OrangeFS改造说明文档

版本：V0.2

最后修改时间：2023-6-10

一、本次改造的主要内容

1. **移植了目录树。**OrangeFS开始支持目录结构，能支持目录的创建、删除，在目录下创建、删除文件。
2. **VFS层能够根据文件路径来确定该文件所挂载的文件系统，将文件路径由绝对路径转化为所挂载的文件系统的相对路径。**
3. **OrangeFS从原来以扇区为数据的管理的最小单位变成以数据块为基本单位。**
4. **字符设备、块设备文件的存储放在了`/dev`目录下，对块设备文件进行了完善。 新添加了初始化字符设备的系统调用，**将创建tty文件的任务交给kernel initial而不是OrangeFS的外部format程序；**修改了初始化块设备的系统调用**，块设备文件不再占用硬盘数据块，块设备文件的`inode`中`i_start_block`成员变量用于记录块设备的设备号。修改了`mount`系统调用，mount系统调用会获取指定的块设备文件的`inode`，从中获取到挂载设备的设备号。

二、目录树功能的说明

目录树功能的添加主要涉及到三方面的改动：在目录树中搜索文件或目录、新建目录、删除目录；

2.1 在目录树中搜索文件/目录

`int strip_path(char *filename, const char *pathname, struct inode **ppinode)`

`strip_path`的功能是根据文件/目录的路径找到该路径指向的文件的父目录的inode并将文件名从路径中分离出来。

`char *filename`是函数的输出，指向存储文件名的数组，是函数的输出。

`char *pathname`是路径名，作为函数的输入。

`struct inode **ppinode`指向路径描述的文件的父目录的inode指针，作为函数的输出。

该函数返回0表示正常返回，返回-1表示路径有误。

例子：查找文件`/dir1/dir2/file.txt`时；`pathname`指向的字符数组`/dir1/dir2/file.txt`,函数正常执行时会返回0值，`filename`指向字符串`file.txt`,`ppinode`储存着目录`dir2`的inode地址。

`strip_path`函数与修改之前功能相同，但是实现起来有了很大变化。引入目录树后，`strip_path`会从路径的第一个目录开始，一层一层的搜索目录（调用`find_dir_inode_cur_dir`），搜到最后一层时填充对应的`filename`和`ppinode`指针。

`int search_file_in_dir(char *name, struct inode *dir_inode)`

`search_file_in_dir`功能是在父目录的下搜索对应的文件。

`dir_inode`是父目录的inode指针。

`name`是文件名

返回值是该文件对应的inode id;

在搜索文件时通常会先调用`strip_path`将文件名剥离出来并且找到该文件的父目录，然后在父目录下搜索对应的文件。

2.2 新建目录

`int ora_createdir(MESSAGE *fs_msg)`

该函数会先检查创建的目录是否存在，若不存在则分配inode（`alloc_imap_bit`）、分配扇区(`alloc_smap_bit`)、初始化inode(`new_inode`)、在父目录中添加一个目录项(`new_dir_entry`).

2.3 删除目录

```
int real_deletedir(struct super_block *sb, const char *pathname)
{
    MESSAGE fs_msg;

    fs_msg.type = DELETEDIR;
    fs_msg.PATHNAME = (void *)pathname;
```

```
    fs_msg.NAME_LEN = strlen(pathname);
    fs_msg.source = proc2pid(p_proc_current);
    int flag;
    flag = do_deletecheck(&fs_msg);
    if (flag) // 如果为错误码
    {
        return flag;
    }
    return ora_deletedir(&fs_msg);
}
```

`real_deletedir`由`vfs_deletedir`调用，先检查是否可以删除（调用`do_deletecheck`），`do_deletecheck`主要是检查路径指向的目录是否存在、路径指向的目录的`i_mode`是不是目录类型、以及目录的inode的`i_cnt`是不是为1。`i_cnt`在创建目录时会被设置成1，在该目录下新建文件或者目录时需要将`i_cnt`+1

