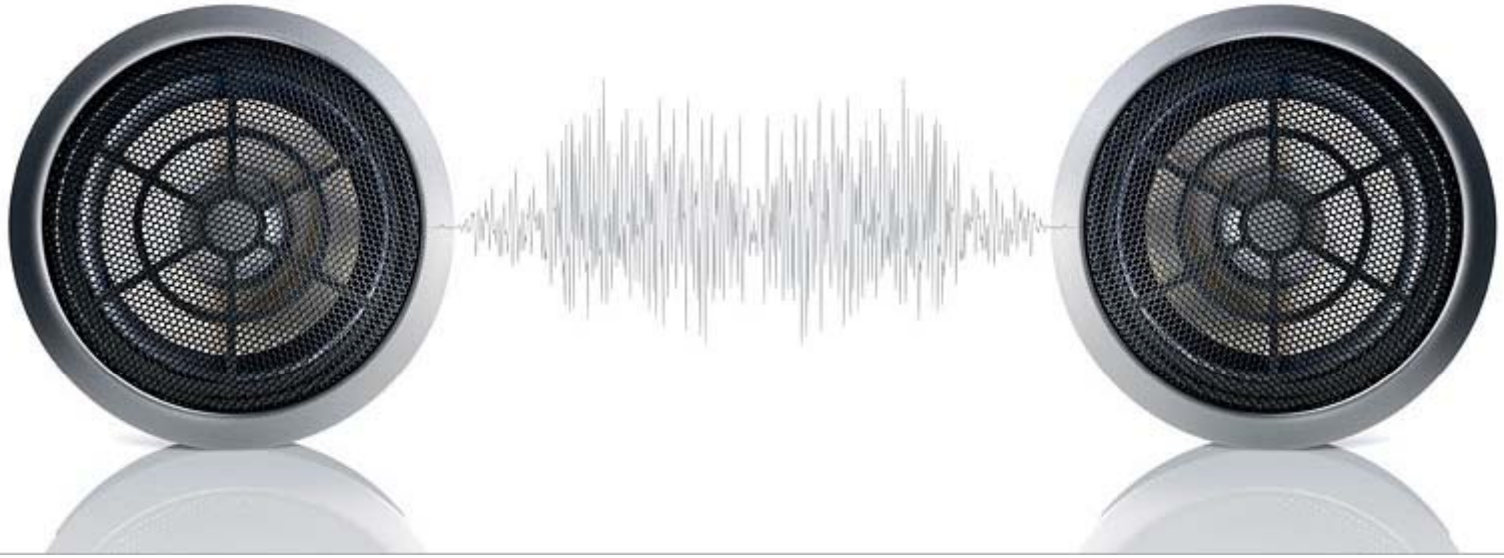
Real Time Audio Streaming con:

PortAudio

Portable Cross-platform Audio I/O

w w w . p o r t a u d i o . c o m



Riferimenti

Ing. Antonio Salini
Mail: antonio.salini@gmail.com
Web: www.antoniosalini.it

Definizioni:

Dispositivo hardware di I/O:
Può essere una scheda audio connessa al computer host tramite USB, Firewire (IEEE1394) o direttamente su bus di sistema.

Driver di periferica:
Modulo software che interfaccia il sistema operativo con la periferica hardware. Fornisce al sistema l'accesso a tutte le funzioni messe a disposizione dalla periferica e si prende carico delle segnalazioni che vengono dalla periferica stessa (interrupt).

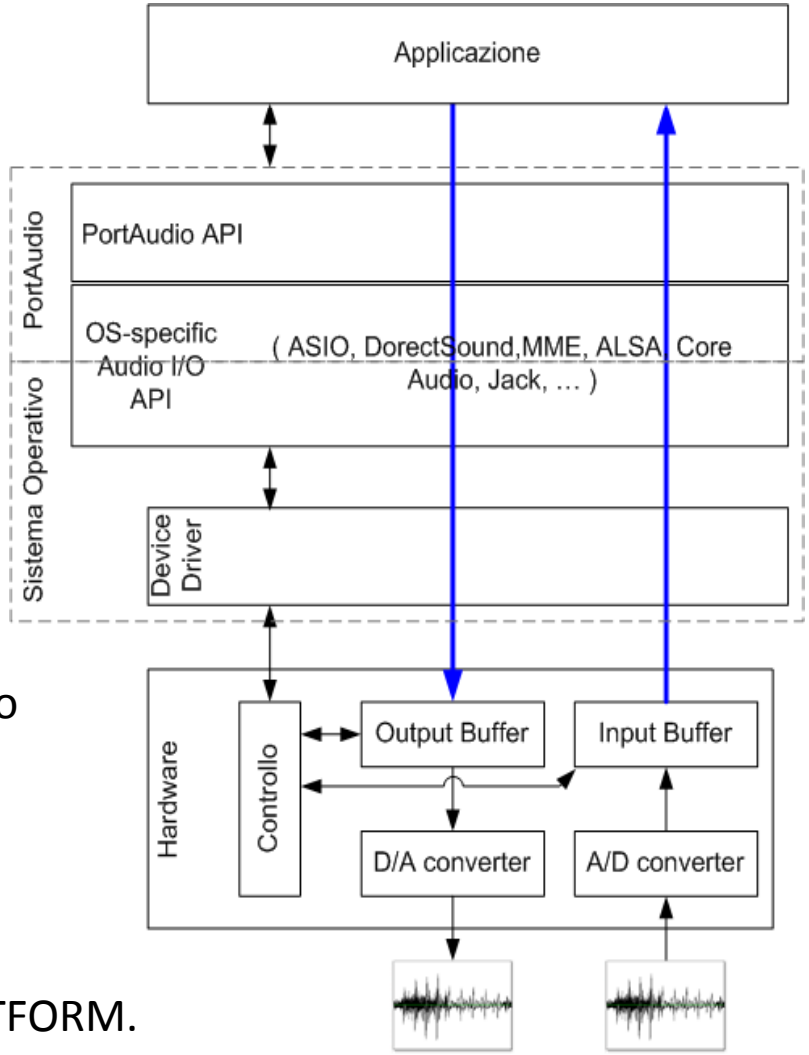
OS-specific API (Application Programming Interface):
Le funzioni, le strutture dati, le costanti, necessarie all'inizializzazione, l'avvio, la gestione degli streams audio, costituisce l'interfaccia di programmazione della libreria. Lo sviluppatore accede alle funzionalità della libreria utilizzando quanto messo a disposizione dalle API. Ciascun sistema operativo ha uno o più possibili sottosistemi audio, ciascuno con le proprie API:

Windows	Mac Os X	Linux
MME	Core Audio	ALSA
ASIO		JACK
DirectSound		OSS
WASAPI		

Cos’è PortAudio:

E’ una libreria C open-source per lo streaming audio real time.
Le sue peculiarità sono le seguenti:

- Essere in grado di interfacciarsi verso il basso con qualsiasi sottosistema audio.
- Fornire allo sviluppatore una API unificata, indipendentemente dal sistema operativo, dal sotto-sitema audio o dell’HW.



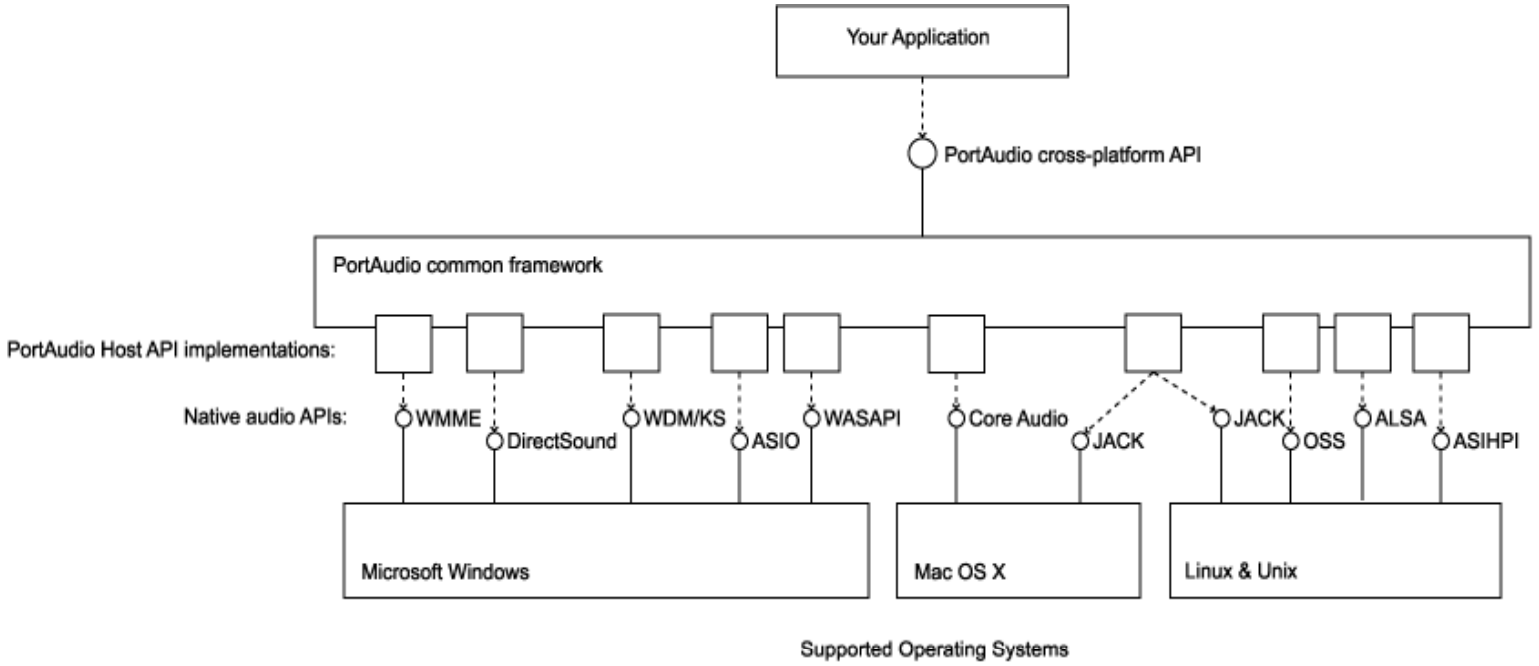
Sviluppare una applicazione audio prendendo a riferimento le specifiche date dalle API PortAudio consente di portare il proprio codice da Windows a Linux a MacOS senza modificare il core audio dell’applicazione.

Questa peculiarità viene definita CROSS-PLATFORM.

