Multimedialne API na przykładzie Linuksa

mgr. inż. Piotr Sokołowski

Podstawy dźwięku

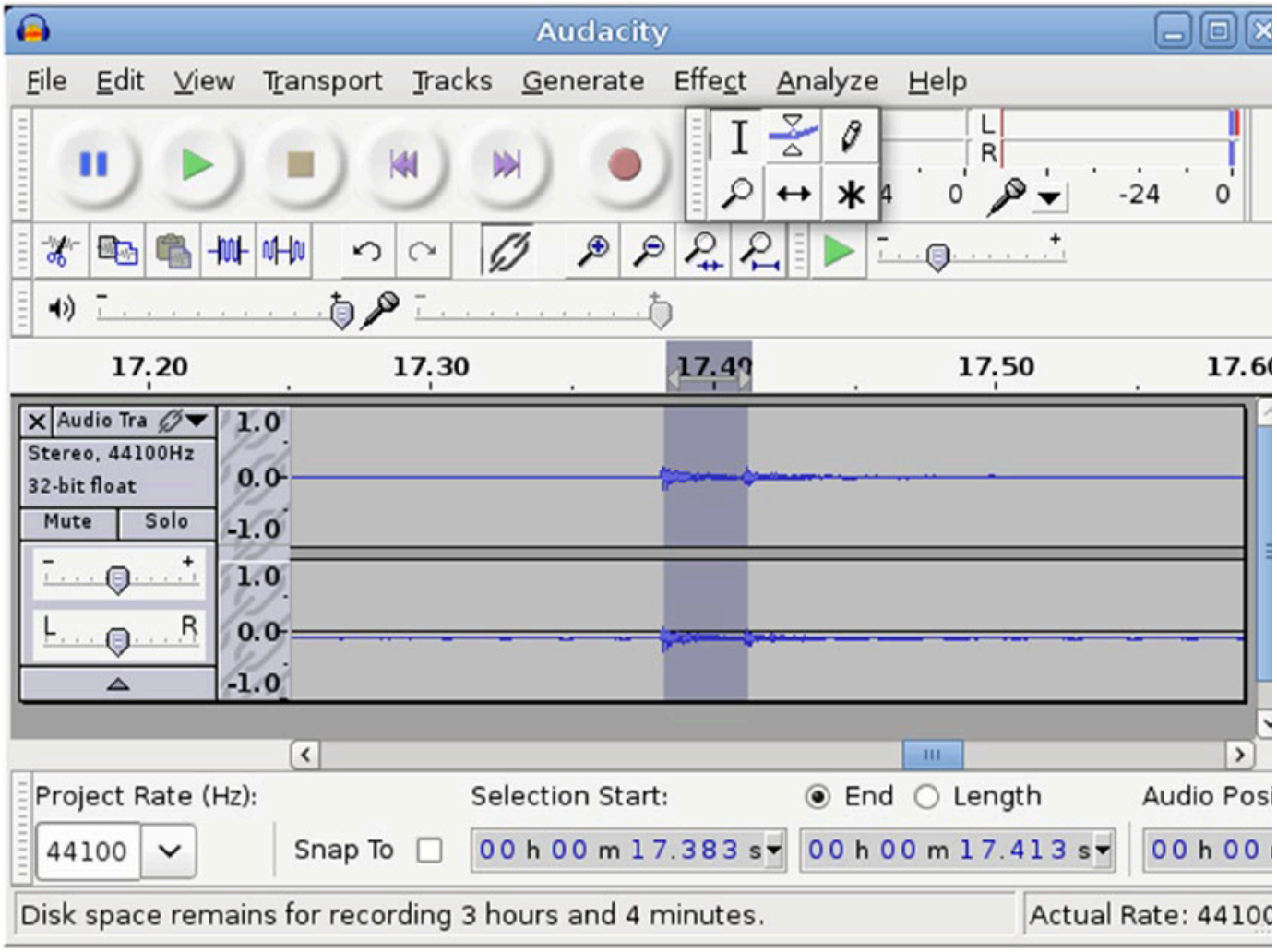
- Próbkowanie dźwięku
- Częstotliwość próbkowania
- Format próbkowania
- Ramki
- Pulse-Code Modulation (PCM)

Najeżdżanie i nienadążanie

“When a sound device is active, data is transferred continuously between the hardware and application buffers. In the case of data capture (recording), if the application does not read the data in the buffer rapidly enough, the circular buffer is overwritten with new data. The resulting data loss is known as overrun. During playback, if the application does not pass data into the buffer quickly enough, it becomes starved for data, resulting in an error called underrun.”

https://www.linuxjournal.com/article/6735?page=0,1

Opóźnienia i jitter

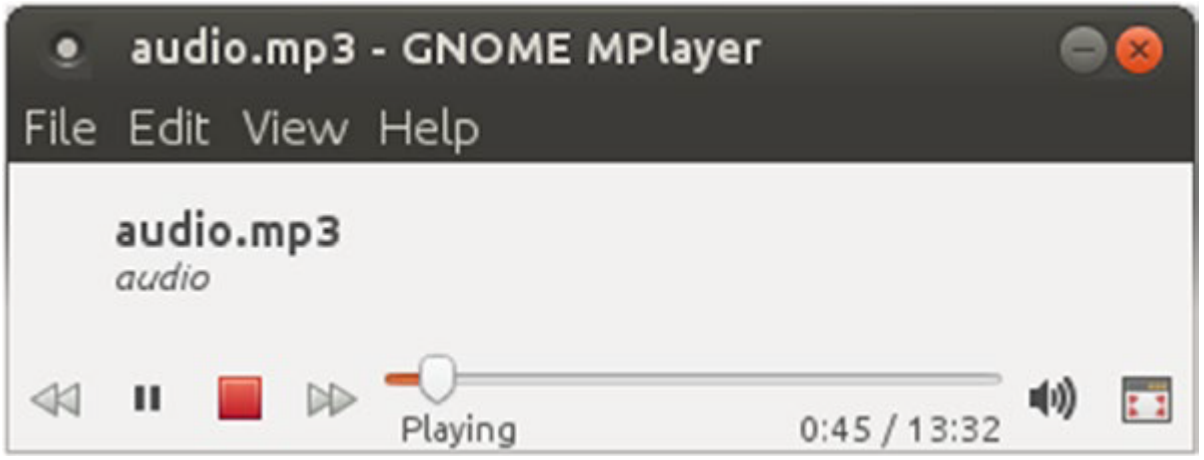


Miksowanie

- Trasowanie między wejściami a wyjściami
- Ustawienie wzmocnienia i poziomów wyjścia dla różnych sygnałów na wejściu i wyjściu
- Nakładanie efektów jak opóźnienie czy pogłos
- Mieszanie wielu wejściowych sygnałów do jednego wyjścia
- Rozdzielanie wejściowego sygnału na wiele wyjść

Źródła

- The Scientist and Engineer's Guide to Digital Signal Processing (www.dspguide.com/) by Steven W. Smith
- Music and Computers: A Theoretical and Historical Approach (http://music.columbia.edu/cmc/MusicAndComputers/) by Phil Burk, Larry Polansky, Douglas Repetto, Mary Roberts, Dan Rockmore



