INTRODUCCIÓN

El teclado es uno de los dispositivos más conocidos y usados en el mundo de la informática. Su misión es la de interfaz entre el usuario humano y la máquina. Es decir, permitir la comunicación entre ambos.

Para llevar a cabo esta comunicación, el primer paso es la activación de la tecla por parte del usuario. De ello resulta una señal eléctrica que expresa la posición de la tecla. Esta señal se gestiona en el *procesador de teclado*, que se encuentra directamente en él.

El **procesador de teclado** convierte la señal eléctrica de la posición de la tecla en un número de tecla, el *Scan Code* (código de muestreo o de rastreo). Este código de muestreo no tiene ninguna relación con el carácter que hay impreso sobre la tecla ni con la función que desempeña. Sólo representa el número de la tecla. El procesador de teclado pasa el código de muestreo al ordenador, donde es recibido por el controlador (driver) de teclado. Esta transmisión se realiza en serie, ya que el cable de teclado dispone sólo de una línea de datos.

Si se pulsan varias teclas simultáneamente, el procesador de teclado las guarda momentáneamente en un buffer interno que nunca llega a llenarse ya que la transmisión se realiza de forma excesivamente rápida.

El controlador de teclado genera códigos de muestreo tanto para la activación de una tecla como para cuando se suelta. Así el sistema puede saber en todo momento si hay aún una tecla pulsada o si se ha soltado ya. Así se reconoce también la activación simultánea de varias teclas.

Para diferenciar los códigos de muestreo que describen la activación de una tecla y su liberación, se habla de ***Make Code***s (pulsación) y ***Break Codes*** (liberación). El Break Code corresponde al Make Code, con la diferencia de que adicionalmente tiene a 1 el séptimo bit. De esto resultan dos consecuencias importantes.

- Por una parte, los Break Codes son siempre mayores o iguales a 128, y por otra parte el teclado podrá tener como máximo 128 teclas, ya que de otro modo los Make Codes colisionarían con los Break Codes y se generarían confusiones.

- Cada vez que el teclado envía un Make Code o Break Code al ordenador, provoca automáticamente una interrupción de hardware. El controlador de teclado es el que recibe los Make Codes y Break Codes, formando a partir de ellos el código ASCII del carácter introducido, que es suministrado al programa que haya actualmente en ejecución.

Para realizar esto, primero, el controlador de teclado ha de pedirle al teclado el Make o Break Code. Si el controlador de teclado ha reconocido el carácter introducido, ha de convertirlo en un código que el programa de aplicación pueda entender. Los Scan Codes no ayudan en este punto, ya que no todos los teclados trabajan con el mismo juegos de Scan Codes.

Estos códigos se transforman sobre el llamado juegos de caracteres **ASCII**, que se utiliza en todo el mundo de los ordenadores.

El carácter transformado no se transmite directamente al programa que hay actualmente en ejecución, sino que primero se deposita en el ***Buffer de Teclado***.

Este buffer de teclado es una cola circular en la que existen dos punteros para señalar el primer y último carácter almacenado en ella. Cada carácter se almacena en dos bytes.

KEYMAPS

Los teclados IBM PC no generan código ASCII directamente. Las teclas se identifican con un número empezando por las teclas que están localizadas en la parte superior izquierda del teclado, así la tecla "ESC" estaría numerada como 1, la "1" como la 2 y así sucesivamente. De esta forma, cada tecla tiene asignado un número, incluidas las teclas 'modificadoras' como el SHIFT, la cual tiene asignado un número para la tecla SHIFT derecha y otro distinto para la izquierda.

Cuando una tecla es pulsada, MINIX recibe el número asignado a la tecla como un código de rastreo o **scan code**. Este código de rastreo o scan code también es generado cuando se suelta la tecla y su única diferencia con el código de rastreo anterior, es que tiene activado el bit más significativo (lo que equivale a sumar 128 al código de la tecla). De esta forma podemos distinguir de manera sencilla entre teclas pulsadas y soltadas. Manteniendo un registro de las teclas 'modificadoras' que han sido pulsadas y que todavía no se han soltado tendremos una gran cantidad de combinaciones posibles.

