On Microkernels and Virtual Machines: The Legacy of Mach

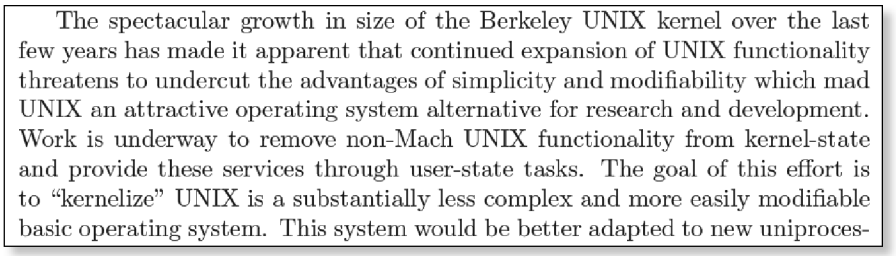
Mahadev Satyanarayanan

School of Computer Science
Carnegie Mellon University

During the early years of Mach, anyone who talked to Rick Rashid soon encountered two ideas that excited him. One was the idea that operating systems had grown too large and complex, and that the time had come to simplify and shrink. The second was the idea that different operating systems offered the same core functionality underneath a lot of superficial diversity, and it should therefore be possible to dynamically choose which guise to interact with. Although these two powerful ideas are interrelated in the Mach implementation, it is useful to think of them as distinct concepts from a historical perspective. In the roughly two decades since active research on Mach ended, how have these concepts fared in the marketplace of ideas? What is their intellectual legacy for OS research? Does anyone still care about them? We examine these issues in this brief note.

2. Simplify and Shrink

Figure 1 is an excerpt from the earliest publication I found that mentions the microkernel concept in Mach. By the mid-1980s, the dominant operating system used by academic researchers (Unix) had grown from a small, tightly organized and easily understood code base into a large and unwieldy body of code that was difficult to understand, debug and maintain. Some of the growth was due to introduction of new functionality such as support for virtual memory and support for networking in the form of the socket interface. Also contributing to size and complexity was the need for portability of the code across many different hardware platforms. The net impact on complexity was multiplicative, not just additive. The Mach



The spectacular growth in size of the Berkeley UNIX kernel over the last few years has made it apparent that continued expansion of UNIX functionality threatens to undercut the advantages of simplicity and modifiability which mad UNIX an attractive operating system alternative for research and development. Work is underway to remove non-Mach UNIX functionality from kernel-state and provide these services through user-state tasks. The goal of this effort is to "kernelize" UNIX is a substantially less complex and more easily modifiable basic operating system. This system would be better adapted to new uniproces-

FIGURE 1: Earliest Mach Reference (1986) to the Microkernel Concept [1]

manifesto proposed a fundamental restructuring of an operating system into a small, simple core combined with mechanisms to easily extend that core.

The importance of extensibility in operating system design had been identified as early as 1970 by Per Brinch Hansen [9]. His design rationale for the RC 4000 operating system stated: "... the main problem in the design of a multiprogramming system is not to define functions that satisfy specific operating needs, but rather to supply a system nucleus that can be extended with new operating systems in an orderly manner." This fundamental insight into operating system structure was lost in the explosive growth of computing hardware and software in the 1970s and early 1980s.

Mach's rediscovery of this attribute came at an opportune time. Its push towards simplicity and extensibility coincided with the emergence of workstation and server hardware with multiple processors. The monolithic structure of Unix, especially its coarse-grain concurrency control around kernel entry and exit, muted the performance benefits of using multiple processors. Introducing mechanisms for finer-grain concurrency control within the kernel was a difficult task, most likely requiring an entire rewrite of the kernel. Such a rewrite was an opportune time to make fundamental changes to the structure of Unix.

By the late 1980s, Mach software releases of growing portability and efficiency were available for a diversity of hardware platforms, some with multiple processors. Initial releases (Mach 2.5 and 2.6) retained the monolithic structure of BSD Unix, but reengineered the design of the virtual memory system, kernel locking mechanisms, and related components. Later in the evolution, starting with Mach 3.0, began the process of fundamental restructuring into a true microkernel architecture.

