UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di laurea triennale in
Scienze e Tecnologie della Comunicazione Musicale

I livelli di astrazione audio nei moderni sistemi operativi

Relatore:
Prof. Carlo Bellettini

Correlatore:
Dott. Mattia Monga

Tesi di laurea di:
Carlo Alberto Boni
matr. 692131

Anno Accademico 2007-2008

“The nice thing about standards is that you have so many to choose from.”

Andrew S. Tanenbaum

“*Science is what we understand well enough to explain to a computer. Art is everything else we do.”*

Donald Knuth

Indice

0. Introduzione

L'arrivo dei personal computer negli anni ottanta ha rivoluzionato il modo di concepire l'informatica, trasformandola in uno strumento della vita di tutti i giorni per centinaia di milioni di persone nel mondo. Lo sviluppo di applicativi per l'elaborazione del segnale audio richiede molta memoria e capacità di calcolo molto rapide per gestire eventi in tempo reale.

La realizzazione di applicazioni audio su personal computer economici è perciò relativamente recente e solo nell'ultimo decennio hanno iniziato a diffondersi interfacce programmative standard per la realizzazione delle applicazioni musicali.

Nel 1983[1] esistevano pochi produttori di hardware audio, questo non era supportato nativamente dai sistemi operativi e non c'era uno standard dei formati di file audio. I convertitori A/D-D/A erano delle periferiche da laboratorio molto costose e non un prodotto consumer. In aggiunta a questo i computer non erano multi-tasking.

Nel 1984 la Apple ha iniziato una strada di ricerca con una particolare attenzione all'usabilità e il suono cominciò ad apparire nei primi software consumer.

Windows nella sua versione 1.0 prevedeva dei semplicissimi suoni, questo nel 1985.

Dai laboratori Creative a Singapore alla fine degli anni '80 nasce la periferica audio

[1] I cenni storici dell'introduzione sono tratti da [93] e da [87].

SoundBlaster, che si impose come standard sul mercato[2] con i suoi convertitori da 8 bit a 23 kHz[3].

SoundBlaster era compatibile con la leader di mercato *AdLib* card e includeva un sintetizzatore FM a 11 voci attraverso l'uso del chip Yamaha YM3812 (conosciuto anche come OPL2).

Questa periferica includeva anche un DSP[4] che era in grado solamente di decomprimere un segnale ADPCM. La prima versione della SoundBlaster non era munita di un filtro anti-aliasing, con il risultato di un suono metallico che adesso considereremmo di scarsa qualità.

In questo ventennio che ci separa dalla nascita della SoundBlaster l'audio digitale ha preso sempre più prepotentemente spazio nella vita quotidiana: l'hardware specifico è diventato sempre più integrato anche nei dispositivi portatili e il software ha raffinato le proprie tecnologie in relazione al supporto hardware disponibile.

Attualmente anche le attrezzature professionali possono essere acquistate a prezzi relativamente bassi.

I sistemi operativi hanno introdotto un supporto audio sempre più integrato e con modalità molto diverse tra loro e nell'ambiente open-source si è aperto un dibattito sullo sviluppo dello stack audio di Linux[5], che ha avuto moltissime modifiche negli anni e che può variare molto in relazione alla distribuzione utilizzata e alla configurazione del sistema.

Al giorno d'oggi Linux cerca di offrire usabilità ed un supporto multimediale paragonabile a quello dei rivali proprietari Windows e Macintosh.

Gli obiettivi e le scelte effettuate

Questo elaborato partirà da un'analisi generale e platform-independent delle strutture di Input/Output audio per poi approfondire le specifiche architetture dei sistemi Linux, Mac Os X e Windows (in particolare la versione XP). Si è scelto di valutare dei sistemi operativi generici

[2] Lo standard viene infatti definito come “SoundBlaster Compatible”

[3] La frequenza di campionamento massima supportata era 23 kHz in riproduzione (D/A) e 12kHz in registrazione (A/D)

[4] Creative Labs per la prima volta usò il termina DSP per indicare Digital Sound Processor invece che Digital Signal Processor

[5] Per confronti e dibattiti tra le tecnologie Linux si rimanda a [19], [20, [24], [28], [29], [79].

e non specificamente orientati all'audio/multimedia o all'elaborazione del segnale perché questi sono i sistemi più comunemente diffusi e più utilizzati da utenti generici. Altri sistemi come quelli sviluppati per il multimedia (es. BeOs, UbuntuStudio,..), quelli realtime (LinuxRT, LynxOS, RMX, ITRON,..) o quelli per dispositivi mobili (es. Symbian, PalmOS, WindowsCE,..) non verranno trattati o vi saranno solamente dei cenni a loro riguardo.

L'obiettivo di questo elaborato è classificare le diverse tecnologie audio presenti nei sistemi operativi in oggetto secondo il modello, tipico dell'ingegneria del software, dei livelli di astrazione.

Nel proporre questa tesi abbiamo supposto che le diverse tecnologie si potessero posizionare, più o meno chiaramente, in uno stack applicativo che potesse nascondere ai livelli superiori le componenti più di basso livello.

Lo stack di livelli di astrazione assunto a prototipo è quello generico dell'I/O, che verrà analizzato nel primo capitolo.

Storicamente la piattaforma Macintosh ha riscosso molto successo tra i musicisti ed i professionisti del settore musicale: uno degli obiettivi di questo lavoro è anche verificare se questa preferenza possa essere dovuta ad una effettiva superiorità tecnica.

Le fonti

La nostra ricerca si propone di indagare e confrontare le tecnologie utilizzate nei sistemi operativi moderni per la gestione dell'audio.

Il campo della programmazione per l'audio e per la musica è un settore molto specialistico e per questo poco documentato.

La nostra ricerca ha spesso incontrato difficoltà nell'approfondire gli argomenti perché gli unici riferimenti autorevoli possibili per molte tecnologie sono le corrispondenti Application Programming Interface (API).

Nello specifico abbiamo notato che: per Windows addirittura la documentazione ufficiale[6] risulta in certi casi lacunosa e link interni non funzionanti, la documentazione Apple risulta

[6] MSDN (MicroSoft Develoloper Network)

piuttosto esauriente per quanto riguarda il limitato numero di argomenti che tratta, mentre per Linux le fonti sono sempre distribuite su molti siti web diversi.

La parte sperimentale è consistita nella realizzazione di applicazioni prototipali che permettessero di confrontare le astrazioni fornite al programmatore dalle diverse piattaforme e ciò ha comportato la necessità di familiarizzarsi con i linguaggi C, C++, C#, Objective-C e Java.

La strumentazione

L'analisi dei sistemi operativi e lo sviluppo di applicativi di esempio sono stati realizzati su un calcolatore Apple MacbookPro con Triple Boot (Linux, MacOS, Windows). Gli ambienti di sviluppo utilizzati sono nel caso di Linux l'editor di testo Emacs più vari compilatori (gcc, javac,..), per Mac Os Xcode 3 e per Windows Visual Studio 2005.

Altre informazioni preliminari

È fondamentale tenere presente in un'analisi comparata dei sistemi operativi che Apple sviluppa con una target hardware molto più definito e preciso rispetto a Microsoft e al mondo Linux: essendo anche produttrice (o meglio assemblatrice) di hardware, Apple in linea di massima conosce tutte le periferiche presenti nei calcolatori sui quali girerà il proprio sistema operativo, per questo dovrebbe poter riuscire a creare un codice più ottimizzato e più performante.

Linux e Microsoft Windows hanno una politica opposta: il target deve essere il più ampio possibile e per questo si suppone che abbiano bisogno di un livello di astrazione hardware (HAL) molto più esteso.

1. Struttura generica dello stack audio platform-independent

Come modello per questa nostra analisi prendiamo in esame la struttura dello stack di I/O generico, che poi verrà specializzato nel caso audio.

Faremo riferimento ai diversi livelli come livelli di astrazione, perché si tratta di livelli logici che non necessariamente devono essere implementati in modo separato nel sistema operativo.

Ormai dagli anni '60 si utilizza una gestione delle periferiche di I/O di tipo asincrono, ovvero la CPU non deve aspettare o controllare ciclicamente lo stato delle periferiche, sprecando risorse di calcolo (busy-waiting), ma può continuare a gestire altri processi, guidata dallo scheduler, e saranno le periferiche stesse a provocare un'interruzione per richiamare a sé l'attenzione della CPU al completamento della loro operazione[7].

[7] La gestione dell'I/O e la la storia dei sistemi operativi sono approfonditi in [9], [11] e [12].

1.1 I livelli di astrazione

Come in [12] possiamo schematizzare i livelli generici di astrazione per la gestione dell'I/O come segue:

- User-level I/O software
- Device-independent operating system software
- Device Drivers
- Interrupt Handlers
- Hardware

1.1.1 L'hardware

Esso è costituito da una parte meccanica ed una parte elettronica. La seconda è generalmente denominata controller. Il controller è in grado di interfacciarsi con il calcolatore attraverso un'interfaccia standard (es. PCI, USB, IEEE1394, SCSI,..). Il compito del controller è quello di effettuare il controllo e la correzione degli errori, segnalare quelli non correggibili e convertire il flusso di dati in un formato compatibile con quello dell'interfaccia utilizzata.

1.1.2 Gestori delle interruzioni

Nell'architettura Intel IA32 le interruzioni hardware vengono inviate dalle periferiche all'interrupt controller (es. Intel 8259) che si occupa di mandare un segnale alla CPU. Il sistema operativo a questo punto avvia una routine, definita come interrupt handler che si occupa di gestire la specifica interruzione.

Generalmente per le interruzioni di I/O l'interrupt handler richiama il driver specifico della periferica.

1.1.3 I device driver

Sono processi in kernel mode[8] che vengono risvegliati solo quando la periferica deve eseguire delle operazioni. Essi conoscono e sanno come gestire le varie componenti e modalità delle periferiche.

[8] Nei sistemi operativi in esame i driver vengono eseguiti in kernel mode, tuttavia esistono dei sistemi che utilizzano driver in user mode.

Periferiche dello stesso tipo vengono spesso riunite in classi per limitare il proliferare di driver e per semplificare l'installazione e l'uso delle stesse. È il caso tipico dei mouse e delle tastiere, però anche nel caso audio, come vedremo nel capitolo 5, esiste uno standard Microsoft per uniformare l'interfaccia con il sistema operativo delle periferiche (Universal Audio Architecture). Per informazioni tecniche a riguardo si rimanda a [56] e a [90].

Spesso i device driver sono innestati: il driver specifico di una determinata periferica non è necessario che conosca esattamente il protocollo dell'interfaccia alla quale è connessa la periferica stessa perché è presente un driver sottostante che si occupa di gestire la connessione (PCI, USB, IEEE1394,...), di effettuare le conversioni ed i relativi controlli di correttezza necessari.

1.1.4 Device independent operating system software

Si tratta di un livello di astrazione molto complesso che permette ai programmatori di non dover conoscere le specifiche di ogni periferica per cui sviluppano il loro software, fornendo così anche un metodo semplice per permettere la portabilità dei software su hardware differenti.

Le funzioni principali definite in [12] di questo livello di indirettezza sono:

- Definire una modalità unica di interfaccia con le periferiche (o almeno una per ciascuna classe di periferiche).
- Definire un modello di interfaccia tra i driver ed il sistema operativo.
- Fornire dei buffer per la lettura e scrittura delle stesse.
- Fornire una dimensione fissa dei blocchi per tutti i *block device*.
- Segnalare eventuali errori.

1.1.5 User level software

Si tratta dei software applicativi oppure eventualmente dei server (o daemon in terminologia Unix) che offrono un'interfaccia più ad alto livello rispetto alle Kernel API.

1.2 L'Input/Output audio

Il caso audio non è particolarmente diverso dal caso di I/O generico, tuttavia sono presenti delle problematiche maggiori dovute al fatto che il processing audio spesso ha dei requisiti di temporizzazione molto stringenti, per cui l'efficienza diventa ancora più importante.

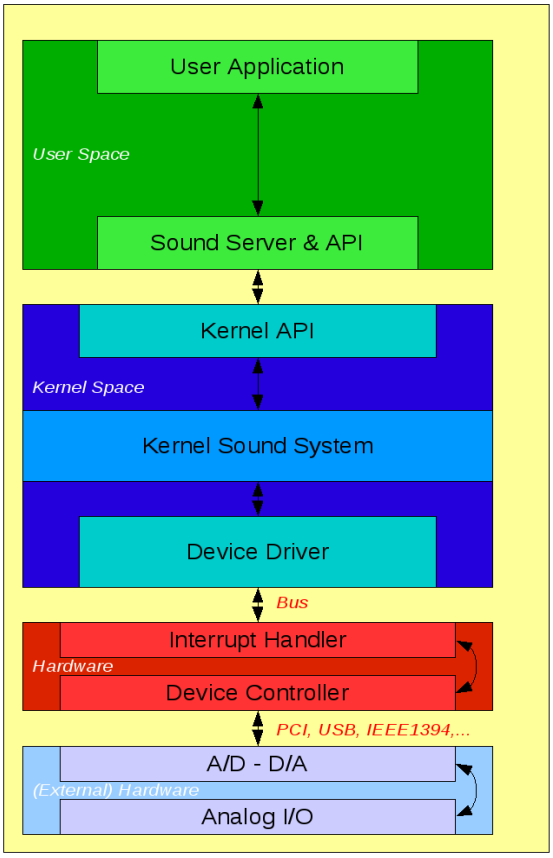

Fig. 1.1- Stack audio

Lo strato software che prima abbiamo definito come device independent operating system software nel caso audio può essere qualificato più propriamente *Kernel Sound System*; questo componente, come evidenziato dalla figura 1.1, si trova tra i device driver e le kernel API e svolge diverse operazioni, tra cui quelle di conversione di formato tra le applicazioni e l'hardware e la gestione dei buffer.

Inoltre i software a livello utente spesso forniscono a loro volta delle librerie per la

costruzione di plug-in che possano fornire sintetizzatori software ed effetti da inserire nella catena audio.

1.3 Modello tradizionale di riproduzione audio

Storicamente la riproduzione audio è realizzata nei sistemi operativi attraverso le interruzioni prodotte delle periferiche audio.

[10] e [79] mostrano che quando un'applicazione apre un device audio per la riproduzione, lo configura per un buffer di dimensione fissa. Successivamente l'applicazione riempie il buffer con dati PCM[9] e quando ha finito comunica all'hardware che è possibile iniziare a riprodurre. L'hardware poi legge i campioni uno per volta e li passa al convertitore D/A[10].

Dopo un certo numero di campioni riprodotti la periferica alza un'interruzione che viene inoltrata all'applicazione[11]. Quando quest'ultima si risveglia scrive nuovamente nel buffer, ricominciando così il ciclo. Quando l'hardware raggiunge la fine del buffer riprende ad utilizzarlo dall'inizio in quanto si tratta di un buffer circolare.

Il numero di campioni dopo i quali viene generata un'interruzione è normalmente chiamato frammento (*fragment*). Il numero di frammenti in cui è diviso il buffer è solitamente un numero intero, in particolare per ovvie ragioni di calcolo è solitamente una potenza di 2.

Il numero di frammenti possibili è dipendente dall'hardware utilizzato. Nell'hardware attualmente in commercio il minimo è sempre e comunque 2 frammenti [10].

---

[9] Pulse Code Modulation è uno dei formati di codifica più di ffusi per l'audio non compresso. Consiste nella rappresentazione dell'onda sonora campione per campione.

[10] Il convertitore D/A si occupa di trasformare il segnale digitale in analogico perché questo possa essere inviato a degli attuatori. D/A è un acronimo per Digital to Analog.

[11] Nei sistemi Unix, e secondo lo standard POSIX, questo avviene tramite poll()/select() : chiamate di sistema che mettono in attesa il processo fino a quando il device di I/O non è pronto.

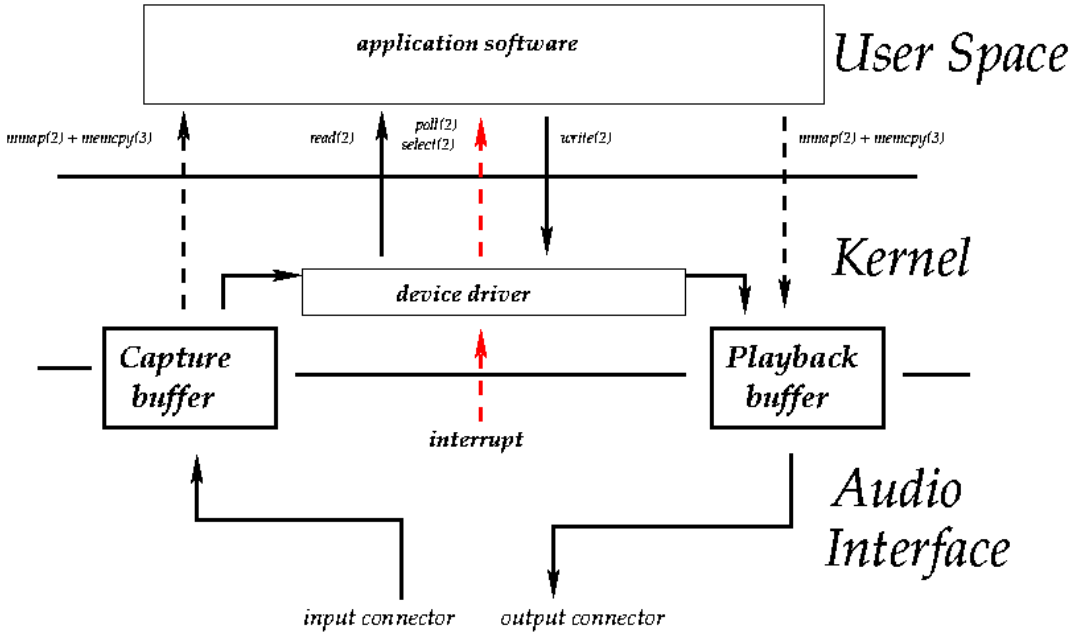

Fig. 1.2 - Modello di interfaccia audio

1.4 I Buffer

Come visto in precedenza le periferiche audio hanno un buffer hardware. Esiste normalmente anche un buffer applicativo che risiede nella memoria centrale.

Nel caso di acquisizione (input audio) quando il buffer della periferica viene sufficientemente riempito esso genera un'interruzione e il driver audio si occupa di trasferirlo via DMA (Direct Memory Access) nel buffer applicativo.

All'opposto, per la riproduzione, al riempirsi del buffer dell'applicazione il driver sarà in grado, sempre via DMA, di trasferirlo nel buffer hardware della periferica.

Entrambi i buffer sono di tipo circolare[12]: vengono mantenuti dei puntatori per tenere traccia delle posizioni correnti in entrambi i buffer.

Una delle sostanziali differenze tra le varie API audio è la possibilità di accedere alle risorse hardware tramite delle chiamate di sistema. Al di fuori del kernel è visibile solamente il buffer applicativo, però determinati Sound System possono fornire delle API per modificare alcuni parametri o scrivere direttamente nel buffer hardware (ad esempio DirectSound o Full ALSA , vedi cap. 2 e 4).

I programmatori che sviluppano i kernel sound system hanno il compito di ottimizzare l'uso del buffer hardware per le diverse situazioni applicative.

La dimensione del buffer e dei fragment può condizionare notevolmente le prestazioni dell'applicazione perché un buffer molto grande impiega molto tempo ad essere letto/scritto completamente. Si determina una latenza[13] inversamente proporzionale alla dimensione del buffer.

Quando un device audio è attivo, vengono continuamente scambiati dati dal buffer applicativo a quello hardware e viceversa. In caso di data-capture (registrazione), se l'applicazione non legge il buffer abbastanza velocemente questo verrà sovrascritto in quanto circolare. La perdita di dati di questo tipo è definita *overrun*. Il caso opposto è quando, durante la riproduzione, l'applicazione non fornisce dati in modo sufficientemente veloce al buffer, che si svuota gradualmente e arriva in una situazione definita come starvation: questo caso è il buffer *underrun.*

Per altri tipi di I/O il buffer underrun potrebbe non avere nessun effetto visibile, mentre invece nel caso audio, che è fortemente dipendente dal tempo, determina un degrado della qualità audio con dei click, dei glitch o altri fenomeni audio non graditi.

[12] Si tratta di un buffer la cui lettura/scrittura avviene sempre in una direzione. Arrivati alla fine del buffer la lettura/scrittura riprende dall'inizio.

[13] Illustrata nel paragrafo successivo.

1.5 La latenza

Con il termine *latenza* si indica il tempo che intercorre tra la digitalizzazione del segnale e la sua successiva riconversione in analogico[14]. È definita in [91]:

> *"Audio latency is defined as the minimum time required for a computer to store a sample from an audio interface into application memory and copy the same sample from application memory to the audio interface output. Both the conversion from analog to digital and back to analog are included."*

Si tratta di un fattore fondamentale per la qualità di un sistema audio digitale, in quanto una latenza superiore ai 30ms risulta assai fastidiosa e può rendere impossibile il lavoro a musicisti e tecnici del suono.

Le moderne Digital Audio Workstation (DAW) arrivano ad ottenere livelli di latenza inferiori a 5ms.

Sono molte le cause che determinano l'aumento della latenza in un elaboratore di tipo general purpose e quando questo vuole diventare uno strumento professionale è necessario poter mantenere più basso possibile questo valore e soprattutto fare in modo che esso sia sostanzialmente costante.

La latenza può essere così classificata:

- **Latenza in ingresso di un sistema:** deriva dal fatto che i dati che arrivano all'interfaccia audio vengono messi in un buffer, questo si riempie e oltre un certo valore effettua una Interrupt Request (IRQ), a questo punto la CPU caricherà il driver.
- **Latenza in uscita di un sistema:** quando i dati sono stati preparati dall'applicazione il driver si blocca in attesa che la periferica alzi un'interruzione per indicare che è pronta a ricevere nuovi dati.

  Quando arriva un IRQ la CPU potrà consegnare i dati perché vengano riprodotti.

  Per questo il ciclo di lettura/scrittura del buffer è almeno diviso in due quanti di tempo

[14] Si rimanda a [75], [84], [91], [92] e all'appendice per maggiori dettagli sulla latenza dei sistemi audio.

distinti: uno per la scrittura da parte dell'applicazione e uno per la lettura hardware.

Sia nel caso in ingresso sia in quello in uscita non sono da sottovalutare i possibili problemi di under/overrun che un'eccessiva latenza può causare.

