

Porting the Xen Hypervisor to ARM

Michael LeMay
UIUC
mdlemay2

Dongyun Jin
UIUC
djin3

Sundeep Reddy
UIUC
skatasa2

Brian Schouderl
UIUC
Brian.S@willinois.edu

ABSTRACT

We are attempting to port the Xen hypervisor, as well as the associated Linux guest operating systems, to the ARM architecture. This would have an important effect on the security and functionality of current and future applications in the realm of embedded systems.

1. INTRODUCTION

Embedded platforms are used in drastically different application domains than standard personal computers. Cellphones, PDAs, manufacturing automation equipment, and critical infrastructure control and monitoring units all rely on processors with restricted capabilities and support infrastructure. The processors often implement the ARM instruction set. Currently, no open source virtualization technologies have been developed that support these architectures. However, there are a number of compelling applications that argue strongly for just such a technology.

Cellphone viruses [3] pose a grave threat to a world that depends critically upon mobile phones for both routine and vital communications. Current cellphone operating systems execute in a single address space, and provide little to no resistance against such viruses. If an unsuspecting user downloads a corrupted game or ringtone, their entire phone could be disabled or coopted for an arbitrary purpose. By supporting virtual machines on the cellphone, such a dire scenario could be averted, as untrusted code could be completely isolated from the rest of the software on the device.

Even more damaging scenarios can be imagined for embedded computers in critical infrastructure applications. Even now, smart electric meters are being deployed in a number of markets such as California and Canada. These meters often have the capability to both accept commands from the power control center and transmit statistics back to the control center. The commands that can be executed include switching the customer's power on or off, transmitting a real time pricing schedule, etc. Conceivably, power

companies may one day wish to install custom applications on these powerful meters, and attackers may possibly exploit those same channels to inject their own illegitimate applications. If these malicious applications could be isolated within a restricted virtual machine environment, their damaging effects could be contained. On the other hand, if smart meters are equipped with operating systems like those on cellphones, they will likely be subject to similar attacks, but with even more serious consequences (such as power outages or corrupted bills). Unfortunately, current researchers do not seem to be aware of these possibilities [4].

It is quite apparent in light of this discussion that providing virtualization support on the ARM architecture could have important implications in a variety of critical technology sectors. In fact, both of the areas we discussed are likely more critical from a societal standpoint than either general web servers or personal computers, which are the most commonly virtualized resources.

2. PROJECT OBJECTIVES

The over-arching objectives for this project are to port both the Xen hypervisor and the associated Linux guest operating systems to the ARM architecture, to enable ourselves and others to achieve the benefits presented by the introduction in future projects. We review our most important milestones here:

1. Repeatably install Linux distribution on QEMU ARM emulator
2. Boot Xen VMM under QEMU
3. Load Linux guest OSs under XEN VMM on QEMU

The first milestone, repeatably installing Xen on the QEMU ARM emulator, is very important because Xen usually hosts Linux guest OSs. If we are unable to build a standard Linux distribution for the ARM architecture that is capable of being run under QEMU, we stand a very slight chance of running a Linux guest OS under the Xen hypervisor later, which is of course our final goal.

We are drawing on previous projects for inspiration and concrete assistance in achieving these milestones. In particular, the IA64 port of Xen [7] has proven to be useful.

3. TOOLS AND BACKGROUND

For all of the following steps, we used the CodeSourcery ARM cross-compiler [5], with support for the ARM EABI. The Subversion source code control system also proved invaluable in managing the substantial changes we have performed and are currently performing on the Xen codebase.

4. CHALLENGES AND SUCCESSES

4.1 Booting Linux on QEMU

We have achieved our first milestone, and have developed a set of repeatable instructions for configuring and compiling a Linux 2.6.16-rc3 kernel to run on QEMU. In fact, we compiled a Linux kernel to which the Xen patches had been applied, although all Xen support was unavailable and disabled on ARM. We also applied the kernel patches available from the download section of arm.com.

Even achieving this preliminary milestone was far from trivial. QEMU has a number of limitations that make it quite difficult to configure a Linux kernel that will actually boot on it. During our first attempts, our kernel would not even get past the following status line:

```
Uncompressing Linux.....done, booting the kernel.
```

Due to the ambiguous nature of this failure, it took a great deal of experimentation to finally detect its root cause. We finally enabled verbose debugging messages in the kernel configuration, and noticed that the kernel was crashing as it was initializing the VGA console driver. We later confirmed by searching online forums and browsing the QEMU sources that QEMU does not provide VGA support for the ARM architecture. Ultimately, we were able to overcome this problem by disabling VGA support in the kernel configuration.

A kernel is only the most basic component necessary in a system. We wanted to construct a realistic and complete Linux installation, to enable us to exercise the full capabilities of our virtual machine after it is completed. Fortunately, ARM (the company) provides a set of scripts and precompiled packages to generate a customized root filesystem image with minimal additional effort. We created a stripped-down filesystem image that includes basic system tools, networking tools, editors, etc. We also included detailed instructions for recreating this image in our collection of deliverables which will be provided at the completion of our project.

