

合肥工业大学

操作系统课程设计报告

设计题目	31 题
学生姓名	谷纪豪
学 号	2020214189
专业班级	计算机科学与技术 20-3 班
指导教师	田卫东、李琳
完成日期	2023 年 6 月

1 课程设计任务、要求、目的

1.1 课程设计任务

本次课程设计的任务是建立基于磁盘存储设备的 FAT 文件系统。任务要求包括分配磁盘存储空间作为文件存储空间，建立相应的文件系统，解决文件的重名、共享和安全控制问题，支持文件的按名存取，并设计适当的数据结构来管理目录、磁盘空闲空间和已分配空间。此外，还需要实现文件的创建、删除、移位、改名等功能，并提供良好的界面来显示磁盘文件系统的状态和空间的使用情况。最后，还需要实现虚拟磁盘转储功能，以实现磁盘与内存之间的信息存储和读取。

1.2 课程设计目的和要求

本次课程设计旨在帮助学生深入理解磁盘文件系统的原理和设计，掌握文件系统的相关操作和数据结构。通过实际操纵磁盘存储空间和建立文件系统，学生可以加深对文件存储和管理的认识，并通过解决文件重名、共享和安全控制等问题，提升对文件系统的设计能力。此外，学生将学习如何设计合适的数据结构来管理目录、磁盘空闲空间和已分配空间，从而提高对数据结构的应用能力。

磁盘文件系统：建立基于磁盘存储设备的 FAT 文件系统。

- 首先分配一定容量的磁盘存储空间，作为文件存储空间；
- 建立相应的文件系统，使用 FAT 文件系统；
- 解决文件的重名、共享和安全控制；
- 支持文件的“按名存取”；
- 为该文件系统设计相应的数据结构来管理目录、磁盘空闲空间、已分配空间等。
- 提供文件的创建、删除、移位、改名等功能。
- 提供良好的界面，可以显示磁盘文件系统的状态和空间的使用情况；
- 提供虚拟磁盘转储功能，可将信息存入磁盘，还可从磁盘读入内存；

2. 开发环境

操作系统：windows 11

编程语言：C

集成开发环境（IDE）：CLion 2023.1.2

其他辅助工具：Markdown、WinDBG、Cmake、Git

3. 相关原理及算法

描述操作系统相关原理和算法等。

3.1 FAT 文件系统介绍

文件系统通常指的是管理存储设备上存储的数据的整个系统，然而，本文档描述了存储设备上 FAT 文件系统的数据格式。

FAT 文件系统起源于 1980 年左右，是最早由 MS-DOS 支持的文件系统。最初它是为适用于容量小于 500KB 的软盘而开发的简单文件系统。随着时间的推移，随着存储媒体容量的增加，其规范已经扩展以支持更大的存储媒体。FAT 是 File Allocation Table 的缩写，它是管理数据区分配的数组，也是文件系统本身的名称。目前有三种 FAT 子类型，分别是 FAT12、FAT16 和 FAT32。这些子类型按照编号的顺序进行开发，并且完全向后兼容旧版本（FAT16 始终包括 FAT12，FAT32 包括所有的 FAT 类型）。

3.2 FAT 文件系统基础

扇区

扇区是在存储设备上读写的最小数据块单元。通常的扇区大小为 512 字节，某些类型的存储介质可能使用更大的扇区大小。存储设备上的每个扇区都由一个按顺序分配的扇区号来寻址。由于卷不总是位于存储的顶部，本文档中的"扇区号"表示相对于卷顶部的位置起点，而"物理扇区号"表示相对于存储设备顶部的绝对位置起点。

FAT 卷

一个完整的 FAT 文件系统被称为逻辑卷（或逻辑驱动器）。FAT 逻辑卷由三个或四个区域组成，每个区域由一个或多个扇区组成，并按照以下顺序位于卷上（FAT 卷映射）：

- 保留区域（卷配置数据）

保留区域是 FAT 文件系统中的第一个区域，用于存储卷的配置数据和引导记录。它通常包含引导代码、文件系统参数、磁盘参数以及其他元数据。该区域在存储设备的开头位置，并且其大小是固定的。

- FAT 区域（数据区的分配表）

FAT 区域包含了文件分配表（File Allocation Table），用于记录存储设备上数据区的分配情况。文件分配表是一个表格，每个表项对应于存储设备上的一个簇（cluster）。簇是存储设备上数据的基本单位，用于存储文件和目录的内容。文件分配表中的每个表项指示了对应簇的使用情况，例如，是否已分配给某个文件或目录。

- 根目录区域（在 FAT32 卷上不存在）

根目录区域位于 FAT 文件系统的根目录位置，用于存储卷的根目录信息。它包含了文件和目录的条目，每个条目描述了一个文件或目录的属性和位置。根目录区域在 FAT32 卷中不再存在，因为 FAT32 采用了更为灵活的目录结构。

- 数据区域（文件和目录的内容）

数据区域是 FAT 文件系统中存储实际文件和目录内容的地方。它由一系列簇组成，每个簇的大小取决于文件系统的类型和配置。文件和目录的内容被分散存储在不同的簇中，文件分配表用于跟踪每个文件或目录所使用的簇的位置。通过读取文件分配表，可以找到文件或目录在数据区域中的具体位置，从而访问其内容。

存储上的数据形式

FAT 文件系统最初是为搭载 x86 处理器的 IBM PC 开发的。一个重要的考虑因素是，FAT 文件系统的数据结构是以小端序（Little Endian）方式存储的。如果在访问 FAT 文件系统的平台使用的是大端序（Big Endian）体系结构，那么在访问 FAT 文件系统的数据结构时就需要进行端序转换。此外，多字节的字数据并不总是对齐到字边界。如果处理器不支持对非对齐字数据的访问，就需要逐字节地访问数据。

因此，将 FAT 卷作为 C 结构的成员进行访问可能会破坏代码的可移植性。为了提供最佳的代码可移植性，最好以简单的字节数组形式逐字节地访问 FAT 卷，而不是将其作为 C 结构进行访问。这样可以确保代码在不同的平台上都能正确地访问 FAT 文件系统，并提高代码的可移植性。

3.3 引导扇区和 BPB（BIOS 参数块）

在 FAT 卷中，BPB（BIOS 参数块）是最重要的数据结构，它存储了 FAT 卷的配置参数。BPB 位于引导扇区中，这个扇区通常被称为 VBR（卷引导记录）或 PBR（私有引导记录），实际上它是保留区域的第一个扇区，也是卷的第一个扇区。

随着 FAT 文件系统不断增加新功能，BPB 经常发生变化。最初引起困惑的是 BPB 的引入。在 MS-DOS Ver.1 中，引导扇区中没有 BPB。在 FAT 文件系统的第一个版本中，磁盘格式是通过参考引导扇区下一个扇区中第一个 FAT 项的第一个字节（第一个 FAT 项的低 8 位）来确定的，而不是通过 BPB。

在 MS-DOS Ver.2 中，通过在引导扇区中引入 BPB 来替代确定磁盘格式的方法，不再支持通过参考 FAT 的第一个字节来确定磁盘格式。现在，所有的 FAT 卷都必须在引导扇区中包含 BPB。BPB 的变化有时会导致确定磁盘格式时的困惑（例如，哪个参数是正确的？这个参数的含义是什么？如何使用这个参数？），关于磁盘格式的确定方法在稍后的 FAT 规范中进行了描述。

随着硬盘的广泛应用和磁盘大小的增加，导致磁盘利用效率下降和卷上文件数量的限制问题。因此，在 MS-DOS Ver.3 中引入了对 FAT16 的支持。然而，这一变化立即引发了新的问题。由于用于指示卷大小的字段大小为 16 位，所以支持的卷大小最大只能是 65536 个扇区（32 MB，每个扇区 512 字节）。为了解决这个问题，在 MS-DOS Ver.3.31 中添加了一个 32 位字段，以支持 128 MB 的 FAT12 卷和 2 GB 的 FAT16 卷（每簇 32 KB）。

BPB 的最后一次变化是在引入 FAT32 时发生的，这发生在 Windows 95 OSR2 中。此时，FAT16 卷的文件数和最大容量已经达到了某些应用程序的限制。作为 FAT 文件系统的最终版本，FAT32 消除了 FAT16 的限制，并支持最多 2 TB 的卷大小（每个扇区 512 字节）。然而，对于大于 32 GB 的卷，微软建议使用除 FAT 之外的其他文件系统，如 NTFS 用于固定磁盘，exFAT 用于可移动磁盘。

下表显示了引导扇区中的数据字段。以 BPB 为标题的字段是 BPB 的一部分，而以 BS 为标题的字段不是 BPB 的一部分，而是引导扇区的一部分。

前 36 个字节的字段是所有 FAT 类型共有的字段，而从偏移量 36 开始的字段取决于 FAT 类型是 FAT32 还是 FAT12/FAT16。关于确定 FAT 类型的方法将在下一节中描述。

FAT12/16/32 公共字段（偏移量从 0 到 35）

字段名称	偏移量	大小	描述
BS_JmpBoot	0	3	引导代码的跳转指令（x86 指令），用于操作系统的引导序列。这个字段有两种格式，首选的是前一种格式：0xEB, 0x??, 0x90（短跳转 + NOP），0xE9, 0x??, 0x??（近跳转）。?? 是一个任意值，取决于要跳转到的位置。如果使用这些格式之外的任何格式，Windows 将无法识别该卷。
BS_OEMName	3	8	推荐使用“MSWIN 4.1”，但通常使用“MSDOS 5.0”。关于这个字段存在许多误解。这只是一个名称。Microsoft 的操作系统不关注这个字段，但一些 FAT 驱动程序会参考它。推荐使用这个字符串，因为它被认为可以最大限度地减少兼容性问题。您可以设置其他值，但是某些 FAT 驱动程序可能无法识别该卷。这个字段通常指示创建卷的系统的名称。
BPB_BytsPerSec	11	2	每个扇区的字节大小。该字段的有效值为 512、1024、2048 或 4096。Microsoft 的操作系统正确支持这些扇区大小，但许多 FAT 驱动程序假设扇区大小为 512，并且不检查该字段。出于最大的兼容性，应该使用 512。但是，不要误解它仅与兼容性有关。该值必须与包含 FAT 卷的存储的扇区大小相同。
BPB_SecPerClus	13	1	每个分配单元的扇区数。在 FAT 文件系统中，分配单元称为“簇”。这是一个或多个连续扇区的块，数据区在这个单位中管理。每个簇的扇区数必须是 2 的幂。因此，有效的值为 1、2、4... 和 128。但是，不应使用任何簇大小（BPB_BytsPerSec * BPB_SecPerClus）超过 32 KB。最近的系统（如 Windows）支持大于 32 KB 的簇大小，例如 64 KB、128 KB 和 256 KB，但这样的卷将无法被 MS-DOS 或旧磁盘工具正确识别。
BPB_RsvdSecCnt	14	2	保留区中的扇区数。该字段不能为 0，因为引导扇区本身在保留区中包含了这个 BPB。为了避免兼容性问题，在 FAT12/16 卷上应该设置为 1。这是因为一些旧 FAT 驱动程序会忽略这个字段，并假设保留区的大小为 1。在 FAT32 卷上，它通常为 32。Microsoft 的操作系统正确支持 1 或更大的任何值。

BPB_NumFATs	16	1	FAT 的数量。该字段的值应始终为 2。也可以使用大于等于 1 的任何值，但强烈建议不要使用 2 以外的值，以避免兼容性问题。Microsoft 的 FAT 驱动程序正确支持除 2 以外的值，但某些工具和 FAT 驱动程序会忽略这个字段，并将 FAT 的数量设为 2。该字段的标准值 2 是为 FAT 数据提供冗余。FAT 条目的值通常从第一个 FAT 中读取，并且对 FAT 条目的任何更改都会反映到每个 FAT 中。如果 FAT 区域的某个扇区损坏，数据将不会丢失，因为它在另一个 FAT 中有副本。因此，它可以最小化数据丢失的风险。对于非基于磁盘的存储介质（如存储卡），这种冗余是一个无用的功能，因此可以将其设置为 1 以节省磁盘空间。但某些 FAT 驱动程序可能无法正确识别这样的卷。
BPB_RootEntCnt	17	2	在 FAT12/16 卷上，该字段表示根目录中 32 字节目录项的数量。该值应设置为根目录大小对齐到 2 个扇区边界，即 $BPB_RootEntCnt * 32$ 成为 $BPB_BytsPerSec$ 的偶数倍。为了最大的兼容性，这个字段在 FAT16 卷上应该设置为 512。对于 FAT32 卷，该字段必须为 0。
BPB_TotSec16	19	2	旧的 16 位字段中的卷的总扇区数。该值是包括卷的所有四个区域的扇区数。当 FAT12/16 卷的扇区数大于等于 0x10000 时，该字段设置为无效值 0，真实值设置为 $BPB_TotSec32$ 。对于 FAT32 卷，该字段始终为 0。
BPB_Media	21	1	该字段的有效值为 0xF0、0xF8、0xF9、0xFA、0xFB、0xFC、0xFD、0xFE 和 0xFF。0xF8 是非可移动磁盘的标准值，0xF0 经常用于未分区的可移动磁盘。另一个重要的点是，相同的值必须放在 FAT[0] 的低 8 位中。这源自 MS-DOS 1 版本的介质确定，不再用于任何目的。
BPB_FATSz16	22	2	FAT 占用的扇区数。该字段仅用于 FAT12/16 卷。在 FAT32 卷上，它必须是一个无效值 0，而使用 $BPB_FATSz32$ 代替。FAT 区域的大小为 $BPB_FATSz?? * BPB_NumFATs$ 个扇区。
BPB_SecPerTrk	24	2	每个磁道的扇区数。该字段仅涉及具有几何结构的介质，并仅用于 IBM PC 的磁盘 BIOS。
BPB_NumHeads	26	2	磁头的数量。该字段仅涉及具有几何结构的介质，并仅用于 IBM PC 的磁盘 BIOS。
BPB_HiddSec	28	4	FAT 卷之前的隐藏物理扇区数。它通常与 IBM PC 的磁盘 BIOS 访问的存储相关，并且设置什么样的值取决于平台。如果卷从存储的开始处开始，例如非分区磁盘（如软盘），则该字段应始终为 0。
BPB_TotSec32	32	4	新的 32 位字段中的 FAT 卷的总扇区数。该值是包括卷的所有四个区域的扇区数。当 FAT12/16 卷的值小于 0x10000 时，该字段必须是无效值 0，真实值设置为 $BPB_TotSec16$ 。对于 FAT32 卷，该字段始终是有效的，旧字段不使用。

