SPECIAL BYTE REPORT

ADVANCED OPERATING SYSTEMS

A look inside the next generation of computing environments, including IBM's Workplace OS, Microsoft's NT, and software from Novell/USL, Sun, Next, and Taligent

The Great OS Debate

Since the dawn of microcomputing, users and developers have jousted with one another to defend the honor of their chosen operating systems. The battle still rages; the dust hasn't even begun to settle. New contenders will exploit mainstream RISC workstations built around MIPS, Alpha, and PowerPC processors even as they ride the Intel performance escalator. But the grounds of the operating-system debate are subtly shifting. Microsoft, IBM, USL (Unix Systems Laboratories), Sun Microsystems, and others are rapidly converging on a set of common design themes—microkernels, objects, and personalities. The battle is no longer about whether to layer object-oriented services and emulation subsystems (i.e., personalities) on a small kernel. Everyone's doing that. The question isn't *whether* to build an operating system in this style but *how* to do the job right. **— Jon Udell, Senior Technical Editor**

MICROKERNELS

In Windows NT, layered subsystems communicate by passing messages through a microkernel. But NT doesn't follow the pure microkernel doctrine, which holds that all nonessential services should run in the processor's nonprivileged (user) mode. IBM, USL, and others say that NT's executive, a layer above the NT microkernel that runs security, I/O, and other services in privileged (kernel) mode, compromises NT's claim to be a microkernel-based system. Microsoft, however, notes that NT's privileged-mode executive subsystems communicate with each other and with the kernel by passing messages, just as its user-mode emulation subsystems do.

IBM's Mach-based Workplace OS, meanwhile, will adhere to the pure microkernel doctrine, relegating the pager, the scheduler, the security system, the file systems, and even major parts of its device drivers to user mode. With this approach, says IBM, its microkernel will be especially valuable as a base that OEMs can customize for specific purposes. USL, however, says that its Chorus microkernel, which can run services in kernel mode or user mode, gives the best of both worlds. It can locate services in kernel mode for performance or in user mode for flexibility.

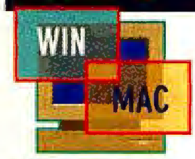
In "Small Kernels Hit It Big," Peter D. Varhol explores these and other issues across a range of microkernel-based systems. And in "The Chorus Microkernel," Dick Pountain takes a close look at the advanced technology chosen by USL as the foundation for future Unixes.

OBJECTS

As applications supporting Microsoft's OLE 2.0 begin to roll out, mainstream users are getting a glimpse of an object-oriented, document-centered style of computing in which applications function as components. Apple, IBM, and partners are countering with OpenDoc, a portable compound-document standard that will bring OLE-like benefits to a broader range of platforms than are supported by OLE. Apple says that OpenDoc's object technology, which relies on IBM's groundbreaking System Object Model, or SOM, offers developers and users the full power of object-oriented programming—including inheritance—while remaining language-neutral. Microsoft says that OLE 2.0's Compound Object Model, which is closely aligned with C++ yet does not support inheritance, will nevertheless yield better results by requiring developers to articulate interfaces precisely and consistently.

On the horizon looms Taligent, an objects-from-the-ground-up system that IBM and Apple say will redefine computing. Meanwhile NextStep, available now on Intel and Motorola platforms, delivers the distributed-object technology that the others are all still talking about. In "Objects on the March," Peter Wayner explores some of the key issues in object and distributed-object computing.

PERSONALITIES



But will it run 1-2-3? For the new breed of operating systems, the answer is almost certainly yes, even on non-Intel hardware, thanks to a hybrid emulation strategy that offsets the inherent inefficiency of pure processor emulation by implementing GUI libraries in native RISC code. Applications lean heavily on GUI libraries nowadays; Windows and Mac libraries are appearing as "personalities" on a variety of new operating systems.

In "Personality Plus," Frank Hayes investigates how Microsoft's Windows NT and IBM's Workplace OS implement personalities. Frank also explores popular third-party solutions like Sun's Wabi (Windows Application Binary Interface), Insignia Solutions' SoftWindows, as well as Quorum Software Systems' Equal.

Small Kernels Hit It Big

PETER D. VARHOL

A microkernel is a tiny operating-system core that provides the foundation for modular, portable extensions. Every next-generation operating system will have one. However, there's plenty of disagreement about how to organize operating-system services relative to the microkernel. Questions include how to design device drivers to get the best performance while abstracting their functions from the hardware, whether to run nonkernel operations in kernel or user space, and whether to keep existing subsystem code (e.g., a legacy version of Unix) or to throw everything away and start from scratch. IBM, Microsoft, and Novell's Unix Systems Laboratories answer these questions differently; each company has strong opinions about how and why its approach will work best.

It was the Next computer's use of Mach that introduced many of us to the notion of a microkernel. In theory, its small privileged core, surrounded by user-mode services, would deliver unprecedented modularity and flexibility. In practice, that benefit was somewhat obscured by the monolithic BSD 4.3 operating-system server that Next wrapped around Mach. However, Mach did enable Next to supply message-passing and object-oriented services that manifest themselves to the end user as an elegant user interface with graphical support for network setup, system administration, and software development.

Then came Microsoft's Windows NT, which touted not only modularity but also portability as a key benefit of the microkernel approach. NT was built to run on Intel-, Mips-, and Alpha-based systems (and others to follow) configured with one or more processors. Because NT would have to run programs originally written for DOS, Windows, OS/2, and Posix-compliant systems, Microsoft exploited the modularity inherent in the microkernel approach by structuring NT so that it did not architecturally resemble any existing operating system. Instead, NT would support each layered operating system as a separate module or subsystem.

More recently microkernel architectures have been announced by Novell/USL, the Open Software Foundation, IBM, Apple, and others. One prime NT competitor in the microkernel arena is Carnegie Mellon University's Mach 3.0, which both IBM and OSF have undertaken to commercialize. (Next still uses Mach 2.5 as the basis of NextStep, but it is looking closely at Mach 3.0.) Another is Chorus 3.0 from Chorus Systems, which USL has chosen as the foundation of its Unix offering (see "The Chorus Microkernel" on page 131). Sun's SpringOS, an object-oriented successor to Solaris, will use a microkernel, and the Taligent Operating Environment will rely on the same microkernel that IBM is developing for its Workplace OS. Clearly, there's a trend away from monolithic systems and toward the small-kernel approach. That's no surprise to QNX Software Systems and Unisys, two companies that have for years offered successful microkernel-based operating systems. QNX Software's QNX serves the real-time market, and Unisys' CTOS is strong in branch banking. Both systems exploit the modularity enabled by a microkernel foundation with excellent results.

Suddenly microkernels are the central design element of new operating systems. But Microsoft, IBM, USL, and others differ on how best to implement one.

Fueling the current microkernel frenzy is the recent fragmentation of the operating-system market. With no one vendor a clear winner in the operating-system sweepstakes, each needs to be able to support the others' applications. AT&T tried this tack a few years ago with Unix System V release 4.0, by including support for the Berkeley and Xenix extensions. But while SVR4 has done well enough, it hasn't been the grand unification of Unix for which AT&T (now Novell's USL) had hoped. On the other hand, Microsoft's NT seems to have succeeded—at least in this respect—by being the first to unify multiple subsystems capable of running Win32, Win16, DOS, OS/2, and Posix applications. IBM is responding with a portable successor to OS/2, the Workplace OS. Its truly modular operating-system architecture, with plug-and-play components and multiple operating-system personalities, may advance expectations still further.

Defining the Microkernel

A microkernel implements essential core operating-system functions. It's a foundation for less-essential system services and applications. Exactly which system services are nonessential and capable of being relegated to the periphery is a matter of debate among competing microkernel implementers. In general, services that were traditionally integral parts of an operating system—file systems, windowing systems, and security services—are becoming peripheral modules that interact with the kernel and each other.

When I first learned about operating systems, the layered approach used by Unix and its variants was the state of the art in operating-system design. Groups of operating-system functions—the file system, IPC (interprocess communications), and I/O and device management—were divided into layers. Each layer could communicate only with the one directly above or below it. Applications and the operating system itself communicated requests and responses up and down the ladder.

While this structured approach often worked well in practice, today it's increasingly thought of as monolithic because the entire operating system is bound together in the hierarchy of layers. You can't easily rip out one layer and swap in another because the interfaces between layers are many and diffuse. Adding features, or changing existing features, requires an intimate knowledge of the operating system, a lot of time, some luck, and the willingness to accept bugs as a result. As it became clear that operating systems had to last a long time and be able to incorporate new features, the monolithic approach began to show cracks. The initial problems vendors encountered when SVR4 shipped in 1990 illustrate this point.

The microkernel approach replaces the vertical stratification of operating-system functions with a horizontal one. Components above the microkernel communicate directly with one another, although using messages that pass through the microkernel itself. The microkernel plays traffic cop. It validates messages, passes them between components, and grants access to hardware.

This arrangement makes microkernels well suited to distributed computing. When a microkernel receives a message from a process, it may handle it directly or pass the message to another process. Because the microkernel needn't know whether the message comes from a local or remote process, the message-passing scheme offers an elegant foundation for RPCs (remote procedure calls). This flexibility comes at a price, however. Message passing isn't nearly as fast as ordinary function calls, and its optimization is critical to the success of a microkernel-based operating system. For example, NT can, in some cases, replace message ports with higher-bandwidth shared-memory communications channels. While costly in terms of nonswappable kernel memory, this alternative can help make the message-passing model practical.

Portability, Extensibility, and Reliability

With all the processor-specific code isolated into the microkernel, changes needed to run on a new processor are fewer and group logically together. Since the processor market seems more likely to fragment with competing designs than to converge on a single architecture, running an operating system on more than one processor may be the only way to leverage buyers' investment in hardware. Intel is still on top of the microprocessor hill, but IBM/Motorola/Apple, DEC, Mips, and Sparc International, among others, are making determined runs at its dominant position.

Extensibility is also a major goal of modern operating systems. While hardware can become obsolete in a few years, the useful life of most operating systems may be measured in decades. Whether the operating system is small like DOS or large like Unix, it will inevitably need to acquire features not in its design. For example, DOS now supports a disk-based file system, large hard disks, memory management, and—most radically—Windows. Few, if any, of these extensions were envisioned when DOS 1.0 shipped.

Operating-system designers have learned their lesson and now build operating systems that make adding extensions manageable. There's no alternative. With increasingly complex monolithic systems, it becomes difficult, if not impossible, to ensure reliability. The microkernel's limited set of well-defined interfaces enables orderly growth and evolution.

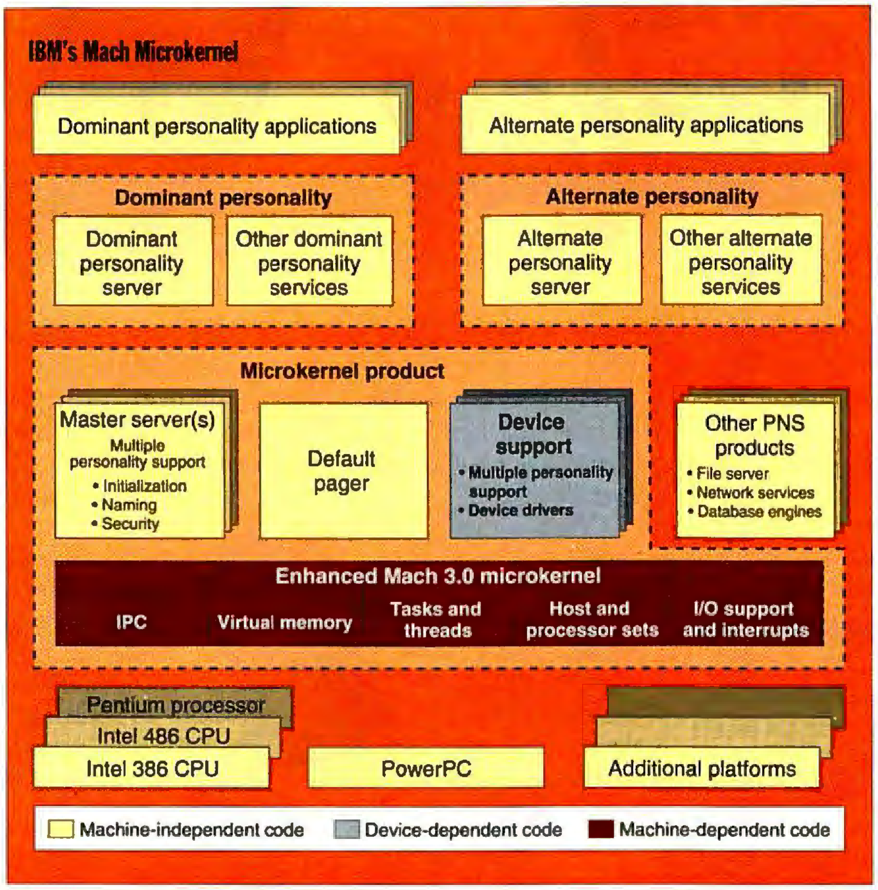
There's also a need to subtract features. More users would flock to Unix or NT if these operating systems didn't require 16 MB of memory and 70 MB or more of hard disk space. *Microkernel* does not necessarily mean *small system*. Layered services, such as file and windowing systems, will add bulk. Of course, not everyone needs C2 security or wants to do distributed computing. If important but market-specific features could be made optional, the base product would appeal to a wider variety of users. Martin McElroy, brand manager for Workplace OS at IBM's Personal Systems Products division, says that IBM's Mach implementation will eventually run the gamut from "palmtops to teraFLOPS." The services riding on the microkernel can be customized to meet the needs of the platform and the market.

The microkernel approach can also help improve the overall quality of the computing environment. Systems like Unix, OSF/1, and NT require hundreds of thousands of lines of code and take years to mature. Programmers who write applications for these systems don't have time to worry about undocumented APIs; they've got their hands full just learning about the hundreds of APIs that are documented. The learning curve for new operating-system calls is becoming so steep that no developer can reasonably expect to know and use them all.

The result is that no one can guarantee the correctness of code making use of several system-service APIs, and no one can guarantee even the correctness of the operating system itself. A small microkernel that offers a compact set of APIs (the OSF microkernel will have about 200, and the tiny QNX microkernel has just 14) improves the chances of producing quality code. This compact API is visible to the systems programmer only; the applications programmer must still wrestle with hundreds of calls. But it certainly enhances the value of microkernels such as IBM's, which the company plans to license to OEMs for customized development.

What's In and What's Out?

As we have seen, the proper division of



IBM uses the Mach microkernel *as the foundation for personality neutral services and multiple operating-system personalities.*

labor between the microkernel and its surrounding modules is a matter of debate. The general idea is to include only those features that absolutely need to run in supervisor mode and in privileged space. That typically means processor-dependent code (including support for multiple CPUs), some process management functions, interrupt management, and message-passing support.

Many microkernel designers include process scheduling, but IBM's implementation of Mach locates scheduling policy outside the microkernel, using the kernel only for process dispatch. IBM's approach separates policy from implementation, but it requires close collaboration between the external scheduler and the kernel-resident dispatcher.

Device drivers may be in-kernel, out-of-kernel, or somewhere in between. Some implementations (e.g., OSF's) locate device drivers in the microkernel. IBM and Chorus locate the device drivers outside of the microkernel but require that some driver code run in kernel space so that interrupts can be disabled and set. In NT, device drivers and other I/O functions run in kernel space but work with the kernel only to trap and pass interrupts.

IBM's Paul Giangarra, system architect for the Workplace OS, says that separating device drivers from the kernel enables dynamic configuration. But other operating systems (e.g., NetWare and OSF) achieve this effect without abstracting the devices from the kernel. While NT doesn't permit dynamic configuration of device drivers, Lou Perazzoli, project leader for NT development, notes that its layered driver model was designed to support on-the-fly binding and unbinding of drivers. But the necessary support for this feature didn't materialize in the first release of NT.

