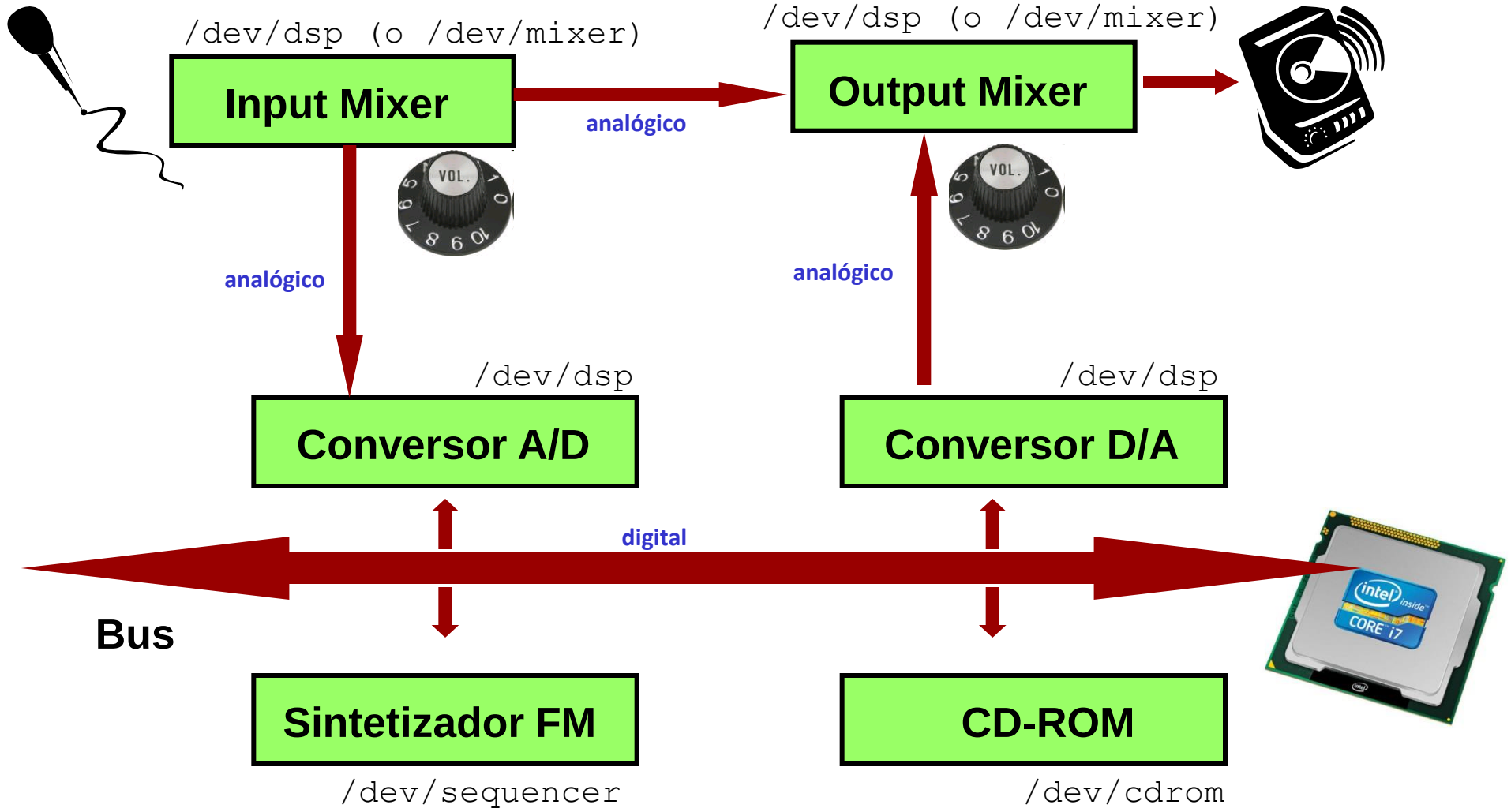
Técnicas de programación de software multimedia
(parte 1)

