Summary of the

Sixth SIGOPS European Workshop on "Matching Operating Systems to Application Needs"

Marc Shapiro, Program Chair

Dagstuhl Castle, Wadern, Germany, 12–14 September 1994

Every other year since 1984, SIGOPS organizes a European Workshop. The EW traditionally invites new ideas and encourages discussions that do not fit into the more result-oriented conferences such as SOSP or Usenix. Along this tradition, this year's theme is "Matching Operating Systems to Applications Needs."

The Program Commitee selected papers, first through an electronic-mail vote, then a physical meeting. Of the 53 submissions, 20 were invited for a formal presentation, on the basis of interest, appropriateness to the workshop topic, technical soundness, an quality of exposition. 20 more were selected to attend the workshop and participate in the discussions. The participants represent both the traditional OS research community and some more unusual application areas. They come from industry and academia, mostly from the US and Europe.

To increase the effectiveness of the meeting, we have encouraged authors to read each other's position paper in advance, via an FTP repository. The proceedings are now accessible by anonymous FTP, at ftp://sigops.informatik.uni-kl.de/sigops.

The following position papers have been selected for inclusion in OSR: "Application-Aware Adaptation for Mobile Computing" by M. Satyanarayanan *et al.*, "New Directions for Integrated Circuit Card Operating Systems" by P. Paradinas and J.-J. Vandewalle, "On Group Communication in Large Scale Distributed Systems" by Ö. Babaoğlu and A. Schiper, "Matching Data Storage to Application Needs" by D. Dean and R. Zippel, "SPIN: An Extensible Micro-Kernel for Application-Specific Operating System Services" by B. Bershad *et al.*, "The Operating System Kernel as a Secure Programmable Machine" by D. Engler *et al.*, "A Caching Model of Operating System Functionality" by D. Cheriton and K. Duda, "Address Space Sparsity and Fine Granularity" by J. Liedtke, and "Objects to the rescue! or httpd: the next-generation operating system" by A. Black and J. Walpole.

Many thanks to all for their participation. Thanks also to Jürgen Nehmer, the general chair, for the excellent site and organization. His students noted the discussions found hereafter. Thanks to the session chairs (most were also PC members) for their help with this summary.

Session 1: Applications (chair: Frans Kaashoek)

The first session focussed on the impact of applications on systems. The applications covered a wide spectrum of systems: (1) mobile applications, (2) IC-card applications, and (3) geographically distributed applications. Brian Noble spoke about Odessey, a system for nomadic machines. He stressed the importance of flexibility of systems for mobiles to allow for difference resource management policies. Pierre Paradinas argued for including a small object-oriented operating system on IC cards to allow for application-specific extensions of the functionality provided. Dag Johansen summarized the lessons learned from building StormCast, a geographically distributed application with real users running on wide variety of architectural platforms connected by low- and high-speed links. The session ended with a discussion on what applications need from the underlying systems, making it an appropriate opening session that matches the theme of the workshop closely.

To what degree does Odyssey's exposure of mobility and diversity make the job of an application

programmer harder? Brian: The key to managing complexity for applications is to provide services and abstractions that solve *most* problems, and then to minimize the amount of work that applications with exceptional requirements must do.

StormCast seems to be a specific system that might be effectively supported by Odyssey. How would you do it? Brian: The separate data types in Odyssey are a more heavy-weight notion than the separate kind of data sources in StormCast. But Odyssey could provide a good structural base for StormCast.

How can the code executed by an IC card be made secure? Pierre: The code has a cryptographic signature that must be checked by the card OS. The signature and cryptographic keys are stored in the applicative memory of the card. Only the card-OS is allowed to access this information.

Wouldn't a PDA be more secure than a card? This way the user does not trust his PIN to to an external reader. Pierre: Regardless of whether on a PDA or a smart card, the CPU and private data must be implemented in a single chip, for security reasons. Another security problem is that a PIN is typed in the clear. Only biometric identification, such as signature recognition, would solve this.

