CMJFS

制作者 Doxygen 1.9.2

Chapter 1

模块索引

1.1 模块

这里列出了所有模块:

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:

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:

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

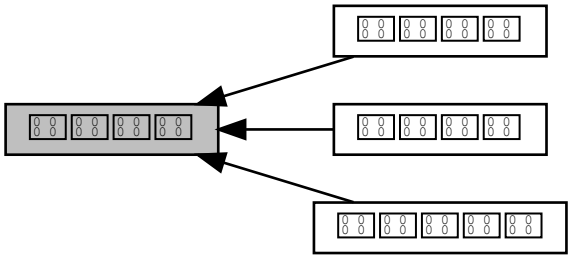
  指定一级索引（直接索引？）数量。目前必须是*10*，因为修改的话大概率出*bug*（因为我无法分辨那些不明所以的字面值）。
- #define **MAX_DIRLIST** 20

  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:



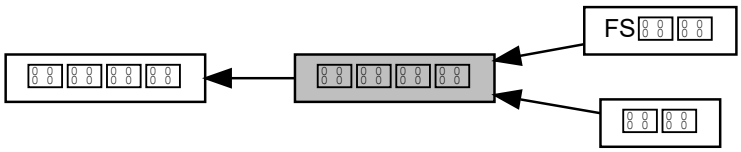
模块

- 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

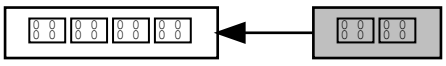
变量

- 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 **dirents_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_BLKN**(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 fino, 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 add_blk_for_file (
            int ino )
```

为文件申请新的数据块。

参数

ino	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

user	用户
cwd	接收修改后工作目录的inode编号
dir_name	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

user	文件属主
cwd	工作目录
args	参数（？不应该是name？）
mode	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

ino	当前目录的inode
f_ino	父目录的inode
dir_name	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

fino	父目录的inode编号
ino	子目录的inode编号
name	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

user	创建者
cwd	工作目录
args	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

user	文件属主
ino	文件获得的inode编号
mode	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

cwd	工作目录
args	参数
ino	inode编号

4.5.2.11 find_name_in_dir()

```
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。

参数

ino	目录项的inode编号
name	待查的文件名
mode	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

cwd	工作目录
argc	没有用
argv	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

ino	要查看的目录的inode编号。
args	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

user	目录属主
cwd	工作目录
dir_name	目录名
mode	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

user	执行操作的用户
cwd	工作目录
args	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

cwd	要取得目录名的inode编号
buf	缓冲区
ino	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

user	用户
cwd	工作目录
argc	没有用
argv	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

user	用户
cwd	工作目录
args	目标文件名？
workIno	希腊奶（？）
mode	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

d	待处理的目录
n	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

user	用户
cwd	工作目录
dir_name	目标目录名
mode	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

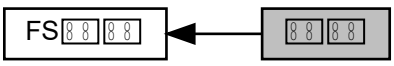
user	用户
cwd	工作目录
argc	没有用
argv	参数

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** [ ]

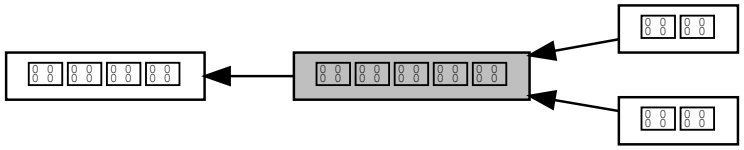
  位图管理，指示哪些块可用。

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

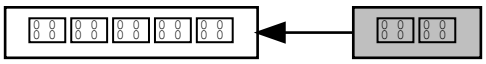
```
int idle_uid  [extern]
```

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

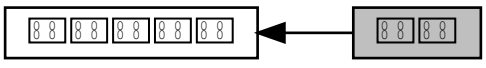
user	用户对象的指针
ino	文件的inode编号
mode	访问模式

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

添加用户。

NOTE：你这咋连返回值都没了？

参数

user	执行操作的用户身份
args	待添加的用户名

4.9.2.2 find_user()

