Trabajo Práctico Especial de Sistemas Operativos

FLEDS

Abramowicz, Pablo Federico
Goñi, Juan Ignacio
Pan, Matias Santiago

Instituto Tecnológico de Buenos Aires

Índice

Resumen

Este informe técnico presenta la implementación de un dispositido de leds y su respectivo controlador para el sistema operativo Minix 2,0,0. Se incluye las descripciones técnicas del dispositivo y las modificaciones realizadas a los códigos fuente para la vinculación con el controlador.
Keywords: *fleds, driver, hardware, interrupciones de hardware, IRQ, handler, serial.*

1. Introducción

Este proyecto surgió debido a la consigna de la cátedra de agregarle alguna funcionalidad a Minix 2,0,0. El primer proyecto que se nos ocurrió fue vincular Minix con GPS mediante el puerto serial, pero el hecho de que Minix no contase con un X imposibilitaba visualizar el mapa de forma coherente, agregando que la complejidad del dispositivo de GPS era parsear la telemetría entregaba. Luego de discutir, surgió la idea de armar un *cartel del leds*, de ahora en más **Fleds**, y a continuación se pasa a detallar los detalles del mismo.

2. Implementación

En principio se pensó en realizar la comunicación entre Minix y el **Fleds** compuesta por 2 capas: una donde el usuario pueda comunicarse y mandarle los comandos, y otra donde el driver se comunique con el fleds por medio del puerto serial. La misma sería:

Usuario → driver fleds → driver serial → fleds hardware

Luego de investigar y revisar el controlador provisto por minix del serial (`/usr/src/kernel/RS232c`) se llegó a la conclución de que era prácticamente inutilizable debido a la estrecha vinculación del mismo con la terminal (`/usr/src/kernel/ttyc`), a lo cual se procedió a realizar un único controlador que haga de ambos.

A su vez, se pensó en dejar la lógica del controlador del lado del usuario por un tema de espacio en la ROM del Hardware, pero satisfactoriamente se pudo incluir ciertas lógicas dentro del Hardware.

También se generó una librería de animaciones para una fácil utilización del mismo.

3. Hardware

Fleds consta de 128 leds rgb, con la capacidad de encenderse con los 3 colores primarios luz, rojo, azul y verde, independientemente. Consta de 16 columnas por 8 filas. Se utilizó un decodificador para la selección de la columna, 3 latches para el dibujado de los colores y un microcontrolador para el refresco de la pantalla. En la figura 1 se muestra el diagrama del circuito.

3.1. Principio de funcionamiento

En el microcontrolador tiene cargado el firmware, el cual se encarga de mantener la comunicación con el driver de Minix vía puerto serial. Cuando recibe un byte, se genera una

