1 PortAudio Application Programming Interface (API)

Material for this lecture is taken from the PortAudio website:

http://www.portaudio.com/

There is a tutorial which can be found at

http://www.portaudio.com/trac/wiki/TutorialDir/TutorialStart

2 Introduction

PortAudio is a free, open-source, cross-platform, 'C' library for audio input and output (I/O). It lets you write simple audio programs in 'C' that will compile and run on many platforms including Linux, OS X, and Windows. This means that you can write a simple 'C' program to process or generate an audio signal, and that program can run on several different platforms just by recompiling the source code.

PortAudio provides a very simple API for recording and/or playing sound using a simple callback function. Prof. De Leon has written a passcode which uses the PortAudio API. This passcode can be used for developing real-time DSP applications using a standard soundcard.

A PortAudio-based application is written with the following steps:

1. Write a *callback function* that will be called by PortAudio when audio processing is needed
2. Initialize the PA library and open a stream for audio I/O
3. Start the stream. Your callback function will be now be called repeatedly by PA in the background
4. In your callback you can read audio data from the inputBuffer and/or write data to the outputBuffer
5. Stop the stream by returning 1 from your callback, or by calling a stop function
6. Close the stream and terminate the library

3 Compiling PortAudio

To begin, you must first download PortAudio

http://www.portaudio.com/download.html

and compile/install for your system. System-dependent directions are given at

http://www.portaudio.com/trac/wiki/TutorialDir/TutorialStart

4 Writing a Callback Function

To write a program using PortAudio, you must include the "portaudio.h" header file. You may wish to read "portaudio.h" because it contains a complete description of the PortAudio functions and constants.

The next task is to write a callback function that is *called* by the PortAudio engine whenever it has captured audio data, or when it needs more audio data for output.

Before we begin, it's important to realize that the callback is a delicate place. This is because some systems perform the callback in a special thread, or interrupt handler, and it is rarely treated the same as the rest of your code. In addition, if you want your audio to reach the speakers on time, you'll need to make sure whatever code you run in the callback runs quickly. What is safe or not safe will vary from platform to platform, but as a rule of thumb, don't do anything like allocating or freeing memory, reading or writing files, printf(), or anything else that might take an unbounded amount of time or rely on the OS or require a context switch. Also do not call any PortAudio functions in the callback except for Pa_StreamTime() and Pa_GetCPULoad().

Your callback function must return an int and accept the exact parameters specified in this typedef:

```
1 typedef int PaStreamCallback( const void *input,
2                                void *output,
3                                unsigned long frameCount,
4                                const PaStreamCallbackTimeInfo* timeInfo,
5                                PaStreamCallbackFlags statusFlags,
6                                void *userData ) ;
```

See Prof. De Leon's passcode lines 40 – 64 at the end of this lecture for implementation of the callback function.

Note that in this example, the samples are of type float and the samples will be in the range -1.0 to $+1.0$.

5 Initializing PortAudio

Before making any other calls to PortAudio, you must call Pa_Initialize(). This will trigger a scan of available devices (soundcard) which can be queried later. Like most PA functions, it will return a result of type paError. If the result is not paNoError, then an error has occurred.

```
err = Pa_Initialize();
if( err != paNoError ) goto error;
```

You can get a text message that explains the error message by passing it to Pa_GetErrorText(err). For Example:

```
printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
```

It is also important, when you are done with PortAudio, to Terminate it:

```
err = Pa_Terminate();
if( err != paNoError )
   printf(  "PortAudio error: %s\n", Pa_GetErrorText( err ) );
```

See Prof. De Leon's passcode lines 73 – 75 at the end of this lecture for implementation of the above.

6 Opening a Stream using Defaults

The next step is to open a stream, which is similar to opening a file. You can specify whether you want audio input and/or output, how many channels, the data format, sample rate, etc. Opening a default stream means opening the default input and output devices, which saves you the trouble of getting a list of devices and choosing one from the list.

See Prof. De Leon's passcode lines 80 – 91 at the end of this lecture for implementation of the above. In the passcode, we choose stereo (2 channels) I/O, 32 bit (float) samples, and 48000 Hz sample rate.

7 Starting, Stopping and Aborting a Stream

PortAudio will not start playing back audio until you start the stream. After calling Pa_StartStream(), PortAudio will start calling your callback function to perform the audio processing.

```
err = Pa_StartStream( stream );
if( err != paNoError ) goto error;
```

You can communicate with your callback routine through the data structure you passed in on the open call. See Prof. De Leon's passcode lines 93 – 95 at the end of this lecture for implementation of the above.

PortAudio will continue to call your callback and process audio until you stop the stream. This can be done in one of several ways, but, before we do so, we'll want to see that some of our audio gets processed by sleeping for a few seconds. This is easy to do with Pa_Sleep(), which is used by many of the examples in the patests/ directory for exactly this purpose.

```
/* Sleep for several seconds. */
Pa_Sleep(NUM_SECONDS*1000);
```

See Prof. De Leon's passcode lines 97 – 103 at the end of this lecture for implementation of the above as well as an implementation which runs until the RETURN is hit.

Now we need to stop the audio stream. There are several ways to do this, the simplest of which is to call Pa_StopStream():

```
err = Pa_StopStream( stream );
if( err != paNoError ) goto error;
```

Pa_StopStream() is designed to make sure that the buffers you've processed in your callback are all played, which may cause some delay. See Prof. De Leon's passcode lines 105 – 107 at the end of this lecture for implementation of the above.