What is the benefit of downloading code to the card? Can't it all be done in the reader? Pierre: A reader is a passive device, just a connector between your card and servers.

Why don't StormCast applications need to execute atomically? Dag: Because of the nature of the application and limitations of underlying technology. For instance, a data collection application over a large area over slow, unreliable networks needs a majority of responses within the time contraints. The parts of an application requiring atomicity should be located in a different environment, *e.g.,* above Isis on a LAN.

How does your application programmer choose from the many diverse mechanisms? Dag: StormCast offers a common, layered architecture and reusable modules at each layer.

What are the required features of a distributed operating system for your application? Dag: Different applications need different abstractions. OSes such as DOS and Unix keep growing to keep up with such requirements. Instead, they should just provide basic mechanisms, and a few flexible high-layer services such as group communication.

Session 2: Communication (Chair: Sacha Krakowiak)

The paper "On Group Communication in Large-Scale Distributed Systems," ws presented by Özalp Babaoğlu. He advocates process groups as an application structuring mechanism well suited to large-scale systems. In a large scale system, reachability is asymmetric, non-transitive, and possibly non-connected.

Technology advances will not solve the scale problem (connectivity will always remain a problem, especially with mobile hosts). Applications should be structured with a relatively small "core" of processes, over which strong view-synchrony needs to be guaranteed. Examples are replicated file systems and cooperative editors. Large applications like StormCast do not need strong consistency, but would benefit from Özalp's Client and Sink communication facilities.

"Optimistic Active Messages: Structuring Systems for High-Performance Communication", was presented by Frans Kaashoek. Optimistic Active Messages enable applications to directly use low-level communication primitives. The compiler incorporates application code into message handlers. The method is optimistic because it expects that message handlers rarely block.

This approach requires a cooperation between the network interface and the processor, for tagging messages. The technique is generally applicable, to RPC, to distributed shared memory, and even outside the communication area. Any software that deals with interrupts (network, disk, etc.) can benefit from the optimistic approach.

Session 3: Objects and Meta-Objects (Chair: Jürgen Nehmer)

Dawson Dean proposes the *microstore*, a universal platform for implementing different storage personalities. Data is stored in variable-length segements. The second and third talk are about meta-objects

as the building blocks for systems. Robert Stroud proposes a persistent storage system that uses reflection to adapt to application needs. Yasuhiko Yokote reports on the current status of Apertos, a system platform base on meta-objects.

Mustn't each storage personality understand the semantics of all other personalities' data? Dawson: Two programs share a file if, either they don't care about the data semantics, or they both understand the semantics. This remains true for microstores.

Give an example application that uses segments. Dawson: Any program that uses sectioned files, *e.g.,* OLE, OpenDoc or HTML. If each section is separate, sections can be shared, may grow and shrink efficiently, or may have different access controls or disk layouts.

What lessons have you learned from building your system? Dawson: The system originally only provided multiple interfaces. Now we realize that multiple implementation policies are important, such as disk layout and cache policy.

What is a reflective system? Robert: In reflective system, an application can observe, feed back into the system, and manipulate its own computational behaviour, via a a "meta-level" interface that exposes the system's structure and behaviour. Reflection cleanly separates application concerns from implementation concerns. Requirements such as dependability and distribution transparency can be addressed more effectively at the meta-level.

Session 4: Consistency in DSM Systems (Chair: Jürgen Nehmer)

Mustaque Ahamad presents "causal memory" as a means to improve DSM performance. This weak consistency model ensures that values returned by read operations are consistent with causal orderings between data accesses. The discussion focused on the distinction between causal memory and related weak consistency models. Causal memory provides non-blocking implementations for shared accesses (no communication required between processors). Also, it naturally captures the semantics of asynchronous, loosely coupled entities. For the class of data-race-free programs, the systems are similar but there are some implementation differences. Causal memory, however, is defined for all programs, including ones lacking explicit synchronization.