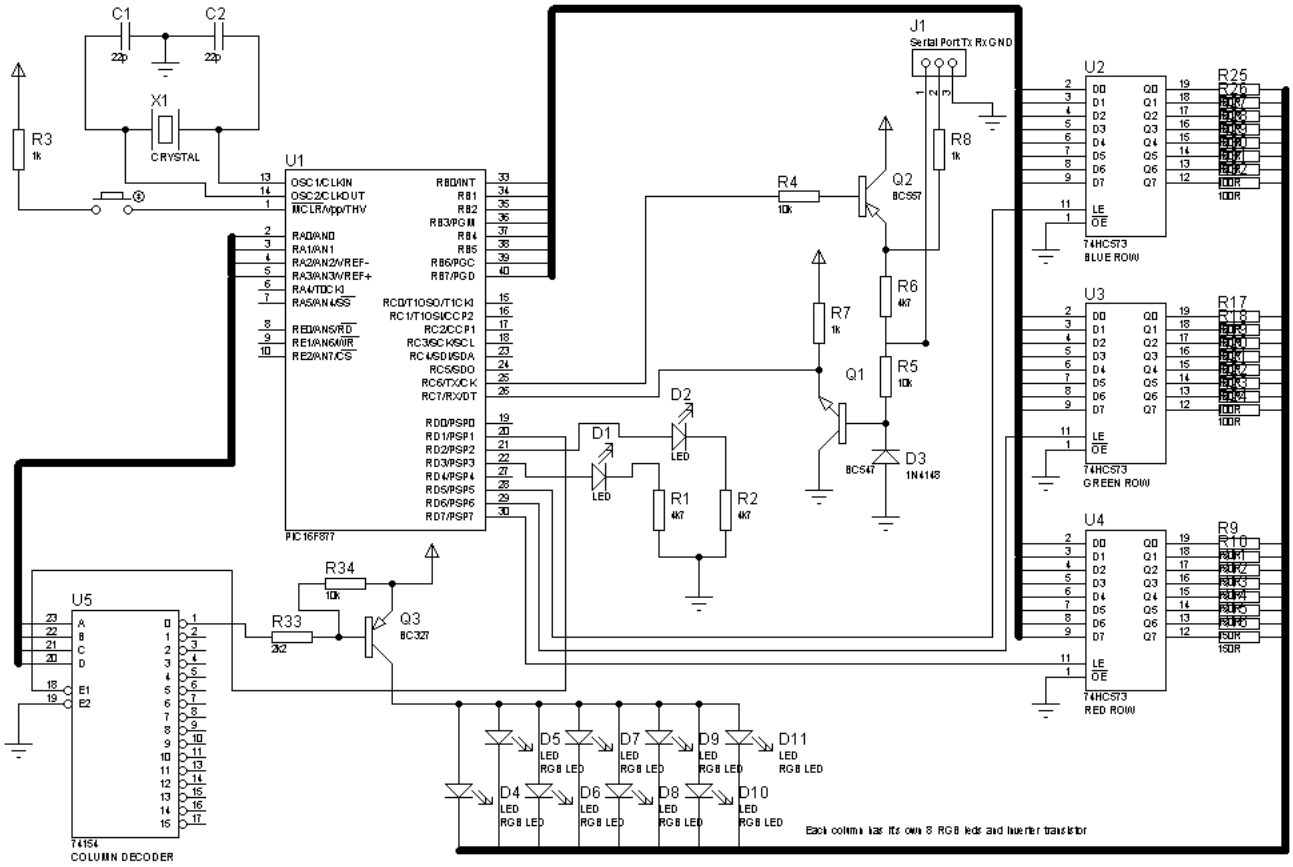

Figura 1: Diseño del circuito eléctrico de FLEDS

interrupción, la cual analiza según el protocolo el comando recibido y lo ejecuta. En estado normal, mantiene el refresco de la pantalla. Primero, le indica al decodificador que debe habilitar la primer columna, luego, se setean en cada latch, la columna del rojo, verde y azul. Se espera un milisegundo, y se proceden a dibujar las siguientes columnas, hasta llegar a la última, donde vuelve a comenzar el ciclo. El refresco es de 60Hz aproximadamente.

3.2. Protocolo

Se enumeran los comandos implementados en el firmware:

0x00: Clear screen. Borra la pantalla

0x40 al 0x4F: Set column. Establece la combinación de colores para la columna del nibble menos significativo (0 a 15). Recibe por parámetros los 3 bytes para el rojo, verde y azul, respectivamente.

0x10 al 0x19: Scroll. Realiza un scroll en base al valor del nibble menos significativo. 0 para ScrollDownCarry, 1 para ScrollDown, 2 para ScrollUpCarry, 3 para ScrollUp, 4 pa-

ra ScrollLeftCarry, 5 para ScrollLeft, 6 para ScrollRightCarry, 7 para ScrollRight, 8 para StartScroll y 9 para StopScroll.

0x08 al 0x8F: Set UART speed. Establece la velocidad de la UART en base al nibble menos significativo. 0 para 1200, 1 para 2400, 2 para 4800, 3 para 9600, 4 para 19200, 5 para 38400, 6 para 57600 y 7 para 115200 baudios.

0x20 al 0x2F: Set scroll speed. Establece la frecuencia para el scroll.

0x5A: Get screen. Transmite por el serial, el contenido actual de la pantalla, enviando 96 caracteres encerrados entre [y], para su mejor lectura.

4. Controlador

La implementación del driver del fleds involucró la modificación de códigos fuentes del kernel de Minix y el agregado de nuevos archivos. Se detalla a continuación las modificaciones correspondientes con sus respectivas explicaciones. *Aclaración: Los paths que encabezan las siguientes subsecciones son relativos a* usr*., y representan la locación del archivo en el file-system.*

4.1. include/minix/com.h

Este archivo contiene los números de tareas que involucra el kernel, a lo cuál se procedió a cambiar el número de tarea de la **TTY** como último y agregarle la tarea del **Fleds** para que el kernel la pueda manejar.

com-min.h

```
1 ...
2 #define TTY                (DL_ETH - 2)
3 /* FLEDS: agrego el pid del fleds */
4 #define FLEDS               (DL_ETH - 1)
5 ...                               /* terminal I/O class */
6
```

4.2. include/minix/config.h

En este archivo se definen configuraciones que se van a utilizar. Como se está agregando un nuevo dispositivo, se agrega la definición del **ENABLE** del mismo.

