Universitat Politècnica de Catalunya

Final year project

User space approach to audio device driving on UNIX-like systems

Author:
Robert Millan Hernandez

Advisor:
Dr. J.R. Herrero Zaragoza

January 17, 2015

Acknowledgements

To my advisor, Dr. J.R. Herrero Zaragoza, for his teachings and guidance, for steering me in the right direction at each turning point, and overall for his generous support during development of this project.

To Dr. Ferran Sabaté for his supervision during the management course, his attention to detail, his many corrections and improvements and his feedback on management aspects of the project.

To Daniel Ruiz Muñoz, for LaTeX advice and allowing me to peek at some of his templates.

Abstract

The proposed project attempts to resolve some of the inconvenience factors related to implementing device drivers in kernel space, by using a different approach.

With the goals of simplifying development efforts and of maximizing performance, design decisions have been taken in the majority of UNIX-like operating systems to implement their kernels using a monolithic design and to build most device driver subsystems into them. This has a number of drawbacks which this proposal document will elaborate on.

The proposed project will attempt to resolve these drawbacks by implementing a driver framework entirely in user space. The proposal is to do this specifically for sound devices because this appears to be a very suitable class of drivers for this kind of experiment.

Abstract (Catalan)

El projecte que proposem pretén resoldre alguns dels inconvenients relacionats amb la implementació de controladors de dispositiu en espai de kernel, tot emprant un mètode diferent.

Amb els objectius de simplificar costos de desenvolupament i maximitzar el rendiment, la majoria dels sistemes operatius de tipus UNIX incorporen com a la implementació dels seus kernels amb un disseny monolític, on la majoria dels subsistemes de control de dispositius s'hi han integrat. Això té determinats inconvenients que seran abordats en aquesta proposta.

Aquest projecte intentarà resoldre aquests inconvenients mitjançant la implementació d'un entorn de controladors completament en espai d'usuari. La proposta consisteix en fer això específicament per a dispositius de so, els quals creiem que constitueixen una àrea molt adient per a aquest tipus d'experiment.

Contents

1 Introduction and state of the art

1.1 Introduction

Traditionally, UNIX-like operating systems have used a monolithic design for their kernels. This applies not just to logical facilities such as the network stack or filesystems, but also to the majority of device drivers. In general there is a good explanation for this design decision: a hardware resource is often shared by many processes (or even users) within the system; sharing requires a routine to manage this resource and multiplex its use requests. Such kind of management is more expensive to do in userland due to Inter-Process Communication (IPC) overhead, which explains why in most cases it is handled by the kernel.

The analysis of kernel-mode driver systems will be focused on Linux-based GNU systems. However, it is likely that similar conclusions can be extrapolated to other operating systems.

1.2 Research context

Previous research on user space device driving is very rich in its conclusions regarding the feasibility and usefulness of such systems.

1.2.1 Easier development

Most researchers who have conducted similar efforts agree that moving device drivers to user space provides a more suitable environment for driver development, allowing programmers to easily test their code without risk of bringing the system down, and making debugging much simpler [12] [17].

Decoupling from internal kernel Application Programming Interfaces (APIs) is also found to open many possibilities. For example, upgrading a device driver without having to upgrade the entire kernel is then possible. This could be very relevant on safety-critical industrial machines, where an upgrade of the entire kernel requires a very expensive certification process [23].

1.2.2 Robustness and system stability

Drivers in user space are also expected to be more robust/stable than their in-kernel counterparts. Research conducted using Linux as a sample code base found that device drivers are seven times more likely to contain coding mistakes than other kind of code [11]. Other experiments show that device drivers are the cause for 85% of operating system failures [21].

1.2.3 Performance

This kind of experiment is not without difficulties. Performance throughput drawbacks are common due to additional IPC overhead. However, this loss of performance has been found to diminish to neglectable levels when optimal block transfer levels are used [12].

1.2.4 Other areas of research

It is important to make a distinction between user space driving of peripheral devices and user space management of hardware resources that are shared by many hardware components and their respective drivers. For instance, it's been found that non-centralized handling of Interrupt Request (IRQ) logic leads to hard-to-track synchronization bugs, non-portable code, and priority inversion, whereas a centralizing IRQ management in a user space server is subject to IPC overhead with a direct impact on interrupt service latency [19].

Another interesting finding is that, although it is tempting to leverage the flexibility provided by user space device drivers to implement a security system (that is, making it possible to use untrusted code with unprivileged user ids), this is much harder than it may seem because drivers may easily obtain read or write access to arbitrary memory using Direct Memory Access (DMA) [12] [17]. Implementing user space drivers with such security constraints is a specialized area of research and is left outside the scope of this project.

1.3 Motivation

There are a few reasons why we think the proposed approach in this project could yield interesting advantages to users and developers of audio device drivers:

1.3.1 Portability

One important drawback to kernel-mode driver systems is the lack of portability.

Userland code has a wide range of portable system APIs at its disposal. When one writes code in userland, chances are that it won't have many dependencies on OS-specific facilities, and it will be easy to port this code to other operating systems. This has enabled projects like CUPS or the X Window System to support a specific class of devices (printers or displays, respectively) on multiple operating systems using the same driver code.

On the other hand, when writing drivers in kernel space, the interfaces the driver will depend on are very specific to the kernel it is being programmed for. In practice it is very likely that this driver will only be useful to one operating system.

1.3.2 Release engineering

Operating system kernels often go a step further in their lack of portability. It is very common that a driver written for a specific version of certain operating system kernel won't build or run correctly on a different version of the same kernel!

While this (not having to maintain API compatibility) may have many advantages to kernel developers, it also means that the release cycle of device drivers has to be synchronized with the release cycle of the kernel as a whole. This can be a significant disadvantage for end users.

If sound drivers weren't tied to internal kernel interfaces, they could be released as a separate software package, and users could run the latest version of sound drivers without compromising the safety of other system components.

1.3.3 Privilege separation

Another important inconvenient to kernel-mode drivers is their lack of privilege separation. A device driver only needs privileged access to a small set of hardware resources. For example a sound driver may need to access the PCI bus, specific areas of memory for DMA operations and synchronization with interrupt events. However, a software bug in this driver has the potential to compromise much more than those resources. A buffer overflow or an invalid pointer could corrupt kernel data or code, or it could leak kernel memory to user space; a small programming mistake could lead the kernel to a panic or deadlock; etc.

It's a significant benefit when the code associated with a hardware driver doesn't need to be given more responsibility than to manage correctly the resources that are strictly required for the task. This can be easily achieved by running the driver in user space.

1.4 Target audience

The project intends to develop a production-ready system. Therefore our target audience is very broad: it includes any computer user who runs audio-capable applications. System administrators and power users should be able to easily deploying our system to manage their audio hardware.

In addition, software developers should be able to integrate new drivers into the system with minimal effort (as seen in previous documents, one of the goals of the project is supporting unmodified drivers) in order to support their hardware.

2 General scope of the project

The goal of this project is to engineer a framework that will perform playback on commodity audio hardware using unmodified drivers, where the device driver logic resides entirely in user space, bypassing kernel interaction in all cases where this is possible.

2.1 Framework

The project doesn't just aim to produce a system that can play audio on user space on a specific device. Obviously, for the purpose of debugging and demonstrating the results, it will have to support at least one device, but the goal is for this system to be extensible; that is, a generic framework for audio in user space where:

1. New drivers can be easily integrated with no changes in the framework itself. This implies driver code will be isolated from the system core via abstraction layers, which will provide:
   (a) Facilities required by drivers to access hardware resources (PCI bus, Input/Output (I/O) space, interrupts...)
   (b) Interaction with upper-level audio APIs
2. Standard audio APIs are provided, so that most user applications can be easily configured to use our framework with minimal or no modification. The following audio APIs are prominent within the libre software ecosystem and will be considered. At least one of them will be provided by the system:

(a) Open Sound System (OSS). Since OSS is designed to provide a low-level interface based on kernel syscalls (open, write, ioctl...), providing this API in user space for unmodified applications could present some challenges as some mechanism would have to be implemented to intercept such calls.

(b) Advanced Linux Sound Architecture (ALSA). Since the ALSA API is provided through a user space library (libasound2), it could be simple to supply this API on user space by providing an Application Binary Interface (ABI)-compatible libasound. However, the ALSA API is much more extensive than OSS and also less portable, which might present some difficulties.

(c) PulseAudio, unlike the other two, is a sound server that takes audio input from user applications and redirects it to other audio APIs for output (generally OSS or ALSA). PulseAudio is deployed as the default audio server in most Linux-based GNU distributions and therefore is a widely supported API among audio applications. In principle PulseAudio would be automatically available if OSS or ALSA are implemented. However, as PulseAudio is able to support multiple audio API back ends, it should also be possible to support PulseAudio directly by implementing the appropriate extensions.

2.2 Unmodified drivers

The project aims to be useful in general, for the vast majority of commodity audio hardware. To fulfill this goal, it must be easy to integrate new drivers in it.

Fortunately there is a wide range of libre software device drivers for such audio hardware, available to us in free operating system kernels such as Linux and the kernels of *BSD operating systems.

After the project is complete, it will be possible to take a working device driver from at least one of those libre operating system kernels and integrate it into the project with minimal effort. In particular, it should be possible to do this without performing any changes in the driver itself. This can be achieved by maintaining a compatibility layer with the internal APIs of the targeted kernels.

2.3 Kernel interaction

The system will take the necessary provisions so that the driver code itself, as well as all the associated driver logic, runs entirely in user space.

Interaction with the kernel will be limited to the minimum that is strictly necessary in order to gain access shared hardware resources.

In some cases (e.g. PCI or interrupts), the resources are also used by other drivers and therefore the kernel will have to maintain its role of arbiter in order to grant access to the user space process implementing our system.

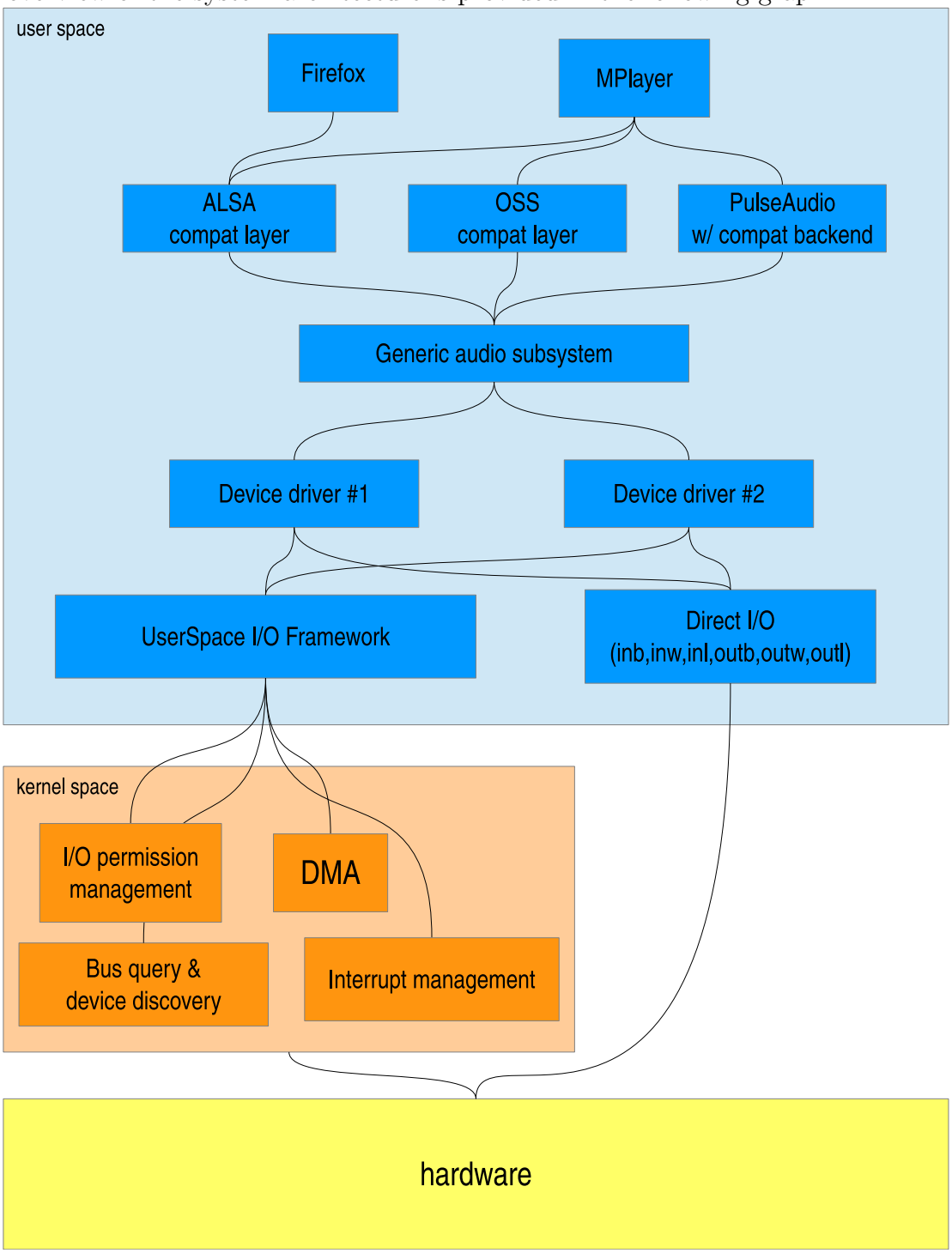
In other cases (e.g. I/O space) special privileges will be requested to the kernel so that our user space process is capable of directly using the resource from that point onwards.

Overall, the goal of the project is that the requirements for kernel interaction are well defined, and limited in scope (section 1.3.2 provides a good perspective on the reasoning behind this goal).

Upon completion, the project will run only on Linux, which is by far the most popular libre operating system kernel, and also provides a number of key facilities we expect to be essential to satisfy the previously-mentioned requirements.

