AUDIO EN LINUX

Una de las cosas a controlar es la administración del audio en Linux, el flujo de datos desde el archivo de audio hasta los altavoces puede volverse una cadena complicada con tantas tecnologías involucradas.

Con el utilidad lshw se obtiene información completa de los dispositivos de audio del sistema. Se sugiere instalarla si no se encuentra en el sistema.

```
$ sudo lshw -c sound
```

ESTRUCTURA DE AUDIO EN LINUX

Es válido en parte, la analogía con el modelo OSI de redes, en cuanto a jerarquía y organización, donde cada capa tiene su función y maneja proceso específicos, pero en la arquitectura de audio de Linux, estas capas varían con la implementación de nuevas tecnologías que permiten jugar con el kernel y con el hardware del sistema.

La capa más baja de la pirámide es una capa física Se encuentran los dispositivos de hardware, como tarjetas internas o interfaces externas de audio.

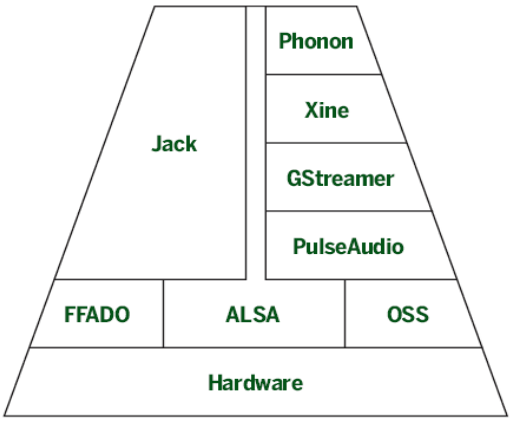

Fig 01. Estructura Audio Linux

En la segunda capa se encuentran las principales tecnologías compatibles con el núcleo del sistema.

FFADO

FFADO es una solución open-source para soporte de dispositivos de audio, mediante conexión FIREWIRE, utilizaos en el audio semi-profesional y profresional. A diferencia del USB donde hay un estándar para el manejo del audio la complejidad de este protocolo hace que necesiten sus propios controladores y por tanto su propia capa

OSS (Open Sound System)

OSS es una arquitectura de sonido alternativa para sistemas compatibles con Unix y Posix.

OSS era el sistema de sonido original de Linux y venía incluido en el kernel hasta el año 2002, cuando fue reemplazado por ALSA y se convirtió en un producto propietario.
Luego en el año 2007 el proyecto volvió a ser libre.

ALSA (Advanced Linux Sound Architecture)

ALSA es el actual componente del kernel de Linux para proporcionar controladores de dispositivos para las tarjetas de audio. ALSA también pone a disposición una amplia biblioteca en el espacio de usuario para desarrolladores.

ALSA fue diseñado para reemplazar OSS (Open Sound System). Sin embargo, el OSS no está realmente muerto, gracias a una capa

de compatibilidad en ALSA diseñada para ejecutar aplicaciones. Es más fácil pensar en ALSA como la capa de control de dispositivos del sistema de sonido de Linux.

SERVIDORES DE AUDIO

Los servidores de audio son paquetes de software, diseñados para administrar los controladores proporcionados por la arquitectura de audio del sistema, sea ALSA, OSS o FFADO. Algunos vienen por defecto con las distribuciones ya que son los utilizados por los navegadores y/o reproductores multimedia de la distribución, otros como Jack es necesario instalarlos y configurarlos.

Jack

Es un servidor de sonido o demonio[1] multiplataforma que provee conexión de baja latencia entre aplicaciones tipo *jackified*, para audio y datos MIDI.
En la actualidad existe para Linux, OS X, Solaris, FreeBSD y Windows. Puede conectar varias aplicaciones cliente a un dispositivo de audio, y permitirles compartir audio entre sí. Los clientes pueden ejecutarse como procesos separados como aplicaciones normales, o en el servidor JACK como "plugins".
Jack fue diseñado desde un principio para audio profesional y su diseño se centra en dos áreas clave: ejecución sincrónica de todos los clientes y el funcionamiento de baja latencia.
El servidor Jack está licenciado bajo GNU GPL, mientras que las bibliotecas están licenciadas bajo GNU LGPL.

Pulse Audio

Es un servidor de sonido multiplataforma capaz de funcionar por red, lo que permite que una aplicación pueda reproducir o grabar sonido en una máquina distinta a aquella en la que se está ejecutando. Usado por entornos de escritorio como KDE y GNOME. Posee una alta calidad de audio para mezcla con la posibilidad de usar varias fuentes.
Permite usar un amplio rango de bibliotecas cliente y aplicaciones como ESD, ALSA, oss, libao y GStreamer.

También existen complementos nativos de PulseAudio para xmms y mplayer

Licenciado bajo los términos de licencia GNU (General Public License). Pulse Audio tiene una arquitectura sencilla basada en plugin con soporte para carga de módulos.

GStreamer

Es un framework multimedia y multiĺataforma para el manejo de datos en distintos tipos de medios, que permite crear aplicaciones audiovisuales, en tareas complejas como mezclar audio y vídeo. Se basa en el uso de grafos de filtros que operan con los datos multimedia También provee una API para programadores.

Las aplicaciones que usan esta biblioteca pueden realizar tareas como procesar sonido en tiempo real, reproducir vídeos, y cualquier otra cosa relacionada con multimedia. Su arquitectura basada en complementos hace que los nuevos tipos de datos o funcionalidades de proceso se puedan añadir simplemente instalando nuevos complementos.

Xine

Xine es un motor de reproducción multimedia para sistemas operativos tipo Unix. Xine consiste en una biblioteca compartida llamada *xine-lib,* varios plugins y una interfaz gráfica o GUI. Muchos otros programas usan la biblioteca de xine para reproducción multimedia como por ejemplo, Amarok, Kaffeine, Tótem

[1]Demonio, en informática es un tipo especial de proceso no interactivo, es decir, se jecuta en segundo plano en lugar de ser controlado por el usuario.

o Phonon.

Phonon

Phonon es el framework multimedia estándar para utilizar las librerias gráficas de QT. Su objetivo es facilitar a los programadores el uso de tecnologías multimedia en sus programas, así como asegurar que las aplicaciones que usen Phonon funcionen en diversas plataformas y arquitecturas de sonido.

Phonon crea una capa intermedia entre los programas de KDE y los diferentes motores multimedia. Esto permite a los programas usar el API estable de Phonon, independientemente de los cambios que se hagan en dichos motores. Un cambio en un motor multimedia sólo requerirá adaptar Phonon a dicho cambio, en vez de tener que adaptar cada una de las aplicaciones que usen dicho motor.

INSTALACIÓN Y CONFIGURACIÓN ALSA

En cuanto a audio moderno el principio es ALSA, esto conecta con el núcleo y proporciona funcionalidad de audio al resto del sistema, puede mezclarse, proporcionar compatibilidad con otras capas, crear una API[2] para programadores y trabajar con una latencia tan baja y estable que pueda competir con los equivalentes ASIO y Core Audio en las plataformas Windows y OSx.

El modulo de Alsa ya viene cargado en el kernel desde la rama 2.6 en adelante, y esta llamado a sustituir a OSS.

Para mirar que versión de drivers de Alsa se encuentran instalados en el sistema

```
 $ cat /proc/asound/version
Advanced Linux Sound Architecture Driver Version 1.0.24.
```

ALSA es responsable de traducir las capacidades del hardware de audio de un dspositivo en una API de software que el resto de su sistema utiliza para manipular el sonido. Fue diseñado para hacer frente a muchas de las deficiencias de OSS (y la mayoría de los otros controladores de sonido en el momento), la más notable de las cuales era que sólo una aplicación tenia acceso al hardware a la vez. Es por esto que ALSA tiene un componente que gestiona las solicitudes de audio según las capacidades de su hardware

Verificar tarejtas de audio del equipo

```
$ cat /proc/asound/cards
 0 [NVidia         ]: HDA-Intel - HDA NVidia
                      HDA NVidia at 0xdfef4000 irq 22
 1 [SAA7134        ]: SAA7134 - SAA7134
                      saa7134[0] at 0xdfffc00 irq 19
```

si aparecen varias tarjetas y esto genera conflicto, sólo las que tengan asignada una IRQ serán físicamente las tarjetas de audio.

Para una instalación básica para desarrollo con la API se necesitan 3 paquetes, alsa-driver, alsa-base, alsa-

[2]API (Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos que ofrece una biblioteca para ser utilizados por otro software como una capa de abstracción.

utils y alsa-lib El paquete alsa-driver es el que viene con el núcleo, utilizarlo es en principio la opción más cómoda aunque puede traer complicaciones debido a las diferentes versiones que existen.

Para mirar la versión del driver en el sistema

```
$ /sbin/modinfo soundcore
filename:       /lib/modules/3.2.0-4-amd64/kernel/sound/soundcore.ko
alias:          char-major-14-*
license:        GPL
author:         Alan Cox
description:    Core sound module
depends:
intree:         Y
vermagic:       3.2.0-4-amd64 SMP mod_unload modversions
parm:           preclaim_oss:int
```

alsa-utils

alsa-utils contiene varias herramientas, alsamixer, aplay, arecord, etc.

```
$ sudo apt-get install alsa-base alsa-utils
```

alsa-lib

Los archivos alsa-lib se descargan de la página oficial del proyecto www.alsa-project.org. Estos son

```
alsa-lib-1.0.28.tar.bz2
```

Se crea el directorio alsa en /usr/src, se copian los archivos y se descomprimen aquí

```
$ mkdir /usr/src/alsa
$ cd archivos_alsa
$ cp *.bz2 /usr/src/alsa
$ cd /usr/src/alsa


$ sudo tar jxvf alsa-lib-1.0.28.tar.bz2

$ cd alsa-lib-1.0.28

$ sudo ./configure
...
Creating asoundlib.h...

$ sudo make install
```

Se verifica usando los camandos arecord y aplay

```
$ arecord -d 10 -c 2 -f S16_LE -r 44100 -D hw:0,0  -t wav test.wav
$ aplay -vv test.wav
```

INSTALACIÓN Y CONFIGURACIÓN JACK

La siguiente herramienta necesaria para la administracición de audio en LInux es el servidor Jack

```
$ sudo apt-get install jackd qjackctl
```

Se ejecuta por terminal como

```
$ qjackctl
```

Y se abre una ventana de control del servidor.

