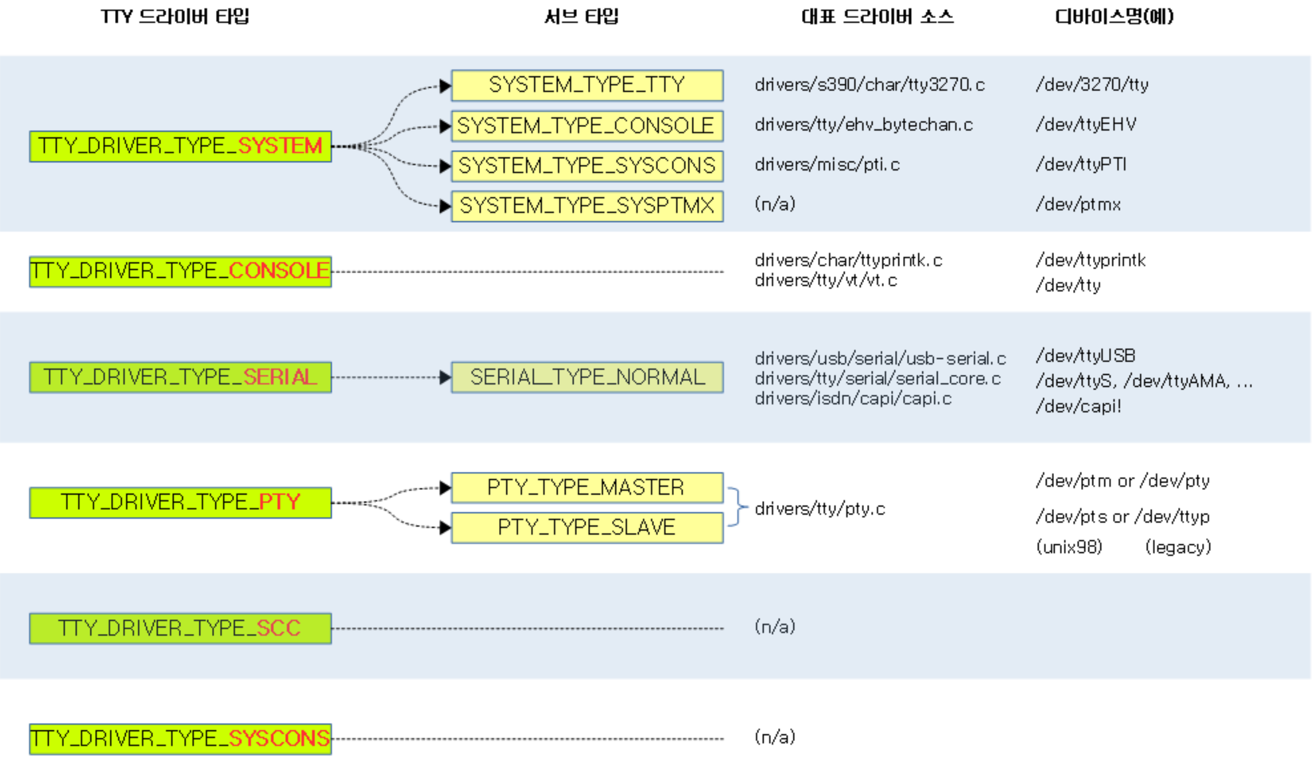
# Console & TTY Driver

2017-12-20 (http://jake.dothome.co.kr/tty/) Moon Young-il (http://jake.dothome.co.kr/author/admin/)
Linux Kernel (http://jake.dothome.co.kr/category/linux/)

## TTY Drivers

There are six types of TTY drivers defined in Linux.

- console
- serial (subtype: normal)
- PTY (subtypes: master, slave)
- system (subtypes: tty, console, syscons, sysptmx)
- scc (not used)
- syscons (not used)



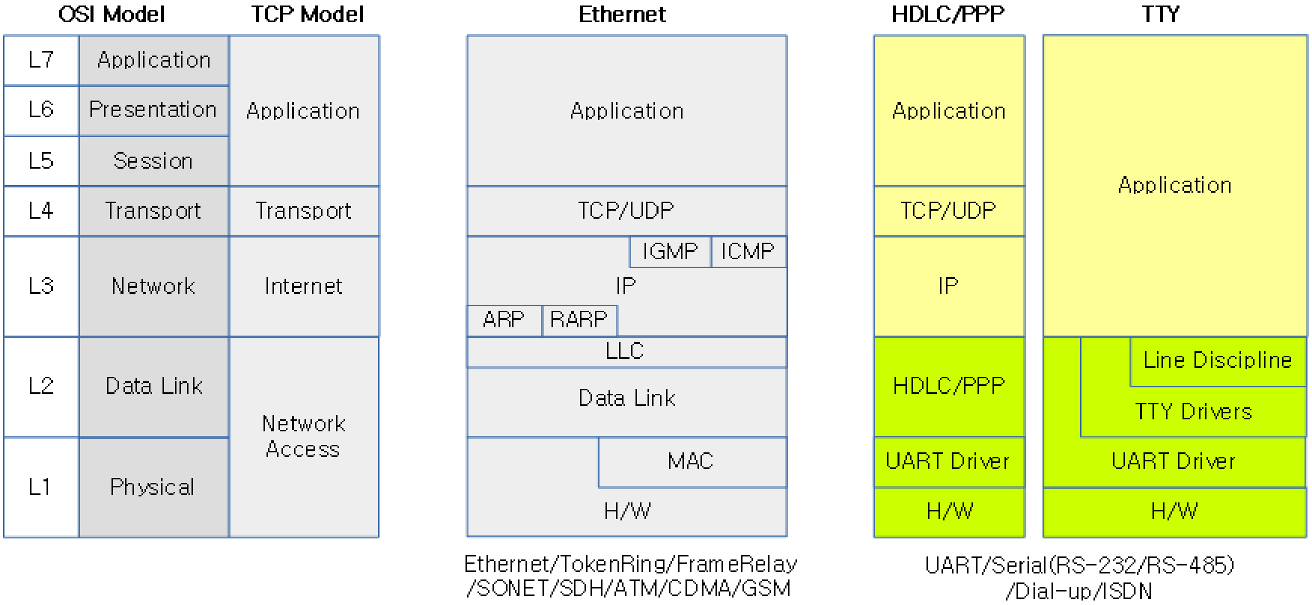
| TTY 드라이버 타입 | 서브 타입 | 대표 드라이버 소스 | 디바이스명(예) |
|---|---|---|---|
| TTY_DRIVER_TYPE_SYSTEM | SYSTEM_TYPE_TTY | drivers/s390/char/tty3270.c | /dev/3270/tty |
| | SYSTEM_TYPE_CONSOLE | drivers/tty/ehv_bytechan.c | /dev/ttyEHV |
| | SYSTEM_TYPE_SYSCONS | drivers/misc/pti.c | /dev/ttyPTI |
| | SYSTEM_TYPE_SYSPTMX | (n/a) | /dev/ptmx |
| TTY_DRIVER_TYPE_CONSOLE | | drivers/char/ttyprintk.c<br>drivers/tty/vt/vt.c | /dev/ttyprintk<br>/dev/tty |
| TTY_DRIVER_TYPE_SERIAL | SERIAL_TYPE_NORMAL | drivers/usb/serial/usb-serial.c<br>drivers/tty/serial/serial_core.c<br>drivers/isdn/capi/capi.c | /dev/ttyUSB<br>/dev/ttyS, /dev/ttyAMA, ...<br>/dev/capi! |
| TTY_DRIVER_TYPE_PTY | PTY_TYPE_MASTER | drivers/tty/pty.c | /dev/ptm or /dev/pty |
| | PTY_TYPE_SLAVE | | /dev/pts or /dev/ttyp<br>(unix98) (legacy) |
| TTY_DRIVER_TYPE_SCC | | (n/a) | |
| TTY_DRIVER_TYPE_SYSCONS | | (n/a) | |

(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-2.png)

# TTY(Teletypewriter)

As shown in the following figure, compare the 7 layers of the OSI model with the TCP model to see which layers correspond to TTY.