SOX

Konwersja do jednego kanału

```
sox audio.mp3 audio.ogg channels 1
```

Dwukrotne podniesienie poziomu głośności

```
sox audio.mp3 audio.ogg vol 2
```

Zmiana częstotliwości próbkowania

```
sox audio.mp3 audio.ogg rate 16k
```

FFmpeg/avconv

Nagranie z urządzenia hw:0

```
ffmpeg -f alsa -i hw:0 test.mp3
```

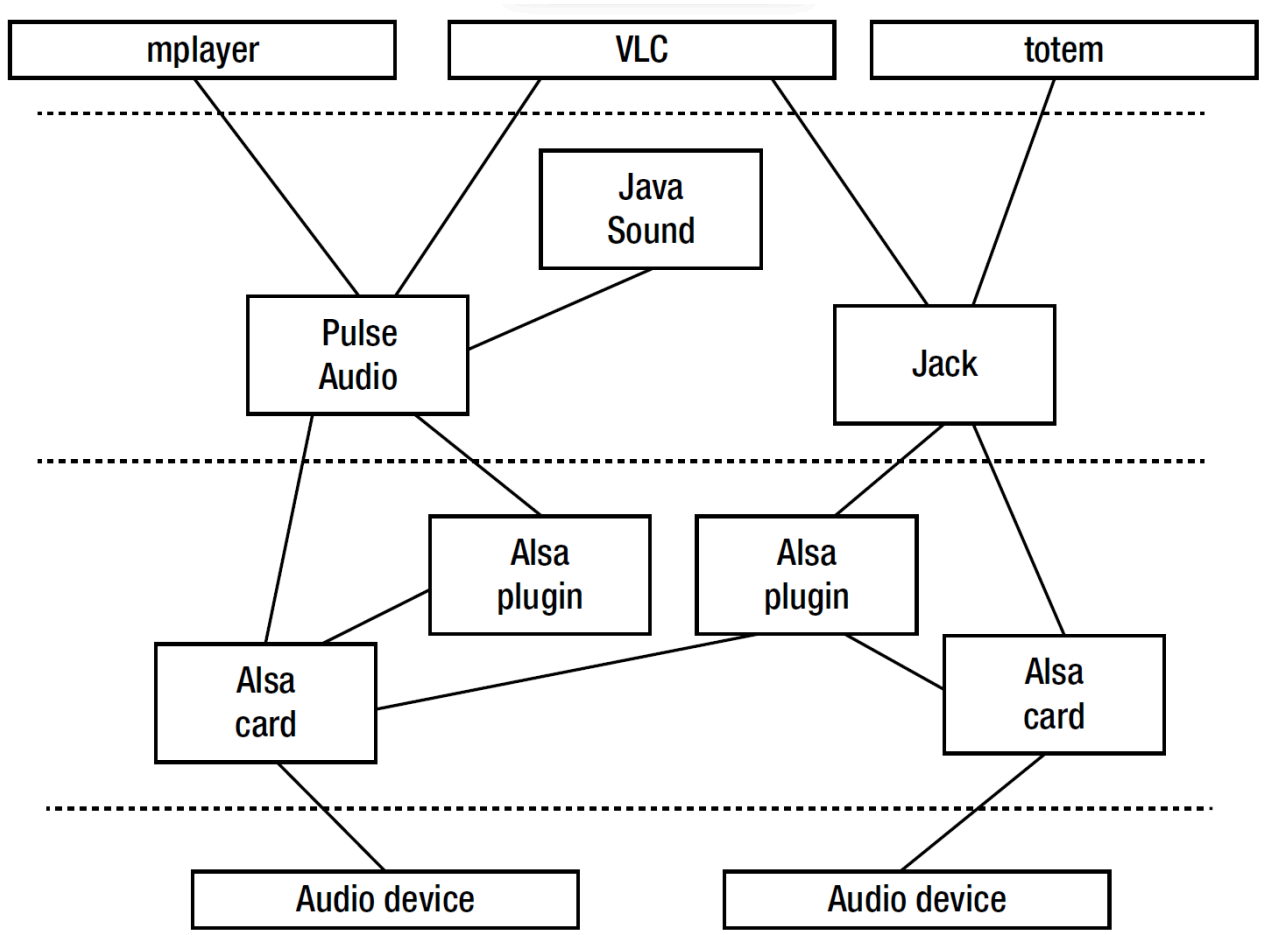
GStreamer

Odtworzenie audio

```
gst-launch-1.0 filesrc location="concept.mp3" ! decodebin ! alsasink
```

Odtworzenie MIDI

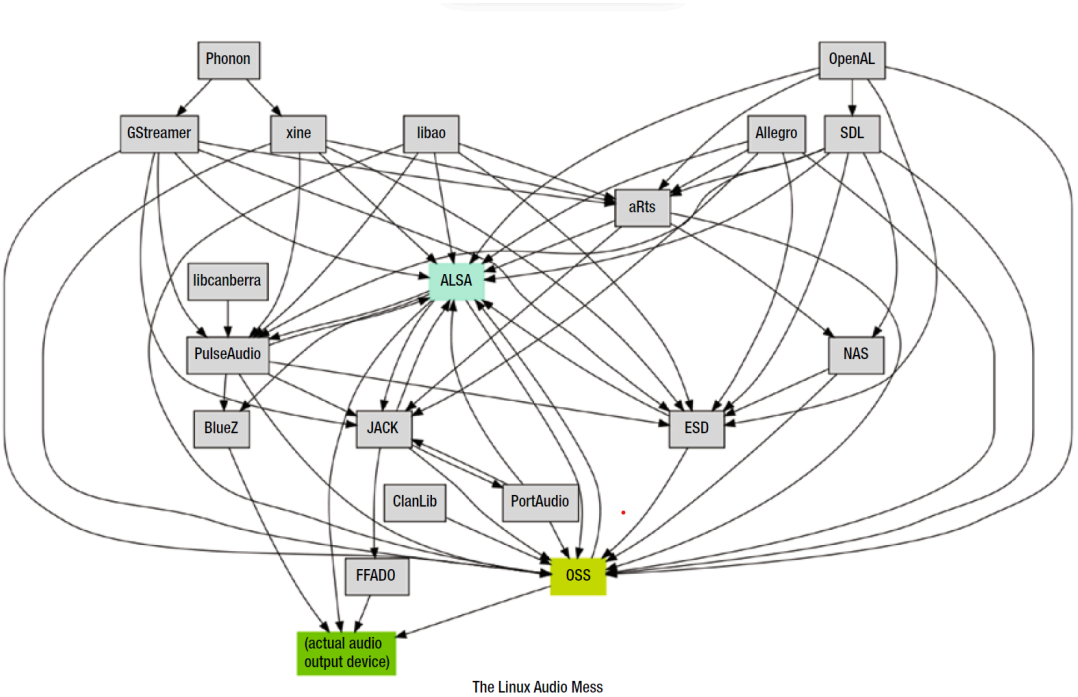
```
gst-launch-1.0 filesrc location="rehab.mid" ! wildmidi ! alsasink
```