Dynamic configuration notwithstanding, there are other reasons to treat device drivers as user-mode processes. For example, a database might include its own device driver optimized for a particular style of disk access, but it can't do this if drivers reside within the kernel. This approach also yields portability since device-driver functions can, in many cases, be abstracted away from the hardware.

Mach and the Workplace OS

IBM's forthcoming Workplace OS uses a Mach 3.0 microkernel that IBM has extended (in cooperation with the OSF Research Institute) to support parallel-processing and real-time operations. This implementation counts five sets of features in its core design: IPC, virtual memory support, processes and threads, host and processor sets, and I/O and interrupt support. Giangarra refers to the Workplace OS microkernel as its *hardware abstraction layer* (not to be confused with NT's HAL, which is just the lowest slice of the NT microkernel). The file system, the scheduler, and network and security services appear in a layer above the microkernel. These are examples of what IBM calls *personality neutral services*, or PNSes, because they're available to any of the individual operating-system personalities layered above them.

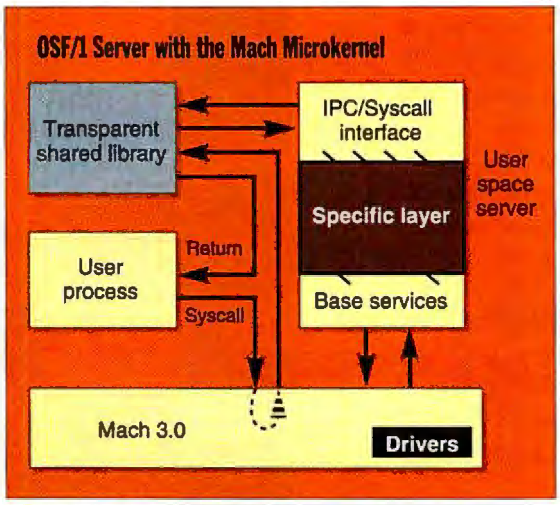
A key distinction between the IBM PNS layer and NT's own service managers is that IBM's PNS layer runs in user space, while the bulk of NT's services run in kernel space. IBM's approach aims to let OEMs add or replace system services freely; NT's system services are intended to remain in place.

Perhaps the best way to describe the relationship of the kernel to the nonkernel processes is that the kernel understands how the hardware works and makes the hardware operation transparent to the processes that set and enforce operating-system policy. In IBM's case, process and thread management is a kernel function. However, only the process dispatcher actually resides in the kernel. The scheduler, which sets policy by checking priorities and ordering thread dispatching, is an out-of-kernel function.

This is an important distinction. Dispatching a thread to run requires hardware access, so it is logically a kernel function. But which thread is dispatched, Giangarra says, is irrelevant to the kernel. So the out-of-kernel scheduler makes decisions about thread priority and queuing discipline.

The other microkernel implementations don't relegate the scheduler to the periphery. Why would you want them to? In IBM's case, the company plans to license its microkernel to other vendors, who might need to swap the default scheduler for one that supports real-time scheduling or some specialized scheduling policy. NT, which embodies the notion of real-time priorities in its kernel-resident scheduler, does not currently expose these to the programmer. You cannot modify or replace the NT scheduler.

Memory management, like scheduling, is divided between the microkernel and a PNS. The kernel itself controls the paging hardware. The pager, operating outside the kernel, determines the page replacement strategy (i.e., it decides which pages to purge from memory to accommodate a page fetched from disk in response to a page fault). Like the scheduler, the pager is a replaceable component. IBM is providing a default pager to boot Workplace OS, but the primary paging mechanism will be integrated with the file system. The Workplace OS file system (like NT's) unifies memory-mapped file I/O, caching, and virtual memory policies.



OSF/1 1.3 runs the OSF/1 *server as a monolithic component on top of the Mach microkernel.*

PNSes can include not only low-level file system and device-driver services but also higher-level networking and even database services. Giangarra believes that locating such application-oriented services close to the microkernel will improve their efficiency by reducing the number of function calls and enabling the service to integrate its own device drivers.

Mach and OSF/1

The OSF, whose OSF/1 1.3 will also incorporate Mach microkernel technology, includes virtually the same microkernel features as does IBM. The code for this version of OSF/1 was frozen in December 1993 and is due to be distributed to OSF licensees in the second quarter of 1994. IBM is a member of the OSF, and the two organizations have been exchanging microkernel technologies. However, OSF's approach differs from IBM's in important ways. OSF/1 was reworked to be able to call Mach for basic system services. Then the entire OSF/1 server system was placed on top of Mach and run in user space. What IBM divides into separate PNSes and layered personalities, OSF lumps into a single structure.

Why the monolithic Unix server riding on top of the microkernel? OSF/1 is mature and proven code, and the OSF says it wasn't feasible to start from scratch. The amount of code reuse between OSF/1 1.3 and the previous version of OSF/1 is over 90 percent. On the other hand, the OSF is also rewriting parts of the Mach kernel in C++, to be able to provide better support for object management.

The net result is that OSF/1 1.3 is less modular than Workplace OS. But by reusing a substantial part of OSF/1, the OSF can ship a more or less complete microkernel-based operating system to its members ahead of the expected debut of the Workplace OS in late 1994. Note that it is precisely this configuration—the OSF/1 server running on Mach—that IBM currently demonstrates as the Unix personality of its Workplace OS.

The OSF's goal is to let the Mach-plus-OSF/1-server combination run efficiently on massively parallel hardware systems. One of the active areas of study in the OSF Research Institute is to configure systems with dozens or hundreds of processors and to observe distributed operating-system behavior as the number of processors grows. The Mach microkernel will run on all processors, but the server—which provides file system, process management, and networking services—need run only on some.

According to Ira Goldstein, vice president of research and advanced development at the OSF Research Institute, future Mach-based versions of OSF/1 will be able to run the OSF/1 server system either in user space or kernel space, depending on the system administrator's choice when configuring the system. Running the OSF/1 server in kernel space will improve performance, because procedure calls will replace message passing, and all server code will remain in memory. Running the server in user space makes it swappable, potentially freeing memory for user programs. Note that USL is planning the same sort of flexibility for its Chorus-based offering. Arthur Sabsevitz, chief scientist at USL, expects the same advantages that NetWare 4.0 developers currently enjoy. Services will be developed and tested in user space. Once debugged and deemed trustworthy, they can move to kernel space for best performance.

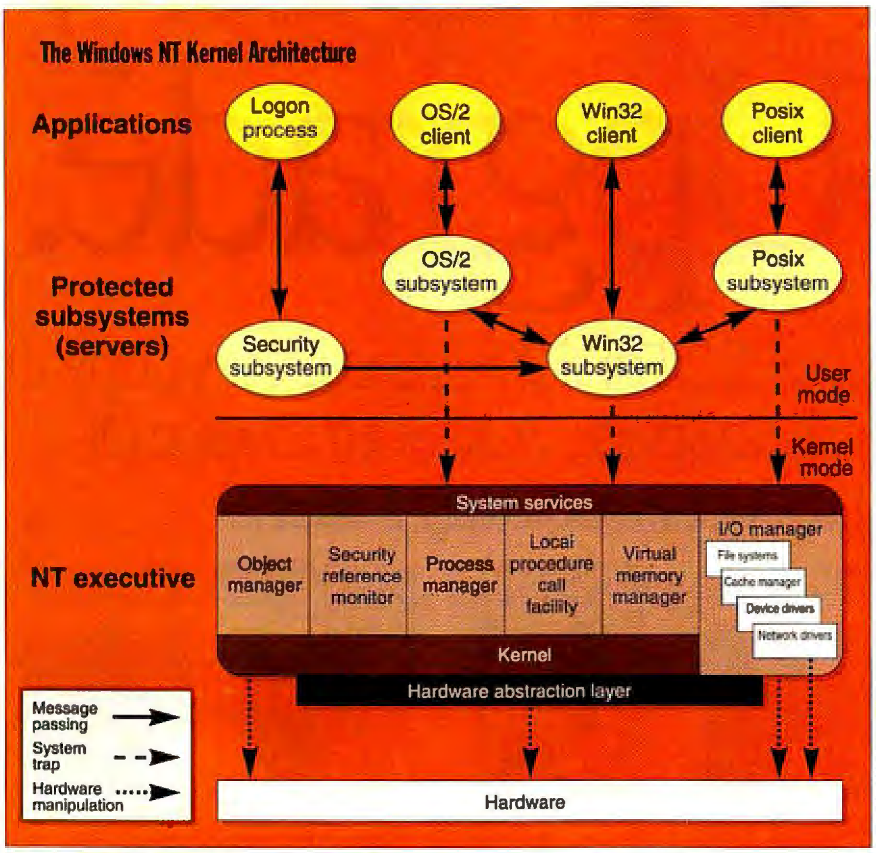
The OSF is still investigating the issue of where to locate device-driver support. Currently, drivers reside within the Mach microkernel. Goldstein says this approach should not preclude dynamic configuration of drivers. Since the OSF is working closely with IBM on microkernel issues, it will look at the IBM approach to device drivers when it receives the technology.

Is NT Really a Microkernel OS?

NT's microkernel serves primarily to support a specific set of user environments on top of a portable base. Its concentration of machine-specific code in the microkernel makes NT relatively easy to port across diverse processors. NT is also extensible, but not in the same way IBM's Workplace OS will be. Whereas IBM wants to license its microkernel separately, it is unlikely that Microsoft will attempt to unbundle NT's microkernel. This is one reason why many observers now conclude that NT is not, in fact, a true microkernel in the same sense that Mach and Chorus are. These critics also note that NT does not rigorously exclude layered services from kernel space (although OSF/1 and Chorus/MiX aren't religious on this point either) and that NT's device drivers cooperate minimally with the kernel, preferring to interact directly with the underlying HAL.

Workplace OS applications talk to user-mode "environment subsystems" that are analogous to the Workplace OS's personalities. Supporting these subsystems are the services provided by the NT executive, which runs in kernel space and does not swap to disk. Executive components include the object manager, the security monitor, the process manager, and the virtual memory manager. The executive, in turn, relies on lower-level services that the NT kernel (or microkernel, if you will) provides. Its services include scheduling threads (the basic level of execution), handling interrupts and exceptions, synchronizing multiple processors, and recovering from system crashes. The kernel runs in privileged mode and is never paged out of memory. It can only be preempted to handle interrupts. The kernel rides on the HAL, which concentrates most hardware-specific code into a single location.

Lou Perazzoli says that NT's design was



Microsoft's Windows NT *separates the device driver from the kernel and runs its operating-system service managers in kernel space.*

driven by strong biases toward performance and networkability, as well as by the requirement to support a specific set of layered personalities. The resulting separation of function between kernel and nonkernel modules reflects these goals. For example, data transfers to the file system and across the network run faster in kernel space, so NT provides in-kernel buffering for the small (16 to 32 KB) reads and writes that typify client/server and distributed applications. Locating these I/O functions in the kernel may violate the academic purity of the NT microkernel, says Perazzoli, but it supports NT's design goals.

Decisions regarding mechanism and policy were motivated by similarly pragmatic concerns. For example, Win32 support did not require a traditional process hierarchy, but other environment subsystems (e.g., OS/2 and Posix) did. The NT executive provides a set of process management services sufficient for the current set of NT personalities, and potentially for others that are similar but not yet supported (e.g., VMS). Radically different alternatives that would require modifying the executive are, however, beyond the scope of NT users.

Because executive components such as the process manager and the virtual memory manager run in kernel space (although they're not technically part of the kernel), some critics say NT is more monolithic than Microsoft likes to admit. However, while these executive-level resource managers do reside in kernel space, they nonetheless function as peers and communicate by passing messages just as the user-level subsystems do.

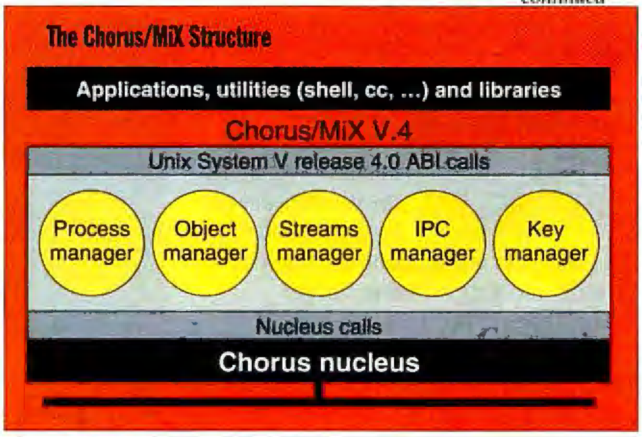
The NT model is object-based, even though not completely object-oriented. System resources such as processes, threads, and files are allocated and managed as objects; each object type exposes a set of attributes and methods. User-visible resources including windows, menus, and files are also built on object foundation. Because of their status as objects, these resources can be named, protected, and shared. NT distinguishes between kernel- and executive-level objects. Kernel objects have threads, events, interrupts, and queues. Executive objects, which executive resource managers create and manipulate, package the more basic kernel objects—adding, for example, names and security descriptors—and, in turn, pass them to user-mode subsystems.

Interrupts and Device Drivers in NT

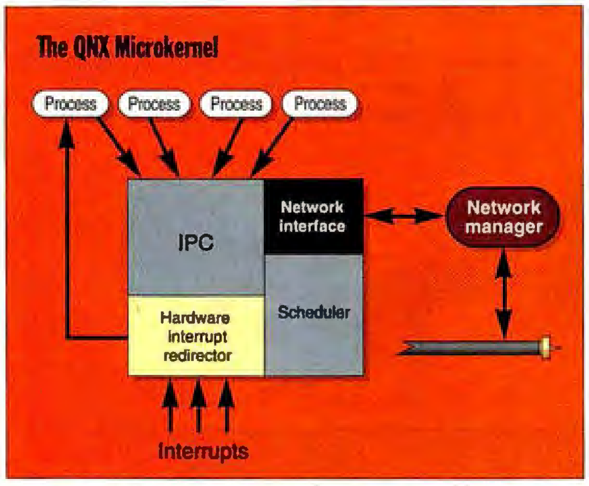
Like other microkernels, the NT kernel also handles interrupts and context switching. An interrupt is handled within the kernel and then dispatched to an ISR (interrupt service routine). The kernel uses an interrupt object to associate an interrupt level with an ISR; this arrangement conceptually separates the device drivers from the interrupt hardware. It also leads to a distinction between NT and most other microkernels in terms of the I/O subsystem. In Mach and in Chorus, device drivers reside above the kernel and access the hardware entirely through its services. In NT, the I/O manager, which includes file systems, device drivers, and networking support, generally bypasses the kernel and works directly with the HAL underneath the kernel. Kernel support is still required for interrupt processing, but in other respects, drivers work autonomously.

Perazzoli says there are good reasons to design the device-driver interface this way. For example, IBM found that it could not accomplish all device-driver functions out-of-kernel and had to find a way to let parts of drivers run in kernel space. NT establishes an object-based link to device drivers for interrupt handling and dispatch and then lets the drivers work directly with their associated devices through the HAL.

continued



Chorus/MiX V.4 *runs Unix services on top of the Chorus nucleus, in much the same way OSF/1 does with the Mach microkernel.*



The small QNX microkernel *is designed to be able to easily add service modules for specific uses.*

Nothing prevents applications vendors from writing specialized device drivers, Perazzoli notes, but these must be distinct from the application and must cooperate with the NT I/O subsystem. Is that a limitation? Perhaps not, in view of the impressive I/O performance NT has shown in benchmark tests.

AT&T and the Chorus Nucleus

The Chorus microkernel resembles IBM's and OSF's implementations of Mach in many respects. Like Mach, it takes a minimalist approach. Chorus includes support for distributed processors, multiple distributed operating-system servers (much like the Mach-OSF/1 combination), memory management, and interrupt handling. It can also communicate transparently with other instances of the Chorus microkernel, making it a good foundation for highly distributed systems.

