT不同的类型使用了不同的内存块大小（在FAT16中一般为1个扇区，但也会有2个扇区情况存在；在FAT32中，一般为32个扇区，但也会有35个扇区情况存在。相应的在BPB[BIOS PARAMETER BLOCK]偏移14字节的位置，共占用2个字节大小，具体可以见第4.3节 FAT分区结构及相关功能），其他地方都采用了每次读取64K字节的方法。

二、 将来的工作

根据前几节的讨论，已经为正式实现磁盘的恢复与研究，做了比较充分的准备。掌握了FAT表的数据结构，知道了分区表的单链结构，根据实验得出了较佳读写内存块大小，分析了BPB表，得出了FAT文件类型的四个区域大小位置等等在恢复时需要使用到的诸多公式。为后面实现在WINDOWS系统下磁盘备份和恢复操作，做了充分的技术储备。在后面一章中，将具体介绍在现阶段被广泛应用的微软WINDOWS操作系统下，磁盘备份恢复的具体实现。

第二部分 基于Windows9x/ NT操作系统下的磁盘恢复与操作实现与研究

第五章 Windows 9x操作系统及相关技术介绍

5.1 基于Windows 9x内核的操作系统概述

几年以前，传统的操作系统（如MS-DOS）提供给用户的界面，是一种以字符为基础的命令接口，用户以行方式输入命令来执行程序和控制程序的运行，如不了解操作系统的命令格式，很难运用自如。

美国Microsoft公司开发的Windows操作系统用窗口界面替代了传统的命令行界面，提供了一种可视化操作环境。它使计算机的操作方式发生了深刻的变化，用户通过窗口中的菜单、图标和对话框等就可以启动和控制程序的执行及文件的管理，再也无需为记不准操作命令的格式和参数而烦恼。[12]

Windows作为一种新型的单用户多任务的操作系统，近年来发展迅猛。不管是单机运行还是联网运行，绝大多数的微型计算机都配置了不同版本的Windows操作系统。包括Windows95、Windows98（操作界面极为相似，统称Windows9X）和Windows 2000等基于NT内核的操作系统。

Windows 98是Windows 9x操作系统中最著名版本。它作为一个操作系统，特别是PC环境的操作系统，终于在系统文件管理方面突破了DOS概念，进入了真正的32位环境。同时它也将最新的多媒体技术、网络技术和Internet/Intranet 技术结合在一起，提供了比较全面的基础设施支持环境软件和工具。

5．2 中断指令INT13

INT13和INT13扩展中断指令[13]

近年来硬盘容量飞速增长，令广大电脑用户惊喜不已。99年主流还是6.4G，而到2000年10G 也不算大了,慢慢得，20G的硬盘逐渐成为市场主流，发展到现在主流已经是80G、120G容量的大硬盘。一时间硬盘容量的增长速度似乎快过了主板、操作系统等配套软硬件的发展脚步。

为了解决好大容量硬盘的使用问题，应当先了解一下这个问题出现的原因。尽管个人电脑的发展飞速，但出于向下兼容等考虑，今天磁盘驱动器的I/O结构仍是建立在早先的DOS-BIOS的分层结构上，即：应用程序--DOS功能调用---INT13中断读写---BIOS磁盘服务例程----ATA界面。

INT13作为一种磁盘I/O读写的重要中断指令函数在WINDOWS基于NT的操作系统出现前，一度被广泛运用在DOS、LINUX和WINDOWS 9X操作系统中。但INT13只有读/写1024柱面之前数据的能力。这也就是为什么操作系统的启动引导程序要放在1024柱面之前。就譬如说，LINUX操作系统的KERNEL程序，不管是通过1）BIOS——>LILO（在MBR中）——>KERNEL；还是2）BIOS——>MBR——>LILO（在活动分区的第一个扇区）——>KERNEL，来用LILO引导LINUX都必须要保证KERNEL放在1024柱面之前。只有在KERNEL起来以后，才有读/写1024柱面以后数据的能力。因为LINUX不使用INT13来进行硬盘操作。从上面说明也可以看到，不存在什么“WIN95可以，而LINUX不可以”的问题，作为操作系统要能被正确引导，在现有的BIOS下，它们的引导部分都必须在1024柱面之前。如果操作系统本身还是基于INT13来进行磁盘操作的话，那么它也只能读/写1024柱面之前的数据。

原先在CHS模式中，只能访问8GB以内的空间。为了访问磁盘的8GB以外的空间，在BIOS中，为了突破INT13的读/写磁盘限制，INT13中断扩展函数被产生，并且在LBA寻址模式中被广泛地使用。在本程序中INT13扩展在读取磁盘的物理参数、锁卷和解卷等方面被广泛应用。但由于在Windows 9x以上版本中，屏蔽了系统INT13中断请求，使得INT13在读写磁盘扇区时，显得力不从心。

5．3 THUNK技术

在应用程序制作过程中，需要在Windows 9x应用程序下调用一个16位的DLL动态链接库，而这个库中具有读取绝对扇区的功能。一般的来说，一个32位程序是无法调用16位API的，但Windows 9x提供了一种THUNK技术，使这变得可能。

任何关于WIN32操作系统的文章，都会出现THUNK这个字，这个术语最初是由发展编译器的人使用的，意思是："a piece of code，generated by the compiler，which evaluated an l-value or r-value of a parameter"。而当WINDOWS使用了THUNK这个字后，它的意义便是："an address that can be called by a process that refers to a system generated piece of code which does something, and then calls a real function address" [14]。

THUNK的行为是在截取程序不同部位时发出的函数调用动作，有时候用来决定地址，有时候用来转换地址等等。Win32s的组成部分（各 DLLs、EXEs）中，最重要的部分，就称之为"thunking layer"。

其实Win32s並不是第一个企图进入32位程序的 Windows程序，第一个有此打算的是 Microsoft 的 WINMEM32.LIB (1991/03 的 MSJ 有一篇 Porting 32 bit AP to Windows 3.1 with the WINMEM32 Lib)。MetaWare 和 Rational Systems 两家公司也有32位程序的。

Windows Extender（它对Windows 的关系相当于32位DOS Extender相对于MS-DOS的关系）。某些32位的产品如 Mathematica (Wolfram Research 出品) 也已经出现在市场上。

但不同于这些已经建立的32位WINDOWS程序，WIN32s承诺的是标准性，它使用的是与WINDOWS NT相同的WIN32 API，并且也因为它来自于MICROSOFT。事实上WIN32s可以说是WINDOWS操作系统在32位程序的扩充部分，绝不只是一个过渡品。每一份WINDOWS 3.1操作系统，都知道什么是WIN32S：WINDOWS 3.1的KRNL386内含一个ExecPE()函数，只要使用者想在Windows 3.1中执行一个Win32 PE(Portable Executable)文件，ExecPE()就会搜寻并企图载入一个W32SYS.DLL文件。此文件是Win32s的组成部分，从这里开始启动Win32s子系统。

如果曾经使用过或者考虑过DOS Extender，是否考虑过：[原先的DOS程序内调用了买来的LIB（没有原始码），现在为了加入DOS Extender，重新编译的程序如何

与这买来的LIB再次成功联结]？这问题同样出现在Win32s身上，但这次有解答，Universal Thunks可以完成使命。这也是唯一一个存在于Win32s，而不存在于Win32的特质（Win32根本不需要什么thunking）。本磁盘备份恢复程序就是以一个32位动态链接库disk32.dll调用一个16位的动态链接库disk16.dll读取磁盘绝对扇区。本程序借助Win32s API的Universal Thunking Mechanism提供的标准界面，在32位应用程序与16位的dll之间搭起了一座鹊桥。简单地说，要为32位应用程序制作一个32位的“interface DLL”，它透过系统提供的“stepdown thunk”与另一端的16位的“interface DLL”搭上，再由16位的interface DLL调用16位的DLL函数。[15]

32 位应用程序

↑

↓

32 位的 interface DLL (程序员负责完成)

↑

↓

stepdown thunk (系统内建)

↑

↓

16 位的 interface DLL (程序员负责完成)