Cos’è PortAudio:

Lo schema seguente è stato ripreso dalla documentazione ufficiale di PortAudio:

- Espone in modo esaustivo tutte le Api audio native con cui PortAudio riesce ad interfacciarsi, in relazione ciascuna con il rispettivo sistema operativo.
- Mostra in modo chiaro come l’interfaccia tra l’applicazione e il framework di PortAudio sia unica ed indipendente dalle strutture di basso livello.



Il meccanismo dello streaming audio:

Lo streaming audio viene generalmente gestito tramite interrupt.

Il dispositivo di I/O ha a sua disposizione dei buffer su cui vengono caricati i campioni:
- dal convertitore A/D nel caso di input stream (buffer di input)
- dal sistema nel caso di output stream (buffer di output)

Al verificarsi della condizone “Buffer pieno” nel caso di buffer di input oppure “Buffer vuoto” nel caso di buffer di output, il dispositivo di I/O chiamerà un interrupt per segnalare al sistema che è il momento di:
- Prelevare i dati dal buffer di input
- Riversare dati sul buffer di output

L’elemento software che si prende immediatamente carico della segnalazione è il driver della periferica.

La segnalazione risale la gerarchia riportata nella slide precedente arrivando infine all’applicazione, che tramite le funzionalità fornite da PortAudio e alle funzioni implementate dallo sviluppatore (***funzione/i di Call-Back***), saprà come gestire i buffer.

Il meccanismo dello streaming audio:

Due parametri importanti per lo streaming sono:

- La dimensione dei buffer: FRAME_PER_BUFFER
- La frequenza di campionamento: SAMPLE_RATE

Naturalmente, una volta impostati questi due valori, gli intervalli di tempo per l'intervento sui buffer saranno prevedibili e calcolabili come:

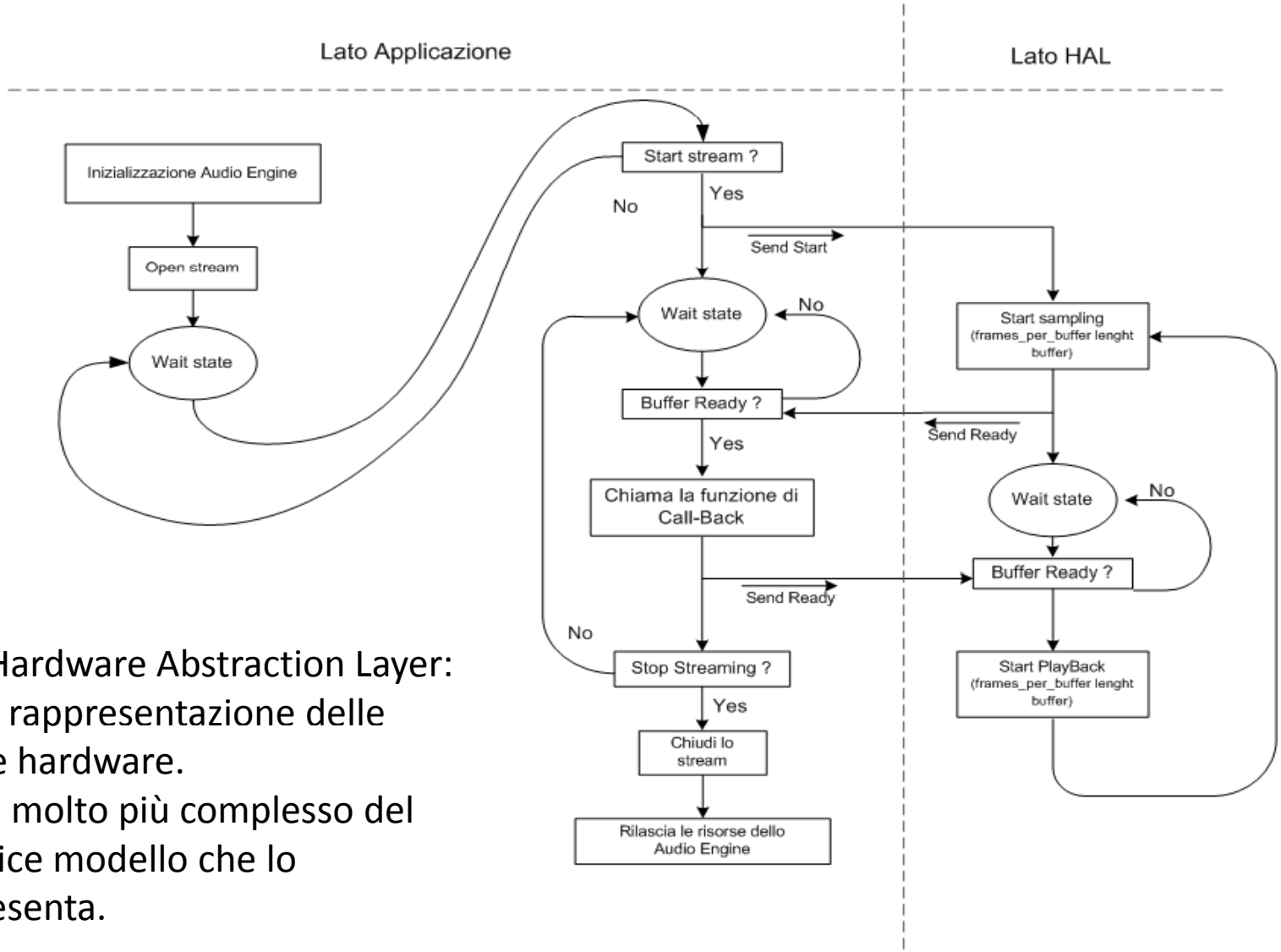
$$\frac{\texttt{frames\_per\_buffer}}{\texttt{sample\_rate}} = t\ [\text{sec}]$$

Es. Con un valore FPB pari a 256 campioni e una SR pari a 44100 Hz, si ottiene un tempo di svuotamento (o riempimento) dei buffer pari a 5.8 ms.

