

# An Architectural Foundation for Net-Centric Computing

*Almost overnight, the "Net" has grown from cult status to mission-critical backbone of mainstream computing. Five years ago, for example, setting up a corporate web site was a relative novelty. Today, many businesses do most of their retail sales, as well as much of their business-to-business commerce, over the Net. And yet, the Net's remarkable growth has just begun.*

## INTRODUCTION

**A**lmost overnight, the "Net" has grown from cult status to mission-critical backbone of mainstream computing. Five years ago, for example, setting up a corporate web site was a relative novelty. Today, many businesses do most of their retail sales, as well as much of their business-to-business commerce, over the Net. One corporation alone has reported sales of over \$20 million through its e-commerce site - every day.

And yet, the Net's remarkable growth has just begun. New bandwidth-hungry services like digital video and audio are quickly gaining a foothold. Meanwhile, "always on" access technologies like xDSL, cable modems, and new wireless devices are encouraging people and businesses to use the Net for more purposes, more often. All this, coupled with the convergence of voice and data over Internet protocols, is creating a demand for new infrastructure at the core of the Net.

Despite its massive size and complexity, the existing infrastructure has done quite well. In fact, it has achieved an availability rating of 99.999 per cent or more - a far cry from the desktop segment of the computing industry.

As new systems are developed to meet the demand for higher performance, and new features and functions, this standard of reliability must be maintained or exceeded. So, faced with the challenge of designing increasingly complex systems with shorter design cycles, how do equipment manufacturers maintain a competitive edge? In a word: architecture.

## BUILDING A SOLID FOUNDATION

The hardware architecture of the equipment employed in large voice and data networks provides a model that can be applied to software systems. Systems are built around bus structures that promote modular construction of complex systems. Individual functions are allocated to cards, which plug in to the bus. The bus defines a structured interface between the system components. System designs incorporate fault-tolerance features to meet reliability targets and facilitate on-line maintenance functions.

Similarly, a realtime operating system (RTOS) architecture should, ideally, eliminate maintenance downtime, allow for failure recovery, and offer hardware platform independence, scalability, software reusability, source-code portability, increased product quality, decreased testing, shorter time-to-market and increased return on investment. No small order!

In discussing the optimum RTOS architecture for net-centric computing, we need to consider a couple of basic design principles. First, smaller is better - as the kernel decreases in size and complexity, the reliability of the system increases exponentially. Second, kernel failures are fatal - if the kernel fails, it can only recover by rebooting. And any component (like a device driver) running without memory protection in kernel mode has the potential to cause the failure of the entire system.

Given these design principles, it's easy to see how RTOS architecture can have an enormous impact on the reliability of net-centric devices. And among RTOSs, there are three basic architectures: realtime executive, monolithic, and microkernel.

### **The realtime executive**

Realtime executives present an executable environment that is multi-threaded with no memory protection. In this model, all operating system and application components share a common address space with the kernel. As a result, a failure (like a corrupted pointer) in any one of these components will cause the system to crash or become unstable. The only way to recover from such a failure is to reboot. Another limitation of this architecture is that it is difficult, if not impossible, to add or change operating system components or application code without taking the system down. Downtime is required for application updates or system maintenance activities. However, downtime may not be acceptable to an Internet Service Provider or their subscribers!

### **The monolithic model**

Monolithic architecture is an improvement over a realtime executive, since all application processes running in user mode have MMU-based memory protection between processes. This ensures that their memory space isn't overwritten and allows application code to

be upgraded without taking the system out of service. However, a significant number of OS components and system drivers run in kernel mode - without MMU protection. As a result, errant pointers or array subscripts in device drivers and other kernel components can cause kernel faults. The system must be reset in order to recover. As the amount of code running in the kernel increases, the likelihood of kernel faults increases significantly.

For a mission-critical application, this architecture is still not ideal, since system outages are required to introduce new drivers or hardware. Monolithic architecture isn't easily extensible by users who wish to support their own drivers, as this would add untested code to the kernel.

