CS/TR-91-104

Next Generation Operating Systems Architecture

Michel Gien

Chorus systèmes

6 avenue Gustave Eiffel, F–78182 Saint-Quentin-en-Yvelines (France)
tel: +33 1 30 64 82 00, fax: +33 1 30 57 00 66, e-mail: `mg@chorus.fr`

What is needed?

Software engineering for operating system development

Operating Systems complexity is increasing. Looking at UNIX evolution, several trends are pulling it away from its roots. As more functionality is continually demanded, it is unavoidable that future versions should be more complex. Unfortunately, a by-product of this increasing complexity is the gradual undermining of UNIX original benefits: simplicity and portability.
Instead of evolving towards more portability, new releases require increasing efforts to implement on new platforms, particularly when they are built around emerging hardware architectures that UNIX was not initially designed to support.
Once a port has been done, maintainability is becoming a major issue. New releases are difficult to integrate in a system in which numerous adaptations and extensions have been made. It often requires almost a complete new port.
Simplicity is required, not in the number of functions performed but in the architecture and the concepts which are at the foundation of the system construction.

‘‘Current operating systems architectures such as UNIX cannot fulfill the expectations of system builders if they continue to increase in complexity, without a well-defined modular architecture, based on simple concepts, to manage this growth.’’

Standard operating system components

Ten years ago system builders developed their own proprietary operating systems of necessity, often simultaneously with the underlying hardware. UNIX introduced the concept of a standard, hardware-independent operating system, whose portability allowed platform builders to reduce their time to market by eliminating the need to develop their own operating systems for each new platform. Similarly, builders of real-time systems have begun to use non-proprietary operating systems, such as pSOS or VRTX, often bundled with boards that can be readily integrated into their systems.

In: Lecture Notes in Computer Science n° 563, Springer-Verlag, Proc. of the International Workshop on Operating Systems of the 90s and Beyond, Dagstuhl Castle, Germany, July 1991

''System builders need to work from standard Operating System components as building blocks for the construction of their modern computing systems.''

Support for an extending scope of applications

The open systems standard that UNIX represents is naturally appealing to application domains that have been dominated by highly-specialized systems. As a result, a contradiction is emerging. Although system builders are adopting UNIX because it is a standard operating system, they are simultaneously demanding extensions to customize it to vertical applications.
For example, UNIX is being extended to support the execution of some categories of real-time applications. Conversely, real-time systems builders want the capabilities of real-time executives to be extended to support UNIX applications. Similar trends can be found in transaction-processing and multi-media environments. This requires extending open systems to areas they sometimes don't fit very well.

''System builders need more generality from the operating system, and more flexibility to tailor it to specific environment without introducing complexity corrupting standard components.''

Support of transparent distribution at all levels

Today's computing environments require that new hardware and software resources, such as specialized servers and applications, be integrated into a single system, distributed over some kind of communication medium. The range of communication media commonly encountered includes shared memory, buses, high speed interconnection networks, local and wide-area networks. This trend will become fundamental as cooperative computing environments emerge to better map natural human organization.

''Facilities to support distribution are now required at the most intimate level of modern computing systems.''

In sum, builders of the next generation of system platforms face three concurrent software engineering challenges:

- to support new multiprocessing and parallel hardware architectures, built around evolving micro-processor architectures (64-bit RISC),
- to incorporate their own value, adding operating system features within a standard open systems framework,
- to gracefully extend the topology and functionality of their system into cooperative computing environments.

How to get there?

Micro-kernel operating system architecture

A recent trend in operating system development, as shown by systems like Amoeba[1], CHORUS[2], Mach[3], Plan 9[4], Topaz[5] or the V-system[6], consists of structuring the operating system as a modular set of system servers which sit on top of a minimal Micro-kernel, rather than using the traditional monolithic structure. This new approach promises to help meet system and platform builders' needs for a sophisticated operating system development environment that can cope with growing complexity, new architectures, and changing market conditions.

In this operating system architecture, the *Micro-kernel* provides system servers with generic tools. Those are limited to generic processor scheduling and memory management functions, independent of a particular operating system environment, and a simple Inter-Process Communication (IPC) facility that allows system servers to interact independently of where they are executed, in a multiprocessor, multicomputer, or network configuration.

This combination of primitive services forms a standard base which in turn supports the implementation of functions that are specific to a particular operating system environment. These system-specific functions can then be configured, as appropriate, into system servers managing the other physical and logical resources of a computer system, such as files, devices and high-level communication services. We refer to such a set of system servers as a *subsystem*. Real-time systems tend to be built along similar lines, with a very simple generic executive supporting application-specific real-time tasks.
Subsystems separate the functions of the operating system into sets of services provided by autonomous servers, and provide operating system interfaces to application programs. In the past, these functions were incorporated in the kernel of monolithic systems. Such an organization of Micro-kernel and subsystems represents the most logical view of an open operating system for cooperative computing environments. Separating functions increases the modularity, portability, scalability and "distributability" of the overall system, which has a small, trustworthy Micro-kernel as its foundation.

System Servers are the building block, the toolset, that one can use to incorporate new distributed functions. This framework for innovation within the open system context can facilitate the addition of product differentiators usually found only in proprietary systems, such as fault tolerance and security features. Price/performance and reliability can also be enhanced because developers can efficiently incorporate new hardware connection technologies and new processors in their next-generation designs.

Micro-kernel based architecture is often justified as an approach to reducing the size and complexity of the operating system. This is not quite true. Restructuring the operating system along a better architecture does not reduce the overall quantity of code that is necessary to perform a given function. The demand from applications for increasingly sophisticated services from the operating system requires ever more code.
What Micro-kernel operating system architecture does provide is a structured, defined way to cope with the increased complexity of operating system development. It also offers great advantage as a framework for the design, development and integration of ''open'' operating systems. Further, this structured approach must be followed especially when operating system functions are distributed among loosely coupled nodes.

