Experience with the Development of a Microkernel-Based, Multiserver Operating System

Freeman L. Rawson III
IBM Austin

Abstract

During the first half of the 1990s IBM developed a set of operating system products called Workplace OS that was based on the Mach 3.0 microkernel and Taligent's object-oriented TalOS. These products were intended to be scalable, portable and capable of concurrently running multiple operating system personalities while sharing as much code as possible. The operating system personalities were constructed out of a set of user-level personality and personality-neutral servers and libraries. While we made a number of important changes to Mach 3.0, we maintained its fundamentals and the multi-server design throughout our project. In evaluating the resulting system, a number of problems are apparent. There is no good way to factor multiple existing systems into a set of functional servers without making them excessively large and complex. In addition, the message-passing nature of the microkernel turns out to be a poor match for the characteristics of modern processors, causing performance problems. Finally, the use of fine-grained objects complicated the design and further reduced the performance of the system. Based on this experience, I believe that more modest, more targeted operating systems consume fewer resources, offer better performance and can provide the desired semantics with fewer compromises.

Introduction

At the beginning of the 1990s IBM was interested in reducing the number and diversity of its non-mainframe operating systems. IBM was investing in OS/2 and PC DOS for Intel-based PCs, AIX for the RS/6000 workstation and server line and OS/400 for the AS/400. Moreover, none of them offered all of the characteristics that seemed necessary. As a result IBM made a major investment in a set of new technologies intended to create an operating system

- that was easily portable from processor to processor and platform to platform
- whose interfaces and implementation scaled from very small embedded systems to very large clusters and MPs
- which supported multiple operating system personalities concurrently and could be configured to provide the "look and feel" of any operating system in the set while maintaining a single system image
- that shared as much code as possible among the personalities
- that offered full compatibility with all of the operating systems in the set and with DOS/Windows 3.1
- which offered the choice of a fully object-oriented operating system including both object-oriented interfaces and an object-oriented implementation
- which provided real time operation.

Since the technologies were modular, the project was structured as a set of related products so that customers could buy precisely the parts that they needed. The project as a whole was known as Workplace OS (WPOS) but was often referred to as the "microkernel-based" system because of the central role played by the microkernel and because the microkernel was marketed separately as the IBM Microkernel Product.

The foundation of the project was the adoption of the Mach 3.0 microkernel from Carnegie Mellon University [11] and the TalOS or Pink operating system from Taligent. Based on the work of Julin [5], we concluded that the facilities of Mach 3.0 with some extensions were sufficient to permit the construction of a set of concurrently executing operating system personalities that were created out of a user-level personality server or servers and a set of non-personality or *personality-neutral* services. These services were either user-level servers or shared libraries. Most of the actual function was in the personality-neutral services to avoid complex cooperation algorithms between personalities, a single, always required personality or the loss of single system image. Figure 1 shows the structure of our system.

Despite the level of investment and the adoption of what was deemed to be the best of operating systems research and advanced development of the 1980s, the project, in retrospect, was not a technical success. For both business and technical reasons, it was terminated in March, 1996, after shipping two releases of the IBM Microkernel and a single limited release of OS/2 Warp for the PowerPC. The rest of this paper briefly describes the key features of the system and then provides a technical evaluation of the design approach and implementation.

