V. JACK

Several third party software developers have built an audio system for OS X that routes audio between applications. Rewire from Propellerhead Software, and Soundflower, originally from Cycling 74 (Matt Ingalls) and now maintained by Joachim Bengtsson (ThirdCog) are two such examples. Neither of these is described in much detail so the exact theory of operation and specifications are difficult to know. In contrast, JACK has been well documented by its developers, particularly Paul Davis and Dan Nigrin. It is an Open Source programme so there is also a considerable amount of information published by other interested parties. For this reason, and the fact that JACK proposes to be a professional solution, JACK is considered here in some depth.
Vucic (nd) has dedicated several pages of his paper (Free Software Audio Applications…an overview of functionality and usability) to describing JACK. On p21 he compares freeware to commercial software, maintaining that the two work on different sets of principles. Features of commercial software occur as a result of market research and are designed to address a particular user segment, as well as working well only with certain hardware, such as proprietary audio I/O interfaces. Non-commercial software is free from such market driven decisions and may include a variety of unusual features. It can also be driven by a purely problem-solving approach, as is the case of JACK. Paul Davis describes how the impetus for developing JACK was one of finding a solution to a problem; "the question of how to get different audio applications to talk to each other…you'd want to do something that seemed very obvious and you just could not do it" (Kerner, 2004).

Theory of Operation

JACK was conceived in 2001 and originally written for the Linux operating system, where it can utilise the scalability and reliability of this OS. Other Linux solutions (eg NAS, artsd, esd) do not provide sample accurate synchronised I/O of multiple audio streams, nor were they designed for performance within low latency systems (Phillips, 2006). Stephane Letz successfully ported JACK to OS X, and the initial public release for OS X was on the 7th of January 2004. This was version 0.4 and was named Jack Tools. In the same month its designer, Paul Davis was awarded a Bronze Open Source Award for the Linux version of JACK (Kerner, 2004). From the outset the design objective was to create a professional specification solution, which would allow streaming of high bandwidth data between independent applications with low latency.
Not all audio software is designed for professional use. JACK does meet this criteria, and therefore must be evaluated against the statement "JACK is different from other audio server efforts in that it has been designed from the ground up to be suitable for professional audio work. This means that it focuses on two key areas: synchronous execution of all clients, and low latency operation." (Davis P, 2008)
It is anticipate that data other than audio could be supported so in future video may be added. It is possible to run more than one JACK server, where each would form its own independent setup. Applications connected to JACK have their own graphical interfaces with JACK making no specifications as to different GUI toolkits or libraries. Consequently, a JACK setup can be spread across multiple system processes. A primary design goal was to provide sample-accurate synchronisation of all clients. To achieve this goal all clients must process the audio for an exact period of time, in other words each client must execute the audio in exact lock-step. Other specifications of JACK are that it uses only mono audio streams (i.e. interleaved audio is not supported), and 32 bit floating point (to IEEE-754[6], normalised to a range of -1.0,+1.0) is used.

[6] IEEE Standard for binary floating-point arithmetic ANSI/IEEE Std 754-1985

Included in the design goal is the ability to add or remove clients while the JACK server is running as well as to connect applications that are running.
Other approaches typically use plugin APIs that are shared objects that must execute in the context of the host application. Rewire, and DirectConnect are examples of high-level plugin APIs which use shared objects. To simplify design these run in a single process by the host application. One of the more difficult tasks any OS has is the real-time scheduling of each audio application correctly. UNIX approaches generally employ sound servers using a 'push' model. This means the applications can write any amount of data from very small to very large when it suits it. The push model does not maintain synchronous operation of all applications so it cannot generate audio at the same time we actually hear it. This is sufficient for consumer applications such as MP3 players but not satisfactory for professional audio due to the large amount of buffering required, increasing latency. Most of these models do not provide inter-application routing. JACK uses a 'pull' model to ensure accurate control of the audio data. In this model data is drawn from the application by the JACK server.

The JACK software consists of several parts. The JACK Server (jackdmp), which manages the clients. Jackdmp is optimised to use the processing power of multi-core CPU machines.
JackPilot is the GUI to allow connection setup and control of jackdmp. The JackRouter is a CoreAudio driver that allows any OS X audio application to be a JACK client. There are also JACK plugins (AU and VST) providing for additional audio routing possibilities. For example a third application can act as an effects insert. A NetJack module has also been developed which allows streaming of audio over a network. At present this feature has been suspended until fixed.

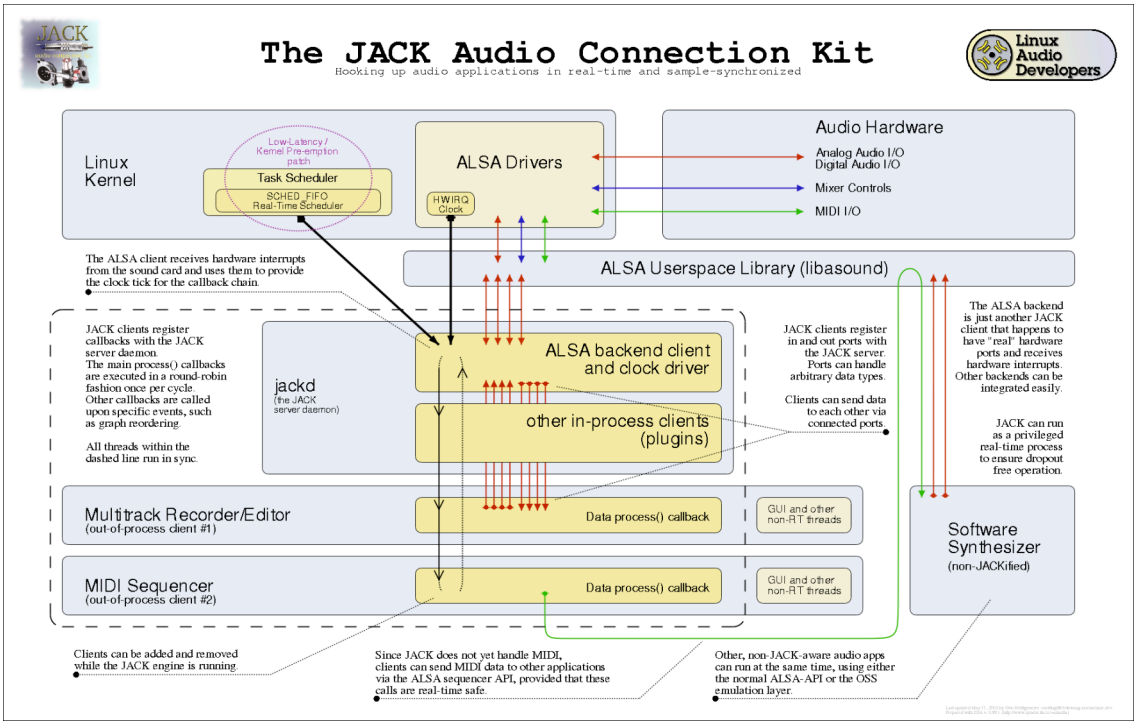

Fig. 36 JACK architecture diagram showing the Linux version. (Davis P., 2003)

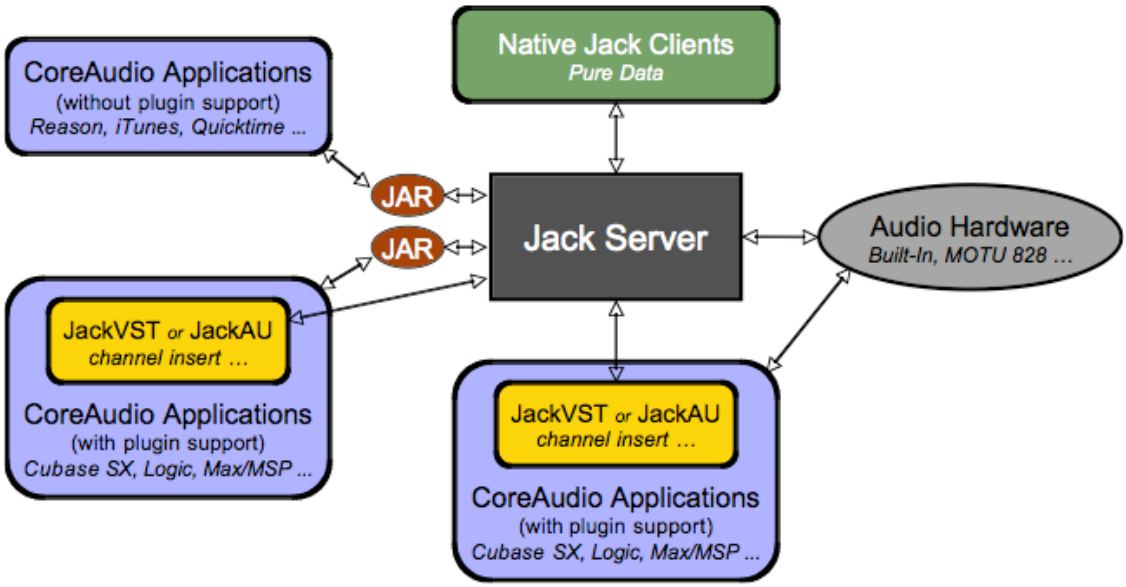

Fig. 37 JACK architecture diagram showing the OS X version. (Davis P., 2003)

The central part of JACK is the JACK Server, which controls the audio data transmission between clients. These may be either applications or hardware devices, such as built in audio or an external audio I/O interface unit. A small number of 'native' JACK applications exist but most use the JackRouter (JAR) to communicate with jackdmp. When the first client signals Jackdmp via the CoreAudio driver Jackdmp activates a client graph. This is a set of nodes to be executed consecutively on a periodic basis. Each client has its own audio process function to be called in a specific order. The driver calling jackdmp at a regular interval determined by the buffer size executes the graph. The server essentially passes this interrupt to each client in turn. The design criteria is that all audio data processing and transfer is completed for all clients within two consecutive interrupts. This must include server/ client communication (Davis P, 2008).

Paul Davis described JACK in some detail at the Linux Audio Developers Conference, ZKM Karlsruhe in March 2003 and much of the information included here is drawn from that presentation.

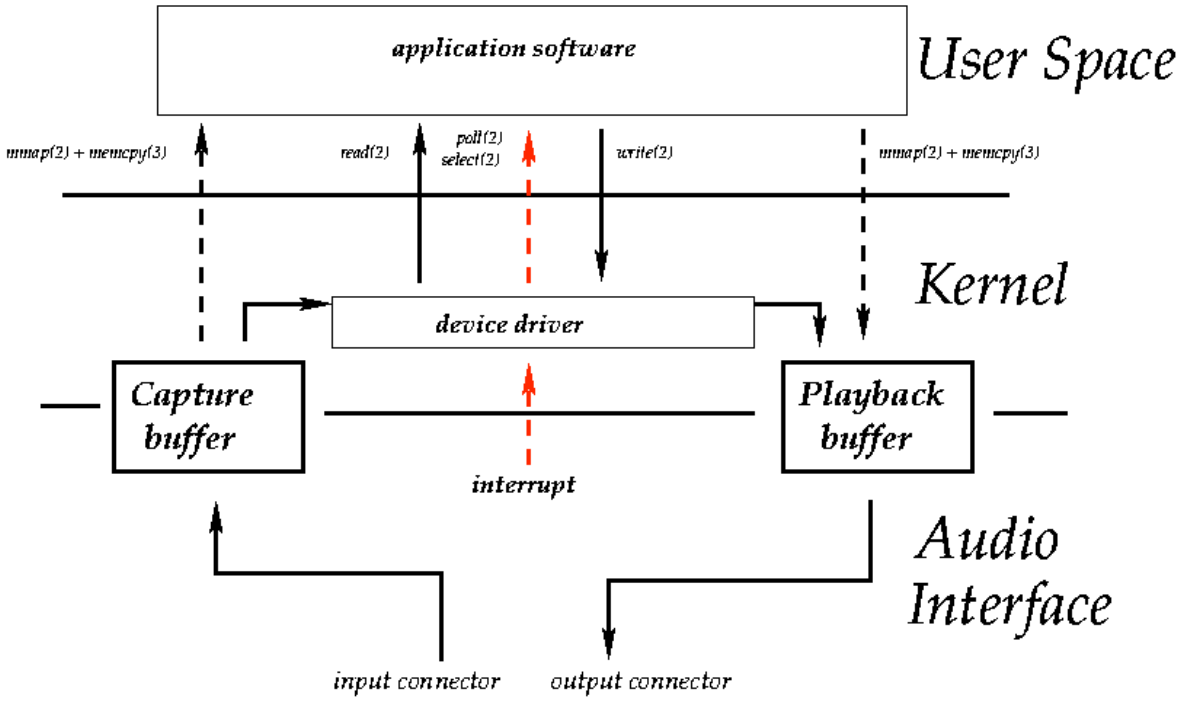

Fig. 38 Typical audio I/O architecture (Davis P., 2003). An interrupt to the OS via the CPU wakes the driver. The read and write lines are the system calls. Data transfer between the User layer and the buffers in the Kernel layer use either a memory copy command or DMA. Davis describes this as the best API model.

Using the method of two interrupts per hardware buffer, JACK can react to the input signal within one interrupt interval and deliver audio within two interrupt intervals.