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

查找用户。

参数

args	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

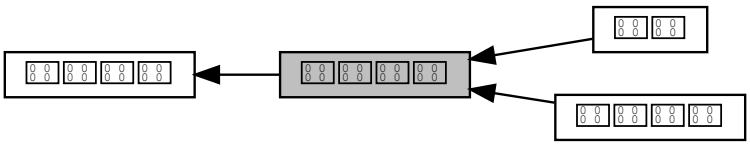
user	接收用户的登录参数
args	用户名。若为NULL，则执行交互式登录

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

p	待添加的指令

4.11.2.2 readline()

```
char * readline (
```

```
const char * str )
```

读入指令。

参数

str	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。

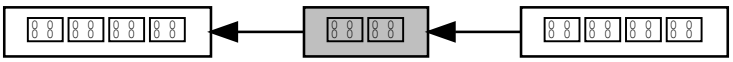
讲道理，这个函数到底是干啥的？

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

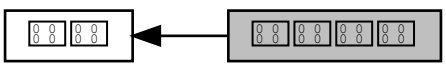
```
#define CMD_N 14
```

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:

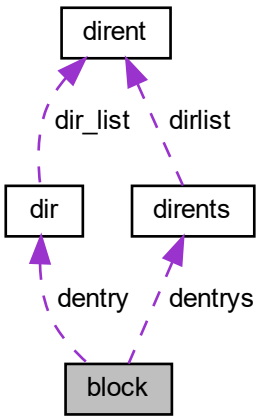
- include/config.h

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:

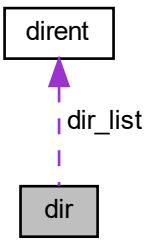
- include/shell/cmd.h

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/dir.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:

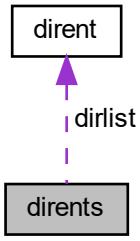
- include/sys/fs/types/dirent.h

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:

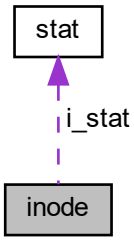
- include/sys/fs/types/dirents.h

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

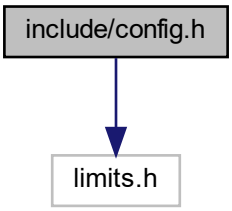
文件说明

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
8 #ifndef __CMJFS__CONFIG_HH__
9 #define __CMJFS__CONFIG_HH__ 1
10
11 #include <limits.h>
12
15
17 #define BLOCK_SIZE 512
19 #define BLOCK_NUM 512
21 #define INODE_NUM 512
23 #define INDIRECT_IDX_1_NUM 10
25 #define MAX_DIRLIST 20
26
28
29 // 下面的是全局编译期检查。**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

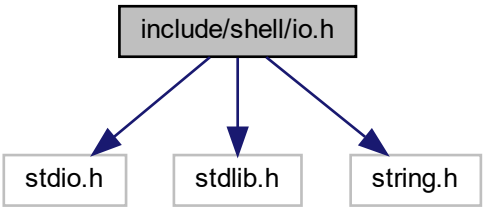
```
1
8 #ifndef __CMJFS__SHELL__COMMAND_H__
9 #define __CMJFS__SHELL__COMMAND_H__ 1
10
13
17 typedef struct cmd {
19     const char* name;
21     char* description;
23     void (*handler)(char* args);
24 } cmd_t;
25
31 #define CMD_N 14
32 // TODO: 这样的话sizeof是不行的，需要另想办法。
33
37 extern cmd_t cmd_table[];
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

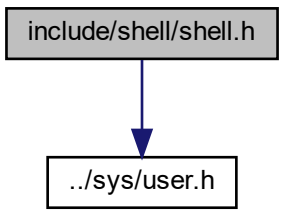
浏览该文件的文档.

```
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 add_history(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.

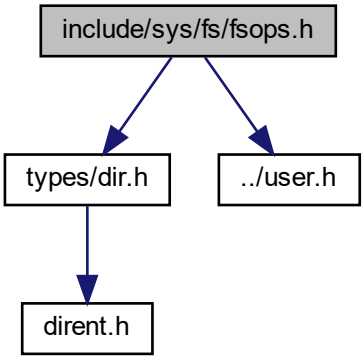
```
1
8 #ifndef __CMJFS__SHELL__SHELL_H__
9 #define __CMJFS__SHELL__SHELL_H__ 1
10
11 #include "../sys/user.h"
12
15
17 extern int work_dir;
19 extern user_t cnt_user;
20
22
23 #endif // __CMJFS__SHELL__SHELL_H__
```

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_BLKN**(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 fino, 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.