0-8186-7834-8/97 $10.00 © 1997 IEEE

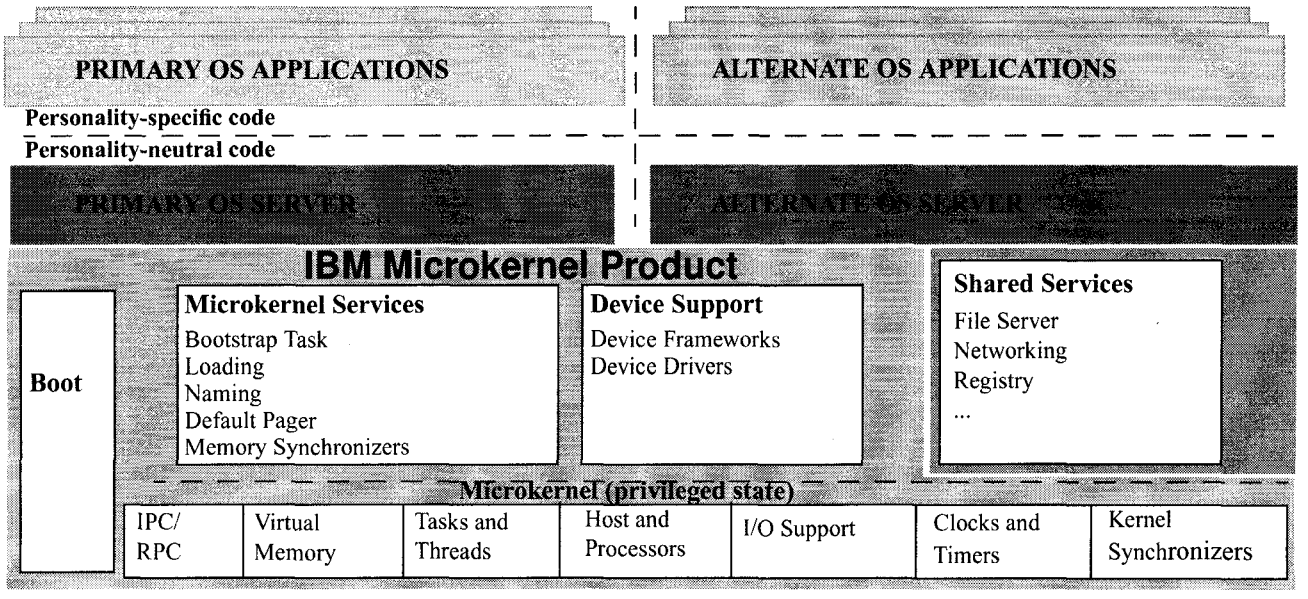

Figure 1 The IBM Microkernel and Workplace OS

The IBM Microkernel

The IBM Microkernel consisted of three major components--the microkernel proper, some user-level personality-neutral services and a set of device drivers.

Microkernel

The basic facilities of the microkernel were those of Mach 3.0 plus several additions and included

- IPC/RPC
- tasks and threads
- virtual memory management
- I/O support
- hosts and processor sets
- clocks and timers
- synchronizers.

Of these components, I/O support, clocks and timers and synchronizers were new while the others were inherited from Mach 3.0.

Of the pre-existing components IPC changed the most during the course of the project: our work was heavily influenced by [6]. Although the basic concepts of port and message were maintained, the change was so radical that we renamed IPC "RPC." Much of the work was careful software engineering, but we also

- removed reply ports
- made message delivery and in most cases reply synchronous
- blocked threads waiting to send or receive messages
- removed message queuing
- passed data too large for the message body by reference, copying it across from sender to receiver
- replaced virtual with physical copy
- optimized and simplified all of the user-level stubs and server loops
- simplified message processing inside the microkernel and the user-level stubs used to call microkernel interfaces
- removed mach_msg and the old implementation of IPC.

The result was a two to ten times improvement in message-passing performance with the improvement's magnitude depending primarily on the number of bytes transmitted. We were constrained by the semantics of the other code already under development, and this limited how closely we could approach the performance of [6]. Even with these improvements, RPC performance was deemed to be a problem to the very end of the project.

We also made a number of changes to virtual memory management including implementing the OSF RI interfaces for external management [1]. Since we ported the system to a number of different processor architectures, we wrote a number of different pmap routines [13] including ones for ARM and several forms of PowerPC. However, the most important change was the introduction of the notion of *coerced memory*--shared memory that is shared at the same range of addresses in every address space. This change is an example of the impact of supporting the semantics of an operating system other than UNIX. Since OS/2 programs assume that if memory is shared, it must be at the same address range in every address space that shares it, the microkernel had to support such a layout.

Mach 3.0 had no precursor to I/O services: its device drivers were linked into the microkernel and directly called any internal routines that they needed to use. We added

several different sets of I/O services during the course of the project to support different device driver models. All of these implementations provided:

- mapping of I/O ports and memory into the address space of the device driver
- loading of interrupt handlers
- interrupt vectoring, revectoring and possibly reflection to a user-level device driver
- DMA channel management and transfers.

The time management in Mach 3.0 was very limited, and we implemented a much more extensive time management component. Mach 3.0 also had no notion of synchronization other than that which can be constructed using the IPC system. Since this was too expensive and too hard to program for many uses, we implemented a comprehensive set of synchronizers including both memory- and kernel-based locks and semaphores.

Microkernel Services

The IBM Microkernel also included some basic user-level personality-neutral services--a personality-neutral runtime environment, a program loader, a default pager and a name service--as well as a booter. These were usually called Microkernel Services to avoid confusion with the generic notion of personality-neutral services. The runtime, the naming service and the loader merit further discussion.

The IBM Microkernel had a set of personality-neutral runtime libraries that provided an ANSI C run-time, the user-level portion of the memory-based synchronizers, a threading package based on the C threads package from Mach 3.0 and some supplemental interfaces modeled on portions of the POSIX standard. This runtime was essential to the goal of supporting personality-neutral code and operating system personalities without requiring UNIX. In contrast, CMU's Mach 3.0 had only a UNIX runtime and ran very little user space code except inside a UNIX process.