The Context Switch

The best audio applications are built with a high level of integration between the GUI and the DSP code. Normally a software designer will want to keep to one developer Toolkit so that both the GUI and DSP code can be run in the same process. This is not always possible with audio software where for example two applications that need to be connected run in different processes. One of the critical aspects in the design of JACK is to allow plugins running in another process to interface with jackdmp. The advantages are that this isolates the host from plugin errors, avoids the requirement for IPC (inter-process communication) between the GUI and the DSP code, and developers can choose their own GUI Toolkits. Paul Davis expresses frustration on this point, "one of the most annoying features of writing audio application software is we cannot integrate good applications written in different toolkits" (Davis P, 2003).
JACK uses a Context Switch to facilitate clients running in a different process. This is the ability of a CPU to switch between processes and saving all the register values for the current process on the stack followed by restoring all the register values for the next process. On a machine using virtual memory (i.e. any personal computer) the same address in two different processes rarely refers to the same physical memory location. Therefore the virtual memory addresses must be mapped to physical memory, making context switching computationally intensive. The task can be handled in hardware eg the Translation Lookaside Buffer (TLB) does this mapping on an Intel processor. Upon each switch, the memory (and cache) must be invalidated, and the impact of this is to introduce some delay. This is linear with CPU speed for each register save/ restore cycle but the TLB effects are dependent on a number of factors outside the programmes control. These

include the frontside buss speed, memory speed, cache size, and the process working set size. On a modern x86 processor register switching time is in the order of 10 to 50 µS. Cache and/or TLB effects can increase this time by a factor of two or four. The total time available to process 64 frames per interrupt at 48kHz is 1333 µS. This is the lowest order of latency that many current PCI interfaces can support. While 50 µS is not a lot, if JACK is serving ten clients this would be 500 µS, which is nearly 40% of the total time available.
One of the most common things effecting the context switch time is the process working size. If the program has to use a large amount of memory it will invalidate a lot of cache and the cache will need refilled (and visa versa). Several of JACK's design goals are significant in determining the low latency. Whereas the internal clients are run as shared object APIs (as are VST plugins, Rewire), the external clients also need to communicate with jackdmp efficiently. Memory copy techniques have been avoided and instead JACK uses shared memory to exchange data. Communication is done using FIFOs rather than signals/slots event notification as these were found to be too slow. FIFOs provide an extremely fast and lightweight IPC mechanism. A compile option in JACK enables FIFOs to be placed on a RAM-based files system. There are other advantages in using FIFOs in the model, namely that they are easily reconfigurable when the graph changes. The FIFOs on the server side can be left open, and clients can close and reopen to reflect the graph order.

This paper will not look at how JACK functions at a coding level but it is informative to briefly consider the operation of the external client. Within the bounds of the external client process delimiter reside a request socket and an event socket, both of which are duplex. The request socket can initiate a request to jackdmp. An example would be a client shutdown message. The request may also communicate with the application GUI or any other process thread (eg for MIDI I/O). The event socket accepts control signals from jackdmp. This enables the client to disconnect or reconnect to another client and is able to communicate with a special audio thread created by jackdmp for the client. This JACK thread runs the main event loop and controls the data to the FIFO from the previous client and the data going to the next client via the output FIFO. Other threads cannot have audio I/O and must not be able to block the JACK thread. They should also register shutdown callbacks so the programme is aware of JACK shutting down its client. Thus the client is responsible for managing at least one thread (the audio thread), and executes process, xrun (buffer under-runs or overruns), and sample rate changes. It must be able to function in real-time without getting any blocking calls. For this reason programming functions such as malloc, sleep, read/write/open/close, and some pthread (POSIX threads) synchronisation functions must be avoided.

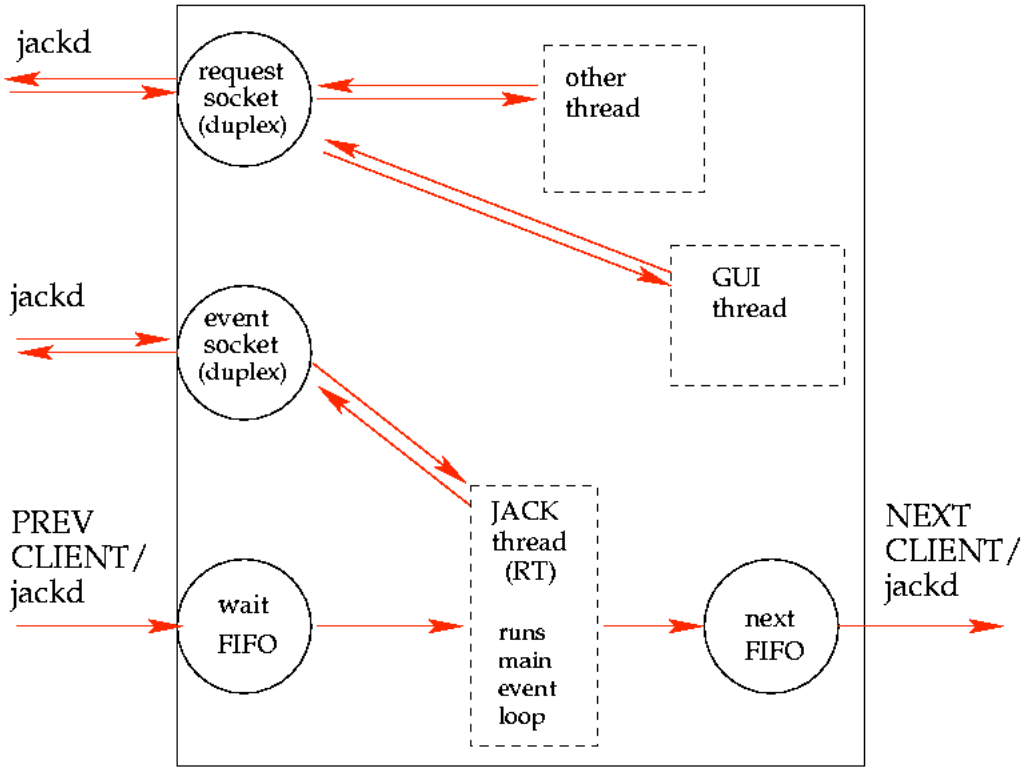

Fig. 39 A JACK external client (Davis, 2003)

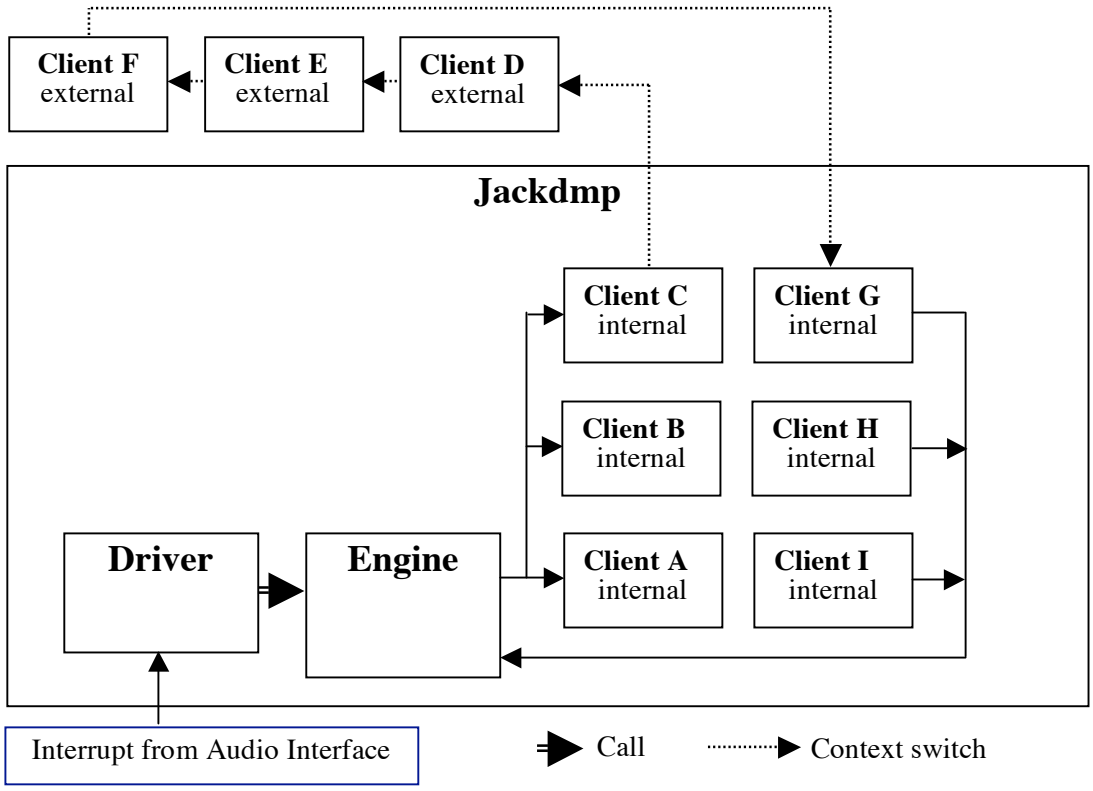

Fig. 40 JACK block diagram (after Davis P, 2003). External clients are run as separate processes, like normal applications.

JACK provides the following options for the CoreAudio driver:

-c, --channels	Maximum number of channels (default: 0)
-i, --inchannels	Maximum number of input channels (default: 0)
-o, --outchannels	Maximum number of output channels (default: 0)
C, --capture	Provide capture ports. Optionally set CoreAudio device name (default: will take default CoreAudio input device)
P, --playback	Provide playback ports. Optionally set CoreAudio device name (default: will take default CoreAudio output device)
-m, --monitor	Provide monitor ports for the output (default: false)
-D, --duplex	Provide both capture and playback ports (default: true)
-r, --rate	Sample rate (default: 44100)
-p, --period	Frames per period (default: 128)
d, --device	CoreAudio device name (default: will take default CoreAudio device name)
-I, --input-latency	Extra input latency (frames) (default: 0)
-O, --output-latency	Extra output latency (frames) (default: 0)
-l, --list-devices	Display available CoreAudio devices (default: true)
-L, --async-latency	Extra output latency in asynchronous mode (percent) (default: 100)

Notes:

1. The period (p) specifies the number of frames between *process ()* function calls. JACK's latency therefore is --period ÷ --rate.
2. The maximum number of output ports defaults to 128. QjackCtl allows selection of 512 ports. More are available with sufficient memory.
3. Several additional settings relating to memory management and sound card performance are available on the Linux version of JACK. For example –n, (--nperiods) specifies the number of periods in the hardware buffer. The default value is 2. The buffer size (bytes) = (-p)*(-n)*4. (Phillips, 2006)

Table 10. JACK parameter settings (after Capela R., 2008)

Paul Davis has indicated several ways in which JACK could be improved in future. Several changes to the Kernel are proposed. Although JACK employs real-time scheduling processes in practice these don't always get run when they are supposed to be run. Deterministic scheduling would overcome this problem. Also a faster IPC method has been considered, using futexes/ doors, as FIFOs must go through various kernel locks. Reducing interrupt blocking is another area where improvements could be made. There are still a number of places in the kernel where interrupts get blocked for a while. Delays may be small but can use up to 25% of total available processing time to achieve lowest latency. Dynamically increasing the number of ports would not require the user to specify a fixed number of ports before starting the JACK server. MIDI implementation has also been considered and it would be easy to add MIDI using the port model (where that data length was a set number of MIDI bytes). Full MIDI support would require a different model that can handle asynchronous MIDI message communication (Davis P., 2003).
JACK is a server/client model, which is hard to make truly robust, as a badly coded application can cause the system to hang. Furthermore it uses a 'pull' model to ensure sample-accurate synchronisation of the streamed audio data and this does not work with applications that are not callback-based. Porting software written on a push model to a pull model can be difficult. It should be kept in mind that JACK is still a work in process. The OS X version has not yet reached

version 1.0. This indicates that although JACK is open source freeware, the development team is serious about creating and improving an audio routing solution suitable for professional use.

QJackCtl v0.3.2 - an alternative GUI

QJackCtl is an alternative GUI for JACK developed by Rui Nuno Capela. It provides several enhancements to the JACK GUI; more parameters can be set and these can be saved between sessions, application transport control is provided, it has a more detailed run status, the connections are presented in graphical (rather than tabular) form, and configurations can be saved in a visual patchbay. The graphical interface of QJackCtl has somewhat of a mixed PC and OS X look, due to it being a cross-platform solution that has been ported from the Linux version.

