Advanced Linux Sound Architecture (ALSA)

The future for the Linux sound!?

Keywords: Linux, sound, soundcards, driver, library, ALSA

Author: Jaroslav Kysela
E−mail: perex@suse.cz

Introduction

The Advanced Linux Sound Architecture (ALSA) [1] project was founded by Jaroslav Kysela at the beginning of 1998 on a non−commercial basis. This project is intended to improve overall Linux sound support. The ALSA proffesional team has been funded in 2nd December 1999. Initially, the team had two members. Since 25th April 2000, the team has had three members: Jaroslav Kysela (Czech Republic), Abramo Bagnara (Italy) and Takashi Iwai (Germany). All three members are employed by SuSE[5]. The project has actually more than ten contributors worldwide.

Main goals of the ALSA project are:

- Create a fully GPL sound driver system for Linux.
- Create a fully modularized sound driver.
- Maintain backwards compatibility with most OSS/Free[2] applications.
- Create the LGPL ALSA Library (C, C++), which simplifies ALSA application development.
- Create the ALSA Manager − alsaconf, an interactive configuration program for the driver.

Soundcard and devices

The ALSA driver uses a quite different look to devices than the OSS/Free[2] driver. The primary identification element is a soundcard. Devices for one interface − like PCM (digital audio) − are not mixed together. Each soundcard has its own set of devices. The ALSA driver identifies devices via two numbers: a soundcard number and a device number. This representation is more real and an user understands the device mapping in an easier way.

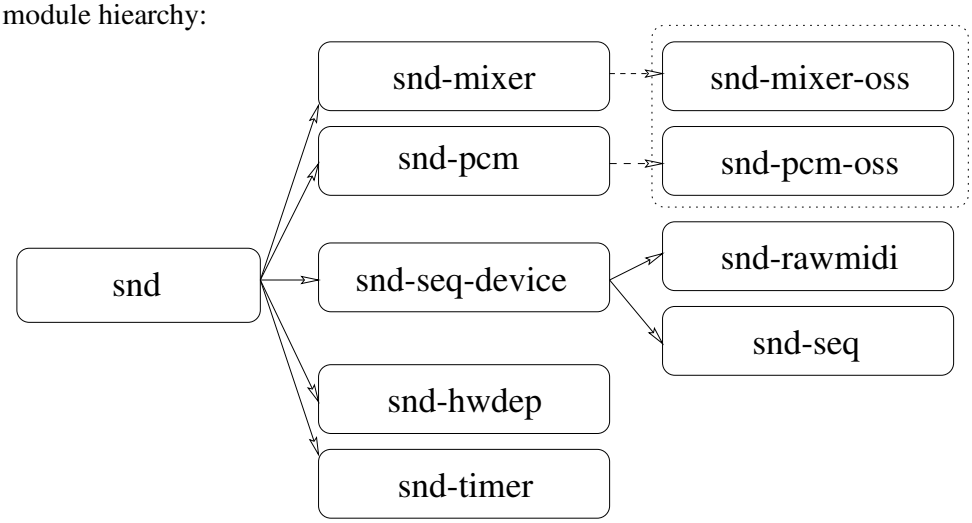
This addressing allows to use the kmod deamon. The soundcard switching is also very easy. A soundcard can be also addressable through a string identifier.

Modularized design

The ALSA driver is very modularized. We are trying to separate every things to independent modules. This separation has big advantages:

- The code can be reused. The most of current soundcards have same components.
- The midlevel code separation from the lowlevel drivers causes that the lowlevel drivers are smaller and more readable. It allows a better maintenance.

