Linux中的File_operations结构体

2011-03-01 nashty
分类：LINUX

File_operations结构体

file_operation就是把系统调用和驱动程序关联起来的关键数据结构。这个结构的每一个成员都对应着一个系统调用。读取file_operation中相应的函数指针，接着把控制权转交给函数，从而完成了Linux设备驱动程序的工作。

在系统内部，I/O设备的存取操作通过特定的入口点来进行，而这组特定的入口点恰恰是由设备驱动程序提供的。通常这组设备驱动程序接口是由结构file_operations结构体向系统说明的，它定义在include/linux/fs.h中。

传统上, 一个 file_operation 结构或者其一个指针称为 fops(或者它的一些变体). 结构中的每个成员必须指向驱动中的函数, 这些函数实现一个特别的操作, 或者对于不支持的操作留置为 NULL. 当指定为 NULL 指针时内核的确切的行为是每个函数不同的。

在你通读 file_operations 方法的列表时, 你会注意到不少参数包含字串 __user. 这种注解是一种文档形式, 注意, 一个指针是一个不能被直接解引用的用户空间地址. 对于正常的编译, __user 没有效果, 但是它可被外部检查软件使用来找出对用户空间地址的错误使用。

注册设备编号仅仅是驱动代码必须进行的诸多任务中的第一个。首先需要涉及一个别的，大部分的基础性的驱动操作包括 3 个重要的内核数据结构，称为 file_operations，file，和 inode。需要对这些结构的基本了解才能够做大量感兴趣的事情。

struct file_operations是一个字符设备把驱动的操作和设备号联系在一起的纽带，是一系列指针的集合，每个被打开的文件都对应于一系列的操作，这就是file_operations,用来执行一系列的系统调用。

struct file代表一个打开的文件，在执行file_operation中的open操作时被创建，这里需要注意的是与用户空间inode指针的区别，一个在内核，而file指针在用户空间，由c库来定义。

struct inode被内核用来代表一个文件，注意和struct file的区别，struct inode一个是代表文件，struct file一个是代表打开的文件

struct inode包括很重要的二个成员：

dev_t i_rdev 设备文件的设备号

struct cdev *i_cdev 代表字符设备的数据结构

struct inode结构是用来在内核内部表示文件的.同一个文件可以被打开好多次,所以可以对应很多struct file,但是只对应一个struct inode.

File_operations的数据结构如下：

struct module *owner

第一个 file_operations 成员根本不是一个操作; 它是一个指向拥有这个结构的模块的指针. 这个成员用来在它的操作还在被使用时阻止模块被卸载. 几乎所有时间中, 它被简单初始化为 THIS_MODULE, 一个在 中定义的宏.

loff_t (*llseek) (struct file *, loff_t, int);

llseek 方法用作改变文件中的当前读/写位置, 并且新位置作为(正的)返回值. loff_t 参数是一个"long offset", 并且就算在 32位平台上也至少 64 位宽. 错误由一个负返回值指示. 如果这个函数指针是 NULL, seek 调用会以潜在地无法预知的方式修改 file 结构中的位置计数器(在"file 结构" 一节中描述).

ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);

用来从设备中获取数据. 在这个位置的一个空指针导致 read 系统调用以 -EINVAL("Invalid argument") 失败. 一个非负返回值代表了成功读取的字节数(返回值是一个 "signed size" 类型, 常常是目标平台本地的整数类型).

ssize_t (*aio_read)(struct kiocb *, char __user *, size_t, loff_t);

初始化一个异步读 -- 可能在函数返回前不结束的读操作. 如果这个方法是 NULL, 所有的操作会由 read 代替进行(同步地).

ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);

发送数据给设备. 如果 NULL, -EINVAL 返回给调用 write 系统调用的程序. 如果非负, 返回值代表成功写的字节数.

ssize_t (*aio_write)(struct kiocb *, const char __user *, size_t, loff_t *);

初始化设备上的一个异步写.

int (*readdir) (struct file *, void *, filldir_t);

对于设备文件这个成员应当为 NULL; 它用来读取目录, 并且仅对文件系统有用.

unsigned int (*poll) (struct file *, struct poll_table_struct *);

poll 方法是 3 个系统调用的后端: poll, epoll, 和 select, 都用作查询对一个或多个文件描述符的读或写是否会阻塞. poll 方法应当返回一个位掩码指示是否非阻塞的读或写是可能的, 并且, 可能地, 提供给内核信息用来使调用进程睡眠直到 I/O 变为可能. 如果一个驱动的 poll 方法为 NULL, 设

备假定为不阻塞地可读可写.

int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);

ioctl 系统调用提供了发出设备特定命令的方法(例如格式化软盘的一个磁道, 这不是读也不是写). 另外, 几个 ioctl 命令被内核识别而不必引用 fops 表. 如果设备不提供 ioctl 方法, 对于任何未事先定义的请求(-ENOTTY, "设备无这样的 ioctl"), 系统调用返回一个错误.

int (*mmap) (struct file *, struct vm_area_struct *);

mmap 用来请求将设备内存映射到进程的地址空间. 如果这个方法是 NULL, mmap 系统调用返回 -ENODEV.

