Micro-kernel Based Operating Systems:

Moving UNIX onto Modern System Architectures

Michel Gien, Lori Grob

Chorus systèmes
6, avenue Gustave Eiffel,
F-78182, Saint-Quentin-en-Yvelines
(France)

High-end Computer Architectures

characterized by multi-processor and distributed memory,

High-volume, low-end systems are driving the pricing and gathering most of the attention of much of the industry, but the high-end systems are driving the technology innovations since the low-end systems are built from standard components. Those innovations in the high-end work their way down to the low end, just as multiprocessing, graphics, etc. have moved down to the PCs. Having a successful strategy in the high end is important for ensuring a continuing technology flow into the medium and low end systems.

Many high-end open systems under development today are characterized by the innovative use of multiple processors in distributed memory configurations ("multicomputers"). These parallel processors provide wider I/O throughput for mainframe-power UNIX systems, redundancy for reliable OLTP systems and enormous computation power for massively parallel supercomputer systems.

need a powerful operating system development environment,

To build these systems, system builders need tools and services that will let them more easily integrate complex hardware architectures into high performance, reliable, distributed environments. To master these more complicated environments and get these more complicated machines to market faster, system builders need an operating system development environment that is as powerful as the development environment provided to applications builders by the UNIX system.

The emphasis for such a development environment should be on two critical areas. First is enabling systems services which let the OS engineer focus on the specific high valued added features of the system and on the most efficient and flexible implementation of those features and not on how to shoe-horn those

---

To appear in the Proceedings of the UniForum'92 Conference, San Francisco, USA, January 22-24, 1992.

CHORUS is a registered trademark of Chorus systèmes
UNIX is a registered trademark of UNIX Systems Laboratories

features into the existing code base. System builders should be able to add value in several ways, such as adding TP, DB and performance monitors to the operating system, supporting multicomputer architectures, improving reliability, high availability and fault tolerance, increasing performance, or extending levels of security.

Transparent re-usability of system components, modularity and scalability of system services, structured integration of device drivers and specific hardware features, are key characteristics of such a powerful operating system development environment. A monolithic operating system such as todays' UNIX implementation does not offer as much flexibility, resulting in greater development cost and later time to market for systems.

enhanced scalability and portability,

Second is scalability and portability, which lets developers build ranges of products where optimization is possible because of the homogeneity of the operating system base. UNIX took the first step in this direction (portability between microprocessors), but now there is a need to go further to be certain the system can be scaled down to real-time embedded systems and scaled up to suit multiprocessor, parallel supercomputer, reliable OLTP, or network systems.

while still fully standards compliant...

Modern operating systems have to meet certain design objectives to meet these requirements and support the inherent distributed environment of high-end systems. And they must do this while providing complete BCS/ABI compatibility with UNIX interfaces and Open Systems standards.

Key Attributes of modern operating systems for high-end computing
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

Micro-kernel Based Operating System

One approach to modern operating system design, which is becoming popular, is to build the distributed operating system as a set of independent system servers using the primitive, generic services of a micro-kernel. The micro-kernel provides a virtual machine for processor use, memory allocation and communication between operating system components. This approach has been used in several key projects such as CHORUS[1] (researched at INRIA, France, then developed and commercialized by Chorus Systems), Amoeba[2] (Free University and Center for Mathematics and Computer Science, Amsterdam), MOS[3] (Hebrew University of Jerusalem), Topaz[4] (DEC/SRC), the V-system[5] (Stanford University). the V-system, and is being looked at for future evolutions of the Mach system (Carnegie Mellon University). [6]

With distributed computing built into their virtual machine layer, micro-kernel architectures provide a means of handling modern systems complexity in a structured design manner. Systems can be designed better and, as a result, their increasing complexity is easier to accommodate and understand. Micro-kernel designs are, therefore, well suited to the requirements of building systems for distributed environments as used in modern high-end computer architectures.

Compared with traditional operating systems, the micro-kernel approach adds two new aspects to the low-level kernel foundation: distribution and subsystem support. In other words, this technology adds to the traditional monolithic architectures the necessary modularity, key to their evolution, introducing the object oriented approach to operating system design.