Fig 01. Ventana de control de Jack

En el botón de Setup se configura para que utilize el controlador de ALSA. Esta configuración define la interfaz hw:1 ya que el equipo tiene dos tarjetas de sonido. En una laptop lo más probable es que se deba definir la interfaz hw:0.

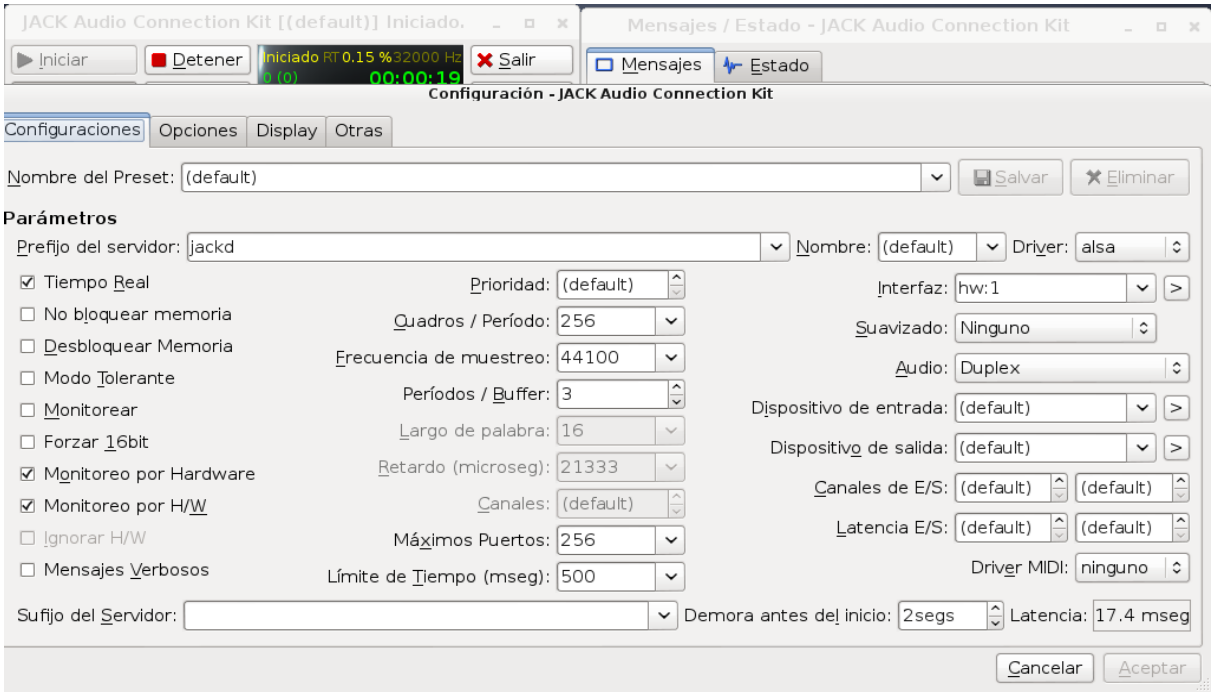

Fig 02. Ventana de configuración de Jack

El servidor ejecutándose solamente con un navegar abierto en el equipo. Se encuentran abiertas las ventanas de Conexiones y de Mensajes, necesarias para administrar y poder llevar un seguimiento del sistema.

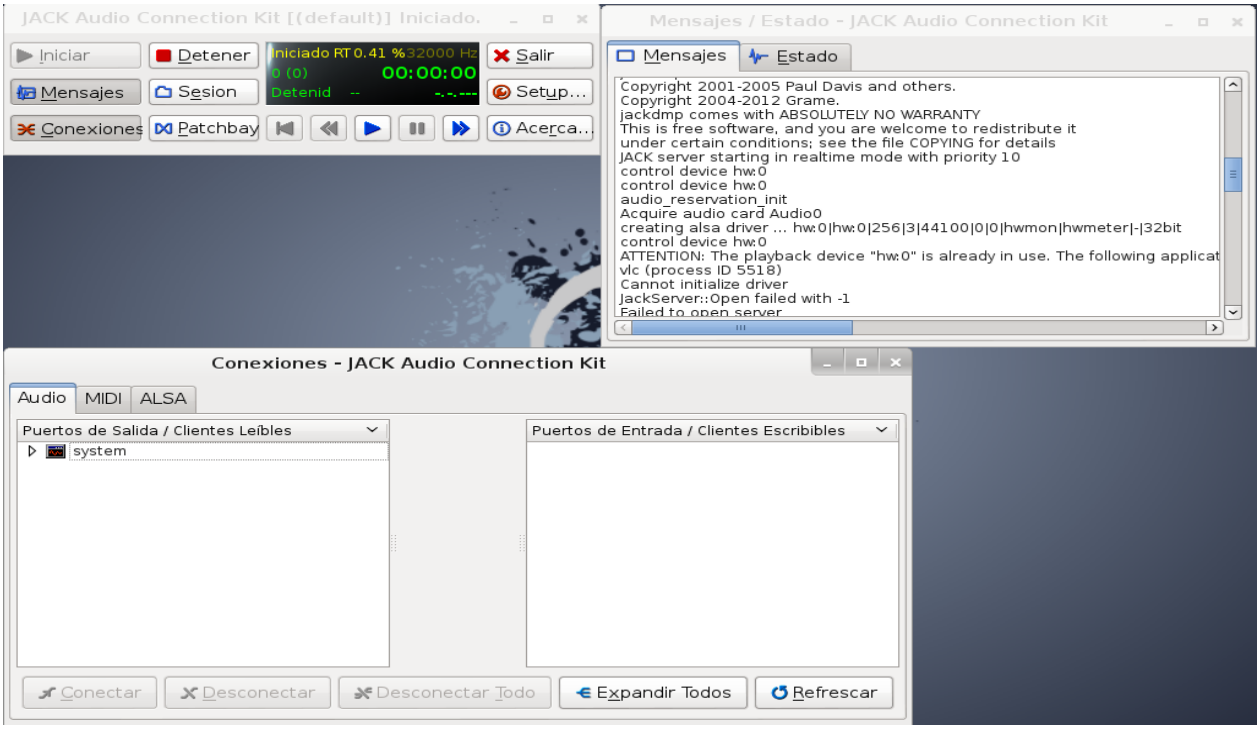

Fig 03. Ventanas de conexiones y mensajes con el servidor ejecutándose

Pueden presentarse errores al momento de ejecutar jack, es importante verificar los ficheros de configuración. En limits.conf se asegura que el modo en tiempo real este permitido por el sistema

```
$ sudo nano /etc/security/limits.conf
```

Debe tener las siguientes líneas al final del archivo

```
# End of file
@audio - rtprio 99
@audio - memlock unlimited
@audio - nice -19
```

También el usuario debe estar agregado al grupo audio del sistema. En el archivo /etc/group se listan los grupos del sistema y los usuarios agregados.

```
$ sudo nano /etc/group
```

Si el usuario no se encuentra agregado al grupo audio, se agrega con el siguiente comando

```
$ sudo adduser usuario audio
```

Donde usuario es el nombre de usuario que se va a agregar. Para remover un usuaurio con el comando deluser

```
$ sudo deluser usuarioAremover audio
```

Donde en este caso pulse es un usuario que se encontraba previamente configurado en el

sistema.

Complementos de Jack

Instalación JAAA

Jaaa (JACK & ALSA Audio Analyzer) es un generador de señal y analizador de espectro muy útil para el análisis de audio.

```
$ sudo apt-get install jaaa
```

Se ejecuta con la opción -J

```
$ jaaa -J
```

Se realizan las respectivas conexiones en jack

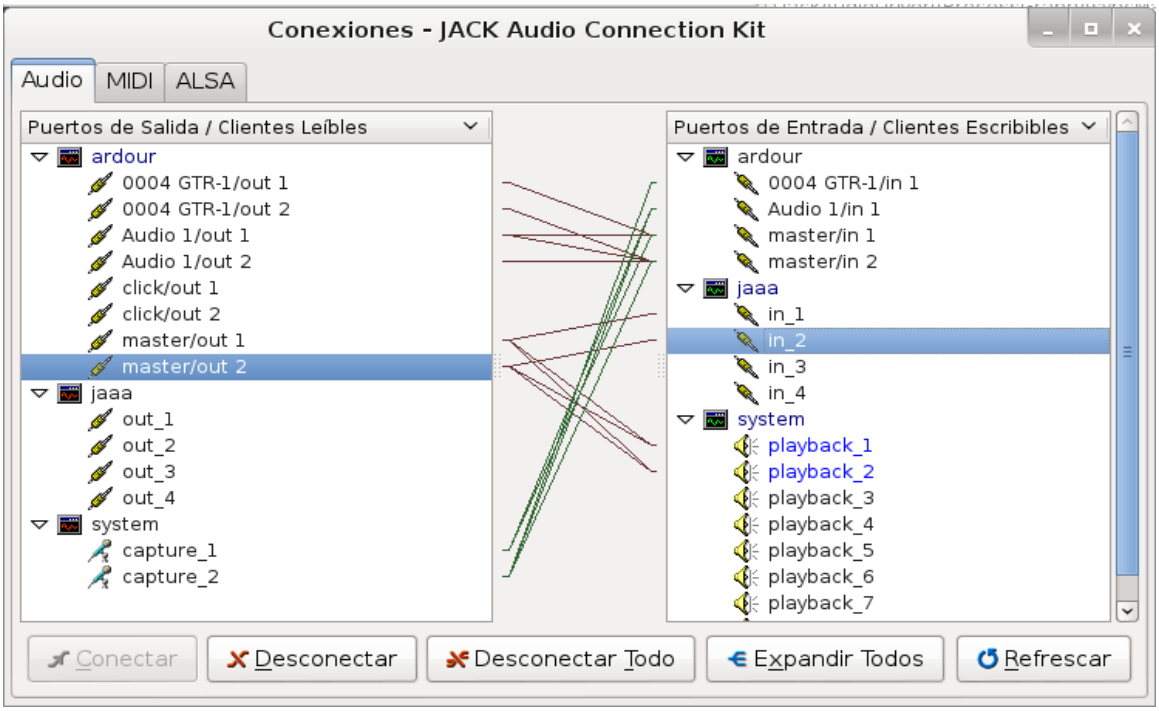

Fig 04. Ventana de conexiones para jaaa

En este caso se esta introduciendo jaaa en el flujo de señal de ardour, un DAW[3] para el trabajo de audio en Linux. Este recibe la señal que ardour monitorea en un canal de audio monofónico, se observa el análisis espectral.

