

# 实验三伪代码

---

```
1 #include <stdio.h>           // 包含标准输入输出库
2 #include <stdlib.h>          // 包含标准库函数, 如malloc和exit
3 #include <string.h>           // 包含字符串处理函数
4
5 #define MAX_FILES 50          // 定义文件系统的最大文件数量
6 #define DISK_SIZE 500           // 定义磁盘大小 (以块为单位)
7 #define GROUP_SIZE 30           // 定义每组空闲块的数量
8
9 // 定义文件结构体, 用于存储文件信息
10 typedef struct {
11     char name[256];           // 文件名的最大长度为255字符加一个终止符
12     int firstBlock;           // 文件的第一个块号
13     int blockCount;           // 文件占用的块数
14 } File;
15
16 // 定义空闲块组结构体, 用于管理每组空闲块
17 typedef struct BlockGroup {
18     int blocks[GROUP_SIZE];    // 存储本组空闲块号的数组
19     int count;                 // 当前组内剩余空闲块数量
20 } BlockGroup;
21
22 // 定义超级块结构体, 用于管理整个空白块链
23 typedef struct {
24     BlockGroup *groups;        // 动态分配的空闲块链数组指针
25     int currentGroup;          // 当前可用的最后一组索引
26     int groupCount;            // 总组数
27 } SuperBlock;
28
29 SuperBlock superBlock;        // 全局变量: 超级块
30 File fileList[MAX_FILES];     // 全局数组: 所有文件列表
31 int fileCount = 0;             // 当前文件数量计数器
32 int totalAllocatedBlocks = 0;   // 分配的物理块总数计数器
33 int diskReads = 0;              // 磁盘读取次数计数器
34
35 // 初始化超级块及各组空闲块链
36 void initSuperBlock(int diskSize, int freeBlocks, int blockSize) {
37     // 验证输入参数是否合理, 防止无效输入导致程序错误
38     if (blockSize <= 0 || diskSize <= 0 || freeBlocks < 0) {
39         fprintf(stderr, "无效的输入参数\n"); // 输出错误信息到标准错误流
40         exit(EXIT_FAILURE);                  // 使用exit退出程序, 表示失败
41     }
42
43     // 计算总组数, 确保至少有一组, 避免除零错误
44     int totalGroups = (diskSize + blockSize - 1) / blockSize;
45     if (totalGroups <= 0) {
```

```

46         fprintf(stderr, "无效的组数: %d\n", totalGroups); // 输出错误信息
47         exit(EXIT_FAILURE); // 使用exit退出
48     }
49
50     // 动态分配内存给超级块中的空闲块链数组，并检查分配是否成功
51     superBlock.groups = malloc(totalGroups * sizeof(*superBlock.groups));
52     if (superBlock.groups == NULL) { // 检查malloc是否返回NULL，即分配失败
53         fprintf(stderr, "内存分配失败\n"); // 输出错误信息
54         exit(EXIT_FAILURE); // 使用exit退出程序
55     }
56
57     // 初始化超级块的基本属性
58     superBlock.currentGroup = 0; // 设置当前使用的最后一组索引为0
59     superBlock.groupCount = totalGroups; // 设置总组数
60
61     // 根据输入的空闲块数初始化每一组的空闲块
62     for (int i = 0; i < totalGroups && freeBlocks > 0; ++i) {
63         // 计算每组实际分配的空闲块数，不超过每组最大容量
64         superBlock.groups[i].count = (freeBlocks > blockSize) ? blockSize
65         : freeBlocks;
66         // 填充每组的空闲块号，从当前组起始位置开始编号
67         for (int j = 0; j < superBlock.groups[i].count; ++j) {
68             superBlock.groups[i].blocks[j] = i * blockSize + j;
69         }
70         // 更新剩余的空闲块数
71         freeBlocks -= superBlock.groups[i].count;
72     }
73
74     // 创建并初始化每个组的第一个空闲块对应的文本文件，模拟实际磁盘操作
75     for (int i = 0; i < superBlock.groupCount; ++i) {
76         if (superBlock.groups[i].count > 0) {
77             char filename[10]; // 准备文件名缓冲区
78             sprintf(filename, "%d.txt", superBlock.groups[i].blocks[0]);
79             // 格式化文件名为"首块号.txt"
80             FILE *file = fopen(filename, "w"); // 以写模式打开或创建文件
81             if (file != NULL) { // 检查文件是否成功打开或创建
82                 // 将该组的所有空闲块号写入文件
83                 for (int j = 0; j < superBlock.groups[i].count; ++j) {
84                     fprintf(file, "%d\n", superBlock.groups[i].blocks[j]);
85                 }
86             } else {
87                 fprintf(stderr, "无法创建文件 %s\n", filename); // 如果文件
88             }
89         }
90     }

```

```

89      }
90  }
91
92 // 打印超级块信息，包括所有空闲块组的信息
93 void printSuperBlock() {
94     printf("SuperBlock:\n"); // 输出标题
95     // 遍历所有已初始化的组，打印其内容
