|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学生姓名 | |  | 年级专业 |  | 学号 |  |
| 项目名称 | | **小型磁盘文件管理系统** | | | | |
| 项  目  实  现  原  理  简  述 | 内容：  （1）支持多级目录结构，支持文件的绝对路径访问；  （2）文件的逻辑结构采用流式（字节流）结构，物理结构采用链式结构中的显式链方式；  （3）磁盘管理采用文件分配表；  （4）实现文件操作命令：创建目录、列表目录、删除空目录、创建文件、删除文件、显示文件内容、打开文件、读文件、写文件（追加方式）、关闭文件、改变文件属性；  （5）通过主函数对所实现的功能进行测试  原理：实现一个模拟磁盘文件管理系统的FAT表（文件分配表）是模拟磁盘管理的核心部分。FAT表记录了每个磁盘块的状态，包括文件的块链表信息、空闲块信息   1. 磁盘模拟   数组模拟   1. 初始化文件系统   **init\_fs()**   1. 查找空闲块   **find\_free\_block()**   1. 分配和释放块   **allocate\_blocks(int length)**  **release\_blocks(int start\_block)**   1. 显示fat   **show\_fat\_table()**函数输出当前FAT表状态，便于调试和观察FAT表的变化。   1. 文件读写   **create\_file(const char\* name, const char\* type, int length)**为文件分配块，并在根目录中创建对应的目录项。  **delete\_file(const char\* name)**从根目录中删除文件，并释放其占用的块  **list\_directory()**函数输出根目录中所有文件的信息 | | | | | |
| 完  成  情  况 | #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #define BLOCK\_SIZE 64     // 每个磁盘块的大小为64字节  #define TOTAL\_BLOCKS 128  // 磁盘总共包含128个块  #define FAT\_SIZE TOTAL\_BLOCKS  #define ROOT\_DIR\_BLOCK 1  #define FREE\_BLOCK 0  #define END\_OF\_FILE -1  #define FAULTY\_BLOCK 254  // 定义目录项结构  typedef struct {      char filename[4];   // 3字节文件名 + '\0'      char filetype[2];   // 1字节文件类型 + '\0'      char attributes;    // 文件属性      char start\_block;   // 起始盘块号      char length;        // 文件长度（单位：块）  } DirectoryEntry;  // 定义目录结构  typedef struct {      DirectoryEntry entries[8]; // 每个目录最多8个目录项  } Directory;  // 定义文件系统结构  typedef struct {      char disk[TOTAL\_BLOCKS][BLOCK\_SIZE]; // 模拟磁盘的二维数组，每个块64字节，总共128块      int FAT[FAT\_SIZE];                   // 文件分配表      Directory root;                      // 根目录  } FileSystem;  FileSystem fs; // 全局文件系统变量  // 初始化文件系统  void init\_fs() {      // 初始化FAT表，将所有块设为空闲块      for (int i = 0; i < TOTAL\_BLOCKS; i++) {          fs.FAT[i] = FREE\_BLOCK;      }      // 将根目录块设置为文件结束块      fs.FAT[ROOT\_DIR\_BLOCK] = END\_OF\_FILE;      // 初始化根目录，设置为8个空目录项      for (int i = 0; i < 8; i++) {          fs.root.entries[i].filename[0] = '$';      }      printf("File system initialized.\n");  }  // 查找第一个空闲块，返回其块号  int find\_free\_block() {      for (int i = 0; i < TOTAL\_BLOCKS; i++) {          if (fs.FAT[i] == FREE\_BLOCK) {              return i;          }      }      return -1;  }  // 为文件分配指定数量的块，并更新FAT表，返回起始块号  int allocate\_blocks(int length) {      int start\_block = -1;      int prev\_block = -1;      for (int i = 0; i < length; i++) {          int free\_block = find\_free\_block();          if (free\_block == -1) {              printf("Not enough free blocks available.\n");              return -1;          }          if (start\_block == -1) {              start\_block = free\_block;          } else {              fs.FAT[prev\_block] = free\_block;          }          prev\_block = free\_block;      }      fs.FAT[prev\_block] = END\_OF\_FILE;      return start\_block;  }  // 释放文件占用的块，并更新FAT表  void release\_blocks(int start\_block) {      int block = start\_block;      while (block != END\_OF\_FILE) {          int next\_block = fs.FAT[block];          fs.FAT[block] = FREE\_BLOCK;          block = next\_block;      }  }  // 显示当前FAT表状态  void show\_fat\_table() {      printf("FAT Table:\n");      for (int i = 0; i < FAT\_SIZE; i++) {          printf("%d ", fs.FAT[i]);      }      printf("\n");  }  // 创建文件，并在根目录中创建相应目录项  int create\_file(const char\* name, const char\* type, int length) {      int start\_block = allocate\_blocks(length);      if (start\_block == -1) {          return -1;      }      for (int i = 0; i < 8; i++) {          if (fs.root.entries[i].filename[0] == '$') {              strncpy(fs.root.entries[i].filename, name, 3);              strncpy(fs.root.entries[i].filetype, type, 1);              fs.root.entries[i].attributes = 0x00;              fs.root.entries[i].start\_block = start\_block;              fs.root.entries[i].length = length;              printf("File '%s.%s' created at block %d.\n", name, type, start\_block);              return 0;          }      }      printf("Root directory is full.\n");      return -1;  }  // 删除文件，并释放其占用的块  void delete\_file(const char\* name) {      for (int i = 0; i < 8; i++) {          if (strncmp(fs.root.entries[i].filename, name, 3) == 0) {              int block = fs.root.entries[i].start\_block;              release\_blocks(block);              fs.root.entries[i].filename[0] = '$';              printf("File '%s' deleted.\n", name);              return;          }      }      printf("File '%s' not found.\n", name);  }  // 列出根目录中所有文件的信息  void list\_directory() {      for (int i = 0; i < 8; i++) {          if (fs.root.entries[i].filename[0] != '$') {              printf("File: %s.%s, Start block: %d, Length: %d blocks\n",                     fs.root.entries[i].filename, fs.root.entries[i].filetype,                     fs.root.entries[i].start\_block, fs.root.entries[i].length);          }      }  }  int main() {      init\_fs(); // 初始化文件系统      // create\_file("abc", "txt", 3); // 创建文件      // create\_file("def", "bin", 2); // 创建文件      // list\_directory(); // 列出根目录文件      // show\_fat\_table(); // 显示FAT表      // delete\_file("abc"); // 删除文件      // list\_directory(); // 列出根目录文件      // show\_fat\_table(); // 显示FAT表        create\_file("abc", "txt", 3); // 创建文件      create\_file("def", "bin", 2); // 创建文件      create\_file("def2", "bin", 3); // 创建文件      list\_directory(); // 列出根目录文件      show\_fat\_table(); // 显示FAT表      return 0;  } | | | | | |
| 主  要  特  点 | 1.初始化fat表：将所有块设置为空闲块，初始化根目录  2.查找空闲块：找到第一个空闲块并返回其块号  3.分配块：分配指定数量的块并更新FAT表。  4.释放块：释放文件占用的块并更新FAT表。  5.输出fat表：输出FAT表的当前状态。 | | | | | |
| 尚存问  题  及  改  进  思  路 | 建立文件目录，查询，有点难在思考 | | | | | |