Linux 内核源代码阅读

- 5130309717
- 朱文豪

We mainly focused on file object and its operations (functions) based in `<linux/fs.h>`. The kernel version is 4.4.3

File Object

The file object is used to represent a file. The file object is represented by struct file and is defined in `<linux/fs.h>`

The object (but not the physical file) is created in response to the `open()` system call and destroyed in response to the `close()` system call. Because multiple processes can open and manipulate a file at the same time, there can be multiple file objects in existence for the same file. The object points back to the dentry (which in turn points back to the inode) that actually represents the open file. The `inode` and `dentry` objects are unique.

```
struct file {
    union {
        struct llist_node   fu_llist /*文件对象列表*/;
        struct rcu_head     fu_rcuhead /*Read-copy update 表*/;
    } f_u;
    struct path     f_path;  /*保存了 dentry 的路径*/
    struct inode        *f_inode;   /* cached value */
    const struct file_operations    *f_op;   /*文件的操作表*/

    /*
     * Protects f_ep_links, f_flags.
     * Must not be taken from IRQ context.
     */
    spinlock_t      f_lock;  /*prefile struct lock*/
    atomic_long_t       f_count;  /*File对象使用计数*/
    unsigned int        f_flags;  /*文件打开标志*/
    fmode_t         f_mode; /*文件访问模式*/
    struct mutex        f_pos_lock;
    loff_t          f_pos;  /*文件偏移量（指针）*/
    struct fown_struct  f_owner; /*文件拥有者*/
    const struct cred   *f_cred;
    struct file_ra_state    f_ra;

    u64         f_version;
```

```
#ifdef CONFIG_SECURITY
    void            *f_security;
#endif
    /* needed for tty driver, and maybe others */
    void            *private_data;

#ifdef CONFIG_EPOLL
    /* Used by fs/eventpoll.c to link all the hooks to this file */
    struct list_head    f_ep_links;
    struct list_head    f_tfile_llink;
#endif /* #ifdef CONFIG_EPOLL */
    struct address_space    *f_mapping;
} __attribute__((aligned(4)));  /* lest something weird decides that 2 is
OK */
```

Similar to the `dentry` object, the file object does not actually correspond to any on-disk data. Therefore, no flag in the object represents whether the object is dirty and needs to be written back to disk. The file object does point to its associated `dentry` object via the `f_dentry` pointer.The `dentry` in turn points to the associated `inode` , which reflects whether the file itself is dirty.

File Operations

I selected some of the functions to describe. The file operations table is associated with struct file are the familiar system calls that form the basis of the standard Unix system calls. The file object methods are specified in `file_operations` and defined in `<linux/fs.h>`

loff_t llseek(struct file *file,loff_t offset, int origin)
Updates the file pointer to the given offset. It is called via the `llseek()` system call.

ssize_t read(struct file *file,char *buf, size_t count,loff_t *offset)
Reads count bytes from the given file at position offset into buf .The file pointer is then updated.This function is called by the `read()` system call.

ssize_t aio_read(struct kiocb *iocb,char *buf, size_t count,loff_t offset)
Begins an asynchronous read of count bytes into buf of the file. This function is called by the `aio_read()` system call.

ssize_t aio_write(struct kiocb *iocb,const char *buf,size_t count, loff_t offset)
Begins an asynchronous write of count bytes into buf of the file.

int open(struct inode *inode,struct file *file)
Creates a new file object and links it to the corresponding `inode` object. It is called by the `open()` system call.

int mmap(struct file *file,struct vm_area_struct *vma)
Memory maps the given file onto the given address space and is called by the `mmap()` system call.

Related data structures

Each process on the system has its own list of open files, root filesystem, current working directory, mount points…

```
struct files_struct {
    atomic_t count; /* usage count */
    struct fdtable *fdt; /* pointer to other fd table */
    struct fdtable fdtab; /* base fd table */
    spinlock_t file_lock;
    int next_fd;
    /* 下一个的缓存 */
    struct embedded_fd_set close_on_exec_init;
    struct embedded_fd_set open_fds_init   /* 打开的文件列表 */
    struct file *fd_array[NR_OPEN_DEFAULT]; /* base files array */
};
```

The `files_struct` is defined in `<linux/fdtable.h>` .This table's address is pointed to by the files entry in the processor descriptor. All pre- process information about open files are in the struct.