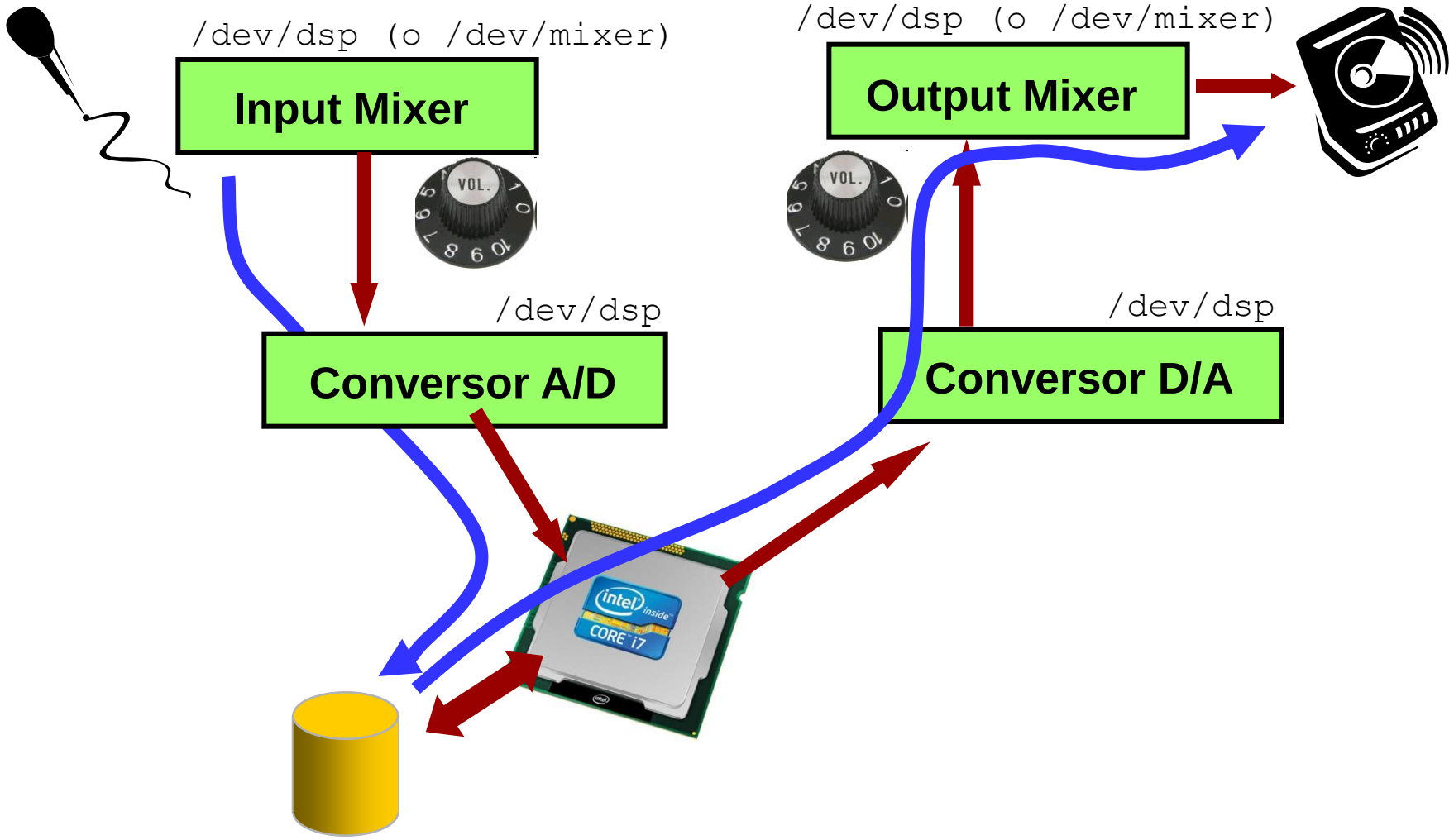
Acceso a los dispositivos en Linux

- **Abstracción común para los dispositivos en Linux (Unix): los dispositivos aparecen como *ficheros***
  - **En `/dev/`…**
  - **Nota: algunos dispositivos también pueden ser accesibles a través de APIs (posiblemente con "más capacidades" o "más fácilmente" que con el interfaz de ficheros)**
- **Los ficheros tienen unas operaciones comunes:**
  - ✓ **`open`, `close`**
  - ✓ **`read`, `write`**
  - ✓ **`ioctl`**
  - **La semántica de las operaciones se adapta a cada dispositivo**
    - ✓ **Pensar en disco duro, línea serie, control de un LED, … tarjeta de sonido**

Tarjeta de sonido



Tarjeta de sonido (para nosotros)