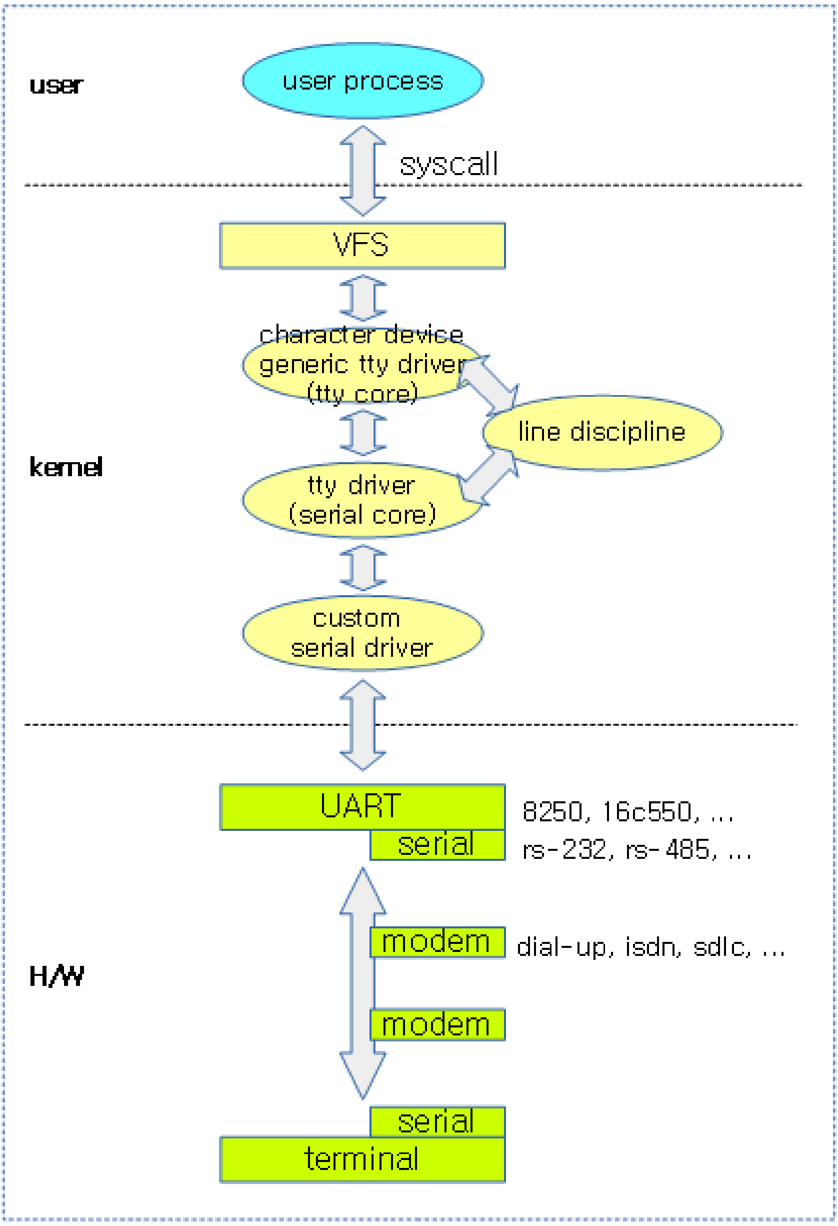
- The Phsical Layer of TTY connects UART, Serial, Dial-up modem, and ISDN modem hardware devices and locates a driver to control them.
- The location of the Data Link Layer includes some of the above hardware drivers, the common TTY driver, and the Line discipline driver.



(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-1b.png)

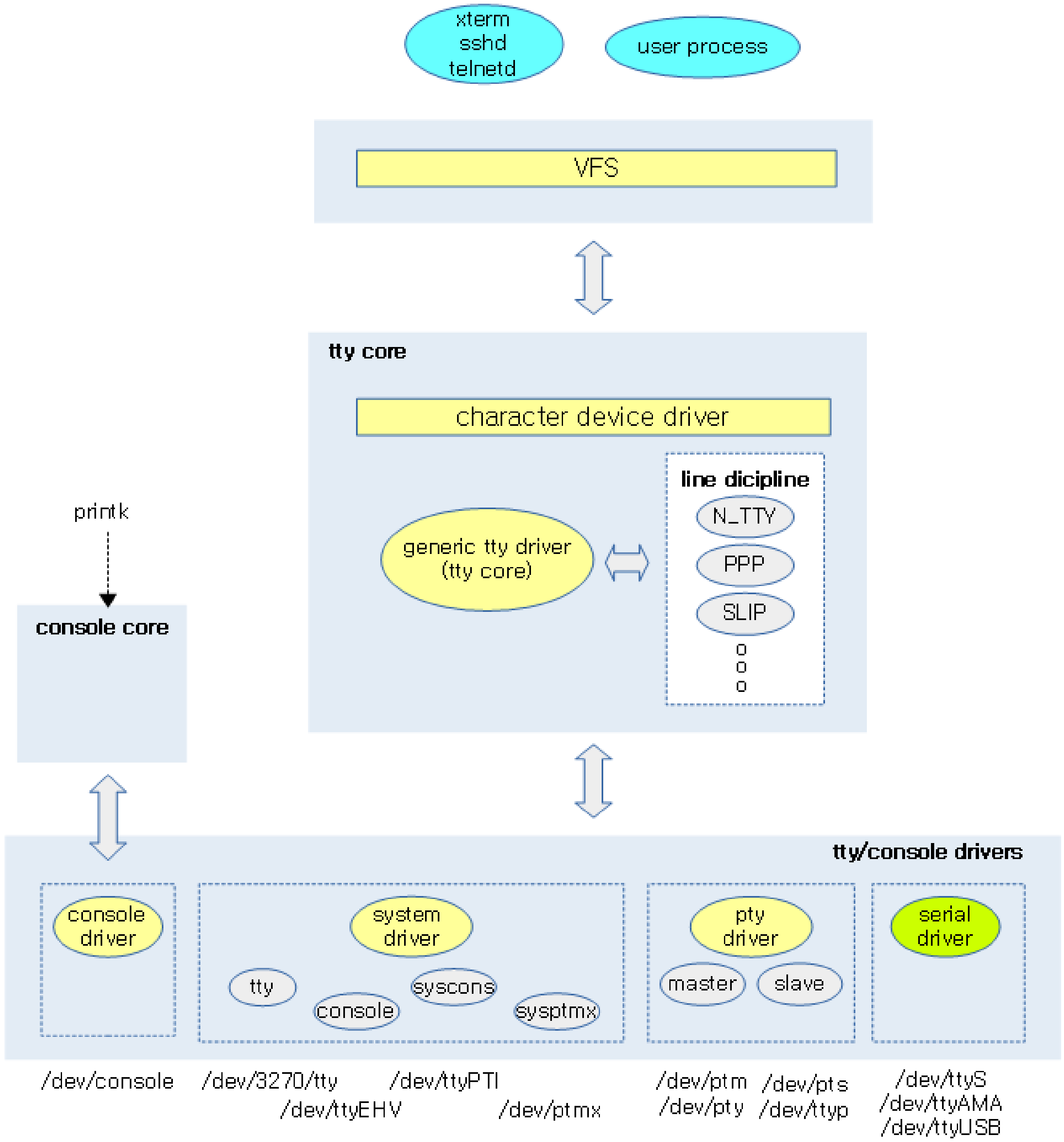
# TTY Drivers

The generic TTY driver is registered as a character device. This generic tty driver can be directly connected to the tty driver associated with the hardware (or pseudo terminal), or it can be connected to a specific line discipline driver in between. There are many different configurations of tty drivers. If you select a serial type from the drivers created by selecting the tty driver type mentioned above, you can describe the layer with the following configuration.



(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-3.png)

The figure below is more comprehensive to show not only the tty of the serial type above, but also the whole.