Basic module hiearchy:



- **snd**
  This module is the ALSA driver kernel. It contains a device multiplexor, soundcard management, /proc information routines and a basic control interface for detection of present interfaces and features.
- **snd−pcm**
  This module contains basic support for PCM (digital audio) layers.
- **snd−timer**
  This module is used, because the snd−pcm module provides timers.
- **snd−mixer**
  This is main mixer module which provides native API.
- **snd−rawmidi**
  This module provides the raw MIDI native and OSS/Free APIs.
- **snd−seq−device**
  This module manages the sequencer drivers. This module is used, because some people do not require to use the sequencer modules automatically. The whole sequencer layer is being very large and loading of all sequencer code implicitly may be wasting of space in the operating memory.
- **snd−seq**
  This module contains the sequencer kernel.
- **snd−pcm−oss**
  This module provides the additional OSS/Free PCM compatibility.
- **snd−mixer−oss**
  This module provides the additional OSS/Free mixer compatibility.

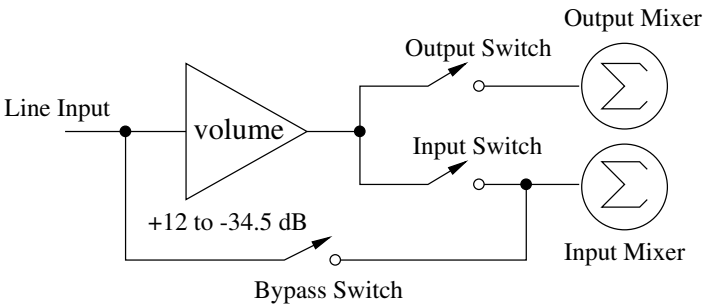
Control interface

The control interface allows to an application to obtain various information from the driver without the exclusive locking of some feature. It also offers reading of the change / modify / add / remove events for the universal switches. The universal switches allow to read and modify various parameters inside the driver. Each switch is accessible through its name. The universal switches provide type value and range. For example: integer type with range from 0 to 16.

Mixer interface

The mixer (version 2) interface has very new design now. It does not follow anything known. The mixer is described via control elements and routes among them. It allows to describe very complicated audio analog/digital mixers available in current soundcards. We have also possibility to enhance this interface in each aspects. The mixer interface has also a group control for the basic elements. This control method was designed to provide the standard mixer behaviour and to retain the compatibility with the OSS/Free mixer interface. Both control methods (element & group) can be used concurrently. The mixer interface also fully supports the change / modify / add / remove notifications through events which can be obtained using the read() function. It allows a perfect synchronization among more mixer applications.

An example of a possible mixer part:



This part contains seven mixer elements:

1. Line Analog Input
2. Volume gain for Line Input
3. Output switch for Line Input
4. Input switch for Line Input
5. Input bypass switch for Line Input
6. Output accumulator
7. Input accumulator

All elements show also routes among them inside the ALSA API. It is very useful for the intuitive graphics visualization.

The mixer group 'Line' has only elements, which can be driven:

1. Volume gain for Line input
2. Output switch for Line input
3. Input switch for Line input
4. Input bypass switch

A lowlevel driver controls the single elements inside group to handle requests: set volume level, set mute or capture source. To accomplish the situation, one element can be in more groups (input signal selector – MUX – is an example).

The mixer API is able to pass the changes (value or structure based) to an application to preserve consistency among more mixer applications and to notify the mixer structure changes.

PCM (digital audio) interface

The PCM interface supports full duplex when hardware is capable. The last added feature is multi open. If hardware is able to mix more PCM streams concurrently, the ALSA driver allows the open call more times, until the resources are not exhausted.

The PCM interface operates with two modes – block and stream. The block mode uses the standard enhanced dual buffer scheme. A whole ring data buffer is separated to equal fragments and the driver takes care about the management of fragments. The stream mode uses a whole ring data buffer without any boundaries. The actual position in the ring buffer is used to determine how many bytes are available for an operation. It allows to use the sample resolution, but on the other side, an user space code must poll the driver in a time period. This method may be used also for software modems and other software which requires accurate real–time synchronization.

The PCM layer also fully supports intelligent mmap control. The mmap control provides an efficient method to access samples directly from a hardware without any additional copying.

