The Potential of Microkernels

ENGL 2201

When designing an operating system, many separate things must be considered, but perhaps the first choice that must be made is the design of the most basic part of any operating system: the kernel. Since the 1980s, there have been two competing kernel design philosophies: monolithic and microkernel. Due to a myriad of reasons, monolithic kernels are commonplace in desktop and server operating systems while microkernels are more common in embedded and special-purpose systems. But with recent improvements in microkernels, and changing requirements for production systems, there is a distinct possibility for microkernels to break into the desktop and server market and improve the security and reliability of computer systems.

The key difference between monolithic and microkernels lies in what each allows to run in kernel space. Kernel space is so named because it is the area of memory reserved by the kernel, which is the first part of the operating system to load. The kernel must accomplish inter-process communication (IPC) and memory, process, file, and I/O management. In a monolithic kernel all of these parts are loaded into kernel space, along with any needed device drivers. By contrast, microkernels typically only place the pieces responsible for IPC, memory, and process management in kernel space while everything runs in user space (Bellevue Linux Users Group, 2007).

One of the earliest and most famous microkernels is Mach, developed as a research project at Carnegie-Mellon University and first released to the public in a finished form in 1984. Mach was almost immediately adopted by the GNU project as a replacement for the UNIX kernel and is used in their open-source Unix-like OS, GNU Hurd (GNU Hurd Project, 2016). Mach was heralded as the microkernel that would revolutionize the desktop and server computer market, with technical publications of the day rejoicing at the possibilities it offered (Johnson, 1993). Another popular microkernel-based OS was MINIX, developed by Andrew Tanenbaum

as a tool to teach operating system design. Tanenbaum's statements on microkernel-based systems, posted on Usenet in 1992 as part of a critique of Linux and its monolithic kernel, illustrate the mindset of many microkernel supporters of the day:

> Microkernels have won. The only real argument for monolithic systems was performance, and there is now enough evidence showing that microkernel systems can be just as fast as monolithic systems… that it is now all over but the shoutin`.

These high hopes for microkernels began to fade as the 1990s dragged on however, as developmental progress on projects including the GNU Hurd and MINIX slowed to a crawl and industry attempts at developing a microkernel-based OS fell flat (Heiser, Elphinstone, Kuz, Klein & Petters, 2007, p. 4). The worst failure was IBM's Workplace OS, which was based on Mach and had a goal of allowing multiple incompatible operating systems to run side-by-side on a single computer. Despite around $2 billion of investment, overzealous ambitions and poor design decisions produced a system with lackluster performance and little of the promised features Heiser, Elphinstone, Kuz, Klein & Petters, 2007, p. 4). While microkernels were stagnating, the monolithic-kernel-based Linux was taking off, helped by its open-source nature and an ever-growing development community. It was these facts that led the development of a microkernel-based desktop and server OS to be all but abandoned at the turn of the century.

One of the main issues with Mach is that it suffers from slow IPC times, often up to 20 microseconds (Ahmed & Gokhale, 2009). In 1993, German computer scientist Jochen Liedtke set out to create a microkernel based from the ground up on performance and efficiency. The result was L4. Much smaller than Mach with fewer services running in the kernel, L4's performance and minimalist design have made it a popular topic of research ever since Liedtke's original version was released, hand-coded in assembly language for the Intel i386 processor. L4

has since been ported to many processor architectures and programming languages and deployed all over the world in embedded and mobile devices (Heiser & Elphinstone, 2016, p. 2). L4 has seen little, if any, use in the realm of desktop and server computing, despite its other successes. This is no doubt due in part to the fact that while microkernels were lagging behind in the 1990s, other, more mature, systems such as Linux, Microsoft Windows, and Apple Macintosh were surging ahead to dominate the desktop and server market. Doubtless the fact that these systems work well and are deeply entrenched in their roles has discouraged further development of microkernel-based desktop and server systems. Rather, almost all significant research and development of microkernels in the 21st century has been geared towards their use in embedded and mobile systems. However, this research has been extensive, and microkernels are now more mature than ever before, and the unique features they offer have the potential to improve the reliability and security of desktop and server systems.