Referencias para programación de tarjeta de sonido

- **Referencia para programar tarjeta de sonido según API de OSS (Open Sound System):**

  http://www.4front-tech.com/pguide/index.html

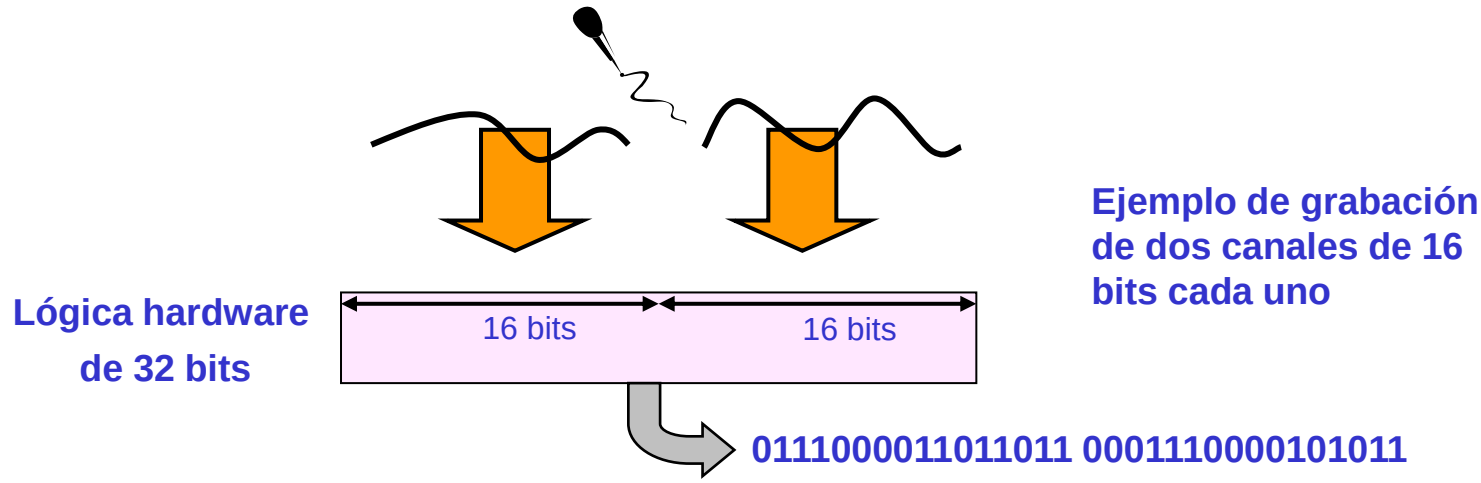
- **Definiciones en fichero `soundcard.h`**
  - **Ejecuta `locate soundcard.h` para saber dónde está**

Dispositivo `/dev/dsp`

- **`/dev/dsp`: conversión analógica/digital (muestreo y cuantificación) y digital/analógica**
  - **Generalmente hay UN dispositivo en un equipo**
  - **Sólo puede tenerlo abierto UN programa a la vez**
    - ✓ **Programas que lo necesitan esporádicamente lo abren, lo usan, y lo cierran inmediatamente**
  - **`read` obtiene datos del conversor analógico/digital (= graba datos)**
  - **`write` vuelca datos en el conversor digital/analógico (= reproduce datos)**

Dispositivo `/dev/dsp`

- `ioctl` a `/dev/dsp` permite configurar los parámetros básicos de funcionamiento de la tarjeta
  - **Frecuencia de muestreo (8000 Hz, 44100 Hz)**
  - **Número de canales (1, 2, 5.1)**
  - **Formato de la muestra para cada canal**
    - ✓ **Número de bits por muestra (8 o 16 bits)**
    - ✓ **Tipo de codificación (uniforme o con algún tipo compresión)**
      - ➢ Puede hacerse por software o por hardware