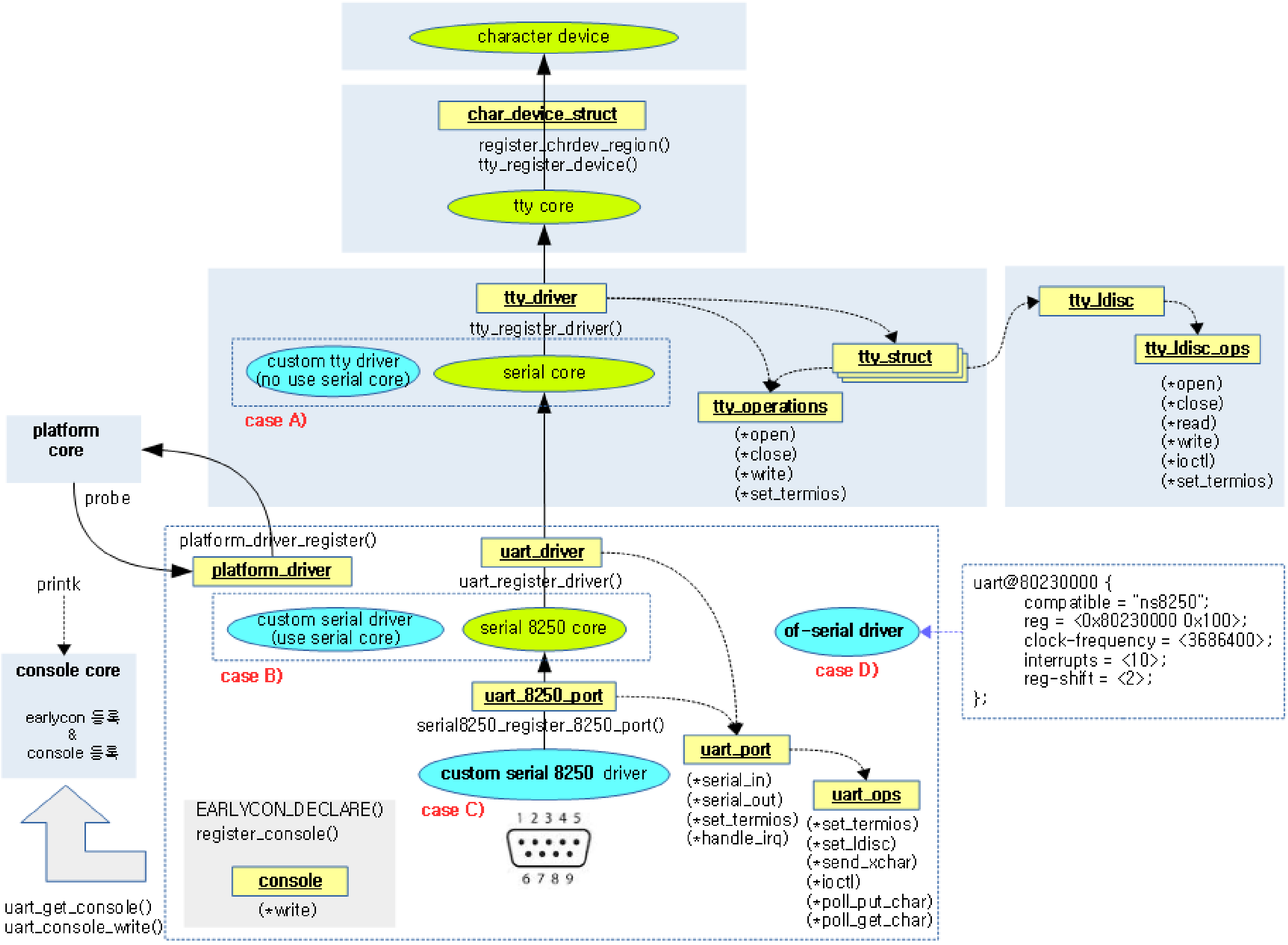


(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-4.png)

Let's take a look at a little more about the tty driver for supporting serial types in the recent kernel. As shown in the figure below, I divided the serial driver types into cases A) ~ D).

- Prior to kernel 2.6, only case A) was supported to develop serial-type tty drivers, and the tty_register_driver() function was used to register tty drivers.
- Serial core (generic) was added in kernel 2.6, and case B) form was also supported to make it easier to develop tty drivers for serials. Use the uart_register_driver() function to register the serial driver.
- In addition, for serial drivers for the 8250/16C550, a serial 8250 core (generic) has been added to the lower layer. Case C form was also supported to develop serial drivers using the 8250 (16C550). Use the serial8250_register_8250_port() function to register the serial driver for the 8250.

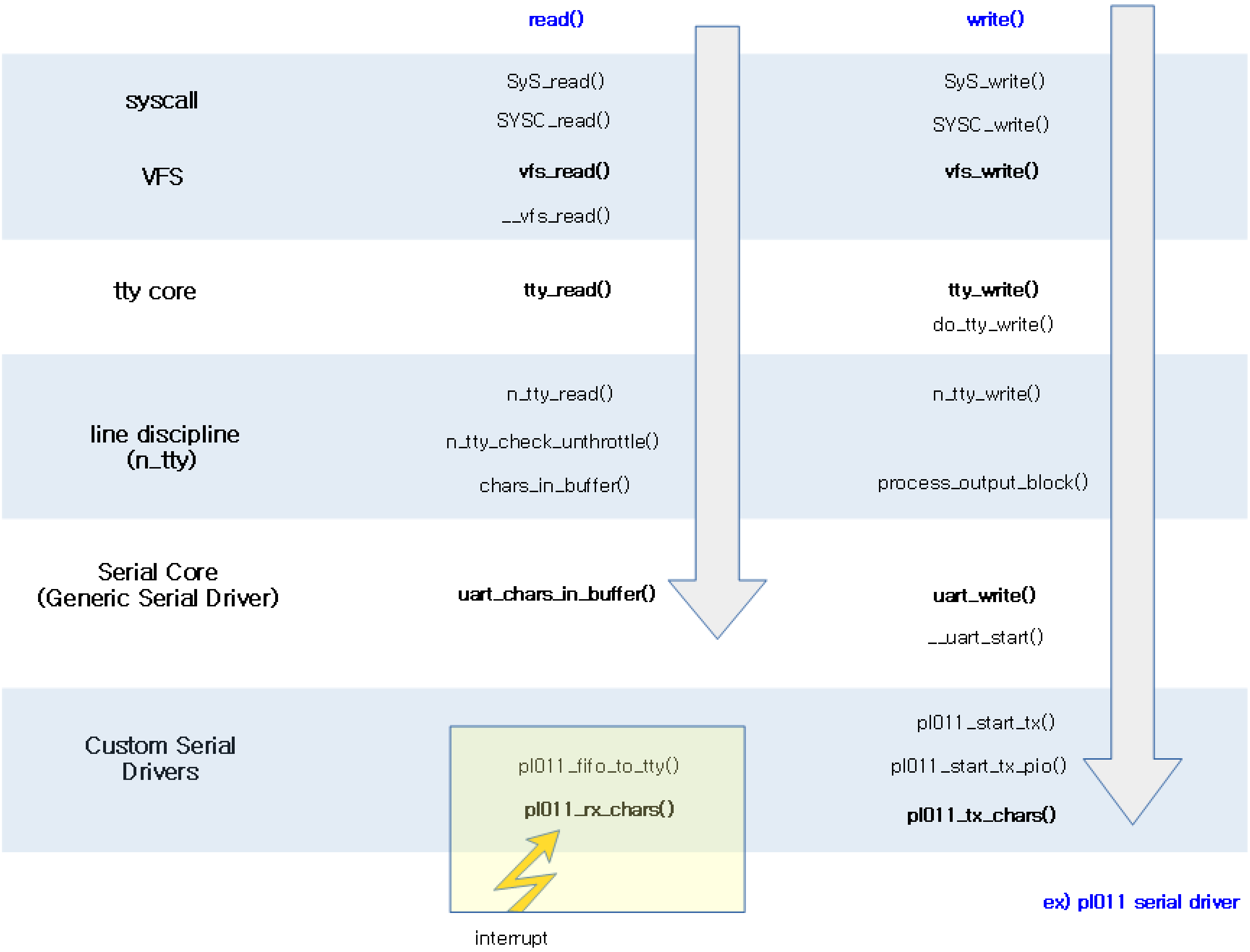
- case D), which can also be used using an of-serial driver for a general-purpose device tree.



```
uart@80230000 {
        compatible = "ns8250";
        reg = <0x80230000 0x100>;
        clock-frequency = <3686400>;
        interrupts = <10>;
        reg-shift = <2>;
};
```