↑

↓

16 位的 DLL

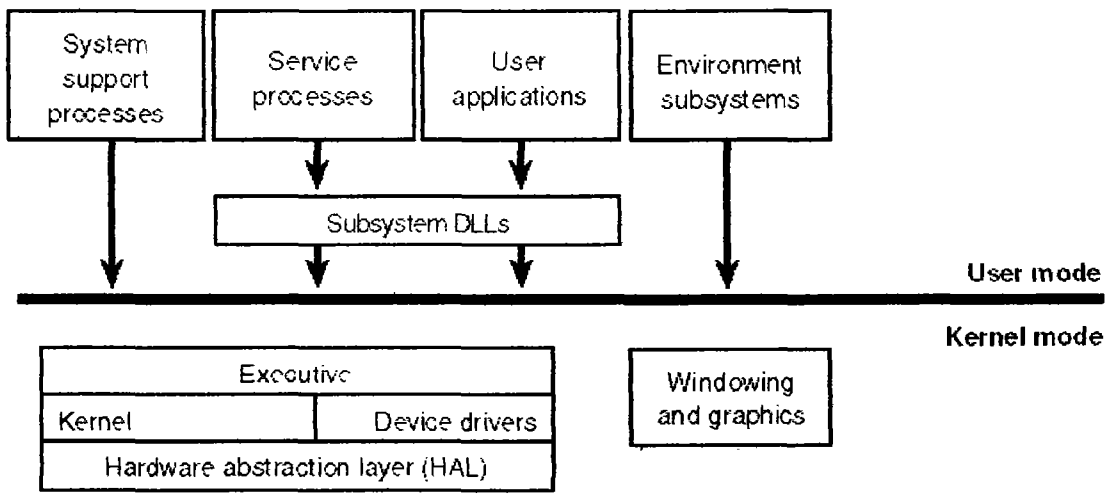
通过了使用了THUNK技术，在应用程序中，使disk32.dll调用一个16位的动态链接库disk16.dll去读取磁盘绝对扇区，解决了Windows 9x操作系统下磁盘绝对扇区的读写技术难题。

第六章　Windows NT操作系统及相关技术介绍

6．1　Windows NT操作系统概述

Windows 2000，原名Windows NT 5.0。它结合了Windows 98和Windows NT 4.0的很多优良的功能/性能于一身，超越了Windows NT的原来含义。 Windows 2000平台操作系统采用NT的技术，并在其上作了大量的改进，使得Windows 2000操作系统平台比此前的Windows操作系统平台更加可靠、更易扩展、更易部署、更易管理、更易使用。在存储部分最关键的就是，它把硬件接口分装，做了一个抽象层（HAL），具体Windows 2000系统结构见下图。HAL使读写外设都变成了标准文件读写，屏蔽了很多外设驱动的异同性，在下一节中，将具体给出说明。

图6-1　Windows 2000系统简单结构示意图



6．2 Windows NT操作系统下跨硬件平台性

基于NT内核的操作系统，特别是从WINDOWS 2000操作系统开始，在设计中多了一个关键的部位就是可以跨各种不同的硬件平台。而使这一切变为可能的就是HAL(Hardware Abstraction Layer)。HAL是一个可装载的内核模块（Hal.dll），它能提供一个低层次的硬件接口。隐藏了硬件依赖性，譬如说I/O接口，中断控制，和多处理器通信机制---其中任何一个功能都是和架构有关，依赖于机器的。

代替了直接对硬件的访问，Windows 2000内部组件就如同用户写的设备驱动，当需要和平台相关信息的时候，通过调用HAL程序保持了一种与硬件无关性。

由于Windows 2000封装了硬件，把读写不同的PC硬件，全部视为是标准部件来看。所以，相对于Windows 9x中纷繁复杂外设驱动而言，Windows 2000中，读写磁盘方便了很多。只要使用标准读写文件API就可以了。譬如说：ReadFile()、WriteFile()、SetFilePointer()等等[18]。通过这些API函数，就可以方便有效的读取磁盘绝对扇区信息。

第七章　不同WINDOWS内核操作系统扇区读写

一、概述

在了解了分区表和不同的文件系统结构后，本章将具体讨论在Windos 9x和Windows 2000下的磁盘扇区读写方法与过程。

在DOS操作系统下，通过BIOS的INT13、DOS的INT25（绝对读）、INT26（绝对写）等功能调用实现对磁盘逻辑扇区或物理扇区的读写是很方便的，C语言中还有对应上述功能调用的函数：biosdisk、absread和abswrite等。但WINDOWS操作系统下编写WIN32应用程序时却再也不能直接使用上述的中断调用或函数了。其实WINDOWS操作系统也提供了读写磁盘扇区的方法，只是在不同的版本中有着不同的方式和使用限制。本章将先介绍不同Windows系统读写磁盘扇区的原理和方法，并在本章最后编写了一个磁盘扇区直接读写类用代码实现。

二、Windows 9x操作系统下读取逻辑扇区的例子

WINDOWS操作系统对所有的存储设备实行了统一管理，而且为了安全起见，操作系统还不允许在WIN32应用程序（工作在Ring3级）中直接调用中断功能，如INT13、INT21、INT25、INT26等。但它同时也提供了一些服务来弥补这种缺憾，在WIN95/98中，VWIN32服务就是其中一种。VWIN32服务是通过一个VXD（虚拟设备驱动）来实现的，它提供了设备IO功能，通过它，使用API函数DeviceIoControl便可以实现WIN32应用程序和磁盘设备驱动程序间的通信，从而实现对磁盘的存取。VWIN32提供的服务是一系列的控制命令字，它们实现诸如DOS操作系统下的INT13、INT25、INT26和INT21等功能调用。下面是它定义的一些控制命令字：

VWIN32_DIOC_DOS_IOCTL	(1)	实现INT21 功能
VWIN32_DIOC_DOS_INT25	(2)	实现INT25 功能
VWIN32_DIOC_DOS_INT26	(3)	实现INT26 功能
VWIN32_DIOC_DOS_INT13	(4)	实现INT13 功能
VWIN32_DIOC_DOS_DRIVEINFO	(6)	实现INT21 730x 功能

如果要对磁盘进行读写，只要使用DeviceIoControl执行相应命令即可，下面的例子用来读取磁盘的一个分区（使用vwin虚拟设备驱动）：

第一步：打开VWIN32服务，HANDLE hDev = CreateFile ("\\\\.\\VWIN32", 0, 0, 0 , 0 , FILE_FLAG_DELETE_ON_CLOSE, NULL);

第二步：填充中断所用到的相关寄存器。这里将寄存器放在一个结构中，结构定义如下（有关INT13使用的寄存器情况，请参阅相关资料）：

```
DIOC_REGISTERS reg;
DISKIO di;
reg.reg_EAX = 0x7305;  // 扩展绝对扇区读写
reg.reg_EBX = (DWORD)&di;
reg.reg_ECX = 0xffff;  // 使用DISKIO结构
reg.reg_EDX = volume+1;  // 基于1的分区盘符号，A代表1；B代表2；依次类推
reg.reg_ESI = 0;          // 读操作
reg.reg_Flags = 0;        // 设置标识符
di.diStartSector = start_sector;    //分区中的逻辑起始扇区
di.diSectors = num_sectors;         //读取的扇区数
di.diBuffer = (DWORD)pBuf;          //读取的内存块
```

第三步：调用设备IO API函数DeviceIoControl执行6号命令（即VWIN32_DIOC_DOS_DRIVEINFO），BOOL bResult = DeviceIoControl(hDev,VWIN32_DIOC_DOS_DRIVEINFO,&reg, sizeof(reg), &reg, sizeof(reg), &cb, 0);

第四步：如果DeviceIoControl函数返回值不等于零，则继续对INT13寄存器的状态标识符进行判断，否则调用失败。

```
if(reg.reg_Flags & 1) return FALSE;
  return TRUE;
```

第四步：关闭服务，CloseHandle(hDev);

三、限制或局限

上面是使用vwin32虚拟设备驱动读取逻辑磁盘扇区的完整步骤，在Windows 95/98下它是可以正确读写逻辑分区的[16]。但是实验证明，vwin32设备驱动对物理磁盘扇区读写却鞭长莫及。物理磁盘不关心该硬盘被分成几个区，在做分区恢复的时候，是需要根据绝对偏移扇区数，对磁盘进行扇区读写的，在这个时候就需要用到5.3节

