

CMJFS

制作者 Doxygen 1.9.2

1 模块索引	1
1.1 模块	1
2 结构体索引	3
2.1 结构体	3
3 文件索引	5
3.1 文件列表	5
4 模块说明	7
4.1 配置项	7
4.1.1 详细描述	7
4.2 核心系统	7
4.2.1 详细描述	8
4.3 文件系统	8
4.3.1 详细描述	8
4.4 结构	8
4.4.1 详细描述	9
4.4.2 类型定义说明	9
4.4.2.1 block_t	9
4.4.2.2 dir_t	10
4.5 FS行为	10
4.5.1 详细描述	11
4.5.2 函数说明	11
4.5.2.1 acq_blk()	12
4.5.2.2 add_blk_for_file()	12
4.5.2.3 cd()	12
4.5.2.4 creat()	13
4.5.2.5 creat_dir()	13
4.5.2.6 creat_dirent()	14
4.5.2.7 creat_file()	14
4.5.2.8 creat_ino()	14
4.5.2.9 creat_stat()	15
4.5.2.10 find_file()	15
4.5.2.11 find_name_in_dir()	15
4.5.2.12 link_file()	16
4.5.2.13 ls()	16
4.5.2.14 mkdir()	17
4.5.2.15 open_file()	17
4.5.2.16 pwd()	18
4.5.2.17 read_file()	18
4.5.2.18 rm()	18
4.5.2.19 rm_dir_item()	19

4.5.2.20 rmdir()	19
4.5.2.21 write_file()	20
4.6 位图	20
4.6.1 详细描述	21
4.7 用户和权限	21
4.7.1 详细描述	21
4.7.2 变量说明	21
4.7.2.1 idle_uid	22
4.8 权限	22
4.8.1 详细描述	23
4.8.2 函数说明	23
4.8.2.1 access()	23
4.9 行为	23
4.9.1 详细描述	24
4.9.2 函数说明	24
4.9.2.1 add_user()	24
4.9.2.2 find_user()	24
4.9.2.3 login()	25
4.10 交互终端	25
4.10.1 详细描述	26
4.11 交互接口	26
4.11.1 详细描述	26
4.11.2 函数说明	26
4.11.2.1 add_history()	26
4.11.2.2 readline()	26
4.11.2.3 rl_gets()	27
4.12 指令	27
4.12.1 详细描述	28
4.12.2 宏定义说明	28
4.12.2.1 CMD_N	28
4.13 指令行为	28
4.13.1 详细描述	29
5 结构体说明	31
5.1 __block_num_validator结构体 参考	31
5.2 __block_size_validator结构体 参考	31
5.3 __indirect_idx_1_num_validator结构体 参考	31
5.4 __inode_num_validator结构体 参考	32
5.5 block联合体 参考	32
5.5.1 详细描述	33
5.6 cmd结构体 参考	33
5.6.1 详细描述	33

5.7 dir结构体 参考	33
5.7.1 详细描述	34
5.8 dirent结构体 参考	34
5.8.1 详细描述	34
5.9 dirents结构体 参考	35
5.9.1 详细描述	35
5.10 inode结构体 参考	35
5.10.1 详细描述	36
5.11 stat结构体 参考	36
5.11.1 详细描述	37
5.12 user结构体 参考	37
5.12.1 详细描述	38
6 文件说明	39
6.1 include/config.h 文件参考	39
6.1.1 详细描述	40
6.2 config.h	40
6.3 include/doc.h 文件参考	41
6.3.1 详细描述	41
6.4 doc.h	41
6.5 cmd.h	42
6.6 include/shell/io.h 文件参考	42
6.6.1 详细描述	43
6.7 io.h	43
6.8 include/shell/shell.h 文件参考	43
6.8.1 详细描述	44
6.9 shell.h	44
6.10 include/sys/fs/fsops.h 文件参考	44
6.10.1 详细描述	46
6.11 fsops.h	47
6.12 include/sys/fs/mediactrl.h 文件参考	47
6.12.1 详细描述	49
6.13 mediactrl.h	49
6.14 include/sys/fs/types/block.h 文件参考	49
6.14.1 详细描述	50
6.15 block.h	50
6.16 include/sys/fs/types/dir.h 文件参考	51
6.16.1 详细描述	52
6.17 dir.h	52
6.18 include/sys/fs/types/dirent.h 文件参考	52
6.18.1 详细描述	53
6.19 dirent.h	53

6.20 include/sys/fs/types/dirents.h 文件参考	53
6.20.1 详细描述	54
6.21 dirent.h	54
6.22 include/sys/fs/types/inode.h 文件参考	55
6.22.1 详细描述	56
6.23 inode.h	56
6.24 include/sys/fs/types/stat.h 文件参考	56
6.24.1 详细描述	57
6.25 stat.h	58
6.26 include/sys/permission.h 文件参考	58
6.26.1 详细描述	59
6.27 permission.h	59
6.28 include/sys/user.h 文件参考	60
6.28.1 详细描述	60
6.29 user.h	61
Index	63

Chapter 1

模块索引

1.1 模块

这里列出了所有模块:

配置项	7
核心系统	7
文件系统	8
结构	8
FS行为	10
位图	20
用户和权限	21
权限	22
行为	23
交互终端	25
交互接口	26
指令	27
指令行为	28

Chapter 2

结构体索引

2.1 结构体

这里列出了所有结构体，并附带简要说明：

__block_num_validator	31
__block_size_validator	31
__indirect_idx_1_num_validator	31
__inode_num_validator	32
block	
定义文件系统数据块。	32
cmd	
描述一个指令条目。	33
dir	
描述目录结构。	33
dirent	
描述一个项目的入口。	34
dirents	
扩展目录结构。	35
inode	
描述inode结点。	35
stat	
描述一个inode的基本信息。	36
user	
描述一个用户的信息。	37

Chapter 3

文件索引

3.1 文件列表

这里列出了所有文档化的文件，并附带简要说明：

include/config.h	定义基本数据，一部分可修改。	39
include/doc.h	帮助doxygen生成文档。 **不要** 试图包含该文件！	41
include/shell/cmd.h	42
include/shell/io.h	提供交互功能。	42
include/shell/shell.h	定义交互终端的核心。	43
include/sys/permission.h	文件权限的验证操作。	58
include/sys/user.h	定义用户信息和用户操作。	60
include/sys/fs/fsops.h	文件系统的核心操作。	44
include/sys/fs/mediactrl.h	实现基本的介质管理。	47
include/sys/fs/types/block.h	定义文件系统块。	49
include/sys/fs/types/dir.h	定义struct dir结构。	51
include/sys/fs/types/dirent.h	定义struct dirent结构。	52
include/sys/fs/types/dirents.h	定义struct dirents。	53
include/sys/fs/types/inode.h	定义inode。	55
include/sys/fs/types/stat.h	定义struct stat结构。	56

Chapter 4

模块说明

4.1 配置项

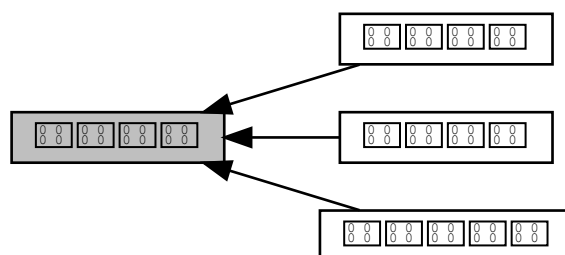
宏定义

- **#define BLOCK_SIZE 512**
指定块尺寸。必须是2的幂。
- **#define BLOCK_NUM 512**
指定块数目。必须是正数。
- **#define INODE_NUM 512**
指定 *inode* 数目。必须是正数。
- **#define INDIRECT_IDX_1_NUM 10**
指定一级索引（直接索引？）数量。目前必须是10，因为修改的话大概率出 *bug*（因为我无法分辨那些不明所以的字面值）。
- **#define MAX_DIRLIST 20**
写了，但没有卵用的东西。你尽管改，起作用算我输。（这是你的代码，我没改。）

4.1.1 详细描述

4.2 核心系统

核心系统 的协作图:



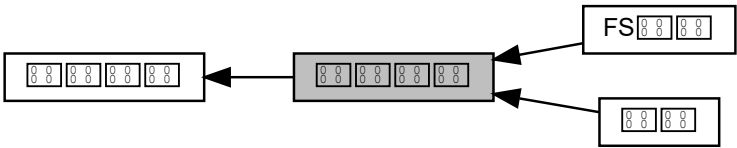
模块

- 文件系统
- 用户和权限
- 交互终端

4.2.1 详细描述

4.3 文件系统

文件系统的协作图:



模块

- 结构
- FS行为

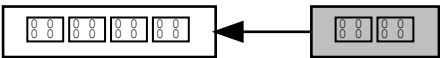
变量

- `block_t BLK []`
全局块数据。
- `inode_t inode []`
全局inode数据。

4.3.1 详细描述

4.4 结构

结构的协作图:



结构体

- union `block`
定义文件系统数据块。
- struct `dir`
描述目录结构。
- struct `dirent`
描述一个项目的入口。
- struct `dirents`
扩展目录结构。
- struct `inode`
描述`inode`结点。
- struct `stat`
描述一个`inode`的基本信息。

类型定义

- typedef union `block block_t`
定义文件系统数据块。
- typedef struct `dir dir_t`
描述目录结构。
- typedef struct `dirent dirent_t`
描述一个项目的入口。
- typedef struct `dirents dirent_t`
扩展目录结构。
- typedef struct `inode inode_t`
描述`inode`结点。
- typedef struct `stat stat_t`
描述一个`inode`的基本信息。

4.4.1 详细描述

4.4.2 类型定义说明

4.4.2.1 `block_t`

```
typedef union block block_t
```

定义文件系统数据块。

NOTE: 我们真的**非常不建议**这样使用union! 初始化一个域后访问其他域是UB。建议额外用一个字段记录union里面存的到底是什么, 或者干脆改用类似`void*`的东西。

NOTE2: 我在这里做了少许修改。你内嵌定义了一个union, 但没有定义变量。认真的? NOTE3: 鉴于实现比较怪(真正的`dentry`肯定不是这么存的), 不进行`block_t`是否能存下`dir_t`和`dirents_t`的校验。

4.4.2.2 dir_t

```
typedef struct dir dir_t
```

描述目录结构。

本质上是产生一个树形数据结构。

注意：记得依次释放dir_list的内容，以免内存泄漏。

NOTE: ext4底层真的是这样的吗？我不确定，以后看看

4.5 FS行为

FS行为的协作图:



模块

- [位图](#)

宏定义

- **#define S_IFREG** 0x0100000
普通文件标识
- **#define S_IFDIR** 0x0040000
目录文件标识
- **#define RECURSIVE** 0x800
递归行为标识
- **#define GET_BLK**(ino, n) (BLK[inode[ino].blk1[n]])
获取文件的第n个块
- **#define GET_PAGE**(ino, page) (page >= 10 ? BLK[inode[ino].blk2].index[page-10] : inode[ino].blk1[page])
获取文件的第n个块
- **#define GET_CHAR**(ino, page, offset) (BLK[GET_PAGE(ino,page)].str[offset])
获取字符(?)

函数

- int `add_blk_for_file` (int ino)
为文件申请新的数据块。
- int `find_name_in_dir` (int ino, const char *name, int mode)
在指定目录下查找具有指定名称的文件。
- int `creat_stat` (const `user_t` *user, int ino, int mode)
创建文件的`stat`。
- int `creat_dirent` (int fno, int ino, const char *name)
为文件创建目录入口。
- int `creat_dir` (int ino, int f_ino, const char *dir_name)
为文件分配`struct dir`结构。
- int `creat_ino` ()
分配`inode`。
- int `creat` (const `user_t` *user, int cwd, const char *args, int mode)
创建文件。
- int `mkdir` (const `user_t` *user, int cwd, const char *dir_name, int mode)
创建目录。
- void `ls` (int ino, char *args)
查看目录信息。
- void `pwd` (int cwd, char *buf, int ino)
取得指定`inode`的目录名。
- void `cd` (const `user_t` *user, int *cwd, const char *dir_name)
修改指定用户的工作目录。
- int `rm_dir_item` (`dir_t` *d, int n)
删除指定目录下的第`n`项。
- void `rm` (const `user_t` *user, int cwd, const char *args, int workIno, int mode)
删除指定文件。
- void `rmdir` (const `user_t` *user, int cwd, char *dir_name, int mode)
删除目录。
- void `creat_file` (const `user_t` *user, int cwd, const char *args)
创建文件。
- int `open_file` (const `user_t` *user, int cwd, const char *args)
打开文件。
- int `read_file` (const `user_t` *user, int cwd, int argc, const char *argv[])
读取文件。
- int `write_file` (const `user_t` *user, int cwd, int argc, const char *argv[])
写入文件。
- int `link_file` (int cwd, int argc, const char *argv[])
创建硬链接。
- void `find_file` (int cwd, const char *args, int ino)
查找文件。
- int `acq_blk` ()
申请一个新块。

4.5.1 详细描述

4.5.2 函数说明

4.5.2.1 acq_blk()

```
int acq_blk ( )
```

申请一个新块。

NOTE: 这个东西实际是申请而不是创建块，创建块是格式化做的事情。所以我把函数名改了。

返回

int 如果成功，返回块号，否则返回-1。

4.5.2.2 add_blk_for_file()

```
int addblk_for_file (
    int ino )
```

为文件申请新的数据块。

参数

<i>ino</i>	文件的inode
------------	----------

返回

int 新数据块的id，失败返回-1。

4.5.2.3 cd()

```
void cd (
    const user_t * user,
    int * cwd,
    const char * dir_name )
```

修改指定用户的工作目录。

参数

<i>user</i>	用户
<i>cwd</i>	接收修改后工作目录的inode编号
<i>dir_name</i>	目标目录

4.5.2.4 creat()

```
int creat (
    const user_t * user,
    int cwd,
    const char * args,
    int mode )
```

创建文件。

参数

<i>user</i>	文件属主
<i>cwd</i>	工作目录
<i>args</i>	参数（？不应该是name？）
<i>mode</i>	文件权限

返回

int 已有文件但不可写返回0，创建失败返回-1，成功创建返回inode编号。

4.5.2.5 creat_dir()

```
int creat_dir (
    int ino,
    int f_ino,
    const char * dir_name )
```

为文件分配struct dir结构。

会分配.和..目录。

NOTE：你给我的注释写着返回块号，但你自己看看，你返回了锤子块号

参数

<i>ino</i>	当前目录的inode
<i>f_ino</i>	父目录的inode
<i>dir_name</i>	当前目录文件名

返回

int 成功返回0，否则返回-1。

4.5.2.6 creat_dirent()

```
int creat_dirent (
    int fino,
    int ino,
    const char * name )
```

为文件创建目录入口。

具体而言：在*fino*指向的目录下，为*ino*文件分配一个名为*name*的入口。

参数

<i>fino</i>	父目录的inode编号
<i>ino</i>	子目录的inode编号
<i>name</i>	文件名

返回

int 成功返回0，否则返回-1。

4.5.2.7 creat_file()

```
void creat_file (
    const user_t * user,
    int cwd,
    const char * args )
```

创建文件。

参数

<i>user</i>	创建者
<i>cwd</i>	工作目录
<i>args</i>	文件名

4.5.2.8 creat_ino()

```
int creat_ino ( )
```

分配inode。

返回

int 成功返回inode编号，失败返回-1。

4.5.2.9 creat_stat()

```
int creat_stat (
    const user_t * user,
    int ino,
    int mode )
```

创建文件的stat。

参数

<i>user</i>	文件属主
<i>ino</i>	文件获得的inode编号
<i>mode</i>	创建模式

返回

int 总是返回0。

4.5.2.10 find_file()

```
void find_file (
    int cwd,
    const char * args,
    int ino )
```

查找文件。

参数

<i>cwd</i>	工作目录
<i>args</i>	参数
<i>ino</i>	inode编号

4.5.2.11 find_name_in_dir()

```
int findname_in_dir (
    int ino,
    const char * name,
    int mode )
```

在指定目录下查找具有指定名称的文件。

参数

<i>ino</i>	目录项的inode编号
<i>name</i>	待查的文件名
<i>mode</i>	查询模式

返回

int 若找到文件，返回inode编号，否则返回-1。

4.5.2.12 link_file()

```
int link_file (
    int cwd,
    int argc,
    const char * argv[] )
```

创建硬链接。

接受两个参数，第一个参数是源文件名，第二个参数是硬链接的文件名。

参数

<i>cwd</i>	工作目录
<i>argc</i>	没有用
<i>argv</i>	参数

返回

int 成功返回0，失败返回-1。

4.5.2.13 ls()

```
void ls (
    int ino,
    char * args )
```

查看目录信息。

NOTE：真正的ls指令应该和权限有关。执行权限被拒绝的话应该不能成功执行ls。我不太确定（

参数

<i>ino</i>	要查看的目录的inode编号。
<i>args</i>	希腊奶，传NULL就完事了！

4.5.2.14 mkdir()

```
int mkdir (
    const user_t * user,
    int cwd,
    const char * dir_name,
    int mode )
```