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.

Challenges for commercial micro-kernel based 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 (Light-Weight RPC as in[7]).

CHORUS

CHORUS-v3 represents the fourth generation of such an architecture. It builds upon the CHORUS−v0, CHORUS−v1, and CHORUS−v2 experiences [8] 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.

The CHORUS product line includes the CHORUS Nucleus, a micro-kernel for core operating system services, and CHORUS/MiX, a binary compatible, multi-server UNIX System V implementation. Some of the critical aspects of a distributed operating system for high end architectures − to support such important trends as distribution, multiprocessing, and open systems − illustrated by CHORUS include:

- Efficient low-level real time foundations.
- Optimized 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 systems, from real time to fault-tolerant, in a single distributed environment.
- Scalability and flexibility, in design, configuration and in run-time resource utilization.
- A framework for operating system innovation, integration, development, testing, debugging, maintenance and extension.
- Absolute conformance to industry standards.

Basic structure − Nucleus and Subsystems

The CHORUS architecture is based on a minimal real time distributed *Nucleus* that integrates distributed processing and communication at the lowest level. The Nucleus provides generic tools − thread scheduling, real time event handling, network-transparent inter-process communications (IPC) and virtual memory management − to sets of independent servers called *subsystems*, which coexist on top of the Nucleus.

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.

The CHORUS organization of Nucleus 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.

The Nucleus

The CHORUS Nucleus provides both local and global management of services (Figure 1). At the lowest level, it manages the physical resources at each “site” with four clearly defined components:

- a powerful *multi-tasking executive* which controls allocation of local processors, manages priority-based preemptive scheduling of CHORUS *threads*, and provides primitives for fine grain synchronization of, and low-level communication between, threads;
- a *distributed memory manager* which can support the full range of memory architectures – linear, segmented or virtual;
- a low level hardware *supervisor* which dynamically dispatches external events such as interrupts, traps and exceptions to dynamically defined routines or ports;
- an *Inter Process Communication* (IPC) manager provides the high-performance global communication services (exchange of *messages* and *Remote Procedure Calls* through *ports*) which are the key to CHORUS architecture's transparent support of services distribution and dynamic reconfiguration capabilities.

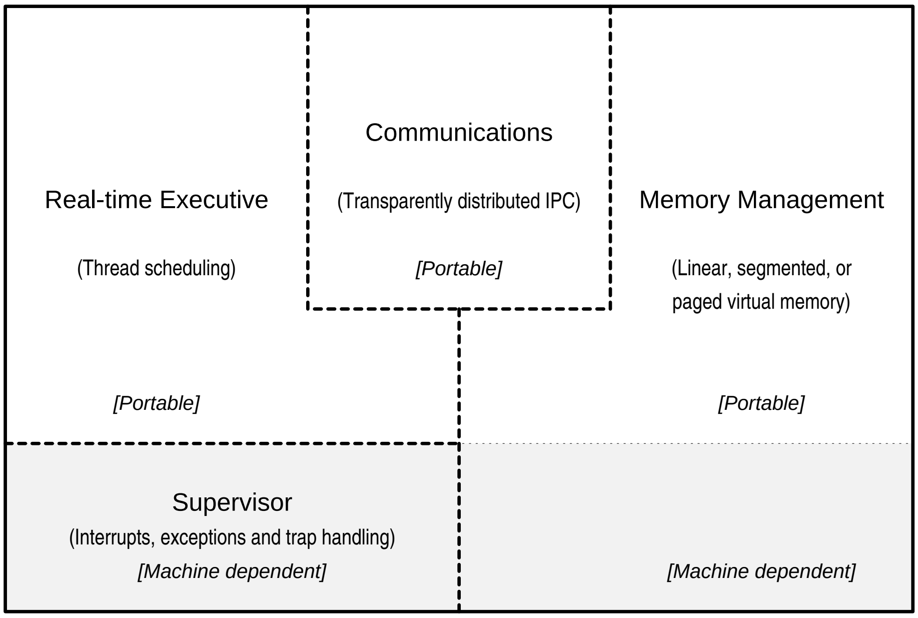


Figure 1. – The CHORUS Nucleus

The CHORUS Subsystems concept

Subsystems in CHORUS architecture are sets of system servers that use the generic services of the Nucleus to provide higher-level services. More simply, a subsystem is an operating system built on top of the CHORUS Nucleus.

