Linux Audio: Origins & Futures

Paul Davis
Linux Audio Systems

Linux Plumbers Conference, 2009

Introduction

- Who am I
- Why am I here
- What is good about Linux audio support?
- What is bad about it?
- How did it get this way?
- What can we do about it?
- What should we do about it?

Who & Why

- First used Linux in 1994 at amazon.com
- Got involved in audio & MIDI software for Linux in 1998
- Helped with ALSA design and drivers in 98-2000, including RME support.
- In 2000, started work on ardour, a digital audio workstation.
- In 2002, designed and implemented JACK
- Worked full time on Linux audio software for more than 10 years

Who & Why: 2

- 2 Linux Desktop Architects meeting
- Encouraged adoption of PulseAudio for desktop use
- Continued dismay at the current state of audio support
- Continued dismay at the misinformation about the state of audio support
- Lessons from other operating systems

What audio support is needed?

- At one of the LDA meetings, I created a longer list. For now...
- Audio in and out of computer, via any available audio interface and/or any network connection
- Audio between any applications
- Share any and all available audio routing between applications
- Reset audio routing on the fly, either at user request or as h/w reconfiguration occurs
- Unified approach to “mixer” controls
- Easy to understand and reason about

Two Perspectives

- How do users interact with whatever audio support exists?
- How do applications interact with whatever audio support exists?
- My focus today is primarily on the latter, with some references to the former

History of Linux Audio Support

- Sometime in the early 1990's, Hannu Savolainen writes drivers for the Creative Soundblaster. Later extends this to form the Open Sound System (OSS), which also runs on other Unixes. Hannu and others add support for more devices
- By 1998, dissatisfaction with the basic design of OSS is growing, Jaroslav Kysela begins work on the Advanced Linux Sound Architecture (ALSA)
- Then: state of the art: 16 bit samples, 48kHz sample rate, no more than 4 channels

History 2

- During 1999-2001, ALSA is redesigned several times to reflect new high end audio interfaces, along with attempts to “improve” the API.
- Frank Van Der Pol implements the ALSA Sequencer, a kernel-space MIDI router and scheduler.
- Near the end of 2001, ALSA is adopted by the kernel community in favor of OSS as the official driver system for audio on Linux.
- OSS continues sporadic development, with NDAs and closed source drivers.

History 3

- 2000-2002: the Linux Audio Developers community discusses techniques to connect applications to each other
- 2001-2002: JACK is implemented as a spin-off from Ardour's own audio engine.
- 2001-present: work on reducing scheduling latency in the kernel begins, continues, improves. Letter from LAD community to Linus and other kernel developers regarding RT patches and access to RT scheduling