中提到的THUNK技术，用32位动态链接库Disk32.Dll调用16位动态链接库Disk16.Dll，进行绝对扇区读写。

extern DLLFUNC *DllThunk32；//引用DLL中的外部函数

bReturnValue= (DllThunk32)(nDiskIndex+0x80, Cylinder, Head, Sector, FIT_IMAGE_TIMES_Abs, pchChange, startinglogicalsector + i * FIT_IMAGE_TIMES_Abs, DllCheckInt13Extension(nDiskIndex+0x80)); //绝对扇区读写。

注释：

1）nDiskIndex是硬盘编号。0：第一个磁盘；1：第二个磁盘；2...依此类推。2）FIT_IMAGE_TIMES_Abs：读写内存块大小，按照4.2.4节给出的最优大小参数，是64K。

3）pchChange内存块。

4）startinglogicalsector + i * FIT_IMAGE_TIMES_Abs：绝对偏移值。

5）DllCheckInt13Extension(nDiskIndex+0x80)函数判断是否支持int13扩展。

四、WINDOWS 2000的磁盘扇区读写

在WINNT和WIN2000中磁盘被看做一种标准设备，如上一章所说，可以使用CreateFile象打开文件一样打开并存取。CreateFile支持两种方式的磁盘设备--逻辑磁盘（格式为"\\.\C:"）和物理磁盘（格式为"\\.\PHYSICALDRIVEx"，其中x为数字），例如打开A：盘进行读取操作，只要这样：HANDLE hDev = CreateFile("\\\\.\\A:",GENERIC_READ,FILE_SHARE_WRITE,0,OPEN_EXISTING,0,0);

如果得到的句柄有效，就可以使用ReadFile来读取了，

ReadFile(hDev,Buffer,512,&dwRet,0);

读取结束要关闭该句柄，

CloseHandle(hDev)；

这比WIN95/98下的磁盘扇区读取方便多了。

另外，上面的例子是读写逻辑磁盘的，它包括软驱、硬盘分区等；物理磁盘指的是实际的硬盘，它不关心该硬盘被分成几个区，硬盘的编号是从0开始的，"\\.\PHYSICALDRIVE0"表示第一块硬盘，其它依此类推。可能马上会想起，利用这

种机制可以对硬盘的分区表进行存取了。确实如此，此时便可以对硬盘的主引导扇区（独立存在的一个扇区，包含分区表信息，不同于磁盘分区的BOOT区）进行操作了。

```
unsigned char Buffer[512]={0};
HANDLE hDev = CreateFile ("\\\\.\\PHYSICALDRIVE0", GENERIC_WRITE, FILE_SHARE_WRITE, 0, OPEN_EXISTING, 0, 0);
WriteFile(hDev,Buffer,512,&dwRet,0);
CloseHandle(hDev);
```

注意：这样将可能改写系统的引导区。

五、一个自适应的磁盘读写类

由上面的例子可以看出，不同的Windows操作系统下对磁盘扇区的读写有不同的方式，为了能够在各类操作系统下能够使用统一的方法读写磁盘扇区，所以设计了一个通用读写扇区类。该类的设计思想如下：首先编写各类操作系统下的磁盘扇区存取函数，然后通过GetVersion()函数来判断操作系统，进而选取对应的函数来实现磁盘扇区的读写。WINDOWS操作系统对INT13的支持是最差的，只能读写软驱数据，所以在这里只使用INT25、INT26、INT21--7305等中断调用来实现。类的定义如下：

```
class CDiskInfo{
public:
    CDiskInfo();
    ~CDiskInfo();
private:
    HANDLE hDev;
    DWORD dwCurrentPlatform;
    //Windows 9x下读写磁盘的函数
    BOOL ReadAbsoluteSectors32( int volume, DWORD start_sector, WORD num_sectors, void* pBuf );
    BOOL WriteAbsoluteSectors32(int volume, DWORD start_sector, WORD num_sectors, void* pBuf );
    //Windows NT下读写磁盘的函数，直接写在公共函数ReadSectors和
    //WriteSectors中
public:
```

```
//对外统一提供Read和Write操作，类内部根据平台选用适合的函数调用
BOOL WriteSectors(int drive, DWORD startinglogicalsector, int numberofsectors, char* buffer);
BOOL ReadSectors(int drive, DWORD startinglogicalsector, int numberofsectors,char* buffer);
//打开关闭句柄
BOOL OpenDeviceHandle(int drive);
BOOL CloseDeviceHandle();
//锁卷/解卷
BOOL LockLogicalVolume( int volume, int lock_level, int permissions, BOOL bFlatFAT );
BOOL UnlockLogicalVolume( int volume, BOOL bFlatFAT);
};
```

该类对外提供了两个接口，即ReadSectors和WriteSectors，其参数是一样的，分别是要读写的磁盘编号drive，要存取磁盘的开始扇区号startinglogicalsector，要读取的扇区数numberofsectors和读写扇区数据的缓冲区buffer。这里磁盘编号是从1开始的，即1代表A；2代表B；3代表C。依此类推。扇区的编号从0开始。使用时也很简单，只要作如下声明即可：

```
BYTE Buffer[1024];
CDiskInfo A;
BOOL bRet=A.ReadSectors(1, 0, 2, Buffer);
```

ReadSectors和WriteSectors函数除了能象上文介绍的那样能够读写逻辑分区扇区，也可以对磁盘的绝对扇区进行读写。只要传入的句柄不再是指向逻辑分区，而是直接指向一个磁盘的就可以了。相对应的，startinglogicalsector参数值就是磁盘绝对扇区的偏移值，drive参数值就显得没有意义了。

六、补充说明

严格来说，在对磁盘进行读写时，应该遵循以下顺序：打开设备句柄（WIN95/98下为VWIN32服务，WIN2000下为磁盘设备）、锁卷、验证卷的有效性、读/写、开锁

卷、关闭设备句柄。这里为了描述上的简洁，忽略了锁卷/开锁卷及验证有效性等操作。

如果要锁卷，只要在打开句柄后，在读写磁盘分区前进行就可以了。在Windows 2000系统使用DeviceIoControl函数的控制码FSCTL_LOCK_VOLUME 对分区进行锁卷，锁卷后，系统将不能对此分区进行读写，保证了备份分区数据的有效性和完整性。在备份完毕后，还需要使用FSCTL_UNLOCK_VOLUME控制码来进行解卷，从而使系统可以继续读此分区进行读写，具体代码如下：

```
bReturnValue= ::DeviceIoControl(hVWin32Device,FSCTL_LOCK_VOLUME,NULL, 0,NULL, 0, &dwOutBytes,(LPOVERLAPPED)NULL);//锁卷
bReturnValue= ::DeviceIoControl(hVWin32Device,FSCTL_UNLOCK_VOLUME,NULL, 0,NULL, 0, &dwOutBytes,(LPOVERLAPPED)NULL);//解卷
```

而在Windows 9x系统下解卷和锁卷都需要通过设置寄存器进行：

1）Windows 9x系统下的锁卷：

```
reg.reg_EAX = 0x440d; // 普通的逻辑控制符
reg.reg_ECX = bFlatFAT ? 0x484a : 0x084a; // 锁逻辑卷
reg.reg_EBX = (volume+1) | (lock_level << 8);
reg.reg_EDX = permissions;
reg.reg_Flags = 1; // 重置标识符
bResult = DeviceIoControl( hVWin32Device, VWIN32_DIOC_DOS_IOCTL,
                &reg, sizeof( reg ), &reg, sizeof( reg ), &cb, 0 );
```