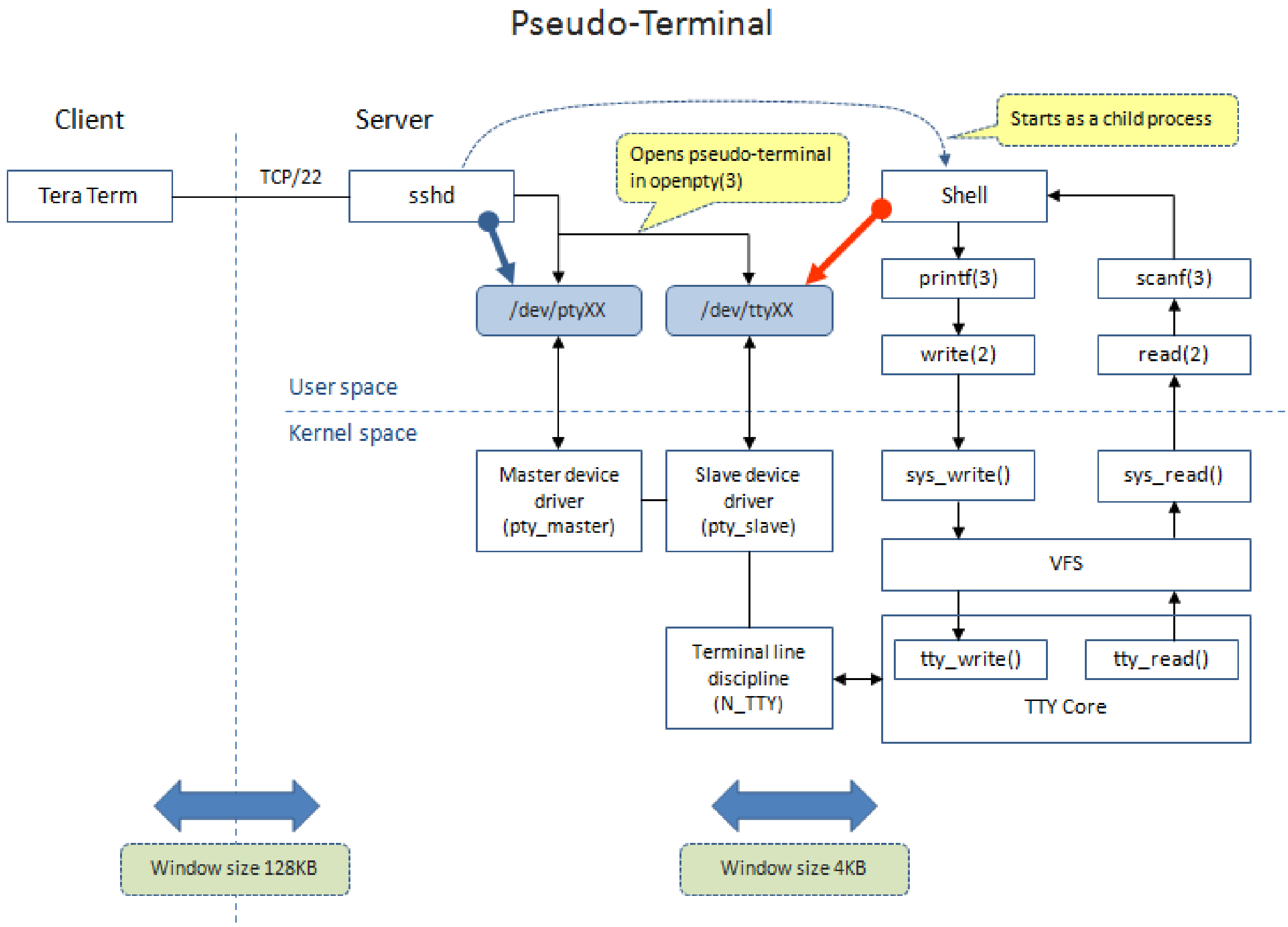
(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-5b.png)

The following shows how the actual functions are called when serial I/O occurs. (Based on RPI2)



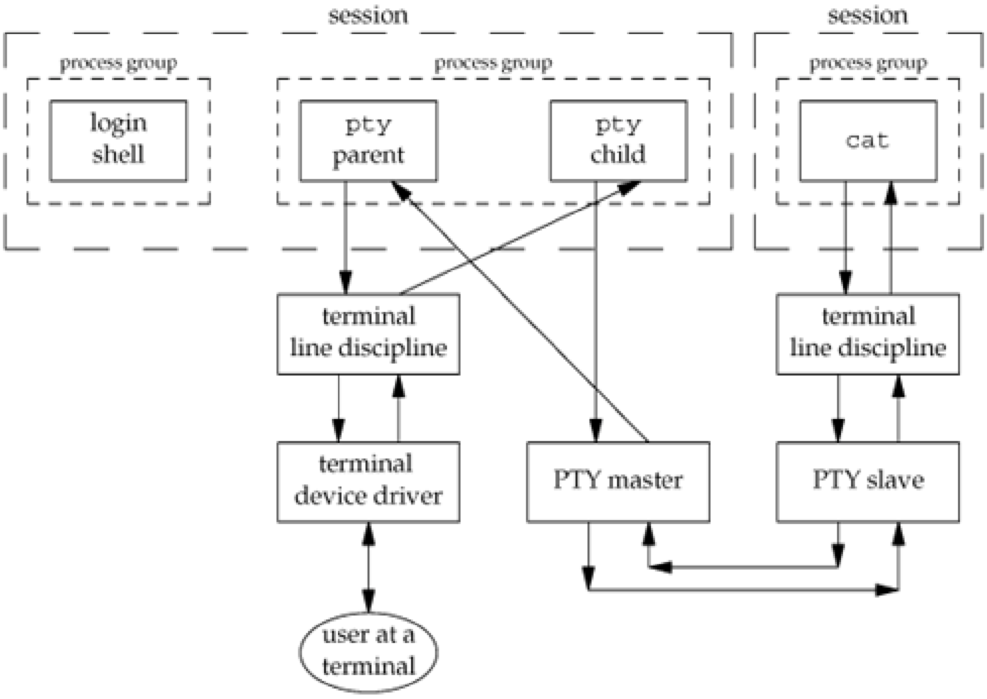
(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-6.png)

Take a look at the diagram detailing how the Pseudo terminal works.



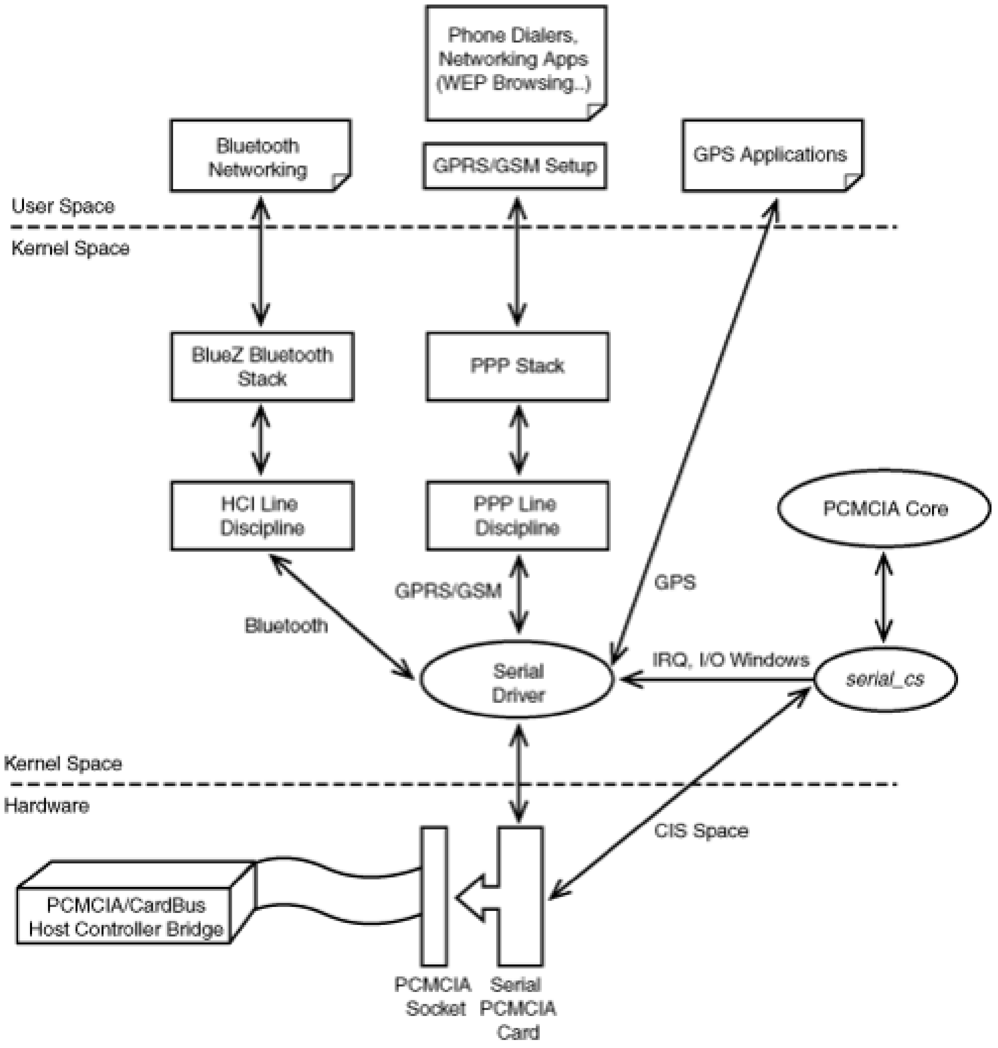
(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-7.png)