Accompanying the successful software releases were a number of scholarly publications that reported on specific aspects of Mach. The combination of working code and evangelism created enthusiastic support for the Mach vision in industry and academia. A consortium of companies (led by Digital Equipment Corporation and initially including Apollo Computer, Groupe Bull, Hewlett-Packard, IBM, Nixdorf Computer, and Siemens AG) formed the "Open Software Foundation (OSF)" and embraced Mach as the foundation of their operating system efforts. The USENIX Association organized multiple Mach-centered conferences in the early 1990s: the USENIX Mach Symposium at Monterey, CA in November 1991, the USENIX

Mach III Symposium at Santa Fe, NM in April 1993, and the USENIX Microkernels and Other Kernel Architectures Symposium in San Diego, CA in September 1993. An indication of Mach's mindshare is suggested by the title of the keynote talk at the USENIX Experiences with Distributed and Multiprocessor Systems symposium in San Diego, CA in September 1993. At this conference the keynote address by Professor Larry Peterson was titled, "Is There Life After Microkernels?" Mach concepts clearly dominated the discourse of the operating systems research community in the early 1990s.

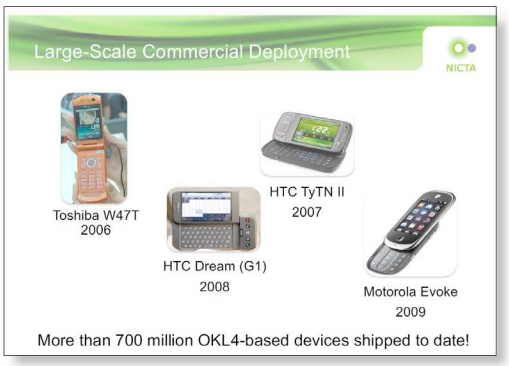


FIGURE 2: Commercial Use of L4 Microkernel (Source: Gernot Heiser)

Yet, by the end of the 1990s this momentum had dissipated. Digital Equipment was no longer an independent company, having been absorbed into Compaq and Intel. The Open Software Foundation was a spent force. The Mach microkernel no longer dominated efforts in commercial operating systems. It was Linux, with a traditional monolithic kernel structure, that was rapidly gaining a foothold in industry. There are ongoing debates about the cause of this turn of events, but establishing a specific causal chain is difficult. The Mach microkernel vision of simplicity and extensibility continued to inspire academic research, as evidenced by work such as SPIN [3] and Exokernel [5]. However, it was no longer seen as a competitive advantage in the marketplace.

Fast forward to 2012. It would seem inevitable that after a further 10–15 years, there would be little left of microkernels in terms of active influence on present-day operating systems. Fortunately, this is not true! The key person who sustained the microkernel vision after Mach's influence faded was Jochen Liedtke (1953-2001). His efforts were later amplified and extended by others such as Hermann Härtig and Gernot Heiser. In a series of papers in the mid-1990s [13, 14, 10], Liedtke and his associates explored microkernel architectures from first principles in the context of the L3 and L4 operating systems. They established that many of the performance challenges of microkernels could be overcome through careful design and attention to implementation details. Today, Heiser's team at the University of New South Wales and NICTA is the center of academic and commercial efforts surrounding the L4 microkernel [12]. Figure 2 (reproduced here by permission of Gernot Heiser) illustrates the extensive use of L4 in embedded systems.

3. Multiple Personalities

Mach's second big idea was the ability to support multiple operating system personalities on a common base. Figure 3 shows the earliest published reference that I could find for this concept. This new capability was especially attractive to companies that were being forced

processes. The reason this implementation strategy is so attractive for open systems, is that it can allow more than one operating system environment to be supported on the same machine, on the same kernel, at the same time. Systems such as UNIX or OS/2 could potentially co-exist in their native form. The kernel becomes a kind of universal "socket" into which more than one operating system environment can be plugged, insulating that software from the hardware itself and greatly simplifying its design and maintenance.

