

/dev/ttyprintk是虚拟的tty device。

in drivers/char/Kconfig

```
1. config TTY_PRINTK
2.     tristate "TTY driver to output user messages via printk"
3.     depends on EXPERT && TTY
4.     default n
5.     ---help---
6.         If you say Y here, the support for writing user messages (i.e.
7.         console messages) via printk is available.
8.
9.         The feature is useful to inline user messages with kernel
10.        messages.
11.        In order to use this feature, you should output user messages
12.        to /dev/ttyprintk or redirect console to this TTY.
13.
14.        If unsure, say N.
```

in drivers/char/Makefile

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

in drivers/char/ttyprintk.c

```

1. static int __init ttyprintk_init(void)
2.
3. {
4.
5.     .....
6.
7.     ttyprintk_driver->driver_name = "ttyprintk";
8.     ttyprintk_driver->name = "ttyprintk";
9.     ttyprintk_driver->major = TTYAUX_MAJOR;
10.    ttyprintk_driver->minor_start = 3;
11.    ttyprintk_driver->type = TTY_DRIVER_TYPE_CONSOLE;
12.    ttyprintk_driver->init_termios = tty_std_termios;
13.    ttyprintk_driver->init_termios.c_oflag = OPOST | OCRNL | ONOCR | ONLRET;
14.    tty_set_operations(ttyprintk_driver, &ttyprintk_ops);①
15.    tty_port_link_device(&tpk_port.port, ttyprintk_driver, 0);
16.
17.    ret = tty_register_driver(ttyprintk_driver);           ②
18.    if (ret < 0) {
19.        printk(KERN_ERR "Couldn't register ttyprintk driver\n");
20.        goto error;
21.    }
22.
23.    return 0;
24.
25.    .....
26. }

```

①

```

1. static const struct tty_operations ttyprintk_ops = {
2.     .open = tpk_open,
3.     .close = tpk_close,
4.     .write = tpk_write,
5.     .write_room = tpk_write_room,
6.     .ioctl = tpk_ioctl,
7. };

```

②

```

1. int tty_register_driver(struct tty_driver *driver)
2. {
3.     int error;
4.     int i;
5.     dev_t dev;
6.     struct device *d;
7.
8.     if (!driver->major) {
9.         error = alloc_chrdev_region(&dev, driver->minor_start,
10.                                     driver->num, driver->name);
11.        if (!error) {
12.            driver->major = MAJOR(dev);
13.            driver->minor_start = MINOR(dev);
14.        }
15.    } else {          ③
16.        dev = MKDEV(driver->major, driver->minor_start);
17.        error = register_chrdev_region(dev, driver->num, driver->name);
18.    }
19.    if (error < 0)
20.        goto err;
21.
22.    if (driver->flags & TTY_DRIVER_DYNAMIC_ALLOC) {
23.        error = tty_cdev_add(driver, dev, 0, driver->num);
24.        if (error)
25.            goto err_unreg_char;
26.    }
27.
28.    mutex_lock(&tty_mutex);
29.    list_add(&driver->tty_drivers, &tty_drivers);
30.    mutex_unlock(&tty_mutex);
31.
32.    if (!(driver->flags & TTY_DRIVER_DYNAMIC_DEV)) {      ④
33.        for (i = 0; i < driver->num; i++) {                  ⑤
34.            d = tty_register_device(driver, i, NULL);
35.            if (IS_ERR(d)) {
36.                error = PTR_ERR(d);
37.                goto err_unreg_devs;
38.            }
39.        }
40.    }
41.
42.    .....
43. }

```

③

ttyprintk走该分支

④

```
1.     ttyprintk_driver = tty_alloc_driver(1,
2.                                         TTY_DRIVER_RESET_TERMIOS |
3.                                         TTY_DRIVER_REAL_RAW |
4.                                         TTY_DRIVER_UNNUMBERED_NODE);
```

所以ttyprintk走该分支

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

⑤

```
1. struct device *tty_register_device(struct tty_driver *driver, unsigned index,
2.                                     struct device *device)
3. {
4.     return tty_register_device_attr(driver, index, device, NULL, NULL);
5. }
```

tty_register_device_attr()

|

|

\|

tty_cdev_add()

```
1. static int tty_cdev_add(struct tty_driver *driver, dev_t dev,
2.                         unsigned int index, unsigned int count)
3. {
4.     /* init here, since reused cdevs cause crashes */
5.     cdev_init(&driver->cdevs[index], &tty_fops);
6.     driver->cdevs[index].owner = driver->owner;
7.     return cdev_add(&driver->cdevs[index], dev, count);
8. }
```

即ttyprintk的file_operations是tty_fops.

in drivers/tty/tty_io.c

```
1. static const struct file_operations tty_fops= {
2.     .llseek      = no_llseek,
3.     .read        = tty_read,
4.     .write       = tty_write,
5.     .poll        = tty_poll,
6.     .unlocked_ioctl = tty_ioctl,
7.     .compat_ioctl = tty_compat_ioctl,
8.     .open         = tty_open,
9.     .release      = tty_release,
10.    .fasync       = tty_fasync,
11. };
```

