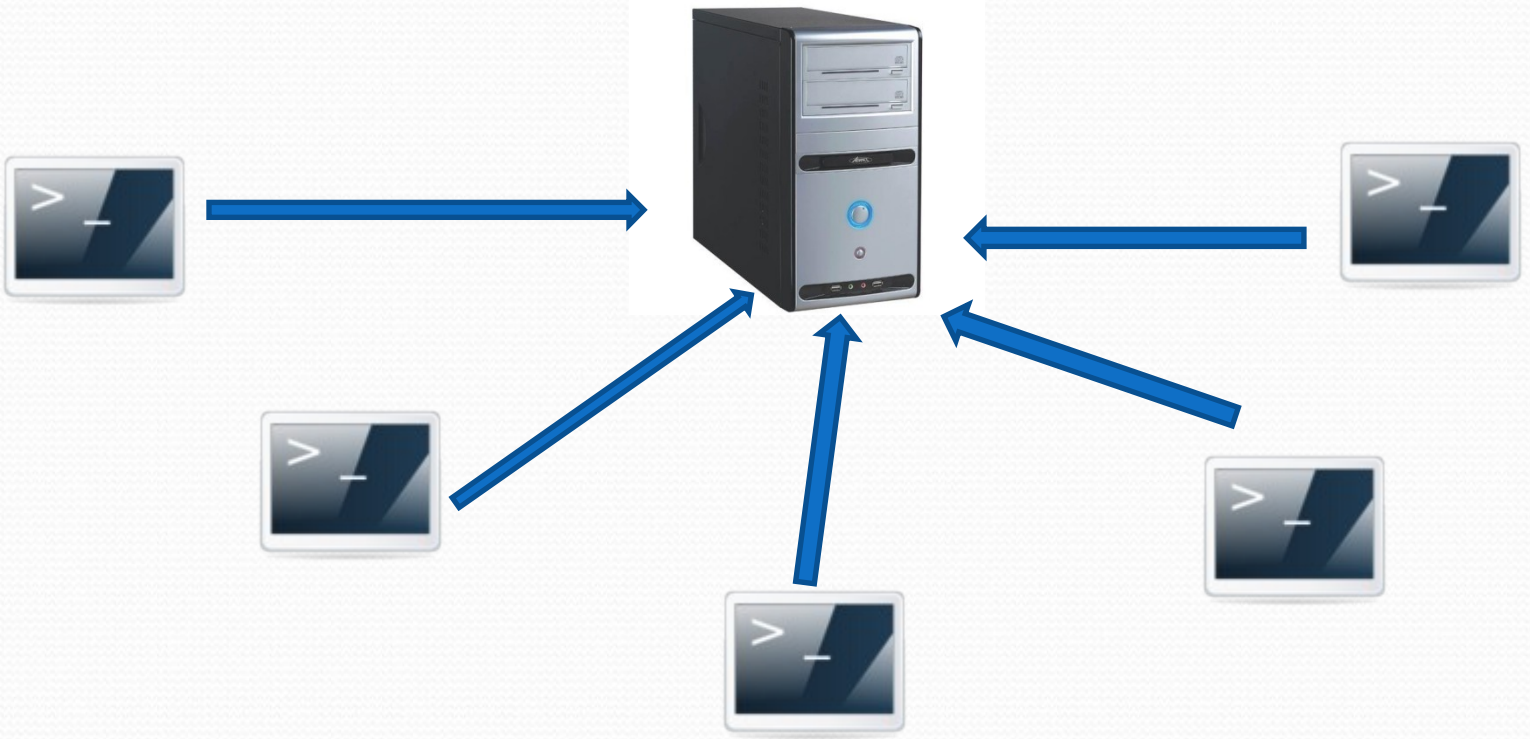
LA CONSOLA

Jorge Hernández Ramírez

Conceptos básicos

Terminal.- Dispositivo que permite al usuario interactuar con una máquina. Tiene la capacidad de enviar y recibir datos mediante un canal de comunicación.