8 Closing a Stream and Terminating PortAudio

When you are done with a stream, you should close it to free up resources:

```
err = Pa_CloseStream( stream );
if( err != paNoError ) goto error;
```

We've already mentioned this in Initializing PortAudio, but in case you forgot, be sure to terminate PortAudio when you are done:

```
err = Pa_Terminate( );
if( err != paNoError ) goto error;
```

See Prof. De Leon's passcode lines 105 – 115 at the end of this lecture for implementation of the above.

9 Utility Functions

In addition to the functions described elsewhere in this tutorial, PortAudio provides a number of Utility functions which are useful in a variety of circumstances.

PortAudio allows you to get error text from an error number.

```
const char *  Pa_GetErrorText (PaError errorCode)
```

See Prof. De Leon's passcode lines 117 – 123 at the end of this lecture for use and implementation of the above.

10 MAIN.C

```
1  /*
2   * MAIN.C
3   *
4   * Purpose: This code uses PortAudio to send/receive samples to/from a sound card.
5   *    DSP algorithms can be implemented within this framework on an ordinary PC
6   *    and easily ported to target hardware. The routine process_signal is where
7   *    the developer should focus the effort.  The file process_signal.c contains
8   *    the routine which sets the outputRightSample, outputLeftSample
9   *    equal to the inputRightSample, inputLeftSample respectively i.e. passes
10  *    input straight to output—passcode.  Finally, the program is stopped by
11  *    hitting return in the console.
12  *
13  * Arguments: None
14  *
15  * Notes:
16  *    Developer must download and build PortAudio first http://www.portaudio.com/download.html
17  *
18  *    See http://www.portaudio.com/trac/wiki/TutorialDir/Compile/MacintoshCoreAudio
19  *    for information on how to build PortAudio for Mac OS X
20  *
21  *    Unless headphones are used, there may be audio feedback.
22  *
23  *    Set Header and Library Search Paths (for PortAudio files) in Edit Project Settings
24  *
25  * References:
26  *   http://www.portaudio.com/trac/wiki/TutorialDir/TutorialStart
27  *
28  * Creation Date: 2009-Mar-31 (P. De Leon)
29  *
30  * Revision History:
31  *    1.0 2009-Mar-31 Base code working (P. De Leon)
32  */
33 #include <portaudio.h>
34 //#include "/opt/local/include/portaudio.h"
35 #include <stdio.h>
36 #include <stdlib.h>
37
38 #include "user_data.h"
39
40 /*******************************/
41 /* PortAudio Callback function */
42 /*******************************/
43 static int pa_callback(void *inputBuffer, void *outputBuffer,
44                        unsigned long framesPerBuffer,
45                        const PaStreamCallbackTimeInfo* timeInfo,
46                        PaStreamCallbackFlags statusFlags,
47                        void *userData )
48 {
49     int i;
50     float *pi, *po;
51     float inputRightSample, inputLeftSample, outputRightSample, outputLeftSample;
52
53     pi = (float*)inputBuffer;
54     po = (float*)outputBuffer;
55     for (i=0; i<framesPerBuffer; i++)
56     {
57         inputRightSample = *pi++;
58         inputLeftSample = *pi++;
59         process_signal(inputRightSample, inputLeftSample, &outputRightSample, &outputLeftSample);
60         *po++ = outputRightSample;
61         *po++ = outputLeftSample;
62     }
63     return 0;
64 }
65
66 /*******************************/
67 int main(void)
68 /*******************************/
69 {
70     PaError err;
71     PaStream *stream;
72
73     /* Initialize PortAudio */
74     err = Pa_Initialize();
```

```
75      if (err != paNoError) goto error;
76
77      /* Program Initialization */
78      initialize_program();
79
80      /* Open a Stream */
81      err = Pa_OpenDefaultStream(
82          &stream,        /* passes back stream pointer */
83          2,              /* 2 channels (stereo) input */
84          2,              /* 2 channels (stereo) output */
85          paFloat32,      /* 32 bit floating point samples */
86          48000,          /* sample rate */
87          256,            /* frames per buffer */
88          // paFramesPerBufferUnspecified,   /* let PA pick best number of frames per buffer */
89          pa_callback,    /* this is the callback function */
90          0 );            /* pass our data through to callback (0 Hans, &data in tutorial???) */
91      if( err != paNoError ) goto error;
92
93      /* Start a Stream */
94      err = Pa_StartStream( stream );
95      if( err != paNoError ) goto error;
96
97      /* Run for 60 seconds. */
98      //Pa_Sleep(60*1000);
99
100     /* Run until user hits RETURN on keyboard */
101     printf("Hit RETURN to quit.\n");
102     fflush(stdout);
103     getchar();
104
105     /* Stop a Stream */
106     err = Pa_StopStream( stream );
107     if( err != paNoError ) goto error;
108
109     /* Close Stream */
110     err = Pa_CloseStream( stream );
111     if( err != paNoError ) goto error;
112
113     Pa_Terminate();
114     printf("Program completed.\n");
115     return err;
116
117 error:
118     Pa_Terminate();
119     fprintf( stderr, "An error occured while using the portaudio stream\n" );
120     fprintf( stderr, "Error number: %d\n", err );
121     fprintf( stderr, "Error message: %s\n", Pa_GetErrorText( err ) );
122     return err;
123 }
```