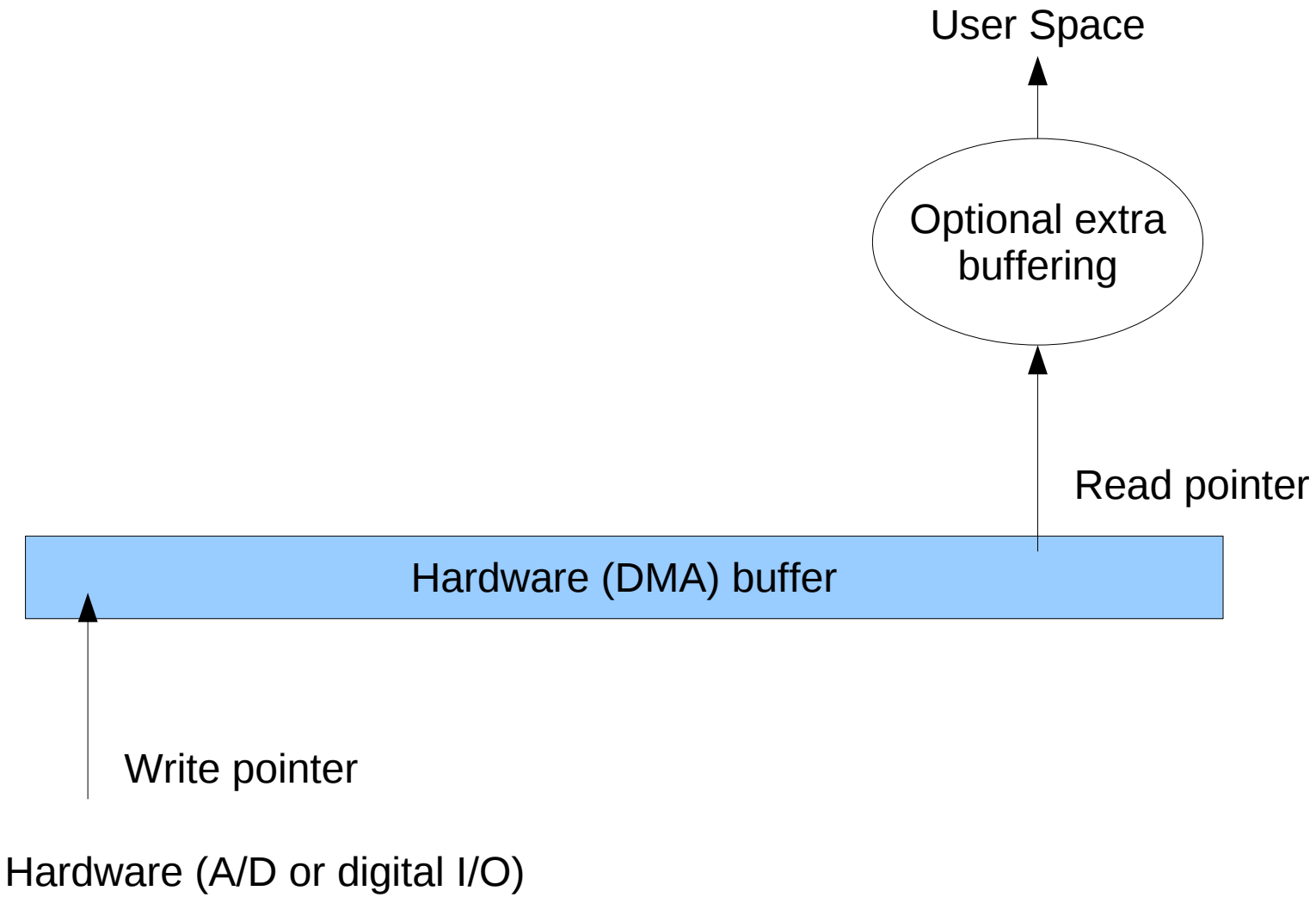
History 4

- Realtime patches from Morton/Molnar et al continue to improve kernel latency
- Access to realtime scheduling tightly controlled
- Requires more kernel patches to use
- Later decisions by kernel community add access to RT scheduling and memory locking to standard mechanisms
- Still requires per-system configuration to allow ordinary user access. Most distributions still do not provide this.

