Yet Another Operating System: Exokernel meets VMM

Navinkumar Adhe, Ayush Jain, and Dixit Paudel
University of California, Davis
{nadhe, jain, meondav}@ucdavis.edu

1 Introduction

Optimal hardware utilization and ownership cost reduction have been the bailiwick of operating systems research for decades [17]. Thus, it is not surprising that the idea of supporting multiple guest operating systems on the same machine simultaneously, continues to be of interest within academic and enterprise environments alike. At first Full-Emulation i.e. faithful reproduction of the underlying processor/peripherals and Binary-Emulation i.e. reimplementation of guest OS APIs within the host OS, may seem viable, but severe performance penalty and OS-specificity limit their wide acceptability respectively [6].

'Virtualization' facilitates resource sharing by presenting an additional 'thin software abstraction layer' for running several 'Virtual Machines' (VMs) over a common set of physical resources. A Virtual Machine Monitor (VMM) exposes an interface identical to the underlying hardware to all VMs, effectively recreating an entire machine capable of running any number of guest operating systems. Initially proposed in 1960s, this technique was resuscitated in the late 1990s by the success of VMware - the first commercial x86 Virtual Machine Monitor (VMM) by the developers of Disco [2] [6]. Finally, with the advent of hardware support (Intel VT-x & AMD-V), virtualization was mainstreamed.

Although virtualization is inherently complex, correct implementation yields several benefits which may be categorized as follows.

- **Extensibility**: Providing auxiliary services in a VM-centric architecture, or revamping the VM itself require lesser effort as compared to modifying the hardware.
- **Security**: The implemented services need only trust the Virtual-Machine-Monitor [4] which is far smaller and simpler to a full-fledged guest OS suffering from multiple vulnerabilities. Resultantly, the TCB (Trusted Computing Base) is substantially minimized.
- **Portability**: Separating the services from guest OSes, allows the former to run across different versions and vendors.
- **Manageability**: VMMs hold isolation to be a key goal [9], resulting in easier scrutiny of individual components.
- **Functionality**: VMs provide a familiar development environment to developers by supporting existing OSes and out-of-the-box applications. Moreover, the ability to clone, save, encrypt, move, restore, and rewind a VM state, can be used for secure logging, intrusion prevention/detection, and environment migration [5] [7] [11] [19].

However, Virtualization is not without its challenges.

1. **Performance**: The performance depends heavily on the degree of modification to a guest OS, necessary for running it on a VM [17]. VMM architectures such as Xen [1] and Denali [20] support 'slightly changed' guest OSes for incremental performance gains. VMware on the other hand, provides a fully virtualized environment [9].

   A significant portion of the time and effort in VMM research has gone towards moderating performance penalties due to an 'additional' layer [9]. Appreciable success has been achieved in this regard e.g. the Xen VMM has a very small overhead [9].

2. **Security**: A majority of the VMM implementations run atop large monolithic kernels [17]. Therefore, in comparison to traditional architectures, VM presents a bigger target area to malicious users due to the presence of guest OS, VMM, and a large TCB.

   Most prior well-known VMM implementations like Xen [1] & KVM [12] as well as commercial solutions like VMware ESXi & Microsoft Hyper-V have a TCB, which is upwards of 100 KLOC in size [17].

The 'Microkernel' theory evangelizes small yet extensible kernels which expose a minimal set of lower-level abstractions such as address spaces, threads, and inter-process communication [14]. Taking cues from 'Microkernel' design, the seminal work on NOVA [17] proposed to minimize the TCB (9 KLOC) for building a secure virtualization architecture. In fact, it has also been argued that 'VMMs are microkernels done right' [9].

The 'Exokernel' OS architecture has gone one step further by bridging the hardware $\Longleftrightarrow$ kernel gap even more. On top of that, by employing novel techniques like secure bindings, visible resource revocation, abort protocols, and libOSes (unprivileged/untrusted, extensible and replaceable library implementations called library operating systems), exokernels provide applications more control over machine resources [8]. Various independent research studies have shed light on the performance benefits of application-level resource management [10] [3] [13] [18]. Such a bare-bones design facilitates formal verification, provides greater flexibility, and provides better performance than monolithic as well as microkernel systems [8]. Aegis/ExOS – a prototype exokernel system, achieved a remarkable speedup (5 times faster as compared to state-of-the-art) in primitive kernel operations like exception handling, IPC, and control transfer [8].