创建目录。

参数

<i>user</i>	目录属主
<i>cwd</i>	工作目录
<i>dir_name</i>	目录名
<i>mode</i>	你尽管改，起作用算我输

返回

int 成功返回0，否则返回-1。

4.5.2.15 open_file()

```
int open_file (
    const user_t * user,
    int cwd,
    const char * args )
```

打开文件。

参数

<i>user</i>	执行操作的用户
<i>cwd</i>	工作目录
<i>args</i>	待查文件名

返回

int 成功返回inode，否则返回-1。

4.5.2.16 pwd()

```
void pwd (
    int cwd,
    char * buf,
    int ino )
```

取得指定inode的目录名。

参数

<i>cwd</i>	要取得目录名的inode编号
<i>buf</i>	缓冲区
<i>ino</i>	递归参数，一般填当前目录即可

4.5.2.17 read_file()

```
int read_file (
    const user_t * user,
    int cwd,
    int argc,
    const char * argv[] )
```

读取文件。

参数

<i>user</i>	用户
<i>cwd</i>	工作目录
<i>argc</i>	没有用
<i>argv</i>	参数

返回

int 成功返回0，否则返回-1。

4.5.2.18 rm()

```
void rm (
    const user_t * user,
    int cwd,
    const char * args,
    int workIno,
    int mode )
```

删除指定文件。

参数

<i>user</i>	用户
<i>cwd</i>	工作目录
<i>args</i>	目标文件名？
<i>workIno</i>	希腊奶（？）
<i>mode</i>	删除模式

4.5.2.19 rm_dir_item()

```
int rm_dir_item (
    dir_t * d,
    int n )
```

删除指定目录下的第n项。

NOTE：讲真，这个是不是不应该暴露出来？

参数

<i>d</i>	待处理的目录
<i>n</i>	见介绍

返回

int 总是返回0。

4.5.2.20 rmdir()

```
void rmdir (
    const user_t * user,
    int cwd,
    char * dir_name,
    int mode )
```

删除目录。

参数

<i>user</i>	用户
<i>cwd</i>	工作目录
<i>dir_name</i>	目标目录名
<i>mode</i>	删除模式

4.5.2.21 write_file()

```
int write_file (
    const user_t * user,
    int cwd,
    int argc,
    const char * argv[] )
```

写入文件。

参数

<i>user</i>	用户
<i>cwd</i>	工作目录
<i>argc</i>	没有用
<i>argv</i>	参数

返回

int 成功返回0，失败返回-1。

4.6 位图

位图 的协作图:



宏定义

- #define SET_BLK_FLAG(i) (blk_flag[i / CHAR_BIT] |= '\x01' << (i % CHAR_BIT))
- #define TEST_BLK_FLAG(i) (blk_flag[i / CHAR_BIT] & '\x01' << (i % CHAR_BIT) ? 1 : 0)
 检查标识位
- #define CLEAR_BLK_FLAG(i) (blk_flag[i / CHAR_BIT] ^= '\x01' << (i % CHAR_BIT))
 清除标识位

变量

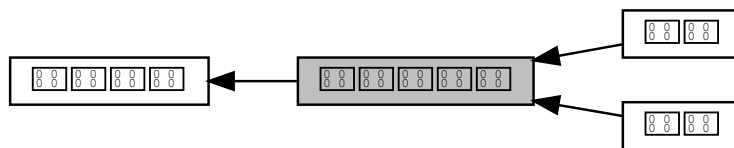
- char blk_flag []
 位图管理，指示哪些块可用。

4.6.1 详细描述

设置标识位

4.7 用户和权限

用户和权限 的协作图:



模块

- 权限
- 行为

结构体

- struct `user`
描述一个用户的信息。

类型定义

- typedef struct `user` `user_t`
描述一个用户的信息。

变量

- int `idle_uid`
可用的下一个用户 *id*。

4.7.1 详细描述

4.7.2 变量说明

4.7.2.1 idle_uid

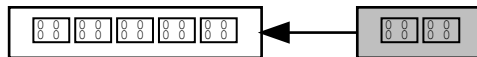
```
int idle_uid [extern]
```

可用的下一个用户id。

这本来是应该放到交互逻辑的，但这个设计下实在难以分离出去。

4.8 权限

权限 的协作图:



宏定义

- **#define S_IRUSR 0x00400**
属主读权限掩码
- **#define S_IWUSR 0x00200**
属主写权限掩码
- **#define S_IXUSR 0x00100**
属主执行权限掩码
- **#define S_IRGRP 0x00040**
属组读权限掩码
- **#define S_IWGRP 0x00020**
属组写权限掩码
- **#define S_IXGRP 0x00010**
属组执行权限掩码
- **#define S_IROTH 0x00004**
其他用户读权限掩码
- **#define S_IWOTH 0x00002**
其他用户写权限掩码
- **#define S_IXOTH 0x00001**
其他用户执行权限掩码
- **#define R_OK 0x04**
读权限掩码
- **#define W_OK 0x02**
写权限掩码
- **#define X_OK 0x01**
执行权限掩码
- **#define HAVE_ACCESS(ino, mode, ch) (inode[ino].i_stat.st_mode & mode ? ch : '-')**
检查文件权限

函数

- int `access` (const `user_t` *`user`, int `ino`, int `mode`)
验证用户权限。

4.8.1 详细描述

4.8.2 函数说明

4.8.2.1 `access()`

```
int access (  
    const user_t * user,  
    int ino,  
    int mode )
```

验证用户权限。

参数

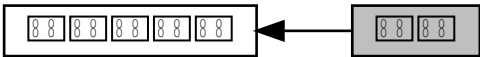
<code>user</code>	用户对象的指针
<code>ino</code>	文件的inode编号
<code>mode</code>	访问模式

返回

int 成功返回0，失败或无权限返回-1。

4.9 行为

行为 的协作图:



函数

- int `login` (`user_t` *`user`, const char *`args`)

登录函数。

- int `find_user` (const char *args)

查找用户。

- void `add_user` (const `user_t` *user, const char *args)

添加用户。

4.9.1 详细描述

4.9.2 函数说明

4.9.2.1 add_user()

```
void adduser (
    const user_t * user,
    const char * args )
```

添加用户。

NOTE: 你这咋连返回值都没了？

参数

<i>user</i>	执行操作的用户身份
<i>args</i>	待添加的用户名

4.9.2.2 find_user()

```
int finduser (
    const char * args )
```

查找用户。

参数

<i>args</i>	用户名
-------------	-----

返回

int 成功返回0，失败返回-1。

4.9.2.3 login()

```
int login (
    user_t * user,
    const char * args )
```

登录函数。

NOTE: 在这里写文件交互? 建议: 外部写一个管理器, 只加载一次。 修改时管理器和文件同步修改。 对大文件, 这样效率比较高。

参数

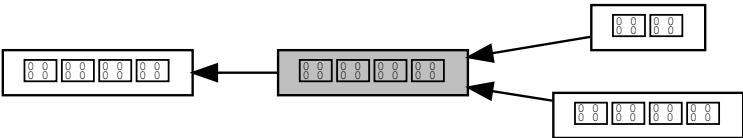
user	接收用户的登录参数
args	用户名。若为NULL, 则执行交互式登录

返回

int 成功返回0, 失败返回-1。

4.10 交互终端

交互终端 的协作图:



模块

- 交互接口
- 指令

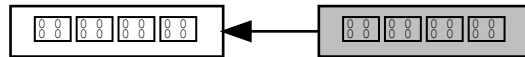
变量

- int work_dir
工作目录的inode编号
- user_t cnt.user
当前登录的用户

4.10.1 详细描述

4.11 交互接口

交互接口 的协作图:



函数

- `char * readline (const char *str)`
读入指令。
- `void add_history (char *p)`
向指令历史添加指令。（未完成）
- `char * rl_gets ()`
接受输入字符。不可重入。

4.11.1 详细描述

4.11.2 函数说明

4.11.2.1 add_history()

```
void add_history (
    char * p )
```

向指令历史添加指令。（未完成）

参数

<i>p</i>	待添加的指令
----------	--------

4.11.2.2 readline()

```
char * readline (
```



```
const char * str )
```

读入指令。

参数

<i>str</i>	命令提示符 (prompt)
------------	----------------

返回

`char*` 读到的指令。

4.11.2.3 rl_gets()

```
char * rl_gets ( )
```

接受输入字符。不可重入。

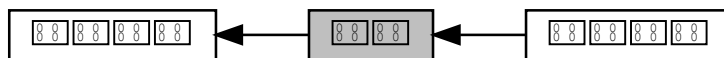
讲道理，这个函数到底是干啥的？

返回

`char*` 大概是读入的指令

4.12 指令

指令 的协作图:



模块

- [指令行为](#)