There are several implementations of the Chorus nucleus microkernel. Chorus/MiX, the version of the Chorus operating system with Unix interfaces, includes separate versions for SVR3.2 and SVR4 compatibility. USL will offer the Chorus/MiX V.4 as a microkernel implementation of SVR4. USL and Chorus Systems plan to work together to develop Chorus/MiX V.4 as the future direction of Unix. The figure "The Chorus/MiX Structure" on page 126 shows how Chorus/MiX V.4 is configured on top of the nucleus microkernel. Chorus also supports an SCO-compatible implementation of Chorus/MiX for use specifically on PCs.

The Chorus nucleus does not include device drivers in the kernel. As with IBM's approach, device drivers work through the kernel to access hardware. According to Michel Gien, general manager and director of R&D for Chorus, this enables a higher-level component called the device manager to keep track of drivers dispersed throughout distributed systems.

On the Drawing Board

Sun, Apple, and Taligent are also moving toward a microkernel-based operating-system architecture for their respective platforms. None of these companies was willing to discuss its plans in any great detail, but all acknowledge that microkernel technology is a crucial ingredient of operating-system design.

Sun's SpringOS, which is still in the design and implementation phase, is incorporating a microkernel and making use of object extensions. While details are sketchy, it appears that SpringOS will use a large amount of existing Solaris code, much in the same way that OSF/1 uses the existing OSF/1 server. Sun has not yet announced support for any of the independent microkernels, and it may be developing its own. Still less is known of Apple's and Taligent's efforts. Although Apple will have the rights to use the Taligent Operating Environment, the company is also rumored to be developing a microkernel for the Mac System 7.

Microkernels Here and Now

QNX and CTOS are two mature microkernel operating systems that have been shipping for years. The 8-KB QNX microkernel handles only process scheduling and dispatch, IPC, interrupt handling, and low-level network services. It exports just 14 kernel calls. The compact kernel can fit entirely in the internal cache of some processors, such as the Intel 486.

A minimal QNX system can be built by adding a process manager, which creates and manages processes and process memory. To make a QNX system usable outside of an embedded or diskless system, add a file system and device manager. These managers run outside of kernel space, so the kernel remains small. QNX Software claims that this message-passing system has performance at least comparable to that of other traditional operating systems.

CTOS, introduced in 1980, was written for Convergent Technologies workstations, a family of Intel-based machines built to run in "cluster networks" linked by ordinary telephone wire. Now sold by Unisys, these CTOS-based machines were demonstrating the benefits of message-based distributed computing long before the term became fashionable. The tiny 4-KB CTOS microkernel concerns itself only with process scheduling and dispatch and message-based IPC. All other system services communicate with the microkernel and with each other through well-defined message interfaces.

Networking is integral to CTOS workstations and effectively transparent to applications, which do not need to know whether a request for service will be handled locally or remotely. The same message-based IPC transmits the request in either case. Building modular system services to service such requests is straightforward. One practical result has been that CTOS applications running unattended in remote branch offices are easily controlled by central management tools.

The Microkernel Advantage

If you're charting the enterprise computing strategy for your organization, you've got to be excited about the trend toward microkernel-based operating systems. Increasingly, you will be able to match kernel-independent networking, security, database, and other services to your available hardware, and customize systems for individual user's needs.

Of course, end users don't care much about how operating systems work, they just want to run the applications that enable them to do their jobs. Will microkernels influence end-user computing? You bet. By abstracting application-level interfaces away from underlying operating systems, microkernels help ensure that an investment in applications will last for years to come, even as operating systems and processors come and go.

The full benefits of microkernels won't be apparent for years. It will take that long to field the operating systems and for useful add-on modules to appear. Some benefits (e.g., quality and robustness) may never be directly apparent to users. However, it's clear that microkernels are here to stay. ■

Peter D. Varhol is an assistant professor of Computer Science and Mathematics at Rivier College in New Hampshire. He can be reached on the Internet or BIX at pvarhol@bix.com.

The Chorus Microkernel

Amid all the hype about microkernel-based operating systems, don't overlook Chorus/MiX, a commercially proven Unix variant from France that offers a number of enhanced features

DICK POUNTAIN

Life has never been tougher for operating-system designers. Any operating system that aspires to cope with all the directions computing will take in the coming decade needs to fulfill a formidable wish list—multitasking, networking, fault tolerance, symmetric multiprocessing, and massive parallelism—while maintaining binary compatibility with industry-standard software across heterogeneous distributed platforms. Oh, and would it also support object orientation, please? As daunting as all this sounds, however, there's an existing, commercially proven operating system that supports all these features. It's made in France, and it's called Chorus/MiX.

Chorus/MiX is a microkernel-based, distributed Unix operating system that grew out of research into packet-switched networks in the late 1970s at INRIA (Institut National de Récherche en Informatique et Automatique), a government-funded laboratory in suburban Paris. In 13 years of development, Chorus has passed through four major versions and has absorbed key concepts from all the most important academic research projects in the distributed-systems field. Message passing was influenced by Stanford University's System V, threads and distributed virtual memory by Carnegie Mellon University's Mach, and network addressing by Amsterdam University's Amoeba.

In 1982, version 0 of Chorus established the basic principle of a small distributed kernel (called the *nucleus*) that directly supports IPC (interprocess communications). By 1986 the Chorus team had spun off from INRIA into a new company, Chorus Systèmes (now Chorus Systems), to exploit Chorus in the commercial arena. The current product, Chorus/MiX, is based on version 3 of the Chorus nucleus. It presents a standard, 100 percent binary-compatible Unix System V release 3.2 or SVR4 interface with added real-time and multithreading features.

Chorus has met with considerable success in its home country; communications giant Alcatel, France's equivalent to AT&T, has just adopted it as the standard operating system for all its future PBX equipment. More recently, Chorus has started to attract attention in the U.S., announcing deals with Unisys, Tandem, Cray Research, The Santa Cruz Operation, and Unix Systems Laboratories. It is available for a wide range of hardware, from the Intel 80x86 family to the Inmos Transputer, and Motorola has recently announced the development of a RISC chip in the PowerPC family that will have the Chorus nucleus "on-chip" for embedded applications.

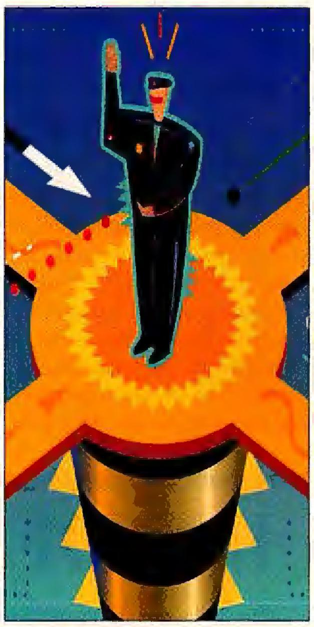
Chorus Basics

Chorus systems are built on a tiny nucleus (typically only 50 to 60 KB in size) that handles scheduling, memory management, real-time events, and communications. Everything else in the operating system is a *server* that sits on top of the nucleus and communicates with it by passing messages. File managers, stream and socket managers, and even device drivers are all treated as servers; a group of such servers is called a *subsystem*. In the case of Chorus/MiX, the complete Unix V implementation is such a subsystem (see the figure "Chorus Nucleus with Layered Unix Services").

This extreme modularity confers many important advantages. For example, in the Unix subsystem, only those servers that are actually being used need to be loaded into memory. The ease of substituting one modular server for another simplifies the implementation of fault tolerance and redundant backup.

The system-level communications abilities allow easy distribution of the operating system by running a separate nucleus on each processor. Combining these abilities lets you build distributed fault-tolerant systems that can reconfigure themselves dynamically.

The ability to support conventional operating systems as subsystems means you could develop multiple "personalities"—say OS/2, Unix, and Windows—and have them interwork transparently via the common underlying commu-


STEVE LYONS © 1994

nications layer. IBM appears to be basing its future operating-system strategy on a similar idea, implementing it on the Mach 3.0 microkernel rather than on Chorus.

Perhaps more important than these advantages is the fact that the modular Chorus system can remain comprehensible and maintainable even as it grows very complex. You can write, test, and debug servers on a running system in piecemeal fashion. In contrast, monolithic operating systems that grow by adding on extra layers tend to reach a crucial complexity barrier beyond which they become very difficult to manage.

The Chorus Nucleus

The IPC manager in the Chorus nucleus (see the text box

"Inside the Nucleus" below) delivers messages between *actors* on the same *site*, but a network manager external to the nucleus is responsible for keeping track of *ports* throughout the system and for the dirty business of network communications. (For definitions of these terms, see the text box "A Chorus Lexicon" on page 136.)

At present, the network manager supports both OSI and Internet protocols. In addition, it acts as a communications server for those special actors that need to access network services directly; for all other actors, IPC is network transparent.

As well as being compact, the Chorus nucleus is also highly portable to different CPU architectures, because only the supervisor and part of the memory manager are hardware dependent. Indeed, this isolation of hardware dependencies is perhaps the strongest commercial rationale for adopting a microkernel approach. Similar reasoning lies behind the HAL (hardware abstraction layer) in Windows NT, which so far supports Intel, Mips, and DEC Alpha processors.

Messages and Efficiency

The choice of a message-passing rather than a shared-memory paradigm for IPC in Chorus is the key to its elegant ease of distribution, particularly in heterogeneous environments where shared memory can be a nightmare to implement. However, message passing has a reputation for being less efficient than shared memory, and since every server in a Chorus subsystem such as Unix ultimately relies on IPC to communicate with other servers, any message-passing overhead will have a serious impact on overall system performance.

Accordingly, Chorus's designers have made great efforts to optimize the IPC system. Chorus messages use a very simple format—just untyped strings of contiguous bytes—and the IPC manager implements no flow control or security checks. System builders add these facilities at the subsystem level using the raw services provided by the nucleus, so that their overhead is incurred only where necessary.

The RPC (remote procedure call) mode of communication employs optimizing algorithms (or *lightweight RPC*) that exploit any locality of client and server. For example, when both client and server threads are executing on the same site, the IPC manager instructs the memory manager to move the message data by simply remapping addresses, without any actual copying. When copying between sites does occur, a copy-on-write scheme ensures that data is transferred only as needed. Given a host processor that provides on-chip communications, such as the Inmos T9000 Transputer, the Chorus IPC service can be mapped directly onto the hardware. The French firm Archipel has done this for its Volvox range of massively parallel supercomputers.

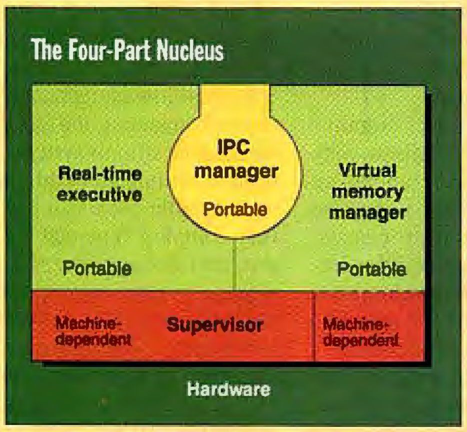
The nucleus's supervisor has also been subject to extensive optimization, both to improve performance and to achieve 100 percent binary compatibility for the Unix subsystem. Version 2 of Chorus employed a pure message-passing interface to Unix and required that all device drivers be part of the nucleus executing in privileged mode. All Chorus/Unix processes had to contain user-level *stubs* to convert system calls into messages; this altered the memory map and spoiled Unix binary compatibility.

Version 3 of Chorus, therefore, introduced a new class of entities, called *supervisor actors*, that execute in the supervisor's address space in privileged mode but are still

INSIDE THE NUCLEUS

The Chorus nucleus is divided into four functional parts:

The multitasking real-time executive allocates local processors and schedules threads using a priority-based preemptive scheme (or, optionally, by time slicing). The executive's programming interface provides primitives for thread creation and destruction, as well as synchronization via semaphores, spin locks, mutexes, or condition variables. Here, as elsewhere, the Chorus philosophy is to provide a variety of efficient but low-level mechanisms, leaving the choice of performance trade-offs to the (sub)system builder.

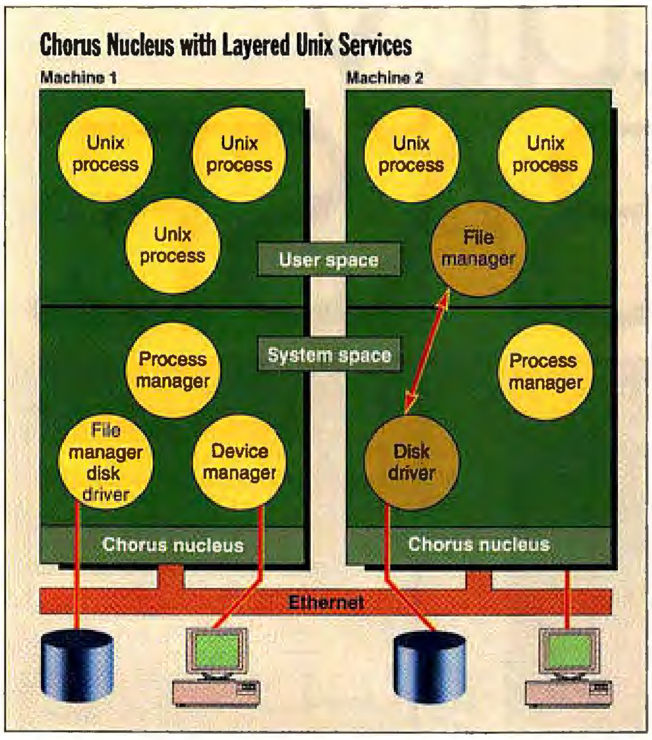


The real-time executive and the IPC manager are fully portable. The supervisor, like NT's HAL (hardware abstraction layer), is fully machine-dependent. The memory manager is partly portable, partly machine-dependent.

The memory manager supports distributed virtual memory. The basic unit of stored data is a *segment* that normally exists on some form of backing store. The virtual address space of an actor is divided into contiguous regions that map a portion of a segment into physical memory. System actors called *mappers* manage segments, maintaining the coherency of distributed shared memory when different threads access the same segment concurrently.

The supervisor dispatches interrupts, exceptions, and traps to dynamically defined device drivers and other real-time event handlers at run time. Its response time is fast enough for Chorus to be applied in real-time control systems.

The IPC (interprocess communications) manager delivers messages between ports throughout the system. Two communications modes are supported: a simple, nonblocking, asynchronous send/receive protocol in which messages are not acknowledged, and an RPC (remote procedure call) with full client-server semantics.



The modular approach *simplifies implementation of fault-tolerant systems that can reconfigure themselves dynamically. (Figure courtesy of Chorus Systèmes)*

compiled and loaded as separate modules. Supervisor actors, alone among Chorus objects, are granted direct access to the hardware event facilities, and they can install threads (called *connected handlers*) that are called directly by nucleus code, like parameterized subroutines, and then return control to the nucleus.

Connected handlers provide a conventional system-trap (rather than message-passing) interface to the nucleus, thus restoring Unix binary compatibility. Their judicious use greatly reduces interrupt response time and enables device drivers to be implemented entirely outside the nucleus. You don't need to modify the nucleus to accommodate new device types, and drivers can be dynamically loaded and destroyed with no loss of interrupt response. While Chorus adheres to its elegant theoretical principles for the most part, it is pragmatic enough to relax them when performance requires it.

Ports and Port Groups

A Chorus port represents both a resource (i.e., a queue of messages waiting to be consumed by one or more threads) and an address to which messages can be sent. Many threads within an actor can use the same port, so you can improve the performance on a multiprocessor machine, transparently to the existing clients, by adding more processors. Ports can also be dynamically migrated to a succession of different actors, which provides the basis for Chorus's run-time reconfiguration abilities.

Chorus can assemble a number of ports into a named port group, which introduces an extra level of indirection into communications. Messages sent to a port group are "multicast" to all its members; since the membership of the group can change over time, this provides a powerful mechanism for the dynamic binding of messages. Before examining groups further, I need to explain a little about naming objects in Chorus.