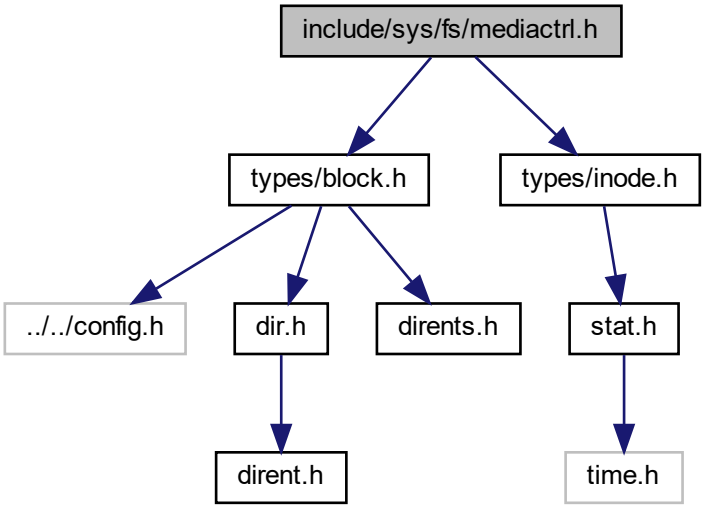
```
1
8 #ifndef __CMJFS__SYS_FS__FS_H__
9 #define __CMJFS__SYS_FS__FS_H__ 1
10
11 #include "types/dir.h"
12 #include "../user.h"
13
16
18 #define S_IFREG 0x0100000
20 #define S_IFDIR 0x0040000
22 #define RECURSIVE 0x800
23
25 #define GET_BLKN(ino,n) (BLK[inode[ino].blk1[n]])
26 // NOTE: 这，，，真就只有直接索引？
27 // TODO: 修改这个东西，我猜可能没法用宏来实现
28
30 #define GET_PAGE(ino,page) (page >= 10 ? BLK[inode[ino].blk2].index[page-10] : inode[ino].blk1[page])
32 #define GET_CHAR(ino,page,offset) (BLK[GET_PAGE(ino,page)].str[offset])
33 // 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。
34 // 以及，为什么是page不是block？
35
42 int add_blk_for_file(int ino);
43
52 int find_name_in_dir(int ino, const char* name, int mode);
53
62 int creat_stat(const user_t* user, int ino, int mode);
63
74 int creat_dirent(int fino, int ino, const char* name);
75
88 int creat_dir(int ino, int f_ino, const char* dir_name);
89 // TODO: 检查潜在bug
90
96 int creat_ino();
97
107 int creat(const user_t* user, int cwd, const char* args, int mode);
108
109 // TODO: 你truncate呢？？这个调用很重要啊
110
120 int mkdir(const user_t* user, int cwd, const char* dir_name, int mode);
121
130 void ls(int ino, char* args);
131
139 void pwd(int cwd, char* buf, int ino);
140
148 void cd(const user_t* user, int* cwd, const char* dir_name);
149
159 int rm_dir_item(dir_t* d, int n);
160
170 void rm(const user_t* user, int cwd, const char* args, int workIno, int mode);
171
180 void rmdir(const user_t* user, int cwd, char* dir_name, int mode);
181
189 void creat_file(const user_t* user, int cwd, const char* args);
190
199 int open_file(const user_t* user, int cwd, const char* args);
200
210 int read_file(const user_t* user, int cwd, int argc, const char* argv[]);
211
221 int write_file(const user_t* user, int cwd, int argc, const char* argv[]);
222
233 int link_file(int cwd, int argc, const char* argv[]);
234
242 void find_file(int cwd, const char* args, int ino);
243
245
246 #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.

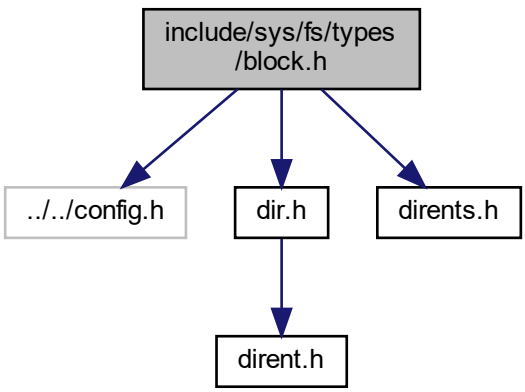
```
1
8 #ifndef __CMJFS__SYS_FS__MEDIACTRL_H__
9 #define __CMJFS__SYS_FS__MEDIACTRL_H__ 1
10
11 #include "types/block.h"
12 #include "types/inode.h"
13
17 #define SET_BLK_FLAG(i) (blk_flag[i / CHAR_BIT] |= '\x01' << (i % CHAR_BIT))
19 #define TEST_BLK_FLAG(i) (blk_flag[i / CHAR_BIT] & '\x01' << (i % 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