config-min1.h

```
1 ...
2 /* FLEDS: incluyo el enabale del fleds */
3 #define ENABLE_FLEDS           1
4 /* Include or exclude device drivers.  Set to 1 to include, 0 to exclude. */
5 #define ENABLE_NETWORKING  0        /* enable TCP/IP code */
6 #define ENABLE_AT_WINI     1        /* enable AT winchester driver */
7 #define ENABLE_BIOS_WINI   1        /* enable BIOS winchester driver */
```

```
#define ENABLE_ESDI_WINI   1        /* enable ESDI winchester driver */
#define ENABLE_XT_WINI     0        /* enable XT winchester driver */
#define ENABLE_ADAPTEC_SCSI 1        /* enable ADAPTEC SCSI driver */
#define ENABLE_MITSUMI_CDROM 0        /* enable Mitsumi CD-ROM driver */
#define ENABLE_SB_AUDIO    0        /* enable Soundblaster audio driver */
...
```

También se procedió a deshabilitar el serial utilizado por la TTY poniendo `NR_RS_LINES` en 0.

config-min2.h

```
...
/* NR_CONS, NR_RS_LINES, and NR_PTYS determine the number of terminals the
 * system can handle.
 */
#define NR_CONS            2        /* # system consoles (1 to 8) */
#define        NR_RS_LINES           0        /* # rs232 terminals (0, 1, or 2) */
#define        NR_PTYS                   0        /* # pseudo terminals (0 to 64) */
...
```

4.3. include/minix/const.h

El `NR_TASK` contiene la cantidad de tareas que corre inicialmeten el kernel, a lo cual, como se está agregando un controlador y se desea que esté activado todo el tiempo, agregamos la tarea

const-min.h

```
...
/* Number of tasks. */
#define NR_TASKS        (9 + ENABLE_WINI + ENABLE_SCSI + ENABLE_CDROM \
                        + ENABLE_NETWORKING + 2 * ENABLE_AUDIO + ENABLE_FLEDS)
...
```

4.4. src/kernel/table.c

En principio se tiene que agregar el tamaño de stack correspondiente al controlador. En `FLEDS_STACK` se define el mismo como un `SMALL_STACK`

table-min.c

```
...
#define SMALL_STACK        (128 * sizeof(char *))

#define        TTY_STACK        (3 * SMALL_STACK)
/* FLEDS: Agrego el stack del fleds */
#define FLEDS_STACK        (3 * SMALL_STACK)
...
```

También se tiene que agregar el mismo al tamaño total de stack definido en `TOT_STACK_SPACE`.

table-min2.c

```
...
/* FLEDS: Incluyo el stack al total */
#define        TOT_STACK_SPACE                (TTY_STACK + DP8390_STACK + SCSI_STACK + \
```

```
4         SYN_ALRM_STACK + IDLE_STACK + HARDWARE_STACK + PRINTER_STACK + \
5         WINCH_STACK + FLOP_STACK + MEM_STACK + CLOCK_STACK + SYS_STACK + \
6         CDROM_STACK + AUDIO_STACK + MIXER_STACK + FLEDS_STACK)
7 ...
```

El **struct tasktab tasktab**[] contiene el punto de entrada de las tareas que se ejecutan, a lo cual se debe agregar la tarea del controlador del **Fleds**. Para agregar una entrada se debe especificar primero la función encargada de atender los pedidos al controlador, en nuestro caso `fleds_task`, luego se define el stack correspondiente de la tarea, en nuestro caso `FLEDS_STACK`, y por último se define el nombre de la tarea a mostrar.

table-min3.c

```
1  ...
2  PUBLIC struct tasktab tasktab[] = {
3          { tty_task,                TTY_STACK,        "TTY"               },
4  /* FLEDS: Agrego la tarea al scheduler */
5  #if ENABLE_FLEDS
6          { fleds_task,               FLEDS_STACK,        "FLEDS"               },
7  #endif
8  #if ENABLE_NETWORKING
9          { dp8390_task,               DP8390_STACK,        "DP8390"        },
10 #endif
11 #if ENABLE_CDROM
12         { cdrom_task,               CDROM_STACK,        "CDROM"               },
13 #endif
14 #if ENABLE_AUDIO
15         { audio_task,               AUDIO_STACK,        "AUDIO"               },
16         { mixer_task,               MIXER_STACK,        "MIXER"               },
17 #endif
18 #if ENABLE_SCSI
19         { scsi_task,               SCSI_STACK,        "SCSI"               },
20 #endif
21 #if ENABLE_WINI
22         { winchester_task,        WINCH_STACK,        "WINCH"               },
23 #endif
24         { syn_alrm_task,        SYN_ALRM_STACK, "SYN_AL"        },
25         { idle_task,               IDLE_STACK,        "IDLE"               },
26         { printer_task,               PRINTER_STACK,        "PRINTER"        },
27         { floppy_task,               FLOP_STACK,        "FLOPPY"        },
28         { mem_task,               MEM_STACK,        "MEMORY"        },
29         { clock_task,               CLOCK_STACK,        "CLOCK"               },
30         { sys_task,               SYS_STACK,        "SYS"               },
31         { 0,                        HARDWARE_STACK,        "HARDWAR"        },
32         { 0,                        0,                "MM"                },
33         { 0,                        0,                "FS"                },
34 #if ENABLE_NETWORKING
35         { 0,                        0,                "INET"                },
36 #endif
37         { 0,                        0,                "INIT"                },
38 };
39 ...
```

Es importante que se agregue en conconrdancia al número de tarea incluido en `/include/minix/com`h. Entonces, como se incluyó en segundo lugar, aquí se hace de la misma manera.

