Virtual machines

The State of the Art

Bachelor Thesis for Computer Science

Erik van der Kouwe
Student number 1397273
E-mail: erik@erisma.nl

Vrije Universiteit Amsterdam
Faculteit Exacte Wetenschappen
Afdeling Wiskunde en Informatica

Supervised by
Andrew S. Tanenbaum

30 July 2006

Abstract

Virtual machine monitors partition a single physical machine into multiple virtual ones. This can be useful for several important applications, such as running multiple isolated servers on a single machine, testing and debugging software, using possibly malicious software and building honeypots.

The widely used IA-32 architecture does not natively support virtualisation. We compare several state-of-the-art methods used to circumvent this problem. Our comparison includes dynamic binary translation, paravirtualisation, previrtualisation, operating system level partitioning, application virtual machines and the recent "Virtual Machine Extensions" added to IA-32 by Intel. Our criteria for this comparison include performance, robustness, relationship with the host operating system and portability.

We use the results of the comparison to determine which techniques are most suitable for which applications. We find that the new hardware virtualisation support looks very promising for many applications, but may not completely replace other methods.

1 Introduction

What is a virtual machine?

Virtualisation is a technique which allows one to partition a computer system in multiple completely separate systems. Each of these provides a software environment which is very similar to that of a complete computer. Such an environment is called a virtual machine (VM).

One will typically want to install an operating system on a virtual machine to be able to run applications. This guest operating system assumes that it has complete control of the computer, and it will attempt to access it's hardware. This cannot be allowed, since the hardware is shared with guest operating systems running on other virtual machines. A program called virtual machine monitor (VMM) or hypervisor is needed to make sure all resources are shared properly.

The role of a VMM dividing resources between operating systems differs from that of a kernel dividing resources between applications. The main difference is that the latter typically provides an abstraction of physical devices, while the former does not change the abstraction level [16]. The VMM should present a faithful low-level interface to virtualised hardware.

The VMM itself may have full hardware access, but it can also can be a normal application running on an operating system. In this case the operating system on which the VMM runs is called the host operating system. This situation is shown in figure 1.

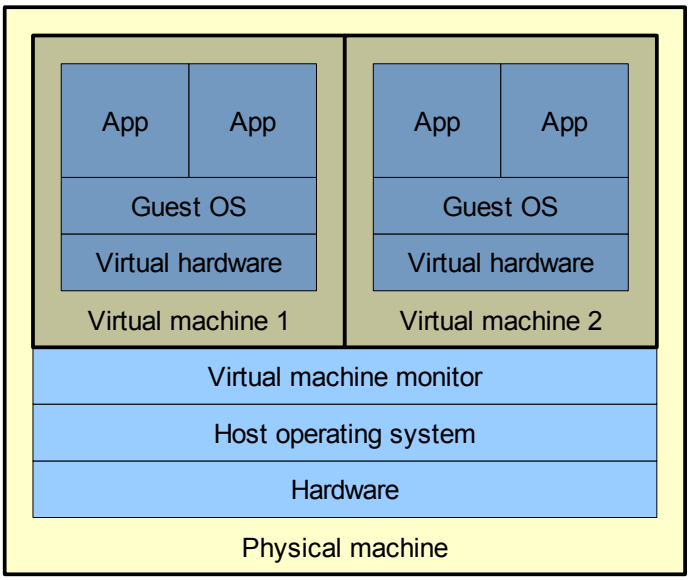

Figure 1: A computer running two virtual machines

Why are virtual machines useful?

Virtual machines have many practical uses in many different kinds of environments.

Servers

One can use virtual machines to run multiple isolated virtual servers on a single physical server. This allows hardware to be used more efficiently and decreases hardware costs. For an internet hosting company this can be used to allow customers full access to a virtual server without endangering other servers on the same physical machine.

Another advantage of using virtual servers is the fact that they can easily be moved to other computers. Typically the interface to the virtualised hardware does not depend on the actual hardware, so the guest operating system does not even notice the move. This can be used to minimise downtime after restoring a backup to different hardware.

Software development

Virtual machines have their uses in software

development as well. They provide a way to switch between different (versions of) operating systems easily and quickly. This is very useful for testing and debugging software on multiple platforms.

VMMs may also allow the user to make snapshots of virtual machines. This means one can test something and revert the machine to it's original state afterwards. This is again very useful for testing and debugging.

Yet another application for development is to debug low-level software, such as kernels. A VMM could provide a kernel debugger with much information about what is going on. It also makes recovery from crashes easier, since one can simply use a previous snapshot.