[3]DAW (Digital Audio Workstation)

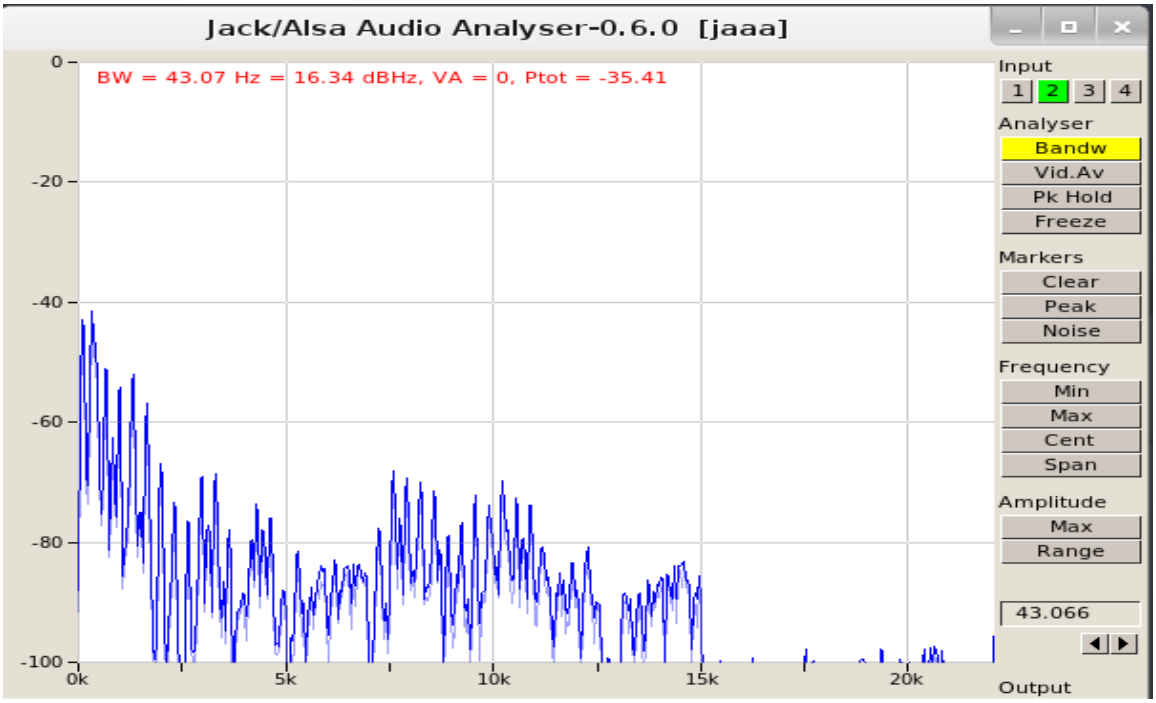

Fig 05. Análisis espectral de jaaa

Instalación japa

```
$ sudo apt-get install japa
```

Se ejecuta igualmente con la opción -J

```
$ japa -J
```

Se realizan las respectivas conexiones en jack

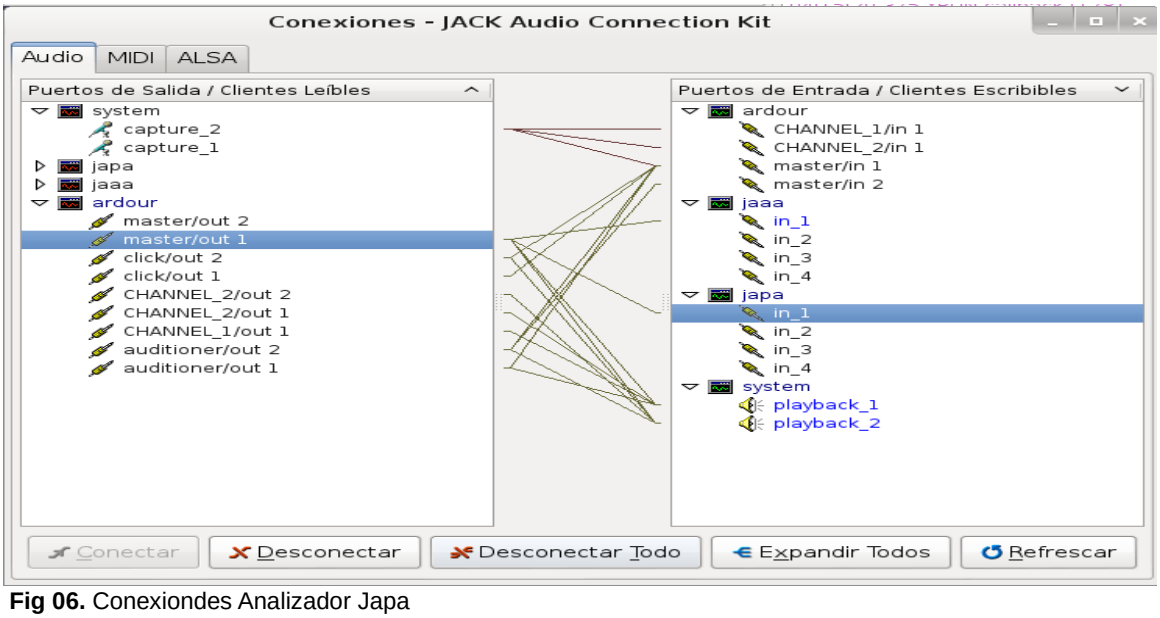

Fig 06. Conexiondes Analizador Japa

Se observa el análisis perceptual

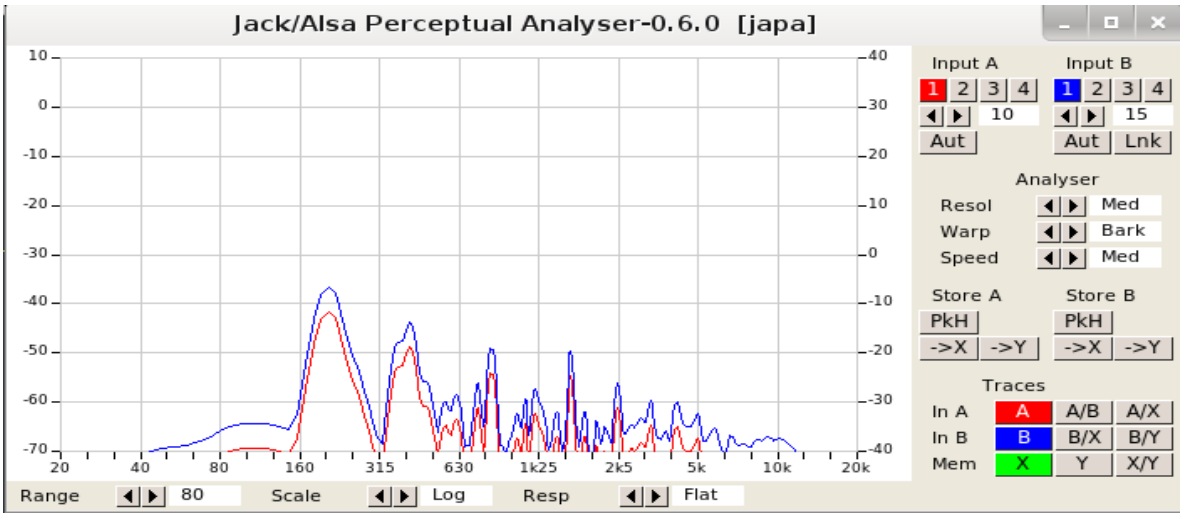

Fig 07. Analizador Perceptual Japa

Instalación jnoise

```
$ sudo apt-get install jnoise
```

Se ejecuta por terminal definiedo el rango de amplitud y en jack se realizan las conexiones del tipo de ruido a generar entre ruido blanco y ruido rosa.

```
$ jnoise -40
```

En este caso se generó ruido rosa a -40 dBFs.

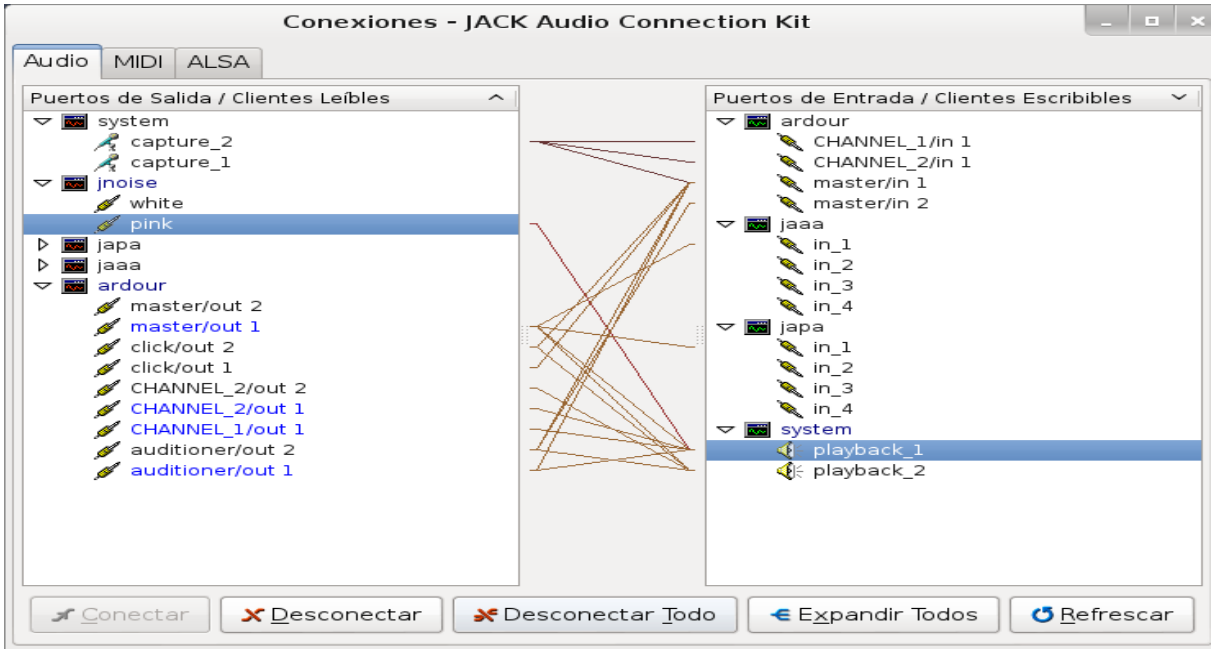

Fig 08. Conexión generador de ruido jnoise

Instalación Meterbridge

Colección de medidores de audio para el servidor jack

```
$ sudo apt-get install meterbridge
```