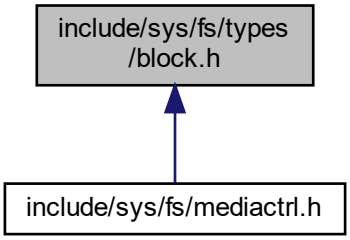
定义文件系统块。

```
#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.

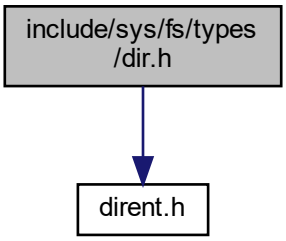
```
1
8 #ifndef __CMJFS__SYS_FS_TYPES__BLOCK_H__
9 #define __CMJFS__SYS_FS_TYPES__BLOCK_H__ 1
10
11 #include "../../config.h"
12 #include "dir.h"
13 #include "dirents.h"
14
26 typedef union block {
28     char str[BLOCK_SIZE];
30     dir_t dentry;
32     int index[BLOCK_SIZE / sizeof(int)];
34     dirents_t dentrys;
35 } block_t;
36
37 #endif // __CMJFS__SYS_FS_TYPES__BLOCK_H__
```

6.16 include/sys/fs/types/dir.h 文件参考

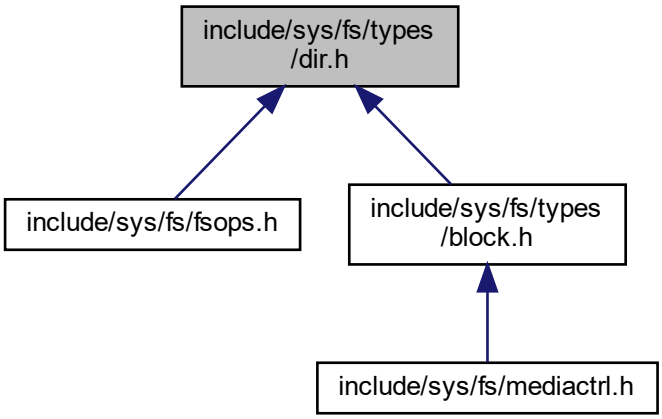
定义struct dir结构。

```
#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

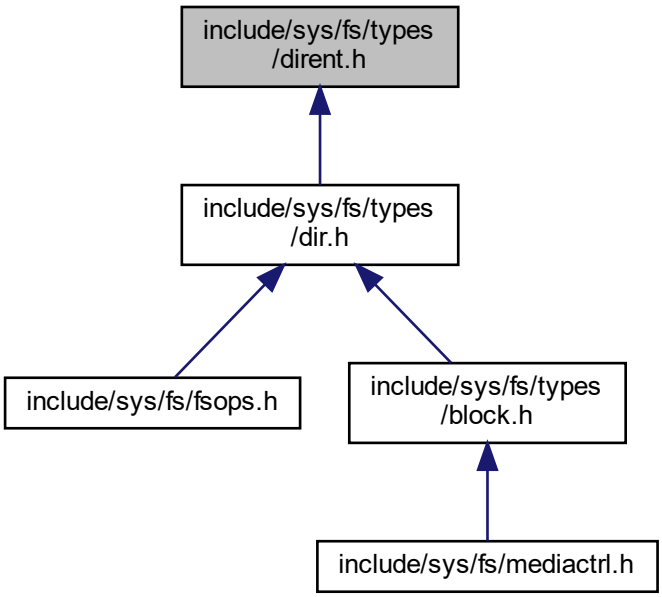
浏览该文件的文档.

```
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

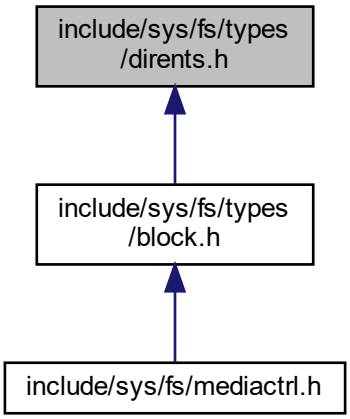
浏览该文件的文档.

