**USB TO AUDIO USERSPACE**

In this case, USB device is the standard audio device that communicates via USB, and ALSA is used to handle the audio stream.

**Overview:**

* Use **libusb** to interact with the USB device at the userspace level.
* Use **ALSA** to send audio data to the sound card.

The following userspace code will capture basic functionality. It does not fully implement all USB-to-audio functionalities (which are more complicated), but it provides a framework to interact with the USB audio device from userspace and playback audio using ALSA.

**Required Libraries:**

* **libusb** (for USB communication)
* **ALSA** (for handling audio playback)

Ensure both libraries are installed:

sudo apt-get install libusb-1.0-0-dev libasound2-dev

**Example Userspace Code:**

This code initializes a USB device and attempts to send audio data through ALSA for playback.

#include <stdio.h>

#include <stdlib.h>

#include <alsa/asoundlib.h>

#include <libusb-1.0/libusb.h>

// ALSA settings

#define SAMPLE\_RATE 44100 // Standard audio sample rate

#define CHANNELS 2 // Stereo

#define PCM\_DEVICE "default"

// Function to initialize ALSA for audio playback

snd\_pcm\_t\* init\_alsa(unsigned int sample\_rate, int channels) {

snd\_pcm\_t \*pcm\_handle;

snd\_pcm\_hw\_params\_t \*params;

unsigned int rate = sample\_rate;

int dir;

// Open PCM device for playback

if (snd\_pcm\_open(&pcm\_handle, PCM\_DEVICE, SND\_PCM\_STREAM\_PLAYBACK, 0) < 0) {

fprintf(stderr, "Error opening PCM device %s\n", PCM\_DEVICE);

return NULL;

}

// Allocate parameters object and fill it with default values

snd\_pcm\_hw\_params\_malloc(&params);

snd\_pcm\_hw\_params\_any(pcm\_handle, params);

// Set parameters for ALSA

snd\_pcm\_hw\_params\_set\_access(pcm\_handle, params, SND\_PCM\_ACCESS\_RW\_INTERLEAVED);

snd\_pcm\_hw\_params\_set\_format(pcm\_handle, params, SND\_PCM\_FORMAT\_S16\_LE);

snd\_pcm\_hw\_params\_set\_channels(pcm\_handle, params, channels);

snd\_pcm\_hw\_params\_set\_rate\_near(pcm\_handle, params, &rate, &dir);

// Write the parameters to the driver

if (snd\_pcm\_hw\_params(pcm\_handle, params) < 0) {

fprintf(stderr, "Unable to set HW parameters\n");

snd\_pcm\_hw\_params\_free(params);

return NULL;

}

snd\_pcm\_hw\_params\_free(params);

return pcm\_handle;

}

// Function to initialize USB using libusb

libusb\_device\_handle\* init\_usb\_audio\_device() {

libusb\_device\_handle \*dev\_handle;

libusb\_context \*ctx = NULL;

int r;

r = libusb\_init(&ctx);

if (r < 0) {

fprintf(stderr, "Error initializing libusb: %d\n", r);

return NULL;

}

// Open the first audio USB device found (you may need to adjust the vendor/product IDs)

dev\_handle = libusb\_open\_device\_with\_vid\_pid(ctx, 0xXXXX, 0xXXXX); // Replace with your device's Vendor and Product IDs

if (!dev\_handle) {

fprintf(stderr, "Cannot open USB device\n");

libusb\_exit(ctx);

return NULL;

}

// Claim the audio interface (you might need to adjust the interface number)

if (libusb\_claim\_interface(dev\_handle, 0) < 0) {

fprintf(stderr, "Cannot claim interface\n");

libusb\_close(dev\_handle);

libusb\_exit(ctx);

return NULL;

}

return dev\_handle;

}

// Main function to play audio

int main() {

snd\_pcm\_t \*pcm\_handle;

libusb\_device\_handle \*usb\_handle;

short \*buffer;

int buffer\_size = 2048; // Size of the buffer in frames

int err, frames\_written;

// Initialize USB audio device

usb\_handle = init\_usb\_audio\_device();

if (!usb\_handle) {

fprintf(stderr, "Failed to initialize USB audio device\n");

return -1;

}

// Initialize ALSA

pcm\_handle = init\_alsa(SAMPLE\_RATE, CHANNELS);

if (!pcm\_handle) {

fprintf(stderr, "Failed to initialize ALSA\n");

libusb\_close(usb\_handle);

return -1;

}

// Allocate buffer for audio data

buffer = (short \*)malloc(buffer\_size \* CHANNELS \* sizeof(short));

// Generate a sine wave as a simple test sound

double frequency = 440.0; // A4 note

double phase = 0.0;

double phase\_increment = 2.0 \* M\_PI \* frequency / SAMPLE\_RATE;

for (int i = 0; i < buffer\_size; i++) {

buffer[2 \* i] = (short)(32767 \* sin(phase)); // Left channel

buffer[2 \* i + 1] = (short)(32767 \* sin(phase)); // Right channel

phase += phase\_increment;

if (phase > 2.0 \* M\_PI)

phase -= 2.0 \* M\_PI;

}

// Play the audio using ALSA

for (int i = 0; i < 10; i++) { // Play 10 times (for demonstration)

frames\_written = snd\_pcm\_writei(pcm\_handle, buffer, buffer\_size);

if (frames\_written < 0) {

frames\_written = snd\_pcm\_recover(pcm\_handle, frames\_written, 0);

}

if (frames\_written < 0) {

fprintf(stderr, "Error writing to ALSA device: %s\n", snd\_strerror(frames\_written));

break;

}

}

// Cleanup

free(buffer);

snd\_pcm\_drain(pcm\_handle);

snd\_pcm\_close(pcm\_handle);

libusb\_release\_interface(usb\_handle, 0);

libusb\_close(usb\_handle);

return 0;

}

**Key Components of the Code:**

1. **USB Initialization with libusb:**
   * The function init\_usb\_audio\_device() initializes the USB audio device using libusb. You'll need to replace 0xXXXX in libusb\_open\_device\_with\_vid\_pid() with your actual **vendor** and **product ID** of the USB audio device.
   * The program claims the interface for the USB audio device to communicate with it.
2. **ALSA Initialization for Audio Playback:**
   * The init\_alsa() function initializes ALSA to handle audio playback. It sets the sample rate, number of channels (stereo), and the audio format (16-bit, little-endian).
   * ALSA is used to send audio data to the sound card for playback.
3. **Sine Wave Generation:**
   * The buffer stores a sine wave with a frequency of **440 Hz** (A4 note). This sine wave is used as a test sound and is played back using ALSA.
4. **Audio Playback:**
   * The sine wave buffer is repeatedly played back using the snd\_pcm\_writei() function from ALSA. It handles audio in **interleaved mode** for stereo output.

**Running the Code:**

1. Compile the program with the following command:

gcc -o usb\_audio usb\_audio.c -lusb-1.0 -lasound -lm

1. Run the program:

./usb\_audio

**Notes:**

* The code assumes basic USB audio handling. For a real-world application, you'll need to add more sophisticated USB control (e.g., handling endpoints, streaming audio data over USB).
* You'll need to replace the **vendor** and **product ID** with the appropriate values for your USB-to-audio device.
* This code is a basic example and may need further adjustment depending on your USB audio device and system setup.