Since the IBM Microkernel like Mach 3.0 used internal capabilities, so that port rights have meaning only within the context of a port space, and since there was no way to resolve names to ports and ports to names in the microkernel itself, the system needed a name service to let clients and servers find each other. We based our interfaces on a subset of the X.500 architecture to support storing attribute information with names, complex naming formats, sophisticated search mechanisms and notifications on name space alteration. Other components including the loader, the OS/2 personality and some of the device drivers made heavy use of these features. However, this design was sufficiently expensive that Release 2 of the IBM Microkernel added an alternative, much simplified name service for embedded configurations.

The Microkernel Services loader loaded programs and shared libraries into address spaces. Originally, the loader was intended to be universal, loading all programs and libraries. Since the original design was that personality-neutral tasks and operating system personality processes did not share libraries, each address space had only a single load module format and a single set of loader semantics. We chose the ELF format and initially the SVR4 semantics for personality-neutral code. We subsequently modified this scheme to permit mixing personality-neutral and personality-specific code in an address space, support address coercion of shared libraries with a more restrictive symbol resolution semantics and limited the Microkernel Services loader to loading programs prior to the initialization of the first personality.

Device Drivers

Mach 3.0 used reworked BSD UNIX device drivers linked into the kernel. These were not acceptable in an IBM product for many reasons including code ownership, limited range of supported devices and the lack of a "device driver model" to make the support of additional devices a small rather than a large coding task. We developed a number of replacements. Our initial design was described in [3] and put almost all of the driver code except the interrupt routines in user space. It implemented the notion of a hardware resource manager to assign hardware resources representing device access paths to drivers based on a request/yield/grant scheme. Subsequently, several other device driver architectures were used with the system. There was some continuing use of drivers in the kernel with a BSD-like structure, especially for networking, but the most architecturally important work was Taligent's Object-Oriented Device Driver Management (OODDM). This device driver architecture was based on fine-grained objects with the goal of making the implementation of a new driver no more than the creation of a subclass with a few lines of unique code. The drivers were mostly in the kernel and required an internal kernel C++ runtime as well as a number of supporting classes to export kernel services.

Workplace OS

Workplace OS was the set of operating system personalities--Taligent's TalOS, OS/2, UNIX and MVM--a DOS/Windows environment--that IBM created on top of the microkernel. It also included a project to merge OS/400, the AS/400 operating system, into the microkernel environment. Since this effort used a different design approach which was uniquely tailored to the AS/400, it is not described here.

Shared Services

WPOS included several major pieces of additional personality-neutral code, known as *shared services*, beyond those in Microkernel Services. The file server was a good example of the type of personality-neutral, standard server that was envisioned as a basic functional building block for all operating system personalities in WPOS. It was a separate user-level task that provided a generic set of file system services and internally used an extended vnode architecture to support a variety of physical file system implementations including FAT, OS/2 HPFS and AIX JFS. The design of the file server made heavy use of ports to manage open files as well as aggressive memory mapping techniques to buffer file data and to share buffers with its clients. Although the file server did its own directory management, it was designed to work with the name service so that all file systems could appear as a part of WPOS's single rooted tree of names.

Another important shared service was the communications and networking code which was based on Taligent's networking frameworks. This was implemented in C++ using fine-grained objects and required a set of C++ wrappers for the interfaces exported by the microkernel. Taligent's notion of fine-grained objects involved the use of complex class hierarchies and extensive subclassing to maximize code reuse. This resulted in a very large number of very short virtual methods. The wrapper classes, rather than being a simple, stateless representation of the kernel interfaces, exported a significantly different set of interfaces that forced them to maintain state.

TalOS

Initially, the key operating system personality for Workplace OS was Taligent's operating system, TalOS, whose application interface was what became the CommonPoint programming environment [12] and included a set of file system facilities, access to communications and a graphical user interface. TalOS was based on the same principles as the networking code, using fine-grained objects, a C++ implementation, and the same C++ microkernel wrappers. The implementation of the TalOS personality was never finished.

OS/2

Second only to Taligent in importance was the OS/2 personality [10]. For OS/2 the goal was to construct its function using the building blocks of a relatively large number of personality-neutral services including the loader, the name service, the file system and the networking and communications code. The design was for an implementation of 32-bit OS/2 only and did not include the original 16-bit interfaces. The OS/2 server provided the OS/2 kernel implementation, but not the Presentation Manager since it and the desktop were user-space programs implemented as shared libraries: these were converted to 32-bit C code but otherwise left unchanged from their previous implementations.

OS/2 on the microkernel was typical of our design for personalities. Each OS/2 process received a microkernel task in which to execute, and each OS/2 thread became a microkernel thread. OS/2 programs were loaded into processes together with some additional shared libraries that provided RPC stubs for accessing function in the microkernel, in Microkernel Services, in shared services and in the OS/2 server. Wherever possible, some of the function was actually implemented in the libraries themselves to reduce the amount of interaction with the microkernel and other servers.