在确定这些字段之前，需要先确定卷是 FAT12/16 还是 FAT32，因为以下字段根据 FAT 类型的不同而发生变化。此外，有一些字段仅存在于 FAT32 卷中，而在 FAT12/16 卷中不存在。

FAT12/16 卷的字段(从 36 偏移)

字段名称	偏移量	大小	描述
BS_DrvNum	36	1	IBM PC 的磁盘 BIOS 使用的驱动器号。此字段在 MS-DOS 引导程序中使用，软盘为 0x00，固定磁盘为 0x80。实际上这取决于操作系统。
BS_Reserved	37	1	保留字段（由 Windows NT 使用）。在创建卷时应设置为 0。

BS_BootSig	38	1	扩展引导签名（0x29）。这是一个签名字节，表示以下三个字段存在。
BS_VolID	39	4	与 BS_VolLab 一起用于跟踪可移动存储介质上的卷的卷序列号。它能够通过 FAT 驱动程序检测到错误的媒体更改。通常在格式化时，该值会根据当前时间和日期生成。
BS_VolLab	43	11	这个字段是 11 字节的卷标，与记录在根目录中的卷标匹配。当根目录中的卷标更改时，FAT 驱动程序应更新此字段。MS-DOS 会执行此操作，但 Windows 不会执行。当卷标不存在时，此字段应设置为“NO NAME”。
BS_FilSysType	54	8	“FAT12”、“FAT16”或“FAT”。许多人认为此字符串在确定 FAT 类型时有影响，但这一个明显的误解。从字段的名称可以看出，这不是 BPB 的一部分。由于该字符串通常不正确或未设置，Microsoft 的 FAT 驱动程序不使用此字段来确定 FAT 类型。然而，一些旧的 FAT 驱动程序使用此字符串来确定 FAT 类型，因此应根据卷的 FAT 类型设置它，以避免兼容性问题。
BS_BootCode	62	448	引导程序。它与平台相关，在未使用时填充为零。
BS_BootSign	510	2	0xAA55。引导签名，表示这是一个有效的引导扇区。
	512		当扇区大小大于 512 字节时，扇区中的其余字段应填充为零。

FAT32 卷的字段(从 36 偏移量)

字段名称	偏移量	大小	描述
BPB_FATSz32	36	4	FAT 的大小，以扇区为单位。FAT 区域的大小为 BPB_FATSz32 * BPB_NumFATs 个扇区。这是在确定 FAT 类型之前需要参考的唯一字段，而且该字段仅存在于 FAT32 卷中。但这不是问题，因为 FAT32 卷中的 BPB_FATSz16 始终无效。
BPB_ExtFlags	40	2	Bit3-0: 活动的 FAT 从 0 开始。当 bit7 为 1 时有效。Bit6-4: 保留 (0)。Bit7: 0 表示每个 FAT 都是活动的并且镜像的。1 表示只有由 bit3-0 指示的一个 FAT 是活动的。Bit15-8-4: 保留 (0)。
BPB_FSVer	42	2	FAT32 版本。高字节是主版本号，低字节是次版本号。本文档描述了 FAT32 版本 0.0。该字段用于将来扩展 FAT32 卷以管理文件系统版本。然而，FAT32 卷将不再更新。
BPB_RootClus	44	4	根目录的第一个簇号。通常设置为 2，即卷的第一个簇，但不一定总是 2。
BPB_FSInfo	48	2	从 FAT32 卷顶部的偏移处的 FSInfo 结构的扇区。通常设置为 1，即引导扇区的旁边。
BPB_BkBootSec	50	2	从 FAT32 卷顶部的偏移处的备份引导扇区的扇区。通常设置为 6，即引导扇区

			的旁边，但不推荐使用 6 或任何其他值。
BPB_Reserved	52	12	保留字段（0）。
BS_DrvNum	64	1	与 FAT12/16 字段的描述相同。
BS_Reserved	65	1	与 FAT12/16 字段的描述相同。
BS_BootSig	66	1	与 FAT12/16 字段的描述相同。
BS_VolID	67	4	与 FAT12/16 字段的描述相同。
BS_VolLab	71	11	与 FAT12/16 字段的描述相同。
BS_FilSysType	82	8	始终为“FAT32 ”，对 FAT 类型的确定没有任何影响。
BS_BootCode32	90	420	引导程序。它与平台相关，在未使用时填充为零。
BS_BootSign	510	2	0xAA55。引导签名，表示这是一个有效的引导扇区。
	512		当扇区大小大于 512 字节时，扇区中的其余部分应填充为零。

引导扇区还有另一个重要的方面。引导扇区只有在引导标志（BS_Sign）包含 0xAA55 时才有效。如果不包含该标志，则引导扇区无效。许多 FAT 文档描述了这个字段，即“引导标志位于引导扇区的末尾”。这只有在扇区大小为 512 字节时才是正确的，但在其他情况下是错误的。引导标志必须始终放置在偏移量 510 处（同时放置在偏移量 510 和扇区末尾也没有问题）。如果扇区大小大于 512 字节，Microsoft 的磁盘格式化程序会用零填充引导扇区的其余部分。使用旧版本 MS-DOS 格式化的卷通常不设置 BS_Sign。

卷的大小，即 BPB_TotSec?? 的值，可能小于包含该卷的容器（存储或分区）的大小。这并不是问题。由于卷的对齐或调整大小，FAT 卷有时会出现这种状态。这样的对齐空间浪费了磁盘空间，但并不意味着 FAT 卷本身存在问题。

然而，如果 BPB_TotSec?? 大于卷的容器大小，可以说 FAT 卷处于严重状态，可能是损坏的卷或创建不正确的卷。对这样的卷进行任何操作都可能导致灾难性的数据丢失，因此，如果 FAT 驱动程序检测到这种情况，应拒绝操作该卷。

3.4 计算参数

通过使用 BPB 中的参数，可以计算出每个区域的偏移量和大小，如下所示：

由于 FAT 区域紧接着保留区域，其偏移量和大小如下计算：


```
FatStartSector = BPB_ResvdSecCnt;  
FatSectors = BPB_FATSz * BPB_NumFATs;
```

根目录的偏移量和大小如下计算：

```
RootDirStartSector = FatStartSector + FatSectors;  
RootDirSectors = (32 * BPB_RootEntCnt + BPB_BytsPerSec - 1) /  
BPB_BytsPerSec;
```

等式中的 32 表示目录项的大小。除法运算时会向上取整，但不建议出现除法的余数。在 FAT32 卷上，BPB_RootEntCnt 始终为 0，因此不存在根目录区域。数据区域是这些区域的剩余部分，计算如下：

```
DataStartSector = RootDirStartSector + RootDirSectors;  
DataSectors = BPB_TotSec - DataStartSector;
```

如果卷不是从存储器的顶部开始的，例如在存储器进行了分区，那么这些起始扇区号与物理扇区号不相等。

3.5 FAT 和簇

另一个重要的区域是 FAT（文件分配表）。FAT 的结构定义了文件的扩展（簇链）的链接列表。需要注意的是，目录和文件都包含在文件中，在 FAT 上没有任何区别。目录实际上是一个具有特殊属性的文件，指示其内容是一个目录表。

数据区域被划分为一定数量扇区的块（BPB_SecPerClus），称为簇，数据区域以这个单位进行管理。FAT 的每个条目与数据区域中的每个簇相关联，FAT 的值表示相应簇的状态。然而，FAT 的前两个条目，FAT[0]和 FAT[1]，是保留的，不与任何簇相关联。第三个 FAT 条目，FAT[2]，与数据区域的第一个簇相关联，有效的簇号从 2 开始。关于 FAT 中记录的数据，请参考文件和簇的关联。

为了冗余起见，通常会对 FAT 进行复制，因为任何 FAT 扇区的损坏都会导致严重的数据丢失。BPB_NumFATs 指示了 FAT 的副本数，FAT 区域的大小为 BPB_FATSz * BPB_NumFATs。FAT 驱动程序通常仅引用第一个 FAT 副本，并且对 FAT 项的任何更新都会在每个 FAT 副本上反映出来。

3.6 确定 FAT 子类型

在挂载 FAT 卷时，需要确定 FAT 的三种类型，即 FAT12、FAT16 和 FAT32。然而，对于确切的确定方法存在相当大的混淆，这导致了各种程度的错误。实际上，确定方法非常简单。

FAT 的类型是通过卷上的簇数来确定，没有其他因素。

簇数是数据区域中可能存在的簇的数量，即数据区域大小除以簇大小的商。如果结果中存在余数，则忽略余数。

```
簇数 = DataSectors / BPB_SecPerClus;
```

一旦知道了簇数，就可以确定 FAT 的类型。具体方法如下：

- 簇数小于等于 4085 的卷是 FAT12 类型。
- 簇数在 4086 到 65525 之间的卷是 FAT16 类型。
- 簇数大于等于 65526 的卷是 FAT32 类型。

这是确定 FAT 类型的唯一方法。FAT12 卷的簇数永远不会超过 4085，而 FAT16 卷的簇数永远不会小于 4086 或大于 65525。如果尝试根据这个规则创建非法的 FAT 卷，正确设计的 FAT 驱动程序将将其识别为不同的 FAT 类型，并无法访问该卷。对于 FAT32 卷的最大簇数没有定义，实际限制为 268435445。

然而，这些边界并没有严格确定。从簇号的最大可能值来看，它将按照上述描述进行划分，但现有的文档和软件实现存在许多变种（1、2、16 或更多）。例如，关于 FAT12 的最大簇数，FAT 规范指出是 4084，而 MSDN 页面指出是 4085，而 Windows 使用的是 4085（FAT 驱动程序）或 4086（chkdsk）。即使授权的文档和标准系统在正确的值上也存在差异，因此建议在创建 FAT 卷时避免接近边界的簇数。FAT 规范指出，簇数应与边界至少相差 16 个簇。

此外，一些不遵循这一规则的 FAT 驱动程序似乎是根据 BS_FilSysType 字符串而不是簇数来确定 FAT 类型。为了支持这种错误的 FAT 驱动程序，在创建 FAT 卷时，建议根据实际的 FAT 类型，使用适当的字符串初始化 BS_FilSysType。

通过了解簇数决定 FAT 类型的含义，可以看出合法的 FAT 类型取决于卷的大小。例如，在簇大小为 512 到 32768 字节的条件下，可以如下表述：

FAT type	Volume size
FAT12	- 128 MB
FAT16	2 MB - 2 GB
FAT32	32 MB - 2 TB

3.7 FAT 条目的访问方法

在访问 FAT 条目时，有一个与 FAT 相关的重要问题需要考虑。首先，你需要知道 FAT 条目在 FAT 中的位置。在 FAT16/32 中，这是相当简单的。FAT 是一个简单的通用整数数组。与内存中的数组不同，FAT 不是连续的内存，而是分为多个块，并存储在从 FAT 的第一个扇区开始的连续磁盘扇区上。FAT 条目的位置 FAT[N]，即扇区号和扇区内字节偏移量，可以通过以下计算得到。

FAT16 条目位置：

```
ThisFATSecNum = BPB_ResvdSecCnt + (N * 2 / BPB_BytsPerSec);
ThisFATEntOffset = (N * 2) % BPB_BytsPerSec;
```

FAT32 条目位置:

```
ThisFATSecNum = BPB_ResvdSecCnt + (N * 4 / BPB_BytsPerSec);
ThisFATEntOffset = (N * 4) % BPB_BytsPerSec;
```