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

尽管这常常是对设备文件进行的第一个操作, 不要求驱动声明一个对应的方法. 如果这个项是 NULL, 设备打开一直成功, 但是你的驱动不会得到通知.

int (*flush) (struct file *);

flush 操作在进程关闭它的设备文件描述符的拷贝时调用; 它应当执行(并且等待)设备的任何未完成的操作. 这个必须不要和用户查询请求的 fsync 操作混淆了. 当前, flush 在很少驱动中使用; SCSI 磁带驱动使用它, 例如, 为确保所有写的数据在设备关闭前写到磁带上. 如果 flush 为 NULL, 内核简单地忽略用户应用程序的请求.

int (*release) (struct inode *, struct file *);

在文件结构被释放时引用这个操作. 如同 open, release 可以为 NULL.

int (*fsync) (struct file *, struct dentry *, int);

这个方法是 fsync 系统调用的后端, 用户调用来刷新任何挂着的数据. 如果这个指针是 NULL, 系统调用返回 -EINVAL.

int (*aio_fsync)(struct kiocb *, int);

这是 fsync 方法的异步版本.

int (*fasync) (int, struct file *, int);

这个操作用来通知设备它的 FASYNC 标志的改变. 异步通知是一个高级的主题, 在第 6 章中描述. 这个成员可以是NULL 如果驱动不支持异步通知.

int (*lock) (struct file *, int, struct file_lock *);

lock 方法用来实现文件加锁; 加锁对常规文件是必不可少的特性, 但是设备驱动几乎从不实现它.

ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);

ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);

这些方法实现发散/汇聚读和写操作. 应用程序偶尔需要做一个包含多个内存区的单个读或写操作; 这些系统调用允许它们这样做而不必对数据进行额外拷贝. 如果这些函数指针为 NULL, read 和 write 方法被调用(可能多于一次).

ssize_t (*sendfile)(struct file *, loff_t *, size_t, read_actor_t, void *);

这个方法实现 sendfile 系统调用的读, 使用最少的拷贝从一个文件描述符搬移数据到另一个. 例如, 它被一个需要发送文件内容到一个网络连接的 web 服务器使用. 设备驱动常常使 sendfile 为 NULL.

ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);

sendpage 是 sendfile 的另一半; 它由内核调用来发送数据, 一次一页, 到对应的文件. 设备驱动实际上不实现 sendpage.

unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);

这个方法的目的是在进程的地址空间找一个合适的位置来映射在底层设备上的内存段中. 这个任务通常由内存管理代码进行; 这个方法存在为了使驱动能强制特殊设备可能有的任何的对齐请求. 大部分驱动可以置这个方法为 NULL.

int (*check_flags)(int)

这个方法允许模块检查传递给 fnctl(F_SETFL...) 调用的标志.

int (*dir_notify)(struct file *, unsigned long);

这个方法在应用程序使用 fcntl 来请求目录改变通知时调用. 只对文件系统有用; 驱动不需要实现 dir_notify.

scull 设备驱动只实现最重要的设备方法. 它的 file_operations 结构是如下初始化的:

struct file_operations scull_fops = {

```
.owner = THIS_MODULE,
.llseek = scull_llseek,
.read = scull_read,
.write = scull_write,
.ioctl = scull_ioctl,
.open = scull_open,
.release = scull_release,
};
```

这个声明使用标准的C标记式结构初始化语法. 这个语法是首选的, 因为它使驱动在结构定义的改变之间更加可移植, 并且, 有争议地, 使代码更加紧凑和可读. 标记式初始化允许结构成员重新排序; 在某种情况下, 真实的性能提高已经实现, 通过安放经常使用的成员的指针在相同硬件高速存储行中.

本文来自：http://hi.baidu.com/cui1206/blog/item/a3c630ec6faddfd42e2e217c.html

上一篇：Linux Kconfig及Makefile学习
下一篇：linux2.6.11 内核基础知识