Finally, we decided to exercise the more advanced functionalities provided by the QEMU ARM emulator, so we configured NFS support. In our configuration, the host system on which QEMU is being executed is setup to export an arbitrary filesystem, and the client running within QEMU is configured to import the filesystem. This worked perfectly, and we even developed a simple script to automatically unpack the cramfs root filesystem created previously, modify the `/etc/fstab` contained therein, and recreate the filesystem image.

In summary, this phase provided us with a great deal of experience working with the idiosyncrasies and limitations of

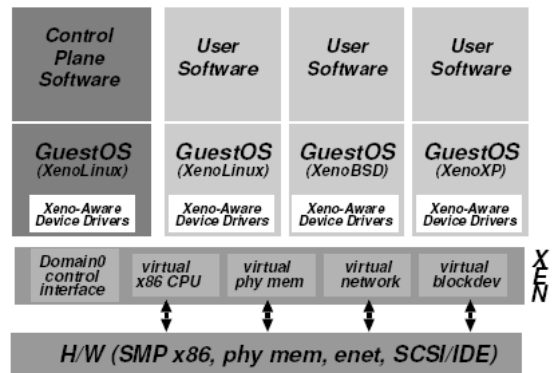


Figure 1: Architectural layers in Xen virtualized system

the QEMU ARM emulation component, as well as experience building a full-featured Linux system image for ARM. This will be very important later when we want to port the Linux guest OSs to the Xen/ARM hypervisor.

4.2 Porting the Xen Virtual Machine Monitor to ARM

The second phase of our project involves porting the Xen hypervisor to run on the ARM. In the layered diagrams that are commonly presented in papers dealing with Xen, such as the one drawn from [1] and presented in Figure 1, this portion of the system is the one labeled “Xen” on the right. The hypervisor is the first piece of software to boot on the virtualized machine, and it subsequently loads guest operating systems.

We could have approached this porting effort in either of two ways. We could have started from scratch and attempted to implement all necessary Xen API functions for the ARM architecture. Alternatively, we could copy an existing implementation and adapt it to the ARM. We selected the second approach, since we believe that it will help us to stay aligned on what functions we actually need to implement, and because we believe that we will not be required to modify all of the code we copied. Some should operate without modification, as long as we adapt the functions upon which the shared code depends.

Having selected an overall approach, we were forced to select between three potential architectures to base our modifications upon. Xen currently supports three main architectures: IA32, AMD64, and IA64. The IA32 and AMD64 architectures share most of their code, since the AMD64 architecture essentially comprises a set of extensions to the IA32 architecture. However, IA64 support was added to Xen after both of the other architectures, and its architecture-specific codebase is much smaller and better organized than that of either IA32 or AMD64. Thus, we copied the IA64 codebase and based our work upon it.

Before proceeding, we present the overall layout of the Xen codebase. We selected the Xen-3.0.1 distribution for our work, since it was the latest “testing” release at the time

we began the porting effort in earnest. This release contains the following top-level directories:

1. **buildconfigs**: Makefile rules.
2. **docs**: Documentation.
3. **extras**: Contains a subdirectory, `mini-os`, which is a minimal OS used for testing purposes.
4. **linux-2.6-xen-sparse**: Stripped-down Linux kernel distribution used for building basic dom0 and domU guest OS kernels.
5. **patches**: Full complement of patches for the Linux kernel source tree, presumably used for building full-featured guest OS kernels.
6. **tools**: Set of tools used for monitoring and controlling virtual machines.
7. **xen**: Source code for the Xen hypervisor itself, which is what we are currently attempting to port.

We will be focusing exclusively on the `xen` subdirectory, because that is the directory containing the source code for the Xen hypervisor, which this phase is concerned with. This subdirectory contains a number of additional subdirectories, which we discuss here:

1. **acm**: Sources for implementing the sHype [2] mandatory access control scheme.
2. **arch**: Architecture-specific Xen hypervisor implementations. Originally contained subdirectories for the IA32, IA32 PAE, AMD64 (`x86`), and IA64 (`ia64`) architectures. We introduced a new subdirectory for the ARM architecture (`arm`). Each subdirectory includes a set of files for actually implementing the Xen architecture-specific API, and also includes a fragment of the Linux 2.6.12 kernel architecture-specific source tree for the appropriate architecture. We actually copied ARM-specific files from kernel 2.6.16-rc3 since that is the version of the kernel that we successfully compiled.
3. **common**: Architecture-independent Xen hypervisor API implementation. Includes such items as the implementation of `printf`, and support for the ELF binary format.
4. **drivers**: Contains drivers for a generic serial-connected console device, as well as ACPI power management functionality. We disabled ACPI support for the ARM build, since ARM includes its own proprietary power management support (which we ignore).
5. **include**: Contains both architecture-independent and architecture-specific include files. Again, we copied the `asm-ia64` directory to form `asm-arm`. Similarly to the `arch` directories, each `include` directory includes the kernel headers specific to the architecture in question. We replaced the copied IA64 kernel headers with ARM headers, again from kernel 2.6.16-rc3.