Untrusted software

Another application which is especially useful nowadays is the ability to run untrusted software in an environment where it can do no damage. Examples are running potential malware downloaded from the Internet and opening suspicious e-mail attachments. If the VMM is secure enough, there is no way for malware and viruses to infect the physical machine or other virtual machines.

Honeypots

Finally virtual machines provide an easy way to build secure "honeypots". These are unprotected machines which are connected to the Internet. The purpose is to get information about new methods to exploit flaws in operating systems and applications. By using virtual honeypots, these exploits are less likely to do damage to the system inself. This may make investigation and recovery easier. They also allow having multiple honeypots with different (versions of) operating systems running simultaneously.

2 Related work

There are several papers comparing different virtualisation techniques. Kiyanclar [10] compares several virtualisation programs to select the one which is most suitable for secure on-demand cluster computing. His criteria are similar to ours. Nanda and Chiueh [12] discuss some implementation details for a large number of VMMs. Both focus more on specific implementations than on characteristics inherent to techniques. Rose [15] discusses advantages and disadvantages of full system virtualisation and paravirtualisation.

In-depth articles comparing virtual machines are typically limited to a single technique. An example is Gaugh [5], who compares the Java Virtual Machine and .NET, which are both application virtual machines. Ars Technica [2] compares two major dynamic binary translation VMMs, but considers only performance and ease of use.

A large comparison table of virtualisation software is found on Wikipedia [21]. This table provides a simple overview over the basic characteristics of many virtualisation programs. The comparison focusses more on software than on techniques and more on features than on applications. Note that, at the moment of writing, this article is considered to be in need of expert attention and clean-up.

3 Theoretical background

Popek and Goldberg

Popek and Goldberg [13] investigated sufficient conditions which allow a computer architecture to support virtual machines. They defined a virtual machine as "an efficient, isolated duplicate of the real machine". Although their research was motivated by the question why IBM 360/67 could support virtual machines while DEC PDP-10 could not, their criterion still applies to modern architectures.

Central to Popek and Goldberg's theorem is the distinction between supervisor and user modes. The supervisor mode allows complete access to the machine and is typically used by operating system kernels and VMMs, while the user mode is more limited and is typically used by applications. They further define a trap operation, which places the processor in a stored state (typically the supervisor mode) while saving the current state.

Popek and Goldberg consider some properties that individual instructions can have:

- A privileged instruction performs a trap operation in user mode, but does not trap in supervisor mode;
- A control sensitive instruction can change the operating mode or virtual memory mappings;
- A behaviour sensitive instruction executes in

different ways depending on operating mode or virtual memory mappings;

- A sensitive instruction is control sensitive and/or behaviour sensitive.

Having defined these terms, we can state the theorem: "A virtual machine monitor may be constructed if the set of sensitive instructions for the computer is a subset of the set of privileged instructions". The virtual machine is constructed by letting the VMM operate in supervisor mode and the virtual machine in user mode. This way the monitor can emulate sensitive instructions, since it is notified by a trap operation. Non-sensitive instructions can safely be executed directly.

Virtualisability of IA-32

The IA-32 architecture is very widely used in personal computers and servers which run Windows, Linux or, more recently, Mac OS. It evolved from (and is still compatible with) the instruction set architectures used by the 8086 and it's successors and is therefore also commonly known as x86. The Pentium D and Core Duo chips from Intel and Opteron and Athlon 64 X2 from AMD are examples of modern implementations of IA-32.

IA-32 defines four security rings numbered 0 through 3. Ring 0 can be considered the supervisor mode, while the other rings correspond with the user mode. The architecture defines privileged instructions which, when executed in user mode, trap by calling an interrupt handler in ring 0. Virtual memory is implemented through segmentation and paging. Segments are identified by 16-bit segment selectors which contain the number of the least privileged ring allowed to access them. The operating system's perspective of IA-32 is described at length in [7].

Robin and Irvine [14] have investigated the IA-32 architecture and have found many instructions that are sensitive, but not privileged. An example is the "PUSH CS" instruction, which pushes onto the stack the selector for the segment containing the currently executing code. Since this segment selector contains the number of the current security ring, this instruction is behaviour sensitive.

The lack of native virtual machine support has lead to the use of many different techniques on the IA-32 platform. The approaches we mention are applicable to other architectures, but we will focus on IA-32. We will also discuss two approaches which are highly similar to virtual machines and provide the same advantages, but which are not virtual machines according to the definition we presented before.

Recently Intel has added true virtualisation support to their newest IA-32 chips by including an extension instruction set called "Virtual Machine Extensions". We will discuss this technology as well.

