**基于N\_tty的tty驱动**

# 1. 驱动注册

1) alloc\_tty\_driver（device numbers）分配一个tty\_driver数据结构；

2)对tty\_driver数据结构进行配置，主要包括设置init\_termios, flag和 major minor\_start;

TTY\_DRIVER\_DYNAMIC\_DEV表明该驱动在sysfs中的设备通过后面tty\_register\_device动态生产，如果不设置该flag，则在注册driver时创建所有设备；

3)tty\_register\_driver注册驱动；该函数会申请一个char设备的region，其中driver->ttys 中为对应设备个数的tty\_struct指针，driver->termios为对应设备个数的struct ktermios指针；

4)tty\_register\_device(smd\_tty\_driver, idx, 0);注册特定index的设备；

数据机构功能：

tty\_driver：用于描述一类tty驱动；

tty\_operations：用于描述该类tty驱动的所支持的控制接口，及该驱动类型的特性；

tty\_struct：用于描述包含在此类tty驱动中的一个设备；

ktermios ：用于描述包含在此类tty驱动中的一个设备的IO配置；

# **2.驱动接口**

对于用户层，tty设备是以一个char设备的形式来表述的，主要实现在tty\_io.c中，char设备接口中调用相应的N\_tty和tty\_operation接口；

## **2.1 tty\_open**



通过用户传入的major和minor来查找对应的tty驱动，并根据minor计算打开设备在tty驱动中的index，创建对应的tty\_struct结构来描述该tty设备。需要注意在tty\_driver中ttys[idx]中会保留已经创建的tty\_struct，termios[index]会保留已创建的termios结构，如果对应的tty设备已经存在tty\_struct结构，则在open中直接使用，从而实现快速open，此时将保留上次用户使用该设备的所有状态，例如buffer数据和termio配置等；

## 2.2 tty\_write



Write数据流控：通过分析可以看出，当底层驱动因为某些因素无法接受数据时，可在tty驱动的tty->ops->write接口调用时返回0，以阻塞用户写数据，等到相关因素变化而允许接收用户数据时再通过tty\_wakeup接口来唤醒前面被阻塞的用户写调用；

另外一边用户会对tty设备采用select的调用方式来进行数据读写，而ld->ops->poll中会把相关的读写分别阻塞在tty->read\_wait和tty->write\_wait这两个等待队列上，对于写使能的条件是

if ([tty](http://lxr.free-electrons.com/ident?i=tty)->[ops](http://lxr.free-electrons.com/ident?i=ops)->[write](http://lxr.free-electrons.com/ident?i=write) && //tty驱动定义了write接口

![tty\_is\_writelocked](http://lxr.free-electrons.com/ident?i=tty_is_writelocked)([tty](http://lxr.free-electrons.com/ident?i=tty)) && //write未lock，即没有写操作正在进行

[tty\_chars\_in\_buffer](http://lxr.free-electrons.com/ident?i=tty_chars_in_buffer)([tty](http://lxr.free-electrons.com/ident?i=tty)) < [WAKEUP\_CHARS](http://lxr.free-electrons.com/ident?i=WAKEUP_CHARS) && //tty驱动的缓冲数据小于256字节

[tty\_write\_room](http://lxr.free-electrons.com/ident?i=tty_write_room)([tty](http://lxr.free-electrons.com/ident?i=tty)) > 0)//tty驱动的缓冲区空闲大于0

[mask](http://lxr.free-electrons.com/ident?i=mask) |= [POLLOUT](http://lxr.free-electrons.com/ident?i=POLLOUT) | [POLLWRNORM](http://lxr.free-electrons.com/ident?i=POLLWRNORM);

故用户只要对于定义好tty驱动中的tty->ops->chars\_in\_buffer和tty->ops->write\_room这两个接口，当缓冲区状态从不可接收数据变为可以接收时调用tty\_wakeup以触发挂起在tty->write\_wait上的select调用进行一次poll检查，通知用户可以写；

一般如果tty设备无特别要求，不定义tty->ops->chars\_in\_buffer，则[tty\_chars\_in\_buffer](http://lxr.free-electrons.com/ident?i=tty_chars_in_buffer)则就直接返回0，而tty->ops->write\_room都会定义；

需要流控用户写到tty设备的数据，可以通过：

1. tty->ops->write返回0，阻塞用户写操作，可写时调用tty\_wakeup解除阻塞；
2. tty->ops->write\_room返回0，使用户select时写操作不被允许，可写时调用tty\_wakeup；
3. tty->ops->chars\_in\_buffer返回大于[WAKEUP\_CHARS](http://lxr.free-electrons.com/ident?i=WAKEUP_CHARS)（256），使用户select时写操作不被允许，可写时调用tty\_wakeup；