4.5. src/fs/table.c

La tabla `dmap` determina la vinculación entre el número principal y la tarea que recibe el mensaje. En la misma se agregó en una posición libre, la 9, la tarea de Fleds.

fs-table-min.c

```
...
PUBLIC struct dmap dmap[] = {
/*  ?   Open       Read/Write   Close       Task #      Device  File
    -   ----       ----------   -----       -------     ------  ----       */
  DT(1, no_dev,    no_dev,      no_dev,     0)           /* 0 = not used   */
  DT(1, dev_opcl,  call_task,   dev_opcl,   MEM)         /* 1 = /dev/mem   */
  DT(1, dev_opcl,  call_task,   dev_opcl,   FLOPPY)      /* 2 = /dev/fd0   */
  DT(ENABLE_WINI,
        dev_opcl,  call_task,   dev_opcl,   WINCHESTER)  /* 3 = /dev/hd0   */
  DT(1, tty_open,  call_task,   dev_opcl,   TTY)         /* 4 = /dev/tty00 */
  DT(1, ctty_open, call_ctty,   ctty_close, TTY)         /* 5 = /dev/tty   */
  DT(1, dev_opcl,  call_task,   dev_opcl,    PRINTER)     /* 6 = /dev/lp    */

#if (MACHINE == IBM_PC)
  DT(ENABLE_NETWORKING,
        net_open,  call_task,   dev_opcl,   INET_PROC_NR)/* 7 = /dev/ip    */
  DT(ENABLE_CDROM,
        dev_opcl,  call_task,   dev_opcl,   CDROM)       /* 8 = /dev/cd0   */
/* FLEDS: agrego el dispositivo del fleds al mapeo */
  DT(1, dev_opcl,  call_task,   dev_opcl,   FLEDS)           /* 9 = /dev/fleds   */
  DT(ENABLE_SCSI,
        dev_opcl,  call_task,   dev_opcl,   SCSI)        /*10 = /dev/sd0   */
  DT(0, 0,         0,           0,          0)           /*11 = not used   */
  DT(0, 0,         0,           0,          0)           /*12 = not used   */
  DT(ENABLE_AUDIO,
        dev_opcl,  call_task,   dev_opcl,   AUDIO)       /*13 = /dev/audio */
  DT(ENABLE_AUDIO,
        dev_opcl,  call_task,   dev_opcl,   MIXER)       /*14 = /dev/mixer */
#endif /* IBM_PC */

#if (MACHINE == ATARI)
  DT(ENABLE_SCSI,
        dev_opcl,  call_task,   dev_opcl,   SCSI)        /* 7 = /dev/hdscsi0 */
#endif
};
...
```

4.6. src/kernel/fleds.c

Este archivo contiene realmente el controlador del dispositivo. Contiene una única función pública `fleds_task()` que es el proceso controlador y la comunicación con el mismo se hace mediante las primitivas `open`, `close`, `read` y `write`. Inicialmente, cuando arranca el controlador inicializa el serial debido que ya no lo hace más la tty. En este proceso se involucra los seteos de la velocidad, paridad, interrupción, buffers internos, etc. Luego la tarea queda esperando a recibir algún mensaje. Una de las complejidades surgidas fue el tema de los direccionamientos que utiliza minix en modo kernel y en modo usuario y el requerimento de copiar los datos entre los distintos segmentos (en el caso de `write` y `read`).

