MANEJADOR

DE

CONSOLA

Daniel Ramírez Viera
David Cortes Alvarez

Índice

1.- Introducción pág.
2.- Terminales con mapa de memoria pág.
2.1.- Terminales orientados a carácter pág.
2.2.- Terminales orientados a bit pág.
3.- Software de salida pág.
4.- Generalidades del controlador. Salida a terminales pág.
4.1.- Funcionamiento de la memoria pág.
5.- Diagrama de relación de llamadas a console desde tty pág.
6.- Console.c pág.
6.1.- Estructura de datos básica pág.
6.2.- Función *cons_write* pág.
6.3.- Función *out_char* pág.
6.4.- Función *flush* pág.
6.5.- Función *cons_echo* pág.
6.6 Función *scroll_screen* pág.
6.7 Función *parse_escape* pág.
6.8 Función *do_escape* pág.
6.9 Función *beep* pág.
6.10 Función *stop_beep* pág.
6.11 Función *src_init* pág.
6.12 Función *putk* pág.
6.13 Función *toggle_scroll* pág.
6.14 Función *cons_stop* pág.
6.15 Función *select_console* pág.
6.16 Función *cons_org0* pág.
6.17 Funciónes *con_loadfont* y *ga_program* pág.

1.- Introducción.

Los ordenadores de propósito general disponen de una o mas terminales que sirven para comunicarse con ellas. Debido a que existe un número muy grande de tipos de terminales distintos, le corresponde al controlador de la terminal (tty) ocultar todas las diferencias a la parte del sistema operativo independiente del dispositivo y los programas de usuario. Este controlador utiliza unas rutinas determinadas para el tratamiento de las consolas, que es el caso que nos ocupa. Para entender completamente el funcionamiento de las consolas, en las siguientes secciones examinaremos primero el hardware de las terminales y luego estudiaremos el software implicado.

2.- Terminales con mapa de memoria.

Estudiamos sólo esta categoria de hardware de terminales debido a que la consola de MINIX es una pantalla con mapa de memoria.

En estos sistemas, el elemento que enlaza los demas dispositivos es la **tarjeta de video**. La interfaz con las terminales con mapa de memoria se establece principalmente a través de la memoria que poseen, denominada **RAM de vídeo**, que fsta forma parte del espacio de direcciones del ordenador y es direccionada por la CPU de la misma manera que el resto de la memoria del sistema. Otro componente de la tarjeta es el **controlador de vídeo**, que se encarga de extraer los códigos de caracteres de la RAM de video y generar posteriormente la señal de video que maneja la pantalla.

Dentro de los terminales mapeados en memoria, el IBM PC dispone de dos modos para la pantalla, que comentamos a continicación.

2.1.- Terminales orientados a carácter.

Es este caso se usa la pantalla con mapa de caracteres para la consola y la unidad mínima que se maneja es el carácter. En la memoria RAM de video, cada carácter ocupa dos bytes. El byte de orden mas bajo representa el código ASCII para el carácter que se exhibe, mientras que el de orden alto es el byte de atributio (especifica el color, vídeo inverso, parpadeo, etc.). Por lo tanto, la pantalla completa de 25 por 80 caracteres ocupa un total de 4000 bytes de RAM de vídeo. Todo esto queda representado mediante la figura 1.

2.2.- Terminales orientados a bit.

En estas terminales, se controla individualmente cada pixel de la pantalla. Necesitaremos un byte de memoria RAM de vídeo para indicar la intensidad del pixel, en el caso de una pantalla monocromática, o tres bytes para cada pixel en las pantallas a color, un para cada color básico (rojo, verde y azul). Entonces, una pantalla a color de 768 x 1024 con 24 bits por pixel requeriría 2MB de RAM únicamente para contener la imagen.

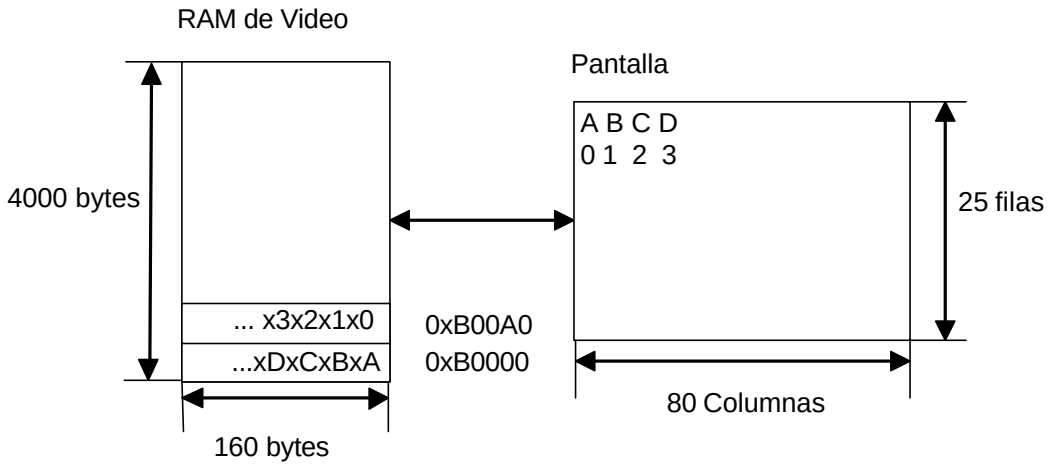


Figura 1

3.- Software de salida.

