Micro-kernel Architecture
Key to Modern Operating Systems Design

Michel Gien

A recent trend in operating system development consists of structuring the operating system as a modular set of system servers sitting on top of a minimal micro-kernel, rather than using the traditional monolithic structure. This approach promises to help meet systems and platform builders' needs for a sophisticated development environment that can cope with growing complexity, new architectures and changing market conditions. The top priority among these needs is the ability to integrate additional functionality, new hardware technologies and architectures, and distributed environments, all within an ''open systems'' context.

In this type of operating system architecture, the micro-kernel provides system servers with generic services independent of a particular operating system, such as the scheduling of one or more processors and memory management. The micro-kernel also provides a simple interprocess communication facility (IPC) that allows system servers to call each other and exchange data independently, whether they work in a multiprocessor, multicomputer or network configuration.

This combination of elementary services forms a standard base which can support all other system-specific functions. System-specific functions can then be configured into appropriate system servers, managing the other physical and logical resources of a computer system, such as files, devices and high-level communication services. Real-time systems tend to be built along similar lines, with a very simple generic executive supporting specific real-time tasks.

Micro-kernel architecture has been the subject of operating system research for the last ten years, illustrated by such projects as: Amoeba[1] (Free University and Center for Mathematics and Computer Science, Amsterdam); CHORUS[2] (INRIA, France, then developed and commercialized by Chorus Systems); MOS[3] (Hebrew University of Jerusalem); Topaz[4] (DEC/SRC); and the V-system[5] (Stanford University). Perhaps the most visible kernel technology is the Mach[6] operating system, an experimental system under development at Carnegie Mellon University.

Ridge's ROS and Convergent Technology's CTOS are two early proprietary systems that borrowed from micro-kernel design. More recently, the Open Software Foundation and Next, Inc. have adapted concepts from Mach in their offerings. Chorus Systems is the first company to offer a commercial micro-kernel based operating system product family comprising a real-time micro kernel (CHORUS/Nucleus) and a fully distributed implementation of UNIX System V (CHORUS/MiX).

This is a revised version of an article published in UNIX REVIEW, Vol. 8, No. 11, November 1990

Micro-kernel architectures have been, and remain, strongly related to distributed computing. Both research and commercial work listed above 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. Years of engineering work have led to mature techniques for structuring operating system functions, and the data structures they manipulate, to minimize their interactions, and to optimize message-passing algorithms by taking advantage of the locality of correspondents.

CHORUS–v3 represents the fourth generation of such an architecture. It builds upon the CHORUS–v0, CHORUS–v1, and CHORUS–v2 experiences and integrates contributions from the V-system in the area of IPC and RPC mechanisms, Mach for distributed virtual memory architecture and threads, Amoeba for addressing and binding capabilities. Also, at the server level, design of several generations of distributed UNIX servers have been required to mature the technology to a stable state, which can now be widely generalized into a family of operating system products.

This article examines how micro-kernel based operating system architectures can change the way modern systems are designed, outlines systems builders' needs to take advantage of new hardware and machine architecture technologies in the context of open systems standards, and discusses the key characteristics that modern operating systems should exhibit to satisfy those needs.

The Software Engineering Argument. Micro-kernel based architecture is often justified as an approach to reducing the size and complexity of the OS. 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 actually a software engineering argument, which applies to system builders.''

System builders need 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.

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

UNIX Complexity is Increasing. Several trends are pulling UNIX away from its roots. As more functionality is continually demanded of UNIX it is unavoidable that today's 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 UNIX 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.

''UNIX cannot fulfill the expectations of system builders if it continues to increase in complexity, without a well-defined modular architecture to manage this growth.''

UNIX Scope of Application is Being Extended. 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 environments, which have already led to the addition of on-line transaction processing facilities to UNIX. 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.''

Distributed Environments Need to be Supported at All Levels. Adding to the demands on UNIX is the move toward distributed environments. 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 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,
- to incorporate their own value, adding operating system features within an open systems framework,

- to gracefully extend the topology and functionality of their system into cooperative computing environments.