Se lanza por terminal con la opción de medidor a visualizar, en el caso del vúmetro de la figura 09.

```
$ meterbridge -t vu alsa_pcm:playback_1 alsa_pcm:playback_2
```

Otras opciones de medidores son

```
$ meterbridge -t ppm alsa_pcm:playback_1 alsa_pcm:playback_2
$ meterbridge -t dpm alsa_pcm:playback_1 alsa_pcm:playback_2
$ meterbridge -t jf alsa_pcm:playback_1 alsa_pcm:playback_2
$ meterbridge -t sco alsa_pcm:playback_1 alsa_pcm:playback_2
```

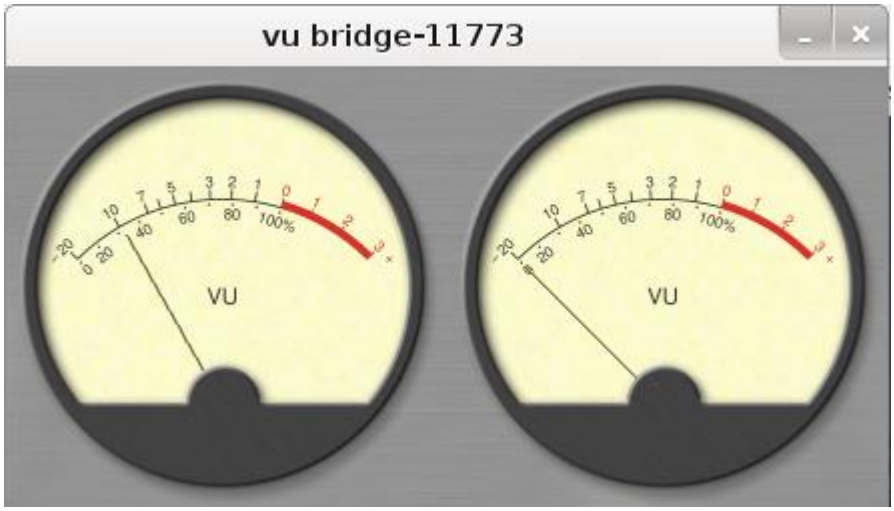

Fig 09. vúmetro de meterbridge

Se verifican conexiones en jack

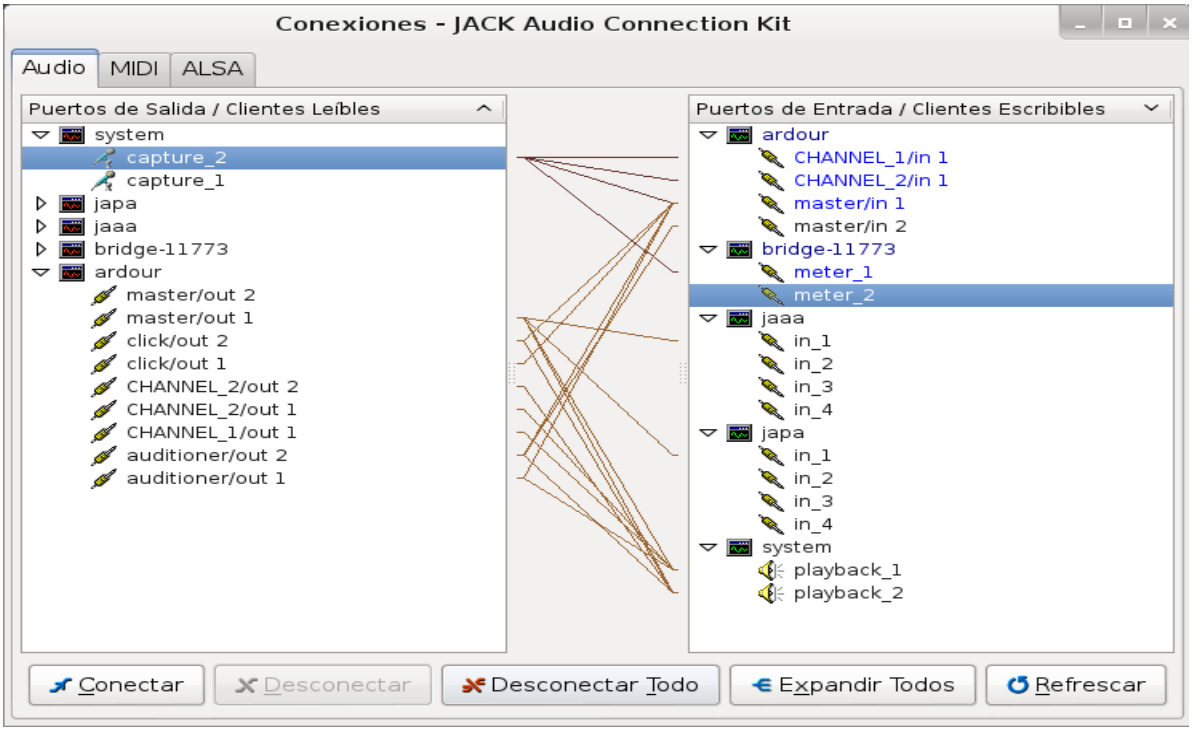

Fig 10. Conexiones módulo bridge-11773 de meterbridge

Instalación Timemachine

Timemachine es una herramienta de grabación de audio donde se pueden espeificar diferentes parámetros para la captura. Se instala igual desde los repositorios

```
$ sudo apt-get install timemachine
```

Se ejecuta con la opción -f para el tipo formato del archivo

```
$ timemachine -f wav
```

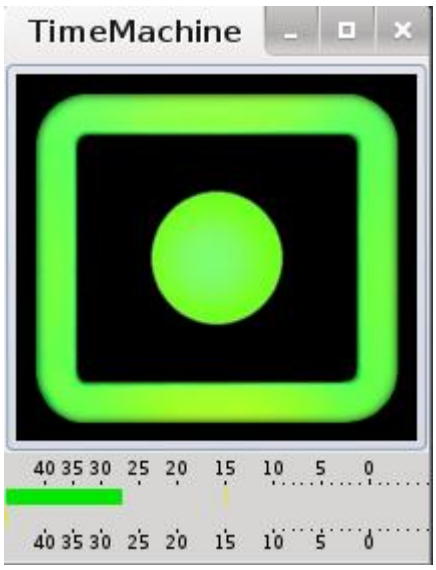

Fig 11. Monitoreo Timemachine

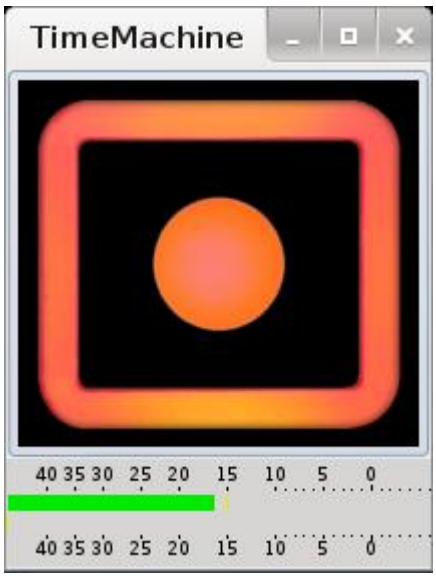

Fig 12. Gracación Timemachine

Se verifican conexiones en jack

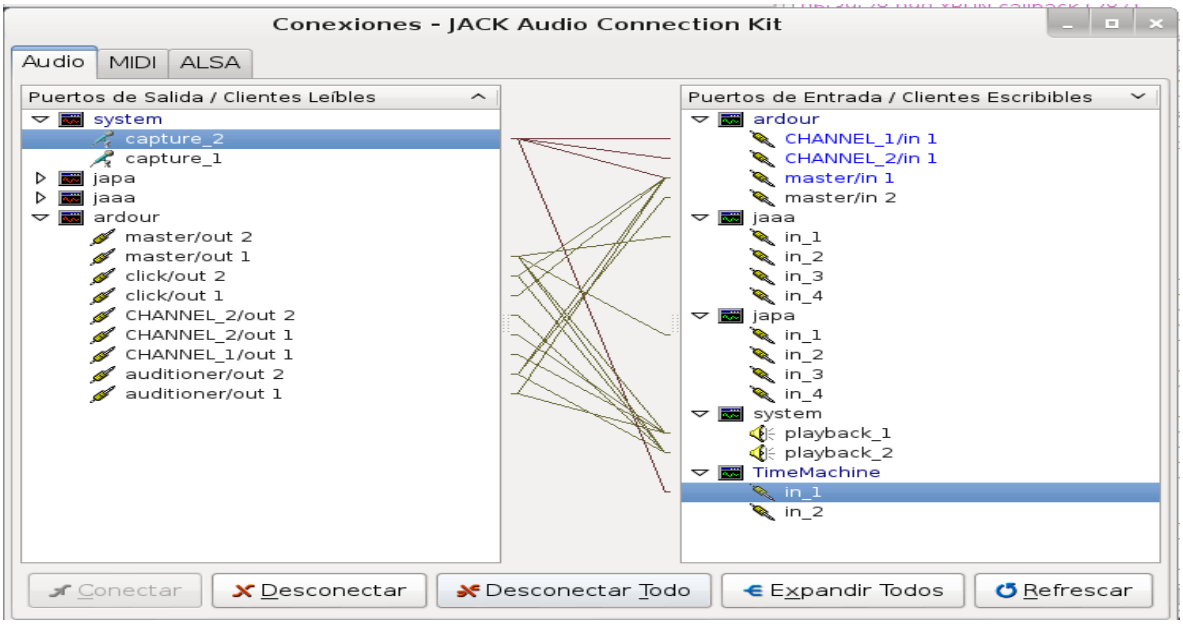

Fig 13. Conexiones Jack Timemachine

Configuración Jack con PulseAudio

Pulseaudio es el servidor de audio por defecto de algunas aplicaiones como navegadores weby reproductores de medios multimedia,