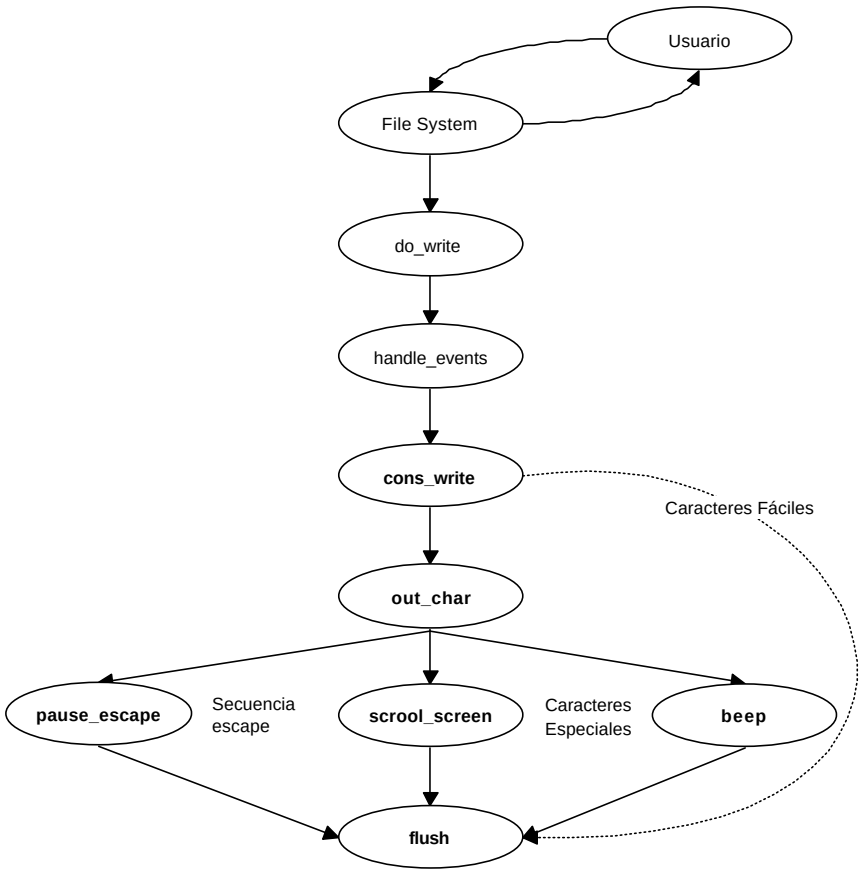
A continuación comentaremos el trabajo básico del controlador de consola en MINIX. En las terminales mapeadas en memoria orientadas a carácter (el caso que nos ocupa), los caracteres que se van a exhibir en panalla se extraen uno por uno del espacio de usu ario y se colocan (junto con el atributo correspondiente) directamente cn la memoria RAM de vídeo. Algunos caracteres requieren un tratamiento en especial, como el retroceso, el retorno de carro, el salto de línea y la alarma audible. La mayoría de ellos (las mencionadas anteriormente menos la alarma) requieren que el controlador software actualice de forma apropiada la posicion actual del cursor.

En particular, cuando un salto de línea es introducido con la posicion del cursor en la última línea, es preciso avanzar la pantalla, es decir, realizar un scroll. Esto requeriría copiar un bloque de memorie de 24x80 caracteres (cada carácter son dos bytes), lo cual sería bastante costoso. Por fortuna, el hardware ofrece ayuda en muchas ocasiones. La mayor parte de los controladores devideo contienen un registro que determina de qué punto de la RAM de video debe comenzarse a ver los caracteres en la pantalla. Bastaría con cambiar el valor de este registro 160 caracteres (una línea en pantalla) mas adelante en la memoria de vídeo. Entonces, la línea que era antes la numero dos pasara a ser la primera, y toda la pantalla se desplazará una línea más arriba.

En otro orden de cosas, los editores de pantalla y muchos otros programas avanzados necesitan actualizar la pantalla de formas mas complejas que las que permiten las operaciones por defecto. A fin de darles servicio, muchos controladores de terminal reconocen diveras secuencias de escape, que se mostrarán y detallarán en apartados posteriores.

4.- Generalidades del controlador. Salida a terminales.

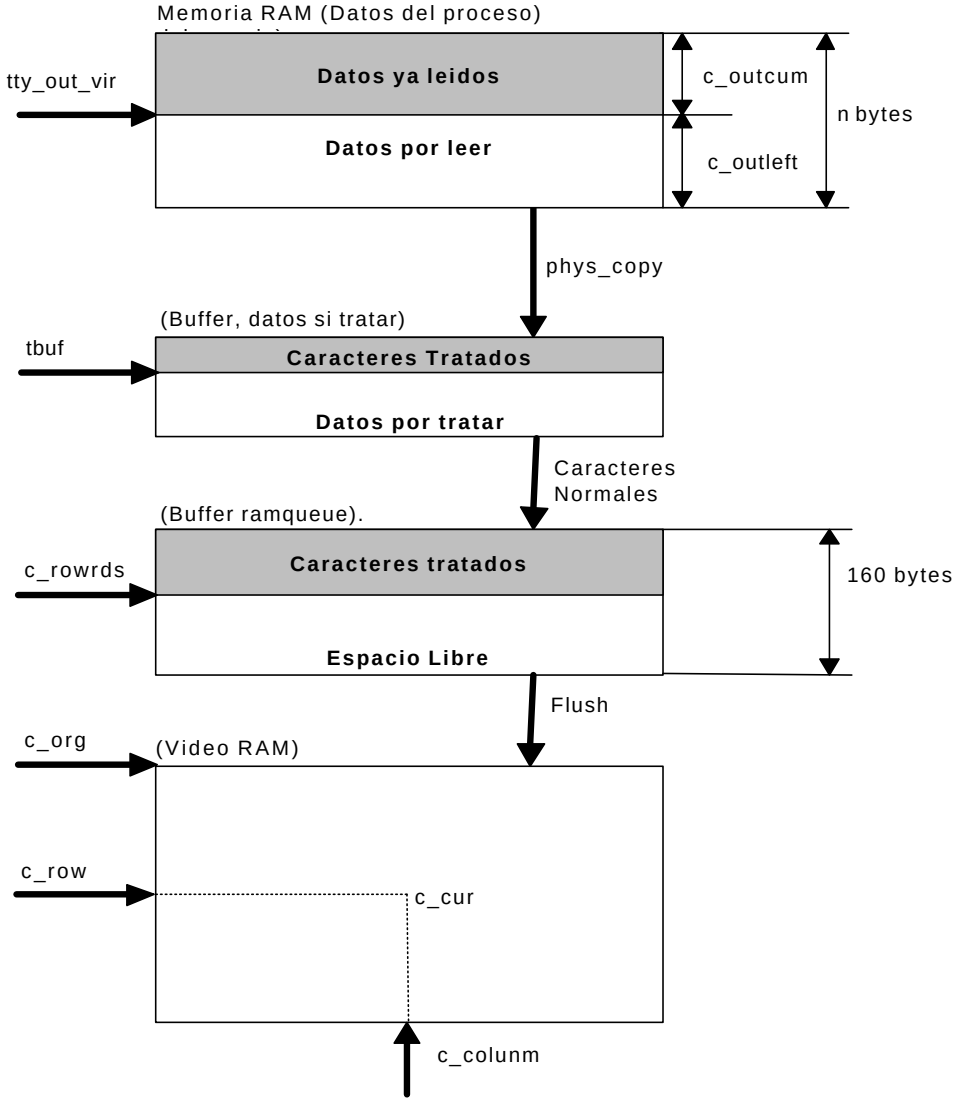
En esta sección rastrearemos los pasos que se siguen para enviar caracteres a la pantalla de la consola. Cuando un proceso desea exhibir algo, generalmente invoca *prinft*, la cual invoca WRITE para enviar un mensaje al sistema de archivos. Hay que mencionar que en mensaje contiene un apuntador a los caracteres por exhibir y no a los caracteres mismos. Entonces, el *file system* llama a driver del terminal, que invoca a *cons_write* para comenzar el trabajo de *console.c* (obtener los caracteres y copiarlos en la RAM de vídeo). La siguiente figura muestra los principales procedimientos que intervienen en las salidas.