Introducing an Operating System Development Environment. In order to develop such platforms, system builders need to be provided with an *operating system development environment* as powerful as the development environments which are available to application developers. They need to be able to develop more and more complex systems as easily as applications can be developed.

Micro-kernel architectures meet these needs by providing a *standard, generic minimum base* that provides enabling system services. System builders can then concentrate on the creation of specific, innovative, system components supported by the standard base.

System builders need to be able to add value not only by means of the hardware organization, but also in the operating systems' combination of services and facilities. A monolithic, fully-packaged operating system structure, as provided by today's UNIX implementations, does not furnish this capability.

The system must also provide enhanced portability over a wide range of scalable architectures allowing the system builder to track the evolution of hardware technologies to achieve the best fit with specific performance requirements.

Portability of the same system base to a wide range of architectures also provides system builders with a homogeneous operating system environment over the entire product range, thus providing economies of scale as well as reusability of system components. UNIX provided the first steps in this direction. The need now is to push this further by insuring that the standard operating system base is able to be:
- scaled down to real-time embedded systems,
- scaled up to multi-processors and parallel architectures,
- scaled laterally to specialized network nodes in fully distributed environments.

A Mature Micro-kernel Based Operating System Architecture. The real design issue in micro-kernel architectures is the definition of the services that need to appear at the micro-kernel interface. This interface must represent a good balance between fully supporting those functions that are key to all systems and enabling customization in order to adapt the system to the requirements of specific application areas and to take advantage of particular machine architectures.

Over the last several years this balance has been achieved. The original message-based micro-kernels that were initially developed to support distributed, specialized systems have since been enhanced or totally redesigned to comply with ever more general system needs. Key design criteria are now well understood and a well engineered set of enablers has been worked out. Mature micro-kernel-based operating systems are indeed feasible. The following features characterize mature designs.

Strong Structuring Concepts. Modularity of design and operational configurations requires strong structuring concepts such as the actors/threads in CHORUS, or the tasks/threads in Mach, as well as clear isolation of interactions provided through message passing-based communications, as in the V-system, Amoeba, and CHORUS.

Transparent Reusability of System Components. Service encapsulation provided by the client-server based model of interactions allows system servers supporting this model to be used in different operating system environments, as is done by Amoeba or CHORUS servers.

Portability Over a Full Range of Computer Architectures. Portability requires the ability to support various types of multiprocessor architectures, as is done in Topaz, Mach and CHORUS, but also the ability to adapt to embedded systems or scientific computers with no virtual memory capabilities, as with CHORUS.

Support for Scalability. Scalability is necessary for system services as well as at the hardware level. Threads, as in Topaz, Mach or CHORUS for example, can be used to scale request handling in servers by adding threads to handle more requests, as well as to optimize performance (using parallelism on symmetric multiprocessor configurations). Operating system scalability should match hardware scalability as examplified by X-terminals equipped with minimal CHORUS/Mix systems. These provide only the UNIX interface systems calls necessary to an X-server and some local X-clients without requiring a full UNIX kernel.

Support for Server Development. Development and debugging of system servers should be done at the user level, as in V, Amoeba, Mach or CHORUS. In mature designs, servers can also dynamically be moved into system space for better performance of operational configurations, as in CHORUS.

Support for Structured Integration of Hardware Specific Capabilities such as Device Drivers. Mature micro-kernel designs define clean ways for device drivers or interrupt/trap handlers to get easy access to hardware engines without being integrated into the micro-kernel itself. As an example, CHORUS' ''privileged actors'' and ''supervisor threads'' provide such capabilities.

Enablers for Transparent Distribution of Operating System Services. Message passing micro-kernels, such as the V, Amoeba, CHORUS and Mach micro-kernels, provide location transparent interprocess communication facilities (and RPC services), logical addressing and naming, that enable distribution transparency.