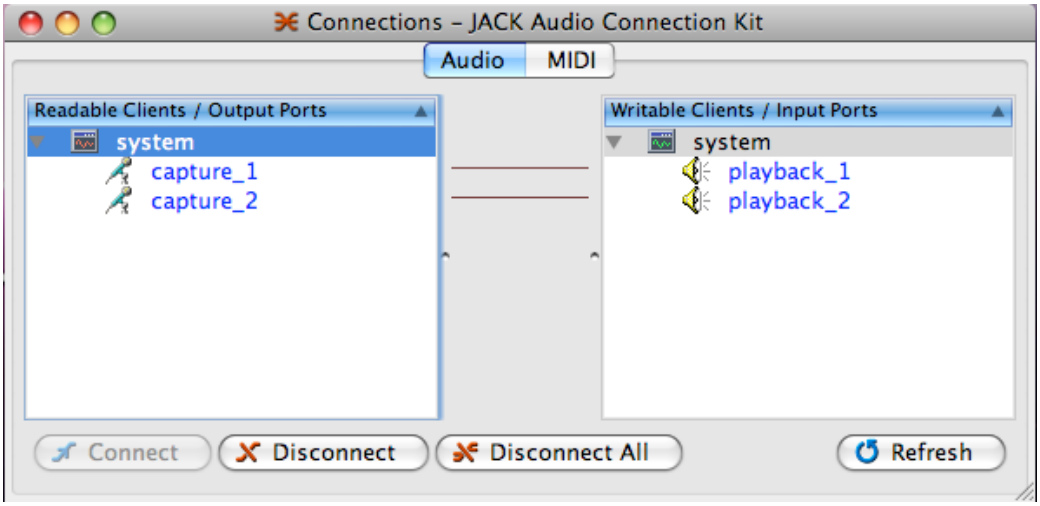

Fig. 41 The QJackCtl Connections window (Capela R., 2008) is very similar to JACK's but cables (in this case horizontal lines) show the connections made

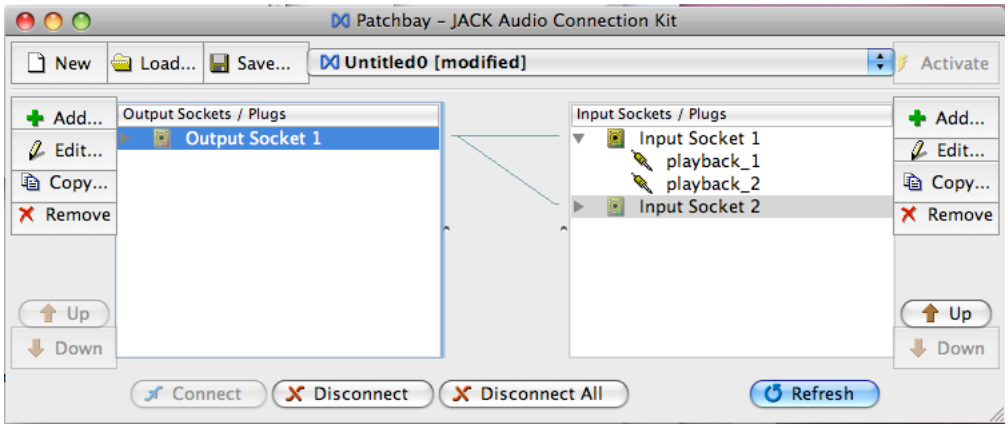

Fig. 42 The Patchbay (Capela R., 2008), like JACK, allows saved configurations but also with cables showing the connections

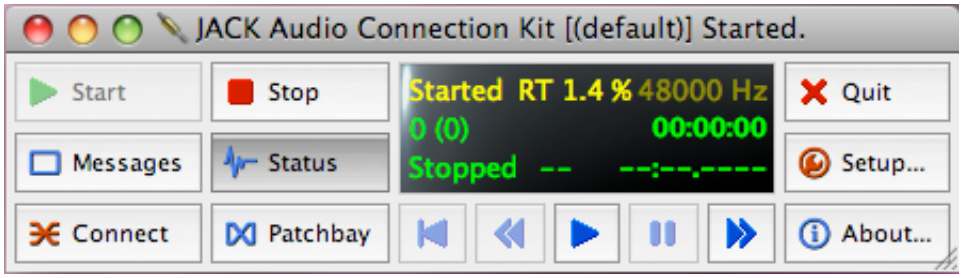

Fig. 43 QJackCtl main window showing the transport controls (Capela R., 2008)

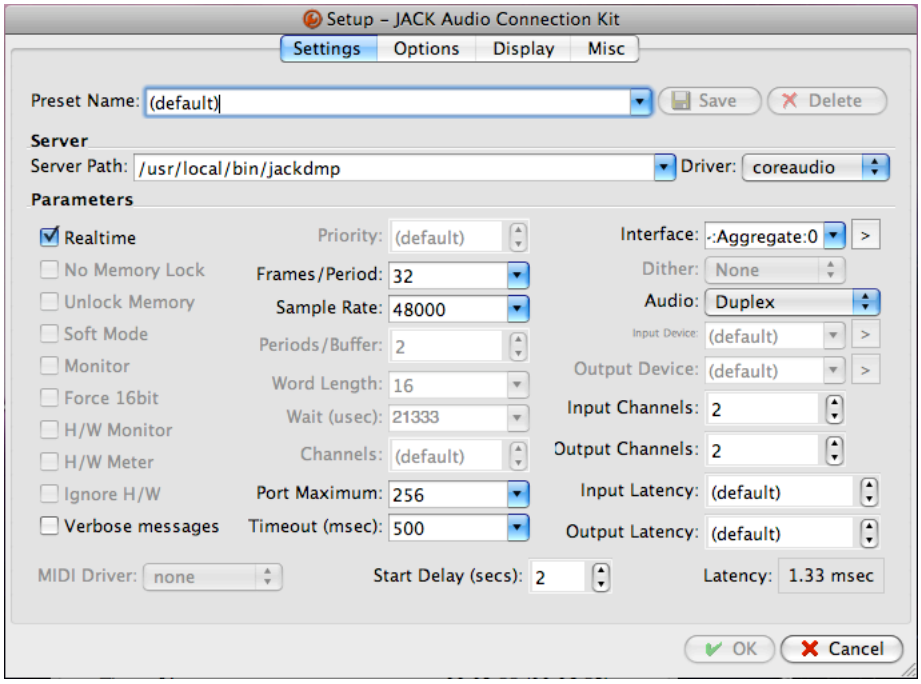

Fig. 44 The settings window (Capela R., 2008) allows GUI access to many more parameters than JACK. Latency is calculated and shown here, according to the sample rate and samples per buffer.

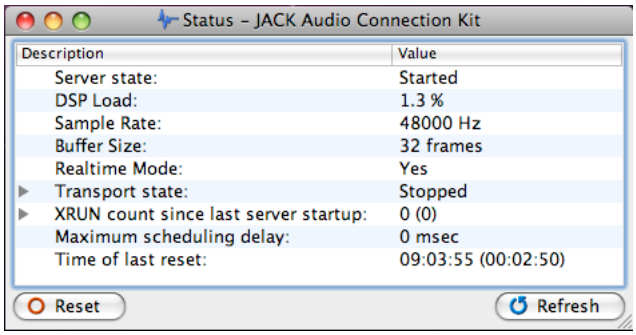

Fig. 45 The Status window (Capela R., 2008) provides considerably more information than JackPilot. Transport state can also show BPM and Timecode. XRUN can detail six types of buffer error count

VI. Latency

Latency can be defined as the delay occurring between an action and its effect. Latency is a central issue when using a personal computer for audio work. There is no processing on a sample by sample basis – instead the CPU will input, process, and output the audio in blocks. One solution is to have dedicated DSP chips to handle the audio processing eg Pro Tools TDM systems). Any delay introduced by any inter-application audio system will also be significant if it is more than a few milliseconds. As the total delay will be cumulative so it is essential that low latency is a priority in the routing design.

Latency can be split into Input and Output delays. The input latency is defined as the amount of time a system takes to respond to the input signal. First, data arrives at the audio I/O, then it is buffered until the next interrupt, when the CPU can access it. For output latency the process is; the data is ready for the I/O and starts filling the output buffer. When the next interrupt executes the CPU can deliver the data. If other data is already queued on the audio interface hardware then additional delays can be incurred before the audio output takes effect. Shortest latency will be achieved if there are two interrupts per hardware buffer. This is a variation of double buffering, which is common in graphical programming. The buffer is divided into two, with an interrupt as it crosses from one half to the other. The hardware works on one half; either writing to it or reading data from it, and the software works on the other half. One layer of the buffer is used for the audio input, and one for the audio output. While the user application writes to the output buffer 0, the hardware driver writes to input buffer 1; after that, starting from the following frame, the buffers are switched to output buffer 1 and input buffer 0 respectively.

To summarise, the total output latency will equal the buffer size and the input latency will equal the interrupt interval. Through latency, therefore, will equal the output latency.

At 44.1kHz each sample is 22.676µS so a latency of 100 samples takes 2.27mS. Sound travels at approximately 344 m/s so two musicians spaced 3m apart will each hear the other with a delay of 8.7mS. While this is acceptable for live performance, in more critical situations such as overdub recording using headphones, any delay of more than 3mS will make singing or playing in time difficult (Davis, 2003).

Virtual Instrument developers have gradually reduced the minimum buffer sizes as faster computers have become available. For example Native Instruments latest versions allow the user to set the output latency to 8mS. The Arturia Jupitor-8V audio buffer size can be set at low as 14 samples (0.3 mS at 44.1kHz). To achieve stable performance with this order of latency requires a fast computer to prevent buffer overruns occurring.

On a 2GHz or greater Intel Mac good results can be obtained but only by careful optimisation and dedicating the computer for sequencing use. Although CPU power has increased dramatically in the last five years there has also been an even greater demand for it by the DSP of audio software. Some Virtual Instruments and other signal processing plugins use complex algorithms requiring numerous computations, and these put a considerable workload on the CPU. A buffer setting of 5mS is a good compromise between minimal delay and reliable audio streaming without glitching. The CPU workload increases as buffer size is reduced so a fast computer is required to achieve latencies of this order. The relationship between buffer size, latency, and CPU load for audio (using the ASIO API) has been explored at the University of British Colombia (Corbett, van den Doel, Pai, 2002).

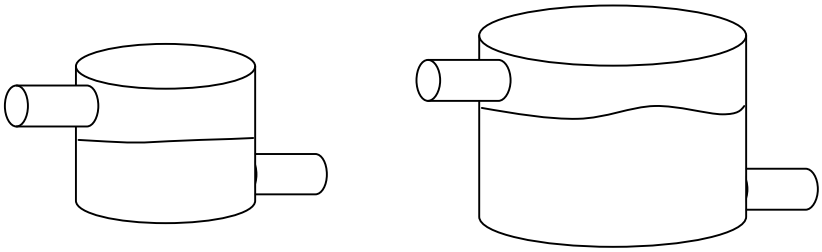

Fig. 46 A water tank analogy of buffer size. The tank represents the buffer size. Water can flow into the tank sporadically but will leave the tank in a continuous flow. The larger tank can store more water to compensate for irregular inflow but takes longer to fill (latency).

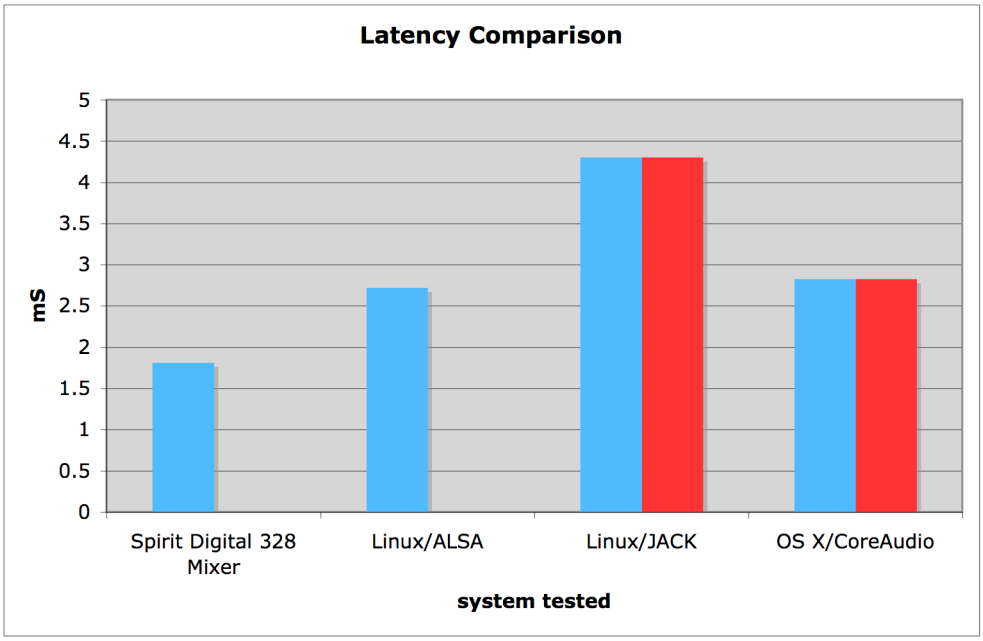

Fig. 47 Best case latency measurements of comparable systems (after MacMillan, Droettboom, Ichiro, 2001). Figures in red show results for testing under extra load. Best overall result was CoreAudio.

VII. How Audio is handled in OS X

a. CoreAudio

Apple's objectives for the design of audio on OS X were twofold. It is a system where latency is inherently low and is determined by the application. This provides a specification suitable for professional multi-channel audio I/O. Secondly, Apple wanted to address a problem that had arisen in OS 9, where third party developers were left to find many of the solutions for audio support (eg ASIO, EASI, MAS[7]). Although often referred to as drivers these are actually user-level APIs and latency results vary depending on whether a generic driver is enabled or one specified for the actual hardware. For many users OS X succeeds in providing a standardised and integrated audio system solution.
The reasons that CoreAudio does not allow inter-application audio routing are stated by Jeff Moore in an Apple Mailing List, "Mac OS X's audio system was designed first and foremost for performance…we have opted for better performance at the cost of not being able to provide this feature" (Moore, 2006). CoreAudio is designed for minimal latency whilst retaining glitch-free audio and uses a "pull" model to achieve this with reliability. Callbacks are run in real-time threads so there are some restrictions on what they can do.