6. **tools**: Contains two auxiliary, architecture-independent tools, one of which (`figlet`) is used to generate ASCII-art block letters automatically. Apparently Xen's creators had a sense of style.

After copying the appropriate files into ARM-specific locations, as explained above, we began our modifications. The first modifications we performed were to the Makefile rules in the newly-created `arch` directory. Xen automatically includes an architecture-specific Makefile fragment for the selected architecture. We modified the makefile to use the CodeSourcery cross compiler, and to pass the appropriate compiler flags for the target architecture (ARMv5).

Next, we attempted to compile the architecture-independent portions of the hypervisor. However, these include architecture-specific files for the target architecture, so we needed to perform significant modifications to the header files simply to get the architecture-independent components to compile. We did succeed after a short while however, and then proceeded to the architecture-dependent implementation files. We have only ported a couple of these files thus far, but have not encountered many specific difficulties in doing so. It simply takes a long time to analyze and port each file, so we were not able to generate a bootable hypervisor image by this time, in spite of our initial schedule projections.

Of the difficulties we have encountered, a few are notable. First, the ARM instruction set does not provide any way to access a timestamp counter, such as those accessible via the `rdtsc` instruction on Pentiums, and a special register on the IA64. As explained in [6], most of the time sources provided by Xen rely on the TSC to some extent. Thus, we must either emulate the TSC, or rewrite the functions that depend upon it. We expect it to be easier to adopt the latter approach, since Xen clocks appear to be implemented independently by each architecture. Thus, we should be able to easily modify the functions in `arch/arm/xen/xentime.c` to be governed by a periodic timer interrupt rather than a time stamp counter.

Another problem that appears frequently during compilation attempts is the lack of SMP support in the ARMv5 architecture. (The ARMv6 does support SMP, however) Of course, the devices we are targeting likely will not actually use SMP for some time (I can't wait for my 32-core smartphone :-), but support for SMP is ingrained very deeply in x86/IA64 Linux, as well as Xen. We would not be encountering this problem if QEMU enabled us to target the ARMv6 architecture, but a large majority of devices use the ARMv5 or older architecture, so it is probably best that we are forced to confront this issue. We hope to overcome it by convincing the kernel sources that they are operating in a non-SMP environment, while retaining SMP support in the rest of the sources to facilitate future development on newer architectures.

Third, we expect to encounter a number of problems when migrating to the ARM MMU. This MMU is quite different from that in the x86, and we will most likely need to rely very heavily on the MMU code in the ARM Linux kernel for guidance while porting this portion of the system.

Last but not least, the ARM architecture only supports two protection rings, rather than the four provided by IA32 and IA64. However, the AMD64 architecture also dispensed with the two extra rings, so we hope to borrow some of the management techniques included in the x86_64 architecture-specific implementation.

5. FUTURE WORK AND CONCLUSIONS

We have completed the first milestone of our project, and have made significant, steady progress towards our second milestone. We are not currently blocked by any unsolvable problems, and are very optimistic about the challenges we have just discussed. After we complete our second milestone, we will immediately begin work on porting the Linux guest OS to run on the hypervisor. Since we will have a solid understanding of what aspects of the hypervisor we modified, we should be able to predict which parts of the kernel will need to be adapted. Overall, we are very optimistic about completing this project by the end of the semester, and believe it will enjoy broad applicability after being released to the general community.

6. ACKNOWLEDGEMENTS

Ellick Chan has been a great resource throughout this project and has helped keep us on track and moving forward. We have also received helpful email responses from Paul Brook during our work on compiling Linux on QEMU.

7. REFERENCES

- [1] P. Barham, et.al.; “Xen and the Art of Virtualization”, <http://www.cl.cam.ac.uk/Research/SRG/netos/papers/2003-xensosp.pdf>
- [2] “sHype - Secure Hypervisor”, http://www.research.ibm.com/secure_systems_department/projects/hypervisor/
- [3] J. Wexler, “Smartphone virus, spam threats loom”, <http://www.networkworld.com/newsletters/wireless/2004/1220wireless1.html>
- [4] E. Beronet, “Securing Meter Data on Automated Meter Reading (AMR) Systems: How important is it?”, <http://www.electricenergyonline.com/article.asp?m=9&mag=32&article=254>
- [5] “CodeSourcery”, <http://www.codesourcery.com/>
- [6] “Xen interfaces”, <http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/interface.pdf>
- [7] H.K.F. Bjerke; “HPC Virtualization with XEN on Itanium”, http://openlab-mu-internal.web.cern.ch/openlab-mu-internal/Documents/2_Technical_Documents/Master_thesis/Thesis_HarvardBjerke.pdf