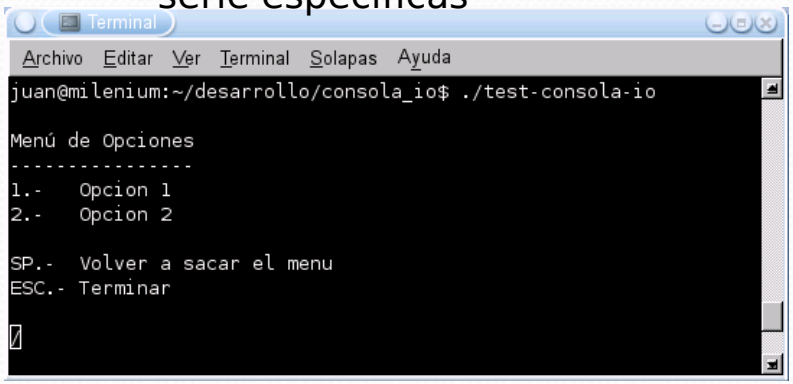
Conceptos básicos

Los terminales se representan por archivos especiales conocidos como **archivos de modo carácter** , en los que es posible leer cuando un usuario presiona una tecla del teclado, o escribir cuando el datos es enviado vía modem por el pue

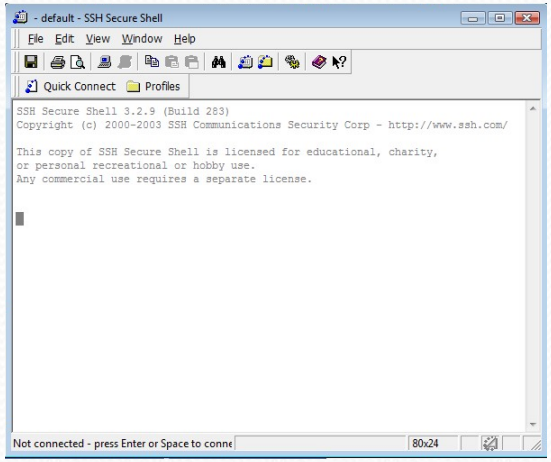
Tipos de terminales

Terminales particulares. Usado por las tarjetas de serie específicas

Puertos serie. Usado por modems y ratones



Consolas virtuales. Cuando estamos fisicamente en la máquina (local)



Pseudo terminales. Acceso remoto (local)

Modo de uso de un terminal

Modo canónico.- Un programa que intenta leer una línea en un terminal debe esperar que una línea completa haya sido introducida antes de poder tratarla.

Modo no canónico.- Los caracteres son tratados e interpretados por el programa de usuario. Los valores MIN y TIME se utilizan para determinar la manera como se reciben los caracteres. MIN corresponde al número mínimo de caracteres que deben recibirse antes de que la lectura sea satisfecha. TIME corresponde a un timer en décimas de segundo que se utiliza para hacer accesibles los datos tras un cierto lapso de tiempo.