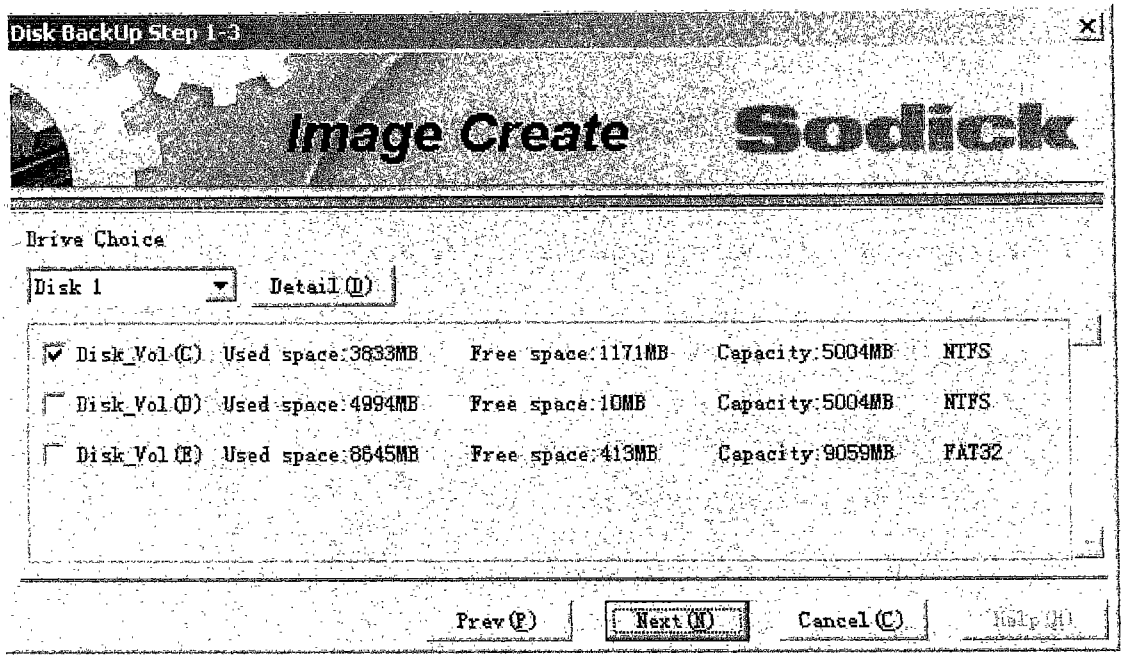
2）Windows 9x系统下的解卷：

```
reg.reg_EAX = 0x440d;
reg.reg_ECX = bFlatFAT ? 0x486a : 0x086a; // 解逻辑卷
reg.reg_EBX = volume+1;
reg.reg_Flags = 1; //重置标识符
bResult = DeviceIoControl( hVWin32Device, VWIN32_DIOC_DOS_IOCTL,
                &reg, sizeof( reg ), &reg, sizeof( reg ), &cb, 0 );
```

第八章　分区盘符确定

在对分区进行备份的时候，都需要指定分区的盘符号，从而能够准确进行分区数据备份。实际代码编写过程中，常常会发现分区与逻辑盘符无法对应的问题。尤其是在Windows 98下，经常发生盘符交错的情况。在Windows 2000中，拥有了一套完整的API程序，可以比较方便地确定分区盘符。可是在Windows 98中，却没有这整套API，需要使用另外的方法，对盘符进行确定。下面将详细指明在Windows 2000和Windows 98下的盘符分配方法。

图8-1　程序对话框显示了分区状况



一、Windows 2000操作系统：

Windows 2000系统对盘符在一般情况下都会依照磁盘先后顺序、分区顺序自动分配盘符。譬如：有C、D、E、F、G、H六个分区，二个硬盘。每个硬盘各自拥有三个分区，则很自然，C、D、E就属于第一个硬盘；剩下的就属于第二个硬盘。但是，比较灵活的是，操作系统可以根据用户需求重新设定分区的盘符号。根据这一点，就不

Disk Backup Step 1-3

Image Creator

Drive Choice

Disk 1 | Default (D)

Disk_Vol (C:)	Used space:3833MB	Free space:11717MB	Capacity:20048MB	NTFS
Disk_Vol (D:)	Used space:4884MB	Free space:1040MB	Capacity:20048MB	NTFS
Disk_Vol (E:)	Used space:8842MB	Free space:4713MB	Capacity:80208MB	FAT32

Prev (P) | Next (N) | Cancel (C) | Help (H)

能套用简单盘符分配顺序这个原则，获取磁盘的分区盘符。幸运的是，Windows 2000 提供了很多方便的API函数，能够比较方便的进行盘符号确定。

```
char achPartitionName[7]="\\\\.\\P:";
achPartitionName[4] = *DriveName;
hDisk = ::CreateFile(achPartitionName,
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL);
STORAGE_DEVICE_NUMBER stStorage;
bReturnValue = ::DeviceIoControl(hDisk,
    IOCTL_STORAGE_GET_DEVICE_NUMBER,
    NULL, 0,
    &stStorage, sizeof(STORAGE_DEVICE_NUMBER),
    &dwOutBytes,
    (LPOVERLAPPED)NULL);
PARTITION_INFORMATION stPartition;
bReturnValue = ::DeviceIoControl(hDisk,
    IOCTL_DISK_GET_PARTITION_INFO,
    NULL, 0,
    &stPartition, 100,
    &dwOutBytes,
    (LPOVERLAPPED)NULL);
```

通过代入盘符号，获得此分区盘符的句柄，再通过把这个句柄作为参数传入 DeviceIoControl 函数，结合 IOCTL_STORAGE_GET_DEVICE_NUMBER 和 IOCTL_DISK_GET_PARTITION_INFO 控制码，得到此分区所位于的磁盘于 stStorage.DeviceNumber和具体偏移值stPartition.StartingOffset.QuadPart，然后再按磁

盘号各自分组依偏移值排序（注意：不是按盘符字符顺序），最后就可以获得每个磁盘所拥有分区顺序。

二、**Windows 98操作系统：**

Windows 98操作系统的盘符分配，又和Windows 2000略有不同。它的分配顺序是先按照IDE（不包括活动硬盘）顺序，分配各IDE磁盘活动主分区的盘符，然后再对各IDE硬盘的其他Windows 98可识别分区进行盘符分配，其中NTFS文件系统不被支持，也就没有被分配盘符号。譬如说，存在两个硬盘，各具有一个活动主分区和二个逻辑分区，共存在六个分区C、D、E、F、G、H。则第一磁盘的分区号为C、E、F，第二个磁盘的分区号为D、G、H。因为C和D被分别分配给两个磁盘的主活动分区，所以就产生了Windows 98下的盘符交错问题，由于这个问题的存在，使实际程序在开始编制初始，Windows 98版本下常常会产生不正常的盘符分区对应。在掌握了以上的一般性规则，程序就相应的按照规则做了修改，可是不久又发现，以上规则并不是绝对的。一，它对活动硬盘不适合；二、如果在BIOS检测这一部中，屏蔽了第二个硬盘的检测，则分区盘符会顺序分配；三、如果通过FDISK程序，把第二个硬盘重新分区，则分区盘符会顺序分配。作为一个商用软件，必须面对这种不确定性，于是由一般性规则产生的分配原则必须再次修改。实际操作中，在Windows 98系统的注册表HKEY_LOCAL_MACHINE下存在ENUM目录[17]，这个目录下，ESDI和SCSI字键下都会有IDE硬盘和活动硬盘的一些信息，包括此硬盘中拥有的分区盘符，可是仔细一看字键值中有些盘符却有可能是重复的，这是因为Windows 98并不会主动去更新硬盘上的盘符信息，即当此硬盘被卸载后，如果因为有新的硬盘加挂，导致分区盘符被修改的话，源硬盘的信息，包括分区盘符被依然保留。从此可以看出，这里唯一可以相信的磁盘盘符记录，就是包括C盘符的磁盘信息，因为C盘总是被包括在Windows 98的当前可启动且已经启动的硬盘，所以此条记录中，包括C盘符，都是第一个磁盘的盘符。然后，把所有从GetLogicalDriveStrings()函数获得的（去除A、B盘符，光驱盘符和一些Windows 98错误生成的盘符[此类盘符是系统错误生成，里面空无一个文件，大小为0]）的已用盘符集，删去注册表中包含C盘符记录的盘符之集后，再按照原来的规则，对剩余的磁盘进行盘符顺序分配。譬如说，有三个硬盘，第一个硬盘有三个可识别分区；第二个磁盘有两个可识别分区；第三个磁盘也有两个可识别分区，那么总的可供分配的盘符集是C、D、E、F、G、H、I。注册表中包含C盘符的记录是C、