Subsystems manage physical and logical resources such as files, devices and high-level communication services and they communicate via the *IPC* facility provided by the Nucleus. Their position "on top" of the Nucleus provides a structured, well-defined way for system builders to cope with complexities of operating system development.

In the operating system context, a subsystem offers the means by which to supply a complete standard operating system interface to standard application programs (Figure 2). New servers can then be added to the set of servers delivering this interface as a means of gracefully extending system and operating system capability. *Servers* are the building block, the toolset, that system builders 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.

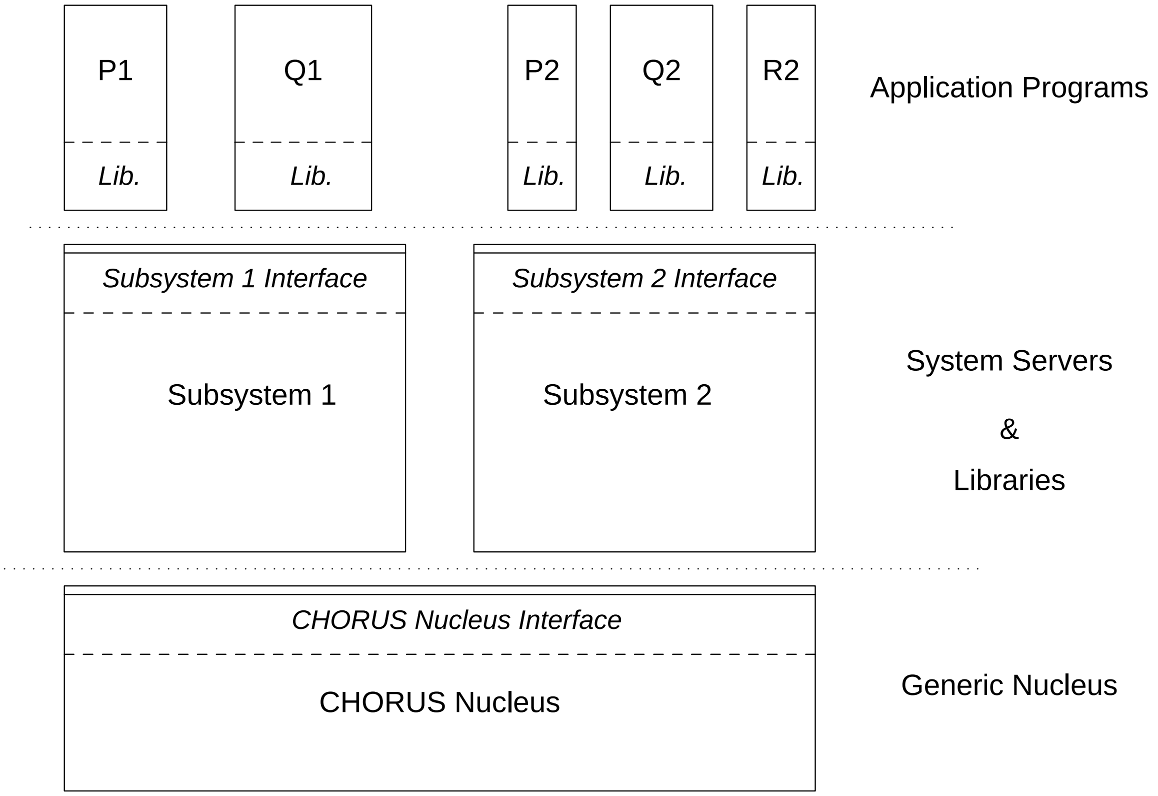


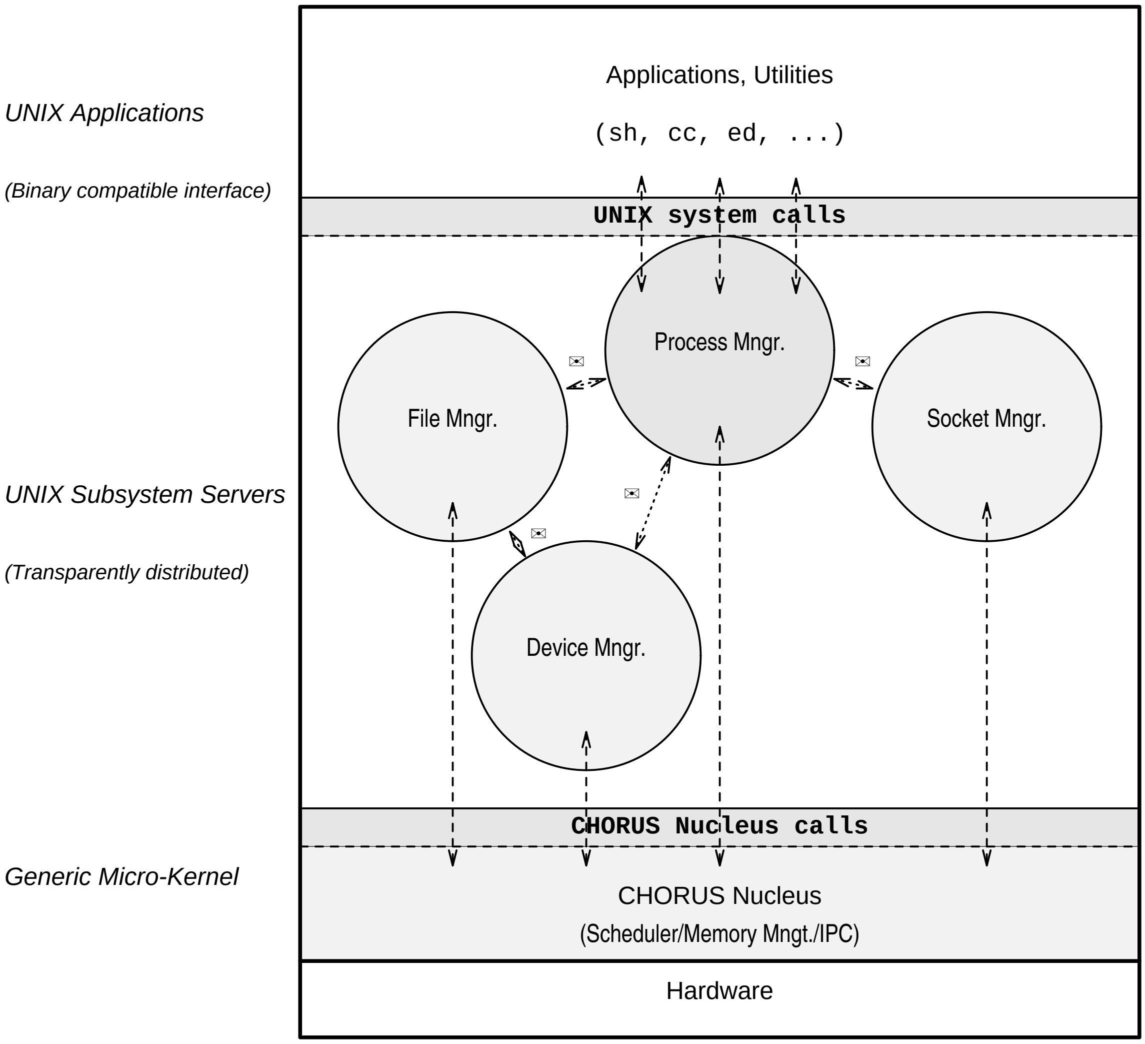
Figure 2. CHORUS/Nucleus supporting different operating systems interfaces

The CHORUS/MiX Subsystem

CHORUS/MiX integrates a UNIX System V subsystem with the CHORUS Nucleus to provide system builders with a standards-based, real time, transparently distributed UNIX environment (Figure 4). CHORUS/MiX UNIX services have been extended to real time and to the distributed environment (distribution of programs as well as files), all in a way that is completely transparent to UNIX application programs. CHORUS/MiX is compatible at *the executable binary code level* with System V 3.2. A first version of a fully SVR4 compliant CHORUS/MiX implementation has just been released.