FIGURE 3: Earliest Mach Reference (1989) to the Concept of Multiple Personalities [17]

to make difficult choices between operating systems on their new hardware. For example, on its new PowerPC hardware platform IBM had to choose between supporting Microsoft's Windows 3.1, its own OS/2, or its AIX flavor of Unix. On its new Alpha hardware platform, Digital Equipment faced a similar decision between VMS and OSF/1. Supporting multiple operating systems on the same hardware was theoretically possible, but it would have been very expensive in terms of software development and maintenance. Mach's ability to support multiple personalities on a common base appeared to be a cost-effective approach to solving this problem.

In 1992, a development team from Digital Equipment described a proof-of-concept implementation of VMS on Mach 3.0 [20]. Figure 4, reproduced from that work, illustrates the layering of VMS on Mach. Their work provided independent confirmation that multiple OS personalities could be supported on the Mach microkernel. At the same time, it exposed certain limitations. For example, it proved impossible to accurately emulate VMS scheduling policies using Mach. As another example, it was not possible to emulate VMS' strong isolation of kernel resource usage by different users. Preliminary measurements also suggested that layering VMS on Mach resulted in unacceptable performance overhead. Due to these technical concerns and other nontechnical considerations, Digital Equipment did not follow through with a production quality implementation of VMS on Mach.

A much bigger bet was made by IBM. Rather than a tentative exploration, it wholeheartedly embraced the concept of multiple personalities on a common base. In 1991, IBM initiated a major development effort called Workplace OS that would unify its diverse menagerie of operating systems that included OS/2, PC DOS, AIX and OS/400. Fleisch et al [6, 7] and Rawson [18] give candid retrospective accounts of this effort. Figure 5 from Rawson's account illustrates the structure of the system. At its peak, over 2000 software developers spanning multiple IBM divisions were engaged in implementing Workplace OS. Starting from the Mach 3.0 kernel, there was initially rapid progress. OS/2, DOS, and Unix personalities on PowerPC and Intel hardware were demonstrated by IBM at COMDEX in Fall 1992. But a year later, in Fall 1993, IBM announced that AIX would remain a separate system outside Workplace OS. Version 1 of the IBM Microkernel was released in Fall 1995 and adopted

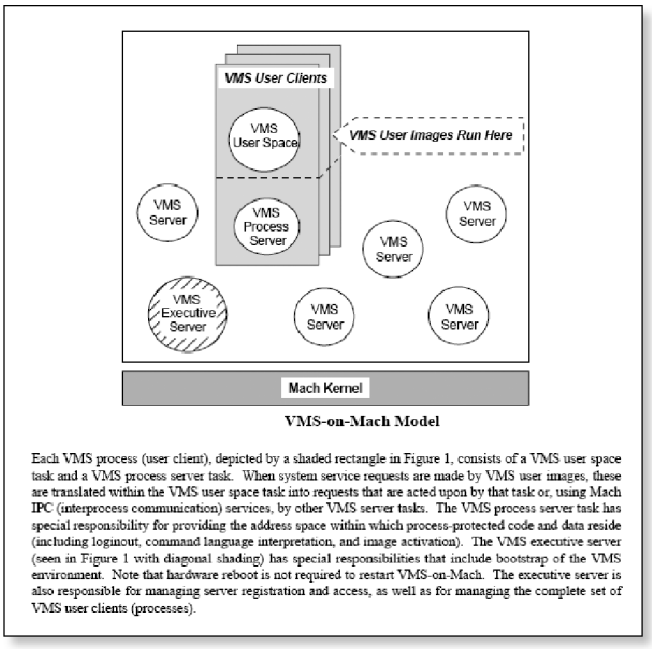


FIGURE 4: Digital Equipment's VMS on Mach Microkernel Prototype (Source: Wiecek et al [20])