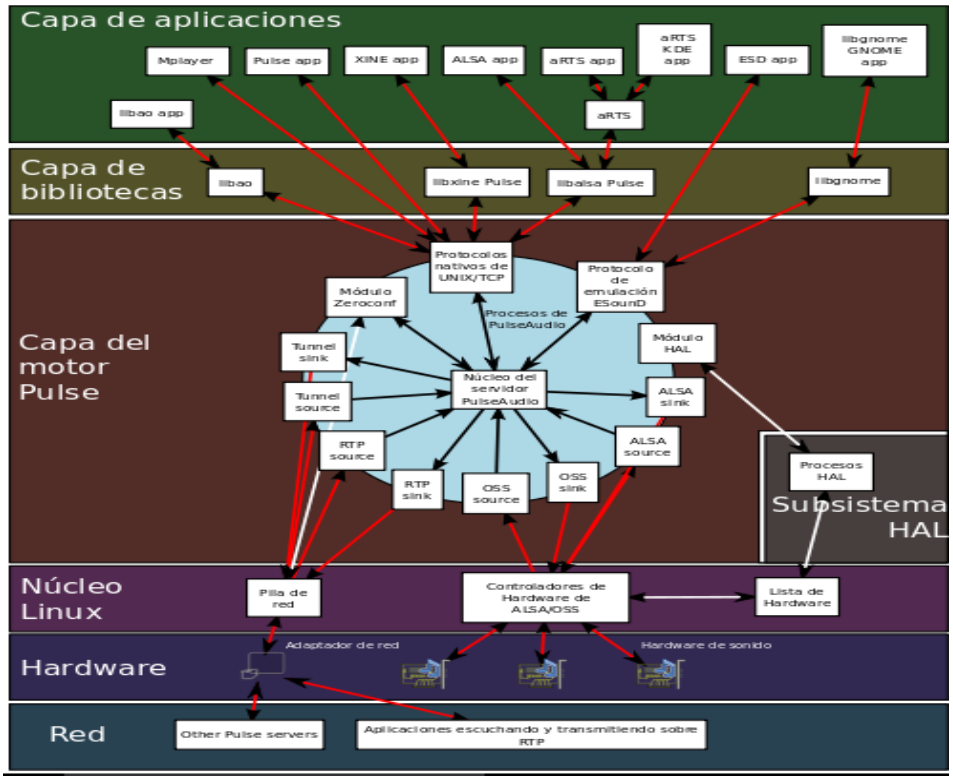

Fig 14. Arquitectura de PulseAudio

Para manejar el audio de aplicaciones que por defecto trabajan con pulseaudio es necesario instalar y activar un módulo de jack.

```
$ sudo apt-get install pulseaudio-module-jack
```

y se edita el fichero de configuración default.pa en /etc/pulse/

```
$ cd /etc/pulse
```

Se crea una copia del archivo, por seguridad

```
$ sudo cp default.pa default.pa.old
```

Y se edita el original

```
$ sudo nano -c default.pa
```

Se agregan la líneas

```
load-module module-jack-source
load-module module-jack-sink
```

Al final de la línea que posee la cadena

```
load-module module-pipe-sink
```

Y se comentarea todo el siguiente bloque que esta relacionado con *module-udev-detect,* para que

quede de la siguiente manera

```
### Automatically load driver modules depending on the hardware available
#.ifexists module-udev-detect.so
#load-module module-udev-detect
#.else
### Use the static hardware detection module (for systems that lack udev/hal support)
#load-module module-detect
#.endif
```

Se ejecuta en una terminal

```
$ pulseaudio --kill
```

Para detener el servidor pulseaudio, y visualizarlo en las conexiones de Jack

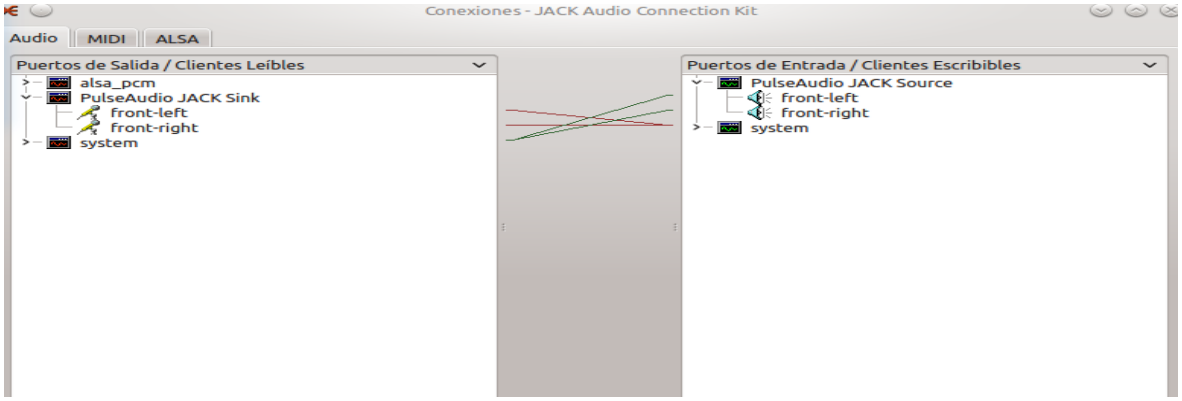

Fig 15. Cotrolador pulseaudio para jack

Con el comando

```
$ pulseaudio --start
```

Se reestablece el servicio al terminar de usar jack.

API DE ALSA

El API de la biblioteca de ALSA trabaja con nombres lógicos de dispositivos en lugar de archivos de dispositivos. Los nombres de dispositivos pueden ser dispositivos reales de hardware, o pueden ser plugins. Los dispositivos reales de hardware utilizan el formato hw:i,j, donde i es el núero de la tarjeta y j es el dispositivo de la tarjeta, por tanto, el primer dispositivo de sonido es hw:0,0.

Los Plugins utilizan otro identificador o nombre único, que es 'plughw', por ejemplo, un plugin puede permitir el acceso al dispositivo de hardare pero añade caraterísticas como la conversión de frecuencias de muestreo.

BUFFER DE AUDIO Y TRANSFERENCIA DE DATOS

Una tarjeta de sonido cuenta con un buffer de hardware que almacena las muestras registradas. Cuando el buffer se llena, se genera una interrupción, aquí el controlador de sonido del kernel utiliza el acceso directo a memoria (DMA)[4] para transferir muestras a un aplicación de buffer en memoria. De la misma manera para la reproducción de sonido, otra aplicación de buffer se transfiere de la memoria, al buffer de hardware de la tarjeta mediante DMA. Estos buffer de hardware son memorias cíclicas, es decir, cuando se alcanza el tope de la capacidad, los datos se devuelven al comienzo. Por esto se mantiene un apuntador para hacer seguimiento a las posiciones de memoria en ambas, tanto en buffer de harware como en buffer de aplicación, pero por fuera del núcleo sólo se utiliza el buffer de aplicación.

El tamaño del buffer puede ser programado por llamadas de la biblioteca de alsa, este puede ser lo bastante grande para causar demoras en la transferencia, denominadas **latencia**. Para solucionar esto ALSA divide el buffer en una serie de periodos y transfiere los datos en unidades de un (1) periodo. Existen dos modelos para esto, el modelo entrelazado y el modelo no entrelazado, pero en general un periodo almacena **frames** o **flujo de datos** , cada uno de los cuales contiene las muestras capturadas en un instante en el tiempo. En un dispositivo estéreo, los frames contienen muestras para dos canales.

La siguiente figura ilustra la descomposición de un buffer en periodos, que a su vez almacenas frames que a su vez almacenan muestras, en este caso estéro. También muestra el apuntador de posición.

[4]DMA : Acceso directo a memoria o Direct Acces Memory permite a a ciertos componentes de un ordenador acceder a la memoria del sistema para lectura y/o escritura.

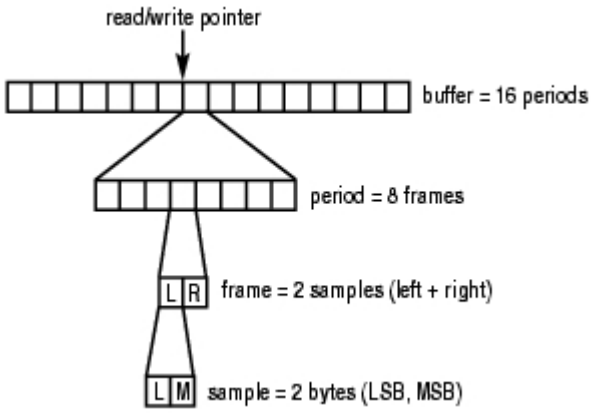

Fig 16. Arquitectura Buffer de Aplicación

Esta descomposición es el modelo entrelazado. En el modelo no entrelazado todas las muestras para un canal se almacenan seguidas de todas las muestrs para el otro canal.

Cuando un dispositivo de sonido se activa, se transfieren datos continuamente entre el hardware y el buffer de aplicación. En grabación por ejemplo, si la aplicación no lee los datos en el buffer con suficiente rapidez, el buffer (que es cíclico) se sobreescribe causando pérdida de los datos, a esto se le llama rebasamiento. Durante la reproducción, si la aplicación no transfiere los datos dentro del buffer con suficiente rapidez, se comienza a producir carencia de datos, lo que resulta en un error llamado empotramiento. o insuficiencia de datos.

La documentación de alsa se suele referir a estas dos condiciones utilizando el término XRUN. Las aplicaciones correctamente diseñadas pueden minimizar XRUN y reuperar si se produce.

Los problemas más comunes para los búferes son:

- Desbordamiento de búfer (*overflow*), también conocido como *overrun*, el cual se da cuando el tamaño del búfer es insuficiente para almacenar la información que se desea ingresar normalmente ocasionado por un problema de programación en donde el programa a usar el búfer malinterpreta el tamaño de dicho búfer.
- Subdesbordamiento de búfer (*underflow*), también conocido como *underrun*, es un problema generado por la falta de información en el búfer cuando es requerida por el otro programa ocasionada generalmente por un error de tiempos, velocidad inestable de transferencia de datos.
- Sobreescritura, también conocido como *overwrite*, es un problema generalmente creado por la mala programación del búfer ya que se escribe información en espacios de memoria donde todavía no se utiliza esa información ocasionando la perdida de dicha información.