4 How to compare

Before one can meaningfully compare virtual machine techniques, one has to determine which criteria to use to distinguish the methods. We will use several characteristics to do this, each of which may or may not be relevant depending on the reason we use virtual machines.

Performance

One important criterion is performance. Good performance is desirable for each of the applications we mentioned, and essential for some of them.

From the applications we discussed, performance is most important in case virtual machines are used for isolated server consolidation. The more efficient the VMM is, the more virtual servers can be run on a single physical server. This means that solutions which perform better require fewer servers.

When used for debugging, a VMM which performs well is pleasant, but not essential. The same goes when virtual machines are applied to isolate untrusted programs or to secure honeypots.

Note that we will focus on techniques rather than on their implementations. As such, our aim is not to compare benchmark results. Instead we will look at the performance potentials inherent to the methods we evaluate.

Robustness

Virtual machines should provide absolute isolation and, as such, perfect security. The only means for communication between virtual machines should be the virtual network connection. Unfortunately, perfection is hard to come by. Both the VMM and the host kernel are pieces of software and, as such, we can expect them to contain bugs. We will therefore evaluate the robustness of the security in the presence of bugs. We will find that in some cases

the VMM can increase robustness, while in others the isolation entirely depends on the ability of the kernel to isolate applications.

Secure isolation is the core feature when virtual machines are applied to isolate untrusted programs or to secure honeypots. It is also of utmost importance when used for server consolidation, since a breach of security would mean down-time. In each of these cases we can expect malicious attempts to break security.

For debugging purposes we rely less on the robustness of VMMs. Although crashes of the virtual machine are possible and should not effect the physical machine, we do not expect any malice.

Relationship with host operating system

A VMM can have different kinds of relationships with the host operating system. It may be an unprivileged application, but it may also be require cooperation from inside the kernel. A third possibility is that no host operating system is present. In this case one could say the VMM itself acts as a minimal operating system.

When used for debugging purposes, it is desirable that the VMM can run as an application. This avoids restarting and allows one to run other applications besides the VMM while debugging. It is preferable that kernel cooperation is not required, because one would typically want to avoid installing kernel-mode drivers since these may make the operating system less stable. Another problem is that installing such a driver is normally only allowed for the root user. The same reasoning goes when virtual machines are used for running untrusted applications. Note that need for kernel support is not a problem if the feature is integrated in the kernel by default.

For servers it is preferred that the VMM run directly on the machine, without a host operating system present. This reduces overhead and may be more stable. It does require that the server hardware is supported by the VMM.

For honeypots the relationship with the operating system is not very important.

Portability to multiple guests

Being able to support many different guest operating systems without additional effort is important when virtual machines are used for debugging applications. A developer typically wants to test not only on multiple operating systems, but also on different versions of the same operating system. The same goes for honeypots, since different versions may have different vulnerabilities.

When virtual machines are used for server consolidation or running untrusted applications, it is sufficient that one recent version of each needed operating system is supported.

5 Techniques in use

Dynamic Binary Translation

Dynamic binary translation can be seen as an advanced way to do emulation. In case of pure emulation, the host software implements the instructions that are available on the CPU. This allows it to interpret the instructions supplied by the guest. As such, emulation is the most obvious approach to build virtual machines on hardware platforms that have no native support for them. Bochs is an example of a program which emulates IA-32 CPUs.

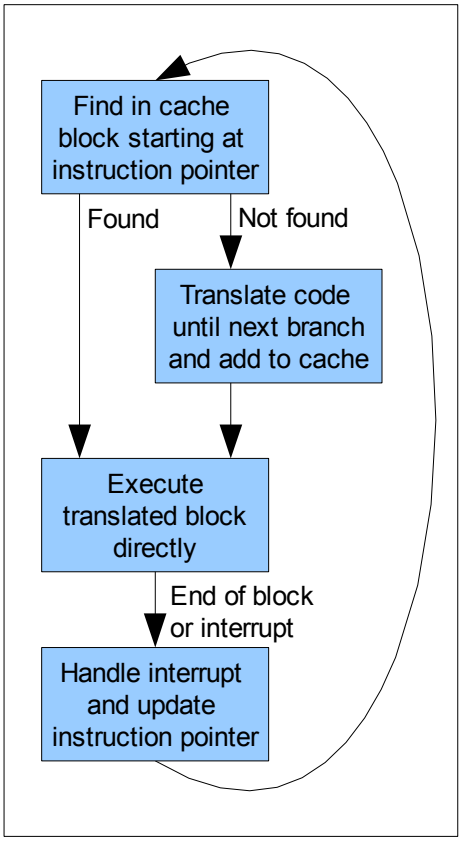

Figure 2: Dynamic binary translation

Unfortunately pure emulation provides very poor performance, since executing a single guest instruction typically takes many instructions on the host machine. This lack of efficiency means that an emulated machine does not satisfy the Popek and Goldberg definition for a virtual machine. As such, we will not consider emulation separately.