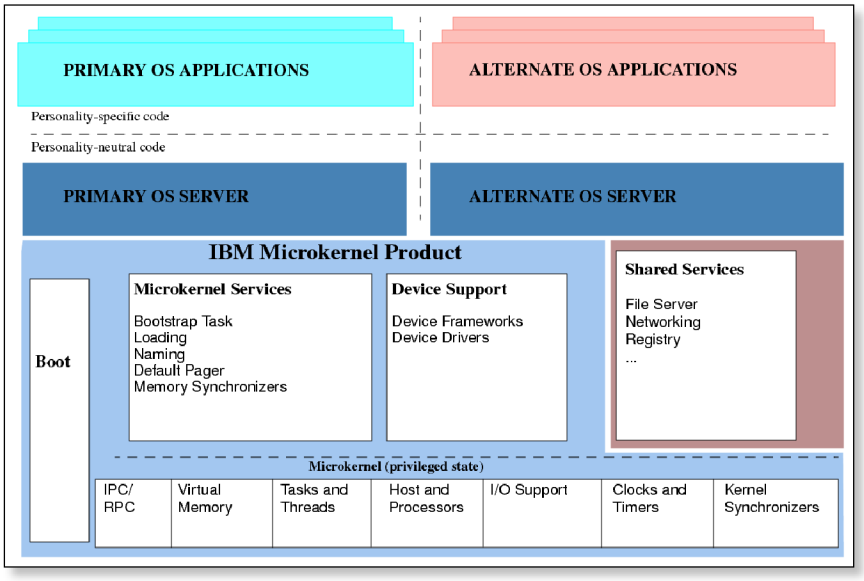


FIGURE 5: Structure of IBM's Workplace OS (Source: Rawson [18])

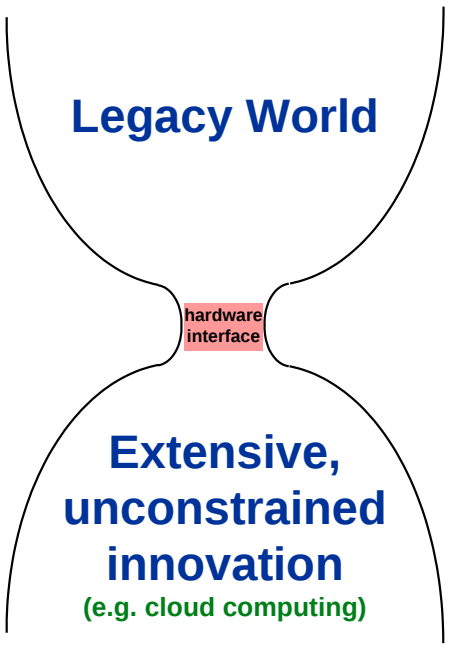


FIGURE 6: Legacy-compatible VM Ecosystem

by a few companies and universities. But a few months later, in late 1995, the entire Workplace OS project was cancelled.

As with the microkernel concept, the 1990s ended with the apparent repudiation of Mach's vision. Yet, a dozen years later, we see exactly this vision transformed into everyday reality: various flavors of Windows and Linux coexisting on a common base in cloud computing. Each of these diverse environments is encapsulated within a virtual machines (VM), and the "common base" is a virtual machine monitor (VMM) or hypervisor. It is interesting to speculate on why this approach has succeeded almost effortlessly, when the same end goal proved so hard to attain earlier on a different foundation.

VMs are an even older idea than microkernels or multiple personalities. They were not created for aesthetic or philosophical reasons, but as a very pragmatic solution to a real problem. VMs were invented by IBM in the mid-1960s as a timesharing approach that enabled concurrent, cleanly isolated system software development on a single mainframe by multiple programmers. Since mainframe hardware of that era was expensive, VMs were a cost-effective approach to enhancing programmer productivity by providing a private "mainframe" for each developer rather than reserving dedicated time on real hardware. Accuracy of hardware emulation was paramount because the system software developed on a VM was intended for use in a mainframe operating system. That software had to work with negligible changes on the real hardware. The techniques for efficient and accurate hardware virtualization that were developed in this era have proved to be of lasting value.