Reliability has been a focus of microkernel research for a long time. It is especially critical in an always-on, embedded environment where even seconds of downtime ae unacceptable, such as flight control systems in aircraft or monitoring stations in industrial facilities (Heiser & Elphinstone, 2016, p. 3). Microkernels are uniquely suited to this task due to their modular design: with every important system service running outside the kernel in its own address space, any services that crash or otherwise fail can simply be restarted without affecting the rest of the system (Ahmed & Gokhale, 2009). Another aspect of reliability is the timing of the system; that is, the average and longest-possible time it takes the code of the operating system to execute a particular type of instruction. In real-time applications where it is imperative that incoming data is always acted on within a short time frame, this measurement becomes paramount. As far back as 2007, a project known as Potoroo was engaged in measuring the

temporal behavior of the L4 microkernel, with the goal of using their results to further improve its performance (Heiser, Elphinstone, Kuz, Klein & Petters, 2007, p. 7). A version of L4 produced by Open Kernel Labs has been deployed on billions of mobile devices as the operating system driving Qualcomm's wireless modem chipsets present on many cellular devices released since 2006 (Heiser & Elphinstone, 2016, p. 3). Such a massive deployment is a testament to modern microkernels' reliability, and something they could bring to the desktop and server market. The last thing anyone wants is their personal computer- let alone their enterprise server- to crash due to a hung service. With a microkernel-based system, such services could be restarted smoothly with little to no user intervention, helping to provide a seamless user experience.

One of the biggest concerns in the computing world today is security. While security solutions exist on all desktop and server operating systems, microkernels have the potential to increase security even further. By design, a microkernel separates all running programs into their own address space in memory, isolating them from each other, which can serve to thwart the attempts of a malicious program trying to gain heightened access to the system. Experimental systems have taken this even further, implementing subsystems capable of *forking* system services, such as the networking stack, among the various programs that need it. Doing this means that each program effectively gets its own copy of the service and anything it does with that service cannot affect any other programs' use of the service (Bloom, Parmer & Simha, 2016). Such isolation makes attacks on the system that rely on exploiting other programs harder to pull off. Other promising research involves the use of virtualization to increase security. While virtualization is commonplace in the server world, its use elsewhere has been limited. The modular design of microkernels makes them a promising choice for hosting virtualized systems, and in fact there are a few hypervisors (software that hosts a virtual

machine) based on microkernels, namely the open-source Xen and Microsoft's HyperV (Naji & Zbakh, 2017). Virtualizing entire operating systems can add an extra layer of security, as each guest operating system can use its own security measures as well. Finally, there are microkernels that have been built from the ground up for security, such as the version of L4 known as seL4 (Heiser, Elphinstone, Kuz, Klein & Petters, 2007, p. 6).

As a final point in the viability of desktop and server microkernel-based systems, consider that three of the most popular desktop and server systems today- Linux, Windows, and Mac- have all incorporated microkernel design concepts over the years to extend their functionality. The Linux kernel added the ability to load device drivers as modules after the system boots. Each module runs in user space and will have minimal effect on the rest of the system should it fail (Heiser & Elphinstone, 2016). This is very similar to how microkernels load device drivers- and much more- in user space, and always have. Microsoft Windows has used a hybrid kernel ever since Windows NT. This means that a microkernel is used to provide some of the most basic functions of the system, but code that would normally run outside kernel space in a true microkernel is loaded in kernel space to improve performance Apple incorporated code from the Mach-based microkernel used by BSD Unix into its Mac OS in the late 1990s, and all of their subsequent systems have made use of this as well. Like Windows, Mac is not a true microkernel, but what it chooses to load in kernel space differs from Windows (Bellevue Linux User's Group, 2007). This illustrates that even though microkernels themselves never took off in the desktop and server world, key concepts carried over, much to the benefit of all involved. If these concepts have worked out so well, why not take the next step, and implement full microkernels outside of the embedded systems world? Microkernels have already proven themselves there; it is time to see them prove themselves elsewhere.

References

Ahmed, M., & Gokhale, S. (2009). Reliable operating systems: Overview and techniques. *IETE Technical Review, 26*(6), 461-469. http://dx.doi.org/10.4103/0256-4602.57831

Bellevue Linux Users Group. (2007). The Linux Information Project. Retrieved March 21, 2018, from http://www.linfo.org/

Bloom, G., Parmer, G., & Simha, R. (2016). LockDown: An operating system for achieving service continuity by quarantining principals. Paper presented at the 9th European Workshop on System Security. doi:10.1145/2905760.2905764

GNU Hurd Project. (2016, December 18). GNU Hurd. Retrieved March 19, 2018, from https://www.gnu.org/software/hurd/

Heiser, G., & Elphinstone, K. (2016). L4 microkernels: The lessons from 20 years of research and deployment. *ACM Transactions on Computer Systems, 34*(1), 1.

Heiser, G., Elphinstone, K., Kuz, I., Klein, G., & Petters, S. (2007). Towards trustworthy computing systems: Taking microkernels to the next level. *ACM SIGOPS Operating Systems Review, 41*(4), 3-11. doi:10.1145/1278901.1278904

Johnson, J. (1993). Operating systems move to microkernel. *Software Magazine, 13*(7), 105.

Naji, H. Z., & Zbakh, M. (2017). A secure virtualization architecture based on a nested nova hypervisor. Paper presented at the 3rd International Conference of Cloud Computing Technologies and Applications. doi:10.1109/CloudTech.2017.8284737