Enablers for Fault Tolerance, Reconfiguration and Duplication of System Resources. Dynamic binding facilities, such as CHORUS ''Port Groups'', provide powerful tools for building dynamically reconfigurable operating system services, by providing an extra level of indirection for addressing a server. As a result, clients actually address a service rather than a particular server. Such facilities also provide support for building redundancy and transparency in distributing operating system services.

Enablers for Security Features. Security enforcement can take advantage of modularity and encapsulation of resource management on specific servers. These servers can handle their own required authentication procedures. Security may be costly in performance and therefore should be provided at the appropriate levels according to each system's requirements. Examples of security enablers can be found in Amoeba's and CHORUS' object naming capabilities.

Enablers for Performance Optimization. The operating system should let system builders choose their own trade-offs to optimize system performance, particularly when this involves juggling conflicting requirements. System builders should also be provided with the ability to optimize system performance at configuration time rather than during programming where the emphasis is on easing the debugging environment. This is of particular importance in real time applications.

Binary Compatibility with Existing Standard Interfaces, such as UNIX. Mach and V-system's ''transparent-shared libraries'' as well as CHORUS' ''Process Managers'' provide for such emulations.

The combination of these features which can now be found in a mature micro-kernel based operating system design, provides system and platform builders with the same type of environment that application builders have enjoyed. They allow them to now build complex systems faster by taking advantage of the availability of a portable, ‘‘standard’’, and qualified base. Equally important, well engineered hooks and services enable system and platform builders to easily add their own specific value and to configure members of a product family to meet specific performance requirements.

Modern Operating Systems. Modern design, engineering, and manufacturing techniques are speeding up the pace at which new hardware technologies and machine architectures become available. Simultaneously, demand for enlarging the scope of application of computing systems is steadily increasing. Thanks to the emergence of open system standards and the availability of powerful application development environments, application builders are increasingly able to create and deliver new products. System builders, however, seem curiously left out of the picture, perhaps because until recently they have often been buried inside hardware-oriented system manufacturers or application-oriented system integrators.

Operating system development lags significantly behind hardware development and is often the bottleneck in system/platform time-to-market. And the evolution of traditional operating systems, built along a monolithic architecture, faces an increasingly evident complexity barrier. Monolithic operating systems do not and cannot provide the tools system builders now need to cope with the rate at which complexity is growing, new architectures are being developed, and market conditions are changing.

Mature micro-kernel operating system architectures 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, microkernel 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.

Microkernel architectures indeed provide a sound foundation for meeting modern operating systems needs, while maintaining the best of UNIX’s heritage in this new generation of cooperative computing environments.

Author’s Biography

Michel Gien is co-founder, general manager and director of R&D at Chorus Systems. He joined the Cyclades computer network team at INRIA in 1971 after graduating from Ecole Centrale des Arts et Manufactures de Paris. He was responsible for Francés participation in the European Informatics Network Projects, which was designed to link computer research centers in Europe and became a major contributor in the early ISO/OSI standardization efforts. He then led a project that introduced UNIX in France and helped to understand how it could be re-architectured along the Chorus distributed systems concepts. Michel Gien is a leading figure within the European UNIX community. He was recently named chairman of EurOpen, the European Forum for Open Systems (formerly EUUG) after serving as vice chairman since 1985.

References

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

[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] Amon Barak and Ami Litman, ''MOS : A Multicomputer Distributed Operating System,'' *Software Practice & Experience*, vol. 15, no. 8, (Aug. 1985), pp. 725-737.

[4] 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), 100 p.

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

[6] Rick Rashid, ''Threads of a New System,'' *Unix Review*, (Aug. 1986).

