Project 6 File System 设计文档

马月骁· 2019K8009915025

1. 文件系统初始化设计

（1）文件系统对磁盘的布局

```
/*****************************************************************************
***************************
Disk layout:
[ boot block | kernel image ] (512MB)
[ Superblock(512B) | block map(128KB) | inode map(512B) | inode
blocks(256KB) | data blocks ] (512MB)
*****************************************************************************
***************************/
```

（2）superblock，inode，dentry， file_descriptor 数据结构

Superblock:

```
typedef struct superblock{
    u32 magic;
    u32 fs_num;
    u32 fs_start_block;

    u32 block_map_offset;
    u32 block_map_num;

    u32 inode_map_offset;
    u32 inode_map_num;

    u32 inode_block_offset;
    u32 inode_block_num;

    u32 data_block_offset;
    u32 data_block_num;
}superblock_t;
```

Inode:

```
typedef struct inode{
    u8  ino;
    u8  mode;
    u8  num; //link
    u16 used_size;
    u16 ctime;
    u16 mtime;
    u16 direct[MAX_DIR];
    u16 indirect1;
    u16 indirect2;
    u32 bytes; //file
```

```
}inode_t;
```

Dentry:

```
typedef struct dentry{
    char name[MAX_NAME_LENGTH];
    u8   type;
    i16  ino;
}dentry_t;
```

File_descriptor:

```
typedef struct file{
    u32 inode;
    u32 access;
    //u32 addr;
    u32 rd_pos;
    u32 wr_pos;
    u32 valid;
}file_t;
```

（3）初始化操作

```
void do_mkfs(){
    prints("[FS] Start initialize filesystem!\n");
    prints("[FS] Setting up superblock...\n");
    superblock_t sb;
    sb.magic = FS_MAGIC;
    sb.fs_num = FS_SIZE;
    sb.fs_start_block = FS_START;
    sb.block_map_offset = BLOCK_MAP_OFFSET;
    sb.block_map_num = BLOCK_MAP_NUM;
    sb.inode_map_offset = INODE_MAP_OFFSET;
    sb.inode_map_num = INODE_MAP_NUM;
    sb.inode_block_offset = INODE_BLOCK_OFFSET;
    sb.inode_block_num = INODE_BLOCK_NUM;
    sb.data_block_offset = DATA_NUM_OFFSET;
    sb.data_block_num = DATA_BLOCK_NUM;
    sbi_sd_write(kva2pa(&sb), 1, FS_START);

    prints("[FS] Checking superblock:\n");
    u8 tmp[512];
    sbi_sd_read(kva2pa(tmp), 1, FS_START);
    superblock_t *tmp_sb_p = (superblock_t *)tmp;
    prints("     magic: 0x%x\n", tmp_sb_p->magic);
    prints("     num sector: %d, start sector: %d\n", tmp_sb_p->fs_num,
tmp_sb_p->fs_start_block);
```

```
    prints("     block map offset: %d, block map num: %d\n",
tmp_sb_p->block_map_offset, tmp_sb_p->block_map_num);
    prints("     inode map offset: %d, inode map num: %d\n",
tmp_sb_p->inode_map_offset, tmp_sb_p->inode_map_num);
    prints("     inode block offset: %d, inode block num: %d\n",
tmp_sb_p->inode_block_offset, tmp_sb_p->inode_block_num);
    prints("     data block offset: %d, data block num: %d\n",
tmp_sb_p->data_block_offset, tmp_sb_p->data_block_num);

    //initialize block map, inode map, inode , data
    //1: clear, 2: set
    kmemset(tmp, 0, 512);
    prints("[FS] Setting up block map...\n");
    for(int i = 0; i < BLOCK_MAP_NUM; i++){
        sbi_sd_write(kva2pa(tmp), 1, FS_START + BLOCK_MAP_OFFSET + i);
    }
    u32 block_num = alloc_block();

    kmemset(tmp, 0, 512);
    prints("[FS] Setting up inode map...\n");
    for(int i = 0; i < INODE_MAP_NUM; i++){
        sbi_sd_write(kva2pa(tmp), 1, FS_START + INODE_MAP_OFFSET + i);
    }
    u32 inode_num = alloc_inode();

    kmemset(tmp, 0, 512);
    prints("[FS] Setting up inode block...\n");
    for(int i = 0; i < INODE_BLOCK_NUM; i++){
        sbi_sd_write(kva2pa(tmp), 1, FS_START + INODE_BLOCK_OFFSET + i);
    }
    kmemset(tmp, 0, 512);
    prints("     inode_num = %d\n", inode_num);
    inode_t *inode_p = tmp;
    inode_p->ino = inode_num;
    inode_p->mode = O_RW;
    inode_p->num = inode_num;
    inode_p->used_size = 2;
    inode_p->ctime = get_timer();
    inode_p->mtime = get_timer();
    inode_p->direct[0] = block_num;
    sbi_sd_write(kva2pa(inode_p), 1, FS_START + INODE_BLOCK_OFFSET +
inode_num);
    global_inode_num = inode_num;
```

```
    kmemset(tmp, 0, 512);
    prints("[FS] Setting up data block(root directory)...\n");
    prints("     block_num = %d\n", block_num);
    dentry_t *root_dentry = (dentry_t *)tmp;
    kmemcpy(root_dentry->name, ".", 1);
    root_dentry->type = T_DIR;
    root_dentry->ino = inode_num;
    sbi_sd_write(kva2pa(tmp), 1, FS_START + DATA_NUM_OFFSET +
block_num);
    kmemset(tmp, 0, 512);
    kmemcpy(root_dentry->name, "..", 2);
    root_dentry->type = T_DIR;
    root_dentry->ino = inode_num;
    sbi_sd_write(kva2pa(tmp), 1, FS_START + DATA_NUM_OFFSET + block_num
+ 1);

    prints("[FS] Initialize filesystem finished.\n");
}
```

