in drivers/tty/n_tty.c

```
struct tty_ldisc_ops tty_ldisc_N_TTY = {
        .magic           = TTY_LDISC_MAGIC,
        .name            = "n_tty",
        .open            = n_tty_open,
        .close           = n_tty_close,
        .flush_buffer    = n_tty_flush_buffer,
        .chars_in_buffer = n_tty_chars_in_buffer,
        .read            = n_tty_read,
        .write           = n_tty_write,
        .ioctl           = n_tty_ioctl,
        .set_termios     = n_tty_set_termios,
        .poll            = n_tty_poll,
        .receive_buf     = n_tty_receive_buf,
        .write_wakeup    = n_tty_write_wakeup,
        .fasync          = n_tty_fasync,
        .receive_buf2    = n_tty_receive_buf2,
};


static void n_tty_receive_buf(struct tty_struct *tty, const unsigned char *cp,
                              char *fp, int count)
{
        n_tty_receive_buf_common(tty, cp, fp, count, 0);
}

static int n_tty_receive_buf2(struct tty_struct *tty, const unsigned char *cp,
                              char *fp, int count)
{
        return n_tty_receive_buf_common(tty, cp, fp, count, 1);
}

static int
n_tty_receive_buf_common(struct tty_struct *tty, const unsigned char *cp,
                         char *fp, int count, int flow)
{
        struct n_tty_data *ldata = tty->disc_data;
        int room, n, rcvd = 0;

        down_read(&tty->termios_rwsem);

        while (1) {
                room = receive_room(tty);       ①
                n = min(count, room);
                if (!n) {
                        if (flow && !room)          ②
                                ldata->no_room = 1;
                        break;
                }
                __receive_buf(tty, cp, fp, n);  ③
                cp += n;
                if (fp)
                        fp += n;
                count -= n;
```

```
54.                     rcvd += n;
55.             }
56.
57.             tty->receive_room = room;
58.             n_tty_check_throttle(tty);          ④
59.             up_read(&tty->termios_rwsem);
60.
61.             return rcvd;
62.     }
```

unsigned char *cp中是data,而char *fp是每个char对应的flag

①

得到n_tty line disciple接收buffer (ldata->read_buf)的空闲空间

②

n = 0，三种可能，一是ldata->read_buf[]还有空闲空间，但已经接收完count的data；二是虽然接收的data还未到count，但ldata->read_buf[]已经满了；三是接收完所有的count的data,并且ldata->read_buf[]也满了。

这里如果是n_tty_receive_buf()，即flow = 0，则直接break处while loop；而对于n_tty_receive_buf2(),

即flow = 1，那么在ldata->read_buf[]已经满了的情况下，ldata->no_room = 1.

对ldata->no_room的access

```
static void n_tty_set_room(struct tty_struct *tty)
{
        struct n_tty_data *ldata = tty->disc_data;

        /* Did this open up the receive buffer? We may need to flip */
        if (unlikely(ldata->no_room) && receive_room(tty)) {
                ldata->no_room = 0;

                WARN_RATELIMIT(tty->port->itty == NULL,
                                "scheduling with invalid itty\n");
                /* see if ldisc has been killed - if so, this means that
                 * even though the ldisc has been halted and ->buf.work
                 * cancelled, ->buf.work is about to be rescheduled
                 */
                WARN_RATELIMIT(test_bit(TTY_LDISC_HALTED, &tty->flags),
                               "scheduling buffer work for halted ldisc\n");
                queue_work(system_unbound_wq, &tty->port->buf.work);
        }
}
```

即在n_tty_receive_buf()接收data的情况下，ldata->no_room总是被初始化为0，没有机会变1，

那么n_tty_set_room()等于是empty function。而在n_tty_receive_buf2()接收data的情况下，

if (unlikely(ldata->no_room) && receive_room(tty))

即在n_tty_receive_buf2()中ldata->read_buf[]已经满，而此时，在invoke n_tty_set_room()

时ldata->read_buf[]又有空闲空间了，那么除了ldata->no_room = 0外，运行->buf.work?

③

根据tty的mode来填充n_tty line disciple的buffer,ldata->read_buf[]

④

这里是一种line disciple与low-level driver间的flow control。如果line disciple接收

buffer空闲空间太小了，则要让low-level driver控制一下传入data的速度。

```
1.  static int receive_room(struct tty_struct *tty)
2.  {
3.          struct n_tty_data *ldata = tty->disc_data;
4.          int left;
5.
6.          if (I_PARMRK(tty)) {        ①
7.                  /* Multiply read_cnt by 3, since each byte might take up to
8.                   * three times as many spaces when PARMRK is set (depending on
9.                   * its flags, e.g. parity error). */
10.                 left = N_TTY_BUF_SIZE - read_cnt(ldata) * 3 - 1;
11.         } else                      ②
12.                 left = N_TTY_BUF_SIZE - read_cnt(ldata) - 1;
13.
14.         /*
15.          * If we are doing input canonicalization, and there are no
16.          * pending newlines, let characters through without limit, so
17.          * that erase characters will be handled.  Other excess
18.          * characters will be beeped.
19.          */
20.         if (left <= 0)      ③
21.                 left = ldata->icanon && ldata->canon_head == ldata->read_tail;
22.
23.         return left;
24. }
```

