Real Time in a Distributed Computing Environment

Marc Guillemont

distribution: Larry Lettieri (Regis McKenna)
Gérard Yon

summary: Article for *Computer Technology Review*

© Chorus systèmes, 1990

Real Time in a Distributed Computing Environment

Marc Guillemont, Chorus systèmes

A real-time distributed computing environment places different demands on today's operating system software than do non distributed environments. This means it is necessary to look at the operating system in different way. First, requirements put on the different machines in a distributed environment may vary from multipurpose operator workstations or PCs, to dedicated servers or rapid-response, real-time environments; the traditional boundary between "host workstation" and "target board" with incompatible interfaces is out of date. A highly modular operating system with built-in support for efficient cooperative computing may appear to be a solid basis to support these various requirements uniformly. Second, a distributed environment has, by nature, a dynamically changing configuration: it makes sense for the operating system to provide facilities for flexible and dynamic application configuration and reconfiguration. When real-time response requirements are added to the distributed equation, the problem of providing services to users becomes greater still.

Operating system have evolved towards real-time distributed environments along two paths. On the one hand, real-time executives, originally small and efficient, have been progressively overloaded with additional services, in particular those needed to reduce the gap towards UNIX®, nothing in the original system being prepared to support this addition. On the other hand, UNIX has been disemboweled to extend its use to real-time, despite of its unsuitability for such applications.

The main obstacle to further evolution is rooted deeply in operating system technology. Since the beginning, operating systems have been built as monoliths, and often dedicated to a single hardware architecture. UNIX has marked an important step in operating system evolution by being portable. The new real-time, distributed computing environments demand another evolutionary step – a *new technology* for building operating systems.

This article describes the architecture and features of CHORUS®, a UNIX-compatible operating system designed for real-time distributed environments, and how that architecture is suited for such environments. CHORUS provides support for real-time applications with true real-time performances and features. Also, CHORUS provides support for distributed applications, enabling applications to take easily full advantage of the benefits of distribution (e.g. transparent localisation, dynamic reconfiguration, etc.). And, finally, CHORUS may be complemented by modular subsystems which can provide to applications the whole set of high-level services of traditional operating systems, such as UNIX.

Traditional monolithic operating system become too complex [1]

® UNIX is a registered trademark of UNIX System Laboratories, Inc. in the U.S.A. and other countries.

® CHORUS is a registered trademark of Chorus systèmes.

1. In my original text, I did distinguish major section titles characterised by larger letter titles and a large letter to begin the text, and minor sections. I thought it suggested a structure in the text which could help its reading.

   Here, I have unified the title sizes, but kept the large initial letters.

© Chorus systèmes, 1990

Traditional operating systems, real-time as well as UNIX-like, are built as a single piece of code where potentially, and in fact too often, every module may interact with any other and get access to shared global data. Operating systems have to evolve to provide new services to their applications, to be adapted to new hardware architectures or to support new classes of applications. Each evolutionary step aimed at solving all performance or side-effect issues affects a large part of the operating system. As the operating system grows, mastering the evolution becomes a problem whose complexity begins to exceed the ability of software engineering to handle it.

Adding services is difficult

As the services provided increase, real-time executives tend towards a catalogue of unrelated and heterogeneous features, adding complexity to the system and making its features difficult to use. In the same way, several trends are pulling UNIX away from its roots. As more functions has been demanded of UNIX it is unavoidable that today's versions should be more complex. Unfortunately, a byproduct of this increasing complexity is the gradual undermining of the original benefits of UNIX original benefits: simplicity and portability.

[Here your text contains a duplicated section which should be removed.]

Adaptation to new hardware architectures is a challenge

Real-time executives have traditionally been built with one objective in mind: performance. Consequently, every hardware feature has been exploited to squeeze one more nano-second of speed, the result being a very non-portable [word missing?]. The stability of interfaces between various implementations on different hardware platforms simplifies the programmer's job, but does not help porting the operating system to new architectures such as multi-processors, or hypercubes.

Instead of evolving towards greater portability, new UNIX releases require increasing effort to implement on new platforms, particularly when they are built around emerging hardware architectures which UNIX was not initially designed to support.

Distributed environments need to be supported at all levels.

Adding to other demands is the move toward distributed environments. Today's computing environments require that new hardware and software resources, such as specialised 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 critical as collective computing environments emerge to better map natural human organisation of systems.

The scope of applications is extending

The open systems standard that UNIX represents is naturally appealing to application domains that have been up to now dominated by highly specialised 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 customise it to vertical applications.