2.4 System architecture

An overview of the system architecture is provided in the following graph:



3 Development plan

3.1 Overview

1. The project will attempt to focus on integration of existing technologies whenever possible. Rather than developing new components from scratch, it will attempt to

leverage existing libre software projects and invest effort into:

   (a) Finding existing technologies under permissive licensing terms that could be integrated as components of the project.
   (b) Evaluating existing technologies under permissive licensing terms that could be integrated as components of the project. For each technology: its code quality, documentation, the developer community behind it, etc.
   (c) Devising how said technology can be integrated into the project, and implementing the necessary glue code, bindings, missing features, etc.
   (d) Reading, understanding and ultimately debugging foreign code in order to fix any outstanding problems within that component which affect the project as a whole.

   The advantage of this is two-fold: On one hand, the project will be based on a foundation of widely tested code, which has gone under review, scrutiny and improvement for long periods of time. On the other, this will allow to direct development effort to achieve ambitious goals that no one else has fulfilled before, and as a result provide something that is new and uniquely useful to average computer administrators.

2. Once the desired technologies have been identified, the project will focus its effort in implementing a proof of concept playback using, rather than any standard audio APIs, a modified audio player application, specifically tailored for the purpose of testing the lower end of the system's audio stack.

3. When basic playback in user space is finally achieved, the missing pieces can be implemented, making it possible for unmodified applications to use the system.

The reason for this division of steps is that each landmark can be accurately tested on its own, which makes it easier for debugging and also to identify clearly when it has been completed.

3.2 Development resources

The resources available for development of this project are enumerated as follows:

1. Research information available on the Internet about implementing drivers on user space (academical papers and other sort of material).

2. Existing projects that already implement other areas of device driving in user space. These may be used as reference, to gain understanding or insight, and/or as integral part of the project if considered suitable.

3. Development tools:

   (a) A desktop PC at the student's workplace will be used for most of the development.
   (b) VirtualBox will be used to run the system in an isolated environment, as well as to provide emulated audio hardware for testing purposes.

(c) Debian GNU/Linux operating system will be used to host VirtualBox instances, as well as a guest OS.

(d) The GNU C Compiler (GCC) and associated development environment (binutils, make, etc) will be used to compile the project components directly from guest OS.

Notably missing from this list is any specific audio hardware. The system device driving will use one of the VirtualBox emulated audio devices, and therefore the result of the project will be usable (for testing, development and demonstration purposes) on any PC / OS combination that is equipped with audio hardware and also capable of running VirtualBox.

3.3 Potential technologies

There exist a few frameworks available which have been written to facilitate the development of user space device drivers:

1. DDE, and its companion project DDEkit, comprise a very complete system as they provide abstraction for all the necessary kernel facilities in a single layer. Its functionality includes interrupt management (using User Space I/O (UIO) as back end), PCI enumeration (using libpci as back end), direct I/O space access (using ioperm/iopl) and even DMA by means of a custom Linux module. Additionally, the DDE driver glue code is separated from DDEkit, allowing for multiple kernels to be supported (currently, Linux is supported but variants exist for other such as the kernel of FreeBSD [15])

2. RUMP provides a useful solution for running drivers in user space, as drivers written for NetBSD are easy to integrate with. Being a component of NetBSD itself, this provides strong assurance of continued support. It is also conceptually simpler than DDE/DDEkit. However, it has the disadvantage that it can only be used by drivers written for NetBSD (or other drivers after porting to internal NetBSD kernel APIs).

Both frameworks seem suitable options for implementing device driving in user space. Using either of them will allow the project to focus effort on integration of the various components, and on resolving problems derived from this integration, rather than on implementing a new framework from scratch.

In addition, both frameworks provide a solution for interfacing with existing device drivers that have been written for a specific kernel: DDE implements glue code for Linux drivers, and RUMP implements the glue code for drivers from the NetBSD kernel.

A generic audio subsystem will have to be developed, providing extensions to higher-level sound APIs like OSS.

In case ALSA support is provided, an ALSA emulation library has been identified, libsalsa. Libsalsa implements ALSA on top of OSS, using the latter as back end. For this project, it could potentially be modified to send its output to our audio subsystem instead of OSS.

3.4 Potential risks & mitigation strategies

1. The project intends to rely on the facilities made available by a large body of external code. Although this allows it to leverage on great technology already available in the libre software ecosystem, relying on external code opens the door for unforeseen problems to appear, specially when trying to use it for purposes that haven't tried before. There are two mitigation strategies for this problem:

   (a) Take advantage that all the code we're using is open source as an aid for debugging, and fix the problems by patching such code when necessary.

   (b) If a particular technology is found not to be suitable for some reason, and an alternative is available (as is the case with DDE/RUMP), fall back to the alternative.

2. The project intends to run entirely in user space, and only rely on standard kernel facilities for initialization. This approach is known to work, as similar projects have tried it before. However, the specifics of audio device driving may impose additional latency constraints: We could find that an event originated by hardware requires a quick response from the device driver, and due to scheduling problems we cannot guarantee this from user space. In case this happens, we could mitigate the problem by developing a helper kernel module to provide this quick response, while still allowing for the bulk of the driver logic to reside in user space.

3.5 Schedule

The project is broken down in four major tasks:

1. Back end development
2. Back end testing and debugging
3. Design and implementation of the sound API front end
4. Result validation

with each of these tasks broken down into smaller subtasks. The tables in Annex I give an exhaustive list of all the steps, and their associated Gantt diagram a schedule of their planned execution.

4 Validation

The initial milestone document of this project explained (in section 2.5) the difficulties in finding a proper validation method. The dilemma was that a simple quantitative approach cannot satisfy our desire to validate the result, because the motivation factors for this project are clearly qualitative ones.

When trying to solve this problem, we found that the Broadcasting Service (BS) recommendations published by the Radiocommunication Sector of the International Telecommunication Union (ITU-R) are among the most rigorous and recognized standards for validation of audio quality.

Among them, recommendation BS.1116-2 [16] appears to be the most suitable for this kind of experiment as we expect that impairments in audio quality in our system will be very small (if perceivable at all) in comparison with those in other kinds of experiment such as analog radio transmission or lossy audio compression.

However, upon careful examination of BS.1116-2 [16], we've come to the conclusion that this standard is too ambitious for a project of this size, as the resources required for implementing it cannot be possibly fit in our budget. Because of this, we've opted to use BS.1116-2 [16] for inspiration, and drafted a validation method based on similar ideas.

The validation method consists of a test procedure that needs to be carried out with collaboration of volunteer subjects over a single session (per subject), using a series of audio samples. Finally, results can be interpreted using statistical analysis.

A more elaborate description of each topic is covered in the subsections below. Please note, that references to specific section numbers refer to sections in BS.1116-2 [16], not to sections in the Followup report itself (in order to facilitate its reading this is not stated explicitly with each reference).

4.1 Samples

BS.1116-2 [16] establishes that the number of samples per session should be between 10 and 15, as a larger number might result in subject getting tired and losing auditive capacities (c.f. section 4.2).

Following this advice, we decided to make the sample selection as large as possible (15 samples), in order to compensate for the shortcomings in the number of subjects (refer to section 4.2 below for details).

Samples should have the following properties:

1. They should be neither too attractive nor too wearisome to listen. This is to prevent the subject from being distracted (c.f. section 6).

2. They should be presented without accompanying pictures or visual stimulus of any kind. This is, again, to prevent the subject from being distracted (c.f. section 2).

3. They should last between 10 and 25 seconds. With longer samples, it is believed that the subject will remember less details since auditive memory has a very short time span (c.f. section 4.2).

4. They should be different from each other in order to increase the chances that they expose different problems. The sample set should include samples that feature different combinations of pitch and timbre (c.f. section 6).

4.2 Subjects

Based on field experience, BS.1116-2 [16] states that 20 is a sufficient number of subjects in order to draw appropriate conclusions from the experiment (c.f. section 3.3).

Furthermore, BS.1116-2 [16] also recommends that the listening panel is composed exclusively from subjects who have expertise in detection of small auditive impairments (c.f. section 3.1).

We have no doubt that the recommendations in BS.1116-2 [16] are solidly grounded and widely recognised as best practice when performing validation of this kind. However,

our position for this project will have to differ. We find the recommendations are excessively demanding for a project of this size and budget. Forming a listening panel of 20 subjects would require that financial compensation is offered to them.

In addition, if we needed each subject to be an audition expert, this would make selection even harder and require a substantial expense to recruit them.

Therefore, in order to simplify validation of the project, we will rely only on volunteer subjects and will not establish a mandatory minimum number of subjects. We will attempt to encourage participation of as many subjects as possible, within our available means.

We will not require specific expertise or perform any kind of selection when forming the listening panel. As long as a candidate has a healthy auditive system, he will be accepted.

4.3 Test procedure

Before the test procedure begins, it is expected of the subjects that they have familiarised themselves with the samples (c.f. attachment 3 to annex 1). This will improve their ability to identify impairments when the test is conducted.

A triple-stimulus with hidden reference (ABC/HR) test procedure will be followed, as recommended by BS.1116-2 [16], but with a few divergences. The divergences will be described below; however, description of the specific details of ABC/HR is outside the scope of this document. The reader is encouraged to consult section 4 of BS.1116-2 [16] before proceeding.

Specifics of our implementation are as follows:

1. BS.1116-2 [16] recommends that a *double-blind* method is used for the test procedure. However, this would place the additional requirement of recruiting and training a control group, which is again not affordable within our budget. We have therefore opted to implement *single-blind* test method instead.

2. BS.1116-2 [16] provides a grading scale with five pre-defined anchor points. It recommends that the scale be treated as continuous, allowing the subject to pick any intermediate value, but without providing intermediate anchor points, as studies show that this may introduce bias [20]. Although BS.1116-2 isn't strict about this (the option of providing them is also contemplated), we find this to be a sensible recommendation and will implement it.

4.4 Result interpretation

Result interpretation will be based on statistical analysis, following the recommendations in BS.1116-2 [16]. The reader is encouraged to consult section 9 of that document for the details.

5 Budget

5.1 Foundations

Foundations for drafting of the project budget:

1. This project is a not-for-profit venture. Its purpose is to develop a framework for audio playback in user space and release it to the public in order to encourage better driver development in UNIX-like operating system kernels.

2. The project manager, Robert Millan Hernandez, will work under the direction of Dr. J.R. Herrero Zaragoza (associate professor in the Computer Architecture Department at UPC), who has kindly offered to take this role.

3. The project will be developed by a single engineer (Robert himself), acting at the same time as Project Manager, Designer and Programmer. Robert will work on this project every day since the beginning date (including weekends) until completion of the project, with a work schedule of 8 hours per day. Robert will also act as sponsor for the project and provide funding for all the liabilities enumerated in this budget, at his own personal cost and expense.

4. Project development will require access to several resources, split in two categories:

   (a) Initial investment. The project will require resources that are not consumed during development and have a considerable cost, therefore must be lent rather than purchased:

      i. A small office to be used as workplace environment. This office includes air conditioning infrastructure to make work environment comfortable, one desk with chairs and a drawer to store and classify project documentation. Fees for energy supply, bottled water supply and Internet service provider (ISP) are included with office rental payment.

   (b) Ongoing expenses:

      i. Depreciation of computer equipment (a high-end workstation is to be purchased in order to build large code bases in a short time).

      ii. As the project is a not-for-profit venture, there are no costs associated with legal permits or municipal taxes. The project will, however, have to assume the cost of VAT (Value-Added Tax) for materials (which in Catalonia stands for 21% of the base price).

   Possible events that may delay or impede project execution within expected results and/or schedule, and their associated contingency plans:

Event	Contingency plan
DDE (Device Drive Environment) is found unsuitable	RUMP is used instead
Binding DDE with USB is found unviable or too costly	PCI hardware is used instead
Implementing OSS as front end sound API is found unviable or too costly	PulseAudio is used instead

   In order to minimize possible impact caused by these undesired events, a task to thoroughly evaluate each possible path before implementation has been scheduled. If evaluation fails in first option, when the alternative (contingency plan) is the only remaining option, it is executed directly without evaluation.

5.2 Budget

The following project budget is elaborated using results from Gantt diagram and lists resources separated by task allocation category. Please refer to section 3.5 for details on which amount of the resource is spent on each individual task.

Resource	Amount	Price	Info	Days	Cost	Notes
Robert Millan's time	100%	200 €/day	This only includes time spent in ideal situation (no contingency plans activated)	61	12200 €	This expense is tax-free as Robert is the project sponsor and doesn't require payment
Office	100%	5 €/day	Includes Internet service, energy supply and bottled water supply	61	305 €	Price includes VAT
Depreciation of PC workstation	100%	1 €/day	High-end workstation designed for strong CPU workloads	61	61 €	1500 € amortized over a period of 4 years. Price includes VAT
SUBTOTAL		206 €/day		61	12566 €	
Optional expenses that may be caused by contingency plans:						
Robert Millan's time	100%	200 €/day	This only includes time spent in ideal situation (no contingency plans activated)	2	400 €	This expense is tax-free as Robert is the project sponsor and doesn't require payment
Office	100%	5 €/day	Includes Internet service, energy supply and bottled water supply	2	10 €	Price includes VAT
Depreciation of PC workstation	100%	1 €/day	High-end workstation designed for strong CPU workloads	2	2 €	1500 € amortized over a period of 4 years. Price includes VAT
SUBTOTAL		206 €/day		2	412 €	
TOTAL		206 €/day		63	12978 €	

5.3 Financial viability of the project

The total cost of the project in worst-case scenario (all contingency plans have to be activated) amounts to 12978 €. This cost is assumed in full by the project sponsor.

It should be noted that a significant portion of this cost (12600 €) isn't being spent in currency (€), but rather contributed directly in time and effort by the project sponsor.

The remaining 378 €are already funded in cash by the project sponsor, so there is no risk of insolvency.

6 Project execution

6.1 Back end development

6.1.1 Potential problems found that question the viability of using DDE

The prospect of using DDE as the abstraction layer to embed existing, production-quality device drivers into our project was very promising. Our alternative (RUMP) provides similar functionality; however, DDE was chosen as our first option as it is more kernel-agnostic: whereas RUMP is designed to provide access to drivers from the kernel of NetBSD, DDE is a generic framework, whose main focus is on Linux but has been successfully used to wrap device drivers from other kernels.

