Nuttx FS

原文 http://blog.csdn.net/chengwenyang/article/details/70213085 2017-04-17 19:02:41 6 ℃ 0 评论

This article present how the FS in Nuttx works, we will use fat fs as the example to explain it, at the last, this article shows the mount process of MMC block driver.

inode

inode is one of the basic data structure in Nuttx OS

```
struct inode {
    struct inode *i_peer;
    struct inode *i_child;
    int_16_t        i_crefs;
    unit16_t        i_flags;
    union inode_ops_u u;
    mode_t      i_mode;
    void        *i_private;
    char            i_name[1];
}
```

the member of inode structure is self-explained

i_peer points to its brother inode at the same level.
i_child points to its child inode
i_crefs is used to track how many user are access the inode
i_flags is a bit mask to distinguish the type of the inode, usually it may be one of three types, drive, block or mountpoint. besides, if the inode is deleted, its i_flag domanin should be colored.
inode_ops_u is an union structure, if the inode is a driver inode, this domain is the file operation pointer for operating the driver, if the inode is block device, then inode_ops_u is the pointer to the block device’ s file operation structure. similarly, if the inode represent the mount point, this domain pointes to the mount point file operation.
i_private is a private pointer which will be used if the driver needs it
the last domain is the name of the inode

For example, the devices under /dev are registered in the system when we call **register_blockdriver** or **register_driver**.

let’ s take register_blockdriver as example:

```
int register_blockdriver(const char *path, struct block_operation *bops, mode_t mode, void *priv)
{
    ...
    ret = inode_reserve(path, &node);
    if (ret >= 0) {
        node->u.i_bops = bops;
        node->i_mode = mode;
        node->i_private = priv;
        ret = OK;
```


标签列表

Java Android JavaScript Linux
Python 程序员 HTML iOS PH
跨平台 mysql 数据库 创业
iOSDeveloper 算法 iOS开发 CC
Oracle Windows C语言 Object
android开发 首页投稿 Shell Sp
LeetCode JS 数据结构 Ubuntu
jQuery acm android知识 机器学
设计 SQL Swift 设计模式 Had

```
        return ret;
    }
}
```

the first operation of registering block driver is to reserve an inode to representing the driver, it can be /dev/led, or /dev/mmcsd0. etc. the path parameter contains the name of the inode, it will be /dev/mmcsd0 if the path param here is "/dev/mmcsd0". Then it initialize its ops, mode and priv domains. for an MMC device, its ops member is the block_operation for mmc. member priv records the info for the mmcdevice in the mmc instance.

After registering the driver, one inode will be allocated for the device in inode tree.

file

the high level concept of file inherents from Linux(unix), everything is file. the file in Nuttx is very simple.

```
struct file {
    int f_oflags; // open mode
    off_t f_pos; // file position
    struct inode *f_inode; // driver interface
    void *f_priv;
}
```

f_oflags is open mode such as read mode or write mode, then the file inode should support read or write opreation methods
f_pos is the position of the file being operating
f_inode points to the inode that the path param in open system call described
f_pirv is a private pointer for the file operation. Usually the domain can be used to contain user specific data structure.

each task has its own file descriptor allocation pool in Nuttx, the file descriptors of two or more task s may be same, but they don't share the same file.

Mount block driver to FS

we can't access the block device untill we mount it to the system FS because the Nuttx uses VFS in order ot support multiple file system, such fat32, romfs. VFS abbstract the common properties of file operation, the difference between the underline media is handled by the specific file system.

The protype of mount is:

```
int mount(const char *source, const char *target, const char *filesystemtype, long mountflags, void *data)
```

For example if we mount "/dev/mmcsd0" to "/mnt" with vfat file system type, the command line is:

```
#mount -t vfat /dev/mmcsd0 /mnt
```

In this example, the source is "/dev/mmcsd0", the target is "/mnt", file system type is "vfat".

the mount operation will create a new inode for the mounted file, and then use the bind method of the filesystem type, for example vfat, to bind the device/drive to the file system.

we introduce a data structure fat_mountpt_s to describe the new fat mount point which will be binded to file system.

```
struct fat_mountpt_s {
    struct inode *fs_blkdriver; // the block driver inode that hosts the FAT32 fs
    struct fat_file_s *fs_head; // a list to all files opened on this mount point
    sem_t fs_sem; // used to assume thread-safe access
    off_t fs_hwsectorsize; // HW: sector size reported by the block driver
    off_t fs_hwnsetors; //HW:  the number of sectors reported by the hardware
    off_t fs_fatbase; // logical block of start of filesystem (past resd sectors)
    off_t fs_rootbase; // MBR: Cluster no of 1st cluster of root dir
    off_t fs_data_base; // logical block of start data sectors
    off_t fs_fsinfo; // MBR: Sector number of FSINFO sector
    off_t fs_currentsector; // the sector number buffered in the fs_buffer
    uint32_t fs_nclusters; // Maximum number of data colusters
```


