Formalizing Interrupt Routing

Lukas Humbel
ETH Zurich
lukas.humbel@inf.ethz.ch

Timothy Roscoe
ETH Zurich
timothy.roscoe@inf.ethz.ch

February 13, 2017

I. Introduction

Due to recent trends, such as SoCs and rack-scale architectures, hardware is getting more *diverse*. Accelerator cards increase the *heterogeneity* inside a single system. On the software side, the trend goes towards direct hardware access in user space and virtualization, creating *new demands* for an operating system. State of the art operating systems are often designed with simplistic assumptions about the underlying hardware.

The complexity of current hardware makes it hard to write correct system software. One particular task system software has to deal with, is receiving interrupts and configuring interrupt controllers. Interrupts may originate from a variety of sources like devices or other CPUs. Unlike often stated in textbooks, each interrupt has individual constraints to which CPUs it may be delivered. For instance, the TI OMAP 4460 SoC contains multiple processors, such as two Cortex A9 cores, a DSP and two Cortex M3 cores. The DMA engine is able to generate different interrupts, some of them can be sent to all cores, while others can not be sent to the DSP. Furthermore, they appear as different vectors on each CPU.

To understand such hardware better, we started developing a formal model of interrupt delivery. It became clear that it overlapped considerably with the problem of modeling memory addressing [Ach17], and the same basic model could encompass both [AHCR17]. To give an intuition: The dual of a device issuing interrupt vectors is a CPU issuing memory addresses. A memory access undergoes multiple translations, for instance a virtual to physical address translation, likewise an interrupt undergoes multiple different representations, for instance when passing through an interrupt controller.

Similar phenomena do not only occur in SoCs, but also in commodity X86 systems, especially when combined with accelerator cards such as a Xeon Phi or hardware assisted virtualization. In the context of emerging rackscale architectures, we expect a great variety of interrupt systems.

Current approaches in operating systems are driven by the goal of configuring the system into a working state. Assumptions about both the hardware and the desired configuration often influence the design of data structures and algorithms. For instance Linux assumes that interrupts are always broadcast to all CPU's [BC05] and therefore does not differentiate between the same vector reaching different CPUs.

This has lead to abstractions, that are not flexible enough to cope well with today's hardware heterogeneity. The above assumptions prevents a single instance of Linux executing on both, the A9 and M3 cores. While it is possible to execute an isolated Linux on the M3 cores, it will not share any state with the one running on the A9 cores. One of the drawbacks of this solution is that threads are not migrated automatically between A9 and M3 cores.

II. Goals and challenges

The primary goal is to develop an all-embracing model of hardware notification delivery and an operating system built on top of that. It should cope with the diversity and heterogeneity found in modern machines. Unlike current designs, our approach is driven by a generic model that can express all hardware. Representations used in the operating system are either equivalent or additional assumptions are made explicit.

We would like to include a broader class of notification mechanisms: For instance the OMAP SoC mentioned in the introduction has a mailbox device to create interrupts on the M3 cores from the A9 cores. In a rackscale setting, RDMA networking can deliver notifications to remote machines.

We would like to proof correctness properties of configuration algorithms, such as creating a configuration that minimizes vector sharing or latency.

Configuration To do so, our model must be extended to contain information about the configuration options of interrupt translating nodes. This is complicated, since each interrupt controller

presents a unique set of configuration constraints. Typically the set of nodes where an interrupt can be directed is limited, but some controllers present other constraints, for instance: ARMs GIC controller can route vectors to different nodes, but the vector number must remain the same. MSI enabled PCI cards will always generate consecutive interrupt vectors. Also, to write a generic configuration algorithm, we need a representation of the model inside system software. Encoding the model in a space efficient and useful way is not easy since the configuration space is large and due to varying controller constraints very heterogeneous.

Multicast In notification systems multicast is often encountered. While our static model does express multicast, it needs to be extended to contain configuration options, especially what sets of destination nodes are permissible. Furthermore it should express the delivery mode: Whether the notification is delivered to all nodes in the destination set, to just one or to any other subset.

Reconfiguration The next step will be reconfiguration of the system. For example when migrating a device driver to another core, the destination of a device interrupt has to be changed. We would like to show that a reconfiguration algorithm does so without creating any notifications that are delivered to the wrong node. To do that, we must include a notion of atomicity in the model, since not all controllers support atomic updates.

Feedback In the process of working with this abstraction, we would also like to give feedback to hardware designers on what is good hardware from a systems programmer perspective. Such feedback can be derived from our model by either introducing a complexity metric on a given representation (like the maximum number of nodes an interrupt passes) or by investigating additional properties that allows our model to be represented much more compactly. For instance, we expect the configuration space representation to become much smaller, if each vector on all controllers can be controlled independently.

Authorization Since we were able to use the same model for memory accesses and interrupts [AHCR17], it might be possible to use the same protection mechanism for both. Thus reducing the amount of code that has to be trusted.

We plan to implement a prototype on the Barrelfish operating system. Specifically leveraging the System Knowledge Base [SBRP11], a declarative Prolog/CLP database that contains facts about the current system, to express the model and algorithms that work on it in a high level language.

III. Related Work

Some representations exist that are trying to informally describe hardware: The closest work to this proposal is made by Device Trees [dev16], which describes a binary file format now used extensively in the Linux kernel to handle non-discoverable devices. While a Device Tree captures some information about, for example, the addresses of devices as seen from a single core (the root of the tree), it has no well-defined semantics for interpreting the data. Moreover, it is not well-suited for heterogeneous systems where a single hierarchy is a poor match for hardware, and does not capture caches, TLBs, or the view of the system from DMA-capable devices.

Schüpbach *et al.* [SBRP11] applied a declarative approach to the configuration of the memory windows of PCI bridges. However, no comprehensive representation of the memory or interrupt subsystem is presented.

References

[Ach17] Reto Achermann. Provable correct memory management. In Submission to 11th EuroSys Doctoral Workshop (EuroDW 2017), April 2017.

[AHCR17] Reto Achermann, Lukas Humbel, David Cock, and Timothy Roscoe. Formalizing address spaces and interrupts. In Submission to 2nd Workshop on Models for Formal Analysis of Real Systems (MARS 2017), April 2017.

[BC05] Daniel P Bovet and Marco Cesati. *Understanding the Linux kernel*, chapter 4.6. O'Reilly Media, Inc., 3rd edition, 2005.

[dev16] devicetree.org. *Devicetree Specification*, release 0.1 edition, May 2016. `http://www.devicetree.org/specifications-pdf`.

[SBRP11] Adrian Schüpbach, Andrew Baumann, Timothy Roscoe, and Simon Peter. A Declarative Language Approach to Device Configuration. In *ASPLOS XVI*, pages 119–132, Newport Beach, California, USA, 2011. ACM.