A continuación se muestran algunas de las formas en que se administran los diferentes tipos de búferes, cada método a utilizar depende de la información a tratar o de los procesos que la van a utilizar.

1. Almacenamiento temporal único por demanda: este sistema no consta más que de el búfer y el programa que lo utiliza. El búfer es simplemente un algoritmo de cola utilizado por el programa, por lo que es este último quien le especifica cuándo lo va a utilizar por medio de demandas.
2. Almacenamiento temporal por anticipación: en este caso, se le añade una bandera (*flag*) al búfer, que va a estar cambiando de valor constantemente para indicar cuando este está lleno o vacío. Esto con el fin de que

el búfer se anticipe a las demandas del programa y pueda llenarse por sí mismo cuando se detecte que ya está vacío.

3. Almacenamiento temporal con bloques: aparte de la bandera, en este caso, el búfer también tiene una variable contador, que le permite saber el índice del último registro que pudo almacenar en su memoria. Así, cuando se leen bloques grandes de información, el búfer sabe exactamente cuál es el siguiente registro a procesar.
4. Doble almacenamiento temporal: este método consta de dos búferes idénticos al del caso del almacenamiento temporal con bloques, y estos trabajan de forma coordinada almacenando información que el otro no alcanzó a almacenar antes de llenarse. La idea es que un búfer se llene mientras el otro se vacía, con el fin de optimizar el proceso.
5. Triple almacenamiento temporal: en este caso, se cuenta con tres búferes coordinados. Al menos uno siempre tiene que estar lleno, así el programa puede estar procesando información sin interrupciones.

Una típica aplicación de sonido que utilice la interfaz PCM sigue el siguiente pseudo-código.

Abrir interfaz
ya sea para captura o reproducción

Fijar parámetros de hardware
(modos de acceso, formato de datos, canales, muestreo, etc**)**

MIentras haya datos para ser procesados,
Leer datos PCM (captura)
o escribir datos PCM (reproducción)

Cerrar interfaz

Para empezar a trabajar con el API de alsa se incluye en los headers del código la librería **asoundlib.h**, y para compialar se vincula el archivo con la biblioteca de ALSA libasound, incluye el parámetro **-lasound** al final de la línea de compilación, por ejemplo :

```
$ gcc -Wall programaAlsa.c -o programaAlsa_exec -lasound
```

para cross-compilarl el programa se incluye la ruta de las librerias compiladas para la arquitectura deseada de ALSA

```
$ sudo /usr/local/arm/4.3.2/bin/arm-none-linux-gnueabi-gcc programaAlsa.c -o programaAlsa_ARM -L
/home/goss/alsaARM/alsalibfiles/usr/lib -I /home/goss/alsaARM/alsalibfiles/usr/include -lasound
```

El siguiente es un ejemplo de un programa en C, que muestra algunos de los tipos de datos PCM y parámetros utilizados por ALSA.

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asound.h>

int main()
{
        int val;
        printf("==================================\n");
        printf("  Parametros PCM\n");
```

```
        printf("==================================\n");
        printf("Version libreria de Alsa\n");
        printf("  %s\n\n",SND_LIB_VERSION_STR);

        printf("Tipos de stream PCM\n\n");
                for(val=0 ; val <= SND_PCM_STREAM_LAST ; val++)
                {
                        printf("  %s\n",
                            snd_pcm_stream_name((snd_pcm_stream_t)val));
                }

        printf("\nTipos de acceso PCM\n\n");
                for (val = 0; val <= SND_PCM_ACCESS_LAST; val++)
                {
                   printf("  %s\n",
                     snd_pcm_access_name((snd_pcm_access_t)val));
                }
        printf("\nEstados PCM\n\n");
                  for (val = 0; val <= SND_PCM_STATE_LAST; val++)
                  {
                   printf("  %s\n",snd_pcm_state_name((snd_pcm_state_t)val));
                  }

        printf("\n");

        return 0;
}
```

Se inicia incluyendo las librerias de alsa en los headers del archivo, luego se accede a cada tipo de flujo de datos, mostrando una cadena descriptiva de cada valor a la salida.

En el siguiente ejemplo se abre el dispositivo PCM, se fija parámetros en él, y se imprimen por pantalla estos parámetros.

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

int main() {
  int rc;
  snd_pcm_t *controlador;
  snd_pcm_hw_params_t *parametros;
  unsigned int val, val2;
  int dir;
  snd_pcm_uframes_t frames;

  /* Abrir dispositivo PCM para reproducción. */
  rc = snd_pcm_open(&controlador, "hw:0,0", SND_PCM_STREAM_PLAYBACK, 0);
            if (rc < 0)
            {
                        fprintf(stderr,"Problema al abrir dispositivo PCM:\n%s\n",
        snd_strerror(rc));
                        exit(1);
            }
```

```
/* Asignar las configuraciónes parámetros a parametros mediante alloca */
snd_pcm_hw_params_alloca(&parametros);

/*Inicializar la variable  con _any que es la que recibe
  el flujo pcm abierto anteriormente */
snd_pcm_hw_params_any(controlador, parametros);

/* Fijar los parámetros que se quieren tener */

/* Modo de acceso, en este caso RW_INTERLEAVED */
snd_pcm_hw_params_set_access(controlador, parametros,
            SND_PCM_ACCESS_RW_INTERLEAVED);

/* Formato,  16 bits */
snd_pcm_hw_params_set_format(controlador, parametros,
                  SND_PCM_FORMAT_S16_LE);

/* Canales, 2 (stereo) */
snd_pcm_hw_params_set_channels(controlador, parametros, 2);

/* Frecuencia de muestreo */
val = 44100;
snd_pcm_hw_params_set_rate_near(controlador,parametros, &val, &dir);

/* Escribe los parámetros en el driver */
rc = snd_pcm_hw_params(controlador, parametros);

          if (rc < 0)
          {
                    fprintf(stderr,
       "unable to set hw parameters: %s\n",
       snd_strerror(rc));
                    exit(1);
          }

/* Imprime por pantalla información de la interfaz PCM  */

printf("Nombre controlador PCM = '%s'\n",snd_pcm_name(controlador));

printf("Estado de PCM  = %s\n",snd_pcm_state_name(snd_pcm_state(controlador)));

snd_pcm_hw_params_get_access(parametros,(snd_pcm_access_t *) &val);
printf("Tipo de Acceso = %s\n",snd_pcm_access_name((snd_pcm_access_t)val));

snd_pcm_hw_params_get_format(parametros, &val);
printf("Formato       = '%s' (%s)\n",snd_pcm_format_name((
snd_pcm_format_t)val),snd_pcm_format_description((snd_pcm_format_t)val));

snd_pcm_hw_params_get_subformat(parametros,(snd_pcm_subformat_t *)&val);
printf("Sub-formato    = '%s' (%s)\n",
 snd_pcm_subformat_name((snd_pcm_subformat_t)val),
 snd_pcm_subformat_description(
            (snd_pcm_subformat_t)val));
```

```
snd_pcm_hw_params_get_channels(parametros, &val);
printf("Canales       = %d\n", val);

snd_pcm_hw_params_get_rate(parametros, &val, &dir);
printf("Freq Muestreo  = %d bps\n", val);

snd_pcm_hw_params_get_period_time(parametros,&val, &dir);
printf("period time    = %d us\n", val);

snd_pcm_hw_params_get_period_size(parametros,&frames, &dir);
printf("period size    = %d frames\n", (int)frames);

snd_pcm_hw_params_get_buffer_time(parametros,&val, &dir);
printf("buffer time    = %d us\n", val);

snd_pcm_hw_params_get_buffer_size(parametros,(snd_pcm_uframes_t *) &val);
printf("buffer size    = %d frames\n", val);

snd_pcm_hw_params_get_periods(parametros, &val, &dir);
printf("periods per buffer = %d frames\n", val);

snd_pcm_hw_params_get_rate_numden(parametros,&val, &val2);
printf("exact rate    = %d/%d bps\n", val, val2);

val = snd_pcm_hw_params_get_sbits(parametros);
printf("significant bits = %d\n", val);

snd_pcm_hw_params_get_tick_time(parametros,&val, &dir);
printf("tick time = %d us\n", val);

val = snd_pcm_hw_params_is_batch(parametros);
printf("is batch = %d\n", val);

val = snd_pcm_hw_params_is_block_transfer(parametros);
printf("is block transfer = %d\n", val);

val = snd_pcm_hw_params_is_double(parametros);
printf("is double = %d\n", val);

val = snd_pcm_hw_params_is_half_duplex(parametros);
printf("is half duplex = %d\n", val);

val = snd_pcm_hw_params_is_joint_duplex(parametros);
printf("is joint duplex = %d\n", val);

val = snd_pcm_hw_params_can_overrange(parametros);
printf("can overrange = %d\n", val);

val = snd_pcm_hw_params_can_mmap_sample_resolution(parametros);
printf("can mmap = %d\n", val);

val = snd_pcm_hw_params_can_pause(parametros);
printf("can pause = %d\n", val);

val = snd_pcm_hw_params_can_resume(parametros);
```

```
 printf("can resume = %d\n", val);

 val = snd_pcm_hw_params_can_sync_start(parametros);
 printf("can sync start = %d\n", val);

 snd_pcm_close(controlador);

 return 0;
}
```

En este programa no se realizó ningún tipo de captura o reproducción de sonido. En la llamada a **snd_pcm_open,** se abre el dispositivo pcm **hw:0,0** y se establece en modo de reproducción. Esta función retorna un controlador que tomaran otras funciones como primer argumento en las llamadas para manipular el flujo de pcm.

Como en la mayoría de las llamadas a la biblioteca ALSA, la función devuelve un estado de retorno entero, un valor negativo indica una condición de error.

Con el fin de establecer los parámetros de hardware para el flujo de datos, es necesario asignarlos a la variable parametros que es de tipo snd_pcm_hw_params_t, esto se hace con la estructura snd_pcm_hw_params_alloca, enseguida se inicializa la variable mediante la función snd_pcm_hw_params_any pasandole el flujo pcm abierto previamente.