从该位置读取的 2 字节 (FAT16) 或 4 字节 (FAT32) 数据就是要访问的 FAT 条目。字节顺序为小端序。FAT 条目在 FAT16/32 中不会跨越扇区边界。

关于 FAT32 还有另一个重要的问题。FAT32 卷的 FAT 条目占据 32 位, 但其高 4 位保留, 只有低 28 位有效。在创建 FAT32 卷时, 这些保留位被初始化为零, 并且在常规使用中不应更改。因此, 在从 FAT32 卷的 FAT 条目中加载值时, 需要对高 4 位进行与操作, 并将其与 0x0FFFFFFF 进行屏蔽。此外, 当将值存储到 FAT 条目时, 需要保留 FAT 条目中的高 4 位。

加载 FAT32 条目的值:

```
ReadSector(SecBuff, ThisFATSecNum);
ThisEntryVal = *(uint32*)&SecBuff[ThisFATEntOffset] & 0x0FFFFFFF;
```

存储 FAT32 条目的值:

```
ReadSector(SecBuff, ThisFATSecNum);
tmp = *(uint32*)&SecBuff[ThisFATEntOffset];
tmp = (tmp & 0xF0000000) | (NewEntryVal & 0x0FFFFFFF);
*(uint32*)&SecBuff[ThisFATEntOffset] = tmp;
WriteSector(SecBuff, ThisFATSecNum);
```

在 FAT12 卷上有一点复杂。FAT12 条目是位域, 并且需要进行复杂的操作。

FAT12 条目位置:

```
ThisFATSecNum = BPB_ResvdSecCnt + ((N + (N / 2)) / BPB_BytsPerSec);
ThisFATEntOffset = (N + (N / 2)) % BPB_BytsPerSec;
```

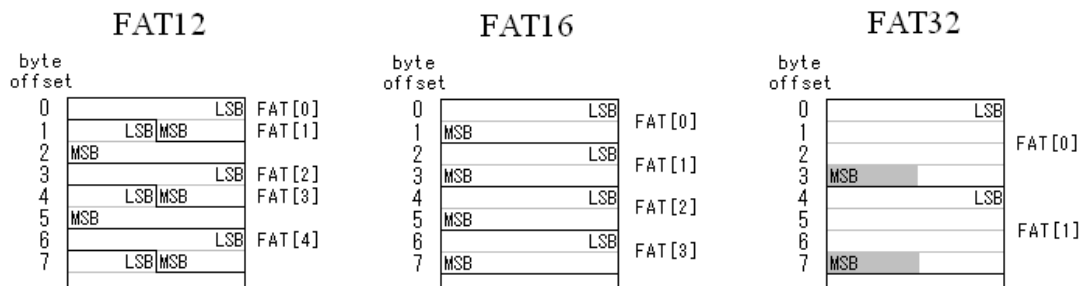
加载 FAT12 条目的值:

```
ReadSector(SecBuff, ThisFATSecNum);
if (N & 1) { /* 奇数条目 */
    ThisEntryVal = (SecBuff[ThisFATEntOffset] >> 4)
        | ((uint16)SecBuff[ThisFATEntOffset + 1] << 4);
} else { /* 偶数条目 */
    ThisEntryVal = SecBuff[ThisFATEntOffset]
        | ((uint16)(SecBuff[ThisFATEntOffset + 1] & 0x0F) <<
8);
}
```

存储 FAT12 条目的值:

```
ReadSector(SecBuff, ThisFATSecNum);
if (N & 1) { /* 奇数条目 */
    SecBuff[ThisFATEntOffset] = (SecBuff[ThisFATEntOffset] & 0x0F)
    | (NewEntryVal << 4);
    SecBuff[ThisFATEntOffset + 1] = NewEntryVal >> 4;
} else { /* 偶数条目 */
    SecBuff[ThisFATEntOffset] = NewEntryVal;
    SecBuff[ThisFATEntOffset + 1] = (SecBuff[ThisFATEntOffset + 1] &
0xF0)
    | ((NewEntryVal >> 8) & 0x0F);
}
WriteSector(SecBuff, ThisFATSecNum);
```

不幸的是，由于位域跨越了扇区边界，所以这段代码不能正常工作，需要适当处理。
下图显示了每种 FAT 类型的 FAT 使用情况。



3.8 文件与簇的关联

在 FAT 卷上，文件由目录进行管理，目录是 32 字节目录项结构的数组。目录项的详细信息如下所述。目录项包含文件名、文件大小、时间戳和文件数据的第一个簇号。簇号是跟随文件数据的簇链的入口点。如果文件大小为零，则将第一个簇号设置为零，并且没有数据簇分配给文件。

如上所述，簇号 0 和 1 是保留的，有效的簇号从 2 开始。簇号 2 对应于数据区的第一个簇。因此，在具有 N 个簇的卷中，有效的簇号范围是从 2 到 N + 1，FAT 条目的计数是 N + 2。数据簇 N 的位置计算如下：

$$\text{FirstSectorofCluster} = \text{DataStartSector} + (N - 2) * \text{BPB_SecPerClus};$$

如果文件大小大于扇区大小，文件数据将跨越簇中的多个扇区。如果文件大小大于簇大小，文件数据将跨越簇链中的多个簇。FAT 条目的值指示下一个存在的簇号，从而可以通过跟随簇链来访问文件中的任意字节偏移量。无法向后跟随簇链。簇链的最后一个链接具有特殊值（簇链的末尾，EOC 标记），该值永远不会与任何有效的簇号匹配。各种 FAT 类型的 EOC 标记如下所示：

FAT12: 0xFF8 - 0xFFF (通常为 0xFFF)

FAT16: 0xFFF8 - 0xFFFF (通常为 0xFFFF)

FAT32: 0x0FFFFFF8 - 0x0FFFFFFF (通常为 0x0FFFFFFF)

还有一个特殊值，坏簇标记。坏簇标记表示簇中存在有缺陷的扇区，不能使用该簇。在格式化、表面检查或磁盘修复中发现的坏簇会记录在 FAT 中作为坏簇标记。坏簇标记的值为 0xFF7 (FAT12)、0xFFF7 (FAT16) 和 0x0FFFFFF7 (FAT32)。

在 FAT12/16 卷上，坏簇标记的值永远不等于任何有效的簇号。然而，它可以等于任何可分配的簇号，因为 FAT32 的最大簇数未定义。这样的 FAT32 卷可能会使磁盘工具混淆，因此应避免创建这样的 FAT32 卷。因此，FAT32 卷的簇数上限实际上是 268435445 (大约 256M 个

簇)。

由于实现原因，一些系统对最大簇数有限制。例如，Windows9X/Me 支持最大 16 MB 的 FAT 大小，限制了最大 4 M 个簇的数量。

每个可分配的 FAT 条目 (FAT[2]及其后面的条目) 的初始值为零，表示该簇未使用，可供新分配使用。如果值不为零，则表示该簇已使用或为坏簇。FAT12/16 卷中没有记录任何地方的空闲簇计数，需要进行完整的 FAT 扫描才能获得此信息。FAT32 支持 FSInfo 来存储空闲簇计数，以避免进行完整的 FAT 扫描，因为其 FAT 结构非常庞大。

前两个 FAT 条目，FAT[0]和 FAT[1]，是保留的，不与任何簇关联。在创建卷时，这些 FAT 条目按以下方式进行初始化：

FAT12: FAT[0] = 0xF??; FAT[1] = 0xFFF;

FAT16: FAT[0] = 0xFF??; FAT[1] = 0xFFFF;

FAT32: FAT[0] = 0xFFFFFFFF??; FAT[1] = 0xFFFFFFFFF;

FAT[0]中的??值与 BPB_Media 的相同值，但该条目没有任何功能。FAT[1]中的某些位记录了错误历史。

卷的脏标志 (FAT16: bit15、FAT32: bit31)：在启动时清除，在正常关闭时恢复。在启动时已经清除表示脏关闭，并可能在卷中存在逻辑错误的可能性。

硬错误标志 (FAT16: bit14、FAT32: bit30)：在无法恢复的读/写错误时清除，表示需要进行表面检查。

这些标志指示卷上可能存在错误的的可能性。支持此功能的某些操作系统在启动时检查这些标志并自动启动磁盘检查工具。Windows 9X 系列使用这些标志。Windows NT 系列不使用这些标志，而使用 BPB 中的替代功能。

关于 FAT 区域还有两个重要的事项。首先，FAT 的最后一个扇区可能没有完全使用。在大多数情况下，FAT 在扇区的中间结束。FAT 驱动程序不对未使用的区域做任何假设。在格式化卷时，应使用零填充，之后不应更改。另一个问题是，BPB_FATSz16/32 可以指示一个大于卷要求的值。换句话说，未使用的扇区可以跟

随每个 FAT。这可能是数据区对齐或其他原因的结果。这些扇区在格式化时用零填充。

下表显示了各种 FAT 类型的 FAT 值范围和含义：

FAT12	FAT16	FAT32	含义
0x000	0x0000	0x00000000	Free
0x001	0x0001	0x00000001	Reserved
0x002 - 0xFF6	0x0002 - 0xFFF6	0x00000002 - 0x0FFFFFFF6	In use (value is link to next)
0xFF7	0xFFF7	0x0FFFFFFF7	Bad cluster
0xFF8 - 0xFFF	0xFFF8 - 0xFFFF	0x0FFFFFFF8 - 0x0FFFFFFF	In use (end of chain)

3.9 FSInfo 扇区结构和备份引导扇区

在 FAT12 卷上，FAT 的大小可达 6 KB，在 FAT16 卷上可达 128 KB，但在 FAT32 卷上通常达到几 MB。因此，FAT32 卷支持 FSInfo 结构，以避免读取整个 FAT 来查找空闲簇或获取空闲簇的计数。该结构被放置在由 BPB_FSInfo 指示的 FSInfo 扇区中。

FAT32 FSInfo sector

字段名	偏移	大小	描述
FSI_LeadSig	0	4	0x41615252。这是一个引导签名，用于验证这确实是一个 FSInfo 扇区。
FSI_Reserved1	4	480	保留字段。这个字段应该始终初始化为零。
FSI_StrucSig	484	4	0x61417272。另一个签名，更局部地位于扇区中，用于标识所使用的字段。
FSI_Free_Count	488	4	此字段指示卷上已知的最后一个空闲簇的计数。如果值为 0xFFFFFFFF，则实际上是未知的。这并不一定是正确的，因此 FAT 驱动程序需要确保它对于卷是有效的。
FSI_Nxt_Free	492	4	此字段为 FAT 驱动程序提供提示，指示驱动程序应从哪个簇号开始寻找空闲簇。由于 FAT32 的 FAT 很大，在 FAT 的开头有很多分配的簇时，如果驱动程序从第一个簇开始寻找空闲簇，可能会非常耗时。通常情况下，此值设置为驱动程序最后分配的簇号。如果值为 0xFFFFFFFF，则表示没有提示，驱动程序应从簇 2 开始查找。这可能不正确，因此 FAT 驱动程序需要确保它对于卷是有效

			的。
FSI_Reserved2	496	12	保留字段。这个字段应该始终初始化为零。
FSI_TrailSig	508	4	0xAA550000。这个结束签名用于验证这确实是一个 FSInfo 扇区。
	512		当扇区大小大于 512 字节时，剩余的字节应初始化为零。

另一个 FAT32 卷的特性是备份引导扇区。这是一项功能，为 FAT 卷上唯一存在的引导扇区提供冗余。如果由于任何原因引导扇区损坏，这可以增加卷恢复的可能性。备份引导扇区的位置由 **BPB_BkBootSec** 指示。

强烈推荐将 6 作为备份引导扇区字段的值，因为引导加载程序和 FAT 驱动程序在尝试读取主引导扇区失败时会硬编码为尝试读取第 6 扇区的引导扇区。实际上，FAT32 引导扇区实际上是由三个 512 字节的扇区组成。从 **BPB_BkBootSec** 指示的扇区开始，存在这三个扇区的副本。即使在备份引导扇区的 **BPB_FSInfo** 字段设置为与第 0 扇区相同的值，FSInfo 扇区的副本也在那里。这三个扇区的偏移量 510 处都有引导标识 0xAA55。

备份引导扇区提供了对引导扇区和相关数据的冗余备份，以增加 FAT32 卷的可靠性和恢复能力。在引导扇区损坏时，这些备份数据可以用于快速恢复卷的正常操作。

3.10 FAT 目录

本节介绍了 FAT 卷的基本特性，即短文件名（SFN）。目录实际上是一个带有特殊属性的文件。它包含了存储在卷上的文件的元数据的目录条目表。每个目录条目的大小为 32 字节，对应于卷上的一个文件或目录。目录的最大大小为 2MB（65536 个条目）。

根目录是一个特殊的目录，必须始终存在，并成为卷中层次结构的顶级节点。在 FAT12/16 卷上，根目录不是一个文件，而是放置在与数据区分开的根目录区域中。根目录条目的数量在格式化过程中确定，并在 **BPB_RootEntCnt** 中指示。在 FAT32 卷上，除了没有任何条目指示其以外，子目录与根目录之间没有区别，并且起始簇号由 **BPB_RootClus** 指示。

