1 Paper 1

The first paper for 2012 Advanced Operating Systems exam is titled 'ARMvisor: System Virtualisation for ARM' and was published in the 2012 Linux Symposium; based on the submission deadlines the paper would have been submitted before June.

1.1 Summary

The paper presents an implementation of a type 2 hypervisor built on Linux's Kernel-based Virtual Machine (KVM) infrastructure and based on the KVM for ARM project. The authors have implemented the hypervisor on a single processor ARMv7 architecture and have performed micro-benchmarking (with LMBench) of a single guest OS to compare their initial design, and their subsequent improved designs.

The motivation for this paper seems to be that virtualisation is inherently valuable, and current embedded systems built on earlier versions of ARM are not able to be virtualised. In addition the authors argue that existing hypervisors for ARM platforms are insufficient: OKL4 is closed-source, Xen for ARM uses para-virtualisation and requires maintenance to operate on different guest-ISA-hypervisor combinations, and KVM for ARM uses a costly memory virtualisation model that requires regular page table flushes.[1]

Because versions of ARM prior to ARMv7-A lack hardware support for virtualisation the authors have used para-virtualisation techniques to handle sensitive instructions from guests. In particular the authors used the Linux kernel's reverse map on frames to manage shadow page table synchronisation. Additionally the authors have implemented two abstractions — Direct Register File Access (DRFA) and Fast Instruction Trap (FIT) — to reduce virtualisation CPU and memory overhead. In the paper, they clearly show an improvement associated with these techniques over their initial design.

1.2 Critical Analysis

1.2.1 Strengths

Firstly in the paper's defence, the authors have built a hypervisor for ARM that seems to improve an existing system (KVM for ARM) and the implementation details they focus on seem sensible. Specifically they manage page-table synchronisation more sanely than KVM for ARM with a reverse

[1] I am trusting the author's assessment of KVM for ARM here. Also, I am not sure what a page table flush is. I assume it is like a TLB flush — a costly, undesirable synchronisation mechanism to ensure consistency — since the shadow page table has similar semantics to a TLB.

map lookup between guest pages and the shadow page table and deliberate protection faults on guest writes.

The authors have also implemented several other small optimisations. The optimised system keeps a shadow register file and avoids trapping to the host via software interrupts on several non-privileged sensitive instructions, such as writes to the CPSR. The system also adds a faster instruction trap to the guest OS's address space to avoid unnecessary lightweight traps into the host. Finally the authors have added some hyper-calls to give hints to ARMvisor about the guest page table usage. These optimisations all seem sensible and the tests prove that these improve on the author's original system.

In addition to their optimisations the authors have made their work freely available to interested parties. Firstly, this means that the systems the authors built is robust enough to be used in the wild. Secondly, their work is open-sourced (via the GNU GPLv2 license) and available on Github. There is also a user-guide to install ARMvisor (possibly on an emulated ARM CPU on top of x86) for anyone who is interested.[2]

1.2.2 Weaknesses

The first criticism I have for this paper is that I feel its motivation and its conclusion are weak. The authors begin by arguing that virtualisation for older versions of ARM is worthwhile for its security benefits in the context of embedded systems. This is a passing reference, and they do not treat the topic in any depth; they even use a Ubuntu host and guest OS which is probably the last choice for an embedded distribution of Linux. Finally, at the end of the paper the authors conclude nothing about ARMvisor, except that they improved its performance by focusing on two well known bottlenecks of hypervisors. They implement optimisations which improve the system, but they do not show that the system is practical or usable. I honestly don't see any significant point to this paper.

Closely related to the paper's motivation is the author's discussion of related work. If we put aside the fact that they discount all commercial solutions out of hand there are still two alternative projects: Xen on ARM and KVM for ARM. Xen on ARM is discounted apparently due to the maintenance cost of para-virtualised hypervisors. This seems particularly hypocritical as ARMvisor handles non-privileged sensitive instructions with para-virtualisation and implements custom fast instruction traps in the guest OS's address space[3]. In addition, as of the 7th of October Xen on ARM is a part of upstream Linux and is the first hypervisor supported by Linux on

[2]Neither of these facts reflect the quality of their work, but they are positive points in the interests of collaboration and reproducibility.

[3]The authors make a passing reference to research into pre-virtualisation as a solution to the maintenance issue but it is not very convincing.

ARMv5 and upwards.[4] Now obviously this occurred after the paper was submitted, but I think that it is unlikely the authors would have been unaware of the development of Xen on ARM while writing their paper; basically I think they may have deliberately avoided discussing Xen on ARM because it makes their work on a hypervisor for ARM seem far less significant.