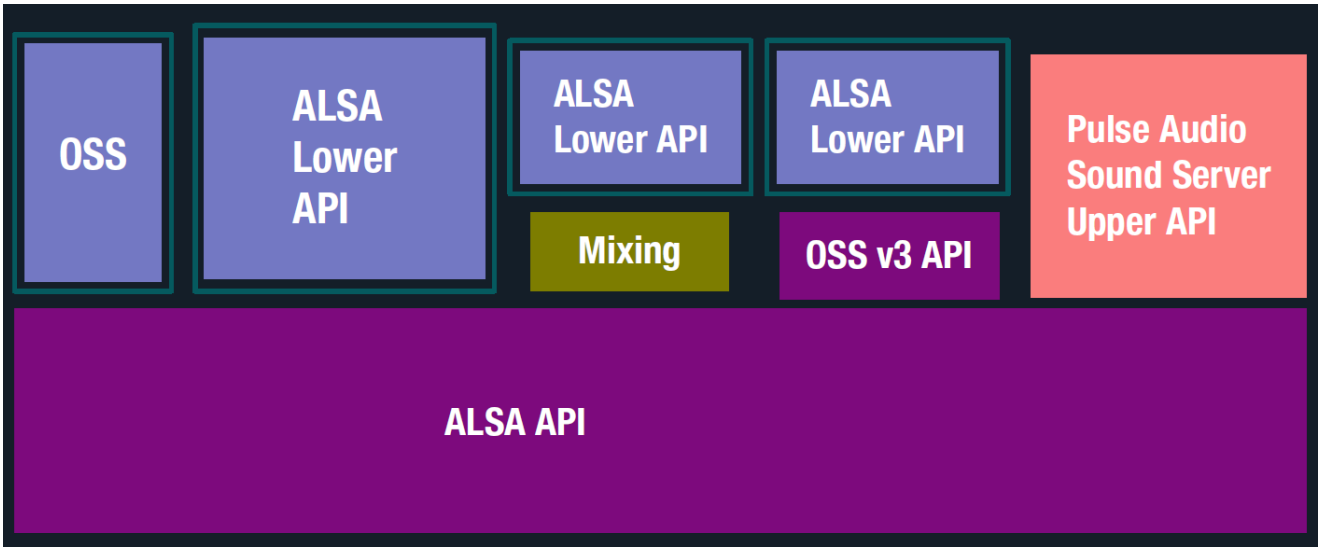
Serwery dźwięku

- Odtwarzacze multimedialne (jak VLC) - GStreamer lub ewentualnie Phonon (KDE)

- Dźwięki powiadomień w aplikacji - libcanberra lub KNotify (KDE). Pliki audio zgodne z XDG
- Profesjonalne aplikacje audio takie jak DAW - JACK lub ALSA
- Podstawowe odtwarzanie audio - ALSA
- Gry - Audio API z SDL lub w przypadku prostszych, okienkowych gier libcanberra
- Mikser - zależnie od warstwy, PulseAudio dla rozszerzeń środowiska graficznego lub w przypadku wykorzystania wsparcia sprzętowego ALSA
- Zastosowania wbudowane - ALSA (zależy od konkretnego przypadku)



The Linux Audio Mess
Origin: Mike Melanson, http://blogs.adobe.com/penguin.swf/
Updated October 10, 2008



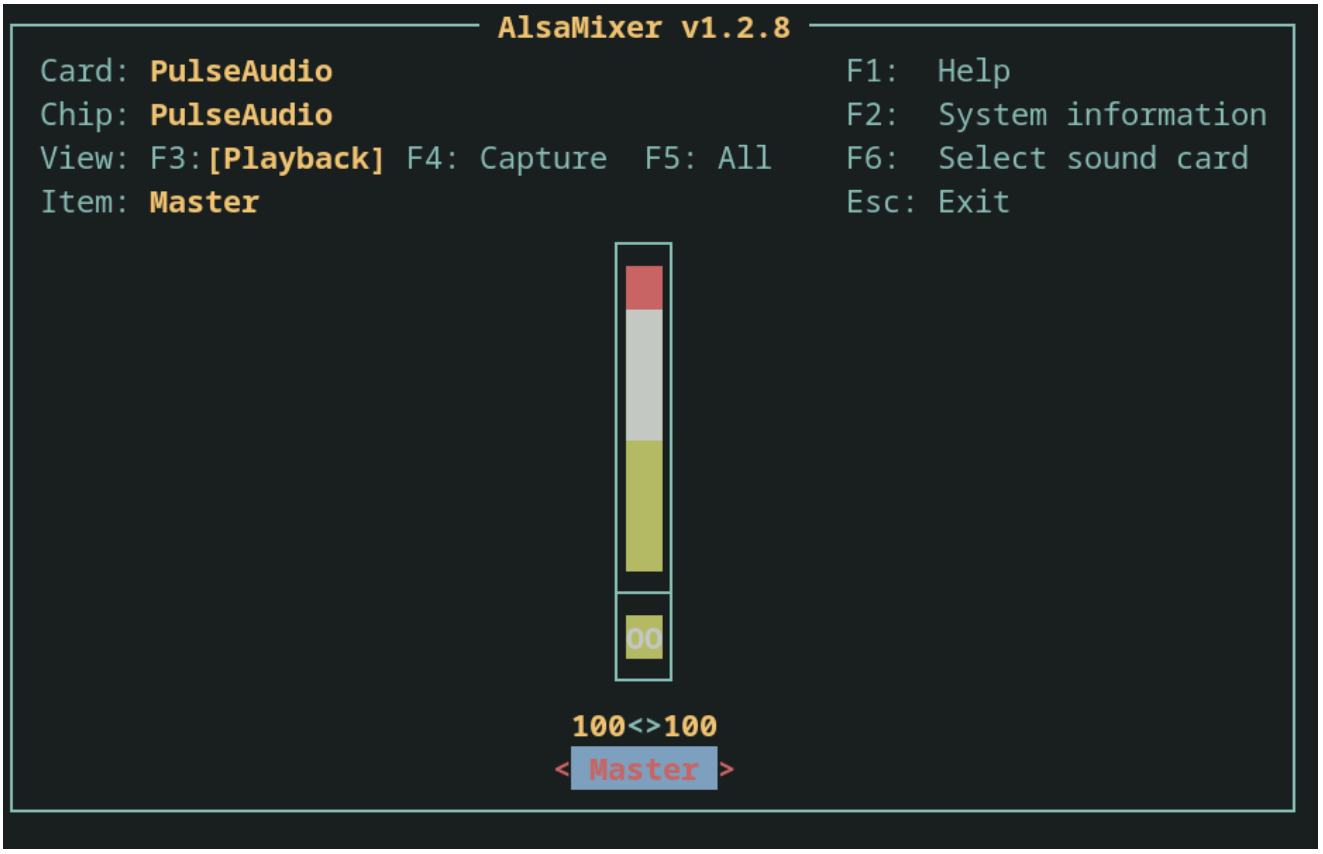
PulseAudio poniżej i powyżej ALSA?