结构体

- `struct cmd`

描述一个指令条目。

宏定义

- `#define CMD_N 14`
指令数。

类型定义

- `typedef struct cmd cmd_t`
描述一个指令条目。

变量

- `cmd_t cmd_table []`
指令表。

4.12.1 详细描述

4.12.2 宏定义说明

4.12.2.1 CMD_N

```
#define CMD_N 14
```

指令数。

为了能分离实现和定义而写了硬编码。

4.13 指令行为

指令行为 的协作图:



函数

- void **cmd_pwd** (char *)
- void **cmd_ls** (char *)
- void **cmd_mkdir** (char *)
- void **cmd_cd** (char *)
- void **cmd_rmdir** (char *)
- void **cmd_su** (char *)
- void **cmd_whoami** (char *)
- void **cmd_useradd** (char *)
- void **cmd_creat** (char *)
- void **cmd_rm** (char *)
- void **cmd_read** (char *)
- void **cmd_write** (char *)
- void **cmd_ln** (char *)
- void **cmd_find** (char *)

4.13.1 详细描述

Chapter 5

结构体说明

5.1 `__block_num_validator`结构体 参考

成员变量

- `int _[__CHECK_POS\(BLOCK_NUM\)]`

该结构体的文档由以下文件生成:

- `include/config.h`

5.2 `__block_size_validator`结构体 参考

成员变量

- `int _[__CHECK_POS\(BLOCK_SIZE >=8\)]`
- `int _[__CHECK_NEG\(__IS_POW_2\(BLOCK_SIZE\)\)]`

该结构体的文档由以下文件生成:

- `include/config.h`

5.3 `__indirect_idx_1_num_validator`结构体 参考

成员变量

- `int _[__CHECK_NEG\(INDIRECT_IDX_1_NUM - 10\)]`

该结构体的文档由以下文件生成:

- `include/config.h`

5.4 __inode_num_validator结构体 参考

成员变量

- `int _[CHECK_POS(INODE_NUM)]`

该结构体的文档由以下文件生成:

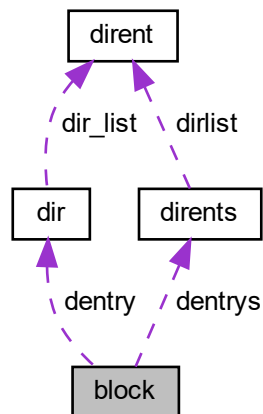
- `include/config.h`

5.5 block联合体 参考

定义文件系统数据块。

```
#include <block.h>
```

block 的协作图:



成员变量

- `char str [BLOCK_SIZE]`
数据块
- `dir_t dentry`
目录块
- `int index [BLOCK_SIZE/sizeof(int)]`
索引块
- `dirents_t dentrys`
扩展目录结构

5.5.1 详细描述

定义文件系统数据块。

NOTE: 我们真的**非常不建议**这样使用union! 初始化一个域后访问其他域是UB。建议额外用一个字段记录union里面存的到底是什么, 或者干脆改用类似void*的东西。

NOTE2: 我在这里做了少许修改。你内嵌定义了一个union, 但没有定义变量。认真的? **NOTE3:** 鉴于实现比较怪(真正的dentry肯定不是这么存的), 不进行block_t是否能存下dir_t和dirents_t的校验。

该联合体的文档由以下文件生成:

- include/sys/fs/types/block.h

5.6 cmd结构体 参考

描述一个指令条目。

```
#include <cmd.h>
```

成员变量

- `const char * name`
指令名称
- `char * description`
指令描述
- `void(* handler)(char *args)`
指令回调函数

5.6.1 详细描述

描述一个指令条目。

该结构体的文档由以下文件生成:

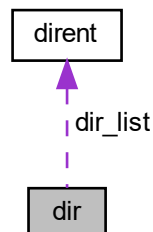
- include/shell/cmd.h

5.7 dir结构体 参考

描述目录结构。

```
#include <dir.h>
```

dir 的协作图:



成员变量

- **char `dir_name`** [28]
目录名称
- **int `dir_list_size`**
目录大小
- **`dirent_t` `dir_list`** [20]
具体的目录列表

5.7.1 详细描述

描述目录结构。

本质上是产生一个树形数据结构。

注意：记得依次释放`dir_list`的内容，以免内存泄漏。

NOTE: `ext4`底层真的是这样的吗？我不确定，以后看看

该结构体的文档由以下文件生成：

- `include/sys/fs/types/dirent.h`

5.8 `dirent`结构体 参考

描述一个项目的入口。

```
#include <dirent.h>
```

成员变量

- **int `d_ino`**
关联的`inode`编号
- **char `d_name`** [20]
项目的名称

5.8.1 详细描述

描述一个项目的入口。

该结构体的文档由以下文件生成：

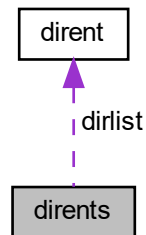
- `include/sys/fs/types/dirent.h`

5.9 **dirents**结构体 参考

扩展目录结构。

```
#include <dirents.h>
```

dirents 的协作图:



成员变量

- `int dir_list_size`
目录项数目
- `dirent_t dirlist [21]`
具体目录项列表

5.9.1 详细描述

扩展目录结构。

该结构体的文档由以下文件生成:

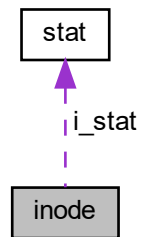
- `include/sys/fs/types/dirents.h`

5.10 **inode**结构体 参考

描述**inode**结点。

```
#include <inode.h>
```

inode 的协作图:



成员变量

- `stat_t i_stat`
 结点状态信息
- `int blk1 [INDIRECT_IDX_1_NUM]`
 一级索引（确定不是直接索引？）
- `int blk2`
 二级索引（确定不是一级索引？）

5.10.1 详细描述

描述inode结点。

该结构体的文档由以下文件生成:

- `include/sys/fs/types/inode.h`

5.11 stat结构体 参考

描述一个inode的基本信息。

```
#include <stat.h>
```

成员变量

- **int st_ino**
inode结点号
- **int st_mode**
文件类型 (?)
- **int st_nlink**
引用计数
- **int st_size**
文件大小
- **int st_uid**
文件所有者
- **int st_gid**
文件属组
- **int st_blksize**
文件块大小 (?)
- **int st_blocks**
文件块数量
- **time_t st_atime**
最后访问时间
- **time_t st_mtime**
最后修改时间
- **time_t st_ctime**
状态改变时间

5.11.1 详细描述

描述一个inode的基本信息。

该结构体的文档由以下文件生成:

- `include/sys/fs/types/stat.h`

5.12 user结构体 参考

描述一个用户的信息。

```
#include <user.h>
```

成员变量

- **char pw_name [10]**
用户名
- **char pw_passwd [20]**
明文密码
- **int pw_uid**
用户id
- **int pw_gid**
用户所属组id

5.12.1 详细描述

描述一个用户的信息。

该结构体的文档由以下文件生成:

- `include/sys/user.h`

Chapter 6

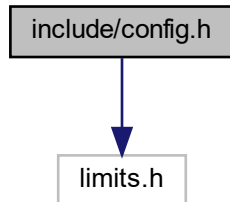
文件说明

6.1 include/config.h 文件参考

定义基本数据，一部分可修改。

```
#include <limits.h>
```

config.h 的引用(Include)关系图:



结构体

- struct `__block_size_validator`
- struct `__block_num_validator`
- struct `__inode_num_validator`
- struct `__indirect_idx_1_num_validator`

宏定义

- `#define BLOCK.SIZE 512`
指定块尺寸。必须是2的幂。
- `#define BLOCK.NUM 512`
指定块数目。必须是正数。
- `#define INODE.NUM 512`

指定 *inode* 数目。必须是正数。

- `#define INDIRECT_IDX_1_NUM 10`

指定一级索引（直接索引？）数量。目前必须是 **10**，因为修改的话大概率出 *bug*（因为我无法分辨那些不明所以的字面值）。

- `#define MAX_DIRLIST 20`

写了，但没有卵用的东西。你尽管改，起作用算我输。（这是你的代码，我没改。）

- `#define __CHECK_POS(cond) (1 - (((!(cond)) - 1) << 1))`
- `#define __CHECK_NEG(cond) (1 - (((!(cond)) - 1) << 1))`
- `#define __IS_POW_2(x) ((x) & ((x) - 1))`
- `#define BLK_FLAG_SIZE (((BLOCK_SIZE) + (CHAR_BIT - 1)) / CHAR_BIT)`