CHORUS/MiX offers the traditional set of UNIX functions for managing signals, but extends them to manage real time, multi-threaded and distributed processes. This extension permits the creation – and manipulation from a distance – of processes on any machine, while respecting rules of UNIX regarding environments, open files and so on.

Finally, because CHORUS/MiX is fully compatible with – and actually *contains* UNIX System V source code, it allows UNIX processes to take advantage of such CHORUS facilities as multi-threading, location transparent IPC, virtual memory management, device drivers and servers development in user space and more.



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 location 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 3. – CHORUS/MiX: Micro-kernel based multi-server UNIX system

CHORUS/MiX support of distributed architectures

The modular design of CHORUS/MiX allows binary compatibility with UNIX, while maintaining a well-structured, portable and efficient implementation. It also includes several significant extensions which offer, at the UNIX subsystem level, access to CHORUS facilities.

Powerful real time facilities available at the UNIX subsystem level make it possible to dynamically connect handlers to hardware interrupts. UNIX processes enjoy the benefit of priority-based preemptive scheduling provided by the CHORUS Nucleus, possibly at a higher priority than the CHORUS/MiX subsystem servers themselves.

UNIX processes running on CHORUS can communicate with other UNIX processes, bare CHORUS processes or entities from other subsystems using the CHORUS IPC mechanisms. Processes, for example, are able to create and manipulate CHORUS ports; send and receive messages; and issue remote procedure calls.

User defined servers.

The CHORUS architecture provides a powerful, convenient platform for operating system development. The homogeneity of server interfaces provided by the CHORUS 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 (See Figure 4). Later, the server can be migrated easily within the system for performance consideration.