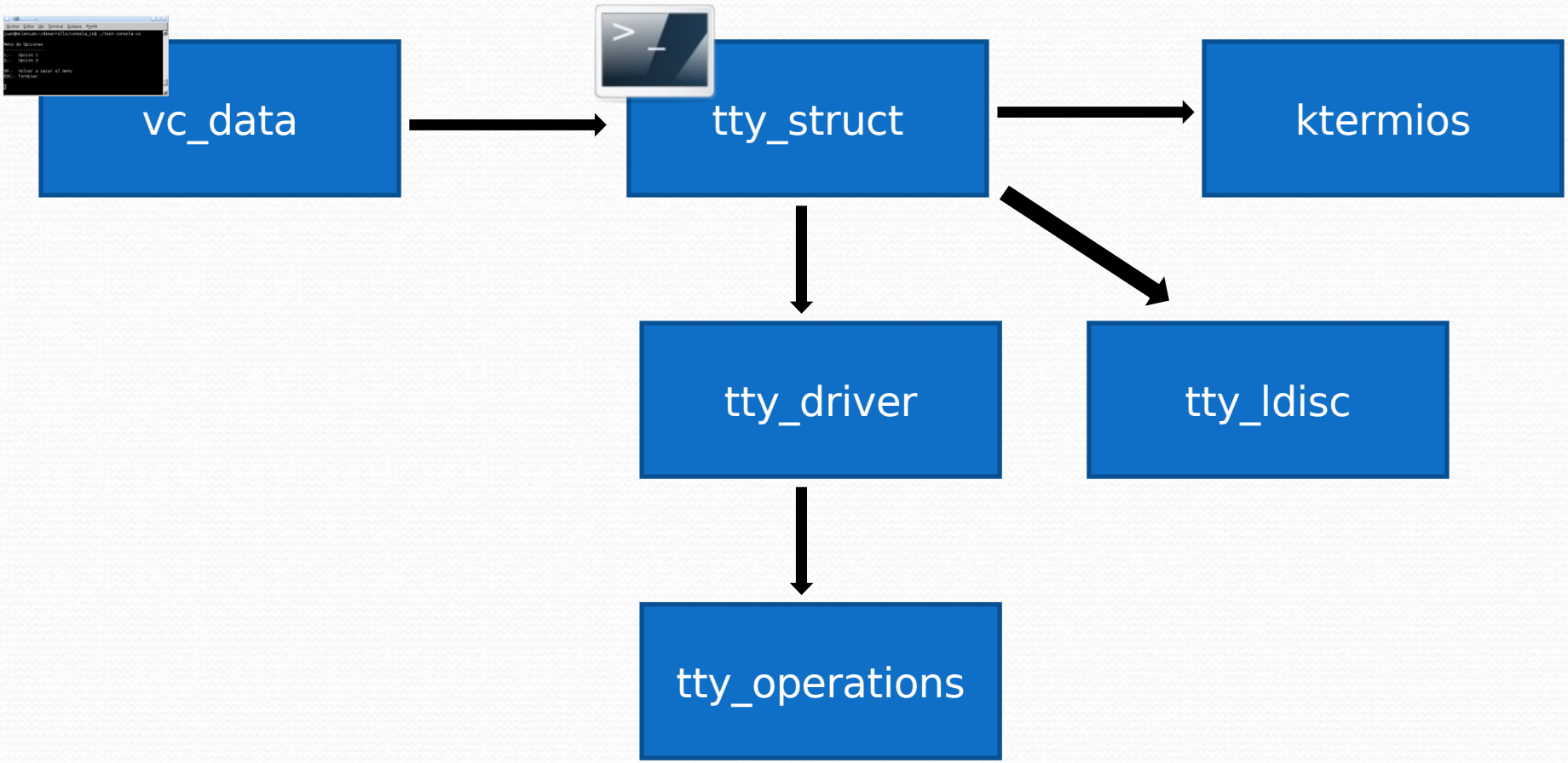
Comunicación proceso - terminal

Se realiza mediante memorias intermedias

El terminal tiene su protocolo de comunicación, y representación de los caracteres propio

Cuando un carácter pasa del terminal a la memoria debe sufrir un proceso de transformación y viceversa

Estructuras



Estructuras tty_struct

Estructura principal a partir del cual se efectúan todas las operaciones sobre el núcleo. Define al terminal a alto nivel

```
struct tty_struct {
 221        int     magic;
223        struct tty_driver *driver; /*Interfaz de acceso al disp. Asociado al terminal*/
 224        const struct tty_operations *ops; /*Operaciones de la consola. Se inicializarán las
operaciones de esta estructura con funciones como con_open.... */
 225        int index; /*Indice del terminal. Podemos tener varios*/
227        struct tty_ldisc ldisc; /*Interfaz para la disciplina de linea*/
231        struct ktermios *termios, *termios_locked; /*Configuración del terminal*/
236        unsigned long flags;  /*Estado del terminal*/
238        struct winsize winsize;         /* Tamaño de la ventana */
248        wait_queue_head_t write_wait; /*Lista de procesos en espera de escritura*/
 249        wait_queue_head_t read_wait; /*Lista de procesos en espera de lectura*/
252        void *driver_data; /*Puntero a la estructura de datos de la consola Si es nulo es
que no tiene asignado esta estructura*/
};
```

