Project 6 File System 设计文档

中国科学院大学
葛忠鑫
2021.1.15

1. 文件系统初始化设计

（1）请阐述你设计的文件系统对磁盘的布局（可以使用图例表示），包括从磁盘哪个位置开始，superblock、inode map、block 或 sector map、inode table 以及数据区各自占用的磁盘空间大小

文件系统采用 block map 表示数据块的空间占用情况，并从磁盘偏移 512MB 的开始。下表记录了 superblock、inodemap、blockmap、inode table 以及数据区各自占用的磁盘空间大小，相对文件系统开始处的偏移和占用块数。

	SIZE	OFFSET	BLOCK
SuperBlock	512B	0	1
Block Bitmap	256K / 8 * 1B= 32KB	4KB	8
Inode Bitmap	32K / 8 * 1B= 4KB	36KB	1
Inode table	32K * 128B = 4MB	40KB	1K
Data	1GB – 4MB - 40KB	4MB 40KB	254K 1014

其中每个 inode 结构体的大小为 128B。

```
/*
* File System
* FS size    : 1GB
* Block size : 4KB
* Inode size : 128B
* Inode Bitmap size : 32K / 8 * 1B= 4KB
* Block Bitmap size : 256K / 8 * 1B= 32KB
*-------------------------------------------------------------------------
*|  Superblock  | Block Bitmap  | Inode Bitmap  |      Inode      |  Blocks  |
*| 1 Block 4KB  | 8 Blocks 32KB | 1 Block  4KB  | 1024Blocks 4MB  |  Others  |
*-------------------------------------------------------------------------
*/
```

（2）请列出你设计的 superblock 和 inode 数据结构，并阐明各项含义。请说明你设计的文件系统能支持的最大文件大小，最多文件数目，以及单个目录下能支持的最多文件/子目录数目。

superblock 结构体如下：

```
1.  typedef struct super_block
2.  {
3.      uint32_t magic;
4.      uint32_t start_sector;
5.      uint32_t fs_size;
6.      uint32_t block_size;
7.
```

```
8.      uint32_t block_bmp_offset;
9.      uint32_t inode_bmp_offset;
10.     uint32_t inode_offset;
11.     uint32_t data_offset;
12.
13.     uint32_t total_inode_num;
14.     uint32_t free_inode_num;
15.
16.     uint32_t total_block_num;
17.     uint32_t free_block_num;
18.
19.     uint32_t inode_size;
20.     uint32_t dir_size;
21. } super_block_t;
```

super_block 结构体各项的含义为：魔法数、文件系统起始位置相对 SD 卡偏移、文件系统大小、块大小、blockmap 相对文件系统偏移、inodemap 偏移、inode table 偏移、数据区偏移、inode 总数、可用（空闲）inode 数目、文件系统总块数、可用（空闲）数据块数目、每个 inode 大小以及每个目录项大小。

inode 结构体如下：

```
1.  typedef struct inode_entry
2.  {
3.      uint32_t id;    // inode number
4.      uint32_t type;
5.      uint32_t mode;
6.      uint32_t links_cnt;
7.      uint32_t fsize;
8.      uint32_t fnum;  // 目录中文件数
9.      uint32_t timestamp;
10.     uint32_t direct_table[MAX_DIRECT_NUM];
11.     uint32_t indirect_1_ptr;
12.     uint32_t indirect_2_ptr;
13.     uint32_t indirect_3_ptr;
14.     uint32_t padding[12];
15. } inode_entry_t;    // 32 * 4B = 128B
```

inode 结构体各项的含义为：inode num、文件类型（文件、目录和软连接等），权限（可读可写、只读和只写）、硬链接数目、文件大小、目录中的文件数、时间戳、10 个直接索引、一个一级索引、一个二级索引和一个三级索引。

此文件系统采用三级索引，所支持最大文件大小为 4TB 4GB 4MB 40KB。其最多支持 32K = 32768 个文件和目录。本文件系统默认一个目录下只用一个数据块即可存下所有目录项，则单个目录下能支持的最多文件/子目录数目为 4KB / 32B = 128 个。