This is an illustration of the above configuration changed to a session and process group perspective.



(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-8.png)

Other device configurations are also examined.



(http://jake.dothome.co.kr/wp-content/uploads/2017/12/tty-9.png)

# TTY Core (Generic)

## TTY Driver Registration

### tty_register_driver()

drivers/tty/tty_io.c

```
/*
 * Called by a tty driver to register itself.
 */
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);
                }
18         } else {
19                 dev = MKDEV(driver->major, driver->minor_start);
20                 error = register_chrdev_region(dev, driver->num, driver->name);
21         }
22         if (error < 0)
23                 goto err;
24 
25         if (driver->flags & TTY_DRIVER_DYNAMIC_ALLOC) {
26                 error = tty_cdev_add(driver, dev, 0, driver->num);
27                 if (error)
28                         goto err_unreg_char;
29         }
30 
31         mutex_lock(&tty_mutex);
32         list_add(&driver->tty_drivers, &tty_drivers);
33         mutex_unlock(&tty_mutex);
34 
35         if (!(driver->flags & TTY_DRIVER_DYNAMIC_DEV)) {
36                 for (i = 0; i < driver->num; i++) {
37                         d = tty_register_device(driver, i, NULL);
38                         if (IS_ERR(d)) {
39                                 error = PTR_ERR(d);
40                                 goto err_unreg_devs;
41                         }
42                 }
43         }
44         proc_tty_register_driver(driver);
45         driver->flags |= TTY_DRIVER_INSTALLED;
46         return 0;
```

Register the tty driver that you received as a takeover. If it registers normally and succeeds, it returns 0, and if an error occurs, it returns an error value.

- If the major number is not registered in the TTY driver on code lines 11~23, find the unused number of the major number 254 to 0 of the character device and register the character device with that number.
    - character device numbers are divided into 32 bits, which can be used to create 32-bit values using the MKDEV (major, minor) macro constants.
        - major: The top 12 bits, which can be determined using the MAJOR() macro constant.
        - minor: This is the lower 20 bits, which can be determined using the MINOR() macro constant.
- If you used the TTY_DRIVER_DYNAMIC_ALLOC flag in the tty driver in lines 25~29 of the code, register the cdevs[] of this driver as the character device on the system. (Be careful not to register twice)
- In lines 31~33 of code, add the requested tty driver to the global tty_drivers list.
- If you used the TTY_DRIVER_DYNAMIC_DEV flag in the tty driver in lines 35~43, have the tty device registered as many as driver->num.
- In line 44 of code, add the requested tty driver to the proc interface.
- Add a TTY_DRIVER_INSTALLED flag to recognize the tty driver requested in line 45 as having finished installing.

```
01 err_unreg_devs:
02         for (i--; i >= 0; i--)
03                 tty_unregister_device(driver, i);
04 
```

```
        mutex_lock(&tty_mutex);
        list_del(&driver->tty_drivers);
        mutex_unlock(&tty_mutex);

err_unreg_char:
        unregister_chrdev_region(dev, driver->num);
err:
        return error;
}
EXPORT_SYMBOL(tty_register_driver);
```

If an error occurs, cancel the number of registered devices and remove it from the global tty_drivers list. And finally, remove the character from your system's device.

## Initialize TTY

The function below is called at the end of the chr_dev_init() function, which is initialized by the "mem" character device that can read and write memory, to initialize the TTY.

- fs_initcall(chr_dev_init) -> tty_init()

### tty_init()

drivers/tty/tty_io.c

```
/*
 * Ok, now we can initialize the rest of the tty devices and can count
 * on memory allocations, interrupts etc..
 */
int __init tty_init(void)
{
        cdev_init(&tty_cdev, &tty_fops);
        if (cdev_add(&tty_cdev, MKDEV(TTYAUX_MAJOR, 0), 1) ||
            register_chrdev_region(MKDEV(TTYAUX_MAJOR, 0), 1, "/dev/tty") < 0)
                panic("Couldn't register /dev/tty driver\n");
        device_create(tty_class, NULL, MKDEV(TTYAUX_MAJOR, 0), NULL, "tty");

        cdev_init(&console_cdev, &console_fops);
        if (cdev_add(&console_cdev, MKDEV(TTYAUX_MAJOR, 1), 1) ||
            register_chrdev_region(MKDEV(TTYAUX_MAJOR, 1), 1, "/dev/console") < 0)
                panic("Couldn't register /dev/console driver\n");
        consdev = device_create(tty_class, NULL, MKDEV(TTYAUX_MAJOR, 1), NULL,
                              "console");
        if (IS_ERR(consdev))
                consdev = NULL;
        else
                WARN_ON(device_create_file(consdev, &dev_attr_active) < 0);

#ifdef CONFIG_VT
        vty_init(&console_fops);
#endif
        return 0;
}
```

Register the default tty character device (/dev/tty) and the default console character device (/dev/console ) in the system. If your kernel supports Virtual Terminal, initialize it (using /dev/tty) using the default ops for console.

- In line 7~11 of the code, register the default tty character device (/dev/tty). (major=5, minor=0)
- In lines 13~18 of code, register the default console character device (/dev/console). (major=5, minor=1)
- In line 19~22 of the code, if the console is successfully registered, it creates an active file with read access for all users and groups.
  - /sys/devices/virtual/tty/console directory followed by an active file.
  - The directory above is linked to the directory name /sys/dev/char/<major>:<minor>
- In line 24~26 of the code, if the kernel supports Virtual Terminal, initialize it.
  - Using the basic OPS for console, I create a device 4:0 /dev/tty0.
  - Create a /dev/vcs file 7:0 and a /dev/vcsa file 7:128.
  - Create file 7:1 /dev/vcs1 and 7:129 /dev/vcsa1.
  - Assign console drivers to up to 63 ports and register console-type tty drivers.

Below, an active file for the console has been created with the attribute read only, and you can read that file to find out which tty driver the console is currently configured with.

```
# ls -la /sys/dev/char/5:1/active
-r--r--r-- 1 root root 4096 12월 21 23:13 /sys/dev/char/5:1/active
# cat /sys/dev/char/5:1/active
tty0
```

### tty_driver Structure

include/linux/tty_driver.h

```
struct tty_driver {
        int     magic;          /* magic number for this structure */
        struct kref kref;       /* Reference management */
        struct cdev *cdevs;
        struct module   *owner;
        const char      *driver_name;
        const char      *name;
        int     name_base;      /* offset of printed name */
        int     major;          /* major device number */
        int     minor_start;    /* start of minor device number */
        unsigned int    num;    /* number of devices allocated */
        short   type;           /* type of tty driver */
        short   subtype;        /* subtype of tty driver */
        struct ktermios init_termios; /* Initial termios */
        unsigned long   flags;          /* tty driver flags */
        struct proc_dir_entry *proc_entry; /* /proc fs entry */
        struct tty_driver *other; /* only used for the PTY driver */

        /*
         * Pointer to the tty data structures
         */
        struct tty_struct **ttys;
        struct tty_port **ports;
        struct ktermios **termios;
        void *driver_state;

        /*
         * Driver methods
         */

        const struct tty_operations *ops;
        struct list_head tty_drivers;
};
```

A tty driver has one or more tty_struct, tty_port, ktermios configurations and statuses.