Figure 6 illustrates the value proposition of the VM abstraction today. A large legacy world of software, including operating system software, represents a substantial intellectual and financial investment that has been already been made on existing computer hardware. This legacy world can be completely closed source: there is no requirement for availability of source code, nor a requirement for recompilation or relinking. All that is needed is standard software installation, just as on real hardware. Beneath the implementation of the emulated hardware interface, there can be extensive innovation along many dimensions. This innovation is totally isolated from the idiosyncrasies of the legacy world above. Certain specific attributes of the VM abstraction accounts for its longevity and success. The hourglass shape in Figure 6 could depict the interface specification of any abstraction (thin waist), applications dependent on this interface (above), and implementations of this interface (below). More specifically,

it could represent an execution engine such as the Java Virtual Machine (JVM) [15] or the Dalvik Virtual Machine for the Android platform [21]. While these alternatives (collectively referred to as software virtualization) have successful niches, they do not approach the VM abstraction (also referred to as hardware virtualization) in terms of longevity, widespread usage and real-world impact. Why is hardware virtualization so much more successful? First, the interface presented by the VM abstraction is compatible with legacy operating systems and their valuable ecosystems of applications. The ability to sustain these ecosystems without code modifications is a powerful advantage of VMs. The ecosystems supported by software virtualization tend to be much smaller. For example, a JVM is only valuable in supporting applications written in specific languages such as Java. In contrast, a VM is language agnostic and OS agnostic. In fact, a JVM can be part of the ecosystem supported by a VM. Hardware virtualization can thus subsume software virtualization. Second, a VM interface is narrow and stable relative to typical software interfaces. In combination, these two attributes ensure adequate return on investments in the layer below. By definition, a narrow interface imposes less constraints on the layer below and thus provides greater freedom for innovation. The stability of a VM interface arises from the fact that the hardware it emulates itself evolves very slowly and almost always in an upward compatible manner. In contrast, the pliability of software results in more rapid evolution and obsolescence of interfaces. Keeping up with these changes requires high maintenance effort. Pliability also leads to widening of narrow interfaces over time, because it is difficult to resist the temptation to extend an interface for the benefit of a key application. Over time, the burden of sustaining a wide interface constrains innovation below the interface.

The importance of the narrowness and stability of an interface can be seen in the contrasting fortunes of process migration and VM migration, which are essentially the same concept applied at different levels of abstraction. Process migration is an operating system capability that allows a running process to be paused, relocated to another machine, and continued there. It has been a research focus of many experimental operating systems built in the past 20 years. Examples include Demos [16], V [19], Mach [23], Sprite [4], Charlotte [2], and Condor [22]. These independent validation efforts have shown beyond reasonable doubt that process migration can indeed be implemented with acceptable efficiency. Yet, in spite of its research popularity, no operating system in widespread use today (proprietary or open source) supports process migration as a standard facility. The reason for this paradox is that a typical implementation of process migration involves such a wide interface that it is easily rendered incompatible by a modest external change such as an operating system upgrade. Long-term maintenance of machines with support for process migration involves too much effort relative to the benefits it provides. The same concept of pausing an executing entity, relocating it, and resuming execution can be applied to an entire VM rather than a single process. This is VM migration, a capability that is in widespread use in cloud computing systems today. Guest OS changes do not affect an implementation of VM migration. This insulation from change, embodied in the narrow and stable VM interface, is crucial to the real-world success of VM migration.

The relationship between microkernels and VMMs has been the subject of debate in the research literature. In 2005, Hand et al published a position paper entitled, "Are Virtual Machine Monitors Microkernels Done Right?" [8]. In response, Heiser et al offered a rebuttal [11]. The debate continues!

4. Closing Thoughts

More than a quarter of a century has passed since Rick first envisioned Mach. Yet, looking at Figures 1 and 3, one is struck by how relevant those ideas are even today. The attributes of simplicity and extensibility continue to inspire operating system research, even as commercial operating systems grow ever more bloated in size and complexity. From a user's viewpoint, the ability to seamlessly and effortlessly switch between operating system worlds continues to be valuable. Mach pointed the way, and it remains the intellectual forerunner of so many aspects of today's systems.

References

[1] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and, Young, M. Mach: A New Kernel Foundation for UNIX Development. In *Proceedings of the Summer USENIX Conference* (1986), pp. 93–112.

[2] Artsy, Y., and Finkel, R. Designing a Process Migration Facility: The Charlotte Experience. *IEEE Computer 22*, 9 (1989).