## 2.3 tty\_read



Tty设备中读到的数据有两层缓存

1. 规程中的数据缓存，采用数据结构tty->ttybuf\_head来描述的存储池，该数据结构中维护着2个tty\_buffer结构的链表，head和tail表示当前存有数据的链表，free表示空闲链表；该缓存的最大空间是65535；该缓存主要用来存储tty设备驱动上传上来的数据，当tty驱动从物理通道上收到数据后，先通过tty\_buffer\_request\_room在这个存储池中找到一个可以存储该大小的对应节点，在通过tty\_filp\_buffer\_push来触发tty->buf.work，该work会把ld缓存池中的数据拷贝到tty设备缓存tty->read\_buf中；

1. [64](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L64) struct [tty\_buffer](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=tty_buffer) {
2. [65](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L65) struct [tty\_buffer](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=tty_buffer) \*[next](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=next);
3. [66](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L66) char \*char\_buf\_ptr;
4. [67](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L67) unsigned char \*flag\_buf\_ptr;
5. [68](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L68) int [used](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=used); //the size of data saved in buffer
6. [69](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L69) int [size](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=size); //the total size of buffer
7. [70](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L70) int [commit](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=commit);//the size of data committed to tty->read\_buf
8. [71](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L71) int [read](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=read);//the size of data already read in tty->read\_buf

1. [72](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L72) ***/\* Data points here \*/***
2. [73](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L73) unsigned long [data](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=data)[0];
3. [74](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L74) };
4. struct [tty\_bufhead](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=tty_bufhead) {
5. [88](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L88) struct [work\_struct](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=work_struct) [work](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=work);
6. [89](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L89) [spinlock\_t](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=spinlock_t) [lock](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=lock);
7. [90](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L90) struct [tty\_buffer](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=tty_buffer) \*[head](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=head); ***/\* Queue head \*/***
8. [91](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L91) struct [tty\_buffer](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=tty_buffer) \*[tail](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=tail); ***/\* Active buffer \*/***
9. [92](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L92) struct [tty\_buffer](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=tty_buffer) \*[free](http://lxr.oss.org.cn/ident?v=3.2.20;a=alpha;i=free); ***/\* Free queue head \*/***
10. [93](http://lxr.oss.org.cn/source/include/linux/tty.h?v=3.2.20;a=alpha" \l "L93) int memory\_used; ***/\* Buffer space used***
11. 94 }

2） tty设备缓存tty->read\_buf ，这是一个环形buffer，最大默认是4096; 用户空间的读操作都是从这个缓存中拷贝到用户提供的存储空间中；



read数据的流控：下图是tty架构设计的从上到下的接收数据的流控点，流控的触发条件tty设备缓存区的变化（tty缓存的大小N\_TTY\_BUF\_SIZE为4096），但因注意以下几点

1. 实际物理设备本身就有FIFO，当FIFO不够时会自动触发硬件流控；
2. 规程的缓存空间为65535远远大于tty设备缓存4096，为了数据传输效率，即使tty设备层触发throttle后，tty驱动还是可以把数据传到规程缓存中，实际情形下，很多设备驱动不会去实现throttle和unthrottle就是这个原因；
3. 如果为了避免对方发送的数据过多的积压在规程缓存中，即tty设备缓存的4096大小足够满足传输要求，则可配合实现tty驱动的throttle和unthrottle接口，在throttle中禁止对端再发送数据，在unthrottle接口中恢复对端的数据发送；
4. 当throttle被触发时，表明用户的读取速度远远小于底层tty驱动接收数据的速度；



## 2.4 tty\_poll

tty\_poll中直接调用ld->ops->poll，在N\_tty的poll实现上：

1. read的poll操作会挂在tty->read\_wait这个等待队列上，当tty设备缓存中的数据量满足用户对tty设置的Vmin要求时就设置POLLIN | POLLRDNORMAL标志；
2. 当tty设备处于TTY\_OTHER\_CLOSED或tty\_hang\_up\_p时会设置POLLUP标志，以通知用户该设备已经读到文件尾；对于用户的select调用，设置该标志后会进程会被通知为可读，但当用户去执行读动作时，ld->ops->read会对应返回 –EIO或0；
3. tty驱动接口中的tty->ops->chars\_in\_buffer小于256且tty->ops->write\_room大于0时，设置POLLOUT | POLLWRNORM，当缓冲区状态从不可接收数据变为可以接收时调用底层驱动调用tty\_wakeup以触发挂起在tty->write\_wait上的select调用进行一次poll检查，通知用户可以写；

## 2.5 tty\_release

一个tty设备tty\_stuct中可能被多个用户打开使用，每次tty\_release会对tty->count减1，当count减到0时，就会调用tty\_ldisc\_release和release\_tty来销毁对应的规程和tty\_sturct数据结构，销毁后所有的tty设备缓存和规程缓存都会被free，tty\_driverz中的tty[index]会赋值为NULL，而其中的 termios[index]只在设置了TTY\_DRIVER\_RESET\_TERMIOS时才会被释放否则下次用户打开时会保持前面使用的termios设置；

另外对应uart这种标准串口设备(使用serial\_core)，在其tty->ops->close中如果发现tty\_chars\_in\_buffer的值不为0则会等待30秒，以便tty驱动把数据发缓存中的数据发出去，等待时间长短可通过ioctl的TIOCSSERIAL的closing\_wait来进行配置；