Ejemplo de grabación de dos canales de 16 bits cada uno

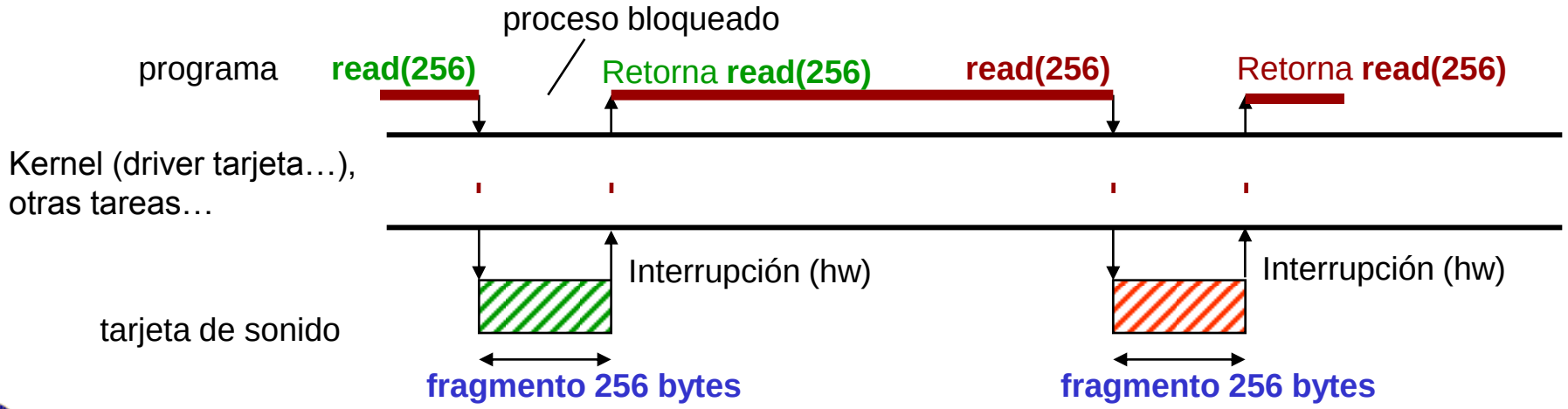
- Los parámetros usados al ejecutar `read` o `write` son los últimos configurados en el dispositivo `/dev/dsp`
  - **= la tarjeta de sonido mantiene estado**

Fragmentos

- **La tarjeta dispone de una zona de memoria interna**
  - **Esta memoria está gestionada en bloques de igual tamaño llamados "*fragmentos*"**
    - ✓ **El tamaño del fragmento es configurable, aunque debe ser potencia de 2**
      - 128, 256, 512 … bytes
  - **Interesante configurar explícitamente el tamaño de los fragmentos si trabajamos con `select` y la aplicación es sensible a los bloqueos**