Estructuras tty_driver

Define al terminal a bajo nivel

```
struct tty_driver {
 270        int     magic;          /* numero mágico */
274         const char      *driver_name; /*Nombre del driver*/
281         short   type;           /* type of tty driver */
284         int     flags;          /* tty driver flags */
291         struct tty_struct **ttys; /*Puntero a la estruc. De datos de los terminales*/
300         const struct tty_operations *ops; /*Puntero a los operaciones del terminal*/
 301        struct list_head tty_drivers;
 302};
```

Estructuras tty_operations

Operaciones que se pueden realizar sobre el terminal. Posteriormente deberán ser asignadas dichas funciones

```
struct tty_operations {
231         int  (*open)(struct tty_struct * tty, struct file * filp); /*Apertura del
terminal*/
 232        void (*close)(struct tty_struct * tty, struct file * filp); /*Cierre del
terminal*/
234         int  (*write)(struct tty_struct * tty,
 235                     const unsigned char *buf, int count); /*Escritura del terminal*/
};
```

Estructuras tty_ldisc

Operaciones a nivel de lineas

```
145struct tty_ldisc {
 146        struct tty_ldisc_ops *ops
; /*Puntero a las operaciones sobre la
disciplina de linea*/
 147        int refcount;
 148};
```

```
struct tty_ldisc_ops {
 108        int     magic;   /*Numero mágico*/
110        int     num;     /*Identificador de la línea*/
 111        int     flags;   /*Tipo de linea*/
116        int     (*open)(struct tty_struct *);  /*Apertura de
la línea*/
 117        void    (*close)(struct tty_struct *);  /*Cierre de
la linea*/
120        ssize_t (*read)(struct tty_struct * tty, struct file
* file,
 121                        unsigned char __user * buf, size_t nr
);
 122        ssize_t (*write)(struct tty_struct * tty, struct file
* file,
 123                         const unsigned char * buf, size_t nr
```

Estructuras ktermio

Permite examinar la configuración del terminal y modificar los parámetros

```
40struct ktermios {
 41        tcflag_t c_iflag;             /* Modo de entrada */
 42        tcflag_t c_oflag;              /* Modo de salida */
 43        tcflag_t c_cflag;              /* Modo de control*/
 44        tcflag_t c_lflag;             /* Modo locales*/
 45        cc_t c_line;                 /* Caracteres de control*/
 46        cc_t c_cc[NCCS];                /* Disciplina de la línea*/
 47        speed_t c_ispeed;               /* Velocidad de entrada*/
 48        speed_t c_ospeed;               /* Velocidad de salida*/
 49};
```

Estructuras vc_data

Define la consola virtual, que deberá estar asociada a un terminal

```
struct vc_data {
 24        unsigned short  vc_num;                 /* Console number */
           unsigned int    vc_cursor_type;      /*Tipo de cursor*/
 59        struct tty_struct *vc_tty;             /*Terminal en el que se
encuentra la consola
};
```

