DOI:10.1145/2880150

Peter J. Denning

The Profession of IT
Fifty Years of Operating Systems

A recent celebration of 50 years of operating system research yields lessons for all professionals in designing offers for their clients.

Operating systems are a major enterprise within computing. They are hosted on a billion devices connected to the Internet. They were a $33 billion global market in 2014. The number of distinct new operating systems each decade is growing, from nine introduced in the 1950s to an estimated 350 introduced in the 2010s.[a]

Operating systems became the subject of productive research in late 1950s. In 1967, the leaders of operating systems research organized the SOSP (symposium on operating systems principles), starting a tradition of biannual SOSP conferences that has continued 50 years. The early identification of operating system principles crystallized support in 1971 for operating systems to become part of the computer science core curriculum (see the sidebar).

In October 2015, as part of SOSP-25, we celebrated 50 years of OS history. Ten speakers and a panel discussed the evolution of major segments of OS, focusing on the key insights that were eventually refined into OS principles (see http://sigops.org/sosp/sosp15/history). A video record is available in the ACM Digital Library. I write this summary not only because we are all professional users of operating systems, but also because these 50 years of operating systems research yield important lessons for all computing professionals who design systems for customers.

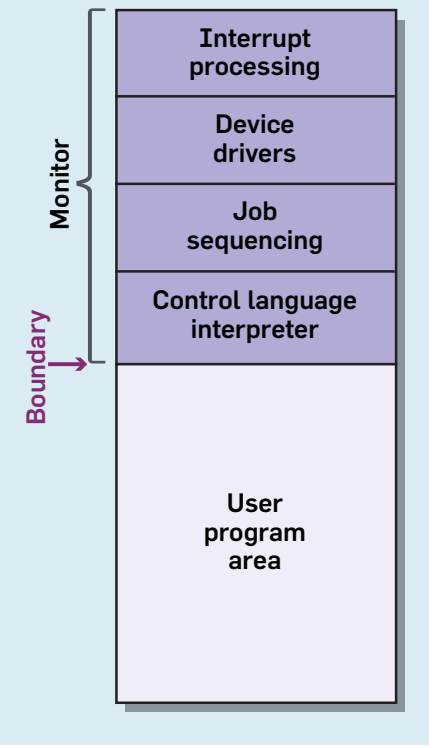
Timeline

A remarkable feature of our history is that the purposes and functions of an operating system have changed so much, encompassing four stages:

- Batch systems: one job at a time (1950–1960);
- Interactive systems: many users on multiple systems constantly interacting, communicating, and sharing resources (1960–1975);
- Desktop systems: Immersive personalizable distributed systems to manage work in an office (1975–2005); and
- Cloud-mobile systems: Immersive personalizable systems to manage all aspects of one's life, work, and social relations (2005 onward)

The accompanying figure depicts a memory layout of an early batch operating system.

Memory layout of an early batch operating system.



The Great Confluence of 1965

The very first operating systems were little more "manual operating procedures" for the first computers in the 1950s. These procedures established a queue of jobs waiting to be executed; an operator put the jobs on the machine one by one and returned output to the requesting users. These procedures were soon automated in the late 1950s; IBM's 1401 front end to the IBM 709x number crunchers was the best known of commercial "spooling" systems. From that time on, computer system engineers became interested in automating all aspects of computing including in-execution job scheduling, resource allocation, and user interaction, and pre-execution job design, preparation, testing, and debugging. By 1965, their experiments yielded a set of eight principles that became the starting point for a new generation of operating systems:

a See https://en.wikipedia.org/wiki/Timeline_of_operating_systems

- Interactive computing (time-sharing)
- Hierarchical file systems
- Fault tolerant structures
- Interrupt systems
- Automated overlays (virtual memory)
- Multiprogramming
- Modular programming
- Controlled information sharing

The MIT Multics project (http://multicians.org) and the IBM System 360 project were the first to bring forth systems with all these characteristics; Multics emphasized interactivity and community, System 360 a complete line of low to high-end machines with a common instruction set. Moreover, Multics used a high-level language (a subset of PL/I) to program the operating system because the designers did not want to tackle a system of such size with assembly language. Developed from 1964 to 1968, these systems had an enormous influence later generations of operating systems.