Questi due parametri, e la semplice relazione che li lega, hanno un importante ruolo nella definizione della latenza tra la generazione algoritmica della forma d'onda e l'effettiva resa del suono in uscita dall'hw.

Questo aspetto va tenuto in considerazione soprattutto nel caso di algoritmi molto complessi in cui la macchina potrebbe richiedere molto tempo per produrre la sequenza di campioni da caricare nel buffer.

Il meccanismo dello streaming audio:



HAL: Hardware Abstraction Layer:
E' una rappresentazione delle risorse hardware.
L'hw è molto più complesso del semplice modello che lo rappresenta.

Il meccanismo dello streaming audio:

Come fa l'applicazione a reagire alla segnalazione dell'interrupt ?

PortAudio affida questo compito ad una funzione di tipo "Call-Back", cioè, senza entrare nei dettagli, ad una funzione invocata dal sistema.
Tale funzione, che può avere un qualsiasi nome, deve essere passata come parametro alla funzione di PortAudio che si occupa di avviare lo stream:

```
Pa_OpenStream(… , … , … , … , My_Callback , …  )
```

Di tutti i parametri di questa funzione ci occuperemo nelle slides seguenti.

Naturalmente la funzione di Call-Back deve rispettare degli standard, in particolare il **tipo di ritorno** e la **lista dei parametri di ingresso**:

```
Static int My_Callback(const void *inputBuffer,
                       Void *outputBuffer,
                       Unsigned long framesPerBuffer,
                       Const PaStreamCallbackTimeInfo* timeInfo,
                       PaStreamCallbackFlags statusFlags,
                       Void* userData)
{
   //codice di elaborazione dei dati audio
}
```

Il meccanismo dello streaming audio:

La sequenza di attivazione dello streaming è relativamente semplice:

1. **Inizializzazione del motore audio**

   E' necessario definire ed inizializzare le variabili di cui PortAudio ha bisogno:

   - Di strutture per memorizzare i parametri degli stream che verranno creati:

     ```
     PaStreamParameters parametriIngresso;
     PaStreamParameters parametriUscita;
     ```

   - Una variabile di tipo puntatore allo stream (stream handler)

     ```
     PaStream *stream;
     ```

   - Una variabile per la gestione degli eventuali messaggi di errore (o di successo) ricevuti dalle funzioni PortAudio.

     ```
     PaError err;
     ```

   Si può a questo punto inizializzare il PortAudio:

   ```
   Err= Pa_Initialize();
   ```

   ..effettuare un controllo sul risultato dell'inizializzazione:

   ```
   If (err != paNoError ) goto error;
   ```

   Error è una etichetta che identifica una sezione di codice.

Il meccanismo dello streaming audio:

Impostazione dei valori dei parametri delle strutture:

```
parametriIngresso.device == PaGetDefaultInputDevice();
parametriIngresso.channelCount = 2;
parametriIngresso.sampleFormat = PA_SAMPLE_TYPE;
parametriIngresso.suggestedLatency =Pa_GetDeviceInfo(inputParameters.device)>defaultLowInputLatency;
parametriIngresso.hostApiSpecificStreamInfo = NULL;

parametriUscita.device == PaGetDefaultOutputDevice();
parametriIngresso.channelCount = 2;
parametriIngresso.sampleFormat = PA_SAMPLE_TYPE;
parametriIngresso.suggestedLatency = Pa_GetDeviceInfo(inputParameters.device)->defaultLowInputLatency;
parametriIngresso.hostApiSpecificStreamInfo = NULL
```

- *Device* imposta l'hw sul quale eseguire lo streaming. Prima di avviare l'applicazione è necessario andare ad impostare come "default" il device desiderato tramite le impostazioni di sistema del sistema operativo.
- *ChannelCount* imposta il numero di canali su cui effettuare lo stream
- *sampleFormat* imposta il formato dei campioni, dichiarato come direttiva #include all'inizio del programma
- *suggestedLatency* imposta la latenza di I/O, indipendentemente dalla latenza di elaborazione dovuta all'algoritmo. Qui è possibile inserire un valore numerico che esprime la latenza desiderata in secondi.
- *hostApiSpecificStreamInfo* è un puntatore ad eventuali strutture dati necessarie al setup della periferica o all'elaborazione dello stream ma esula dal contesto. Potrebbe essere anche ignorato ma è meglio forzare a NULL piuttosto che lasciarlo in uno stato indefinito.

Il meccanismo dello streaming audio:

2. **Apertura dello stream**

```
err = Pa_OpenStream( &stream, &parametriIngresso, &parametriUscita, SAMPLE_RATE,
                                                    FREAMES_PER_BUFFER,0, My_Callback, NULL);
```

   - I primi tre parametri sono dei riferimenti allo stream handler e alle strutture contenenti i parametri.
   - SAMPLE_RATE e FRAMES_PER_BUFFER sono le costanti dichiarate tramite direttive #define all'inizio del programma
   - My_Callback è la funzione chiave nell'elaborazione dello stream.
   - L'ultimo parametro nella definizione della funzione è di tipo void*, può essere tutto e niente, verrà eventualmente tipizzato nella funzione tramite casting. Questo parametro è utile per portare nella funzione di Call-Back eventuali strutture dati necessarie per l'elaborazione.