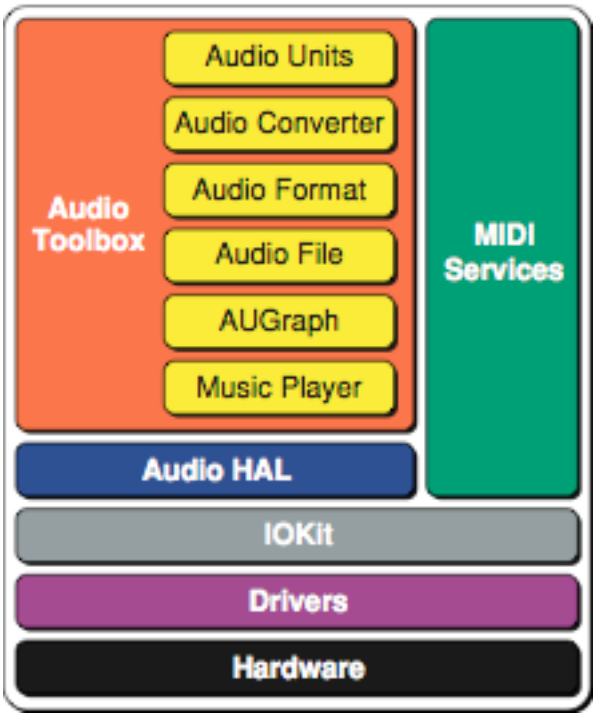

Fig. 48 CoreAudio architecture (Apple, 2004)

The CoreAudio is generally expressed in terms of layers where the Kernel layer includes the audio drivers and the I/O kit. Audio drivers use the I/O kit to perform many of its tasks, such as providing timing mechanisms and buffers.
Above this layer resides the User layer, which includes the Hardware Abstraction Layer (HAL). The HAL is used to present a consistent interface to any application to interact with hardware. Timing information and latency values are therefore handled by a standardised part of the OS.

[7] ASIO: audio stream input output (Steinberg), EASI: Enhanced audio streaming interface (Emagic), MAS: MOTU audio system (Mark of the Unicorn).

Sample accurate timing is fundamental to CoreAudio and is achieved with timestamps. A specific AU (the AUHAL) is used to get audio in and out of the HAL.

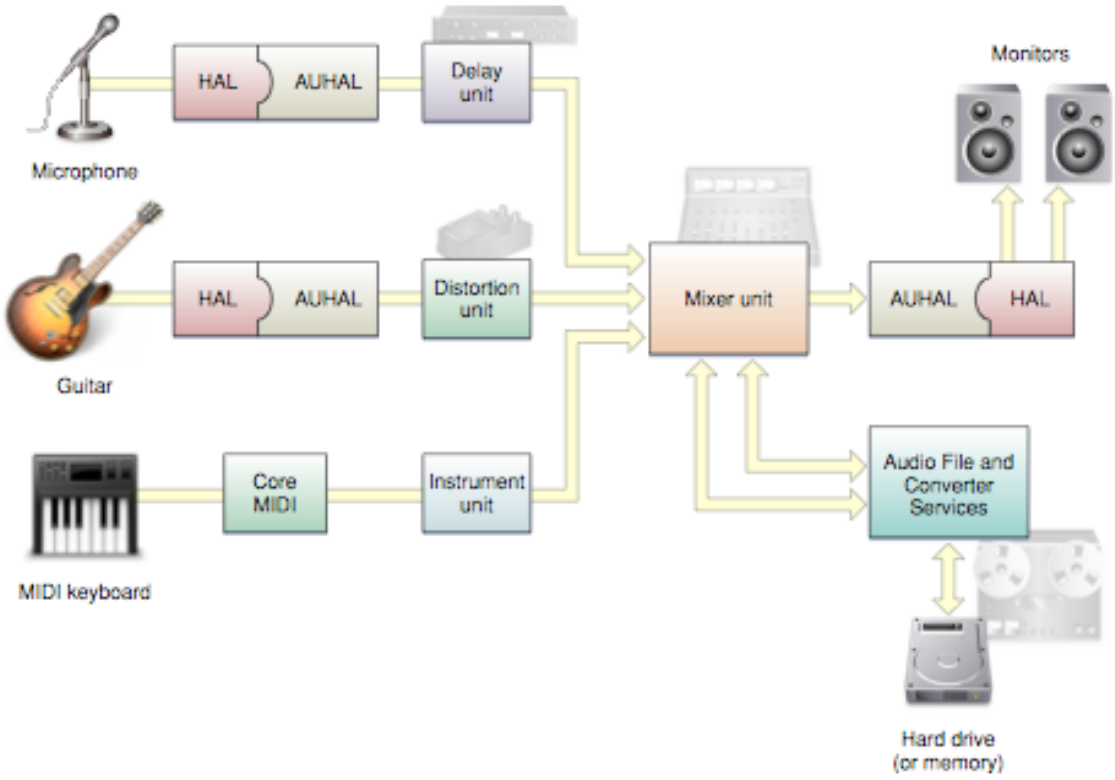

Fig. 49 The recording studio concept of CoreAudio (Apple, 2007)

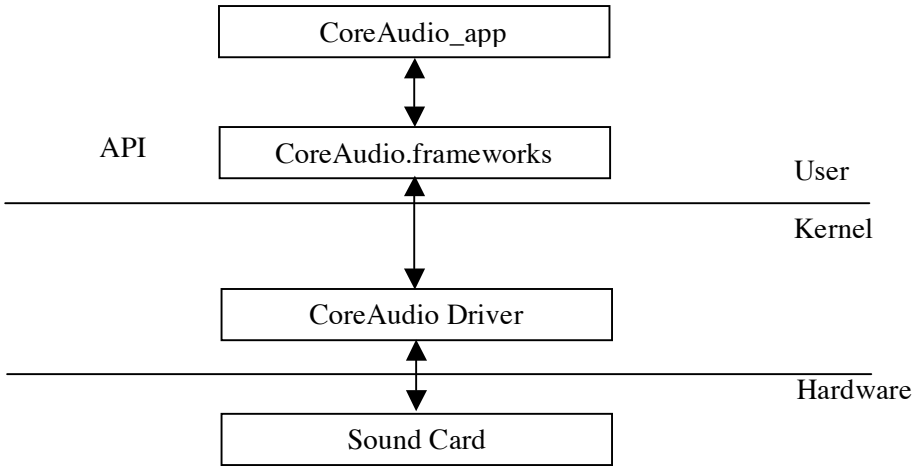

Fig. 50 CoreAudio layers (after Ekeroot, 2007)

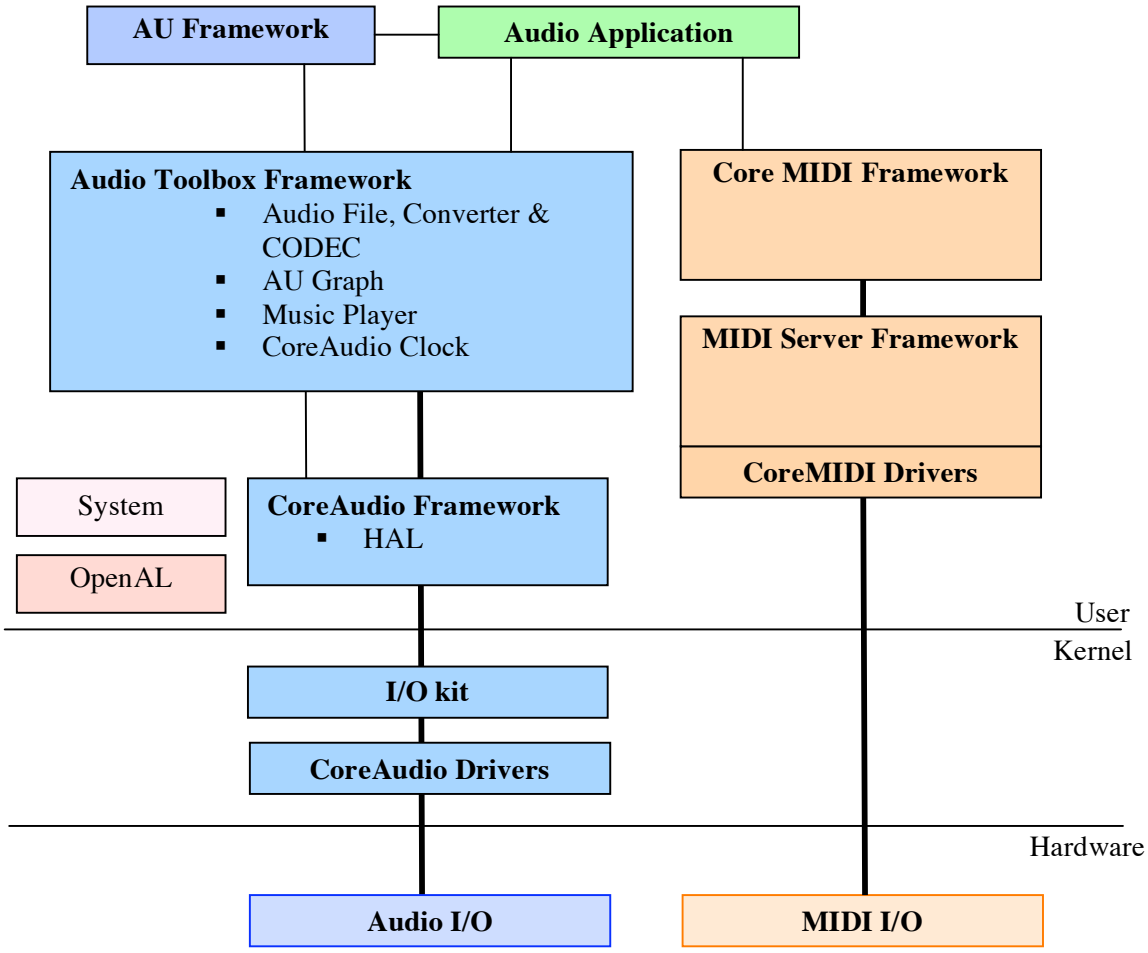

Fig. 51 CoreAudio architecture

The audio HAL is central to obtaining reliable audio on OS X. The audio HAL uses extremely accurate timecode on OS X to ensure clients perform their I/O at the proper time, according to their buffer size. HAL predicts when the I/O cycle has completed. The driver sets up the hardware buffer and takes a timestamp for very buffer cycle. "The Audio HAL keeps track of each timestamp and uses the sequence of timestamps to predict the current location of the audio I/O engine (in terms of sample frame read or written) at any time. Given that information, it can predict when a cycle will complete and sets its wake-up timestamp accordingly. This model, combined with the ability of the I/O Kit Audio family to receive audio data from each Client asynchronously, allows any number of clients to provide audio data that gets mixed into the Final output. It also allows different client buffer sizes; one client can operate at a very low buffer size (and a correspondingly low latency) while at the same time another client may use a much larger buffer. As long as the timestamps provided by the driver are accurate, the family and the Audio HAL do all of the work to make this possible." (Apple, 2006)

Audio Toolbox APIs Audio File and Converter	File read/ writes to file or buffer. File conversion (CODEC or bit depth).
AU Graph	Allows dynamic modification. Data is pulled from the head node (the output AU node).
Music Player	Playback of sequenced MIDI data or CoreAudio events. Can output to AU Graph.
CoreAudio Clock (added in version 10.3)	Reference clock; can supply SMPTE, samples, bars, beats & clicks.
Audio Units API	Supports 32 bit FP linear PCM (non-interleaved)

Table 11. OS X Audio API features

User access to CoreAudio in OS X is usually via the Sound pane of System Preferences or Audio MIDI Setup Utility. In accordance with Apple GUI design philosophy, there are very few things that can be adjusted in the Sound Pane. The AMS allows setting a few more parameters such as sample rate, where these may be adjusted for a given device. Interestingly, the MIDI side of the AMS has more functionality. There is a MIDI patchbay where virtual MIDI devices can be connected to and from any MIDI interface. This includes naming and setting which MIDI channels will be allowed for a given device. There are also two special devices in the MIDI device pane. The IAC driver allows for unlimited MIDI busses to be created for inter-application MIDI data. Secondly, the Network device allows MIDI communication between computers, using Bonjour over Ethernet.

b. QuickTime

QuickTime is a cross-platform multimedia technology that is primarily used to read, write, convert and stream video and/ or audio data. It supports over 100 data types, including image, video, audio[8] and web streaming formats. Media is handled in a data structure called a QuickTime movie, which acts as either a pointer or container file for the media. The QuickTime movie has a track or set of tracks for each data type.
CoreAudio handles the audio output of QuickTime. By default this is sent to the default audio device, but from version 7 it is possible to specify any output device. There are also settings for track volume, balance, channel assignment, pitch shift, playback speed, and an eight-band real-time spectrum analyser. Full functionality, including basic editing (truncate, loop), is provided in the Pro version.
An extensible component of QuickTime is the QuickTime Music Architecture (QTMA). This can play a sequence of music notes from an application, either internally or externally. Using QuickTime Pro, a Standard MIDI File (SMF) can be imported and saved as an audio file in as a QuickTime movie. In this case the convention of General MIDI applies, with channel 10 set to drums.

[8] Supported audio formats are: uncompressed 8, 16, 24 or 32 bit, IMA, µLaw, aLaw, AVI, WAV, DV audio, MP3, MPEG-4 audio.