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/ttysize
```

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/ttprintk
```

```
* Exporting directories for NFS kernel daemon...
* Starting NFS kernel daemon
Pyro name server daemon: disabled, see /etc/default/pyro-nsd
speech-dispatcher disabled; edit /etc/default/speech-dispatcher
* Starting CUPS printing spooler/server

* Starting VirtualBox kernel modules
aned disabled; edit /etc/default/saned
                                         * Checking battery state...

[ 232.577256] BUG: unable to handle kernel NULL pointer dereference at 000000000000000020
[ 232.577279] IP: [ffffffffff81429e28] tty_port_open+0x88/0xe0
[ 232.577291] PGD 3cff2a067 PUD 3baddd067 PMD 0
[ 232.577300] Oops: 0000 [#1] SMP
[ 232.577306] Modules linked in: nls_iso8859_1(F) usb_storage(F) pci_stub(F) vboxpci(OF) vboxnetadp
tng_probe_statedump(OF) lttng_probe_signal(OF) lttng_probe_sched(OF) lttng_probe_kvm(OF) lttng_probe
lient_mmap_overwrite(OF) lttng_ring_buffer_client_mmap_discard(OF) lttng_ring_buffer_metadata_client(
fsd(F) nfs_acl(F) lttng_kretprobes(OF) lttng_ring_buffer(OF) auth_rpcgss(F) lttng_kprobes(OF) lt
tdi_sio(F) snd_usb_audio(F) snd_usbmidi_lib(F) usbserial(F) snd_hda_codec_realtek(F) snd_hda_intel(F
ic(F) snd_rawmidi(F) snd_seq_midi_event(F) snd_seq(F) snd_timer(F) snd_seq_device(F) snd(F) soundcore
) ahci(F) libahci(F) e1000e(F)
[ 232.577513] CPU 3
[ 232.577519] Pid: 3761, comm: bash Tainted: GF          0 3.8.0-27-generic #40~precise3-Ubuntu Dell
[ 232.577530] RIP: 0010:[<ffffffffff81429e28>] [ffffffffff81429e28] tty_port_open+0x88/0xe0
[ 232.577541] RSP: 0018:ffff8803bada1b48 EFLAGS: 00010246
[ 232.577546] RAX: 0000000000000000 RBX: ffffffffff81f1d720 RCX: 0000000000000000
[ 232.577553] RDX: ffff8803bada1fd8 RSI: 0000000000000282 RDI: ffffffffff81f1d7f0
[ 232.577560] RBP: ffff8803bada1b78 R08: 0000000000000000 R09: 0000000000000001
[ 232.577566] R10: 0000000000000000 R11: 0000000000000000 R12: ffffffffff81f1d7f0
[ 232.577573] R13: ffff8803bac85800 R14: ffff8803dad89b00 R15: ffffffffff81f06078
[ 232.577580] FS: 00007fd871887700(0000) GS:ffff88041dd80000\(0000\) knlGS:0000000000000000
[ 232.577587] CS: 0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 232.577593] CR2: 0000000000000020 CR3: 00000009bad29000 CR4: 00000000001407e0
[ 232.577600] DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
[ 232.577607] DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400
[ 232.577614] Process bash (pid: 3761, threadinfo ffff8803bada0000, task ffff8803dae5dd00)
[ 232.577620] Stack:
[ 232.577624] ffff8803bada1b58 ffffffffff81f1f38fe ffff8803bac85800 ffff8803dad89b00
[ 232.577635] 0000000000500003 ffff8803dae5dd00 ffff8803bada1b88 ffffffffff814502c5
[ 232.577645] ffff8803bada1c08 ffffffffff81421982 ffff8803bada1c08 ffffffffff8146906b
[ 232.577655] Call Trace:
[ 232.577664] [<ffffffffff81f1f38fe>] ? _raw_spin_lock+0xe/0x20
[ 232.577673] [<ffffffffff814502c5>] tpk_open+0x25/0x30
[ 232.577682] [<ffffffffff81421982>] tty_open+0x172/0x420
[ 232.577691] [<ffffffffff8146906b>] ? kobj_lookup+0x10b/0x170
[ 232.577699] [<ffffffffff811a0472>] chrdev_open+0xb2/0x1a0
[ 232.577707] [<ffffffffff811a03c0>] ? cdev_put+0x30/0x30
[ 232.577715] [<ffffffffff81199a6e>] do_dentry_open+0x21e/0x2a0
[ 232.577723] [<ffffffffff81199ee9>] vfs_open+0x49/0x50
[ 232.577731] [<ffffffffff811a8356>] do_last+0x246/0x820
[ 232.577739] [<ffffffffff811aa883>] path_openat+0xb3/0x4d0
[ 232.577748] [<ffffffffff811ab6a2>] do_filp_open+0x42/0xa0
[ 232.577755] [<ffffffffff811b9325>] ? __alloc_fd+0xe5/0x170
[ 232.577763] [<ffffffffff8119b2ca>] do_sys_open+0xfa/0x250
[ 232.577771] [<ffffffffff8119b441>] sys_open+0x21/0x30
[ 232.577779] [<ffffffffff816fc75d>] system_call_fastpath+0x1a/0x1f
[ 232.577785] Code: 48 89 df e8 0b ff ff 4c 89 e7 e8 d3 76 2c 00 48 8b 83 b8 00 00 00 a9 00 00 00 80
5 33
[ 232.578589] RIP [ffffffffff81429e28] tty_port_open+0x88/0xe0
[ 232.579477] RSP <ffff8803bada1b48>
[ 232.580372] CR2: 0000000000000000
```