E、F，那么去除C、E、F后，还剩余D、G、H、I。第二个磁盘的盘符就是D、G。相应的，第三个磁盘的盘符就是H、I。具体代码省略。总结分配方法，就是根据注册表，先获得第一个硬盘的分区，然后根据分配规则，在余下的子集中顺序分配获得盘符。

三、总结

通过前面针对两个不同系统确认盘符号的方法，可以正确的获得磁盘相对应的盘符，不管它顺序，或是系统盘符分配的各种法则。在有了系统正确的盘符号后，才可以在进行磁盘备份时，输入正确的需要备份的盘符号，同样的情况也发生在恢复的时候。

第九章　引导扇区

一、系统引导

在顺利的完成分区数据文件的恢复后，如果没有考虑到同步恢复每个分区表前面的引导扇区，将可能使原先可以启动的分区无法启动。

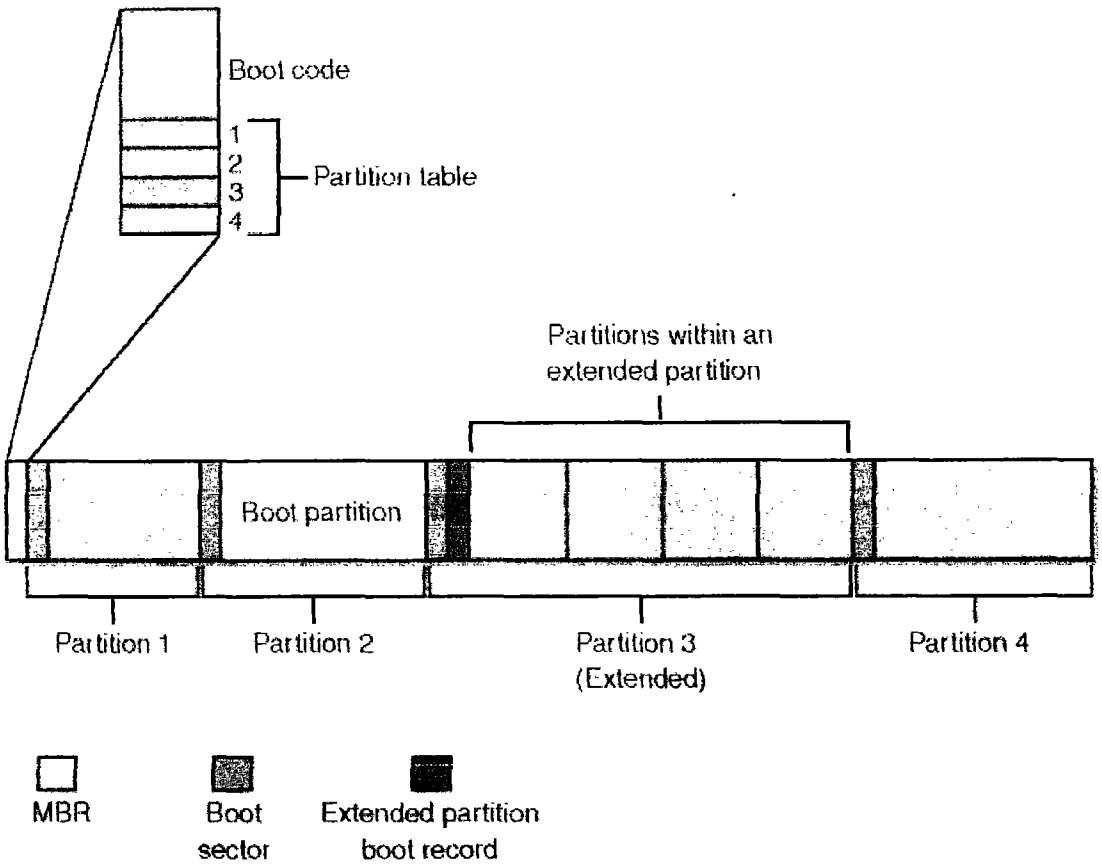
下面将以Windows 2000为例详细阐述系统引导问题。具体将从Windows 2000的安装和Ntldr和Ntdect的执行开始。下表列出了所有引导过程中使用到的组件、及它们的执行模式和作用。

表9-1 引导过程组件对象

组件	处理器执行模式	功能
Master boot record (MBR) code	16位实模式	读入、装载分区引导扇区
Boot sector	16位实模式	读根目录，并装载Ntldr程序
Ntldr	16位实模实和32位保护模实;打开分页管理	读入 Boot.ini, 提供 boot menu, 还装载 Ntoskrnl.exe, Bootvid.dll, Hal.dll, 和 boot-start 设备驱动
Ntoskrnl.exe	分页管理下的32位保护模式	初始化可执行的子系统，并且引导和启动系统的设备驱动，为系统运行系统内应用和运行Smss.exe做准备
Smss	32位系统内部应用	装载 Win32 子系统，包括 Win32k.sys 和 Csrss.exe, 并且启动Winlogon 进程
Winlogon	32位系统内部应用	启动服务控制管理器 (SCM), 本地安全子系统(Lsass), and 提供交互可登录的对话框
Service control manager (SCM)	32位系统内部应用	装载、初始化自动启动的设备驱动和Win32服务。

二、准备引导

图9-1　一个磁盘分区布局



当一个机器引导的时候，首先将从BIOS中运行引导代码。BIOS把MBR读入内存，并把控制交还给MBR。MBR代码搜索主分区表，直到它定位在一个可引导分区。当MBR发现至少一个可引导分区时，将会把此分区第一个扇区读入内存，并把控制转给此分区的代码。这种分区就称之为引导分区。这种分区的第一个扇区就称之为引导扇区。

引导扇区将会根据格式不同而不同。例如，如果引导扇区是一个FAT分区，Windows 2000将在引导扇区写一个可识别FAT分区的代码。同理，如果是ntfs，也将会有个可识别NTFS的引导扇区代码。所以，引导扇区中只读代码的作用就是告诉Windows 2000有关逻辑盘符的结构和形式，而且去逻辑硬盘的根目录下去读取Ntldr文件。当引导扇区把Ntldr文件导入内存后，引导扇区将把控制权返回给Ntldr文件的

入口。如果引导扇区不能在磁盘根目录下找到Ntldr的话，他将会显示错误信息“BOOT:Couldn't find NTLDR”（如果引导文件系统是FAT）或者是“NTLDR is missing”（如果文件系统是NTFS）。

Ntldr启动于系统执行x86实模式的时候。在实模式下，没有发生虚拟到物理内存转换，这意味着程序使用的全是物理地址，但只能访问此机器的最先的1兆物理地址。简单的MS-DOS程序在实模式下执行。然而，Ntldr接手后的第一件事，就是切换到保护模式下。仍旧没有虚拟到物理间的内存转换在此引导过程中发生，但是可以访问一个32位的内存了，也就是说Ntldr可以访问所有的物理内存。在产生了足够的分页表，并Ntldr启用了分页管理机制，使得能够把线性地址转换成物理地址，使低16兆空间能够被访问。保护模式下的分页管理就是Windows 2000系统下在常规操作下运行的模式。

当Ntldr使能了分页管理机制，就可以完全运行启动代码了。然而，它就像要显示引导区一样，仍旧还需要依赖引导代码中的函数来访问基于IDE的系统和引导盘。引导代码函数简短地关闭了分页管理，把处理器切回BIOS提供的，能被执行的模式。Ntldr然后读取Boot.ini文件。像引导扇区代码，Ntldr也包含了只读的NTFS和FAT代码；但也不同于引导扇区代码，Ntldr文件系统代码能够读取子目录。如果Boot.ini显示了有多个可供引导的系统存在，屏幕上将给出操作系统菜单。如果只有一个系统，就不会显示菜单。接着系统就可以逐个装载硬件驱动和系统服务等等，直到系统启动完毕。