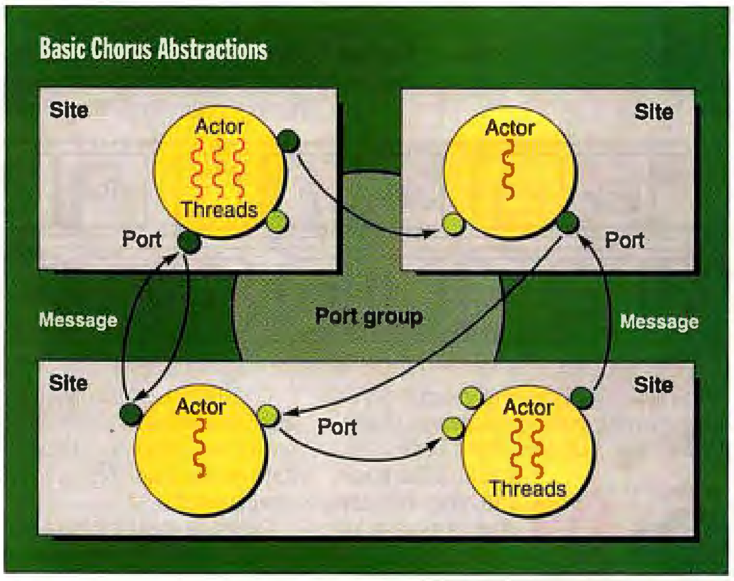
Chorus employs a single, global name space with names that are usable at any level, from nucleus to application. This contrasts with systems such as the DNS (Domain Name System) servers used under TCP/IP on the Internet, in which names are local to each site and a central name server routes messages. Chorus's name management is fully distributed, which removes a potential point of failure in the name server and makes it easier to achieve high-reliability systems.

Chorus generates names called UIs (unique identifiers) for all actors, virtual memory segments, and IPC addresses (i.e., ports and port groups), in such a way that the UIs are unique in both time and space; no two objects in a distributed Chorus system will ever use the same UI for as long as the life of the system.

UIs are 128-bit quantities formed by concatenating a site number, which records the birthplace of the object, with a "stamp" chosen from a very large, sparse random-number space. If you need to build a gateway from one distributed Chorus system to another, you can preface each system's UIs with an extra domain name identifying the system.

Chorus supplies the raw means for protecting names, although the actual protection policies must be implemented in subsystems. Objects created by external servers (e.g., segments) rather than by the nucleus are named by global *capabilities* constructed by combining the UI of a port of the server that manages the object with a 64-bit key that holds access control information. Protection in Chorus can be summed up by the following three rules:

1. Only possession of a port gives the right to receive on it. Ports cannot be shared between actors.
2. Only knowledge of the name of a port or port group gives the right to transmit to it. The knowledge of names is protected against forgery by the



Threads and messages *work much as you'd expect if you're familiar with Mach or Windows NT, and you won't go far wrong if you think of actors as the Mach or NT equivalents of processes. Port groups introduce a multicast capability that's a powerful mechanism for dynamic binding of messages.*

A Chorus Lexicon

Actor. The equivalent of a Unix process; it provides an execution context for one or more threads. An actor is the unit of distribution in Chorus, the smallest software entity that can be allocated to a site. It is not the smallest unit that can be allocated to an individual processor, however; Chorus can allocate the individual threads within an actor to different processors on a multiprocessor site, so that Chorus supports tightly coupled parallel computers as well as loosely coupled networked computers.

Ports. Queues attached to actors by which threads of one actor send messages to threads of another. Sending messages via ports rather than directly to the other thread decouples communication from execution, so communication in Chorus becomes transparent with respect to distribution; one thread need not know where another is executing in order to communicate with it. A thread can only ever belong to one actor, but a port can migrate from one actor to another, redirecting all messages to the new actor.

Site. The basic unit of computing hardware under Chorus, consisting of one or more processors and some memory and I/O devices. It might be a whole computer or just a board in a rack. Each site runs one nucleus.

Thread. The unit of execution in Chorus. It has the same meaning (i.e., a lightweight process) as it does in Windows NT and OS/2. Unlike a heavyweight Unix process, a thread does not need a private address space but only its own stack, and many threads can share the same address space. Under Chorus, that address space belongs to an actor.

sparse and random nature of name generation.
3. Only knowledge of the key of a port group gives the right to update it (i.e., to insert or remove ports).

The Chorus IPC system also supports authentication, issuing to every new actor and port a protection identifier that cannot be altered except by a special superuser. Every message is stamped with the identifiers of its sender actor and port. The receiver can read, but not modify, this stamp and apply its own authentication policies (e.g., traditional Unix file permissions).

The UI of a port group names all the ports in the group so that when a thread sends a message to that UI, the message will be received by every port in the group. A newly created port group is just an empty UI, into which ports can be inserted and removed dynamically. A port can belong to more than one group at the same time.

This group concept is very important to Chorus, because the group UI provides a single stable name for what might be a changing group of entities. In effect, a group UI names a system service rather than the actual servers that provide the service.

Groups permit a degree of immortality, because they persist even after the ports they contain have terminated. This property allows failed servers to be dynamically replaced (i.e., *hot reconfiguration*) without disrupting any transactions in progress.

Take, for example, a RAID-style file server built from a bank of drives. Each drive's server will have one or more ports by which actors elsewhere in the system can exchange data with it. If these ports are all inserted into a single group and remote threads send messages to the group rather than to the individual ports, you can replace a failed drive with a backup unit, and programs that are running will never notice any difference.

Objects Are COOL

With Unix pretty well tamed, Chorus Systems has turned its attention to object orientation. COOL (Chorus Object-Oriented Layer) is an ongoing research project, now into its second iteration, being carried out with INRIA and two European Esprit projects. COOL-2 defines three layers that sit on top of the Chorus nucleus.

COOL-base, the first layer, encapsulates the Chorus nucleus to present a new object-oriented microkernel with a system-call interface. COOL-base deals with abstractions called *clusters*, which are simply collections of virtual memory regions mapped into an address space. From a higher-level viewpoint, clusters are the places where objects exist. The COOL-base layer manages clusters, mapping them into multiple address spaces to produce distributed cluster spaces. Clusters are the units of persistence and are subject to garbage collection.

On top of COOL-base lies the GRT (generic run-time) layer, which provides support for finer-grained objects within clusters. In particular, the GRT provides for object execution, virtual object memory, a single-level persistent object store similar in concept to that used in Apple's Newton architecture, interobject communications based on nucleus RPC, and a protection subsystem to enforce protection of objects during application execution.

The final layer is the language-specific run-time layer, which maps the object model of particular programming languages, such as C++ or Smalltalk, onto the GRT's abstractions. This layer uses preprocessors to generate an upcall table for every type of object created at the GRT level, through which the GRT can call to obtain language-specific information about the semantics of certain operations. For example, it could find out how to convert in-memory object pointers to persistent pointers for storage, or how to handle method dispatch. This mechanism will enable COOL to support many different OOP (object-oriented programming) languages with reasonable efficiency.

The toughest outstanding problem in COOL right now is how to group objects that invoke one another into the same cluster, so as to maximize efficiency. Current versions do this statically, scanning the source code for object interactions, but the long-term plan is to investigate dynamic clustering based on the run-time execution patterns of objects.

When COOL makes it to product status, then Chorus, alone among current operating systems, will be able to claim that it can handle every item on that wish list at the beginning of this article. It's beginning to look as though Taligent (the IBM/Apple joint venture) and Microsoft may be busy reinventing wheels that they could have bought on a shopping trip to Paris. ■

Dick Pountain is a BYTE contributing editor based in London. He specializes in programming languages and system architectures. You can reach him on the Internet or BIX at dickp@bix.com.

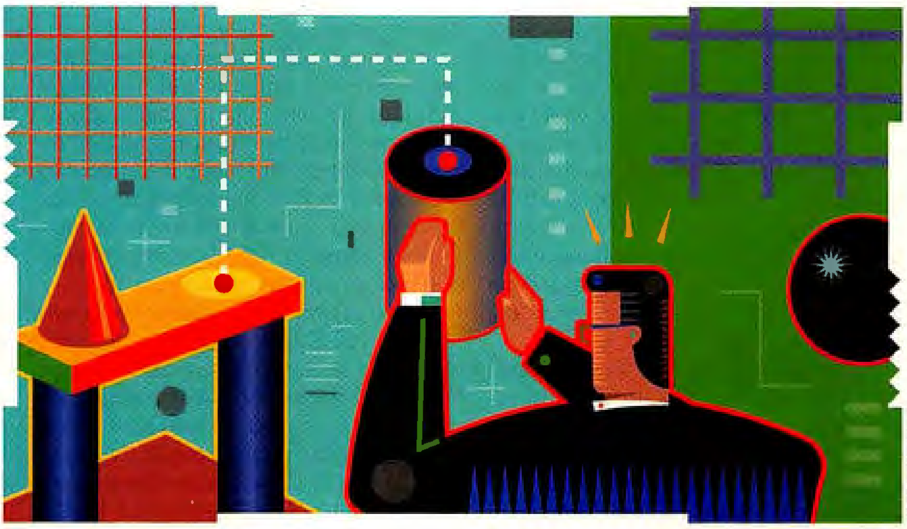
About the Company

Chorus Systems
6 Avenue Gustave Eiffel
F-78182
Saint-Quentin-en-Yvelines
Cedex, France
+33 1 30648200
fax: +33 1 30570066
Circle 1090 on Inquiry Card.

Chorus Systems, Inc.
15262 Northwest
Greenbrier Pkwy.
Beaverton, OR 97006
(503) 690-2300
fax: (503) 690-2320
Circle 1091 on Inquiry Card.

Objects on the March

PETER WAYNER



Microkernel technology lays a foundation for modular systems that can evolve in an orderly manner, but it doesn't guarantee results. For example, you could argue, with some justification, that MS-DOS already is a microkernel to which users add extensions such as networking and Windows. Of course, redefining DOS in this way doesn't sweep away the instabilities and conflicts that arise when you pile on arbitrary mixtures of TSR programs, device drivers, and memory managers. Similarly, Macintosh users find that INITs and other system extensions often lead to trouble.

Clearly what's needed is an object-oriented approach to the design of operating systems—one that lends discipline to the process of adding modular extensions to a small kernel. Microsoft, Apple, IBM, Novell/USL (Unix Systems Laboratories), and Sun Microsystems are all moving their operating systems in this direction. Taligent, the IBM/Apple joint venture, hopes to leapfrog everybody else with its from-scratch object-oriented operating system. Next, meanwhile, ships Motorola and Intel versions of NextStep, the most advanced microkernel-based and object-oriented operating system available. NextStep lacks the bottom-to-top object orientation that will be Taligent's hallmark, but at least it's available today.

Fully object-oriented operating systems will appeal strongly to systems programmers and users alike. At the system level, objects will enable programmers to dig deeply into the depths of the operating system to customize it to their needs, without disrupting system integrity. At the application level, users will find that they can mix and match features and accessories.

Objects also pave the road to distributed computing. *Objects* are units of code and data that communicate by sending and receiving messages. When built correctly, the objects in a system are highly interchangeable, and it can be a relatively straightforward task to swap remote objects for local objects and thereby extend object communication across a network. Programmers must compensate for the latency inherent in such a distributed system, but that's not the hardest problem that these systems introduce. The tough nut to crack will be uniform directory services that enable programmers to name and search for objects on a network that may be scattered worldwide.

The seamless nature of object systems will radically alter the way we think about *where* our data is. Data will be encapsulated in objects that will in some cases be able to roam to where they are most needed. We are in the habit of thinking that a document is simply stored on a particular hard disk. Distributed object systems will ask us to surrender that comfortable certainty in exchange for the power and flexibility of location-transparent storage.

Object-oriented technologies will help the next generation of operating systems evolve in an orderly way and reach out across the network

If we're to entrust our data to object systems, we'll have to be sure they can handle it securely. What's to prevent a malicious user from forging messages to access information? The next generation of operating systems will include cryptographic protocols that will enable objects to authenticate messages. Complete object systems will also have to provide ways to authorize some forms of inter-object communication while denying others.

All this won't happen overnight; it's going to be a long, evolutionary process. But it's important to understand how the technologies available today and those available in the near future—Microsoft's OLE; the OpenDoc standard from Apple, IBM, WordPerfect, Novell, and Borland; IBM's DSOM (Distributed System Object Model); Next's PDO (Portable Distributed Objects); and Taligent's frameworks—will prepare users for life in a world of distributed objects.

The Evolution of Microsoft's OLE

Applications at the top of the object food chain will be most users' first taste of these emerging object systems. For Windows users, that means applications that use Microsoft's OLE technology. With the first version of OLE, which debuted with Windows 3.1, users could insert objects into *client* documents. Those objects referred to (in the case of linking) or contained (in the case of embedding) data in a format recognized by *server* applications. Users double-clicked on the objects to launch the server applications and transfer data to them for editing.

OLE 2.0, available now as a Windows 3.1 extension, redefines the client document as a *container*. When a user double-clicks on an OLE 2.0 object that's been inserted into a container document, it can be activated in place. Suppose, for example, that the container is a Microsoft Word 6.0 document and the inserted object represents a range of cells in Excel 5.0 format. When you double-click on the spreadsheet object, Word's menus and frame controls magically become those of Excel. In effect, the word processor becomes a spreadsheet while the contained spreadsheet object has focus.

Clearly, the user benefits from this compound document model, but for programmers, OLE 2.0 requires a radical mind shift. They're used to writing applications that can, to a large extent, control the user interface. Under OLE 2.0 or similar systems, the programmer must build an application that's prepared to surrender substantial autonomy and function as a cog in a machine. Programs have to conform to rigid interfaces in order to interact successfully with other objects. OLE's designers strove to find the right balance: The interface had to be sufficiently rigorous to ensure trouble-free object interaction, yet flexible enough to allow objects to evolve in interesting and useful ways.

The root interface supported by all OLE 2.0 objects is called IUnknown. It provides a method, QueryInterface, that describes other, more specialized interfaces supported by each object. To inquire about one of these, your program consults QueryInterface, which supplies the name of the interface. How do you know which names to inquire about? They're listed in the system registry.

When you call through an interface to the methods it supports, you're using a virtual function table, or vtable, that is quite similar to the vtables generated by C++ compilers. But while the structures generated by C++ compilers can differ from machine to machine and from compiler to compiler, OLE's vtables present a standard, well-known mechanism.

The similarity to C++ does mean, however, that OLE 2.0 is much easier to use in C++ than in any other language. Calling OLE 2.0 objects from C, for example, requires substantial effort. You have to create and initialize vtables explicitly, duplicating work that's done automatically by a C++ compiler. The C++ bias of OLE 2.0 stands in sharp contrast to the language neutrality of IBM's SOM (System Object Model), the object-dispatch mechanism at the heart of OpenDoc (see the table "OLE vs. OpenDoc").

OLE objects can support a wide range of interfaces to functions for such things as memory management, name binding, data transfer, and object storage. Among the most important are the interfaces that provide a common way for an object to negotiate with the container for display real estate in the container's window and for storage space in the container's document.

The infrastructure required to support these complex object interactions is so extensive that Microsoft has described OLE 2.0 as "one-third of an operating system." Object storage, for example, utilizes a *docfile*, which is really a miniature file system contained within an ordinary MS-DOS file. Docfiles provide their own internal mechanisms for subdirectories, locking, and transaction (i.e., commit/rollback) semantics.

What doesn't OLE do yet? Networking is the most glaring omission, and it's the top priority for future OLE development. The next major iteration of OLE will appear in a distributed, object-based version of Windows called Cairo, which is due in 1995.

Apple's OpenDoc

Apple, along with WordPerfect, Novell, Sun, Xerox, Oracle, IBM, and Taligent—collectively known as the Component Integration Laboratories—is also pursuing an object-oriented compound document architecture called OpenDoc. Designed as a cross-platform technology, the project lags behind OLE 2.0 considerably and won't enter its alpha stage until about the time this article sees print. Apple expects to ship beta OpenDoc development kits this summer, in time for the Apple World-Wide Developer's Conference.