Si presentano anche altre situazioni che possono aumentare la latenza di un sistema tra cui è di notevole rilievo il problema causato dalla grafica delle applicazioni: in relazione al sistema operativo e al linguaggio di programmazione utilizzato sono disponibili diverse librerie grafiche tra cui ricordiamo quelle di sistema (Win32, Quartz), e quelle applicative (GTK+, Qt, FLTK, OpenGL,...).

Le applicazioni audio spesso utilizzano librerie diverse da quelle di sistema per avere una grafica più accattivante, per esempio per dei plug-in con un design in stile analogico. Questo determina il fatto che il sistema debba caricare in memoria molti più dati e può causare più context-switch[15] per la gestione delle finestre o più passaggi in kernel-mode per gestire le interruzioni.

Un sistema adatto per gestire audio in real-time deve contenere il più possibile il numero di context-switch perché, oltre al tempo necessario a salvare e ripristinare i registri, questo comporta nei moderni sistemi operativi che supportano la memoria virtuale[16] il fatto che venga invalidato il Translation Lookaside Buffer (TLB)[17]. In sostanza il continuo context-switch rende inutile l'uso di sistemi di caching della traduzione degli indirizzi logici in indirizzi fisici e causa così un ulteriore latenza perché diventa poi necessario consultare più spesso la Page

[15] Con questo termine si intende il cambiamento di contesto entro cui opera il processore. Il context-switch è necessario in un sistema multitasking. Il context-switch generalmente causa il salvataggio dei registri principali del processore, tra cui, in un'architettura IA32: l'Instruction Pointer (EIP), lo stack pointer (ESP), i descrittori di segmento (SS, CS; DS) e i registri EFLAGS. Per maggiori dettagli si rimanda a [11] e [12].

[16] La maggior parte dei sistemi operativi commerciali di tipo general-purpose.

[17] TLB è una memoria cache che permette la memorizzazione delle traduzioni degli indirizzi logici nei corrispondenti indirizzi fisici. Il Translation Lookaside Buffer normalmente si trova nella MMU (Memory Management Unit) ed è composto al più da 64 traduzioni. Rende possibile un incremento delle prestazioni del sistema in quanto permette di diminuire l'accesso alla Page Table (che risiede in memoria centrale, molto più lenta di una memoria cache).

Table[18].

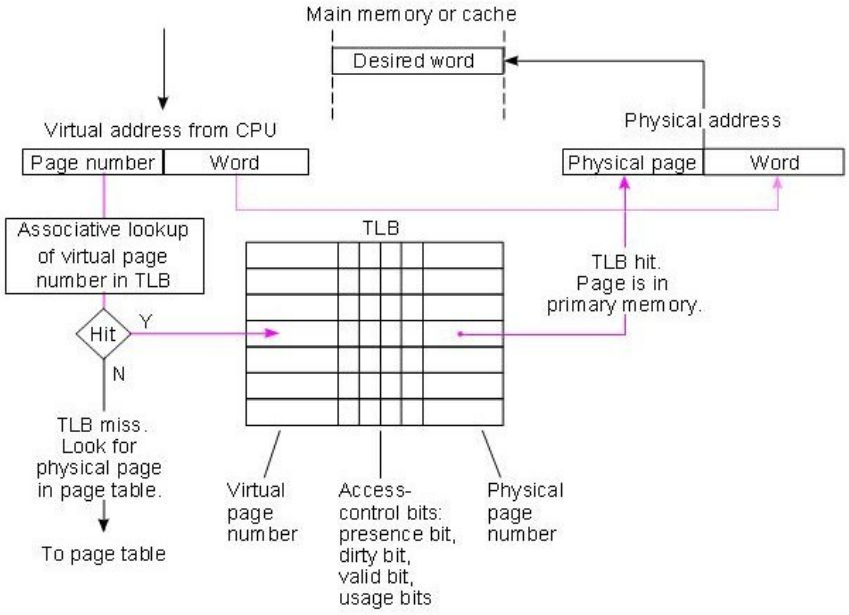

Fig. 1.3 - Translation Lookaside Buffer

Come affermato in [75] il costo del cambio di contesto dal punti di vista dei registri (save/restore) è lineare con il clock della CPU mentre gli effetti della cache/TLB dipendono da molti più fattori (spesso nei personal computer non professionali la componentistica è di scarsa qualità), tra cui la velocità del bus sulla scheda madre, la velocità della memoria centrale, la dimensione della cache.

Nei processori moderni della famiglia x86 la velocità di context-switch per i registri è nell'ordine dei 10-50 microsecondi: gli effetti dovuti al TLB possono duplicare o addirittura quadruplicare questo tempo, e questo può creare dei problemi in quanto il tempo disponibile per effettuare processing audio real time è molto ridotto: 1333 microsecondi sono il tempo necessario per processare 64 frame a 48kHz.

[18] Nei sistemi operativi che utilizzano la paginazione la Page Table contiene tutte le traduzioni necessarie degli indirizzi delle pagine di memoria e delle informazioni riguardanti la presenza delle specifiche pagine nella memoria centrale piuttosto che in un supporto di memoria di massa (disco di swap).

1.6 Ottimizzazione dei buffer

In relazione ai requisiti dell'applicazione verrà scelta la dimensione del buffer di riproduzione (in realtà dei frammenti, in quanto il buffer hardware ha una dimensione fissa in relazione alla periferica audio in uso). Può essere impostato a 4ms per una applicazione a bassa latenza, come un sintetizzatore, oppure anche a 2s per applicazioni per le quali non è fondamentale una risposta in tempo reale, come un generico player audio.

Formalizzando come in [10]:
BUF_SIZE → dimensione del buffer hardware in campioni
FRAG_SIZE → dimensione di un frammento in campioni
NFRAGS → numero di frammenti che compongono il buffer
RATE → frequenza di campionamento

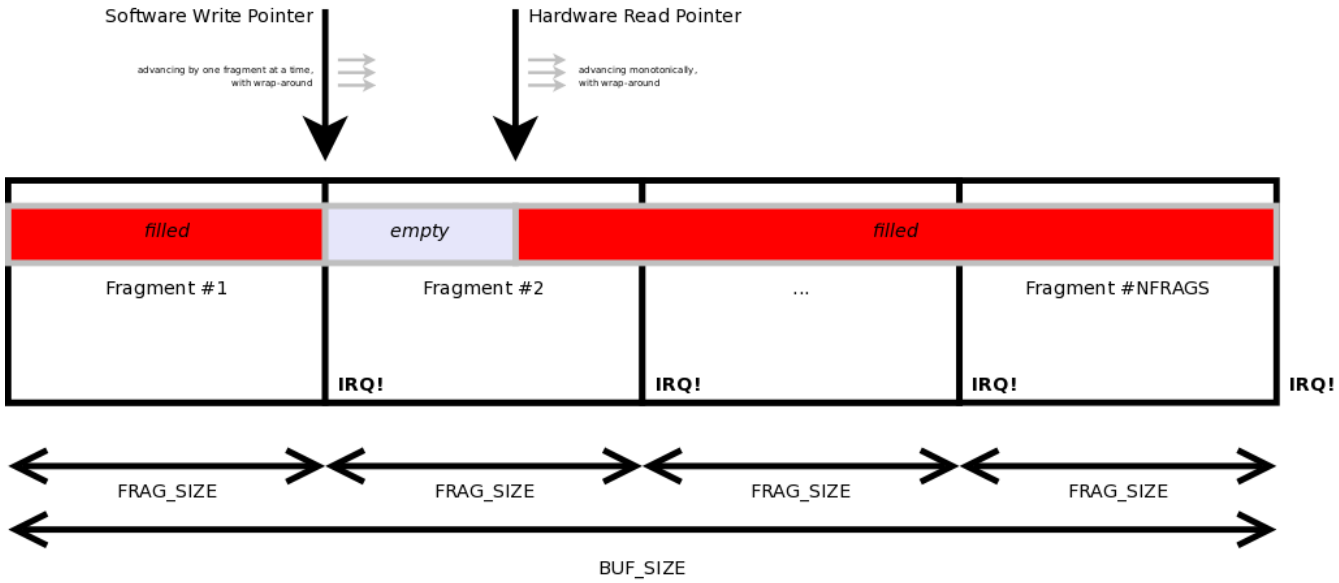

Fig. 1.4 - Buffer e frammenti

avremo la latenza complessiva pari a BUF_SIZE / RATE
un'interruzione viene alzata ogni FRAG_SIZE / RATE

se l'applicazione non perde nessuna interruzione ha a disposizione un tempo per adempiere le

richieste pari a

(NFRAGS – 1) * FRAG_SIZE / RATE

Se l'applicazione necessita di più tempo avremo un buffer underrun.

Il livello di riempimento del buffer dovrebbe rimanere tra BUF_SIZE e (NFRAGS – 1) * FRAG_SIZE.

In caso di interruzioni perse potrebbe anche scendere ulteriormente, nel peggiore dei casi fino a 0, che sarebbe di nuovo un buffer underrun.

La dimensione del buffer dovrebbe essere :

- Il più grande possibile per minimizzare il rischio di buffer overrun.
- Il più piccolo possibile per garantire basse latenze

La dimensione dei frammenti dovrebbe essere:

- Il più grande possibile per minimizzare il numero di interruzioni e conseguentemente minimizzare i context-switch, l'uso della CPU e minimizzare i consumi di energia elettrica.
- Il più piccolo possibile per dare all'applicazione il più tempo possibile per riempire il buffer di riproduzione.

È evidente che non esiste una soluzione ottimale in assoluto: ogni sistema ha dato maggiore importanza ad alcuni parametri rispetto ad altri.

I problemi del tradizionale modello di riproduzione sono i seguenti:

- Le specifiche dei buffer sono strettamente dipendenti dall'hardware, per cui un software che vuole perseguire degli obiettivi di portabilità dovrà tenere conto che la maggior parte delle periferiche hardware può fornire un numero limitato di dimensioni dei buffer e dei frammenti.
- Le caratteristiche del buffer possono essere specificate solo all'apertura del device.

  Questo può creare dei problemi quando esistono diverse applicazioni che concorrentemente vogliono mandare in output dell'audio e queste hanno requisiti di latenza differenti.
- Non è praticamente possibile scegliere bene le esatte caratteristiche di cui la nostra applicazione ha bisogno, perché non ci è dato sapere nulla riguardo alle interruzioni che

potranno avvenire e allo scheduling della CPU[19].

- Poiché il numero di frammenti è intero ed è pari ad almeno 2 su tutto l'hardware esistente, genereremo almeno 2 interruzioni per ogni iterazione del buffer. Se impostiamo la dimensione del buffer a 2s avremo un'interruzione al secondo e successivamente 1 secondo per riempire di nuovo il buffer.

1.7 I Sound Server

Un sound server è un applicativo nato per gestire i device audio in modo più complesso e più ad alto livello rispetto ad un Kernel Sound System: esso è situato in user-space e vuole spesso nascondere completamente l'interfaccia dei livelli sottostanti.

I sound server sono nati principalmente per sopperire alle mancanze dell'environment in kernel space in ambiente Linux. Tuttavia alcuni sound server professionali, come Jack, sono stati implementati anche per altre piattaforme[20].

1.7.1 L'allocazione delle risorse

I primi sistemi di gestione dell'audio permettevano che solo un'applicazione alla volta utilizzasse il device audio, impedendo a tutte le altre di emettere suoni. Questo poteva addirittura bloccare alcune applicazioni in attesa che si liberasse il device audio. È il caso di Windows fino alla versione 95 compresa e di Linux con OSS3.

[19] Si considera il caso comune di sistemi operativi non specifici per il tempo reale.

[20] Nello specifico Jack esiste per Linux e per Mac Os X.
Pulse Audio, pur non essendo di taglio professionale, ha sviluppato libsydney per essere cross-platform su Linux, MacOs e Windows, interfacciandosi rispettivamente con ALSA, CoreAudio e DirectSound.

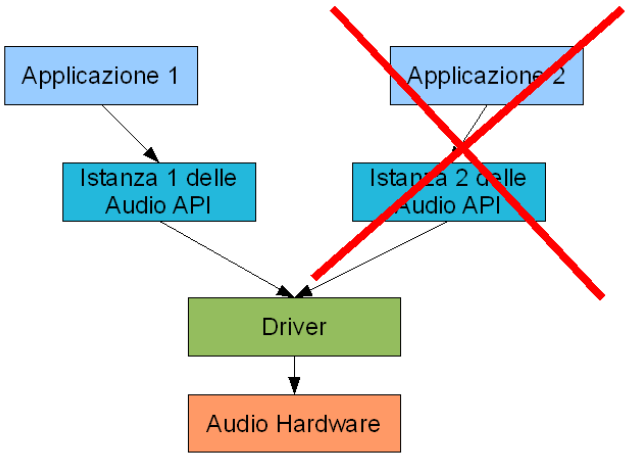

Fig 1.5 – Allocazione esclusiva delle risorse audio da parte di un'applicazione.

Successivamente è stato inserito un livello di indirettezza per fare in modo che l'allocazione del device audio non venisse fatta direttamente dall'applicazione utente, bensì da questo servizio di sistema. Quest'ultimo è stato realizzato con modalità differenti ed in tempi diversi nei diversi sistemi operativi: a livello utente oppure come servizio all'interno del kernel.

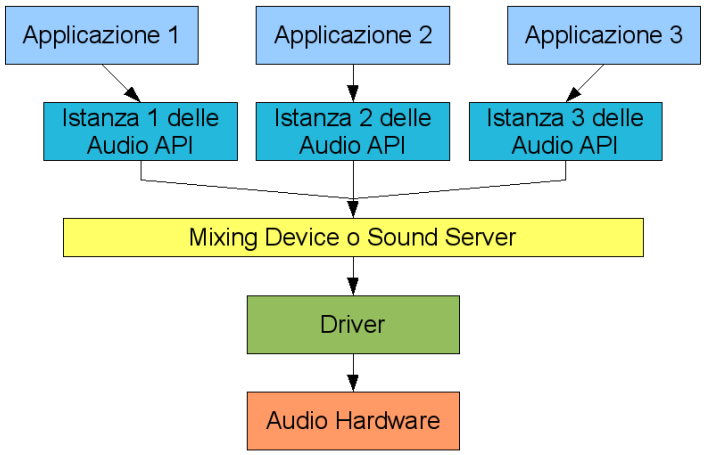

Fig 1.6 – Condivisione delle risorse audio tramite un sistema di indirettezza.

1.7.2 I sound server e il mixaggio

Quando due o più applicativi richiedono l'uso dell'audio contemporaneamente il sistema deve essere in grado di gestire e soddisfare queste richieste.

Uno stack audio molto semplice può non prevedere la possibilità di più flussi audio contemporanei.

Per avere una visione più completa è necessario anche tenere in considerazione che esistono periferiche audio stereo (la maggior parte di quelle consumer) ma anche schede audio surround (es. 5.1) o multicanale (generalmente professionali).

Per le periferiche multicanale la soluzione è spesso semplice ed efficiente: il driver è in grado di gestire più flussi audio contemporaneamente.

Per le periferiche stereo è necessario effettuare un mixaggio dei flussi audio prima di inviarli al driver.

Il mixaggio può essere gestito da un componente del kernel (es. in Windows fino a XP : il Kernel Mixer) oppure da dei “sound server”, applicativi in user-space che permettono il mixaggio e forniscono delle librerie di funzioni di livello medio-alto, quali ad esempio degli effetti sui canali audio o il routing (rewire) dei flussi audio tra le applicazioni.

Generalmente non si parla di sound server in ambiente Microsoft e Apple, anche se spesso troviamo delle componenti audio del sistema operativo a livello utente, in particolare il framework CoreAudio di OSX e la maggior parte dello stack di Windows Vista sono in user-space.

Esistono però dei sound server nati in ambiente Linux che sono stati esportati anche per gli altri sistemi e che offrono comunque un valore aggiunto allo stack nativo del sistema.

1.7.3 I sound server e l'IPC[21]

In uno scenario di editor multitraccia, sequencer, software synthesizer, drum machine, MIDI pattern sequencer nasce il problema di come interconnettere tutti questi software diversi.

Sistemi Operativi come Mac OS e Windows hanno scelto di implementare software monolitici che ospitino tutti questi componenti come moduli o plug-in.

Alternative sono il *ReWire*[22], creato dalla *Propellerheads* in collaborazione con la *Steinberg* nel 1998 per il sintetizzatore *ReBirth* e oggi supportato dalla maggior parte delle applicazioni commerciali, e *DirectConnect* (*Digidesign*). Queste tecnologie permettono di interconnettere i singoli canali audio di più applicazioni così come sarebbe possibile connettere fisicamente degli strumenti musicali a mixer, processori, effetti, etc.

Attualmente è importante sottolineare, come in [75], che solo i plugin che utilizzano API di livello più alto, attraverso oggetti condivisi ottemperano al loro compito servendosi di un unico thread, per cui il problema della condivisione della memoria e della comunicazione tra i processi è fondamentale per tutto il processing audio.

L'approccio Unix è quello “push”, che permette a tutti di scrivere quanto e quando vogliono: esempi di server che utilizzano questo modello sono *esd*, *artsd*, *NAS*. Questo metodo non permette nella maggior parte dei casi il routing tra le applicazioni e non mantiene una sincronia tra le stesse, in quanto non c'è controllo sullo scheduling che effettua il sistema operativo. Il risultato è soddisfacente solo per un mercato di tipo consumer e non professionale.

I sound server attuali offrono molte caratteristiche interessanti come l'utilizzo in rete di periferiche audio in modo completamente trasparente oppure offrono delle API di livello più alto rispetto a quelle di sistema.

[21] Inter Process Communication (IPC): insieme di tecniche per lo scambio di dati tra due o più thread o processi, ad esempio il message passing, la memoria condivisa, la sincronizzazione.

[22] Maggiori informazioni possono essere trovate in [86].

1.8 Il modello Glitch-Free

Si tratta di un modello adottato da parecchi anni da Apple CoreAudio, recentemente adottato da Microsoft WASAPI e quest'anno in ambiente Linux con l'ultima release di Pulse Audio.

La spiegazione di questo modello è stata possibile principalmente attraverso la documentazione di Pulse Audio[23], in quanto tecnologia open source: le affermazioni riportate si suppone possano essere condivise anche dalle tecnologie dei sistemi Windows e Macintosh, in quanto citati in [10] come antesignani del glitch-free rispetto a Pulse Audio e in quanto confermato da [37], [55], [65] e [69].

L'idea di base è quella di non dipendere dalle interruzioni delle periferiche audio per effettuare lo scheduling delle risorse audio ma di utilizzare dei timer di sistema, i quali sono molto più flessibili di quelli basati sui frammenti dei buffer hardware. Possono essere configurati a piacere ed essere indipendenti dalle politiche di buffering dei device.

La seconda idea fondante di questa tecnologia è quella di utilizzare dei buffer più grandi possibile: fino a 2 o 5 secondi.

Il terzo concetto fondamentale è quello di rendere possibile in qualunque momento la riscrittura del buffer hardware: questo rende possibile una reattività molto maggiore all'input dell'utente, nonostante la grandissima latenza imposta dal buffer hardware di riproduzione.

Le interruzioni audio vengono disabilitate oppure ridotte al minimo (attraverso la riduzione del numero di frammenti del buffer). Il sistema si occuperà di raccogliere eventuali richieste di specifiche latenze da parte delle applicazioni e, in ogni caso adotterà la minima latenza richiesta. Qualora non vi fossero richieste la latenza sarà pari alla dimensione del buffer.

[23] Si rimanda a [10], [79], [80], [81].

Viene poi configurato un timer per risvegliare l'applicazione N millisecondi prima (es. 10ms per Pulse Audio) che il buffer si svuoti completamente per riempirlo parzialmente e poi tornare in stato di attesa.

Se la latenza complessiva L deve essere inferiore ad N il timer verrà configurato per risvegliare l'applicazione ogni L/2.

Se il timer risulta essere troppo lungo avremo un *buffer underrun* al quale il sistema reagirà dimezzando il timer.

Con questa soluzione adattiva il buffer underrun potrà accadere una volta ma poi non accadrà più.