History 5

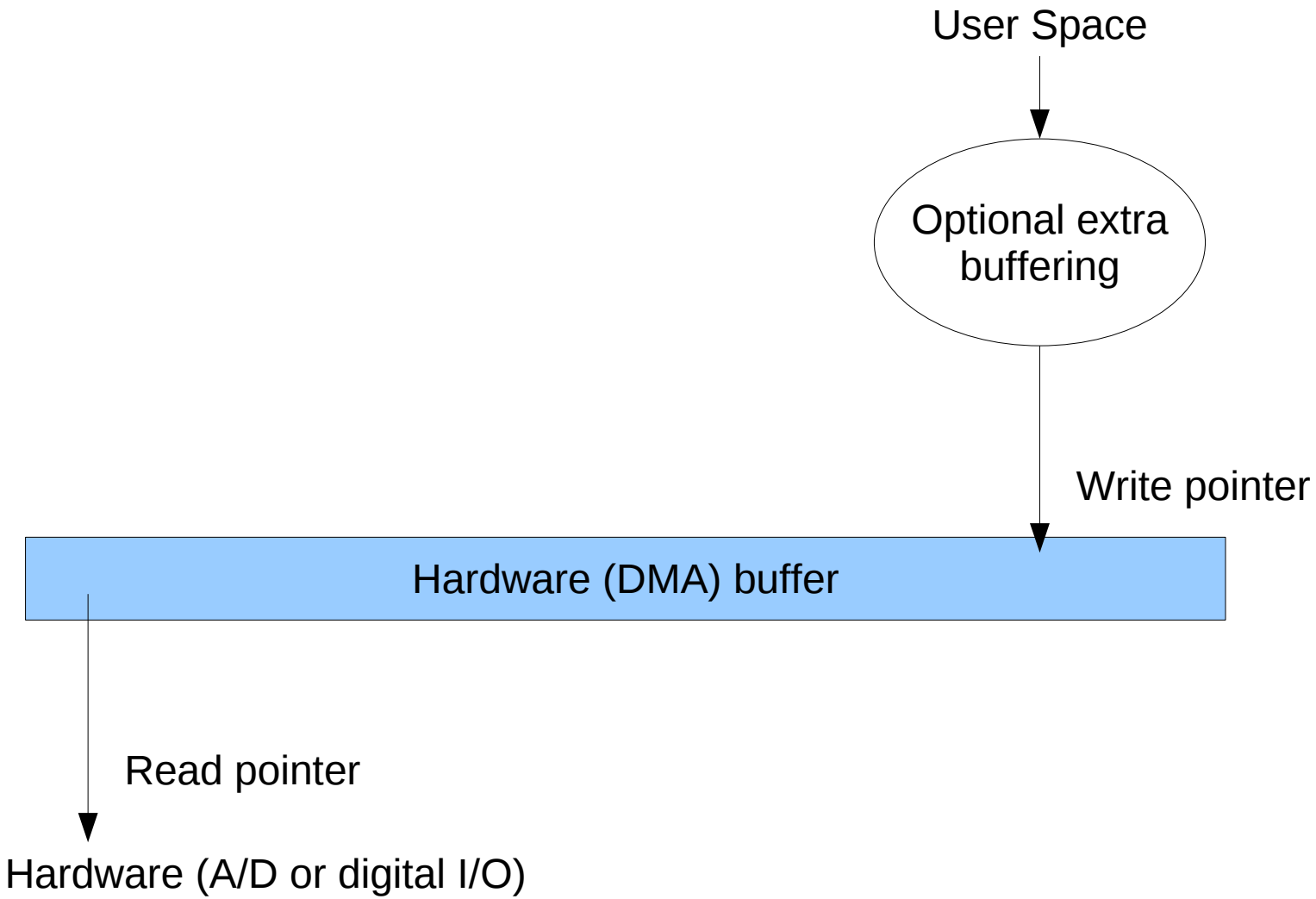
- Mid-2000's: Lennart begins work on PulseAudio
- KDE finally drops aRts as a sound server
- Gstreamer emerges for intra-application design
- Desktops want to provide “simple” audio access for desktop applications, start implementing their own libraries (Phonon, libsydney)
- JACK is the only way to access firewire audio devices
- Confusion reigns teh internetz

Audio H/W Basics



CAPTURE/RECORDING

Audio H/W Basics



PLAYBACK

Push & Pull Models

- Push: application decides when to read/write audio data, and how much to read/write.
- Pull: hardware or other lower levels of system architecture determine when to read/write audio data and how much to read/write.
- Push requires enough buffering in the system to handle arbitrary behaviour by the application
- Pull requires application design and behaviour capable of meeting externally imposed deadlines.

Key Point

- Supporting a push model on top of a pull model is easy: just add buffering and an API
- Supporting a pull model on top of a push model is hard, and performs badly.
- Conclusion: audio support needs to be based on the provision of a pull-based system. It can include push-based APIs layered above it to support application design that requires it.

How Should Applications Access Audio Support?

- OSS provided drivers that were accessed via the usual Unix open/close/read/write/ioctl/ select/mmap system calls.
- ALSA provides drivers that are accessed via a library (libasound) that presents a huge set of functions to control:
  - Hardware parameters & configuration
  - Software parameters & configuration
  - Different application design models

The Video/Audio Comparison

- Both involve some sort of medium that generates human sensory experience (vision/display, hearing/speakers)
- Both are driven by periodically rescanning a data buffer and then “rendering” its contents to the output medium
- Differences: refresh rate, dimensionality, effect of missing refresh deadlines
- Other than when using video playback applications, the desired output doesn't change very often (eg. GUIs)

Audio/Video Comparison 2

- Is anyone seriously proposing writing an application that wants to display stuff on a screen with open/read/write/close?
- For years, all access to the video device has been mediated by either a server, or a server-like API.
- Why is there any argument about this for audio?

The Problem with Unix

- open/read/write/close/ioctl/mmap are GREAT
- Lack temporal semantics
- Lack data format semantics
- Require that implicit services be provided in the kernel.
- Writing to a filesystem or a network: “deliver the bytes I give you, whenever”
- Reading from a filesystem or a network: “give me whatever bytes you discover, ASAP”

Unix Issues 2

- open/read/write/close/ioctl/select/mmap CAN be used to interact with realtime devices
- BUT ... does not promote a pull-based application design, does not encourage considering temporal aspects, does not deal with data formats.
- Conclusion: inappropriate API for interacting with video or audio interfaces.
- Yes, there are always special cases.

