

Live-coding Audio in C

Claude Heiland-Allen

2018-05-12

Contents

- ▶ Overview
- ▶ JACK Audio
- ▶ Dynamic Reloading
- ▶ Detecting File Changes
- ▶ Future Work

Overview

How Clive Works

- ▶ watches a directory for file changes
- ▶ recompiles changed sources
- ▶ reloads recompiled library
- ▶ maintains memory area between reloads

Clive Strengths

- ▶ two-phase edit/commit cycle
- ▶ no explicit “DSP graph of UGens” separation
- ▶ compilation is realtime safe
- ▶ reloading is (almost) realtime safe
- ▶ processing uses single-sample callbacks (simple)

Clive Weaknesses

- ▶ explicit state maintenance, append-only
- ▶ tiny UGen library
- ▶ compilation is high latency
- ▶ reloading is at JACK block boundaries
- ▶ processing uses single-sample callbacks (slow)

JACK Audio

JACK Client Setup

```
client = jack_client_open("live", JackNoStartServer, 0);
jack_set_process_callback(client, processcb, 0);
out_port = jack_port_register(client, "live:output_1",
    JACK_DEFAULT_AUDIO_TYPE, JackPortIsOutput, 0);
jack_activate(client);
jack_connect(client,
    jack_port_name(out_port), "system:playback_1");
```


JACK Process Callback

```
int processcb(jack_nframes_t nframes, void *arg) {
    jack_default_audio_sample_t *in, *out;
    in = jack_port_get_buffer(in_port, nframes);
    out = jack_port_get_buffer(out_port, nframes);
    for (jack_nframes_t i = 0; i < nframes; ++i)
        out[i] = state.func(state.data, in[i]);
    return 0;
}
```

Clive Concept

- ▶ change `state.func` at runtime
- ▶ preserve contents of `state.data`

Dynamic Code Reloading

Using libdl

```
void *old_dl = 0;
void *new_dl = 0;
if ((new_dl = dlopen("go.so", RTLD_NOW)) {
    callback *new_cb;
    new_cb = dlsym(new_dl, "go");
    if (new_cb) {
        while (inprocesscb) ; // race condition
        state.func = new_cb;
        state.reload = 1;
        if (old_dl) dlclose(old_dl);
        old_dl = new_dl;
    } else dlclose(new_dl);
}
```

Race Condition

- ▶ don't unload running code (otherwise... boom!)

```
volatile inprocesscb = 0;
int processcb(/* ... */) {
    inprocesscb = 1;
    // ...
    inprocesscb = 0;
    return 0;
}
```

Cache Circumvention

- ▶ `dlopen` caches based on filenames
- ▶ need to copy the `go.so` file to a new location
- ▶ double buffering works (two copies are enough)

Detecting File Changes

Getting INotify Events

```
int ino = inotify_init();
int wd = inotify_add_watch(ino, ".", IN_CLOSE_WRITE);
ssize_t buf_bytes =
    sizeof(struct inotify_event) + NAME_MAX + 1;
char *buf = malloc(buf_bytes);
while (1) {
    memset(buf, 0, buf_bytes);
    ssize_t r = read(ino, buf, buf_bytes); // blocking
    if (r == -1) sleep(1);
    else /* parse events */ ;
}
```


Parsing INotify Events

```
char *bufp = buf;
while (bufp < buf + r) {
    struct inotify_event *ev = bufp;
    bufp += sizeof(*ev) + ev->len;
    if (ev->mask & IN_CLOSE_WRITE) {
        if (0 == strcmp("go.so", ev->name))
            /* reload */ ;
        if (0 == strcmp("go.c", ev->name))
            /* recompile */ ;
    }
}
```

Future Work

Embiggen UGen Library

- ▶ oscillators (bandlimited wavetables, ...)
- ▶ filters (Butterworth, ...)
- ▶ spatialisation (ambisonics, ...)
- ▶ sequencing (varistep, ...)

Block-based Processing

- ▶ more CPU efficient
- ▶ more awkward
- ▶ FFT-based spectral processing

Embedded DSP

- ▶ low power audio processing devices
- ▶ process external inputs with very low latency
- ▶ compile on host and transfer over USB/network
- ▶ also support sensors/electronics
- ▶ rapid prototyping of embedded instruments

Thanks

Clive

- ▶ `claudio@mathr.co.uk`
- ▶ `https://mathr.co.uk/clive`
- ▶ `https://code.mathr.co.uk/clive`