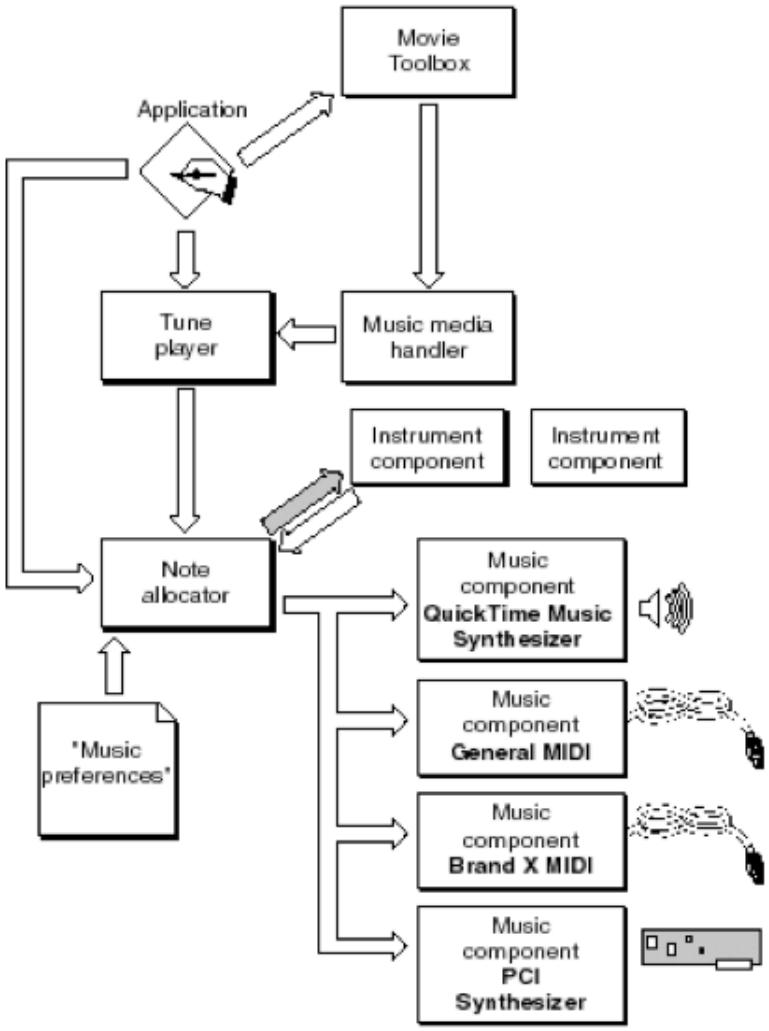

Fig. 52 QTMI architecture (Apple, 2006)

The Note Allocator plays individual notes with the application setting which instrument will sound. The synthesiser is selectable from any Soundfonts installed. The default device is the QuickTime Music Synthesizer, which has a specification similar to General MIDI 2, with 24-voice polyphony. Timing of a sequence of notes is handled by the Tune Player, which can play an entire sequence in an asynchronous manner, without application intervention. Tune Player is also responsible for volume setting and transport control. QuickTime music events (such as note, controller, marker) are held in a QuickTime movie track, which uses a media handle to access the Tune Player.

A problem arises when trying to bounce MIDI tracks as part of a stereo mix in Logic. During playback both audio tracks and any MIDI tracks assigned to the QuickTime Music Synthesiser are heard as they are both routed to the default audio device (i.e. the internal speakers). A bounce operation is handled inside Logic but the MIDI data is sent out to QuickTime so any instruments playing from the QuickTime Music Synthesiser are excluded. This condition is identified in Apple article 300898 but no fix is given. The solution is to use Soundflower or JACK to route the audio as detailed in Appendix III.

c. *Audio on a Windows PC*

For the purpose of comparison, a short description of how audio is handled on the Windows operation system is included in the section.
Audio routing on the PC is more flexible than on a Macintosh due to the Kernel Mixer. As well as a master volume, independent volume, balance and muting controls are present for Line input, Mic input, CD playback, MIDI sounds (from the soundcard), and Wave (from hard disk or memory).

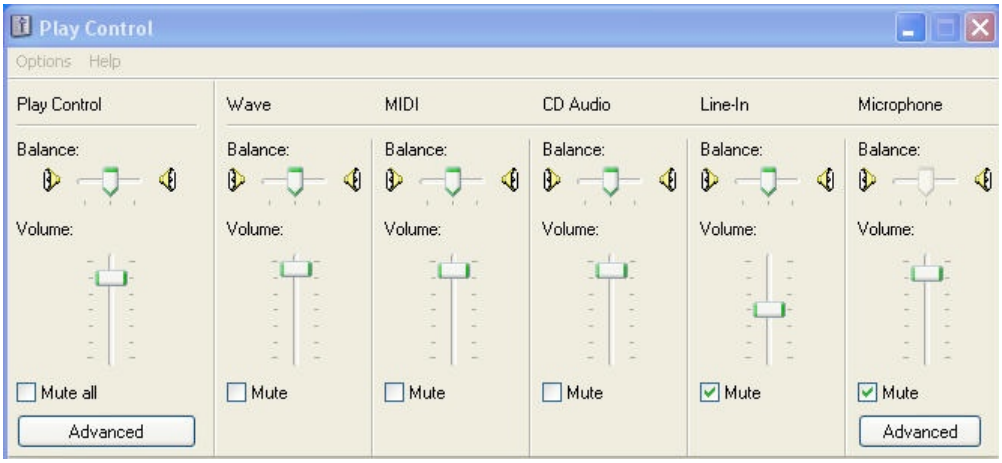

Fig. 53 The playback mixer in Windows XP (Microsoft, 2001)

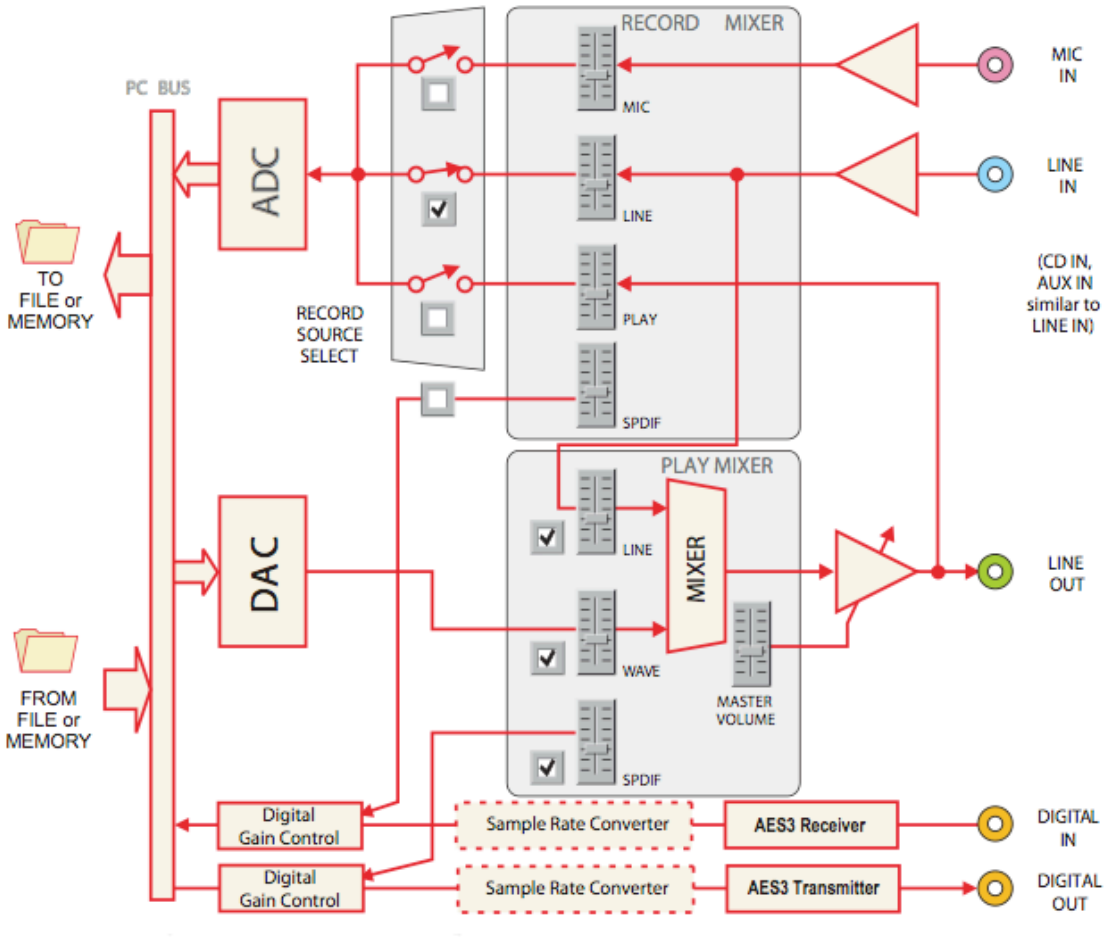

Fig. 54 Audio paths on the Windows PC (AES, 2006). Some paths have been omitted. For example the Mic input can also be directed to the Playback Mixer.

WDM[9] is a unified driver model, which provides audio mixing and resampling using a system component named KMixer. This allows multi-client access to hardware and unlimited audio streams mixed in real-time. In a Windows XP system internal buffering in the KMixer adds 30mS of latency to audio playback which most applications are stuck with, as they have no method to bypass the KMixer. Windows Vista introduced a new model named WaveRT. This has picked up on several methods used in the Linux ALSA architecture such as mapping the hardware buffer and control registers into user space so that the driver never touches the audio data. This eliminates using IRPs (I/O Request Packets) and associated kernel-user transitions. WaveRT supports both push and pull models of data transfer.

[9] The Win32 Driver Model

VIII. Software Reliability

While it is exceptionally rare to find a piece of software that simply does not work it is probably true to say it is also exceptionally rare to find a piece of software that never fails. In the ever-changing world of computer software we are accustomed to a certain amount of downtime due to software 'misbehaving'. With audio software there tends to be less certainty of stability than with software for other purposes. This is largely because of the real-time scheduling required.

Audio and Music software has remained highly competitive, and is generally provided by small software developer companies. Exceptions are Logic Audio, which is now a product of Apple Computers, and Pro Tools from Digidesign, which is a part of the Avid Company. It is interesting to note that Digidesign maintain a stable software product by specifying exactly (in some detail) what hardware and Operation System are required for compatibility with any given version of Pro Tools. Problems do still arise; an extreme case was Pro Tools LE which was not qualified for use with the Leopard OS until mid 2008 – this resulted in a wait of over six months for many users who had upgraded to a new Macintosh which came with Leopard preinstalled. In most cases incompatibilities can be fixed within a shorter timeframe.

Historically, music and audio software has been relatively immature when released to the public and while there has been some improvement in this area this remains a non-trivial issue for any professional user. In comparison a product such as Microsoft Office is virtually bug-free, and furthermore 'patches' are rare. The rapid evolution of OS X has exacerbated the problem, with some audio developers having to play catchup because newer versions of the OS are released frequently. In my experience, each major release of OS X has had some issues in handling audio. As the minor releases get up to about 5 (eg 10.4.5) things seem to settle down and maximum performance is obtained with few reliability problems. As further minor releases are added there is sometimes an increase in stability issues. This was the case in Tiger but 10.4.11 is quite reliable now and this is probably due to the audio developers being able to eventually fix all the bugs, since Tiger is no longer a moving target. Leopard has been particularly troublesome to some audio users. In the case of 10.5.2 several audio software developers issued statements advising against using it altogether (Kirn, 2008).

Several commentators (Sellers, 2004, Mertin, 2004) have noted that audio plugins in general have a high degree of incompatibility, either with the host application or the OS.

IX. Combinations

Due to the variety and complexity of sound operations that we now expect to perform on a computer, there are some situations where a combination of two or more of the sound utilities are needed to achieve the desired result.
A related aspect is how well the sound utilities 'live together' on the same computer. It will be infrequent that most users will require using several audio streaming utilities simultaneously. Several sound utilities will most likely be installed on one machine and used at various times. It is important that the software will flush completely out of the machines RAM when quit and that any processes that automatically load do not interfere with other applications.
During the testing phase I ran the following utilities on a single computer: JACK, Soundflower/Soundflowerbed, SoundSource, Sound Menu, WireTap Anywhere, PTHVolume, Rewire, and AudioHijack. Some of these are very compatible, but care must be taken when using other audio utilities with Soundflower, Rewire and JACK.
A problem arose while I was testing on an Intel iMac (OS 10.4.5) where all the audio hung. This was a situation where JACK, the Digidesign CoreAudio driver, QuickTime, Logic Express, and Soundflower were all activated. For some reason QuickTime failed to appear in the JACK connections manager and JACK hung. CPU usage was at a constant 51% and almost all of this was caused by Soundflower. Force-quitting Soundflower fixed this but JACK, Logic Express, and QuickTime Player refused to be force quit as CoreAudio had hung. The computer needed a hardware restart before work could recommence.

