Lab – ALSA usage and programming

Objective: get familiar with sound manipulation with ALSA

After this lab, you will be able to

- Play sound on a Linux system with ALSA
- Define ALSA PCM devices
- Use and manipulate PCM plugins
- Record sound
- Make other people in the room crazy!

Setup

Go to `/home/<user>/felabs/sound/lab1`

Edit the `/etc/exports` file to add the `/home/<user>/felabs/sound/lab1/nfsroot` directory. See the training labs on http://free-electrons.com/training/drivers for details on how to boot Linux on an NFS exported directory.

Run a virtual PC

We are going to practice with ALSA sound on a virtual PC emulated by QEMU.

Start the virtual machine by running the script found in the current directory:

```
./run_qemu
```

Create ALSA device files

Find the name of the sound card emulated by QEMU.

The driver for this sound card is already loaded in your virtual system.

Your virtual PC already contains all the libraries and software needed to manipulate sound in this lab. However, the filesystem doesn't have device files in `/dev/snd` yet, to let ALSA compatible applications access the hardware through the alsa-lib library.

Using information from `/proc`, create the ALSA device files corresponding to the sound card in the emulated PC.

See your lecture slides for details.

Play sound samples

Go to the `/root/samples` directory and try to play some of the sound samples. You can use the `aplay` command, which is a simple ALSA compatible player, which mainly supports the playback of WAV files. The nice thing about `aplay` is that it doesn't require to compile a codec library to play WAV files.

Naming your sound card

Give a name to your soundcard, and play samples using this name (`aplay -D` *name*).

Sound mixing

Try to play multiple files in parallel and this way, test ALSA's software mixing capabilities.

Now, try to play some files in parallel. Here's how you can do it:

```
for f in pig1.wav duck.wav cat.wav
> do
> aplay $f &
> done
```

You can also add a `sleep 1` command before `aplay` to add a 1s pause before playing a new sound. This should make the result sound a little bit better.

Using other PCM plugins

Discard the sound output from your sound player.

Play a sound into a file, without playing it on the sound card.

Play a sound into a file and play it on the sound card at the same time.

This is a convenient way of recording an audio stream, for example.

Tee PCM device

Create a PCM device that plays sound on the default sound card, and dumps the audio into a file. This file should always have the same name.

As opposed to using the tee plugin in the command line, this allows to dump played audio into a file without having to specify the name of this file in the command line.

Test that this device works fine.

Once this works, you will notice that the `tee` and `file` plugins always overwrite their target file. There is no straightforward way of keeping everything that was played from the beginning.

ALSA PCM programming

We will develop a small program to play a WAV file.

First, install the toolchain available at http://free-electrons.com/labs/tools/i686-unknown-linux-uclibc-gcc-4.3.2.tar.bz2.

Using the documentation of ALSA PCM API in the slides and at http://www.alsa-project.org/alsa-doc/alsa-lib/pcm.html, create a program that:

1. Opens the default PCM device with `snd_pcm_open()`
2. Prepare a `snd_pcm_hw_params` structure
   1. interleaved access mode
      `snd_pcm_hw_params_set_access()`
   2. 16 bits little endian format
      `snd_pcm_hw_params_set_format()`
   3. 2 channels
      `snd_pcm_hw_params_set_channels()`
   4. 44.100 Hz rate
      `snd_pcm_hw_params_set_rate_near()`
   5. Set a small period size, such as 32 frames
      `snd_pcm_hw_params_set_period_size_near()`
3. Set the parameters of this `snd_pcm_hw_params` structure
4. Get the period size and allocate a buffer large enough to hold that number of frames (the frame size is 4 bytes since we do 16 bits stereo audio)

5. Get the period time
6. Compute the number of loops to do to play 5 seconds of audio: `loops = 5000000 / periodtime`
7. Loop for each period
    1. Read a period (32 frames of 4 bytes) from the standard input (file descriptor 0) using the `read()` function
    2. Write to the PCM device using `snd_pcm_writei()`. If the return value is `-EPIPE`, call `snd_pcm_prepare()` to reset the device
8. After the loop, drain the device with `snd_pcm_drain()` and close it with `snd_pcm_close()`

To compile your application, do:

```
i686-unknown-linux-uclibc-gcc \
--sysroot /home/<user>/felabs/sound/lab1/build/ \
-o play play.c -lasound
```

To run you application, in Qemu, do:

```
cat cat.wav | ./play
```

Lab - Basic ALSA sound driver

Objective: Practice with the basic ALSA driver API

After this lab, you will

- be able to compile your own kernel with ALSA support
- practice a little bit with sound card registration.

