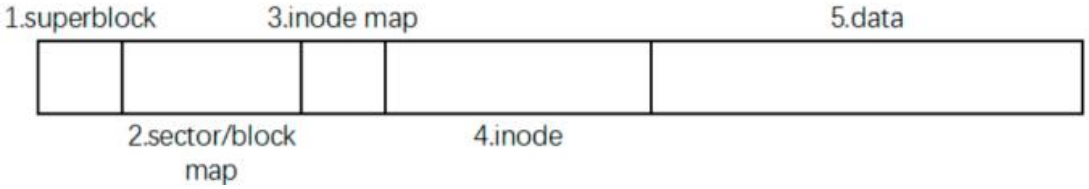
Project 6 File System 设计文档

中国科学院大学
[薛峰]
[2019/1/8]

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/sector map，inode table 以及数据区各自占用的磁盘空间大小。

答：整个文件系统大小为 1G，分布如下：



其中，superblock 在磁盘的 512MB 处，占据 1 个 block，即 4KB；
block_map 块有 8 个 block，占 32KB；
inode_map 块有 1 个 block，占 4KB；
inode 块有 246 个 block，占 4 * 246 KB；
其余为数据块，占 1G - 1M.

Superblock	Block Bitmap	Inode Bitmap	Inode	Data
1 Block 4KB	8 Block 32KB	1 Block 4KB	246 Block	1023*256

（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。

答：inode 块如右图所示。

```
typedef struct inode_t{
    int  mode;
    int  file_size;
    int  file_indirect_blocks_num;

    type file_type;
    int  dentry_num;

    int  blocks_number;
    int  direct_block[DIRECT_BLOCK_NUM];
    int  indirect_block;
    int  links;

    uint64_t create_time;
    uint64_t last_access_time;
    uint64_t last_modified_time;
} inode_t;    // 64B
```

其中，mode 域用来区分文件和目录；

File_size 域用来表明目录/文件的大小；

File_indirect_block_num 用来表明目录/文件一级间址指针的数目；

File_type 用来表明文件的类型；

Block_number 用来表明该目录/文件的数据块数；

Direct_block[DIECT_BLOCK_NUM] 是间址指针；

Indirect_block 是一级间址指针；

Links 为连接数；

Create_time 为创建时间；

Last_access_time 为最后访问时间；

Last_modified_time 为最后修改时间；

superblock 数据结构如右图：

```
typedef struct superblock_t
{
    uint32_t magic;
    uint32_t fs_size;
    uint32_t used_block_num;
    uint32_t fs_start_addr;
    uint32_t fs_end_addr;

    uint32_t block_map_num;
    uint32_t block_map_size;
    uint32_t block_map_start;

    uint32_t inode_map_num;
    uint32_t inode_map_size;
    uint32_t inode_map_start;

    uint32_t inode_num;
    uint32_t inode_size;
    uint32_t inode_start;

    uint32_t data_start;
    uint32_t block_size;
    uint32_t dentry_size;

} superblock_t;
```

其中 magic 域为文件系统的标识，用于区分不同的文件系统，并且在初始化的时候可以检测出磁盘中是否已经存在一个文件系统；

Fs_size 域表明整个文件系统的大小；

Used_block_num 表示已经用的块的数目；

Fs_start_addr 记录文件系统在磁盘上的起始位置，即 superblock 的位置；

Fs_end_addr 记录文件系统在磁盘上的末尾位置；

Block_map_num 表明 block_bitmap 块的 block 数目；

Block_map_size 表明 block_bitmap 块的大小；

Block_map_start 表明 block_bitmap 块的起始位置；

inode_map_num 表明 inode_bitmap 块的 block 数目；

inode_map_size 表明 inode_bitmap 块的大小；

inode_map_start 表明 inode_bitmap 块的起始位置；

inode_num 表明 inode 块的 block 数目；

inode_size 表明 inode 块的大小；

Inode_start 表明 inode 的起始位置；

Data_start 表明数据块的起始位置；

Blcok_size 表明文件系统中一个 block 的大小；

Dentry_size 表明文件系统中一个目录项的大小。

一个数据块大小为 4KB，一个指针大小为 4B，因此一个数据块最多可以存储 1KB 个指针。所以一个一级间址指针可以索引的最大文件为：4K * 1KB = 4MB。又因为还有两个直接指针，因此该文件系统的最大文件大小为 4MB + 8KB。

因为 inode 的大小为 64B，并且 inode 块大小为 246 * 4K，因此一共的 inode 的数目为：15744 个，所以最多的文件数目为 15744 个。

一个目录项大小为 32B，因此一个数据块最多可以存储 128 个目录项。所以两个直接指针能够表示 256 个目录项，一个一级间址指针能够索引 1K * 256 = 256K 个指针，但是 256K 已经超出 inode 的数目，因此一个目录下最多文件/子目录数目与最多的 inode 数目相同，及 15744 个。

（3）设计或实现过程中遇到的问题和得到的经验。

该项目代码量比较多，稍微不细心便会出现错误，因此花费了很多时间在 debug 上面，并且都是一些效的错误，所以以后还是要多注意，尽量不犯低级错误。

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。

首先要对新的文件分配 inode，这就要求对这个 inode 进行初始化操作。

其次，要修改其所在目录的 inode，必定修改的有：file_size，dentry_num，links，

last_access_time，last_modified_time。如果当前存放目录项的 block 块满了的话，会修改 file_indirect_block_num，block_num，可能还会修改 Direct_block[DIECT_BLOCK_NUM]，Indirect_block。

（3）设计或实现过程中遇到的问题和得到的经验。

做完 task1 之后，task2 较为简单。

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。

首先对输入的路径进行分析，将绝对路径分为多个目录名，假设输入为 A/B，则分解成 A、B。然后从根目录开始，找到当前所在目录的 i-node（寻找的过程与下文类似），然后从该 i-node 找到其对应的数据块，从该目录里面搜索名字是 A 的目录项，从而找到目录 A 的 ino，根据该 ino 找到对应的 i-node，从该 i-node 可以找到目录 A 的数据块。从该数据块中寻找名字是 B 的目录项，找到 B 的 ino。从而读取 B 的 i-node，在 B 的 i-node 中找到其数据块，将 B 的数据块中的每一个目录项的名字打印出来，即完成了 ls 指令。

（2）设计或实现过程中遇到的问题和得到的经验

通过该实验，对文件系统的理解更加深刻，尤是知道了如何一级一级地查找一个文件。