目录项结构体如下：

```
1.  typedef struct dir_entry
2.  {
3.      uint32_t id;
```

```
    uint32_t type;
    uint32_t mode;
    char name[MAX_NAME_LENGTH];
} dir_entry_t;      // 32B
```

目录项中额外存储了类型和权限，方便 ls 查看，减少读操作，提高效率。

（3）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

文件系统实验时，修改 Makefile 时需要注意：不要在 make clean 时删除 disk。否则会造成下一次生成 disk 后启动 qemu，产生 USB timeout 错。

2. 文件操作设计

（1）请说明创建一个文件所涉及的元数据新增和修改操作，例如需要新增哪些元数据，需要修改哪些元数据

创建新文件时，需要分配并修改一个 inode、分配一个数据块。在分配这两项时需要修改 superblock 的空闲 inode 块数、空闲数据块数，inodo bitmap 和 block bitmap。

需要修改其所在目录的数据块，新增一个目录项，并更新其父目录中的相关内容：文件数加 1、大小加一个目录块大小、并更新时间戳。

创建文件操作如下：

```
int init_file(char *name, uint32_t access)
{
    // TODO: check name
    kprintf("Initializing file: %s...\n", name);

    uint32_t id = alloc_inode();
    uint32_t block = alloc_block();
    init_inode(block, current_dir_entry.id, id, O_RDWR, TYPE_FILE);

    // modify current dir block
    read_block(current_dir_entry.direct_table[0]);
    dir_entry_t *dir = (dir_entry_t *)(BUFFER + (current_dir_entry.fnum + 2) * sizeof(dir_entry_t));
    dir->id = id;
    dir->type = TYPE_FILE;
    dir->mode = access;
    strcpy(dir->name, name);
    write_block(current_dir_entry.direct_table[0]);

    // modify current dir entry
    current_dir_entry.fsize += sizeof(dir_entry_t);
    current_dir_entry.fnum += 1;
    // printk("current id = %d inode.fnum = %d\n\r", current_dir_entry.id, current_dir_entry.fnum);
    current_dir_entry.timestamp = get_timer();
```

```
24.     memcpy((uint8_t *)&inode_buffer, (uint8_t *)&current_dir_entry,
    sizeof(inode_entry_t));
25.     write_inode(current_dir_entry.id);
26.     return id;
27. }
```

3. 目录操作设计

（1）请说明文件系统执行 ls 命令查看一个绝对路径时的操作流程

本实现有些偷懒：首先，保存当前目录的 inode num 到一个全局变量（如下）。

```
1.  void save_current_dir()
2.  {
3.      current_dir_id = current_dir_entry.id;
4.  }
```

然后，调用 cd 操作对应的内核函数 do_enterdir 定位到该绝对路径，大引其数据块中的所有目录项信息。最后，恢复到操作前的目录（如下）。

```
1.  void restore_current_dir()
2.  {
3.      read_inode(current_dir_id);
4.      memcpy((uint8_t *)&current_dir_entry, (uint8_t *)&inode_buffer,
    sizeof(inode_entry_t));
5.  }
```