La funcion *cons_write* utiliza un buffer local para copiar bloques de 64 bytes desde la zona de datos del proceso que llamo. Cuando éste se llena, cada byte de 8 bits se trasnfiere a otro buffer, *ramqueue*, que es un arreglo de palabras de 16 bits. Cada segundo byte se llena con el valor actual de atributo. Si es posible, los caracteres se van transfiriendo directamente a *ramqueue*, pero los caracteres de control o los que formasn parte de una secuencia de escape necesitan un manejo especial. En estos casos se invoca *out_char*, que se encarga de realizar las acciones adecuadas y actualizar los parámetros correspondientes (scrolls, posición del cursor, ete.). El búffer *ramqueue* es copiado posteriormente en memoria vídeo RAM mediante *flush*.

4.1.- Funcionamiento de la memoria.

A continuación se muestra un esquema de todas las variables implicadas en el manejo de la memoria durante el todo el proceso de exhibir caracteres en pantalla.



Podemos tener asociada a cada monitor varias pantallas virtuales. Los parámetros *c_start* y *c_limit* identifican la dirección de memoria vídeo RAM de inicio y final para cada una de estas terminales virtuales. En el parámetro *c_org* se introduce la dirección a partir de la cual queremos mostrar en pantalla, el chip 6845 visualizara desde esta posición hasta los siguientes 4000 bytes (modo carácter).

Al arranque de la computadora, la pantalla de despeja, se exriben salidas en RAM de vídeo comenzando en la posicion *c_start*, y se asigna a *c_org* el mismo valor que tiene *c_start*. Por lo tanto, la primera línea aparece en la línea superior de la pantalla. Cuando es necesario exhibir salidas en una nueva línea, las salidas se escriben en al posicion dada por *c_start* mas 80. Tarde o temprano se llenan las 25 líneas, y es necesario desplazar, haciendo avanzar la pantalla hacia arriba (SCROLL UP)

Algunos programas, como los editores, requieren también desplazamiento hacia abajo (SCROLL DOWN), cuando el cursor está en la línea superior y es necesario moverse más hacia arriba dentro del texto.

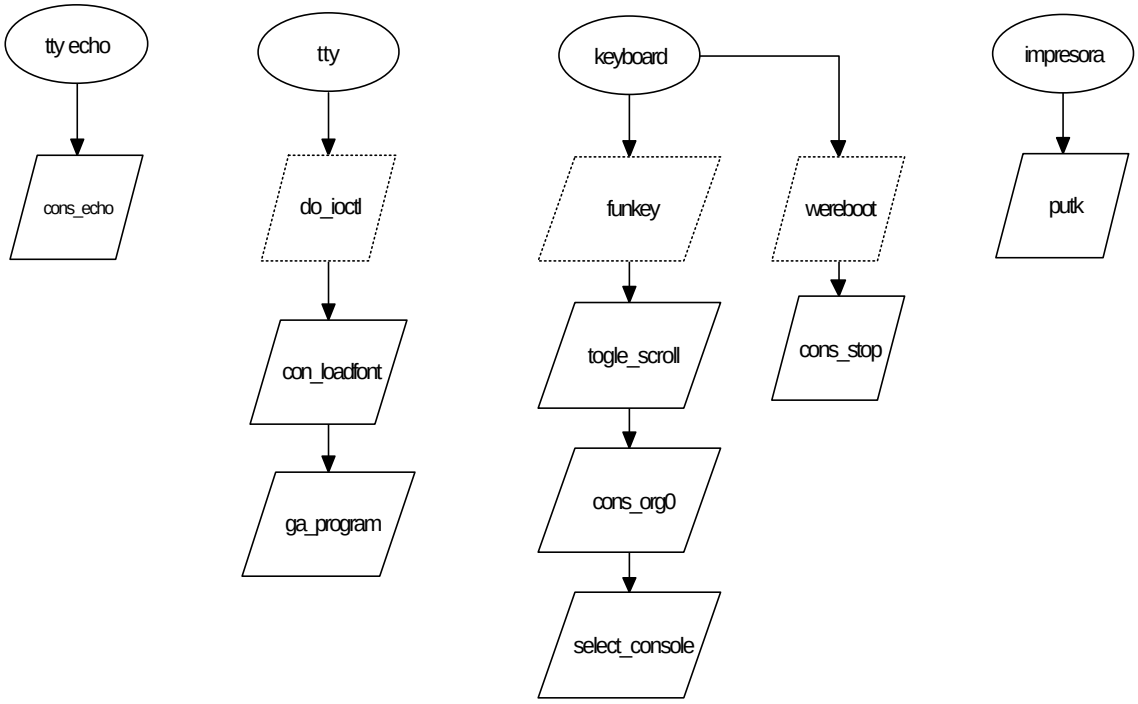
Hay dos formas de manejar el desplazamiento de la pantalla:

a) Desplazamiento por software: Cuando es detectado que se debe realizar un scroll, copiamos los bloques de memoria desde *c_start* mas 160 bytes hasta *c_start* (SCROLL UP), o desde *c_start* hasta *c_start* mas 160 bytes (SCROLL DOWN). El costo es que la CPU ha movido 80x24=1920 palabras.

b) Desplazamiento por hardware: No hay movimientos de bloques de memoria, sino que cambiamos el registro *c_org*, pasándoselo al chip 6845 para que empiece a visualizar desde esa posición. Para ello se requiere que el chip controlador tenga la suficiente inteligencia como para manejar la RAM de vídeo en forma "circular", tomando los datos del principio de la RAM (*c_start*) después de haber llegado al final (*c_limit*), o bien que la RAM de vídeo tenga más capacidad que las necesarias para almacenar una sola pantalla.

Cuando se configuran consolas virtuales, la memoria disponible dentro de un adaptador de vídeo se divide equitativamente entre el número de consolas deseadas inicializando debidamente los campos *c_start* y *c_limit* para cada consola. Esto afecta al desplazamiento en pantalla. En cualquier adaptador con el tamao suficiente para apoyar consolas virtuales, el desplazamiento por software ocurre con cierta frecuencia, incluso si normalmente está vigente el desplazamiento por hardware. Cuanto menor sea la cantidad de memoria disponible para cada pantalla de consola, con mayor frecuencia será necesario usar el desplazamiento por software. Se llega al límite cuando se configura el número máximo posible de consolas. Entonces, toda operación de desplazamiento tendrá que efectuarse por software.

5.- Diagrama de relación de llamadas a console desde tty.