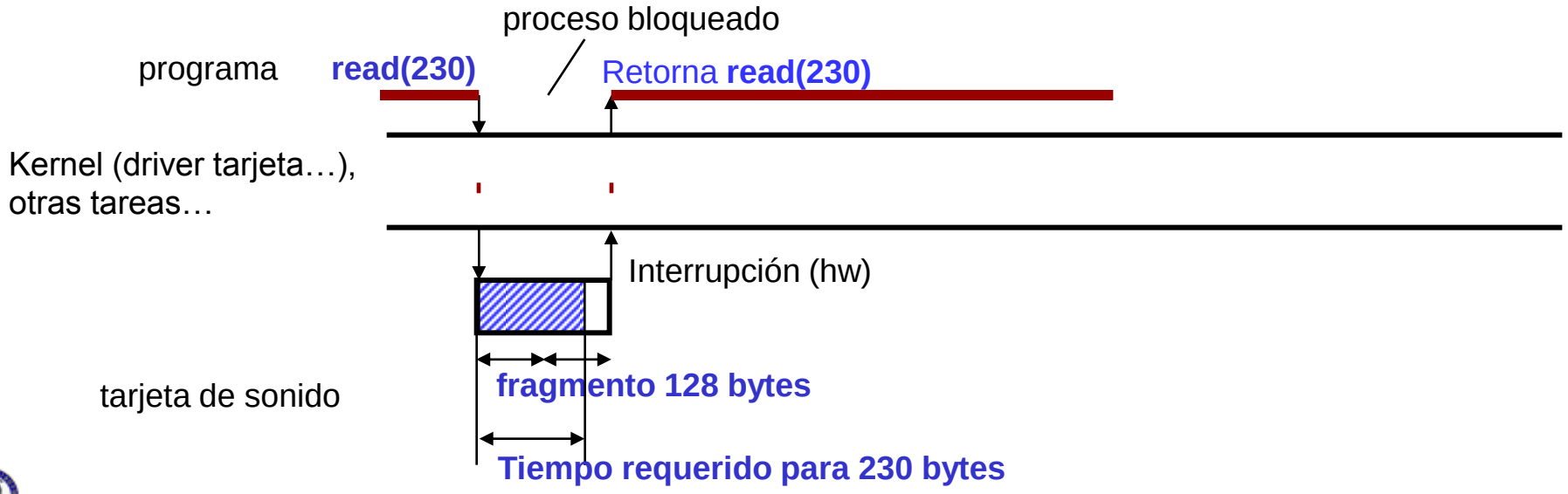
Bloqueos en la tarjeta de sonido

- **Lectura (`read`) esporádica: el proceso solicitante se bloquea hasta obtener la cantidad de datos correspondiente al un número de fragmentos necesarios para obtener esa cantidad de datos**
  - **No hasta obtener los datos, sino más (si el fragmento > datos solicitados)!**
  - **Mínimo tiempo de bloqueo si el número de datos que se solicita en el `read` es igual que el tamaño del fragmento**



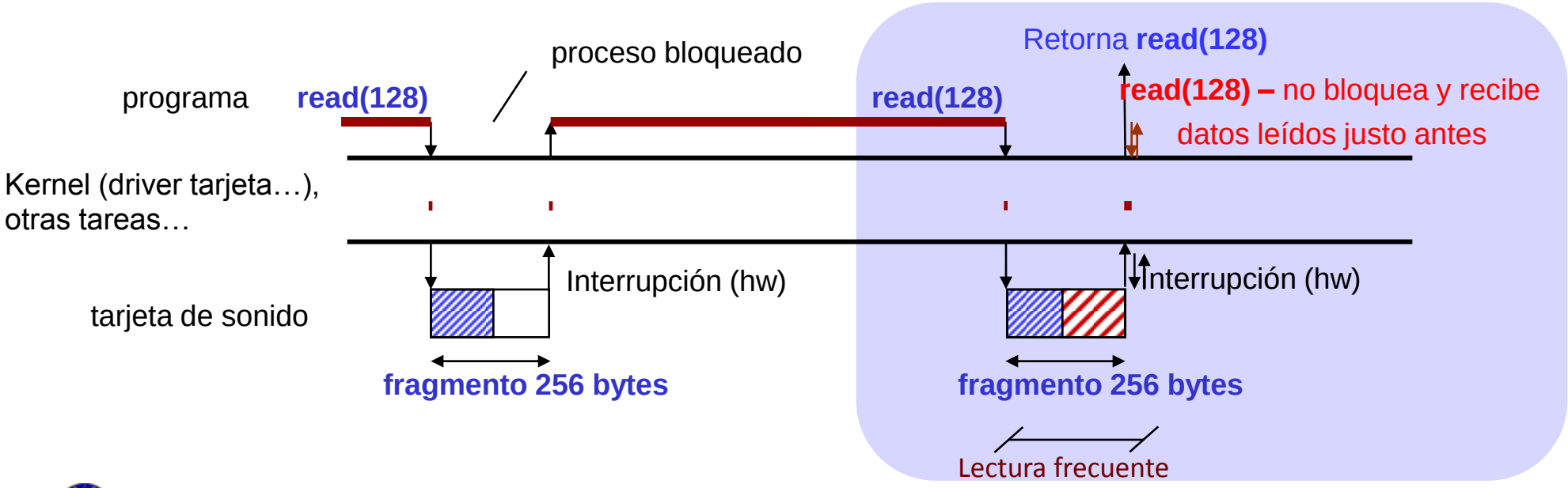
Bloqueos en la tarjeta de sonido

- **Lectura si el tamaño solicitado es distinto del tamaño del fragmento**
  - ✓ **Ej: tamaño de fragmento = 128 bytes.**
  - ✓ **Se solicita lectura de 230 bytes**
  - ✓ **Tarjeta bloquea durante el tiempo necesario para leer 2 fragmentos de 128 bytes (=256 bytes, potencia de 2 inmediatamente superior a 230 bytes)**
  - ✓ **La operación de lectura devuelve 230 bytes**



Bloqueos en la tarjeta de sonido

- **No obstante, si la lectura es frecuente, y la cantidad de datos solicitados es menor que un fragmento, la tarjeta devuelve datos que habían sido leídos con anterioridad**
  - **Ej: fragmento 256 bytes, lecturas (`read`) que solicitan 128 bytes**
    - ✓ **En este caso, algunas lecturas no son bloqueantes**
  - **No obstante, conviene evitar este funcionamiento: para el tipo de aplicaciones que queremos, introduce un retardo indeseado (algunas lecturas de 128 bytes tardan como leer 256)**