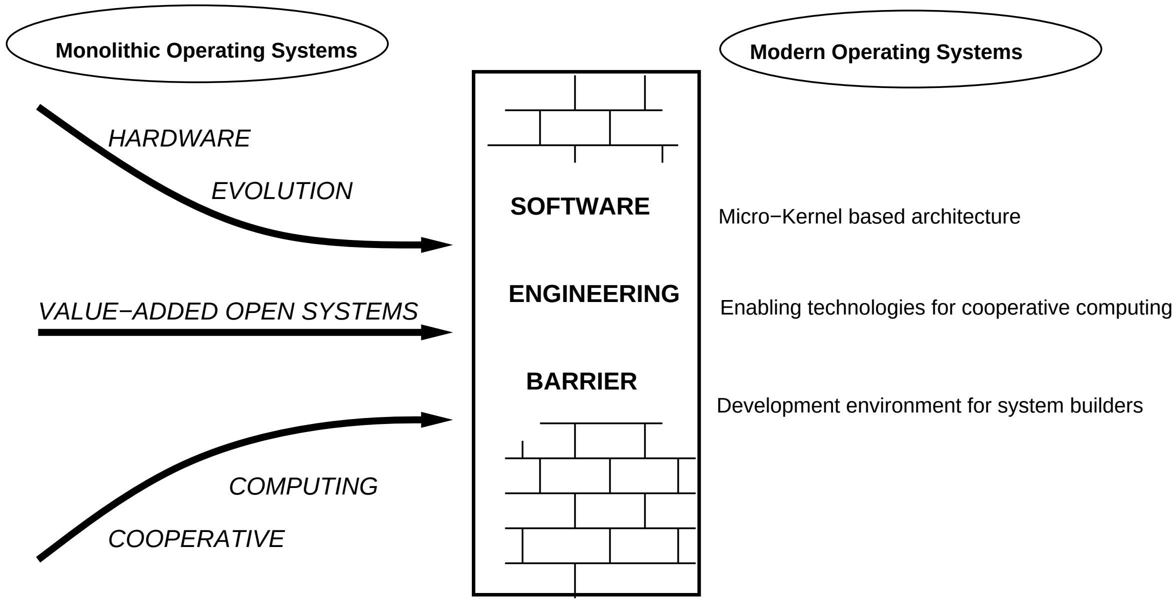

Figure 1. – The Software Engineering Argument for Micro-Kernel Based Operating Systems

---

CHORUS is a registered trademark of Chorus Systems
UNIX is a registered trademark of AT&T

Figure 2. – Key Attributes of a Modern Operating System

Key Attributes of mature micro-kernel based operating systems
Strong structuring concepts, allowing distribution of individual components
Transparent reusability of system components (client-server model)
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
Support for dynamic configuration of hardware dependent servers (device drivers)
Enablers for transparent distribution of operating system services
Enablers for fault tolerance and duplication 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