Rachid Guerraoui reported on the GARF library, an extensible set of mechanisms for DSM, at varying degrees of consistency. An object is bound at run time to a "mailer" class that manages its consistency (changing mailer changes the consistency model). The discussion raised the issue of adequate support for helping application programmers to select the most appropriate consistency model.

Session 5: System Customization (Chair: Paul Leach)

Gilles Muller presented "Towards Safe and Efficient Customization in Distributed Systems." He proposes to customize systems by loading code into a separate protection domain, with supervisor privilege, of the application's address space. Protection domains are implemented by software fault isolation. Dawson Engler presented the exokernel concept in "The Operating System Kernel as a Secure Programmable Machine." An exokernel exports hardware features to applications. Code is downloaded safely into the exokernel using compilation and code inspection techniques. Brian Bershad presents "SPIN: An Extensible Micro-Kernel for Application-Specific Operating System Services." User code can dynamically compile new "spindles" into the kernel; aggressive compiler technology provides both safety and performance.

Is overflowing kernel address space a problem in practice? Gilles: Kernel address space is a bounded shared resource, and loading different user modules in the kernel poses protection problems. In DP-Mach, customized code is private an application.

To what extent do you expose hardware issues to the application programmer? Gilles: The application programmer should ignore hardware implementation details. Customized functions are automatically generated.

Dawson: The OS is the wrong place to put a virtual machine. Insted of abstracting hardware resources, Exokernel safely exposes the hardware to the user. The application no longer needs special privileges to alter the underlying virtual machine.

Doesn't the application have to do a lot of work? Are you giving up portability when exposing hardware? Dawson: The low-level interface will be abstracted by a set of libraries that are linked into the application. Libraries isolate machine specifics; in Exokernel this layer is in user-space and does not requires special privilege to modify.

Is dynamic linking reflection? Brian: No. Dynamic linking alone just lets you late-bind identifiers, which you manipulate explicitly (*e.g.,* call, read, write). Reflection implies late-bound behaviour of implicitly referenced services. If A calls B calls C, then C is implicitly called by A. If A can control what C is, then we have reflection.

Are you making the application programmers' job more complex? Brian: Probably, since we are assigning the programmer greater responsibility in determining system behaviour. On the other hand, the wrong fixed set of interfaces, no matter how fast, could be more complicated that the right set, so we could make some of the programmers' lives easier.

Session 6: Memory Management (Chair: Andy Tanenbaum)

Xavier Rousset presented "Single Address Space or Private Address Spaces." The advent of 64-bit address spaces provides the opportunity for multiple processes to share an address space without a danger of running out of address space. The key issue here is protection. Xavier proposes using segments protected by software capabilities to implement a distributed shared memory across machines.

Jochen Liedtke presented "Address Space Sparsity and Fine Granularity". Sparse 2^{64} virtual address spaces are not well supported either by conventional page table trees nor inverted page tables. Guarded Page Tables (GPTs) are tree-structured and perform well in time and space for arbitrary sparsity. They support fine-granular spaces and mixed-sized pages. Jochen had convincing answers to the audience's many questions, that are too technical to reproduce here.

Doesn't the "short-pointer space" of the PA-RISC solve your problem? Xavier: It does, in a limited fashion. A big issue is to know on a page fault whether the current domain can map the page, *i.e.,* mapping addresses back to capabilities.

Have you considered fragmentation of address spaces? Xavier: Here are some ideas: (i) never reclaim virtual memory allocated to segments; physical memory is reclaimed by capability-based garbage collection. (ii) On each node a hash table locates the segments "offered" on this node.

Will TLB bandwidth dominate performance if intensive sharing at arbitrary granularity occurs? Jochen: We have sound idea of the programming styles that will evolve as fine grained address spaces become available. Therefore, we cannot yet predict the required TLB bandwidth. But fortunately, it is not correlated to the set of pages allocated in RAM.

Session 7: Kernels (Chair: Özalp Babaoğlu)

In this year's meeting, the theme "Extensible Microkernels" appeared to be a popular one. The two presentations in this session reported on prototype systems being developed in order to verify different aspects of this theme.