[3] Bershad, B. N., Savage, S., Pardyak, P., Sirer, E. G., Fiuczynski, M. E., Becker, D., Chambers, C., and, Eggers, S. Extensibility Safety and Performance in the SPIN Operating System. In *Proceedings of the Fifteenth ACM symposium on Operating Systems Principles* (Copper Mountain, CO, 1995).

[4] Douglis, F., and Ousterhout, J. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software Practice and Experience 21*, 8 (1991).

[5] Engler, D. R., Kaashoek, M. F., and, O'toole, Jr., J. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (Copper Mountain, CO, 1995).

[6] Fleisch, B. D. The Failure of Personalities to Generalize. In *Proceedings of the 6th IEEE Workshop on Hot Topics in Operating Systems* (Cape Cod, MA 1997).

[7] Fleisch, B. D., Co, M. A. A., and, Tan, C. Workplace Microkernel and OS: A Case Study. Tech. Rep. CS984, Department of Computer Science, University of California at Riverside, April 1998.

[8] Hand, S., Warfield, A., Fraser, K., Kotsovi Nos, E., and, Magenheimer, D. Are Virtual Machine Monitors Microkernels Done Right? In *Proceedings of the 10th conference on Hot Topics in Operating Systems* (Santa Fe, NM, 2005).

[9] Hansen, P. B. The Nucleus of a Multiprogramming System. *Communications of the ACM 13*, 4 (April 1970), 238–250.

[10] Hārtig, H., Hohmuth, M., Liedtke, J., Wolter, J., and, Schōnberg, S. The Performance of µkernel-based Systems. In *Proceedings of the sixteenth ACM symposium on Operating Systems Principles* (Saint Malo, France, 1997).

[11] Heiser, G., Uhlig, V., and, Levasseur, J. Are Virtual Machine Monitors Microkernels Done Right? *SIGOPS Operating Systems Review 40*, 1 (January2006).

[12] Klein, G., Elphinstone, K., Heiser, G., Andron Ick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and, Winwood, S. seL4: Formal Verification of an OS Kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles* (October 2009).

[13] Liedtke, J. Improving IPC by Kernel Design. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles* (Asheville, NC, 1993).

[14] Liedtke, J. On Microkernel Construction. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (Copper Mountain, CO, 1995).

[15] Lindholm, T., and Yellin, F. The Java Virtual Machine Specification (2nd Edition). Prentice Hall, 1999.

[16] Powell, M., and Miller, B. Process Migration in DEMOS/MP. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles* (Bretton Woods, NH, Oct. 1983).

[17] Rashid, R., Baron, R., Forin, A., Golub, D., Jones, M., Julin, D., Orr, D., and Sanzi, R. Mach: A Foundation for Open Systems. In *Proceedings of the Second Workshop on Workstation Operating Systems* (1989).

[18] Rawson III, F. L. Experience With the Development of a Microkernel-Based, Multiserver Operating System. In *Proceedings of the 6th IEEE Workshop on Hot Topics in Operating Systems* (Cape Cod, MA, May 1997).

[19] Theimer, M., Lantz, K., and Cheriton, D. Pre-emptable Remote Execution Facilities for the VSystem. In *Proceedings of the 10th Symposium on Operating System Principles* (Orcas Island, WA, December 1985).

[20] Wiecek, C. A., Kaler, C. G., Fiorelli, S., William C. Davenport, J., and, Chen, R. C. A Model and Prototype of VMS Using the Mach 3.0 Kernel. In *Proceedings of the USENIX Workshop on Microkernels and Other Kernel Architectures* (Seattle, WA, April 1992).

[21] Wikipedia. Dalvik (software). http://en.wikipedia.org/wiki/Dalvik_(software), September 2011.

[22] Zandy, V., Miller, B., and Livny, M. Process Hijacking. In *8th International Symposium on High Performance Distributed Computing* (Redondo Beach, CA, August 1999).

[23] Zayas, E. Attacking the Process Migration Bottleneck. In *Proceedings of the 11th ACM Symposium on Operating System Principles* (Austin, TX, November 1987).