[I had cancelled the above section in my text, because I found it too much UNIX-oriented, whereas the article should be emphasised towards real-time.]

Real-time systems builders want the capabilities of real-time executives to be extended to support UNIX applications. Conversely, UNIX is being extended to support the execution of some categories of real-time 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 in which they don't often fit well.

The micro-kernel based view of operating system

A recent trend in operating system development is structuring the operating system as a modular set of system servers sitting on top of a minimal µ-kernel, rather than using the traditional monolithic structure. This approach promises to help meet systems and platform builders' needs to support sophisticated and varied environments and that can cope with growing complexity and new architectures. The high priority among these needs is the ability to integrate real-time applications, with new hardware technologies and distributed environments, all within an "open systems" context.

The µ-kernel provides generic services

In this operating system architecture, the µ-kernel provides system servers with generic services independent of a particular operating system, such as (real-time) scheduling of one or more processors and memory management. The µ-kernel also provides a simple Interprocess Communication facility (IPC) that allows system servers to call each other and exchange data independently of where they are executed in a multiprocessor, multicomputer or network configuration.

This combination of elementary services forms a standard base which can support all other system-specific functions. These 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.

Compared with traditional real-time executives, the µ-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, the key to their evolution.

IPC as a building technology

[Your title, as I understand it, suggests that IPC is a technology used to build the µ-kernel itself, which is not true; IPC is *provided* by the µ-kernel and used in order *to build the subsystems*.]

µ-kernel architecture has been, and remains, strongly related to real-time distributed computing. Message passing, which has come to be one of the often distinguishing characteristics of µ-kernel architecture, is a natural way to structure systems components distributed over a loosely-coupled set of individual processors, boards or complete machines. Message passing enforces very clear isolation between each individual component of the system, by making explicit the communications rules used between them, while 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 and 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 automation-driven systems, parallel scientific applications and transaction-oriented systems.

Challenges for commercial micro-kernel based systems

Traditionally, there have been two major challenges to developing commercial µ-kernel systems. First, the virtues of a generalised, 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. Second, when applied to shared memory, a closely coupled environment, or a single-processor architecture, the challenge is more significant. But, years of engineering work at Chorus systèmes [2] have led to now mature techniques for structuring operating system functions and the data structures they manipulate, to minimise their interactions, and to optimise message-passing algorithms by taking advantage of the locality of correspondents.

Modularity, key of the evolution

µ-kernel technology adds a natural modularity to an operating system architecture. Services are provided by independent servers to which any machine can get access, as a result of the power of the underlying IPC. Distribution, therefore, is a natural quality of the subsystem built on top of the µ-kernel.

In contrast [3], servers can be configured on the different machines according to specific requirements, allowing the varying requirements to be met with a single set of servers. The evolution of a server or the development of new servers is largely facilitated by the clear interfaces between components and the absence of side-effects between them.

Adaptation to new hardware architectures is left to the µ-kernel which is the most hardware dependent part of the overall system; modern µ-kernel, developed in high-level languages with efficient and clear structures, make this task simple.

CHORUS

Unlike traditional operating systems, the CHORUS product line is developed around this µ-kernel technology. CHORUS treats system communications as a fundamental resource which must therefore be managed by the operating system. As a result, the functions needed to implement distribution and multiprocessing, which are the major trends in hardware architecture today, are the very foundation of the CHORUS line.

CHORUS's µ-kernel-based architecture is also designed with the view that next-generation systems will be open systems. As a result, CHORUS has many key features which are critical in building the new systems that will launch tomorrow's cooperative computing environment. These 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.

---

2. Chorus systèmes has a grave accent on the "e".
3. Is it the right term? This section is not in contradiction with the previous one, but more its consequence.

- 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 utilisation.
- A well-engineered framework for operating system innovation, integration, development, testing, debugging, maintenance and extension.
- Absolute conformance to industry standards.

The CHORUS product line includes the CHORUS Nucleus, a well structured µ-kernel for core operating system services, and CHORUS/MiX, a binary compatible, multi-server UNIX System V implementation. By conforming to open system standards, the CHORUS line ensures software compatibility in the face of hardware evolution and can be gracefully introduced into either new or existing systems platforms. With real-time, distributed technology as its base, CHORUS is the first µ-kernel-based operating system line that achieves *performance equivalent to or greater than* that of traditional monolithic operating systems while facilitating the creation of specific, innovative, value-added operating system components.