Kubernetes
配置与服务部

kubernetes
集群部署

用自己的数据
Faster-RCNN

微信小程序之
Material

内网嗅探与欺
探（1）

一键安装Doc
图形化管理界

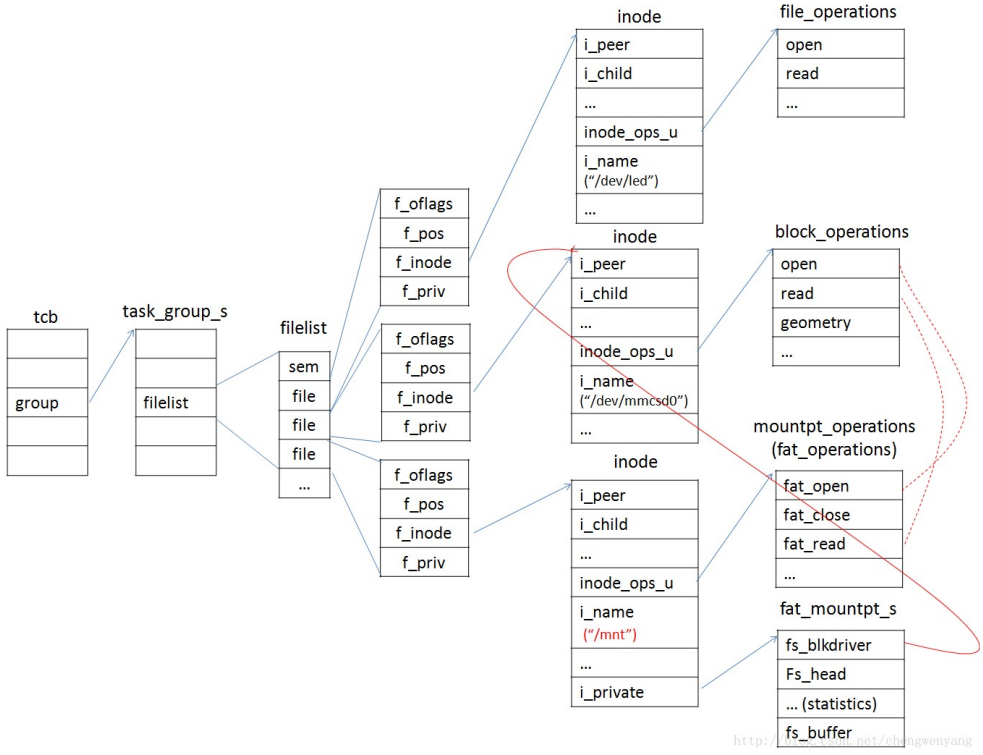
electron-vue
webpack: A

```
    uint32_t fs_nfatsecoters; // MBR: Count of sectors occupied by one fat
    uint32_t fs_fattotsec; // MBR: Total count of sectors on the volume
    uint32_t fs_fsifreecount; // FSI: last free cluster count on the volume
    uin32_t fsinextfree; // FSI: Cluster number of 1st free cluster
    uint16_t fs_fatrewvdeccount; // MBR:  The Total number of reserved sectors
    uint16_t fs_rootentcnt; // MBR: Count of 32-bit root derecotry entries
    bool fs_mounted // true: the file system is ready
    bool fs_dirty; // true: the buffer is dirty
    bool fs_fsidirty; // true: FSINFO sector must be written to disk
    uint8_t fs_type // FSTYPE_FAT12, FSTYPE_FAT16, FSTYPE_FAT32
    uint8_t fs_fatnumfats; // MBR:Number of FATs probably 2
    uint8_t fs_fatsecperclus; // MBR: Sector per allcoation unit: 2**n, n=0,..7
    uint8_t *fs_buffer; // this is the allcoated buffer to hold one sector from the device
}
```

the first member is the pointer to the block dirver that this mount point is driven
the second is a list files that was opened from this mount point
The rest are the meat data of about the media or the control flags
the mount process we can image does the following things:
allocate new inode for the mount point
allocate specific fs mount descriptor for the new mount point, for examle structure fat_mout_s
point fs_blkdriver in fat_mount_s to the block inode which will be binded.
read the media to get the info of fat fs on the media
point the mount point ops to the new inode’ s ops domain, for example fat_operations
and point the new inode’ s priv member to the specific fs, for example structure fat_mountpt_s

at this point, the new fs is mounted to the system. we can use the general operation method, like read, write... to access the underlying media by mount_point_inode->private->fs_blockdriver->read(write. etc), but actually it doesn’ t use this way.

> **a breif view of the data structures within Nuttx FS**



In Nuttx system, basicly there are three kinds of file operations, **file_operation**, **block_operation** and **mountpt_operation**.

> **file_operation**

file_operation is used for general character device driver like an LED, mems sensors, it porvides methods to access these devices.

> **block_operation**

Block device is different from character device, we can only read or write the device sequentially, for example, read 512 bytes from the front of one file in SD card, so we have to locates the file from fatfs and get the meta info of the file, then we strart to read the file. without the help of fatfs, we even don’ t know where the file is in the SD card. The block_operation has methods to access the media, but we shouldn’ t use these methods directly, and why?

> **mountpt_operation**

mount point operations is designed as intermedium between block driver and FS, its operation methods are binded with the specific fs, for example fatfs, romfs.

Through the mount point operation has read/write method, but they don’ t touch the underlying media directly, as the fact, they recall the corresponding block driver method.

Tags : Nuttx file-syste linux Nuttx Linuxkernel file-syste

上一篇：linux安全配置检查脚本（持续更新中）　　下一篇：struts2 poi和struts2 jxl实现读取EXCEL

猜你喜欢

- locale 命令查看环境语言 2017-04-18
- dos2unix详解 2017-04-17
- [看门狗]内部看门狗和外部看门狗 2017-04-17
- 【Linux】了解服务器的情况 2017-04-17
- Intellij Idea配置MapReduce编程环境 2017-04-17
- Linux内存管理相关 2017-04-17
- shell脚本必知会 2017-04-17
- Linux学习笔记——进程间的通信-文件和文件锁 2017-04-17
- linux:输入输出重定向 2017-04-17
- linux中查看软件文件安装路径 2017-04-17