### ***A true microkernel***

Microkernel architecture offers both superior memory protection and the ability to extend the OS. Under this architecture, only a small set of essential services resides in the kernel, including scheduling, interrupt handling, and message-based interprocess communication (IPC) - commonly known as message-passing. These facilities support a team of optional cooperating processes that provide higher-level operating system services, such as file systems, network communications, protocol stacks, and graphical user interfaces. Each of these high-level services - as well as each user-written application - runs as a separate process in its own memory-protected address space. This prevents programming errors, such as the misuse of pointers, from corrupting the memory used by other processes or the kernel.

(It's important to note that some RTOSs claim to implement a microkernel architecture, but do not provide integrated memory management and protection. This means that software faults, such as errant pointers, can overwrite another application thread or the kernel. In effect, this is no better than the architecture of a real-time executive.)

The microkernel, and the message-passing IPC facilities it provides, acts as a software bus. This allows individual operating system and application processes to be added or removed from the bus dynamically. As a result, operating system services, device drivers, and application components can be added or changed on the fly without bringing the system down. This capability is significant, given the high reliability and availability demands of net-centric equipment.

The well-defined interface provided by the software bus ensures that operating system services and device drivers are virtually indistinguishable from user-written applications. This allows equipment manufacturers to provide custom extensions to the RTOS. For instance, new device driver or a file system manager for a new persistent storage media can be added seamlessly to the RTOS without compromising the reliability of the kernel. A unique benefit of this approach is that the same development process (compilers and full source-code symbolic debuggers) can be used to develop operating system services, device drivers, and application modules. Net-centric equipment manufac-

turers can therefore provide new features at a faster rate - without compromising quality or reliability.

In contrast, extensions to the kernel of a realtime executive or a monolithic architecture often require a special kernel development process which is significantly less productive than a standard application development process.

### **BEYOND THE SINGLE CPU**

Microkernel RTOSs provide an operating environment that is also highly scalable, an important feature for net-centric systems. While microkernel architecture offers obvious benefits in a single CPU context, its true value becomes evident when application requirements dictate multiple processors.

To address the needs of computation-intensive applications, microkernel architecture is easily extended to support SMP (Symmetrical Multi-Processing). SMP is typically associated with general-purpose operating systems found in enterprise server roles. Most of these operating systems are based on the monolithic kernel architecture, which means that the changes to support SMP are extensive due to the sheer size of the code running within the kernel.

In contrast, the changes required to support SMP in a microkernel architecture are minimal. Since OS services and device drivers are implemented as cooperating processes and do not reside with the kernel, the size of the kernel can be very small. By modifying the microkernel alone, all other OS services and application processes gain the full advantage of SMP without the need for coding changes. In the case that these processes are multi-threaded, their threads will be scheduled concurrently across the available processors in the SMP system.

SMP provides a tightly coupled multiprocessing solution, but often multiprocessing systems are composed of loosely coupled processors distributed across a backplane, a local area network, or the Net. The message-passing IPC facilities of a properly designed microkernel architecture can transparently extend the software bus to provide transparent distributed processing. Any application on any machine can directly make use of any resource, operating system service, or custom application located on any machine - whether local or remote - in a distributed environment. The microkernel transparently redirects messages destined for remote services or applications.

For complex system designs, one of the traditional ways of achieving high reliability and availability is to distribute components of an application across multiple processors. That way, even a CPU failure won't stop the application from providing service. Dual redundant systems that provide hot-swap support for critical functions are easy to implement within the architectural framework provided by a message-passing microkernel. Often, communications links are potential weak points in the architecture of a distributed system. Support for link redundancy, load-balancing, and configurable quality-of-service is required to implement robust distributed systems. These features, along with an IP-based protocol, are requirements for compatibil-

## MARKET

ity with the Net's core infrastructure.

Finally, for Net-centric systems, performance is key. The integration of network transparent message-passing primitives into the heart of the microkernel architecture ensures performance that is as close as possible to the bandwidth of the communications medium. Through the use of multiple links, which may be a mix of LAN, WAN, or backplane connections, even higher performance is achievable while providing a flexible foundation to meet reliability requirements.

As applications become more complex and net-centric, embedded systems developers need to take a closer look at operating system architecture. By choosing an RTOS that is truly reliable, extensible, scalable, and delivers productivity advantages, developers can go a long way towards easing time-to-market pressures, while creating products that meet customer demands ■

---

*Brent Daniel is Senior Technology Analyst at QNX Software Systems Limited.*