However, upon closer inspection we found that DDE was not actively maintained software and in its current state was not even usable with modern versions of Linux:

1. Latest update of DDE/Linux26 (the layer that encapsulates Linux drivers) was performed on 2008, and it was based on Linux 2.6.6.

   Later a new version based on Linux 2.6.29 was developed [23]; however, this version has never been integrated in the main branch of DDE hosted at the facilities of Technische Universität Dresden: Operating Systems (TUD:OS), nor has it been updated to newer versions of Linux, nor has received updates of any kind.

   On the other hand, DDE/FreeBSD (the layer that encapsulates FreeBSD drivers) is even older (dates back to 2006).

   This, however, is not a severe show stopper, since dependency between DDE and Linux 2.6.29 source code is restricted to DDE's role as glue layer; therefore, this condition would prevent us from using drivers from recent versions of Linux, but not from running the software on a recent platform.

2. Regarding DDEkit, the latest Linux version was released in 2011 [23]. The specific Linux version this release intended to support is not specified. However, through a combination of trial and error and binary search, we found that it is most likely targeted at Linux 2.6.37 (or close to that), which was released in the same year (2011).

   In contrast with DDE, DDEkit doesn't just use specific kernel APIs from user space, it comes with kernel modules of its own, which of course are strongly tied to internal Linux interfaces.

   In this case, limitations caused by interface dependencies are much more demanding: If left unfixed, they would prevent our software from being used on modern

operating systems based on a recent version of Linux. This would imply a mandatory port of DDEkit internals to up-to-date Linux APIs in case we decided to go this route.

3. Problems were found when attempting to build DDEkit on a test PC running AMD64 architecture. The problems themselves were not that significant, but their presence suggests this software has only been tested on computers running 32-bit x86 architecture. For a low-level component like DDEkit, runtime portability problems -in case they were found- could be quite difficult to debug and overcome. We think this implies a risk for the project schedule that needs to be taken into account.

4. We setup a test system with 32-bit x86 binaries for the purpose of evaluating DDE and DDEkit. An operating system distribution based on an old version of Linux was used in order to accommodate for DDE version requirements. During this test, severe runtime errors were found that prevented successful execution of the example code in DDE package. In case we used DDE / DDEkit, these problems would have to be debugged and fixed. Again, we think this is indicative of the abandoned state of this code base and it is likely that similar problems will be found if we proceed further with this route.

5. Another potential problem is that of licensing. We find that DDE and DDEkit rely heavily on code licensed under GNU General Public License (GPL) version 2. This license is incompatible with widely adopted free software licenses such as GPL version 3, the Apache license, etc. This is not usually a problem when running such code as a stand alone program (or even a kernel such as Linux). However, as our project aims to provide a framework that can be used directly by most application programs, this would place additional restriction on which programs can benefit.

In light of the potential problems that were found with DDE and DDEkit, decision was taken to discard this option and rely on the RUMP framework instead. RUMP is an actively maintained project, as is made clear by the considerable activity found in its source repository and its mailing lists. In addition it is an integral part of NetBSD, which means that it will automatically be integrated with newer versions of the NetBSD kernel APIs and therefore enjoy the availability of driver updates and support for new hardware with relative ease.

6.1.2 Attempts to use USB Audio Class devices led to a dead end

Initially, we intended to focus on integration of drivers for USB Audio Class devices. This seemingly would involve less difficulties than PCI because the usual abstraction layer for accessing USB from user space, LibUSB, relies on a complete, self-contained kernel API and provides all the necessary facilities in one single library. In contrast, with PCI in order to access hardware facilities like DMA or IRQs one has to rely on many distinct APIs. This involves additional effort, as we expect that not all those facilities will be readily implemented in our framework of choice.

Before looking at RUMP in detail, we held the belief that we would be able to integrate drivers for USB devices into RUMP, and use those on top of a Linux-based system with little effort. Extensions for RUMP to integrate USB drivers had already been implemented [17], so we didn't expect significant trouble in this area.

However, we found that RUMP didn't use the generic LibUSB library for interfacing with USB (if it had, it would have been easy to port it to Linux). Instead, it relied on a NetBSD-specific interface known as ugenhc. This interface is very different from the one provided by LibUSB.

When faced with this problem, we decided that it would be more worthwhile to direct efforts towards overcoming the PCI-related problems than with performing a complete rewrite of the USB back end in RUMP.

6.1.3 RUMP with PCI

In contrast with the USB situation, we've found that PCI support with RUMP on top of Linux is in a more advanced stage. As a recent development, we found a plugin that adds PCI support on Linux is now available for RUMP [9]. However, it had some limitations that had to be overcome to be used in our environment (more details are provided below).

A summary of the tasks we've had to perform in order to implement the PCI back end follows:

1. We've chosen an audio chipset model for testing purposes and to use as the reference target hardware for the whole project. For consistency, everything in the project, from development to validation will be done using this chipset.

   The chipset chosen is the *Intel 82801AA AC97* because of its wide availability both in real hardware and as a virtual device in our emulated environment.

2. We integrated the AUICH driver from NetBSD into RUMP. This just required the addition of a few configuration files and an initialization routine for the NetBSD driver configuration framework (c.f. [18], page 6):

   https://github.com/.../commit/dc106d26e3bf0595c105d9ed1423aa3e1c3cf91b

6.2 Back end testing and debugging

The back end configuration we built included the NetBSD "audio(4)" driver. This is the abstraction layer in the kernel of NetBSD that sits on top of specific device drivers and provides generic audio facilities to userland.

The API exported by this layer is the native audio API of NetBSD (c.f. [6]). It originates from the Solaris operating system, and is generally known as "Sun audio" in other projects (such as the MPlayer code base) or just "audio(4)" in the context of NetBSD or RUMP.

This is very relevant to our project, because before we could move on to front end development using one of the standard APIs we targeted, we had to setup a simpler, more direct way to test the back end and confirm it was working correctly.

Since the application we wanted to use for testing purposes (MPlayer) already included "Sun audio" support, this seemed like the more obvious approach.

Below is the list of tasks we carried on for back end testing and debugging, starting with the MPlayer modifications we mentioned:

1. We modified MPlayer to add support for a new back end (selectable using "-ao rump" flag). Rather than writing the new back end code from scratch, the Sun audio back end was reused, with some C pre-processor magic to redirect its requests to RUMP using the same API. This comprises three distinct sets of changes:

(a) *Fix portability problems in the "Sun audio" extension of MPlayer.* As Sun audio is not a native API provided by Linux, this code has never been used on Linux-based systems and required portability adjustments:

```
--- libao2/ao_sun.c       2014-07-29 11:33:20 +0000
+++ libao2/ao_sun.c       2014-07-30 16:45:34 +0000
@@ -25,6 +25,7 @@
 #include <unistd.h>
 #include <fcntl.h>
 #include <errno.h>
+#include <time.h>                                   /*
    nanosleep */
 #include <sys/ioctl.h>
 #include <sys/time.h>
 #include <sys/types.h>
@@ -455,7 +465,7 @@
                 else
                     info.play.balance = (vol->right -
                        vol->left + volume) *
                        AUDIO_RIGHT_BALANCE / (2*volume);
             }
-#if !defined (__OpenBSD__) && !defined (__NetBSD__)
+#if !defined (__OpenBSD__) && !defined (__NetBSD__) &&
    !defined (__linux__)
             info.output_muted = (volume == 0);
 #endif
             ioctl( fd,AUDIO_SETINFO,&info );
```

(b) *Avoid collisions in NetBSD ioctl namespace.* The "Sun audio" API definitions are provided on NetBSD by sys/audioio.h header. In turn, this header relies on internal ioctl definitions, which are provided on NetBSD by sys/ioccom.h header.

These definitions are also required by applications that use RUMP, so the header had to be extracted from NetBSD source tree [**?**] and manually installed system-wide.

However, neither of these headers could be used unmodified, because of namespace collisions in internal ioctl macros with their Linux counterparts. We had to ensure applications linked with these headers used the NetBSD version of these macros in order for their ABI to remain compatible with RUMP.

In order to avoid namespace collision, a few macros had to be renamed as well as all their users:

```
diff -ur src/sys/sys/audioio.h new/sys/sys/audioio.h
--- src/sys/sys/audioio.h       2011-09-06
    03:16:43.000000000 +0200
+++ new/sys/sys/audioio.h       2014-07-29
    13:10:17.590114112 +0200
@@ -169,26 +169,26 @@
 /*
```

```
   * Audio device operations
   */
-#define AUDIO_GETINFO  _IOR('A', 21, struct audio_info)
-#define AUDIO_SETINFO  _IOWR('A', 22, struct audio_info
    )
-#define AUDIO_DRAIN    _IO('A', 23)
-#define AUDIO_FLUSH    _IO('A', 24)
-#define AUDIO_WSEEK    _IOR('A', 25, u_long)
-#define AUDIO_RERROR   _IOR('A', 26, int)
-#define AUDIO_GETDEV   _IOR('A', 27, struct
    audio_device)
-#define AUDIO_GETENC   _IOWR('A', 28, struct
    audio_encoding)
-#define AUDIO_GETFD    _IOR('A', 29, int)
-#define AUDIO_SETFD    _IOWR('A', 30, int)
-#define AUDIO_PERROR   _IOR('A', 31, int)
-#define AUDIO_GETIOFFS _IOR('A', 32, struct
    audio_offset)
-#define AUDIO_GETOOFFS _IOR('A', 33, struct
    audio_offset)
-#define AUDIO_GETPROPS _IOR('A', 34, int)
+#define AUDIO_GETINFO  _NB_IOR('A', 21, struct
    audio_info)
+#define AUDIO_SETINFO  _NB_IOWR('A', 22, struct
    audio_info)
+#define AUDIO_DRAIN    _NB_IO('A', 23)
+#define AUDIO_FLUSH    _NB_IO('A', 24)
+#define AUDIO_WSEEK    _NB_IOR('A', 25, u_long)
+#define AUDIO_RERROR   _NB_IOR('A', 26, int)
+#define AUDIO_GETDEV   _NB_IOR('A', 27, struct
    audio_device)
+#define AUDIO_GETENC   _NB_IOWR('A', 28, struct
    audio_encoding)
+#define AUDIO_GETFD    _NB_IOR('A', 29, int)
+#define AUDIO_SETFD    _NB_IOWR('A', 30, int)
+#define AUDIO_PERROR   _NB_IOR('A', 31, int)
+#define AUDIO_GETIOFFS _NB_IOR('A', 32, struct
    audio_offset)
+#define AUDIO_GETOOFFS _NB_IOR('A', 33, struct
    audio_offset)
+#define AUDIO_GETPROPS _NB_IOR('A', 34, int)
 #define  AUDIO_PROP_FULLDUPLEX 0x01
 #define  AUDIO_PROP_MMAP       0x02
 #define  AUDIO_PROP_INDEPENDENT        0x04
 #define  AUDIO_PROP_PLAYBACK   0x10
 #define  AUDIO_PROP_CAPTURE    0x20
-#define AUDIO_GETBUFINFO       _IOR('A', 35, struct
    audio_info)
```

```
+#define AUDIO_GETBUFINFO        _NB_IOR('A', 35, struct
    audio_info)

 /*
  * Mixer device
@@ -261,9 +261,9 @@
 /*
  * Mixer operations
  */
-#define AUDIO_MIXER_READ                _IOWR('M', 0,
    mixer_ctrl_t)
-#define AUDIO_MIXER_WRITE               _IOWR('M', 1,
    mixer_ctrl_t)
-#define AUDIO_MIXER_DEVINFO             _IOWR('M', 2,
    mixer_devinfo_t)
+#define AUDIO_MIXER_READ                _NB_IOWR('M', 0,
     mixer_ctrl_t)
+#define AUDIO_MIXER_WRITE               _NB_IOWR('M', 1,
     mixer_ctrl_t)
+#define AUDIO_MIXER_DEVINFO             _NB_IOWR('M', 2,
     mixer_devinfo_t)

 /*
  * Well known device names
diff -ur src/sys/sys/ioccom.h new/sys/sys/ioccom.h
--- src/sys/sys/ioccom.h        2011-10-19
    12:53:12.000000000 +0200
+++ new/sys/sys/ioccom.h        2014-07-29
    13:10:22.638114329 +0200
@@ -44,32 +44,32 @@
  *      | I/O | Parameter Length          | Command Group
     | Command         |
  *
     +-------------------------------------------------------------------------

  */
-#define         IOCPARM_MASK    0x1fff          /*
    parameter length, at most 13 bits */
-#define         IOCPARM_SHIFT   16
-#define         IOCGROUP_SHIFT  8
-#define         IOCPARM_LEN(x)  (((x) >> IOCPARM_SHIFT)
    & IOCPARM_MASK)
-#define         IOCBASECMD(x)   ((x) & ~(IOCPARM_MASK <<
     IOCPARM_SHIFT))
-#define         IOCGROUP(x)     (((x) >> IOCGROUP_SHIFT)
     & 0xff)
+#define         NB_IOCPARM_MASK 0x1fff          /*
    parameter length, at most 13 bits */
```

```
+#define         NB_IOCPARM_SHIFT        16
+#define         NB_IOCGROUP_SHIFT       8
+#define         NB_IOCPARM_LEN(x)       (((x) >>
    NB_IOCPARM_SHIFT) & NB_IOCPARM_MASK)
+#define         NB_IOCBASECMD(x)        ((x) & ~(
    NB_IOCPARM_MASK << NB_IOCPARM_SHIFT))
+#define         NB_IOCGROUP(x)  (((x) >>
    NB_IOCGROUP_SHIFT) & 0xff)

-#define         IOCPARM_MAX     NBPG    /* max size of
    ioctl args, mult. of NBPG */
+#define         NB_IOCPARM_MAX  NBPG    /* max size of
    ioctl args, mult. of NBPG */
                                        /* no parameters */
-#define         IOC_VOID        (unsigned long)0
    x20000000
+#define         NB_IOC_VOID     (unsigned long)0
    x20000000
                                        /* copy parameters out
                                           */
-#define         IOC_OUT         (unsigned long)0
    x40000000
+#define         NB_IOC_OUT              (unsigned long)0
    x40000000
                                        /* copy parameters in */
-#define         IOC_IN          (unsigned long)0
    x80000000
+#define         NB_IOC_IN               (unsigned long)0
    x80000000
                                        /* copy parameters in
                                           and out */
-#define         IOC_INOUT       (IOC_IN|IOC_OUT)
+#define         NB_IOC_INOUT    (NB_IOC_IN|NB_IOC_OUT)
                                        /* mask for IN/OUT/VOID
                                           */
-#define         IOC_DIRMASK     (unsigned long)0
    xe0000000
+#define         NB_IOC_DIRMASK  (unsigned long)0
    xe0000000

-#define         _IOC(inout, group, num, len) \
-     ((inout) | (((len) & IOCPARM_MASK) << IOCPARM_SHIFT
    ) | \
-     ((group) << IOCGROUP_SHIFT) | (num))
-#define         _IO(g,n)        _IOC(IOC_VOID,  (g), (n)
    , 0)
-#define         _IOR(g,n,t)     _IOC(IOC_OUT,   (g), (n)
    , sizeof(t))
```

```
-#define         _IOW(g,n,t)     _IOC(IOC_IN,    (g), (n)
    , sizeof(t))
+#define         _NB_IOC(inout, group, num, len) \
+     ((inout) | (((len) & NB_IOCPARM_MASK) <<
    NB_IOCPARM_SHIFT) | \
+     ((group) << NB_IOCGROUP_SHIFT) | (num))
+#define         _NB_IO(g,n)     _NB_IOC(NB_IOC_VOID,
    (g), (n), 0)
+#define         _NB_IOR(g,n,t)  _NB_IOC(NB_IOC_OUT,
    (g), (n), sizeof(t))
+#define         _NB_IOW(g,n,t)  _NB_IOC(NB_IOC_IN,
    (g), (n), sizeof(t))
 /* this should be _IORW, but stdio got there first */
-#define         _IOWR(g,n,t)    _IOC(IOC_INOUT, (g), (n)
    , sizeof(t))
+#define         _NB_IOWR(g,n,t) _NB_IOC(NB_IOC_INOUT,
    (g), (n), sizeof(t))

 #endif /* !_SYS_IOCCOM_H_ */
```

