Are Virtual-Machine Monitors Microkernels Done Right?

Gernot Heiser
National ICT Australia* and University of New South Wales
Sydney, Australia
gernot@nicta.com.au

Volkmar Uhlig
IBM T.J. Watson Research Center, Yorktown Heights, NY
vuhlig@us.ibm.com

Joshua LeVasseur
University of Karlsruhe, Germany
jtl@ira.uka.de

Abstract

A paper by Hand *et al.* at the recent HotOS workshop re-examined microkernels and contrasted them to virtual-machine monitors (VMMs). It found that the two kinds of systems share architectural commonalities but also have a number of technical differences which the paper examined. It concluded that VMMs are a special case of microkernels, "microkernels done right".

A closer examination of that paper shows that it contains a number of statements which are poorly justified or even refuted by the literature. While we believe that it is indeed timely to reexamine the merits and issues of microkernels, such an examination needs to be based on facts.

1 Introduction

At the HotOS workshop in June this year, Hand and coauthors presented a paper [HWF$^+$05] titled "Are virtual machine monitors microkernels done right?" The paper compares and contrasts microkernels and virtual-machine monitors (VMMs) as platforms for systems design and implementation. While identifying architectural similarities, it examines the difference in the approaches, and concludes that VMMs are one specific point in the microkernel design space, the "right" one. Unstated but implied is the assertion that VMMs such as Xen [BDF$^+$03] are the (to date) only "right" approach to building microkernels.

Taking a closer look at the main assertions made by Hand *et al*, we find that they are hard to justify, or even squarely at odds with the literature. While we think that reexamining the merits and failures of microkernels is a potentially valuable exercise, we strongly believe that such a discussion must be performed in accordance with established scientific principles, and most of all, be grounded in facts. As a contribution to an informed discussion, we examine the assertions made by Hand *et al* in the light of the public record.

*National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs.

2 Background

Before addressing the specific assertions made in Hand *et al*'s paper, we provide some (we hope) useful background for the discussion.

2.1 History Revisited

Microkernels and virtual machine monitors both have a long history, dating back to the early 1970's [BH70, Gol74]. Given that there are significant similarities, it is useful to look at a somewhat narrow definition of both.

Goldberg [Gol74] defines a virtual machine monitor as "*[...] software which transforms the single machine interface into the illusion of many. Each of these interfaces (virtual machines) is an efficient replica of the original computer system, complete with all of the processor instructions [...]*".

Liedtke [Lie96] describes the microkernel approach as "*... to minimize the kernel and to implement whatever possible outside of the kernel*".

Both definitions appear sufficiently distinct to raise the question of how much commonality there can be. Examining the *goals* of the two approaches shows that there is more similarity than is evident from the definitions: Goldberg lists software reliability, data security, alternative system APIs, and improved and new mechanisms as benefits; Liedtke lists flexibility and extensibility, fault isolation, maintainability, and restricted interdependencies.

It seems that while VMMs and microkernels share a common set of goals, they take a different approach towards the solution. Yet both approaches consider minimality important. While for microkernels it is a key objective, Goldberg reports it as a result of the system structure: "*A key principle in the analysis of software reliability is that the VMM is likely to be correct—i.e., the probability of failure is near zero. This assumption is reasonable because the VMM is likely to be a very small program [...]*".

2.2 Core primitives

In the effort to minimise kernel functionality, microkernels offer a minimal set of abstractions with a central primitive for extensibility: inter-process communication (IPC). In a microkernel, IPC serves three primary purposes:

1. IPC is the mechanism for kernel-controlled change of execution flow between protection domains;
2. IPC is the mechanism for kernel-controlled data transfer between protection domains;
3. IPC is the mechanism for resource delegation between protection domains that requires mutual agreement between multiple (potentially distrusting) parties.

Combining these three orthogonal operations into a single primitive reduces the number of security mechanisms, reduces the code complexity, and reduces the code size. A smaller code base reduces the number of errors in the privileged kernel, as well as reducing the cache footprint. An obvious key requirement for any microkernel is thus a low-overhead IPC primitive. All other operations that require a combination of the three mechanisms can be implemented via the single IPC primitive.

VMMs in comparison, closely resemble processor hardware and offer a rich variety of primitives. Each primitive requires a dedicated set of security mechanisms, resources, and kernel code. A comprehensive list is beyond the scope of this paper, thus we only list the common subset of primitives that can be found in most VMMs:

1. synchronous switch of protection domain from guest user to guest kernel;
2. synchronous switch of protection domain from guest kernel to guest user;
3. asynchronous communication channels across domains (virtual machine (VM) to virtual machine);
4. resource allocation per VM via VMM hypercall interface;
5. resource allocation within the VM (e.g., via hardware page-table virtualisation);
6. resource re-allocation (e.g., via page flipping);
7. page-fault and exception handling via exception virtualisation;

8. asynchronous event notification across domains via virtual-interrupt signalling mechanism;

9. hardware interrupt notification via virtualized interrupt controller;

10. a set of common devices, such as NIC and disk.

