MANEJADOR PARA

MEZCLADOR

SOUND BLASTER

German Jerez Cárdenes

Miguel Mederos Sánchez

Indice

1. INTRODUCCCION AL MEZCLADOR.

El mezclador tiene la tarea de controlar los altavoces de las distintas fuentes de entrada y de salida, de definir la vinculación de las diferentes fuentes y liberar las señales de entrada para el muestreo.

Cada diferente versión de la SoundBlaster va acompañada de una diferente versión del mezclador. Hasta el momento han aparecido tres tipos de mezcladores:

- El relativamente primitivo CT1335, que se utiliza con la SoundBlaster 2.0, si siempre y cuando haya instalada una extensión para conectar una unidad CD-ROM.
- Está el CT1345, que se encuentra en las tarjetas SoundBlaster Pro.
- Y el más claramente eficiente CT1745, introducido con la SoundBlaster 16.

La diferencia entre primitivo y eficiente la marcan las posibilidades de configuración del mezclador, que se han ido incrementando de versión en versión. Esto se refiere a la as y salidas, la disponibilidad de filtros para reducir los ruidos y la resolución en lo que respecta al volumen de una fuente de sonido o de una señal de salida. Mientras más alta sea la resolución, mejor podrán diferenciarse las diferentes fuentes lo que implica una mayor calidad.

2. ACCESO AL MEZCLADOR.

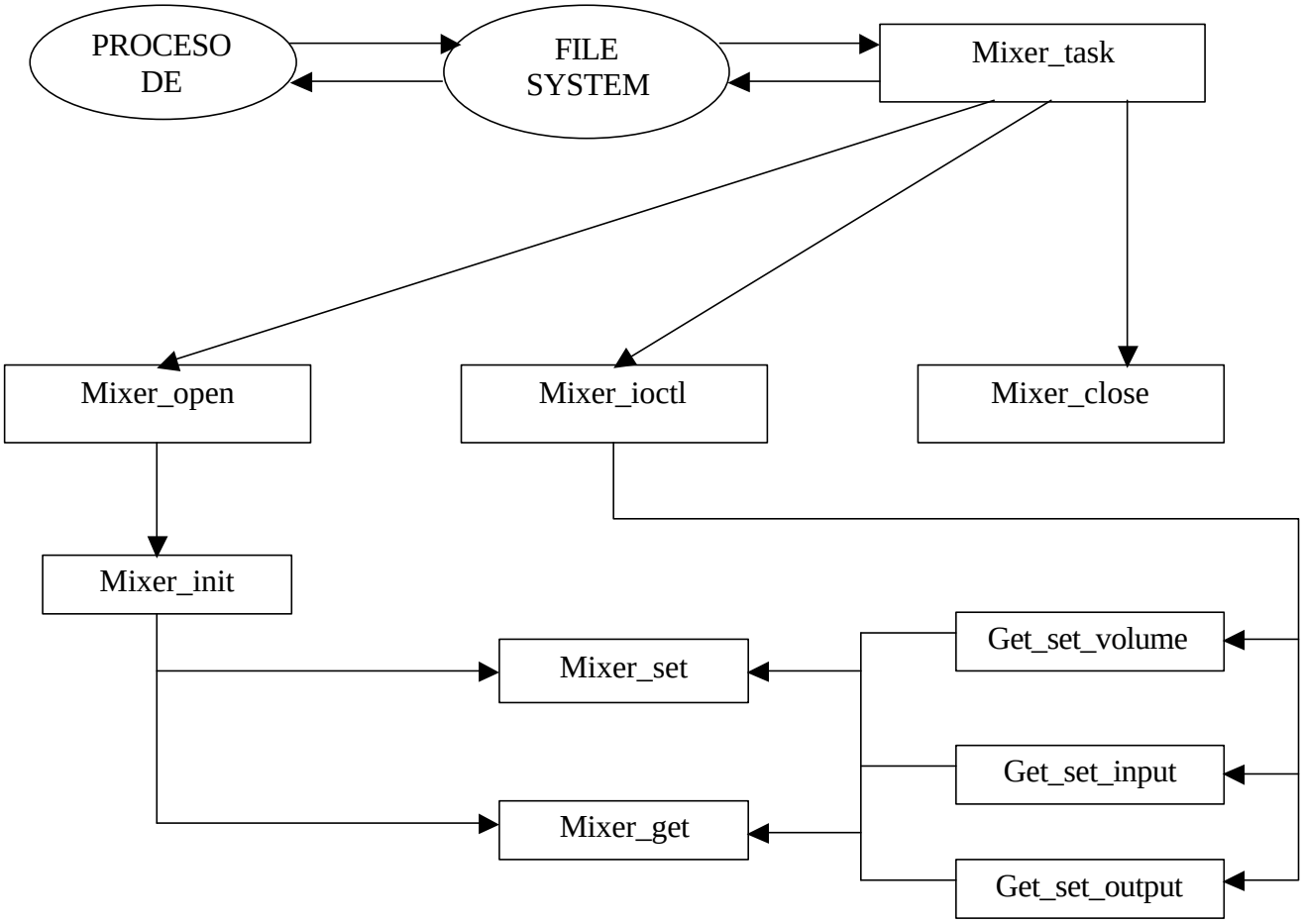
A los registros del mezclador se accede a través de puertos diferentes para datos y para direcciones. A través del puerto de direcciones tiene que especificarse primero el número del registro deseado del mezclador, antes de poder leer o escribir su contenido. La dirección del puerto de direcciones y de datos se define de forma relativa a la dirección base de la tarjeta SoundBlaster.