(c) *Modify the "Sun audio" extension of MPlayer to send its output to RUMP.* Finally, the "Sun audio" extension was modified, linking with appropriate RUMP libraries, calling RUMP initialization routines and using C pre-processor magic to redirect the bulk of external audio calls to RUMP:

```
=== modified file 'configure'
--- configure   2014-07-29 11:34:07 +0000
+++ configure   2014-07-30 16:42:04 +0000
@@ -5538,6 +5539,8 @@
  if test "$_sunaudio" = yes ; then
    def_sunaudio='#define CONFIG_SUN_AUDIO 1'
    aomodules="sun $aomodules"
+   extra_ldflags="$extra_ldflags -lrumpvfs -lrumpdev -
    lrumpdev_audio"
+   extra_ldflags="$extra_ldflags -lrumpdev_pci -
    lrumpdev_pci_auich"
  else
    def_sunaudio='#undef CONFIG_SUN_AUDIO'
    noaomodules="sun \$noaomodules"

=== modified file 'libao2/ao_sun.c'
--- libao2/ao_sun.c     2014-07-29 11:33:20 +0000
+++ libao2/ao_sun.c     2014-07-30 16:45:34 +0000
@@ -49,6 +50,15 @@
 #include "mp_msg.h"
 #include "help_mp.h"

+/* Redirect calls to rump */
+#include <stdint.h>
```

```
+#include <rump/rump.h>
+#include <rump/rump_syscalls.h>
+#define open            rump_sys_open
+#define write           rump_sys_write
+#define ioctl           rump_sys_ioctl
+#define close           rump_sys_close
+
 static const ao_info_t info =
 {
     "Sun audio output",
@@ -381,7 +391,7 @@
 {
     if (audio_dev == NULL) {
         if ((audio_dev = getenv("AUDIODEV")) == NULL)
-            audio_dev = "/dev/audio";
+            audio_dev = "/dev/audio0";
     }

     if (sun_mixer_device == NULL) {
@@ -477,6 +487,8 @@
     int ok;
     int convert_u8_s8;

+                rump_init();
+
     setup_device_paths();

     if (enable_sample_timing == RTSC_UNKNOWN
```

2. *Implement PCI I/O space support.* We found that the RUMP PCI plugin [9] was missing support for mapping I/O space of PCI devices. Attempting to run the driver would result in:

   ```
   auich0: can't map codec i/o space
   ```

   We fixed this problem thus:

   (a) We implemented the missing routines, using iopl(2) [4] system facility to request I/O access privileges to Linux for our user space process.

   (b) We implemented the missing stubs for bus_space(9) [7] routines using GCC inline assembly [14].

   Aforementioned changes can be consulted in:

   https://github.com/.../commit/8f2fd610334072e92406e78d14eefb0f8466d28b

3. *Implement missing bits of DMA support.* We found existing DMA support in the RUMP PCI plugin, which allowed mapping physical memory for use with DMA, by reserving anonymous memory using mmap() [8]. However this part of the code was incomplete, as it had no provision to free mapped memory after it is no longer needed. Again this made our execution fail, this time with:

```
panic: bus_dmamem_free not implemented
```

We implemented the missing routine (bus_dmamem_free) by storing information relative to the virtual memory address of allocated blocks in their meta-data structures, which would later allow us to liberate it using munmap() [8].

This change is split in two parts, one for RUMP itself, and one for the PCI plugin:

https://github.com/.../commit/17fc0ea0ee88f51ca31345a5ad222c66426b56f6

https://github.com/.../commit/92c8b2ed831dc76fcfdc4fe5642426480b3f3b34

4. *Fix division by zero in auich_calibrate().* We found that our process often received a Floating Point Exception (SIGFPE) during initialization phase.

   This turned to be very hard to debug, as the problem only manifested itself occasionally. And once it did, it put the hardware in an inconsistent state (see note below about unexpected process termination leading to deadlocks) where it was no longer reproducible.

   Ultimately (after fixing the deadlock problem), a backtrace (captured with the GNU Debugger (GDB)) revealed that this exception was raised at the following line of auich_calibrate():

   ```
   actual_48k_rate = (bytes * UINT64_C(250000)) / wait_us;
   ```

   in which "wait_us" represents the time (measured in μs) it has taken to exit the calibration loop at the beginning of this routine:

   ```
   /* wait */
   nciv = ociv;
   do {
           microtime(&t2);
           if (t2.tv_sec - t1.tv_sec > 1)
                   break;
           nciv = bus_space_read_1(sc->iot, sc->aud_ioh
              ,
                                                   ICH_PCMI + ICH_CIV);
   } while (nciv == ociv);
   microtime(&t2);

   /* omitted ~rmh */

   /* turn time delta into us */
   wait_us = ((t2.tv_sec - t1.tv_sec) * 1000000) + t2.
      tv_usec - t1.tv_usec;
   ```

   As we can see, this loop polls an I/O hardware address, and it is thus at least a theoretical possibility that the hardware response time is lower than 1 μs, which would cause the loop to exit with a "wait_us" value of zero, thereby raising a division by zero SIGFPE.

   Obviously, this being a production-ready device driver, this problem isn't expected to happen under normal conditions. We believe our particular setup (emulated

virtual hardware and driver running in userspace) triggers faster response time, which uncovers this previously undetected bug.

We fixed the problem by adding a special case to handle the "wait_us == 0" situation:

https://github.com/.../commit/6a38ce4778d18a569ab593099929afe24e24d811

This is one of the changes we've made to the RUMP code base which directly revert back as improvements to the NetBSD device drivers it is based on, and could potentially be useful in situations other than user space device driving.

5. *Prevent a situation in which unexpected process termination leads to subsequent deadlocks.* We think that this problem is an interesting finding, because it exemplifies the kind of new situations that arise when moving kernel-based device drivers to user space.

   As we've found, a very important difference between driving hardware devices from kernel vs user space is that user space code flow can be aborted at any time if the process that contains it is killed.

   This could be triggered by multiple reasons. For example, the process is killed because it raised an exception (like SIGFPE example above), a user or system administrator outright kills the process, etc.

   In this particular case, if our process were aborted during execution of certain parts of auich_read_codec() or auich_write_codec() routines, I/O Controller Hub (ICH) Codec Access Semaphore (ICH_CAS) register may be left in an inconsistent state, which would lead to deadlock:

   (a) a command is expected by ICH_CAS before granting access to the driver

   (b) the driver has requested codec access and is polling ICH_CAS before issuing a command

   This would be triggered if abortion happened immediately after codec access is granted by ICH_CAS, i.e. after this poll loop has finished:

```
/* wait for an access semaphore */
for (i = ICH_SEMATIMO / ICH_CODECIO_INTERVAL; i-- &&
    bus_space_read_1(sc->iot, sc->aud_ioh,
        ICH_CAS + sc->sc_modem_offset) & 1;
    DELAY(ICH_CODECIO_INTERVAL));
```

   but before the codec access transaction is finished and ICH_CAS is cleared (see auich_clear_cas() below):

```
*val = bus_space_read_2(sc->iot, sc->mix_ioh
    ,
     reg + (sc->sc_codecnum *
        ICH_CODEC_OFFSET));
DPRINTF(ICH_DEBUG_CODECIO,
    ("auich_read_codec(%x, %x)\n", reg, *val
        ));
```

```
                status = bus_space_read_4(sc->iot, sc->
                    aud_ioh,
                     ICH_GSTS + sc->sc_modem_offset);
                if (status & ICH_RCS) {
                        bus_space_write_4(sc->iot, sc->
                            aud_ioh,
                                          ICH_GSTS + sc->
                                              sc_modem_offset
                                              ,
                                          status & ~(ICH_SRI
                                              |ICH_PRI|
                                              ICH_GSCI));
                        *val = 0xffff;
                        DPRINTF(ICH_DEBUG_CODECIO,
                            ("%s: read_codec error\n",
                                device_xname(sc->sc_dev)));
                        if (reg == AC97_REG_GPIO_STATUS)
                                auich_clear_cas(sc);
                        return -1;
                }
                if (reg == AC97_REG_GPIO_STATUS)
                        auich_clear_cas(sc);
```

Our solution to this problem has been to make auich_read_codec() and auich_write_codec() more permissive when run for first time, so that they can recover from this inconsistent state:

https://github.com/.../commit/c8ee3813f55586e02c660e4e4a90df54cc77ab5a

6. At this point playback is partially working:

```
# RUMP_VERBOSE=2 ./mplayer /home/rmh/split-track02.flac -ao
    sun
MPlayer SVN-r34540-4.7 (C) 2000-2012 MPlayer Team
mplayer: could not connect to socket
mplayer: No such file or directory
Failed to open LIRC support. You will not be able to use
    your remote control.

Playing /home/rmh/split-track02.flac.
libavformat version 53.19.0 (internal)
Audio only file format detected.
Load subtitles in /home/rmh/
==========================================================================


Opening audio decoder: [ffmpeg] FFmpeg/libavcodec audio
    decoders
libavcodec version 53.32.2 (internal)
AUDIO: 44100 Hz, 2 ch, s16le, 976.0 kbit/69.16% (ratio:
    121996->176400)
```

```
Selected audio codec: [fffflac] afm: ffmpeg (FFmpeg FLAC
   audio)
==========================================================

Copyright (c) 1996, 1997, 1998, 1999, 2000, 2001, 2002,
   2003, 2004, 2005,
2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014
The NetBSD Foundation, Inc. All rights reserved.
Copyright (c) 1982, 1986, 1989, 1991, 1993
The Regents of the University of California. All rights
   reserved.

NetBSD 6.99.43 (RUMP-ROAST) #0: Tue Nov 11 13:52:59 CEST
   2014
rmh@tfg:/home/rmh/rump/buildrump.sh/obj/lib/librump
total memory = unlimited (host limit)
timecounter: Timecounters tick every 10.000 msec
timecounter: Timecounter "rumpclk" frequency 100 Hz quality
   0
cpu0 at thinair0: rump virtual cpu
cpu1 at thinair0: rump virtual cpu
root file system type: rumpfs
mainbus0 (root)
pci0 at mainbus0 bus 0
pci0: i/o space, memory space enabled, rd/line, rd/mult, wr/
   inv ok
auich0 at pci0 dev 0 function 0: i82801AA (ICH) AC-97 Audio
auich0: interrupting at pausebreak
auich0: ac97: SigmaTel STAC9700 codec; no 3D stereo
auich0: ac97: ext id 0x809<AC97_23,VRM,VRA>
ac97_write: reg=MASTER_VOLUME, written=0x8020, read=0x803f
auich0: ac97 link rate calibration timed out after 0 us
audio0 at auich0: full duplex, playback, capture, mmap,
   independent
ao2: 44100 Hz 2 chans s16le [0x9]

AO: [sun] 44100Hz 2ch s16le (2 bytes per sample)
Video: no video
Starting playback...
A: 0.3 (00.2) of 121.0 (02:01.0) ??,?%
```

However, after a few seconds, the process gets stuck and sound is interrupted. This issue required careful investigation:

(a) First we run MPlayer using GDB, up to the point where playback gets stuck, then manually interrupted the process and extracted a backtrace:

```
AO: [sun] 44100Hz 2ch s16le (2 bytes per sample)
Video: no video
```

```
Starting playback...
A:   0.3 (00.2) of 121.0 (02:01.0) ??,?%
Program received signal SIGINT, Interrupt.
pthread_cond_wait@@GLIBC_2.3.2 () at ../nptl/sysdeps/
   unix/sysv/linux/x86_64/pthread_cond_wait.S:162
162../nptl/sysdeps/unix/sysv/linux/x86_64/
   pthread_cond_wait.S: El fitxer o directori no
   existeix.
(gdb) bt
#0  pthread_cond_wait@@GLIBC_2.3.2 () at ../nptl/sysdeps
   /unix/sysv/linux/x86_64/pthread_cond_wait.S:162
#1  0x00007fffeadcb21a in rumpuser_cv_wait (cv=0x197c8b0
   , mtx=0x197c550)
    at /home/rmh/rump/buildrump.sh/src/lib/librumpuser/
       rumpuser_pth.c:547
#2  0x00007fffefacb21d in docvwait (cv=0x1a6b0e0, mtx=0
   x1a61c10, ts=0x0)
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/librump/rumpkern/locks.c:359
#3  0x00007fffefacb3b3 in cv_wait_sig (cv=0x1a6b0e0, mtx
   =0x1a61c10)
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/librump/rumpkern/locks.c:409
#4  0x00007ffff5917c10 in audio_waitio (sc=0x1a6b000,
   chan=0x1a6b0e0)
    at /home/rmh/rump/buildrump.sh/src/sys/rump/dev/lib/
       libaudio/../../../../dev/audio.c:1261
#5  0x00007ffff591a0d2 in audio_write (sc=0x1a6b000, uio
   =0x7fffffffd2a0, ioflag=16)
    at /home/rmh/rump/buildrump.sh/src/sys/rump/dev/lib/
       libaudio/../../../../dev/audio.c:2279
#6  0x00007ffff5917fd6 in audiowrite (dev=10880, uio=0
   x7fffffffd2a0, ioflag=16)
    at /home/rmh/rump/buildrump.sh/src/sys/rump/dev/lib/
       libaudio/../../../../dev/audio.c:1374
#7  0x00007fffefa98607 in cdev_write (dev=10880, uio=0
   x7fffffffd2a0, flag=16)
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/../kern/subr_devsw.c:886
#8  0x00007ffff5d6ec76 in spec_write (v=0x7fffffffd1c0)
    at /home/rmh/rump/buildrump.sh/src/lib/librumpvfs
       /../../sys/rump/../miscfs/specfs/spec_vnops.c:790
#9  0x00007ffff5d962c4 in rump_vop_spec (v=0
   x7fffffffd1c0)
    at /home/rmh/rump/buildrump.sh/src/lib/librumpvfs
       /../../sys/rump/librump/rumpvfs/rumpfs.c:1680
#10 0x00007fffefa79411 in VOP_WRITE (vp=0x1aab818, uio=0
   x7fffffffd2a0, ioflag=16, cred=0x19aff00)
```

```
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/../kern/vnode_if.c:430
#11 0x00007ffff5d783ca in vn_write (fp=0x1acbe80, offset
   =0x1acbe80, uio=0x7fffffffd2a0, cred=0x19aff00, flags
   =1)
    at /home/rmh/rump/buildrump.sh/src/lib/librumpvfs
       /../../sys/rump/../kern/vfs_vnops.c:571
#12 0x00007fffefa5ba52 in dofilewrite (fd=3, fp=0
   x1acbe80, buf=0x1b02010, nbyte=16384, offset=0
   x1acbe80, flags=1,
    retval=0x7fffffffd480) at /home/rmh/rump/buildrump.
       sh/src/lib/librump/../../sys/rump/../kern/
       sys_generic.c:355
#13 0x00007fffefa5b9a9 in sys_write (l=0x19b6800, uap=0
   x7fffffffd460, retval=0x7fffffffd480)
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/../kern/sys_generic.c:323
#14 0x00007fffefad27f9 in sy_call (sy=0x7fffefd0acc0, l
   =0x19b6800, uap=0x7fffffffd460, rval=0x7fffffffd480)
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/../sys/syscallvar.h:61
#15 0x00007fffefad28cb in sy_invoke (sy=0x7fffefd0acc0,
   l=0x19b6800, uap=0x7fffffffd460, rval=0x7fffffffd480,
    code=4)
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/../sys/syscallvar.h:85
#16 0x00007fffefad3d3a in rump_syscall (num=4, data=0
   x7fffffffd460, dlen=24, retval=0x7fffffffd480)
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/librump/rumpkern/rump.c:846
#17 0x00007fffefac4061 in rump___sysimpl_write (fd=3,
   buf=0x1b02010, nbyte=16384)
    at /home/rmh/rump/buildrump.sh/src/lib/librump
       /../../sys/rump/librump/rumpkern/rump_syscalls.c
       :110
#18 0x00000000004f39f9 in play ()
#19 0x00000000004af7bf in main ()
```

the backtrace revealed that our process was stuck in audio_waitio(), which is a hardware synchronization routine.

Ultimately the backtrace leads to pthread_cond_wait() [2], which is a POSIX Threads library (pthread) waiting call that yields the CPU until a condition is met. Since this thread of execution is halted, rather than actively using the CPU to poll hardware registers, and it can only be awakened by another thread, it is clear that it is waiting for an interrupt-triggered event to resume its operation. This suggests that something is failing with hardware interrupts for this device or their handler routines.

(b) We checked the interrupt counters for UIO driver in Linux and verified in-

terrupts are being received as the counters increment between two successive checks:

```
$ grep uio_pci_generic /proc/interrupts
 21:        8138   IO-APIC-fasteoi   uio_pci_generic

$ grep uio_pci_generic /proc/interrupts
 21:        8515   IO-APIC-fasteoi   uio_pci_generic
```

(c) Having discarded the possibility that interrupts aren't being received by the driver, we went on to check the kernel side of interrupt handling. We reviewed the Linux source code (specifically, version 3.2.60) to find out what conditions could lead to interrupts not being processed.

Close inspection of the source code reveals the only possible way this would happen is if the interrupt service routine (uio_interrupt()) didn't execute uio_event_notify(), which in turn is a condition determined by irqhandler() return value:

```
struct uio_device *idev = (struct uio_device *)
    dev_id;
irqreturn_t ret = idev->info->handler(irq, idev
    ->info);

if (ret == IRQ_HANDLED)
        uio_event_notify(idev->info);
```

As we can see, IRQ_HANDLED is the only possible return value that would result in the interrupt event being notified. And when we analyze the irqhandler() routine of the UIO driver, we find that there is only one way to exit this routine without returning IRQ_HANDLED:

```
/* Check interrupt status register to see whether
   our device
 * triggered the interrupt. */
if (!(status & PCI_STATUS_INTERRUPT))
    goto done;
```

As a quick way to verify our thesis, we added a printk() at the part of the routine that is never supposed to run:

```
--- linux-3.2.60.orig/drivers/uio/uio_pci_generic.c
    2014-06-09 14:29:18.000000000 +0200
+++ linux-3.2.60/drivers/uio/uio_pci_generic.c
    2014-08-17 16:18:00.202810091 +0200
@@ -70,6 +70,8 @@
        if (!(status & PCI_STATUS_INTERRUPT))
                goto done;

+       printk(KERN_EMERG "irq for us!\n");
+
        /* We triggered the interrupt, disable it. */
        newcmd = origcmd | PCI_COMMAND_INTX_DISABLE;
        if (newcmd != origcmd)
```

And indeed, we found that our message is never printed.

(d) The check we found is related to shared IRQs. When receiving an interrupt, UIO driver needs to verify it is directed at our device before processing it. But, for some reason it was always thinking the interrupt is meant for someone else.

Since supporting the driver using shared IRQs was not part of the goals of the project, and we can (as we've found) successfully demonstrate user space audio drivers without this feature, we've opted for modifying the UIO driver to disabled shared IRQs. This can be accomplished simply by:

i. Removing he IRQF_SHARED flag in the uio_pci_generic_dev structure.

ii. Disabling the aforementioned check.

Thus, changes were performed as follows:

```
Index: linux-3.2.60/drivers/uio/uio_pci_generic.c
===================================================================
--- linux-3.2.60.orig/drivers/uio/uio_pci_generic.c
    2014-07-24 18:59:42.916804862 +0200
+++ linux-3.2.60/drivers/uio/uio_pci_generic.c
    2014-07-25 09:15:17.660640451 +0200
@@ -65,10 +65,12 @@
         origcmd = cmd_status_dword;
         status = cmd_status_dword >> 16;

+#if 0
         /* Check interrupt status register to see
            whether our device
          * triggered the interrupt. */
         if (!(status & PCI_STATUS_INTERRUPT))
                 goto done;
+#endif

         /* We triggered the interrupt, disable it. */
         newcmd = origcmd | PCI_COMMAND_INTX_DISABLE;
@@ -154,7 +156,7 @@
         gdev->info.name = "uio_pci_generic";
         gdev->info.version = DRIVER_VERSION;
         gdev->info.irq = pdev->irq;
-        gdev->info.irq_flags = IRQF_SHARED;
+        gdev->info.irq_flags = 0;
         gdev->info.handler = irqhandler;
         gdev->pdev = pdev;
```

After performing these changes, playback finally works reliably without time constraints. **This means the first major milestone of the project (audible playback) is finally achieved**.

6.3 Design and implementation of the front end

As we mentioned earlier on section 2, as a result of using RUMP our system already provides a standard audio API that can be used by applications, which is generally known as "Sun audio" or "audio(4)" (c.f. [6]).

However, our goal as defined in section 2.1 was to provide one of the more widespread audio APIs (OSS, ALSA and PulseAudio). The reason we wanted to provide one of these interfaces was to maximize the usefulness of our platform. Given that "Sun audio" API is seldom known or used outside of NetBSD, we feel there is no reason to deviate from our initial goal, as exporting another interface is still necessary.

As established in the project schedule (refer to Gantt diagram in section 3.5), we would begin evaluating the implementation of an OSS front end as our first option.

In this regard, our use of "Sun audio" turned out to be quite productive. After some research, we found that a compatibility library libossaudio existed for the sole purpose of allowing OSS applications to run using "Sun audio".

This library, however, was targeted at a very different use case than we intended, and required a number of modifications in order to serve our purpose:

1. It used native NetBSD audio(4) API as back end, as it was only intended to be useful on NetBSD. Instead, we wanted it to use the same API on a different back end (RUMP).

   Fixing this required some C pre-processor magic to redirect ioctl() calls issued by libossaudio to RUMP:

   https://github.com/.../commit/c499f3a84f8040e123641b4c0a8f0565b3f3bfb0

2. It relied on system-wide availability of NetBSD audio(4) API definitions (which were obviously not present when run on our target Linux-based system).

   We fixed this by importing a copy of "sys/audioio.h" and "sys/ioccom.h" from the NetBSD source tree [**?**], and afterwards renaming a few macros (and their users) to avoid namespace collisions with system-wide ioctl definitions:

   https://github.com/.../commit/37bfb1e7c82fe927d276d6df37ff5badcdbfff0c

   https://github.com/.../commit/dd5596f20126e54b723d312ba5ad182747a38b9b

   this is very similar to what we did when implementing the "Sun audio" back end for MPlayer (refer to section for details).

3. It included its own version of the "sys/soundcard.h" header, as it expected applications to use it. However, this was contrary to our goals: we intended to support unmodified programs as already provided on Linux-based systems, which had been linked with a different implementation of "sys/soundcard.h".

   In order to maintain ABI compatibility, we had to make some portability improvements in libossaudio as to allow it to link using the Linux version of "sys/soundcard.h" instead:

   https://github.com/.../commit/d6c7605ad28b9db5ae19bbba2b25204adda7d99f

4. It expected applications to use libossaudio's own version of ioctl(), which is exported as the "_oss_ioctl" symbol by the library. Conveniently, it provided C pre-processor define in "sys/soundcard.h" to redirect "ioctl()" function calls to "_oss_ioctl()". However, this imposed two requirements that we were unable to meet:

(a) Applications had to be rebuilt. However, this wasn't compatible with our goal of supporting unmodified applications.

(b) The libossaudio version of "sys/soundcard.h" had to be deployed system-wide. However, this posed ABI-related problems as explained above.

In order to avoid this situation, we developed an upper layer to intercept calls to "ioctl()" and related calls (such as "open()" or "close()"). It would then redirect them to our own handlers when they were intended for OSS audio playback, or to standard system facilities otherwise.

Note that this problem is in fact close to our initial expectation, as we already stated in section 2.1 that implementing seamless support for OSS would require a system to intercept system calls.

The implementation of this layer, however, is more complex than we expected: It has to assign a fake file descriptor to each new OSS handle requested by the application, and keep track of them in a global, Mutual Exclusion (mutex)-protected data structure (a hash table, in order to do this efficiently). Complete source code to accomplish this can be found at:

https://github.com/.../commit/0b3e2de52cf30678eb3a948c5acb8957bdf5fd43

The final result for the front end (dubbed "rumposs") is a RUMP-based application, which can be consulted at:

https://github.com/robertmillan/rumposs

6.4 Result validation

6.4.1 Programmatic buffer control

Prior to the field validation based on experimental audition that we described previously, we considered it would also be useful to monitor the status of playback buffers used by audio hardware to see if there could be any risk of buffer underrun problems because of the additional overhead when running the driver in user space.

In order to achieve this, we modified Kernel-based Virtual Machine (KVM) and added a routine that calculates the current amount of used space at a given point, and regularly prints it every time a new chunk of data is processed:

```
--- qemu-kvm-1.1.2+dfsg.old/hw/ac97.c    2012-09-09
    15:21:39.000000000 +0200
+++ qemu-kvm-1.1.2+dfsg/hw/ac97.c        2014-12-31
    18:33:04.054061226 +0100
@@ -967,6 +967,19 @@

     while (temp) {
         int copied;
+
+        /* Print buffer utilization */
+        printf("%u\n",
+               /* Total implied buffer utilization because of
    the distance
```

```
+                   between current buffer and last buffer. */
+                ((r->lvi - r->civ + 32) % 32) * (r->bd.ctl_len &
    0xffff)
+                /* Plus number of bytes pending to process in our
     current
+                   buffer */
+                + r->picb
+                /* Minus what we've just written in this
    execution of write_audio()
+                   before PICB is updated */
+                - (written >> 1));
+
          to_copy = audio_MIN (temp, sizeof (tmpbuf));
          pci_dma_read (&s->dev, addr, tmpbuf, to_copy);
          copied = AUD_write (s->voice_po, tmpbuf, to_copy);
```

Then we executed our modified KVM (playing a 2 min sample) and plotted the output (once using the native kernel driver in Linux, and once using the RUMP-based user space approach).