The core technologies in OpenDoc are the Bento storage mechanism (named after the Japanese plates with compartments for different foods); a scripting technology that borrows heavily from AppleScript; and IBM's SOM. In a Bento document, each object has a persistent ID that moves with it from system to system. Storage is not only transactional as in OLE, but it is capable of storing and tracking multiple revisions of each object. If there are several drafts of a document, only the incremental changes from one revision to the next will actually be stored. The upper limit to the number of extant revisions will be user-configurable.

This incremental approach will significantly reduce the disk space that's needed to maintain multiple revisions of a document. Because the Bento system will be transactional and multiuser-safe, it will lend itself to the development

OLE VS. OPENDOC
Two models for object-oriented compound documents.

	OLE	OPENDOC
Openness	Controlled by Microsoft.	Controlled by the CIL (Component Integration Lab). Many vendors, including Apple, Borland, Claris, and WordPerfect, are participating in the project.
Language	C++-oriented.	Language-neutral.
Inheritance	Simulated with aggregation.	Genuinely supported.
Storage Model	Compound file with transaction controls.	Compound file with transaction and revision controls.
Availability	For programmers, now. For users, OLE-2.0 capable applications are now shipping.	For programmers, alpha and beta versions will appear during 1994.

To Inherit or Not to Inherit?

The ability of objects to be derived from and specialize more general objects is fundamental to any object-oriented system. Yet Microsoft deliberately excluded inheritance from OLE 2.0's object model. The problem, according to OLE developers, is that it's hard to specify a precise interface between a base object and a derived one.

For example, suppose an object inherits half of its behavior from the operating system and provides the other half itself. Now suppose that a new version of the operating system revises the base object while preserving its interface. In theory, the derived object should still work perfectly. This is the major selling point for object-oriented systems. IBM, for example, touts SOM (System Object Model) as a way to achieve binary reuse of objects.

But there can be hidden pitfalls, say OLE developers. Suppose the derived object defines a virtual method that supersedes a method in the base object. Suppose also that the original version of the base object called this virtual method once after all its data was initialized. What if the new base object called the virtual method before some piece of data was initialized? The interface wouldn't be violated—parameters would still be passed correctly—but tacit assumptions made by the derived object's programmer could lead to trouble.

Microsoft therefore came up with the notion of *aggregation*, whereby programmers must explicitly build in the pointers from a derived object to a base object. This approach allows the programmer to build in controls that would stop the object from inheriting something in a dangerous way. The programmer could, for example, force the derived object to check the revision number of the base object.

In IBM's SOM, on the other hand, the dispatcher automatically uses the first instance of a base-class object that it can find. This approach requires more discipline on the part of programmers, who must try to ensure that the derived code they write interacts with base-class objects from one revision to another.

Apple's Kurt Piersol is familiar with this dilemma, because OpenDoc's object model is SOM. He believes, however, that talented programmers deserve the freedom that inheritance brings and can handle the responsibility that it demands. Jim Green, director of the DOE (Distributed Objects Everywhere) project at Sun Microsystems, agrees, and he notes that Microsoft's is the only object system that imposes such strictness.

Who's right? Only time will tell. Objects are not standard equipment yet. When there's a broader base of experience, we'll see whether programmers will run amok with inheritance and come begging for forgiveness like the prodigal son.

---

of collaborative applications. Note that OLE does not currently support revision control, although Microsoft says this feature will appear in Cairo.

OpenDoc's scripting, which is modeled on the Mac's AppleScript, implements a set of standard verbs that are intended to be as general as possible. Fourteen core verbs will apply polymorphically to almost all applications supporting OpenDoc. A verb might specify, for example, "move to next item," which could mean "move to the next word" in a text document and "move to the next cell" in a spreadsheet.

Apple's decision to introduce object-oriented polymorphism to the OpenDoc scripting language grew out of the company's experience with HyperCard, according to OpenDoc developer Kurt Piersol. HyperCard's XCMD mechanism enabled programmers to add arbitrary commands to the HyperCard scripting language. But programmers had to resort to difficult and inelegant tricks that could have been avoided if HyperCard's language model had been stronger.

Apple has learned its lesson, says Piersol. Thanks to IBM's SOM, which is a language-independent engine that implements inheritance and method-dispatching, OpenDoc's script language will enable programmers to write clean, clear code that makes it much easier to integrate different applications.

The team at Apple plans to make OpenDoc compatible with Microsoft's OLE. If the plan succeeds, the OpenDoc system will be able to wrap OLE objects with a layer of message-translation software. An OpenDoc container would see an embedded OLE object as an OpenDoc object, and the OLE object would see its container as an OLE container. Apple says that the reverse translation should also be possible. In that scenario, OpenDoc objects function in OLE containers. The translation layers are being developed by WordPerfect, with help from Borland, Claris, Lotus, and others.

Can it work? It's a tall order, but the fact that both OpenDoc and OLE are built with object technology makes the notion at least conceivable. Given that editing a document involves universal conventions such as "save" and "delete," Microsoft and Apple are certain to express their interfaces in similar ways.

Dueling Object Models: SOM and COM

Underlying OLE and OpenDoc are two competing object models: Microsoft's COM (Component Object Model) and IBM's SOM. Each defines protocols that objects use to communicate with one another. How do they differ? Most visibly, SOM is language-neutral and supports inheritance, while COM is strongly biased toward C++ and eschews inheritance in favor of an alternative mechanism that Microsoft calls aggregation. See the text box "To Inherit or Not to Inherit?" for a summary of the inheritance/aggregation debate.

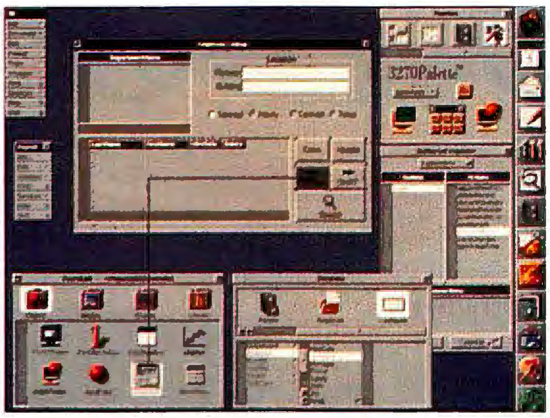
IBM first used SOM to support the class hierarchy of the Workplace Shell in OS/2 2.0. But that's just one application of what is in fact a fully general system for defining object hierarchies and invoking object methods. When one SOM object invokes another, the SOM run-time engine intercepts the call, locates the target object, activates it, and passes parameters in a standard binary format.

SOM solves a problem that has long plagued OOP (object-oriented programming) languages. Such language systems interoperate poorly because no binary standard supports inheritance and method dispatching across compilers—never mind across languages. You can't take a class library written in Borland C++ and extend it using Microsoft C++. Nor can you inherit from or extend Borland or Microsoft class libraries using COBOL, C, or Smalltalk. But you *can* do all these things if you make SOM, rather than C++ or some other OOP language system, responsible for inheritance and method dispatch.

This approach yields another important benefit: rapid development. I quit programming with one set of object-oriented libraries supplied for the Mac because I grew tired of waiting for lengthy compilations whenever I made the slightest modification to the root of the class hierarchy. Everything needed to be recompiled because the parts were in some way dependent on the root class.

SOM solves this "fragile base class" problem, according to IBM, by eliminating the need to recompile in many cases. You can add new methods and local variables to a base class without recompiling its derived classes, and the derived classes can continue to call methods of the base class as before.

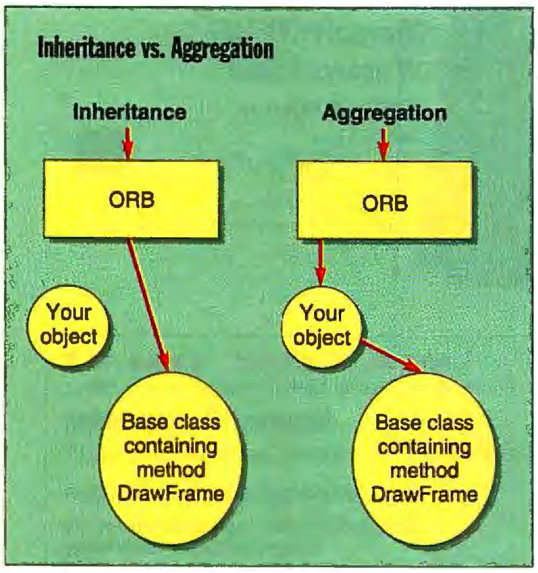
This flexibility is essential if a system is to be extended cleanly. If you use the system's window object and build your application around the features in it, you don't want to have to recompile your entire application when IBM decides to add more features to the system window object. SOM ensures that the new features won't get in your way. You may choose to use them in a later revision of your software, but there is no need to recompile the software to remain compliant with the base system.


The NextStep interface builder. Visual tools are all the rage, but Next's are still the best around.

This flexibility does come at a price, however. Using SOM means that compilers cannot optimize interobject communications. In conventional OOP implementations, compilers can sometimes place small objects in-line, effectively creating an instance of the object and removing the interobject communication code. A flexible object model like SOM must inevitably trade away such optimizations.

The SOM model was recently extended to work in a distributed manner on IPX/SPX, TCP/IP, and NetBIOS networks. DSOM looks the same as SOM to a programmer, but the DSOM run-time engine can match up objects with requests for their services even when those requests reach across process or machine boundaries.

How will IBM handle the naming of objects in a distributed system? DSOM provides its own, somewhat limited directory service, but for large-scale systems IBM plans to rely on the global directory services of the Open Software Foundation's DCE (Distributed Computing Environment).


Inheritance vs. Aggregation
Inheritance
ORB
Your object
Base class containing method DrawFrame
Aggregation
ORB
Your object
Base class containing method DrawFrame

In both cases, *your object passes on calls to draw its frame to a method called DrawFrame. In the SOM inheritance model, the ORB (object request broker) vectors the DrawFrame call directly to the base class object where it is implemented. In the COM aggregation model, your object must add to its vtable the necessary pointer to the DrawFrame method in the base class object.*

Microsoft's COM

Microsoft's COM, developed for OLE 2.0, tackles the same problems that IBM's SOM does, yet in startlingly different ways. The most visible difference is that COM doesn't explicitly support inheritance. Instead it offers another mechanism, called *aggregation,* that requires objects to explicitly include pointers to objects higher up in the hierarchy (see the figure "Inheritance vs. Aggregation").

As an example, imagine you're creating a spreadsheet object in a document, but you want it to have flexible column widths instead of the fixed columns provided by the standard object. With conventional OOP you'd inherit most capabilities (e.g., formula translation and constraint propagation) from the base class and then override the display function to implement variable-width columns. The compiler in C++, or the SOM run-time engine in the case of SOM, would redirect the display calls to your code while routing other calls to the ancestral object.

Microsoft's OLE, however, won't do such redirection automatically. You must explicitly expand your object's vtable to include pointers to the reference class. In Microsoft's terms, you "aggregate" the pointers into your object. Why is this necessary? The QueryInterface method in each OLE object only knows how to read local vtables; it can't search upward through an inheritance chain, because there isn't one.

Microsoft's architects chose this approach because they thought that it would be more resistant to the "fragile base class" problems that emerge when a base class is redefined. "It is significantly easier for programmers to not be clear about the actual interface between a base and derived class than it is [for them] to be clear," says Bob Atkinson, one of the principal developers of COM and OLE. "In practice, the base-derived interface will not be well articulated, thus preventing the base-class provider from revising his product," he notes.

But OLE developers didn't want to rule out inheritance completely, so they allowed objects to effectively inherit functions by adding them to their internal dispatch table. In this scenario, the spreadsheet object you've created would contain your own display functions, along with pointers to all the functions in the main spreadsheet object.

The Taligent Revolution

Taligent (Santa Clara, CA) is building a new, object-oriented operating system from the bottom up. Everything in the system, from device drivers to applications, will share a common object model. The company expects that this bold approach will produce a clean operating system that will be completely extensible.

continued

Taligent engineers talk obsessively about *frameworks*, by which they mean structures that harness collections of objects. Conventional frameworks include Borland's Object Windows Library, or OWL, and Apple's MacApp. These, however, govern only the creation of applications that run under Windows and the Macintosh. They include classes for windows, controls, menus, and other GUI paraphernalia. By relying on these frameworks to handle simple, standard user interactions, programmers can concentrate on more complex and application-specific tasks.

Taligent's frameworks, by contrast, will reach down into the bowels of the operating system. But with this unprecedented freedom will come an equal measure of responsibility. Programmers will have to tread carefully: If you want to add a derived class that takes control of a certain feature of the system, you have to be sure not to violate any of the assumptions built into the base class.

This principle holds true for any operating system, of course, but I have always found programming in frameworks to be like writing sonnets: There are many possible themes, but there are also some rules that just cannot be broken. Nevertheless, Taligent's radical openness and malleability are alluring.

Complicating the future of Taligent is the company's relationship with its parents, IBM and Apple. Taligent plans to release in 1996 its own operating system, which shares IBM's SOM and its microkernel. But the company also plans to release a personality module that sits in IBM's Workplace OS milieu. It is not clear yet whether, or how, Apple intends to move the Taligent technology onto the Macintosh platform.

Next Got There First

The furor surrounding the object-oriented futures of Microsoft, Apple, IBM, and Taligent can obscure the fact that NextStep delivers many of the same benefits today. It allows you to spin together reusable objects to build a slick user interface in no time flat (see the screen on page 144), and Next supplies powerful frameworks for database and 3-D graphics work.

Over the last five years, NextStep's performance has improved dramatically, says Avadis Tevanian, manager of Next's RISC business unit. A key challenge for developers was to optimize memory allocation so that objects were kept together in memory. Early versions of the system swapped excessively because they couldn't achieve locality of reference with respect to objects.

The NextStep compiler now also performs some object-level optimizations. Each method is assigned a unique number, and objects can invoke a method by number rather than by name. This approach speeds up context switching and makes NextStep extremely responsive to the user.

NextStep also tackles the problem of distributing objects across a network. A technology called Distributed Objects simplifies the task of creating systems of objects that communicate across a network. A programmer makes an object available throughout the network by *vending* it—that is, registering its name in the Network Name Service. Programmers who use Distributed Objects can avoid dealing with the lowest level of interaction with Mach, the network, and RPCs (remote procedure calls).

Next is now making Distributed Objects available on other operating systems, in a form called PDO—Portable Distributed Objects. PDO for HP-UX, which shipped in mid-November, contains the Objective C language compiler (i.e., the language in which NextStep objects are written) as well as code for handling distributed object requests. Next intends to ship PDOs for Data General, NCR, and other Unix platforms and eventually non-Unix operating systems, possibly including Windows NT.

Does the requirement to use Objective C limit the appeal of PDO? Not according to Ricardo Parada, software engineer with Pencom Software. "Nothing beats Objective C for objects," he says. "NextStep is the platform that made me see that C++ is not good enough for OOP."

At press time, Next and SunSoft announced a joint licensing agreement that will marry Sun's developing object technology with the NextStep application environment. Next will freely publish a specification describing OpenStep, an operating system–independent software layer encompassing NextStep APIs and application frameworks. Sun will license the OpenStep application layer from Next, along with development tools including Interface Builder, and will make these standard parts of Solaris. The OpenStep specification will be written in terms of Objective C, but it can also be implemented in C++. "We've been investing for three years building low-level object plumbing," said Sun chairman and CEO Scott McNealy at the joint announcement. "OpenStep gives us the application framework we need to layer on top of that plumbing." In exchange for OpenStep, Sun will license that object plumbing to Next.

The CORBA Connection

Hewlett-Packard, Sun Microsystems, and DEC began experimenting with objects long ago. These companies have now joined with many others to fund an industrywide coalition known as the OMG (Object Management Group), which develops standards for object exchange. The OMG's CORBA (Common Object Request Broker Architecture) lays the groundwork for distributed computing with portable objects. CORBA defines how objects locate other objects and invoke their methods.