与子目录的另一个区别是，根目录不包含子目录中始终存在的点条目（"."，".."），但它可以包含一个卷标（具有 **ATTR_VOLUME_ID** 属性的条目）。下表显示了目录条目的结构：

目录项结构

字段名	偏移量	大小	描述
DIR_Name	0	11	对象的短文件名（SFN）。
DIR_Attr	11	1	文件属性的组合标志位。最高的 2 位为保留位，必须为零。0x01：ATTR_READ_ONLY（只读）0x02：ATTR_HIDDEN（隐藏）0x04：ATTR_SYSTEM（系统）0x08：ATTR_VOLUME_ID（卷标）0x10：ATTR_DIRECTORY（目录）0x20：ATTR_ARCHIVE（存档）0x0F：ATTR_LONG_FILE_NAME（LFN 条目）
DIR_NTRes	12	1	可选标志位，指示 SFN 的大小写信息。0x08：文件名的每个字母都为小写。0x10：扩展名的每个字母都为小写。
DIR_CrtTimeTenth	13	1	可选的亚秒信息，对应于 DIR_CrtTime。DIR_CrtTime 的时间分辨率为 2 秒，因此该字段提供了亚秒的计数，有效值范围为 0 到 199，以 10 毫秒为单位。如果不支持，设置为零，之后不再更改。
DIR_CrtTime	14	2	可选的文件创建时间。如果不支持，设置为零，之后不再更改。
DIR_CrtDate	16	2	可选的文件创建日期。如果不支持，设置为零，之后不再更改。
DIR_LstAccDate	18	2	可选的最后访问日期。最后访问时间没有时间信息，因此最后访问时间的分辨率为 1 天。如果不支持，设置为零，之后不再更改。
DIR_FstClusHI	20	2	簇号的高位部分。在 FAT12/16 卷上始终为零。参见 DIR_FstClusLO。
DIR_WrtTime	22	2	文件最后修改时间（通常在关闭时）。
DIR_WrtDate	24	2	文件最后修改日期（通常在关闭时）。
DIR_FstClusLO	26	2	簇号的低位部分。当文件大小为零时，不分配任何簇给文件，此项必须为零。如果是目录，则始终为有效值。
DIR_FileSize	28	4	文件的大小（以字节为单位）。如果是目录，则不使用，并且该值必须始终为零。

DIR_Name 字段的第一个字节，DIR_Name[0]，是一个重要的数据，用于指示目录条目的状态。当值为 0xE5 时，表示该条目未使用（可用于新分配）。当值为 0x00 时，表示该条目未使用（与 0xE5 相同），此外，在该条目之后没有分配的条目（在该条目之后的所有条目的 DIR_Name[0] 也设置为 0）。DIR_Name[0] 中的任何其他值表示该条目正在使用中。关于以 0xE5 开头的文件名，有一个例外情况。在这种情况下，设置为 0x05。

DIR_Name 字段是一个 11 字节的字符串，分为两部分，主体和扩展名。文件名存储在 8 字节的主体+3 字节的扩展名中。文件名中用于分隔主体和扩展名的点在目录条目中被删除。如果任何部分的名称不适合该部分，该部分的其余字节将用空格（0x20）填充。用于文件名的代码页取决于系统。

文件名	DIR_Name[]	描述
"FILENAME.TXT"	"FILENAME.TXT"	点号被移除。
"DOG.JPG"	"DOG JPG"	每个部分都用空格填充。
"file.txt"	"FILE TXT"	小写字母被转换为大写。
"厩気楼.JPG"	"・気楼 JPG"	"厩"的第一个字节 0xE5 被替换为 0x05。
"NOEXT"	"NOEXT "	没有扩展名。
".cnf"		（不合法）不允许没有文件名主体的名称。
"new file.txt"		（不合法）不允许有空格。
"file[1].2+2"		（不合法）不允许使用 [] +。
"longext.jpeg"		（不合法）超出了 8.3 格式。
"two.dots.txt"		（不合法）超出了 8.3 格式。

文件名的合法字符包括 ASCII 字符中的 0~9、A~Z、!、#、\$、%、&、'、(、)、-、@、^、_、`、{、}、~，以及扩展字符（\x80 - \xFF）。输入文件名中的小写 ASCII 字符（a-z）在匹配和记录之前会被替换为大写字符。至于扩展字符，在不同系统之间替换的方式存在许多差异，例如 Ää→ÄÄ（CP852）和 Ää→AA（CP850）。因此，即使具有相同的二进制名称，使用扩展字符可能会在不同系统之间引起兼容性问题（例如，文件打开失败）。至于日语环境中的双字节字符，可参考下面的兼容性说明。

目录中的每个文件名都是唯一的，不存在同名的其他条目。DIR_Attr 字段指示条目的属性。

文件属性

标志	含义
----	----

ATTR_READ_ONLY	只读文件。不允许对文件进行更改或删除。
ATTR_HIDDEN	正常的目录列表不显示该文件（系统相关）。
ATTR_SYSTEM	表示这是一个系统文件（系统相关）。
ATTR_DIRECTORY	表示这是一个目录的容器。
ATTR_ARCHIVE	用于备份工具。在创建、修改或重命名文件时由 FAT 驱动器设置。备份工具能够轻松找到要备份的文件，并在备份完成后清除此属性。
ATTR_VOLUME_ID	具有此属性的条目包含卷的卷标。根目录中只能存在一个条目。DIR_FstClusHI、DIR_FstClusLO 和 DIR_FileSize 字段必须始终为零。某些系统可能会设置 ATTR_ARCHIVE，但它没有意义。
ATTR_LONG_NAME	此组合属性表示该条目是长文件名的一部分。详细信息请参见下文描述。

3.11 目录操作

创建文件

要创建文件，FAT 驱动程序会在目录中找到一个空闲条目进行创建。如果目录中没有找到空闲条目，则会延伸目录一个簇以分配新的空闲条目，但目录的大小不能超过 2MB（65536 个条目）。静态目录的大小（在 FAT12/16 卷中的根目录）是固定的，无法更改。新条目的名称存储在 DIR_Name 中，DIR_Attr 中的 ATTR_ARCHIVE 标志，以及 DIR_FstClusHI、DIR_FstClusLO 和 DIR_FileSize 的初始值为 0。当对文件写入任何数据并且文件大小从 0 更改时，会创建一个新的簇链，并将第一个簇号存储到 DIR_FstClusHI 和 DIR_FstClusLO 中。随着文件大小的增加，簇链会被扩展。

创建子目录

要创建子目录，FAT 驱动程序会像创建文件一样创建一个目录条目。该条目需要具有 ATTR_DIRECTORY 属性。子目录没有大小信息，DIR_FileSize 字段必须始终为零。一个簇最初被分配给子目录，并将簇号设置为 DIR_FstClusHI 和 DIR_FstClusLO 字段。簇中的每个条目都被初始化为零。当目录表变满时，簇链会被扩展，并且簇被初始化为零。目录的最大长度为 2MB（64K 个条目）。

子目录在目录的顶部有两个特殊条目（点条目），DIR[0]为"."，DIR[1]为"..". 这些条目具有 ATTR_DIRECTORY 属性，但它们不具有任何簇，而是指向另一个目录的簇。"."条目指向当前目录，".."条目指向父目录。如果父目录是根目录，则即使在 FAT32 卷上，将 DIR_FstClusHI 和 DIR_FstClusLO 字段设置为零。

删除文件

要删除文件，将 0xE5 设置为 DIR_Name[0]以释放条目。如果文件有簇链，还需要从 FAT 中删除该链。

删除子目录

删除子目录与删除文件相同。在删除目录之前，需要扫描目录下方的所有节点，并删除目录中的所有文件和子目录，否则这些对象的簇将丢失。

卷标

FAT 卷可以有一个称为卷标的名称，该名称作为具有 ATTR_VOLUME_ID 属性的目录条目记录在根目录中。卷标不是一个文件，而只是卷的名称。它的命名空间独立于文件，并且该名称可以与目录中的任何其他文件重复。卷标的可允许字符类似于 SFN 条目，但它可以在名称中的任何位置包含空格，并且不能包含点。

卷标不适用于 LFN 扩展。对卷标进行任何更改时，应该反映到 BS_VolLab 中，但 Windows 不这样做。Windows 在以 0xE5 开头的卷标的行为上存在问题。它不将其替换为 0x05，并且更改将没有效果，因此不应使用此类卷标。

3.12 时间戳

目录条目中有一些与时间和日期相关的字段。大多数 FAT 驱动程序仅支持必须支持的 DIR_WrtTime 和 DIR_WrtDate 字段，不支持可选字段。不支持的字段在创建条目时应初始化为零，并且以后不应更改。时间和日期的格式如下所述：

字段名	位字段
DIR_WrtDate DIR_CrtDate DIR_LstAccDate	位 15-9: 从 1980 年开始计算的年数，范围从 0 到 127（1980-2107）。位 8-5: 月份，范围从 1 到 12。位 4-0: 日期，范围从 1 到 31。
DIR_WrtTime DIR_CrtTime	位 15-11: 小时，范围从 0 到 23。位 10-5: 分钟，范围从 0 到 59。位 4-0: 以 2 秒为单位的计数，范围从 0 到 29（0-58 秒）。

请注意，这些时间和日期字段的位表示适用于 DIR_WrtDate、DIR_CrtDate 和 DIR_LstAccDate 字段，以及 DIR_WrtTime 和 DIR_CrtTime 字段。

3.15 长文件名（LFN）

当将长文件名（LFN）作为 FAT 文件系统的新功能添加时，需要与现有系统实现向后兼容。以下是所需的具体示例：

- LFN 的存在在现有系统上需要是不可见的，特别是在 MS-DOS 和 Windows 上的任何文件 API 上。

- LFN 需要物理上靠近对应文件的目录条目，以防止对性能产生不良影响。
- 如果磁盘工具在 FAT 卷中的某个地方发现了 LFN 信息的记录，文件系统需要保持完整性并且不受影响。

为了实现这些要求，LFN 信息被记录为带有特殊属性的目录条目。如上所述，LFN 条目的属性（ATTR_LONG_NAME）是由现有属性位（ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_SYSTEM | ATTR_VOLUME_ID）组合定义的，并且还定义了掩码值（ATTR_LONG_NAME_MASK = ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_SYSTEM | ATTR_VOLUME_ID | ATTR_DIRECTORY | ATTR_ARCHIVE）。当 DIR_Attr 与（ATTR_LONG_NAME_MASK 匹配 ATTR_LONG_NAME）进行按位掩码操作时，该条目就是一个 LFN 条目，其字段定义如下。

LFN 的目录条目结构

字段名	偏移量	大小	描述
LDIR_Ord	0	1	序列号（1-20），用于标识此条目在组成 LFN 的 LFN 条目序列中的位置。值为 1 表示 LFN 的顶部部分，带有 LAST_LONG_ENTRY 标志（0x40）的任何值表示 LFN 的最后部分。
LDIR_Name1	1	10	LFN 的一部分，从第 1 个字符到第 5 个字符。
LDIR_Attr	11	1	LFN 的属性。始终为 ATTR_LONG_NAME，表示这是一个 LFN 条目。
LDIR_Type	12	1	必须为零。
LDIR_Chksum	13	1	与此条目相关联的 SFN 条目的校验和。
LDIR_Name2	14	12	LFN 的一部分，从第 6 个字符到第 11 个字符。
LDIR_FstClusLO	26	2	必须为零，以避免旧磁盘工具进行错误修复。
LDIR_Name3	28	4	LFN 的一部分，从第 12 个字符到第 13 个字符。

LFN 条目始终与相应的 SFN 条目关联，以便将 LFN 添加到文件中。LFN 条目从不独立存在，每个文件只有 SFN 或同时具有 SFN 和 LFN。LFN 条目只包含名称信息，不包含有关文件的其他内容。如果存在没有与 SFN 条目关联的 LFN 条目，则该孤立的 LFN 条目是无效的并被视为垃圾。这是为了与旧系统保持向后兼容性。如果文件具有 LFN 条目，LFN 将成为文件的主要名称，而 SFN 将作为备用名称。不支持 LFN 的旧系统不会识别 LFN 条目，但可以使用 SFN 访问文件。LFN 系统可以使用 LFN 或 SFN 访问文件。下表显示了如何在目录中记录 LFN 和 SFN 的集合。

与文件关联的 LFN "MultiMediaCard System Summary.pdf" 的关联情况

位置	第一个字节	名称字段	属性	内容
DIR[N-3]	0x43	ary.pdf	--VSHR	LFN 的第 3 部分 (lfn[26..38])
DIR[N-2]	0x02	d System Summ	--VSHR	LFN 的第 2 部分 (lfn[13..25])
DIR[N-1]	0x01	MultiMediaCar	--VSHR	LFN 的第 1 部分 (lfn[0..12])
DIR[N]	'M'	MULTIM~1PDF	A-----	相关的 SFN 条目