Figure 1: Native kernel driver in Linux

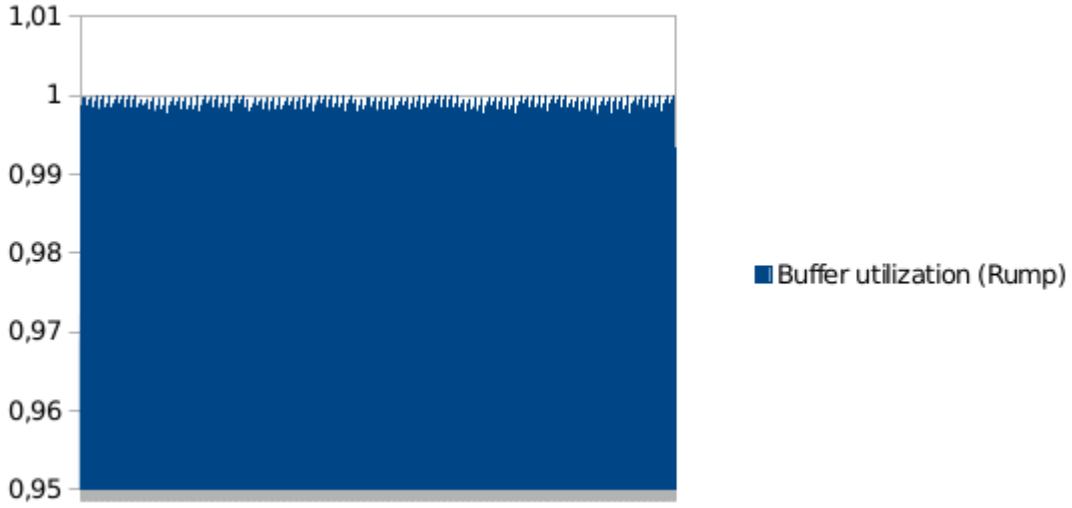
As we can see in the figures 1 and 2, in both cases buffer utilization is maintained very close to 100% during the whole test. One can observe that the user space version takes a more conservative approach in that it refills the buffer more often and keeps it to a higher minimum, but other than that we don't see any significant differences when validating our work from this perspective.

6.4.2 Field validation based on subject experiments

ABC/HR is a repeated measures design. In fact, it implements repeated measures in two different dimensions: intra-subject and intra-sample. This eliminates possible error sources that would be derived from cross-comparison of results between different subjects or different samples.

In addition to this, when subjects compare samples with the original reference, they base their assessments in a recent memory footprint of the original reference. This elim-

Figure 2: RUMP-based user space approach



inates possible distortions derived from time lapse (recall that human auditive memory has a very short time span, as mentioned in section 4.2 of BS.1116-2 [16]).

Finally, the order in which the hidden reference is presented to the subject has been counter-balanced to compensate possible order derived distortions.

We consider that such exhaustive control of error sources guarantees a very high degree of statistical significance, and therefore are confident that we will obtain reliable results despite the fact that the number of subjects (5) is below BS.1116-2 [16] recommendation (20 or more).

It should be noted as well that each subject evaluated 15 samples, allowing for a grand total of 150 test results after each of our subjects compared each sample twice (once with the userland version and once with the hidden reference).

A complete listing of the test results, as well as the paired differences for each individual test, and their corresponding average and standard deviation is provided in Annex III of this document.

For statistical analysis of the results we used parametric statistics as recommended by section 9 of BS.1116-2 [16]; however, our understanding is that the two compared sets are too similar for Analysis of Variance (ANOVA) to yield useful results. Therefore we performed our analysis using Student's t-test for paired samples [24].

We obtained t-value by applying Student's formula:

$$t = \frac{\overline{X}_D}{s_D \div \sqrt{n}} \tag{1}$$

where $\overline{X}_D$ is the average of the paired differences, s_D is the standard deviation of the paired differences and n is the number of subject-sample combinations.

Thus:

$$t = \frac{\overline{X}_D}{s_D \div \sqrt{n}} = \frac{0,0066666667 - 0}{0,4754324726 \div \sqrt{75}} = 0,1214368606 \tag{2}$$

For a bilateral test, given a 0,95 degree of confidence and 74 degrees of freedom, our threshold for statistical significance is 1,99, which is a much larger value than the t-value we obtained (0,1214368606).

Therefore, we can assure with a 0,95 level of confidence that both of the studied sets are the same. That is, there is no observable difference between sound output produced using our user space solution and the reference (native Linux sound system).

6.5 Schedule deviations and final time line

It hasn't been possible to carry the project according to plan. A number of deviations have been necessary. Some of them were already foreseen and had contingency plans, which have been activated; in another case, the problem was unforeseen due to a mistake in the initial plan, and caused unexpected delay.

1. During the initial stages of back end development, we concluded that some of the choices we made on interfaces or technology to be relied on, were not suitable for successful execution of the project. Fortunately we had contingency plans to manage these situations.

   An elaborate explanation on the problems found, as well as the reasoning for our design choices can be found in section 6.1.

2. Debugging of MPlayer + Backend combination took much longer than expected. We planned to perform this step in less than a week; however, it took almost an entire month to completion. Unfortunately we found a lot of unforeseen problems, many of which were related to low level kernel and hardware subtleties and required expensive debugging work.

   A complete listing of the problems we found, with description for each problem as well as the solution we implemented, can be found in section 2.

3. Initial planning didn't account for some of the tasks. Due to a mistake when drafting the initial project schedule, some of the tasks that are implied for development of this project were not explicitly included:

   (a) Definition of the validation method. Section 2.5 of the initial project document (section 4 in this document) established that a validation method was yet to be defined, and implied that a specific task with this purpose would be added to the project.

   However, the schedule as defined in section 3.5 of the initial project document (or section 3.5 in this one) did not take this into account. As a result it had to be refactored to include it.

   (b) Actual execution of the result validation method wasn't included in the initial schedule either.

   (c) Drafting of the Follow up report merited a task of its own. However this hadn't been accounted for in the initial schedule.

   (d) Drafting of this document (the final report) also merited a task of its own. Again, this hadn't been accounted for in the initial schedule.

Consequently, a number of adjustments have had to be performed in the project schedule in order to accommodate the missing tasks, correcting all of the above mistakes. This pushes the target completion date back several days, from Nov 10th 2014 to Jan 2nd 2015.

For the revised project schedule (including Gantt diagram), refer to Annex II.

7 Sustainability and social impact

7.1 Applicable laws and regulations

To the best of our knowledge, there are no areas of law and regulation that could be of concern for our project plan. However, for completeness we considered some of which could possibly affect us in order to provide a rebuttal:

1. Directive 2009/490/EC of the European Parliament [22] establishes certain safety requirements for personal music players. It places on device manufacturers the responsibility of implementing security controls that restrict device owners in order to prevent them from harming themselves.

   Specifically, Article 3 states the following requirements:

   (a) *"Exposure to sound levels shall be time limited to avoid hearing damage. At 80 dB(A) exposure time shall be limited to 40 hours/week, whereas at 89 dB(A) exposure time shall be limited to 5 hours/week. For other exposure levels a linear intra- and extrapolation applies. Account shall be taken of the dynamic range of sound and the reasonably foreseeable use of the products."*

   (b) *"Personal music players shall provide adequate warnings on the risks involved in using the device and to the ways of avoiding them and information to users in cases where exposure poses a risk of hearing damage."*

   However, as the software we're developing is only a middleware framework, and not a complete application, our understanding is that it doesn't fall upon us to implement any sort of warning or restriction, but rather to end manufacturers or application vendors.

2. From a copyright standpoint, we think our project is on safe ground as well: In our current plan, it will only be integrated with code from RUMP. RUMP is a derivative of NetBSD, which is available under liberal license terms for use and redistribution in either modified or unmodified form.

   More details on the copyright and license terms of RUMP and NetBSD are available at:

   http://www.netbsd.org/about/redistribution.html

3. Finally, a word on patents. Article 52 of the European Patent Convention [1] (section 2.c) establishes that computer programs cannot be regarded as an invention in order to qualify for an European patent. Since the project we're developing is comprised of software only, we believe that patent law is of no concern for our project.

7.2 Environmental sustainability

As the project produces only software it has no direct impact on the environment. It should be noted, however, that its development will require energy supply for a high-end workstation during at least 488 h, amounting to a rough energy expenditure of 878 MJ, which equals to emitting a small amount of CO_2 to the atmosphere.

7.3 Social impact

Section 1.3 already elaborates on the technical factors that we think justify the development of user space device drivers. We expect that the social impact of developing user space device drivers in general will be the logical consequence of these factors:

1. As user space device drivers tend to be more portable than kernel-mode drivers, we expect end users of different operating systems will benefit from wider driver availability, as they are no longer restricted to using drivers written for their own operating system.

2. As user space device drivers allow for decoupled release cycles and more flexible release engineering, we expect end users will benefit from installing new drivers as they desire without compromising the stability of their systems.

3. As user space device drivers run inside the same isolated sandbox environment as all user space processes, we expect end users will benefit from higher reliability and stability in their computer experience. (section 1.2.2 provides more detail on existing research regarding device driver stability).

We think all the aforementioned benefits are applicable to our project to some degree.

On the other hand, we found that development of user space device drivers is not entirely possible without modifying the original drive code, as in some cases it would have made assumptions that are only valid in kernel environment. Fixing this problem involves getting rid of the assumptions, making such code more robust and thereby improving its quality as a collateral effect.

Two notable examples of such improvements are explained in section 2 (division by zero in auich_calibrate() and hardware inconsistency caused by unexpected process termination).

8 Conclusions

We have found that audio device driving in user space is a viable option: Our work complies with existing laws and regulations (c.f. section 7.1); once the infrastructure provisions are in place, integration of new drivers requires little effort (c.f. section 6.1.3); the resulting framework works reliably, ensuring a sustainable data flow (c.f. section 6.4.1) and with audible output of very high quality, which has been experimentally proven to be indistinguishable from the traditional kernel-based approach (c.f. section 6.4.2).

In contrast, we've learnt that the initial goal of supporting unmodified drivers cannot be fulfilled: device driving in user space doesn't just require provisions to access I/O facilities; it presents new problems because of the fundamental differences between kernel and user space modes.

We have learnt about one such difference from one of the problems we fixed in AUICH (c.f. section 2): user space processes can die, even while inside a critical section! The system needs to be able to cope with that event and handle it gracefully. In contrast, when the kernel panics it doesn't have to worry about cleaning things up, as a panic is immediately followed by a hardware reboot.

That means if a device driver contains any code region that temporarily puts the hardware in an inconsistent state (e.g. an incomplete transaction), it needs to take into

account the possibility that execution is interrupted precisely inside this region and be able to detect (and fix) this inconsistency afterwards when the process is restarted.

It is possible that other fundamental differences between kernel and user space modes generate new problems for drivers in user space. So far we haven't found any, but given what we've seen, our expectation is that more could be found in case user space drivers are developed in future directions.

When it comes to user space *audio* driving on UNIX-like operating systems, we think after having proved that the concept is viable, there are many exciting possibilities for expanding this work in other directions. To name a few:

1. Our work only developed support for PCI devices, but support for other buses could also be implemented. We think USB Audio Class devices would be an interesting complement. We've explored this possibility only on the surface (c.f. section 6.1.2).

2. Even within the scope of PCI bus devices, more drivers could be integrated. As we've seen, integration itself is a simple process (c.f. section 6.1.3) but it could lead to discovery of new bugs in the drivers when moved to their new user space environment (c.f. section 2).

3. So far we've developed support for only OSS audio API. We've performed some superficial analysis on other front ends (ALSA and PulseAudio) and we think it would be a nice addition to this project in order to extend the number of supported applications a bit further.

4. Our work has only focused on Linux as the host kernel, on one hand because it's the most widespread UNIX-like kernel and on the other because it provided readily available facilities for user space driver development (such as UIO). However, we think with some effort it could support other kernels. For example, the kernel of FreeBSD implements a PCI pass-through facility similar to UIO [10], which could potentially be adapted for this purpose.

Overall, we're quite satisfied with the result of this project. Now that it has been shown to be a viable alternative, end users can automatically obtain the benefits initially laid out in section 1.3.

We think there's a lot more to explore in this area and that in the end it can be very rewarding both from an engineering or research perspective as well as from the point of view of the end user or system administrator.