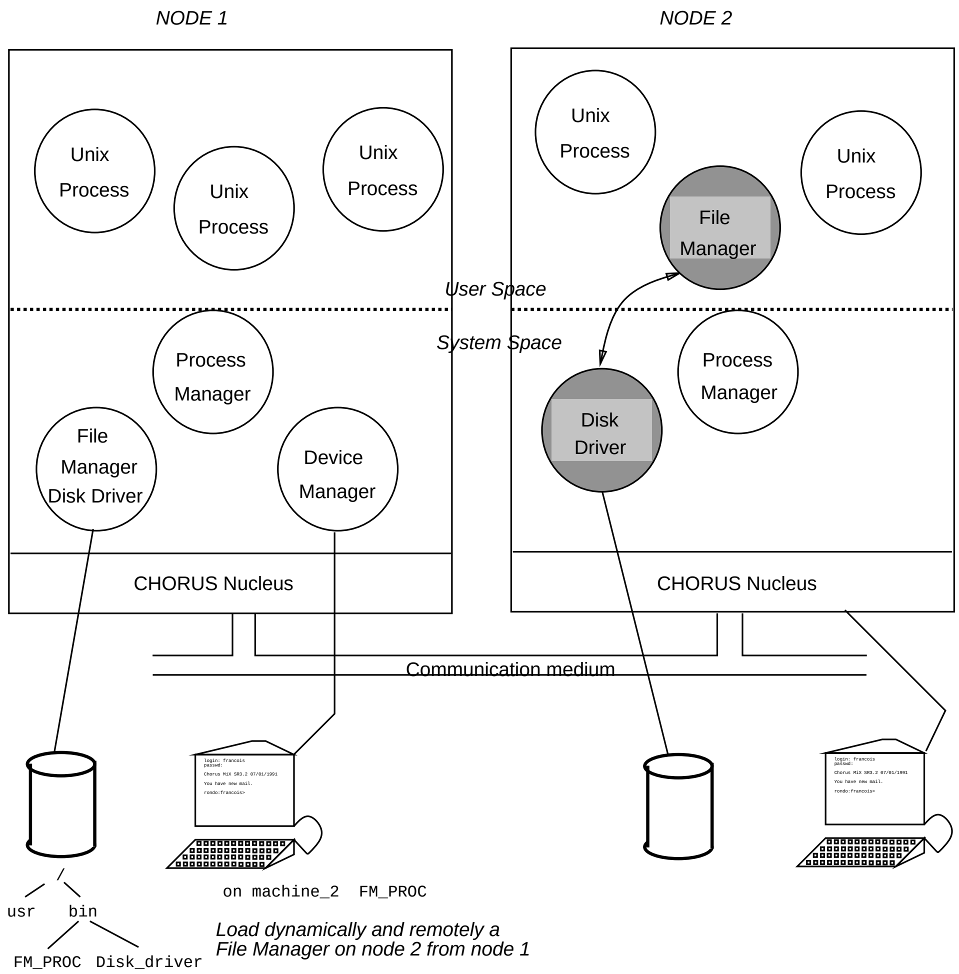


Figure 4. UNIX servers in user space

Modularity.

CHORUS incorporates both distribution support concerns at the lowest levels in a manner consistent with emerging computer and communications requirements. Further, modularity extends from the CHORUS Nucleus to higher levels and in between. The result is a set of well-engineered *enablers* for the design, implementation and operation of next-generation systems.

Because CHORUS uses modular, independent servers, to divide the operating system into manageable units, along an object-oriented approach, debugging and testing can be handled more efficiently. The clear, message-based interfaces between server modules permit greater independence, which simplifies and speeds integration. The simpler micro-kernel interfaces can also be used to build higher level semantics.

A fully distributed operating system configuration.

As shown on Figure 5, a high-end distributed memory configuration (or a LAN based distributed computing environment) will be able to take advantage of such a modular distributed operating system implementation by transparently distributing the UNIX servers among the various nodes of the distributed configuration according to tradeoffs decided upon by the system builder (or system adminisatrator, as opposed to the operating system designer). Shared memory multi-processors can be transparently integrated into the distributed configuration as well (as shown on the Figure).

