Reminiscences on the 25th SOSP's History Day Workshop

Peter G. Neumann, Senior Principal Scientist,
SRI International Computer Science Laboratory

Close to 400 people joined us for the 25th SOSP History Day workshop. It was a spectacular event with something for everyone. The older generations had the opportunity to see how ideas they grew up with interacted with other lines of ideas that evolved into present systems. The middle generation has the opportunity to see where the foundations of their areas came from and to meet some of the people who built those foundations. The newest generation had the opportunity to see the rich heritage of ideas, principles, and concerns they have inherited from 50 years of prior work by thousands of researchers. We also scheduled a less formal evening session to allow anyone among the conference participants to reminisce about operating system. Here are a few thoughts on the entire event, including contributed text from some of the participants.

We owe enormous thanks to Peter Denning for creating this event and to Ken Birman for his close coordination with PJD from the very beginning. The organizing committee for History Day was phenomenally responsive during the planning stages. Not only did they help choose the topics and the speakers, but they also helped the speakers by reviewing draft slide sets. The committee included three authors who were at SOSP 1 in Gatlinburg (Denning, Jack Dennis, Butler Lampson) and two authors from SOSP 2 in Princeton (Jerry Saltzer and myself -- I was also at SOSP 1), with the rest from later years. Jeanna Matthews, past chair of SIGOPS, was emcee for the day; with her deft hand at time management, all the speakers kept to the allotted 25 minutes and allowed time for questions from the audience.

The History Day slides and paper abstracts are on the History Day website: http://sigops.org/sosp/sosp15/history/ They are diverse, but comprehensive. In addition, copies of all past SOSP proceedings are accessible online: http://sigops.org/sosp/sosp15/archive Younger researchers especially are encouraged to look them up, together with the cited references, all of which represent a gold mine of treasures that are likely to contain nuggets useful for addressing today's problems.

Peter Denning spoke about the emergence of OS principle and showed several time lines of their evolution. He delved into memory management, just one timeline that unfolded the principles of location-independent addressing and locality. He was a leading researcher in those areas through the 1970s. Both areas were motivated by virtual memory -- which was a newly alluring but controversial technology at the time. The early concerns were whether the automation of mapping could be made transparent, and whether the automation of paging would perform well. A common principle across all implementations of virtual memory over the years is location independent addressing, and the methods of mapping addresses to the physical locations of objects. This principle was found in the original virtual memory, which had a contiguous address space made of pages, and is present in today's Internet, which provides a huge address space made of URLs, DOIs, and functionality. Performance was the other concern for early virtual

memory because the speed gap between a main memory access and a disk address was 10,000 or more; even a few page faults hurt performance. The locality principle emerged from years of study of paging algorithms, multiprogramming, and thrashing, where it was harnessed to measure working sets, avoid thrashing, and optimize system throughput. It is harnessed today in all levels of systems, including the many layers of cache built into chips and memory control systems, the platforms for powering cloud computing, and in the Internet itself to cache pages near their frequent users and avoid bottlenecks at popular servers.

Butler Lampson spoke of the evolution of principles and thought on protection and security, emphasizing how isolation is at the core of security but is sacrificed in the mania to share. For many years the principles that would lead to more secure systems took back seat to features that users found more convenient. He summarized the core principles of security with the Golden Rule of the three **AU**s: AUthentication, AUthorization, and AUditing. (I found Butler's speech and the entire event to be AUspicious, AUgmentational, AUthoritative, AUguring a better future, and overall AUsome.) In the final session, Butler insisted that we all need to be concerned about the growing cybercrime problems that have been exacerbated by our computer systems and networks. We cannot solve these problems alone, but we can contribute to solutions. He also maintained that users almost always choose convenience over security, and are unwilling to acknowledge that user attitude is the elephant in the room.

Butler wondered how many people would want to buy a car that contains ten logic bombs in its embedded computers that could go off at any random time and cause an accident. That is possible now, and no one is doing anything to stop it. In the early years of the SOSP there was a lot of concern about protection and security. In 1975 Jerry Saltzer and Mike Schroeder distilled eight principles from all the discussion and research into a paper that became a classic -- still read today by students of security. They did not mention a 9th principle, which appears in the Saltzer-Kaashoek book, Principles of Computer System Design (2009) -- minimizing the part of the system that must be trusted. That principle was formulated during the design of Multics (around 1965) and was part of its ring protection structure. The principle of minimal trust has been one of my guides for over 50 years.