I benefici di questo modello sono molteplici:

- Minimizza le interruzioni, migliorando le prestazioni ed il consumo di energia
- Minimizza il rischio di *buffer overrun*
- Fornisce una soluzione rapida in caso di *buffer underrun*
- Fornisce *latenza zero* perché è sempre possibile scrivere nel buffer
- Migliora la portabilità in quanto è meno dipendente dalle specifiche dell'hardware
- È in grado di fornire quasi qualunque latenza a richiesta dell'applicazione con la possibilità di riconfigurarsi senza discontinuità nell'audio

Ci sono anche alcune complicazioni:

- I timer di sistema e delle periferiche non sono sincronizzati, inoltre non tutte le schede audio permettono di richiedere l'indice dei frame di riproduzione in qualunque momento ma solamente appena dopo un'interruzione. Per questo sono necessari degli algoritmi che facciano una stima in ogni momento del livello di riempimento del buffer.
- I timer Unix non sono di alta precisione. Su Linux tradizionale i watchdog sono arrotondati a multipli di 10ms. I kernel più recenti sono in grado di fornire una temporizzazione migliore. Esistono anche dei kernel creati apposta per questo tipo di utilizzo e vengono infatti denominati *low-latency kernels*.
- Una gestione dei flussi audio di questo tipo rischia di pesare sull'utilizzo della memoria centrale, per questo vengono adottati sistemi di *zero-copy* dei flussi audio.
- Si dipende comunque dalla massima dimensione del buffer supportata dall'hardware.
- Non è un sistema backward-compatibile con il modello tradizionale

2. Linux

2.1 Architettura dello stack audio

Per registrare o riprodurre suoni su un sistema Linux si passa attraverso una catena di componenti software che può essere schematizzata come segue:

1. Applicazione che riproduce o registra
2. Libreria che fornisce le API all'applicazione e si interfaccia con un intermediario prima del kernel
3. Intermediario tra la libreria e il kernel, generalmente definito come “sound server”
4. Kernel API
5. Kernel Sound System, che interpreta i comandi
6. Kernel Hardware driver

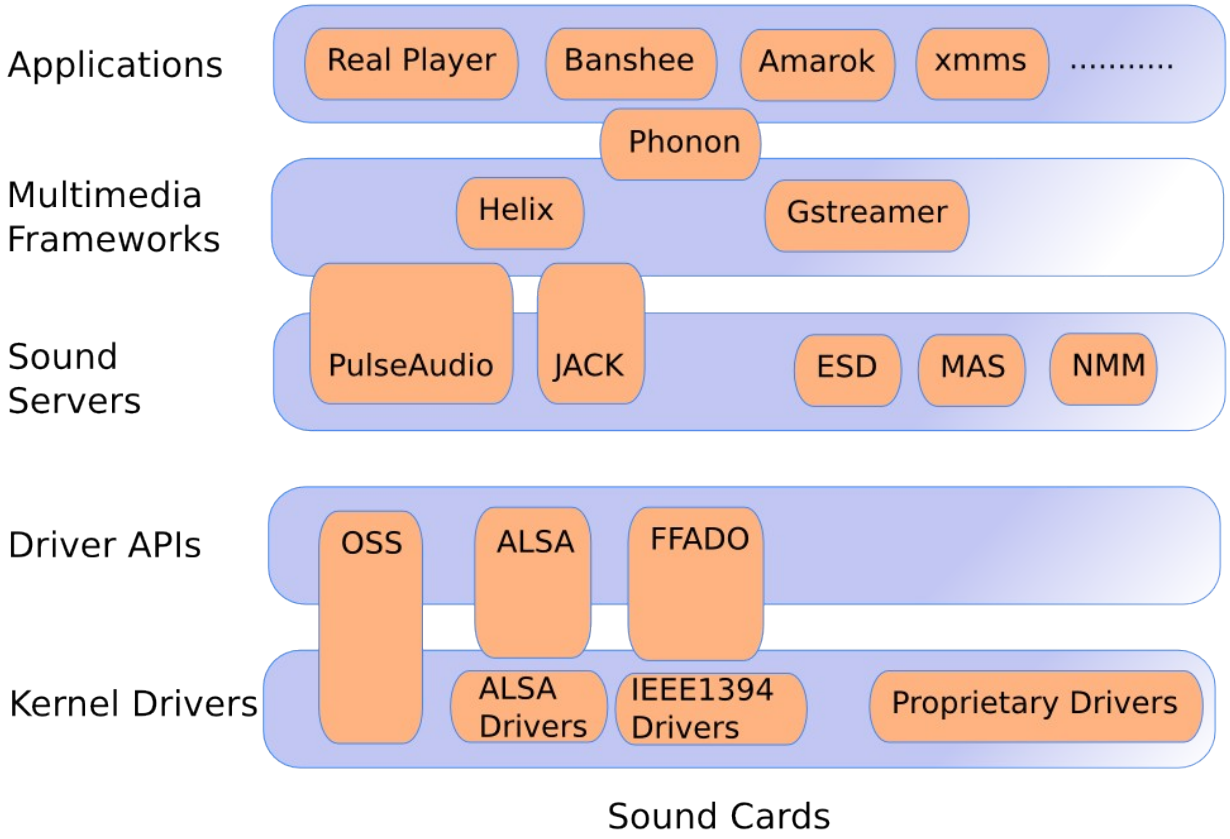

Fig. 2.1 - Linux Audio Stack

Come si può notare dalla figura 2.1 in Linux possono esistere molteplici alternative (di cui quelle in figura sono solo un piccolo sottoinsieme) per i medesimi livelli, in relazione alla scelta dell'utente e alla distribuzione utilizzata.

Linux presenta un'architettura modulare in tutto lo stack perché ad ogni livello è possibile scegliere tra diverse alternative o almeno tra diverse configurazioni. Tuttavia è da sottolineare che non tutti i software sono compatibili in assoluto con ogni configurazione di Linux. Spesso l'installazione di alcuni pacchetti rispetto ad altri è una scelta condizionata dal proprio desktop environment oppure dal desiderio di utilizzare applicativi di livello più alto che necessitano un determinato background. È il caso di Jack, il sound server più professionale, che è richiesto da tutti i software più potenti nell'ambiente audio di Linux (es. Ardour, Rosengarten,..).

2.2 Kernel Sound System

Storicamente su Linux si sono avvicendati due sound system differenti: OSS e ALSA.

Nella figura 2.1 si vede allo stesso livello anche *FFado* (Free Firewire Audio Drivers) che in realtà è un progetto della comunità Linux per lo sviluppo di driver per il supporto di periferiche audio professionali con connessione IEEE1394 e non vuole in alcun modo porsi come alternativa a OSS o ad ALSA.

2.2.1 Open Sound System

OSS è un'interfaccia standard per creare e catturare l'audio nei sistemi operativi Unix.

È basato sull'uso standard dei device Unix (es. POSIX read, write, ioctl, etc.).

Spesso il termine OSS è utilizzato anche per indicare la parte audio del kernel che fornisce l'interfaccia OSS: in questo senso si può vedere OSS come una collezione di driver per gestire l'hardware audio.

OSS è stato creato nel 1992 da Hannu Savolainen ed è disponibile per gli 11 principali sistemi operativi della famiglia Unix.

Il progetto Open Sound System è nato nella filosofia Linux come free software, tuttavia, vedendo il successo del progetto la compagnia 4FrontTechnologies assunse Savolainen e rese il suo supporto per le successive distribuzioni di OSS di tipo proprietario.

Di conseguenza la comunità Linux non inserì più l'implementazione OSS/free(3.x) nel kernel, mentre la 4FrontTechnologies continuava a lavorare sulla versione successiva (4.x).

Nel luglio 2007 la 4FrontTechnologies è tornata al modello commerciale del free-software e ha rilasciato i sorgenti di OSS per OpenSolaris in licenza CDDL e GPL per Linux. Nel gennaio 2008 ha rilasciato la versione per FreeBSD in licenza BSD.

I programmi che si basano su OSS effettuano delle chiamate di sistema a livello kernel e richiedono che il programmatore conosca i nomi specifici dei file corrispondenti ai device audio perché l'utilizzo di un device nel mondo Unix corrisponde all'apertura del file corrispondente:

```
fd = open(name, mode, 0);
```

Inoltre per impostare i parametri audio è necessario effettuare delle chiamate di controllo delle periferiche (ioctl):

```
ioctl(fd, SNDCTL_DSP_SPEED, &SR);
```

L'utilizzo dei device audio di OSS permette inoltre di scrivere e di leggere sulle porte audio in modo estremamente intuitivo (anche se estremamente limitato) direttamente dalla shell. Sono infatti possibili (e funzionanti) comandi come:

```
cat /dev/random > /dev/dsp
cat ~/Music/Peaches_and_Regalia.au > /dev/audio
cat /dev/audio > ./record.au
```

2.2.2 Advanced Linux Sound Architecture

Alcuni dei principali obiettivi del progetto ALSA sono la configurazione automatica della scheda audio e il supporto in modo semplice di multipli device audio.

Il progetto, gestito da Jaroslav Kysela, nacque nel 1998 dallo sviluppo di un driver per la scheda audio Gravis Ultrasound ed è rimasto con un ciclo di sviluppo separato dal kernel Linux fino al 2002 (Linux kernel 2.5.4-2.5.5). Una delle prime distribuzioni che ha incoraggiato l'uso di ALSA è stata SuSE.

Le caratteristiche mancanti a OSS nel 1998 che fecero dare avvio al progetto ALSA furono

- sintesi MIDI hardware-based
- mix hardware di canali multipli
- full-duplex
- device drivers thread safe e compatibili con hardware multiprocessore

ALSA è costituita da una serie di device drivers per le diverse possibili periferiche e da una

libreria di API (libasound.h).

Le API di ALSA permettono un'interfaccia di livello più alto rispetto alle API di OSS. Per esempio non è necessario conoscere i nomi specifici dei device audio perché ALSA si occupa di fornire al programmatore dei nomi logici. Il sistema di naming adotta il formato hw:i,j , dove i corrisponde al numero della periferica mentre j corrisponde al dispositivo sulla periferica. Ad esempio il primo device audio del sistema verrà chiamato *hw:0,0.* Esiste anche un alias per indicare la prima periferica in modo più intuitivo: *default.*

ALSA fornisce anche una serie di programmi a linea di comando per la gestione, tra cui un'applicazione di rilevamento e configurazione guidata delle periferiche audio (*alsaconf*),un mixer (*alsamixer*), due player (*aplay, alsaplayer*) e alcune utilità per la gestione di particolari caratteristiche di alcune periferiche audio.

Alcuni plugin di ALSA (il suo sviluppo modulare è uno dei motivi del suo successo) permettono di mixare diversi flussi audio (dmix) oppure di inviare lo stesso flusso a più canali (dshare).

Le API di ALSA possono essere raggruppate secondo le principali interfacce che supporta[24]:

- **Information Interface (/proc/asound):** directory comprendente diversi file di informazioni sulla configurazione di ALSA.
- **Control Interface (/dev/sndControlCx)**: per il controllo dei registri delle periferiche audio e dei device disponibili.
- **PCM Interface (/dev/snd/pcmCxDx)**: l'interfaccia per gestire l'audio digitale nella registrazione e nella riproduzione. È composta dalle API più utilizzate per le applicazioni audio.
- **Raw MIDI Interface (/dev/snd/midiCxDx)**: supporta Musical Instrument Digital Interface e fornisce accesso al bus MIDI delle schede audio. Le API gestiscono direttamente gli eventi MIDI e il programmatore è responsabile di gestire il protocollo e la temporizzazione.
- **Timer Interface (/dev/snd/timer)**: fornisce accesso all'hardware di temporizzazione delle periferiche audio per permettere la sincronizzazione degli eventi sonori.
- **Sequencer Interface (/dev/snd/seq)**: un'interfaccia di più alto livello rispetto a

[24] Il percorso del file corrispondente potrebbe variare per configurazioni particolari di ALSA.

quella raw per la gestione dei dati MIDI. Gestisce gran parte del protocollo MIDI e della temporizzazione.

- **Mixer Interface (/dev/mixer)**: controlla i dispositivi di instradamento del segnale audio nelle periferiche e controlla i livelli del volume. Questa interfaccia è costruita al di sopra di quella di controllo.

Le API di ALSA sono suddivisibili in due categorie: Full ALSA e Safe ALSA.
Safe ALSA è un sottoinsieme più limitato delle potenzialità di ALSA che tuttavia garantisce la portabilità del codice in quanto non prevede l'utilizzo di funzionalità specifiche dell'hardware in uso.

ALSA supporta dei device virtuali: per esempio è possibile riprodurre file MIDI senza avere un device MIDI fisico, semplicemente attraverso un device software che si occupa di gestire un Virtual Instrument come ALSAModularSynth o ZynAddSubFX**.**

ALSA permette di ottenere delle buone prestazioni e una bassa latenza perché permette di impostare la suddivisione del buffer applicativo in frammenti (cfr. cap. 1). Sono supportate entrambe le modalità stereo del PCM lineare, sia quella interfogliata sia quella non-interfogliata (quella di default).

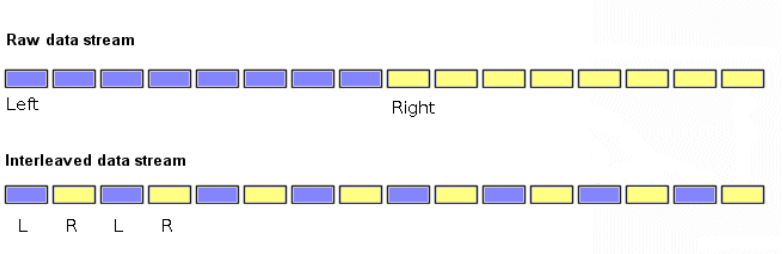

Fig. 2.2 - Interleaving

Di seguito un esempio di applicazione tipica audio in pseudocodice.

```
open interfaccia per registrazione
set parametri hardware
(modo di accesso, formato dei dati, canali, campionamento, etc. )
while (dati da processare)
  read dati PCM
close interfaccia
```

2.2.3 OSS vs. ALSA

Fino alla versione 2.4 del kernel Linux era standard Open Sound System, successivamente si è passati ad ALSA, e infatti dal kernel 2.5 OSS è marcato come deprecato.

È importante sottolineare però che ALSA è uno standard compatibile solo con Linux, mentre OSS è uniforme su tutto il mondo Unix.

Uno dei principali problemi che aveva OSS era la possibilità di utilizzare solo una sorgente audio alla volta: questo ostacolo poteva causare il crash di un'applicazione alla sua apertura se ce ne era un'altra che usava un device audio nello stesso momento.

Per aggirare questo problema furono creati diversi Sound Server che potessero ricevere più input audio, mixarli e mandarli ad OSS come un'unica sorgente.

ALSA supporta inoltre la possibilità di registrare molteplici sorgenti audio contemporaneamente o di gestire molti output diversi (per esempio Dolby 5.1): per questo è considerata molto più orientata al multimedia.

ALSA fornisce una modalità di emulazione OSS per poter permette l'utilizzo di software OSS anche con versioni del kernel Linux più recenti. Nello stesso modo esiste un'emulazione ALSA anche nell'implementazione Linux di OSS.

2.3 Kernel API

Open Sound System e ALSA utilizzano delle kernel API differenti, per questo un programma scritto per un sistema non è direttamente compatibile con l'altro.

Le modalità di emulazione OSS che ALSA fornisce sono due: In-Kernel e Out of Kernel.

Per fare in modo che un programma utilizzi l'emulazione di OSS su ALSA se non è configurato di default inserire aoss prima del nome del programma:

```
aoss  realplay
```

2.3.1 In-kernel OSS emulation

Questa modalità ha due possibili configurazioni: compilata direttamente nel kernel oppure compilata come un modulo.

La prima possibilità può creare dei problemi perché non può essere rimossa senza dover ricompilare tutto il kernel, può rendere la configurazione e il debugging più difficili, non può essere disattivata.

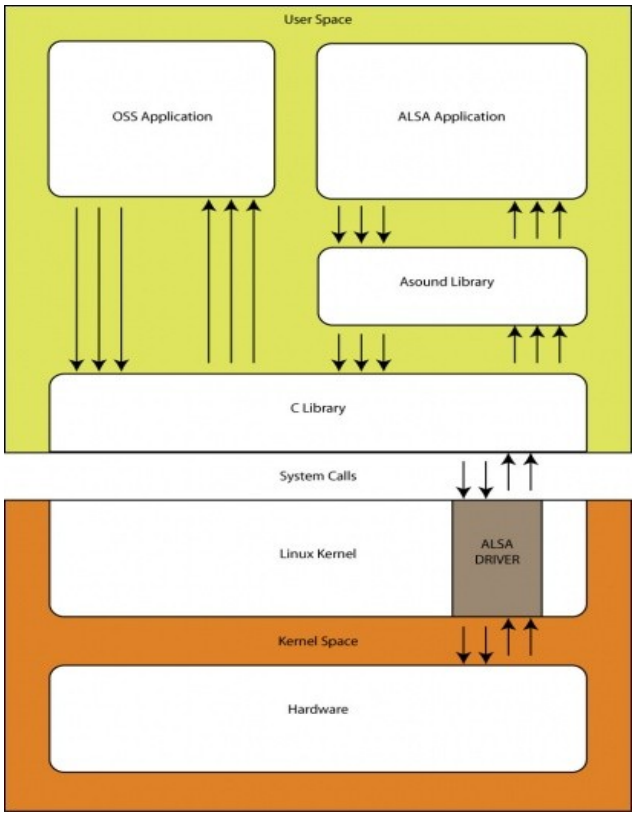

Fig. 2.3 - In-Kernel OSS Emulation

È di solito conveniente usare la modalità di emulazione interna al kernel perché l'efficienza è molto migliore ed è più semplice da usare.

Vantaggi:

- È il metodo più semplice e più efficiente
- Funziona sempre se c'è una sola sorgente audio
- Alcune schede audio hanno implementato il mixaggio hardware per cui è possibile usare più sorgenti.

Svantaggi:

- Non è possibile implementare un mixaggio software in modo elegante.
- Con la maggior parte delle schede audio integrate è possibile riprodurre un solo suono alla volta.
- Quando si usa il modulo OSS il resto di ALSA è bloccato.
- Se un programma apre /dev/dsp (o /dev/audio) e non lo chiude, il device audio è bloccato fintantochè è prelazionato da quel determinto programma

2.3.2 Out of kernel OSS emulation

Esiste un sistema di emulazione architetturalmente più semplice: la libreria libaoss.h: un wrapper che permette di tradurre le chiamate di OSS a delle chiamate ALSA (libasound) attraverso una libreria inserita tra l'applicazione e la libc.

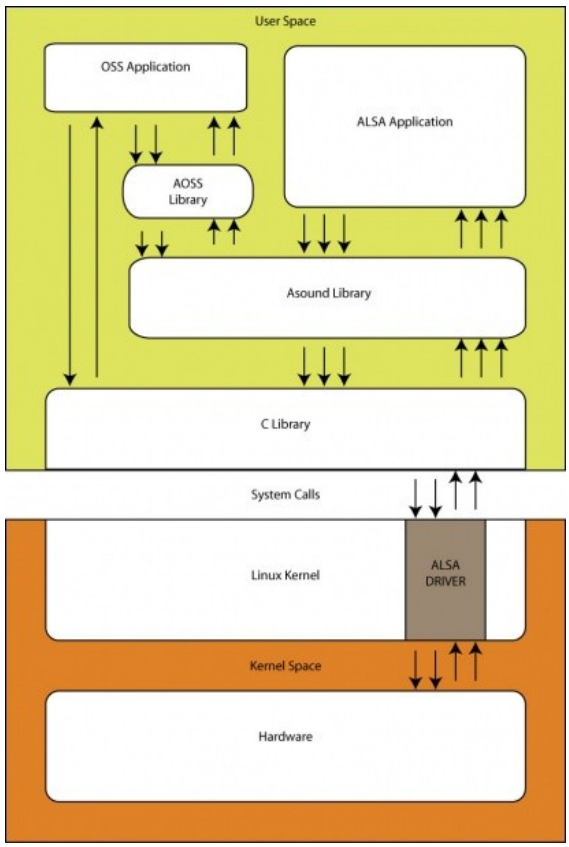

Fig. 2.4 - User-space OSS Emulation Mode

Vantaggi:

- Con questo metodo il sistema può fornire mixaggio software.
- Per la maggior parte delle system-call l'overhead aggiunto è minimo.

Svantaggi:

- Non è sempre possibile inserire la libreria tra l'applicazione e la libc.
- È possibile che sia necessario dover reimplementare l'intera libc per far funzionare questa modalità.

2.3.3 *Forward* out of kernel OSS emulation

Un sistema più complesso ma più elegante (chiamato Forward) è un modulo del kernel che intercetta le chiamate OSS e risveglia uno specifico daemon (Forward OSS Daemon), che permette di interfacciarsi con la libreria Asound.

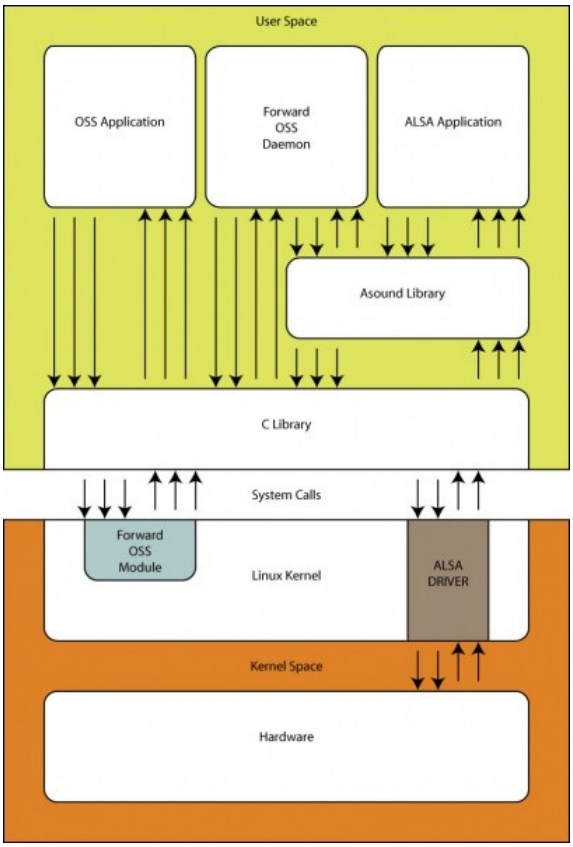

Fig. 2.5 - Forward OSS Emulation Mode

Vantaggi:

- Funziona con tutte le applicazioni.
- Fornisce mixaggio software.
- Non crea overhead nelle chiamate di sistema non destinate al device audio.

Svantaggi:

- Questa tecnica crea molto overhead per le chiamate audio: ci possono essere dei problemi di latenza nel flusso audio.

2.4 Sound Server

I sound server di Linux nascono per l'esigenza di aumentare le funzionalità offerte dal sistema operativo al programmatore, tra cui il mixaggio, l'utilizzo trasparente in rete, l'astrazione delle API audio e dell'hardware.

2.4.1 Dmix

Si tratta in realtà di un modulo di ALSA, installato ormai di default, che agisce come livello di indirettezza per il mixaggio di diversi flussi.

Ad esempio rende possibile la riproduzione di due flussi audio contemporanei tramite il lancio di due istanze concorrenti di alsaplayer che usano il modulo *dmix*:

```
alsaplayer -o alsa -d plug:dmix pippo.mp3 & alsaplayer -o alsa -d
plug:dmix pluto.mp3
```

È anche possibile configurare alsa in modo che *dmix* venga utilizzato di default, creando o modificando il file /etc/asound.conf[25]:

```
pcm.!default {
    type plug
    slave.pcm "dmixer"
}


pcm.dsp0 {
    type plug
    slave.pcm "dmixer"
}
pcm.dmixer {
    type dmix
    ipc_key 1024
    slave {
        pcm "hw:0,0"
        period_time 0
        period_size 1024
        buffer_size 8192
        rate 44100   #molte nuove periferiche sono solo a 48kHz!!
    }
    bindings {
        0 0
        1 1
    }
}

ctl.dmixer {
    type hw
    card 0
}
```

[25] Attualmente dmix viene installato di default insieme ad alsa con già questa configurazione.

2.4.2 Vmix

Per chi non utilizzi ALSA esiste *vmix*, un modulo kernel di OSS4 che permette il mixaggio e ha delle funzionalità molto simili a dmix.

2.4.3 NAS

Network Audio System è un sound server progettato con funzionalità di rete alla Network Computing Devices da Jim Fulton, Greg Renda e Dave Lemke nei primi anni '90 per i sistemi Unix[26].