Puerto de direcciones del mezclador = Dirección Base-SoundBlaster + 4

Puerto de datos del mezclador = Dirección Base-SoundBlaster + 5

Después del envío del número de registro deseado al puerto de direcciones del mezclador no es necesario como en otros drivers crear un bucle de espera para darle al chip un poco de tiempo para el acceso, sino que es perfectamente posible realizar, inmediatamente después, un acceso de lectura al puerto de datos, cuando el registro especificado debe ser leído o un acceso a escritura, cuando se quiere modificar su contenido.

4. DIAGRAMA DE FUNCIONES DEL MEZCLADOR.



5. CODIGO DEL DRIVER DEL MEZCLADOR.

```
 * The driver supports the following operations (using message format m2):
 *
 *    m_type      DEVICE    PROC_NR     COUNT    POSITION  ADRRESS
 * ----------------------------------------------------------------
 * |  DEV_OPEN  | device  | proc nr |         |         |         |
 * |------------+---------+---------+---------+---------+---------|
 * |  DEV_CLOSE | device  | proc nr |         |         |         |
 * |------------+---------+---------+---------+---------+---------|
 * |  DEV_IOCTL | device  | proc nr |func code|         | buf_ptr |
 * ----------------------------------------------------------------
 *
 *
 * The file contains one entry point:
 *
 *   mixer_task:  main entry when system is brought up
 *


#include "kernel.h"
#include <minix/com.h>
#include <minix/callnr.h>
#include <sys/ioctl.h>
#include <minix/sound.h>
#if __minix_vmd
#include "config.h"
#endif
#include "sb16.h"

#if ENABLE_SB_AUDIO


/* Definición de las funciones que va a usar el mezclador */
FORWARD _PROTOTYPE( int mixer_init, (void));
FORWARD _PROTOTYPE( int mixer_open, (message *m_ptr));
FORWARD _PROTOTYPE( int mixer_close, (message *m_ptr));
FORWARD _PROTOTYPE( int mixer_ioctl, (message *m_ptr));
FORWARD _PROTOTYPE( int mixer_get, (int reg));
FORWARD _PROTOTYPE( int get_set_volume, (message *m_ptr, int flag));
FORWARD _PROTOTYPE( int get_set_input, (message *m_ptr, int flag, int
channel));
FORWARD _PROTOTYPE( int get_set_output, (message *m_ptr, int flag));


/*Variable usada para indicar si el mezclador está disponible o no */
PRIVATE int mixer_avail = 0;
```

5.1. Mixer_task

Es el punto de entrada al driver del mezclador. Consiste de un bucle infinito a la espera de recibir un mensaje. Una vez ese mensaje llega comprueba su origen y el tipo de servicio que solicita. En función del servicio ejecuta una rutina específica. Por último crea un mensaje de respuesta al proceso que lanzó la petición indicándole el estado final de la acción solicitada.

```
PUBLIC void mixer_task()
{
  message mess;
  int err, caller, proc_nr;

  /* Here is the main loop of the mixer task. It waits for a message,
carries
   * it out, and sends a reply.
   */
  while (TRUE)  /*Bucle infinito */
  {
      receive(ANY, &mess); /*Recepción del mensaje */

      caller = mess.m_source;
      proc_nr = mess.PROC_NR;

  /*Comprobación del origen del mensaje. Sólo tratará mensajes que vengan
del Fyle Sistem. */
      switch (caller)
        {
        case HARDWARE:
            /* Leftover interrupt. */
            continue;
        case FS_PROC_NR:
            /* The only legitimate caller. */
            break;
        default:
            printf("sb16: got message from %d\n", caller);
            continue;
      }

/*Detecta el tipo de operación solicitada y ejecuta una función */

      switch(mess.m_type)
        {
          case DEV_OPEN:      err = mixer_open(&mess);break;
          case DEV_CLOSE:     err = mixer_close(&mess);break;
          case DEV_IOCTL:     err = mixer_ioctl(&mess);break;
          default:           err = EINVAL;break;
      }

      /* Finalmente prepara y envía un mensaje de respuesta indicando el
estado de la ejecución de la función. */
      mess.m_type = TASK_REPLY;
      mess.REP_PROC_NR = proc_nr;

      mess.REP_STATUS = err;  /* error code */
      send(caller, &mess);    /* send reply to caller */
  }
}
```