3. **Avvio dello stream**

```
err = Pa_StartStream(stream);
```

4. **Chiusura dello stream**

```
err = Pa_CloseStream(stream);
```

5. **Deallocazione delle risorse**.

```
Pa_Terminate();
```

Es1-Un semplice programma 1/2:

```
#include “portaudio.h”
#include “stdio.h”

#define SAMPLE_RATE                    44100
#define PA_SAMPLE_TYPE    paFloat32
#define FRAMES_PER_BUFFER 256

typedef float SAMPLE;

static int My_Callback(const void *inputBuffer, void *outputBuffer, unsigned long framesPerBuffer,
                       const PaStreamCallbackTimeInfo* timeInfo, paStreamCallbackFlags statusFlags,
                       void* userData)
{
   SAMPLE *out = (SAMPLE*)outputBuffer;
   const SAMPLE *in = (const SAMPLE*)inputBuffer;
   unsigned int count;
   (void) timeInfo;
   (void) statusFlags;
   (void) userData;

   if (inputBuffer == NULL)
      for (count = 0; count < framesPerBuffer; count++)
      {
         *out++ = 0;
         *out++ = 0;
      }
    else
      for (count = 0; count < framesPerBuffer; count++)
      {
         *out++ = *in++;
         *out++ = *in++;
      }
 return paContinue;
}
```

Es1-Un semplice programma 2/2:

```
int main(void)
{
    PaStreamParameters parametriIngresso, parametriUscita;
    PaStream *stream;
    PaError err;

    err = Pa_Initialize();
    if (err != paNoError) goto Error;

    parametriIngresso.device == Pa_GetDefaultInputDevice();
    if (parametriIngresso.device == paNoDevice) goto Error;
    parametriIngresso.channelCount = 2;
    parametriIngresso.sampleFormat = PA_SAMPLE_TYPE;
    parametriIngresso.suggestedLatency = Pa_GetDeviceInfo(parametriIngresso.device)->defaultLowInputLatency;
    parametriIngresso.hostApiSpecificStreamInfo = NULL;

    parametriUscita.device == Pa_GetDefaultOutputDevice();
    if (parametriUscita.device == paNoDevice) goto Error;
    parametriUscita.channelCount = 2;
    parametriUscita.sampleFormat = PA_SAMPLE_TYPE;
    parametriUscita.suggestedLatency = Pa_GetDeviceInfo(parametriUscita.device)->defaultLowInputLatency;
    parametriUscita.hostApiSpecificStreamInfo = NULL;

    err = Pa_OpenStream( &stream, &parametriIngresso, &parametriUscita, SAMPLE_RATE, FREAMES_PER_BUFFER,
                                                                                    0, My_Callback, NULL);
    err = Pa_StartStream(stream);
    printf("Premi un tasto qualsiasi per interrompere lo streaming.");
    getchar();
    err = Pa_CloseStream(stream);
    Pa_Terminate();
    return 0;

Error:
    Pa_Terminate();
    return -1;
}
```

Qualche commento al codice:

Prendiamo a riferimento le istruzioni che compaiono tra le prime righe della funzione di Call-Back:

```
SAMPLE *out = (SAMPLE*)outputBuffer;
const SAMPLE *in = (const SAMPLE*)inputBuffer;
(void) timeInfo;
(void) statusFlags;
(void) userData;
```

Queste istruzioni effettuano la conversione forzata tra tipi di dato, in gergo informatico ci si riferisce a questa tecnica come “casting”.
Le prime due righe sono le più importanti poiché forzano il sistema ad interpretare i valori numerici letti dal buffer come numeri in virgola mobile a 32 bit, alla luce di quanto dichiarato nella direttiva:

```
typedef float SAMPLE;
```

all’inizio del file di programma.

Qualche commento al codice:

Cosa fa questo programma ?
Nulla di particolarmente utile, prende lo stream proveniente da una sorgente di ingresso e lo ripropone esattamente uguale in uscita.

Come fare per inserire qualche tipo di elaborazione del segnale ?
Bisogna inserire il codice all'interno della funzione My_Callback:

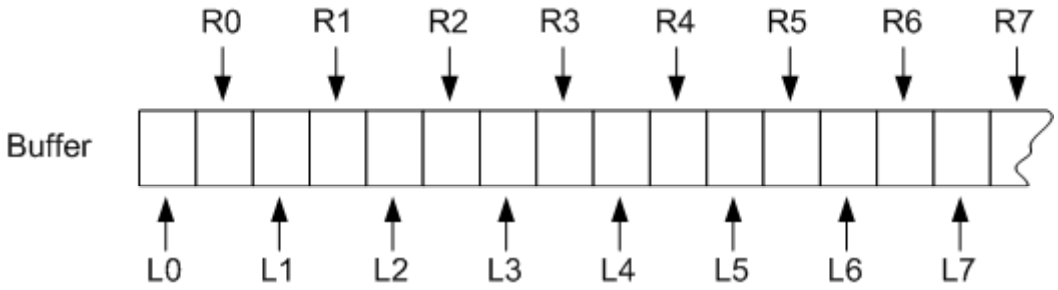
```
static int My_Callback(const void *inputBuffer, void *outputBuffer, …, …, …, …, …)
{
   …
   …

   if (inputBuffer == NULL)
      for (count = 0; count < framesPerBuffer; count++)
      {
         *out++ = 0;
         *out++ = 0;
      }
    else
      for (count = 0; count < framesPerBuffer; count++)
      {
         *out++ = unaQualsiasiMiaFunzione(*in++);
         *out++ = unaQualsiasiMiaFunzione(*in++);
      }
 return paContinue;
}
```