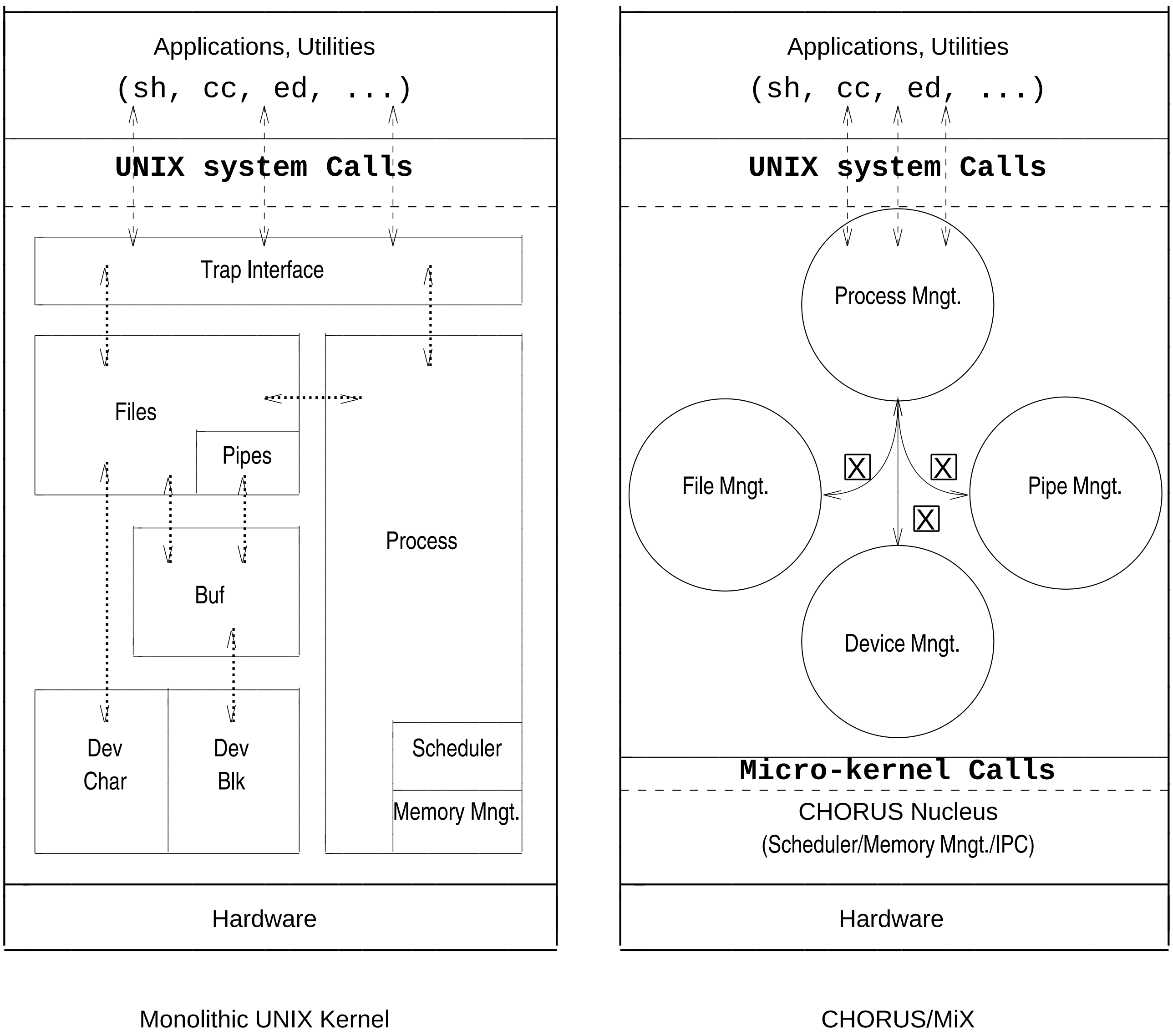

Figure 3. – Micro Kernel Architecture Applied to the UNIX Kernel

Micro-kernel operating systems provide a more structured architecture than conventional, monolithic UNIX kernels. When the micro-kernel architecture is applied to UNIX, a small, generic micro-kernel, such as the CHORUS/Nucleus provides support for basic operations such as processor real-time scheduling, (virtual) memory management and localisation transparent IPC between servers that implement more complex operating system-dependent functions. UNIX system calls are made to these servers through the Process Manager which transforms them into (Remote) Procedure Calls.