- magic
    - The magic number for the tty driver
    - TTY_DRIVER_MAGIC(0x5402)
- kref
    - Reference Management
- *cdevs
    - Character Device Pointer
- *owner
    - Modules.
- *driver_name
    - Driver Name
- *name
    - Device Name
- name_base
    - The starting number of the name to be printed.
- major
    - Major Number
- minor_start
    - Minor Start Number
- num
    - Number of tty devices
- type
    - TTY driver type (6 types)
        - TTY_DRIVER_TYPE_SYSTEM(1)
        - TTY_DRIVER_TYPE_CONSOLE(2)
        - TTY_DRIVER_TYPE_SERIAL(3)
        - TTY_DRIVER_TYPE_PTY(4)
        - TTY_DRIVER_TYPE_SCC(5)
        - TTY_DRIVER_TYPE_SYSCONS(6)
- subtype
    - tty driver subtype
        - system subtype
            - SYSTEM_TYPE_TTY(1)
            - SYSTEM_TYPE_CONSOLE(2)
            - SYSTEM_TYPE_SYSCONS(3)
            - SYSTEM_TYPE_SYSPTMX(4)
        - pty subtypes

        - PTY_TYPE_MASTER(1)
        - PTY_TYPE_SLAVE(2)
      - serial subtype
        - SERIAL_TYPE_NORMAL(1)
- *init_termios
  - Terminal Init Hook Function
- flags
  - Driver Flags
    - TTY_DRIVER_INSTALLED(0x0001)
    - TTY_DRIVER_RESET_TERMIOS(0x0002)
    - TTY_DRIVER_REAL_RAW(0x0004)
    - TTY_DRIVER_DYNAMIC_DEV(0x0008)
    - TTY_DRIVER_DEVPTS_MEM(0x0010)
    - TTY_DRIVER_HARDWARE_BREAK(0x0020)
    - TTY_DRIVER_DYNAMIC_ALLOC(0x0040)
    - TTY_DRIVER_UNNUMBERED_NODE(0x0080)
- *proc_entry
  - proc interface start directory entry
- *other
  - Used only with tty drivers of type PTY
- **ttys
  - tty_struct refers to an array.
- **ports
  - tty_port refers to an array. (as many tty ports as there are ports)
- **termios
  - It points to a ktermios array for terminal control.
- *driver_state
  - Driver Status
- *ops
  - Operations of the tty driver
- tty_drivers
  - Node links used when added to the global tty_drivers list

### tty_struct & tty_port Structs

tty_struct specifies the line discipline for the tty device and manages to perform flow control to perform the data link role corresponding to the OSI L2 layer. A tty driver can have more than one tty_port, so there is a tty_port struct with content for it.

- Struct code omitted

## TTY Core operations

Here's the OPS for tty and console that tty core uses:

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,
};

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

## TERMINAL CONTROL SETTINGS (TERMIOS)

If the tty driver uses the standard terminal settings, you can use the following settings:

```
struct ktermios tty_std_termios = {     /* for the benefit of tty drivers */
        .c_iflag = ICRNL | IXON,
        .c_oflag = OPOST | ONLCR,
        .c_cflag = B38400 | CS8 | CREAD | HUPCL,
        .c_lflag = ISIG | ICANON | ECHO | ECHOE | ECHOK |
                   ECHOCTL | ECHOKE | IEXTEN,
        .c_cc = INIT_C_CC,
        .c_ispeed = 38400,
        .c_ospeed = 38400
};

EXPORT_SYMBOL(tty_std_termios);
```

The following shows the terminal configuration values set on the tty device device via the stty command.

```
$ sudo stty -F /dev/tty0 -a
speed 38400 baud; rows 25; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>; swtch = <undef>;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd -cmspar cs8 -hupcl -cstopb cread -clocal -crtscts
ignbrk -brkint ignpar -parmrk -inpck -istrip -inlcr -igncr -icrnl -ixon -ixoff -iuclc -ixany -imaxbel -iutf8
-opost -olcuc -ocrnl -onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
-isig -icanon -iexten -echo -echoe -echok -echonl -noflsh -xcase -tostop -echoprt -echoctl -echoke
```

include/uapi/asm-generic/termbits.h

The terminal configuration consists of a set of 4 mode flags, a line discipline, 19 control characters, and an I/O speed setting, as shown below.

```
struct ktermios {
        tcflag_t c_iflag;               /* input mode flags */
        tcflag_t c_oflag;               /* output mode flags */
        tcflag_t c_cflag;               /* control mode flags */
        tcflag_t c_lflag;               /* local mode flags */
        cc_t c_line;                    /* line discipline */
        cc_t c_cc[NCCS];                /* control characters */
        speed_t c_ispeed;               /* input speed */
        speed_t c_ospeed;               /* output speed */
};
```

# console

The drivers that correspond to the keyboard and screen of the system are called consoles. On a PC, the keyboard device and the graphics output via the monitor are used as inputs and outputs of the console. Note that embedded systems do not have keyboards and monitors, so UART (serial) ports are usually used for input/output to replace them. In this case, a UART (serial) device is used as a console.

### console_init()

drivers/tty/tty_io.c

```
/*
 * Initialize the console device. This is called *early*, so
 * we can't necessarily depend on lots of kernel help here.
 * Just do some early initializations, and do the complex setup
 * later.
 */
void __init console_init(void)
{
        initcall_t *call;

        /* Setup the default TTY line discipline. */
        tty_ldisc_begin();

        /*
         * set up the console device so that later boot sequences can
         * inform about problems etc..
         */
        call = __con_initcall_start;
        while (call < __con_initcall_end) {
                (*call)();
                call++;
        }
}
```

Registers the default tty line discipline and calls the setup functions of the console device driver that is configured and registered in the kernel.

- Each system uses different console devices, and there are dozens of types registered in the kernel, some of which are used by embedded devices:
    - con_init() – CONFIG_VT_CONSOLE
    - serial8250_console_init() – CONFIG_SERIAL_8250_CONSOLE
    - bcm63xx_console_init() – CONFIG_SERIAL_BCM63XX_CONSOLE
    - s3c24xx_serial_console_init() – CONFIG_SERIAL_SAMSUNG_CONSOLE

Place the console driver's start function pointer in the ".con_initcall.init" section via the following macro action:

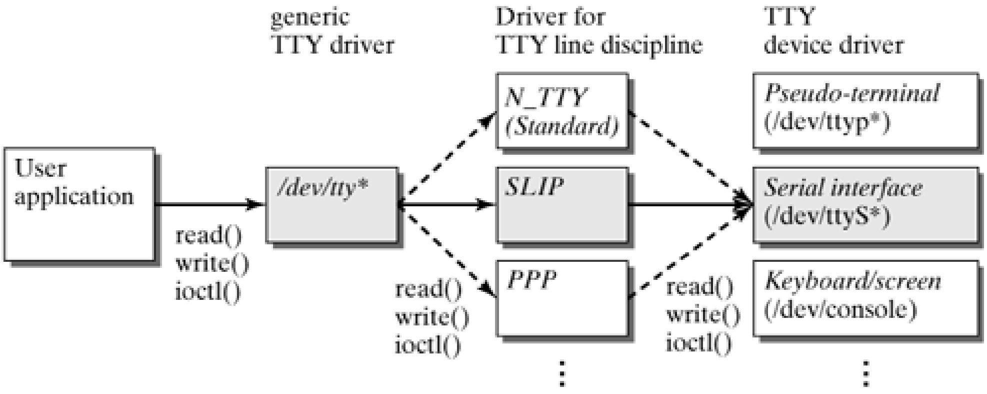
**console_initcall()**

include/linux/init.h

```
#define console_initcall(fn) \
        static initcall_t __initcall_##fn \
        __used __section(.con_initcall.init) = fn
```

# Line Discipline

The Line Discipline driver sits between the top-level character device (generic tty driver) and the tty device driver that is responsible for the hardware or pseudo terminal. Functions include controlling the flow of I/O data between them, or allowing special commands and data changes to be applied. There are many different types of line discipline drivers.



