/dev/ttyprintk是虚拟的tty device。

in drivers/char/Kconfig

```
config TTY_PRINTK
        tristate "TTY driver to output user messages via printk"
        depends on EXPERT && TTY
        default n
        ---help---
          If you say Y here, the support for writing user messages (i.e.
          console messages) via printk is available.

          The feature is useful to inline user messages with kernel
          messages.
          In order to use this feature, you should output user messages
          to /dev/ttyprintk or redirect console to this TTY.

          If unsure, say N.
```

in drivers/char/Makefile

```
obj-$(CONFIG_TTY_PRINTK)        += ttyprintk.o
```

---

in drivers/char/ttyprintk.c

```
static int __init ttyprintk_init(void)
{
        int ret = -ENOMEM;

    ......

        ttyprintk_driver->driver_name = "ttyprintk";
        ttyprintk_driver->name = "ttyprintk";
        ttyprintk_driver->major = TTYAUX_MAJOR;
        ttyprintk_driver->minor_start = 3;
        ttyprintk_driver->type = TTY_DRIVER_TYPE_CONSOLE;
        ttyprintk_driver->init_termios = tty_std_termios;
        ttyprintk_driver->init_termios.c_oflag = OPOST | OCRNL | ONOCR | ONLRET;
        tty_set_operations(ttyprintk_driver, &ttyprintk_ops);①
        tty_port_link_device(&tpk_port.port, ttyprintk_driver, 0);

        ret = tty_register_driver(ttyprintk_driver);                ②
        if (ret < 0) {
                printk(KERN_ERR "Couldn't register ttyprintk driver\n");
                goto error;
        }

        return 0;

    ......
}
```

①

```
static const struct tty_operations ttyprintk_ops = {
        .open = tpk_open,
        .close = tpk_close,
        .write = tpk_write,
        .write_room = tpk_write_room,
        .ioctl = tpk_ioctl,
};
```

②

```
int tty_register_driver(struct tty_driver *driver)
{
        int error;
        int i;
        dev_t dev;
        struct device *d;

        if (!driver->major) {
                error = alloc_chrdev_region(&dev, driver->minor_start,
                                                driver->num, driver->name);
                if (!error) {
                        driver->major = MAJOR(dev);
                        driver->minor_start = MINOR(dev);
                }
        } else {            ③
                dev = MKDEV(driver->major, driver->minor_start);
                error = register_chrdev_region(dev, driver->num, driver->name);
        }
        if (error < 0)
                goto err;

        if (driver->flags & TTY_DRIVER_DYNAMIC_ALLOC) {
                error = tty_cdev_add(driver, dev, 0, driver->num);
                if (error)
                        goto err_unreg_char;
        }

        mutex_lock(&tty_mutex);
        list_add(&driver->tty_drivers, &tty_drivers);
        mutex_unlock(&tty_mutex);

        if (!(driver->flags & TTY_DRIVER_DYNAMIC_DEV)) {        ④
                for (i = 0; i < driver->num; i++) {
                        d = tty_register_device(driver, i, NULL);       ⑤
                        if (IS_ERR(d)) {
                                error = PTR_ERR(d);
                                goto err_unreg_devs;
                        }
                }
        }

    ......
}
```

③

ttyprintk走该分支

④

```
        ttyprintk_driver = tty_alloc_driver(1,
                        TTY_DRIVER_RESET_TERMIOS |
                        TTY_DRIVER_REAL_RAW |
                        TTY_DRIVER_UNNUMBERED_NODE);
```

所以ttyprintk走该分支

1 --- 只有一个/dev/ttyprintk virtual device

⑤

```
struct device *tty_register_device(struct tty_driver *driver, unsigned index,
                                   struct device *device)
{
        return tty_register_device_attr(driver, index, device, NULL, NULL);
}
```

tty_register_device_attr()

|

|

\|/

tty_cdev_add()

```
static int tty_cdev_add(struct tty_driver *driver, dev_t dev,
                unsigned int index, unsigned int count)
{
        /* init here, since reused cdevs cause crashes */
        cdev_init(&driver->cdevs[index], &tty_fops);
        driver->cdevs[index].owner = driver->owner;
        return cdev_add(&driver->cdevs[index], dev, count);
}
```

即ttyprintk的file_operations是tty_fops.

in drivers/tty/tty_io.c

```
static const struct file_operations tty_fops= {
        .llseek         = no_llseek,
        .read           = tty_read,
        .write          = tty_write,
        .poll           = tty_poll,
        .unlocked_ioctl = tty_ioctl,
        .compat_ioctl   = tty_compat_ioctl,
        .open           = tty_open,
        .release        = tty_release,
        .fasync         = tty_fasync,
};
```

这里的file_operations中的callback是怎样route到ttyprintk_ops中的callback的？

---

analyse accessing of "/dev/ttyprintk" device

Open "/dev/ttyprintk" device

tty_open()

    |

|

\|/

tpk_open()

read "/dev/ttyprintk" device

由于该virtual device是write-only，所以实际上没有read处理。但tty_read()会怎么样呢？

write "/dev/ttyprintk" device

tty_write()

|

|

\|/

n_tty_write()

|

|

\|/

tpk_write()

ioctl "/dev/ttyprintk" device

tty_ioctl()

|

|

\|/

tpk_ioctl()

/dev/printk的用处

用户可以往kernel log中写message.

echo "walter message\n" > /dev/ttyprintk

则该message会出现在kernel message中

[U]walter message\n

在ubuntu 12.04的3.8.0-27 kernel上测试该feature，发行system crash.可能是kernel的bug吧。

walterzh@walterzh-Precision-T1650:~$ uname -a

Linux walterzh-Precision-T1650 3.8.0-27-generic #40~precise3-Ubuntu SMP Fri Jul 19 14:38:30 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux

cat test

walterzh

cat test > /dev/ttyprintk