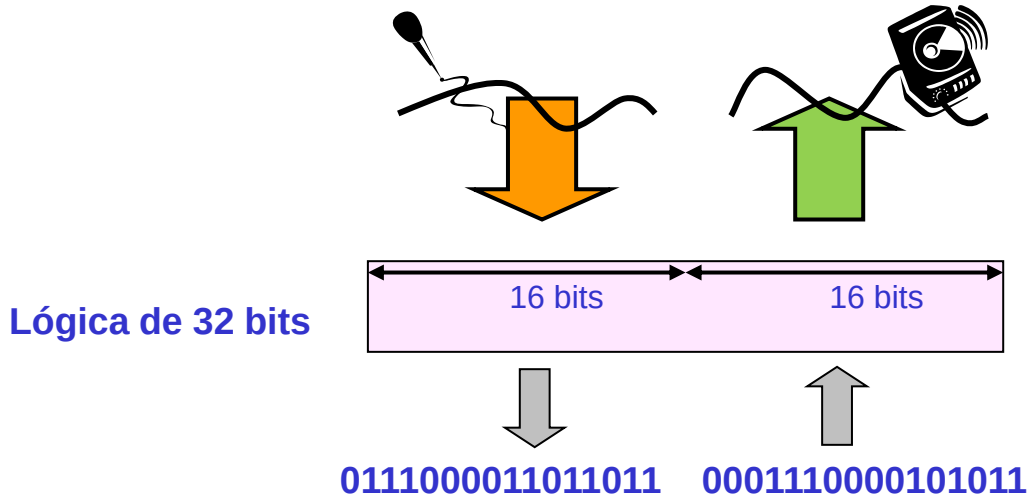


Bloqueos en la tarjeta de sonido

- La escritura es **no bloqueante**: mientras haya memoria suficiente en la tarjeta de sonido
- La escritura es **bloqueante** si la memoria de la tarjeta de sonido asociada a escritura está llena, en cuyo caso bloquea
  - **El proceso se desbloquea cuando hay memoria suficiente para almacenar los datos de la solicitud de escritura**

Modo dúplex

- **Modo Dúplex:**
  - La tarjeta puede grabar y reproducir A LA VEZ



  - Ej: el código puede hacer un `read` e inmediatamente después un `write`... y la tarjeta graba y reproduce simultaneamente
  - Nota: si el 'ancho' del conversor A/D, D/A es de 32 bits, cada sentido sólo puede disponer de 16 bits, limitando las opciones a
    - ✓ 1 canal de 16 bits
    - ✓ 2 canales de 8 bits

/* Truco de programación */

- **Comprobar siempre resultados de llamadas al Sistema Operativo**
  - **¡pueden haber salido mal!**
  - **Habitualmente, las llamadas que salen mal devuelven `-1`**
    - ✓ **Leer siempre la página de manual de la llamada, para ver cómo se codifica el éxito o fallo en la llamada al SO (si devuelven un puntero, el error se notifica devolviendo NULL)**
- **Automatizando el chequeo del éxito en llamadas al SO**
  - ***Macro C* que comprueba si una llamada al SO devuelve `-1` => mensaje de error y detiene la ejecución de la función**

```
#define CALL(v,m) {if ( (v)==-1) {perror (m);
printf ("Error number: %d, %s\n", errno, strerror
(errno)); exit (1); }};
```

  - **Ejemplo de uso**

```
CALL (callX (argumento1, argumento2..), "La
llamada a callX fracasó después de haberse
ejecutado…");
```

Configurando parámetros

```
int ioctl_arg;      /* argumentos para ioctl */
```

```
/* abrimos dispositivo de audio */
CALL (
 (descSnd = open("/dev/dsp", O_RDWR),
 "Fallo en apertura de  /dev/dsp"
 );
```

```
  /* fijar parametros de muestreo */
  ioctl_arg = formato;
 /* Ej: 8 es PCM (con cuantificación lineal de 8 bits);
        16 es PCM con 16 bits little endian con signo;
    ver /usr/include/linux/soundcard.h para más posibilidades */
CALL(
  ioctl(descSnd, SNDCTL_DSP_SETFMT, &ioctl_arg),
   "Fallo en establecimiento del formato de grabac/reproduc"
  );
   if (ioctl_arg != formato)
  perror("Imposible seleccionar codificación pedida");
```

¡Comprobar si la tarjeta hizo lo que se le pedía!