Un teclado standard inglés tiene definidas 47 teclas ordinarias (26 alfabéticas, 10 numéricas, y 11 de puntuación), las cuales pueden ser usadas en combinación con las teclas 'modificadoras' (CTRL, ALT, SHIFT...), pero hemos de tener en cuenta que no todos los lenguajes poseen los mismos caracteres y símbolos de puntuación, por lo que muchos sistemas operativos proporcionan la posibilidad de elegir el mapeo entre los códigos de rastreo del teclado y los códigos que se pasarán a la aplicación. Esto es posible gracias al **KEYMAP**, el cual determina que código de carácter se pasa a la aplicación basándose en la tecla que ha sido pulsada y los modificadores activos en ese momento.

Lógicamente el KEYMAP de MINIX es un array de 128 filas por 6 columnas. Las **filas** representan los posibles valores de los códigos de rastreo (Se escogieron 128 filas para acomodar sin problemas a los teclados japoneses que tienen más teclas que los europeos y americanos). Las **columnas** representan: no modificador, tecla SHIFT, tecla CTRL, tecla ALT izquierda, tecla ALT derecha, y la combinación de cualquiera de los ALT más la tecla SHIFT. De este modo hay 720 ((128-6)x6) códigos de carácter que pueden ser generados con este esquema. Esto requiere que cada entrada en la tabla sea una cantidad de 16 bit. A efectos del teclado americano o inglés las columnas correspondientes a las dos teclas ALT son

idénticas, cosa que no ocurre en otros lenguajes en donde la tecla ALT2 es denominada ALTGR y es tratada de forma distinta al ALT1.

Un **keymap** standard definido en la línea *#include keymaps/us-std.src* de **keyboard.c** es compilado en el kernel de MINIX en tiempo de compilación, pero podemos llamar a ***ioctl(0,KIOCSMAP,keymap)*** para cargar un mapa diferente en el kernel en la dirección *keymap*. Un keymap completo ocupa 1536 bytes, y keymaps extra son almacenados en forma comprimida.

Un programa denominado ***genmap*** es usado para hacer un nuevo keymap comprimido. Cuando se compila *genmap* incluye el código de *keymap.src* para un *keymap* particular, por lo que dicho mapa es compilado dentro de *genmap*. Generalmente *genmap* es ejecutado inmediatamente después de ser compilado, y en ese momento pasa la versión comprimida a un archivo, mientras que el *genmap* binario es borrado.

El comando ***loadkeys*** lee un keymap comprimido, lo expande internamente, y luego realiza una llamada a IOCTL para transferir el keymap a la memoria del kernel. MINIX puede ejecutar loadkeys automáticamente nada más empezar, aunque el programa también puede ser invocado por un usuario en cualquier momento.

La figura 1 nos muestra en forma tabular los contenidos de unas pocas líneas del *src/kernel/keymap/us-std.src*.

Scan code	Carácter	Regular	SHIFT	ALT1	ALT2	ALT + SHIFT	CTRL
00	ninguno	0	0	0	0	0	0
01	ESC	C('[')	C('[')	CA('[')	CA('[')	CA('[')	C('[')
02	'1'	'1'	'!'	A('1')	A('1')	A('!')	C('A')
13	'='	'='	'+'	A('=')	A('=')	A('+')	C('@')
16	'q'	L('q')	'Q'	A('q')	A('q')	A('Q')	C('Q')
28	CR/LF	C('M')	C('M')	CA('M')	CA('M')	CA('M')	C('J')
29	CTRL	CTRL	CTRL	CTRL	CTRL	CTRL	CTRL
59	F1	F1	SF1	AF1	AF1	ASF1	CF1
127	???	0	0	0	0	0	0

Figura 1. Ejemplo de entradas del keymap

En esta tabla de la figura 1, se pueden ilustrar varios aspectos de los keymaps:

- No hay ninguna tecla en los teclados IBM-PC que genere un código de rastreo igual a 0.

- Para muchos teclados, europeos y americanos, muchas entradas de la tabla de las últimas posiciones, estarán rellenadas con ceros al no haber tantas teclas para generar todos los posibles códigos. (Pe. La última entrada mostrada en la figura 1).