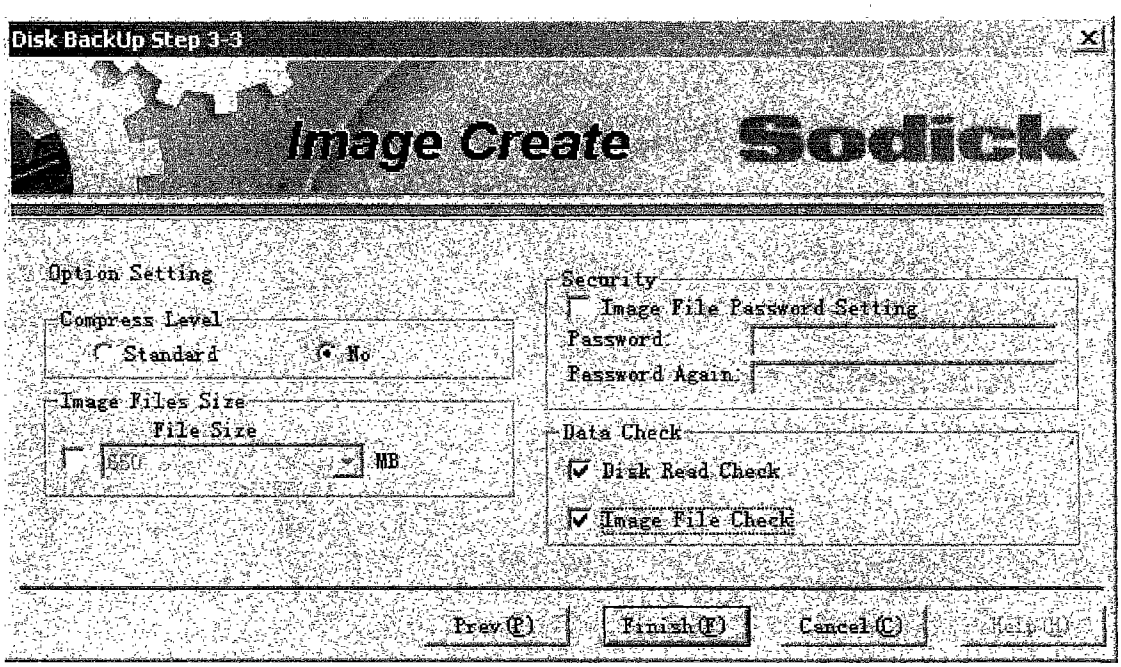
纵观整个系统启动的过程，可以发现从引导扇区到Ntldr是最可能导致系统不能正常启动的原因。在代码编写过程中，常常可能会发现，原先可以引导的分区在恢复到一个新的分区后，常常无法正常启动。可能就会报告象上文中列示的警告词“Ntldr is missing”、“BOOT:Couldn't find NTLDR”一样，启动不了系统，根据上文中对Windows 2000的研究和分析，可以比较清楚的知道，这主要是由于目的引导扇区中的引导程序没有根据源分区的引导情况正确改写，从而导致系统不能正常启动。针对这种可能产生问题的情况，修改了恢复分区数据的代码。除了保留正常恢复文件数据的代码外，还增添了修改引导扇区的代码，使源分区引导扇区中引导信息被正确写入目的扇区中。使可引导分区在恢复后不仅保证了数据的完整性，也可以像原样，正常地进行引导启动。

说到引导扇区中的引导代码，那谈谈它们所处的位置。在FAT16文件系统中，它位于引导扇区中偏移62位以后，前面的都为BPB部分；与FAT16文件系统相似的是，FAT32文件系统的引导代码位于引导扇区偏移90位以后，同样的前面的部分也为BPB部分，相关解释说明可以参看前面章节关于BPB部分的解释。了解了引导代码的位置，那么当开始分区复原的时候，就可以知道该恢复引导扇区中的哪些字节，同时保留BPB中的数据。

第十章　压缩算法LZO和CRC32校验算法

本程序在进行程序备份和恢复的时候，还可以根据用户实际需要，对生成的文件进行压缩处理和CRC数据校验。在实际的测试中，压缩处理对分区文件数据一般可以达到50%左右，而使用了CRC32数据校验处理后保证了数据的完整无错性，但同时也不可避免的用更多的时间作为了交换条件。

图10-1　实际压缩、校验对话框效果图



Disk Backup Step 3-3
Image Create
Quick
Options Setting
Compress Level
Standard
No
Image File Size
MB
Data Check
Disk Read Check
Image File Check
Security
Image File Password Setting
Password:
Password Again:
Next(N)
Finish(F)
Cancel(C)
Help(H)

10．1 压缩算法LZO及其性能

当对磁盘分区进行备份的时候，如果当前目的保存分区大小不够的话，那么就需要使用压缩算法，把目标生成文件大小尽可能最小化，减少其在磁盘上的占用空间。根据诸多的调查分析，本程序选用了压缩性能卓越的lzo算法。

Lzo是一种实时的数据压缩库，它是一种微小无损失的数据压缩库。它能提供相当好的快速压缩和解压缩。解压缩不需要任何内存。此外，它也可以用比较慢的速度来达到一个很有竞争力的压缩比率，而且在解压缩时，仍旧保持一个很快的速度。

Lzo是“Lempel-Ziv-Oberhumer”的缩写。用ANSI C写成。源代码和被压缩的数据形式被设计成可轻松跨平台。[19]

整个Lzo包包含了诸多算法，具有如下的特征：

- 解压缩简单快速。
- 解压缩不需要内存。
- 压缩也很快。
- 压缩只需要64KB的内存
- 允许你使用更大的压缩比率，但这会耗费更多的时间。解压缩的时间不会减少。
- 每次压缩时都会达到尽可能大的压缩比率。
- 存在一种压缩率，只需要8kb的内存来进行压缩。
- 算法在线程级安全。
- 算法达到数据无损失。

表10-1 lzo各算法的性能比较：

Algorithm	Length	CxB	ComLen	%Remn	Bits	Com K/s	Dec K/s
LZO1-1	224401	1	117362	53.1	4.25	4665.24	13341.98
LZO1-99	224401	1	101560	46.7	3.73	1373.29	13823.40
LZO1A-1	224401	1	115174	51.7	4.14	4937.83	14410.35
LZO1A-99	224401	1	99958	45.5	3.64	1362.72	14734.17

LZ01B-1	224401	1	109590	49.6	3.97	4565.53	15438.34
LZ01B-2	224401	1	106235	48.4	3.88	4297.33	15492.79
LZ01B-3	224401	1	104395	47.8	3.83	4018.21	15373.52
LZ01B-4	224401	1	104828	47.4	3.79	3024.48	15100.11
LZ01B-5	224401	1	102724	46.7	3.73	2827.82	15427.62
LZ01B-6	224401	1	101210	46.0	3.68	2615.96	15325.68
LZ01B-7	224401	1	101388	46.0	3.68	2430.89	15361.47
LZ01B-8	224401	1	99453	45.2	3.62	2183.87	15402.77
LZ01B-9	224401	1	99118	45.0	3.60	1677.06	15069.60
LZ01B-99	224401	1	95399	43.6	3.48	1286.87	15656.11
LZ01B-999	224401	1	83934	39.1	3.13	232.40	16445.05
LZ01C-1	224401	1	111735	50.4	4.03	4883.08	15570.91
LZ01C-2	224401	1	108652	49.3	3.94	4424.24	15733.14
LZ01C-3	224401	1	106810	48.7	3.89	4127.65	15645.69
LZ01C-4	224401	1	105717	47.7	3.82	3007.92	15346.44
LZ01C-5	224401	1	103605	47.0	3.76	2829.15	15153.88
LZ01C-6	224401	1	102585	46.5	3.72	2631.37	15257.58
LZ01C-7	224401	1	101937	46.2	3.70	2378.57	15492.49
LZ01C-8	224401	1	100779	45.6	3.65	2171.93	15386.07
LZ01C-9	224401	1	100255	45.4	3.63	1691.44	15194.68
LZ01C-99	224401	1	97252	44.1	3.53	1462.88	15341.37
LZ01C-999	224401	1	87740	40.2	3.21	306.44	16411.94
LZ01F-1	224401	1	113412	50.8	4.07	4755.97	16074.12
LZ01F-999	224401	1	89599	40.3	3.23	280.68	16553.90
LZ01X-1(11)	224401	1	118810	52.6	4.21	4544.42	15879.04
LZ01X-1(12)	224401	1	113675	50.6	4.05	4411.15	15721.59
LZ01X-1	224401	1	109323	49.4	3.95	4991.76	15584.89
LZ01X-1(15)	224401	1	108500	49.1	3.93	5077.50	15744.56
LZ01X-999	224401	1	82854	38.0	3.04	135.77	16548.48
LZ01Y-1	224401	1	110820	49.8	3.98	4952.52	15638.82
LZ01Y-999	224401	1	83614	38.2	3.05	135.07	16385.40
LZ01Z-999	224401	1	83034	38.0	3.04	133.31	10553.74
LZ02A-999	224401	1	87880	40.0	3.20	301.21	8115.75
memcpy()	224401	1	224401	100.0	8.00	60956.83	59124.58