Programando la tarjeta de sonido

Configurar **parámetros conversión A/D y D/A:** ioctl…

SNDCTL_DSP_SETFMT – número de bits por muestra y codificación (ver soundcard.h para qué codificaciones)

SNDCTL_DSP_CHANNELS – número de canales

SNDCTL_DSP_SPEED – frecuencia de muestreo

```
Duplex:  CALL (ioctl (descSnd, SNDCTL_DSP_SETDUPLEX,
         0), "Fallo al poner la  tarjeta de sonido en
         modo DUPLEX");
```

Configurando **el volumen**

```
CALL (ioctl (descSnd, MIXER_WRITE (SOUND_MIXER_MIC),
 &volFinal), "Fallo en onfiguración volumen del microfono");
```

volFinal: 8 bits más signif= canal derecho / 8 menos significativos canal izquierdo

```
CALL (ioctl (descSnd, MIXER_WRITE (SOUND_MIXER_PCM),
 &volFinal), "Fallo en configuración volumen de la salida");
```

Programando la tarjeta de sonido: tamaño de fragmento

- **Ajustar el tamaño del fragmento al tiempo de respuesta requerido por la aplicación**
  - La llamada al `ioctl` que ajusta el tamaño de los fragmentos tiene que hacerse ANTES de otros `ioctls`, `read`, `write`…
    - ✓ Si se quiere cambiar el tamaño del fragmento, hay que cerrar y abrir otra vez el dispositivo
  - Datos: `requestedFragmentSize` es el número de bytes que se quieren manejar en la tarjeta.

    Código obtiene tamaño en bytes que es potencia de 2 inmediatamente inferior al solicitado (función logaritmo).

    Configura tarjeta para que el fragmento sea de ese tamaño
  - Es decir,
    - ✓ Si queremos tamaño de fragmento 32, hay que utilizar 5 como argumento
    - ✓ Tamaño de fragmento 64, hay que utilizar 6 como argumento
    - ✓ Tamaño de fragmento 43… tendremos que poner 5, y el tamaño de fragmento real será 32 (argumento=5)

Referencia: http://www.4front-tech.com/pguide/audio2.html

Programando la tarjeta de sonido

```
void configSndcard (int *descSnd, struct structSndQuality *datosQosSnd, int
    *requestedFragmentSize) {
unsigned int frgmArg, requestedFrgmArg; /* local variables for fragment conf */
/* Argument to construct: 0xMMMMSSSS  , being MMMM fragments (MMMM=number of
    fragments) of size 2^SSSS (SSSS= log2 of the size of the fragments.
We set the size of the fragments, and we do not request anything about the number of
    fragments */
requestedFrgmArg = floor_log2 (*requestedFragmentSize); /* floor_log2 is defined at
    the end of this file. Sets 'SSSS' part of fragment argument */

requestedFrgmArg = requestedFrgmArg | 0x7FFF0000 ; /* this value is used to request
    the soundcard not to limit the number of fragments available. Sets 'MMMM' part of
    fragment argument */

frgmArg = requestedFrgmArg ;
CALL (ioctl ( (*descSnd), SNDCTL_DSP_SETFRAGMENT, & frgmArg ), "Failure when setting
    the fragment size\n");
if (frgmArg != requestedFrgmArg )
    { printf ("Fragment size could not be set to the requested value: requested argument was %d,
    resulting argument is %d\n", requestedFrgmArg, frgmArg); exit (-1);}
/* returns configured fragment value, in bytes, so it performs 2^frgmArg by shifting
    one bit the appropriate number of times */
* requestedFragmentSize = 1 << frgmArg; /* returns actual configured value of the
    fragment size, in bytes*/
}
```