6.1.1 详细描述

定义基本数据，一部分可修改。

版本

0.1

日期

2021-11-14

6.2 config.h

[浏览该文件的文档.](#)

```
1
2
3
4
5
6
7
8 #ifndef __CMJFS__CONFIG_HH__
9 #define __CMJFS__CONFIG_HH__ 1
10
11 #include <limits.h>
12
13
14
15
16
17 #define BLOCK_SIZE 512
18 #define BLOCK_NUM 512
19 #define INODE_NUM 512
20 #define INDIRECT_IDX_1_NUM 10
21 #define MAX_DIRLIST 20
22
23
24
25
26
27
28
29 // 下面是全局编译期检查。**不要修改**!!
30 // NOTE: 用struct是为了避免产生实际符号，干扰编译过程。
31
32 // 检查是否非0，不是则返回负值。用于进行编译期检查
33 // NOTE: 有些编译器可能不允许数组长度为0，但有些可以。我们利用这点实现编译期约束
34 // 原理：
35 // - 先把cond转化为数字0F/1T
36 // - 利用数组长度必须为正的数特性，减去1后乘2，取相反数再加1
37 // - 则分别可以取到1和-1，目的达到
38 #define __CHECK_POS(cond) (1 - (((!(cond)) - 1) << 1))
39 // 检查是否为0
40 #define __CHECK_NEG(cond) (1 - (((!(cond)) - 1) << 1))
41 // 检查x是否是2的幂，是则返回0，否则返回非0
42 #define __IS_POW_2(x) ((x) & ((x) - 1))
43
44 // 检查BLOCK_SIZE是否是2的幂。不满足的话就会在这里报错
45 struct __block_size_validator {
46     // 约束1: BLOCK_SIZE必须是不小于8的正整数
47     int _[__CHECK_POS(BLOCK_SIZE) >= 8];
48     // 约束2: BLOCK_SIZE是2的幂
49     // - 当x是2的幂时，x & (x-1)一定是0，反之非0
50     // - 逻辑取反后，满足为1，反之为0
51     int _[__CHECK_NEG(__IS_POW_2(BLOCK_SIZE))];
52 };
53
```

```

54 // 检查BLOCK_NUM是否是正数
55 struct __block_num_validator {
56     int _[CHECK_POS(BLOCK_NUM)];
57 };
58
59 // 检查INODE_NUM是否是正数
60 struct __inode_num_validator {
61     int _[CHECK_POS(INODE_NUM)];
62 };
63
64 // 验证INDIRECT_IDX_1_NUM
65 struct __indirect_idx_1_num_validator {
66     int _[CHECK_NEG(INDIRECT_IDX_1_NUM - 10)];
67 };
68
69 // 下面是依赖配置信息自动运算的部分，请勿修改
70
71 // blk_flag需要申请多少字节
72 #define BLK_FLAG_SIZE (((BLOCK_SIZE) + (CHAR_BIT - 1)) / CHAR_BIT)
73
74 #endif // __CMJFS__CONFIG_HH__

```

6.3 include/doc.h 文件参考

帮助doxygen生成文档。****不要****试图包含该文件！

6.3.1 详细描述

帮助doxygen生成文档。****不要****试图包含该文件！

版本

0.1

日期

2021-11-14

6.4 doc.h

[浏览该文件的文档.](#)

```

1
33
34 // !!! @defgroup wtf 完全不知道干什么用的东西
35
36 // /**
37 //  * @brief 兄啊，你连实现都没实现，我怎么会知道这是做什么的
38 //  *
39 //  * @ingroup wtf
40 //  *
41 //  * @param ino 希腊奶
42 //  * @param len 希腊奶
43 //  * @return char* 就不写con.....哦这个不需要const啊，那没事了
44 //  */
45 // char* creat.reg(int ino, int len);
46
47 #error DO NOT include this file!
48

```

6.5 cmd.h

```

1
8 #ifndef __CMJFS__SHELL__COMMAND_H__
9 #define __CMJFS__SHELL__COMMAND_H__ 1
10
13
17 typedef struct cmd {
19     const char* name;
21     char* description;
23     void (*handler)(char* args);
24 } cmd_t;
25
31 #define CMD_N 14
32 // TODO: 这样的话sizeof是不行的，需要另想办法。
33
37 extern cmd_t cmd_table[];
38
40
43
44 // TODO: 写详细注释（我懒得做了）
45
46 void cmd_pwd(char*);
47 void cmd_ls(char*);
48 void cmd_mkdir(char*);
49 void cmd_cd(char*);
50 void cmd_rmdir(char*);
51 void cmd_su(char*);
52 void cmd_whoami(char*);
53 void cmd_useradd(char*);
54 void cmd_creat(char*);
55 void cmd_rm(char*);
56 void cmd_read(char*);
57 void cmd_write(char*);
58 void cmd_ln(char*);
59 void cmd_find(char*);
60
62
63 #endif // __CMJFS__SHELL__COMMAND_H__

```

6.6 include/shell/io.h 文件参考

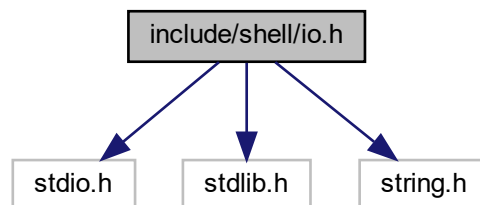
提供交互功能。

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

io.h 的引用(Include)关系图:



函数

- char * [readline](#) (const char *str)

读入指令。

- void `add.history` (char *p)
向指令历史添加指令。（未完成）
- char * `rl.gets` ()
接受输入字符。不可重入。

6.6.1 详细描述

提供交互功能。

版本

0.1

日期

2021-11-14

6.7 io.h

[浏览该文件的文档.](#)

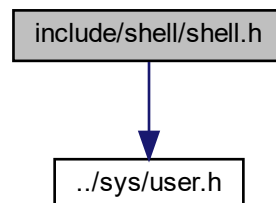
```
1
8 #ifndef __CMJFS__SHELL__IO_H__
9 #define __CMJFS__SHELL__IO_H__ 1
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14
17
24 char* readline(const char* str);
25
31 void addhistory(char* p);
32
40 char* rl.gets();
41
43
44 #endif // __CMJFS__SHELL__IO_H__
```

6.8 include/shell/shell.h 文件参考

定义交互终端的核心。

```
#include "../sys/user.h"
```

shell.h 的引用(Include)关系图:



变量

- `int work_dir`
工作目录的*inode*编号
- `user_t cnt_user`
当前登录的用户

6.8.1 详细描述

定义交互终端的核心。

版本

0.1

日期

2021-11-14

6.9 shell.h

[浏览该文件的文档.](#)

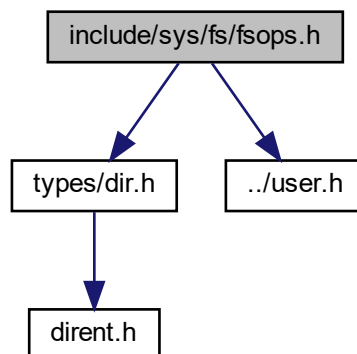
```
1
8 #ifndef __CMJFS__SHELL__SHELL_H__
9 #define __CMJFS__SHELL__SHELL_H__ 1
10
11 #include "../sys/user.h"
12
13
14
15
16
17 extern int work_dir;
18
19 extern user_t cnt_user;
20
21
22
23 #endif // __CMJFS__SHELL__SHELL_H__
```

6.10 include/sys/fs/fsops.h 文件参考

文件系统的核心操作。

```
#include "types/dir.h"
#include "../user.h"
```

fsops.h 的引用(Include)关系图:



宏定义

- **#define S_IFREG** 0x0100000
普通文件标识
- **#define S_IFDIR** 0x0040000
目录文件标识
- **#define RECURSIVE** 0x800
递归行为标识
- **#define GET_BLKNO**(ino, n) (`BLK[inode[ino].blk1[n]]`)
获取文件的第 n 个块
- **#define GET_PAGE**(ino, page) (`page >= 10 ? BLK[inode[ino].blk2].index[page-10] : inode[ino].blk1[page]`)
获取文件的第 n 个块
- **#define GET_CHAR**(ino, page, offset) (`BLK[GET_PAGE(ino,page)].str[offset]`)
获取字符(?)

