

Device Driver File Operations

**Advanced Embedded Linux
Development**
with **Dan Walkes**



University of Colorado **Boulder**

Learning objectives:

Driver data structures file_operations
and file

Open and Release

Fundamental Driver Data Structures

- file_operations
 - Connect drivers operations to device numbers
 - open, read, write, etc
 - object oriented program, where file is an object and functions are methods.
 - Also called fops

Fundamental Driver Data Structures

- file
 - Not related to FILE* for buffered I/O
 - Represents an open file (specific open/close instance)
 - Driver open from mknod node /dev/yourdev
 - File in the filesystem
 - Called either a “file” or “filp” (file pointer)
- inode
 - Represents files (not an open file descriptor)

file_operations

- Registered with register_chrdev
- Definitions use C tagged structure initialization syntax

```
static ssize_t sleepy_read (struct file *filp, char __user *buf, size_t count, loff_t *pos)
{
}

struct file_operations sleepy_fops = {
    .owner = THIS_MODULE,
    .read = sleepy_read,
    .write = sleepy_write,
};

static int sleepy_init(void)
{
    /*
     * Register your major, and accept a dynamic number
     */
    result = register_chrdev(sleepy_major, "sleepy", &sleepy_fops);
```

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*read_iter) (struct kiocb *, struct iovec *);
    ssize_t (*write_iter) (struct kiocb *, struct iovec *);
    int (*iopoll) (struct kiocb *, bool spin);
    int (*iterate) (struct file *, struct dir_context *);
    int (*iterate_shared) (struct file *, struct dir_context *);
    __poll_t (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    unsigned long mmap_supported_flags;
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area) (struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
    int (*check_flags) (int);
    int (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write) (struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
    ssize_t (*splice_read) (struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
    int (*setlease) (struct file *, long, struct file_lock *, void **);
```

Linux Device Drivers 3rd Edition Chapter 3

<https://github.com/cu-ecen-5013/ldd3/blob/master/misc-modules/sleepy.c>

<https://stackoverflow.com/a/3017026/1446624>

<https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h#L1821>

file_operations function definitions

- Definitions in the book are out of date.
- How to versions matching your kernel?
 - RTSL - Read The Source Luke!
 - Linux kernel git source
<https://github.com/torvalds/linux>
 - Linux kernel cross reference
<https://elixir.bootlin.com>

file_operations function definitions

- Latest list from cross reference
 - Most operate on the struct file* object as passed parameter

```
struct file_operations {  
    struct module *owner;  
    loff_t (*llseek) (struct file *, loff_t, int);  
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);  
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);  
    ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);  
    ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);  
    int (*iopoll) (struct kiocb *kiocb, bool spin);  
    int (*iterate) (struct file *, struct dir_context *);  
    int (*iterate_shared) (struct file *, struct dir_context *);  
    __poll_t (*poll) (struct file *, struct poll_table_struct *);  
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);  
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);  
    int (*mmap) (struct file *, struct vm_area_struct *);  
    unsigned long mmap_supported_flags;  
    int (*open) (struct inode *, struct file *);  
    int (*flush) (struct file *, fl_owner_t id);  
    int (*release) (struct inode *, struct file *);  
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);  
    int (*fasync) (int, struct file *, int);  
};
```

file structure

- `f_mode` - readable or writable
- `f_pos` - current read/write position
- `f_flags` - `O_RDONLY`, `O_NONBLOCK`, `O_SYNC`
- `struct file_operations *f_op` - pointer to the file ops table
- `void *private_data` - your driver can use this location to hold a pointer to allocated memory.

```
struct file {
```

```
    fmode_t      f_mode;
```

```
    loff_t       f_pos;
```

```
    unsigned int f_flags;
```

```
    const struct file_operations *f_op;
```

```
    void         *private_data;
```


Char Device Registration

- Option 1: Allocated as a chunk of memory

```
struct cdev *my_cdev = cdev_alloc();  
my_cdev->ops = &my_fops;
```

- Option 2: Initialized within your own structure <-
More typically used

```
void cdev_init(struct cdev *cdev, struct file_operations *fops);
```

- Then tell the kernel about it

```
int cdev_add(struct cdev *dev, dev_t num, unsigned int count);
```

```
void cdev_del(struct cdev *dev);
```

Char Device Registration

- Scull device registration

```
struct scull_dev {  
    struct scull_qset *data; /* Pointer to first quantum set */  
    int quantum;             /* the current quantum size */  
    int qset;                /* the current array size */  
    unsigned long size;      /* amount of data stored here */  
    unsigned int access_key; /* used by sculluid and scullpriv */  
    struct semaphore sem;    /* mutual exclusion semaphore */  
    struct cdev cdev;        /* Char device structure */  
};
```

open method

- Open should typically:
 - Check for device errors (not ready or other hardware problems)
 - Initialize the device on first open
 - Update the `f_ops` pointer to `file_operations`
 - Allocate and fill/set `private_data`
- Open prototype:

```
int (*open) (struct inode *, struct file *);
```
- Notably missing from prototype: `c_dev` structure

open method

```
int (*open) (struct inode *, struct file *);
```

- How do we get the c_dev pointer?

```
struct inode {
```

```
    union {  
        struct pipe_inode_info *i_pipe;  
        struct block_device *i_bdev;  
        struct cdev *i_cdev;  
        char *i_link;  
        unsigned i_dir_seq;  
    };
```

container_of

- How do we get the `scull_dev` structure from `c_dev` pointer?

```
struct scull_dev {  
    struct scull_qset *data; /* Pointer to first quantum set */  
    int quantum;             /* the current quantum size */  
    int qset;                /* the current array size */  
    unsigned long size;      /* amount of data stored here */  
    unsigned int access_key; /* used by sculluid and scullpriv */  
    struct semaphore sem;    /* mutual exclusion semaphore */  
    struct cdev cdev;        /* Char device structure */  
};
```

```
container_of(pointer, container_type, container_field);
```

```
int scull_open(struct inode *inode, struct file *filp)  
{  
    struct scull_dev *dev; /* device information */  
  
    dev = container_of(inode->i_cdev, struct scull_dev, cdev);  
    filp->private_data = dev; /* for other methods */  
}
```

release method

- release is the reverse of open
 - Deallocate anything open() allocated in filp->private_data
- What does the scull implementation need to do on release?
 - Nothing, since the data pointed to in filp->private_data structure was allocated in init_module (not open())
 - Therefore should be freed in module_exit()

```
int scull_release(struct inode *inode, struct file *filp)
{
    return 0;
}
/*
```

```
int scull_init_module(void)
{
```

```
    scull_devices = kmalloc(scull_nr_devs * sizeof(struct scull_dev), GFP_KERNEL);
```

release method

- release is the reverse of open
 - Deallocate anything open() allocated in filp->private_data
- What if the user space programmer forgets to close the file?
 - Kernel cleans up automatically at process exit
 - Your driver is guaranteed exactly one release() per open()
 - Multiple processes (fork/dup) may close(), only the last close() results in release()