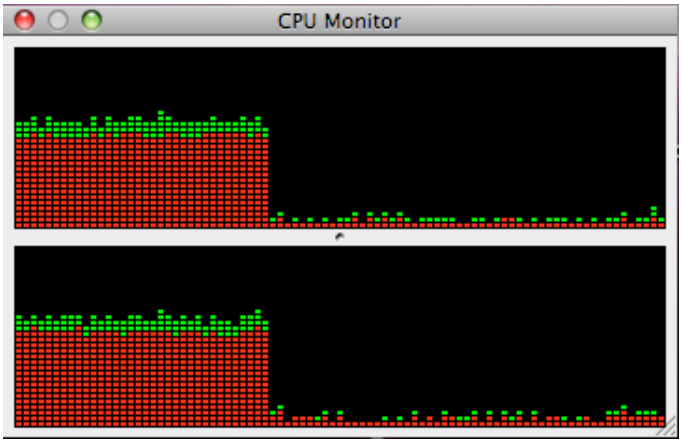

Fig. 55 CPU usage during the hang, and after Soundflower was force quit

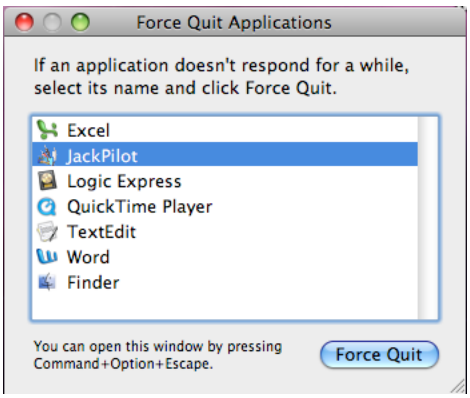

Fig. 56 The Force Quit window, which is invoked using, keystrokes Option-Apple-Escape (Apple, 2008)

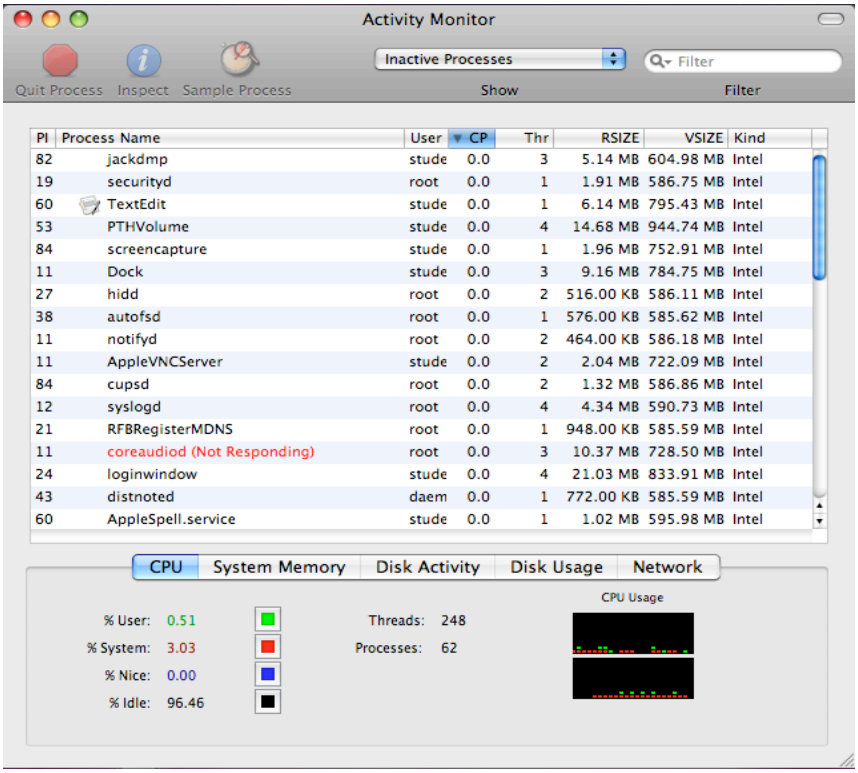

Fig. 57 Activity Monitor (Apple, 2008) shows CoreAudio has hung. Jackdmp was still active with 3 threads but refused to force-quit.

A basic setup of bringing 2 channels of audio into Pro Tools (v7.4) from Reason (v4) using the Rewire plugin was also connected successfully. An incompatibility with MIDI was encountered while testing Rewire and is detailed in Appendix IX.
The following tables indicate if any two of the trialled applications were basically compatible when tested together. It should be noted that comprehensive compatibility testing of all the practical combinations would take an enormous amount of time and was not possible in the timeframe allotted for this research.

	JACK	Rewire	Audio Hijack	Line In	Sound Menu	Sound Source	PTH Vol	WireTap Anywhere	Digidesign CoreAudio	Sound flower
JACK		Y	Y	Y	Y	Y	Y	Y	Y	Y
Rewire			Y	Y	Y	Y	Y	Y	N	Y
Audio Hijack				Y	Y	Y	Y	Y	Y	Y
Line In					Y	Y	Y	Y	Y	Y
Sound Menu						Y	Y	Y	Y	Y
SoundSource							Y	Y	Y	Y
PTHVolume								Y	Y	Y
WireTap Anywhere									NT	Y
Digidesign CoreAudio driver										Y
Soundflower										

Table 12 Compatibility chart 1. Does the software work while other software is active? Long term testing was not implemented on all programmes, so this is only a guide – your mileage may vary. NA = not applicable, NT =not tested.

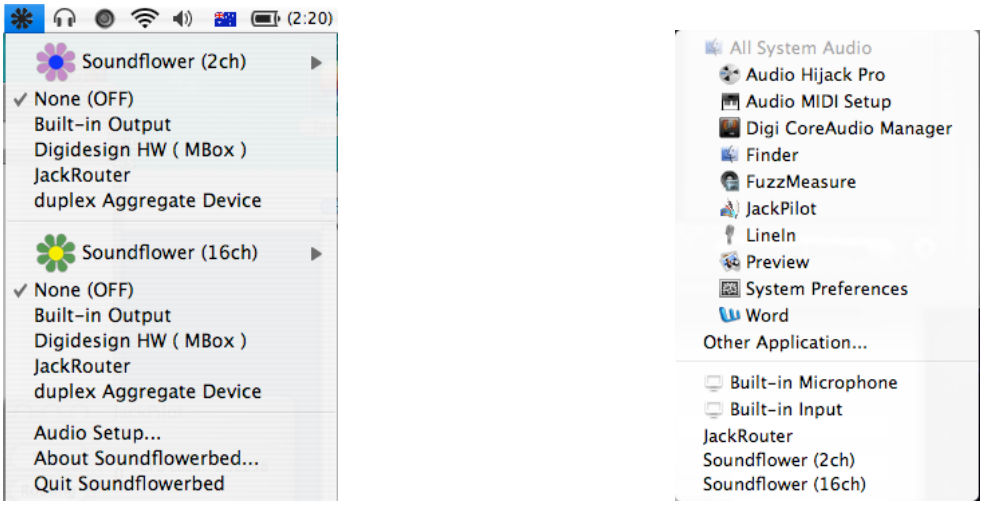

Fig. 58 (left image) Soundflower (Cycling74, 2008) is able to sense the Digidesign CoreAudio driver, and JACK on OS 10.4.11 but does not see the Digidesign CoreAudio driver on OS 10.5.4; (right image) the WireTap Anywhere (Ambrosia, 2008) 'add source' menu. It can see almost any other audio application.

Acknowledges ->										
	JACK	Rewire	Audio Hijack	Line In	Sound Menu	Sound Source	PTH Vol	WireTap Anywhere	Digi Core Audio	Sound flower
JACK		N	N	N	NA	NA	NA	N	N	N
Rewire	N		N	N	NA	NA	NA	N	NT	N
Audio Hijack	Y			N	NA	NA	NA	N	N	Y
Line In	NA	N	N		NA	NA	NA	N	NT	Y
Sound Menu	N	N	N	N		NA	NA	N	N	Y
SoundSource	N	N	N	N	NA		NA	N	N	Y
PTHVolume	Y	N	N	N	NA	NA		N	N	Y
WireTap Anywhere	Y	N	N	NA	NA	NA	NA		N	Y
Digidesign CoreAudio driver	N	N	N	N	N	N	N	N		N
Soundflower	Y	N	N	N	N	N	N	N	Y	

Table 13 Compatibility chart 2. Does the software acknowledge other software? NA = not applicable, NT =not tested.

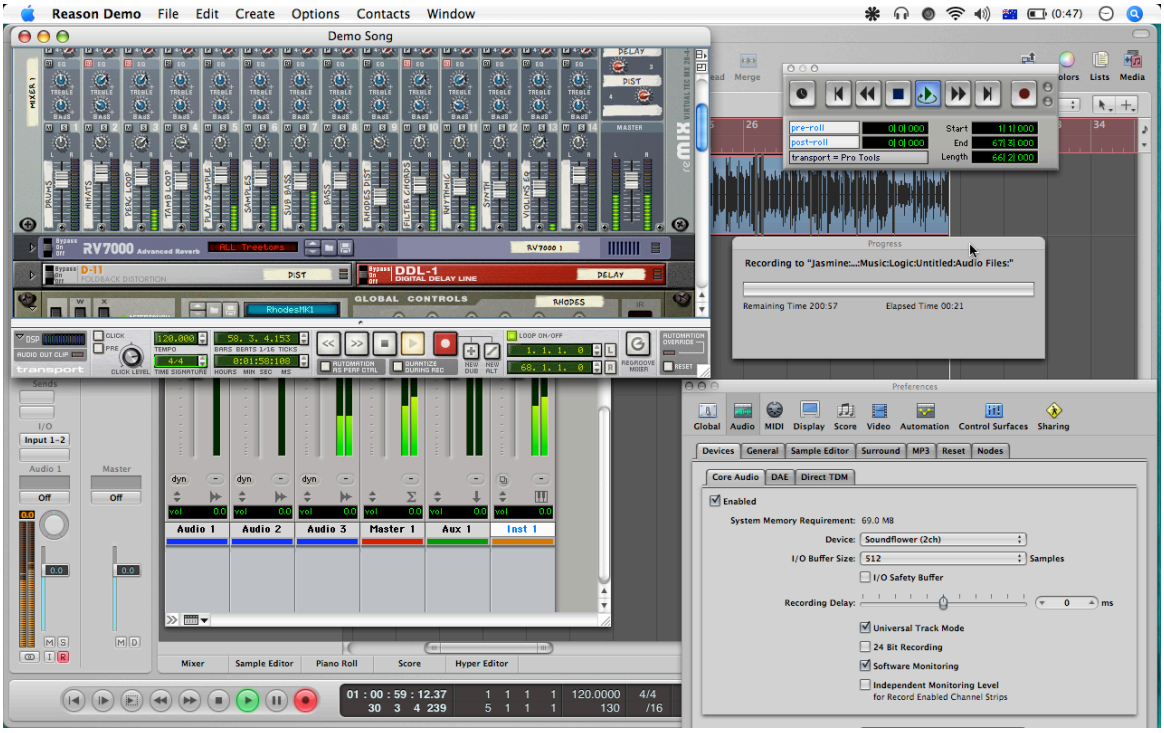

Fig. 59 Pro Tools/Rewire/Reason are outputting audio to an Mbox. At the same time, QuickTime Player is streaming audio through Soundflower to be recorded into Logic Pro (MacBook)

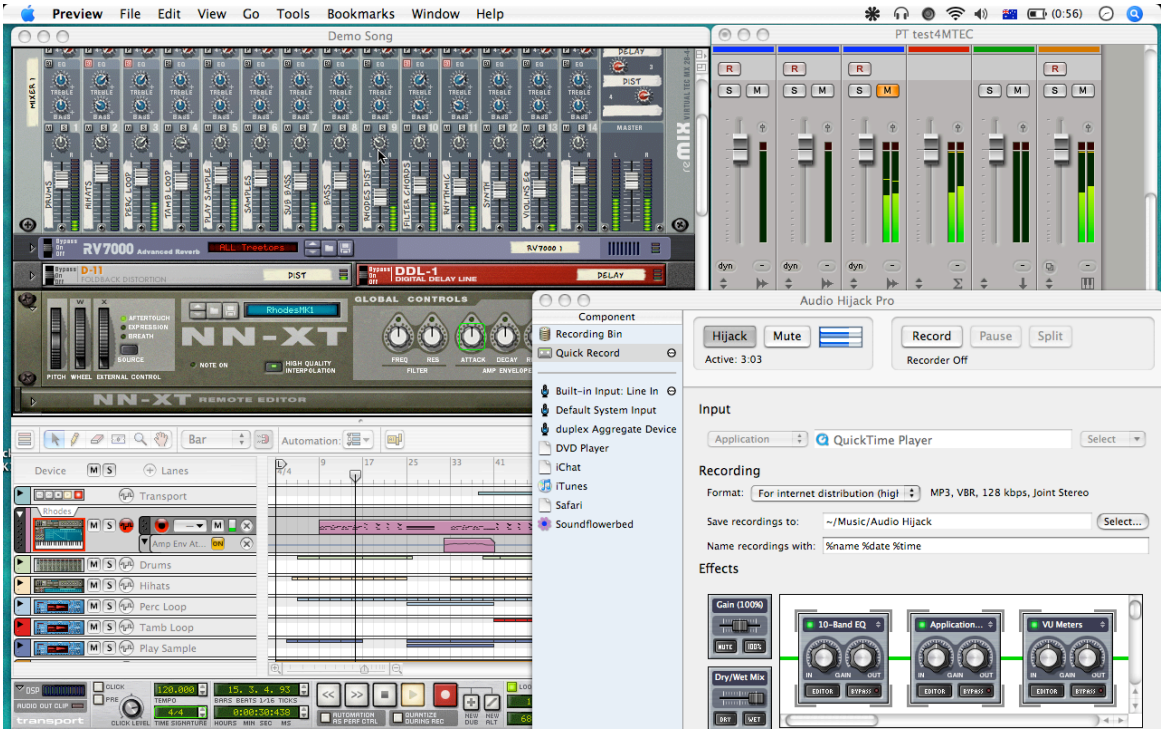

Fig. 60 Here, Pro Tools/Rewire/Reason are outputting audio to an Mbox while Audio Hijack Pro captures audio from the QuickTime Player (MacBook)

Pro Tools provides a special case as it has its own audio API; the Digidesign Audio Engine (DAE). This is a unique audio routing solution involving both kernel extensions and a user layer Framework. I was interested to see if it was possible to run a Pro Tools session simultaneously

with audio streaming though JACK and discovered that this was quite possible on an Intel iMac, with no glitches or hangs.