- PulseAudio jest w stanie miksować dźwięki aplikacji, czego nie potrafi ALSA
- PulseAudio instaluje się jako domyślne urządzenie wyjściowe ALSA
- Aplikacje wysyłają dźwięk do domyślnego urządzenia ALSA, którym jest PulseAudio
- PulseAudio miksuje dźwięk i wysyła go z powrotem do wskazanego urządzenia w ASLA
- ALSA odtwarza dźwięk

Źródła

- A Guide Through The Linux Sound API Jungle, Lennart Poettering (http://0pointer.de/blog/projects/guide-to-sound-apis.html).
- “How it works: Linux audio explained” (http://tuxradar.com/).
- http://insanecoding.blogspot.com.au/2009/06/state-of-sound-inlinux-not-so-sorry.html

---



---

aplay/arecord

```
arecord -r 44100 -f S16_LE --buffer-size=128 | aplay --buffer-size=128
```

```
arecord -f dat -d 20 -D hw:0,0 test.wav
```

---

amixer

```
Simple mixer control 'Master',0
  Capabilities: pvolume pswitch pswitch-joined
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 65536
  Mono:
  Front Left: Playback 65536 [100%] [on]
  Front Right: Playback 65536 [100%] [on]
Simple mixer control 'Capture',0
  Capabilities: cvolume cswitch cswitch-joined
  Capture channels: Front Left - Front Right
  Limits: Capture 0 - 65536
  Front Left: Capture 10387 [16%] [on]
  Front Right: Capture 10387 [16%] [on]
```

---

sudo alsactl info

```
#
# Sound card
#
- card: 0
  id: PCH
  name: HDA Intel PCH
  longname: HDA Intel PCH at 0xd5228000 irq 132
  driver_name: HDA-Intel
  mixer_name: Realtek ALC3246
  components: HDA:10ec0256,10280785,00100002 HDA:8086280b,80860101,00100000
  controls_count: 41
  pcm:
    - stream: PLAYBACK
      devices:
        - device: 0
          id: ALC3246 Analog
          name: ALC3246 Analog
```

```
          subdevices:
            - subdevice: 0
              name: subdevice #0
        - device: 3
          id: HDMI 0
          name: HDMI 0
          subdevices:
            - subdevice: 0
              name: subdevice #0
        - device: 7
          id: HDMI 1
          name: HDMI 1
          subdevices:
            - subdevice: 0
              name: subdevice #0
        - device: 8
          id: HDMI 2
          name: HDMI 2
          subdevices:
            - subdevice: 0
              name: subdevice #0
    - stream: CAPTURE
      devices:
        - device: 0
          id: ALC3246 Analog
          name: ALC3246 Analog
          subdevices:
            - subdevice: 0
              name: subdevice #0
alsactl: rawmidi_device_list:105: snd_ctl_rawmidi_next_device
```

---

arecord -l

```
**** List of CAPTURE Hardware Devices ****
card 0: PCH [HDA Intel PCH], device 0: ALC3246 Analog [ALC3246 Analog]
 Subdevices: 1/1
 Subdevice #0: subdevice #0
```

---

aplay -l

```
**** List of PLAYBACK Hardware Devices ****
card 0: PCH [HDA Intel PCH], device 0: ALC3246 Analog [ALC3246 Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 0: PCH [HDA Intel PCH], device 3: HDMI 0 [HDMI 0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 0: PCH [HDA Intel PCH], device 7: HDMI 1 [HDMI 1]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 0: PCH [HDA Intel PCH], device 8: HDMI 2 [HDMI 2]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

/etc/asound.conf

```
pcm.hdmi0 {
    type hw
    card 1
    device 3 }

pcm.hdmi1 {
    type hw
    card 1
    device 7 }

pcm.hdmi2 {
    type hw
    card 1
    device 8 }
```

Narzędzia korzystające z ALSA

```
mplayer -ao alsa:device=hw=0.3 -srate 48000 bryan.mp3
```

```
vlc --aout alsa ...
```

```
timidity -Os ...
```

Programowanie ALSA

- Interfejs informacyjny (/proc/asound)
- Interfejs kontrolny (/dev/snd/controlCX)
- Interfejs miksera (/dev/snd/mixerCXDX)
- Interfejs PCM (/dev/snd/pcmCXDX)
- Surowy interfejs MIDI (/dev/snd/midiCXDX)
- Interfejs sekwencera (/dev/snd/seq)
- Interfejs zegara (/dev/snd/timer)

---

ls -l /dev/snd

```
razem 0
drwxr-xr-x  2 root root       60 03-26 09:51 by-path
crw-rw----+ 1 root audio 116,  9 03-26 09:51 controlC0
crw-rw----+ 1 root audio 116,  7 03-26 09:51 hwC0D0
crw-rw----+ 1 root audio 116,  8 03-26 09:51 hwC0D2
crw-rw----+ 1 root audio 116,  3 03-26 09:51 pcmC0D0c
crw-rw----+ 1 root audio 116,  2 03-26 09:57 pcmC0D0p
crw-rw----+ 1 root audio 116,  4 03-26 09:51 pcmC0D3p
crw-rw----+ 1 root audio 116,  5 03-26 09:51 pcmC0D7p
crw-rw----+ 1 root audio 116,  6 03-26 09:51 pcmC0D8p
crw-rw----+ 1 root audio 116,  1 03-26 09:51 seq
crw-rw----+ 1 root audio 116, 33 03-26 09:51 timer
```

Urządzenia sprzętowe

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>
#include <locale.h>
```

```c
// used by gettext for i18n, not needed here
#define _(STR) STR

static char *command;
#define error(...) do {\
    fprintf(stderr, "%s: %s:%d: ", command, __FUNCTION__, __LINE__); \
    fprintf(stderr, __VA_ARGS__); \
    putc('\n', stderr); \
} while (0)

static void device_list(snd_pcm_stream_t stream)
{
    snd_ctl_t *handle;
    int card, err, dev, idx;
    snd_ctl_card_info_t *info;
    snd_pcm_info_t *pcminfo;
    snd_ctl_card_info_alloca(&info);
    snd_pcm_info_alloca(&pcminfo);

    card = -1;
    if (snd_card_next(&card) < 0 || card < 0) {
        error(_("no soundcards found..."));
        return;
    }
    printf(_("**** List of %s Hardware Devices ****\n"),
         snd_pcm_stream_name(stream));
    while (card >= 0) {
        char name[32];
        sprintf(name, "hw:%d", card);
        if ((err = snd_ctl_open(&handle, name, 0)) < 0) {
            error("control open (%i): %s", card, snd_strerror(err));
            goto next_card;
        }
        if ((err = snd_ctl_card_info(handle, info)) < 0) {
            error("control hardware info (%i): %s", card,
snd_strerror(err));
            snd_ctl_close(handle);
            goto next_card;
        }
        dev = -1;
        while (1) {
            unsigned int count;
            if (snd_ctl_pcm_next_device(handle, &dev)<0)
                error("snd_ctl_pcm_next_device");
            if (dev < 0)
                break;
            snd_pcm_info_set_device(pcminfo, dev);
            snd_pcm_info_set_subdevice(pcminfo, 0);
            snd_pcm_info_set_stream(pcminfo, stream);
```

```
            if ((err = snd_ctl_pcm_info(handle, pcminfo)) < 0) {
                if (err != -ENOENT)
                    error("control digital audio info (%i): %s",
                     card,
                     snd_strerror(err));
                continue;
            }
            printf(_("card %i: [%s,%i] %s [%s], device %i: %s [%s]\n"),
             card, name, dev, snd_ctl_card_info_get_id(info),
             snd_ctl_card_info_get_name(info),
             dev,
             snd_pcm_info_get_id(pcminfo),
             snd_pcm_info_get_name(pcminfo));
            count = snd_pcm_info_get_subdevices_count(pcminfo);
            printf( _("  Subdevices: %i/%i\n"),
             snd_pcm_info_get_subdevices_avail(pcminfo),
             count);
            for (idx = 0; idx < (int)count; idx++) {
                snd_pcm_info_set_subdevice(pcminfo, idx);
                if ((err = snd_ctl_pcm_info(handle, pcminfo)) < 0) {
                    error("control digital audio playback info (%i): %s",
                     card,
                     snd_strerror(err));
                    } else {
                    printf(_("  Subdevice #%i: %s\n"),
                     idx,
                     snd_pcm_info_get_subdevice_name(pcminfo));
                    }
                }
            }
            snd_ctl_close(handle);
      next_card:
            if (snd_card_next(&card) < 0) {
                error("snd_card_next");
                break;
            }
      }
}


int main (int argc, char *argv[])
{
    device_list(SND_PCM_STREAM_CAPTURE);
    device_list(SND_PCM_STREAM_PLAYBACK);
}
```

```
**** List of CAPTURE Hardware Devices ****
card 0: [hw:0,0] PCH [HDA Intel PCH], device 0: ALC3246 Analog [ALC3246
Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
**** List of PLAYBACK Hardware Devices ****
card 0: [hw:0,0] PCH [HDA Intel PCH], device 0: ALC3246 Analog [ALC3246
Analog]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 0: [hw:0,3] PCH [HDA Intel PCH], device 3: HDMI 0 [HDMI 0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 0: [hw:0,7] PCH [HDA Intel PCH], device 7: HDMI 1 [HDMI 1]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 0: [hw:0,8] PCH [HDA Intel PCH], device 8: HDMI 2 [HDMI 2]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

Urządzenia PCM