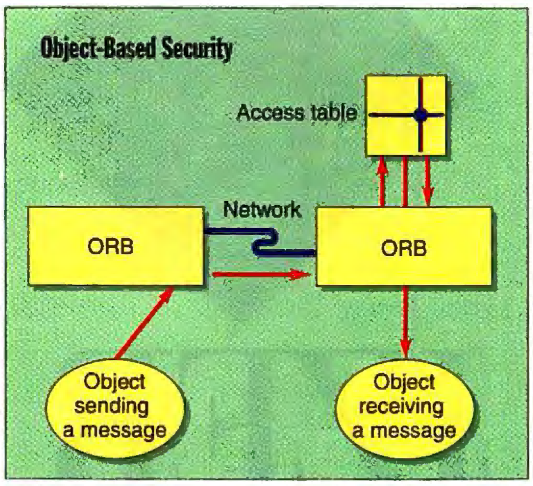
If this sounds strikingly similar to IBM's SOM, it should. SOM is CORBA compliant. If you're using DSOM under OS/2 (or AIX), you'll be able to invoke CORBA-compliant objects running on HP's, Sun's, or other architectures. Does this mean you will be able to edit an OpenDoc object created on the Macintosh from within a container document on a RISC workstation? Probably not. CORBA can guarantee only a low-level mechanism by which objects can invoke other objects. To interact successfully, the two objects also have to understand each other's messages.

The OMG hopes to synchronize the efforts of many leading workstation vendors. SunSoft, for instance, is working with the OMG to transform much of its technology into open standards. SunSoft's work in the realm of distributed objects has yielded a series of Solaris extensions that have been incorporated into the Common Object Services Specification, or COSS, which are now approved as OMG standards.

The naming service links an object to a human-readable name that a programmer or system can use to find the object on a network. The event notification service, which enables objects to synchronize their operations, supports client/server or peer-to-peer interaction. The association service joins objects together into collections. The properties service lets anyone bind annotations to objects. This object-level graffiti could support store-and-forward messaging or store configuration data.

Security in a World of Distributed Objects

The more that we link our computers together, the more difficult our security prob-



In a CORBA environment, *ORBs ensure that only authorized objects can transmit messages. The access table specifies which connections are permitted.*

lems become. The inherent flexibility of distributed object systems brings new security challenges. Designers want to make it easy for one object to call another object, even if the two occupy different address spaces, ZIP codes, and time zones. Speedy communication is critical.

Unfortunately, security gets in the way. There are strong mathematical algorithms for sealing messages from prying eyes and proving that the identity of an object or a person is authentic. But the problem is that these algorithms chew up compute cycles. That's acceptable on an occasional basis—say, once per log-in session—but too burdensome if every object call needs to pay this extra computational price.

Emerging solutions take two basic forms. Novell and Apple are concentrating on public-key algorithms based on patents held by RSA Data Security (Redwood City, CA) and Public Key Partners (Sunnyvale, CA). In these systems, keys come in pairs. One is published while the other is kept private to the owner. A central authority dispenses public keys to users.

The other common method, which is used by Apple, IBM, DEC, and many other Unix manufacturers, is based on the Kerberos system developed at MIT during the 1980s. This system is based completely on private keys that are dispensed by a central, trusted authority. In this case, though, the central authority must provide a new key whenever a secure link between two entities must be generated. In public-key systems, the central authority is consulted only when two computers first communicate.

The latest security provisions come bundled in Apple's latest revision of the Macintosh operating system, called System 7 Pro. Apple is recommending that all users in networked environments shift over to this version because it offers a variety of options for building collaborative environments. The security provisions take two different forms: digital signatures and secure collaborative sessions.

Digital signatures are generated with an RSA algorithm. When you join the network, a pair of keys, one public and one private, are issued in your name. When you want to "sign" a document, you drop it onto the DigiSign program. This action will fetch your private key from disk, where it is kept in encrypted form. You type in a password that decrypts the private key (which is too long for a user to remember), and a signature is then generated and attached to the document's resource fork.

Apple hopes that this technology will reduce the flow of paper in offices. If you want to question the veracity of a signature, you ask the central authority for the person's public key. It will verify signatures generated with the corresponding private key. The only way that someone can forge a signature is by obtaining the private key or the password. Apple has designed the algorithm so the private key is held in memory in unencrypted form only for as long as it's needed.

Object-Based Security

IBM is working with the OMG and with other companies to add a layer of security software on top of the SOM and DSOM object managers. The challenge is to ensure that messages can reach objects only when the sender has the appropriate authorization. The goal is to provide a secure standard that meets or exceeds the Orange Book criteria formulated by the National Security Agency.

IBM's approach is to delegate authentication work to the ORBs (object request brokers) that make connections between the objects over the network (see the figure "Object-Based Security"). While it's possible to add a layer of protection to the objects themselves, this severely constrains an object's reusability in applications that do not require security. IBM plans to embed access control in the ORB, which will filter out unauthorized requests. Programmers can then create objects without worrying about security precautions.

Secure ORBs will maintain access tables that control which outside objects can access objects under its control. The ORB will be able to check the identity of the message sender by using public-key algorithms. It will also negotiate keys for encrypting messages. Messages will be decrypted before they are passed to their target objects.

Windows NT takes a similar approach with its built-in security. Each object's creator sets its access privileges. The object broker in the kernel controls the connections so that only authorized messages get through.

The U.S. government issues standards that specify degrees of security. At level C2, for example, a system guarantees that any object can be made secure at the discretion of its creator. Windows NT systems can be made C2-secure because all interactions must pass through the object dispatcher. The simplicity of the model makes it possible to analyze the system and ensure that there are no "trapdoors" available for anyone to exploit. Sun Microsystems, HP, and DEC also produce operating systems that are C2-secure or better.

Objects Are Closer Than They Appear

The transition to object-oriented operating systems will dominate the rest of this century. Programmers will need to rewrite huge quantities of code to exploit the benefits of these new systems.

The OLE 2.0–compatible applications that are now emerging are an important first step. OLE 2.0 is the carrot and stick that Microsoft hopes will ensure a supply of applications for Cairo when it emerges. The members of the OpenDoc consortium are pursuing a similar strategy that, unlike OLE 2.0, is not tightly coupled to the Windows platform. And Unix vendors, always advanced in their network orientation, are rapidly converging on interoperable CORBA-compliant distributed object systems.

Not everything must be described in the future tense, however. IBM's CORBA-compliant DSOM toolkit is shipping now, as is Next's PDO. Adventurous and forward-looking developers can today explore the kinds of object technologies that will appear on the mainstream platforms of tomorrow. ■

Peter Wayner is a BYTE consulting editor based in Baltimore, Maryland. He can be reached on the Internet or BIX at pwayner@bix.com.

Personality Plus

FRANK HAYES

STEVE LYONS © 1994

The new breed of operating systems won't just do the same old things better. Instead, they'll offer capabilities that we've never expected before. Some of these (e.g., microkernels and objects) will live deep in the bowels of the systems, and users may never know they exist. But one new capability will affect almost every desktop computer user: the ability to run foreign applications.

Currently, add-on software lets Mac and Unix users run DOS and Windows applications. But in the generation of operating systems now emerging, the ability to run foreign software will be a standard part of the system and will work well. Your choice of operating system will no longer drastically limit your choice of applications. The collision of user interfaces that occurs when Mac, Windows, and Unix applications all share the same screen will take some getting used to. Still, multiple operating-system personalities are here to stay, and soon they'll be as standard as mice and menus.

What won't be standard, though, is the way in which operating systems implement their ability to run nonnative applications. OS/2, Windows NT, Unix, Workplace OS, and the Mac will all take distinctively different tacks. These differences will affect how well you are able to take advantage of the wider range of applications that the extra personalities will support.

There are two competing sets of requirements. The mission of a foreign personality is to run existing applications, so it must support them as fully and faithfully as possible. But the needs of those applications may conflict with the design of an advanced operating system. Specialized device drivers may be at odds with the need for security. Memory management schemes and windowing systems may conflict. Business issues (e.g., the cost of licensing code and threats of legal action) also affect the design of foreign personalities. But the biggest potential issue is performance: A personality must run applications at an acceptable speed.

The Emulation Equation

For one computer to run software intended for another (e.g., a Mac running DOS software), the computer must perform instructions that it doesn't natively understand. For example, a Mac's 680x0 processor must execute binary code that was intended for a PC's 80x86 CPU. The 80x86 comes with its own instruction decoder, registers, and internal architecture; it executes each instruction through hard-wired circuitry or by executing a microcode routine within the CPU.

The 680x0 doesn't understand 80x86 code, so typically it has to collect each instruction, decode it to determine what it's intended to do, and perform the equivalent routine using external 680x0 code rather than internal microcode. Because the 680x0 also doesn't come equipped with exactly the same registers, flags, and internal arithmetic and logic units as an 80x86, it must also imitate those elements, either in its own registers or in memory. And it must accurately reproduce the results of each instruction, which requires 680x0 routines specifically written to make sure that the emulated registers and flags will be exactly the same as they would be on a real 80x86 after executing each instruction.

The ability to run Windows and Macintosh software is the order of the day, and the name of the game is "multiple personalities"

For the CPU, it's not hard work, just exacting and very tedious—the sort of job at which computers excel. But it's also very slow work, because the microcode inside a real 80x86 runs at a much faster clip than the external 680x0 instructions that must emulate it. In the time it takes the 680x0 to perform one 80x86 instruction, a real 80x86 CPU might be able to execute dozens of instructions. The result: A DOS program running under pure emulation on a Mac is certain to be incredibly slow compared to one running on a PC.

The problem isn't the Mac, though—Macintosh software being emulated instruction-by-instruction on a Unix workstation runs like molasses, too. The emulation equation is easy to understand: The processor's ordinary performance, minus all the overhead of emulation, will equal how much work it can do. Thus, unless the processor performing the emulation is spectacularly faster to compensate for the emulation overhead, the software running under emulation will simply be very, very slow.

A Dime a Dozen

What makes the new personalities better than emulation in the past? Faster processors help, of course. But the big difference is that many of today's applications run under GUIs like Windows, the Mac, or Unix's Motif. That means the new personalities can "cheat" on the emulation process.

An application running under a GUI spends much of its running time doing

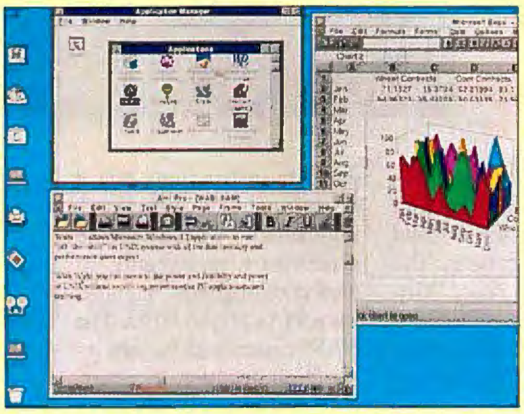
SunSelect's Wabi vs. Insignia Solutions' SoftWindows

SunSelect's Wabi (Windows Application Binary Interface), which will be bundled with many Unix workstations, uses the workstation's normal X Window System display protocols for creating the images called for by a Windows application and Unix's usual facilities for handling files, memory, and other resources.

Wabi is based on technology acquired by SunSelect from Praxsys Technologies, but it functions much like other personality translators. While working its way through the code in a Windows application, Wabi decodes and mimics individual 80x86 instructions until it encounters a call to a DOS or Windows function. Then the emulator switches to native mode, performing the DOS or Windows function by making the appropriate calls to X, Unix, or other facilities. The technical challenge comes in translating the parameters of each Windows call to the appropriate format for Unix and then translating the results from the function call into the appropriate information to be returned in the appropriate Windows data structures.

The first release of Wabi claims to support the Windows 3.1 API, with DDE and OLE supported only as external DLLs that must be interpreted by Wabi's 80x86 emulator. Networking is limited to access to remote file systems and printers. SunSelect says improved network support and native versions of DDE and OLE will come in a future release of Wabi.

Windows applications running under Wabi have the look of an X-based Unix GUI such as Motif or OpenLook, rather than that of Microsoft Windows. And instead of running the entire Windows desktop environment within a window,


Wabi running Windows applications on the Solaris desktop.

as Insignia Solutions' SoftPC and SoftWindows currently do, Wabi opens a new window on the Unix desktop for each Windows-based application. Using a standard X display means both text and graphics can be cut and pasted between Windows and Unix applications (although most Unix applications can't automatically convert to and from the Windows bit-map format).

However, SunSelect isn't religious about its X implementation of Windows. To make sure TrueType fonts are properly handled for the Windows applications, the company has licensed font-handling technology from Bitstream. As a result, when a Windows application issues a call to display text in a particular TrueType face, Wabi converts the request to X calls but also provides the appropriate fonts for the display.

Wabi can't currently handle plenty of Windows-related features, including multimedia extensions, ODBC (Open Database Connectivity), MAPI (Messaging API), and networking beyond access to remote file systems and printers. Are those limitations Wabi-killers? SunSelect doesn't think so, arguing that Wabi's purpose is to run the popular Windows applications Sun's customers have asked for, not to convert Unix into a close copy of Windows. The current list of "Wabi-certified" applications is short. Only 13 packages from Lotus, WordPerfect, Microsoft, Borland, and other major Windows software vendors are guaranteed to run under Wabi.

According to SunSelect's director of research and development, Andy Halford, another 50 packages seem to work fine, but they haven't been run through the Wabi testing and certification program. Software that uses APIs Wabi doesn't support may fail to install or exit gracefully with an option to close files—or even cause Wabi to abort.

But a Microsoft-backed competitor thinks Wabi's approach is far too limited. The day before SunSelect unveiled Wabi, Microsoft launched a preemptive strike by announcing it would license Windows source code to Insignia Solutions. The product that Insignia produced from that agreement, SoftWindows, runs Windows applications on Unix workstations, but there the similarity to Wabi ends.

SoftWindows is actually Windows 3.1 and MS-DOS, recompiled for Unix. Initially, SoftWindows fully supports OLE, DDE, and DLLs; Insignia says it is now working on multimedia and other extensions. The image that appears in a some very predictable things. It repeatedly makes calls to the GUI's libraries to manipulate windows and perform other GUI-related functions. And that's where a personality can make up for some of the time lost doing instruction-by-instruction emulation. A carefully crafted personality can come complete with libraries that mimic the GUI's own internal libraries but that are written in native code. Some vendors call this approach *translation*, to distinguish it from the slower process of emulating code one instruction at a time.

For example, on a Mac executing a Microsoft Windows program, performance might be very slow when it's interpreting 80x86 instructions. But when a call is made to open a window, the personality module could switch to a precompiled 680x0 window-opening routine. Because the GUI libraries don't have to decode and imitate each 80x86 instruction, performance can speed up dramatically in sections of the code that call the GUI's ABI (Application Binary Interface). The result is that in those sections of the code, the application can approach (or possibly exceed) its performance on its native processor.

And there's a *lot* of code that calls the GUI ABI in typical applications today. Apple claims that a Mac application spends up to 90 percent of its processing time performing Mac toolbox routines, rather than executing code that's unique to the application. SunSelect says that Windows applications spend 60 percent to 80 percent of their time in the Windows kernel. As a result, there can be a much smaller performance penalty for emulation of GUI-based applications. In fact, SunSelect claims that its new Windows personality, Wabi (Windows Application Binary Interface), can outperform real Microsoft Windows on the same hardware when running some benchmarks, thanks to highly optimized libraries.

The rise of GUIs has also resulted in another change in the way most desktop applications software is written today. Until the advent of the Mac, most desktop software treated operating-system calls with a sort of "do-it-yourself" philosophy. If the programmer didn't think the operating system would perform the routine fast enough, he or she would often dispense with the available operating-system calls and write an equivalent routine that directly manipulated hardware or software. This approach was commonly used for time-critical functions like display scrolling and getting data from a serial port.