What we want

- A server-esque architecture, including a server-esque API
- API explicitly handles:
  - Data format, including sample rate
  - Signal routing
  - Start/stop
  - Latency inquiries
  - Synchronization
- Server handles device interaction
- If multiple APIs, stack them vertically.

What we don't want

- Services required to be in the kernel because the API requires it
- Pretending that services are in the kernel when they are actually in user space (via user space FS)
- Applications assuming that they can and should control hardware devices
- Push-model at the lowest level
- Lots of effort spent on corner cases

What We Can (maybe) Factor Out

- We don't need to provide data format support or sample rate conversion in a system audio API.
- But we do, and we can, and we could.
- Alternative is to make sure that there are clearly identified, high quality libraries for format conversion and SRC.
- We don't need to provide the illusion of device control, even for the hardware mixer.
- Can be left to specialized applications.

OSS API MUST DIE!

- OSS provides an API that requires that any services provided are implemented in the kernel
- OSS provides an API that allows, even encourages, developers to use application design that doesn't play well with others
  - No temporal semantics
  - No signal routing
  - Explicit (if illusory) control of a hardware device
- Some claim the drivers “sound better”. Not relevant.

Why didn't we fix this in 2000?

- Back compatibility with OSS was felt to be important
- No heirarchy of control to impose a new API
- Even today, it is still impossible to
  - Stop users from installing OSS as an alternative to ALSA
  - Stop developers from writing apps with the OSS API
- ALSA wasn't really that different from OSS anyway
- Didn't believe that RT in user space could work

CoreAudio's introduction on OS X

- Apple's MacOS 9 had a limited, crude and simplistic audio API and infrastructure
- Apple completely redesigned every aspect of it for OS X
- Told developers to migrate – no options, no feedback requested on basic issues.
- Today, CoreAudio is a single API that adequately supports desktop, consumer media and professional applications.

The Windows Experience

- MME, WDM, WaveRT
- Windows has taken much longer than Linux or OS X to support really low latency audio I/O
- Windows has retained back compatibility with older APIs
- Current and future versions layer push model user space APIs on top of pull model system architecture.
- Driver authors forced to switch. Application developers could switch if desired.

JACK & PulseAudio

- Different targets
- JACK: low latency is highest priority, sample synchronous execution of clients, single data format, single sample rate, some h/w requirements. Pro-audio & music creation are primary targets.
- Pulse: application compatibility and power consumption are high priorities, desktop and consumer usage dominate feature set, few h/w requirements.

Do We Need Both?

- Right now, absolutely
- JACK conflicts with application design in many desktop and consumer apps.
- PulseAudio incapable of catering to pro-audio and music creation apps; offers valuable services for desktops and consumer uses.
- Similar to the situation on Windows until Vista
- MME et al (desktop/consumer, 30msec of kernel mixing latency) + ASIO (similar design to JACK, used by almost all music creation apps)

Do We Have A Stick?

- Is it feasible to force adoption of a new API on Linux?
- If so, how?
- If not, what are the implications for attempting to improve the audio infrastructure?
  - Continued existence of subsystems with different design goals (e.g. PulseAudio and JACK)
  - Continued support required for older APIs
  - Continued confusion about what the right way to do audio I/O on linux really is

Where is the mixer?

- only one audio interface, no support for mixing in hardware (most don't), access to the device by multiple apps requires a mixer, somewhere.
- CoreAudio and WaveRT (Windows) puts it in the kernel
- OSS4 (“vmix”) puts it in the kernel
- JACK, PulseAudio put it in user space
- Which is right? Why? What about SRC?
- Note: user space daemons look different to users

The Importance of Timing

- Current ALSA implementation relies on device interrupts and registers to understand where the h/w read and write ptrs are
- Doesn't work well for devices where the interrupts are asynchronous WRT the sample clock (USB, FireWire, network audio)
- Much better to use a DLL (or similar 2nd order control system) to predict positions
- Supporting different SR's (and even different access models) much easier because accurate time/position predictions are always available

A Unified API?

- Can it be done?
- CoreAudio says yes
- No inter-application routing, but JACK shows that can be done w/CoreAudio too
- What do application developers use? One library? Many libraries?
- What happens to device access? (hint: think about X Window, DirectFB etc)

Thanks...