fleds.c

```
1  #include "kernel.h"
2  #include <termios.h>
3  #include <sys/ioctl.h>
4  #include <minix/callnr.h>
5  #include <minix/com.h>
6  #include <sys/types.h>
7  #include "proc.h"
8  #include "fleds.h"
9
10 FORWARD _PROTOTYPE( void send_serial, (int data)                );
11 FORWARD _PROTOTYPE( void do_open, (void) );
12 FORWARD _PROTOTYPE( void do_close, (void) );
13 FORWARD _PROTOTYPE( void do_write, (message *m_ptr) );
14 FORWARD _PROTOTYPE( void do_read, (message *m_ptr) );
15 FORWARD _PROTOTYPE( void fleds_init, (void) );
16 FORWARD _PROTOTYPE( void reply, (int code, int replyee, int process,
17                 int status) );
18
19 FORWARD _PROTOTYPE( int setParity, (int parity, int odd) );
20 FORWARD _PROTOTYPE( int setInterrupt, (int on) );
21 FORWARD _PROTOTYPE( int setMCR, (void) );
22 FORWARD _PROTOTYPE( int setbps, (serialSpeedT speed) );
23 FORWARD _PROTOTYPE( int setStop, (int stopBit)  );
24 FORWARD _PROTOTYPE( int setBitCount, (int bitCount) );
25 FORWARD _PROTOTYPE( int doReset, (void) );
26 FORWARD _PROTOTYPE( int fleds_handler, (int irq) );
27 FORWARD _PROTOTYPE( int add_buffer,(int data) );
28 FORWARD _PROTOTYPE( char get_buffer,(void) );
29 FORWARD _PROTOTYPE( int bufferNotFull, (void) );
30 FORWARD _PROTOTYPE( int bufferNotEmpty, (void) );
31
32 typedef struct {
33         unsigned char LCR;
34         unsigned char DLL;
35         unsigned char DLM;
36         unsigned char MCR;
37         unsigned char IER;
38 } serialStateT;
39
40 PRIVATE serialStateT serialState;
41
42 PRIVATE int initialized = 0;
43
44 PRIVATE int commBase = COMM1_BASE;
45
46 typedef struct
47 {
48         int buffer[FLEDS_BUFFER_SIZE];
49         int write;
50         int read;
51         int qty;
52 }fledsBufferT;
53
54 fledsBufferT k_buffer;
55
56 PRIVATE int bufferNotEmpty () {
57         return (k_buffer.qty != 0);
58 }
59
```

```
60  PRIVATE int bufferNotFull () {
61          return (k_buffer.qty < FLEDS_BUFFER_SIZE);
62  }
63
64
65  PUBLIC void fleds_task()
66  {
67          message mess;
68          int status;
69
70          /* inicializo el fleds */
71          fleds_init();
72
73          while(1)
74          {
75                  receive(ANY, &mess);
76                  switch(mess.m_type)
77                  {
78                          case DEV_OPEN:
79                                  do_open();
80                                  reply(TASK_REPLY, mess.m_source, mess.PROC_NR, OK);
81                                  break;
82                          case DEV_WRITE:
83                                  do_write(&mess);
84                                  break;
85                          case DEV_READ:
86                                  do_read(&mess);
87                                  break;
88                          case DEV_CLOSE:
89                                  do_close();
90                                  reply(TASK_REPLY, mess.m_source, mess.PROC_NR, OK);
91                                  break;
92                          case DEV_IOCTL:
93                                  reply(TASK_REPLY, mess.m_source, mess.PROC_NR, OK);
94                                  break;
95                          case HARD_INT:                                  break;
96                  }
97
98          }
99
100 }
101
102 /*===========================================================================*
103  *                                do_write                                   *
104  *===========================================================================*/
105
106 PRIVATE void do_write(m_ptr)
107 register message *m_ptr;        /* pointer to the newly arrived message */
108 {
109
110         register int r;
111         int i, count=0;
112         char obuf[FLEDS_BUFFER_SIZE];
113         phys_bytes user_phys;
114
115         /* Reject command if last write is not finished, count not positive, or
116         * user address bad.
117         */
118
119         if (m_ptr->COUNT <= 0)
```

```
120         {
121                 r = EINVAL;
122         }
123         else
124                 if (numap(m_ptr->PROC_NR, (vir_bytes) m_ptr->ADDRESS, m_ptr->COUNT) == 0)
125                 {
126                         r = EFAULT;
127                 }
128                 else
129                 {
130                         #if FLEDS_PING_ENABLE
131                         disable_irq(RS232_IRQ);
132                         #endif
133                         /* cantidad de bytes a copiar */
134                         count = m_ptr->COUNT;
135                         /* obtengo  la direccion fisica del puntero enviado */
136                         user_phys = proc_vir2phys(proc_addr(m_ptr->PROC_NR), (vir_bytes)
    m_ptr->ADDRESS);
137                         /* copio los datos a una posicion local */
138                         phys_copy(user_phys, vir2phys(obuf), (phys_bytes) count);
139                         /* envio al serial */
140                         for (i=0; i < count; i++)
141                                 send_serial(obuf[i]);
142                         #if FLEDS_PING_ENABLE
143                         enable_irq(RS232_IRQ);
144                         #endif
145                         r = OK;
146                 }
147
148         reply(TASK_REPLY, m_ptr->m_source, m_ptr->PROC_NR,r);
149 }
150
151
152 /*===========================================================================*
153  *                                do_read                                    *