Dynamic binary translation overcomes the performance

limitations of emulation by translating the instruction stream to host instructions which can be executed natively. In this step sensitive instructions are replaced with calls to the VMM. The translated instruction stream can be cached. Because of this, only a small part of the time is spent on translation. The main loop of a dynamic binary translation VMM is shown in figure 2.

Ung and Cifuentes [18] provide details on how one can implement binary translation.

Translation can be more efficient if the host has the same architecture as the guest, and many of the available dynamic binary translation VMMs only support running IA-32 guests on IA-32 hosts.

Dynamic binary translation is widely used. Market leader VMWare created several VMMs which are based on this technique: VMWare Workstation runs as an application on Windows and Linux and VMWare ESX server runs without an operating system. The architecture of the virtualisation software is described in [20]. For the server version, this software runs on a proprietary microkernel operating system [19].

Another example of a VMM which uses dynamic binary translation is QEMU. This program provides more insight in the implementation of this technique, since it is open source. It's author describes the internals in [4]. QEMU translates CPU instructions to C code, which is compiled using the GCC compiler. This results in very good portability, since GCC has been ported to many platforms. Translation happens in blocks ending at the next potential jump or important change in CPU state (such as changing mode of operation). These blocks are stored in a cache. Pages containing translated code are marked write-only, and by handling the resulting protection faults QEMU can invalidate the cache when code changes.

Other examples of virtualisation products which use dynamic binary translation include Microsoft VirtualPC and VirtualServer, Parallels Workstation and Serenity Virtual Station.

With dynamic binary translation much of the code is translated only once and can be executed natively. We therefore expect the chaining of translated blocks to have the most important impact on performance. Benchmarks show [2] that a highly optimised binary translation VMM such as VMWare Workstation can reach good speeds, but worse results should be expected in branch-intensive or self-modifying code.

The host kernel should ensure that the guest code is executed in a user mode ring, typically ring 3. All of this code is generated by the VMM. This provides the additional guarantee that the guest cannot attempt to call the host kernel directly. This results in double protection, making it unlikely that a single bug in either the kernel or the VMM compromises the isolation. As such, dynamic binary translation can be very robust.

Our examples show that binary translation VMMs can be highly portable. In principle, the guest operating system does not matter at all, as long as the VMM faithfully implements all hardware interfaces it requires.

A binary translation VMM can run either as an unprivileged application within an operating system or with full control without operating system. We have seen examples of both.

Paravirtualisation

The approach we described before tries to mimic the environment in which the operating system is running on a real machine. For this reason it is also called full system virtualisation. More efficient virtualisation may be possible if the VMM cooperates with the guest operating system. This approach is called paravirtualisation.

Xen is an example of a virtual machine monitor which uses paravirtualisation. It requires that guest operating systems be changed to make their kernel run in ring 1 instead of ring 0. The ported kernel uses “hypercalls” to replace sensitive instructions. This is done by placing parameters in registers and then calling an interrupt handler. Xen itself runs in ring 0. The approach is shown in figure 3.

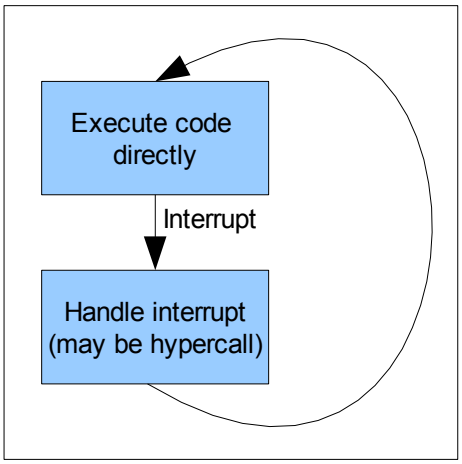

Figure 3: Paravirtualisation

Xen's creators ported Windows XP, Linux and BSD to run on Xen [3], although Windows is not publicly available. Minix for Xen is also available, and the document [9] describing the

porting process shows how a simple microkernel operating system can be ported to Xen.

Another example of paravirtualisation is User Mode Linux. This is a modified version of the Linux kernel which is capable of running as a user-mode application inside Linux.

