DisOrder Uniform Audio

Problem

DisOrder has supports three audio backends from two programs, plus network play. There is considerable model mismatch and duplicated code.

Solution

Define a single audio interface and use that.

Requirements

- Support OSS, ALSA and Core Audio; and maybe others later
- Support RTP network play; and maybe other network sound protocols later.
- Select at runtime, so that Linux binaries are independent of the chosen sound API
- Per-API configuration, e.g.:
    - Device name
    - Source address and port, target address and port
    - Compression scheme

Interface

Global:

- List APIs

For each API:

- List devices
    - Maybe
- Open device
    - For each platform there's a default interface (preferably a physical one)
    - For physical interfaces there's always a default device
- Set device configuration
    - Non-generic
    - e.g. network parameters
- Start
    - Supply a callback to get sample data; see below
- Activate
    - Starts calling callback from a background thread
- Deactivate
    - Stops calling callback
    - If callback is blocking, so might this
- Stop
    - Implicitly deactivates
- Get/set volume

Start/stop are potentially expensive setup and teardown. Activate and deactivate are supposed to be (comparatively) cheap.

The API gets to call a user function to supply more audio; something like:

```
typedef int ua_get_audio_data_fn(int16_t *buffer,
```

```
                                int max_samples,
                                int latency,
                                void *userdata);
```

Here:

- `buffer` is the buffer to fill
- `max_samples` is the buffer size
- `latency` is the number of milliseconds until this data will play, or -1 if this is unknown/unknowable
- `userdata` is an opaque pointer

The return value is the number of samples actually supplied.

TODO what are you supposed to do if no data is *currently* available?

TODO is this allowed to be 0?

Implementations

OSS and ALSA

These are almost the same and probably share much of their code.

`Start` will create a background thread which, while activated, sits in a tight loop awaiting audio data and playing it. Activation, deactivation and stopping are managed via flags which are checked after each call plus a condition variable to allow asynchronous notification.

Core Audio

A straightforward translation of the Core Audio interface.

RTP Play

As for OSS/ALSA we have a background thread. However it will deliberately wait between callbacks if it's significantly ahead of itself. This implies maintaining the relationship between the current time and the current sample count.

We could do a FLAC compress step between getting audio data and transmitting it, looking ahead.

-- RichardKettlewell - 22 Feb 2009