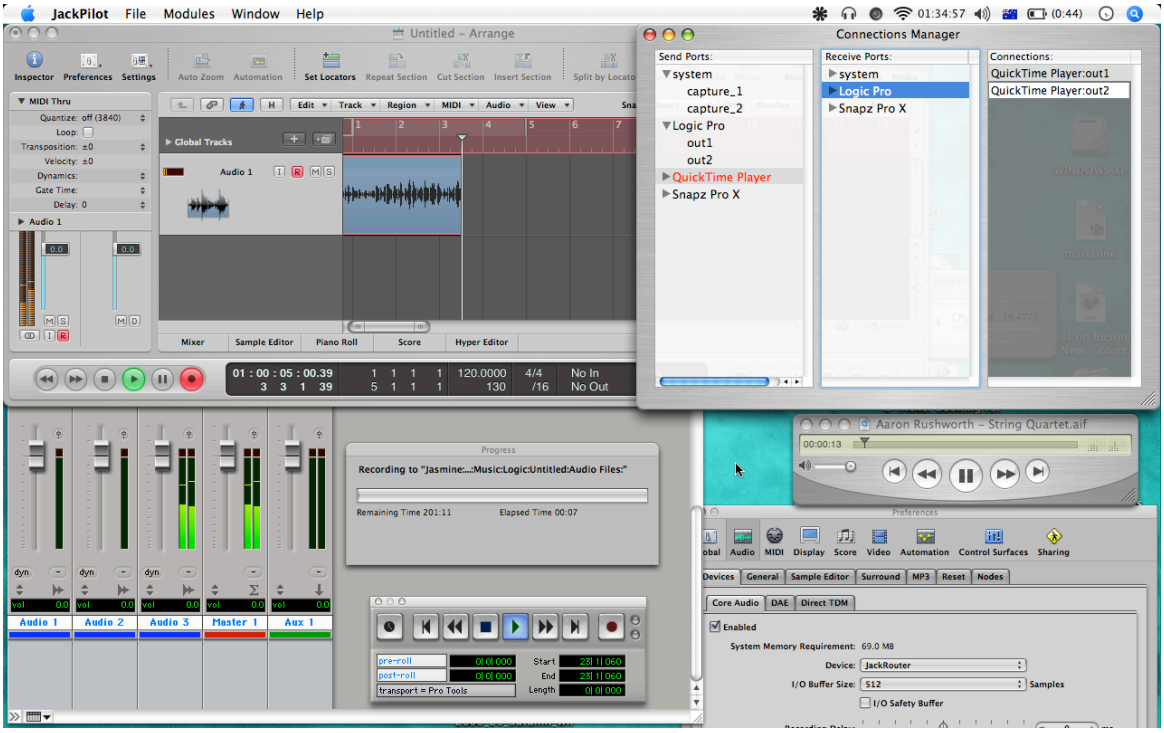

Fig. 61 Here, audio is playing from Pro Tools to an Mbox. At the same time audio is routed through JACK from the QuickTime Player and recorded in Logic Pro.

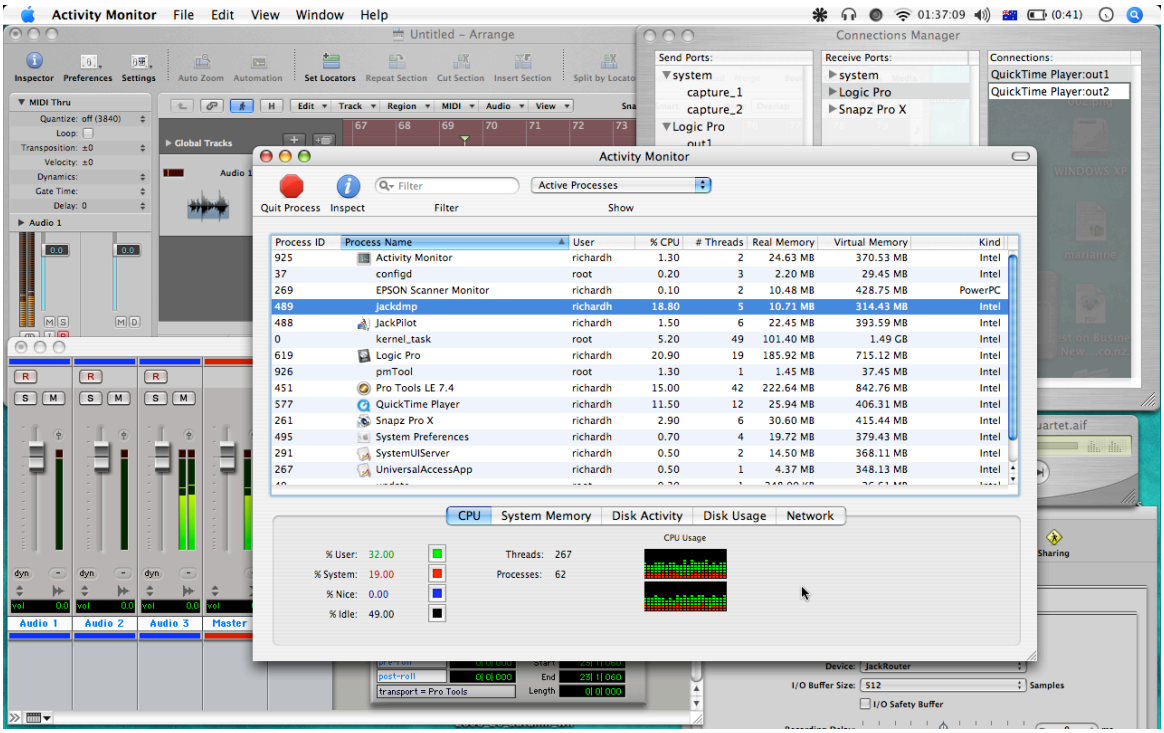

Fig. 62 Activity Monitor shows that CPU usage is moderate (JACK 20%, Logic 21%, Pro Tools 15%, and QuickTime 12%.

It was also possible to use a Pro Tools session simultaneously with Soundflower. As the DAE runs completely separately from CoreAudio it simply ignores either JACK or Soundflower.

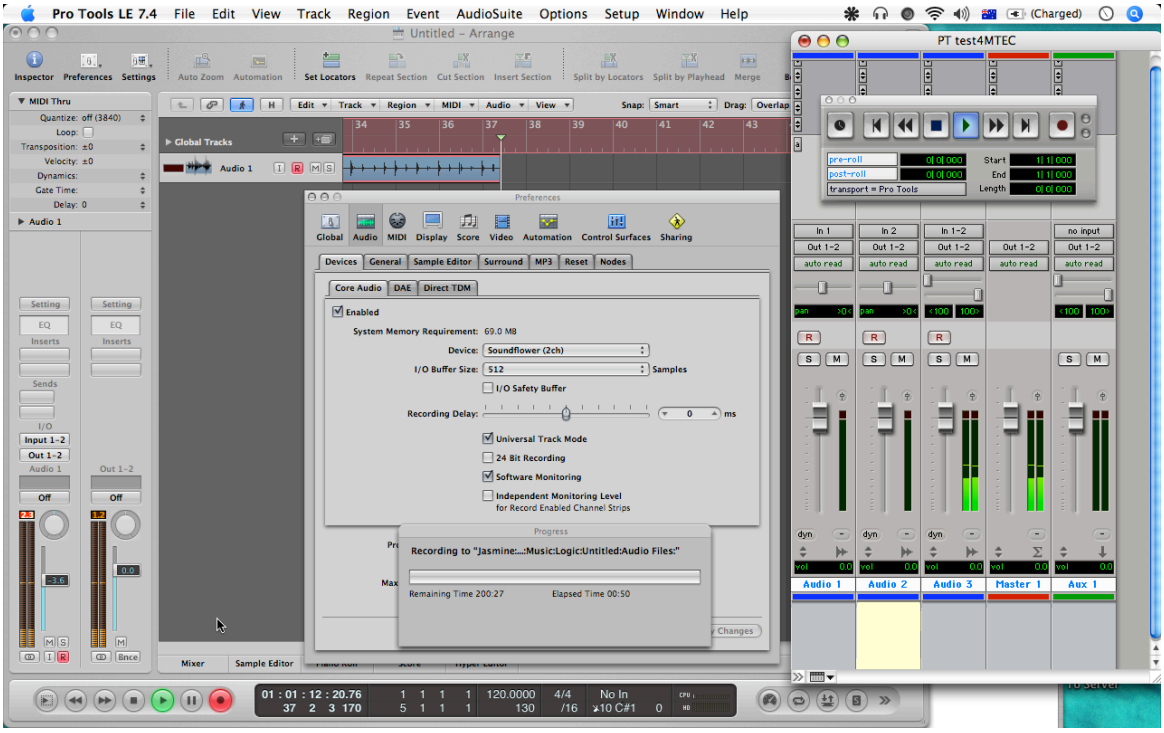

Fig. 63 Playing audio from Pro Tools to an Mbox. At the same time audio is routed through Soundflower from the QuickTime Player and recorded in Logic Pro.

Hugo Schotman has described setup for podcasting that uses both Audio Hijack Pro and Soundflower. Audio Hijack Pro is used to record and Soundflower routes the sound in and out of the setup. He identifies that Soundflowerbed has given some problems when used to monitor Soundflower. A partial solution has been to reset the sample rate to 44.1kHz as it defaults to 48kHz. A workaround is to use the Auxiliary Device Output instead of Soundflowerbed to monitor the mix, or to have an additional Audio Hijack session to send audio from Soundflower to the headphones. Schotman notes that the disadvantage to these methods is increased latency (Schotman, 2005).

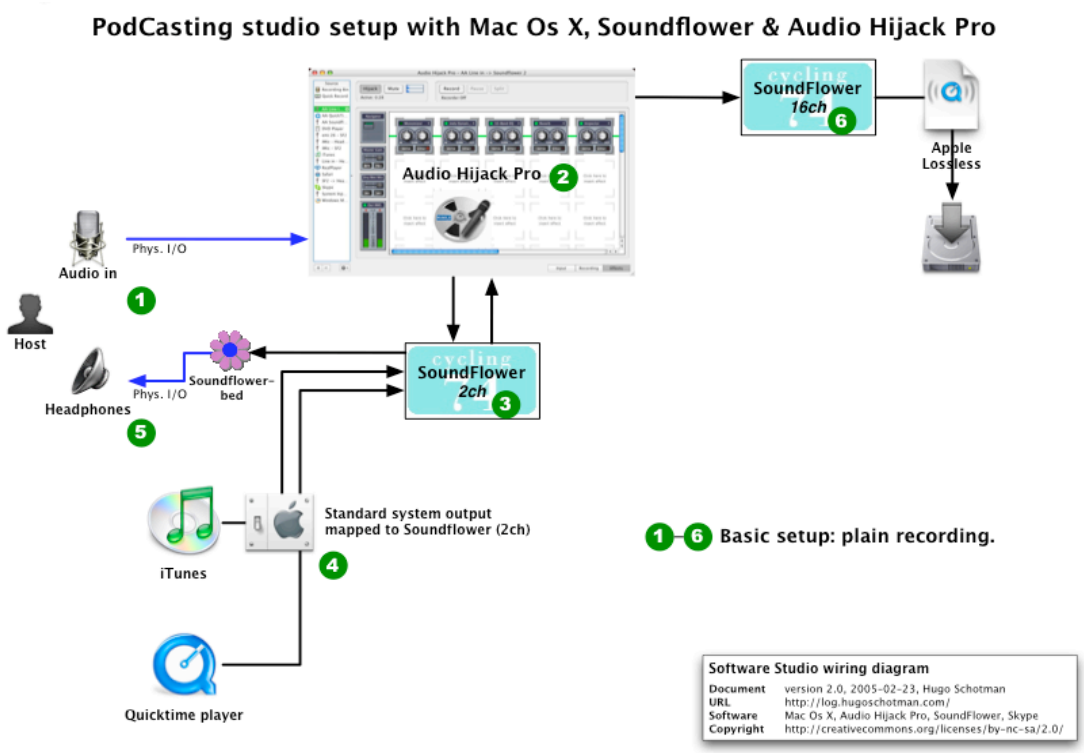

Fig. 63 Podcasting setup using Soundflower and Audio Hijack Pro (Schotman H., 2005)

X. A Rewire MIDI Problem

Usually the Rewire MIDI setup is straightforward as MIDI data comes in from an external keyboard and drives both Reason and a Pro Tools MIDI track. On this occasion I had no external keyboard and used a small virtual keyboard application called midikeys (http://www.manyetas.com/). The setup was to generate a simple MIDI sequence in Logic Pro (v8) and send that to Pro Tools and Reason via midikeys, using two IAC busses, as follows:

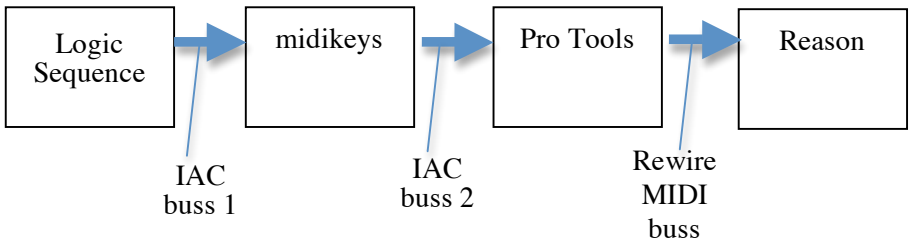

Fig. 64 The inter-application MIDI setup

When the Rewire plugin was activated Reason failed to load, its CPU usage went to 90%, and it could not be force quit. Furthermore Reason or Rewire was holding Pro Tools and not allowing it to be quit.