Qualche commento al codice:

```
for (count = 0; count < framesPerBuffer; count++)
      {
         *out++ = unaQualsiasiMiaFunzione(*in++);
         *out++ = unaQualsiasiMiaFunzione(*in++);
      }
```

In realtà i campioni de due buffer dei canali left e right sono memorizzati in un unico buffer fisico, in modo “interleaved”:



L’istruzione *out++ scrive il campione sul buffer di uscita e salta alla locazione successiva.

Qualche commento al codice:

```
*out++ = unaQualsiasiMiaFunzione(*in++);
```

Questa istruzione implica un'elaborazione "istantanea" dei campioni dello stream, in cui cioè non è necessario tenere in considerazione lo stato di un eventuale sistema; ove per sistema si intende un filtro, un generatore d'onda, ecc..

Per questo tipo di applicazioni PortAudio mette a disposizione l'ultimo parametro della funzione di Call-Back:

```
Static int My_Callback(…,…,…,…,…,Void* userData)
```

La variabile userData è un puntatore ad una qualsiasi struttura dati. Il fatto che sia dichiarata come void* è legato alla sua generalità.
Tra le prime righe della funzione di Call-Back è necessario effettuare un casting sulla variabile per utilizzarla così come desiderato dallo sviluppatore:

```
myWaveData * data    = (myWaveData *)userData;
```

La variabile "data" sarà un puntatore alla struttura dati contenente le informazioni.

Esempio di definizione di struttura dati:

Poiché è il sistema a chiamare la funzione di Call-Back, come si fa a passargli il parametro “userData” ?

Il parametro va passato alla funzione Pa_OpenStream() e sarà poi PortAudio a passarlo alla funzione di Call-Back.

```
typedef struct
{
    float frequenza;
    float fase;
    float ampiezza;
} myWaveData;

int main(void)
{
    PaStreamParameters parametriIngresso, parametriUscita;
    PaStream *stream;
    PaError err;
    myWaveData data;
    data.frequenza = 440;
    data.fase = 0;
    data.ampiezza = 1;
    …
    …

    err = Pa_OpenStream( &stream, &parametriIngresso, &parametriUscita, SAMPLE_RATE, FREAMES_PER_BUFFER,
                                                                        0, My_Callback, &data);

    err = Pa_StartStream(stream);
    …
    …
    return 0;
}
```

Esempio di generatore di forma d'onda:

Struttura Dati che memorizza lo stato dell'oscillatore:

```
typedef struct
{
  float left_phase;
  float right_phase;
}
myOscillatorStatus;
```

Funzione di Call-Back come generatore d'onda Sawtooth:

```
static int patestCallback( …, …, …, …, …, void *userData )
{
   myOscillatorStatus *status = (myOscillatorStatus*)userData;
   …
   unsigned int i;
   for( i=0; i<framesPerBuffer; i++ )
   {
     *out++ = status->left_phase;  /* left */
     *out++ = status->right_phase;  /* right */

     data->left_phase += 0.01f;
     if( data->left_phase >= 1.0f )
       data->left_phase -= 2.0f;
     data->right_phase += 0.03f;
     if( data->right_phase >= 1.0f )
       data->right_phase -= 2.0f;
   }
return paContinue;
}
```

Codice per la generazione dell'onda.
In questo caso la frequenza di oscillazione è specificata dagli incrementi:

- 0.01f sul canale sinistro
- 0.03f sul canale destro.

La compilazione di PortAudio

Come per tutte le librerie C, per utilizzare PortAudio sono necessari due file:

- Il file di intestazione: "*portaudio.h*"
- Il file di libreria entrambi presenti nel sistema:
  - "*portaudio_x86.lib*"
  - "*portaudio_x86.dll*"

In alternativa è possibile inserire nel proprio progetto direttamente i sorgenti di portaudio.
Il consiglio è di compilare una volta per tutte la libreria ed usare i file *.lib* e *.dll*.

E' possibile generare i file di libreria utilizzando un qualsiasi compilatore C e una qualsiasi IDE ma per motivi pratici, soprattutto per evitare complicazioni, il consiglio è di utilizzare "*Visual C++ 2008 express",* reperibile gratuitamente sul sito Microsoft all'indirizzo:

http://www.microsoft.com/downloads/it-it/details.aspx?FamilyID=94de806b-e1a1-4282-abc5-1f7347782553&DisplayLang=it

Nota: Il Visual C++ 2010 non riesce a convertire il file di progetto, si consiglia quindi l'uso dell'edizione 2008

La compilazione di PortAudio

La procedura da seguire è la seguente:

1) Scaricare il pacchetto portaudio:
   http://www.portaudio.com/download.html
   Scegliendo il file:
   pa_stable_v19_20110326.tgz
   Cioè la versione V.19 del 26 marzo 2011

2) Installare la DirectX SDK, ottenibile gratuitamente dal sito:

   http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=3021d52b-514e-41d3-ad02-438a3ba730ba

   Una volta installata Visual C++ dovrebbe individuare i file in automatico, in caso contrario copiare manualmente i due file "*dsound.h*" e "*dsconf.h*" nella cartella "include" di portaudio.
   I file si trovano nella cartella di installazione della DirectX SDK.
   Qualora Visual C++ avesse difficoltà a trovare file di tipo .lib è necessario impostare il linker affinché prenda in carico la cartella contenente le librerie.