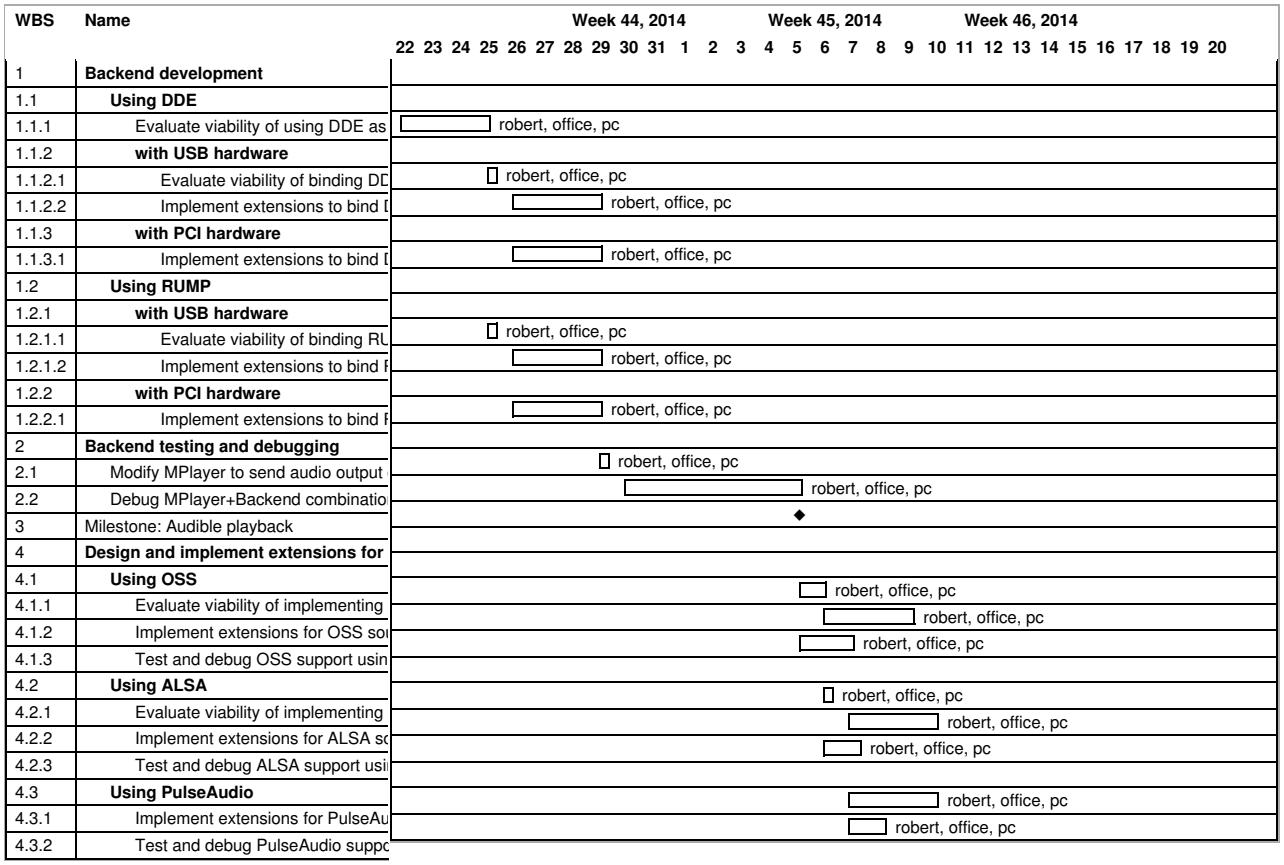
Annexes

I Original project schedule

User space approach to audio device driving on UNIX-like systems

Company: FIB
Manager: Robert Millan
Start: October 22, 2014
Finish: November 10, 2014
Report Date: October 11, 2014

Gantt Chart



Tasks

WBS	Name	Start	Finish	Work	Complete	Cost	Assigned to
1	**Backend development**	**Oct 22**	**Oct 29**	**54d**			
1.1	**Using DDE**	**Oct 22**	**Oct 29**	**32d**			
1.1.1	Evaluate viability of using DDE as framework for userspace device driving	Oct 22	Oct 25	10d	0%	320	robert, office, pc
1.1.2	**with USB hardware**	**Oct 25**	**Oct 29**	**12d**			
1.1.2.1	Evaluate viability of binding DDE with USB hardware for audio output	Oct 25	Oct 25	2d	0%	64	robert, office, pc
1.1.2.2	Implement extensions to bind DDE framework with USB backend	Oct 26	Oct 29	10d	0%	320	robert, office, pc
1.1.3	**with PCI hardware**	**Oct 26**	**Oct 29**	**10d**			
1.1.3.1	Implement extensions to bind DDE framework with PCI backend	Oct 26	Oct 29	10d	0%	320	robert, office, pc
1.2	**Using RUMP**	**Oct 25**	**Oct 29**	**22d**			
1.2.1	**with USB hardware**	**Oct 25**	**Oct 29**	**12d**			
1.2.1.1	Evaluate viability of binding RUMP with USB hardware for audio output	Oct 25	Oct 25	2d	0%	64	robert, office, pc
1.2.1.2	Implement extensions to bind RUMP framework with USB backend	Oct 26	Oct 29	10d	0%	320	robert, office, pc
1.2.2	**with PCI hardware**	**Oct 26**	**Oct 29**	**10d**			
1.2.2.1	Implement extensions to bind RUMP framework with PCI backend	Oct 26	Oct 29	10d	0%	320	robert, office, pc
2	**Backend testing and debugging**	**Oct 29**	**Nov 5**	**22d**			
2.1	Modify MPlayer to send audio output directly to Backend	Oct 29	Oct 29	2d	0%	64	robert, office, pc
2.2	Debug MPlayer+Backend combination and fix any outstanding problems	Oct 30	Nov 5	20d	0%	640	robert, office, pc
3	Milestone: Audible playback	Nov 5	Nov 5				
4	**Design and implement extensions for standard sound API frontend**	**Nov 5**	**Nov 10**	**49d**			
4.1	**Using OSS**	**Nov 5**	**Nov 9**	**17d**			
4.1.1	Evaluate viability of implementing OSS as frontend sound API	Nov 5	Nov 6	2d	0%	64	robert, office, pc
4.1.2	Implement extensions for OSS sound API	Nov 6	Nov 9	10d	0%	320	robert, office, pc
4.1.3	Test and debug OSS support using arbitrary applications with audio output	Nov 5	Nov 7	5d	0%	160	robert, office, pc
4.2	**Using ALSA**	**Nov 6**	**Nov 10**	**17d**			
4.2.1	Evaluate viability of implementing ALSA as frontend sound API	Nov 6	Nov 6	2d	0%	64	robert, office, pc
4.2.2	Implement extensions for ALSA sound API	Nov 7	Nov 10	10d	0%	320	robert, office, pc
4.2.3	Test and debug ALSA support using arbitrary applications with audio output	Nov 6	Nov 7	5d	0%	160	robert, office, pc

4.3	**Using PulseAudio**	**Nov 7**	**Nov 10**	**15d**			
4.3.1	Implement extensions for PulseAudio sound API	Nov 7	Nov 10	10d	0%	320	robert, office, p
4.3.2	Test and debug PulseAudio support using arbitrary applications with audio output	Nov 7	Nov 8	5d	0%	160	robert, office, p

Resources

Name	Short name	Type	Group	Email	Cost
Robert Millan	robert	Work		robert.millan@est.fib.upc.edu	10
Office	office	Material			1
PC workstation (amortization)	pc	Material			1

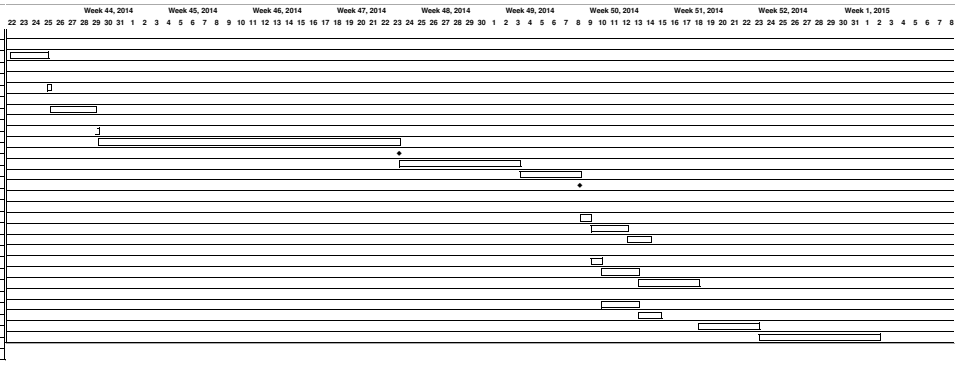
II Revised project schedule

User space approach to audio device driving on UNIX-like systems

Company: FIB
Manager: Robert Millan
Start: October 22, 2014
Finish: January 2, 2015
Report Date: December 8, 2014

Gantt Chart

WBS	Name	Work
1	**Backend development**	**7d 1h**
1.1	**Using DDE**	**3d 2h**
1.1.1	Evaluate viability of using DDE as framework for userspace device driving	3d 2h
1.2	**Using RUMP**	**3d 7h**
1.2.1	**with USB hardware**	**5h**
1.2.1.1	Evaluate viability of binding RUMP with USB hardware for audio output	5h
1.2.2	**with PCI hardware**	**3d 2h**
1.2.2.1	Implement extensions to bind RUMP framework with PCI backend	3d 2h
2	**Backend testing and debugging**	**25d 5h**
2.1	Modify MPlayer to send audio output directly to Backend	5h
2.2	Debug MPlayer+Backend combination and fix any outstanding problems	25d
3	Milestone: Audible playback	
4	Define validation method	10d
5	Write followup report	5d
6	Followup meeting	
7	**Design and implement extensions for standard sound API frontend**	**19d 2h**
7.1	**Using OSS**	**5d 4h**
7.1.1	Evaluate viability of implementing OSS as frontend sound API	5h
7.1.2	Implement extensions for OSS sound API	3d 2h
7.1.3	Test and debug OSS support using arbitrary applications with audio output	1d 5h
7.2	**Using ALSA**	**8d 7h**
7.2.1	Evaluate viability of implementing ALSA as frontend sound API	5h
7.2.2	Implement extensions for ALSA sound API	3d 2h
7.2.3	Test and debug ALSA support using arbitrary applications with audio output	5d
7.3	**Using PulseAudio**	**4d 7h**
7.3.1	Implement extensions for PulseAudio sound API	3d 2h
7.3.2	Test and debug PulseAudio support using arbitrary applications with audio output	1d 5h
8	Result validation	5d
9	Write final report	10d



Tasks

WBS	Name	Start	Finish	Work	Complete	Cost	Assigned to
1	**Backend development**	**Oct 22**	**Oct 29**	**7d 1h**			
1.1	**Using DDE**	**Oct 22**	**Oct 25**	**3d 2h**			
1.1.1	Evaluate viability of using DDE as framework for userspace device driving	Oct 22	Oct 25	3d 2h	0%		
1.2	**Using RUMP**	**Oct 25**	**Oct 29**	**3d 7h**			
1.2.1	**with USB hardware**	**Oct 25**	**Oct 25**	**5h**			
1.2.1.1	Evaluate viability of binding RUMP with USB hardware for audio output	Oct 25	Oct 25	5h	0%		
1.2.2	**with PCI hardware**	**Oct 25**	**Oct 29**	**3d 2h**			
1.2.2.1	Implement extensions to bind RUMP framework with PCI backend	Oct 25	Oct 29	3d 2h	0%		
2	**Backend testing and debugging**	**Oct 29**	**Nov 23**	**25d 5h**			
2.1	Modify MPlayer to send audio output directly to Backend	Oct 29	Oct 29	5h	46%		
2.2	Debug MPlayer+Backend combination and fix any outstanding problems	Oct 29	Nov 23	25d	0%		
3	Milestone: Audible playback	Nov 23	Nov 23				
4	Define validation method	Nov 23	Dec 3	10d	0%		
5	Write followup report	Dec 3	Dec 8	5d	0%		
6	Followup meeting	Dec 8	Dec 8				
7	**Design and implement extensions for standard sound API frontend**	**Dec 8**	**Dec 18**	**19d 2h**			
7.1	**Using OSS**	**Dec 8**	**Dec 14**	**5d 4h**			
7.1.1	Evaluate viability of implementing OSS as frontend sound API	Dec 8	Dec 9	5h	0%		
7.1.2	Implement extensions for OSS sound API	Dec 9	Dec 12	3d 2h	0%		
7.1.3	Test and debug OSS support using arbitrary applications with audio output	Dec 12	Dec 14	1d 5h	0%		
7.2	**Using ALSA**	**Dec 9**	**Dec 18**	**8d 7h**			
7.2.1	Evaluate viability of implementing ALSA as frontend sound API	Dec 9	Dec 10	5h	0%		
7.2.2	Implement extensions for ALSA sound API	Dec 10	Dec 13	3d 2h	0%		
7.2.3	Test and debug ALSA support using arbitrary applications with audio output	Dec 13	Dec 18	5d	0%		
7.3	**Using PulseAudio**	**Dec 10**	**Dec 15**	**4d 7h**			
7.3.1	Implement extensions for PulseAudio sound API	Dec 10	Dec 13	3d 2h	0%		
7.3.2	Test and debug PulseAudio support using arbitrary applications with audio output	Dec 13	Dec 15	1d 5h	0%		
8	Result validation	Dec 18	Dec 23	5d	0%		
9	Write final report	Dec 23	Jan 2	10d	0%		

Resources

Name	Short name	Type	Group	Email	Cost
Robert Millan	robert	Work		robert.millan@est.fib.upc.edu	10
Office	office	Material			1
PC workstation (amortization)	pc	Material			1

III ABC/HR experimental results

Núria

sample	A-B	A-C	HR	REF-REF	REF-RUMP	difference
1	4,90	5,00	C	5,00	4,90	0,10
2	5,00	5,00	B	5,00	5,00	0,00
3	4,50	4,70	C	4,70	4,50	0,20
4	5,00	5,00	B	5,00	5,00	0,00
5	5,00	4,50	C	4,50	5,00	-0,50
6	5,00	5,00	B	5,00	5,00	0,00
7	5,00	5,00	C	5,00	5,00	0,00
8	4,80	5,00	B	4,80	5,00	-0,20
9	4,00	5,00	C	5,00	4,00	1,00
10	5,00	5,00	B	5,00	5,00	0,00
11	4,70	4,60	C	4,60	4,70	-0,10
12	5,00	5,00	B	5,00	5,00	0,00
13	5,00	5,00	C	5,00	5,00	0,00
14	5,00	5,00	B	5,00	5,00	0,00
15	5,00	4,70	C	4,70	5,00	-0,30

Angela

sample	A-B	A-C	HR			
1	5,00	5,00	B	5,00	5,00	0,00
2	5,00	5,00	C	5,00	5,00	0,00
3	4,90	4,90	B	4,90	4,90	0,00
4	5,00	4,80	C	4,80	5,00	-0,20
5	5,00	5,00	B	5,00	5,00	0,00
6	5,00	5,00	C	5,00	5,00	0,00
7	5,00	5,00	B	5,00	5,00	0,00
8	5,00	5,00	C	5,00	5,00	0,00
9	5,00	5,00	B	5,00	5,00	0,00
10	5,00	5,00	C	5,00	5,00	0,00
11	5,00	5,00	B	5,00	5,00	0,00
12	5,00	5,00	C	5,00	5,00	0,00
13	5,00	5,00	B	5,00	5,00	0,00
14	5,00	5,00	C	5,00	5,00	0,00
15	5,00	5,00	B	5,00	5,00	0,00