In the committee for the COSINE-8 report proposing a core course on operating systems principles (1971), Butler took the protection topic. He proposed to represent access rights as a function ***Rights=f(principal,object).*** Other committee members thought this too abstract and suggested he display his function as a matrix. He acceded, and thus was born the access matrix model for representing the security policy of a system. Sometimes committees can be helpful in charting directions that shape history.

Barbara Liskov spoke about abstractions developed within the systems community to make it easier for implementers to organize system code; this work underlies the way systems are structured today. Two different approaches emerged from early work on how to structure systems that ran on a single computer. In one, described in Dijkstra's paper on the THE system, each individual resource (e.g., a printer) is controlled by a specific

"owner" process and code running in other processes use the resource by communicating with the owner using IPC (inter-process communication). In the other (described in Liskov's paper on the Venus system), resources are encapsulated inside modules and processes call operations provided by the module to use the resource; a popular way to control concurrency in this approach was provided by using monitors. Later work by Lauer and Needham showed that these two approaches are logically (but not necessarily operationally) the same, leaving each system developer free to choose the one that best fits the given architecture and application.

In the area of distributed computing, some form of IPC is needed and early work addressed the form this communication should take. Concerns included how to relate requests and replies, communicate arguments and results, allow servers to advertise their availability, and allow clients to select servers of interest. RPCs (remote procedure calls) that address these issues were first described in a paper by Birrell and Nelson. RPCs can be generalized beyond the call/return paradigm, e.g., to allow a client to make a sequence of calls that will be delivered in order, without the client having to wait for a reply from one call before making another. Generalized RPC between clients and servers is the approach used for all online applications today.

Edsger W. Dijkstra, our beloved colleague of the early years, was mentioned frequently throughout the day. His work on hierarchies of abstractions for operating systems, along with related work for programming by David Parnas, was hugely influential. His SOSP 1967 paper on the THE system inspired Barbara's early research on the use of abstraction levels to simplify systems and make them more resistant to errors. She noted that THE system was said to be free of deadlocks -- because function invocations were restricted to downward and upward calls in a process tree, which cannot allow a circular wait. Incidentally, Nico Habermann (Dijkstra's student and later colleague) told me that there had been unexpected deadlocks *within* a single layer, where cooperating processes could wind up in circular loops waiting for each other to send signals. Thus the design was "almost but not quite perfect."

Dijkstra himself added to the allure of the deadlock story at SOSP 1967. When it was his turn to present his paper, he said he would not speak about the THE system -- we could read that for ourselves -- but would instead speak of the problem of "deadly embrace", about which he was being asked in the hallways. This was a completely impromptu change of speaking plans. He sketched out a diagram on the chalkboard showing the state space of cooking stew and pudding in a kitchen with a single burner and single beater that both recipes needed. With his diagram he showed how certain (but not all) action sequences in the kitchen could lead to deadlock. To the surprise of many, there was an unsafe region of the state space such that if the joint progress path of the two recipes wandered into it, a deadlock at some future time was unavoidable. This diagram made such an impression that it wound up in many operating systems textbooks and inspired an entire line of research into deadlocks and preventing them. Two versions of it recreated subsequently are included here.