"Programming on the metal" for performance was a nightmare for emulator writers, because they had to mimic software that was going directly to hardware that usually didn't exist on the computer doing the emulation. It was also a major problem for computer makers such as IBM and Apple, because it locked them into using exactly the same hardware architecture in generation after generation of the IBM PC and the Apple II. Changing many hardware details was out of the question, even if the changes would mean dramatic improvements, because changes would also break lots of software.

Lessons Learned

Apple learned its lesson from the Apple II experience. With the Mac, Apple

SoftWindows window is that of a complete Windows desktop, and because the source code is the same as the original 80x86 version, every nuance of Windows is preserved. When SoftWindows' 80x86 emulator reaches a Windows function call, it doesn't simply mimic the function. It actually performs it, at full processor speed, with appropriate calls made to Unix instead of DOS.

Because it uses authentic Windows source code, SoftWindows is able to run a far wider range of Windows applications than Wabi. By comparison, says Insignia, Wabi offers very little.

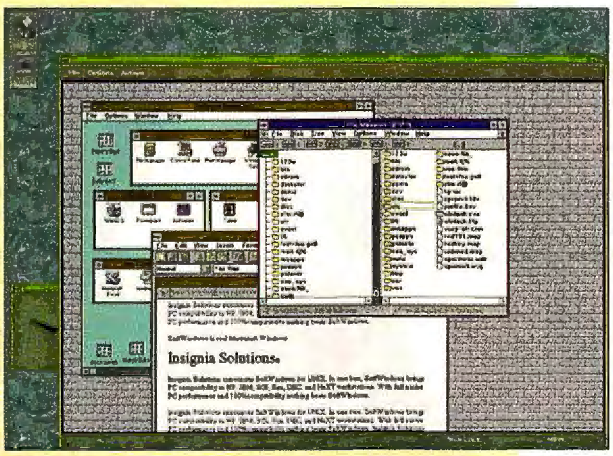
But according to SunSelect, Wabi does claim one major advantage over SoftWindows: blinding speed. Executing every line of authentic Windows code for each function creates an awful lot of overhead, particularly because Windows was designed as a 16-bit application running on top of MS-DOS and was built to perform its own memory management and other advanced functions. By contrast, Unix is a 32-bit operating system that has finely tuned memory management and other facilities.

SunSelect argues that by using Unix to mimic Windows rather than slavishly performing every line of the authentic code, Wabi can outperform genuine 80x86-based Windows. A demonstration performed at SunSelect's original Wabi announcement appears to bear out the claim. Running the Wintach benchmark, a PC running the Intel version of Solaris with Wabi performed 50 percent faster than an identical PC running Microsoft Windows, according to SunSelect.

In response, Insignia points out that Wintach is just one benchmark, and it's strongly geared to graphical functions—the kind of functions where Wabi would be expected to do well. Insignia claims it uses a battery of benchmarks to make sure its RISC Unix versions of SoftWindows will perform at least as well as a 25-MHz 486-based PC in every area. The company says it has not yet benchmarked SoftWindows against Wabi but that the two initially look "competitive."

Ironically, SunSelect is an Insignia customer. The company sells an enhanced version of Insignia's SoftPC as SunPC, and SunSelect acknowledges that for SPARC customers who need more complete PC emulation, that's the way to go. But for those who need to run only the top Windows applications, says SunSelect, Wabi is a better solution.

The choice between SoftWindows and Wabi comes down to whether a customer wants to run full-scale Windows or full-speed Windows applications.



SoftWindows running Windows applications.

worked hard to discourage programmers from "going to the metal" or otherwise departing from a strict set of programming guidelines. (Apple's programmers weren't immune to the temptation to program on the metal, though. Some Apple telecommunications software for early, relatively slow Macs programmed the hardware directly.) The result of that discouragement was that Mac applications software was much less likely to break the rules than PC software. With fewer hardware dependencies, Apple has been able to evolve the architecture of the Mac over time.

The biggest reason programmers used the Mac's "toolbox" of GUI library routines was not a stick, but a carrot. The toolbox routines were so complex and powerful that using them was significantly easier than writing your own version of the code. Microsoft Windows also included a powerful GUI ABI, as did Microsoft and IBM's OS/2 Presentation Manager and Unix GUIs based on the X Window System. When Windows rocketed to popularity in 1990, the tide turned for emulation. Finally, a large body of applications software that spent a large part of its time in a GUI ABI could be mimicked.

With the technical barriers down, there are pressing business reasons why vendors believe multiple personalities are a crucial part of any successful new operating system. DOS, Windows, and Mac programs pack the shelves in software stores; obtaining shelf space for a new incompatible type of software is practically impossible. More important, users have plenty of Windows and Mac software, and they're not about to give up the software they know well, no matter how impressive a new operating system promises to be. In fact, for an increasing number of business customers, the ability to run particular PC applications (e.g., Lotus 1-2-3 and WordPerfect) is becoming a standard requirement for desktop computer purchases, even if the purchase also requires technical applications available only under Unix.

Luckily, the modularity of the new generation of operating systems makes it far easier to support multiple personalities. Unlike older operating systems, which often consist for all practical purposes of a single large block of code divided into arbitrary parts, newer systems are modular, with clearly defined interfaces between the parts. That makes it much easier to design additional modules that bundle together processor emulation and GUI library translation.

So the pieces have all come together, both technological (software style, processor speed, and modular operating systems) and business (popular "must-run" software packages). Multiple personalities are the wave of the future for operating systems.

Who's Got What?

Among the advanced operating systems that will specifically incorporate multiple personalities are IBM's OS/2 2.x and Workplace OS; Microsoft Windows NT; the PowerOpen Association's PowerOpen; and versions of Unix from Sun Microsystems, IBM, and Hewlett-Packard. In addition, some companies are repackaging their user interfaces as personality modules, and still other vendors offer emulation and personality-translation products that can run as applications.

continued

PUT A NEW FACE ON YOUR HOST DATA...

KEATERM VT EMULATION GIVES YOUR HOST DATA A FACELIFT—THE WINDOWS INTERFACE!

Before KEAterm, host data was dull, colorless, and limited by aging host applications. KEAterm transforms your host data—making it better looking, and more useable.

Apply the power and ease of Windows tools, like Excel, 1-2-3, WordPerfect, Visual Basic... to your host data using KEAterm's DDE, hot links, file transfer, and user-defined menus and dialog boxes. And you'll get more done faster with multiple KEAterm sessions.

Advanced features include a powerful macro language and definable on-screen button pads, to make your work even easier. KEAterm speeds your work with high throughput over your serial port or installed TCP/IP or LAT network. ■

KEAterm 420—a powerful link to your VAX and UNIX text applications.

KEAterm 340—for applications requiring ReGIS, Tektronix, or sixel graphics.

KEAterm—for **Windows NT** available soon!

Empower your desktop now!

Call 1-800-663-8702

KEA Systems Ltd.
3738 North Fraser Way, Unit 101
Burnaby, B.C. Canada V5J 5G1
Phone: (604) 431-0727
Fax: (604) 431-0818

KEAterm 340

KEA

KEAlink, KEAterm, ZSTEM, PowerStation, KEA and their respective logos are trademarks of KEA Systems Ltd. All other brand and product names are trademarks or registered trademarks of their respective holders. Copyright © KEA SYSTEMS LTD. 1993. All rights reserved.

Circle 256 on Inquiry Card.

EXISTING AND FORTHCOMING OPERATING SYSTEMS OFFERING MULTIPLE PERSONALITIES

	OS/2 2.x[1]	Workplace OS[2]	Windows NT[3]	PowerOpen[4]	Unix (with Wabi)[5]
Vendor:	IBM	IBM	Microsoft	PowerOpen Association	SunSoft (Solaris), IBM (AIX), Hewlett-Packard (HP-UX), USL (SVR4.2)
Availability:	Now	Future (this year)	Now	Future	Now (Solaris)
Personalities available:	DOS, Windows 3.1	DOS, Windows, OS/2, AIX (Unix), others	DOS, Windows 3.1, Win32, OS/2 1.x, Posix	Macintosh, AIX (Unix)	Windows 3.1
Look and feel:	OS/2 or complete Windows environment within a window	OS/2 Workplace Shell or Unix CDE	Windows	Motif; Mac desktop in a self-contained window	Motif or OpenWindows
Applications supported:	Windows 3.1 applications and device drivers	Unknown (prerelease)	DOS and Windows applications that do not require access to hardware; character-based 16-bit OS/2 applications	RS/6000 AIX, System 7	13 Windows applications from major vendors "certified"; others may run

[1] OS/2 2.x is based on code licensed from Microsoft. OS/2 for Windows incorporates no Microsoft code.
[2] Additional proposed personalities include Mac and BSD Unix. Currently a product in development.
[3] Posix support requires recompilation of source code.
[4] Mac support via Macintosh Application Services.
[5] Wabi Windows personality was reverse-engineered from Windows API. Wabi has been licensed to IBM, Novell, and HP and will be available with every Sun workstation and copy of Solaris for Intel.

THIRD-PARTY PERSONALITY SOFTWARE

	Macintosh Application Services[1]	Liken[2]	Equal Application Adapter[3]	SoftPC[4]	SoftWindows[5]	Merge[6]
Vendor:	Apple	Andataco	Quorum Software Systems	Insignia Solutions	Insignia Solutions	Locus Computing
Availability:	Future (this year)	Now	Now	Now	Now	Now
Operating systems supported:	Unix (PowerOpen, others)	Unix (Solaris, HP-UX)	Unix (Solaris, Silicon Graphics)	Mac and Unix (many varieties)	Unix (Solaris and HP-UX now; AIX, Silicon Graphics, and DEC OSF/1 in March)	Unix (80x86 versions)
Personalities available:	Mac	Mac	Mac System 7	DOS, Windows 3.1	Windows 3.1	DOS
Look and feel:	Mac using X Window System widgets	Complete Mac desktop in a window	Motif or OpenWindows	Windows, character-mode DOS in a window	Complete Windows environment in a window	Character-mode DOS
Applications supported:	Unknown	Monochrome System 6–based applications	Microsoft Word and Excel "certified"; others run but are not guaranteed and may end Equal session unexpectedly	Most DOS and Windows applications that do not require direct hardware access	Most existing Windows applications and device drivers that do not require direct hardware access	Most DOS applications that do not require direct hardware access

[1] "Statement of direction" from Apple.
[2] Requires a copy of System 6.0.7. Emulates 680x0 CPU and Mac hardware environment.
[3] Reverse-engineered from System 7 specifications. Runs Mac applications but does not mimic entire Mac environment.
[4] Emulates 80x86 and PC hardware environment.
[5] Based on Windows source code licensed from Microsoft.
[6] Microsoft Windows can be run over Merge.

Perhaps the most familiar multiple-personality operating system is also the one that opened the floodgates by showing that the ability to run other systems' software can be a big plus. OS/2 2.0 ran DOS and Windows 3.0 applications, and version 2.1 improved on this, upgrading to Windows 3.1 software and making the Windows windows a regular part of the desktop.

At first glance, IBM developers would seem to have had a comparatively easy task in adding the Windows personality to OS/2. After all, like Windows, OS/2 runs on 80x86 CPUs, so no processor emulation was required. In addition, IBM had access to actual Microsoft Windows source code and the right to use it, for a licensing fee, in OS/2. So IBM's work largely consisted of integrating the Windows code into OS/2.

But it still wasn't easy. The requirements of the two environments created difficult problems, some of which IBM has never satisfactorily resolved. For example, Windows incorporates its own memory manager. So does OS/2. Unable to modify the Windows code to use OS/2's memory management services directly, the OS/2 developers settled on using the Windows memory manager within the OS/2 memory manager. Windows' manipulations of memory can spill over into the OS/2 swap file. Similarly, OS/2's "seamless Windows" mode required major work on the display drivers to enable the two window systems to share screen real estate.

Windows NT offers five operating-system personalities: DOS, Windows, an advanced 32-bit version of Windows, OS/2 1.x, and a Unix-like personality that meets the IEEE's Posix.1 specification. NT runs on several different CPUs, including the Mips R4000/R4400 and DEC's Alpha, as well as the 80x86. To run DOS and Windows applications on non-80x86 platforms, NT incorporates emulation technology licensed from Insignia Solutions, which also makes the DOS emulator SoftPC for the Mac and Unix workstations. (NT's OS/2 personality is not supported on non-80x86 processors.)

Naturally enough, to provide the ability to run Windows applications, Microsoft used its own Windows source code, modified and recompiled for each CPU that NT runs on. The 16-bit Windows and DOS personalities run on top of the 32-bit Windows (Win32) NT subsystem. On 80x86 machines, where the CPU is not emulated, DOS and 16-bit Windows applications run in V86 mode, and 16-bit calls are "thunked" (converted to 32-bit versions) and serviced by Win32.

NT's major trade-off in DOS and Windows support is that, in keeping with NT's security and reliability goals, device drivers and other DOS and Windows programs are not allowed access to the hardware. As a result, some DOS and Windows programs simply won't run under NT. (In contrast, OS/2's DOS and Windows support allows more complete DOS and Windows support, but for that capability trades away robustness.)

NT's OS/2 support has special limitations compared to the DOS and Windows personalities, but it is still a thoroughly usable version. It is available only on 80x86 NT, does not support the PM GUI, and is designed to handle only software written for OS/2 1.2 and earlier versions, which limits applications to 16-bit versions. In practice, though, NT's OS/2 personality can run current versions of many OS/2 packages—particularly server applications, which don't require PM.

In contrast to the OS/2 personality, NT's Posix personality isn't actually mimicking an existing operating system at all. Although there are versions of Unix (on which Posix is modeled) for each CPU that NT runs on, NT's Posix can't run shrink-wrapped Unix software; it requires programs to be recompiled before running.

The Unix Strategies

While Windows NT can't run Unix binaries, some Unix vendors are convinced they need the ability to run Windows software. That ability has been available for several years through third-party software like SoftPC (now available with Windows), which runs on Macs and Sun, HP, IBM, Next, and Silicon Graphics Unix workstations. On 80x86-based computers, Locus Computing's Merge also enables DOS applications to run under Unix. Merge runs a standard copy of Windows on top of the DOS environment.

In addition, Insignia's new SoftWindows was scheduled to begin shipping in December. SoftWindows uses a recompiled version of the Windows source code to speed up Windows applications running on Sun, HP, IBM, DEC, Next, and Silicon Graphics Unix workstations. If that approach sounds familiar, it should: It's almost exactly the same approach used for non-80x86 versions of Windows NT. But while SoftWindows and NT are conceptually close cousins, NT can also run 32-bit Windows code, while SoftWindows is limited to running16-bit Windows applications.

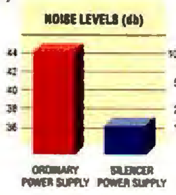
However, the most aggressive approach to bringing Windows and Unix together comes from Sun Microsystems' SunSelect division, which has developed Wabi. While SoftWindows uses recompiled Windows source code from Microsoft, Wabi is an attempt to reverse-engineer Windows based on its functional specifications, with all operating-system-related functions (e.g., display, memory management, and interprocess communication) handled by Unix. Instead of the Windows desktop, each Windows application running under Wabi appears in its own screen window and uses the Motif or OpenLook screen appearance rather than that of Microsoft Windows.

The result is a mixed success. SunSelect initially guarantees that Wabi can run only the most popular Windows software, including Lotus 1-2-3 and Ami Pro; WordPerfect; Microsoft Word, Excel, PowerPoint, and Project; Borland Paradox and Quattro Pro; Aldus PageMaker; Harvard

There are two ways to quiet a noisy computer.

The irritating whine of a noisy power supply can really rattle your nerves! You could seek relief with heavy-duty ear muffs, but the *real* solution is a **Silencer®** power supply. Appreciated by users since1986, its custom, high-efficiency fan & low-turbulence circuitry reduce noise by up to 84% (8 db). You'll have clean, reliable ultra-quiet power!