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 organisation of Nucleus and subsystems (see Figure 1.) 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 µ-kernel as its foundation.

The Nucleus

The CHORUS Nucleus (see Figure 2.) provides both local and global management of services. At the lowest level, it manages the physical resources at each "site" with four clearly defined components:

- a classical **real-time multi-tasking executive** which controls allocation of local processors, manages priority-based preemptive scheduling of CHORUS *threads*, and provides primitives for fine grain synchronisation 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* through *ports*) which are the key to CHORUS architecture's real-time, distributed capabilities.

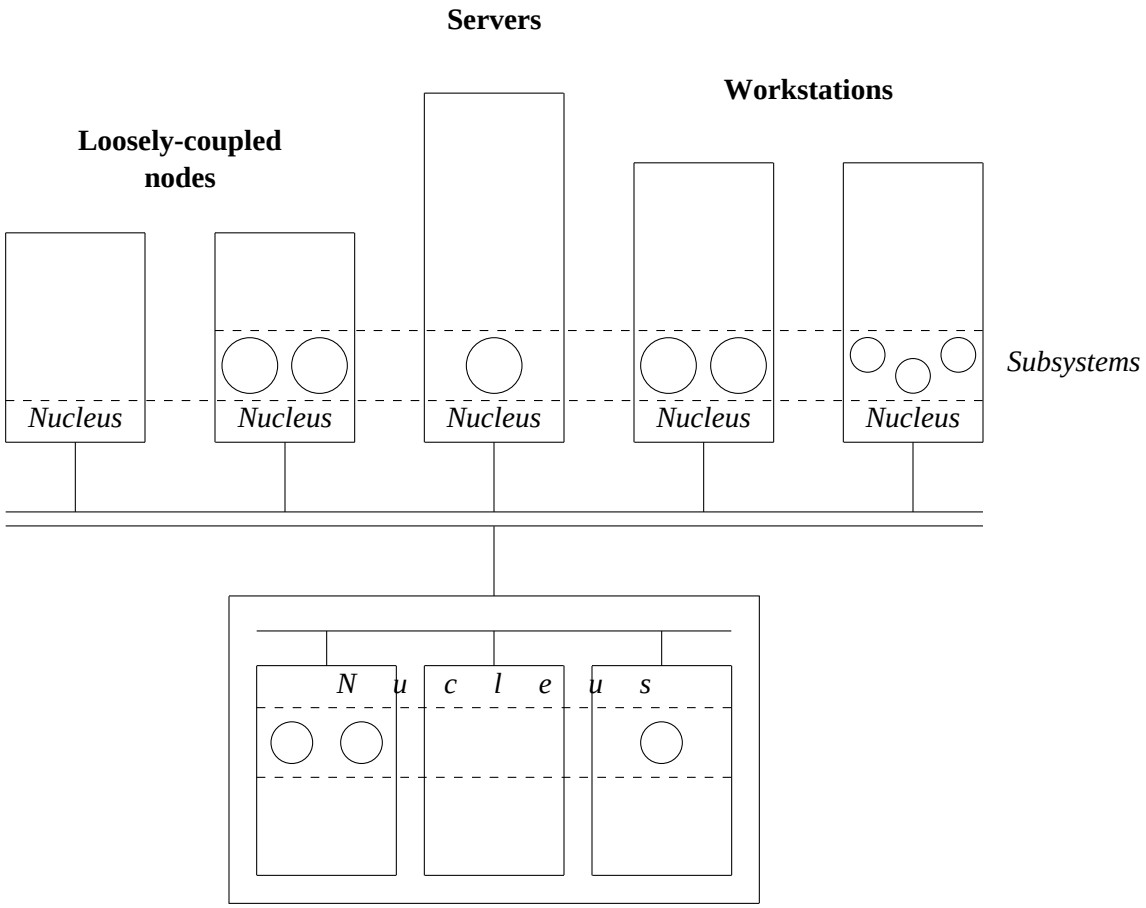

Figure 1. The CHORUS Nucleus and Subsystems

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. 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