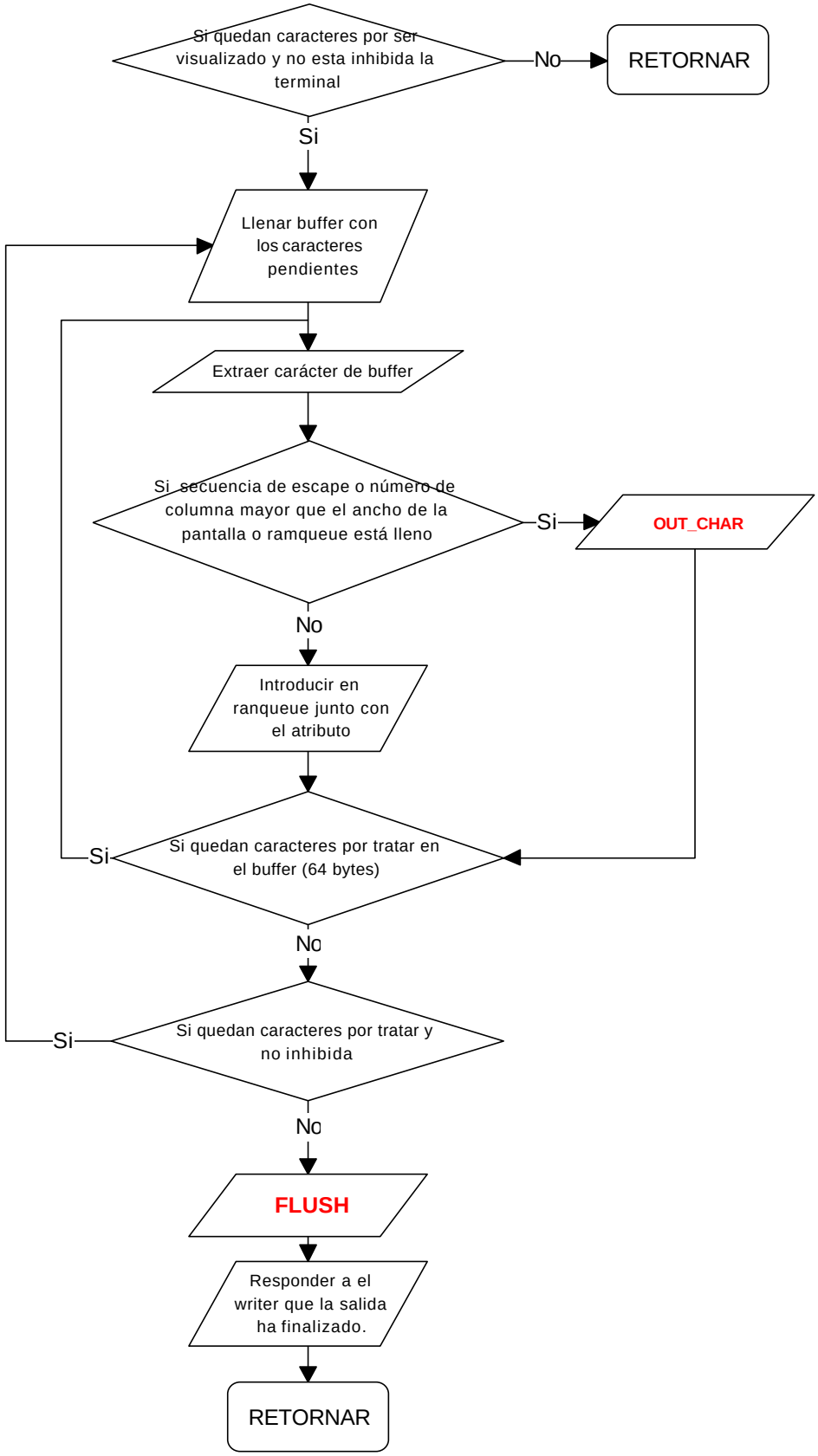
6.- Console.c

Este apartado se encargará de describir con detalle las estructuras de datos y las funciones pertenecientes a console.c. Para cada función existente, se mostrará su diagrama de flujo, los datos más utilizados y se detallará su funcionamiento.

6.1.- Estructura de datos básica

```
typedef struct console {
      tty_t *c_tty;       /* struct TTY asociado */
      int c_colum;        /* columna actual */
      int c_row;          /* fila actual */
      int c_rwords;       /* num. Words en outqueue */
      unsigned c_start;   /* inicio de mem video  */
      unsigned c_limit;   /* límite de memoria de video */
      unsigned c_org;     /* lugar en RAM donde apunta base 6845 */
      unsigned c_cur;     /* pos. actual del cursor */
      unsigned c_attr;    /* atributo del caracter */
      unsigned c_blank;   /* atributo blanco */
      char c_esc_estate;  /* 0=normal, 1=ESC, 2=ESC[ */
      char c_esc_intro;   /* carácter distintivo sigue a ESC */
      int *c_esc_parmp;   /* apunta al parametro de ESC actual */
      int c_esc_parmv[MAX_ESC_PARMS];        /* lista de parametros ESC */
      u16_t c_ramqueue[CONS_RAM_WORDS];      /* buffer p/RAM de video */
} console_t;
```

6.2.- Función *cons_write*.



DEFINE

scr_width	80 (caracteres en una línea).

STRUCT TTY

tty_priv	puntero a los datos particulares de un device.
tty_outleft	caracteres pendientes de salida
tty_outcum	caracteres a los que ya se ha dado salida.
tty_inhibited	1 si (^S) para parar la salida
tty_out_vir	Dirección virtual desde donde vienen los datos.

STRUCT CONSOLE

c_column	Posición actual de la columna
c_rowrds	número de palabras (WORD) en ranqueue
c_ramqueue	buffer para vídeo RAM
c_attr	atributo del carácter
c_esc_state	0=normal, 1=ESC, 2=ESC[