如果长文件名（LFN）超过 13 个字符，它会被分成多个 LFN 条目。LFN 的最大名称长度为 255 个字符，因此一个 LFN 可以占用多达 20 个 LFN 条目。LFN 放置在与相关 SFN 条目紧挨着的目录中。对于上面的示例，长度为 33 个字符的 LFN 由 3 个 LFN 条目组成，它们的 LDIR_Ord 字段分别为 0x43、0x02、0x01。LFN 使用的字符编码是 UTF-16 编码的 Unicode，而 SFN 的字符编码是取决于系统的本地代码页中的 ANSI/OEM 编码。如果最后一部分不足 13 个字符，将以空字符（U+0000）结束，并且名称字段的其余部分必须填充为 U+FFFF。LDIR_Ord 必须从 1 开始，并按降序记录。LFN+SFN 的块在连续的条目中记录。如果 LFN 条目的任何条件不满足，LFN 将不再有效。

此外，使用校验和来确保 LFN 和 SFN 之间的关联性。每个 LFN 条目在 LDIR_Chksum 中具有与其关联的 SFN 的校验和。校验和是根据以下算法生成的。

```
uint8_t create_sum(const DIR* entry)
{
    int i;
    uint8_t sum;
    for (i = sum = 0; i < 11; i++) { /* 计算DIR_Name[]字段的和 */
        sum = (sum >> 1) + (sum << 7) + entry->DIR_Name[i];
    }
    return sum;
}
```

如果 LFN 条目中的任何校验和不匹配，LFN 将无效。这是为了防止非 LFN 系统对目录进行更改（删除、创建或重命名）而导致错误的关联。然而，迷失的 LFN 条目仍然会占用目录，使磁盘使用量变得更糟。这将成为固定长度目录（FAT12/16 卷上的根目录）的问题。这些垃圾条目会被磁盘工具删除。

3.16 命名空间

短文件名 SFN

SFN，通常称为 8.3 格式文件名，是最初在 MS-DOS 上使用的一种传统风格的文件名格式，由主体部分（1-8 个字符）和可选的扩展名部分（1-3 个字符）组成。这两个部分用点号（.）分隔。SFN 允许的字符包括 ASCII 字母数字字符、一些 ASCII 标点符号（\$%'-_@~`!(){}^#&）和扩展字符（\x80 - \xFF）。

SFN 以 OEM 字符集（在 MS-DOS 上使用的字符集）的形式存储，取决于系统的区域设置。文件名中的小写字符会被转换为大写字符，然后存储和匹配，因此 SFN 的大小写信息会丢失。

长文件名 LFN

LFN 的长度可以达到 255 个字符。LFN 允许的字符包括空格和一些 ASCII 标点符号（+, ;=[]），除此之外还包括 SFN 字符。文件名中的点号可以嵌入在任何位置，但末尾的点号和空格被视为名称的结束，并在文件 API 中被截断。前导的空格和点号是有效的，但某些用户界面，如 Windows 公共对话框，会拒绝此类文件名。

LFN 以不进行大写转换的形式存储在 LFN 条目中。LFN 使用的字符编码为 Unicode。

由于 SFN（OEM 字符集）和 LFN（Unicode）使用不同的字符编码，通用的实现需要进行编码转换。如果 OEM 代码是单字节代码，则没有问题，但是当 OEM 代码是双字节字符集（DBCS）时，需要一个庞大的（几百 KB）转换表，因此在内存有限的小型嵌入式系统中实现 LFN 是困难的。

3.17 名称匹配

在目录中，每个文件名都是唯一的，不论是 LFN 还是 SFN，它都不会与任何其他名称匹配。LFN 可以包含小写字符，并且在不区分大小写的情况下进行匹配，因此这三个名称"LongFileName.Txt"、"longfilename.txt"、"LONGFILENAME.TXT"被视为相同的名称。因此，在目录搜索中，名称匹配总是不区分大小写进行的。

当在 8.3 格式中找到一个输入文件名时，会比较目录中文件的 LFN 和 SFN。当文件名不符合 8.3 格式时，只会比较 LFN。

当列出目录中的文件名时，只会输出 LFN，除非指定了 SFN。如果文件没有 LFN，或者 LFN 中的任何字符无法转换为 OEM 代码（当 API 使用 OEM 代码时），则会输出 SFN。

3.18 物理驱动器分区

这不在 FAT 文件系统的范围内。然而，这是关于磁盘使用的通用知识，每个人在使用嵌入式系统中的存储设备时都需要了解。

为了有效地使用硬盘驱动器的大容量空间，通常在物理驱动器上使用多个分区。例如，一个 100 GB 的硬盘分为 3 个分区，10、30 和 60 GB，并创建用于系统、数据和缓存的卷。在通用 Windows PC 上，硬盘上会存在两个分区，一个用于系统，另一个用于恢复。

有两种分区规则，MBR 格式（也称为 FDISK 格式）和 SFD 格式（Super-floppy Disk）。MBR 格式通常用于硬盘和存储卡。它可以在 MBR 上（物理驱动器的 LBA 0，即主引导记录）使用分区表将物理驱动器划分为一个或多个分区。SFD 格式是非分区磁盘格式。FAT 卷从物理驱动器的 LBA 0 开始，没有任何磁盘分区。它通常用于软盘、光盘和大多数类型的超级软盘介质。

某些系统和介质类型的组合仅支持其中一种格式，而不支持另一种格式。Windows 操作系统不支持可移动驱动器上的第二个分区和硬盘上的 SFD 格式（前者从 Windows 10 1703 开始支持）。

MBR and 分区表

字段名称	偏 移 量	大小	描述
MBR_bootcode	0	446	引导程序。根据系统而定。未使用时填充为零。
MBR_Partation1	446	16	分区表条目 1。指示分区类型和状态。
MBR_Partation2	462	16	分区表条目 2。
MBR_Partation3	478	16	分区表条目 3。
MBR_Partation4	494	16	分区表条目 4。
MBR_Sig	510	2	0xAA55。指示这是有效的 MBR（主引导记录）。

下表显示了分区表条目的字段，MBR 中最多可以记录四个条目。这意味着一个存储设备可以分为四个分区。有一种分区类型可以包含其他分区，但有关其详细信息，请参考其他资料。

分区表表项

字段名	偏 移 量	大小	描述
PT_BootID	0	1	引导指示符。非引导分区（0x00）或引导分区（0x80）。引导分区表示系统从该分区引导，但这与具体的系统相关。只能设置一个分区为引导分区。

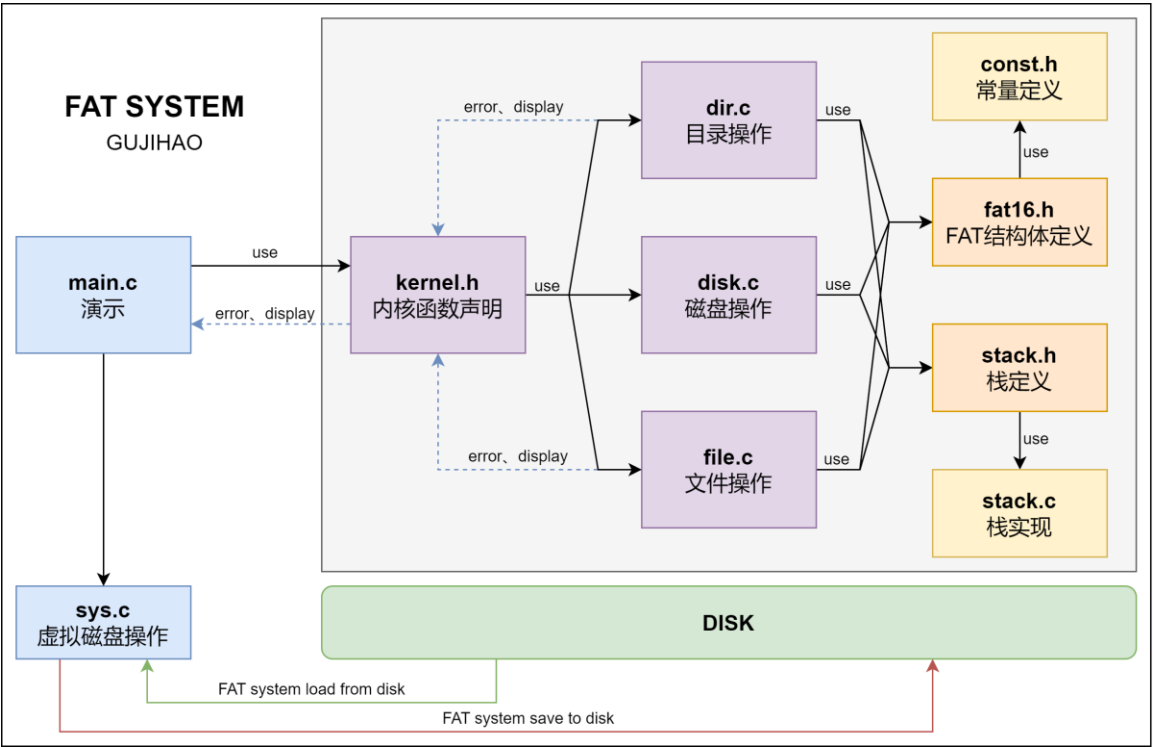
PT_StartHd	1	1	分区起始扇区的磁头号（CHS 格式，0 - 254）。
PT_StartCySc	2	2	分区起始扇区的柱面号（位 9-0：0-1023）和柱面内的扇区号（位 15-10：1-63）（CHS 格式）。
PT_System	4	1	分区类型。常见的取值有：0x00：空白条目。其他字段必须为零。0x01：FAT12（CHS/LBA，扇区数<65536）0x04：FAT16（CHS/LBA，扇区数<65536）0x05：扩展分区（CHS/LBA）0x06：FAT12/16（CHS/LBA，扇区数>=65536）0x07：HPFS/NTFS/exFAT（CHS/LBA）0x0B：FAT32（CHS/LBA）0x0C：FAT32（LBA）0x0E：FAT12/16（LBA）0x0F：扩展分区（LBA）
PT_EndHd	1	1	分区结束扇区的磁头号（CHS 格式，0 - 254）。
PT_EndCySc	2	2	分区结束扇区的柱面号（位 9-0：0-1023）和柱面内的扇区号（位 15-10：1-63）（CHS 格式）。
PT_LbaOfs	8	4	分区起始扇区的 32 位逻辑块地址（32 位 LBA，1 - 0xFFFFFFFF）。
PT_LbaSize	12	4	分区大小（以扇区为单位，1 - 0xFFFFFFFF）。

每个分区占据驱动器的一部分，彼此之间没有重叠，而分区的第一个扇区是 **VBR**（卷引导记录）。在大多数情况下，只使用第一个条目，其余的条目为空白。

有两种格式用于表示分区的分配情况，即 **CHS** 格式和 **LBA** 格式。这两个参数都存储在分区表条目中。**CHS** 字段用于具有几何结构的驱动器，但实际上这取决于系统。**LBA** 字段用于以 **LBA**（逻辑块寻址）方式控制的驱动器。如果分区重叠了无法用 **CHS** 格式表示的区域（8 GB 及以上），**CHS** 字段将不再有效，只能使用 **LBA** 字段。

4 系统结构和主要的算法设计思路

描述所设计系统的系统架构，主要的算法思想、流程图等。



main.c

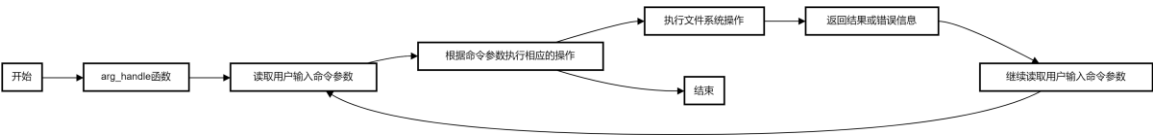
系统架构:

- 该文件系统由多个源文件组成，包括"kernel.h"和"fat16.h"等。
- 主要功能由 `arg_handle` 函数实现，该函数根据用户输入的命令参数来执行相应的文件系统操作。

算法思想:

- 系统通过读取用户输入的命令参数，根据参数类型来执行相应的操作。
- 使用条件判断和循环结构来处理不同的命令和参数。
- 调用相应的文件系统函数来执行对文件和目录的操作，如创建文件、删除文件、读取文件、更改当前目录等。

流程图:



sys.c

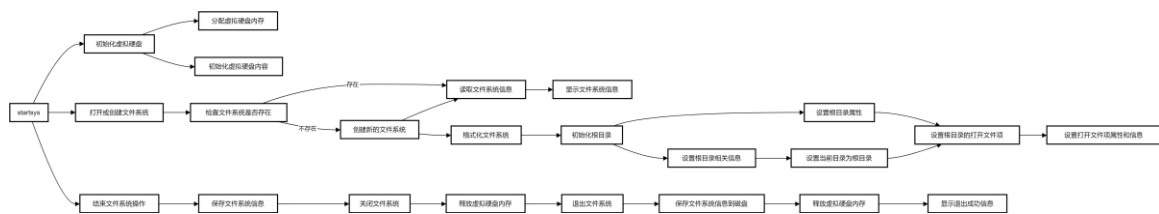
系统架构:

- 该文件系统由多个源文件组成, 包括"kernel.h"和"fat16.h"等。
- 主要功能由 startsys 和 existsys 函数实现, 其中 startsys 用于启动文件系统, existsys 用于退出文件系统。

算法思想:

- startsys 函数通过读取文件系统的数据结构和元数据来初始化系统状态, 包括虚拟硬盘的分配和读取、文件控制块的设置等。
- existsys 函数用于保存虚拟硬盘的数据, 并释放相关的内存资源。

流程图:



kernel.h

系统架构:

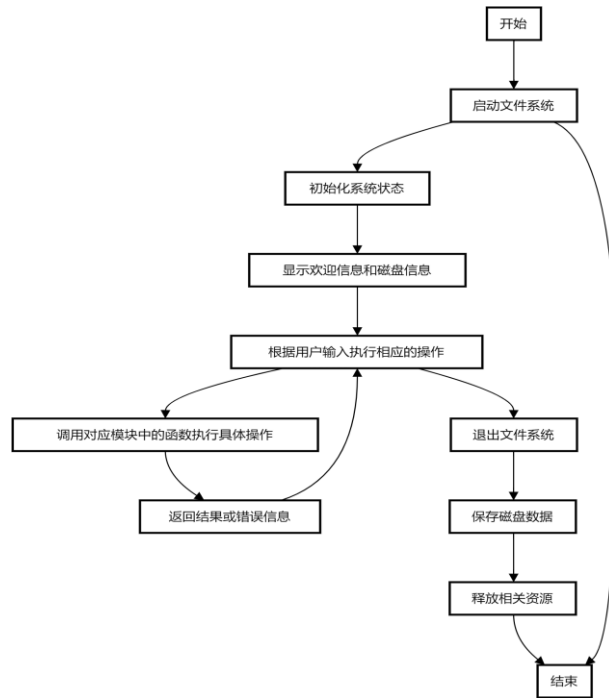
- 该文件系统包含了多个模块, 分别对应不同的功能, 例如系统函数、磁盘操作、目录操作和文件操作等。
- 系统函数模块定义了系统的启动和退出函数, 以及显示信息和错误处理的函数。
- 磁盘操作模块包含格式化磁盘和显示磁盘使用情况的函数。
- 目录操作模块定义了创建目录、改变当前目录、删除目录和列出目录内容的函数。
- 文件操作模块定义了创建文件、删除文件、打开文件、关闭文件、读取文件和写入文件的函数。
- 还有一些其他功能的函数, 如交叉检查文件、更改文件的最后一个块等。

算法思想:

- 系统通过调用不同模块中的函数来实现不同的功能, 通过函数参数传递参数信息和数据。

- 不同的函数根据具体的操作需求使用不同的算法思想，例如目录操作需要遍历目录结构，文件读写需要使用指针和缓冲区等。
- 系统函数提供了显示信息和错误处理的功能，用于向用户提供反馈和处理异常情况。

流程图:



dir.c

系统架构:

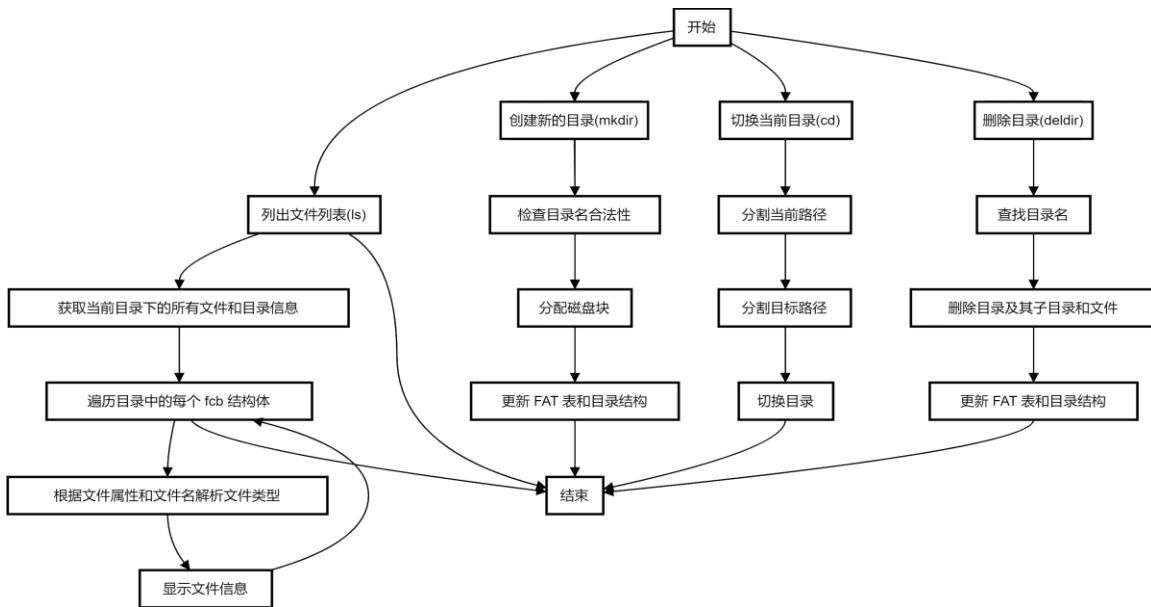
- 该文件系统包含了多个模块，如系统函数、磁盘操作、目录操作和文件操作等。
- 系统函数模块定义了系统的启动和退出函数，以及显示信息和错误处理的函数。
- 磁盘操作模块包含了格式化磁盘和读写磁盘数据的函数。
- 目录操作模块定义了创建目录、切换目录和删除目录等函数。
- 文件操作模块定义了创建文件、删除文件和读写文件等函数。

算法思想:

- `ls()` 函数用于列出当前目录的文件和子目录信息。它遍历当前目录的所有 `fc` 结构体，并根据文件属性和文件名解析文件类型。
- `mkdir()` 函数用于创建新的目录。它检查目录名的合法性，分配磁盘块，更新 FAT 表和目录结构。

- `cd()` 函数用于切换当前目录。它将当前路径分割为目录数组，并根据指定的目录名遍历目录结构，找到目标目录的位置。
- `deldir()` 函数用于删除目录。它在目录结构中查找指定的目录名，并删除该目录及其所有子目录和文件。

流程图:



disk.c

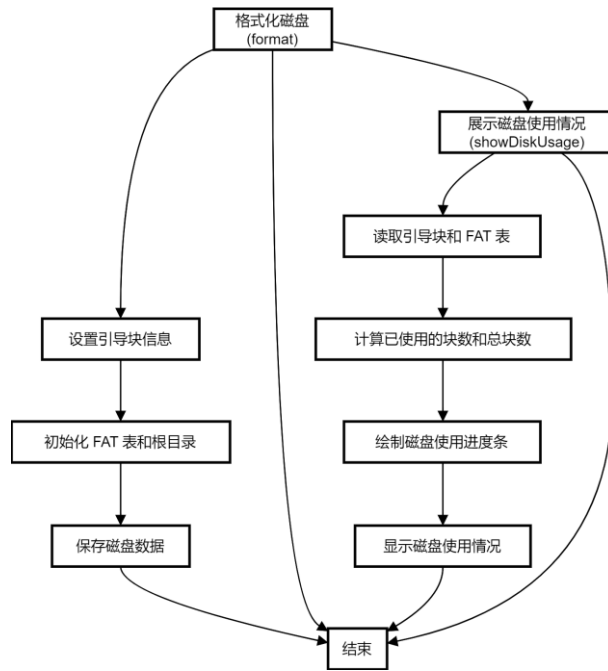
系统架构:

- 该文件系统包含了多个模块，如系统函数、磁盘操作、格式化和磁盘使用情况展示等。
- 系统函数模块定义了系统的启动和退出函数，以及显示信息和错误处理的函数。
- 磁盘操作模块包含了读写磁盘数据的函数。
- 格式化模块定义了格式化磁盘的函数。
- 磁盘使用情况展示模块用于展示磁盘的使用情况，包括已使用块数、总块数和磁盘使用进度。

算法思想:

- `format()` 函数用于格式化磁盘。它根据指定的块大小、磁盘总大小和最大打开文件数等参数，设置引导块的信息，并初始化 FAT 表和根目录。
- `showDiskUsage()` 函数用于展示磁盘的使用情况。它读取磁盘上的引导块和 FAT 表，计算已使用的块数和总块数，然后根据计算结果绘制磁盘使用进度条。

流程图:



file.c

系统架构:

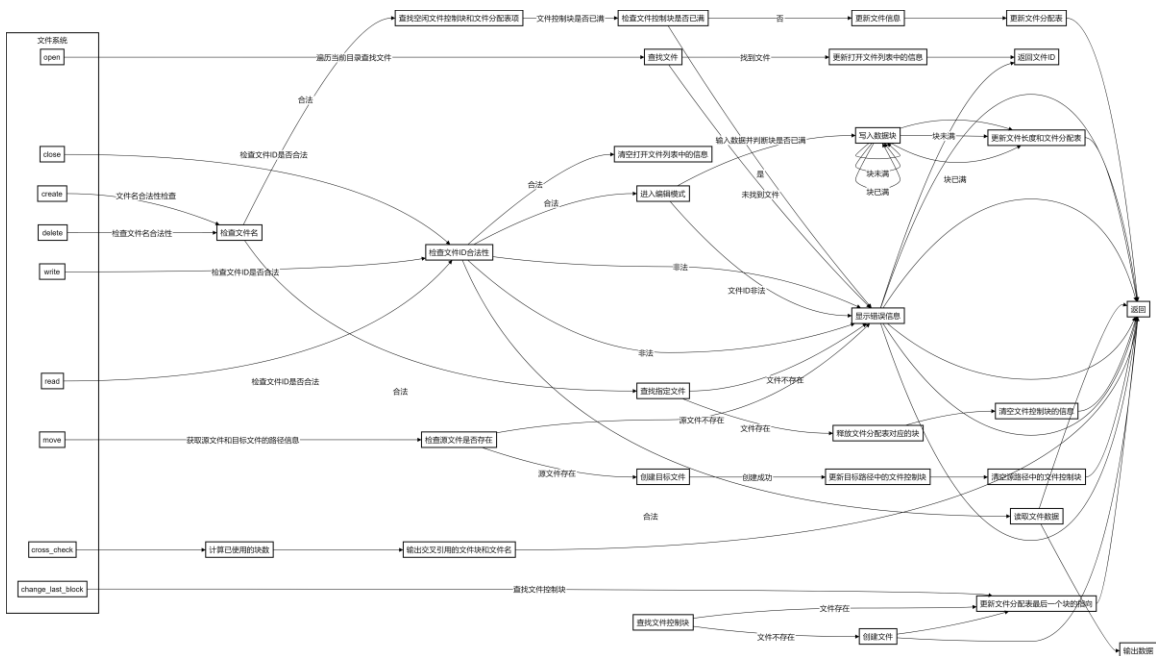
- 该系统使用了 C 语言编写，实现了一个简单的文件系统。
- 系统中的主要组件包括内核(kernel)、FAT16 文件系统(fat16)和堆栈(stack)。
- 内核部分提供了文件系统的基本功能，包括文件的创建、打开、关闭、写入和读取等操作。
- FAT16 文件系统部分实现了 FAT16 文件系统的的数据结构和操作方法，包括文件控制块(fcb)和文件分配表(fat)的定义和管理。
- 堆栈部分提供了一些辅助函数，用于处理堆栈的操作。

算法思想:

- 系统的核心算法思想是基于 FAT16 文件系统，通过管理文件控制块和文件分配表来实现文件的创建、打开、写入和读取等操作。
- 文件的创建(create)函数通过检查文件名的合法性，查找空闲的文件控制块和文件分配表项，创建新的文件，并更新相关信息。
- 文件的打开(open)函数通过遍历当前目录查找指定的文件，如果找到了对应的文件，则将文件的相关信息存储到打开文件列表(openfilelist)中，方便后续操作。

- 文件的关闭(close)函数用于关闭已打开的文件，将打开文件列表中对应文件的信息清空。
- 文件的写入(write)函数用于向已打开的文件中写入数据。根据输入的数据长度，判断是否需要分多个块进行写入，并更新文件长度和文件分配表。
- 文件的读取(read)函数用于从已打开的文件中读取数据。根据文件长度判断是否需要分多个块进行读取，并输出读取的数据。
- 文件的删除(delfile)函数用于删除指定的文件。通过查找文件控制块，释放文件分配表对应的块，并清空文件控制块的信息。
- 文件的移动(move)函数用于将文件从源路径移动到目标路径。通过获取源文件和目标文件的路径信息，切换到目标路径并检查目标路径的合法性，然后更新目标路径中的文件控制块，并清空源路径中的文件控制块。
- 文件的交叉检查(cross_check)函数用于检查文件分配表中的交叉引用情况，输出交叉引用的文件块和文件名。
- 文件的修改最后一个块(change_last_block)函数用于修改指定文件的最后一个文件分配表项，用于在写入数据时定位到文件的末尾块。