We propose to build a paravirtual Virtual Machine Monitor over JOS – a pedagogical Operating System. Implemented in an 'exokernel-style', JOS is simple but provides all key OS functionalities – booting, memory management, file system, command shell and network drivers. In short, our work will make the following key contributions:

1. We show why an 'Exokernel' should be selected over traditional monolithic/microkernel systems for hosting a VMM. A minimal kernel results in a smaller TCB, greater flexibility, and faster performance.

2. We present the design and implementation of a paravirtual VMM. By exporting an interface to the VM which is similar (but non-identical) to the underlying hardware, we would achieve lower performance overhead.

2 Evidence

Figure 1: sysdiagram

3 Status

Towards implementing an exokernel based VMM, we started with a base JOS code which included:

- **Bootloader**
- **Memory Management**
- **User Environments**

The VMM module we will develop will run as user environment on the top of this base JOS base kernel.

We used the Stony Brook CSE591 [16] skeleton code as a starting point as it will guide us to building a VMM through lab exercises. We ported the bootloader and the memory management modules from MIT's 6.828 [15] JOS implementation.

The choice of the x-86 emulator in the MIT 6.828 course is qemu, but qemu doesn't support VT-x which is essential for building our VMM. We therefore switched to the bochs emulator as demanded by the Stony Brook CSE591 course. There were some configuration issues caused by a version mismatch, but we were able to overcome those and get the emulator to bootload JOS.

The base bootloader code provided by the CSE591 course handles multi booting, and exceeded the allowable bootfile size limit of 510 bytes. We changed it to boot a single kernel image, and contained the bootfile size within the allowable limit. We also successfully ported the memory management code from JOS.

Our next step is to port user environment code from MIT's JOS implementation. This will include a shell environment from which our VMM will be run. Our first step towards the VMM implementation will require us to implement a guest jos bootloader and kernel. Further steps require implementing extended page table support, vmlaunch, vmresume, and vmexit.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, volume 37, pages 164–177. ACM, 2003.

[2] E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems (TOCS)*, 15(4):412–447, 1997.

[3] P. Cao, E. W. Felten, and K. Li. Implementation and performance of application-controlled file caching. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, page 13. USENIX Association, 1994.

[4] P. M. Chen and B. D. Noble. When virtual is better than real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, HOTOS '01, pages 133–, Washington, DC, USA, 2001. IEEE Computer Society.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the*

2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2, pages 273–286. USENIX Association, 2005.

[6] C. L. Coffing. *An x86 protected mode virtual machine monitor for the mit exokernel*. PhD thesis, Massachusetts Institute of Technology, 1999.

[7] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review*, 36(SI):211–224, 2002.

[8] D. R. Engler, M. F. Kaashoek, et al. *Exokernel: An operating system architecture for application-level resource management*, volume 29. ACM, 1995.

[9] S. Hand, A. Warfield, K. Fraser, E. Kotsovinos, and D. Magenheimer. Are virtual machine monitors microkernels done right? In *Proceedings of the 10th Conference on Hot Topics in Operating Systems - Volume 10*, HOTOS'05, pages 1–1, Berkeley, CA, USA, 2005. USENIX Association.

[10] K. Harty and D. R. Cheriton. *Application-controlled physical memory using external page-cache management*, volume 27. ACM, 1992.

[11] S. T. King, G. W. Dunlap, and P. M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 1–1, 2005.

[12] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230, 2007.

[13] K. Krueger, D. Loftesness, A. Vahdat, and T. Anderson. *Tools for the development of application-specific virtual memory management*, volume 28. ACM, 1993.

[14] J. Liedtke. *On micro-kernel construction*, volume 29. ACM, 1995.

[15] MIT. 6.828: Operating system engineering, 2017.

[16] D. Porter. Paravirtual vmm, 2014.

[17] U. Steinberg and B. Kauer. Nova: A microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 209–222, New York, NY, USA, 2010. ACM.

[18] C. A. Thekkath and H. M. Levy. *Hardware and software support for efficient exception handling*, volume 29. ACM, 1994.

[19] A. Whitaker, R. S. Cox, S. D. Gribble, et al. Configuration debugging as search: Finding the needle in the haystack. In *Osdi*, volume 4, pages 6–6, 2004.

[20] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and performance in the denali isolation kernel. *ACM SIGOPS Operating Systems Review*, 36(SI):195–209, 2002.