- Los valores compilados en varias columnas son determinados por macros definidas en *include/minix/keymap.h*:

```
#define C(c)     ((c)& 0x1F)      /* mapea al código de control*/
#define A(c)     ((c)| 0x80)      /*activa octavo bit (ALT)*/
#define CA(c)    A(C(c))          /*CTRL-ALT*/
#define L(c)     ((c)| HASCAPS)   /*añade atributo "el Caps Lock
                                  tiene efecto"*/
```

Las tres primeras macros manipulan bits en el código de los caracteres entre comillas para producir el código que será devuelto a la aplicación. La última activa el bit de HASCAPS en el byte superior de los 16 bits. Este flag nos indica que el estado de la variable **capslock** (que guarda el estado de la tecla Caps Lock) tiene que ser chequeado y el código posiblemente modificado antes de ser retornado.

- Las teclas de función F1, F2. F12, no devuelven valores ASCII ordinarios. En la fila con código de rastreo 59 se muestran simbólicamente los valores que retorna la tecla F1 en combinación con otros modificadores. Estos valores son: F1: 0x0110, SF1: 0x1010, AF1: 0x0810, ASF1: 0x0C10, y CF1: 0x0210.

IMPLEMENTACIÓN DEL DRIVER DE TECLADO

Aunque el sistema operativo ve al teclado y a la consola como parte de un mismo dispositivo, */dev/console*, estos requieren soporte software separado que se encuentra en los archivos **keyboard.c** y consola.c.

Nosotros veremos el software correspondiente al teclado en este apartado, pero no debemos olvidar ciertos aspectos como son:

- Lógicamente el teclado es un sirviente de la consola y la rutina que contiene el código de inicialización del teclado, *kb_init*, es llamada desde *console.c*.

- Si el sistema soporta **consolas virtuales**, el teclado será único, es decir, sólo un teclado es usado para las operaciones de entrada sea cual sea la consola que está activa.

KEYBOARD.C

Keyboard.c comienza como es lógico con varias declaraciones de *#include*, uno de los cuales, *keymap/usr-std.src*, contiene el keymap compilado por defecto.

Posteriormente encontramos macros que nos definen varias constantes. El primer grupo de estas macros es usado en interacciones de bajo nivel con el controlador de teclado. Muchas de ellas son direcciones de puertos de E/S o combinaciones de bits que tienen sentido en estas interacciones. El siguiente grupo incluye nombres simbólicos para teclas especiales.

En las siguientes líneas se definen variables que almacenan el estado de ciertas teclas (shift, alt..) que deben ser recordados para interpretar correctamente la pulsación de una tecla. Estas variables son usadas de distintas formas, por ejemplo, el valor de *capslock*, es activado a verdadero la primera vez que se aprieta dicha tecla, a falso la segunda vez que se presiona y así sucesivamente, mientras que la tecla *shift* se pone a verdadero cuando se aprieta y a falso cuando se suelta.

La estructura *kb_s* es usada para mantener una traza de los códigos de rastreo tal cual van entrando. Dentro de esta estructura los códigos son almacenados en un buffer circular, *ibuf*, del tamaño KB_INT_BYTES. Se declara un array, *kb_lines[NR_CONS]*, de estas estructuras, una por consola, pero de hecho sólo la primera de ellas es usada, ya que la macro *kbaddr* siempre se usa para determinar la dirección del actual *kb-s*. De esta forma si algún fabricante de PC's proporciona soporte hardware para múltiples teclados, Minix estará preparado realizando solamente una modificación en la macro *kdaddr*.

****

map_key0

****

Está definida como una macro. Devuelve el código ASCII que corresponde a un código de rastreo ignorando los modificadores.

****

map_key

****

Realiza el mapeo completo de un código de rastreo, teniendo en cuenta las teclas modificadoras que estén presionadas al mismo tiempo que las teclas ordinarias.

****

kbd_hw_int

****

Es la rutina del servicio de interrupción del teclado, la cual es llamada cada vez que una tecla es presionada o soltada. Realiza una llamada a *scan_keyboard* para obtener el código de rastreo del chip que controla el teclado. El bit más significativo del código de rastreo se activa cada vez que el soltado de una tecla provoca una interrupción, en ese caso la tecla es ignorada a menos que se trate de una tecla modificadora. Si la interrupción es causada por la presión de una tecla, o por la suelta de una tecla modificadora, el código de rastreo es guardado en el buffer circular si este no está lleno, y el flag de *tty->tty_events* de la consola actual es activado. Posteriormente se realiza una llamada a *force_timeout*.

La figura 2 muestra una secuencia de códigos de rastreo que se han ido almacenando en el buffer.