''The argument for Micro-kernel architectures is indeed a software engineering argument, which applies to system builders.''

A distributed computing technology

Micro-kernel architectures have been, and remain, strongly related to distributed computing. Both research and commercial work presumed a set of distributed computing requirements, within or between networked ''boxes''. Message passing, which has come to be one of the often distinguishing characteristics of Micro-kernel architecture, is a very natural way to structure systems in which components are distributed over a loosely-coupled set of individual processors, boards or complete machines. It enforces very clear isolation between each individual component of the system, by making explicit the communications rules used between them, while at the same time providing a very flexible way to assemble distributed components into a higher level global entity. Message passing can take the form of simple send/receive protocols exchanged for transferring data between remote entities as well as means for synchronizing parallel independent execution flows. Simple send/reply messages can be combined into a form of remote procedure call (RPC) to better suit client-server types of communications. This concept has been used under various forms in automaton-driven systems, real-time distributed systems, parallel scientific applications, and transaction-oriented systems.

The virtues of a generalized, message passing foundation for assembling operating system functions are well known as long as these functions do not share common state information and global data. When applied to shared memory, a closely coupled environment, or a single-processor architecture, the challenge is more significant. Techniques for structuring operating system functions, and the data structures they manipulate, can been developed to minimize their interactions, and to optimize message-passing algorithms by taking advantage of the locality of correspondents (i.e., light-weight RPC).

UNIX as a Subsystem.

UNIX can be integrated as a subsystem on top of the Micro-kernel to provide system builders with a standards-based UNIX environment. UNIX services can thus be extended to real time, multiprocessor and distributed environments support, all in a way that remains transparent to standard UNIX application programs.

User Defined Servers.

Such an architecture provides a powerful, convenient platform for operating system development. The homogeneity of server interfaces provided by the low-level IPC allows system builders to develop new servers and integrate them at will into a system, either as user or as system servers. For example, new file management strategies such as real time file systems, or fault-tolerant servers can be developed and tested as a user level utility without disturbing a running system, using the powerful debugging tools available for user-level application development. Later, the server can be migrated easily within the system space for performance consideration.

Conclusion

Some of the critical aspects of new generation operating systems to support such trends as real-time, distribution, multiprocessing, and open systems, include:

- Efficient low-level real time foundations.
- Optimised interprocess and interprocessor communications and user-transparent distribution of resources.
- Portability across hardware architectures from single processor to multiprocessors (loosely or tightly coupled) to advanced distributed environments, from linear to segmented to virtual memory architectures.
- Interoperability of a wide range of operating environments, from real time to fault-tolerant, in a single transparent (secure) distributed environment.
- Scalability and flexibility, in design, configuration and in run-time resource utilization.

- A framework for operating system innovation, development, debugging, maintenance, extension and integration.
- Absolute conformance to industry standards to allow for evolutionary transitions.

Mature Micro-kernel operating system architectures (such as CHORUS), are now available that allow modern operating systems to be built along a modular approach, consistent with the way modern hardware and application environments are being constructed. Moreover, Micro-kernel architectures meet system builders greatest unmet needs: the *"software engineering need"* for operating system architectures in which system components can be developed and assembled in various ways; and the *"distributed systems technology need"*, for a cooperative framework between distributed system components closely interacting through high performance communication media. By insuring complete compatibility with open system standard, Micro-kernel architectures need not affect the application environment and can therefore be gracefully introduced into new system platforms.

References

[1] Sape J. Mullender Ed., *The Amoeba Distributed Operating System : Selected Papers 1984 -1987,* CWI Tract No. 41, Amsterdam, Netherlands, (1987), p. 309.

[2] Marc Rozier, Vadim Abrossimov, François Armand, Ivan Boule, Michel Gien, Marc Guillemont, Frédéric Herrmann, Claude Kaiser, Sylvain Langlois, Pierre Léonard, and Will Neuhauser, ‘‘CHORUS Distributed Operating Systems,’’ *Computing Systems Journal*, vol. 1, no. 4, The Usenix Association, (Dec. 1988), pp. 305-370.

[3] Rick Rashid, ‘‘Threads of a New System,’’ *Unix Review*, (Aug. 1986).

[4] Rob Pike, Dave Presotto, Ken Thompson, and Howard Trickey, ‘‘Designing Plan 9,’’ *Dr. Dobb’s Journal*, vol. 16, no. 1, (January 1991), pp. 49-60.

[5] Paul R. McJones and Garret F. Swart, ‘‘Evolving the UNIX System Interface to Support Multithreaded Programs,’’ Technical Report 21, DEC Systems Research Center, Palo Alto, CA, (Sept. 1988), p. 100.

[6] David Cheriton, ‘‘The V Distributed System,’’ *Communications of the ACM*, vol. 31, no. 3, (Mar. 1988), pp. 314-333.

Key Attributes of Next Generation Operating Systems
Strong structuring concepts, allowing distribution of individual components
Transparent reusability of system components (client-server model, O-O approach)
Portability over full range of machine architectures, preserving real-time performance (embedded systems, mono/multi processors, multicomputers, network architectures)
Support for scalability of system servers at system configuration time
Support for dynamic configuration of system servers into user or system space, according to ease of development, protection or performance criteria.
Support for dynamic configuration of hardware dependent servers (device drivers)
Enablers for transparent distribution of operating system services
Enablers for fault tolerance and replication of system servers and resources
Enablers for secure behavior
Enablers for performance optimization depending on configuration and application requirements
Binary compatibility with Open System standard interfaces