In "Meeting the Application in User Space," Holger Assenmacher described the "picokernel" of the Panda system. This kernel is only 30K lines and implements a small set of orthogonal abstractions through which flexible interfaces supporting different programming paradigms can be built.

Kenneth Duda reported work at Stanford on the *cache kernel* in "A Caching Model of Operating System Kernel Functionality". In this system, the kernel is just a cache for typical operating system objects such as processes, address spaces and communication channels, rather than a complete manager of these objects. It makes hardware functionality available with minimal overhead. The resulting system is claimed to be smaller than typical microkernels and results in greater flexibility for application support.

What makes the pico-kernel approach different? Holger: It makes no assumptions about the user-level policies, *e.g.,* all kernel calls are non-blocking, which removes all obstacles to user-level thread management. It minimizes predetermined mechanisms or policies, such as protection or synchronization.

Is persistence really orthogonal to mobility? How can DSM be independent from synchronization?

How to achieve orthogonality in the general case? Holger: Copying to/from background storage and locking is independent from moving objects to other nodes. Our DSM is only a user-level thread that handles page faults. Mechanisms and policies may be freely combined if the implementation avoids dependence on later combinations, and avoids unencapsulated cross-mechanism optimizations.

How are your processes different from Unix processes? Kenneth: (1) The primary operations are "load" and "unload." (2) "Change priority" is only an optimization. (3) Unix processes have file descriptors, signal masks, nice levels, etc. CK processes have the minimal state to safely/fairly share the CPU.

What makes the caching implementation simpler? Kenneth: (1) CK does not need to page. (2) CK never reports that it is out of resources. (3) CK can assume that objects never change type, that is, memory containing a thread always contains a thread. This simplifies concurrency. (4) CK stores objects in local memory on a multiprocessor.

Session 8: Operating System Architecture (Chair: Marc Shapiro)

According to John Wilkes' paper, "Better Mousetraps," a lot of the current operating systems research is not relevant because it does not take economics into account. Small systems have limited battery reserves, CPU power, memory, network bandwidth, etc. Specialization is important to maximize resource utilization. A declarative interface is more friendly for application programmers, and leaves more room for clever implementations and optimizations. The OS should adapt its behaviour automatically to observed access patterns, according to the declarative specifications. But real-time or bandwidth requirements must be specified in advance and are not optional.

Sape Mullender says that "You and I are Past Our Dancing Days". General-purpose operating systems can only become more arcane and bloated. Sharing resources is getting harder, because: (i) isochronous applications (video and audio) have Quality of Service requirements; (ii) we must take into account new, scarce resources such as wireless bandwidth, battery power, communication costs. One solution is collaborative sharing at the application level rather than competitive multiplexing in the OS. Programmable consumer multimedia applications hardly needs a Unix.

The audience asks why bother with QoS; current machines and networks have adequate power and bandwidth. Sape answers that today, multimedia applications just transport audio and video data but do not process them (certainly not in real time). Later they will. Also we will see a need for 3D real-time video, real-time holography, animation, etc.

The title of Andrew Black's presentation is "Objects to the rescue! or httpd: the next-generation operating system." An object-oriented OS structure is best in terms of modularity. Such an OS must provide uniform naming, persistent objects, invocation mechanisms, a library of useful objects, extensibility to arbitrary new object types, and critical mass. The only system that comes close to fulfilling these requirements is WWW. It's big, it's slow, it's not an OS, but who cares? Not the users; they care that it does something useful. In the long run, specialized object implementations offer opportunity for excellent performance.

The audience was skeptical: should aeronautics engineers worry about the quality of food on board? Andrew answers we should stop looking at super-sonic transporters and start thinking about airbusses. A question: "You are telling us that URLs are good, but most of the time they don't work; the page isn't there." Answer: Right. Why didn't they use something better? Because we — the OS community — did not provide it.