42	35	170	18	38	38	24	57	54	17	182	24	19	38	32
L+	h	L-	e	l	l	o	SP	R+	w	R-	o	r	l	d

Figura 2. Códigos de rastreo en el buffer de entrada, con las correspondientes teclas Presionadas. L+, L-, R+, R-, representan respectivamente, la pulsación y la suelta de las teclas Shift izquierda y derecha.

Cuando ocurre la interrupción del reloj, se despierta a la terminal, *tty_wakeup*, y se inspecciona el flag de *tp->tty_events* para el dispositivo de la consola, si este está activado se produce una llamada a *kb_read*, usando el puntero en el campo *tp->tty_devread*, de la estructura tty de la consola.

****

kb_read

****

Kb_read coge el código de rastreo del buffer circular del teclado y coloca el correspondiente código ASCII en su buffer local, el cual es lo suficientemente grande para almacenar secuencias de escape que deben ser generadas en respuesta a algunos códigos de rastreo provenientes del teclado numérico. Entonces se llama a *in_process* para colocar el carácter en la cola de salida. En esta función (*kb_read*), *lock* y *unlock* son usados para proteger el decremento de la variable *kb->icount* de la aparición de una posible interrupción de teclado al mismo tiempo. La llamada a *make_break* devuelve el código ASCII como un entero. Teclas especiales como las del teclado numérico y teclas de función, tienen valores superiores a 0xFF en este punto. Códigos dentro del rango comprendido entre *HOME* e *INSRT* (0x101 y 0x10C respectivamente, definidos en *include/minix/keymap.h*) resultan de apretar una tecla del teclado numérico, y son convertidos a una secuencia de tres caracteres de escape tal como se muestra en la figura 3 usando el vector *numpad_map*. En este momento, la secuencia es pasada a *in_process*. Códigos mayores que el de *INSRT*, no son pasados a *in_process*, pero se realiza un chequeo para comprobar si los códigos corresponden a *ALT-LEFT-ARROW, ALT-RIGHT-ARROW,* o desde *ALT-F1* hasta *ALT-F12*, en cuyo caso se llama a *select_console* para activar **consolas virtuales**.

Key	Código de rastreo	"ASCII"	Secuencia de Escape
Home	71	0x101	ESC[H
Cursor Arriba	72	0x103	ESC[A
Pg Up	73	0x107	ESC[V
-	74	0x10A	ESC[S
Cursor Izquierda	75	0x105	ESC[D
5	76	0x109	ESC[G
Cursor Derecha	77	0x106	ESC[C
+	78	0x10B	ESC[T
End	79	0x102	ESC[Y
Cursor Abajo	80	0x104	ESC[B
Pg Dn	81	0x108	ESC[U
Ins	82	0x10C	ESC[@

Figura 3. Códigos de escape generados por el teclado numérico. Cuando un código de rastreo de una tecla ordinaria es traducido a código ASCII, las teclas especiales se le asigna una pseudo código ASCII con valores mayores que 0xFF.

****

make_break

****

Make_break convierte los códigos de rastreo a códigos ASCII y actualiza las variables que guardan la traza del estado de las teclas modificadoras. Sin embargo, primero se chequea para comprobar la combinación *CTRL-ALT-DEL* que, como todo el mundo sabe, fuerza el reinicio de la máquina bajo *MS-DOS*. En ese momento MINIX, envía la señal *SIGABRT* a *init*, que es el proceso padre de todos los procesos. *Init* espera capturar esta señal y la interpreta como un comando de inicio ordenado de caída del sistema, primero causando el retorno al arranque del monitor, desde el cual se producirá una restauración del sistema o puede ser ordenado un reinicio de MINIX. Por supuesto, no es realista suponer que esto ocurra siempre. La mayoría de usuarios entiende lo peligroso que supone una caída repentina del sistema y no presionan *CTRL-ALT-DEL* hasta que algo va realmente mal y el control normal del sistema se vuelve imposible. En este punto es probable que el sistema esté tan destrozado que ordenar el envío de una señal a otro proceso puede ser imposible. Es por esto por lo porque existe una variable estática *CAD_count* in *make_break.* La mayoría de los sistemas que se han venido abajo permiten que el sistema de interrupciones continúe funcionando, así la entrada de teclado puede continuar recibiendo datos. Aquí MINIX toma ventaja sobre el proceder esperado de los usuarios de ordenadores, quienes muy probablemente apretarán las teclas

repetidamente cuando algo no parece trabajar correctamente. Si el intento de enviar la señal *SIGABRT* a *init* falla y el usuario presiona *CTRL-ALT-DEL* dos veces más, se produce directamente una llamada a *wreboot*, causando el retorno al monitor sin realizar la llamada a *init*.

