Writing Device Drivers Under Linux

TTY Device Drivers

TTY Device Drivers

- A subclass of character devices
- The Linux tty layer provides extra functionality
  - POSIX tty termios handling
  - PPP/SLIP pseudo-network devices

TTY Layer organization

- See page 22 in the example sources
- Three major classes of the TTY objects
  - TTY high-level processing layer
  - TTY line discipline
  - TTY low-level driver

TTY high-level processing layer

- Manages tty device structures
- Associates a tty device with:
  - A TTY low-level driver
  - A TTY line discipline

TTY line discipline

- Provides processing of the raw incoming TTY data
- Examples of line disciplines
  - POSIX termios (N_TTY)
  - PPP/SLIP
- Interfaces with the linux filesystem
  - File_operations and inode_operations

TTY low-level driver

- Communicates directly with the hardware
- Receives incoming tty data and routes it to the line discipline
- The line discipline sends data to the TTY driver for transmission
- What you need to implement to support a new hardware device

The Comtrol Rocketport

- Manufactured by Comtrol, Inc.
- Supports 4 to 32 ports per board
- No interrupts
- Large (1024 byte) buffers

The Rocketport driver

- Page 30 in the example sources (linux.c)
- Distributed as a module
  - Advantages
    - Easier to update to new versions
    - Kernel independent
  - Disadvantage
    - Hard to make it kernel version independent

Initialization

- See init_module() (linux.c:1751)
- Initialization steps
  - Detect hardware
  - Reserve kernel resources (timer, irq, ioports)
  - Register tty driver structure

Detect hardware

- Be careful probing for hardware!!
- Don't assume the device is at an I/O address until you've checked carefully
- Try not to hang the system when you check
- In a module, be especially careful

Guides for I/O probing

- Always check to see if another driver has reserved the I/O ports first
- Reading ports is safer than writing
- If you must to a write to a port, restore its contents afterwards
- Do safer tests before more invasive tests

The tty driver structure

- See init_module() (linux.c:1825)
- This structure is the major interface between the tty code and driver
  - Control parameters (major number, minor number ranges)
  - Function pointers (open, close, write, etc.)

The open function

- See rp_open() (linux.c:781)
- Basic operations
  - Allocate data structures (lines 788--834)
  - Initialize hardware (lines 840-864)
  - Do callout/dialin arbitration (block_til_ready)
  - Initialize termios and configure port

Allocating data structures

- Be careful of race conditions!
- What if two processes try to open the port at the same time?
- Memory allocation functions can block
- After calling functions which can block, you must recheck the state of the world

Callout/dialin arbitration

- See block_til_ready() (linux.c:654)
- Functions
  - Provide mutual exclusion between callout and dialin devices
  - In some cases, block the open from completing until carrier detect is high
- As an implementation strategy, save this for last

January 7, 1997 TTY Device Drivers 15

Configuring the port

- See configure_r_port (linux.c:582)
- This function initializes the serial port parameters as specified by the tty termios structure
  - Called by rp_open() when the port is opened
  - Called by rp_set_termios() (linux.c:1010) when the termios structure is changed.

January 7, 1997 TTY Device Drivers 16

The close function

- See rp_close() (linux.c:888)
- Basic operations
  - Maintain in-use counts
  - Flush xmit buffers
  - Free resources

The polling routine

- See rp_do_poll (linux.c:428)
- Heart of the Rocketport driver

Overview of rp_do_poll()

- Quickly check to see if port needs to be serviced
- If necessary, process the port (rp_handle_port(), linux.c:369)
  - Check carrier detect
  - If the are characters to be read, call rp_do_receive() (linux.c:216)
- Load more data into the ports' transmit buffers (rp_do_transmit(), linux.c:311)

Character reception

- See rp_do_receive() (linux.c:216)
- Basic operations
  - Determine how many characters to read in
  - Read them into the flip buffer
    - Character buffer
    - Flags buffer for parity, frame errors
  - Mark the flip buffer for flipping

TTY Flip Buffers

- For efficiency reasons, it is better to batch characters to the line discipline
  - But not too much or latency becomes a problem
- Flip buffers are provided by the high-level tty code to solve this problem
  - Optional --- tty drivers don’t have to use it
  - Allows drivers to very quickly stuff characters into the flip buffer to minimize interrupt processing time

TTY Flip Buffers II

- Each tty has a pair of flip buffers
- The driver fills one of the flip buffers in its receive_characters interrupt routine
- Periodically, the flip buffers are atomically swapped, and the filled flip buffer is processed by the line discipline

Character transmission

- Two calling methods into a tty driver
  - rp_write (linux.c:1438)
  - rp_put_char (linux.c:1405)
    - ...and flush_chars() (tty-abstractions:279), if present
    - Kernel will emulate put_char() if necessary
- Related routines
  - rp_write_room (linux.c:1542)
  - rp_chars_in_buffer (linux.c:1563)

Handling ioctls

- See rp_ioctl() (linux.c:1167)
- TTY ioctls
  - Sending break characters
  - Setting/reading modem control lines
  - Device specific configuration

Hangup Processing

- Hangup initiation
  - By the driver (rp_handle_port(), linux.c:411)
  - By the tty layer under various circumstances
    - pty closure
    - vhangup
- tty_hangup() calls rp_hangup() (linux.c:1352)

Hangup Processing II

- One of the more subtle parts of the tty layer
- The hangup routine must:
  - Shutdown the device
  - Reset the usage counts to zero
- In rp_close() (linux.c:903), the driver must ignore "hung-up" fd's
- Beware of race conditions!

Implementation Suggestions

- Implement as a kernel module first
  - Faster compile/debug cycle
- Implement the driver incrementally
  - Easier to debug
  - Psychologically more satisfying

Implementation Order

- Driver initialization and cleanup
- Port open and close
- Transmission path
- Receive path

Implementation Order II

- UART configuration
- Miscellaneous ioctls
- Dialin/callout arbitration
- Hangup processing

January 7, 1997 TTY Device Drivers 29