下面介绍关于路径的解析（位于 do_enterdir 函数中）：

a) 删除路径最后的‘/’；

b) 开始循环解析，当路径长度为 0 时结束；

c) 首先判断路径开头是否为‘/’，若是，则定位到根目录 root，根目录的 inode num 为 0；

d) 判断“..”，若是，根据目录数据块中所存父目录信息进入父目录（根目录 root 的父目录为自己），并跳转到 h 执行；

e) 判断“.”，若是，则跳转到 h 执行；

f) 读出路径中的下一个名字，在当前目录项中查找，若找到，则进入该目录，并跳转到 h 执行；

g) 若未找到，则报错，函数返回错误（-1），

h) 若未解析完，跳过‘/’符号，回到 b 循环继续解析。

i) 返回成功（0）。

具体实现代码如下：

```
1.  int do_enterdir(char *name)
2.  {
3.      int id;
4.      // printk("In cd...\n\r");
5.      if (name[strlen(name) - 1] == '/' && strlen(name) != 1)
6.      {
7.          name[strlen(name) - 1] = '\0';
8.      }
```

```
    char name_buf[10];
    while (strlen(name))
    {
        // printk("name len: %d, %s\n\r", strlen(name), name);
        if (name[0] == '/')
        {
            id = ROOT_ID;
            strcpy(name_buf, "root");
            name++;
        }
        else if (name[0] == '.' && name[1] == '.')
        {
            if (name[2] == '/' || name[2] == '\0')
            {
                read_block(current_dir_entry.direct_table[0]);
                dir_entry_t *dir = (dir_entry_t *)BUFFER; // ..
                id = dir->id;
                if (name[2] == '/')
                {
                    name += 3;
                }
                else
                {
                    name += 2;
                }
                strcpy(name_buf, "..");
            }
            else
            {
                kprintf("Invalid path: %s\n", name);
                return -1;
            }
        }
        else if (name[0] == '.')
        {
            if (name[1] == '/' || name[1] == '\0')
            {
                if (name[1] == '/')
                {
                    name += 2;
                }
                else
                {
                    // '\0'
```

```
53.                     name += 1;
54.                     break;
55.                 }
56.             }
57.             else
58.             {
59.                 kprintf("Invalid path: %s", name);
60.                 return -1;
61.             }
62.             continue;
63.         }
64.         else
65.         {
66.             int j;
67.             for (j = 0; j < strlen(name) && name[j] != '/'; j++)
68.             {
69.                 name_buf[j] = name[j];
70.             }
71.             name_buf[j] = '\0';
72.             // printk("name buf: %s\n\r", name_buf);
73.             int i = is_name_in_dir(name_buf);
74.             if (i == -1)
75.             {
76.                 kprintf("Dir: %s not found...\n", name);
77.                 // do_mkdir(name_buf);
78.                 // i = is_name_in_dir(name_buf);
79.                 return -1;
80.             }
81.             dir_entry_t *dir = (dir_entry_t *)(BUFFER + (i + 2)
    * sizeof(dir_entry_t));
82.             id = dir->id;
83.             if (name[j] == '\0')
84.             {
85.                 name += j;
86.             }
87.             else
88.             {
89.                 name += j + 1;
90.             }
91.         }
92.         kprintf("Opening dir: %s (id = %d)...\n", name_buf, id);
93.         read_inode(id);
94.         memcpy((uint8_t *)&current_dir_entry, (uint8_t *)&inode_
    buffer, sizeof(inode_entry_t));
```

```
95.     }
96.     return 0;
97. }
```

ls 对应的内核函数 do_readdir 实现如下：

```
1.  int do_readdir(char *name)
2.  {
3.      save_current_dir();
4.      do_enterdir(name);
5.      read_block(current_dir_entry.direct_table[0]);
6.      int i;
7.      dir_entry_t *dir = (dir_entry_t *)(BUFFER + 2 * sizeof(dir_e
    ntry_t));
8.      kprintf(". ..\n");
9.      if (current_dir_entry.fnum)
10.     {
11.         kprintf("  Name \t TYPE \t Mode\n");
12.         // kprintf("current_dir_entry.fnum = %d\n\r", current_di
    r_entry.fnum);
13.     }
14.     for (i = 0; i < current_dir_entry.fnum; i++)
15.     {
16.         dir_entry_t *dir = (dir_entry_t *)(BUFFER + (i + 2) * si
    zeof(dir_entry_t));
17.         kprintf("  %s    %s    %s\n",
18.                 dir->name,
19.                 (dir->type == TYPE_DIR) ? "DIR" : "FILE",
20.                 (dir->mode == O_RDWR) ? "O_RDWR" : (dir->mode ==
     O_WRONLY) ? "O_WRONLY" : "O_RDONLY");
21.     }
22.     restore_current_dir();
23.     return 0;
24. }
```

（2）设计或实现过程中遇到的问题和得到的经验（如果有的话可以写下来，不是必需项）

对于不存在目录项的处理：直接报错，而不是创建路径。