Le sue caratteristiche principali sono:

- Audio in rete indipendente dalla periferica
- Supporto per molte tipologie di file e codifiche audio
- Possibile caching per successive riproduzioni più veloci
- Mixing e manipolazione separata delle tracce audio
- Piccola dimensione in memoria

2.4.4 aRTS

Analog ReaTime Synthesizer è un framework audio creato da Stefan Westerfeld il cui sviluppo oggi non viene più portato avanti.

È stato lo standard di KDE[27] e la sua componente principale è il sound server (*artsd*) che permette delle buone performance in realtime.

2.4.5 ESD

Enlightened Sound Daemon è il sound server per Enlightenment e GNOME. È stato sviluppato nel 1998 da Eric Mitchel e ad oggi molti programmi lo supportano. Esso fornisce mixing e funzionalità di rete. La sua documentazione in rete è piuttosto scarsa in quanto quasi tutti i documenti rimandano al codice stesso di ESD.

[26] Per ulteriori dettagli si rimanda a [82].

[27] Per KDE 2 e 3

2.4.6 Pulse Audio

PA[28] è un Sound Server molto potente cross-platform (al momento supporta in modo nativo Linux, Solaris, FreeBSD, Windows), ha un'architettura estensibile a plug-in attraverso moduli a caricamento dinamico automatico ed ha le seguenti caratteristiche principali:

- Supporto per molteplici porte di I/O (definite come sink per l'output e source per l'input)
- Dall'ultima release applica il modello Glitch Free
- Comportamento a bassa latenza con misurazione molto accurata
- Incapsulabile in altro software (il core è disponibile come libreria c)
- API C completamente asincrone con due varianti sincrone
- Interfaccia a linea di comando semplice e riconfigurabile
- Conversioni di formato e ricampionamento automatici
- Architettura zero-copy per ridurre le copie in memoria e ottimizzare le prestazioni
- Permette l'utilizzo di più periferiche hardware audio contemporaneamente con sistemazione automatica della frequenza di campionamento
- Permette di sincronizzare completamente multipli flussi audio

2.4.7 Jack

È un sound server professionale a bassissima latenza.

È pensato con un'architettura client-server nella quale è presente un server Jack[29] e diversi plug-in.

Le applicazioni che utilizzano Jack possono avere una qualunque interfaccia grafica, in quanto esso non specifica GUI toolkits o librerie.

Jack è pensato per fornire una completa sincronizzazione di tutti i client e utilizza per la codifica audio il formato standard IEEE 32bit floating point non interfogliato.

[28] Dettagli in [10], [79], [80] e [81].

[29] Nulla vieta che possano coesistere più server jack, i quali tuttavia saranno completamente indipendenti. Per maggiori dettagli si rimanda a [75], [76], [77]

Ogni client Jack può produrre o consumare un numero arbitrario di flussi di dati[30].

Permette che vengano aggiunti o rimossi client durante l'esecuzione del server.

Lo scambio di dati con i client avviene tramite memoria condivisa mentre la comunicazione tramite pipe.

I thread audio non utilizzano chiamate di sistema bloccanti: malloc, sleep, read/write/open/close, pthread_cond_wait, pthread_mutex_lock.

Le API di Jack non forniscono:

- nessuna configurazione hardware
- nessuna configurazione software
- nessuna negoziazione di formato (tutti i canali sono FP 32bit)
- nessun main-loop

richiedono però:

- capacità di programmazione multithread lock-free
- l'utilizzo di un sistema client server con un modello di tipo pull

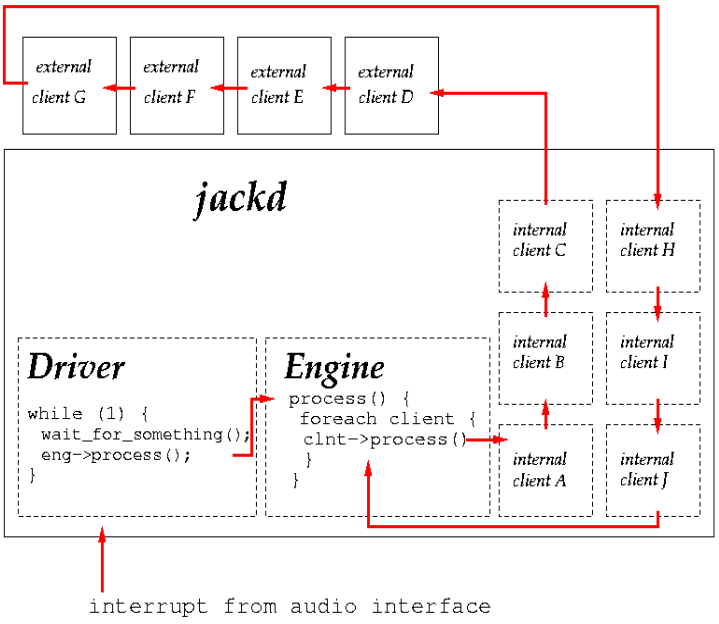

Fig. 2.6 - struttura dei client di Jack

[30] Il disegno originale prevede che Jack possa essere utilizzato anche per dati non-audio (es. video) anche se al momento non è supportata che la parte audio.

3. Mac OS X

Con il nome Core Audio si intende generalmente tutto lo stack audio in ambiente Macintosh, anche se in particolare questo nome si riferisce principalmente al livello utente ed in particolare è il nome del SDK e di uno dei framework che riguardano la parte audio di OSX.

Nella documentazione sono ancora presenti dei riferimenti a Sound Manager, il sistema di gestione dell'audio per Macintosh nelle versioni precedenti a OS X, tuttavia è ormai deprecato per cui si è scelto di non trattarlo in questa sede.

Alcune delle possibilità che offre Core Audio sono:

- Interfacce per lo sviluppo di Plug-in per la sintesi audio e l'elaborazione del segnale (DSP)
- Supporto integrato per la lettura / scrittura in una grande varietà di formati di file e di codifica
- Interfacce per lo sviluppo di Plug-in per la gestione di formati di file e di codifica personalizzati
- Un approccio modulare per costruire delle catene audio (signal-chain)
- Gestione scalabile dell'audio multicanale

- Semplice sincronizzazione dei dati audio e MIDI in fase di registrazione e di riproduzione
- Un'interfaccia standard per tutti i device audio, sia interni che esterni, senza distinzione in base al tipo di connessione (PCI, USB, IEEE1394,...)

3.1 Architettura

L'architettura di Core Audio è composta di diversi livelli di astrazione diversamente integrati nel sistema operativo.

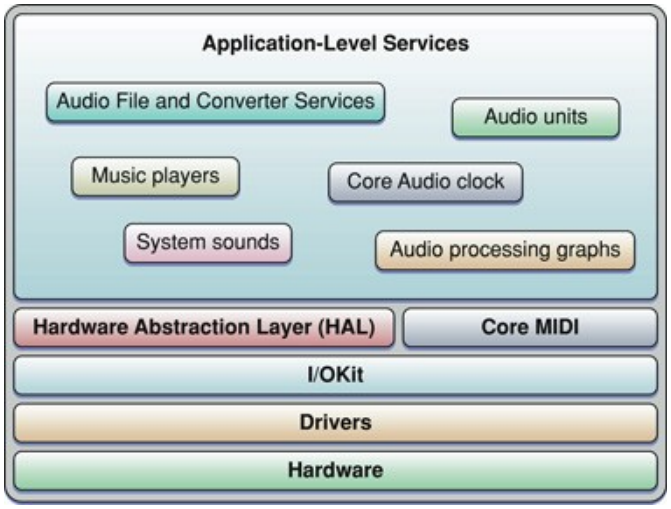

Fig.3.1 - Struttura dello stack audio di Mac OS X

Al di sopra dei Driver è presente un I/O Kit che fornisce un'interfaccia comune di comunicazione con le periferiche, di qualunque genere esse siano (PCI, USB, IEEE1394,...). Questo livello non è specifico per l'audio ma è generico per ogni tipo di periferica.

È presente poi un Hardware Abstraction Layer (HAL) audio, che fornisce un'interfaccia stabile per le applicazioni per interagire con l'hardware. L'HAL fornisce anche informazioni riguardo alla temporizzazione, alla sincronizzazione e alla latenza delle periferiche in uso.

A fianco al livello di astrazione Hardware si trova Core MIDI, che gestisce i device e i canali MIDI, fornendo un'interfaccia a bassa latenza.

Al di sopra di questi moduli di Core Audio si trovano i servizi a livello applicazione, quali i player, i suoni di sistema, il clock di CoreAudio, le Audio Units, le catene audio, i servizi di gestione dei file e di conversione.

3.2 I/O kit

Le principali caratteristiche che offre l'I/O kit sono:

- configurazione dinamica ed automatica dei device (plug-and.play)
- supporto di nuove tipologie di periferiche multimediali
- gestione del risparmio energetico (es. sleep mode)
- spazio di indirizzamento della memoria separato tra il kernel e i programmi utente
- preemptive multitasking
- gestione del multiprocessing simmetrico
- astrazioni comuni condivise tra le tipologie di device
- Prestazioni molto elevate
- Supporto per una complessità arbitraria per la gestione dei livelli di oggetti

Inoltre i programmatori hanno a disposizione attraverso l'I/O kit SDK:

- Un framework orientato agli oggetti che implementa il comportamento comune condiviso tra tutti i driver e le famiglie di periferiche
- primitive di IPC e gestione dei dati per il multiprocessing, il controllo dei task e dei trasferimenti I/O
- I/O Registry: un database che traccia gli oggetti istanziati e fornisce informazioni a loro riguardo
- I/O Catalog: un database di tutte le classi di I/O disponibili su un sistema.
- una serie di interfacce e meccanismi di plug-in che permettono alle applicazioni utente di comunicare con i driver

3.2.1 I driver

L'ambiente di esecuzione dei device driver è composto da un'architettura dinamica e strutturata a livelli, che permette ai driver di essere caricati e disallocati in qualunque momento, risparmiando risorse del sistema. Esso offre delle funzionalità standard di gestione dei dati durante le comuni operazioni di I/O ed un sistema robusto per proteggere l'accesso alle risorse dei driver attivi, che libera i programmatori dall'onere di scrivere loro stessi il codice per abilitare e disabilitare le interruzioni e gestire le variabili di lock sulle risorse specifiche dei driver.

È anche fornito l'accesso ai servizi nella libreria libkern in C++ per gestire le collezioni, effettuare operazioni atomiche e scambiare byte tra differenti tipologie di hardware.

L'I/O kit offre tutte le funzionalità e le strutture dati necessarie agli sviluppatori dei driver.

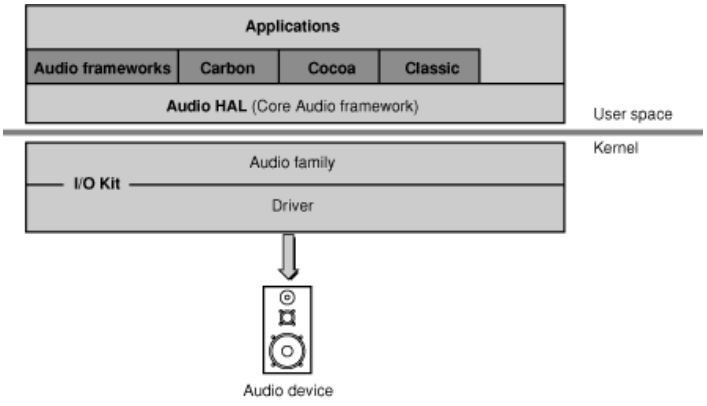

Fig. 3.2 - L'I/O kit all'interno di Mac OS X

L'I/O kit audio può essere suddiviso in due livelli: le API per i Driver e la Audio Family, che ha il compito di fornire gli oggetti necessari per la comunicazione con i livelli superiori e di mantenere il sample buffer e il mix buffer per l'hardware.

3.2.2 La Audio Family

I driver costruiti con l'I/O kit possono supportare qualunque tipo di hardware.

Uno dei compiti di questo strato di software è quello di effettuare le conversioni di formato necessarie per mettere in comunicazione HAL con l'hardware: come abbiamo già detto il formato standard di HAL è il FP32bit mentre il formato dell'hardware varia a seconda della

periferica.

Le interazioni tra il motore DMA, il driver e l'audio HAL sono basate sull'assunzione che in una direzione il flusso di dati audio proceda continuamente con la stessa velocità (determinata dalla frequenza di campionamento). Audio Family imposta diversi timer per sincronizzare le azioni degli agenti coinvolti nel trasferimento dei dati. questi meccanismi di sincronizzazione assicurano che i dati audio siano processati con la massima velocità e la minima latenza (cfr. cap. 1).

Esempio di flusso in ingresso: subito dopo la scrittura da parte di DMA dei campioni audio nel buffer del driver, quest'ultimo legge i dati, li converte (generalmente da interi a FP32bit[31]) e scrive i campioni risultanti nel mixer buffer, che permette il passaggio all'Audio HAL.

L'Hardware Abstraction Layer (HAL) funziona come interfaccia "device" per l'Audio Family.

HAL si trova appena sotto la linea di confine tra il kernel e lo spazio utente e il suo compito è quello di rendere accessibili ai software i flussi in ingresso dai driver e di trasmettere a quest'ultimi i flussi in uscita dai livelli superiori.

3.3 Hardware Abstraction Layer

Audio HAL è contenuto nel framework Core Audio (CoreAudio.framework) e presenta delle API per i linguaggi C e JAVA.

Le API di Audio HAL sono composte da tre principali astrazioni:

-Le **Audio Hardware API** danno accesso alle entità audio che hanno una visibilità globale, quali la lista dei device presenti e il device di default.

-Le **Audio Device API** permettono di gestire una specifica periferica e i motori di I/O che

[31] Tutti i dati audio trattati dai vari livelli di Mac OSX sono 32bit Floating Point.

questa contiene. Un audio device viene visto da HAL come un singolo componente di I/O con il suo corrispondente clock e tutti i buffer che sono sincronizzati con esso.

-Le **Audio Stream API** rendono possibile il controllo e l'interrogazione di un flusso audio. Ogni periferica possiede almeno un flusso audio che corrisponde al buffer di memoria utilizzato per trasferire l'audio tra il confine utente/kernel. Queste API specificano inoltre il formato dei dati audio.

Una parte critica delle API per l'hardware audio riguarda le proprietà delle periferiche e le loro notifiche.

Queste API permettono alle applicazioni di ricevere (get) ed impostare (set) le proprietà audio.

I metodi get sono sincroni ma i metodi set sono asincroni, ovvero necessitano di notifiche. I client dell'Audio HAL (programmi che utilizzano le API Audio HAL) implementano dei metodi "listener" (in ascolto) delle funzioni associate all'hardware audio.

Un esempio di un generatore monofonico sinusoidale utilizza HAL per inviare il proprio output alla periferica audio.

Il primo passo necessario è quello di conoscere le proprietà hardware:

```
AudioHardwareGetProperty(kAudioHardwarePropertyDefaultOutputDevice,
&count, (void *) &device);
```

In seguito bisogna conoscere la dimensione del buffer:

```
 AudioDeviceGetProperty(device, 0, false,
kAudioDevicePropertyBufferSize, &count, &deviceBufferSize);
```

e il formato dei dati accettati dalla periferica:

```
AudioDeviceGetProperty(device, 0, false,
kAudioDevicePropertyStreamFormat, &count, &deviceFormat);
```

A questo punto è possibile creare il processo di I/O:

```
AudioDeviceAddIOProc(device, appIOProc, (void *) def);
```

e farlo eseguire:

```
AudioDeviceStart(device, appIOProc);
```

Di seguito un esempio di struttura del processo di I/O:

```
OSStatus appIOProc (AudioDeviceID  inDevice,
                    const AudioTimeStamp*  inNow,
                    const AudioBufferList*   inInputData,
                    const AudioTimeStamp*  inInputTime,
                    AudioBufferList*  outOutputData,
                    const AudioTimeStamp* inOutputTime,
                    void* defptr)
{
/* Generazione della sinusoide e invio in output */
}
```

3.4 Framework secondari

Mac OS X ha molti framework secondari oltre a quello CoreAudio, per esempio *Audio Units* e *Audio Toolbox* sono costruiti direttamente sopra CoreAudio.

MIDI System Services, composto a sua volta dai due moduli Core MIDI e Core MIDI Server, non dipende direttamente da Core Audio, ma è comunque un utente dei servizi audio.

Il framework *Audio Units* fornisce supporto per generare, processare, ricevere e manipolare flussi di dati audio.
Questa funzionalità è basata sulla nozione di Audio Unit.

Le Audio Unit (AU) sono una particolare istanza di un elemento detto Component. Un Component è un pezzo di codice che fornisce un insieme di servizi ad uno o più client.

Nel caso delle AU questi client possono usare le AU sia singolarmente che interconnesse tra

loro a formare un grafo audio (Audio signal graph). Per connettere più AU i client possono usare le API AUGraph nel Toolbox framework.

Audio Toolbox complementa le AU con due grandi astrazioni: *AUGraph* e *Music Player*. Audio Toolbox contiene i servizi di *Audio File Stream* e di *Audio Queue*. Si tratta di una collezione di API scritte in C per la gestione dei flussi audio.

Audio File Stream è in grado di identificare i pacchetti audio in un gruppo "raw" di byte, mentre Audio Queue si occupa della riproduzione.

AUGraph fornisce una completa descrizione di una rete di elaborazione del segnale.

Le API di Music Player usano AUGraph per fornire i servizi di un elemento sequencer che raccoglie gli eventi audio in tracce che possono essere copiate, incollate e riprodotte a loop all'interno di una sequenza. Un Music Player è in grado di riprodurre una Music Sequence che può essere creata da un file MIDI standard o da altre metodologie di input.

MIDI System Services è una tecnologia che permette la comunicazione tra le applicazioni ed i device MIDI che comprende due framework: CoreMIDI.framework e CoreMIDIServer.framework.

Il server MIDI si occupa della gestione dei driver MIDI, che normalmente non sono di competenza dell'I/O kit. Il framework CoreMIDI, offre un'interfaccia per la configurazione dei dispositivi MIDI e per il loro utilizzo. In particolare fa distinzione tra i *device*, le *entities* e gli *endpoints*, che rispettivamente corrispondono alle periferiche MIDI nella loro interezza, ai componenti (sintetizzatori, porte di I/O) e alle sorgenti e destinazioni dei flussi dei 16 canali MIDI.

3.5 Audio Unit

Si tratta di Plug-in per gestire dati audio[32].

All'interno di Mac Os X quasi qualunque tipo di manipolazione e gestione di dati audio può essere effettuata tramite audio unit.

Essendo Plug-in si tratta di moduli software non indipendenti: è necessaria un'applicazione che li richiami come modulo di processing (Es. *AULab*, *GarageBand*, *Logic Pro*, *SoundTrackPro*, *ProTools*,...)

Questo tipo di plug-in ha generalmente una grande efficienza e per questo è possibile utilizzare catene di AU molto complesse senza appesantire particolarmente il sistema.

Le Audio Unit possono essere delle seguenti tipologie:

- DSP (processore di segnale), per esempio un filtro passa basso, un flanger, un compressore, un overdrive,..
- Instrument: sintetizzatore software che risponde ad un input MIDI
- Signal source: sintetizzatore programmato che non risponde a messaggi MIDI
- Interfaccia di I/O
- Signal converter: permette di codificare / decodificare il segnale
- Mixer o splitter
- Offline effect unit: per eseguire le operazioni troppo onerose dal punti di vista del calcolo o non possibili da eseguire in real-time (es. reverse di un intero file).

Core Audio utilizza come formato nativo il PCM lineare little endian floating point 32bit, anche se è possibile creare dei convertitori per utilizzare altre varianti del PCM oppure formati di codifica compressi. Core Audio fornisce diversi codec per la codifica /decodifica dei formati compressi lossless e lossy, anche se del MP3[33] non fornisce l'encoder (in quanto è un brevetto della Thompson-Fraunhofer il cui uso commerciale viene concesso solo su licenza).

[32] Nello specifico le Audio Unit sono componenti di Component Manager.

[33] Maggiori dettagli in [85].

3.6 Core Audio SDK

Per favorire lo sviluppo di applicazioni audio Apple fornisce un development kit specifico per Core Audio.

Esso contiene molti esempi di codice per lo sviluppo di applicazioni audio e MIDI oltre a tool di diagnostica e applicazioni per il testing.

In particolare CoreAudio SDK v1.4.3 offre le seguenti caratteristiche:

- AULab

  Applicazione di testing delle Audio Unit
- HALLab

  Applicazione di testing e di diagnostica per interagire con lo stato globale dell'audio del sistema, compresi i device hardware
- Esempi di codice
- Un Framework C++ per la costruzione di Audio Unit

Questo framework semplifica e riduce il carico di lavoro necessario per sviluppare un plug-in audio, permettendo al programmatore di non dover conoscere i dettagli dell'interfaccia plug-in del Component Manager. Sostanzialmente per lo sviluppo di un'audio unit molto semplice è necessario solo implementare i metodi ereditati dalla classe di riferimento.