流程图:



fat16.h

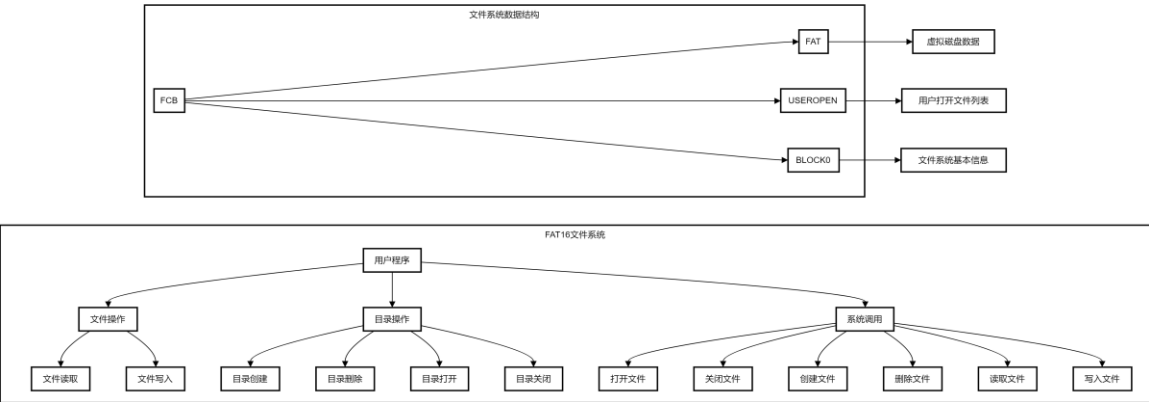
系统架构:

- 文件系统由文件控制块（FCB）、文件分配表（FAT）、用户打开表（USEROPEN）和引导块（BLOCK0）等数据结构组成。
- 文件控制块（FCB）用于描述文件的属性和信息，包括文件名、属性、时间、日期、起始磁盘块号和文件长度等。
- 文件分配表（FAT）用于记录文件在磁盘上的存储情况，每个 FAT 表项表示一个磁盘块的状态。
- 用户打开表（USEROPEN）维护了用户打开的文件的信息，包括文件名、属性、时间、日期、起始磁盘块号、文件长度等，并提供了对文件的读写操作。
- 引导块（BLOCK0）记录了文件系统的一些基本信息，如文件系统名称、版本号、块大小、磁盘大小、最大打开文件数和根目录磁盘块号等。

算法思想:

- 文件创建（create）操作：检查文件名的合法性，查找空闲的文件控制块和文件分配表项，更新文件信息，并更新文件分配表。
- 文件打开（open）操作：遍历当前目录查找文件，更新打开文件列表中的信息，返回文件 ID。
- 文件关闭（close）操作：检查文件 ID 的合法性，清空打开文件列表中的信息。
- 文件写入（write）操作：检查文件 ID 的合法性，进入编辑模式，根据块的状态写入数据块，并更新文件长度和文件分配表。
- 文件读取（read）操作：检查文件 ID 的合法性，读取文件数据并输出。
- 文件删除（delete）操作：检查文件名的合法性，查找指定文件，释放文件分配表对应的块，并清空文件控制块的信息。
- 文件移动（move）操作：获取源文件和目标文件的路径信息，检查源文件是否存在，创建目标文件并更新目标路径中的文件控制块，清空源路径中的文件控制块。
- 交叉引用（cross_check）操作：计算已使用的块数，并输出交叉引用的文件块和文件名。
- 更新最后一个块（change_last_block）操作：根据文件是否存在，更新文件分配表最后一个块的指向。

流程图:



stack.h & stack.c

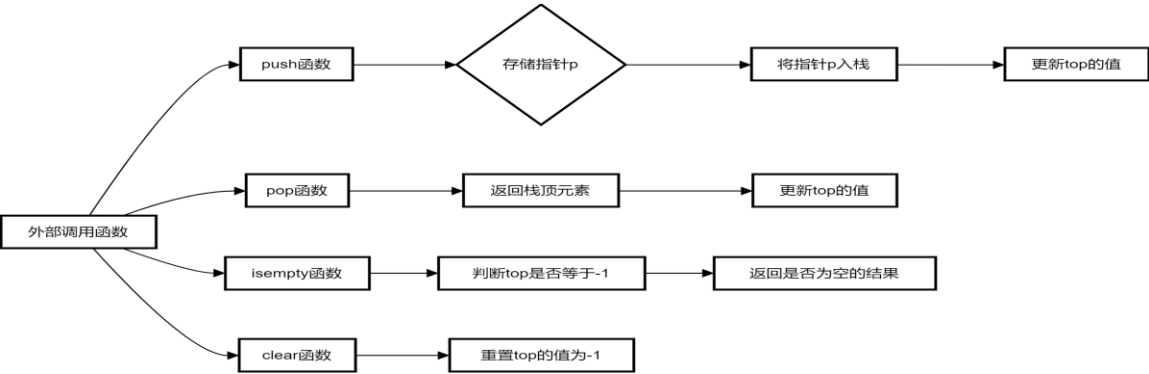
系统架构:

- 栈 (Stack) 是一种数据结构，遵循"先进后出" (Last-In-First-Out, LIFO) 的原则。它包含以下操作：push (入栈)、pop (出栈)、isempty (判断栈是否为空) 和 clear (清空栈)。
- 该系统的架构是一个基于数组的栈实现。数组 stack 用于存储栈中的元素，top 表示栈顶的索引位置。

主要算法思想:

- push 函数：将传入的指针 p 入栈，即将其放入数组 stack 中的下一个位置，并更新 top 的值。
- pop 函数：返回栈顶元素，并将 top 的值减一，表示栈顶向下移动一个位置。
- isempty 函数：检查栈是否为空，即判断 top 是否等于-1。如果是，则栈为空；否则，栈不为空。
- clear 函数：将 top 的值重置为-1，表示清空栈。

流程图:



const.h

常量定义:

- BLOCKSIZE: 磁盘块的大小, 为 1024 字节。
- VERSION: 系统版本号, 为 0.01。
- SIZE: 虚拟磁盘的大小, 为 1024000 字节。
- END: FAT 中表示文件结束的标志, 为 65535。
- FREE: 磁盘块空闲的标志, 为 0。
- ROOTBLOCKNUM: 根目录块的数量, 为 2。
- MAXOPENFILE: 同时打开的文件数的最大限制, 为 10。
- ARG_CMD_LENGTH: 命令参数的最大长度, 为 1024。
- PROGRESS_BAR_WIDTH: 进度条的长度, 为 20。

错误码定义:

- ERR1: 输入的目录名为空的错误码。
- ERR2: 目录名或文件名包含不允许的字符的错误码。
- ERR3: 已经存在同名的文件或目录的错误码, 需要提供新的名称。
- ERR4: 目录不存在的错误码。
- ERR5: 输入的目录名为空的错误码。
- ERR6: 修改 FAT 表时发生错误的错误码。
- ERR7: 打开文件时发生错误的错误码。
- ERR8: 当前目录已满, 无法创建新文件的错误码。
- ERR9: 文件不存在的错误码。
- ERR10: 非法的文件 ID 的错误码。
- ERR11: 尚未打开具有指定文件 ID 的文件的错误码。
- ERR12: 没有已打开的文件或已打开的文件无法写入的错误码。
- ERR13: 写文件错误的错误码。
- ERR14: FAT 表错误的错误码。

- ERR15: 文件资源 ID 错误的错误码, 可能是尚未打开它。
- ERR16: 没有已打开的文件或已打开的文件无法写入的错误码。
- ERR17: 输入的文件名为空的错误码。
- ERR18: 源文件不存在的错误码。
- ERR19: 目标目录不存在的错误码。
- ERR20: 目标目录已满, 无法移动文件的错误码。
- ERR21: 打开文件时发生错误的错误码。
- ERR22: 命令错误的错误码。

5 程序实现---主要数据结构

描述所设计系统的关键数据结构。

关键数据结构包括:

1. **BootSector** (引导扇区): 表示文件系统的引导扇区, 占据扇区 0。它包含了文件系统的各种参数和信息。

```
typedef struct {
    // offset 0~35 扇区编号 0
    unsigned char BS_JmpBoot[3];           // 0 3字节 跳转指令
    char BS_OEMName[8];                    // 3 8字节 OEM名称
    unsigned short BPB_BytsPerSec;         // 11 2字节 每个扇区的字节数
    unsigned char BPB_SecPerClus;         // 13 1字节 每簇的扇区数
    unsigned short BPB_RsvdSecCnt;        // 14 2字节 保留扇区数 (2字节)
    unsigned char BPB_NumFATs;            // 16 1字节 FAT表的数量
    unsigned short BPB_RootEntCnt;        // 17 2字节 根目录区的目录项数
    unsigned short BPB_TotSec16;          // 19 2字节 总扇区数
    unsigned char BPB_Media;              // 21 1字节 媒体类型
    unsigned short BPB_FATSz16;           // 22 2字节 每个FAT表的扇区数
    unsigned short BPB_SecPerTrk;        // 24 2字节 每个磁道的扇区数
    unsigned short BPB_NumHeads;         // 26 2字节 磁头数
    unsigned int BPB_HiddSec;             // 28 4字节 隐藏扇区数
    unsigned int BPB_TotSec32;           // 32 4字节 总扇区数
    // offset 36~511
    unsigned int BPB_FATSz32;             // 36 4字节 FAT的扇区数
    unsigned short BPB_ExtFlags;          // 40 2字节 扩展标志
    unsigned short BPB_FSVer;            // 42 2字节 FAT32版本
    unsigned int BPB_RootClus;           // 44 4字节 根目录的第一个簇号
    unsigned short BPB_FSInfo;           // 48 2字节 FSInfo结构所在扇区的偏移量
    unsigned short BPB_BkBootSec;        // 50 2字节 备份引导扇区所在扇区的偏移量
    unsigned char BPB_Reserved[12];      // 52 12字节 保留字段
    unsigned char BS_DrvNum;             // 64 1字节 磁盘驱动器编号
    unsigned char BS_Reserved;           // 65 1字节 保留字段
    unsigned char BS_BootSig;            // 66 1字节 扩展引导标志
    unsigned int BS_VolID;               // 67 4字节 卷序列号
    char BS_VolLab[11];                 // 71 11字节 卷标
    char BS_FilSysType[8];               // 82 8字节 文件系统类型
    unsigned char BS_BootCode32[420];    // 90 420字节 引导程序
    unsigned short BS_BootSign;          // 510 2字节 引导标记 0xAA55
} BootSector; // 1扇区
```

2. **FSInfoSector** (FSInfo 扇区): 表示文件系统的 FSInfo 扇区, 占据扇区 1。它包含了一些文件系统信息和状态。

```
typedef struct {
    // offset 512~1023 扇区编号 1
    unsigned int FSI_LeadSig;             // 0 4字节 FSIInfo主要签名
    unsigned char FSI_Reserved1[480];     // 4 480字节 保留字段
    unsigned int FSI_StrucSig;            // 484 4字节 FSIInfo结构签名
    unsigned int FSI_Free_Count;          // 488 4字节 上次已知的空闲簇计数
    unsigned int FSI_Nxt_Free;           // 492 4字节 提示FAT驱动程序开始查找空闲簇的簇号
    unsigned char FSI_Reserved2[12];     // 496 12字节 保留字段
    unsigned int FSI_TailSig;            // 508 4字节 FSIInfo尾部签名
} FSInfoSector; // 引导扇区 占1扇区
```

3. **BackupBootSector**（备份引导扇区）：表示备份的引导扇区，占据扇区 2 到扇区 7。用于存储备份的引导扇区数据。

```
typedef struct {  
    // offset 1024~4095 扇区编号 2~7  
    unsigned char Reserved1[512]; // 0 512字节 保留字段  
    unsigned char Reserved2[512]; // 512 512字节 保留字段  
    unsigned char Reserved3[512]; // 1024 512字节 保留字段  
    unsigned char Reserved4[512]; // 1536 512字节 保留字段  
    unsigned char Reserved5[512]; // 2048 512字节 保留字段  
    unsigned char Reserved6[512]; // 2560 512字节 保留字段  
} BackupBootSector; // FSInfo扇区 占6扇区
```

