JONATHAN S. SHAPIRO

extracting the lessons of Multics

Jonathan Shapiro is an assistant professor at Johns Hopkins University. His research interests focus on operating systems and system security. He is also a recidivist entrepreneur and has seen the development of several products from technical concept to market.

shap@eros-os.org
http://www.eros-os.org/~shap

THE KEYNOTE SPEAKER FOR THIS year's USENIX Security Conference was Earl Boebert, a key participant in the Multics project. Interspersed with pointed (and painfully accurate) insights about the state of the computing field, Dr. Boebert's talk focused on the structure of the Multics system, its key features, how many security problems it didn't have, and how little has been learned from it in the intervening 40 years. Karger and Schell made much the same claim in their retrospective paper.[1] In spite of being one of the best documented early operating system projects in existence,[2] and a staggeringly innovative effort in its own right, Multics is primarily remembered today as the system whose cost and ongoing delays caused Bell Labs to withdraw from the Multics collaboration and ultimately led Thompson, Ritchie, McIlroy, and Ossana to start the UNIX project. It seems fitting to try to answer the question Boebert posed in his talk: What can we learn from the past? Why did Multics fail?

The Multics project was proposed in 1962 by J.C.R. Licklider (the founder of DARPA) as part of his commitment to connected, multi-user computing. It was a year of dramatic invention. The notion of computer-supported collaboration, and therefore multi-user, time-shared computing in some form, was definitely "in the ether." Doug Engelbart and Ted Nelson would independently invent hypertext in 1962.[3] Berkeley, Dartmouth, and several other groups were exploring time-sharing systems, and 1962 was also the year that brought us Spacewar, the first computer game. The mouse would come shortly as Engelbart gained experience with Augment and, later, NLS (early hypertext systems). In the middle of this, a bunch of technologists decided to invent the modern computing utility. The Multics contract was awarded by DARPA in August 1964.

While few of the major innovations embodied in the Multics system were original to Multics, it was probably the first attempt to integrate so many ideas effectively. Virtual memory, segment-based protection, a hierarchical file system, shared memory multiprocessing, security, and online reconfiguration were all incorporated into the Multics design. Multics may have

been the first system implemented primarily in a high-level programming language, and it was one of the first to support multiple programming languages for creating applications. MACLISP, troff, and many other early tools trace their origins to the Multics system, as does much of the modern UNIX command line. Multics originated the term "computer utility," a concept that we have yet to fully explore 40 years later The integrated circuit was patented in 1959 and by 1963 was just entering the scene in the form of the 7400-series logic parts. Volume customers might soon expect to get as many as four *gates* on a single chip. . Considered in the context of then-available electronics technology, Multics was an incredible undertaking.

Today, we take for granted (at least, we *say* we do) many of the software techniques pioneered in the Multics effort. Multics was one of the first attempts at serious software engineering in a large, general-purpose system. It established the use of small, enforcedly encapsulated (isolated) software components (subsystems) that could be invoked only through their published interfaces. Using this fundamental building block, the Multicians crafted an end-to-end design that was both robust and secure. Even a casual reading of the Orange Book (TCSEC) standard reveals that many of the ideas of the Orange Book originated in either the Multics architecture or the Multics software process.

A skeptic might be prompted to ask, "If Multics was so wonderful, why aren't we using it today?" It is tempting to think, as Boebert implies, that Multics was a victim of the American desire for "crap in a hurry," and that this did not allow for the emphasis on quality engineering that delayed the completion of Multics. The truth, I suspect, is a matter of economics rather than bad taste. Multics was largely doomed by the intersection of two forces: the exploding growth of the computer and semiconductor industries, and a rising national sensibility of individual empowerment that brought, inevitably, the trend toward decentralized computing.

Exponential advances in integrated circuit design conspired against the Multics effort. In 1962, Fairchild Semiconductor was still shipping individual transistors, and this was "state of the art." By the time Bell Labs withdrew from the Multics project in 1969, Intel was shipping 64-bit memory chips. By the time Multics was presented commercially in 1973, Digital Equipment Corporation was shipping entry-level versions of PDP-11 systems running either RSTS-11 or RSX-11. DEC would ship the LSI-11 and the PDP-11/70 two years later, and in doing so would establish the features that would ultimately define mid-range computer architecture. In the microprocessor world, the 8080 would be running early versions of CP/M by 1974. A new era had arrived.

When it was announced in 1973, Multics was arguably the perfect answer to the problems of 1964, but it was too late, too expensive, too dependent on a proprietary hardware architecture, and too focused on centrally shared computing to be relevant in a world where decentralized, departmental computing was becoming the order of the day. Following the pattern of every other competitive market in history, the mainframe was being commoditized from below, and the era of "personal computing" would soon take over the world. Key elements of Multics—virtual memory, the hierarchical file system, multiple user support, and, to a lesser degree, online reconfiguration—would be rediscovered and incrementally introduced on Digital's VAX line of hardware, burdened at each step with the requirements of backwards compatibility. The same process of reinvention is happening now as Microsoft reshapes the underlying PC standard to support advanced server and management features. Not until the arrival of universal, always-on connectivity would the lessons of Multics once again seem relevant.

Ultimately, Multics failed because high-end computer-architecture ideas consolidated around the VAX and System 360 feature set, and shifted away from fea-

tures such as segmentation and multiple protection rings, on which Multics relied. This left the Multics operating system nonportable and unable to exploit the shift to cheaper, more open hardware systems. Unfortunately, it occurred at a time when circuit integration still wasn't far enough along to support basic protection features on low-end microprocessors. A decade later, UNIX would benefit from being able to ride a more stable, mature technology curve, reaping benefit from the same forces that doomed the Multics effort.

Sadly, later systems would show that neither segmentation nor multilevel protection rings were necessary. A simple user/supervisor split coupled with a paged memory management unit is sufficient. The Gemini project would construct an A1-certifiable operating system on a 386-class microprocessor. KeyKOS would provide Multics-comparable levels of robustness and security on commodity microprocessors. The EROS research effort has directly adopted many of these ideas. They seem to be contagious—the L4 project is now adopting many of them as well.[4] The "lots of small, protected memory objects" approach of Multics is reframed in KeyKOS, EROS, and successors as a "lots of small, protected processes" approach, which has stood up well to both formal and practical testing.

Unfortunately, neither UNIX nor Microsoft Windows managed to preserve (or successfully replace) the key security underpinnings of Multics, and we are now committed to a large body of insecure legacy software that will be difficult to overcome at a time when software patents make overcoming an entrenched legacy provider nearly impossible.

Given the economic and technology environment into which it emerged, the wonder of Multics is not that it has been ignored, but that so many of its key ideas have been adopted and adapted so pervasively in later efforts. Multics largely defined modern time-sharing systems, and its influence can be seen in every multi-user system that is shipping today.

FURTHER READING

1. Paul A. Karger and Roger R. Schell, "Thirty Years Later: Lessons from the Multics Security Evaluation." *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)* (December 2002)

2. Elliot I. Oganick, *The Multics System: An Examination of Its Structure* (Cambridge, MA: MIT Press, 1972). The best starting point today for information about the Multics system is the Multicians' Web site at http://www.multicians.org.

3. D.C. Engelbart, "Augmenting Human Intellect: A Conceptual Framework" (Stanford Research Institute, 1962); see Theodor H. Nelson, *Literary Machines 931* (Mindful Press, 1982)

4. KeyKOS: http://www.cis.upenn.edu/~KeyKOS; EROS: http://www.eros-os.org. L4: http://www.l4ka.org.