5.3. Mixer_close

Esta función se utiliza para cerrar el mezclador

```
PRIVATE int mixer_close(m_ptr)
message *m_ptr;
{
#if SB_DEBUG
  printf("mixer_close\n");
#endif
  return OK;
}
```

5.4. Mixer_ioctl

Esta es la última de las tres operaciones que se pueden solicitar al mezclador, y que permite leer o modificar el volumen, cambiar la fuente de entrada o de salida y leer datos sobre la fuente actual.

```
PRIVATE int mixer_ioctl(m_ptr)
message *m_ptr;
{
  int status;

#if SB_DEBUG
  printf("mixer: got ioctl %d\n", m_ptr->REQUEST);
#endif

/*Lee del mensaje la opción solicitada y ejecuta una función. */
  switch(m_ptr->REQUEST)
  {
    case MIXIOGETVOLUME:      status = get_set_volume(m_ptr, 0);break;
    case MIXIOSETVOLUME:      status = get_set_volume(m_ptr, 1);break;
    case MIXIOGETINPUTLEFT:   status = get_set_input(m_ptr, 0, 0);break;
    case MIXIOGETINPUTRIGHT:  status = get_set_input(m_ptr, 0, 1);break;
    case MIXIOGETOUTPUT:      status = get_set_output(m_ptr, 0);break;
    case MIXIOSETINPUTLEFT:   status = get_set_input(m_ptr, 1, 0);break;
    case MIXIOSETINPUTRIGHT:  status = get_set_input(m_ptr, 1, 1);break;
    case MIXIOSETOUTPUT:      status = get_set_output(m_ptr, 1);break;
    default:                  status = ENOTTY;
  }
  return status; /*Devuelve el estado final de ejecución de la rutina */
}
```

5.5. Mixer_init

Esta función trata de inicializar el dispositivo y para ello en primer lugar trata de detectarlo y en caso afirmativo, activa el AGC y devuelve un mensaje de ejecución correcta.

```
PRIVATE int mixer_init()
{
  /* Intenta detectar el mezclador. Para ello escribe en el registro
interno MIXER_DAC_LEVEL y si el valor escrito se puede leer entonces el
mixer ha sido detectado */
  mixer_set(MIXER_DAC_LEVEL, 0x10);
  if (mixer_get(MIXER_DAC_LEVEL) != 0x10)
  {
    printf("sb16: Mixer not detected\n");
    return EIO;
  }
  /* Habilita el Amplificador Automático del Micrófono (AGC) */
  mixer_set(MIXER_AGC, 0x01);

#if SB_DEBUG
  printf("Mixer detected\n");
#endif
  mixer_avail = 1; /*El mezclador está ocupado */
  return OK;
}
```

5.6. Mixer_set

Escribe en el registro 'reg', los datos pasados por parámetros 'data'.

```
PUBLIC int mixer_set(reg, data)
int reg;
int data;
{
  int i;
/*Escribe en el puerto de direcciones el registro a modificar */
  out_byte(MIXER_REG, reg);
  for(i=0;i<100;i++);
/*Se realiza un acceso de escritura al puerto de datos */
  out_byte(MIXER_DATA, data);
  return OK;
}
```

5.7. Mixer_get.

Lee los datos del registro 'reg'.

```
PRIVATE int mixer_get(reg)
int reg;
{
  int i;
/*Escribe en el puerto de direcciones el registro a leer */
  out_byte(MIXER_REG, reg);
  for(i=0;i<100;i++);
/*Se realiza un acceso de lectura al puerto de datos */
  return (in_byte(MIXER_DATA) & 0xff);
}
```