En seguida se establecen los parámetros a utilizar mediante las llamadas a la API que toman el flujo del controlador (hw:0,0).

En este caso se usó el modo entrelazado, un tamaño de muestra de 16 bits a 2 canales y con una frecuencia de muestreo de 44100 bps..

Los parámetros de hardware no están realmente activos hasta la llamada a la función **snd_pcm_hw_params**.

El resto del programa obtiene y muestra algunos de los parámetros del flujo PCM, incluyendo el periodo y la longitud del buffer. El resultado varía dependiendo del hardware de sonido.

En el siguiente programa se muestra un ejemplo de playback

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

int main()
{
        int pcm, tmp,dir;
        int channels, seconds, buff_size, loops;
        unsigned int rate;
        char *buffer;
        snd_pcm_t *controlador;                 //apuntador para usar el dispositivo pcm
```

```
snd_pcm_hw_params_t *parametros;//apuntador para fijar parámetros
snd_pcm_uframes_t frames;

/*Abro dispositivo pcm y defino que se va a usar para reproducción,
 paso como parámetro la dirección de memoria del controlador
 */
        pcm = snd_pcm_open(&controlador, "hw:0,0", SND_PCM_STREAM_PLAYBACK,0);
                if(pcm < 0)
                {
                        fprintf(stderr,"Problema al abrir dispositivo PCM",
                                        snd_strerror(pcm));
                        exit(1);
                }

/*se llama a la estructura que almacena las configuraciones de parametros de
 hardware mediante _alloca
 */
        snd_pcm_hw_params_alloca(&parametros);

/*Inicializa parámetros con la función _any que es la que recibe el flujo pcm
 anteriormente abierto
*/
        snd_pcm_hw_params_any(controlador, parametros);

/*Establecer los parámetros de Hardware deseados*/

        if ((pcm = snd_pcm_hw_params_set_access(controlador, parametros,
                                                SND_PCM_ACCESS_RW_INTERLEAVED)) < 0)
                printf("ERROR: Can't set interleaved mode. %s\n", snd_strerror(pcm));

        if ((pcm = snd_pcm_hw_params_set_format(controlador,parametros,
                                                SND_PCM_FORMAT_S16_LE)) < 0)
                printf("ERROR: Can't set format. %s\n", snd_strerror(pcm));

        if ((pcm = snd_pcm_hw_params_set_channels(controlador, parametros, 1)) < 0)
                printf("ERROR: Can't set channels number. %s\n", snd_strerror(pcm));

        rate = 44100;
        if ((pcm = snd_pcm_hw_params_set_rate_near(controlador, parametros, &rate, NULL)) < 0)
                printf("ERROR: Can't set rate. %s\n", snd_strerror(pcm));

        frames = 64;
        snd_pcm_hw_params_set_period_size_near(controlador,parametros, &frames, &dir);

        /*Escribir los parámetros en el driver
         */
        pcm = snd_pcm_hw_params(controlador, parametros);
                if (pcm < 0)
                {
                  fprintf(stderr, "unable to set hw parameters: %s\n",snd_strerror(pcm));
                  exit(1);
                }


        snd_pcm_hw_params_get_channels(parametros, &channels);
```

```
                printf("channels: %i ", channels);

                if (channels == 1)
                        printf("(mono)\n");
                else if (channels == 2)
                        printf("(stereo)\n");

        snd_pcm_hw_params_get_rate(parametros, &tmp, &dir);
                printf("rate: %d bps\n", tmp);

                printf("seconds: %d\n", seconds);

        snd_pcm_hw_params_get_format(parametros, &tmp);
                printf("Formato : '%s' (%s)\n",snd_pcm_format_name((
        snd_pcm_format_t)tmp),snd_pcm_format_description((snd_pcm_format_t)tmp));


        snd_pcm_hw_params_get_period_size(parametros, &frames, &dir);
                printf("Period_size: %i frames\n", frames);

        buff_size = frames * 2 * channels; /* 2 bites por sample */
        buffer = (char *) malloc(buff_size);

        snd_pcm_hw_params_get_period_time(parametros, &tmp, &dir);
                printf("Period_time: %i us\n", tmp);
        for(loops = 512; loops > 0; loops--)
                {
                if((pcm = read(0,buffer,buff_size)) == 0)
                {
                        fprintf(stderr,"Early end of file\n");
                        break;
                }
                if((pcm = snd_pcm_writei(controlador, buffer, frames)) == -EPIPE)
                {
                        fprintf("XRUN\n",pcm);
                        snd_pcm_prepare(controlador);
                }
                else if(pcm < 0)
                {
                printf("NO se puede escribir dispositivo\n",snd_strerror(pcm));
                }
        }

        snd_pcm_drain(controlador);
        snd_pcm_close(controlador);
        free(buffer);

        return 0;
}
```

Este programa puede ser usado para ejecutar archivos o fuentes de audio en general

```
$ ./programaPlayback < /dev/urandom
```

lo que devolvera ruido blanco a la salida

```
$ ./programaPlayback > /path/to/sound.wav
```

El siguiente programa ofrece una captura que puede ser redireccionada a un archivo de extensión .wav

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

int main()
{
        int pcm, tmp,dir;
        int channels, seconds, buff_size;
        int loops;
        unsigned int rate;
        char *buffer;
        snd_pcm_t *controlador;                 //apuntador para usar el dispositivo pcm
        snd_pcm_hw_params_t *parametros;//apuntador para fijar parámetros
        snd_pcm_uframes_t frames;

        /*Abro dispositivo pcm y defino que se va a usar para reproducción,
         paso como parámetro la dirección de memoria del controlador
         */
                pcm = snd_pcm_open(&controlador, "default", SND_PCM_STREAM_CAPTURE,0);
                        if(pcm < 0)
                        {
                                fprintf(stderr,"Problema al abrir dispositivo PCM",
                                                snd_strerror(pcm));
                                exit(1);
                        }
/*se llama a la estructura que almacena las configuraciones de parametros de
          hardware mediante _alloca
         */
                snd_pcm_hw_params_alloca(&parametros);

        /*Inicializa parámetros con la función _any que es la que recibe el flujo pcm
         anteriormente abierto
   */
                snd_pcm_hw_params_any(controlador, parametros);

        /*Establecer los parámetros de Hardware deseados*/

                if ((pcm = snd_pcm_hw_params_set_access(controlador, parametros,
                                                        SND_PCM_ACCESS_RW_INTERLEAVED))< 0)
                printf("ERROR: Can't set interleaved mode. %s\n", snd_strerror(pcm));

                if ((pcm = snd_pcm_hw_params_set_format(controlador,parametros,
                                                        SND_PCM_FORMAT_S16_LE)) < 0)
                printf("ERROR: Can't set format. %s\n", snd_strerror(pcm));

                if ((pcm=snd_pcm_hw_params_set_channels(controlador, parametros, 1))< 0)
                printf("ERROR: Can't set channels number. %s\n", snd_strerror(pcm));
```

```
rate = 44100;
if ((pcm=snd_pcm_hw_params_set_rate_near(controlador,parametros,&rate, &dir))<0)
                        printf("ERROR: Can't set rate. %s\n", snd_strerror(pcm));

frames = 64;
snd_pcm_hw_params_set_period_size_near(controlador,parametros, &frames, &dir);

/*Escribir los parámetros en el driver
 */
pcm = snd_pcm_hw_params(controlador, parametros);
if (pcm < 0)
{
fprintf(stderr, "unable to set hw parameters: %s\n",snd_strerror(pcm));
exit(1);
}


snd_pcm_hw_params_get_channels(parametros, &channels);
            printf("channels: %i ", channels);

            if (channels == 1)
                        printf("(mono)\n");
            else if (channels == 2)
                        printf("(stereo)\n");

snd_pcm_hw_params_get_rate(parametros, &tmp, &dir);
            printf("rate: %d bps\n", tmp);

            printf("seconds: %d\n", seconds);

snd_pcm_hw_params_get_format(parametros, &tmp);
            printf("Formato       = '%s' (%s)\n",snd_pcm_format_name((
snd_pcm_format_t)tmp),snd_pcm_format_description((snd_pcm_format_t)tmp));

snd_pcm_hw_params_get_period_size(parametros, &frames, &dir);
            printf("Period_size: %i\n", frames);

buff_size = frames * 2 * channels;
buffer = (char *) malloc(buff_size);

snd_pcm_hw_params_get_period_time(parametros, &tmp, &dir);
            printf("Period_time: %i\n", tmp);

for(loops = ((10 * 1000000)/(tmp)) ; loops > 0 ; loops--)
{
            if((pcm = snd_pcm_readi(controlador, buffer, frames)) == -EPIPE)
            {
                        fprintf(stderr,"OVERRUN\n");
                        snd_pcm_prepare(controlador);
            }
            else if(pcm < 0)
            {
                        fprintf(stderr,"ERROR DE LECTURA\n",snd_strerror(pcm));
            }
```

```
                else if((pcm = write(1,buffer,buff_size)) != buff_size)
                {
                        fprintf(stderr, "Lectura Corta :\n %d bytes leidos\n",pcm);
                }
        }

        snd_pcm_drain(controlador);
        snd_pcm_close(controlador);
        free(buffer);

    printf("\n");
    return 0;
}
```

Puede ser ejecutado

```
$ ./programaCaptura > sound.wav
```

Ambos programas pueden ser ejecutados si la tarjeta es FULL DUPLEX

```
$ ./programaCaptura | ./programaPlayback
```

De esta manera.

El siguiente programa captura datos y los guarda en un archivo de extensión .wav, para esto es necesario crear el archivo wav y reescribirlo.