Paravirtualisation can be expected to be very fast, since all code can be executed without runtime translation. The only overhead is having to use hypercalls instead of accessing hardware directly. This may affect the performance slightly, but this is unavoidable since direct hardware access is unacceptable in a virtual machine.

With paravirtualisation, code is not translated. This means that we must be careful not to allow it any access to the physical machine. Since all of this code runs in user mode, the only way out should be calling the host kernel (at least, if the host kernel properly sets up permissions). This should be prevented, as it might allow virtual machines more access than they are entitled to.

If direct access to the host kernel is prevented, then the security of a paravirtualising VMM relies on the ability of the host operating system to securely isolate applications. If no host operating system is installed, such protection should be provided by the kernel included in the VMM. In both cases the robustness can be expected to be at the same level as the protection between applications delivered by the host operating system. Paravirtualisation lacks the double security provided by binary translation.

For the relationship with the host operating system, we have seen that a paravirtualisation VMM can run directly on the system (like Xen) or as an unprivileged application inside a host operating system (like User Mode Linux).

With paravirtualisation, each guest operating system has to be ported specifically. This takes some effort and can only be done if the source code is available. A closed-source operating system can only be used if it's owner ports it. Hence portability is rather bad for paravirtualisation.

Previrtualisation

LeVasseur et al. [11] suggest a modified version of paravirtualisation, which they call previrtualisation. In this case sensitive instructions are replaced by the compiler, drastically reducing the effort needed to perform the port. This can only be done if the hypercall interface is sufficiently similar to the interface sensitive instructions provide to the CPU,

We are not aware of current real-world usage of previrtualisation. The University of Karlsruhe is currently developing experimental VMMs which apply previrtualisation. Marzipan runs without an operating system and is based on the L4Ka Pistachio microkernel developed at the same university. It is used for research. BurnNT runs as an application on Windows XP. This program is currently in an early stage of development, being able to start the Linux kernel, but not supporting user applications running on it yet.

Previrtualisation is very similar to paravirtualisation, but some differences in characteristics can be expected.

The restriction that the hypercall interface be similar to the interface provided by the CPU may result in lower performance than paravirtualisation, where for example multiple actions may be merged in a single hypercall. Performance can therefore be expected to be slightly worse than for paravirtualisation. Because code is not translated at runtime, a significant performance benefit over binary translation can still be expected.

Portability is significantly better than for paravirtualisation, since it should be possible to port open source guest operating systems with little effort. Closed-source operating systems still require that owner of the operating system cooperate with the porting effort.

Operating system level partitioning

We will discuss operating system level partitioning only briefly, as it is not really a virtualisation technique as defined by Popek and Goldberg. It is still an interesting technique because it is much more light-weight than the solutions discussed before.

In case of partitioning, virtualisation support is provided by the operating system. The applications running on the operating system are partitioned in several isolated groups. The implementation for system calls make sure that these groups cannot interact in ways different than interaction between separate machines. This means that they do not share the same filesystem and that one who has root access to a single partition may not have this privilege for other partitions or the machine itself. Kamp and Watson [8] describe partitioning as implemented by

FreeBSD, where the feature is called "jails".

Solaris 10 implements this feature and names it "Zones". Partitioning on Solaris is highly configurable, allowing partitions to share some read-only directories to avoid duplication of data. OpenVZ and Linux V-Server are patches for the Linux kernel which allow one to use partitioning on Linux.

Performance for partitioning is very good. Both applications and the kernel are executed natively and hypercalls are not needed. The approach is very lightweight because there is no need (and indeed, no possibility) to run multiple operating systems. Overhead is limited to some extra security checks in the implementations for system calls.

The approach is as robust as the kernel itself, since no separate VMM is used.

The relationship with the operating system is inherently that the VMM is part of the kernel.

Portability of the partitioning solution is bad, since different partitions are all running the same operating system as the host.

Application virtual machine

Yet another virtualisation approach which is interesting despite not satisfying the Popek and Goldberg definition is the use of application virtual machines.

An application virtual machine does not duplicate a real machine, but instead exposes an instruction set architecture especially designed for virtualisation. Such an instruction set is designed for running applications, not operating systems. A simple VMM is in between the guest application and the host operating system. This VMM typically compiles the guest instructions and the caches the result. This is similar to dynamic binary translation, but in this case the job is simplified by a good choice of virtual instruction set architecture. This allows compiling larger chunks of code at once and is typically called "just in time compilation" in this context.