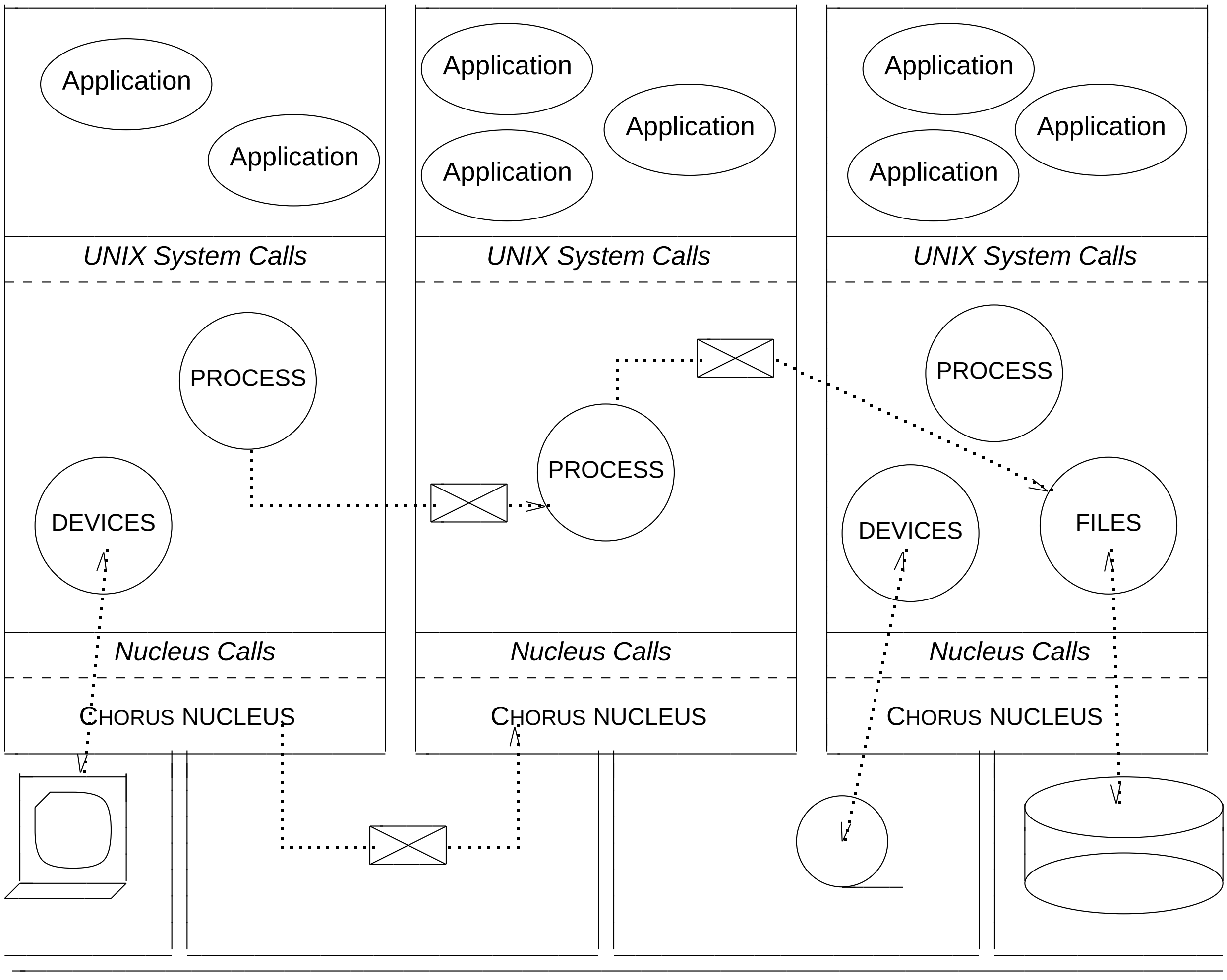

Figure 4. – Cooperative UNIX Operating System

There are four key points to an effective micro-kernel based operating system implementation:

[1] The definition of the services that need to appear at the micro-kernel interface is a real design issue. It must represent a good balance between fully supporting those functions that are key to all systems, and a well engineered set of enablers to facilitate customization of specific application areas to take advantage of particular machine architectures.

[2] Efficient message passing is another key to the micro-kernel architecture's ability to deliver high performance as well as distribute functions over communications media from shared memory to a wide area network.

[3] Correct structuring of higher level operating system services into system servers according to a Client-Server based model. Special care is indeed necessary to split the various system data structures on which they operate in order to optimize overall performance by minimizing interactions.

[4] Providing binary compatibility with standard Open Systems interfaces is required to insure complete portability of existing applications while providing a path to further system interface extensions.