La classe principale del Plugin deve estendere AUEffectBase, che offre le strutture dati e le funzioni necessarie per il plugin.

```
class TremoloUnit : public AUEffectBase
```

inoltre la parte di elaborazione del segnale verrà effettuata in una classe che estende AUKernelBase:

```
class TremoloUnitKernel : public AUKernelBase
```

successivamente avremo nell'implementazione la componente che istanzia l'interfaccia:

```
TremoloUnit::TremoloUnit(AudioUnit component)
    : AUEffectBase(component)
{
    CreateElements();
    Globals()->UseIndexedParameters(kNumberOfParameters);
    SetAFactoryPresetAsCurrent (kPresets[kPreset_Default]);
    SetParameter(kParameter_Gain, kDefaultValue_Tremolo_Gain );
    SetParameter(kParameter_Frequency, kDefaultValue_Tremolo_Freq);
    SetParameter(kParameter_Depth, kDefaultValue_Tremolo_Depth );
    SetParameter(kParameter_Waveform, kDefaultValue_Tremolo_Waveform );

#if AU_DEBUG_DISPATCHER
    mDebugDispatcher = new AUDebugDispatcher (this);
#endif

}
```

una componente preparatoria per il processing (crea le forme d'onda che poi possono essere applicate, ad es. sinusoide e onda quadra):

```
TremoloUnit::TremoloUnitKernel::TremoloUnitKernel(AUEffectBase
*inAudioUnit)
:
    AUKernelBase (inAudioUnit), mSamplesProcessed (0), mCurrentScale (0)
{
    for (int i = 0; i < kWaveArraySize; ++i){
          double radians = i * 2.0 * pi / kWaveArraySize;
          mSine[i] = (sin(radians) + 1.0) * 0.5;
    }

    for (int i = 0; i < kWaveArraySize; ++i){
          double radians = i * 2.0 * pi / kWaveArraySize;
          radians = radians + 0.32;
          mSquare[i] =
                (
```

```
                sin(radians) +
                0.3 * sin(3 * radians) +
                0.15 * sin(5 * radians) +
                0.075 * sin(7 * radians) +
                0.0375 * sin(9 * radians) +
                0.01875 * sin(11 * radians) +
                0.009375 * sin(13 * radians) +
                0.8
                ) * 0.63;
    }
    mSampleFrequency = GetSampleRate();
}
```

ed il processing vero e proprio:

```
   void        TremoloUnit::TremoloUnitKernel::Process(
       const Float32      *inSourceP,
       Float32            *inDestP,
       UInt32             inSamplesToProcess,
       UInt32             inNumChannels, /* for version 2 AudioUnits
                                            inNumChannels is always 1 */
       bool               &ioSilence )
   {

   /* [...] */

   for (int i = inSamplesToProcess; i > 0; --i){
         int index = static_cast<long>(mSamplesProcessed * mCurrentScale)
   %     kWaveArraySize;

         if((mNextScale != mCurrentScale) && (index == 0))
               mSamplesProcessed = 0;

         rawTremoloGain = waveArrayPointer[index];
```

```
        tremoloGain             = (rawTremoloGain * tremoloDepth -
                                      tremoloDepth + 100) * 0.1;
        inputSample             = *sourceP;
        outputSample            = gain*(inputSample * tremoloGain);

        *destP                  = outputSample;
        sourceP                 += 1;
        destP                   += 1;
        mSamplesProcessed       += 1;
        }
    }
```

4. Windows

La Microsoft ha sviluppato in questi ultimi dodici anni moltissime tecnologie a supporto dello sviluppo di applicazioni audio, soprattutto per la spinta commerciale dovuta ai videogiochi e, più recentemente, alle console di gaming[34].

Come sarà illustrato sono presenti due tipi di approcci all'audio in ambiente Windows: uno più "guidato", con prestazioni inferiori, mirato ad un mercato non professionale, e un metodo di utilizzo delle risorse audio più diretto, genericamente attraverso primitive audio sviluppate da terze parti.

4.1 Componenti audio di Windows XP

Windows Xp offre al livello applicativo diverse API per utilizzare l'audio. Queste interfacce programmative hanno un'organizzazione differente tra loro all'interno della struttura del

[34] L'utilizzo di Windows per l'entertainment ha determinato lo sviluppo di prodotti specifici: Microsoft Windows Embedded è stato spesso utilizzato per i videogiochi delle sale giochi (arcade) e nel 2001 il colosso di Redmond ha rilasciato negli USA la console di gaming Xbox.

sistema operativo, per cui è possibile utilizzare stack audio composti diversamente in relazione all'utilizzo di applicazioni differenti.

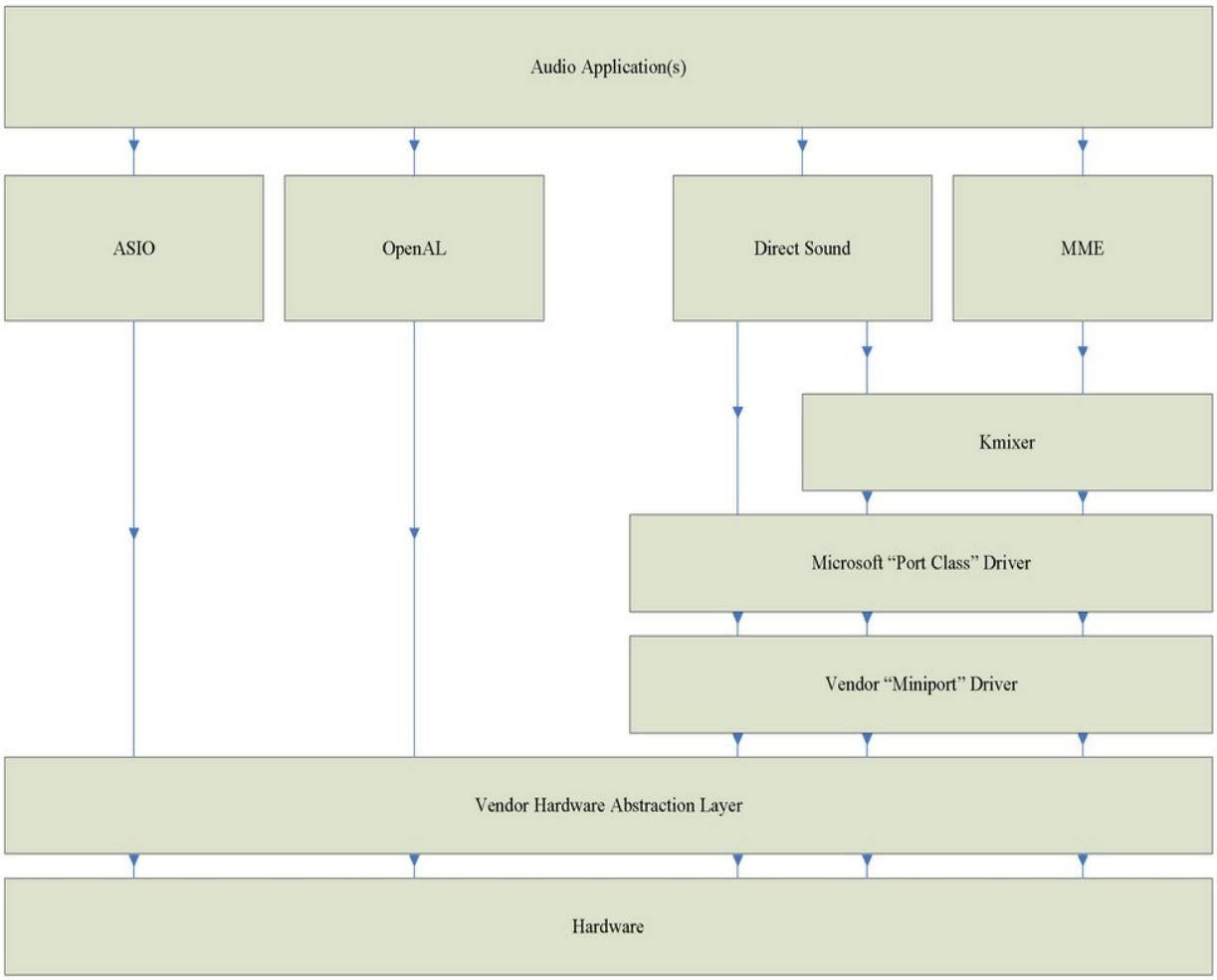

Fig. 4.1 – Organizzazione dello stack audio di Windows XP

Le componenti principali sviluppate da Microsoft sono le MME (MultiMedia Extensions), DirectX, il Kernel Mixer e il Port Class Driver.

4.1.1 MME

Si tratta di un'insieme di tecnologie di cui alcune introdotte dalla versione 3.0 di Microsoft Windows, che offrono un'interfaccia semplice al programmatore e all'utente.

Molte di queste tecnologie sono oggi in disuso, perché superate e perché Windows Vista ha completamente cambiato il sistema audio di Windows.

4.1.2 MCI

Windows Media Control Interface è una componente di MME che fornisce delle API per la gestione di periferiche di vario genere in modo indipendente dall'hardware. Un maggiore approfondimento su MCI sarà sviluppato nel paragrafo 4.5.

4.1.3 KMIXER

Il Kernel Audio Mixer gestisce i buffer di mixaggio di diverse sorgenti di I/O. È in esecuzione attraverso il processo KMixer.sys e fornisce le seguenti funzionalità:

- Missaggio di multipli flussi audio PCM
- Conversione di formato, quantizzazione e frequenza di campionamento
- Configurazione degli attuatori e mappatura dei canali audio.

Il Kernel Mixer è nato principalmente come Sound Server per semplificare la gestione di multipli flussi audio, specialmente per periferiche audio economiche che non supportano più di un canale stereo.

Tuttavia esso ha introdotto una serie di problemi tra cui:

- Latenza che si aggira intorno ai 30ms e non può essere ridotta, in quanto il Kernel Mixer è logicamente situato appena al di sopra dei driver.
- Esso prova a mixare qualunque flusso che passi attraverso di lui, anche se in un formato non supportato. Ad esempio si possono creare problemi se un player cerca di invare un flusso in surround AC3 ad un ricevitore home-cinema esterno tramite S-PDIF[35].

Per evitare questi problemi è stato necessario introdurre delle nuove kernel-API per bypassare il Kernel Mixer.

[35] È l'acronimo di Sony Philips Digital Interface Format: è una collezione di specifiche hardware e protocolli a basso livello per il trasporto di segnali audio digitali PCM stereo tra periferiche e componenti stereo. S/PDIF è la versione consumer dello standard conosciuto come AES/EBU (Audio Engineering Society / European Broadcasting Union) da cui si distingue per alcune differenze e componenti più economici.

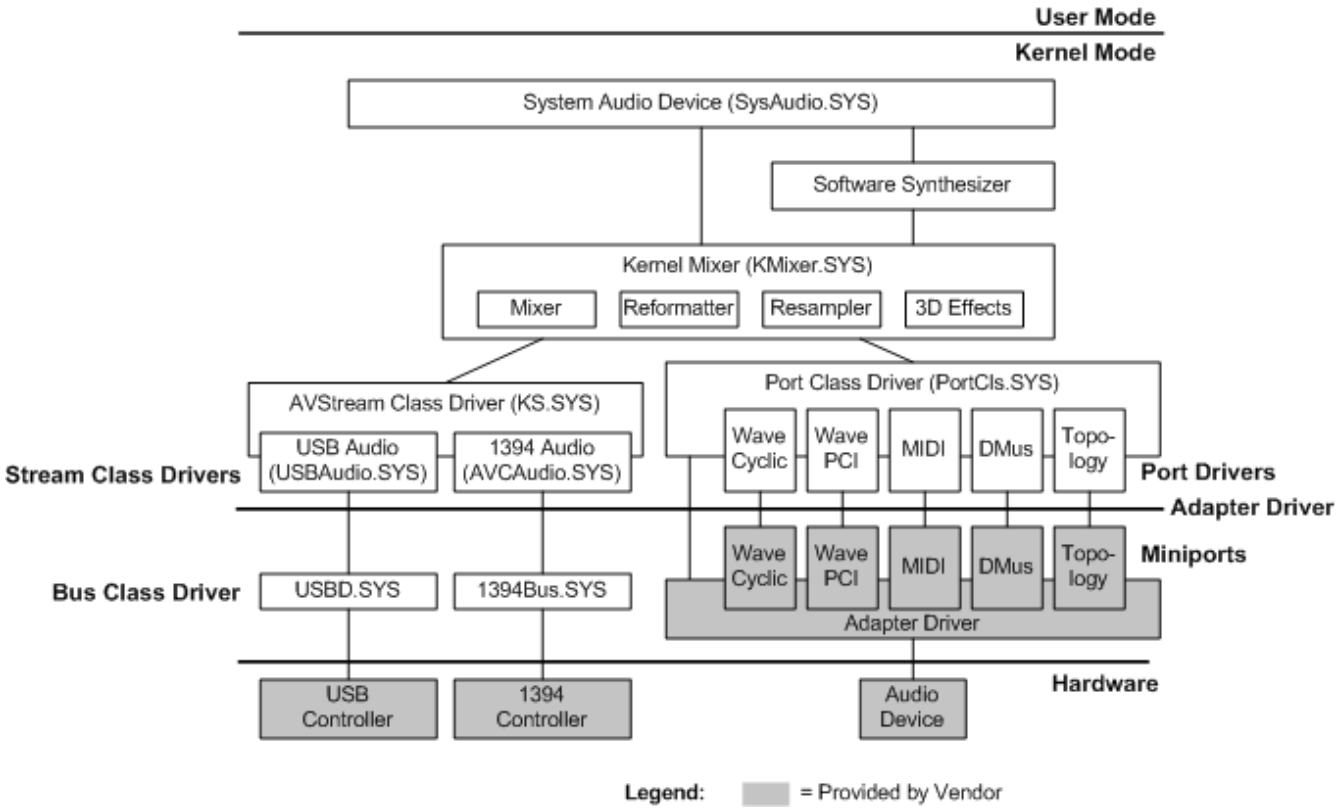

Fig. 4.2 - Struttura Kernel e driver audio di Windows

4.2 Driver

Il sistema di comunicazione con i driver in ambiente Windows segue un modello imposto dalla Microsoft: il *Windows Driver Model* (WDM).

I driver audio sono composti da due parti a loro volta suddivise in più parti tra loro corrispondenti:

- Il *Port Class Driver* si occupa di comunicare tramite il WDM con le applicazioni e contiene i *Port Driver* che corrispondono ciascuno ad una componente specifica del device.
- L'*Audio Adapter Driver* contiene i *Miniport Driver* i quali corrispondono uno ad uno ai Port Driver del Port Class Driver e contengono il codice necessario a comunicare con la periferica hardware in riferimento alle sue specifiche componenti.

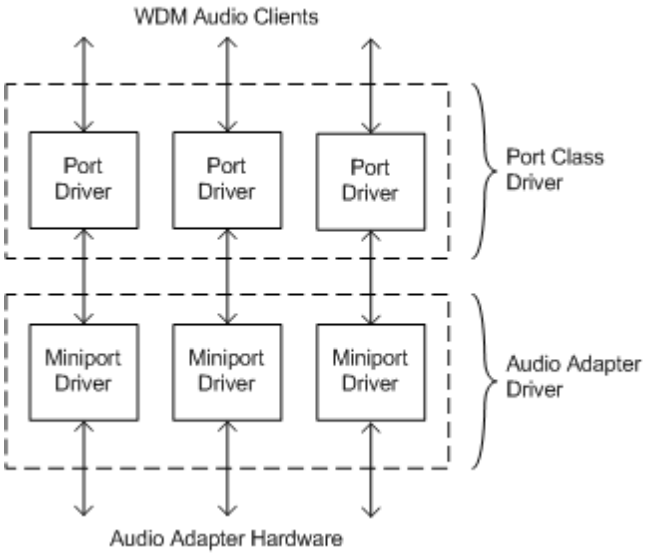

Fig. 4.3 – Struttura dei driver audio di Windows

4.3 Win 32 API

Le API WIN32 comprendono in realtà la maggior parte delle interfacce programmative di Windows, a partire dalla versione Windows95 in quanto è stata la prima versione a supportare l'elaborazione a 32 bit.

Le primitive di I/O nelle Win32 API sono:

Communication Resources

Device fisico o logico che fornisce un data-stream singolo, bidirezionale e asincrono (es. porte seriali, parallele, modem,..)

CreateFile, **CloseHandle**, **ReadFile**, **ReadFileEx**, **WriteFile**, **WriteFileEx**

Device Input and Output Control (POSIX ioctl)

Ad esempio per comunicare direttamente con il Device Driver è possibile chiamare direttamente DeviceIoControl.

Device Management

Notifica alle applicazioni cambianti che possono modificare il loro accesso al device

Image Mastering API

Permette alle applicazioni di mostrare o masterizzare informazioni audio o immagini di dischi su CD-R (o CD-RW)

es. per il CD-DA esiste l'interfaccia **IRedbookDiscMaster** con i seguenti metodi

Method	Description
GetTotalAudioTracks	Returns total tracks staged in the image.
GetTotalAudioBlocks	Returns total audio blocks available on disc.
GetUsedAudioBlocks	Returns total audio blocks staged in image.
GetAvailableAudioTrackBlocks	Returns blocks available for current track.
GetAudioBlockSize	Returns the audio block size in bytes (2352).
CreateAudioTrack	Indicates a new audio track to be staged.
AddAudioTrackBlocks	Adds blocks of audio to track being staged.
CloseAudioTrack	Closes out staging of an audio track.

Removable Storage Manager

Facilita la comunicazione con le applicazioni e librerie

4.4 Media Control Interface (MCI)

MCI permette di creare applicazioni che controllano periferiche audiovisive in modo indipendente dallo specifico device.

4.4.1 Struttura generale

Le periferiche MCI sono driver che forniscono funzionalità indipendenti ai programmi basati su Windows per il controllo di componenti hardware e software multimediali. La rimozione e la reinstallazione di periferiche MCI può consentire di risolvere problemi relativi alla riproduzione di determinati file.

L'interfaccia MCI è presente in Windows fin dalla versione 3.11.

La libreria di questa interfaccia è *winmm.dll* che si trova nella cartella *%windir%\system32* Usando le funzioni di questa libreria è possibile per esempio riprodurre audio, controllare gli attuatori del computer, gestire i codec.

Esistono due sistemi per interagire con MCI: i *Command-Messages* e le *Command-Strings*:

I primi sono formati da costanti e strutture dati, le Command-Strings, invece, sono la versione più user-friendly perché è una traduzione testuale del messaggio (Command-Message).

```
MCIERROR mciSendCommand(
  MCIDEVICEID IDDevice,
  UINT        uMsg,
  DWORD       fdwCommand,
  DWORD_PTR   dwParam
);

MCIERROR mciSendString(
  LPCTSTR lpszCommand,
  LPTSTR lpszReturnString,
  UINT cchReturn,
  HANDLE hwndCallback
);
```

Queste funzioni hanno un valore di ritorno uguale al numero corrispondente ad un eventuale errore di esecuzione. Il valore 0 corrisponde ad un'esecuzione corretta.

Esempi di comandi MCI sono MCI_CAPTURE, MCI_PLAY, MCI_INFO (responsabili rispettivamente di salvare su file un buffer, riprodurre un oggetto multimediale e ottenere informazioni su uno specifico device.)

Per utilizzare i comandi MCI la prassi è costituita da tre passaggi:

- Apertura del device
- Invio dei comandi
- Chiusura del device

Per scendere più nel dettaglio analizziamo un comando di esempio: play

```
wsprintf(
  lpszCommand,
  "play %s %s %s",
  lpszDeviceID,
  lpszPlayFlags,
  lpszFlags
);
```

Il comando play è realizzato tramite una funzione wsprintf che formatta una serie di caratteri e valori in un buffer (il primo parametro: lpszCommand). Il secondo parametro specifica la formattazione della stringa.

Il comando play può essere applicato a: cdaudio, digitalvideo, sequencer, vcr, videodisc, waveaudio

La versione Command-Message di play è MCI_PLAY

```
MCIERROR mciSendCommand(
  MCIDEVICEID wDeviceID,
  MCI_PLAY,
  DWORD dwFlags,
  (DWORD) (LPMCI_PLAY_PARMS ) lpPlay
);
```

MCI_PLAY è l'evidente versione codificata della stringa passata tramite “play”.

LPMCI_PLAY_PARAMS è la struttura dati che specifica i parametri dell'oggetto da riprodurre:

```
typedef struct {
    DWORD_PTR dwCallback;
    DWORD     dwFrom;
    DWORD     dwTo;
} MCI_PLAY_PARMS;
```

I Device riconosciuti da “open” (MCI_OPEN) sono i seguenti: cdaudio, dat, digitalvideo, other, overlay, scanner, sequencer, vcr, videodisc, waveaudio.

4.4.2 Componenti Audio

La componente Audio di MCI si suddivide in *Audio Compression Manager*, *Audio Mixers*, *MIDI* e *Waveform Audio*.

ACM

L'Audio Compression Manager è il framework multimediale che gestisce i codec audio. ACM può anche essere considerata una specifica API perché un codec deve essere conforme all'implicita specifica ACM per funzionare con Windows Multimedia. Microsoft oggi incoraggia l'uso di DirectShow perché ACM non supporta alcune delle tecnologie utilizzate nei

nuovi codec, tra cui il VBR[36].

Le funzionalità che offre ACM sono:

- Compressione e decompressione audio trasparente in tempo di escuzione
- Selezione del formato dei dati audio
- Selezione del filtro dei dati audio
- Conversione di formato dei dati audio
- Filtraggio dei dati audio

Audio Mixers

L'elemento base del mixer di MCI è la *audio line*, che consiste in uno o più canali di dati originati da un'unica sorgente o risorsa di sistema. L'architettura del mixer permette l'instradamento delle varie audio line e la regolazione del volume e di altri effetti.

Musical Instrument Digital Interface

Le API MCI offrono dei buffer e dei servizi temporizzati con time-code MIDI e una interfaccia per la completa gestione del protocollo.

Waveform Audio

Offre un'ampia gamma di funzioni per la gestione dell'audio in formato wave.

PlaySound è la funzione più semplice che si occupa delle riproduzione audio.

```
#include <windows.h>
int main()
{
PlaySound("C:\\Documents and Settings\\Carlo\\Music\\mysound.wav");
}
```

La classe MCIWnd

Si tratta di una classe molto particolare che permette la riproduzione e la gestione dei

[36] Variable Bit Rate: tecnologia utilizzata nella compressione lossy per indicare una quantizzazione variabile del segnale in relazione alle caratteristiche di dinamica e di este in frequenza.

device multimediali ad un livello più semplice ed unificato per i diversi media rispetto a Multimedia Audio, Multimedia Input, Video for Windows,...

È comodo utilizzare questa classe quando le operazioni da compiere non sono complesse.