La variable *make* recuerda si el código de rastreo fue generado por la presión o soltado de una tecla, y entonces llama a *map_key* devolviendo el código ASCII sobre la variable *ch*. Seguidamente se entra en una estructura *SWITCH* evaluada sobre *ch*. En este punto se distinguen dos casos, el que sea una tecla ordinaria y de una tecla especial. Para el caso de una tecla ordinaria, ninguno de los *switch* se activará ni siquiera el caso *default*, puesto que se supone que los códigos de las teclas ordinarias sólo son aceptados en la fase de presionado de la tecla. Si de algún modo se acepta el código de una tecla ordinaria cuando es soltada, aquí se asigna el valor -1 a *ch* y esto provoca que sea ignorada por el llamador, *kb_read* al creer que se trata de una tecla especial. Una tecla especial, por ejemplo *CTRL*, se identifica en la parte apropiada del *SWITCH*. La variable correspondiente, *control* en este caso, toma el valor de *make* y asigna el valor –1 a *ch* que será devuelto al llamador (*kb_read*) e ignorado. Para el caso de las teclas *CALOCK, NLOCK* y *SLOCK* es algo más complicado pero el efecto es similar. La diferencia radica en que estas teclas una vez pulsadas cambian su estado y no vuelven a modificarlos hasta que son pulsadas de nuevos, a diferencia de las demás que recuperan su estado inicial una vez soltadas. Por ello se comprueba su estado y se invierte, produciendo también una llamada a *set_leds* encargada de encender o apagar los ‘leds’ del teclado.

Existe un caso más a considerar, que es el del código *EXTKEY* y la variable *esc*. Esta no debe ser confundida con la tecla *ESC* del teclado la cual retorna el código ASCII 0x1B. No existe ninguna forma de generar el código de *EXTKEY* mediante la presión de una tecla o combinación de ellas; es el prefijo de tecla extendida del teclado de PC. El primero de los dos bytes del código de rastreo indica que una tecla que no formaba parte de las teclas del complemento del PC original pero que tiene el mismo código de rastreo que otra que si lo era, ha sido pulsada. En muchos casos, el software trata las dos teclas de forma idéntica. Por ejemplo, es el típico caso de la tecla normal “/” y la tecla “/” del teclado numérico. En otros casos, se desearía distinguir entre las dos teclas. Por ejemplo, muchos teclados abogan por otros lenguajes distintos al inglés que traten el *ATL* derecho e izquierdo de manera deferente, soportando teclas que deben generar tres códigos de caracteres diferentes. Ambas teclas *ATL* generan el mismo código de rastreo (56), pero el código *EXTKEY* precede a éste cuando es presionado el *ALT* derecho. Cuando el código *EXTKEY* es

devuelto, el flag *esc* es activado. En este caso, *make_break* retorna desde dentro de la estructura *SWITCH*, sin pasar por el retorno normal del final de la función en el que *esc* es puesto a cero para todos los casos. Esto tiene el efecto de hacer *esc* efectivo sólo para el siguiente código recibido, ya que al tratar el siguiente código, *esc* será puesto de nuevo a cero.

****

set_leds

****

Enciende o apaga las luces del teclado que indican si las teclas *Num Lock*, *Caps Lock*, o *Scroll Lock* han sido pulsadas. Un byte de control, *LED_CODE*, se escribe en un puerto de salida para indicar al teclado que el siguiente byte escrito en ese puerto es para el control de las luces, y que el estado de las tres luces lo encontrará en los tres bits menos significativos del siguiente byte. Para ello hace uso de dos funciones *kb_wait* que le indicará cuando el teclado está listo para recibir una secuencia de comandos y *kb_ack* que le indicará si el comando ha sido reconocido. Ambas funciones actúan en espera activa puesto que están leyendo continuamente hasta que ven un código. Esta no es una técnica recomendable para la mayoría de los manejadores de entrada/salida pero debido a que el encendido y apagado de las luces no es algo que se realice con mucha frecuencia, hacerlo de esta forma no gasta demasiado tiempo. Notar también, que ambas funciones pueden fallar y se podría determinar desde el código de retorno que pasó. Pero encender las luces en el teclado no merece la suficiente importancia para chequear el valor devuelto por las funciones.