Gough [5] discusses application virtual machines and describes and compares two important examples: the Java Virtual Machine and Microsoft .NET. Both use an instruction set which is stack based and inherently object oriented. The main difference between the two is that Java is more oriented towards emulation, while .NET was designed with just in time compilation in mind. In practice both use just in time compilation on common operating systems running on IA-32. On other platforms only emulation may be available.

Because for application virtual machines the instruction set architecture is designed with virtualisation in mind, one can expect performance which is better than dynamic translation.

An application virtual machine can be very robust. The instruction set architecture is normally designed with security in mind. We also have the same kind of double security that we had in case of dynamic binary translation: we know that all code passes through a translator, so no code goes through unchecked.

Application virtual machines do not typically offer the level of isolation that normal virtual machines do. Still the VMM is in complete control of all interaction between the application and the operating system and it can selectively block or alter interaction with the system, if so configured by the user. This may be even more useful than total isolation in some situations.

The relationship with the operating system is inherently that of an unprivileged application.

An application virtual machine can only run guests using the specially defined instruction set. This means that portability is very bad.

It deserves mention that application virtual machines such as Java have very good host portability. One can execute an application on different operating systems and architectures without recompilation as long as one has the proper version of the VMM. Note, however, that for our comparison we are only looking at the ease of supporting different guest platforms, not supporting different host platforms.

Hardware supported

Recently Intel has introduced instruction set extensions which allow hardware supported virtual machines. This new technology is called Virtual Machine Extensions (VMX). An overview of this technology is presented by Uhlig et al [17] and complete documentation can be found in [7].

Intel introduces a distinction between VMX root and non-root modes. When the CPU runs in root mode, it can set up an environment for the non-root mode and switch. When it runs in non-root mode, sensitive instructions which do not trap and interrupts will cause a VM exit. This saves the complete CPU state

and restores the root mode situation. It is possible for the root to prevent certain instructions from causing a VM exit for performance reasons.

Both modes are just like the original situation, including four security rings. In effect the new chips therefore have 8 different security rings. This allows a guest operating system running in non-root mode to use ring 0, while the VMM stays in control. Therefore these operating systems can run entirely without modification. Figure 4 shows this approach.

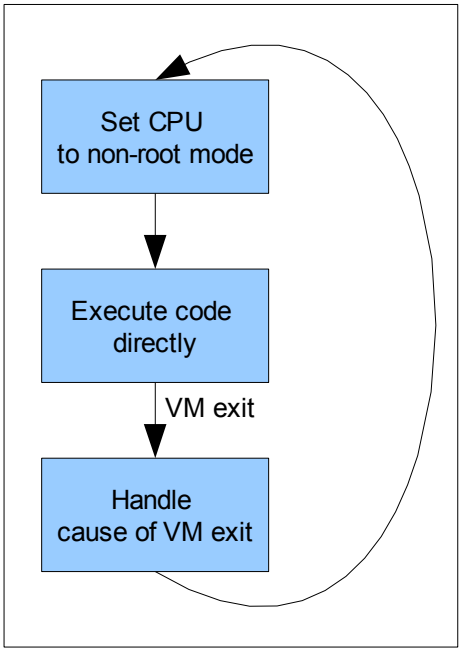

Figure 4: Hardware supported

Now we can consider ring 0 of the VMX root mode to be supervisor mode, the VMX non-root mode to be user mode, and a VM exit to be a trap. We see that the VMX extensions eliminate exactly the problem that made IA-32 not satisfy the conditions for the Popek and Goldberg theorem. Ring 1 through 3 of the VMX root mode still do not satisfy these conditions, but they now can be avoided in the VMM.

AMD has announced that it will introduce similar technology called Secure Virtual Machine Architecture. This extension is documented in [1]. To our knowledge no processors with this instruction set have been released yet.

The performance of hardware supported virtualisation depends heavily on the implementation of the chip. It is likely that performance is better than for binary translation, but for paravirtualisation it is hard to tell. This depends on which is more expensive: a hypercall or a VM exit. The state data which needs to be updated on a VM exit is stored in a 4096-byte block of data, so more memory needs to be updated than for a hypercall. It is not unthinkable, however, that the implementation highly optimises this specific case.

A VMM which uses the hardware solution is likely to be very simple, and as such likely to be robust. Security would only be compromised if a bug causes guest code to execute in root mode. Such a severe bug should be relatively unlikely in a simple, well-tested VMM. As such, we consider hardware virtualisation to be very robust.

VMX instructions trap outside of ring 0. This means that kernel-mode support is needed to be able to use them.

Portability between guest operating systems is very good. Like for dynamic binary translation, it does not really matter what code executes on the Virtual Machine. Unlike binary translation, porting in such a way that guests can run on a different architecture is not possible.