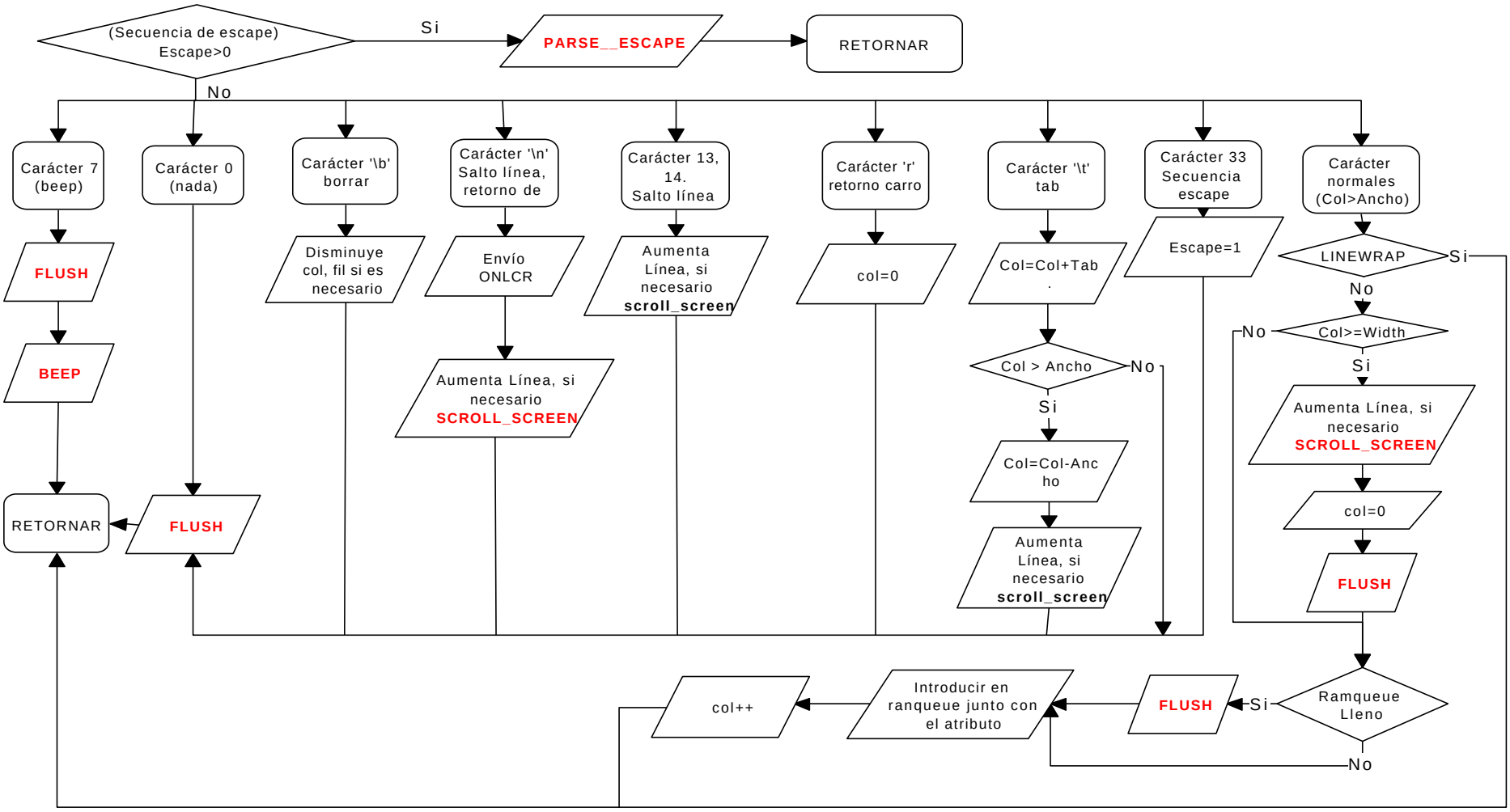
PARAMETROS ENTRADA

Struct tty *tp	La terminal con la que trabajaremos

VARIABLES

Buf	buffer intermedio (64 bytes) donde se almacena datos por tratar.
tbuf	puntero a caracteres en buf
Cont	número de caracteres por tratar de buf

6.3.- Función *out_char*.



DEFINE	
scr_lines	(25) número de filas de la terminal
scr_width	(80) número de columnas de la terminal
TAB_SIZE	tamaño de la tabulación
LINEWRAP	indica si se ha completado la salida de una línea en la pantalla.
STRUCT CONSOLE	
c_row	posición actual de la fila
c_column	posición actual de la columna
c_rowrds	número de palabras (WORD) en ranqueue
c_ramqueue	buffer para video RAM
c_attr	atributo del carácter
c_esc_state	0=normal, 1=ESC, 2=ESC[
PARÁMETROS DE ENTRADA	
Console_t *cons	Puntero a la estructura consola.
int c	Carácter que vamos a tratar

6.4.- Función *flush*.

Este procedimiento copia los datos desde el buffer ramqueue (datos ya tratados) a la memoria de Video.

1. Si hay caracteres en ramqueue los transfiere a memoria video y vacía ramqueue.

2. Chequea las variables columna y fila por si debe actualizarlas (otros procedimientos las pueden dejar con valores negativos).

3. Calcula la posición del cursor y si es distinta a la anterior indica donde se debe dibujar el cursor.

STRUCT TTY	
tty_priv	puntero a los datos particulares de un device.
tty_position	Actual posición en la pantalla para enchoing
tty_reprint	Flag que indica si se está haciendo enchoing.
STRUCT CONSOLE	
c_row	Posición actual de la fila
c_column	Posición actual de la columna
c_cur	posición actual del cursor en Vídeo RAM.
c_org	Posición Vídeo RAM desde la que el chip 6845 mostrará los caracteres.
c_rowrds	número de palabras (WORD) en ranqueue
c_ramqueue	buffer para vídeo RAM
PARAMETROS	
console_t *cons	Puntero a la estructura consola.
VARIABLES	
cur	Para calcular la posición del cursor.