Referencia: http://www.4front-tech.com/pguide/audio2.html

Programando la tarjeta de sonido: datos en el buffer de escritura

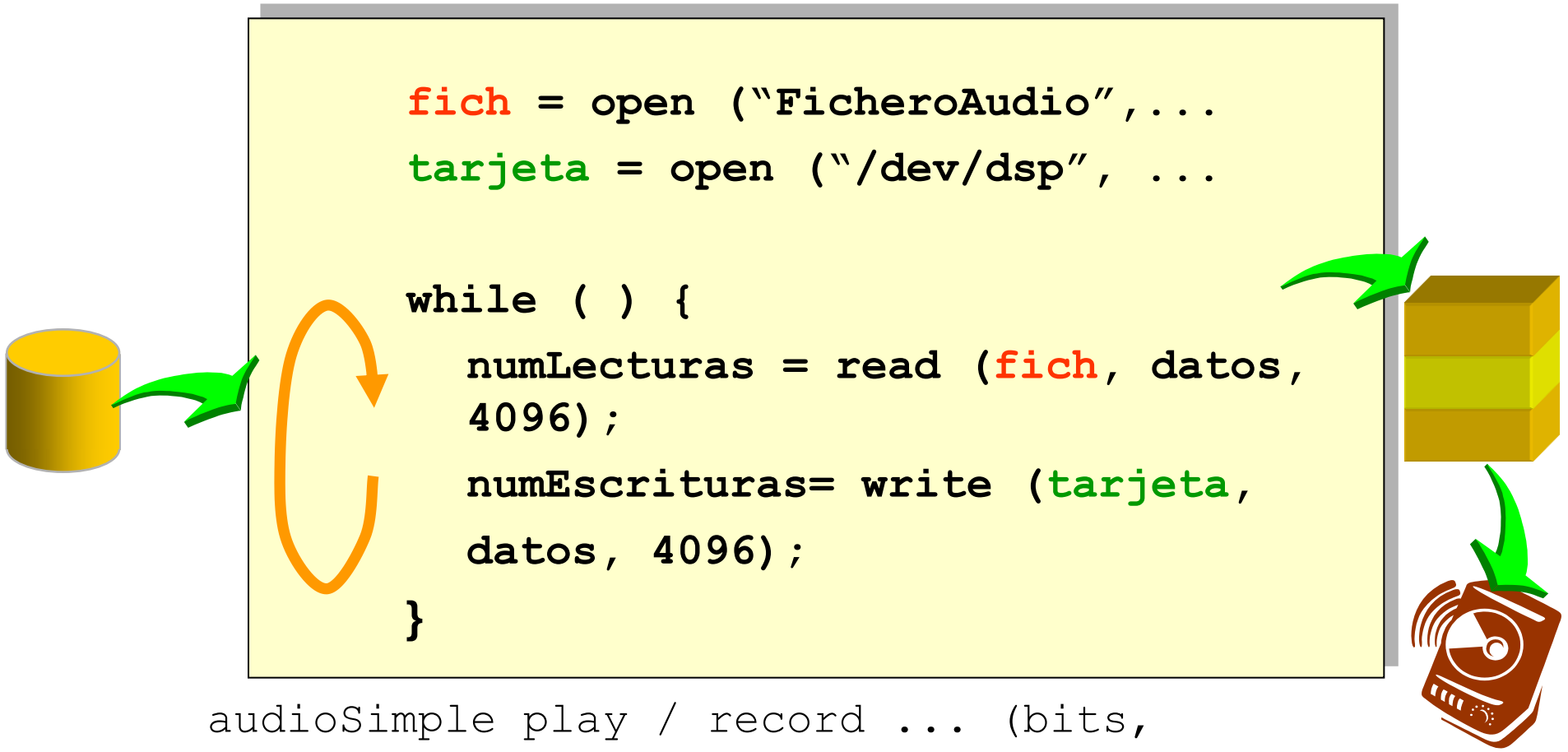
```
int bytes;

ioctl(soundcard, SNDCTL_DSP_GETODELAY,
&bytes);
```

- Indica el número de bytes que quedan por reproducir en la tarjeta de sonido en el momento en el que se hace la llamada
- Convertir en segundos=

bytes / (sample_rate * num_of_channels * sample_size_in_bytes)

Reproductor/Grabador de Audio

En la página de web hay un programa sencillo para grabar y reproducir audio (ver Práctica 0)



```
fich = open ("FicheroAudio",...
tarjeta = open ("/dev/dsp", ...

while ( ) {
   numLecturas = read (fich, datos,
   4096);
   numEscrituras= write (tarjeta,
   datos, 4096);
}
```

audioSimple play / record ... (bits, estereo, duración, frecuencia ...)