函数

- int **add_blk_for_file** (int ino)
为文件申请新的数据块。
- int **find_name_in_dir** (int ino, const char *name, int mode)
在指定目录下查找具有指定名称的文件。
- int **creat_stat** (const user_t *user, int ino, int mode)
创建文件的 *stat*。
- int **creat_dirent** (int fno, int ino, const char *name)
为文件创建目录入口。
- int **creat_dir** (int ino, int f_ino, const char *dir_name)
为文件分配 *struct dir* 结构。
- int **creat_ino** ()
分配 *inode*。
- int **creat** (const user_t *user, int cwd, const char *args, int mode)

- 创建文件。
- `int mkdir (const user_t *user, int cwd, const char *dir_name, int mode)`
创建目录。
- `void ls (int ino, char *args)`
查看目录信息。
- `void pwd (int cwd, char *buf, int ino)`
取得指定`inode`的目录名。
- `void cd (const user_t *user, int *cwd, const char *dir_name)`
修改指定用户的工作目录。
- `int rm_dir_item (dir_t *d, int n)`
删除指定目录下的第 n 项。
- `void rm (const user_t *user, int cwd, const char *args, int workIno, int mode)`
删除指定文件。
- `void rmdir (const user_t *user, int cwd, char *dir_name, int mode)`
删除目录。
- `void creat_file (const user_t *user, int cwd, const char *args)`
创建文件。
- `int open_file (const user_t *user, int cwd, const char *args)`
打开文件。
- `int read_file (const user_t *user, int cwd, int argc, const char *argv[])`
读取文件。
- `int write_file (const user_t *user, int cwd, int argc, const char *argv[])`
写入文件。
- `int link_file (int cwd, int argc, const char *argv[])`
创建硬链接。
- `void find_file (int cwd, const char *args, int ino)`
查找文件。

6.10.1 详细描述

文件系统的核心操作。

版本

0.1

日期

2021-11-14

6.11 fsops.h

[浏览该文件的文档.](#)

```

1
8 #ifndef __CMJFS__SYS_FS__FS_H__
9 #define __CMJFS__SYS_FS__FS_H__ 1
10
11 #include "types/dir.h"
12 #include "../user.h"
13
14
15
16
17
18 #define S_IFREG 0x0100000
19 #define S_IFDIR 0x0040000
20 #define RECURSIVE 0x800
21
22
23
24 #define GET_BLKNO(ino,n) (BLK[inode[ino].blk1[n]])
25 // NOTE: 这, , , 真就只有直接索引?
26 // TODO: 修改这个东西, 我猜可能没法用宏来实现
27
28
29 #define GET_PAGE(ino,page) (page >= 10 ? BLK[inode[ino].blk2].index[page-10] : inode[ino].blk1[page])
30 #define GET_CHAR(ino,page,offset) (BLK[GET_PAGE(ino,page)].str[offset])
31 // NOTE: 宏必须加括号, 这是原则。这里我帮你加上了。这样的原则是为了防止疏忽和减少维护成本。
32 // 以及, 为什么是page不是block?
33
34
35
36 int add_blk_for_file(int ino);
37
38
39 int findname_in_dir(int ino, const char* name, int mode);
40
41
42 int creat_stat(const user_t* user, int ino, int mode);
43
44
45 int creat_dirent(int fno, int ino, const char* name);
46
47
48 int creat_dir(int ino, int fno, const char* dir_name);
49 // TODO: 检查潜在bug
50
51
52 int creat_ino();
53
54
55 int creat(const user_t* user, int cwd, const char* args, int mode);
56
57
58 // TODO: 你truncate呢? ? 这个调用很重要啊
59
60
61 int mkdir(const user_t* user, int cwd, const char* dir_name, int mode);
62
63
64 void ls(int ino, char* args);
65
66
67 void pwd(int cwd, char* buf, int ino);
68
69
70 void cd(const user_t* user, int* cwd, const char* dir_name);
71
72
73 int rm_dir_item(dir_t* d, int n);
74
75
76 void rm(const user_t* user, int cwd, const char* args, int workIno, int mode);
77
78
79 void rmdir(const user_t* user, int cwd, char* dir_name, int mode);
80
81
82 void creat_file(const user_t* user, int cwd, const char* args);
83
84
85 int open_file(const user_t* user, int cwd, const char* args);
86
87
88 int read_file(const user_t* user, int cwd, int argc, const char* argv[]);
89
90
91 int write_file(const user_t* user, int cwd, int argc, const char* argv[]);
92
93
94 int link_file(int cwd, int argc, const char* argv[]);
95
96
97 void find_file(int cwd, const char* args, int ino);
98
99
100
101 #endif // __CMJFS__SYS_FS__FS_H__

```

6.12 include/sys/fs/mediactrl.h 文件参考

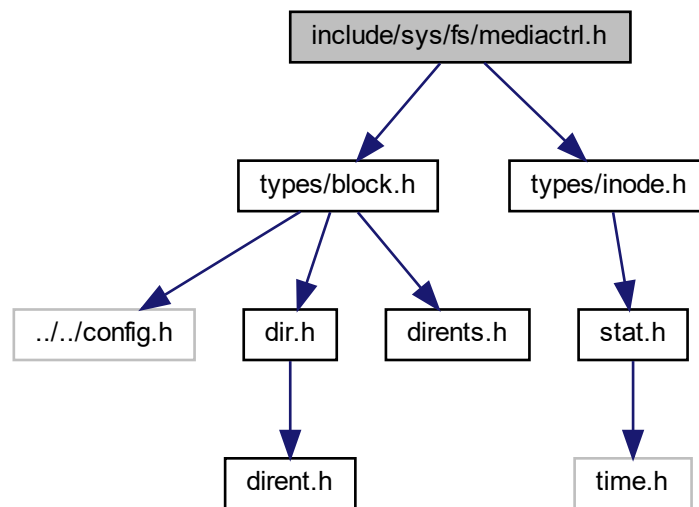
实现基本的介质管理。

```

#include "types/block.h"
#include "types/inode.h"

```

mediactrl.h 的引用(Include)关系图:



宏定义

- `#define SET_BLK_FLAG(i) (blk.flag[i / CHAR_BIT] |= '\x01' << (i % CHAR_BIT))`
- `#define TEST_BLK_FLAG(i) (blk.flag[i / CHAR_BIT] & '\x01' << (i % CHAR_BIT) ? 1 : 0)`
检查标识位
- `#define CLEAR_BLK_FLAG(i) (blk.flag[i / CHAR_BIT] ^= '\x01' << (i % CHAR_BIT))`
清除标识位

函数

- `int acq_blk ()`
申请一个新块。

变量

- `block_t BLK []`
全局块数据。
- `char blk.flag []`
位图管理，指示哪些块可用。
- `inode_t inode []`
全局inode数据。

6.12.1 详细描述

实现基本的介质管理。

版本

0.1

日期

2021-11-14

6.13 mediactrl.h

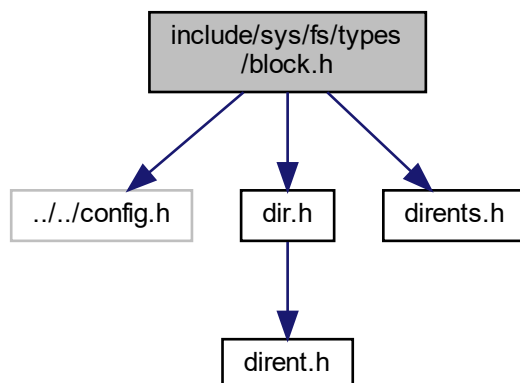
[浏览该文件的文档.](#)

```
1
8 #ifndef __CMJFS__SYS_FS__MEDIACTRL_H__
9 #define __CMJFS__SYS_FS__MEDIACTRL_H__ 1
10
11 #include "types/block.h"
12 #include "types/inode.h"
13
17 #define SET_BLK_FLAG(i) (blk_flag[i / CHAR_BIT] |= '\x01' << (i % CHAR_BIT))
19 #define TEST_BLK_FLAG(i) (blk_flag[i / CHAR_BIT] & '\x01' << (i % CHAR_BIT) ? 1 : 0)
21 #define CLEAR_BLK_FLAG(i) (blk_flag[i / CHAR_BIT] ^= '\x01' << (i % CHAR_BIT))
23
28 extern block_t BLK[];
29
34 extern char blk_flag[];
35
40 extern inode_t inode[];
41
50 int acq_blk();
51
52 #endif // __CMJFS__SYS_FS__MEDIACTRL_H__
```

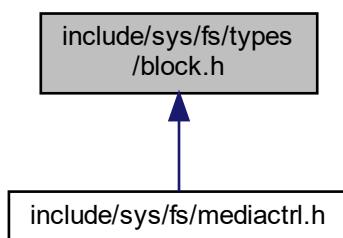
6.14 include/sys/fs/types/block.h 文件参考

定义文件系统块。

```
#include ".././config.h"
#include "dir.h"
#include "dirents.h"
block.h 的引用(Include)关系图:
```



此图展示该文件直接或间接的被哪些文件引用了:



结构体

- union [block](#)
定义文件系统数据块。

类型定义

- typedef union [block](#) [block_t](#)
定义文件系统数据块。

6.14.1 详细描述

定义文件系统块。

版本

0.1

日期

2021-11-14

6.15 block.h

[浏览该文件的文档.](#)