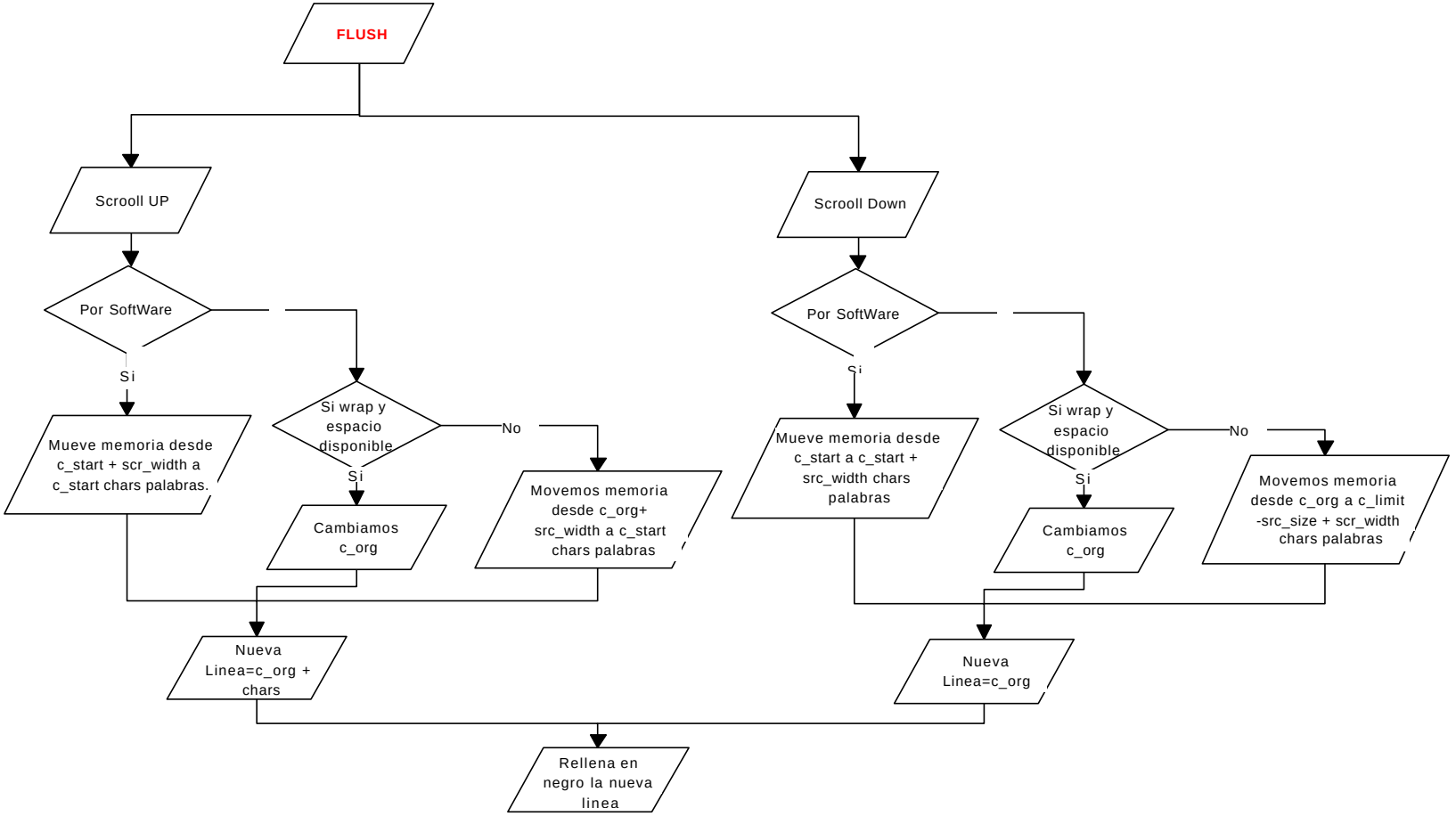
6.5.- Función *cons_echo*.

Este procedimiento se utiliza cuando esta activado el modo echo, su funcionamiento es el siguiente:

1. Llama a out_char.
2. Llama a flush.

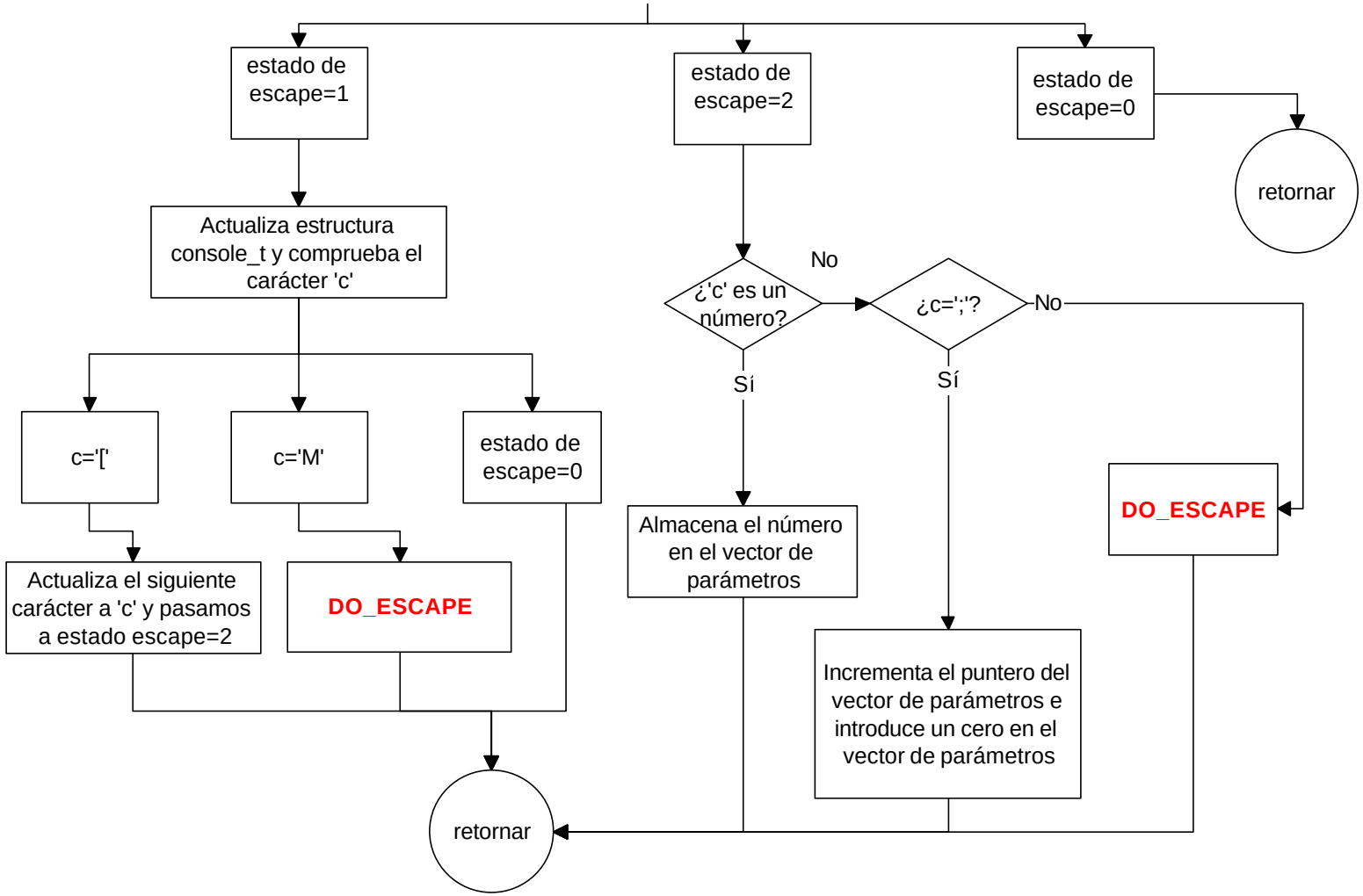
STRUCT TTY	
tty_priv	puntero a los datos particulares de un device.
PARAMETROS	
Struct tty *tp	La terminal con la que trabajaremos
int c	Carácter que vamos a tratar