Inicialización

```
int __init vty_init(const struct file_operations *console_fops)
2931{
2940        console_driver = alloc_tty_driver(MAX_NR_CONSOLES);/*Reserva memoria para el
dispositivo de la consola*/
2941        if (!console_driver)  //Console driver es de tipo tty
2942                panic("Couldn't allocate console driver\n");
        //Se inicializa la consola
2943        console_driver->owner = THIS_MODULE;
2944        console_driver->name = "tty";
2945        console_driver->name_base = 1;
2946        console_driver->major = TTY_MAJOR;
2947        console_driver->minor_start = 1;
2948        console_driver->type = TTY_DRIVER_TYPE_CONSOLE;
2949        console_driver->init_termios = tty_std_termios;
2950        if (default_utf8)
2951                console_driver->init_termios.c_iflag |= IUTF8;
2952        console_driver->flags = TTY_DRIVER_REAL_RAW | TTY_DRIVER_RESET_TERMIOS;
        //Inicializa el conjunto de operaciones posibles
2953        tty_set_operations(console_driver, &con_ops);
        //Esta función inserta el dispositivo en la lista encadenada de dispositivos (registro)

2954        if (tty_register_driver(console_driver))
2955                panic("Couldn't register console driver\n");
2964        return 0;
2965}
2966
```

Creación y configuración de la consola

```
static int __init con_init(void)
2841{
 acquire_console_sem();  //Adquirimos el cerrejo
//Se crea y se configura la consola

2880        for (currcons = 0; currcons < MIN_NR_CONSOLES; currcons++) {
                                //Se reserve memoria para albergar la consola
2881                vc_cons[currcons].d = vc = alloc_bootmem(sizeof(struct vc_data));
2882                INIT_WORK(&vc_cons[currcons].SAK_work, vc_SAK);
2883                visual_init(vc, currcons, 1);
2884                vc->vc_screenbuf = (unsigned short *)alloc_bootmem(vc->
vc_screenbuf_size);
2885                vc->vc_kmalloced = 0;
                  /*Función de configuración de la consola (color...)*/
2886                vc_init(vc, vc->vc_rows, vc->vc_cols,
2887                        currcons || !vc->vc_sw->con_save_screen);
16}
/*Actualiza la pantalla*/
18update_screen(vc);
/*Registra la consola e imprime los mensajes del kernel*/
20register_console(&vt_console_driver);
21}
```

Operaciones

Antes vimos que la estructura tty_driver tenía un campo struct tty_operations *ops que apuntaba a la estructura tty_operations. Es aquí donde se asignan dichas operaciones. Nos centraremos en las funciones con_open, con_close, con_write.

```
static const struct tty_operations con_ops = {
    .open = con_open,
    .close = con_close,
    .write = con_write,
    .write_room = con_write_room,
    .put_char = con_put_char,
    .flush_chars = con_flush_chars,
    .chars_in_buffer = con_chars_in_buffer,
    .ioctl = vt_ioctl,2920        .stop = con_stop, 1
    .start = con_start,2922        .throttle = con_throttle,
    .unthrottle = con_unthrottle,
    .resize = vt_resize,
    .shutdown = con_shutdown};
```

CON-OPEN

```
static int con_open(struct tty_struct *tty, struct file *filp)
2752{
        /*Index indica que numero de terminal queremos asociar a la consola a abrir.*/
2753        unsigned int currcons = tty->index;
2754        int ret = 0;
2755
              //Adquirimos el cerrojo
2756        acquire_console_sem();
        /*Si no tiene asignado una estructura de datos de la consola*/
2757        if (tty->driver_data == NULL) {
        /* Esta función inicializa el entorno gráfico de la consola, comprueba que hay memoria, la reserva si no hay más de un número máximo de terminales abiertos, genera el terminal gráfico con los colores por defecto, etc. Si la llamada a esta función se realiza con éxito, esta función devolverá un 0*/

2758                ret = vc_allocate(currcons);
```

```
2759            if (ret == 0) {
                                        /*Inicializamos la consola virtual que
corresponde al terminal currcons*/
2760                    struct vc_data *vc = vc_cons[currcons].d;
2761
2767                    tty->driver_data = vc;
2768                    vc->vc_tty = tty; //Se le asigna el terminal a la consola
2769//Se le asigna el tamaño de la ventana
2770                    if (!tty->winsize.ws_row && !tty->winsize.ws_col) {
2771                            tty->winsize.ws_row = vc_cons[currcons].d->vc_rows;
2772                            tty->winsize.ws_col = vc_cons[currcons].d->vc_cols;
2773                    }
2778                    vcs_make_sysfs(tty); //Se libera el semaforo
2779                    release_console_sem();//Guardamos en un fichero el
Terminal creado

2780                    return ret;
2781            }
2782    }
2783    release_console_sem();
2784    return ret;
2785}
```