5.8. Get_set_volume.

La función lee o modifica el nivel de volumen de la fuente indicada en el mensaje del proceso de usuario.

```
PRIVATE int get_set_volume(m_ptr, flag)
message *m_ptr;
int flag;   /* 0 = get, 1 = set */
{
  phys_bytes user_phys;
  struct volume_level level;
  int cmd_left, cmd_right, shift, max_level;

/*Dirección física del proceso de usuario */
user_phys = numap(m_ptr->PROC_NR, (vir_bytes) m_ptr->ADDRESS,
                                        sizeof(struct volume_level));
  if (user_phys == 0) return(EFAULT);
  phys_copy(user_phys, vir2phys(&level), (phys_bytes) sizeof(level));
/*Se copian los datos del usuario en el Kernel. */
  shift = 3;
  max_level = 0x1F;
/*En función de la fuente escoge unos registros del mezclador u otros */
switch (level.device)
  {
    case Master:  { cmd_left = MIXER_MASTER_LEFT;
                    cmd_right = MIXER_MASTER_RIGHT;
                  };break;
    case Dac:     { cmd_left = MIXER_DAC_LEFT;
                    cmd_right = MIXER_DAC_RIGHT;
                  };break;
    case Fm:      { cmd_left = MIXER_FM_LEFT;
                    cmd_right = MIXER_FM_RIGHT;
                  };break;
    case Cd:      { cmd_left = MIXER_CD_LEFT;
                    cmd_right = MIXER_CD_RIGHT;
                  };break;
    case Line:    { cmd_left = MIXER_LINE_LEFT;
                    cmd_right = MIXER_LINE_RIGHT;
                  };break;
    case Mic:     { cmd_left = cmd_right = MIXER_MIC_LEVEL;
                  };break;
    case Speaker: { cmd_left = cmd_right = MIXER_PC_LEVEL;
                    shift = 6;
                    max_level = 0x03;
                  };break;
    case Treble:  { cmd_left = MIXER_TREBLE_LEFT;
                    cmd_right = MIXER_TREBLE_RIGHT;
                    shift = 4;
                    max_level = 0x0F;
                  };break;
    case Bass:    { cmd_left = MIXER_BASS_LEFT;
                    cmd_right = MIXER_BASS_RIGHT;
                    shift = 4;
                    max_level = 0x0F;
                   };break;
    default:     return EINVAL;
  }
```

```
  if (flag)  /* Modificar Nivel de Volumen */
  {
/*Comprobación de que el dato a insertar está dentro de un rango */
    if (level.right < 0) level.right = 0;
    else if (level.right > max_level) level.right = max_level;
    if (level.left < 0) level.left = 0;
    else if (level.left > max_level) level.left = max_level;

    mixer_set(cmd_right, (level.right << shift));
    mixer_set(cmd_left, (level.left << shift));
  }
  else       /* Leer Nivel de Volumen */
  {
    level.left = mixer_get(cmd_left);
    level.right = mixer_get(cmd_right);

    level.left >>= shift;
    level.right >>= shift;

    /* Se realiza una copia de los datos al usuario */
    phys_copy(vir2phys(&level), user_phys, (phys_bytes) sizeof(level));
  }

  return OK;
}
```

5.9. Get_set_input.

Se encarga de leer datos de los canales izquierdo y derecho de la fuente de entrada o de modificar la fuente de entrada de los canales.

```
PRIVATE int get_set_input(m_ptr, flag, channel)
message *m_ptr;
int flag;   /* 0 = get, 1 = set */
int channel;    /* 0 = left, 1 = right */
{
  phys_bytes user_phys;
  struct inout_ctrl input;
  int input_cmd, input_mask, mask, del_mask, shift;

/*Dirección física del proceso de usuario */
  user_phys = numap(m_ptr->PROC_NR, (vir_bytes) m_ptr->ADDRESS,
                                        sizeof(struct inout_ctrl));
  if (user_phys == 0) return(EFAULT);

  phys_copy(user_phys, vir2phys(&input), (phys_bytes) sizeof(input));
/*Copia de datos del usuario a la fuente de entrada */

  input_cmd = (channel == 0 ? MIXER_IN_LEFT : MIXER_IN_RIGHT);
  mask = mixer_get(input_cmd);

/* En función de la fuente de entrada son los valores de desplazamiento y

  switch (input.device)
  {
    case Fm:   { shift = 5;
                 del_mask = 0x1F;
               };break;
    case Cd:   { shift = 1;
                 del_mask = 0x79;
```

```
                };break;
    case Line: { shift = 3;
                 del_mask = 0x67;
                };break;
    case Mic:  { shift = 0;
                 del_mask = 0x7E;
                };break;
    default:   return EINVAL;
  }
  if (flag)  /* Modificar la fuente de entrada */
  {
    input_mask =
           ((input.left == ON ? 1 : 0) << 1) | (input.right == ON ? 1 : 0);

    if (shift > 0)
     input_mask <<= shift;
    else
     input_mask >>= 1;

    mask &= del_mask;
    mask |= input_mask;

    mixer_set(input_cmd, mask);
  }



  else      /* Obtener datos de la fuente de entrada */
  {
    if (shift > 0)
    {
      input.left = ((mask >> (shift+1)) & 1 == 1 ? ON : OFF);
      input.right = ((mask >> shift) & 1 == 1 ? ON : OFF);
    }
    else
      input.left = ((mask & 1) == 1 ? ON : OFF);

    /* Copia de datos de la fuente de entrada al usuario */
 phys_copy(vir2phys(&input), user_phys, (phys_bytes) sizeof(input));
  }

  return OK;
}
```