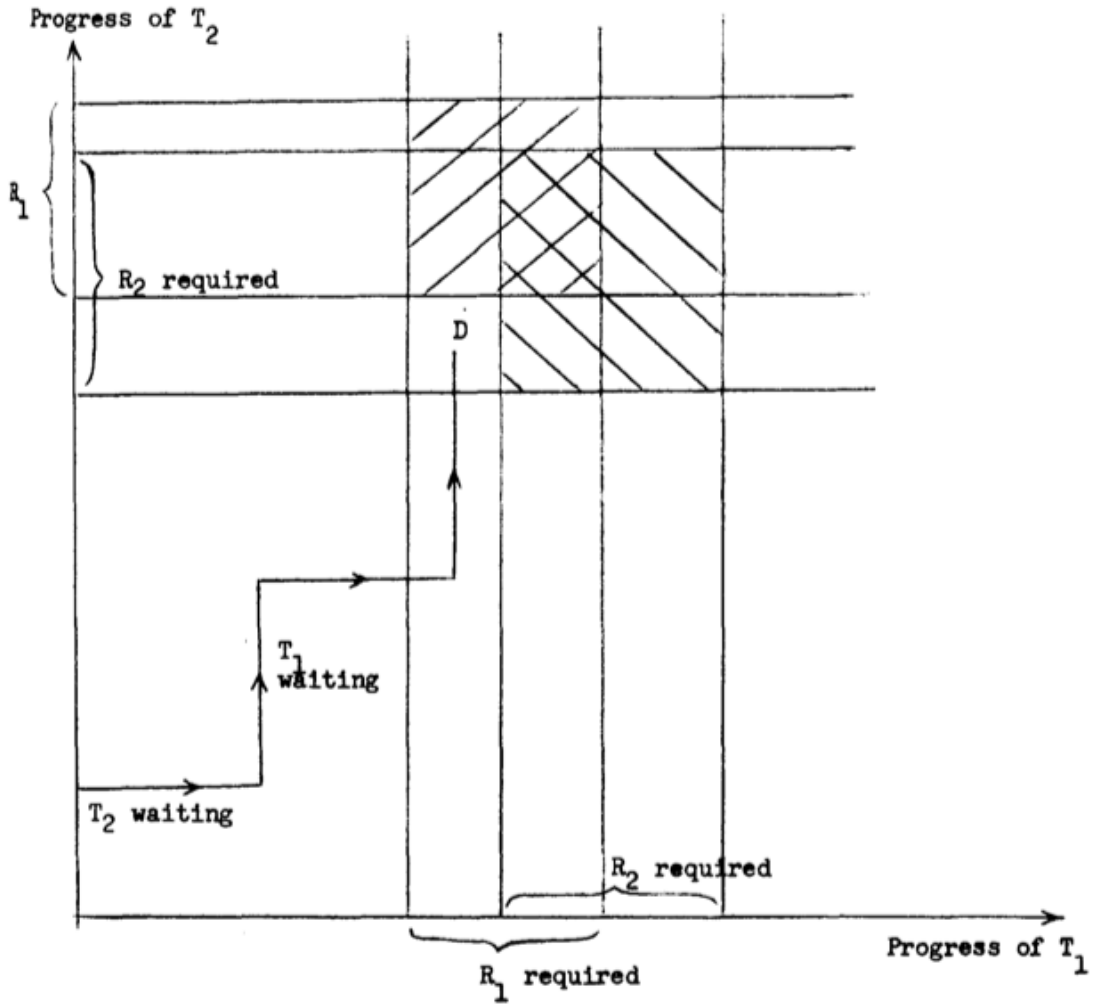

FIG. 2. Joint progress of tasks T_1, T_2.

Figure source: System Deadlocks, Coffman, Elphick, Shoshani, CSURV, June 1971.