4.4.3 Esempi Applicativi

Il seguente esempio riproduce la sesta traccia di un CD audio :

```
mciSendString("open cdaudio", lpszReturnString,
            lstrlen(lpszReturnString), NULL);
mciSendString("set cdaudio time format tmsf", lpszReturnString,
            lstrlen(lpszReturnString), NULL);
mciSendString("play cdaudio from 6 to 7", lpszReturnString,
            lstrlen(lpszReturnString), NULL);
mciSendString("close cdaudio", lpszReturnString,
            lstrlen(lpszReturnString), NULL);
```

Il seguente esempio riproduce un file audio wave:

```
mciSendString( "open c:\mmdata\purplefi.wav type waveaudio alias
finch", lpszReturnString, lstrlen(lpszReturnString), NULL);
mciSendString("set finch time format samples", lpszReturnString,
    lstrlen(lpszReturnString), NULL);
mciSendString("play finch from 1 to 10000", lpszReturnString,
    lstrlen(lpszReturnString), NULL);
mciSendString("close finch", lpszReturnString,
    lstrlen(lpszReturnString), NULL);
```

Per avere un player audio che supporti anche file in formato compresso avremo un comando di apertura file:

```
Pcommand = "open \"" + this.url + "\" type mpegvideo alias MediaFile";
mciSendString(Pcommand, null, 0, IntPtr.Zero);
```

un comando di riproduzione:

```
Pcommand = "play MediaFile";
mciSendString(Pcommand, null, 0, IntPtr.Zero);
```

un comando per interrompere la riproduzione (stop):

```
Pcommand = "stop MediaFile";
mciSendString(Pcommand, null, 0, IntPtr.Zero);
```

un comando per fermare la riproduzione (pause):

```
Pcommand = "pause MediaFile";
mciSendString(Pcommand, null, 0, IntPtr.Zero);
```

un comando per riprendere la riproduzione:

```
Pcommand = "resume MediaFile";
mciSendString(Pcommand, null, 0, IntPtr.Zero);
```

un comando per impostare il volume:

```
Pcommand = String.Format("setaudio MediaFile volume to {0}", volume);
mciSendString(Pcommand, null, 0, IntPtr.Zero);
```

e un comando per impostare la posizione di riproduzione:

```
Pcommand = String.Format("seek MediaFile to {0}", Millisecs);
mciSendString(Pcommand, null, 0, IntPtr.Zero);
```

4.5 DirectSound

DirectSound è un componente della libreria DirectX sviluppata da Microsoft che si occupa di gestire l'interazione con le periferiche audio in modo semplice ed efficace. Questa libreria permette sia operazioni di livello medio-alto, come il mixaggio o l'applicazione di filtri già pronti ad un flusso audio, sia la gestione di parametri hardware.

DirectX è composta da un gruppo di oggetti basati su COM[37] che permettono agli sviluppatori di giochi e multimedia di interfacciarsi con l'hardware di un PC basato su Microsoft Windows.

É pensato per astrarre dalle caratteristiche specifiche dell'hardware, dando così la possibilità ai programmatori di concentrarsi sulle caratteristiche multimediali di livello più alto. DirectX è formata da diverse librerie tra cui ricordiamo DirectDraw, DirectPlay, DirectSound, Direct3D e DirectMusic.

Si è scelto ai fini di questa ricerca di approfondire DirectSound come modello di programmazione DirectX in quanto permette di effettuare una grande varietà di operazioni audio ed è più confrontabile alle tecnologie presenti sugli altri sistemi operativi. Direct Music è in grado di gestire anche dati MIDI e di sintetizzare suoni a partire da questi dati. Per maggiori informazioni si rimanda all'appendice “Differenze tra Direct Sound e Direct Music”.

Le caratteristiche principali di DirectSound sono:

- Interfaccia per accedere alle funzioni simili di una grande varietà di periferiche hardware audio.
- Mixaggio a bassa latenza di flussi audio.
- Funzionalità di spazializzazione 3D dell'audio.
- Proprietà per accedere alle funzionalità hardware non supportate direttamente da DirectSound.
- Funzioni di verifica delle caratteristiche audio dell'hardware in uso per permettere

[37] Component Object Model: Interfaccia standard per i componenti software, introdotta da Microsoft nel 1993. È responsabile della comunicazione tra i processi e la creazione dinamica degli oggetti in qualunque linguaggio di programmazione che supporta questa tecnologia.

all'applicazione di impostare i propri parametri in modo da ottimizzare le prestazioni e la qualità.

DirectSound è composta da un livello di astrazione hardware (HAL) ed un livello di emulazione hardware (HEL). HAL è composto prevalentemente dai driver delle periferiche e processa le richieste provenienti dall'oggetto DirectSound. Le caratteristiche e le capacità dell'hardware vengono rese note al sistema attraverso il livello di astrazione hardware. Se non è presente un driver adeguato oppure non sono supportate determinate operazioni in hardware, il livello di emulazione provvederà a fornire il supporto per quelle operazioni attraverso le funzioni di editing della forma d'onda di Windows Multimedia. L'accelerazione hardware è inversamente proporzionale all'uso di HEL in quanto quest'ultimo è nato proprio per gestire le situazioni in cui non è possibile sfruttare l'accelerazione hardware.

DirectSound utilizza dei buffer per gestire i dati audio.
Il formato di codifica utilizzato è il PCM e i parametri di default sono:

- quantizzazione a 8bit
- campionamento a 22050 Hz
- codifica di tipo lineare stereo interfogliato

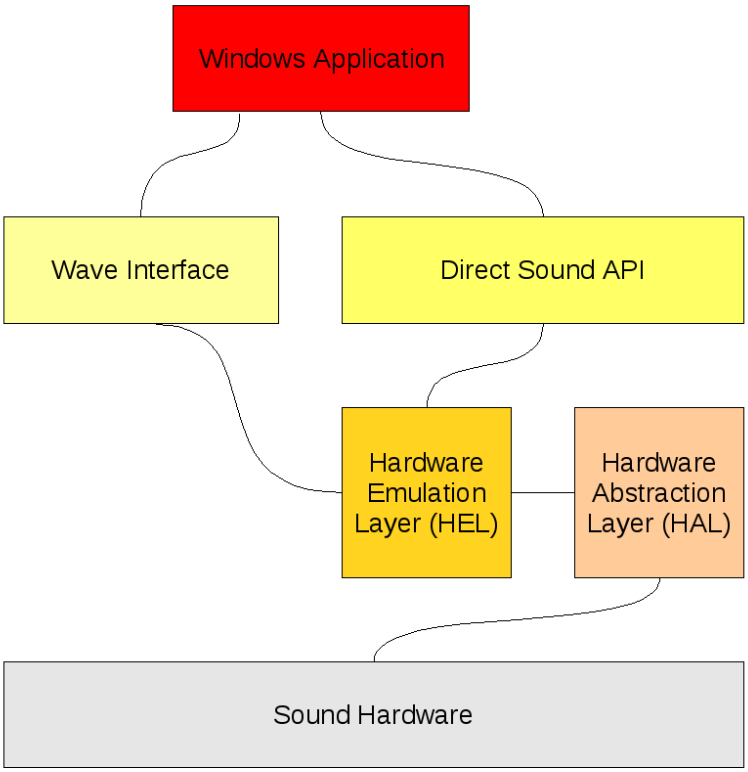

Fig. 4.4 - Interazioni tra le tecnologie usate in un applicativo DirectSound

I diversi tipi di oggetti che vengono utilizzati in DirectSound sono: *Device*, *PrimaryBuffer*, *SecondaryBuffer*, *Effect*.

Sono presenti in ogni applicazione un solo *Device* ed un solo *Primary Buffer*, mentre esiste un *Secondary Buffer* per ogni suono ed esso può avere associati zero o più *Effect*.

Il *Primary Buffer* supporta le funzionalità di mixaggio degli altri buffer e di riproduzione. I *Secondary Buffer* vengono utilizzati per il processing e le altre operazioni audio e possono essere di due tipi diversi: statici oppure streaming.

Un esempio di applicativo che utilizza DirectSound per la riproduzione di file wave avrà bisogno di:

definire un oggetto di tipo Buffer (si intende PrimaryBuffer):

```
protected Microsoft.DirectX.DirectSound.Buffer mySound;
```

definire le impostazioni del buffer:

```
BufferDescription desc = new BufferDescription();
            desc.CanGetCurrentPosition = true;
            desc.ControlFrequency = true;
            desc.ControlPan = true;
            desc.ControlVolume = true;
            desc.PrimaryBuffer = false;
            desc.GlobalFocus = true;
```

ed istanziarlo a partire dall'oggetto audio corrispondente al file il cui percorso è contenuto nella stringa filesrc:

```
 mySound = new Microsoft.DirectX.DirectSound.Buffer(filesrc, desc,
dev.Device);
```

Sarà possibile poi controllare attraverso delle semplici varibili le impostazioni di volume, panning, frequenza di lettura del Buffer[38] e posizione di lettura del file:

```
public int Volume {
            get { return mySound.Volume; }
            set { mySound.Volume = value; }
        }
public int Frequency {
            get { return mySound.Frequency; }
            set { mySound.Frequency = value; }
        }
```

[38] Come abbiamo già detto il default è 22,05 kHz

```
public int Pan {
            get { return mySound.Pan; }
            set { mySound.Pan = value; }
        }

public int Position {
            get { return mySound.PlayPosition; }
            set { mySound.SetCurrentPosition(value); }
        }
```

4.6 ASIO

Audio Stream Input Output è una tecnologia non-Microsoft, in quanto sviluppata dalla Steinberg, che fornisce un'interfaccia a bassa latenza ed alta fedeltà tra le applicazioni e le schede audio.

Il target di uso di questa tecnologia è differente rispetto a quello di DirectSound in quanto la prima è sviluppata per un'utenza di musicisti ed ingegneri del suono, mentre la seconda per un'utenza prevalentemente non professionale.

L'idea di base di questa tecnologia è quella di evitare i livelli intermedi di processing di Windows (quali ad esempio il KernelMixer) e in questo modo abbassare la latenza e migliorare le prestazioni.

Questa tecnologia è stata sviluppata solamente per Windows perché gli altri sistemi operativi non hanno gli stessi problemi di latenza, avendo già implementato uno stack audio più performante.

4.7 Windows Vista e le sue novità

4.7.1 Architettura

Windows Vista presenta moltissime innovazioni per quanto riguarda l'audio, sia per gli utenti (vedi appendice 3) che per i programmatori.

La grande innovazione di Windows Vista dal punto di vista audio è che tutto lo stack, ad eccezione dei driver audio è implementato in user space.

Ne deriva il fatto che non è più possibile sfruttare l'accelerazione hardware permessa nelle precedenti versioni di Windows da DirectSound[39].

Lo spostamento di una così grande parte dello stack audio (anche il kernel mixer non è più presente, sostituito da un componente in user space) nello spazio utente porterebbe un forte degrado delle prestazioni se non fosse presente una componente migliorativa che riduce notevolmente la copia dei dati in memoria: è permesso dal livello utente scrivere nel buffer DMA tramite apposite chiamate di sistema.

Questo determina la suddivisione in due modalità possibili di accesso all'audio di Vista: accesso diretto al buffer DMA dall'applicazione utente (caso tipico dei software professionali) e accesso tramite delle sessioni che vengono programmate attraverso le Windows Audio Session API.

Le due modalità sono:

- **Exclusive Mode** (DMA mode): i flussi di dati audio vengono inviati al driver in modo diretto e non mixato. Questa modalità permette un'alta efficienza, attraverso la minore mediazione software possibile. Non è possibile effettuare elaborazione del segnale. È possibile inviare audio in formato compresso tramite connessioni S/PDIF a device esterni che supportino Dolby Digital, DTS, o WMA Pro.

- **Shared Mode**: i flussi audio vengono processati dall'applicazione. È possibile inserire degli effetti pre-mix (LFX = Local Effects), effettuare un mixaggio via sofware e poi applicare degli effetti prima dell'output (GFX = Global Effects).

[39] Direct Sound viene emulato tramite una sessione WASAPI

La figura 4.5 è una semplificazione dello scenario reale per mostrare la distinzione tra le due modalità di utilizzo dell'audio in Windows Vista.

È ben visibile come a sinistra venga applicato tutto lo stack audio delle Windows Audio Session API (WASAPI) attraverso un uso shared del sistema.

ASIO e Open AL utilizzano il sistema in Exclusive Mode, evitando completamente la complessità introdotta dalle sessioni e da tutti i passaggi intermedi, ottenendo così una latenza molto minore.

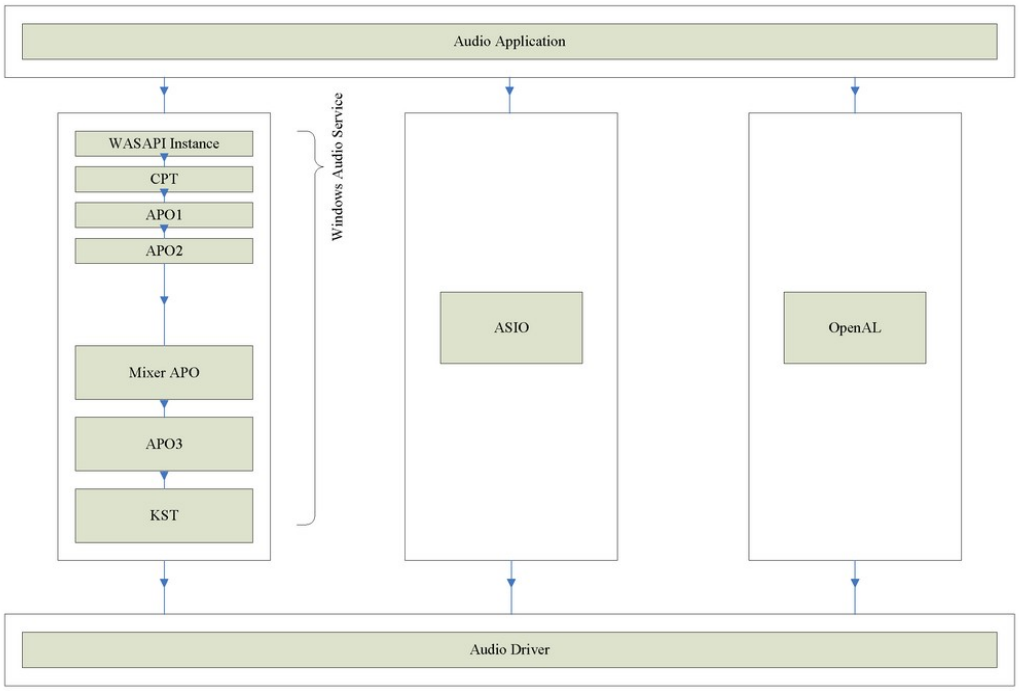

Fig. 4.5 – Windows Vista Audio Stack semplificato

4.7.2 API

In Windows Vista è possibile utilizzare una grande varietà di API per fare programmazione Audio, in quanto vengono introdotte nuove tecnologie ma rimangono comunque disponibili gran parte delle API precedenti, che tuttavia hanno un comportamento differente perché vengono emulate tramite delle sessioni.

È quindi possibile usare le seguenti API tramite sessioni: DirectX (Direct Sound, Direct3D, DirectMusic, DirectShow,..), Windows Multimedia (MCI), Xaudio2, XACT3, XAPO.

Per utilizzare in modo nativo il sistema audio di Windows Vista senza le sessioni (Exclusive Mode) sono state introdotte delle nuove API, con un nome che richiama il mondo Apple: Windows Core Audio API.

In realtà il nome Core Audio API comprende anche le WASAPI, tuttavia è spesso inteso come interfaccia per l'Exclusive Mode.

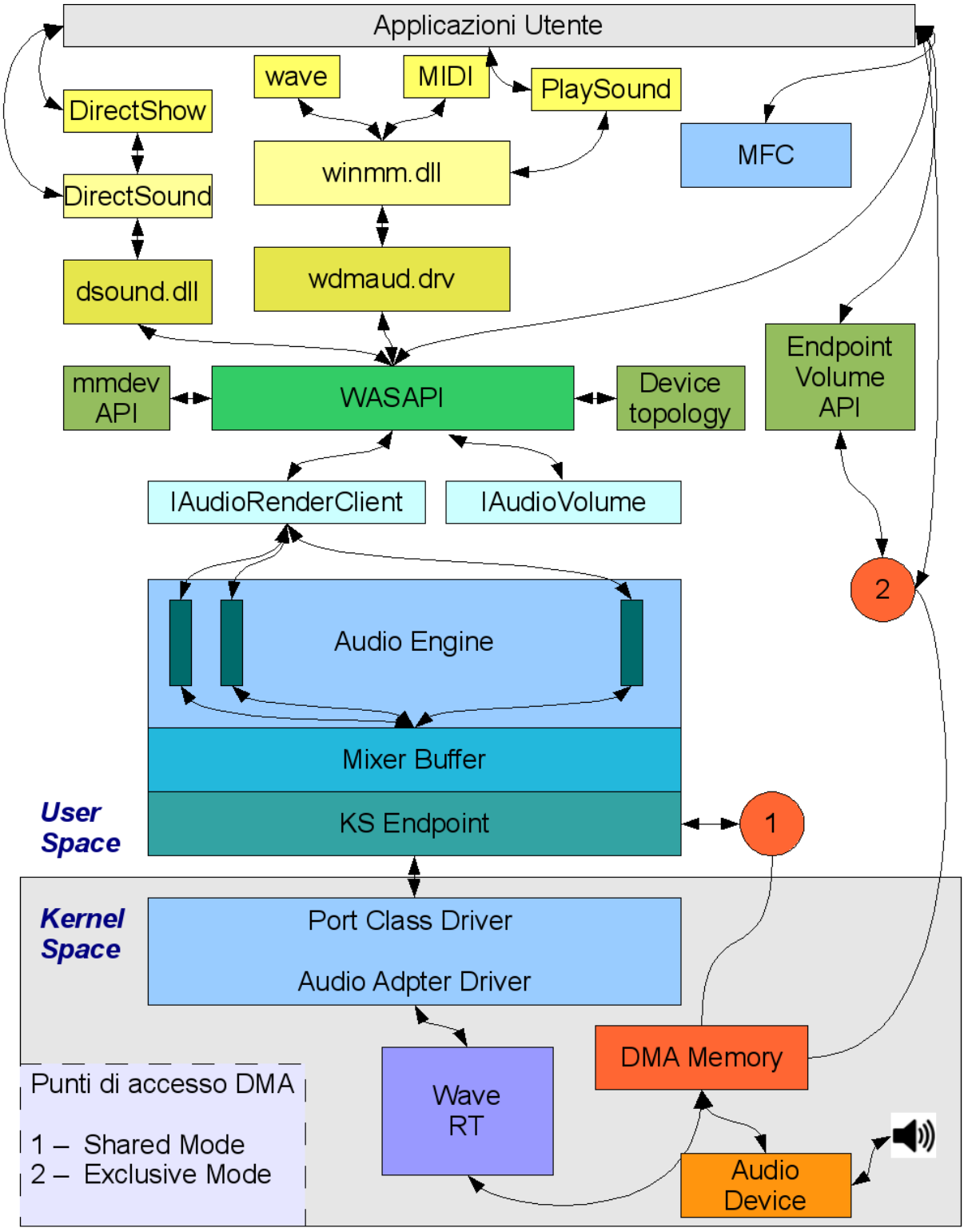

Fig. 4.6 – Audio in Windows Vista

Le Windows Core Audio API vengono utilizzate generalmente solo dalle applicazioni professionali che hanno bisogno performance elevate e di un grande controllo del segnale audio a basso livello.

Le Windows Core Audio API si suddividono come segue:

- **MultiMedia Device (MMDevice) API:** per enumerare gli endpoint ed i device nel sistema.
- **Windows Audio Session API (WASAPI):** per gestire l'audio tramite le sessioni
- **Device Topology API:** per accedere direttamente a caratteristiche quali il controllo dei volumi e del mixer.
- **Endpoint Volume API:** utilizzata in Exclusive mode per regolare i volumi della periferica.

4.7.3 XAUDIO2 e XACT3

Microsoft ha recentemente sviluppato una libreria che andrà a sostituire DirectSound: Xaudio2.

Si tratta di una libreria audio a basso livello cross-platform (relativamente alle piattaforme di gaming Microsoft: Windows e Xbox360). La libreria è stata rilasciata a marzo 2008 insieme al nuovo DirectX SDK. Le funzioni di mixing a basso livello e di elaborazione del segnale vengono realizzate tramite le API Xaudio2, mentre le funzioni più ad alto livello come la riproduzione e la spazializzazione sono incluse nelle API XACT e X3DAudio.

Le nuove funzionalità di Xaudio sono:

- Separazione dei dati sonori generici dalla “voce”
- Submixing con un numero arbitrario di livelli e di instradamenti
- Processing con molteplici frequenze di campionamento
- Filtraggio e aggiunta di effetti sui singoli canali audio
- Sintetizzatori Software programmabili
- Conversione di frequenza di campionamento on-the-fly
- Elaborazione del segnale software
- Supporto nativo e scalabile di formati compressi (XMA, ADPCM, xWMA)
- Gestione separata della spazializzazione del suono (tramite X3DAudio)

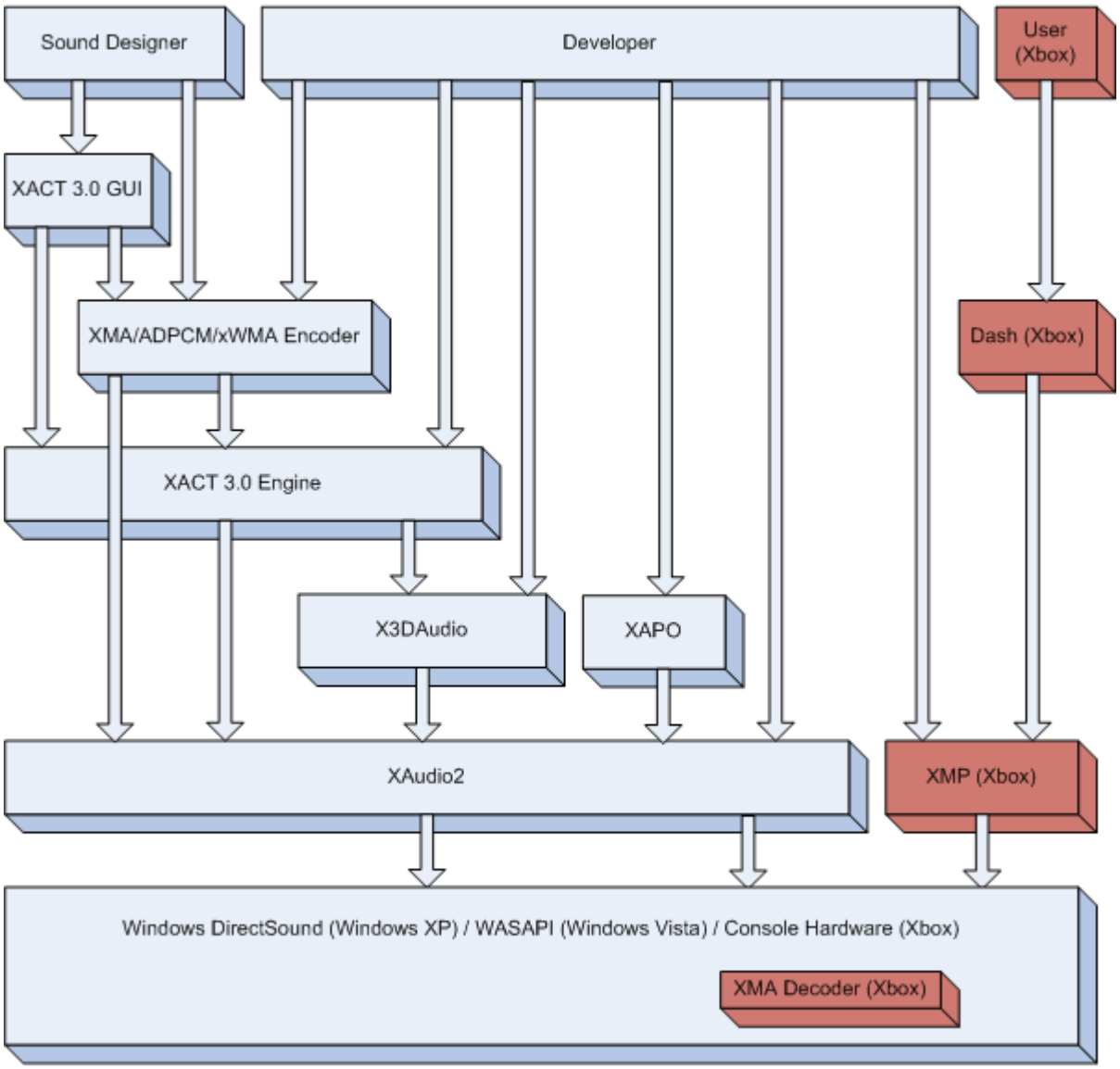

Fig. 4.7 - Nuove tecnologie audio per Windows e Xbox

4.7.4 UNIVERSAL AUDIO ARCHITECTURE

Si tratta di un'iniziativa promossa dal 2002 dalla Microsoft per la standardizzazione dei driver audio e pienamente implementata solo a partire dal 2006 con il miniport WaveRT di Windows Vista. L'obiettivo di UAA è quello di risolvere molti problemi relativi al proliferare dei più svariati driver audio diversi per periferiche audio di tipo consumer. A causa della mancanza di un'interfaccia comune per descrivere e controllare le molteplici capacità delle periferiche audio i produttori di hardware (es. Creative Labs, Realtek, Turtle Beach) hanno dovuto sviluppare applicativi di controllo ed interfacce specifiche.