154  *===========================================================================*/
155
156 PRIVATE void do_read(m_ptr)
157 register message *m_ptr;        /* pointer to the newly arrived message */
158 {
159
160         register int r;
161         int i, count=0;
162         char orBuf[FLEDS_BUFFER_SIZE];
163         phys_bytes user_phys;
164
165         /* Reject command if last write is not finished, count not positive, or
166         * user address bad.
167         */
168
169         if (m_ptr->COUNT <= 0)
170         {
171                 r = EINVAL;
172         }
173         else
174                 if (numap(m_ptr->PROC_NR, (vir_bytes) m_ptr->ADDRESS, m_ptr->COUNT) == 0)
175                 {
176                         r = EFAULT;
177                 }
178                 else
```

```
179                 {
180                         #if FLEDS_PING_ENABLE
181                         disable_irq(RS232_IRQ);
182                         #endif
183                         /* cantidad de bytes a copiar */
184                         count = m_ptr->COUNT;
185                         /* obtengo  la direccion fisica del puntero enviado */
186                         user_phys = proc_vir2phys(proc_addr(m_ptr->PROC_NR), (vir_bytes)
m_ptr->ADDRESS);
187                         /* copio  los datos del buffer a una variable local (por que el
buffer es ciclico */
188                         for (i=0; i < count; i++)
189                                 orBuf[i]=get_buffer();
190                         /* copio los datos a la posicion fisica del usuario */
191                         phys_copy(vir2phys(orBuf), user_phys, (phys_bytes) count);
192                         #if FLEDS_PING_ENABLE
193                         enable_irq(RS232_IRQ);
194                         #endif
195                         r = OK;
196                 }
197
198         reply(TASK_REPLY, m_ptr->m_source, m_ptr->PROC_NR,r);
199 }
200
201
202 /*==========================================================================*
203  *                                setInterrupt                                *
204  *==========================================================================*/
205
206 PRIVATE int setInterrupt(on)
207 int on;
208 {
209         unsigned char ie = 0;
210
211         if (on)
212                 ie = 0x01;
213
214         out_byte(commBase + 3, serialState.LCR & 0x7F);
215         out_byte(commBase + 1, ie);
216         out_byte(commBase + 3, serialState.LCR);
217
218         return 1;
219 }
220 /*==========================================================================*
221  *                                setParity                                 *
222  *==========================================================================*/
223
224 PRIVATE int setParity(parity, odd)
225 int parity;
226 int odd;
227 {
228         unsigned char p = 0;
229
230         if (parity)
231         {
232                 p |= 0x08;
233                 if (!odd)
234                         p |= 0x10;
235         }
236
```

```
237         serialState.LCR = (serialState.LCR & 0xE7) | p;
238
239         out_byte(commBase + 3, serialState.LCR);
240
241         return 1;
242 }
243
244 /*=========================================================================*
245  *                                setMCR                                   *
246  *=========================================================================*/
247
248 PRIVATE int setMCR()
249 {
250         out_byte(commBase + 4, 0x08);
251         return 1;
252 }
253
254 /*=========================================================================*
255  *                                setbps                                   *
256  *=========================================================================*/
257
258 PRIVATE int setbps(speed)
259 serialSpeedT speed;
260 {
261         char speedReg[][2] = { {0x00, 0x09}, {0x80, 0x01}, {0x60, 0x00}, {0x30, 0x00},
262         {0x18, 0x00}, {0x0C, 0x00}, {0x06, 0x00}, {0x03, 0x00}, {0x02, 0x00}, {0x01, 0x00}};
263
264         out_byte(commBase + 3, serialState.LCR | 0x80);
265         out_byte(commBase + 0, serialState.DLL = speedReg[speed][0]);
266         out_byte(commBase + 1, serialState.DLM = speedReg[speed][1]);
267         out_byte(commBase + 3, serialState.LCR);
268
269         return 1;
270 }
271
272 /*=========================================================================*
273  *                                setStop                                  *
274  *=========================================================================*/
275
276 PRIVATE int setStop(stopBit)
277 int stopBit;
278 {
279         unsigned char s = 0;
280
281         if (stopBit)
282                 s = 0x04;
283
284         serialState.LCR = (serialState.LCR & 0xFB) | s;
285
286         out_byte(commBase + 3, serialState.LCR);
287
288         return 1;
289 }
290
291 /*=========================================================================*
292  *                                setBitCount                              *
293  *=========================================================================*/
294
295 PRIVATE int setBitCount(bitCount)
296 int bitCount;
```

```
297 {
298         if (bitCount >= 5 && bitCount <= 8)
299                 bitCount -= 5;
300
301         serialState.LCR = (serialState.LCR & 0xFC) | bitCount;
302
303         out_byte(commBase + 3, serialState.LCR);
304
305         return 1;
306 }
307
308 PRIVATE int doReset()
309 {