```c
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>
#include <locale.h>
#define _(STR) STR

static void pcm_list(snd_pcm_stream_t stream )
{
    void **hints, **n;
    char *name, *descr, *descr1, *io;
    const char *filter;

    if (snd_device_name_hint(-1, "pcm", &hints) < 0)
        return;

    n = hints;
    filter = stream == SND_PCM_STREAM_CAPTURE ? "Input" : "Output";

    while (*n != NULL) {
        name = snd_device_name_get_hint(*n, "NAME");
        descr = snd_device_name_get_hint(*n, "DESC");
```

```
        io = snd_device_name_get_hint(*n, "IOID");

        if (io != NULL && strcmp(io, filter) == 0)
            goto __end;

        printf("%s\n", name);

        if ((descr1 = descr) != NULL) {
            printf(" ");

            while (*descr1) {
                if (*descr1 == '\n')
                    printf("\n ");
                else
                    putchar(*descr1);
                descr1++;
            }

            putchar('\n');
        }

        __end:
        if (name != NULL)
            free(name);
        if (descr != NULL)
            free(descr);
        if (io != NULL)
            free(io);
        n++;
    }

    snd_device_name_free_hint(hints);
}

main (int argc, char *argv[])
{
    printf("************ CAPTURE ************\n");
    pcm_list(SND_PCM_STREAM_CAPTURE);

    printf("\n\n************ PLAYBACK ************\n");
    pcm_list(SND_PCM_STREAM_PLAYBACK);
}
```

```
************ CAPTURE ************
null
    Discard all samples (playback) or generate zero samples (capture)
```

```
default
    Playback/recording through the PulseAudio sound server
lavrate
    Rate Converter Plugin Using Libav/FFmpeg Library
samplerate
    Rate Converter Plugin Using Samplerate Library
speexrate
    Rate Converter Plugin Using Speex Resampler
jack
    JACK Audio Connection Kit
oss
    Open Sound System
pulse
    PulseAudio Sound Server
speex
    Plugin using Speex DSP (resample, agc, denoise, echo, dereverb)
upmix
    Plugin for channel upmix (4,6,8)
vdownmix
    Plugin for channel downmix (stereo) with a simple spacialization
hw:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    Direct hardware device without any conversions
hw:CARD=PCH,DEV=3
    HDA Intel PCH, HDMI 0
    Direct hardware device without any conversions
hw:CARD=PCH,DEV=7
    HDA Intel PCH, HDMI 1
    Direct hardware device without any conversions
hw:CARD=PCH,DEV=8
    HDA Intel PCH, HDMI 2
    Direct hardware device without any conversions
plughw:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    Hardware device with all software conversions
plughw:CARD=PCH,DEV=3
    HDA Intel PCH, HDMI 0
    Hardware device with all software conversions
plughw:CARD=PCH,DEV=7
    HDA Intel PCH, HDMI 1
    Hardware device with all software conversions
plughw:CARD=PCH,DEV=8
    HDA Intel PCH, HDMI 2
```

```
    Hardware device with all software conversions
sysdefault:CARD=PCH
    HDA Intel PCH, ALC3246 Analog
    Default Audio Device
front:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    Front output / input
surround21:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    2.1 Surround output to Front and Subwoofer speakers
surround40:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    4.0 Surround output to Front and Rear speakers
surround41:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    4.1 Surround output to Front, Rear and Subwoofer speakers
surround50:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    5.0 Surround output to Front, Center and Rear speakers
surround51:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    5.1 Surround output to Front, Center, Rear and Subwoofer speakers
surround71:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    7.1 Surround output to Front, Center, Side, Rear and Woofer speakers
hdmi:CARD=PCH,DEV=0
    HDA Intel PCH, HDMI 0
    HDMI Audio Output
hdmi:CARD=PCH,DEV=1
    HDA Intel PCH, HDMI 1
    HDMI Audio Output
hdmi:CARD=PCH,DEV=2
    HDA Intel PCH, HDMI 2
    HDMI Audio Output
dmix:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    Direct sample mixing device
dmix:CARD=PCH,DEV=3
    HDA Intel PCH, HDMI 0
    Direct sample mixing device
dmix:CARD=PCH,DEV=7
    HDA Intel PCH, HDMI 1
    Direct sample mixing device
```

```
dmix:CARD=PCH,DEV=8
    HDA Intel PCH, HDMI 2
    Direct sample mixing device
usbstream:CARD=PCH
    HDA Intel PCH
    USB Stream Output


********** PLAYBACK **********
null
    Discard all samples (playback) or generate zero samples (capture)
default
    Playback/recording through the PulseAudio sound server
lavrate
    Rate Converter Plugin Using Libav/FFmpeg Library
samplerate
    Rate Converter Plugin Using Samplerate Library
speexrate
    Rate Converter Plugin Using Speex Resampler
jack
    JACK Audio Connection Kit
oss
    Open Sound System
pulse
    PulseAudio Sound Server
speex
    Plugin using Speex DSP (resample, agc, denoise, echo, dereverb)
upmix
    Plugin for channel upmix (4,6,8)
vdownmix
    Plugin for channel downmix (stereo) with a simple spacialization
hw:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    Direct hardware device without any conversions
plughw:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    Hardware device with all software conversions
sysdefault:CARD=PCH
    HDA Intel PCH, ALC3246 Analog
    Default Audio Device
front:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    Front output / input
```

```
dsnoop:CARD=PCH,DEV=0
    HDA Intel PCH, ALC3246 Analog
    Direct sample snooping device
usbstream:CARD=PCH
    HDA Intel PCH
    USB Stream Output
```

Parametry urządzeń PCM

- kanały
- częstotliwość próbkowania
- wielkość ramki
- czas okresu odświeżania (w mikrosekundach)
- rozmiar okresu
- liczba okresów na bufor
- czas buforu
- rozmiar buforu

Właściwości