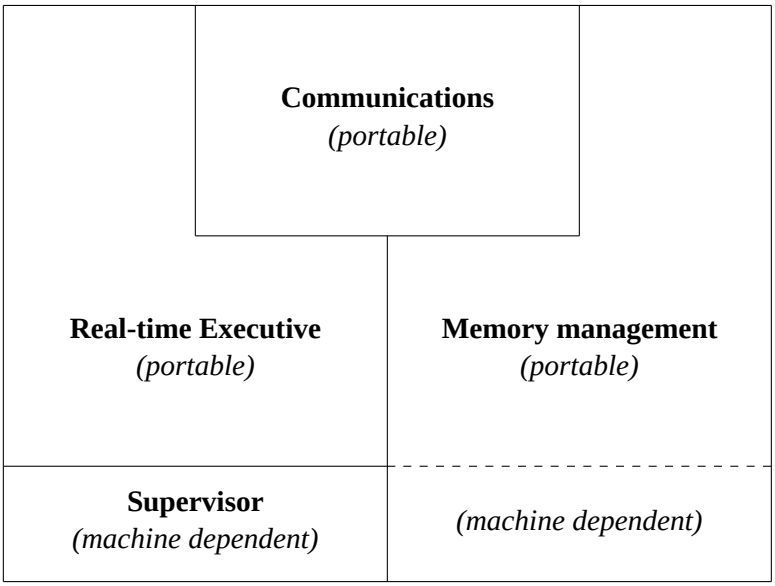

Figure 2. The CHORUS Nucleus

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.

The CHORUS/MiX Subsystem

CHORUS/MiX (see Figure 3.) integrates a UNIX System V subsystem with the CHORUS Nucleus to provide system builders with a standards-based, real-time, transparently distributed UNIX environment. CHORUS/MiX UNIX services conform to X/Open® specifications and 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 a vanilla UNIX. A CHORUS/MiX that conforms UNIX System V.4, with real-time and distributed features, is under development.

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

® X/Open is a registered trademark of X/Open.

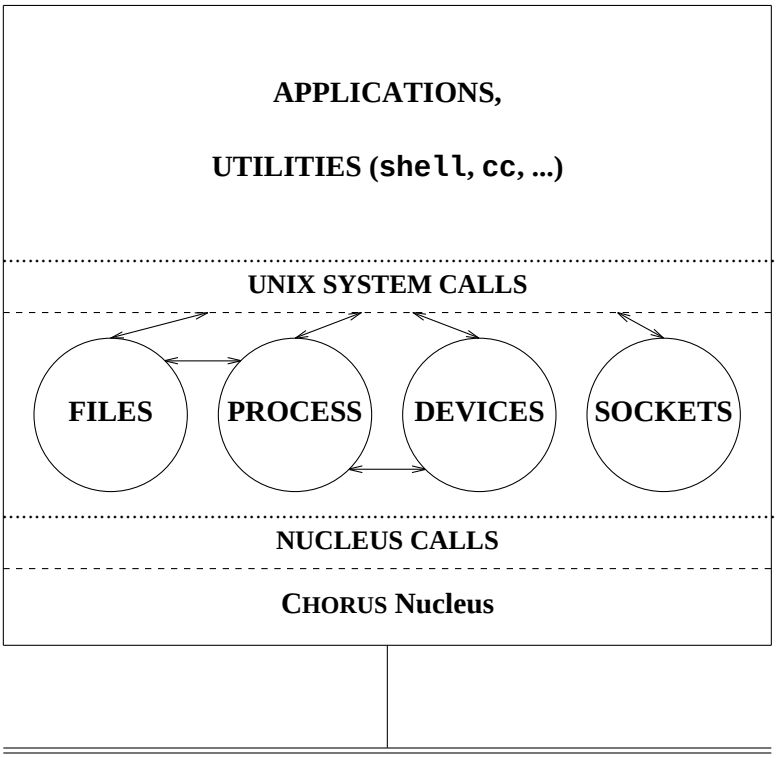

Figure 3. The CHORUS/MiX subsystem

development in user space and more. All within an open, standards-centered context.

Real-time and distribution in CHORUS/MiX

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. Later, the server can be migrated easily within the system for performance consideration.

Enablers

CHORUS incorporates both distributed system and real-time 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, 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 µ-kernel interfaces can also be used to build higher level semantics.

CHORUS reconciles real-time, distribution and UNIX

The future for real-time distributed computing environment is the renewal of operating systems technology without impacting their interface, allowing smooth migration from old-fashioned current operating systems to new generations of operating systems. CHORUS is the first commercially available product to reconcile real-time, distribution and UNIX.

The CHORUS Nucleus maintains real-time in the heart of the operating system, while associating it gracefully with distribution. The consecutive modularity, applied to subsystems, allows system builders to construct a compatible UNIX which, in addition, benefits from CHORUS's underlying real-time and distribution nature.