310         out_byte(commBase + 3, serialState.LCR & 0x7F);
311         out_byte(commBase + 2, 0x06);
312         out_byte(commBase + 3, serialState.LCR);
313
314         return 1;
315 }
316
317
318
319 /*==========================================================================*
320  *                                fleds_init                                *
321  *==========================================================================*/
322
323 PRIVATE void fleds_init()
324 {
325         doReset();
326
327         /*  inicializo las variables a usar */
328         serialState.LCR = 0;
329         serialState.DLL = 0;
330         serialState.DLM = 0;
331         serialState.MCR = 0;
332         serialState.IER = 0;
333         /* Inicializo las variables del buffer */
334         k_buffer.write = 0;
335         k_buffer.read = 0;
336         k_buffer.qty = 0;
337
338
339         setInterrupt(SERIAL_INT_E_OFF);
340         setParity(SERIAL_PARITY_OFF,SERIAL_PARITY_ODD);
341         setMCR();
342         setbps(SERIAL_S_115200);
343         setStop(SERIAL_STOP_1);
344         setBitCount(8);
345
346         /* si esta habilitado el ping lo uso */
347         #if FLEDS_PING_ENABLE
348         put_irq_handler(RS232_IRQ, fleds_handler);
349         enable_irq(RS232_IRQ);
350         #endif
351
352         doReset();
353
354         setInterrupt(SERIAL_INT_E_ON);
355
356 }
```

```
357
358
359
360 /*=========================================================================*
361  *                                add_buffer                               *
362  *=========================================================================*/
363
364 PRIVATE int add_buffer(data)
365 int data;
366 {
367         if( bufferNotFull() ) {
368                 k_buffer.buffer[k_buffer.write] = data;
369                 k_buffer.qty++;
370                 if ( ++(k_buffer.write) == FLEDS_BUFFER_SIZE)
371                         k_buffer.write = 0;
372                 return 1;
373         }
374         return 0;
375 }
376
377 /*=========================================================================*
378  *                                get_buffer                               *
379  *=========================================================================*/
380
381 PRIVATE char get_buffer()
382 {
383         char data;
384
385         if ( bufferNotEmpty() ) {
386                 k_buffer.qty--;
387                 data = k_buffer.buffer[k_buffer.read];
388                 if ( ++(k_buffer.read) == FLEDS_BUFFER_SIZE)
389                                         k_buffer.read = 0;
390                 return data;
391         }
392         return 0;
393 }
394
395
396 /*=========================================================================*
397  *                                fleds_handler                                *
398  *=========================================================================*/
399
400 PRIVATE int fleds_handler(irq)
401 int irq;
402 {
403         char data;
404
405         switch (data = in_byte(commBase))
406         {
407                 #if FLEDS_PING_ENABLE
408                 case FLEDS_PING:
409                         send_serial(FLEDS_PONG);
410                         break;
411                 #endif
412                 default:
413                         add_buffer(data);
414         }
415         return(1);        /* reenable serial interrupt */
416 }
```

```
417
418
419 /*===========================================================================*
420  *                                do_open                                    *
421  *===========================================================================*/
422
423 PRIVATE void do_open()
424 {
425         return;
426 }
427
428 /*===========================================================================*
429  *                                do_close                                   *
430  *===========================================================================*/
431
432 PRIVATE void do_close()
433 {
434         return;
435 }
436
437
438 /*===========================================================================*
439  *                                reply                                      *
440  *===========================================================================*/
441
442 PRIVATE void reply(code, replyee, process, status)
443 int code;                         /* TASK_REPLY or REVIVE */
444 int replyee;                        /* destination for message (normally FS) */
445 int process;                        /* which user requested the printing */
446 int status;                        /* number of  chars printed or error code */
447 {
448 /* Send a reply telling FS that printing has started or stopped. */
449
450   message pr_mess;
451
452   pr_mess.m_type = code;        /* TASK_REPLY or REVIVE */
453   pr_mess.REP_STATUS = status;        /* count or EIO */
454   pr_mess.REP_PROC_NR = process;        /* which user does this pertain to */
455   if ((status = send(replyee, &pr_mess)) != OK)
456         panic("reply failed, status\n", status);
457 }
458
459
460 /*===========================================================================*
461  *                                send_serial                                *
462  *===========================================================================*/
463
464 PRIVATE void send_serial(data)
465 int data;
466 {
467         out_byte(commBase, data);
468 }
469
```