Il fatto che i driver vengano eseguiti in kernel-mode determina problemi di stabilità del sistema nel caso in cui i driver stessi non siano sviluppati con particolare cura.

Questi problemi hanno determinato la scelta della Microsoft di disabilitare di default lo stack audio nei sistemi Windows 2003 Server.

UAA fornisce un'interfaccia unica per i device audio, in modo da non aver bisogno di driver aggiuntivi. In aggiunta assicura una compatibilità dei device UAA-compliant per le future versioni di Windows, perché non sono necessari i driver del produttore.

Un altro obiettivo di UAA è quello di fornire un migliore supporto per l'audio multicanale, ad esempio i flussi WMA Pro possono essere riprodotti senza speciali driver.

UAA supporta tre classi di device: USB, IEEE1394 e Intel High Definition Audio[40] (PCI e PCI Express).

Per raggiungere performance migliori o per sfruttare tecnologie più professionali vengono utilizzati driver proprietari, ad esempio sfruttando la tecnologia ASIO.

[40] Si rimanda a [90].

5. Conclusioni

Il confronto delle delle diverse tecnologie in molti casi risulta complesso, perché si tratta di librerie audio molto diverse tra di loro e soprattutto collocate in sistemi operativi architetturalmente diversi.

Per questo motivo è importante definire alcuni semplici parametri di confronto: le funzionalità offerte, i linguaggi di programmazione disponibili per le API, la latenza, la portabilità.

5.1 Applicativi realizzati

Una delle difficoltà incontrate nello sviluppo delle applicazioni nelle varie piattaforme è stata quella di comprendere l'organizzazione delle API e di apprendere l'utilizzo dei diversi ambienti di sviluppo.

Mostriamo di seguito una tabella riepilogativa degli esempi applicativi realizzati ai fini di comprendere meglio le tecnologie discusse in questo testo.

	Nome	Tecnologia	SO	Linguaggio	Difficoltà incontrata	Numero righe
1	synth_c	OSS	Linux	C	Bassa	185
2	synth_alsa	ALSA	Linux	C	Bassa	136
3	synth_mt	ALSA	Linux	C	Media	211
4	Wave	HAL/Core Audio	Mac OS	ObjC	Medio-Alta	201
5	Player	CoreAudio	Mac OS	ObjC	Medio-Alta	163
6	CarloUnit	CoreAudio AU framework	Mac OS	C++	Media	696
7	MCIPlayer	MME	Windows	C#	Medio-Bassa	671
8	DXPlayer	Direct Sound	Windows	C#	Media	556

Tabella 5.1 - Applicativi realizzati

Gli esempi applicativi in ambito Linux sono molto semplici e consistono in applicazioni a linea di comando che effettuano una sintesi di una sinusoide di frequenza determinata da un parametro a linea di comando passato dall'utente. Opzionalmente l'utente può indicare se sintetizzare anche un certo numero di armoniche della frequenza fondamentale ed eventualmente se applicare a queste una pesatura esponenziale in ampiezza.

Il terzo applicativo permette la sintesi in multithreading di più sinusoidi contemporaneamente, che verranno poi mixate dal modulo dmix.

In ambiente Mac OS la difficoltà iniziale è stata molto maggiore, soprattutto a causa del linguaggio Objective-C sul quale è basata gran parte della documentazione Apple.

Abbiamo realizzato un sintetizzatore monofonico che si basa sulle API del livello di astrazione hardware, un player audio e un modulo plug-in che permetta di effettuare processing

del segnale audio in qualunque applicazione che supporti Audio Unit.

Su Windows abbiamo sviluppato due diversi player per confrontare le tecnologie MultiMedia Extensions e Direct Sound. Si è riscontrato che MME risulta più semplice da utilizzare perchè le API sono di livello più alto. Direct Sound risulta di fatto molto più potente e flessibile, in quanto si ha il controllo diretto sui buffer applicativi. Con Direct Sound è stato possibile aggiungere al player la funzionalità di modifica in termpo reale della frequenza di lettura del buffer, ottenendo così un effetto di variazione della velocità del brano riprodotto e di corrispondente pitch-shift[41].

Abbiamo inoltre schematizzato i diversi sistemi per un semplice confronto in appendice. Per fare questo siamo dovuti scendere a dei compromessi per quanto riguarda la precisione e la completezza a vantaggio di una migliore chiarezza.

5.2 Confronto

Nella tabella 5.2 troviamo un confronto tra le caratteristiche offerte dalle diverse tecnologie: abbiamo definito 6 categorie di servizi offerti al programmatore:

Gestione Hardware della periferica, attraverso la modifica dei parametri della stessa

Driver: la possibilità costruirne attraverso apposite API oppure la presenza degli stessi all'interno del pacchetto kernel della specifica tecnologia.

Conversioni: lo strato che rende la tecnologia Device Independent effettuando in automatico le conversioni di formato tra i dati dell'applicazione e i dati della periferica.

Mix: la capacità di effettuare un mixaggio di molteplici flussi audio di processi distinti. Senza l'intervento del programmatore.

DSP: funzionalità di elaborazione del segnale (es. applicazione di filtri)

Funzionalità avanzate: la gestione del Surround, delle interfacce di rete e il routing dei flussi

[41] Il Pitch Shift è quell'effetto per il quale varia l'altezza dei suoni che vengono riprodotti.

audio tra le applicazioni.

	OSS	**ALSA**	**JACK**	**Pulse Audio**	**WIN 32 API**	**MME**	**Direct Sound**	**Xaudio 2**	**XACT 3**	**ASIO**	**Core Audio**	**HAL I/O kit**
Funzionalità avanzate			x	x			x		x		x	
DSP			x	x			x	x	x		x	
Mix		°	x	x		x	x	x	x		x	
Conversioni	x	x				x	x	x		x		x
Driver	x	x			x					x		x
Gestione Hardware	x	°°			x		x[42]			x		

Tabella 5.2 – Funzionalità offerte dalle diverse tecnlogie

Legenda

x → funzionalità presente

° → funzionalità disponibile tramite il modulo dmix

°° → funzionalità presente attraverso le API Full ALSA

[42] È importante sottolineare che le tecnologie Windows hanno comportamenti diversi in relazione alla versione del sitema operativo sul quale vengono utilizzate: la gestione dell'Hardware non è possibile tramite Windows Vista. Il Mix avviene in realtà attraverso il KernelMixer o il Global Audio Environment e non è compito specifico né di DS né di Xaudio2, anche se entrambe le tecnologie permettono un mixaggio esplicito di più buffer audio.

5.2.1 Linguaggi di programmazione

La tabella 5.2 mostra il supporto per i vari linguaggi nelle API delle diverse tecnologie.

	OSS	ALSA	JACK	Pulse Audio	WIN 32 API	MME	Direct Sound	Xaudio 2	XACT 3	ASIO	Core Audio	HAL I/O kit
C	x	x	x	x	x	x	x	x	x	x	x	x
C++	x	x	x	x	x	x	x	x	x	x	x	x
C#						x	x	x	x			
ObjC											x	x

Tabella 5.3 – Supporto dei diversi linguaggi di programmazione

Esistono dei wrapper delle librerie C per Java di alcune tecnologie come Jack (JJack) la cui documentazione risulta tuttavia molto scarsa. È possibile comunque utilizzare Java attraverso la JNI per qualunque libreria C.

Altri linguaggi come Ruby e Python risultano molto poco documentati riguardo all'utilizzo di tecnologie audio, per cui abbiamo deciso di non considerarli importanti ai fini di questa ricerca.

5.2.2 Latenza

La latenza è sicuramente uno dei fattori più importanti per valutare la qualità di una tecnologia audio.

La figura 5.1 mostra le latenze degli esperimenti di [91], le cui tabelle complete sono riportate in appendice.

Si nota che CoreAudio ha delle prestazioni mediamente molto migliori degli altri sistemi presi in esame. ALSA risulta certamente una alternativa molto valida, seguita da ASIO: questi sistemi permettono latenze medie anche inferiori ai 5 ms.

Dalle ricerche [91] appare che nelle piattaforme Macintosh e Linux non vi sia sostanziale differenza tra le misurazioni con e senza carico, mentre invece su Windows 2000 la situazione è notevolmente diversa, in quanto con carico la latenza può aumentare anche di quattro volte.

Le misure con il carico sono state effettuate facendo eseguire altri due programmi per occupare risorse di CPU e di uso dell'hard-disk. Le simulazioni di carico permettono

generalmente una visione più realistica della situazione di utilizzo di software audio, perché spesso quest'ultimo si avvale dell'utilizzo di processori di segnale sofware o di sintetizzatori che aumentano il carico del sistema.

System	Operating System	Details	Latency (ms)
Spirit	-	-	1.81
A	Linux 2.4.1 (AM)	ALSA	2.72
A	Linux 2.4.1	ALSA	2.72
C	MacOS X	CoreAudio	2.83
F	Windows 2000	ASIO	3.11
F	Linux 2.4.1 (IM)	ALSA (L)	4.30
F	Linux 2.4.5 (AM)	ALSA (L)	4.30
F	Linux 2.4.2	ALSA (L)	4.30
G	MacOS 9.04	ASIO	6.80
A	Linux 2.2.16	ALSA	7.19
F	Windows 2000	MME (P)	11.45
E	Linux 2.4.0	OSS	12.20
E	Windows 98	MME (P)	60.86
E	Windows 98	DirectSound (P)	63.24
D	Windows ME	MME (P)	73.51
A	Windows 2000	MME (P)	75.85
D	Windows ME	DirectSound (P)	82.86
B	MacOS 8.6	SM (P)	106.24
A	Windows 2000	DirectSound (P)	123.11
C	MacOS 9.04	SM (P) VM Off	195.58
C	MacOS 9.04	SM (P) VM On	935.53

Table 4. Latency Test results for systems without load.
P – PortAudio. L – LAAGA. VM – MacOS virtual memory settings. AM – Andrew Morton. IM – Ingo Molnar.

System	Operating System	Details	Latency (ms)
C	MacOS X	CoreAudio	2.83
F	Linux 2.4.1 (IM)	ALSA (L)	4.30
F	Linux 2.4.5 (AM)	ALSA (L)	4.30
F	Linux 2.4.2	ALSA (L)	4.30
F	Windows 2000	ASIO	6.03
G	MacOS 9.04	ASIO	6.80
F	Windows 2000	MME (P)	245.17

Table 5 – Latency Test Results with system load.
P – PortAudio. L – LAAGA.
AM – Andrew Morton. IM – Ingo Molnar.

Fig 5.1 – Confronto Latenza

È importante tenere conto che le ricerche di [91] sono state effettuate nel 2001 e sicuramente bisogna tenere in considerazione un forte miglioramento dell'hardware in questi sette anni, tuttavia, gli stack applicativi dei diversi sistemi risultano confrontabili[43]: si presume

[43] Un caso a parte è quello di Windows Vista che tuttavia offre come supporto alle applicazioni uno stack lento in user mode. L'utilizzo della modalità Exclusive presuppone la rinuncia a tutte le tecnologie Microsoft di

pertanto che anche le proporzioni tra le differenze di latenza nei vari sistemi siano sostanzialmente invariate.

Risulta evidente come alcuni sistemi non siano indicati per un utilizzo professionale in quanto hanno una latenza anche di diversi ordini di grandezza maggiore.

ALSA, CoreAudio e ASIO risultano quindi le scelte più performanti per i corrispondenti sistemi operativi in quanto con queste API è possibile ottenere dei ritardi inferiori ai 4-5ms.

5.3 Cenni sulla Portabilità

Le API analizzate in questa ricerca sono strettamente legate alla piattaforma per la quale sono sviluppate, per cui risulta sostanzialmente impossibile portare in modo semplice e veloce il codice sviluppato per una piattaforma su un'altra. Questo è anche uno dei motivi principali per cui molti software audio anche professionali sono stati realizzati per una piattaforma soltanto. Tuttavia esistono delle alternative che possono rendere gli applicativi piu' facilmente portabili su altri sistemi.

Pulse Audio è stato rilasciato sia per Windows che per Linux, Solaris e FreeBSD e le sue API possono pertanto essere utilizzate su tutte le suddette piattaforme in modo equivalente.[44]

Jack è presente sia nella versione Linux che in quella OSX.

ASIO è uno standard utilizzabile anche in ambiente OSX, tuttavia le prestazioni di CoreAudio fanno in modo che l'utilizzo di questa tecnologia non sia necessario.

OpenAL è uno standard per la gestione efficiente ed avanzata dell'audio posizionale a tre dimensioni. Le sue API ricalcano deliberatamente quelle delle OpenGL.

sviluppo Audio.

[44] Spesso anche l'aspetto di interfaccia grafica necessita di una cura particolare per garantire la portabilità.

Appendice 1

Schemi riassuntivi delle architetture audio

I seguenti schemi sono delle approssimazioni dell'architettura reale e delle possibili configurazioni degli stack audio nei sistemi operativi trattati in questa ricerca.

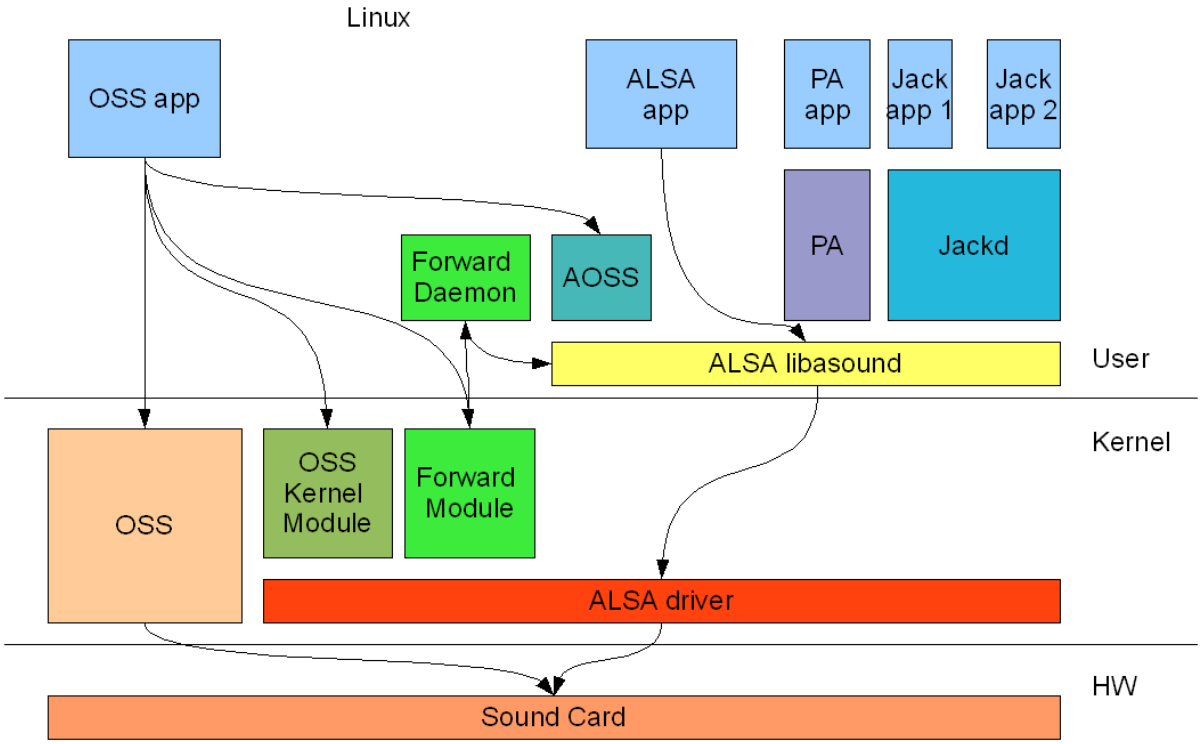

Fig. A1.1 - Schema riassuntivo dell'architettura audio di Linux

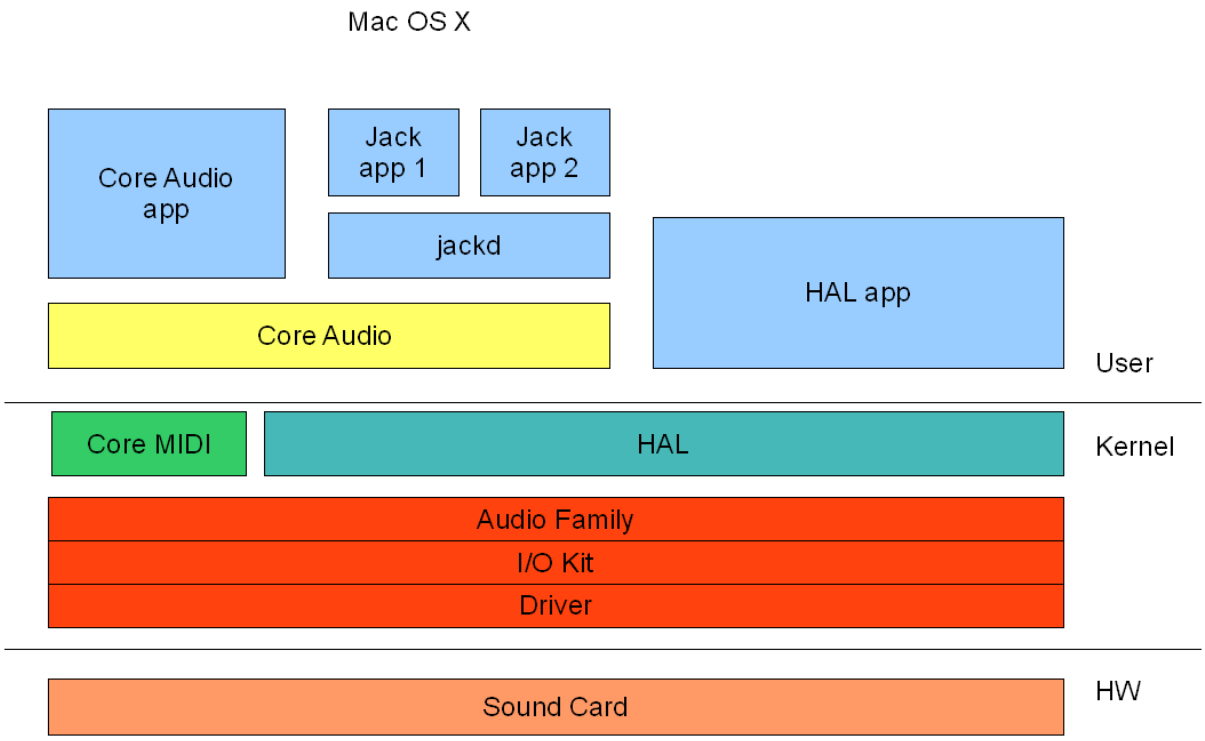

Fig. A1.2 – Schema riassuntivo dell'architettura audio di Mac OS X

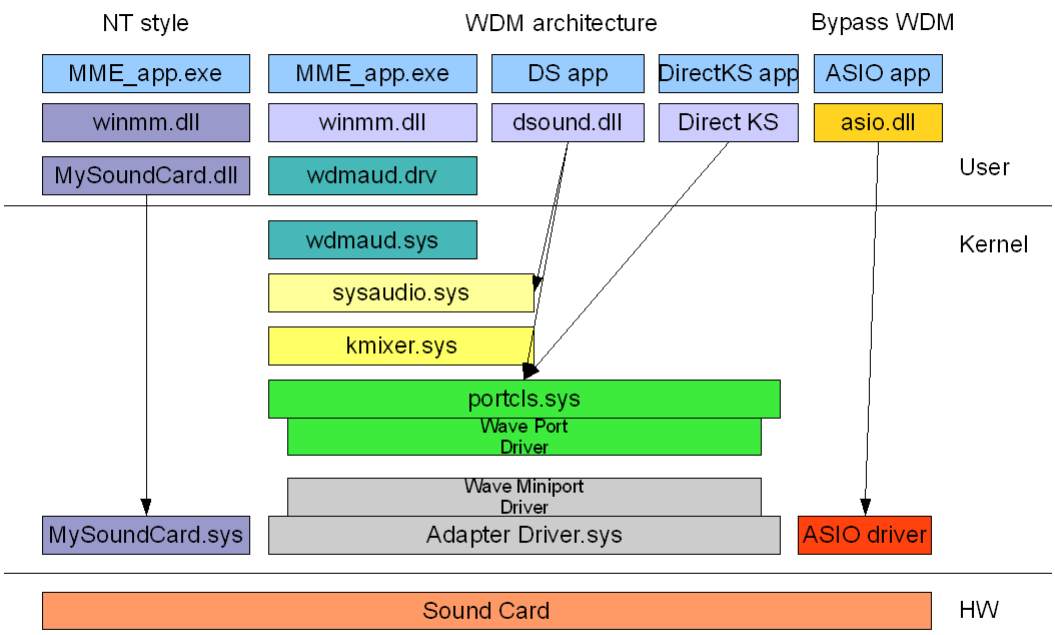

Fig. A1.3 – Schema riassuntivo delle architetture Audio Legacy di Windows

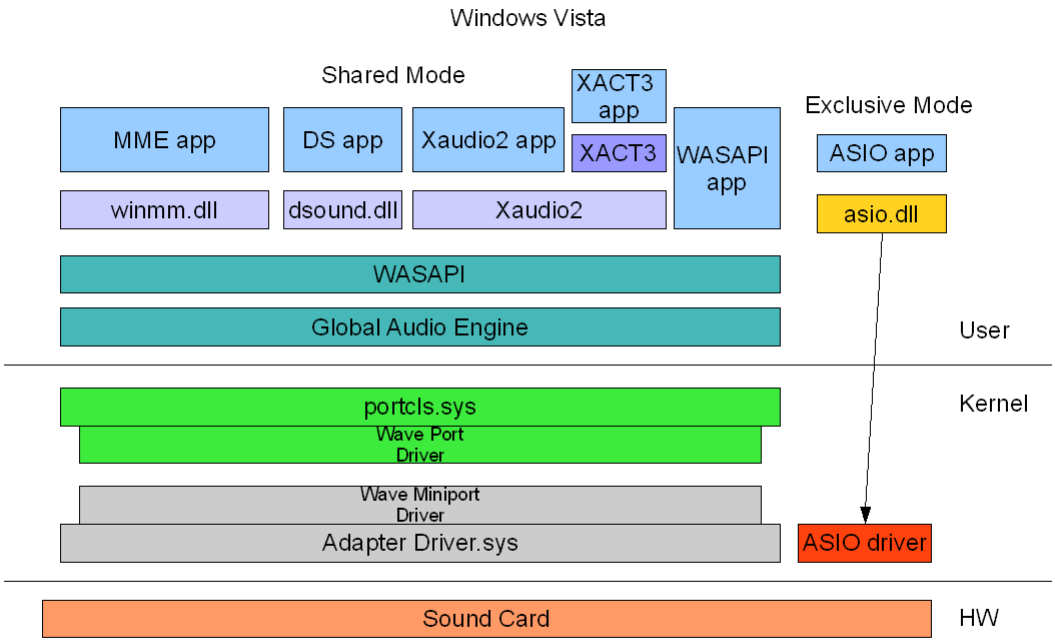

Fig. A1.4 – Schema riassuntivo dell'architettura audio di Windows Vista

Appendice 2

Differenze tra Direct Sound e Direct Music

La tabella qui riportata è tratta da [70].