96     for (int i = 0; i <= superBlock.currentGroup; ++i) {
97         printf("Group %d: ", i); // 输出组号
98         // 打印该组内的所有空闲块号
99         for (int j = 0; j < superBlock.groups[i].count; ++j) {
100             printf("%d ", superBlock.groups[i].blocks[j]);
101         }
102         printf("\n"); // 换行
103     }
104 }
105
106 // 分配指定数量的空闲块，并返回分配的起始块号
107 int allocateBlocks(int count) {
108     int startBlock = -1; // 初始化返回值为-1，表示分配失败
109     if (count > 0) { // 只有当请求的块数大于0时才进行分配
110         // 查找是否有足够空闲块的组，从后向前遍历
111         while (superBlock.currentGroup >= 0 && superBlock.groups[superBlock.currentGroup].count < count) {
112             --superBlock.currentGroup; // 更新currentGroup索引
113         }
114
115         // 如果找到合适的组，则进行分配
116         if (superBlock.currentGroup >= 0) {
117             startBlock = superBlock.groups[superBlock.currentGroup].blocks[0];
118             s[0]; // 获取起始块号
119             superBlock.groups[superBlock.currentGroup].count -= count;
120             // 更新该组剩余空闲块数
121             // 将未分配的块号向前移动，覆盖已分配的块号
122             memmove(&superBlock.groups[superBlock.currentGroup].blocks[0]
123                     ,
124                     &superBlock.groups[superBlock.currentGroup].blocks[count],
125                     sizeof(int) * (superBlock.groups[superBlock.currentGroup].count));
126
127             // 如果当前组已无剩余空闲块，则更新currentGroup
128             if (superBlock.groups[superBlock.currentGroup].count == 0) {
129                 --superBlock.currentGroup;
130             }
131         }
132     }
133     return startBlock; // 返回分配的起始块号，或-1表示分配失败

```

```
131    }
132
133    // 释放指定起始块号及其后连续的指定数量的块回空闲块链
134    void releaseBlocks(int startBlock, int count) {
135        int groupIndex = startBlock / GROUP_SIZE; // 计算要释放的块属于哪一组
136        // 将释放的块号重新加入到对应组的空闲块列表中
137        for (int i = 0; i < count; ++i) {
138            superBlock.groups[groupIndex].blocks[superBlock.groups[groupIndex]
139                .count++] = startBlock + i;
140        }
141        // 更新currentGroup索引, 如果必要的话
142        if (groupIndex > superBlock.currentGroup) {
143            superBlock.currentGroup = groupIndex;
144        }
145
146        // 更新对应的文本文件, 模拟实际磁盘操作
147        char filename[10]; // 准备文件名缓冲区
148        sprintf(filename, "%d.txt", superBlock.groups[groupIndex].blocks[0])
149        ; // 格式化文件名为"首块号.txt"
150        FILE *file = fopen(filename, "w"); // 以写模式打开文件
151        if (file != NULL) { // 检查文件是否成功打开
152            // 将该组的所有空闲块号写入文件
153            for (int j = 0; j < superBlock.groups[groupIndex].count; ++j) {
154                fprintf(file, "%d\n", superBlock.groups[groupIndex].blocks[j]
155            );
156            }
157            fclose(file); // 关闭文件
158            diskReads++; // 增加磁盘读取次数计数器
159        } else {
160            fprintf(stderr, "无法打开文件 %s 进行更新\n", filename); // 如果文件打
开失败, 输出错误信息
161        }
162
163    // 添加新文件到文件系统中
164    void addFile() {
165        if (fileCount >= MAX_FILES) { // 检查文件系统是否已满
166            printf("文件系统已满, 无法添加更多文件。\\n");
167            return;
168        }
169        printf("请输入文件名\\n"); // 提示用户输入文件名
170        scanf("%255s", fileList[fileCount].name); // 限制输入长度, 防止缓冲区溢出
171        printf("请输入申请块数\\n"); // 提示用户输入申请的块数
172        int blockCount;
173        scanf("%d", &blockCount); // 读取用户输入的块数
174        // 尝试分配请求的块数
```

```

175     int startBlock = allocateBlocks(blockCount);
176     if (startBlock != -1) { // 如果分配成功
177         fileList[fileCount].firstBlock = startBlock; // 设置文件的首块号
178         fileList[fileCount].blockCount = blockCount; // 设置文件占用的块数
