Writing Device Drivers Under Linux

Block Device Drivers

Block Device Drivers

- Major distinction: devices which can be mounted as filesystems
- Kernel provides a buffer cache abstraction
- Low-level devices drivers supplies blocks to the buffer cache abstraction

Warning: Still under construction

- Block device abstraction still not yet clean
  - Design dates back to the original 0.01 kernel
- Likely to be redesigned during the 2.1 development cycle
- Important to look at sample device drivers
- Also check ll_rw_blk.c and genhd.c

Kernel-provided functionality

- Interface to the VFS (block_read and block_write)
- Maintenance of queue of requests to the device driver
- Request queue sorted using elevator algorithm
- Therefore, "all" you have to provide is the request() function

The request() function

- Interface between the kernel-provided, device independent code and the device driver
- Called by the device independent layer where there is a read or write request which needs to be handled.

January 7, 1997 Block Device Drivers 5

The request() function II

- Device driver reads or writes the requested block into (or out of) the provided buffer page
- Device driver calls end_request() when the transfer is complete

January 7, 1997 Block Device Drivers 6

Block driver framework

- See ramdisk.c (page 68 in the Example Sources)
- blk.h magic (page 63 in the Example Sources)
  - See RAMDISK specific lines at blk.h:87

The request queue

- The kernel supplies a single request queue for each major number
- Contains a sorted list of work to be done
- See do_rd_request() (ramdisk.c:33)

Servicing a request

- INIT_REQUEST
- If the request is an error, call end_request(0) and repeat
- If the request is successful, call end_request(1) and repeat

Initialization function

- See rd_init() (ramdisk.c:78)
- Basic operations
  - Register the driver
    - the file_operations table
    - the request function
  - Ramdisk-specific initializations
  - Set up the blksize_size array, which specifies the natural block size
    - Also an optional blk_size array

XT driver sample

- See xd.c (page 71 in the Example Sources)
- Simple device driver for pre-IDE disk controllers
- Uses interrupts, but written in a polling style
- Supports use of partition tables

Partition table services

- A partition table divides up a disk into multiple smaller areas
- Represents a convention honored by all of the users of the disk
- Linux supports several different partitioning schemes
  - MS-DOS, BSD disklabel, Sun disklabel, etc.
- Support provided by the gendisk abstraction

Writing gendisk device drivers

- See gen_hd.h (page 68 in your Example Sources Book)
- struct gendisk used as the interface point between the driver and the gendisk abstraction
- See declaration of xd_gendisk (xd.c:110)

Initialization function

- See xd_init() (xd.c:148)
- Basic operations
  - Register the driver
    - the file_operations table
    - the request table
  - Add xd_gendisk to the gendisk linked list

The gendisk function

- See xd_geninit() (xd.c:199)
- Called by the gendisk layer
- Performs hardware initialization
- Determine the size of the full disk

Standard block device ioctl's

- See xd_ioctl() (xd.c:301)
- Standard block device ioctl functions
  - BLKRASET
  - BLKFLSBUF
  - BLKRPART

Re-reading the partition table

- See xd_reread_partitions() (xd.c:375)
- Called by the BLKRPART ioctl
  - Clears buffered inodes/blocks
  - Calls resetup_one_dev() to reread partition table
- Drivers that support removeable media will need to call this when there is a disk change. See ide.c driver for more details.

The request function

- See do_xd_request() (xd.c:263)
- Basic framework for a polling driver
  - But not really polling!
  - xd_command() (xd.c:575) calls sleep_on(&xd_wait_int)
- More complicated approaches possible
  - See drivers/block/ide.c
  - See drivers/scsi/sd.c