```c
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

void info(char *dev_name, snd_pcm_stream_t stream) {
  snd_pcm_hw_params_t *hw_params;
  int err;
  snd_pcm_t *handle;
  unsigned int max;
  unsigned int min;
  unsigned int val;
  unsigned  int dir;
  snd_pcm_uframes_t frames;

  if ((err = snd_pcm_open (&handle, dev_name, stream, 0)) < 0) {
    fprintf (stderr, "cannot open audio device %s (%s)\n",
         dev_name,
         snd_strerror (err));
    return;
  }
```

```
  if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_any (handle, hw_params)) < 0) {
    fprintf (stderr, "cannot initialize hardware parameter structure
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_get_channels_max(hw_params, &max)) < 0) {
    fprintf (stderr, "cannot  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max channels %d\n", max);

  if ((err = snd_pcm_hw_params_get_channels_min(hw_params, &min)) < 0) {
    fprintf (stderr, "cannot get channel info  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min channels %d\n", min);

  /*
  if ((err = snd_pcm_hw_params_get_sbits(hw_params)) < 0) {
      fprintf (stderr, "cannot get bits info  (%s)\n",
           snd_strerror (err));
      exit (1);
  }
  printf("bits %d\n", err);
  */

  if ((err = snd_pcm_hw_params_get_rate_min(hw_params, &val, &dir)) < 0) {
    fprintf (stderr, "cannot get min rate (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min rate %d hz\n", val);

  if ((err = snd_pcm_hw_params_get_rate_max(hw_params, &val, &dir)) < 0) {
    fprintf (stderr, "cannot get max rate (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max rate %d hz\n", val);
```

```
  if ((err = snd_pcm_hw_params_get_period_time_min(hw_params, &val, &dir)) <
0) {
    fprintf (stderr, "cannot get min period time  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min period time %d usecs\n", val);

  if ((err = snd_pcm_hw_params_get_period_time_max(hw_params, &val, &dir)) <
0) {
    fprintf (stderr, "cannot  get max period time  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max period time %d usecs\n", val);

  if ((err = snd_pcm_hw_params_get_period_size_min(hw_params, &frames,
&dir)) < 0) {
    fprintf (stderr, "cannot  get min period size  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min period size in frames %lu\n", frames);

  if ((err = snd_pcm_hw_params_get_period_size_max(hw_params, &frames,
&dir)) < 0) {
    fprintf (stderr, "cannot  get max period size (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max period size in frames %lu\n", frames);

  if ((err = snd_pcm_hw_params_get_periods_min(hw_params, &val, &dir)) < 0)
{
    fprintf (stderr, "cannot  get min periods  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min periods per buffer %d\n", val);

  if ((err = snd_pcm_hw_params_get_periods_max(hw_params, &val, &dir)) < 0)
{
    fprintf (stderr, "cannot  get min periods (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max periods per buffer %d\n", val);

  if ((err = snd_pcm_hw_params_get_buffer_time_min(hw_params, &val, &dir)) <
```

```
0) {
    fprintf (stderr, "cannot get min buffer time (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min buffer time %d usecs\n", val);

  if ((err = snd_pcm_hw_params_get_buffer_time_max(hw_params, &val, &dir)) <
0) {
    fprintf (stderr, "cannot get max buffer time  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max buffer time %d usecs\n", val);

  if ((err = snd_pcm_hw_params_get_buffer_size_min(hw_params, &frames)) < 0)
{
    fprintf (stderr, "cannot get min buffer size (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("min buffer size in frames %lu\n", frames);

  if ((err = snd_pcm_hw_params_get_buffer_size_max(hw_params, &frames)) < 0)
{
    fprintf (stderr, "cannot get max buffer size  (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("max buffer size in frames %lu\n", frames);
}

int main (int argc, char *argv[])
{
  int i;
  int err;
  int buf[128];
  FILE *fin;
  size_t nread;
  unsigned int rate = 44100;

  if (argc != 2) {
    fprintf(stderr, "Usage: %s card\n", argv[0]);
    exit(1);
  }

  printf("********** CAPTURE **********\n");
  info(argv[1], SND_PCM_STREAM_CAPTURE);

  printf("********** PLAYBACK **********\n");
```

```
  info(argv[1], SND_PCM_STREAM_PLAYBACK);

  exit (0);
}
```

Przykład sprzętowy

```
********** CAPTURE **********
max channels 2
min channels 2
min rate 44100 hz
max rate 192000 hz
min period time 83 usecs
max period time 11888617 usecs
min period size in frames 16
max period size in frames 524288
min periods per buffer 2
max periods per buffer 32
min buffer time 166 usecs
max buffer time 23777234 usecs
min buffer size in frames 32
max buffer size in frames 1048576
********** PLAYBACK **********
cannot open audio device hw:0 (Device or resource busy)
```

Przykład programowy

```
********** CAPTURE **********
max channels 32
min channels 1
min rate 1 hz
max rate 384000 hz
min period time 2 usecs
max period time -1 usecs
min period size in frames 1
max period size in frames 1398102
min periods per buffer 3
max periods per buffer 1024
```

```
min buffer time 7 usecs
max buffer time -1 usecs
min buffer size in frames 3
max buffer size in frames 4194304
********** PLAYBACK **********
max channels 32
min channels 1
min rate 1 hz
max rate 384000 hz
min period time 2 usecs
max period time -1 usecs
min period size in frames 1
max period size in frames 1398102
min periods per buffer 3
max periods per buffer 1024
min buffer time 7 usecs
max buffer time -1 usecs
min buffer size in frames 3
max buffer size in frames 4194304
```

Nagrywanie

```c
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>
#include <signal.h>

#define BUFSIZE 128
#define RATE 44100

FILE *fout = NULL;

/*
 * quit on ctrl-c
 */
void sigint(int sig) {
  if (fout != NULL) {
    fclose(fout);
  }
  exit(1);
}

int main (int argc, char *argv[])
```

```
{
  int i;
  int err;
  short buf[BUFSIZE];
  snd_pcm_t *capture_handle;
  snd_pcm_hw_params_t *hw_params;
  unsigned int rate = RATE;
  int nread;

  if (argc != 3) {
    fprintf(stderr, "Usage: %s cardname file\n", argv[0]);
    exit(1);
  }

  if ((fout = fopen(argv[2], "w")) == NULL) {
    fprintf(stderr, "Can't open %s for writing\n", argv[2]);
    exit(1);
  }


  signal(SIGINT, sigint);

  if ((err = snd_pcm_open (&capture_handle, argv[1], SND_PCM_STREAM_CAPTURE,
0)) < 0) {
    fprintf (stderr, "cannot open audio device %s (%s)\n",
         argv[1],
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_any (capture_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot initialize hardware parameter structure
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_access (capture_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type (%s)\n",
         snd_strerror (err));
    exit (1);
  }
```