The interfaces provided by the VMM have an intriguing benefit for an important class of highly complex software: *existing* operating systems. Available operating systems already program to the interface provided by the hardware and resembled by the VMM. Thus existing operating systems require no or only minimal changes to run on a VMM, whereas adaptation to the microkernel primitives often requires significant modifications. However, this benefit is being eroded by the increasing divergence of VMMs from *pure* virtualisation (faithful representation of the underlying hardware) to paravirtualisation (representation of modified hardware that lends itself better to efficient support of legacy OSen).

The diversity of interfaces also leads to structural compromises, such as *centralized super-VMs* that combine and colocate significant critical system functionality. Such a structure potentially decreases overall reliability and poses the risk of a single point of failure. This problem becomes even more inherent if this super-VM runs a legacy operating system and thus re-introduces a large number of software bugs [CYC$^+$01].

For extensions that are *not an existing operating system*, the VMM's interfaces significantly increase the complexity of software design. As, per definition, a VMM presents an interface that is close to the underlying architecture, software developed for one VMM is *inherently unportable* across architectures. In contrast, a microkernel abstracts and hides the peculiarities of the hardware platform behind its common set of abstractions. For example, software that is written for an L4 microkernel [Lie95] naturally runs on nine different processor platforms, from embedded devices such as ARM, to desktop and small servers such as x86, up to large multiprocessor PowerPC and Itanium machines. Hence, it is possible to leverage and reuse system components across a wide variety of hardware platforms, thereby minimising porting and maintenance overhead.

3 Architectural Lessons

Now we reexamine the architectural lessons presented by Hand *et al* in detail, following the headings of their paper, and clarify the role of microkernels.

3.1 Avoid Liability Inversion

The paper states that moving system services out of the kernel *relaxes the dependability boundaries within the system*. Applications and even the kernel depend on user-level code. This situation is called *liability inversion* and an example from Mach [YTR$^+$87] is used to argue that "inelegant" mechanisms are required to ensure correct system operation as a consequence of the "kernel abdicating its liability". It is further argued that one of the principal design guidelines of Xen were to avoid liability inversion.

At the workshop, Butler Lampson was quick to point out that this liability inversion is in fact an issue in Xen as well. An example for this is actually given in another paper at the same workshop by some of the same authors: the Parallax storage system [WRF$^+$05] essentially uses external pagers to provide file service. While that paper argues that the design avoids liability inversion, Parallax is "providing a critical system service for a set of VMMs". This is exactly what a user-level server does in a microkernel-based system. The argument is made that a failure of the Parallax server only affects its clients — exactly the same situation as if a server fails in an L4-based system. Hence, we fail to see the difference between a VMM and a microkernel in this respect.

Possibly this apparent conflict is a result of a lack of understanding of microkernels (even though this has been thoroughly explained in the literature [Lie96]). The confusion might in fact be the result of an invalid generalisation of a specific example (a particular design fault of Mach) onto a whole class of systems (microkernels).

3.2 Make IPC performance irrelevant

Here Hand *et al*. argue that, while microkernel designers have spent considerable effort on optimising inter-process communication (IPC) mechanisms, this is irrelevant as it is "not a critical design concern in the construction of high-performance VMMs."

They further argue that IPC between virtual machines is much less frequent and thus not performance critical, as a consequence of the VMM scheduling and protecting complete operating systems.

This is an interesting line of argument, as it is at odds with the reality of Xen-based systems in at least two respects:

- Xen uses a separate virtual machine (called Dom_0) to encapsulate legacy device drivers [FHN+04]. Hence, any I/O operation implies at least one round-trip communication between the guest VM and Dom_0. The authors call this a "simple asynchronous unidirectional event mechanism" — it is nothing else than a form of asynchronous IPC.

  And performance-critical it is indeed. A recent paper [CG05] examines the CPU overhead of Dom_0 drivers under high load, and finds that the CPU load generated by Dom_0 accounts for almost all of the CPU load of the system under test! They also find that the Dom_0 CPU time is proportional to the number of Xen's page-flipping operations, that is, message transfers, irrespective of the message size. The clear implication of this data is that IPC costs dominate the driver overhead in Xen systems under high I/O load.

- While it is true that Xen schedules complete operating systems, this does not mean that there is no other interaction with the VMM. In fact, each guest-application exception and system call causes a trap into the VMM, which then invokes corresponding functionality in the guest OS. This is nothing but an IPC operation between the guest application and the guest OS.