CON-CLOSE

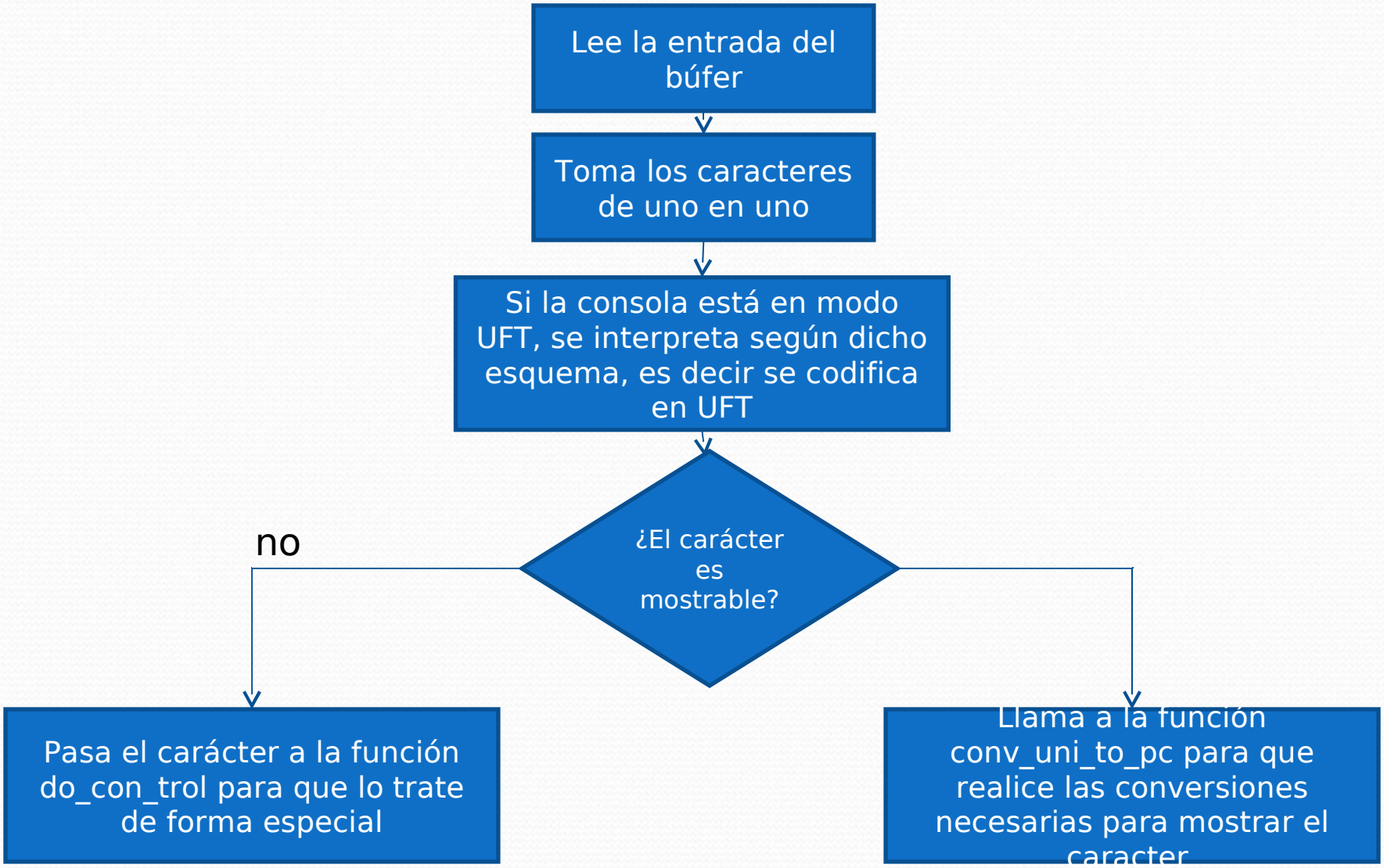
```
2787static void con_close(struct tty_struct *tty, struct file *filp)
2788{
2789        /* Nothing to do - we defer to shutdown */
2790}

2792static void con_shutdown(struct tty_struct *tty)
2793{
          //Obtenemos la 'consola' de ese terminal
2794         struct vc_data *vc = tty->driver_data;
2795         BUG_ON(vc == NULL);
          //Adquirimos el cerrojo
2796         acquire_console_sem();
          //Desvinculamos la 'consola' del terminal
2797         vc->vc_tty = NULL;
2801}
```