```
  if ((err = snd_pcm_hw_params_set_format (capture_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
    fprintf (stderr, "cannot set sample format (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_rate_near (capture_handle, hw_params,
&rate, 0)) < 0) {
    fprintf (stderr, "cannot set sample rate (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  fprintf(stderr, "rate set to %d\n", rate);

  if ((err = snd_pcm_hw_params_set_channels (capture_handle, hw_params, 2))
< 0) {
    fprintf (stderr, "cannot set channel count (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params (capture_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  snd_pcm_hw_params_free (hw_params);

  /*
  if ((err = snd_pcm_prepare (capture_handle)) < 0) {
    fprintf (stderr, "cannot prepare audio interface for use (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  */

  while (1) {
    if ((nread = snd_pcm_readi (capture_handle, buf, BUFSIZE)) < 0) {
      fprintf (stderr, "read from audio interface failed (%s)\n",
           snd_strerror (err));
      /* recover */
      snd_pcm_prepare(capture_handle);
    } else {
      fwrite(buf, sizeof(short), nread, fout);
    }
  }
```

```
  snd_pcm_close (capture_handle);
  exit(0);
}
```

```
./alsa_capture default tmp.s16
sox -c 2 -r 44100 tmp.s16 tmp.wav
mplayer tmp.wav
```

Odtwarzanie

```
#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

int main(int argc, char *argv[])
{
  int i;
  int err;
  int buf[128];
  snd_pcm_t *playback_handle;
  snd_pcm_hw_params_t *hw_params;
  FILE *fin;
  size_t nread;
  unsigned int rate = 44100;

  if (argc != 3) {
    fprintf(stderr, "Usage: %s card file\n", argv[0]);
    exit(1);
  }

  if ((err = snd_pcm_open (&playback_handle, argv[1],
SND_PCM_STREAM_PLAYBACK, 0)) < 0) {
    fprintf (stderr, "cannot open audio device %s (%s)\n",
         argv[1],
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
```

```
  }

  if ((err = snd_pcm_hw_params_any (playback_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot initialize hardware parameter structure
(%s)\n",
          snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_access (playback_handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type (%s)\n",
          snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_format (playback_handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
    fprintf (stderr, "cannot set sample format (%s)\n",
          snd_strerror (err));
    exit (1);
  }


  if ((err = snd_pcm_hw_params_set_rate_near (playback_handle, hw_params,
&rate, 0)) < 0) {
    fprintf (stderr, "cannot set sample rate (%s)\n",
          snd_strerror (err));
    exit (1);
  }
  printf("Rate set to %d\n", rate);

  if ((err = snd_pcm_hw_params_set_channels (playback_handle, hw_params, 2))
< 0) {
    fprintf (stderr, "cannot set channel count (%s)\n",
          snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params (playback_handle, hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters (%s)\n",
          snd_strerror (err));
    exit (1);
  }

  snd_pcm_hw_params_free (hw_params);

  /*
  if ((err = snd_pcm_prepare (playback_handle)) < 0) {
    fprintf (stderr, "cannot prepare audio interface for use (%s)\n",
```

```
          snd_strerror (err));
    exit (1);
  }
  */

  if ((fin = fopen(argv[2], "r")) == NULL) {
      fprintf(stderr, "Can't open %s for reading\n", argv[2]);
      exit(1);
  }

  while ((nread = fread(buf, sizeof(int), 128, fin)) > 0) {
    //printf("writing\n");
    if ((err = snd_pcm_writei (playback_handle, buf, nread)) != nread) {
      fprintf (stderr, "write to audio interface failed (%s)\n",
           snd_strerror (err));
      snd_pcm_prepare(playback_handle);
    }
  }

  snd_pcm_drain(playback_handle);
  snd_pcm_close (playback_handle);
  exit (0);
}
```

```
while (1) {
    int nread;
    if ((nread = snd_pcm_readi (capture_handle, buf, BUF_SIZE)) != BUF_SIZE)
{
        fprintf(stderr, "read from audio interface failed (%s)\n",
snd_strerror(nread));
        snd_pcm_prepare(capture_handle);
        continue;
    }
    printf("copying %d\n", nread);
    if ((err = snd_pcm_writei (playback_handle, buf, nread)) != nread) {
        if (err < 0) {
            fprintf(stderr, "write to audio interface failed (%s)\n",
snd_strerror (err));
        } else {
            fprintf (stderr, "write to audio interface failed after %d
frames\n", err);
        }
            snd_pcm_prepare(playback_handle);
    }
}
```

```c
#define PERIOD_SIZE 1024
#define BUF_SIZE (PERIOD_SIZE * 2)

#include <stdio.h>
#include <stdlib.h>
#include <alsa/asoundlib.h>

void print_pcm_state(snd_pcm_t *handle, char *name) {
  switch (snd_pcm_state(handle)) {
  case SND_PCM_STATE_OPEN:
    printf("state open %s\n", name);
    break;

  case SND_PCM_STATE_SETUP:
    printf("state setup %s\n", name);
    break;

  case SND_PCM_STATE_PREPARED:
    printf("state prepare %s\n", name);
    break;

  case SND_PCM_STATE_RUNNING:
    printf("state running %s\n", name);
    break;

  case SND_PCM_STATE_XRUN:
    printf("state xrun %s\n", name);
    break;

  default:
    printf("state other %s\n", name);
    break;

  }
}


int setparams(snd_pcm_t *handle, char *name) {
  snd_pcm_hw_params_t *hw_params;
  int err;


  if ((err = snd_pcm_hw_params_malloc (&hw_params)) < 0) {
    fprintf (stderr, "cannot allocate hardware parameter structure (%s)\n",
         snd_strerror (err));
    exit (1);
  }
```

```
  if ((err = snd_pcm_hw_params_any (handle, hw_params)) < 0) {
    fprintf (stderr, "cannot initialize hardware parameter structure
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_access (handle, hw_params,
SND_PCM_ACCESS_RW_INTERLEAVED)) < 0) {
    fprintf (stderr, "cannot set access type (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params_set_format (handle, hw_params,
SND_PCM_FORMAT_S16_LE)) < 0) {
    fprintf (stderr, "cannot set sample format (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  unsigned int rate = 48000;
  if ((err = snd_pcm_hw_params_set_rate_near (handle, hw_params, &rate, 0))
< 0) {
    fprintf (stderr, "cannot set sample rate (%s)\n",
         snd_strerror (err));
    exit (1);
  }
  printf("Rate for %s is %d\n", name, rate);

  if ((err = snd_pcm_hw_params_set_channels (handle, hw_params, 2)) < 0) {
    fprintf (stderr, "cannot set channel count (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  snd_pcm_uframes_t buffersize = BUF_SIZE;
  if ((err = snd_pcm_hw_params_set_buffer_size_near(handle, hw_params,
&buffersize)) < 0) {
    printf("Unable to set buffer size %i: %s\n", BUF_SIZE,
snd_strerror(err));
    exit (1);;
  }

  snd_pcm_uframes_t periodsize = PERIOD_SIZE;
  fprintf(stderr, "period size now %lu\n", periodsize);
  if ((err = snd_pcm_hw_params_set_period_size_near(handle, hw_params,
&periodsize, 0)) < 0) {
    printf("Unable to set period size %li: %s\n", periodsize,
```