Dennis Ritchie and Ken Thompson at Bell Labs loved the services available from Multics, but loathed the size and cost. They extracted the best ideas and blended with a few of their own to produce Unix (1971), which was small enough to run on a minicomputer and was written in a new portable language C that was close enough to code to be efficient and high level enough to manage OS program complexity. Unix became a ubiquitous standard in the configuration interfaces of operating systems and in the middleware of the Internet. In 1987, Andy Tanenbaum released Minix, a student-oriented version of Unix. His student, Linus Torvalds, launched Linux from Minix.

OS Principles

By the late 1960s OS engineers believed they had learned a basic set of principles that led to reliable and dependable operating systems. The SOSP institutionalized their search for OS principles. In my own work, I broadened the search for principles to include all computing[3,4] (see http://greatprinciples.org).

I am often asked, "What is an OS (or CS) principle?" A principle is a statement either of a law of computing (Box 1) or of design wisdom for computing (Box 2).

Box 1. Examples of Laws

Semaphore invariant c(t) = min(a(t), s(t)+I) [5]

Space-time law: memory used = (spacetime per job)x(system throughput)

Mean value equations for throughput and response time in a queueing network

Locality principle

Box 2. Examples of Design Wisdom

Information hiding

Levels or layers of abstraction

Atomic transactions

Virtual machines

Least privilege

Of the many possible candidates for principle statements, which ones are worthy of remembering? Our late colleague Jim Gray proposed a criterion: A principle is great if it is "Cosmic"—it is timeless and incredibly useful. Operating systems contributed nearly one-third of the 41 great principles listed in a 2004 survey (see http://greatprinciples.org). The accompanying table gives examples—OS is truly a great contributor to the CS field.

Lessons

As I looked over the expanse of results achieved by the over 10,000 people who participated in OS research over the past 50 years, I saw some lessons that apply to our daily work as professionals.

Even though it seems that research is academic and does not apply to professional work, a closer look at

Both the researcher and professional search for answers. The one pushes the frontier of knowledge, the other makes systems more valuable to customers.

Founding History

My first volunteer position in ACM was editor of the SICTIME newsletter in 1968. SICTIME was the special interest committee on time-sharing, a small group of engineers and architects of experimental time-sharing systems during the 1960s. Jack Dennis (SICTIME) and Walter Kosinski (SICCOMM) organized the first symposium on operating systems principles (SOSP) in 1967 to celebrate the emergence of principles from the experimental systems and to promote research that would clearly articulate and validate future operating system principles. It is significant that they recognized the synergy between operating systems and networks before the ARPANET came online; Larry Roberts presented the ARPANET architecture proposal at the conference. The conference inspired such enthusiasm that the leaders of SICTIME wanted to convert their SIC to a SIG (special interest group); they recruited me to spearhead the conversion. I drafted a charter and bylaws and proposed to rename the group to operating systems because time-sharing was too narrow. The ACM Council approved SIGOPS in 1969 and ACM President Bernard Galler appointed me as the first chair. One of my projects was to organize a second SOSP at Princeton University in 1969. That conference also inspired much enthusiasm, and every two years since then SIGOPS has run SOSP, which evolved into the premier conference on operating systems research. SIGOPS has identified 48 Hall of Fame papers since 1966 that had a significant shaping influence on operating systems (see http://www.sigops.org/award-hof.html).