注释：

- CxB是块的数量
- K/s 是每秒1000个未被压缩字节被压缩的速度
- 使用汇编后，解压缩将会更快。

鉴于以上的事实，在磁盘备份恢复程序中，应用程序便使用了lzo压缩库对磁盘分区数据按照用户的需要进行压缩和解压缩操作。实验测试下来，对一个128K大小的内存块进行Lzo压缩处理，压缩比率最高可以达到50%。实测速度也和不压缩时，相差不多。

可是，在某些特殊情况下，特别是对某些已经被压缩过的文件，譬如说jpg文件、rar文件、zip文件等等，进行Lzo压缩处理时，压缩后产生的数据块不但没有变小，反而会稍大于被处理的数据块。针对这种情况，代码中将仍旧按照原大小进行数据块存储。

10．2 CRC32校验算法

数据备份的目的就是要把磁盘分区数据及时可靠地恢复到目的分区磁盘，因此要求一个程序必须可靠与快速进行备份恢复操作，在进行备份恢复操作中可靠与快速往往是一对矛盾。为了解决可靠性，本程序采用了差错控制。本节将概要介绍循环冗余校验CRC（Cyclic Redundancy Check）的差错控制原理及其算法实现。

一、循环冗余校验码（**CRC**）

CRC校验采用多项式编码方法。被处理的数据块可以看作是一个n阶的二进制多项式，由$a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\cdots+a_1x+a_0$。如一个8位二进制数10110101可以表示为：$1x^7+0x^6+1x^5+1x^4+0x^3+1x^2+0x+1$。多项式乘除法运算过程与普通代数多项式的乘除法相同。多项式的加减法运算以2为模，加减时不进，错位，和逻辑异或运算一致。[20]

采用CRC校验时，发送方和接收方用同一个生成多项式g（x）,并且g（x）的首位和最后一位的系数必须为1。CRC的处理方法是：发送方以g（x）去除t（x），得到余数作为CRC校验码。校验时，以计算的校正结果是否为0为据，判断数据帧是否出错。

CRC校验可以100％地检测出所有奇数个随机错误和长度小于等于k（k为g（x）的阶数）的突发错误。所以CRC的生成多项式的阶数越高，那么误判的概率就越小。采用CRC-4方案，使用的CRC校验码生成多项式g（x）= x^4+x+1。采用16位CRC校验，可以保证在10^{14} bit码元中只含有一位未被检测出的错误，其生成多项式g（x）= $x^{16}+x^{15}+x^2+1$。CRC-32的生成多项式g（x）= $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$。CRC-32出错的概率比CRC-16低$10^{-5}$倍。由于CRC-32的可靠性，把CRC-32用于重要数据校验变得十分合适，所以在通信、计算机等领域运用十分广泛。因此在本程序中，在对磁盘读写和文件数据读写进行校验的时候，也采用CRC-32进行差错控制。

二、CRC校验码的算法分析

CRC校验码的编码方法是用待发送的二进制数据t（x）除以生成多项式g（x），将最后的余数作为CRC校验码。其实现步骤如下：

设待发送的数据块是m位的二进制多项式t（x），生成多项式为r阶的g（x）。在数据块的末尾添加r个0，数据块的长度增加到m+r位，对应的二进制多项式为$x^r t(x)$。

用生成多项式g（x）去除$x^r t(x)$，求得余数为阶数为r-1的二进制多项式y（x）。此二进制多项式y（x）就是t（x）经过生成多项式g（x）编码的CRC校验码。

用$x^r t(x)$以模2的方式减去y（x），得到二进制多项式$x^r t'(x)$。$x^r t'(x)$就是包含了CRC校验码的待发送字符串。

从CRC的编码规则可以看出，CRC编码实际上是将代发送的m位二进制多项式t（x）转换成了可以被g（x）除尽的m+r位二进制多项式$x^r t'(x)$，所以解码时可以用接受到的数据去除g（x），如果余数位零，则表示传输过程没有错误；如果余数不为零，则在传输过程中肯定存在错误。许多CRC的硬件解码电路就是按这种方式进行检错的。同时$x^r t'(x)$可以看做是由t（x）和CRC校验码的组合，所以解码时将接收到的二进制数据去掉尾部的r位数据，得到的就是原始数据。

为了更清楚的了解CRC校验码的编码过程，下面用一个简单的例子来说明CRC校验码的编码过程。由于CRC-32、CRC-16、CCITT和CRC-4的编码过程基本一致，只有位数和生成多项式不一样。为了叙述简单，用一个CRC-4编码的例子来说明CRC的编码过程。

设待发送的数据t（x）为12位的二进制数据100100011100；CRC-4的生成多项式为g（x）$=x^4+x+1$，阶数r为4，即10011。首先在t（x）的末尾添加4个0构成$x^4 t(x)$，数据块就成了1001000111000000。然后用g（x）去除$x^4 t(x)$，不用管商是多少，只需要求得余数y（x）。下表为给出了除法过程。

表10-2 CRC除法过程

除数次数	被除数/ g（x）/ 结果	余数
0	1 001000111000000	100111000000
	1 0011	
	0 001001111000000	
1	1 00111000000	1000000
	1 0011	
	0 00001000000	
2	1 000000	1100
	1 0011	
	0 001100	

从上面表中可以看出，CRC编码实际上是一个循环移位的模2运算。对CRC-4，假设有一个5 bits的寄存器，通过反复的移位和进行CRC的除法，那么最终该寄存器中的值去掉最高一位就是所要求的余数。所以可以将上述步骤用下面的流程描述：

//reg是一个5 bits的寄存器

把reg中的值置0.

把原始的数据后添加r个0.

While (数据未处理完)

Begin

If (reg首位是1)

reg = reg XOR 0011.

把reg中的值左移一位，读入一个新的数据并置于register的0 bit的位置。

End

reg的后四位就是所要求的余数。

这种算法简单，容易实现，对任意长度生成多项式的G（x）都适用。在发送的数据不长的情况下可以使用。但是如果发送的数据块很长的话，这种方法就不太适合了。它一次只能处理一位数据，效率太低。为了提高处理效率，可以一次处理4位、8位、16位、32位。由于处理器的结构基本上都支持8位数据的处理，所以一次处理8位比较合适。

为了对优化后的算法有一种直观的了解，先将上面的算法换个角度理解一下。在上面例子中，可以将编码过程看作如下过程：

由于最后只需要余数，所以我们只看后四位。构造一个四位的寄存器reg，初值为0，数据依次移入reg0（reg的0位），同时reg3的数据移出reg。有上面的算法可以知道，只有当移出的数据为1时，reg才和g（x）进行XOR运算；移出的数据为0时，reg不与g（x）进行XOR运算，相当于和0000进行XOR运算。就是说，reg和什么样的数据进行XOR移出的数据决定。由于只有一个bit，所以有2^1种选择。上述算法可以描述如下，

//reg是一个4 bits的寄存器

初始化t[]={0011,0000}

把reg中的值置0.

把原始的数据后添加r个0.

While (数据未处理完)

Begin

把reg中的值左移一位，读入一个新的数据并置于register的0 bit的位置。

reg = reg XOR t[移出的位]

End

上面算法是以bit为单位进行处理的，可以将上述算法扩展到8位，即以Byte为单位进行处理，即CRC-32。构造一个四个Byte的寄存器reg，初值为0x00000000，数据依次移入reg0（reg的0字节，以下类似），同时reg3的数据移出reg。用上面的算法类

推可知，移出的数据字节决定reg和什么样的数据进行XOR。由于有8个bit，所以有2^8种选择。上述算法可以描述如下：

//reg是一个4 Byte的寄存器

初始化t[]＝{...}//共有2^8＝256项

把reg中的值置0.