`int ora_deletedir(MESSAGE *fs_msg)`

该函数负责执行目录删除的具体操作，会删掉待删目录的父目录中存储的对应dentry记录，释放待目录的inode和占用扇区数。

三、VFS层修改的详细说明

`inode`中新增了一个成员变量`i_mnt`，当`inode`描述的文件是一个挂载点时，`i_mnt`存储的是`mnt_table`中的索引。在执行mount系统调用时，会给该变量赋值。

orangefs中新增了一个函数`int vfs_path_transfer(char *path, int *fs_index)`，该函数主要功能是根据传入的文件路径`path`来确定该文件所在的文件系统是哪个，并且将文件路径由绝对路径转化为挂载的文件系统内的相对路径。该函数具体实现是。根据`path`的路径一级一级的打开目录，查看目录的`inode`中`i_mode`是否为挂载点类型；若是挂载点类型，则根据`inode`中`i_mnt`变量记录的索引去访问`mnt_table`,从中获取到该挂载点挂载的文件系统在`vfs_table`中的索引。之后就将绝对路径中将挂载点的路径删掉，从而完成了绝对路径到文件系统内的相对路径的转化。`kern_vopen`等vfs向系统调用提供的接口的实现中，会根据`vfs_path_transfer`函数所确定的具体文件系统，调用该文件系统的`open`函数，将转化后的相对路径传进去，完成具体的`open`等操作。

VFS与具体文件系统的接口函数的参数发生了修改.

```
struct file_op{
    int (*create)   (struct super_block *,const char*);
    int (*open)    (struct super_block *,const char* ,int);
    int (*close)   (int);
    int (*read)    (int,void * ,int);
    int (*write)   (int,const void* ,int);
    int (*lseek)   (int,int,int);
    int (*unlink)  (struct super_block *,const char*);
    int (*delete) (struct super_block *,const char*);
    int (*opendir) (struct super_block *,const char *);
    int (*createdir) (struct super_block *,const char *);
    int (*deletedir) (struct super_block *,const char *);
    int (*readdir) (struct super_block *,char*, int*, char*);
    int (*chdir) (struct super_block *,const char*); //added by ran
    //int tag;
    }
```

主要是一些接口中添加了`superblock`类型的指针变量。添加该变量的原因是遇到了像FAT32这种类型的文件系统可以格式化到多个块设备中，但是open等函数无法与具体格式化的设备绑定，`superblock`描述了一个设备的具体文件系统,在VFS层调用具体文件系统的接口时需要提供`superblock`。

四、OrangeFS变为以块作为基本读写单位

修改前的OrangeFS以扇区作为读写单位和管理单位，每次读写读写一个扇区，分区的0号扇区作为启动扇区、1号扇区作为superblock所在的扇区。

修改后OrangeFS以数据块作为基本的读写单位和管理单位，每个块大小为4KB（8个扇区），每次读写都是以块为单位进行读写，0号块作为启动块、1号块作为superblock所在的块。inode和superblock中原先描述扇区相关的成员变量都变成了描述块相关的成员变量。新建文件时每次会分配256个数据块（1MB的空间）。

对应的外部format程序和cloader以及orangefs_boot都进行了相应的修改。

五、重写了字符设备文件和块设备文件的创建

重写了`init_block_dev`对应的系统调用、新增了`init_char_dev`，这两个系统调用都是在根文件系统的/dev目录下创建对应块设备文件或者字符设备文件。

`init_block_dev`系统调用用于创建块设备文件。块设备文件的文件命名与linux一致，比如sda1、hd1等，这些文件全部放在/dev目录下，块设备文件分配硬盘中的存储空间，块设备文件的inode的`i_start_block`成员变量存储的是该块设备文件描述的块设备的设备号，当执行mount操作时，mount会读取该成员变量以确定要挂载的设备而不是修改之前那种根据sda1这个文件名来一步步确定设备号。

字符设备（tty设备）文件之前是在orangeFS外部format程序中创建的，现在的版本中将tty字符设备文件的创建操作放在kernel启动之后，通过`init_char_dev`系统调用在/dev目录下创建三个tty设备文件。