Functionality	Direct Music	Direct Sound
Play WAV sounds	Yes	Yes
Play MIDI	Yes	No
Play DirectMusic Producer segments	Yes	No
Load content files and manage objects	Yes	No
Control musical parameters at run-time	Yes	No
Manage timeline for cueing sounds	Yes	No
Use downloadable sounds (DLS)	Yes	No
Set Volume, Pitch and Pan of individual sounds	Yes, through Direct Sound API	Yes
Set Volume on multiple sounds (audiopaths)	Yes	No
Implement 3D sounds	Yes, through Direct Sound API	Yes
Apply effects (DMOs)	Yes, through DirectMusic Producer content or Direct Sound API	Yes

Chain buffers for mix-in (send) effects	Yes, through DirectMusic Producer content	No
Capture WAV sounds	No	Yes
Implement Full Duplex	No	Yes
Capture MIDI	Yes	No
Control Allocation of Hardware Buffers	No	Yes

Appendice 3

Nuove funzionalità audio offerte agli utenti di Windows Vista

Queste funzionalità sono tratte da [58].

- **Loudness Equalization DSP**: per mantenere la stessa intensità audio percepita nel passaggio tra suoni diversi.
- **Forward Bass Management**: offre un crossover per la gestione di un eventuale subwoofer
- **Reverse Bass Management**: ridistribuzione del canale LFE in audio multicanale in assenza di subwoofer
- **Low Frequency Protection**
- **Speaker Fill (SF)**: ridistribuzione in impianti multicanale di suoni stereo.
- **Room Correction**: calibrazione dell'equalizzazione in relazione alla risposta all'impulso della stanza in cui ci si trova.
- **Virtual Surround (VS)**: per permettere la comunicazione di segnali audio multicanale tramite una periferica stereo ad impianti *Surround enhancement*
- **Channel Phantoming (CP)**: ottimizzazione di suoni surround su un impianto 2.1
- **Virtualized Surround Sound over Headphones:** spazializzazione del suono tramite HRTF[45]
- **Bass Boost**
- **Noise Suppression**
- **Automatic Gain Control**
- **Voice Activity Detection**

[45] Head Related Transfer Function

Appendice 4

Tabelle della ricerca [91]

System	CPU	Speed (MHz)	Memory (MB)	Disk	Soundcard
A	PIII	933	256	SCSI	1
B	G3	400	128	SCSI	2
C	G4	400	128	IDE	2
D	PIII	700	128	IDE	3
E	Celeron	366	192	IDE	4
F	PIII(dual)	933	512	SCSI	5
G	G4	500	256	IDE	6

Table 1. Hardware Details of the Test Systems.

Soundcard	Brand	Model	Grade
1	Crystal	CS4614	Consumer
2	Apple	Built-in	Consumer
3	ESS	Maestro 3	Consumer
4	ESS	Solo-1	Consumer
5	RME	Hammerfall (9652)	Professional
6	MOTU	2408	Professional

Table 2. Soundcard Details.

API	Platforms	Standard
Microsoft DirectSound and DirectSoundCapture	Windows 98, ME, 2000	No
Microsoft Multimedia Extensions (MME)	Windows 98, ME, 2000	Yes
Steinberg Audio Stream Input Output (ASIO) 2.0	Windows 98, ME, 2000, MacOS 8 and 9	No
Apple SoundManager (SM)	MacOS 8, 9	Yes
Apple CoreAudio	MacOS X	Yes
Open Sound System (OSS)	Linux 2.2, 2.4	Yes
Advanced Linux Sound Architecture (ALSA)	Linux 2.2, 2.4	No

Table 3. Common Audio APIs.

System	Operating System	Details	Latency (ms)
Spirit	-	-	1.81
A	Linux 2.4.1 (AM)	ALSA	2.72
A	Linux 2.4.1	ALSA	2.72
C	MacOS X	CoreAudio	2.83
F	Windows 2000	ASIO	3.11
F	Linux 2.4.1 (IM)	ALSA (L)	4.30
F	Linux 2.4.5 (AM)	ALSA (L)	4.30
F	Linux 2.4.2	ALSA (L)	4.30
G	MacOS 9.04	ASIO	6.80
A	Linux 2.2.16	ALSA	7.19
F	Windows 2000	MME (P)	11.45
E	Linux 2.4.0	OSS	12.20
E	Windows 98	MME (P)	60.86
E	Windows 98	DirectSound (P)	63.24
D	Windows ME	MME (P)	73.51
A	Windows 2000	MME (P)	75.85
D	Windows ME	DirectSound (P)	82.86
B	MacOS 8.6	SM (P)	106.24
A	Windows 2000	DirectSound (P)	123.11
C	MacOS 9.04	SM (P) VM Off	195.58
C	MacOS 9.04	SM (P) VM On	935.53

Table 4. Latency Test results for systems without load.
P – PortAudio. L – LAAGA. VM – MacOS virtual memory settings. AM – Andrew Morton. IM – Ingo Molnar.

System	Operating System	Details	Latency (ms)
C	MacOS X	CoreAudio	2.83
F	Linux 2.4.1 (IM)	ALSA (L)	4.30
F	Linux 2.4.5 (AM)	ALSA (L)	4.30
F	Linux 2.4.2	ALSA (L)	4.30
F	Windows 2000	ASIO	6.03
G	MacOS 9.04	ASIO	6.80
F	Windows 2000	MME (P)	245.17

Table 5 – Latency Test Results with system load.
P – PortAudio. L – LAAGA.
AM – Andrew Morton. IM – Ingo Molnar.

Indice delle Illustrazioni

Bibliografia

Programmazione:

[1] Deitel H.M., Deitel P.J. (2001), C++ Fondamenti di Programmazione, Apogeo

[2] Eckel Bruce (2000), Thinking in C++, 2nd Ed. Vol. 1, Upper Saddle River (NJ), Person Prentice Hall, <http://www.silab.dsi.unimi.it/manual/Thinking%20in%20C++,%202nd%20ed.%20Volume%201/>

[3] Kernighan B.W., Ritchie D.M. (1989), Linguaggio C, seconda edizione, Milano, Gruppo Editoriale Jackson

[4] Mayo Joe (2000), The C# Station Tutorial, <http://www.csharp-station.com/Tutorials/Lesson01.aspx>

[5] O'Tierney Tristan, Objective-C Beginner's Guide, <http://www.otierney.net/objective-c.html>

[6] Prezl F. (2006), Laboratorio di Calcolo II – materiale del corso, <http://labmaster.mi.infn.it/Laboratorio2/serale/>

[7] Solinas Marco (2004), Programmazione Concorrente, <http://www.gameprog.it/pub/IDS/MarcoAllanonSolinas/ProgConcorr1.pdf>

Sistemi Operativi e gestione dei device:

[8] A.A.V.V. (2008), Interrupt <http://en.wikipedia.org/wiki/Interrupt>

[9] Dijkstra Edsger W. , My recollections of operating system design (EWD1303),

<http://www.cs.utexas.edu/users/EWD/ewd13xx/EWD1303.PDF>

[10]Poettering L. (2008), What's Cooking in PulseAudio's glitch-free Branch, <http://www.0pointer.de/blog/projects/pulse-glitch-free.html>

[11]Tanenbaum Andrew S. (2001), Modern Operating Systems, 2nd Edition, Upper Saddle River (NJ), Pearson Prentice Hall

[12]Tanenbaum Andrew S., Woodhull Albert S. (2006), Operating Systems Design and Implementation, 3rd Edition, Upper Saddle River (NJ), Pearson Prentice Hall

Linux:

[13]A.A.V.V. (2008), Advanced Linux Sound Architecture, <http://en.wikipedia.org/wiki/Advanced_Linux_Sound_Architecture>

[14]A.A.V.V. (2008), Forward OSS Kernel Module, <http://wiki.freespire.org/index.php/Forward-oss_Kernel_Module>

[15]A.A.V.V. (2008), Linux Audio Developers Simple Plugin API <http://www.ladspa.org/>

[16]A.A.V.V. (2008), Open Sound System <http://en.wikipedia.org/wiki/Open_Sound_System>

[17]Iwai Takashi, Proc Files of ALSA Drivers, <http://alsa.opensrc.org/index.php/Procfile.txt>

[18]Iwai Takashi (2005),Writing an ALSA Driver, 0.3.7 Edition, <http://www.kernel.org/pub/linux/kernel/people/tiwai/docs/writing-an-alsa-driver.pdf>

[19]Littler John (2007), An Introduction to Linux Audio,

<http://www.linuxdevcenter.com/pub/a/linux/2007/08/02/an-introduction-to-linux-audio.html>

[20]Marti Don (2008), Linux Audio: it's a mess, <http://lwn.net/Articles/299211/>

[21]Müller Ingo (2007), AlsaStackAKB,
<http://alsa.opensrc.org/index.php/AlsaStackAKB>

[22]Müller Ingo (2007), OSS emulation,
<http://alsa.opensrc.org/index.php/OssEmulation>

[23]Pappalardo Filippo (2004), An Introduction to Linux Audio,
<http://www.osnews.com/story/6720&page=2>

[24]Phillips Dave (2008), Java Sound & Music Software for Linux, Seattle (WA), Linux Journal, <http://www.linuxjournal.com/content/java-sound-music-software-linux-part-1>

[25]Phillips Dave (2004), On the ALSA track, Seattle (WA), Linux Journal,
<http://www.linuxjournal.com/article/7391>

[26]Phillips Dave (2007), The Sound of Linux, Seattle (WA), Linux Journal,
<http://www.linuxjournal.com/node/1005969>

[27]Piccardi Simone (2005), Guida alla Programmazione in Linux,
<http://www.cact.unile.it/facilities/XC6000/html-man/gapil/gapil.html>

[28]Poettering Lennart (2008), A Guide through the Linux Sound API Jungle,
<http://0pointer.de/blog/projects/guide-to-sound-apis.html>

[29]Torelli Daniele (2006), ALSA OSS e JACK,
<http://relug.linux.it/wiki/index.php/ALSA,_OSS,_JACK>

[30]Tranter Jeff (2004), Introduction To Sound Programming with ALSA, Seattle (WA), Linux Journal, <http://www.linuxjournal.com/article/6735>

[31]Tranter Jeff (1999), The Linux Sound Howto, revision 1.22,
<http://www.faqs.org/docs/Linux-HOWTO/Sound-HOWTO.html>

[32]4Front Technologies, Audio Programming,
<http://www.opensound.com/pguide/audio.html>

[33]4Front Technologies, Open Sound System – Programmer's Introduction,
<http://www.opensound.com/pguide/intro.html>

[34]4Front Technologies (2007), Programming with OSS,
<http://manuals.opensound.com/developer/programming.html>

Apple:

[35]Anguish Scott, Buck Eric M. Yacktman Donald A. (2002), Cocoa Programming,Sams Publishing, <http://www.promac.ru/book/Sams%20-%20Cocoa%20Programming/0672322307_main.html>

[36]Apple Inc. (2007), Accessing Hardware from Applications, Cupertino (CA),
<http://developer.apple.com/documentation/DeviceDrivers/Conceptual/AccessingHardware/AccessingHardware.pdf>

[37]Apple Inc.(2001), Audio and MIDI on Mac OS X,
<http://developer.apple.com/audio/pdf/coreaudio.pdf>

[38]Apple Inc. (2008), Audio Reference, Cupertino (CA),
<http://developer.apple.com/reference/MusicAudio/index.html>

[39]Apple Inc. (2006), Audio Unit Programming Guide, Cupertino (CA),
<http://developer.apple.com/documentation/MusicAudio/Conceptual/AudioUnitProgrammingGuide/AudioUnitProgrammingGuide.pdf>

[40]Apple Inc. (2007), Cocoa Application Tutorial, Cupertino (CA),
<http://developer.apple.com/documentation/Cocoa/Conceptual/ObjCTutorial/ObjCTutorial.pdf>

[41]Apple Inc. (2007), Cocoa Fundamentals Guide, Cupertino (CA), <http://developer.apple.com/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf>

[42]Apple Inc. (2007), Core Audio Overview, Cupertino (CA), <http://developer.apple.com/documentation/MusicAudio/Conceptual/CoreAudioOverview/CoreAudioOverview.pdf>

[43]Apple Inc. (2008), I/O kit Framework Reference, Cupertino (CA), <http://developer.apple.com/documentation/Darwin/Reference/IOKit/index.html>

[44]Apple Inc. (2006), Sound Programming Topics for Cocoa, Cupertino (CA), <http://developer.apple.com/documentation/Cocoa/Conceptual/Sound/Sound.pdf>

[45]Apple Inc. (2006), TN2091 Device input using the HAL Output Audio Unit, Cupertino (CA),<http://developer.apple.com/documentation/Darwin/Reference/IOKit/index.html>

[46]Apple Inc. (2006), TN2097 Playing a sound File using the Default Output Audio Unit, Cupertino (CA), <http://developer.apple.com/documentation/Darwin/Reference/IOKit/index.html>

[47]Dalrymple Mark (2003), The machine that goes ping, <http://old.macedition.com/bolts/bolts_20030410.php>

[48]Harmony Central (2008), Audio Programming, <http://www.harmony-central.com/Computer/Programming/>

Windows:

[49]A.A.V.V. (2008), Audio Stream Input/Output,
<http://en.wikipedia.org/wiki/Audio_stream_input_output>

[50]A.A.V.V. (2008), DirectX, <http://en.wikipedia.org/wiki/DirectX>

[51]A.A.V.V. (2008), Direct Music,
<http://en.wikipedia.org/wiki/DirectMusic>

[52]A.A.V.V. (2008), Direct Sound, <http://en.wikipedia.org/wiki/DirectSound>

[53]A.A.V.V. (2008), DirectSound Programming Tutorials,
<http://blueshogun96.wordpress.com/2008/05/19/directsound-programming-tutorials-lessons-1-6/>

[54]A.A.V.V. (2008), How to start writing DirectX 9 Applications,
<http://www.csharp-home.com/index/tiki-read_article.php?articleId=105>

[55]A.A.V.V. (2008), Technical Features New to Windows Vista,
<http://en.wikipedia.org/wiki/Technical_features_new_to_Windows_Vista>

[56]A.A.V.V. (2008), Universal Audio Architecture,
<http://en.wikipedia.org/wiki/Universal_Audio_Architecture>

[57]A.A.V.V. (2008), Unravelling the reasons for Vista Audio Glitches,
<http://www.itwriting.com/blog/492-unravelling-the-reasons-for-vista-audio-glitches.html>

[58]A.A.V.V. (2008), Vista Audio for Musicians? Review,
<http://www.thewhippinpost.co.uk/news/vista-review-for-musicians.htm>

[59]A.A.V.V. (2008), Windows Driver Model,
<http://en.wikipedia.org/wiki/Windows_Driver_Model>

[60]A.A.V.V. (2008), Working with MultiMedia,

<http://helloworld.siteburg.com/content/cplusplus/cpp3/ch15lev1sec3.htm>

[61]Andera Craig (2005), Direct Sound Basics Tutorial,
<http://alt.pluralsight.com/wiki/default.aspx/Craig.DirectX/DirectSoundBasicsTutorial.html>

[62]Creative Labs (2006), Audio in Windows Vista

<http://forums.creative.com/creativelabs/board/message?board.id=Vista&message.id=1694>

[63]Gold Mike (2003), Virtual Piano in C#,
<http://www.c-sharpcorner.com/UploadFile/mike4/DotNetPiano11122005012432AM/DotNetPiano.aspx>

[64]Microsoft Corporation, Audio Device Technologies for Windows,
<http://www.microsoft.com/whdc/device/audio/default.mspx>

[65]Microsoft Corporation, MSDN, <http://msdn.microsoft.com/it-it/default.aspx>

[66]Microsoft Corporation, Programming Guide for Direct Sound,

<http://msdn.microsoft.com/en-us/library/bb219822(VS.85).aspx>

[67]Microsoft Corporation, Windows Driver Model Audio Terminology,
<http://msdn.microsoft.com/en-us/library/ms790016.aspx>

[68]Miles Brook (2003), The Forger's WIN32 API Programming Tutorial,
<http://www.winprog.org/tutorial/start.html>

[69]Osterman L., Omiya E.H. (2005), Vista Audio Stack and API,
<http://channel9.msdn.com/shows/Going+Deep/Vista-Audio-Stack-and-API/>

[70]Programmer's Heaven, What's the difference between Direct Sound and Direct Music?,
<http://www.programmersheaven.com/2/FAQ-DIRECTX-Difference-Between-

DirectSound-and-DirectMusic>

[71]Root M., Boer J. (1999), DirectX Complete, New York, McGraw-Hill

[72]Solomon D.A., Russinovich M.E. (2000), Inside Microsoft Windows 2000 – 3rd Edition, Redmond, Microsoft Press

Sound Server e Plug-in

[73]A.A.V.V. (2008), Allow Multiple Programs To Play Sound At Once, <http://wiki.archlinux.org/index.php/Allow_multiple_programs_to_play_sound_at_once>

[74]A.A.V.V. (2008), Dmix, <http://alsa.opensrc.org/home/w/org/opensrc/alsa/index.php?title=DmixPlugin>

[75]Davis Paul (2003), The Jack Audio Connection Kit, LAD Conference, ZKM Karlsruhe, <http://lad.linuxaudio.org/events/2003_zkm/slides/paul_davis-jack/title.html>

[76]Gulden Jens (2007), JJack Overview, <http://jjack.berlios.de/>

[77]Jack Audio <http://jackaudio.org/>

[78]Phillips Dave (2001), Linux Audio Plug-Ins: A look into LADSPA, O'Reilly Media, <http://www.linuxdevcenter.com/pub/a/linux/2001/02/02/ladspa.html>

[79]Poettering Lennart (2008), Glitch Free Audio, <https://fedoraproject.org/wiki/Features/GlitchFreeAudio>

[80]Poettering Lennart (2008), Pulse Audio Documentation, <http://0pointer.de/lennart/projects/pulseaudio/doxygen/>

[81]Pulse Audio, <http://www.pulseaudio.org/>

[82]Trulson Jon (2008), The Network Audio System,
<http://radscan.com/nas.html>

Multimedia:

[83]A.A.V.V. (2008), Azalia, <http://it.wikipedia.org/wiki/Azalia>

[84]A.A.V.V. (2008), Latenza, <http://it.wikipedia.org/wiki/Latenza>

[85]A.A.V.V., MP3 (2008), <http://en.wikipedia.org/wiki/MP3>

[86]A.A.V.V. (2008), ReWire, <http://en.wikipedia.org/wiki/ReWire>

[87]A.A.V.V. (2008), Sound Blaster <http://en.wikipedia.org/wiki/Sound_Blaster>

[88]Brandt Eli, Dannenberg Roger B. (1998), Low Latency Music Software Using Off-the-shelf operating systems, School of Computer Science, Carnegie Mellon University

[89]Creative Labs (2008), OpenAL
<http://connect.creativelabs.com/openal/default.aspx>

[90]Intel Corporation (2004), High Definition Audio Specification, Revision 1.0,
<http://download.intel.com/standards/hdaudio/pdf/HDAudio_03.pdf>

[91]MacMillan K., Droettboom M., Fujinaga I. (2001), Audio Latency Measurements of Desktop Operating Systems, Peabody Institute of the Johns Hopkins University

[92]Muheim Men (2003), Design and Implementation of a Commodity Audio System, Diss. ETH No 15198, Zurich, Swiss Federal Institute of Technology

[93]Ternström Sten (2007), Audio Software Architectures,
<http://www.csc.kth.se/utbildning/kth/kurser/DT2410/_notes/audio_f_software_arch.pdf>