2. 文件操作设计

创建一个文件需要为新文件分配一个 inode，初始化 inode 中的内容并分配一个 block 给新文件，修改父目录的修改时间和大小，修改 inodemap 和 blockmap 使用情况。

删除文件则需要进行 inode 和 block 的回收。

3. 目录操作设计

以文件系统执行 ls 命令查看一个绝对路径时的操作流程为例

```
void do_fs_ls(char *name){
    if(!check_fs()){
        prints("ERROR: no filesystem\n");
        return ;
    }
    dentry_t dy_match;
    if(name[0] == '/'){
        dy_match = find_dir(0, &name[1]);
    }else{
        dy_match = find_dir(global_inode_num, &name[1]);
    }
    if(dy_match.ino == -1){
        prints("ERROR: no such directory\n");
        return ;
    }

    u8 tmp[512];
    superblock_t *sb = tmp;
```

```
    sbi_sd_read(kva2pa(sb), 1, FS_START);
    u32 fs_block_start = sb->fs_start_block;
    u32 inode_block_start = sb->inode_block_offset + fs_block_start;
    u32 data_block_start = sb->data_block_offset + fs_block_start;
    inode_t *inode_p = (inode_t *)tmp;
    sbi_sd_read(kva2pa(inode_p), 1, inode_block_start + dy_match.ino);
    u8 dentry_buffer[512];
    for(int i = 0; i < inode_p->used_size; i++){
        dentry_t *dy = (dentry_t *)dentry_buffer;
        sbi_sd_read(kva2pa(dy), 1, data_block_start + inode_p->direct[i
/ 8] +  i % 8);
        if(dy->ino >= 0){
            if(dy->type == T_DIR){
                prints("[DIR]  %s\n", dy->name);
            }
            else if(dy->type == T_FILE){
                prints("[FILE] %s\n", dy->name);
            }
        }
    }
}
```