4.7. src/kernel/fleds.h

Este archivo contiene la configuración del Fleds.

fleds.h

```
#define COMM1_BASE                0x3F8
#define COMM2_BASE                0x2F8

#define SERIAL_INT_E_ON                0x01
#define SERIAL_INT_E_OFF        0x00

#define SERIAL_PARITY_ON        0x01
#define SERIAL_PARITY_OFF        0x00

#define SERIAL_PARITY_ODD        0x00
#define SERIAL_PARITY_EVEN        0x01

#define SERIAL_STOP_1                0x00
#define SERIAL_STOP_2                0x01

#define FLEDS_PING_ENABLE        0
#define FLEDS_PING                '?'
#define FLEDS_PONG                '~'

#define FLEDS_BUFFER_SIZE        128

typedef enum{SERIAL_S_50, SERIAL_S_300,SERIAL_S_1200, SERIAL_S_2400, SERIAL_S_4800,
SERIAL_S_9600, SERIAL_S_19200, SERIAL_S_38400, SERIAL_S_57600, SERIAL_S_115200} serialSpeedT;

PUBLIC _PROTOTYPE( void fleds_task, (void) );
```

5. Aplicación

A modo de utilización del Fleds se implementó una librería libFleds.c destinada a facilitarle el uso del mismo al usuario, la misma incluye las siguientes funciones:

5.1. fledsADT newFleds (int height, int width, char * device)

Crea una nueva instacia del Fleds.

5.2. int freeFleds (fledsADT fleds)

Libera la instancia del Fleds.

5.3. int loadText (fledsADT fleds, char * text, int x, int y)

Carga un texto dado en el Fleds dadas las coordenadas x, y.

5.4. int loadPic (fledsADT fleds, pic_t pic, int x, int y)

Carga una imagen dado un pic_t en las coordenadas dadas.

5.5. int loadMovie (fledsADT fleds, pic_t * pic)

Carga una película en un vector de pic_t.

5.6. int clear (fledsADT fleds)

Limpia la pantalla del Fleds.

5.7. int show (fledsADT fleds)

Activa la imagen que tiene cargado el Fleds. Prende los leds.

5.8. int hide (fledsADT fleds)

Oculta lo que está mostrando el Fleds. Apaga los leds.

5.9. int animate (fledsADT fleds, animation_t animation, int iterations, int speed)

Anima los datos cargados en el Fleds con un tipo de animación. Existen diferentes tipos de animación, los cuales son:

5.9.1. SCROLL_NONE

No hace ningún tipo de scroll.

5.9.2. SCROLL_RIGHT

Scroll hacia la derecha y borra el contenido.

5.9.3. SCROLL_LEFT

Scroll hacia la izquierda y borra el contenido.

5.9.4. SCROLL_UP

Scroll hacia arriba y borra el contenido.

5.9.5. SCROLL_DOWN

Scroll hacia abajo y borra el contenido.

5.9.6. SCROLL_RIGHT_CARRY

Scroll hacia la derecha y hace cíclico el muestreo.

5.9.7. SCROLL_LEFT_CARRY

Scroll hacia la izquierda y hace cíclico el muestreo.

5.9.8. SCROLL_UP_CARRY

Scroll hacia arriba y hace cíclico el muestreo.

5.9.9. SCROLL_DOWN_CARRY

Scroll hacia abajo y hace cíclico el muestreo.

5.9.10. SCROLL_ROW_LEFT

Scrolea fila por fila hacia la izquierda limpiando la pantalla.

5.9.11. SCROLL_ROW_RIGHT

Scrolea fila por fila hacia la derecha limpiando la pantalla.

5.9.12. SCROLL_COLUMN_DOWN

Scrolea columna por columna hacia abajo limpiando la pantalla.

5.9.13. WAVE

Hace *la ola* con la imagen puesta en el fleds.

5.9.14. TWINKLE

Parpadea el contenido de la pantalla.

5.9.15. LSD

Tira animaciones re-locas.

5.9.16. MOVIE

Anima la pelicula.

5.9.17. CAMEL

Genera el efecto camel que tanto se comenta entre los jóvenes.

6. Conclusiones

Creemos haber cumplido el objetivo que nos planteamos al comienzo del tpe, desarrollar un dispositivo físico de leds y su respectivo driver para el sistema operativo Minix.
Se comprendió la interacción de Minix internamente, el armado de un controlador nuevo y la vinculación entre sí.
Finalmente, estamos satisfechos con el resultado obtenido y consideramos ésta una experiencia muy positiva.

7. Posibles extensiones

Se podría agregar dentro de lo que es el controlador la posibilidad de cambiarle parámetros de conexión con el dispositivo utilizando ioctl. Actualmente está definido pero no se está utilizando. Respecto al dispositivo, agregar más módulos para mostrar imágenes con una mayor resolución o textos más largos.

Referencias

[1] "*Sistemas Operativos. Diseño e implementación*" - Andrew S. Tanenbaum, Albert S. Woodhull

[2] "*Intro_minix.pdf*" - Cátedra de Sistemas Operativos

[3] "*Diseño de Sistemas Operativos*" - http://gsd.unex.es/ jdiaz/asig/dso/lab/p8/p8_v2.pdf

[4] "*Librerías fuentes de Minix*" - /usr/ dentro de minix.