(http://jake.dothome.co.kr/wp-content/uploads/2017/12/ldisc-1.png)

**tty_ldisc_begin()**

drivers/tty/tty_ldisc.c

```
void tty_ldisc_begin(void)
{
        /* Setup the default TTY line discipline. */
        (void) tty_register_ldisc(N_TTY, &tty_ldisc_N_TTY);
}
```

Register the default tty line discipline driver.

### tty_register_ldisc()

drivers/tty/tty_ldisc.c

```
/**
 *      tty_register_ldisc      -       install a line discipline
 *      @disc: ldisc number
 *      @new_ldisc: pointer to the ldisc object
 *
 *      Installs a new line discipline into the kernel. The discipline
 *      is set up as unreferenced and then made available to the kernel
 *      from this point onwards.
 *
 *      Locking:
 *              takes tty_ldiscs_lock to guard against ldisc races
 */

int tty_register_ldisc(int disc, struct tty_ldisc_ops *new_ldisc)
{
        unsigned long flags;
        int ret = 0;

        if (disc < N_TTY || disc >= NR_LDISCS)
                return -EINVAL;

        raw_spin_lock_irqsave(&tty_ldiscs_lock, flags);
        tty_ldiscs[disc] = new_ldisc;
        new_ldisc->num = disc;
        new_ldisc->refcount = 0;
        raw_spin_unlock_irqrestore(&tty_ldiscs_lock, flags);

        return ret;
}
EXPORT_SYMBOL(tty_register_ldisc);
```

Register the LINE DISCIPLINE driver corresponding to the requested disc device.

# TTY line discipline table

There is a pointer array of 30 line discipline ops declared as shown below.

drivers/tty/tty_ldisc.c

```
/* Line disc dispatch table */
static struct tty_ldisc_ops *tty_ldiscs[NR_LDISCS];
```

These are the line disciplines of the device that is registered. Starting with a basic TTY device, it can handle a wide variety of protocols.

include/uapi/linux/tty.h

```
/*
 * 'tty.h' defines some structures used by tty_io.c and some defines.
 */

#define NR_LDISCS               30

/* line disciplines */
```

```
#define N_TTY           0
#define N_SLIP          1
#define N_MOUSE         2
#define N_PPP           3
#define N_STRIP         4
#define N_AX25          5
#define N_X25           6       /* X.25 async */
#define N_6PACK         7
#define N_MASC          8       /* Reserved for Mobitex module <kaz@cafe.net> */
#define N_R3964         9       /* Reserved for Simatic R3964 module */
#define N_PROFIBUS_FDL  10      /* Reserved for Profibus */
#define N_IRDA          11      /* Linux IrDa - http://irda.sourceforge.net/ */
#define N_SMSBLOCK      12      /* SMS block mode - for talking to GSM data */
                                /* cards about SMS messages */
#define N_HDLC          13      /* synchronous HDLC */
#define N_SYNC_PPP      14      /* synchronous PPP */
#define N_HCI           15      /* Bluetooth HCI UART */
#define N_GIGASET_M101  16      /* Siemens Gigaset M101 serial DECT adapter */
#define N_SLCAN         17      /* Serial / USB serial CAN Adaptors */
#define N_PPS           18      /* Pulse per Second */
#define N_V253          19      /* Codec control over voice modem */
#define N_CAIF          20      /* CAIF protocol for talking to modems */
#define N_GSM0710       21      /* GSM 0710 Mux */
#define N_TI_WL         22      /* for TI's WL BT, FM, GPS combo chips */
#define N_TRACESINK     23      /* Trace data routing for MIPI P1149.7 */
#define N_TRACEROUTER   24      /* Trace data routing for MIPI P1149.7 */
```

## Line Discipline Operations

It is an OPS structure that corresponds to the line discipline device.

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,
};
```

# Serial Core (Generic)

## uart_register_driver()

drivers/tty/serial/serial_core.c

```
/**
 *      uart_register_driver - register a driver with the uart core layer
 *      @drv: low level driver structure
 *
 *      Register a uart driver with the core driver.  We in turn register
 *      with the tty layer, and initialise the core driver per-port state.
 *
 *      We have a proc file in /proc/tty/driver which is named after the
 *      normal driver.
 *
 *      drv->port should be NULL, and the per-port structures should be
 *      registered using uart_add_one_port after this call has succeeded.
 */
int uart_register_driver(struct uart_driver *drv)
{
        struct tty_driver *normal;
        int i, retval;

        BUG_ON(drv->state);

        /*
         * Maybe we should be using a slab cache for this, especially if
         * we have a large number of ports to handle.
         */
        drv->state = kzalloc(sizeof(struct uart_state) * drv->nr, GFP_KERNEL);
        if (!drv->state)
                goto out;

        normal = alloc_tty_driver(drv->nr);
        if (!normal)
                goto out_kfree;

        drv->tty_driver = normal;

        normal->driver_name     = drv->driver_name;
        normal->name            = drv->dev_name;
        normal->major           = drv->major;
        normal->minor_start     = drv->minor;
        normal->type            = TTY_DRIVER_TYPE_SERIAL;
        normal->subtype         = SERIAL_TYPE_NORMAL;
        normal->init_termios    = tty_std_termios;
        normal->init_termios.c_cflag = B9600 | CS8 | CREAD | HUPCL | CLOCAL;
        normal->init_termios.c_ispeed = normal->init_termios.c_ospeed = 9600;
        normal->flags           = TTY_DRIVER_REAL_RAW | TTY_DRIVER_DYNAMIC_DEV;
        normal->driver_state    = drv;
        tty_set_operations(normal, &uart_ops);
```

```
        /*
         * Initialise the UART state(s).
         */
        for (i = 0; i < drv->nr; i++) {
                struct uart_state *state = drv->state + i;
                struct tty_port *port = &state->port;

                tty_port_init(port);
                port->ops = &uart_port_ops;
        }

        retval = tty_register_driver(normal);
        if (retval >= 0)
                return retval;

        for (i = 0; i < drv->nr; i++)
                tty_port_destroy(&drv->state[i].port);
        put_tty_driver(normal);
out_kfree:
        kfree(drv->state);
out:
        return -ENOMEM;
}
```

### uart_driver structs

include/linux/serial_core.h

```
struct uart_driver {
        struct module           *owner;
        const char              *driver_name;
        const char              *dev_name;
        int                      major;
        int                      minor;
        int                      nr;
        struct console          *cons;

        /*
         * these are private; the low level driver should not
         * touch these; they should be initialised to NULL
         */
        struct uart_state       *state;
        struct tty_driver       *tty_driver;
};
```

### uart_state & uart_port Structs

A UART driver consists of a state and one or more uart_port.

- Code omitted

## uart Operations

include/linux/serial_core.h

```
/*
 * This structure describes all the operations that can be done on the
 * physical hardware.  See Documentation/serial/driver for details.
 */
struct uart_ops {
        unsigned int    (*tx_empty)(struct uart_port *);
        void            (*set_mctrl)(struct uart_port *, unsigned int mctrl);
        unsigned int    (*get_mctrl)(struct uart_port *);
        void            (*stop_tx)(struct uart_port *);
        void            (*start_tx)(struct uart_port *);
        void            (*throttle)(struct uart_port *);
        void            (*unthrottle)(struct uart_port *);
        void            (*send_xchar)(struct uart_port *, char ch);
        void            (*stop_rx)(struct uart_port *);
        void            (*enable_ms)(struct uart_port *);
        void            (*break_ctl)(struct uart_port *, int ctl);
        int             (*startup)(struct uart_port *);
        void            (*shutdown)(struct uart_port *);
        void            (*flush_buffer)(struct uart_port *);
        void            (*set_termios)(struct uart_port *, struct ktermios *new,
                                       struct ktermios *old);
        void            (*set_ldisc)(struct uart_port *, struct ktermios *);
        void            (*pm)(struct uart_port *, unsigned int state,
                              unsigned int oldstate);