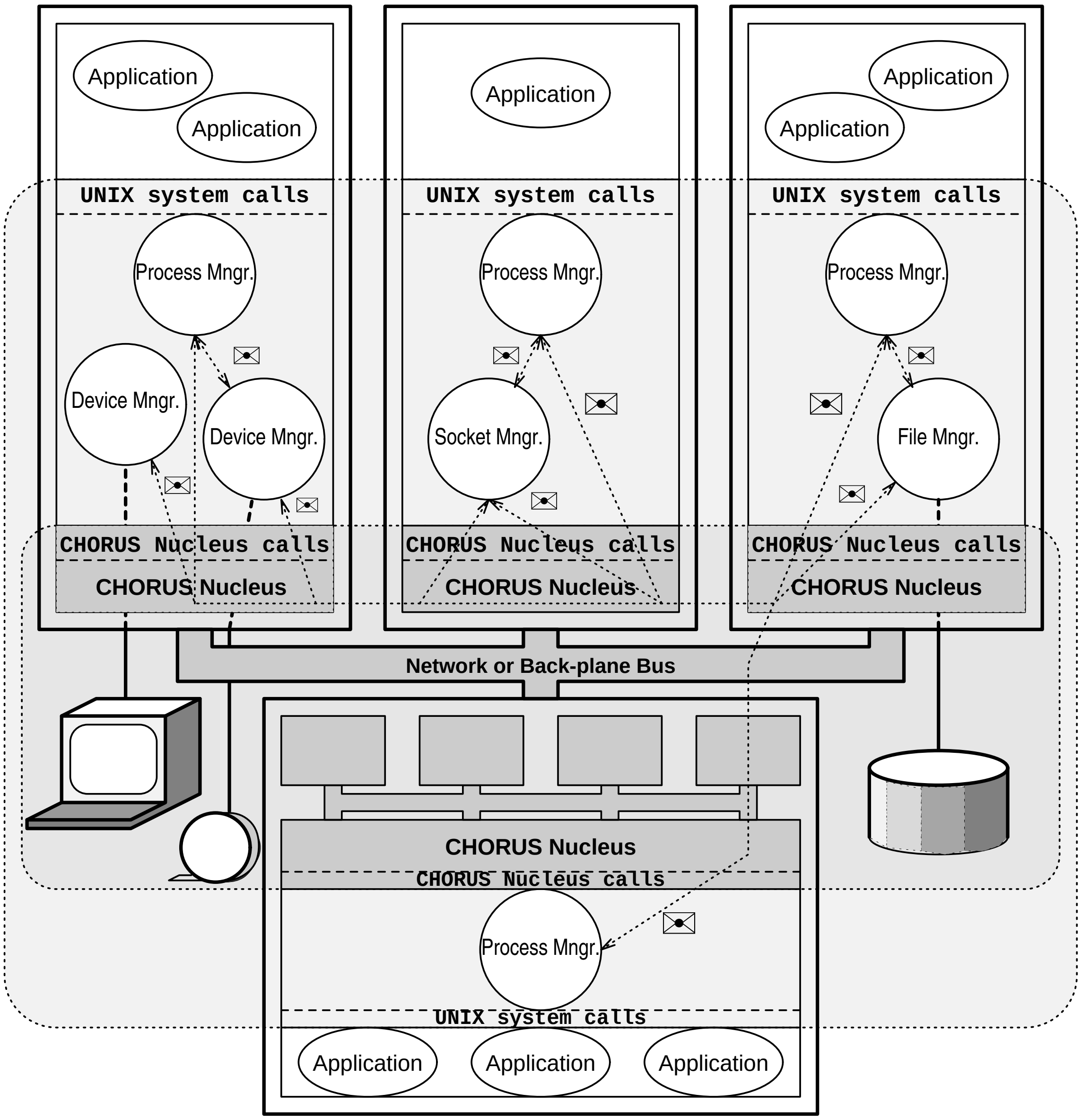
Replication of system servers can be envisaged for fault tolerance purposes, thus moving open systems forward into supporting not only hardware but also system software failures transparently for the application program.

Application level distributing computing environments (as well as object-oriented platforms) can also take advantage of such an operating system structure to exhibit higher performance characteristics by making direct use of the efficient underlying IPC facilities rather than user level networking protocols.

Conclusion

Mature micro-kernel based distributed 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, 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.



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 customisation 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 optimise overall performance by minimising 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.

Figure 5. – CHORUS/MiX: A Fully Distributed Micro-kernel based UNIX Operating System

References

[1] 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.

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

[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), p. 100.

[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).

[7] Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy, ‘‘Lightweight Remote Procedure Call,’’ *ACM Transactions on Computer Systems*, vol. 8, no. 1, (February 1990), pp. 37-55.

[8] Allan Bricker, Michel Gien, Marc Guillemont, Jim Lipkis, Douglas Orr, and Marc Rozier, ‘‘A New Look at Microkernel-Based UNIX Operating Systems: Lessons in Performance and Compatibility,’’ in *Proc. of EurOpen Spring 1991 Conference*, EurOpen, Tromso, Norway, (20-24 May, 1991), pp. 13-32.

Authors’s Biography

Michel Gien *is co-founder, General Manager and Director of R&D at Chorus systèmes, which was created at the end of 1986. He joined the Cyclades computer network team at INRIA in 1971 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 is chairman of EurOpen, the European Forum for Open Systems (formerly EUUG).

Lori Grob *is a senior system engineer, Manager of Technical Support at Chorus systèmes. From 1983 until 1985, she was an Adjunct Faculty Member and an Instructor at NYU Courant Institute of Mathmatical Sciences. In 1984, she joined the NYU Ultracomputer Research Project as a research scientist, working on their symmetric, scalable unix kernel and parallel programming support for scientific applications. She joined Chorus systèmes in 1989, as a member of the development group.*
Lori Grob was the co-Chair of the 1988 Usenix workshop on Unix and Supercompters and the technical program chair of the 1991 Winter Usenix Technical Conference. She is the author of several publications and has been on more program committees than she cares to think about.