Setup

Go to the `/home/<user>/felabs/sound/lab2` directory.

Download the Linux 2.6.23 sources and extract them in the current directory. Apply the Linux kernel patch in the `data/` subdirectory.

Don't hesitate to ask your instructor if you need any help with Kernel configuration and compiling.

Configure your kernel with the initial configuration file available in the `data/` subdirectory.

Open the kernel configuration interface:

```
> make xconfig
```

Enable ALSA support and just enable the below options (all of them statically compiled into the kernel, not as modules):

- `OSS mixer API`
- `OSS PCI (digital audio) API`
- `OSS PCI (digital audio) API - Include plugin system`
- `Verbose procfs contents`
- `Verbose printk`
- `Debug options`
- `Support for the AK4531 codec` (In the `PCI devices` submenu)

Compile your kernel.

Booting

The root filesystem was created with scratchbox, and just contains BusyBox and some sound playing tools (alsaplayer, alsa-utils), and libraries (alsa-base, libvorbis, libogg and ncurses).

You are going to boot your kernel on the filesystem in `felabs/sound/lab1/nfsroot`, which contains everything you need to play and record sound, while being reasonably small in size.

The hardware platform is a PC emulated by qemu. It emulates an Ensoniq 1370 PCI sound card. You are going to experiment with a driver for this card. What you play with the virtual card is eventually played on your PC's hardware, and what you record from the virtual card's microphone input comes from your PC's input line.

To make your experiments easier, the filesystem will be mounted through NFS, and driver compiling will be done in the root filesystem, boot visible on the target and on the development host. Add `/home/<user>felabs/sound/lab1/nfsroot` to your `/etc/exports` file, and restart your NFS server.

Have a look at the `run_qemu` command and run it. This should start a virtual PC booting Linux and reaching a text console.

Compiling a driver

Creating a ALSA driver from scratch obviously takes a lot of time. To keep the lab short enough, we will thus give you pieces of the final driver, and you will take care of filling some gaps in the puzzle. It would also take time to cover details about this specific hardware, so

low-level functions taking care of accessing hardware registers will also be given to you.

On your host, go to `nfsroot/src`. This directory contains the driver which you will complete, as well as a Makefile to compile it.

Run the `make` command to compile your driver.

In the virtual machine, go to the `/src/` directory. This is the same directory as `nfsroot/src` on your host. Load your driver and check that it is successfully loaded:

```
> insmod my1370.ko
> lsmod
```

Check whether there are any ALSA sound cards on your system.

PCI and module initialization

In the code of your driver, explicit the list of supported PCI devices, here a sound card with the `0x1274` vendor id and the `0x5000` product id.

Add the missing hooks to the module init and exit functions, to register your driver, with its supported devices and its `probe` and `remove` functions, which you will find in the code.

Reload your module, and check that you now see an ALSA sound card on your system.

Adding PCM devices to your card

Modify the `snd_audiopci_probe` function to add support for the mixer and the 2 PCM devices in your sound card, for which initialization code is already available.

Notice that `/proc/asound/cards` lists your new card.

Testing your card

Using the `/samples` directory, use the `aplay` or `alsaplayer` commands to make sure that playback works fine. You can even try to copy Ogg/Vorbis files to your root filesystem (easy with NFS), and play them with alsaplayer.

Using the `alsamixer` command, make sure the mixer works fine.

Using the `arecord` command, test that recording works fine.

Download a free Ogg/Vorbis file from the Internet, play it, and try to record at the same time, to check the full-duplex capabilities of your card.