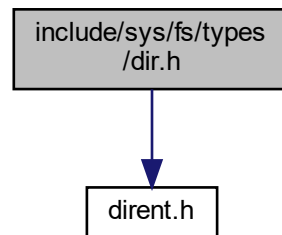
```
1
2
3
4
5
6
7
8 #ifndef __CMJFS__SYS_FS_TYPES__BLOCK_H__
9 #define __CMJFS__SYS_FS_TYPES__BLOCK_H__ 1
10
11 #include "../config.h"
12 #include "dir.h"
13 #include "dirents.h"
14
15
16
17
18
19
20
21
22
23
24
25
26 typedef union block {
27     char str[BLOCK_SIZE];
28     dir_t dentry;
29     int index[BLOCK_SIZE / sizeof(int)];
30     dirent_t dentrys;
31 } block_t;
32
33
34
35
36
37 #endif // __CMJFS__SYS_FS_TYPES__BLOCK_H__
```


6.16 include/sys/fs/types/dir.h 文件参考

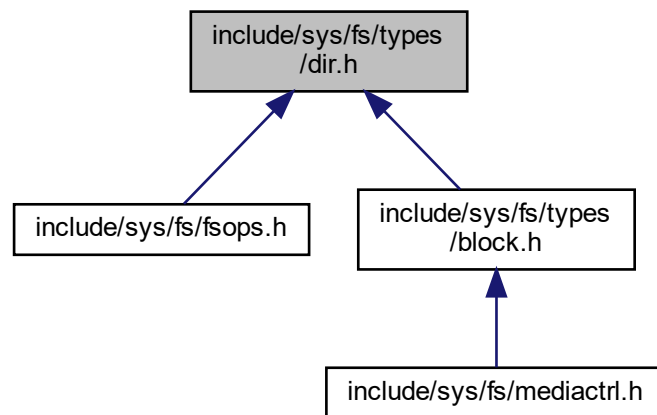
定义struct dir结构。

```
#include "dirent.h"
```

dir.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



结构体

- struct **dir**
描述目录结构。

类型定义

- typedef struct **dir** **dir_t**
描述目录结构。

6.16.1 详细描述

定义struct dir结构。

版本

0.1

日期

2021-11-14

6.17 dir.h

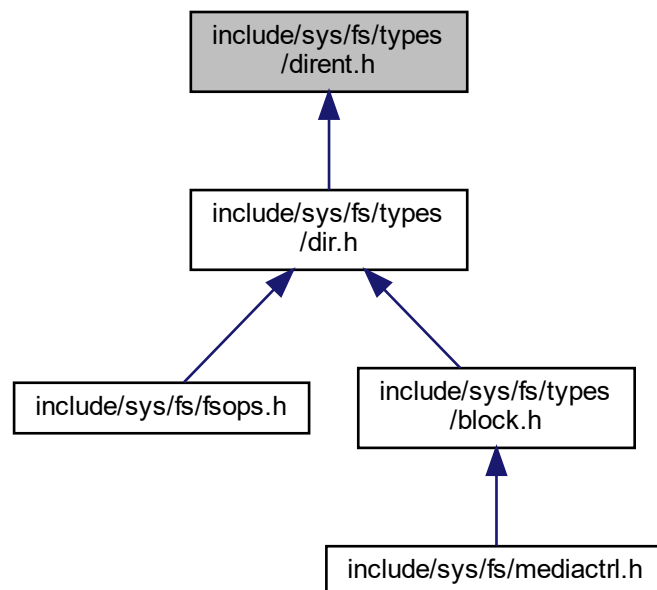
[浏览该文件的文档.](#)

```
1
8 #ifndef __CMJFS__SYS_FS_TYPES__DIR_H__
9 #define __CMJFS__SYS_FS_TYPES__DIR_H__ 1
10
11 #include "dirent.h"
12
23 typedef struct dir {
25     char dir_name[28];
27     int dir_list_size;
29     dirent_t dir_list[20];
30 } dir_t;
31
32 #endif // __CMJFS__SYS_FS_TYPES__DIR_H__
```

6.18 include/sys/fs/types/dirent.h 文件参考

定义struct dirent结构。

此图展示该文件直接或间接的被哪些文件引用了：



结构体

- struct `dirent`
描述一个项目的入口。

类型定义

- typedef struct `dirent` `dirent_t`
描述一个项目的入口。

6.18.1 详细描述

定义struct `dirent`结构。

版本

0.1

日期

2021-11-14

6.19 dirent.h

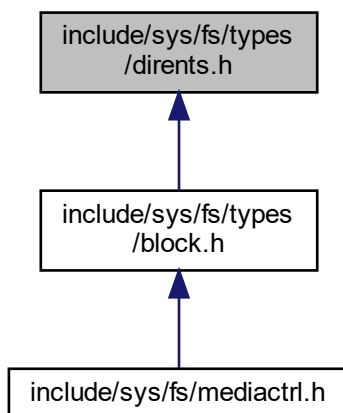
[浏览该文件的文档.](#)

```
1
8 #ifndef __CMJFS__SYS_FS_TYPES__DIRENT_H__
9 #define __CMJFS__SYS_FS_TYPES__DIRENT_H__ 1
10
15 typedef struct dirent{
17     int d_ino;
19     char d_name[20];
20 } dirent_t;
21
22 #endif // __CMJFS__SYS_FS_TYPES__DIRENT_H__
```

6.20 include/sys/fs/types/dirents.h 文件参考

定义struct `dirents`。

此图展示该文件直接或间接的被哪些文件引用了:



结构体

- struct [dirents](#)
扩展目录结构。

类型定义

- typedef struct [dirents](#) **dirents_t**
扩展目录结构。

6.20.1 详细描述

定义struct **dirents**。

版本

0.1

日期

2021-11-14

6.21 dirents.h

[浏览该文件的文档.](#)

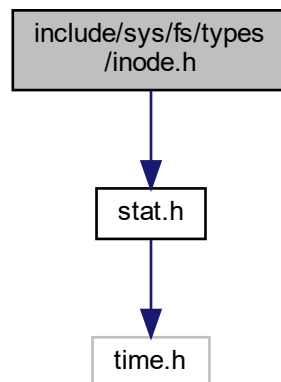
```
1
8 #ifndef __CMJFS__SYS_FS_TYPES_H__
9 #define __CMJFS__SYS_FS_TYPES_H__ 1
10
15 typedef struct dirents {
17     int dir_list_size;
19     dirent_t dirlist[21];
20 } dirents_t;
21
22 #endif // __CMJFS__SYS_FS_TYPES_H__
```

6.22 include/sys/fs/types/inode.h 文件参考

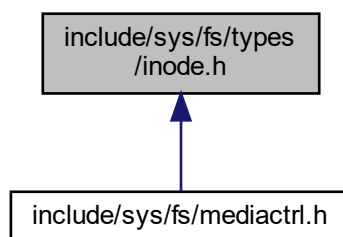
定义inode。

```
#include "stat.h"
```

inode.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了:



结构体

- struct [inode](#)
描述inode结点。

类型定义

- typedef struct [inode](#) **inode_t**
描述inode结点。

6.22.1 详细描述

定义inode。

版本

0.1

日期

2021-11-14

6.23 inode.h

[浏览该文件的文档.](#)

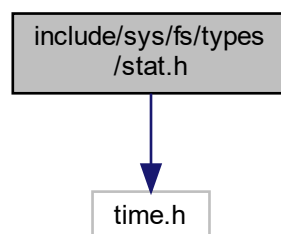
```
1
8 #ifndef __CMJFS__SYS_FS_TYPES__INODE_H__
9 #define __CMJFS__SYS_FS_TYPES__INODE_H__ 1
10
11 #include "stat.h"
12
13 typedef struct inode {
14     stat_t i_stat;
15     int blk1[INDIRECT_IDX_1_NUM];
16     int blk2;
17 } inode_t;
18
19 #endif // __CMJFS__SYS_FS_TYPES__INODE_H__
```

6.24 include/sys/fs/types/stat.h 文件参考

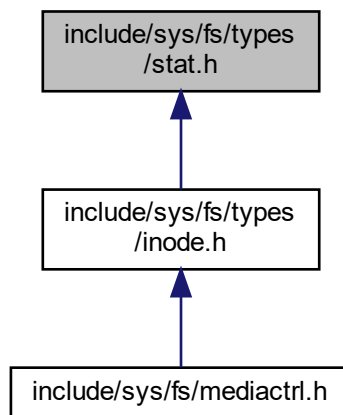
定义struct stat结构。

```
#include <time.h>
```

stat.h 的引用(Include)关系图:



此图展示该文件直接或间接的被哪些文件引用了：



结构体

- struct [stat](#)
描述一个 *inode* 的基本信息。

类型定义

- typedef struct [stat](#) **stat_t**
描述一个 *inode* 的基本信息。

6.24.1 详细描述

定义 `struct stat` 结构。

版本

0.1

日期

2021-11-14

6.25 stat.h

[浏览该文件的文档.](#)