The PCM API has build–in support for AES/EBU/IEC–958 transfer protocols. This makes a possibility to share same user space code, which uses these industrial standards, among many sound applications with minimum effort.

Raw MIDI interface

The Raw MIDI interface is fully supported. It allows to read and write unchanged data from the MIDI ports. These ports can connect either external or internal devices.

Timer interface

The timer interface allows to use timers from soundcards. It also offers a slave mode. It means that some piece of code from the user space can be synchronized with any kernel device which uses same timer. The PCM stream can be also used as a primary timer source. It allows a perfect synchronization between a digital audio stream and a MIDI stream.

Sequencer

The sequencer inside the ALSA driver is probably the most interesting part of the ALSA driver, but especially clients for internal (mostly wavetable) synthesis are still under development. The original proposal from Frank van de Pol have been very enhanced and the current code is able to drive 192 clients at same time. Each client has two memory pools for input and output events. The features like event merging, a client notification about the internal changes or changes provoked with an another application are a matter of course.

Hardware dependent interface

This interface does not provide an abstract layer to applications except the protocol identification. It is intended for hardware which does not fit to any abstract interface like FX processors, a special access to synthesizer chips etc.

Related code

The ALSA driver also contains the full support for ISA Plug & Play cards. The new isapnp module is not designed only for soundcards, but it can be used with an another driver for an ISA Plug & Play card. This slightly modied code (to follow Linus requests) is present in the 2.3.14 kernel.

The C ALSA library

The ALSA C library fully covers the ALSA driver API. The ALSA library extents the driver functionality and hides kernel calls. There is a large PCM plugin layer which allows an equal access to different hardware. This layer provides functionality as stream conversion on the fly and so on.

OSS[3] compatible interfaces (emulation layer) in the driver

- **Mixer**
  The OSS/Free mixer API is fully emulated.
- **PCM (digital audio)**
  The OSS/Free PCM API is emulated. A few applications have trouble with this emulation. These trouble are mostly caused with wrong assumptions of authors and volatile PCM API specification.
- **Raw MIDI**
  The OSS/Free raw MIDI API is emulated.
- **Sequencer**
  Both levels of the OSS sequencer API are emulated.

Supported hardware

The current ALSA driver supports wide range of soundcards. There are drivers for consumer hardware like cards from the Creative Labs production and the list is closed with the professional hardware like MidiMan Delta and RME9652 (Hammerfall) cards.

The complete list of all supported cards is out of scope of this presentation and is available on the ALSA web server[1].

The future

The interfaces are mostly finished. We would like to create clear API and retain the binary compatibility for the future. Our work continues with adding new lowlevel drivers for more soundcards and API improvements. We also need to finish the documentation for application and lowlevel driver programmers.

The ALSA team hopes that the ALSA driver will be merged into the main kernel tree after all application interfaces will be stable. Alan Cox partly confirmed this fact. The ALSA should be in the development kernel tree 2.5.x. We expect that OSS/Free and ALSA drivers will co−exist together for a while. Once ALSA becomes fully featured, OSS/Free may be removed from the Linux kernel.

About the author

Jaroslav Kysela is the founder and current leader of the ALSA project. He has been working with Linux since 1993. Between 1994 and 1997 he worked on the Linux Ultra Sound Project[4]. In 1995 he initiated and still maintains the development of the driver for 100Mbit/s Voice Any Lan network adapters from Hewlett Packard (the hp100.c driver). Since 1998 he has been working on the ALSA project. As part of the ALSA project, he designed ISA Plug & Play routines actually used in Linux kernel 2.3.14 and higher. He was employed by SuSE GmbH[5] in April 1999.

WWW links

[1] Advanced Linux Sound Architecture – http://www.alsa−project.org
[2] Open Sound System (OSS) Free – http://www.linux.org.uk/OSS
[3] Open Sound System (OSS) Programmer's Guide – http://www.opensound.com/pguide
[4] The Linux Ultra Sound Project – http://www.perex.cz/~perex/ultra
[5] SuSE – http://www.suse.com