①

if (I_PARMRK(tty))

判断接收的data是否有校验错，这里要多预留空间，没找到详细的资料解释 × 3的原因。

②

返回空闲空间

③

？

```
static void __receive_buf(struct tty_struct *tty, const unsigned char *cp,
                          char *fp, int count)
{
        struct n_tty_data *ldata = tty->disc_data;
        bool preops = I_ISTRIP(tty) || (I_IUCLC(tty) && L_IEXTEN(tty));

        if (ldata->real_raw)
                n_tty_receive_buf_real_raw(tty, cp, fp, count);
        else if (ldata->raw || (L_EXTPROC(tty) && !preops))
                n_tty_receive_buf_raw(tty, cp, fp, count);
        else if (tty->closing && !L_EXTPROC(tty))
                n_tty_receive_buf_closing(tty, cp, fp, count);
        else {
                if (ldata->lnext) {
                        char flag = TTY_NORMAL;

                        if (fp)
                                flag = *fp++;
                        n_tty_receive_char_lnext(tty, *cp++, flag);
                        count--;
                }

                if (!preops && !I_PARMRK(tty))
                        n_tty_receive_buf_fast(tty, cp, fp, count);
                else
                        n_tty_receive_buf_standard(tty, cp, fp, count);

                flush_echoes(tty);
                if (tty->ops->flush_chars)
                        tty->ops->flush_chars(tty);
        }

        if ((!ldata->icanon && (read_cnt(ldata) >= ldata->minimum_to_wake)) ||
                L_EXTPROC(tty)) {
                kill_fasync(&tty->fasync, SIGIO, POLL_IN);
                if (waitqueue_active(&tty->read_wait))
                        wake_up_interruptible_poll(&tty->read_wait, POLLIN);
        }
}
```

对raw mode的处理比较简单，因为实质上line discipline不需要处理什么，而对canon mode则

由于没有详细处理的资料，所以只能看懂个大概。

```
1.  static void
2.  n_tty_receive_buf_real_raw(struct tty_struct *tty, const unsigned char *cp,
3.                             char *fp, int count)
4.  {
5.          struct n_tty_data *ldata = tty->disc_data;
6.          size_t n, head;
7.
8.          head = ldata->read_head & (N_TTY_BUF_SIZE - 1);         (A)
9.          n = N_TTY_BUF_SIZE - max(read_cnt(ldata), head);
10.         n = min_t(size_t, count, n);
11.         memcpy(read_buf_addr(ldata, head), cp, n);
12.         ldata->read_head += n;
13.         cp += n;
14.         count -= n;
15.
16.         head = ldata->read_head & (N_TTY_BUF_SIZE - 1);         (B)
17.         n = N_TTY_BUF_SIZE - max(read_cnt(ldata), head);
18.         n = min_t(size_t, count, n);
19.         memcpy(read_buf_addr(ldata, head), cp, n);
20.         ldata->read_head += n;
21. }
```

n_tty line discipline对接收到的data不做任何处理，直接放入ldata->read_buf[]。

这里分(A)，（B）两部分copy是源于ldata->read_buf[]实际上是个环形buffer,为了处理如下情况。

| | | |

| free 1| data | free 2 |

(A)先填充free 2 buffer，(B)填充free 1 buffer.

另外由于是raw mode(完全raw),所以这里fp所指向的flags buffer是没意义的。

```
1.  static void
2.  n_tty_receive_buf_raw(struct tty_struct *tty, const unsigned char *cp,
3.                        char *fp, int count)
4.  {
5.          struct n_tty_data *ldata = tty->disc_data;
6.          char flag = TTY_NORMAL;
7.
8.          while (count--) {
9.                  if (fp)
10.                         flag = *fp++;                        (A)
11.                 if (likely(flag == TTY_NORMAL))          (B)
12.                         put_tty_queue(*cp++, ldata);         (C)
13.                 else
14.                         n_tty_receive_char_flagged(tty, *cp++, flag);    (D)
15.         }
16. }
```

这里的raw不太彻底，还是要处理一些特殊字符，所以要判断字符的flag。

(A)

获得当前处理character的flag。TTY_NORMAL表示不是特殊处理的字符。

(B)

normal character

(C)

static inline void put_tty_queue(unsigned char c, struct n_tty_data *ldata)

{

    *read_buf_addr(ldata, ldata->read_head) = c;

    ldata->read_head++;

}

指示放入line discipline buffer即可

(D)

不是normal character，则要处理一下

```
static void
n_tty_receive_char_flagged(struct tty_struct *tty, unsigned char c, char flag)
{
        char buf[64];

        switch (flag) {
        case TTY_BREAK:
                n_tty_receive_break(tty);
                break;
        case TTY_PARITY:
        case TTY_FRAME:
                n_tty_receive_parity_error(tty, c);
                break;
        case TTY_OVERRUN:
                n_tty_receive_overrun(tty);
                break;
        default:
                printk(KERN_ERR "%s: unknown flag %d\n",
                       tty_name(tty, buf), flag);
                break;
        }
}
```

由于资料缺乏，所以只看懂了个大概。