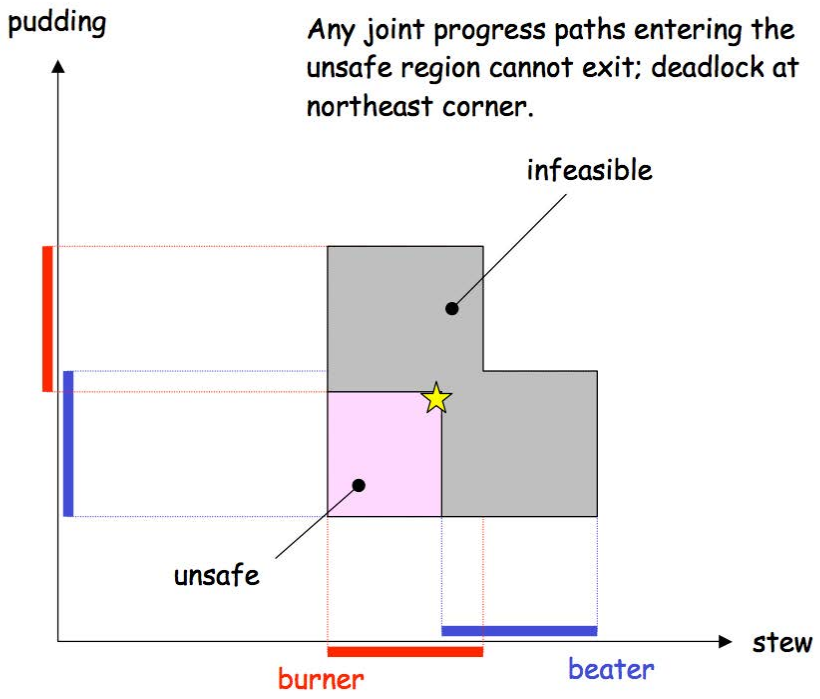

Figure source: Peter J. Denning, 2002, diagram for his students

I recall a WG2.3 meeting in Santa Cruz in the later 1970s when Edsger and I walked back through the redwoods after lunch. He was working for Burroughs at the time. I asked him what he was teaching Burroughs programmers about developing operating systems. (They wrote in a proprietary higher-level language, BALGOL; together with the PL/I subset and its Multics compiler -- developed by Doug McIlroy and Bob Morris -- they were the primary early adopters of higher-level languages for OS development.) Edsger's response was quite characteristic: "I don't do that. If I cannot write a program on the back of an envelope and prove it is correct, I have no interest in it." I do not know how many Burroughs programmers took his advice. I do know that his philosophy of building correctness into a system by design became very influential within the OS community.

In his comprehensive discussion of the evolution of memory and file systems, Mahadev Satyanarayanan (Satya) focused on the quests for scale, performance, transparency, and robustness at differing points in history, as well as gaining almost 13 orders of magnitude reduction in the cost per bit. As Bob Daley's co-designer of the Multics hierarchical file system with ACLs to help manage access (Fall Joint Computer Conference, 1965), I was particularly intrigued by Satya's comments refuting predictions that such file systems would soon be extinct! (Satya pointed out the origin of file-system hierarchicalization stems from Herb Simon in 1962.)

Ken Birman discussed the history of fault-tolerance and the CATOCS/CAP controversies (CATOCS = Causally and Totally Ordered Communication Support. This topic is rather complicated, and deserves considerably more detail from Ken than could include in his slides. However, his talk was a poignant reminder that we really need trustworthy systems that can help enforce a wide range of total-system trustworthiness properties.

Andrew Herbert spoke about the origins and evolution of virtualization systems. From beginnings as a testbed for exploring segmentation and paging algorithms (e.g., in Multics), early IBM work led to the successful and long-lasting mainframe operating system VM/370. In addition to timesharing, VM/370 had two other important capabilities: it allowed multiple IBM operating systems to be run on the same machine and it enabled operating systems to be developed using the same tools and environment as for applications. The virtual machine concept remained in the background of OS research until it reappeared in the early 21st century, first as a means to enable different desktop operating systems to coexist on the same personal computer and then as a means enable server consolidation. Today, virtual machines are a key component of cloud computing and the means by which server workloads are managed. Virtual machine research reported at SOSP has included hybrid virtualization, paravirtualization, virtual machine monitor resource management, virtualization for fault tolerance and for misuse detection and monitoring. Andrew also looked at the relationship between virtualization and abstraction (going back to Dijkstra's THE operating system) and the development of the concept of uniform location-independent names for virtualized resources.

Dave Clark spoke about the early days of networking. In the 1970s, the early implementations of TCP/IP were dreadfully slow. Dave and his colleagues spent years understanding in detail all the sources of inefficiency and one by one worked them out. When they were done TCP/IP was tightly integrated into the operating system and was very efficient. Without that work, the Internet would not have taken off. This is one of the lessons of history. It often takes a lot of engineering work to transform a great idea (such as TCP/IP) into a working system with acceptable performance. We tend to remember the people who discovered the idea but not the many engineers who figured out how to make it work.

Dave Patterson spoke about the evolution of computer hardware and how gross differences between instruction sets of machines was a major impediment to interoperability. The first attempts to standardize machines around instruction sets were with the IBM 360 and evolved into machines with very complex instruction sets (CISC). Beginning with John Cocke, researchers found that simplifying instructions around an execution pipeline significantly improved performance and gave birth to the RISC revolution in the early 1980s. Some of the great ideas of earlier eras, such as capability addressing, were lost in the RISC revolution. Today there is an effort to produce open instruction sets that all chip makers could use, and some researchers are aiming to include capability addressing in that architecture. I think many participants came to a new appreciation of hardware they did not have before.

Frans Kaashoek's talk divided research on parallelism in operating systems in 4 periods. He summarized foundational ideas for parallel programming, covering three types of parallelism in operating systems: user-generated parallelism, I/O parallelism, and processor parallelism. With the arrival of commodity small-scale multiprocessors, the OS community "rediscovered" the importance of processor parallelism and contributed techniques to scale operating systems to large number of processors. These techniques found their way in today's mainstream operating systems largely because of the emerging ubiquity of multicore processors. Frans observed a "rebirth" of research in parallel computing as a result. Given the advent of cloud computing and the presence of frequently compromised operating systems, Frans's talk was especially timely as a cry for future research.