```
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.

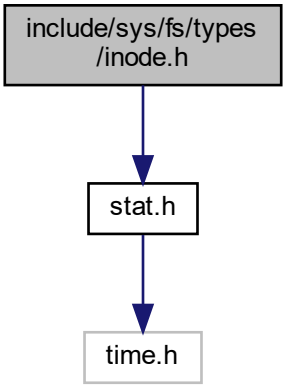
```
1
8 #ifndef __CMJFS__SYS_FS_TYPES_H__
9 #define __CMJFS__SYS_FS_TYPES_H__ 1
10
15 typedef struct dirents {
17     int dir_list_size;
19     dirent_t dirlist[21];
20 } dirents_t;
21
22 #endif // __CMJFS__SYS_FS_TYPES_H__
```

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

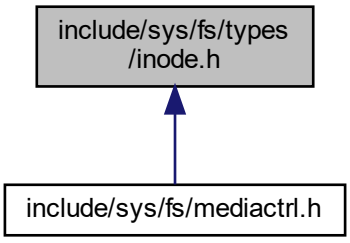
定义inode。

```
#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

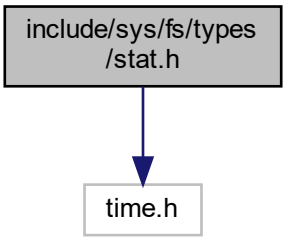
浏览该文件的文档.

```
1
8 #ifndef __CMJFS__SYS_FS_TYPES__INODE_H__
9 #define __CMJFS__SYS_FS_TYPES__INODE_H__ 1
10
11 #include "stat.h"
12
17 typedef struct inode {
19     stat_t i_stat;
21     int blk1[INDIRECT_IDX_1_NUM];
23     int blk2;
24 } inode_t;
25
26 #endif // __CMJFS__SYS_FS_TYPES__INODE_H__
```

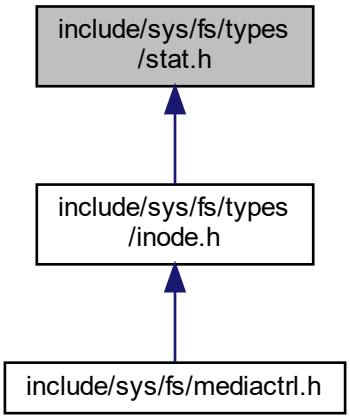
6.24 include/sys/fs/types/stat.h 文件参考

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

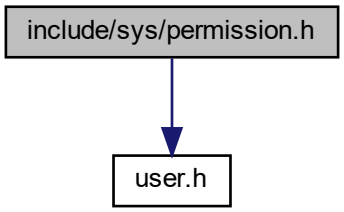
浏览该文件的文档.

```
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

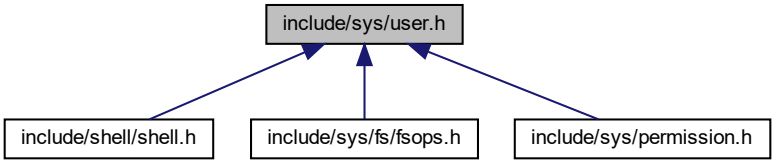
浏览该文件的文档.

```
1
8 #ifndef __CMJFS__SYS_FS__PERMISSION_H__
9 #define __CMJFS__SYS_FS__PERMISSION_H__ 1
10
11 #include "user.h"
12
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
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