```
snd_strerror(err));
    exit (1);
  }

  if ((err = snd_pcm_hw_params (handle, hw_params)) < 0) {
    fprintf (stderr, "cannot set parameters (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  snd_pcm_uframes_t p_psize;
  snd_pcm_hw_params_get_period_size(hw_params, &p_psize, NULL);
  fprintf(stderr, "period size %lu\n", p_psize);

  snd_pcm_hw_params_get_buffer_size(hw_params, &p_psize);
  fprintf(stderr, "buffer size %lu\n", p_psize);

  snd_pcm_hw_params_free (hw_params);

  if ((err = snd_pcm_prepare (handle)) < 0) {
    fprintf (stderr, "cannot prepare audio interface for use (%s)\n",
         snd_strerror (err));
    exit (1);
  }

  return 0;
}

int set_sw_params(snd_pcm_t *handle, char *name) {
  snd_pcm_sw_params_t *swparams;
  int err;

  snd_pcm_sw_params_alloca(&swparams);

  err = snd_pcm_sw_params_current(handle, swparams);
  if (err < 0) {
    fprintf(stderr, "Broken configuration for this PCM: no configurations
available\n");
    exit(1);
  }

  err = snd_pcm_sw_params_set_start_threshold(handle, swparams,
PERIOD_SIZE);
  if (err < 0) {
    printf("Unable to set start threshold: %s\n", snd_strerror(err));
    return err;
  }
  err = snd_pcm_sw_params_set_avail_min(handle, swparams, PERIOD_SIZE);
  if (err < 0) {
    printf("Unable to set avail min: %s\n", snd_strerror(err));
```

```
    return err;
  }

  if (snd_pcm_sw_params(handle, swparams) < 0) {
    fprintf(stderr, "unable to install sw params:\n");
    exit(1);
  }

  return 0;
}

/************** some code from latency.c *****************/

int main (int argc, char *argv[])
{
  int i;
  int err;
  int buf[BUF_SIZE];
  snd_pcm_t *playback_handle;
  snd_pcm_t *capture_handle;
  snd_pcm_hw_params_t *hw_params;
  FILE *fin;
  size_t nread;
  snd_pcm_format_t format = SND_PCM_FORMAT_S16_LE;
  if (argc != 3) {
    fprintf(stderr, "Usage: %s in-card out-card\n", argv[0]);
    exit(1);
  }

  /**** Out card *******/
  if ((err = snd_pcm_open (&playback_handle, argv[2],
SND_PCM_STREAM_PLAYBACK, 0)) < 0) {
    fprintf (stderr, "cannot open audio device %s (%s)\n",
         argv[2],
         snd_strerror (err));
    exit (1);
  }

  setparams(playback_handle, "playback");
  set_sw_params(playback_handle, "playback");


  /*********** In card **********/

  if ((err = snd_pcm_open (&capture_handle, argv[1], SND_PCM_STREAM_CAPTURE,
0)) < 0) {
    fprintf (stderr, "cannot open audio device %s (%s)\n",
         argv[1],
         snd_strerror (err));
    exit (1);
```

```
  }

  setparams(capture_handle, "capture");
  set_sw_params(capture_handle, "capture");

  if ((err = snd_pcm_link(capture_handle, playback_handle)) < 0) {
    printf("Streams link error: %s\n", snd_strerror(err));
    exit(0);
  }

  if ((err = snd_pcm_prepare (playback_handle)) < 0) {
    fprintf (stderr, "cannot prepare playback audio interface for use
(%s)\n",
         snd_strerror (err));
    exit (1);
  }

  /***************** stuff something into the playback buffer
****************/
  if (snd_pcm_format_set_silence(format, buf, 2*BUF_SIZE) < 0) {
    fprintf(stderr, "silence error\n");
    exit(1);
  }

  int n = 0;
  while (n++ < 2) {
    if (snd_pcm_writei (playback_handle, buf, BUF_SIZE) < 0) {
      fprintf(stderr, "write error\n");
      exit(1);
    }
  }

  /************** COPY ************/
  while (1) {
    int nread;
    if ((nread = snd_pcm_readi (capture_handle, buf, BUF_SIZE)) != BUF_SIZE)
{
      if (nread < 0) {
    fprintf (stderr, "read from audio interface failed (%s)\n",
         snd_strerror (nread));
      } else {
    fprintf (stderr, "read from audio interface failed after %d frames\n",
nread);
      }
      snd_pcm_prepare(capture_handle);
      continue;
    }

    if ((err = snd_pcm_writei (playback_handle, buf, nread)) != nread) {
      if (err < 0) {
```

```
    fprintf (stderr, "write to audio interface failed (%s)\n",
         snd_strerror (err));
      } else {
    fprintf (stderr, "write to audio interface failed after %d frames\n",
err);
      }
      snd_pcm_prepare(playback_handle);
    }
  }


  snd_pcm_drain(playback_handle);
  snd_pcm_close (playback_handle);
  exit (0);
}
```

Miksowanie z dmix

```
alsa_playback plug:dmix tmp1.s16 &
alsa_playback plug:dmix tmp2.s16 &
alsa_playback plug:dmix tmp3.s16
```

Miksowanie z PulseAudio

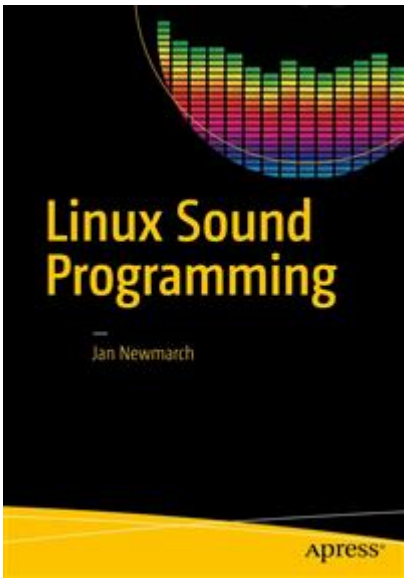
```
alsa_playback default tmp1.s16 &
alsa_playback pulse tmp2.s16 &
alsa_playback default tmp3.s16
```

Prosty mikser

```
#include <alsa/asoundlib.h>
#include <alsa/mixer.h>
#include <stdlib.h>

int main(int argc, char **argv) {

    snd_mixer_t *mixer;
    snd_mixer_selem_id_t *ident;
```

```
    snd_mixer_elem_t *elem;
    long min, max;
    long old_volume, volume;

    snd_mixer_open(&mixer, 0);
    snd_mixer_attach(mixer, "default");
    snd_mixer_selem_register(mixer, NULL, NULL);
    snd_mixer_load(mixer);

    snd_mixer_selem_id_alloca(&ident);
    snd_mixer_selem_id_set_index(ident, 0);
    snd_mixer_selem_id_set_name(ident, "Master");
    elem = snd_mixer_find_selem(mixer, ident);
    snd_mixer_selem_get_playback_volume_range(elem, &min, &max);
    snd_mixer_selem_get_playback_volume(elem, 0, &old_volume);
    printf("Min %ld max %ld current volume %ld\n", min, max, old_volume);

    if (argc < 2) {
    fprintf(stderr, "Usage: %s volume (%ld - %ld)\n", argv[0], min, max);
    exit(1);
    }
    volume = atol(argv[1]);
    snd_mixer_selem_set_playback_volume_all(elem, volume);
    printf("Volume reset to %ld\n", volume);

    exit(0);
}
```

Żródła


Linux Sound
Programming
Jan Newmarch
Apress