On the topic of discounting other similar systems, I think there is serious issue with how the authors have discussed KVM for ARM. The authors claim is that they based a hypervisor on KVM for ARM and improved it, particularly with respect to page table synchronisation. Firstly they do not clearly describe how much their of system is based on KVM for ARM. Secondly they do not benchmark their system against KVM for ARM, despite it being an open-source project. Their claim that they cannot evaluate it because benchmarks and profiling results are not available seems very weak; even if benchmarks were available they should have reproduced them on their own experimental setup for an objective comparison. Their design may be better, or it may in fact be worse. I don't know and it was the author's job to convince me their design was better. On a last note, I think the improvements the paper describes seems quite logical which makes it hard to believe they've achieved something, or something that couldn't be achieved with minor changes to KVM for ARM.

The benchmarking in this paper is deeply flawed. My issues are as follows.

- The authors only present the results of micro-benchmarks (LMBench). They give no information about the overall system behaviour and efficiency.
- All the benchmarks compare the performance of different versions of ARMvisor. I have little idea of how effective the system really is.
- The benchmarking tool MiBench is mentioned but never used as far as I can tell.
- The system seems similar to KVM for ARM, except that the authors claim their design is better. Since KVM for ARM is also open-source, I don't see a reason not to include a comparison between these two systems.
- All tests involve a single guest on a single CPU. I don't feel this is a reasonable test setup for virtualisation.
- Table 7 presents a trap count for a do nothing program in the guest. Since these numbers are not referred to a native operating system, or a similar hypervisor I don't know what the numbers mean. Having said this, 7000 traps for a do nothing program seems very high.

[4] `http://blog.xen.org/index.php/2012/10/08/xen-arm-in-linux/`

- Figure 6, which presents the performance slowdown ratio of ARMvisor against a native system has so many problems. The extremely poor performance of their unoptimised system (100 times slower!) makes the slowdown of their optimisations seem much better (and much harder to read). In reality a slowdown of five or ten times on a virtualised system is abysmal.

Not only do the authors avoid macro benchmarks and real system performance, they completely avoid a discussion of I/O virtualisation and optimisation. As far as I can tell the I/O path though the system is:

1. guest user calls guest driver,
2. guest driver traps into ARMvisor,
3. ARMvisor passes the request to a virtual device in QEMU,
4. QEMU passes the request to the host driver,
5. finally the host driver accesses the hardware.

If this is the case then I'm not surprised the authors have avoided macro-benchmarks and discussing their virtualisation of I/O.

Finally, the paper's cost model seems pointless. The authors break down the emulation cost of running their system and use this model to conclude that a better hypervisor will use fewer traps with shorter delays (a completely banal claim). I am not sure why they included this section, by I feel like it was an attempt to add formality and gravitas to the paper for the purposes of publication. Their final conclusion that the cost model is a crucial tool to improve the performance of hypervisors is unfounded and weak.

1.3 Errata

This section lists several small grievances with the paper that I feel are not relevant to a technical criticism, but I found to be annoying. I appreciate that marking an extra page of material isn't appealing, so don't feel obliged to read this.

- Firstly the paper is afflicted by generally poor English and grammar. There are far too many examples to bother listing them all, but in particular the authors frequently fail to use the singular and plural forms of nouns and verbs correctly. In addition, their use of the word vowing — admittedly only twice — was particularly annoying.
- In the cost model section, which I have already criticised, there are several other minor issues. Firstly equation 3 is incorrect: it omits

T_{io} and adds an undefined delay T_{idle}. I think the correct equation would replace T_{idle} with T_{io}. Secondly, several symbolic system delays are not defined, such as: T_{inst}, T_{excpt}, T_{abt}, T_{shadow}, T_{sync}, T_{mmio}, and T_{portio}. Finally they neither define the trap counts, Cx, or the limits of the summation, or what the terms of the summation, i and j, actually refer to and why the delays depend on these terms. That was a long sentence and I apologise.

- The authors use both the terms hypervisor and VMM. I am under the impression these are synonyms, and I feel that using synonyms in a technical paper is unnecessary and adds ambiguity.

- When reviewing alternative systems the authors phrase the commercial success and support for security of OKL4 as 'claims'. This terminology may be innocent, but I feel that it is more likely a subtle attempt to put down a competing system. Furthermore the authors state that OKL4 has been 'ported to millions of mobile handsets'? A more appropriate description would be 'hundreds of millions', or 'over a billion' and again this feels like an attempt to misrepresent the significance of a competing design.

- The choice to virtualise Ubuntu on top of Ubuntu is curious. The authors do not specify what distribution they use in their benchmarks and so I would guess that it is also Ubuntu. I'm not sure why the authors made this decision (as opposed to a more light weight distribution or one better suited to embedded or real time systems) and they haven't really offered any explanation.