- Lennart for the invitation
- Linux Foundation for travel funding
- Dylan & Heidi for the bed
- No thanks to Delta Airlines, US Airways and Alaska Airlines for taking 13 hours to get me across the country.
- The Ardour community for making it possible for me to work full time on libre audio & MIDI software.

Linux Audio:
Origins & Futures

Paul Davis
Linux Audio Systems

Introduction

- Who am I
- Why am I here
- What is good about Linux audio support?
- What is bad about it?
- How did it get this way?
- What can we do about it?
- What should we do about it?

Who & Why

- First used Linux in 1994 at amazon.com
- Got involved in audio & MIDI software for Linux in 1998
- Helped with ALSA design and drivers in 98-2000, including RME support.
- In 2000, started work on ardour, a digital audio workstation.
- In 2002, designed and implemented JACK
- Worked full time on Linux audio software for more than 10 years

Who & Why: 2

- 2 Linux Desktop Architects meeting
- Encouraged adoption of PulseAudio for desktop use
- Continued dismay at the current state of audio support
- Continued dismay at the misinformation about the state of audio support
- Lessons from other operating systems

What audio support is needed?

- At one of the LDA meetings, I created a longer list. For now...
- Audio in and out of computer, via any available audio interface and/or any network connection
- Audio between any applications
- Share any and all available audio routing between applications
- Reset audio routing on the fly, either at user request or as h/w reconfiguration occurs
- Unified approach to “mixer” controls
- Easy to understand and reason about

Two Perspectives

- How do users interact with whatever audio support exists?
- How do applications interact with whatever audio support exists?
- My focus today is primarily on the latter, with some references to the former

History of Linux Audio Support

- Sometime in the early 1990's, Hannu Savolainen writes drivers for the Creative Soundblaster. Later extends this to form the Open Sound System (OSS), which also runs on other Unixes. Hannu and others add support for more devices
- By 1998, dissatisfaction with the basic design of OSS is growing, Jaroslav Kysela begins work on the Advanced Linux Sound Architecture (ALSA)
- Then: state of the art: 16 bit samples, 48kHz sample rate, no more than 4 channels

History 2

- During 1999-2001, ALSA is redesigned several times to reflect new high end audio interfaces, along with attempts to “improve” the API.
- Frank Van Der Pol implements the ALSA Sequencer, a kernel-space MIDI router and scheduler.
- Near the end of 2001, ALSA is adopted by the kernel community in favor of OSS as the official driver system for audio on Linux.
- OSS continues sporadic development, with NDAs and closed source drivers.

History 3

- 2000-2002: the Linux Audio Developers community discusses techniques to connect applications to each other
- 2001-2002: JACK is implemented as a spin-off from Ardour's own audio engine.
- 2001-present: work on reducing scheduling latency in the kernel begins, continues, improves. Letter from LAD community to Linus and other kernel developers regarding RT patches and access to RT scheduling

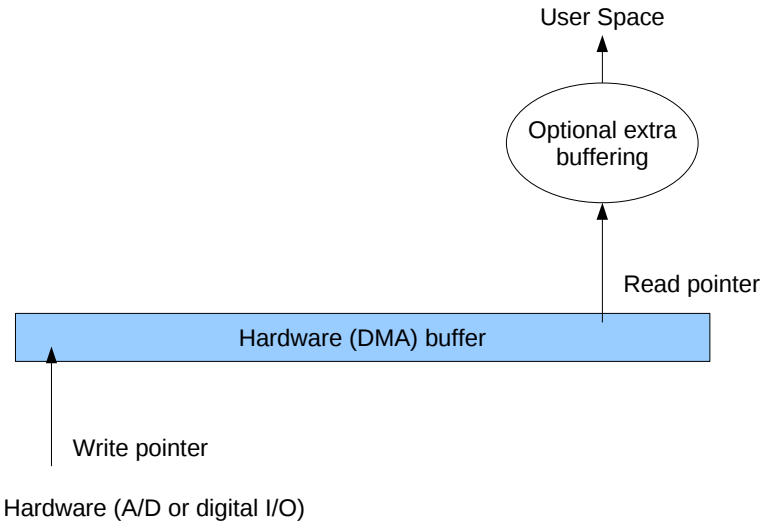
History 4

- Realtime patches from Morton/Molnar et al continue to improve kernel latency
- Access to realtime scheduling tightly controlled
- Requires more kernel patches to use
- Later decisions by kernel community add access to RT scheduling and memory locking to standard mechanisms
- Still requires per-system configuration to allow ordinary user access. Most distributions still do not provide this.