Carles

sample	A-B	A-C	HR			
1	5,00	5,00	C	5,00	5,00	0,00
2	5,00	5,00	B	5,00	5,00	0,00
3	5,00	5,00	C	5,00	5,00	0,00
4	5,00	5,00	B	5,00	5,00	0,00
5	5,00	5,00	C	5,00	5,00	0,00
6	5,00	5,00	B	5,00	5,00	0,00
7	5,00	5,00	C	5,00	5,00	0,00
8	5,00	5,00	B	5,00	5,00	0,00
9	5,00	5,00	C	5,00	5,00	0,00
10	5,00	5,00	B	5,00	5,00	0,00
11	5,00	5,00	C	5,00	5,00	0,00
12	5,00	5,00	B	5,00	5,00	0,00
13	5,00	5,00	C	5,00	5,00	0,00
14	5,00	5,00	B	5,00	5,00	0,00
15	5,00	5,00	C	5,00	5,00	0,00

Nil

sample	A-B	A-C	HR			
1	4,00	4,00	B	4,00	4,00	0,00
2	4,00	4,00	C	4,00	4,00	0,00
3	4,80	4,80	B	4,80	4,80	0,00
4	4,00	4,00	C	4,00	4,00	0,00
5	4,00	3,50	B	4,00	3,50	0,50
6	4,00	3,50	C	3,50	4,00	-0,50
7	4,50	5,00	B	4,50	5,00	-0,50
8	4,00	4,00	C	4,00	4,00	0,00
9	4,50	4,00	B	4,50	4,00	0,50
10	5,00	4,00	C	4,00	5,00	-1,00
11	4,00	4,00	B	4,00	4,00	0,00
12	4,00	4,00	C	4,00	4,00	0,00
13	4,50	4,00	B	4,50	4,00	0,50
14	4,00	4,00	C	4,00	4,00	0,00
15	4,00	4,00	B	4,00	4,00	0,00

Cristina

sample	A-B	A-C	HR			
1	4,00	3,00	C	3,00	4,00	-1,00
2	5,00	4,00	B	5,00	4,00	1,00
3	2,00	4,00	C	4,00	2,00	2,00
4	4,00	5,00	B	4,00	5,00	-1,00
5	5,00	4,00	C	4,00	5,00	-1,00
6	3,00	2,00	B	3,00	2,00	1,00
7	5,00	5,00	C	5,00	5,00	0,00
8	4,00	4,00	B	4,00	4,00	0,00
9	4,00	3,00	C	3,00	4,00	-1,00
10	4,00	4,00	B	4,00	4,00	0,00
11	4,00	3,00	C	3,00	4,00	-1,00
12	4,00	4,00	B	4,00	4,00	0,00
13	4,00	5,00	C	5,00	4,00	1,00
14	4,00	4,00	B	4,00	4,00	0,00
15	4,00	5,00	C	5,00	4,00	1,00

AVG: 0,00666667
STDEV: 0,47543247

Student's t-value for paired samples: 0,12143686

Acronyms

ABC/HR triple-stimulus with hidden reference. 14, 37

ABI Application Binary Interface. 8, 21, 35, 36

ALSA Advanced Linux Sound Architecture. 8, 11, 35, 42, 52, *See glossary:* Advanced Linux Sound Architecture

ANOVA Analysis of Variance. 38

API Application Programming Interface. 5–8, 10, 11, 15, 18–21, 35, 42, 51–53

BS Broadcasting Service. 12

BSD Berkeley Software Distribution. 51, 52

DDE Device Driver Environment. 3, 11, 12, 15, 18, 19, *See glossary:* Device Driver Environment

DMA Direct Memory Access. 6, 7, 11, 19, 26

GCC the GNU C Compiler. 11

GDB the GNU Debugger. 27, 30

GPL GNU General Public License. 19

I/O Input/Output. 7, 8, 11, 26, 27, 41

ICH I/O Controller Hub. 28, 50, 51, *See glossary:* I/O Controller Hub

ICH_CAS ICH Codec Access Semaphore. 28, *See glossary:* ICH Codec Access Semaphore

IPC Inter-Process Communication. 5, 6

IRQ Interrupt Request. 6, 19, 34

ITU-R Radiocommunication Sector of the International Telecommunication Union. 12

KVM Kernel-based Virtual Machine. 36, 37, *See glossary:* Kernel-based Virtual Machine

mutex Mutual Exclusion. 36, *See glossary:* Mutual Exclusion

OSS Open Sound System. 8, 11, 15, 35, 36, 42, 51, 52, *See glossary:* Open Sound System

PCI Peripherial Component Interconnect. 3, 7, 8, 11, 15, 19, 20, 26, 27, 42, 51

pthread POSIX Threads library. 32

SIGFPE Floating Point Exception. 27, 28

TUD:OS Technische Universität Dresden: Operating Systems. 18

UIO User Space I/O. 11, 32–34, 42, *See glossary:* User Space I/O

USB Universal Serial Bus. 3, 15, 19, 20, 42, 52, 53

Glossary

ICH Codec Access Semaphore Semaphore register in ICH which is read by software to check whether a codec access is currently in progress. For details, refer to Intel datasheet [3] (page 328). 28, 50

Advanced Linux Sound Architecture ALSA was designed to support a set of advanced features which at that time were not available in OSS. Although only implemented by Linux-based systems, it's the most commonly used API on such systems, and enjoys widest support among audio-capable applications. 8, 50

AUICH NetBSD driver for Intel 82801AA AC97 audio hardware. 20, 41

critical section A piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution. 41, 52

DDEkit Abstraction layer which maps hardware access facilities provided by the different subsystems of the underlying kernel (currently, only Linux is supported) into a unified API that is exported to userland programs. 11, 18, 19, 51

Device Driver Environment Wrapper library which emulates the internal API of certain kernels (including Linux) and uses DDEkit as back end to implement the necessary facilities. It allows for drivers written for Linux to be used unmodified on a user space environment. 3, 50

double-blind Test method in which the subject, as well as any operator of the experiment whom the subject interacts with, are agnostic about the expected outcome of the subject's evaluation. 14

FreeBSD Free, UNIX-like operating system that descends from Berkeley Software Distribution (BSD). 11, 18, 42

I/O Controller Hub A highly-integrated, multi-functional I/O Controller Hub that provides the interface to the PCI Bus and integrates many of the functions needed in today's PC platforms [3]. 28, 50

kernel The definition of this word is a bit lax. Traditionally, the kernel is interpreted to be the core of the operating system. This includes, among other things, management of shared resources (hardware or otherwise) and most of the time device drivers too. However, sometimes certain resources can be managed in user space, and sometimes devices can be driven by user space too. To avoid ambiguities, in

this document we take a practical approach: if the processor is set in privileged mode when running a piece of code, then this code is part of the kernel; otherwise it is part of userland. In contrast with userland, UNIX-like kernels typically exhibit very tight integration of their internal components, with direct communication governed by constantly evolving APIs. 2, 3, 5–8, 11, 12, 15, 18–20, 28, 33, 37, 41, 42, 51–53

Kernel-based Virtual Machine Full virtualization solution for Linux-based hosts on x86 hardware with x86 guests. KVM is intended for systems where the processor has hardware support for virtualization. 36, 50

libossaudio A NetBSD compatibility library which provides emulation of the OSS audio interface. [5]. 35, 36

LibUSB A portable interface for accessing USB from userland (http://www.libusb.org/). 19, 20

Linux A very popular kernel implementation, used as the foundation for most UNIX-like operating systems. Traditionally used only in combination with GNU userland (so-called Linux-based GNU systems, or GNU/Linux systems), nowadays it's almost ubiquitous as a component of mobile operating systems such as Android. 5, 8, 11, 18–21, 26, 32, 33, 35, 37, 39, 42, 52

MPlayer Free media player for UNIX-like operating systems. 20, 21, 25, 30, 35, 39

Mutual Exclusion The requirement of ensuring that no two concurrent processes are in their critical section at the same time. 36, 50

NetBSD Free, UNIX-like operating system that descends from BSD. 11, 18–21, 28, 35, 40, 51–53

Open Sound System The traditional audio API for UNIX-like operating systems. Its API follows the UNIX "everything is a file" philosophy which makes it relatively simple to understand and implement. It has lost traction over the years since it was deprecated in favour of ALSA on Linux-based GNU systems, but is still supported as an output option by the vast majority of applications. 8, 50

PulseAudio Audio server which takes audio input from user applications and redirects it to other audio APIs for output (generally OSS or ALSA). PulseAudio is deployed as the default audio server in most Linux-based GNU distributions and therefore is a widely supported API among audio applications. 8, 15, 35, 42

RUMP Framework for running internal subsystems of the NetBSD kernel directly as a user space process. It maps the required facilities for hardware access to the internal NetBSD kernel APIs, thereby making it possible for NetBSD device drivers to run in user space. 3, 11, 12, 15, 18–21, 25–28, 35–38, 40

single-blind Test method in which the subject is agnostic about the expected outcome of his evaluation. 14

Solaris A proprietary derivative of AT&T UNIX operating system. 20

ugenhc An interface for accessing USB from userland specific to NetBSD (http://nxr.netbsd.org/source/xref/src/sys/rump/dev/lib/libugenhc/ugenhc.c). 20

UNIX-like operating system Broad family of operating systems whose design (and sometimes code heritance) derives from AT&T Unix System V. All references to operating systems in this document refer to this kind, unless stated otherwise. 5, 15, 52

user space System components which are not part of the kernel. This includes the most noticeable part of the vast majority of processes in the system (although some part of each process runs in the kernel, and many kernel implementations provide a set of kernel-only processes). An important characteristic of user space well-defined layer separations that use very stable APIs to communicate. Examples of these are the C library API, and many of the public APIs exported by the kernel. 5–8, 10–12, 15, 18, 19, 26, 28, 34, 36–39, 41, 42, 51–53

User Space I/O Public kernel API provided by Linux which implements certain facilities useful to perform device driving in user space. Despite its name, its main purpose is not enabling I/O access to user space (that is the domain of ioperm and iopl), but to give user space the ability to synchronize with hardware interrupts [13]. 11, 51

userland Same as user space. In this document they may be used interchangeably. 5, 6, 38, 51–53

VirtualBox Free ia32/amd64 virtualization solution allowing a wide range of operating systems. 10, 11

References

[1] The European Patent Convention. Article 52. URL http://www.epo.org/law-practice/legal-texts/html/epc/2013/e/ar52.html. 1973.

[2] pthread_cond_wait(3). URL http://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_cond_ 1997.

[3] http://www.intel.com/design/chipsets/datashts/290655.htm. 1999.

[4] iopl(2). URL http://linux.die.net/man/2/iopl. 2004.

[5] ossaudio(3). URL http://netbsd.gw.com/cgi-bin/man-cgi?ossaudio+3. 2009.

[6] audio(4). URL http://netbsd.gw.com/cgi-bin/man-cgi?audio+4+NetBSD-current. 2011.

[7] bus_space(9). URL http://netbsd.gw.com/cgi-bin/man-cgi?bus_space++NetBSD-current. 2011.

[8] mmap(2). URL http://linux.die.net/man/2/mmap. 2012.

[9] Kernel PCI drivers in userspace. URL https://github.com/rumpkernel/wiki/wiki/Repo:-pci-userspace. 2014.

[10] PCI passthrough support in bhyve. URL https://wiki.freebsd.org/bhyve/pci_passthru. 2014.

[11] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. *An empirical study of operating systems errors*, volume 35. ACM, 2001.

[12] P. Chubb et al. Linux kernel infrastructure for user-level device drivers. In *In Linux Conference*, 2004.

[13] J. Corbet. UIO: user-space drivers. *LWN.net*, 2007.

[14] Free Software Foundation. How to Use Inline Assembly Language in C Code. URL https://gcc.gnu.org/onlinedocs/gcc/Using-Assembly-Language-with-C.html. 2014.

[15] T. Friebel. *Übertragung des Device-Driver-Environment-Ansatzes auf Subsysteme des BSD-Betriebssystemkerns*. PhD thesis, Masters thesis, Technische Universität Dresden, March 2004. Available at http://os.inf.tu-dresden.de/papers_ps/friebel-diplom.pdf. 7, 22, 2006.

[16] ITU-R. BS.1116. *Methods for the Subjective Assessment of Small Impairments in Audio Systems Including Multichannel Sound Systems," International Telecommunication Union, Geneva, Switzerland (1994 March). Available at http://www.itu.int/rec/R-REC-BS.1116/en*, 1994.

[17] A. Kantee. Rump device drivers: Shine on you kernel diamond. *Proceedings of AsiaBSDCon*, pages 75–84, 2010.

[18] J. Kunz. *Writing Drivers for NetBSD. URL http://netbsd.mirrors.tds.net/pub/NetBSD/misc/agc/writing_drivers.pdf.* 2003.

[19] J. Lser and M. Hohmuth. Omega0: A portable interface to interrupt hardware for l4 systems. 1999.

[20] E. C. Poulton. *Bias in quantifying judgements*. Taylor & Francis, 1989.

[21] R. Short. Vice President of Windows Core Technology. *Microsoft Corp. private communication*, 2003.

[22] The European Commission. Directive 2009/490/EC of the European Parliament on the safety requirements to be met by European standards for personal music players pursuant to Directive 2001/95/EC. URL http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2009:161:0038:0039:EN:PDF. *Official Journal of the European Union*, 2009.

[23] H. Weisbach, B. Döbel, and A. Lackorzynski. Generic User-Level PCI Drivers. In *Proceedings of the 13th Real-Time Linux Workshop. URL http://lwn.net/images/conf/rtlws-2011/proc/Doebel.pdf*, 2011.

[24] Wikipedia. Student's t-test — Wikipedia, The Free Encyclopedia. URL http://en.wikipedia.org/wiki/Student's_t-test, 2014. [Online; accessed 30-December-2014].