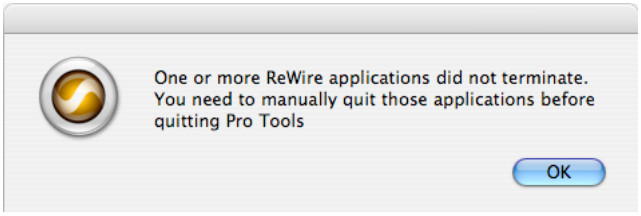

Fig. 65 The Pro Tools error message (Digidesign, 2008)

The problem was fixed by first opening Reason in stand alone mode and setting up the MIDI input by disabling the USB MIDI port driver and enabling the IAC buss after the IAC driver was enabled in Audio MIDI Setup. Once this was done the normal sequence of starting Reason automatically by inserting the Rewire plugin into Pro Tools allowed normal operation.

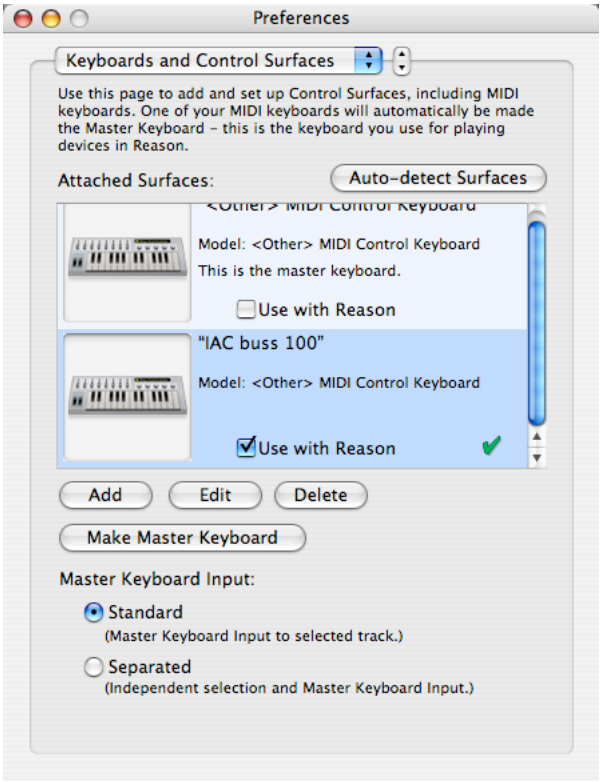

Fig. 66 The Reason Preferences window where the MIDI input device is set (Propellerhead, 2008)

XI. Software Resources

Sound Menu v1.5.1
Mort D'ivresse
Aspirine Software Products
http://www.aspirine.li

Jack v0.77, 0.78
Paul Davis, Stephane Letz, Johnny Petrantoni, Dan Nigrin
http://www.jackaudio.org
http://www.jackosx.com
http://sourceforge.net/projects/jackosx

QJackCtl v0.3.2
Rui Nuno Capela
http://qjackctl.sourceforge.net

Soundflower v1.2.1, 1.3, 1.3.1, 1.4
Cycling '74
30 Clementina Street
San Francisco, CA 94103
USA
http://www.cycling74.com
http://code.google.com/p/soundflower/

Rewire
Propellerhead Software
Rosenlundsgatan 29c
11863 Stockholm
Sweden
http://www.propellerheads.se/

Pro Tools LE 7.4
Digidesign
Avid Technology, Inc.
Avid Technology Park
One Park West Tewksbury,
MA 01876
U.S.A.
http://www.digidesign.com/

VST Systemlink
Steinberg Media Technologies GmbH
Neuer Hoeltigbaum 22-32
22143 Hamburg
Germany
http://www.steinberg.net/en/home.html

WireTap Studio v1.0.4, WireTap Anywhere v1.0.1
Ambrosia Software
PO Box 23140
Rochester, NY 14692
USA
http://www.AmbrosiaSW.com

Audio Hijack Pro v2.8.1, Detour v1.5.5, Line In v2.0.3, SoundSource v2.0
Rogue Amoeba
22 Kidder Ave. #3
Somerville, MA 02144
USA
www.rogueamoeba.com

Apple OS X 10.4.11 (Tiger), Apple OS X 10.5.3, 10.5.4, 10.5.5 (Leopard), including System Preferences/Sound v3.0, Audio MIDI Setup v2.2.2, CoreAudio v3.1.0, Activity Monitor v10.5
Apple QuickTime v7.4.5, 7.5, Apple iTunes v7.6
Apple Logic Pro 8
Apple
1 Infinite Loop , Cupertino, CA 95014
USA
http://www.apple.com/

PTHVolume v2.2.0
PTH Consulting
Flower Mound,
Texas,
USA
http://pth.com

XII. Glossary

ADC
Analogue to Digital Converter

AGGREGATE DEVICE
A feature of the AMS that allows for inputs and outputs on multiple hardware devices to be addressed as one virtual device.

AMS (AUDIO MIDI SETUP)
The OS X version of MIDI Manager or OMS, the AMS allows the user to select audio devices, and configure a virtual MIDI patchbay. It provides multiple configurations, device naming, channel filtering. In version 10.3 (Panther) the IAC has been included with unlimited busses. There is support for USB, firewire, PCMCIA, and PCI. OMS was developed for Opcode by Doug Wyatt, who moved to Apple to develop AMS.

API
Application Programming Interface
A high level set of computer instructions provided as part of the operating system for the purpose of providing easy and standard procedures with which an application can call the OS.
There are five APIs available as part of OS X; namely, POSIX, Cocoa, Carbon, Java, and Toolbox. Audio routing software such as JACK or Rewire can also be considered APIs.
Linux has several audio APIs, the most common being OSS, ALSA, and LADSPA.

ASIO
Audio Stream Input Output
ASIO is a low latency, sample accurate audio API developed by Steinberg software. It provides callbacks and double buffering to achieve this. On OS X it has been largely superceded by CoreAudio.

AU
Audio Unit
The native format for OS X audio plugins.

AUGRAPH
TheAUGraph is a high-level representation of a set of AudioUnits, along with the connections between them. It provides for realtime routing changes, and maintaining representation even when AudioUnits are not instantiated.

ASYNCHRONOUS
A system of data communication where no clocking is present. For MIDI, start and stop bits must be added to the data to indicate what the transmission state is to the receive device register. Asynchronous Transfer Mode (ATM) is an example of a sophisticated method of streaming audio and video on the Internet. Instead of data packets it uses ‘cells’ with a standard data length of 48 bytes to avoid jitter and delay.

BUFFER MEMORY
A relatively small amount of RAM dedicated to compensate for different speeds of input and output data. Usually associated with some hardware device (eg hard disk, video card). Buffer under-runs occur if output demand exceeds data supply; buffer over-runs are where output data cannot be used fast enough. Inter-application audio streaming also requires buffer memory (see also Latency).

CALLBACK
A type of computer instruction, which allows a low level process to call a function defined in a higher level. The advantage of this method of programme execution for audio routing is that it allows relative isolation between the driver and the audio data.

CORE MIDI
MIDI implementation in OS X. Features include: applications can share multi-port MIDI interfaces, inter-application bussing, USB MIDI class specification compliance, and low latency (< 1mS). Jitter is designed to be less than 200µS.

CPU
Central Processing Unit
The microprocessor that handles overall control of the computer system and is responsible for most of the data processing.

DAC
Digital to Analogue Converter

DIGIDESIGN COREAUDIO DRIVER
The Digidesign CoreAudio Driver is a single-client, multichannel sound driver that allows CoreAudio-compatible applications to record and play back through Digidesign hardware. Full-duplex recording and playback of 24-bit audio is supported at sample rates up to 96 kHz. The CoreAudio Driver provides up to 18 channels of I/O depending on the Pro Tools System. Buffer sizes can be set from 128 to 2048 samples.

DUPLEX
½ duplex means that data can be transferred only in one direction at a time. Full duplex means that simultaneous bi-directional data communication can occur.

FIFO (Named Pipe)
A method of inter-process communication. In computing a Pipeline is a method for connecting the output of one process into the input of another. Named Pipes are created and deleted outside of the attached process. FIFO stands for “First In, First Out” and refers to the property that the order of bytes going in is the same coming out.

FRAME
A time-coincident set of samples of the various channels in the audio stream. For PCM audio 1 Frame = 1 Packet.

FRAMEWORK
A type of software bundle that packages a dynamic shared library (executable code) with the resources that the library requires.

GLITCH
A transient interruption to the audio signal.

GUI
Graphical User Interface

HAL
Hardware Abstraction Layer
A high level software layer between the hardware and application so that it can communicate with the hardware in a consistent way, and without needing to address the specifics of the hardware.

JACK
Jack Audio Connection Kit

JACKDMP
The JACK server.

KERNEL
The core or lowest level of the operating system.

LATENCY
The delay between the time of input stimulus and observed output. In a computer handling audio this will generally be the time difference between when audio enters the system and when it leaves the system. As audio must be streamed in and/or out of the computer in real time without glitching a buffer memory is required. The main factor affecting latency is therefore buffer size. Other considerations are the ADC and DAC conversion delays, and CPU efficiency (ie how many clock cycles are needed to execute operations).

MIDI
Musical Instrument Digital Interface
A Simplex (unidirectional) asynchronous data communications protocol running at 31.25kb/s.

MIDI MANAGER
Apple's MIDI Manager offered a high level interface to the Mac OS to correctly support the timing accuracy required by MIDI hardware and software under MultiFinder. It was for doing Inter-Application Communication and for allowing multiple applications to address the serial port. MIDI Manager did not come with the System - it was available to developers or as licensed software with MIDI application packages.

MULTI-THREADING
Where a CPU can virtually be executing more than one series of instructions simultaneously (using a procedure called scheduling). The scheduler can set thread priorities.

MULTI-PROCCESSING
Sharing the CPU workload between two or more processors. The Intel Core 2 Duo technology effectively combines two processors on one chip.

OMS
OMS (Open MIDI System) is similar to MIDI Manager in that it extends the Mac OS for MIDI applications. It has some features not found in MIDI Manager such as SMPTE synchronization (SMPTE synchronization is the job of MIDI Time Code; it does not require OMS or MIDI Manager). OMS was essential to many pre OS X Macs for achieving full MIDI functionality. OMS allows an application to address a large number of discrete MIDI cables through one serial port, and also allows real-time IAC (Inter-Application Communication) between applications, like a sequencer and a softsynth (maximum of 4 busses). A feature is the OMS patchnames library.
Development ceased at version 2.3.8. It was supplied free to all interested MIDI/Music developers.

OPEN SOURCE
Software where the Source code is provided freely to users (usually along with the compiled application). Some limitations of use and distribution of the software may still apply but will usually allow anyone to use, extend, fix, or modify the software.

OPERATING SYSTEM (OS)
A software interface between the hardware, the user interface, and the application software.

PLUGIN
An application which is designed only to work from within its host application. An audio plugin adds some extra sound processing or generating functionality and allows audio (and/or MIDI data) to automatically stream between the plugin and host applications. Parameters are controlled using the particular plugin's GUI.

PPC
Power Performance Computing
A RISC architecture CPU used in Macintosh computers (PowerMac series and later G series).

RAM
Random Access Memory
The amount of RAM is an important determiner of computer performance. The OS X POSIX API does not support locking pages into real memory. Page outs can cause audio dropouts, so running unnecessary applications should be avoided when running audio communication software.

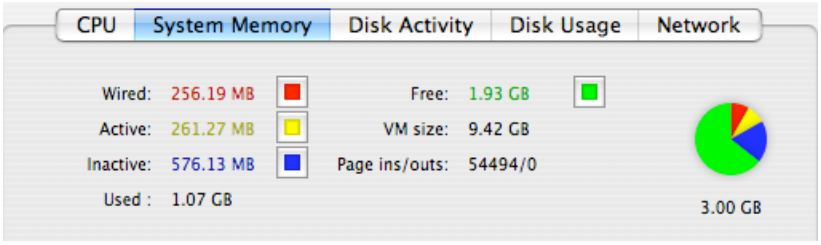

Fig. 67 The System Memory tab in Activity Monitor (Apple, 2008). The high ratio of page ins to page outs indicates a good surplus of RAM.

SYNCHRONOUS
Where multiple data is stepped in time, or 'clocked'. For digital audio to be truly synchronous each frame (ie sample word) of the multiple audio streams must be kept in step.

UNIX
An operating system started at Bell Laboratories in 1969. OS X uses the BSD version of UNIX.

XRUN
Either a buffer under-run or over-run. In the first case an application or the CPU is not fast enough delivering audio data to the buffer. In the second case the output from the buffer cannot be processed fast enough by an application or the CPU. XRUNs will usually result in audible pops.