

tty driver 似乎可以分为3层，从上到下依次为

- ① tty layer
- ② line discipline
- ③ tty low-level driver

tty low-level driver运行是异步的，而tty layer运行是同步的。

vt and serial 属于tty low-level driver

tty_io.c, tty_port.c, tty_ioctl.c, tty_ldisc, etc属于tty layer

n_tty.c is line discipline

①和②之间的主要interface是

```

1. struct tty_ldisc_ops {
2.     .....
3.
4.     /*
5.      * The following routines are called from above.
6.      */
7.     int      (*open)(struct tty_struct *);
8.     void      (*close)(struct tty_struct *);
9.     void      (*flush_buffer)(struct tty_struct *tty);
10.    ssize_t    (*chars_in_buffer)(struct tty_struct *tty);
11.    ssize_t    (*read)(struct tty_struct *tty, struct file *file,
12.                      unsigned char __user *buf, size_t nr);
13.    ssize_t    (*write)(struct tty_struct *tty, struct file *file,
14.                       const unsigned char *buf, size_t nr);
15.    int        (*ioctl)(struct tty_struct *tty, struct file *file,
16.                        unsigned int cmd, unsigned long arg);
17.    long       (*compat_ioctl)(struct tty_struct *tty, struct file *file,
18.                               unsigned int cmd, unsigned long arg);
19.    void        (*set_termios)(struct tty_struct *tty, struct ktermios *old);
20.    unsigned int (*poll)(struct tty_struct *, struct file *,
21.                        struct poll_table_struct *);
22.    int         (*hangup)(struct tty_struct *tty);
23.
24.     .....
25.
26. }

```

The comments:

```

1.  /*
2.  * int (*open)(struct tty_struct *);
3.  *
4.  * This function is called when the line discipline is associated
5.  * with the tty. The line discipline can use this as an
6.  * opportunity to initialize any state needed by the ldisc routines.
7.  *
8.  * void (*close)(struct tty_struct *);
9.  *
10. * This function is called when the line discipline is being
11. * shutdown, either because the tty is being closed or because
12. * the tty is being changed to use a new line discipline
13. *
14. * void (*flush_buffer)(struct tty_struct *tty);
15. *
16. * This function instructs the line discipline to clear its
17. * buffers of any input characters it may have queued to be
18. * delivered to the user mode process.
19. *
20. * ssize_t (*chars_in_buffer)(struct tty_struct *tty);
21. *
22. * This function returns the number of input characters the line
23. * discipline may have queued up to be delivered to the user mode
24. * process.
25. *
26. * ssize_t (*read)(struct tty_struct * tty, struct file * file,
27. *                 unsigned char * buf, size_t nr);
28. *
29. * This function is called when the user requests to read from
30. * the tty. The line discipline will return whatever characters
31. * it has buffered up for the user. If this function is not
32. * defined, the user will receive an EIO error.
33. *
34. * ssize_t (*write)(struct tty_struct * tty, struct file * file,
35. *                  const unsigned char * buf, size_t nr);
36. *
37. * This function is called when the user requests to write to the
38. * tty. The line discipline will deliver the characters to the
39. * low-level tty device for transmission, optionally performing
40. * some processing on the characters first. If this function is
41. * not defined, the user will receive an EIO error.
42. *
43. * int (*ioctl)(struct tty_struct * tty, struct file * file,
44. *              unsigned int cmd, unsigned long arg);
45. *
46. * This function is called when the user requests an ioctl which
47. * is not handled by the tty layer or the low-level tty driver.
48. * It is intended for ioctls which affect line discipline
49. * operation. Note that the search order for ioctls is (1) tty
50. * layer, (2) tty low-level driver, (3) line discipline. So a
51. * low-level driver can "grab" an ioctl request before the line
52. * discipline has a chance to see it.
53. *

```

```

54. * long (*compat_ioctl)(struct tty_struct * tty, struct file * file,
55. *                          unsigned int cmd, unsigned long arg);
56. *
57. *      Process ioctl calls from 32-bit process on 64-bit system
58. *
59. * void (*set_termios)(struct tty_struct *tty, struct ktermios * old);
60. *
61. *      This function notifies the line discipline that a change has
62. *      been made to the termios structure.
63. *
64. * int (*poll)(struct tty_struct * tty, struct file * file,
65. *              poll_table *wait);
66. *
67. *      This function is called when a user attempts to select/poll on a
68. *      tty device. It is solely the responsibility of the line
69. *      discipline to handle poll requests.
70. *
71. * int (*hangup)(struct tty_struct *)
72. *
73. *      Called on a hangup. Tells the discipline that it should
74. *      cease I/O to the tty driver. Can sleep. The driver should
75. *      seek to perform this action quickly but should wait until
76. *      any pending driver I/O is completed.
77. */

```

②和③之间的主要interface是

```

1. struct tty_ldisc_ops {
2.
3.     .....
4.
5.     /*
6.      * The following routines are called from below.
7.      */
8.     void (*receive_buf)(struct tty_struct *, const unsigned char *cp,
9.                          char *fp, int count);
10.    void (*write_wakeup)(struct tty_struct *);
11.    void (*dcd_change)(struct tty_struct *, unsigned int);
12.    void (*fasync)(struct tty_struct *tty, int on);
13.    int (*receive_buf2)(struct tty_struct *, const unsigned char *cp,
14.                        char *fp, int count);
15.
16.    .....
17.
18. };

```

```

1.  /*
2.  * void (*receive_buf)(struct tty_struct *, const unsigned char *cp,
3.  *                      char *fp, int count);
4.  *
5.  *      This function is called by the low-level tty driver to send
6.  *      characters received by the hardware to the line discipline for
7.  *      processing. <cp> is a pointer to the buffer of input
8.  *      character received by the device. <fp> is a pointer to a
9.  *      pointer of flag bytes which indicate whether a character was
10. *      received with a parity error, etc. <fp> may be NULL to indicate
11. *      all data received is TTY_NORMAL.
12. *
13. * void (*write_wakeup)(struct tty_struct *);
14. *
15. *      This function is called by the low-level tty driver to signal
16. *      that line discipline should try to send more characters to the
17. *      low-level driver for transmission. If the line discipline does
18. *      not have any more data to send, it can just return. If the line
19. *      discipline does have some data to send, please arise a tasklet
20. *      or workqueue to do the real data transfer. Do not send data in
21. *      this hook, it may leads to a deadlock.
22. *
23. *
24. * void (*fasync)(struct tty_struct *, int on)
25. *
26. *      Notify line discipline when signal-driven I/O is enabled or
27. *      disabled.
28. *
29. * void (*dcd_change)(struct tty_struct *tty, unsigned int status)
30. *
31. *      Tells the discipline that the DCD pin has changed its status.
32. *      Used exclusively by the N_PPS (Pulse-Per-Second) line discipline.
33. *
34. * int (*receive_buf2)(struct tty_struct *, const unsigned char *cp,
35. *                     char *fp, int count);
36. *
37. *      This function is called by the low-level tty driver to send
38. *      characters received by the hardware to the line discipline for
39. *      processing. <cp> is a pointer to the buffer of input
40. *      character received by the device. <fp> is a pointer to a
41. *      pointer of flag bytes which indicate whether a character was
42. *      received with a parity error, etc. <fp> may be NULL to indicate
43. *      all data received is TTY_NORMAL.
44. *      If assigned, prefer this function for automatic flow control.
45. */

```

Documentation/serial/tty.txt是很有价值的文章。