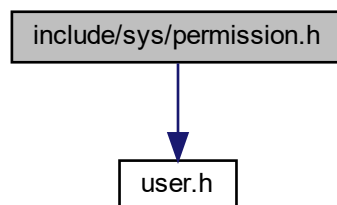
```
1
8 #ifndef __CMJFS__SYS_TYPES__STAT_H__
9 #define __CMJFS__SYS_TYPES__STAT_H__ 1
10
11 #include <time.h>
12
17 typedef struct stat {
19     int st_ino;
21     int st_mode;
23     int st_nlink;
25     int st_size;
27     int st_uid;
29     int st_gid;
31     int st_blksize;
33     int st_blocks;
35     time_t st_atime;
37     time_t st_mtime;
39     time_t st_ctime;
40 } stat_t;
41
42 #endif // __CMJFS__SYS_TYPES__STAT_H__
```

6.26 include/sys/permission.h 文件参考

文件权限的验证操作。

```
#include "user.h"
```

permission.h 的引用(Include)关系图:



宏定义

- **#define S_IRUSR** 0x00400
属主读权限掩码
- **#define S_IWUSR** 0x00200
属主写权限掩码
- **#define S_IXUSR** 0x00100
属主执行权限掩码
- **#define S_IRGRP** 0x00040
属组读权限掩码
- **#define S_IWGRP** 0x00020
属组写权限掩码

- `#define S_IXGRP 0x00010`
属组执行权限掩码
- `#define S_IROTH 0x00004`
其他用户读权限掩码
- `#define S_IWOTH 0x00002`
其他用户写权限掩码
- `#define S_IXOTH 0x00001`
其他用户执行权限掩码
- `#define R_OK 0x04`
读权限掩码
- `#define W_OK 0x02`
写权限掩码
- `#define X_OK 0x01`
执行权限掩码
- `#define HAVE_ACCESS(ino, mode, ch) (inode[ino].i_stat.st_mode & mode ? ch : '-')`
检查文件权限

函数

- `int access (const user_t *user, int ino, int mode)`
验证用户权限。

6.26.1 详细描述

文件权限的验证操作。

版本

0.1

日期

2021-11-14

6.27 permission.h

[浏览该文件的文档.](#)

```

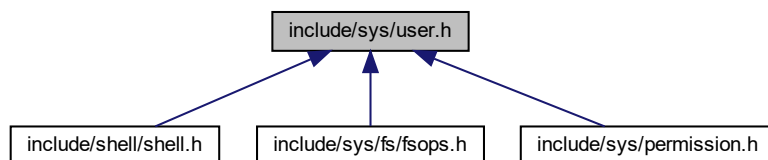
1
8 #ifndef __CMJFS__SYS_FS__PERMISSION_H__
9 #define __CMJFS__SYS_FS__PERMISSION_H__ 1
10
11 #include "user.h"
12
13
14
15
17 #define S_IRUSR 0x00400
19 #define S_IWUSR 0x00200
21 #define S_IXUSR 0x00100
23 #define S_IRGRP 0x00040
25 #define S_IWGRP 0x00020
27 #define S_IXGRP 0x00010
29 #define S_IROTH 0x00004
31 #define S_IWOTH 0x00002
33 #define S_IXOTH 0x00001
35 #define R_OK 0x04
37 #define W_OK 0x02
39 #define X_OK 0x01
40
42 #define HAVE_ACCESS(ino,mode,ch) (inode[ino].i_stat.st_mode & mode ? ch : '-')
43
44 // TODO: 三种特殊权限码
45
54 int access(const user_t* user, int ino, int mode);
55
56
57
58 #endif // __CMJFS__SYS_FS__PERMISSION_H__

```

6.28 include/sys/user.h 文件参考

定义用户信息和用户操作。

此图展示该文件直接或间接的被哪些文件引用了：



结构体

- struct `user`
描述一个用户的信息。

类型定义

- typedef struct `user` `user_t`
描述一个用户的信息。

函数

- int `login` (`user_t` *`user`, const char *args)
登录函数。
- int `find_user` (const char *args)
查找用户。
- void `add_user` (const `user_t` *`user`, const char *args)
添加用户。

变量

- int `idle_uid`
可用的下一个用户 *id*。

6.28.1 详细描述

定义用户信息和用户操作。

版本

0.1

日期

2021-11-14

6.29 user.h

[浏览该文件的文档.](#)

```
1
8 #ifndef __CMJFS__SYS__USER_H__
9 #define __CMJFS__SYS__USER_H__ 1
10
15 typedef struct user {
17     char pw_name[10];
19     char pw_passwd[20];
21     int pw_uid;
23     int pw_gid;
24 } user_t;
25
33 extern int idle_uid;
34
37
48 int login(user_t* user, const char* args);
49
56 int find.user(const char* args);
57
66 void add.user(const user_t* user, const char* args);
67
69
70 #endif // __CMJFS__SYS__USER_H__
```


Index

[__block_num.validator](#), 31
[__block_size.validator](#), 31
[__indirect.idx_1_num.validator](#), 31
[__inode_num.validator](#), 32

[access](#)
 [权限](#), 23
[acq_blk](#)
 [FS行为](#), 11
[add_blk_for_file](#)
 [FS行为](#), 12
[add_history](#)
 [交互接口](#), 26
[add_user](#)
 [行为](#), 24

[block](#), 32
[block_t](#)
 [结构](#), 9

[cd](#)
 [FS行为](#), 12

[cmd](#), 33
[CMD_N](#)
 [指令](#), 28
[creat](#)
 [FS行为](#), 12
[creat_dir](#)
 [FS行为](#), 13
[creat_dirent](#)
 [FS行为](#), 13
[creat_file](#)
 [FS行为](#), 14
[creat_ino](#)
 [FS行为](#), 14
[creat_stat](#)
 [FS行为](#), 14

[dir](#), 33
[dir_t](#)
 [结构](#), 9
[dirent](#), 34
[dirents](#), 35

[find_file](#)
 [FS行为](#), 15
[find_name_in_dir](#)
 [FS行为](#), 15
[find_user](#)
 [行为](#), 24
[FS行为](#), 10

[acq_blk](#), 11
[add_blk_for_file](#), 12
[cd](#), 12
[creat](#), 12
[creat_dir](#), 13
[creat_dirent](#), 13
[creat_file](#), 14
[creat_ino](#), 14
[creat_stat](#), 14
[find_file](#), 15
[find_name_in_dir](#), 15
[link_file](#), 16
[ls](#), 16
[mkdir](#), 17
[open_file](#), 17
[pwd](#), 17
[read_file](#), 18
[rm](#), 18
[rm_dir_item](#), 19
[rmdir](#), 19
[write_file](#), 20

[idle_uid](#)
 [用户和权限](#), 21
[include/config.h](#), 39, 40
[include/doc.h](#), 41
[include/shell/cmd.h](#), 42
[include/shell/io.h](#), 42, 43
[include/shell/shell.h](#), 43, 44
[include/sys/fs/fsops.h](#), 44, 47
[include/sys/fs/mediactrl.h](#), 47, 49
[include/sys/fs/types/block.h](#), 49, 50
[include/sys/fs/types/dir.h](#), 51, 52
[include/sys/fs/types/dirent.h](#), 52, 53
[include/sys/fs/types/dirents.h](#), 53, 54
[include/sys/fs/types/inode.h](#), 55, 56
[include/sys/fs/types/stat.h](#), 56, 58
[include/sys/permission.h](#), 58, 59
[include/sys/user.h](#), 60, 61
[inode](#), 35

[link_file](#)
 [FS行为](#), 16
[login](#)
 [行为](#), 24

[ls](#)
 [FS行为](#), 16

[mkdir](#)
 [FS行为](#), 17

- open_file
 - FS行为, 17
- pwd
 - FS行为, 17
- read_file
 - FS行为, 18
- readline
 - 交互接口, 26
- rl_gets
 - 交互接口, 27
- rm
 - FS行为, 18
- rm_dir_item
 - FS行为, 19
- rmdir
 - FS行为, 19
- stat, 36
- user, 37
- write_file
 - FS行为, 20
- 交互接口, 26
 - add_history, 26
 - readline, 26
 - rl_gets, 27
- 交互终端, 25
- 位图, 20
- 指令, 27
 - CMD_N, 28
- 指令行为, 28
- 文件系统, 8
- 权限, 22
 - access, 23
- 核心系统, 7
- 用户和权限, 21
 - idle_uid, 21
- 结构, 8
 - block_t, 9
 - dir_t, 9
- 行为, 23
 - add_user, 24
 - find_user, 24
 - login, 24
- 配置项, 7