Linux Device Driver

(Block Devices)

Amir Hossein Payberah

payberah@yahoo.com

Contents

- Registering the Driver
- blk.h
- Handling request
- Mount and umount
- Ioctl
- Removable devices

Block device

- Like char devices, block devices are accessed by filesystem nodes in the /dev directory.
- A block device is something that can host a filesystem, such as a disk.
- A block device can be accessed only as multiples of a block,
    - □ A block is usually one kilobyte of data or another power of 2.

Registering the driver

- Like char drivers, block drivers in the kernel are identified by major numbers.
- Block major numbers are entirely distinct from char major numbers.
  - □ A block device with major number 32 can coexist with a char device using the same major number since the two ranges are separate.

Registering the driver

- int register_blkdev(unsigned int major, const char *name, struct block_device_operations *bdops);
- int unregister_blkdev(unsigned int major, const char *name);
- They are defined in <linux/fs.h>.

block_device_operations

```
struct block_device_operations
{
   int (*open) (struct inode *inode, struct file *filp);
   int (*release) (struct inode *inode, struct file *filp);
   int (*ioctl) (struct inode *inode, struct file
   *filp,unsigned command, unsigned long
   argument);
   int (*check_media_change) (kdev_t dev);
   int (*revalidate) (kdev_t dev);
};
```

Block device read/write

- There are no read or write operations provided in the block_device_operations structure.
- All I/O to block devices is normally buffered by the system.
- User processes do not perform direct I/O to these devices.
  - User-mode access to block devices usually is implicit in filesystem operations they perform (those operations clearly benefit from I/O buffering).
  - However, even ‘‘direct’’ I/O to a block device, such as when a filesystem is created, goes through the Linux buffer cache.

Block device read/write

- The kernel provides a single set of read and write functions for block devices, and drivers do not need to worry about them.
- In Linux, the method used for these I/O operations is called request.
- The request method handles both read and write operations and can be somewhat complex.

Request method

- For the purposes of block device registration, however, we must tell the kernel where our request method is.
- blk_init_queue(request_queue_t *queue, request_fn_proc *request);
- blk_cleanup_queue(request_queue_t *queue);
- They are defined in <linux/blkdev.h>

Device request queue

- Each device has a request queue that it uses by default.
- BLK_DEFAULT_QUEUE(major)
  - It is used to indicate that queue when needed.
  - This macro looks into a global array of blk_dev_struct structures.

Sample

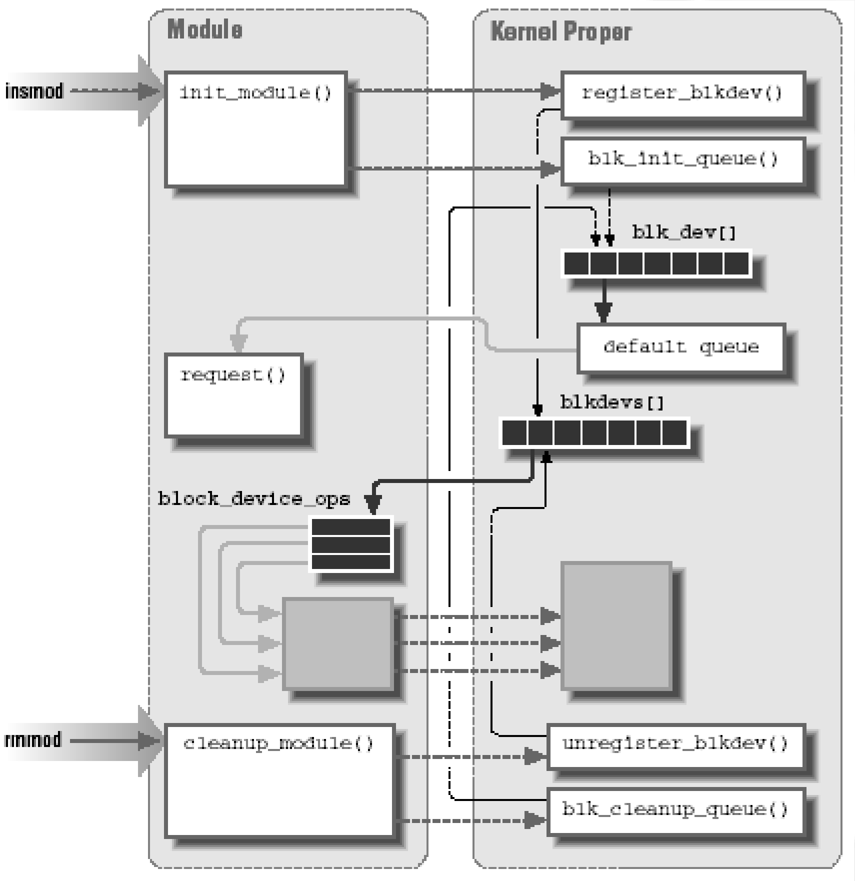
- blk_init_queue(BLK_DEFAULT_QUEUE (major), sbull_request);