5.10. Get_set_output.

Se encarga de leer datos de la fuente de salida o de modificar la fuente de salida actual.

```
PRIVATE int get_set_output(m_ptr, flag)
message *m_ptr;
int flag;   /* 0 = get, 1 = set */
{
  phys_bytes user_phys;
  struct inout_ctrl output;
  int output_mask, mask, del_mask, shift;

/*Dirección física del proceso de usuario */
  user_phys = numap(m_ptr->PROC_NR, (vir_bytes) m_ptr->ADDRESS,
```

```
                                          sizeof(struct inout_ctrl));
  if (user_phys == 0) return(EFAULT);
  phys_copy(user_phys, vir2phys(&output), (phys_bytes) sizeof(output));
/*Copia de datos del usuario a la fuente de salida */

  mask = mixer_get(MIXER_OUTPUT_CTRL);

/* En función de la fuente de salida son los valores de desplazamiento y la

  switch (output.device)
  {
    case Cd:   { shift = 1;
                 del_mask = 0x79;
               };break;
    case Line: { shift = 3;
                 del_mask = 0x67;
               };break;
    case Mic:  { shift = 0;
                 del_mask = 0x7E;
               };break;
    default:   return EINVAL;
  }


  if (flag)  /* Modificar la fuente de salida */
  {
    output_mask =
           ((output.left == ON ? 1 : 0) << 1) | (output.right == ON ? 1 :
0);

    if (shift > 0)
     output_mask <<= shift;
    else
     output_mask >>= 1;

    mask &= del_mask;
    mask |= output_mask;

    mixer_set(MIXER_OUTPUT_CTRL, mask);
  }


  else      /*Obtener datos de la fuente de salida */
  {
    if (shift > 0)
    {
      output.left = ((mask >> (shift+1)) & 1 == 1 ? ON : OFF);
      output.right = ((mask >> shift) & 1 == 1 ? ON : OFF);
    }
    else
      output.left = ((mask & 1) == 1 ? ON : OFF);

/* Copia de datos de la fuente de salida al usuario */
phys_copy(vir2phys(&output), user_phys, (phys_bytes) sizeof(output));
  }

  return OK;
}
#endif /* ENABLE_AUDIO */
```

6. CUESTIONES.

6.1. Tareas que realiza el mezclador de una SoundBlaster.

El mezclador tiene las tareas de controlar los altavoces de las distintas fuentes de entrada y de salida, de definir la vinculación de las diferentes fuentes y liberar las señales de entrada para el muestreo

6.2. Pasos a realizar y funciones que intervienen par cambiar el volumen de una fuente.

En primer lugar un proceso de usuario realiza la petición de cambiar el volumen de la fuente actual al File System quien a su vez traslada el mensaje a la función ***Mixer_task****. Esta función detecta el origen del mensaje y la operación solicitada y llama a la función* ***Mixer_ioctl,*** *quien a su vez llama a la función* ***Get_set_volume*** *que es la que se encarga finalmente de modificar el volumen según lo indicado por el proceso de usuario, devolviendo el resultado de la ejecución a Mixer_ioctl quien a su vez se lo devuelve a Mixer_task, la cual crea un mensaje de respuesta al proceso de usuario que solicitó el servicio, indicándole el resultado final de la ejecución de dicho servicio.*

6.3. ¿Cuál es la forma de acceder al mezclador para leer los datos de un registro?

Para realizar un acceso de lectura a un registro interno del mezclador, en primer lugar hay que indicar el registro que se desea leer en el puerto de direcciones, y a continuación sin tener que realizar un bucle de espera hay que hacer un acceso de lectura de ese registro por el puerto de datos.