****

kb-wait

****

Indicará cuando el teclado está listo para recibir una secuencia de comandos. Espera a que el buffer esté vacío.

****

kb-ack

****

Verificará si el comando ha sido reconocido.

****

kb-init

****

Desde que el teclado es parte de la consola, la inicialización de la rutina *kb_init* es llamada desde *scr_init* en **console.c** y no directamente desde *tty_init* en **tty.c**. Si existen consolas virtuales activas (*NR_CONS* en *include/minix/config.h* es mayor que 1), *kb_init* es llamada una vez para cada una de las consolas lógicas. Después de la primera llamada, la única parte de *kb_init* que es esencial para las demás consolas es la de cargar la dirección de *kb_read* en *tp->tty_devread*, pero no produce ningún problema el repetir el resto de la función. El resto de la función inicializa algunas variables, como son los punteros de la cola, enciende los leds del teclado y rastrea el teclado para estar seguro de no haber dejado alguna tecla pulsada sin leer. Cuando todo esta listo, se llama a *put_irq_handler* que activa el manejador de interrupciones y después a *enable_irq* que guarda toda la inicialización, así que *kbd_hw_int* será ejecutada siempre que una tecla sea pulsada o soltada.

****

kbd-loadmap

****

Carga el nuevo *keymap* del usuario sobreescribiendo el cargado por defecto, mediante la llamada a *phys_copy.*

****

func_key

****

Función llamada desde *kb_read* para ver si una tecla con un significado especial para el proceso actual ha sido pulsada. La figura 4 muestra estas teclas y sus efectos. El código llamado se encuentra en varios ficheros. Los códigos de *F1* y *F2* activan el código *p_dmp* y *map_dmp* respectivamente en *dmp.c* . El código de *F3* activan el código *toggle_scroll* en *console.c*. Los códigos *CF7, CF8* y *CF9* provocan llamadas a *sigchar,* en *tty.c*. Si existe una red conectada a MINIX, hay un caso más, el código de *F5*, que es añadido para mostrar las estadísticas Ethernet.

****

scan_keyboard

****

Trae un carácter desde el hardware de teclado y lo reconoce. Trabaja a nivel de interfaz hardware, leyendo y escribiendo bytes en los puertos de entrada/salida. El controlador de teclado es informado de que un carácter ha sido leído, el cual lee un byte, lo escribe otra vez pero con el bit más significativo a 1, y lo reescribe con el mismo bit puesto a 0. Esto previene que se lea el mismo dato en una lectura posterior. No existe un estado de chequeo de teclado, pero no habría problemas en ningún caso, ya que *scan_keyboard* sólo es llamado en respuesta a una interrupción con la excepción de la llamada hecha desde *kb_init* para eliminar la "basura".

Key	Propósito
F1	Muestra la tabla de procesos
F2	Muestra detalles del uso de memoria del proceso
F3	Cambia entre scroll software y hardware
F5	Muestra estadísticas Etherner
CF7	Envía SINGQUIT, mismo efecto que CTRL-\
CF8	Envía SINGINT, mismo efecto que DEL
CF9	Envía SIGKILL, mismo efecto que CTRL-U