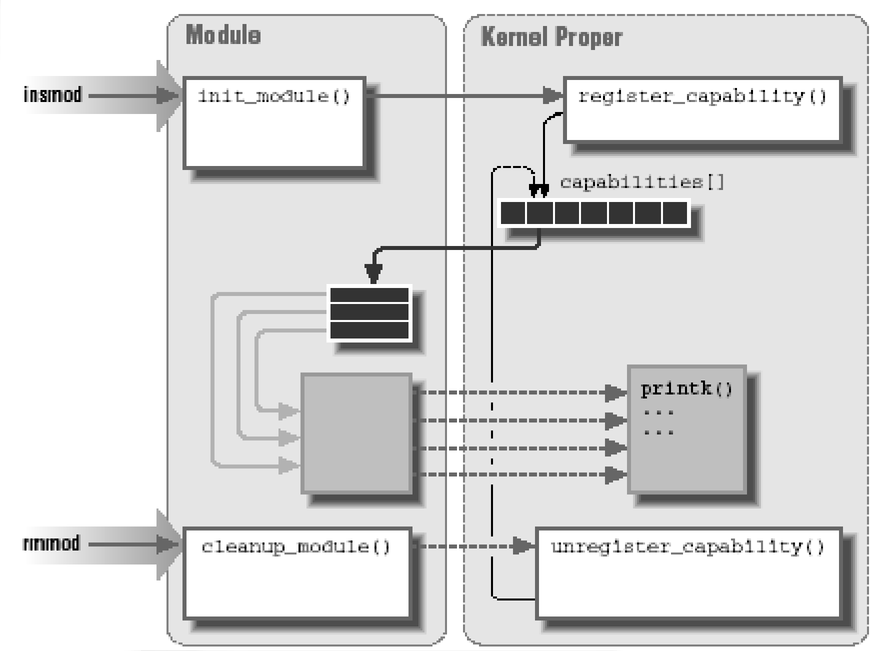
blk_dev_struct

```
struct blk_dev_struct
{
   request_queue_t request_queue;
   queue_proc *queue;
   void *data;
};
```

- The request_queue member contains the I/O request queue.
- The data field may be used by the driver for its own data.

Block vs Character





Block Device

Char Device

Block device global arrays

- struct blk_dev_struct blk_dev[]
- int blk_size[][]
  - It describes the size of each device, in kilobytes.
- int blksize_size[][]
  - The size of the block used by each device, in bytes.
- int hardsect_size[][]
  - The size of the hardware sector used by each device, in bytes.

Block device global arrays

- int read_ahead[] and int max_readahead[][]
  - These arrays define the number of sectors to be read.
- int max_sectors[][]
  - This array limits the maximum size of a single request.
- int max_segments[]
  - This array controlled the number of individual segments that could appear in a clustered request.

Sample

```
read_ahead[major] = sbull_rahead;
sbull_sizes = kmalloc(sbull_devs * sizeof(int), GFP_KERNEL);
for (i=0; i < sbull_devs; i++)
    sbull_sizes[i] = sbull_size;
blk_size[major]=sbull_sizes;
sbull_blksizes = kmalloc(sbull_devs * sizeof(int), GFP_KERNEL);
for (i=0; i < sbull_devs; i++)
    sbull_blksizes[i] = sbull_blksize;
blksize_size[major]=sbull_blksizes;
sbull_hardsects = kmalloc(sbull_devs * sizeof(int),
    GFP_KERNEL);
for (i=0; i < sbull_devs; i++)
    sbull_hardsects[i] = sbull_hardsect;
hardsect_size[major]=sbull_hardsects;
```

Register disk

- One last thing that must be done is to register every ‘‘disk’’ device provided by the driver.
- register_disk(struct gendisk *gd, int drive, unsigned minors, struct block_device_operations *ops, long size);
- A block driver without partitions will work without this call in 2.4.0, but it is safer to include it.