6 Results

We have evaluated several techniques which are used to implement current VMM software. We can now link these techniques to the applications for which they are most suitable.

Table 1 (next page) links possible applications to the criteria we used to evaluate the different techniques. This table summarises chapter 4. Table 2 links techniques to criteria, summarising chapter 5.

Servers

For servers four of the six techniques we considered are likely to be useful.

When the virtual servers only need a single operating system, and this operating system supports partitioning, this is likely to be the most suitable solution because of it's speed. In this case virtualisation comes for free, performance-wise.

Paravirtualisation or previrtualisation can be very useful for operating systems that do not support partitioning. It is also useful if one wants to run multiple guest operating systems. The speed provided is still very good. We currently do not know any real-world IA-32 implementations of previrtualisation. When these are available, benchmarks would be needed to determine if paravirtualisation really does provide a speed advantage over previrtualisation. If it does not, previrtualisation is preferred since it allows easier porting for new versions of operating systems.

In some cases the previously described solutions cannot be applied. One example is if one needs to run a virtual Windows server. This operating system does not support partitioning and a paravirtualised or

	Performance	Robustness	Cooperation with host OS	Portability
Server	Important	Important	No OS	Preferred
Debugging	Preferred	Preferred	Application	Important
Untrusted applications	Preferred	Important	Application	Preferred
Honeypot	Preferred	Important	Any	Important

Table 1: Applications linked with criteria

	Performance	Robustness	Cooperation with host OS	Portability
Binary translation	Moderate	Extra	Any	Very good
Paravirtualisation	Very good	Like OS	Any	Bad
Previrtualisation	Good	Like OS	Any	Moderate
OS-level partitioning	Very good	Like OS	Kernel	Bad
Application VM	Good	Extra	Application	Bad
Hardware supported	Depends	Extra	Kernel / no OS	Good

Table 2: Techniques linked with criteria

previrtualised version is not publicly available. In this case hardware supported virtualisation is likely to be the best choice.

If hardware supported virtualisation is turns out to be sufficiently fast, then it might be the best solution in many cases. This is because the robustness is likely to be higher than for the other cases.

Software development

For debugging, binary translation and hardware supported virtualisation are preferred because of their portability.

If the host operating system makes hardware virtualisation available to applications, then hardware supported virtualisation would probably be the best choice. It is likely to be faster than binary translation. If it is not available, binary translation is preferred, because it can be used from unprivileged applications.

Untrusted applications

When using untrusted applications, each of the technologies we mentioned can be used. Ease of use is probably the most important consideration here. For partitioning or hardware support this makes it desirable that the functionality is part of the kernel by default, rather than having to recompile the kernel or ask a root user to install kernel-mode drivers.

For one distributing applications that people may not trust, application virtual machines can be a good choice. They allow people to run these applications without fearing they will corrupt their system. This requires users to have the runtime for the application virtual machine installed. This is typically more likely than people having complete VMMs installed. Application virtual machines are typically easier to use than full system virtual machines, since to the user they appear no different than native applications.

Note that using application virtual machines to protect the computer requires that the environment be set up to sufficiently protect the computer against the guest application.

Honeypots

For honeypots, binary translation and hardware supported virtualisation are both good choices. Both of these are secure and portable.

If one wishes to test guests which use multiple architectures, then dynamic binary translation would be more useful than hardware supported virtualisation.

7 Conclusion

Originally the IA-32 architecture was designed without virtualisation in mind. Currently, many different approaches are used to circumvent this problem. Recently native support for virtual machines has been added to this architecture. However, the techniques previously used are still being actively developed. Currently even new

techniques, such as previrtualisation, are being researched and look promising.

We found that the new hardware support is likely to be very useful, but that it may not replace other techniques completely. Each of the techniques we discussed has advantages for specific areas of application. Therefore, selecting the proper approach can be hard as many factors should be considered.

8 References

[1] AMD, AMD64 Architecture Programmer's Manual Volume 2: System Programming, Publication 24593, Revision 3.11, December 2005, http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf

[2] A. Baratz. Virtual machine shootout: VMware vs. Virtual PC, *Ars Technica*, August 8, 2004, http://arstechnica.com/reviews/apps/vm.ars

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt and A. Warfield. Xen and the Art of Virtualization, *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ACM Press, New York, NY, USA, 2003, pp. 164-177

[4] Fabrice Bellard. QEMU, A Fast and Portable Dynamic Translator, *Proceedings of the USENIX 2005 Annual Technical Conference, FREENIX Track*, April 2005, pp. 41-46