Jeff Dean described the development of systems that are going to be essential for cloud computing systems. These systems are going to have to address trustworthiness much more systemically, incorporating security, reliability, distributed transactional consistency, high-performance computing, and lots more. My own view of cloud computing is that has the potential to combine the worst aspects of outsourcing to untrustworthy or unaccountable third parties, and will introduce enormous risks unless major improvements in trustworthy operating systems and networking are achieved. Caveat Emptor!

The day ended with a panel looking at the future of system support for security, with Jeanna Matthews as the moderator, and David Mazieres, Mark Miller, Margo Seltzer and YY Zhou. The panel raised many diverse points, and was evidently engaged the audience.

Aside from Clark and Patterson, the other speakers focused mostly on software. They all called for constant attention to the principle that operating systems be designed holistically, with system architectures coordinating hardware, software, and even programming languages around user practices and concerns. Peter Denning mentioned this in his overview of evolution of the purposes of operating systems -- the current-generation OSs present an immersive environment in which users manage their life, work, and social networks. However, total-system properties such as human safety and system survivability in the face of accidental, intentional, and environmental disturbances were not mentioned, yet must also be systemically addressed. The same is true for hardware that can more effectively enforce what is needed for the operation systems and applications.

Bob Morris's 19 September 1988 quote before the National Research Council Computer Science and Telecommunications Board is even more relevant today, with the potentials for crime and abuse in the extended Internet. (Note that the `T' in CSTB later became 'Technology'.)

> "To a first approximation, every computer in the world is connected with every other computer."

Capability-based hardware and operating systems seem to be having a renaissance. In 1971, Butler Lampson discussed the intuitive notion of an access-control matrix, with a row for each subject and a column for each object. In that model, each column represents an access control list for an object (as introduced in Multics, and noted above), and each row represents capabilities for accesses permitted to that subject. In his talk, Lampson noted that short-term capabilities (access tickets with an expiry of a few minutes) are much more useful than long-term capabilities (access tickets that do not expire, causing revocation and auditing problems). History is replete with many attempts to design and implement various types of capability-based systems -- including include Lampson's Berkeley system, Plessey 250, PSOS, Hydra, KeyKos, EROS, the E-system, Cambridge's CAP, and most recently NICTA's seL4 and the SRI-University of Cambridge CHERI (Capability Hardware Enhanced RISC Instructions) system, with MIPS-64-based hardware running FreeBSD. Attendees involved in capability-based systems also included Andrew Herbert, Alan Karp, and Mark Miller, and myself (in addition to Robert Watson on the committee, who was unable to attend). This is a diverse community, but generally quite optimistic about the potential role of capabilities in the future.

In the evening event, Ken Birman took a moment to remind us of Jim Gray (1944-2008), who received the ACM Turing Award in 1998 and was a passionate participant in the ACM systems community. Although his roots were primarily in the transactional and database systems communities, Jim nonetheless made a point of attending SOSP and often spoke about fault-tolerance scalability of consistency mechanisms. Jim's unique mix of technical wisdom and personal warmth, and his style of outreach and engagement with the new generation of researchers, is truly inspirational.

Many other individuals have contributed significantly to our history and the current state of the art. For example, other authors from those first two SOSPs included Brian Randell, Earl Van Horn, and Bob Daley from 1967, and Roger Needham, Brian Kernighan, Bill Wulf, Elliott Organick, and George Mealy from 1969 -- to name just a few. Ken Thompson and Dennis Ritchie were authors in 1971. Also, work by many other people was mentioned. For example, Andrew Herbert noted the importance of the work of Roger Needham, Peter Denning noted the early work on memory management and locality by Les Belady at IBM, Ken Birman noted the recursive recovery-block work led by Brian Randell, and several people mentioned the long-time contributions of Les Lamport on distributed algorithms. Everyone named here is an important part of our history, as are many other authors not explicitly mentioned.

All in all, the 25th SOSP History Day was remarkably valuable for everyone, old and young. Clearly, it would be a major task to do justice to everyone who has contributed to this history, although that is way beyond the intended scope of my brief remarks. However, I hope that the SOSP History Day can inspire a comprehensive project, perhaps beginning with a collection of oral histories akin to what has been done in the computer security community -- where over 300 people were interviewed by the Babbage Institute of the University of Minnesota, under the aegis of NSF funding. I also hope that some of the attendees of the 25th SOSP History Day will be around to organize a 50th SOSP History Day in 2065.

PGN, 23 October 2015