History 5

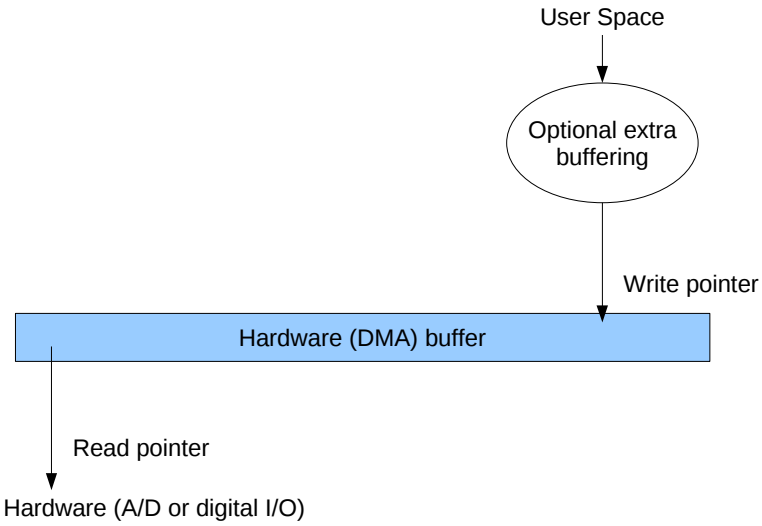
- Mid-2000's: Lennart begins work on PulseAudio
- KDE finally drops aRts as a sound server
- Gstreamer emerges for intra-application design
- Desktops want to provide “simple” audio access for desktop applications, start implementing their own libraries (Phonon, libsydney)
- JACK is the only way to access firewire audio devices
- Confusion reigns teh internetz

Audio H/W Basics



CAPTURE/RECORDING

Audio H/W Basics



PLAYBACK

Push & Pull Models

- Push: application decides when to read/write audio data, and how much to read/write.
- Pull: hardware or other lower levels of system architecture determine when to read/write audio data and how much to read/write.
- Push requires enough buffering in the system to handle arbitrary behaviour by the application
- Pull requires application design and behaviour capable of meeting externally imposed deadlines.

Key Point

- Supporting a push model on top of a pull model is easy: just add buffering and an API
- Supporting a pull model on top of a push model is hard, and performs badly.
- Conclusion: audio support needs to be based on the provision of a pull-based system. It can include push-based APIs layered above it to support application design that requires it.

How Should Applications Access Audio Support?

- OSS provided drivers that were accessed via the usual Unix open/close/read/write/ioctl/ select/mmap system calls.
- ALSA provides drivers that are accessed via a library (libasound) that presents a huge set of functions to control:
  - Hardware parameters & configuration
  - Software parameters & configuration
  - Different application design models

The Video/Audio Comparison

- Both involve some sort of medium that generates human sensory experience (vision/display, hearing/speakers)
- Both are driven by periodically rescanning a data buffer and then “rendering” its contents to the output medium
- Differences: refresh rate, dimensionality, effect of missing refresh deadlines
- Other than when using video playback applications, the desired output doesn't change very often (eg. GUIs)

Audio/Video Comparison 2

- Is anyone seriously proposing writing an application that wants to display stuff on a screen with open/read/write/close?
- For years, all access to the video device has been mediated by either a server, or a server-like API.
- Why is there any argument about this for audio?

The Problem with Unix

- open/read/write/close/ioctl/mmap are GREAT
- Lack temporal semantics
- Lack data format semantics
- Require that implicit services be provided in the kernel.
- Writing to a filesystem or a network: “deliver the bytes I give you, whenever”
- Reading from a filesystem or a network: “give me whatever bytes you discover, ASAP”

Unix Issues 2

- open/read/write/close/ioctl/select/mmap CAN be used to interact with realtime devices
- BUT ... does not promote a pull-based application design, does not encourage considering temporal aspects, does not deal with data formats.
- Conclusion: inappropriate API for interacting with video or audio interfaces.
- Yes, there are always special cases.

What we want

- A server-esque architecture, including a server-esque API
- API explicitly handles:
  - Data format, including sample rate
  - Signal routing
  - Start/stop
  - Latency inquiries
  - Synchronization
- Server handles device interaction
- If multiple APIs, stack them vertically.