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

int main()
{
    FILE *fp;
    int pcm, tmp,dir,i=0;
    int channels, seconds, buff_size;
    int loops;
    unsigned int rate;
    char *buffer;
    snd_pcm_t *controlador;                 //apuntador para usar el dispositivo pcm
    snd_pcm_hw_params_t *parametros;//apuntador para fijar parámetros
    snd_pcm_uframes_t frames;

    /*Abro o creo archivo WAV con permisos de escritura
     */
        if((fp = fopen("sound.wav","w")) < 0)
                {
                        printf("Error al abrir archivo wav\n");

                }
```

```
        /*Abro dispositivo pcm y defino que se va a usar para reproducción,
          paso como parámetro la dirección de memoria del controlador
        */
                pcm = snd_pcm_open(&controlador, "default", SND_PCM_STREAM_CAPTURE,0);

                        if(pcm < 0)
                                {
                                        fprintf(stderr,"Problema al abrir dispositivo PCM",
                                                                snd_strerror(pcm));
                                        exit(1);
                                }

        /*se llama a la estructura que almacena las configuraciones de parametros de
          hardware mediante _alloca
        */

                snd_pcm_hw_params_alloca(&parametros);

        /*Inicializa parámetros con la función _any que es la que recibe el flujo pcm
          anteriormente abierto
        */

                snd_pcm_hw_params_any(controlador, parametros);
        /*
          Establecer los parámetros de Hardware deseados
         */

                if ((pcm = snd_pcm_hw_params_set_access(controlador, parametros,
                                        SND_PCM_ACCESS_RW_INTERLEAVED))< 0)
                printf("ERROR: Can't set interleaved mode. %s\n", snd_strerror(pcm));

                if ((pcm = snd_pcm_hw_params_set_format(controlador,parametros,
                                                                SND_PCM_FORMAT_S16_LE)) < 0)
                                        printf("ERROR: Can't set format. %s\n", snd_strerror(pcm));

        if ((pcm = snd_pcm_hw_params_set_channels(controlador, parametros, 1)) < 0)
                printf("ERROR: Can't set channels number. %s\n", snd_strerror(pcm));

                rate = 44100;
if ((pcm = snd_pcm_hw_params_set_rate_near(controlador, parametros, &rate, &dir))< 0)
                                        printf("ERROR: Can't set rate. %s\n", snd_strerror(pcm));

                frames = 1024;
snd_pcm_hw_params_set_period_size_near(controlador,parametros, &frames, &dir);

        /*Escribir los parámetros en el driver
        */

                pcm = snd_pcm_hw_params(controlador, parametros);
                                        if (pcm < 0)
                                        {
                                          fprintf(stderr, "unable to set hw parameters:
          %s\n",snd_strerror(pcm));
                                          exit(1);
```

```
                        }

        snd_pcm_hw_params_get_channels(parametros, &channels);
                printf("channels: %i ", channels);

                if (channels == 1)
                        printf("(mono)\n");
                else if (channels == 2)
                        printf("(stereo)\n");

        snd_pcm_hw_params_get_rate(parametros, &tmp, &dir);
                printf("rate: %d bps\n", tmp);

                printf("seconds: %d\n", seconds);

        snd_pcm_hw_params_get_format(parametros, &tmp);
printf("Formato   : '%s' (%s)\n",snd_pcm_format_name((
snd_pcm_format_t)tmp),snd_pcm_format_description((snd_pcm_format_t)tmp));


        snd_pcm_hw_params_get_period_size(parametros, &frames, &dir);
                printf("Period_size: %i\n", frames);

        buff_size = frames * 2 * channels;
        buffer = (char *) malloc(buff_size);

        snd_pcm_hw_params_get_period_time(parametros, &tmp, &dir);
                printf("Period_time: %i\n", tmp);

                for(loops = ((10 * 1000000)/(tmp)) ; loops > 0 ; loops--)
                {
                        pcm = snd_pcm_readi(controlador, buffer, frames);
                        printf("%d\n",i++);
                        if(pcm == -EPIPE)
                        {
                                fprintf(stderr,"OVERRUN\n");
                                snd_pcm_prepare(controlador);
                        }

                        else if(pcm < 0)
                        {
                        fprintf(stderr,"ERROR DE LECTURA\n",snd_strerror(pcm));
                        }
                        else if( pcm != (int)frames)
                        {
                        fprintf(stderr, "Lectura Corta :\n %d bytes leidos\n",pcm);
                        }

                        pcm = fwrite( buffer,1, buff_size,fp);

                        if (pcm != buff_size)
                        {
                        fprintf(stderr,  "short write: wrote %d bytes/n", pcm);
                        }
                        else
```

```
                                        {
                                                printf("fwrite buffer success\n");
                                        }
                                }

                        snd_pcm_drain(controlador);
                        snd_pcm_close(controlador);
                        fclose(fp);
                        free(buffer);

                printf("\n");
                return 0;
}
```

El siguiente programa reproduce un archivo de externsión .wav

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

int main()
{
        FILE *fp;
        int pcm, tmp,dir,i=0;
        int channels, seconds, buff_size;
        int loops;
        unsigned int rate;
        char *buffer;
        snd_pcm_t *controlador;                 //apuntador para usar el dispositivo pcm
        snd_pcm_hw_params_t *parametros;   //apuntador para fijar parámetros
        snd_pcm_uframes_t frames;

        /*Abro archivo WAV
         */
                if((fp = fopen("sound.wav","r")) < 0)
                        {
                                printf("Error al abrir archivo wav\n");

                        }
                if(fseek(fp,0,SEEK_SET) < 0)
                {
                        printf("put fp start to first error\n");
                }

/*Abro dispositivo pcm y defino que se va a usar para reproducción,
          paso como parámetro la dirección de memoria del controlador
        */
                pcm = snd_pcm_open(&controlador, "default", SND_PCM_STREAM_PLAYBACK,0);

                        if(pcm < 0)
                                {
                                        fprintf(stderr,"Problema al abrir dispositivo PCM",
                                                        snd_strerror(pcm));
```

```
                                        exit(1);
                                }

/*se llama a la estructura que almacena las configuraciones de parametros de
  hardware mediante _alloca
*/

        snd_pcm_hw_params_alloca(&parametros);

/*Inicializa parámetros con la función _any que es la que recibe el flujo pcm
  anteriormente abierto
*/

        snd_pcm_hw_params_any(controlador, parametros);
/*
  Establecer los parámetros de Hardware deseados
 */

        if ((pcm = snd_pcm_hw_params_set_access(controlador, parametros,
                                                SND_PCM_ACCESS_RW_INTERLEAVED))
                                < 0)
        printf("ERROR: Can't set interleaved mode. %s\n", snd_strerror(pcm));

        if ((pcm = snd_pcm_hw_params_set_format(controlador,parametros,
                                                        SND_PCM_FORMAT_S16_LE)) < 0)
        printf("ERROR: Can't set format. %s\n", snd_strerror(pcm));

        if ((pcm = snd_pcm_hw_params_set_channels(controlador,parametros,1)) < 0)
        printf("ERROR: Can't set channels number. %s\n", snd_strerror(pcm));

        rate = 44100;
        if((pcm=snd_pcm_hw_params_set_rate_near(controlador,parametros,&rate, &dir))< 0)
        printf("ERROR: Can't set rate. %s\n", snd_strerror(pcm));

        frames = 1024;
        snd_pcm_hw_params_set_period_size_near(controlador,parametros, &frames, &dir);

/*Escribir los parámetros en el driver
*/

pcm = snd_pcm_hw_params(controlador, parametros);
        if (pcm < 0)
        {
        fprintf(stderr, "unable to set hw parameters: %s\n",snd_strerror(pcm));
        exit(1);
        }

snd_pcm_hw_params_get_channels(parametros, &channels);
        printf("channels: %i ", channels);

        if (channels == 1)
                printf("(mono)\n");
        else if (channels == 2)
                printf("(stereo)\n");
```

```
snd_pcm_hw_params_get_rate(parametros, &tmp, &dir);
        printf("rate: %d bps\n", tmp);

        printf("seconds: %d\n", seconds);

snd_pcm_hw_params_get_format(parametros, &tmp);
        printf("Formato   : '%s' (%s)\n",snd_pcm_format_name((
        snd_pcm_format_t)tmp),snd_pcm_format_description((snd_pcm_format_t)tmp));


snd_pcm_hw_params_get_period_size(parametros, &frames, &dir);
        printf("Period_size: %i\n", frames);

        buff_size = frames * 2 * channels;
        buffer = (char *) malloc(buff_size);

snd_pcm_hw_params_get_period_time(parametros, &tmp, &dir);
        printf("Period_time: %i\n", tmp);

        for(loops = ((10 * 1000000)/(tmp)) ; loops > 0 ; loops--)
                {
                        pcm = fread(buffer,1,buff_size,fp);
                        //printf("%d\n",i++);
                        if(pcm == 0)
                        {
                                fprintf(stderr,"end of file on input\n");
                                break;
                        }
                        else if(pcm != buff_size)
                        {
                        fprintf(stderr,  "short write: wrote %d bytes/n", pcm);
                        }

                        pcm = snd_pcm_writei(controlador, buffer, frames);

                        if(pcm == -EPIPE)
                        {
                                fprintf(stderr,"OVERRUN\n");
                                snd_pcm_prepare(controlador);
                        }

                        else if(pcm < 0)
                        {
                        fprintf(stderr,"ERROR DE LECTURA\n",snd_strerror(pcm));
                        }
                        else if( pcm != (int)frames)
                        {
                        fprintf(stderr, "Lectura Corta :\n %d bytes leidos\n",pcm);
                        }

                        else
                        {
                                printf("Buffer success\n");
                                printf("leidos %d samples\n",i++);
                        }
```

```
                    }
                snd_pcm_drain(controlador);
                snd_pcm_close(controlador);
                free(buffer);
                fclose(fp);

        printf("\n");
        return 0;
}
```

Interfaces de alsa en el sistema :

1. Information Interface (/proc/asound)
2. Control Interface (/dev/snd/controlCX)
3. Mixer Interface (/dev/snd/mixerCXDX)
4. PCM Interface (/dev/snd/pcmCXDX)
5. Raw MIDI Interface (/dev/snd/midiCXDX)
6. Sequencer Interface (/dev/snd/seq)
7. Timer Interface (/dev/snd/timer)

Variables de entorno

La configuración de estas variables, afecta el funcionamiento de alsa-lib

LIBASOUND_DEBUG

=1 print errors to stderr, dump hw_params
=2 also assert(0)

ALSA_CONFIG_PATH

file to use instead of ALSA_CONFIG_DIR/alsa.conf"

ALSA_MIXER_SIMPLE

file to use instead of ALSA_CONFIG_DIR/smixer.conf

ALSA_MIXER_SIMPLE_MODULES

path to modules .so files

LADSPA_PATH

path to ladspa plugins
Proposed

LIBASOUND_PCM_DUMP

=1 dump pcm_hw_params to console from any app (similar to aplay -v)