Sample

```
for (i = 0; i < sbull_devs; i++)
    register_disk(NULL,
    MKDEV(major, i), 1, &sbull_bdops,
    sbull_size << 1);
```

Cleanup block device

- The call to fsync_dev is needed to free all references to the device that the kernel keeps in various caches.

Sample

```
for (i=0; i<sbull_devs; i++)
    fsync_dev(MKDEV(sbull_major, i));
unregister_blkdev(major, "sbull");
blk_cleanup_queue(BLK_DEFAULT_QUEUE(major));

read_ahead[major] = 0;
kfree(blk_size[major]);
blk_size[major] = NULL;
kfree(blksize_size[major]);
blksize_size[major] = NULL;
kfree(hardsect_size[major]);
hardsect_size[major] = NULL;
```

Contents

- Registering the Driver
- blk.h
- Handling request
- Mount and umount
- Ioctl
- Removable devices

The Header File blk.h

- All block drivers should include the header file <linux/blk.h>.
- This file defines much of the common code that is used in block drivers.
- It provides functions for dealing with the I/O request queue.

Module compile notes

- the blk.h header is quite unusual.
- It defines several symbols based on the symbol MAJOR_NR.
  - It must be declared by the driver before it includes the header.

blk.h symbols

- MAJOR_NR
  - This symbol is used to access a few arrays.
- DEVICE_NAME
  - The name of the device being created.
- DEVICE_NR(kdev_t device)
  - This symbol is used to extract the ordinal number of the physical device from the kdev_t device number.
  - The value of this macro can be MINOR(device).
- DEVICE_INTR
  - This symbol is used to declare a pointer variable that refers to the current bottom-half handler.

blk.h symbols

- DEVICE_ON(kdev_t device) & DEVICE_OFF(kdev_t device)
  - These macros are intended to help devices that need to perform processing before or after a set of transfers is performed.
  - for example, they could be used by a floppy driver to start the drive motor before I/O and to stop it afterward.
- DEVICE_NO_RANDOM
  - By default, the function end_request contributes to system entropy, which is used by /dev/random.
  - If the device isn't able to contribute significant entropy to the random device, DEVICE_NO_RANDOM should be defined.
- DEVICE_REQUEST
  - Used to specify the name of the request function used by the driver.

Sample

```
#define MAJOR_NR sbull_major
static int sbull_major;
#define DEVICE_NR(device) MINOR(device)
#define DEVICE_NAME "sbull"
#define DEVICE_INTR sbull_intrptr
#define DEVICE_NO_RANDOM
#define DEVICE_REQUEST sbull_request
#define DEVICE_OFF(d)
#include <linux/blk.h>
```

Contents

- Registering the Driver
- blk.h
- Handling request
- Mount and umount
- Ioctl
- Removable devices

Request function

- The most important function in a block driver is the request function.
- It performs the low-level operations related to reading and writing data.

Request queue

- When the kernel schedules a data transfer, it queues the request in a list, ordered in such a way that it maximizes system performance.
- The queue of requests is passed to the driver’s request function.
- void request_fn(request_queue_t *queue);

Request function tasks

- Check the validity of the request. This test is performed by the macro INIT_REQUEST.
- Perform the actual data transfer.
  - The CURRENT variable (a macro, actually) can be used to retrieve the details of the current request.
- Clean up the request just processed.
  - This operation is performed by end_request.
- Loop back to the beginning, to consume the next request.

Sample

```
void sbull_request(request_queue_t *q)
{
    while(1)
    {
        INIT_REQUEST;
        printk("<1>request %p: cmd %i sec %li (nr. %li)\n",
    CURRENT,
                CURRENT->cmd,
                CURRENT->sector,
                CURRENT->current_nr_sectors);
        end_request(1);
    }
}
```

CURRENT

- CURRENT is a pointer to struct request.
- kdev_t rq_dev;
  - The device accessed by the request.
- int cmd;
  - This field describes the operation to be performed; it is either READ or WRITE.
- unsigned long sector;
  - The number of the first sector to be transferred in this request.

CURRENT

- unsigned long current_nr_sectors & unsigned long nr_sectors;
  - The number of sectors to transfer for the current request.
- char *buffer;
  - The area in the buffer cache to which data should be written or read.
- struct buffer_head *bh;
  - The structure describing the first buffer in the list for this request.

