Esound

Carsten Haitzler
Red Hat Software, Inc.

raster@redhat.com

Esound (also referred to as ESD) is a small sound daemon for both Linux and UNIX. ESD was created to provide a consistent and simple interface to the audio device, so applications do not need to have different driver support written per architecture. It was also designed to enhance capabilities of audio devices such as allowing more than one application to share an open device. ESD accomplishes these things while remaining transparent to the application, meaning that the application developer can simply provide ESD support and let it do the rest. On top of this, the API is designed to be very similar to the current audio device API, making it easy to port to ESD.

Overview

Esound (ESD) is a stand-alone sound daemon which abstracts the system sound device to multiple clients. Under Linux using the Open Sound System (OSS), as well as other UNIX systems, typically only one process may open the sound device. This is not acceptable in a desktop environment like GNOME, as it is expected that many applications will be making sounds (music decoders, event based sounds, video conferencing, etc). The ESD daemon connects to the sound device and accepts connections from multiple clients, mixing the incoming audio streams and sending the result to the sound device. Connections are only allowed to clients which can authenticate successfully, alleviating the concern that unauthorized users can eavesdrop via the sound device. In addition to accepting client connections from the local machine, ESD can be configured to accept client connections from remote hosts which authenticate successfully.

Applications wanting to contact the ESD daemon do so using the libesd library. Much like with file i/o, a ESD connection is first opened. The ESD daemon will be spawned automatically by libesd if a daemon is not already present. Data is then either read or written to the ESD daemon. For a ESD client local to the machine which the ESD daemon is running on, the data is transferred through a local socket, then written to the sound device by the ESD daemon. For a client on a remote machine, the data is sent by libesd on the remote machine over the network to the ESD daemon. The process is completely transparent to the application using ESD.

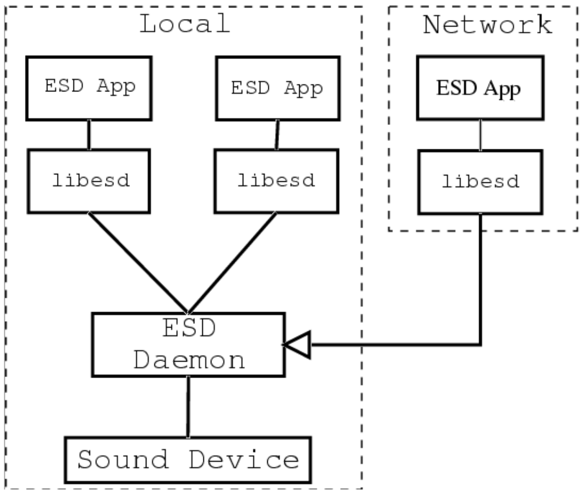

Figure 1. The ESD Process

Bit Stream

ESD will automatically sample an incoming stream from a client to the best format which is supported by the sound device. Therefore, an ESD client does not need to be concerned with the actual format it uses.

This alleviates the common problem of having to write code for each different platform which determines the possible formats available. A developer just selects a format to use and relies upon ESD to map that as best possible to the platform the application is running on.

ESD also supports recording and writing from the audio device. The API allows for different programs to be able to record and write simultaneously if your audio device is *full duplex* - that means the device is able to digitize analog audio input and convert digital to analog audio output at the same time. Many common sound cards are not full duplex, such as Sound Blaster cards. A device can play in 16 bits and record in 8 bits, but not play and record in 16 bits on both streams. Being able to record and play at the same bit resolution, same rate, and same number of channels is what is considered full duplex, for the purposes of this document.

In addition to streams, ESD also supports sample caching. The client can upload a sample of audio, tag it by a name, and receive an ID tag for that sample. At any point the client can ask for the sample to be freed from ESD's memory. The sample can be shared among several programs and allow instant playback of sounds, (For example, for spot effects), with no blocking of calls to the server to play long samples.

Futhermore, if the audio device of the standard Linux kernel supports mixing at the driver level, e.g, ALSA, ESD can act as a simple front-end to ALSA. This allows mixing on older kernels and non-Linux platforms, as well as mixing via the device when available.

Instant Advantages

ESD provides the application developer with some instant advantages:

- ESD has the ability to do network transparent audio if desired.
- ESD can keep ownership of the audio device to one user, such as *audio*, and then grant authentication keys to specific users for access. Removing users is as simple as changing the authentication key.
- Programs that are unable to handle "lesser" audio devices (ones that can only output 16 bit stereo 44.1kHz audio) can still run, as ESD will mix down automatically and transparently for the application.
- With ESD more than one application can access the sound device at once.
- Non ESD-enabled applications can be fooled into being ESD applications by using ESD's hack:

```
esddsp app_name -parameters to -the application
```

This will redirect the application to use ESD instead of **`/dev/dsp`**.

- You can monitor all mixed output to the audio device. **esdmon** is a quick example of this. This is useful for being able to do waveform displays for audio output from your computer.

Problems

ESD is by no means perfect, but it is a small, manageable project and thus can easily be expanded and modified to meet the needs of applications.

Several problems currently in ESD are:

- Lag could be reduced inside of ESD's own mixing routines.
- ESD needs better audio client management support (similar to the X equivalent of Window managers and ICCCM).
- ESD suffers from lack of *real-time* processing. It is liable to "crackle" and become unable to keep up in piping and mixing audio to the device if it does not get sufficient CPU time-slices for a period of time. This is a problem that is hard to overcome in an easy fashion without making ESD an **SUID** root process so that it could ursurp a higher priority.
- Authentication is simplistic, as ESD only accepts a single authentication key.

References

Websites:

- Esound (http://www.tux.org/~ricdude/EsounD.html)
- ALSA (http://www.alsa-project.org/api.html)
- OSS (http://www.4front-tech.com/pguide/)