La compilazione di PortAudio

3) Per il supporto ASIO e necessario scaricare preventivamente l'ASIO SDK:

   http://www.steinberg.net/en/company/developer.html

   e copiare l'intera cartella ASIOSDK2 nella cartella di portaudio /src/hostapi/asio/, rinominandola ASIOSDK.

4) Infine, nella cartella /build/msvc/, all'interno della cartella di portaudio, è possibile trovare un file di progetto di Visual C++.
   Il file va naturalmente aperto con Visual C++ e compilato per ottenere i file di libreria di cui precedentemente discusso.

Nella creazione di un proprio progetto i file "portaudio_x86.lib" e "portaudio.h" possono essere copiati all'interno di un'unica cartella in modo da impostare il linker in modo che vada lì a cercare le risorse.

Es2-Memorizzare uno stream su file in formato raw:

PortAudio non ha funzioni utili alla scrittura di stream audio su disco o alla riproduzione di stream audio prelevati da file su disco, è possibile utilizzare le funzioni base di I/O messere a disposizione dalle librerie standard del C.

In particolare le primitive per la scrittura di dati binari su file sono disponibili attraverso il file “stdio.h”, già disponibile a corredo di qualsiasi compilatore C.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define  SEC_TO_WRITE   5
#define  SAMPLE_RATE    44100
#define  FREQUENZA       440

int main(int argc, char *argv[])
{
  float* vettore;
  vettore = calloc(SAMPLE_RATE * SEC_TO_WRITE, sizeof(float));

  int counter;
  for(counter = 0; counter < (SAMPLE_RATE * SEC_TO_WRITE); counter++)
                vettore[counter] = sin(2*M_PI*FREQUENZA/SAMPLE_RATE*counter);

  FILE *rawfile;
  rawfile = fopen("raw-file.raw","wb");
  fwrite(vettore,sizeof(float),SAMPLE_RATE * SEC_TO_WRITE,rawfile);
  fclose(rawfile);

  return 0;
}
```

Memorizzare uno stream su file in formato raw:

Qualche commento:

```
float *vettore;
vettore = calloc(SAMPLE_RATE * SEC_TO_WRITE, sizeof(float));
```

Questo metodo di dichiarare gli array è peculiare del C; è possibile, qualora si utilizzi un compilatore per C++, ottenere lo stesso risultato con la seguente sintassi:

```
float vettore[] = new float[SAMPLE_RATE * SEC_TO_WRITE];
```

Per generare una forma d'onda a frequenza prestabilita c'è da tener conto del fatto che la variabile tempo è data da un contatore incrementale e tenendo in considerazione la frequenza di campionamento, secondo la formula:

$$s(n) = A \cdot \sin\left(2\pi \cdot \frac{freq}{SampleRate} \cdot n + \varphi\right)$$

Che si traduce in codice come:

```
for(int counter = 0; counter < (SAMPLE_RATE * SEC_TO_WRITE); counter++)
    vettore[counter] = sin(2*M_PI*FREQUENZA/SAMPLE_RATE*counter);
```

Memorizzare uno stream su file in formato raw:

Infine la parte relativa alla creazione del file ed alla scrittura dei dati:

```
FILE *rawfile;
rawfile = fopen("raw-file.raw","wb");
```

Per dichiarare la variabile di riferimento al file e per aprire il file “raw-file-raw” in scrittura in modalità “binary” (“wb”).

```
fwrite(vettore,sizeof(float),SAMPLE_RATE * SEC_TO_WRITE,rawfile);
```

Per scrivere sul file “*rawfile*” un numero di elementi pari a “*SAMPLE_RATE * SEC_TO_WRITE*”, ciascuno di dimensione “*sizeof(float)*”, contenuti nell’array “*vettore*”.
Ed infine:

```
fclose(rawfile);
```

Per chiudere il file.

Memorizzare uno stream su file in formato .wav:

La scrittura di file wav su disco è equivalente a quella già vista per il formato RAW con la differenza che i file WAV sono strutturati, contengono cioè informazioni aggiuntive sul "payload".
E' possibile prendere a riferimento le specifiche dei file WAV e scrivere delle funzioni che rispettino tali specifiche ma è conveniente utilizzare librerie già pronte, come ad esempio libsndfile, reperibile all'indirizzo "http://www.mega-nerd.com/libsndfile/".

Il consiglio è scaricare uno dei due file eseguibili di installazione, quello per i sistemi windows a 32 bit oppure quello per i sitemi windows a 64 bit.

L'installatore creerà una cartella: "*\Mega-Nerd\libsndfile*" all'interno della cartella "*C:\Programmi*" (oppure *Programmi(x86)* nel caso di sistemi a 64 bit).
Qui è possibile trovare:

- il file "*sndfile.h*" nella cartella "*include*"
- Il file di libreria statica "*libsndfile-1.lib*" nella cartella "*lib*"

I due file vanno inseriti all'interno del progetto rispettivamente come file **header** e come file di **libreria**.

Per chi volesse utilizzare librerie con link dinamico è possibile trovare il file "*libsndfile-1.dll*" all'interno della cartella "*bin*".

Esempio3: Registrare un file su disco.

Il programma di questo esempio ha lo scopo di scrivere su disco un file contenente un segnale ottenuto come somma tra l'ingresso della scheda audio e un segnale di tipo "sawtooth" generato nel programma.