  Xen provides a shortcut based on x86's trap gates that avoids invoking the VMM on guest systemcalls. However, this shortcut is specifically targeted and limited to Linux's `int 0x80` system-call variant and restricts the use of segments. Protection can only be preserved if all active segment configurations explicitly exclude the VMM kernel. Since x86's trap mechanism only reloads two of the six segment selectors, the solution is limited; Linux's latest glibc violates the assumption and renders the shortcut useless.

A Xen-based system performs essentially the same number of IPC operations as a comparable microkernel-based system (such as L^{4}Linux [HHL+97]).

3.3 Treat the OS as a component

Under this heading, Hand *et al.* argue that a benefit of VMMs is that they are designed to run complete legacy systems, with familiar programming and development environments, and lending themselves to extensions such as Parallax. The (unstated) implication of such statements has to be that microkernels are somehow not suitable for such use.

This is a really surprising notion, as L4 has demonstrated many years ago that it is perfectly suitable as a VMM supporting a paravirtualised Linux system with excellent performance [HHL+97], and the Dresden DROPS system [HBB+98] is built specifically on extending a paravirtualised Linux system running on a microkernel with real-time services and is in industrial use.

Again, we fail to see the claimed "significant difference" between VMMs and microkernels.

4 Conclusions

In summary, the "important differences" between microkernels and VMMs identified by Hand *et al.* do not seem to hold up to scrutiny. As a consequence, their conclusion "that VMMs are microkernels done right" cannot be inferred from the arguments they presented. Yet, the observation, also made by others [HPHS04], that VMMs and microkernels are closely related, deserves further attention. We believe that a systematic and objective examination of the similarities and differences of microkernels and VMMs is still outstanding, and would make a valuable contribution to OS theory and practice.

References

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfi eld. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on OS Principles*, pages 164–177, Bolton Landing, NY, USA, October 2003.

[BH70] Per Brinch Hansen. The nucleus of a multiprogramming operating system. *Communications of the ACM*, 13:238–250, 1970.

[CG05] Ludmila Cherkasova and Rob Gardner. Measuring CPU overhead ofr I/O processing in the Xen virtual machine monitor. In *Proceedings of the 2005 USENIX Technical Conference*, pages 387–390, Annaheim, CA, USA, April 2005.

[CYC+01] Andy Chou, Jun-Feng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating systems errors. In *Proceedings of the 18th ACM Symposium on OS Principles*, pages 73–88, Lake Louise, Alta, Canada, October 2001.

[FHN+04] Keir Fraser, Steven Hand, Rolf Neugebauer, Ian Pratt, Andrew Warfi eld, and Mark Williamson. Reconstructing I/O. Technical Report UCAM-CL-TR-596, University of Cambridge, August 2004.

[Gol74] Robert P. Goldberg. Survey of virtual machine research. *IEEE Computer*, 7(6):34–45, June 1974.

[HBB+98] Hermann Härtig, Robert Baumgartl, Martin Borriss, Claude-Joachim Hamann, Michael Hohmuth, Frank Mehnert, Lars Reuther, Sebastian Schnberg, and Jean Wolter. Drops — OS support for distributed multimedia applications. In *Proceedings of the 8th SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[HHL+97] Hermann Härtig, Michael Hohmuth, Jochen Liedtke, Sebastian Schönberg, and Jean Wolter. The performance of μ-kernel-based systems. In *Proceedings of the 16th ACM Symposium on OS Principles*, pages 66–77, St. Malo, France, October 1997.

[HPHS04] Michael Hohmuth, Michael Peter, Hermann Härtig, and Jonathan S. Shapiro. Reducing TCB size by using untrusted components —small kernels versus virtual-machine monitors. In *Proceedings of the 11th SIGOPS European Workshop*, Leuven, Belgium, September 2004.

[HWF+05] Steven Hand, Andrew Warfi eld, Keir Fraser, Evangelos Kottsovinos, and Dan Magenheimer. Are virtual machine monitors microkernels done right? In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems*, Sante Fe, NM, USA, June 2005. USENIX.

[Lie95] Jochen Liedtke. Improved address-space switching on Pentium processors by transparently multiplexing user address spaces. Technical Report 933, GMD SET-RS, Schloß Birlinghoven, 53754 Sankt Augustin, Germany, November 1995.

[Lie96] Jochen Liedtke. Towards real microkernels. *Communications of the ACM*, 39(9):70–77, September 1996.

[WRF+05] Andrew Warfi eld, Russ Ross, Keir Fraser, Christian Limpach, and Steven Hand. Parallax: Managing storage for a million machines. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems*, Santa Fe, NM, USA, June 2005. USENIX.

[YTR+87] Michael Young, Avadis Tevanian, Richard Rashid, David Golub, Jeffrey Eppinger, Jonathan Chew, William Bolosky, David Black, and Robert Baron. The duality of memory and communication in the implementation of a multiprocessor operating system. In *Proceedings of the 11th ACM Symposium on OS Principles*, pages 63–76, 1987.