4. **ReservedSectors**（保留扇区）：表示保留扇区，占据扇区 8 到扇区 31。用于保留字段的存储。

```
typedef struct {  
    // offset 4096~16383 扇区编号 8~31  
    unsigned char Reserved1[512]; // 0 512字节 保留字段  
    unsigned char Reserved2[512]; // 512 512字节 保留字段  
    unsigned char Reserved3[512]; // 1024 512字节 保留字段  
    unsigned char Reserved4[512]; // 1536 512字节 保留字段  
    unsigned char Reserved5[512]; // 2048 512字节 保留字段  
    unsigned char Reserved6[512]; // 2560 512字节 保留字段  
    unsigned char Reserved7[512]; // 3072 512字节 保留字段  
    unsigned char Reserved8[512]; // 3584 512字节 保留字段  
    unsigned char Reserved9[512]; // 4096 512字节 保留字段  
    unsigned char Reserved10[512]; // 4608 512字节 保留字段  
    unsigned char Reserved11[512]; // 5120 512字节 保留字段  
    unsigned char Reserved12[512]; // 5632 512字节 保留字段  
    unsigned char Reserved13[512]; // 6144 512字节 保留字段  
    unsigned char Reserved14[512]; // 6656 512字节 保留字段  
    unsigned char Reserved15[512]; // 7168 512字节 保留字段  
    unsigned char Reserved16[512]; // 7680 512字节 保留字段  
    unsigned char Reserved17[512]; // 8192 512字节 保留字段  
    unsigned char Reserved18[512]; // 8704 512字节 保留字段  
    unsigned char Reserved19[512]; // 9216 512字节 保留字段  
    unsigned char Reserved20[512]; // 9728 512字节 保留字段  
    unsigned char Reserved21[512]; // 10240 512字节 保留字段  
    unsigned char Reserved22[512]; // 10752 512字节 保留字段  
    unsigned char Reserved23[512]; // 11264 512字节 保留字段  
    unsigned char Reserved24[512]; // 11776 512字节 保留字段  
} ReservedSectors; // 备份引导扇区 占24扇区
```

5. **FATEntry**（FAT 表项）：表示 FAT 表中的一个条目，用于记录簇号与数据区的对应关系。

```
// FAT[0]和FAT[1]是保留的，并且不与任何簇相关联  
// FAT[0] = 0xFFFFFFFF?; FAT[1] = 0xFFFFFFFF;  
// FAT[0]中的??的值与BPB_Media的值相同，但条目没有任何功能。FAT[1]中的某些位记录错误历史。  
// 第三个FAT项，FAT[2]，是与数据区的第一个簇相关联的项，有效的簇号从2开始  
typedef unsigned int FATEntry;
```

6. **DirectoryEntry**（目录项）：表示文件或目录的目录项。包含文件名、属性、大小等信息。

```
typedef struct {  
    // DIR_Name[0]，是一个重要的数据，用于指示目录项的状态。  
    // 当值为0xE5时，表示该条目未使用（可供新分配）。  
    // 当值为0x00时，表示该条目未使用（与0xE5相同），并且在此之后没有分配的条目（所有后续条目的DIR_Name[0]也都设置为0）  
    char DIR_Name[11]; // 0 11 bytes 文件名  
    unsigned char DIR_Attr; // 11 1 byte 文件属性  
    unsigned char DIR_NTRes; // 12 1 byte 可选的大小写信息标志  
    unsigned char DIR_CrtTimeTenth; // 13 1 byte 可选的文件创建时间的子秒信息  
    unsigned short DIR_CrtTime; // 14 2 bytes 可选的文件创建时间  
    unsigned short DIR_CrtDate; // 16 2 bytes 可选的文件创建日期  
    unsigned short DIR_LstAccDate; // 18 2 bytes 可选的最后访问日期  
    unsigned short DIR_FstClusHI; // 20 2 bytes 簇号高16位  
    unsigned short DIR_WrtTime; // 22 2 bytes 最后修改时间  
    unsigned short DIR_WrtDate; // 24 2 bytes 最后修改日期  
    unsigned short DIR_FstClusLO; // 26 2 bytes 簇号低16位  
    unsigned int DIR_FileSize; // 28 4 bytes 文件大小（单位字节），目录始终为0  
} DirectoryEntry;
```

6 程序实现---主要程序清单

dir.c

函数	功能
ls()	列出当前目录下的文件和目录。
mkdir(char *dirname)	创建具有指定名称的新目录。
cd(char *dirname)	将当前目录更改为指定的目录。
deldir(char *dirname)	删除指定的目录。
main()	主程序，作为系统的入口点。
void ls()	显示当前目录中文件和目录的信息。
void mkdir(char *dirname)	使用给定名称创建新目录。
void cd(char *dirname)	将当前目录更改为指定的目录。
void deldir(char *dirname)	删除指定的目录。

disk.c

函数	功能
void format()	格式化虚拟硬盘，创建文件系统的初始结构
void showDiskUsage()	显示虚拟硬盘的使用情况，包括已使用块数、总块数和使用百分比

file.c

函数	功能
int create(char *filename)	创建文件
int open(char *filename)	打开文件
void close(int fid)	关闭文件
int write(int fid)	写入文件
int dwrite(int fid, char *text, int len, char wstyle)	执行写入操作
int read(int fid)	读取文件
void delfile(char *filename)	删除文件
void move(char *src, char *dest)	将源文件移动到目标位置
void cross_check()	检查文件系统中的交叉引用和重复文件
void change_last_block(char *filename, unsigned short id)	更改指定文件的最后一个块的标识符，即模拟病毒文件

7 程序运行的主要界面和结果截图

列出系统运行的主要界面和运行结果截图。

7.1 运行结果

```
Welcome to FAT32 file system
version:0.01
block size:1024
total size:1024000
max opened files:10
root block location:5

FAT32 ~/root$ |
```

help

```
FAT32 ~/root$ help
zzycami file system
version: 0.01
Useage:[order] --[option]

ls      --list-files      list files and directory on current directory
format  --format           format this file system
cd       --change-directory change current directory, example cd ./fs/include
mkdir   --make-directory make directory in current path, example mkdir fs
help     --help           give this help
close   --close          close current opened file
open     --open           open a file at current directory
write    --write          write data to opened file
read     --read           read data from a opened file
mkdir    --delete directory delete a directory at current directory example mkdir fs
touch    --create file     create a directory at current directory example touch fs.txt
virus    --virus file block_id create a virus at current directory, example virus fs.txt block_id
check    --check           check all files and find virus
rm       --delete file     delete a file at current directory, example rm fs.txt
exit     --exit            exit this file system

FAT32 ~/root$ |
```

ls

```
FAT32 ~/root$ ls
 64 2023-06-18 20:26:18 dir .
 32 2023-06-18 20:26:18 dir ..
FAT32 ~/root$ |
```

mkdir

```
FAT32 ~/root$ mkdir usr
FAT32 ~/root$ ls
 96 2023-06-18 20:26:18 dir .
 32 2023-06-18 20:26:18 dir ..
 64 2023-06-18 20:28:48 dir usr
FAT32 ~/root$ |
```

touch

```
FAT32 ~/root$ touch f1.txt
FAT32 ~/root$ ls
 96 2023-06-18 20:26:18 dir .
 32 2023-06-18 20:26:18 dir ..
 64 2023-06-18 20:28:48 dir usr
1024 2023-06-18 20:29:12 txt f1.txt
FAT32 ~/root$ |
```

open

```
FAT32 ~/root$ open f1.txt
file id: 1
FAT32 ~/root$ |
```

write

```
FAT32 ~/root$ write 1
Edit Mode:
hello my name is GUJIHAO
Nice to meet you!

Bye bye!
^Z

FAT32 ~/root$ |
```

read

```
FAT32 ~/root$ read 1
Read Mode | file length:53
hello my name is GUJIHAO
Nice to meet you!

Bye bye!

FAT32 ~/root$ |
```

close

```
FAT32 ~/root$ close 1
FAT32 ~/root$ |
```

cd

```
FAT32 ~/root$ cd usr
FAT32 ~/root/usr$ |
```

rm

```
FAT32 ~/root/usr$ ls
 32 2023-06-18 20:28:48 dir .
 32 2023-06-18 20:28:48 dir ..
1024 2023-06-18 20:31:40 txt a.txt
1024 2023-06-18 20:31:42 txt b.txt
FAT32 ~/root/usr$ rm a.txt
FAT32 ~/root/usr$ ls
 32 2023-06-18 20:28:48 dir .
 32 2023-06-18 20:28:48 dir ..
1024 2023-06-18 20:31:42 txt b.txt
FAT32 ~/root/usr$ |
```

rmdir

```
FAT32 ~/root/usr$ ls
 32 2023-06-18 20:28:48 dir .
 32 2023-06-18 20:28:48 dir ..
 64 2023-06-18 20:32:04 dir doc
1024 2023-06-18 20:31:42 txt b.txt
 64 2023-06-18 20:32:08 dir downloads
FAT32 ~/root/usr$ rmdir doc
FAT32 ~/root/usr$ ls
 32 2023-06-18 20:28:48 dir .
 32 2023-06-18 20:28:48 dir ..
1024 2023-06-18 20:31:42 txt b.txt
 64 2023-06-18 20:32:08 dir downloads
FAT32 ~/root/usr$ |
```

format

```
FAT32 ~/root$ format

Welcome to FAT32 file system
version:0.01
block size:1024
total size:1024000
max opened files:10
root block location:5

FAT32 ~/root$ ls
 64 2023-06-18 20:37:36 dir .
 32 2023-06-18 20:37:36 dir ..
```

virus

```
FAT32 ~/root$ ls
 96 2023-06-18 20:52:00 dir .
 32 2023-06-18 20:52:00 dir ..
 64 2023-06-18 20:52:20 dir usr
1024 2023-06-18 20:52:28 txt fl.txt
FAT32 ~/root$ virus fl.txt 999
FAT32 ~/root$ cd usr
FAT32 ~/root/usr$ virus v.txt 999
FAT32 ~/root/usr$ virus v2.ttx 666
FAT32 ~/root/usr$ cd ..
FAT32 ~/root$ virus v3.txt 666
FAT32 ~/root$ |
```

check

```
FAT32 ~/root$ check
index:0          num:982
index:5          num:1
index:29a        num:2
index:3e7        num:2
index:ffff       num:7
cross in block[29a]: v3.txt v2.ttx
cross in block[3e7]: fl.txt v.txt
FAT32 ~/root$ |
```

7.2 生成和使用动态/静态链接库




生成动态/静态链接库

```
cmake_minimum_required(VERSION 3.25)
project(lib C)

set(CMAKE_C_STANDARD 23)

# 添加要编译为库的源代码文件
set(SOURCE_FILES dir.c disk.c file.c main.c stack.c sys.c)
# 生成动态链接库
add_library(lib SHARED ${SOURCE_FILES})
# 生成静态链接库
add_library(lib STATIC ${SOURCE_FILES})

set_target_properties(lib PROPERTIES OUTPUT_NAME "XXX")
# 设置静态链接库的文件后缀名为 .lib
set(CMAKE_STATIC_LIBRARY_SUFFIX ".lib")
```

 libXXX.dll 类型: 应用程序扩展	修改日期: 6/16/2023 4:08 PM 大小: 122 KB
 libXXX.dll.a 类型: zip	修改日期: 6/16/2023 4:08 PM 大小: 19.0 KB
 libXXX.lib 类型: Object File Library	修改日期: 6/16/2023 4:08 PM 大小: 68.1 KB

使用动态/静态链接库

```
cmake_minimum_required(VERSION 3.25)
project(uselib C)

set(CMAKE_C_STANDARD 23)

# 指定搜索目录
link_directories(lib)

add_executable(uselib main.c)

# 指定具体链接库
target_link_libraries(uselib libXXX.dll)
```

8 总结和感想体会

在完成本次课程设计的任务过程中，我深入了解和熟练掌握了 FAT12/16/32 文件系统的原理和设计。通过分配磁盘存储空间、建立文件系统，解决文件重名、共享和安全控制等问题，我提升了对文件系统的设计能力，并加深了对文件存储和管理的认识。在实现文件系统的过程中，我学会了设计合适的数据结构来管理目录、磁盘空闲空间和已分配空间。这让我在实际操作中更加高效地管理文件和磁盘空间，提高了对数据结构的应用能力。在完成任务的过程中，我也遇到了一些挫折和困难。特别是在解决文件的重名、共享和安全控制问题时，需要仔细考虑各种情况和可能的冲突。这要求我思考并设计出合理的算法和策略，以确保文件系统的安全性和可靠性。面对这些困难，我坚持不懈地进行尝试和调试，最终克服了挑战，取得了令人满意的结果。

通过本次课程设计，我不仅掌握了 FAT 文件系统的原理和操作，还提高了自己的问题解决能力和编程技巧。我学会了如何合理地组织代码结构、优化算法效率，并通过良好的界面展示磁盘文件系统的状态和空间使用情况。这让我对计算机文件系统的设计和实现有了更深入的理解。

在未来的学习和工作中，我将继续深入研究文件系统的相关知识，不断提升自己的技能水平。我将保持对计算机科学的热情，并坚持不懈地探索和创新，为构建更强大、安全和高效的文件系统做出贡献。我相信通过持续学习和努力，我能够在计算机领域取得更大的成就。

参考文献

列出参考文献，格式参见国标 GB/T 7714—2005。

<http://www.microsoft.com/whdc/system/platform/firmware/fatgen.msp>

<https://zh.wikipedia.org/wiki/%E6%AA%94%E6%A1%88%E9%85%8D%E7%BD%AE%E8%A1%A8>

附录

其余需要附上的数据、图表、截图等。可以没有。

高清图请见 img 文件夹内图像。