179         ++fileCount; // 增加文件计数器
180         totalAllocatedBlocks += blockCount; // 更新分配的物理块总数计数器
181         printSuperBlock(); // 打印当前超级块信息
182     } else {
183         printf("没有足够的空闲块来创建文件。\\n"); // 如果分配失败，输出提示信息
184     }
185 }
186
187 // 删除现有文件，释放其占用的块
188 void deleteFile() {
189     printf("请输入要删除的文件名\\n"); // 提示用户输入要删除的文件名
190     char fileName[256];
191     scanf("%255s", fileName); // 限制输入长度，防止缓冲区溢出
192
193     // 查找并删除指定文件
194     for (int i = 0; i < fileCount; ++i) {
195         if (strcmp(fileList[i].name, fileName) == 0) { // 如果找到了匹配的文
196            件
197             releaseBlocks(fileList[i].firstBlock, fileList[i].blockCount)
198             ; // 释放文件占用的块
199             // 将文件列表中后续文件前移一位，覆盖被删除的文件
200             memmove(&fileList[i], &fileList[i + 1], sizeof(File) * (fileC
201             ount - i - 1));
202             --fileCount; // 减少文件计数器
203             totalAllocatedBlocks -= fileList[i].blockCount; // 更新分配的物
204            理块总数计数器
205             printSuperBlock(); // 打印当前超级块信息
206             return;
207         }
208     }
209     printf("未找到名为 '%s' 的文件。\\n", fileName); // 如果未找到文件，输出提示
210     信息
211 }
212
213 // 列出当前所有文件的信息
214 void listFiles() {
215     printf("-----当前已用的文件目录-----\\n"); // 输出标题
216     // 遍历所有文件，打印其信息
217     for (int i = 0; i < fileCount; ++i) {
218         printf("文件名: %s, 首块号: %d, 占用块个数: %d\\n", fileList[i].name,
219             fileList[i].firstBlock, fileList[i].blockCount);
220         // 打印文件占用的所有物理块号
221         for (int j = 0; j < fileList[i].blockCount; ++j) {
222             printf("物理块号: %d\\n", fileList[i].firstBlock + j);
223         }
224     }
225 }
```

```
217         }
218     }
219 }
220
221 // 主函数, 程序入口
222 int main() {
223     int diskSize, freeBlocks, blockSize;
224
225     // 获取用户输入的磁盘大小、空闲块数和每组空闲块数, 并验证输入合理性
226     printf("请输入磁盘大小 (块数) : \n");
227     if (scanf("%d", &diskSize) != 1 || diskSize <= 0) { // 检查输入是否有效
228         fprintf(stderr, "无效的磁盘大小\n");
229         return EXIT_FAILURE; // 返回失败状态码
230     }
231
232     printf("请输入空闲物理块数: \n");
233     if (scanf("%d", &freeBlocks) != 1 || freeBlocks < 0 || freeBlocks > diskSize) { // 检查输入是否有效
234         fprintf(stderr, "无效的空闲物理块数\n");
235         return EXIT_FAILURE; // 返回失败状态码
236     }
237
238     printf("请输入每组空闲块数: \n");
239     if (scanf("%d", &blockSize) != 1 || blockSize <= 0) { // 检查输入是否有
240         // 效
241         fprintf(stderr, "无效的每组空闲块数\n");
242         return EXIT_FAILURE; // 返回失败状态码
243     }
244
245     // 初始化超级块
246     initSuperBlock(diskSize, freeBlocks, blockSize);
247
248     int choice;
249     do {
250         // 提供一个简单的命令行界面让用户选择不同的操作
251         printf("\n请选择操作: \n1. 创建文件\n2. 删除文件\n3. 列出所有文件\n4. 退
252         出\n选择: ");
253         if (scanf("%d", &choice) != 1) { // 检查输入是否有效
254             printf("无效选项, 请重试.\n");
255             continue;
256         }
257
258         switch (choice) {
259             case 1:
260                 addFile(); // 调用添加文件函数
261                 break;
262             case 2:
263                 deleteFile(); // 调用删除文件函数
```

```
262         break;
263     case 3:
264         listFiles(); // 调用列出所有文件函数
265         break;
266     case 4:
267         printf("退出程序。\\n"); // 用户选择退出程序
268         break;
269     default:
270         printf("无效选项, 请重试。\\n"); // 用户输入了无效选项
271     }
272 } while (choice != 4); // 循环直到用户选择退出
273
274 // 输出统计信息
275 printf("分配的物理块总数: %d\\n", totalAllocatedBlocks);
276 printf("读磁盘次数: %d\\n", diskReads);
277
278 // 释放动态分配的内存, 防止内存泄漏
279 free(superBlock.groups);
280 return 0; // 返回成功状态码
281 }
```