CON-WRITE

```
static int con_write(struct tty_struct *tty, const unsigned char *buf, int count)
{
2660        int     retval;
2661
2662        retval = do_con_write(tty, buf, count); //Se llama a do_con_write
2663        con_flush_chars(tty);
2664
2665        return retval;
2666}
```

DO-CON-WRITE


Lee la entrada del búfer
Toma los caracteres de uno en uno
Si la consola está en modo UFT, se interpreta según dicho esquema, es decir se codifica en UFT
¿El carácter es mostrable?
no
Pasa el carácter a la función do_con_trol para que lo trate de forma especial
Llama a la función conv_uni_to_pc para que realice las conversiones necesarias para mostrar el caracter

DO-CON-WRITE

```
//Especificamos cuanto queremos escribir con la variable count
static int do_con_write(struct tty_struct *tty, const unsigned char *buf, int count)
2094{
2122        acquire_console_sem();  //Adquiere el semáforo
2123        vc = tty->driver_data;  //Obtiene el la consola asociada

                //Si el terminal no tiene asociada ningúna consola error
2124        if (vc == NULL) {
2125                printk(KERN_ERR "vt: argh, driver_data is NULL !\n");
2126                release_console_sem();
2127                return 0;
12}
 //Lee cada uno de los caracteres del buffer y los va tratando
2153        while (!tty->stopped && count) {
2154                int orig = *buf;
2155                c = orig;
2156                buf++;
2157                n++;
2158                count--;
2159                rescan = 0;
2160                inverse = 0;
2161                width = 1;
2162
```

```
2163            /* Do no translation at all in control states */
2164            if (vc->vc_state != ESnormal) {
2165                    tc = c; //Si está en modo normal se coge el carácter tal cual
2166            }
                //Si la consola está en modo UTF, la secuencia de caracteres se
interpreta según el esquema de codificación UTF
        else if (vc->vc_utf && !vc->vc_disp_ctrl) {
                .....
        } else { //Si no esta en modo UTF se traduce
2238            tc = vc_translate(vc, c);
}
```

- **//Se comprueba que el carácter sea imprimible o no**

```
2256            ok = tc && (c >= 32 ||
2257                    !(vc->vc_disp_ctrl ? (CTRL_ALWAYS >> c) & 1 :
2258                            vc->vc_utf || ((CTRL_ACTION >> c) & 1)))
2259                && (c != 127 || vc->vc_disp_ctrl)
2260                && (c != 128+27);
2261
2262        if (vc->vc_state == ESnormal && ok) {
2263                if (vc->vc_utf && !vc->vc_disp_ctrl) {
2264                        if (is_double_width(c))
2265                                width = 2;
2266                }
```

//En caso afirmativo, se llama a la función conv_uni_to_c para determinar las conversiones necesarias a llevar a cabo para imprimir el carácter

```
2268                tc = conv_uni_to_pc(vc, tc);
```

//en caso negativo, se llama a la función do_con_trol para tratar el carácter especial.

```
            }
            …
2352                do_con_trol(tty, vc, orig);
2353        }
```

Bibliografía

Linux Cross References
http://lxr.linux.no

Apuntes y transparencias 2007/2008