```
#include "portaudio.h"
#include "stdio.h"
#include "sndfile.h"

#define  SAMPLE_COUNT     (SAMPLE_RATE * SEC_TO_REC)
#define  NUM_CANALI       2
#define SEC_TO_REC        4
#define SAMPLE_RATE       44100
#define PA_SAMPLE_TYPE    paFloat32
#define FRAMES_PER_BUFFER 256

typedef float SAMPLE;

typedef struct
{
   float phaseLeft;
   float phaseRight;
   float* frameBuffer;
   long indice;
}
myDataStruct;
```

Esempio3: Registrare un file su disco.

```
Static int My_Callback(const void *inputBuffer, void *outputBuffer, unsigned long framesPerBuffer,
                       const PaStreamCallbackTimeInfo* timeInfo, paStreamCallbackFlags statusFlags,
                       void* userData)
{
   SAMPLE *out = (SAMPLE*)outputBuffer;
   const SAMPLE *in = (const SAMPLE*)inputBuffer;
   unsigned int count;
   (void) timeInfo;
   (void) statusFlags;
   myDataStruct* status = (myDataStruct*)userData;
   SAMPLE local;

   if (inputBuffer == NULL)
      for (count = 0; count < framesPerBuffer; count++)
      {
         *out++ = 0;
         status->frameBuffer[indice++] = 0;
         *out++ = 0;
         status->frameBuffer[indice++] = 0;
      }
   else
      for( count=0; count<framesPerBuffer; count++ )
      {
         local = status->phaseLeft + *in++;
         *out++ = local;
         if(status->indice < SAMPLE_COUNT) status->frameBuffer[indice++] = local;
         local = status->phaseRight + *in++;
         *out++ = local;
         if(status->indice < SAMPLE_COUNT) status->frameBuffer[indice++] = local;
         status->phaseLeft += 0.01f;
         if(status->phaseLeft >= 1.0f )   status->phaseLeft -= 2.0f;
         status->phaseRight += 0.03f;
         if(status->phaseRight >= 1.0f )  status->phaseRight -= 2.0f;
      }

return paContinue;
}
```

Esempio3: Registrare un file su disco.

```
int main(void)
{
    PaStreamParameters parametriIngresso, parametriUscita;
    PaStream *stream;
    PaError err;

    SNDFILE *file;
    SF_INFO sfinfo;

    myDataStruct myData;
    myData.frameBuffer = calloc(NUM_CANALI * SAMPLE_COUNT,sizeof(float));
    myData.indice = 0;

    sfinfo.samplerate     = SAMPLE_RATE ;
    sfinfo.frames                       = SAMPLE_COUNT ;
    sfinfo.channels                     = NUM_CANALI ;
    sfinfo.format                       = (SF_FORMAT_WAV | SF_FORMAT_FLOAT) ;

    err = Pa_Initialize();
    if (err != paNoError) goto Error;

    parametriIngresso.device == Pa_GetDefaultInputDevice();
    if (parametriIngresso.device == paNoDevice) goto Error;
    parametriIngresso.channelCount = 2;
    parametriIngresso.sampleFormat = PA_SAMPLE_TYPE;
    parametriIngresso.suggestedLatency = Pa_GetDeviceInfo(parametriIngresso.device)->defaultLowInputLatency;
    parametriIngresso.hostApiSpecificStreamInfo = NULL;

    parametriUscita.device == Pa_GetDefaultOutputDevice();
    if (parametriUscita.device == paNoDevice) goto Error;
    parametriUscita.channelCount = 2;
    parametriUscita.sampleFormat = PA_SAMPLE_TYPE;
    parametriUscita.suggestedLatency = Pa_GetDeviceInfo(parametriUscita.device)->defaultLowInputLatency;
    parametriUscita.hostApiSpecificStreamInfo = NULL;
```

Esempio3: Registrare un file su disco.

```
    file = sf_open("output.wav", SFM_WRITE, &sfinfo);

    err = Pa_OpenStream( &stream, &parametriIngresso, &parametriUscita, SAMPLE_RATE, FREAMES_PER_BUFFER,
                                                                                    0, My_Callback, NULL);
    err = Pa_StartStream(stream);
    printf("Premi un tasto qualsiasi per interrompere lo streaming.");
    getchar();
    err = Pa_CloseStream(stream);

    sf_write_float(file, myData.frameBuffer, SAMPLE_COUNT * NUM_CANALI);
    sf_close (file);

    Pa_Terminate();
    return 0;

Error:
    Pa_Terminate();
    return -1;
}
```

Informazioni aggiuntive.

Le API di PortAudio sono ricche di funzioni per la diagnosi e gestione del sottosistema audio, per la sincronizzazione e altro.
Si consiglia di fare riferimento alla documentazione, reperibile all'indirizzo:

http://portaudio.com/docs/v19-doxydocs/api_overview.html

Un'altra importantissima fonte sono gli esempi presenti nella directory portaudio, nella cartella *test.*