6.6 Función *scroll_screen*.



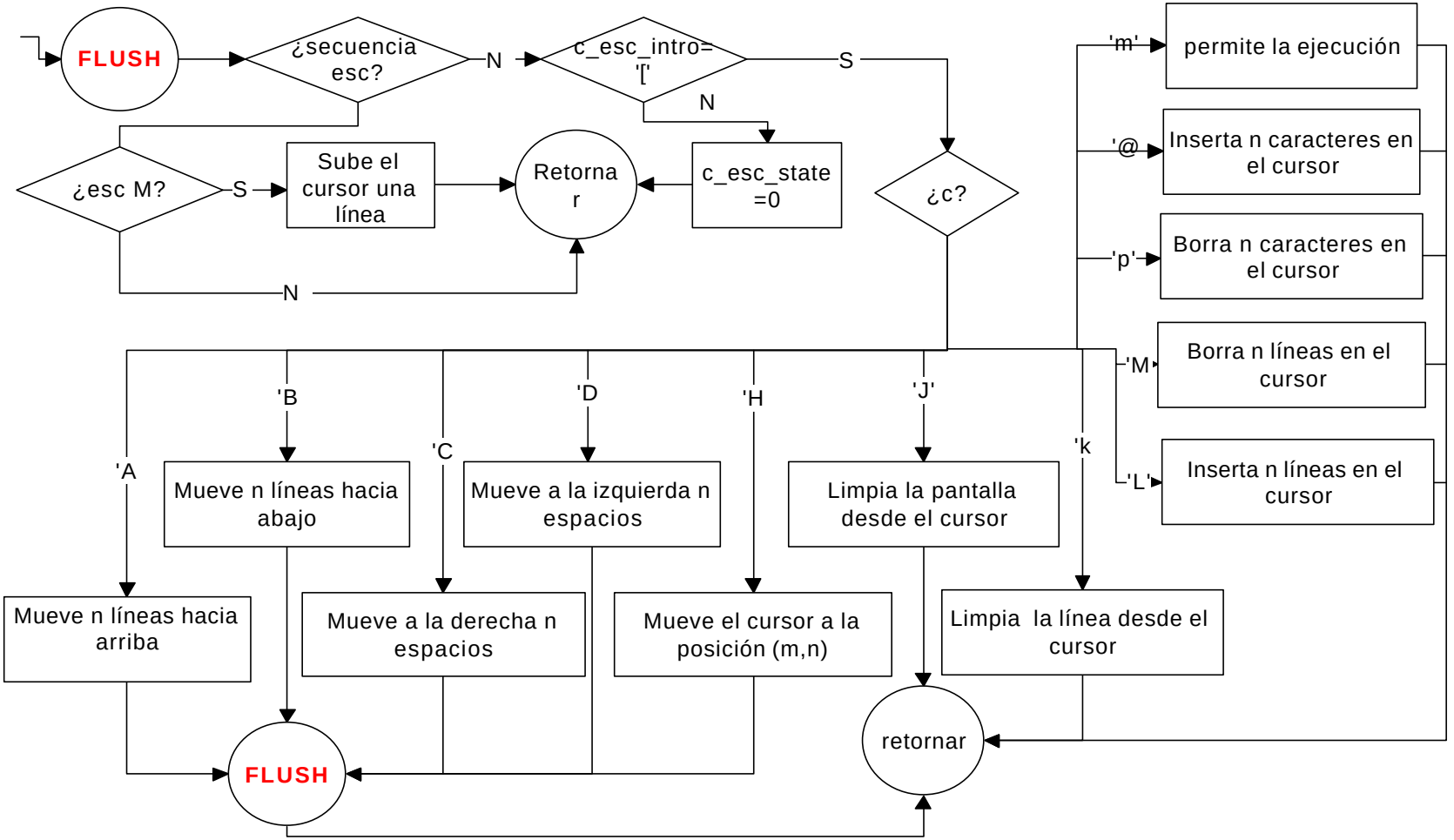
DEFINE	
scr_size	Tamaño de la pantalla (25*80).
scr_width	Ancho de la pantalla (80).
BLACK_MEM	Se utiliza para rellenar de negro la nueva línea.
STRUCT CONSOLE	
c_start	Dirección de comienzo de memoria para la consola.
c_limit	Dirección limite de memoria para la consala.
c_org	Dirección origen desde donde el chip 6845 empieza a visualizar en pantalla.
PARAMETROS DE ENTRADA	
cons	Estructura de la consola.
dir	Dirección del scroll (up o down).
VARIABLES	
new_line	Dirección de la nueva a ser rellenada en negro.
new_org	Dirección del nuevo org.
chars	Número de bloques de memoria a mover.

6.7 Función *parse_escape*.



STRUCT CONSOLE	
c_esc_state	0=normal, 1=ESC, 2=ESC[
c_esc_intro	Carácter siguiente al ESC
c_esc_parmp	Puntero al siguiente parámetro de ESC
c_esc_parmv	Lista de los parámetros de ESC
PARAMETROS DE ENTRADA	
Console_t *cons	Puntero a la estructura console.
char c	Siguiente carácter en la secuencia de escape.

6.8 Función *do_escape*.



DEFINICIONES

src_size	Número de caracteres de la pantalla (80*25).
BLANK_MEM	Valor del carácter negro para el uso en mem_vid_copy.
scr_width	Número de caracteres por línea (80).
scr_lines	Número de líneas de la pantalla (25).
color	Color usado por el controlador (vid_port==c_6845).
scroll_down	Scroll hacia atrás (1).
scroll_up	Scroll hacia delante (0).

STRUCT CONSOLE

c_esc_intro	carácter siguiente al ESC
c_row	fila actual
c_esc_paramv	lista de los parámetros de ESC
c_column	columna actual
c_cur	posición actual del cursor en la video RAM
c_org	base en la video RAM para el 6845
c_blank	atributo blank
c_attr	atributo del carácter

ENTRADA DE PARAMETROS

cons	puntero a la estructura console_t
c	char que contiene el carácter de la secuencia de escape

VARIABLES

src	Dirección origen
dst	Dirección destino
count	Cantidad de palabras a transferir
N	Número de desplazamientos del cursor
value	el primer parámetro de la secuencia ESC[

6.9 Función *beep*.

Produce un sonido en el altavoz, trabajando con el puerto B de el 8255. Las acciones de esta funcion son las siguientes:

1. Si ya esta produciéndose el sonido retornar.
2. Saca los datos para programar el canal dos del timer, en la frecuencia y modo deseado.
3. Inhabilita las interrupciones desde el teclado.
4. Complementa el valor de PORT_B, produciendo el sonido.
5. Habilita las interrupciones.
6. Activa beeping.
7. Actualiza la estructura del mensaje y se la envía a el reloj, para que este se encargue de detener el sonido, llamando a stop_beep.

DEFINE	
BEEP_FREQ	(0x0533) Frecuencia del sonido.
B_TIME	Duración en ticks del sonido.
VARIABLES	
message	mensaje que se le pasa a sendrec.
beeping	Es un flag que indica si se esta beepeando.

6.10 Función *stop_beep*.

Apaga el sonido.