Sample

```
void sbull_request(request_queue_t *q)
{
        while(1)
        {
                INIT_REQUEST; /* returns when queue is empty */
                status = sbull_transfer(device, CURRENT);
                end_request(status);
        }
}
//-----------------------------------------------------------------------
static int sbull_transfer(Sbull_Dev *dev, const struct request *req)
{
        ptr = device->data + req->sector * sbull_hardsect;
        size = req->current_nr_sectors * sbull_hardsect;
        switch(req->cmd)
        {
                case READ:
                                memcpy(req->buffer, ptr, size); /* from sbull to buffer */
                                return 1;
                case WRITE:
                                memcpy(ptr, req->buffer, size); /* from buffer to sbull */
                                return 1;
        }
}
```

Contents

- Registering the Driver
- blk.h
- Handling request
- ⇨ Mount and umount
- Ioctl
- Removable devices

Mount

- When the kernel mounts a device in the filesystem, it invokes the normal open method to access the driver.
- in this case both the filp and inode arguments to open are dummy variables.
- In the file structure, only the f_mode and f_flags fields hold anything meaningful.
  - The value of f_mode tells the driver whether the device is to be mounted read-only (f_mode == FMODE_READ) or read/write (f_mode == (FMODE_READ| FMODE_WRITE)).
- In the inode structure only i_rdev may be used.

Umount

- As far as umount is concerned, it just flushes the buffer cache and calls the release driver method.
- There is no meaningful filp to pass to the release method.

Contents

- Registering the Driver
- blk.h
- Handling request
- Mount and umount
- Ioctl
- Removable devices

The ioctl method

- The only relevant difference between block and char ioctl implementations is that block drivers share a number of common ioctl commands that most drivers are expected to support.

Common commands

- BLKGETSIZE
  - Retrieve the size of the current device, expressed as the number of sectors.
- BLKFLSBUF
  - Literally, ‘‘flush buffers.’’
- BLKRRPART
  - Reread the partition table.
- BLKRAGET & BLKRASET
  - Used to get and change the current block-level read-ahead value for the device.

Common commands

- BLKFRAGET & BLKFRASET
  - Get and set the filesystem-level read-ahead value.
- BLKROSET & BLKROGET
  - used to change and check the read-only flag for the device.
- BLKSECTGET & BLKSECTSET
  - retrieve and set the maximum number of sectors per request.
- BLKSSZGET
  - Returns the sector size of this block device.

Common commands

- BLKPG
  - Allows user-mode programs to add and delete partitions.
- BLKELVGET & BLKELVSET
  - These commands allow some control over how the elevator request sorting algorithm works.
- HDIO_GETGEO
  - Used to retrieve the disk geometry.

Sample

```
int sbull_ioctl (struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)
{
	struct hd_geometry geo;
	switch(cmd)
	{
		case BLKGETSIZE:
				size = blksize*sbull_sizes[MINOR(inode->i_rdev)]/sbull_hardsects[MINOR(inode->i_rdev)];
				copy_to_user((long *) arg, &size, sizeof (long));
				return 0;
		case BLKRRPART:
				return -ENOTTY;
		case HDIO_GETGEO:
				size = sbull_size * blksize / sbull_hardsect;
				geo.cylinders = (size & ˜0x3f) >> 6;
				geo.heads = 4;
				geo.sectors = 16;
				geo.start = 4;
				copy_to_user((void *) arg, &geo, sizeof(geo));
				return 0;
	}
}
```

Contents

- Registering the Driver
- blk.h
- Handling request
- Mount and umount
- Ioctl
- Removable devices

check_media_change

- The checking function receives kdev_t as a single argument that identifies the device.
- The return value is 1 if the medium has been changed and 0 otherwise.

Sample

```
int sbull_check_change(kdev_t i_rdev)
{
    int minor = MINOR(i_rdev);
    Sbull_Dev *dev = sbull_devices + minor;
    if (dev->data)
        return 0; /* still valid */
    return 1; /* expired */
}
```

Revalidation

- The validation function is called when a disk change is detected.

Sample

```
int sbull_revalidate(kdev_t i_rdev)
{
   Sbull_Dev *dev = sbull_devices + MINOR(i_rdev);
   if (dev->data)
         return 0;
   dev->data = vmalloc(dev->size);
   if (!dev->data)
         return -ENOMEM;
   return 0;
}
```

Question?