        /*
         * Return a string describing the type of the port
         */
        const char      *(*type)(struct uart_port *);

        /*
         * Release IO and memory resources used by the port.
         * This includes iounmap if necessary.
         */
        void            (*release_port)(struct uart_port *);

        /*
         * Request IO and memory resources used by the port.
         * This includes iomapping the port if necessary.
         */
        int             (*request_port)(struct uart_port *);
        void            (*config_port)(struct uart_port *, int);
        int             (*verify_port)(struct uart_port *, struct serial_struct *);
        int             (*ioctl)(struct uart_port *, unsigned int, unsigned long);
#ifdef CONFIG_CONSOLE_POLL
        int             (*poll_init)(struct uart_port *);
        void            (*poll_put_char)(struct uart_port *, unsigned char);
        int             (*poll_get_char)(struct uart_port *);
#endif
};
```

## Serial 8250 Core (Generic)

It is prepared for quick configuration of drivers using the following UART chips.

- 8250, 16450, 16550, 16550A, 16C950/954
- Cirrus,
- ST16650, ST16650V2, ST16654
- TI16750
- Startech
- XR16850

- RSA
- NS16550A
- XScale
- OCTEON
- AR7
- U6_16550A
- Tegra
- XR17D15X, XR17V35X
- LPC3220
- TruManage
- CIR port
- Altera 16550 FIFO32, Altera 16550 FIFO64, Altera 16550 FIFO128
- 16550A_FSL64

### serial8250_register_8250_port()

drivers/tty/serial/8250/8250_core.c

```
/**
 *      serial8250_register_8250_port - register a serial port
 *      @up: serial port template
 *
 *      Configure the serial port specified by the request. If the
 *      port exists and is in use, it is hung up and unregistered
 *      first.
 *
 *      The port is then probed and if necessary the IRQ is autodetected
 *      If this fails an error is returned.
 *
 *      On success the port is ready to use and the line number is returned.
 */
int serial8250_register_8250_port(struct uart_8250_port *up)
{
        struct uart_8250_port *uart;
        int ret = -ENOSPC;

        if (up->port.uartclk == 0)
                return -EINVAL;

        mutex_lock(&serial_mutex);

        uart = serial8250_find_match_or_unused(&up->port);
        if (uart && uart->port.type != PORT_8250_CIR) {
                if (uart->port.dev)
                        uart_remove_one_port(&serial8250_reg, &uart->port);

                uart->port.iobase       = up->port.iobase;
                uart->port.membase      = up->port.membase;
                uart->port.irq          = up->port.irq;
                uart->port.irqflags     = up->port.irqflags;
                uart->port.uartclk      = up->port.uartclk;
                uart->port.fifosize     = up->port.fifosize;
                uart->port.regshift     = up->port.regshift;
                uart->port.iotype       = up->port.iotype;
                uart->port.flags        = up->port.flags | UPF_BOOT_AUTOCONF;
                uart->bugs              = up->bugs;
                uart->port.mapbase      = up->port.mapbase;
                uart->port.private_data = up->port.private_data;
                uart->port.fifosize     = up->port.fifosize;
                uart->tx_loadsz         = up->tx_loadsz;
                uart->capabilities      = up->capabilities;
                uart->port.throttle     = up->port.throttle;
                uart->port.unthrottle   = up->port.unthrottle;
                uart->port.rs485_config = up->port.rs485_config;
                uart->port.rs485        = up->port.rs485;
```

```
                /* Take tx_loadsz from fifosize if it wasn't set separately */
                if (uart->port.fifosize && !uart->tx_loadsz)
                        uart->tx_loadsz = uart->port.fifosize;

                if (up->port.dev)
                        uart->port.dev = up->port.dev;

                if (up->port.flags & UPF_FIXED_TYPE)
                        serial8250_init_fixed_type_port(uart, up->port.type);

                set_io_from_upio(&uart->port);
                /* Possibly override default I/O functions.  */
                if (up->port.serial_in)
                        uart->port.serial_in = up->port.serial_in;
                if (up->port.serial_out)
                        uart->port.serial_out = up->port.serial_out;
                if (up->port.handle_irq)
                        uart->port.handle_irq = up->port.handle_irq;
                /*  Possibly override set_termios call */
                if (up->port.set_termios)
                        uart->port.set_termios = up->port.set_termios;
                if (up->port.set_mctrl)
                        uart->port.set_mctrl = up->port.set_mctrl;
                if (up->port.startup)
                        uart->port.startup = up->port.startup;
                if (up->port.shutdown)
                        uart->port.shutdown = up->port.shutdown;
                if (up->port.pm)
                        uart->port.pm = up->port.pm;
                if (up->port.handle_break)
                        uart->port.handle_break = up->port.handle_break;
                if (up->dl_read)
                        uart->dl_read = up->dl_read;
                if (up->dl_write)
                        uart->dl_write = up->dl_write;
                if (up->dma) {
                        uart->dma = up->dma;
                        if (!uart->dma->tx_dma)
                                uart->dma->tx_dma = serial8250_tx_dma;
                        if (!uart->dma->rx_dma)
                                uart->dma->rx_dma = serial8250_rx_dma;
                }

                if (serial8250_isa_config != NULL)
                        serial8250_isa_config(0, &uart->port,
                                        &uart->capabilities);

                ret = uart_add_one_port(&serial8250_reg, &uart->port);
                if (ret == 0)
                        ret = uart->port.line;
        }
        mutex_unlock(&serial_mutex);

        return ret;
```

```
}
```

### uart_8250_port Struct

include/linux/serial_8250.h

```
struct uart_8250_port {
        struct uart_port        port;
        struct timer_list       timer;          /* "no irq" timer */
        struct list_head        list;           /* ports on this IRQ */
        unsigned short          capabilities;   /* port capabilities */
        unsigned short          bugs;           /* port bugs */
        bool                    fifo_bug;       /* min RX trigger if enabled */
        unsigned int            tx_loadsz;      /* transmit fifo load size */
        unsigned char           acr;
        unsigned char           fcr;
        unsigned char           ier;
        unsigned char           lcr;
        unsigned char           mcr;
        unsigned char           mcr_mask;       /* mask of user bits */
        unsigned char           mcr_force;      /* mask of forced bits */
        unsigned char           cur_iotype;     /* Running I/O type */
        unsigned int            rpm_tx_active;
        unsigned char           canary;         /* non-zero during system sleep
                                                 *   if no_console_suspend
                                                 */

        /*
         * Some bits in registers are cleared on a read, so they must
         * be saved whenever the register is read but the bits will not
         * be immediately processed.
         */
#define LSR_SAVE_FLAGS UART_LSR_BRK_ERROR_BITS
        unsigned char           lsr_saved_flags;
#define MSR_SAVE_FLAGS UART_MSR_ANY_DELTA
        unsigned char           msr_saved_flags;

        struct uart_8250_dma    *dma;

        /* 8250 specific callbacks */
        int                     (*dl_read)(struct uart_8250_port *);
        void                    (*dl_write)(struct uart_8250_port *, int);
};
```

# consultation

- Line Discipline (http://www.embeddedlinux.org.cn/essentiallinuxdevicedrivers/final/ch06lev1sec4.html)s – www.embeddedlinux.org
- Linux Device Drivers 18. chapter TTY Drivers – Download PDF (https://static.lwn.net/images/pdf/LDD3/ch18.pdf) | Jonathan Corbet, Alessandro Rubini, Greg

Kroah-Hartman

- Linux kernel serial drivers – Download PDF (https://free-electrons.com/doc/legacy/serial-drivers/linux-serial-drivers.pdf) | Free Electrons
- TERMIOS (http://man7.org/linux/man-pages/man3/termios.3.html) | man7.org
- stty (http://www.skrenta.com/rt/man/stty.1.html) | man

### LEAVE A COMMENT

Your email will not be published. Required fields are marked with *

Comments

name *

email *

Website

WRITE A COMMENT

❮ Kthreadd() (http://jake.dothome.co.kr/kthreadd/)

pidmap_init() ❯ (http://jake.dothome.co.kr/pidmap_init/)

Munc Blog (2015 ~ 2024)