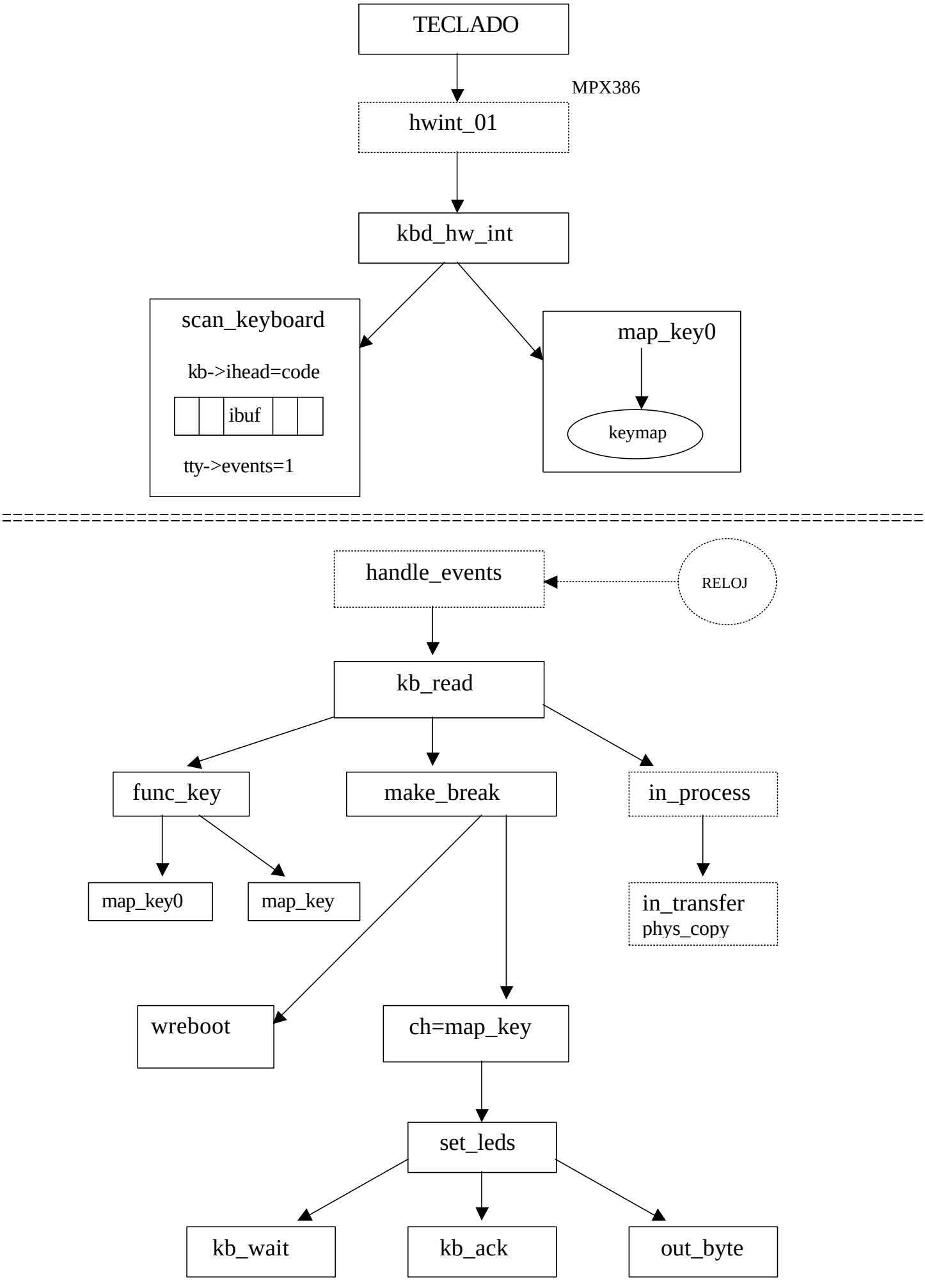
Figura 4. Las teclas de función detectadas por *func_key.*

****

wreboot

****

Si es invocado como resultado de un estado de "pánico" (se han pulsado repetidamente las teclas *CRTL-ALT-DEL*), ofrece la oportunidad de que el usuario use las teclas de función para mostrar por pantalla una "debbug" de la información. Esta función usa también, la espera activa. El teclado es leído continuamente hasta que un *ESC* es presionado. Ciertamente, nadie puede reclamar una técnica más eficiente después de la caída de un sistema mientras se espera por un comando de reinicio. Dentro del bucle , *func_key*, es llamada para dar la posibilidad de obtener información para ayudar a analizar la causa que produjo la caída del sistema.


TECLADO
MPX386
hwint_01
kbd_hw_int
scan_keyboard
kb->ihead=code
ibuf
tty->events=1
map_key0
keymap
handle_events
RELOJ
kb_read
func_key
make_break
in_process
map_key0
map_key
in_transfer
phys_copy
wreboot
ch=map_key
set_leds
kb_wait
kb_ack
out_byte

CUESTIONES

1.- ¿ Cuál es la estructura y la utilidad de un **keymap** ?

2.- ¿Cuáles son esquemáticamente los pasos que se siguen desde que una tecla es pulsada , desde el punto de vista del driver de teclado?.

3. ¿ Qué se entiende por Scan Code? ¿Es lo mismo que código ASCII?