Following these successes, in 1970 Bruce Arden, representing COSINE (computer science in engineering), an NSF-sponsored project of the National Academy of Engineering, asked me to chair a task force to propose an undergraduate core course on operating system principles. A non-math core course was a radical idea at the time, but the existence of so many OS principles gave them confidence it could be done. Our small committee released its recommendation in 1971.[1] Many computer science and engineering departments adopted the course and soon there were several textbooks. I wrote a follow-on paper in 1972 that explained the significance of the paradigm shift of putting systems courses in the CS core.[2] After that, ACM curriculum committees began to include other systems courses in the core recommendations. The place of OS in the CS core has gone unchallenged for 45 years.

—Peter J. Denning

Examples of computing principles contributed by operating systems.

Process	A program in execution on a virtual processor. A process can be started, stopped, scheduled, and interacted with. Operating systems and networks comprise many interacting processes that never terminate.
Interactive Computation	Processes can receive inputs or generate outputs at any time—contrasts with the Turing view that processes get all their input at the start and produce all their output at the end. Interactive computations can implement functions that non-interactive computations cannot.
Concurrency controls	To avoid pathologies such as race conditions, buffer overflows, and deadlocks, processes need explicit mechanisms to wait for and receive signals.
Locality	Processes use small subsets of their address spaces for extended periods. Caches and memory managers detect working sets, position them for significantly improved performance, and protect them to prevent thrashing.
Naming and mapping	Objects can be assigned location-independent names; mappers translate names to object physical locations when needed. Hierarchical naming systems (such as directories and URLs) scale to very large name spaces.
Protection and Sharing	The global name space is visible to everyone (for example, the space of all Web URLs). Objects are by default accessible only to their owners. Owners explicitly state who is allowed to read or write their objects.
System Languages	System programming languages yield systems that are well structured, more easily verified, and fault tolerant.
Levels of Abstraction	System software can be simplified and verified by organizing the functions as a hierarchy that can make only downward calls and upward returns.
Virtual machines	A set of related functions can be implemented as a simulation of a machine whose interface is an "instruction set" and whose internal structure and data are hidden.

what actually happens reveals a great deal of overlap. Both the researcher and the professional seek answers to questions. The one aims to push the frontier of knowledge, the other to make a system more valuable to a customer. If we want to find out what it is like to explore a question, our main sources are academic research papers; there are very few written professional case studies. The typical research paper tells a tidy story of an investigation and a conclusion. But the actual investigation is usually untidy, uncertain, and messy. The uncertainty is a natural consequence of numerous contingencies and unpredictable circumstances through which the investigator must navigate. We can never know how a design proposal will be received until we try it and see how people react.

You can see this in the presentations of the speakers at the conference, as they looked back on their struggles to find answers to the questions they asked. They were successful because they allowed themselves to be beginners constantly searching for what works and what does not work: building, tinkering, and experimenting. From this emerged many insights.

The results of their work were almost always systems that others could use and experiment with. After the messy process of learning what worked, they wrote neat stories about what they learned. Before they produced theories, they first produced prototypes and systems.

Professionals do this too. When sitting around the fire spinning yarns of what they did for their customers, they too tell neat stories and graciously spare their clients their struggles with their designs.

References

1. COSINE Task Force 8 report. An Undergraduate Course on Operating Systems Principles. National Academy of Engineering, 1971; http://denninginstitute.com/pjd/PUBS/cosine-8.pdf
2. Denning, P. Operating systems principles and undergraduate computer science curricula. In *Proceedings of AFIPS Conference*. 40 (SJCC), 1972, 849–855; http://denninginstitute.com/pjd/PUBS/OSprinciples.pdf
3. Denning, P. Great principles of computing. *Commun. ACM 46*, 11 (Nov. 2003), 15–20.
4. Denning, P. and Martell, C. *Great Principles of Computing*. MIT Press, 2015.
5. Habermann, A.N. Synchronization of communicating processes. *Commun. ACM 15*, 3 (Mar. 1972), 171–176.

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for information innovation at the Naval Postgraduate School in Monterey, CA, is Editor of ACM *Ubiquity*, and is a past president of ACM. The author's views expressed here are not necessarily those of his employer or the U.S. federal government.