把原始的数据后添加r/8个0字节.

While (数据未处理完)

Begin

把reg中的值左移一个字节，读入一个新的字节并置于reg的第0个byte的位置。

reg = reg XOR t[移出的字节]

End

算法的依据和多项式除法性质有关。如果一个m位的多项式t（x）除以一个r阶的生成多项式g（x），$t(x)=a_{m-1}x^{m-1}+a_{m-2}x^{m-2}+\cdots+a_2x^2+a_1x^1+a_0$，将每一位$a_kx^k$（0=<k<m）提出来，在后面不足r个0后，单独去除g（x），得到的余式位$y_k(x)$。则将$y_{m-1}(x)\oplus y_{m-2}(x)\oplus\cdots\oplus y_0(x)$后得到的就是t（x）由生成多项式g（x）得到的余式。对于CRC-32，可以将每个字节在后面补上32个0后与生成多项式进行运算，得到余式和此字节唯一对应，这个余式就是上面算法种t[]中的值，由于一个字节有8位，所以t[]共有2^8＝256项。这种算法每次处理一个字节，通过查表法进行运算，大大提高了处理速度，故为大多数应用所采用。

二、**CRC-32的程序实现：**

为了提高编码效率，在实际运用中大多采用查表法来完成CRC-32校验，下面是产生CRC-32校验码的子程序。

DWORD s_arrdwCrc32Table[256]={

0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f, 0xe963a535, 0x9e6495a3,0x0edb8832,..., 0x5a05df1b, 0x2d02ef8d

};//事先计算出的参数表，共有256项，未全部列出。

// Read 4K of data at a time (used in the C++ streams, Win32 I/O, and assembly functions)

```
#define MAX_BUFFER_SIZE    4096
// Map a "view" size of 10MB (used in the filemap function)
#define MAX_VIEW_SIZE  10485760
void CalcCrc32(BYTE byte, DWORD *dwCrc32)
{
    *dwCrc32 = ((*dwCrc32) >> 8) ^ s_arrdwCrc32Table[(byte) ^ ((*dwCrc32) &
0x000000FF)];
}
DWORD CRC32Buffer( BYTE *buffer, int buffer_size, DWORD *dwCrc32 )
{
    int        nLoop;
    DWORD     dwErrorCode = NO_ERROR;
    *dwCrc32 = 0xFFFFFFFF;//设初值
    for( nLoop = 0; nLoop < buffer_size; nLoop++ )
    {
        CalcCrc32(buffer[nLoop], dwCrc32);
    }
    *dwCrc32 = ~(*dwCrc32);
    return dwErrorCode;
}
```

把代码生成Lib文件，加入整个备份恢复软件工程。当用户选中CRC校验选项。则当程序在备份的时候，会把数据从磁盘扇区读出写入文件后，再把刚写入文件的数据读出和磁盘扇区数据进行CRC校验；在恢复的时候，则先把数据从文件中读出写入磁盘扇区后，再把刚写入的扇区取出和文件数据进行CRC32校验，保证数据的完整性。当读写数据出错的时候，将再次进行读写数据进行CRC32数据，当连续五次读写失败后，将强制终止备份或是恢复操作。

第十一章 总结与展望

本论文是根据实际在公司中参与的项目为背景，以著名磁盘备份恢复软件ghost为原型，检索资料撰写完成的。实际完成的软件可以运行在Windows9x以上操作系统版本中，对所有Windows可识别支持的FAT、NTFS文件系统进行磁盘备份和恢复操作。可以按照具体要求进行数据压缩和CRC校验处理，实现映象作成文件的最小化和数据拷贝的高可靠性。也可以按照用户的要求，对做成的文件进行切割，生成相对较小的被切割映象文件，进行刻录光盘等等相关数据保存操作。而且能够在同FAT文件系统类型，同一个簇拥有同等数目的扇区数的情况下，按照FAT表信息，判断是否是已用簇，来进行簇拷贝，在满足条件的情况下，实现分区大的源分区恢复到相对小的目的分区中。但由于时间和可使用资料比较有限的客观事实，当前程序在处理不同文件类型之间恢复的情况时，还是使用了灵活性相对较差的方法，实现了和源分区扇区大小长度一比一的恢复方法，FAT文件类型是根据备份时候收集的分区结构信息（主要是FAT表中对簇是否已经使用的信息），再恢复到目的分区。NTFS文件系统则是根据了其文件目录组织结构，采用了B+树先序搜索方法，遍历文件记录信息，提取有用文件信息，对NTFS文件系统的数据文件进行了备份。然后根据备份时的数据记录偏移值，实现了分区的正常恢复。当源、目的分区大小不一致的情况下，在需要的情况下，实现了对主引导区中的分区表和扩展分区中的分区表的动态重新改写。并能够保证使原来可以启动的分区在备份恢复后，仍旧可以启动引导。今后的工作将会是，实现FAT16和FAT32互为逆向的备份和恢复，在最大限度下不受簇和扇区对应数值的影响，使分区恢复备份更加灵活，不用去修改分区表的数据结构。

参考文献

[1] 徐文军 麻信洛，硬盘应用与维护，2002-1-1，机械工业出版社

[2] http://www.sunattic.com/recover/findouthd/article4.htm

[3] http://newdos.yginfo.net/dosart12.htm

[4] David a. Solomon & Mark E. Russimovivh, Inside Microsoft Windows 2000, 1999, Microsoft Press, P689

[5] http://www.pcworld.com.cn/2000/back_issues/2016/1633.asp

[6] http://www.fanqiang.com/a1/b5/20010508/144234.html

[7] 印涛 秦剑，深入浅出硬盘分区表，VC知识库在线杂志第十五期

[8] Microsoft Corporation Microsoft Extensible Firmware Initiative FAT32 File System Specification Hardware White Paper Version 1.03, December 6, 2000, Microsoft Press, P9-P19

[9] 尤晋元 史美林，Windows 操作系统原理，2003年9月第六次印刷，机械工业出版社，P260

[10] http://courses.cs.vt.edu/~cs2604/SummerI_2003/Notes/C13.TreeIndexing.pdf

[11] http://linux-ntfs.sf.net/ntfsdoc

[12] http://web.sias.net.cn/ec/dzuwu/1/ciyemian/di4/kecheng3.htm

[13] 杨季文 等编著，汇编语言程序设计教程，2002年12月第九次印刷，清华大学出版社

[14] http://www.jjhou.com/review1-16.htm

[15] Andrew Schulman Bona Fide 32-bit Programs that Run on Windows 3.1 Using Win32s, 1993/04

[16] Walter Oney著 侯捷译，Windows 95 系统程序设计－虚拟机器与VxD程序设计，1997年

[17] 苏国彬 唐义胜，精通BIOS与Windows注册表（第2版），2003年5月，机械工业出版社

[18] Charles Petzold, Windows程序设计（上、下），1999年11月，北京大学出版社

[19] http://www.oberhumer.com/opensource/lzo/

[20] 刘东，循环冗余校验CRC的算法分析和程序实现，西南交通大学计算机与通信工程学院

图索引

表索引

致谢

本文是我硕士阶段研究工作的小结。本文的完成是和诸多师长、同学、朋友和亲人的指导、关心和帮助分不开的。

首先要感谢的是我的导师，邢传鼎老师。在学校学习二年多的时间里，他不仅对我的研究工作指导颇多，还教给我治学的态度和研究的方法。在此对他表示特别的感谢。

忻秀珍老师对我的研究工作有很多有益的指导和建议。感谢她对我的学习和研究的帮助。

特别感谢上海市易仁信息技术有限公司的孙军副总经理。在本文中提到的有关98下调用中断和THUNK技术的大部分研究实现工作，是我在易仁信息技术有限公司中，在与他讨论、切磋中完成的。在此对他所给予的热情帮助和真诚合作表示由衷的感谢。

感谢所有教导过我的老师们。

感谢所有帮助过我的同学们。

最后，感谢我的父母。如果没有他们在我硕士学习期间对我的支持和关爱，我硕士论文的顺利完成是难以想象的。

2003年12月26日星期三

于东华大学实验室