[5] K. J. Gough. Stacking them up: a comparison of virtual machines, *Proceedings of the 6th Australasian conference on Computer systems architecture*, IEEE Computer Society, Washington, DC, USA, 2001, pp. 55-61

[6] Intel Corporation. IA-32 Intel Architecture Software Developer's Manual, Volume 3A: System Programming Guide, Order number 253668-019, March 2006, ftp://download.intel.com/design/Pentium4/manuals/25366819.pdf

[7] Intel Corporation. Intel Virtualization Technology Specification for the IA-32 Intel Architecture, C97063-002, April 2005, ftp://download.intel.com/design/Pentium4/manuals/25366819.pdf

[8] P. H. Kamp and R. N. M. Watson. Jails: Confining the omnipotent root, *Proceedings of the 2nd International System Administration and Networking Conference*, 2000, http://www.nluug.nl/events/sane2000/papers/kamp.pdf

[9] I. Kelly. Porting MINIX to Xen, May 8, 2006, http://choices.cs.uiuc.edu/cache/Report.pdf

[10] N. Kiyanclar. A Survey of Virtualization Techniques Focusing on Secure On-Demand Cluster Computing, ACM Computing Research Repository, Technical Report cs.OS/0511010, November 2, 2005 http://www.projects.ncassr.org/cluster-sec/papers/kiyanclarCorr05-virtualization.pdf

[11] J. LeVasseur, V. Uhlig, M. Chapma, P. Chubb, B. Leslie and G. Heiser. Pre-Virtualization: Slashing the Cost of Virtualization, Technical Report PA005520, National ICT Australia, October 2005, http://nicta.com.au/uploads/documents/PA005520_NICTA1.pdf

[12] S. Nanda and T. Chiueh. A Survey on Virtualization Technologies, Technical report TR-179, Department of Computer Science, State University of New York, February 2005, http://www.ecsl.cs.sunysb.edu/tr/TR179.pdf

[13] G. J. Popek and R. P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures, *Communications of the ACM*, ACM Press, New York, NY, USA, Volume 17, Issue 7, July 1974, pp. 412-421

[14] J. S. Robin and C. E. Irvine. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor, *Proceedings of the 9th USENIX Security Symposium*, August 2000, pp. 129-144

[15] R. Rose, Survey of System Virtualization Techniques, March 8, 2004, http://www.robertwrose.com/vita/rose-virtualization.pdf

[16] J. E. Smith and R. Nair. The architecture of virtual machines, *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, Volume 38, Issue 5, May 2005, pp. 32-38

[17] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung and L. Smith. Intel Virtualization Technology, *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, Volume 38, Issue 5, May 2005, pp. 48-56

[18] D. Ung and C. Cifuentes. Machine-adaptable dynamic binary translation, *ACM SIGPLAN Notices archive*, ACM Press New York, NY, USA, Volume 35, Issue 7, July 2000, pp. 41-51

[19] VMWare Inc. ESX Server 2 Security White Paper, 2004, http://www.vmware.com/pdf/esx2_security.pdf

[20] VMWare Inc. Virtualization Architectural Considerations and other evaluation criteria, 2005, http://www.vmware.com/pdf/virtualization_considerations.pdf

[21] Wikipedia contributors. Comparison of virtual machines, *Wikipedia, The Free Encyclopedia*, June 24, 2006, http://en.wikipedia.org/w/index.php?title=Comparis

on_of_virtual_machines&oldid=60326520

9 Products mentioned

Bochs	http://bochs.sourceforge.net/
BurnNT	http://l4ka.org/projects/virtualization/burnnt/
FreeBSD	http://www.freebsd.org/
Java	http://java.sun.com/
Linux V-Server	http://linux-vserver.org/
Marzipan	http://l4ka.org/projects/virtualization/resourcemon/
Microsoft .NET	http://www.microsoft.com/net/
Microsoft VirtualPC	http://www.microsoft.com/Windows/virtualpc/
Microsoft Virtual Server	http://www.microsoft.com/windowsserversystem/virtualserver/
OpenVZ	http://openvz.org/
Parallels Workstation	http://www.parallels.com/en/products/workstation/
Qemu	http://fabrice.bellard.free.fr/qemu/
Serenity Virtual Station	http://www.serenityvirtual.com/
Solaris Zones	http://www.opensolaris.org/os/community/zones/
User Mode Linux	http://user-mode-linux.sourceforge.net/
VMWare ESX Server	http://www.vmware.com/products/vi/esx/
VMWare Workstation	http://www.vmware.com/products/ws/
Xen	http://www.xensource.com/