Silencer 205 (for slims)	**$119**
Silencer 220 (for desktops)	**$129**
Silencer 270 (for towers)	**$179**



"Power supplies you can barely hear" *PC World*, July 1993
"If you value quiet, get the Silencer" *Business Week*, Sept. 18, 1989

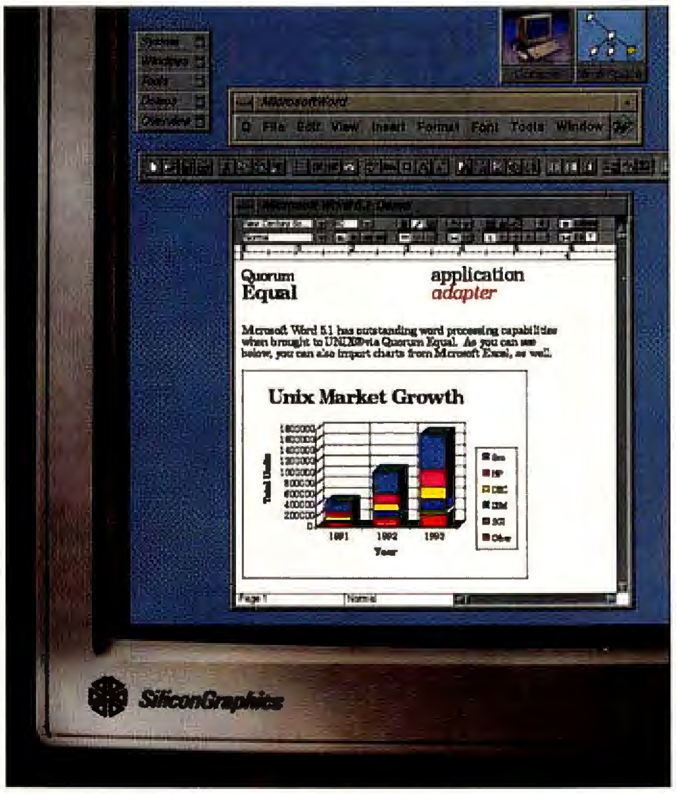
PC POWER & COOLING, INC.
5995 Avenida Encinas, Carlsbad, CA 92008, (619) 931-5700, (800) 722-6555, Fax (619) 931-6988
We accept Visa, MC, COD, or PO on approved credit. ©1993 PC Power & Cooling, Inc. Silencer is a registered trademark of PC Power & Cooling, Inc.

Circle 260 on Inquiry Card (RESELLERS: 261).

Graphics; CorelDraw; and Procomm Plus. The company says that the list of "certified" applications will grow. In the meantime, while some noncertified applications will run, others may not install, or may fail while the application is running due to use of unsupported API calls.

SunSelect says its focus is on running popular applications rather than mimicking Windows in its entirety. But all Windows applications function in a complex environment, with subtleties that may show up only when Wabi's developers tackle support for applications outside the most-wanted list. In addition, Windows will continue to be a moving target; SunSelect may be hard-pressed to keep up with future changes required by new versions of Windows software.

However, Wabi has one huge advantage in any popularity contest for Windows-on-Unix software: SunSoft is making Wabi available with every copy of its Solaris version of Unix, and SunSelect has licensed the product to IBM, HP, and Novell to include in their versions of Unix. If all these vendors include Wabi in their systems as Sun does, Wabi will be shipped with more than 70 percent of all Unix workstations.



Quorum
Equal

application
adapter

Microsoft Word 5.1 has outstanding word processing capabilities when brought to UNIX® via Quorum Equal. As you can see below, you can also import charts from Microsoft Excel, as well.

Quorum Software Systems' Equal running the Macintosh version of Microsoft Word on a Silicon Graphics workstation.

Not to be outdone, Apple is working on its own Mac personality translator to run on Unix systems. The first version, Macintosh Application Services, will run on PowerPC-based workstations running the PowerOpen version of Unix. MAS will let PowerOpen workstations run both Unix applications and shrink-wrapped software intended for 680x0-based Macs. (MAS should not be confused with the new PowerPC-based Macs, which also use processor emulation and GUI translation to run 680x0 Mac software.)

MAS will appear as a "Macintosh window" on PowerOpen-based workstations. Although Apple says that MAS will be compatible with X, Mac applications running under MAS will still have the distinctive Mac look and feel.

In addition, Apple has announced that it will eventually support other Unix workstations. Apple hasn't released details of its plans, and they clearly fall under the category of future product development. However, Sun, HP, and IBM have already said they hope to use the forthcoming Apple technology to let their Unix workstations run unmodified shrink-wrapped Mac software.

In the meantime, two ISVs (independent software vendors) are already emulating the Mac on Unix systems—although with limits. Andataco's Liken is a pure processor emulator; it runs on Sun and HP workstations and mimics the Mac's 680x0 CPU, as well as the Mac hardware environment. However, Liken doesn't try to copy the Mac's toolbox GUI libraries; for that, you need a copy of System 6.0.7.

In contrast to Liken, Quorum Software Systems' Equal is designed to mimic both the 680x0 processor and all Mac system calls, so that Mac applications can run on Sun and Silicon Graphics Unix workstations. Like Wabi, Equal puts each Mac application in its own window, using X to display Motif- or OpenLook-style window decorations. Also like Wabi, Equal currently has a limited set of "certified" applications. Initially, it includes only the Mac versions of Microsoft Word and Excel, although Quorum plans to expand the list of certified software early this year to include Microsoft PowerPoint, QuarkXPress, and other popular Mac software. (According to Quorum, many "uncertified" Mac applications run with no problems.)

Closing the circle is IBM's Workplace OS, the OS/2 successor based on the Mach 3.0 microkernel. Standard Workplace OS personalities will include Unix and OS/2 (along with its DOS and Windows personalities). But IBM hints that other personalities may also be available for the system. Because the Workplace OS interfaces are being developed in close communication with Taligent, the IBM/Apple joint venture to develop an object-oriented operating environment, both Taligent and the Mac GUI are likely candidates as Workplace OS personalities.

Who Wins, Who Loses

The ability to run Windows and Mac software is no longer a minor consideration when it comes to advanced operating systems. But beyond that simple point of agreement lie a welter of strategies for putting the multiple-personalities idea to work—and some of those strategies are diametrically opposed to others. A careful examination of the strategies operating-system vendors are using makes it apparent that there's no single correct way to implement multiple personalities.

In the case of Unix, the personality translator is typically designed to float along the surface of the operating system, like any other application. For more recent operating systems like Windows NT and Workplace OS, the personality module is much more closely linked to the operating system, although it is still highly modular. And for OS/2, with its simpler, less modular structure, the personality capability appears to be deeply embedded in the operating system.

But while operating-system vendors are juggling their approaches to run the largest number of popular applications most effectively, the biggest impact of the trend toward multiple personalities may be on applications software developers. Windows and Mac applications are likely to sell slightly better than before. The big winners will be those Windows applications that are already the most popular, because the ability to run them will be

Windows NT and Workplace OS: Plug It In

While Unix personality modules are designed to function as if they were applications, both Microsoft's and IBM's entries in the portable 32-bit operating-system sweepstakes take a more integrated approach. Microsoft Windows NT and IBM's forthcoming Workplace OS have been specifically designed to support emulation of multiple operating-system personalities, although the difference between the two systems' approaches is striking.

Windows NT supports five operating-system personalities: MS-DOS, 16-bit Windows, OS/2 1.x, Posix, and 32-bit Windows. All five personalities are implemented as NT "environment subsystems"; each runs in its own protected user space. The Win32 subsystem handles display, keyboard, and mouse support for the other four personalities.

DOS and 16-bit Windows applications run via VDMs (virtual DOS machines), each of which emulates a complete 80x86 computer running MS-DOS. In NT, a VDM is a Win32 application; thus, like a typical Unix personality module, NT DOS and 16-bit Windows applications effectively float in a layer directly above the Win32 subsystem.

The OS/2 and Posix subsystems are a different matter. As full-scale NT subsystems themselves, they communicate with the Win32 subsystem for user input and output, but they also communicate directly with the NT Executive for other operating-system services. The OS/2 subsystem can run many current character-mode OS/2 applications, including OS/2 SQL Server, and it supports named pipes and NetBIOS.

But the Posix subsystem is remarkably limited, despite direct access to kernel services. Posix applications must be compiled specifically for Windows NT; NT does not support binary code intended for any other Posix-compliant operating systems, such as Unix. In addition, NT's Posix subsystem does not directly support printing, does not support network access except for remote file systems, and does not support any facilities of the Win32 subsystem such as memory-mapped files or graphics.

Compared to NT, IBM's forthcoming Workplace OS uses a more straightforward organization. While some NT personalities go through the Win32 subsystem and others deal directly with the NT kernel, all Workplace OS personalities have direct access to kernel services. Workplace OS currently supports three personality servers: an OS/2 server for OS/2 applications, an AIX server that mimics IBM's version of Unix, and an MVM (multiple virtual machines) server for DOS and 16-bit Windows applications.

Workplace OS is built on a version of Mach 3.0. The IBM microkernel supplies only a very limited set of services; it is essentially a software backplane into which other modules, called servers, connect. The personality servers function exactly like any other Workplace OS servers. Each runs in its own protected memory space and communicates directly with the microkernel and, through it, other servers.

However, all personality servers are not created equal. IBM initially plans two versions of Workplace OS, one the OS/2 Workplace Shell, the other, Unix CDE (common desktop environment). In each case, the *dominant* personality will do double duty, providing both the capabilities required for its own applications and the desktop GUI and default execution semantics for the other personalities. On a standard Workplace OS system, the OS/2 (or Unix) personality is dominant. The other personality servers, known as *alternative* personalities, don't contain code to provide these services.

However, dominance is entirely arbitrary in Workplace OS. The Workplace OS could be given a Windows look and feel, although IBM has no plans to do so. IBM says the server interfaces for Workplace OS will be published, so constructing dominant and alternative personalities will be practical for ISVs (independent software vendors). Additional personalities can also be added by IBM or other vendors; although none have been announced, a Mac personality is rumored as a future addition.

In practice, announcements and demonstrations are currently the limit of Workplace OS's functionality, because it is a product in development rather than a shipping package like NT. In recent demonstrations, for example, Workplace OS's Unix and DOS personalities were both character-based, and users could only hot-key between them and the OS/2 GUI.

Technically, both Windows NT and Workplace OS use modular subsystems to support multiple operating-system personalities. Paul Giangarra, lead architect for Workplace OS, is enthusiastic about the idea of other software vendors developing additional personalities (or, alternatively, personality-neutral services). Microsoft's director of business development, Bob Kruger, says the whole reason NT includes Posix support is to demonstrate that subsystems can be added, either by Microsoft or other vendors, that connect directly to the NT Executive without running as Win32 applications.

In fact, the two approaches seem very comparable at a technical level. Then why does Workplace OS's approach to multiple personalities seem so robust, promising the potential ability to run every significant desktop operating system, while NT's non-Windows personalities seem thoroughly undeveloped? One reason may be that it's easier to create a robust plan than a working operating system with robust implementations of multiple personalities.

But there's also clearly a difference in business philosophy. IBM is pursuing multiple personalities, while Microsoft appears to be discarding them. "How many people are actually going to write a Posix application?" asks Kruger. And he downplays NT's ability to run OS/2 applications: "At the end of the day, people will buy Windows NT because it runs Windows," Kruger insists. It's true that with good support for Windows applications, NT already has many of the benefits that multiple personalities promise. But only time will tell if a Windows-only philosophy will help or hurt NT in its competition with other advanced operating systems.

A Better OS/2 Than OS/2?

Ironically, the first major operating system to demonstrate the commercial value of supporting multiple personalities is now demonstrating a new way to support them. OS/2 was a serious disappointment to development partners Microsoft and IBM when it was first released. When it was first introduced, analysts predicted that within five years, OS/2 would account for more than half the sales of business PCs, displacing MS-DOS as king of the desktop. Instead, early versions of OS/2 sold fewer than a half-million copies per year—a tiny fraction of expectations. And with OS/2's downfall came the collapse of the close relationship between IBM and Microsoft.

So when IBM relaunched OS/2 in 1992, Big Blue needed an edge. It found that edge by beefing up OS/2's ability to run DOS-based applications software and adding support for Windows applications. While OS/2 1.x offered only a single window for running DOS software, version 2.0 let users run several DOS sessions at once. Windows support in version 2.0 was initially limited to running Windows 3.0 on a full screen, but OS/2 eventually supported both "seamless" Windows applications (each appearing in its own desktop window) and, in version 2.1, support for Windows 3.1 applications.

OS/2's DOS and Windows support came through MVM (multiple virtual machines), an OS/2 subsystem that could imitate a series of DOS PCs. In contrast to the modular approach to multiple personalities used by Unix, Windows NT, and Workplace OS, OS/2's DOS and Windows support was firmly embedded in the operating system's code, which seriously limited its flexibility in adding new operating-system personalities.

What proved to be most important, though, was simply that DOS and Windows support was there. Despite a dearth of OS/2-specific software, OS/2 sold some 2.5 million copies since OS/2 2.0 appeared—far more than in its previous history. While that was less than one-quarter of Microsoft's annual sales of Windows, it represented an astonishing comeback for OS/2 and provided convincing proof that the ability to run popular software could prove to be the difference between success and failure for a new operating system.

The comeback came at a high price. OS/2's Windows support used source code that was provided to IBM by Microsoft as part of the companies' technology-sharing agreement. To use the Windows code, however, IBM was required to pay a royalty to Microsoft for every copy of OS/2 that the company shipped. Although IBM never made public the details of the license, the company has reportedly paid Microsoft $20 per OS/2 copy, or more than $50 million since launching OS/2 2.0. Also, that royalty fee pushed OS/2's list price to more than $200.

But a new version of OS/2 changes both the economics and the technology of its Windows support. Code-named Ferengi when it was under development at IBM's Personal Software Products Division in Boca Raton, Florida, the new version is officially named OS/2 Special Edition for Windows, or OS/2 for Windows for short. As its name suggests, it functions as an upgrade to OS/2 for users who own Microsoft Windows. To install, it requires a system with DOS 5.x or higher and Windows 3.1. Once in place, OS/2 for Windows loads the actual Windows environment, modifying it on the fly, so that Windows support is virtually identical to that under previous versions of OS/2.

The business impact of OS/2 for Windows is clear: Because it incorporates no Microsoft Windows code, IBM pays no royalty to Microsoft. As a result, the list price of the package is less than half that of conventional OS/2.

The technical impact may be just as dramatic, at least for IBM's development team. In effect, OS/2 for Windows lifts up Windows and slips an OS/2 jacket around it. That approach will pose a major challenge for IBM developers with each new release of Windows; developers will have to work feverishly to upgrade OS/2 for Windows to tweak the new Windows binaries correctly. Still, their efforts may be no greater than the work required to integrate a new version of the Windows source code would have been.

Whether IBM's new OS/2-jacket approach to Windows support will have as great an impact on OS/2 sales as the improved DOS and Windows support of OS/2 2.0 remains to be seen. What is clear is that OS/2 for Windows effectively turns OS/2's DOS and Windows inside out.

bundled with a large percentage of Unix workstations in the form of Wabi. Ironically, because they are so popular, the additional software sales may not make a big impact on them.

And the big losers? They're likely to be single-user productivity applications written specifically for Unix. Unix software developers already face major problems. Popular Unix workstations sell in the hundreds of thousands, not millions (like the Mac) or tens of millions (like the PC). Few software retailers carry any Unix applications at all. The combination of low volume and limited distribution means that Unix software vendors will be hard-pressed to compete against similar Windows or Mac programs. That could spell the end of the line for applications that don't take advantage of the special features of Unix—or any other advanced operating system.

In the end, the real impact of multiple personalities will be on users, in the form of easier access to better software and more freedom of choice in operating systems. That may not be great news for all operating-system or applications vendors. But for users who have ever needed software they couldn't run, multiple personalities are an important step toward sanity. ■

Frank Hayes is a writer, communications consultant, and former West Coast news editor for BYTE. You can contact him on BIX as "frankhayes."