1. Inhabilita las interrupciones desde el teclado.
2. Desactiva beeping.
3. Habilita las interrupciones.
4.

VARIABLES	
Beeping	Es un flag que indica si se esta beepeando.

6.11 Función *src_init*.

Procedimiento que inicializa el driver de pantalla. El algoritmo que sigue es el que se muestra a continuación.

1. Asocia CONSOLE y TTY.
2. Inicializa el driver de teclado.
3. Asignación de las funciones de salida.
4. Obtención de la BIOS los parámetros que nos dicen el registro base I/O.
5. Configuración de los parámetros para la memoria de vídeo dependiendo del tipo de tarjeta gráfica que soporta el equipo.
6. Inicialización de los atributos para cuando se limpia la pantalla
7. Llamada a **select_console()**.
8. Retorno

DEFINICIONES

COLOR_BASE	base de la memoria de vídeo (0xB8000L)
COLOR_SIZE	tamaño de la memoria vídeo 16k (0x4000)
MONO_BASE	base de la memoria de vídeo(0xB0000L)
MONO_SIZE	tamaño de la memoria vídeo 4k (0x1000)
EGA_SIZE	32k (deben soportar al menos 32k las VGA y EGA)
VIDEO_SELECTOR	selector de vídeo
NR_CONS	número de consolas permitidas

STRUCT CONSOLE

c_esc_intro	Carácter siguiente al ESC
c_row	Fila actual
c_esc_paramv	Lista de los parámetros de ESC
c_column	Columna actual
c_cur	Posición actual del cursor en la video RAM
c_org	base en la video RAM para el 6845
c_blank	atributo blank
c_attr	atributo del carácter

VARIABLES GLOBALES

nr_cons	número actual de consolas
vid_port	Puerto I/O para el chip 6845
vid_base	base de la vídeo RAM (0xB000 o 0xB800)
vid_size	tamaño de la vídeo Ram para mono (0x2000=8M para color o 0x800=2M para mono)
vid_mask	máscara 0x1FFF color 0x0800
vid_seg	selector de la RAM de vídeo 0xB0000 ó 0xB8000
src_size	número de caracteres por pantalla (80*25)
page_size	tamaño de página.

STRUCT TTY

tp	puntero a tty_t
tty_devwrite	rutina comienzo de salida del dispositivo actual
tty_echo	rutina de entrada para los caracteres echo
tty_table	vector se puntero a la estructura tty_t

STRUCT CONSOLE

c_start	comienzo en la memoria de vídeo para la consola
c_limit	límite de la consola en la memoria de vídeo
c_org	localización en la RAM del punto base para el 6845
c_attr	atributo del carácter

ENTRADA DE PARAMETROS

tp	puntero a la estructura tty_t

VARIABLES

cons	puntero a una estructura conole_t
vid_base	Dirección base de la memoria de vídeo
bios_ctrbase	Valor del puerto I/O para el 6845
page_size	Determina el tamaño de página para la consola

6.12 Función *putk*.

Imprime un carácter, llama a out_char para imprimir el carácter.

Comentario: Este procedimiento es usado por la versión de printf() que se linka con el propio kernel. El printf de la librería envía un mensaje al Sistema de ficheros el cual no es necesario para escribir dentro del kernel. Simplemente lo encola y comienza la salida.

Algoritmo

```
 si c<>0 entonces
     si c="salto de línea" entoces putk("retorno de carro")
     out_char(c)
sino
   flush()

retornar
```

PARÁMETRO DE ENTRADA

C	Carácter a imprimir

6.13 Función *toggle_scroll*.

Este procedimiento intercambia el tipo de scroll, es decir conmuta entre software y hardware scroll. Su algoritmo es el siguiente:

1. Llamada al procedimeinto **cons_org0().**
2. Cambia el valor de softcroll.
3. Imprime por pantalla el tipo de scrolling activado.

VARIABLES GLOBALES

Softscroll	Si su valor es 1 indica scroll por sofware y 0 scroll hecho por hardware.

6.14 Función *cons_stop*.

Prepara el console para una parada o rearranque. Su algoritmo es el siguiente:

1. Llamada a cons_org0.
2. Habilitamos el permiso para que el scroll no pueda ser realizado por hardware.
3. select_console().g
4. Inicialización del atributo BLANK.
5. Retornar.

DEFINICIONES

BLANK_COLOR	Determina el color del cursor para el atributo blank.

VARIABLES GLOBALES

softscroll	Si su valor es 1 indica scroll por sofware y 0 scroll hecho por hardware.
cons_table	vector de puntero a la estructura console_t y contiene la configuración de cada una de las consoles que podemos disponer.

6.15 Función *select_console*.

Establece a la consola actual el número de consola que contiene el parámetro de entrada. Su algoritmo es:

1. Obtenemos las propiedades para la consola de entrada.
2. Llamada a set_6845 para establecer en sus registros el origen.
3. Llamada a set_6845 para establecer en sus registros la posición donde se encuentra el cursor.
4. Retornar.

DEFINICIONES

VID_ORG	Registro del 6845donde contiene el origen.
CURSOR	Registro d el 6845 que contiene la posición del cursor .

VARIABLES GLOBALES

curcons	Puntero a la estructura console_t actualmente visible.

PARÁMETROS DE ENTRADA

cons_line	número consola

6.16 Función *cons_org0*.

Esta función se encarga simplemente de reorganizar la memoria de video RAM. Para cada una de las consolas activas va recorriendo su memoria asignada en la video RAM y estructura esa zona de manera que la memoria que hay sin usar entra la base y donde está el puntero actual de comienzo de los datos, elimina este bloque y desplaza los datos hasta la posición del bloque que ha sido eliminado.

STRUCT CONSOLE

vid_size	tamaño de la video RAM
scr_size	tamaño de la pantalla (80*25)
cons_table	vector de punteros a las diferentes consolas

VARIABLES GLOBALES

c_start	comienzo en la memoria de vídeo para la consola
c_org	localización en la RAM del punto base para el 6845

6.17 Funciónes *con_loadfont* y *ga_program*.

El procedimiento con_loadfont() es llamado por la subrutina do_ioctl perteneciente a tty. Esta llamada se realiza cuando el usuario quiere trabajar con unos atributos definidos por él, es decir, para cambiar la configuración de los diferentes tipos de caracteres. Esta nueva configuración se encuentra en un area de memoria correspondiente al usuario, load_font() carga estos valores desde la dirección de memoria del usuario al controlador de vídeo para que contenga los nuevos valores que definen la configuración de los caracteres.

PARÁMETROS DE ENTRADA

user_phys	Dirección de memoria donde se encuentran los datos del usuario.

PARÁMETROS DE ENTRADA

seq	Estructura que contiene los parámetros para poder realizar la transferencia al controlador.