What we don't want

- Services required to be in the kernel because the API requires it
- Pretending that services are in the kernel when they are actually in user space (via user space FS)
- Applications assuming that they can and should control hardware devices
- Push-model at the lowest level
- Lots of effort spent on corner cases

What We Can (maybe) Factor Out

- We don't need to provide data format support or sample rate conversion in a system audio API.
- But we do, and we can, and we could.
- Alternative is to make sure that there are clearly identified, high quality libraries for format conversion and SRC.
- We don't need to provide the illusion of device control, even for the hardware mixer.
- Can be left to specialized applications.

OSS API MUST DIE!

- OSS provides an API that requires that any services provided are implemented in the kernel
- OSS provides an API that allows, even encourages, developers to use application design that doesn't play well with others
  - No temporal semantics
  - No signal routing
  - Explicit (if illusory) control of a hardware device
- Some claim the drivers “sound better”. Not relevant.

Why didn't we fix this in 2000?

- Back compatibility with OSS was felt to be important
- No heirarchy of control to impose a new API
- Even today, it is still impossible to
  - Stop users from installing OSS as an alternative to ALSA
  - Stop developers from writing apps with the OSS API
- ALSA wasn't really that different from OSS anyway
- Didn't believe that RT in user space could work

CoreAudio's introduction on OS X

- Apple's MacOS 9 had a limited, crude and simplistic audio API and infrastructure
- Apple completely redesigned every aspect of it for OS X
- Told developers to migrate – no options, no feedback requested on basic issues.
- Today, CoreAudio is a single API that adequately supports desktop, consumer media and professional applications.

The Windows Experience

- MME, WDM, WaveRT
- Windows has taken much longer than Linux or OS X to support really low latency audio I/O
- Windows has retained back compatibility with older APIs
- Current and future versions layer push model user space APIs on top of pull model system architecture.
- Driver authors forced to switch. Application developers could switch if desired.

JACK & PulseAudio

- Different targets
- JACK: low latency is highest priority, sample synchronous execution of clients, single data format, single sample rate, some h/w requirements. Pro-audio & music creation are primary targets.
- Pulse: application compatibility and power consumption are high priorities, desktop and consumer usage dominate feature set, few h/w requirements.

Do We Need Both?

- Right now, absolutely
- JACK conflicts with application design in many desktop and consumer apps.
- PulseAudio incapable of catering to pro-audio and music creation apps; offers valuable services for desktops and consumer uses.
- Similar to the situation on Windows until Vista
- MME et al (desktop/consumer, 30msec of kernel mixing latency) + ASIO (similar design to JACK, used by almost all music creation apps)

Do We Have A Stick?

- Is it feasible to force adoption of a new API on Linux?
- If so, how?
- If not, what are the implications for attempting to improve the audio infrastructure?
  - Continued existence of subsystems with different design goals (e.g. PulseAudio and JACK)
  - Continued support required for older APIs
  - Continued confusion about what the right way to do audio I/O on linux really is

Where is the mixer?

- only one audio interface, no support for mixing in hardware (most don't), access to the device by multiple apps requires a mixer, somewhere.
- CoreAudio and WaveRT (Windows) puts it in the kernel
- OSS4 (“vmix”) puts it in the kernel
- JACK, PulseAudio put it in user space
- Which is right? Why? What about SRC?
- Note: user space daemons look different to users

The Importance of Timing

- Current ALSA implementation relies on device interrupts and registers to understand where the h/w read and write ptrs are
- Doesn't work well for devices where the interrupts are asynchronous WRT the sample clock (USB, FireWire, network audio)
- Much better to use a DLL (or similar 2nd order control system) to predict positions
- Supporting different SR's (and even different access models) much easier because accurate time/position predictions are always available

A Unified API?

- Can it be done?
- CoreAudio says yes
- No inter-application routing, but JACK shows that can be done w/CoreAudio too
- What do application developers use? One library? Many libraries?
- What happens to device access? (hint: think about X Window, DirectFB etc)

Thanks...

- Lennart for the invitation
- Linux Foundation for travel funding
- Dylan & Heidi for the bed
- No thanks to Delta Airlines, US Airways and Alaska Airlines for taking 13 hours to get me across the country.
- The Ardour community for making it possible for me to work full time on libre audio & MIDI software.