UNIX and MVM

Mach 3.0 had come with a UNIX implementation known as UX [2]. UX was an implementation of BSD UNIX that had been created by separating the UNIX function from the Mach code in Mach 2.5 and putting it into a single user-level server task running on the microkernel. Since our goal was a multi-server implementation and since UX was out-of-date, we elected to discard it. We planned to replace it with an AIX-compatible implementation based on the personality-neutral server structure, but this project did not progress very far before being dropped.

The last major piece of Workplace OS was the MVM server [4] whose design was an extension of that of [8]. MVM provided multiple DOS and Windows 3.1 environments, each in its own microkernel task, as well as implementing the DOS Protected Mode Interface (DPMI). MVM consisted of a small server plus a set of shared libraries that were loaded into each MVM task. The shared libraries handled the traps generated and used virtual device drivers to communicate with the real device drivers for hardware access. We used the DOS and Windows 3.1 binaries to get the required operating system function. On the PowerPC MVM also included the instruction set translator that translated blocks of Intel instructions to PowerPC instructions for execution.

Evaluation

There are a number of ways to evaluate Workplace OS and the IBM Microkernel. I will look at it from a technical perspective and focus on the semantic choices, performance implications of the design and the use of fine-grained objects.

Semantics

One major difficulty with using a set of personality-neutral servers to create a number of different operating system environments was in defining the semantics of the various servers. There was a sense in which the division into servers was reasonably straightforward, but if the interaction with the operating system personality was to be minimized, all of the stateful semantics of each supported operating system had to find their way into the personality-neutral code. Thus, for example, the file server had to implement the union of the TalOS, the OS/2 and the UNIX file system semantics: we were saved from having to include DOS/Windows since OS/2's file system semantics are a reasonable approximation of those of DOS/Windows. Unfortunately, there were places where this type of combination inevitably led to inconsistencies and implementation compromises. Even where a consistent result compatible with all of the operating system personalities was possible, the resulting implementation was big and complex.

Another problem was with data formats. To be acceptable the system had to permit the use of all of its predecessors' key on-disk data formats, especially its physical file system formats. This created a number of problems since despite the best efforts of file system architects there are places where the physical format limits the logical processing allowed or forces semantic or implementation choices in the code. A good example is the old FAT format used by OS/2: it supports only 8 character file names followed by a "." followed by 3 character extensions. There was no good way to jam long file names into the OS/2 FAT file format without generating an incompatibility.

There were also some semantic inconsistencies between the microkernel and the requirements of the operating systems. Mach 3.0 had been designed to make heavy use of copy-on-write, lazy allocation and large, sparse address spaces at some cost in both size and complexity. Its memory management was page-oriented and did not retain the allocation size. OS/2 programs assumed a commitment-oriented memory management system with eager allocation and relatively minor use of copy-on-write. Worse, OS/2's memory management was on a byte basis and assumed that the operating system retained allocation sizes. The result was essentially two memory management systems, with OS/2's built on the microkernel's, which, while workable, greatly increased the memory footprint.

Performance

A key question about microkernel-based systems has always been their performance, and that was certainly true in the case of the IBM Microkernel and WPOS. There are two sets of numbers that are particularly useful in understanding the performance characteristics of WPOS. The first is a simple comparison between OS/2 Warp on Intel and OS/2 Warp for PowerPC on a set of OS/2 benchmarks. The hardware being used here, a 133 MHz Pentium for OS/2 on Intel and a 133 MHz 604 for WPOS are roughly comparable. However, the PowerPC machine had 64 MB of memory while the Pentium machine had 16 MB.

Test	Application Content	WPOS OS/2 to OS/2 ratio
File Intensive 1	IBM Works Applications	2.96
File Intensive 2	IBM Works ToDo	2.97
Graphics Low	Klondike	0.91
Graphics Medium	Klondike	0.87
Graphics High	Klondike	0.71
PM Tasking Medium	Swp32	0.82
PM Tasking High	Wind32	1.02
Overall		1.21

Table 1: OS/2 Performance Comparisons

As these numbers show, the performance was comparable or better with the microkernel-based system for the graphics-intensive code where the test programs ran primarily at user-level in shared libraries and directly drove the screen buffer. However, interacting using RPC with the file server and the device drivers cost about a factor of 3 in performance versus a standard in-kernel implementation.

The second set of numbers was taken on a Pentium implementation of the microkernel using the performance counter hardware available there to compare RPC versus trap times. The microkernel's traps were conceptually no different from those used to implement most services in a standard operating system, so these numbers serve to illustrate the overhead introduced by using RPC for service access instead of a trap. The particular trap measured, thread_self(), returned the thread port for the current thread while the 32-byte RPC transferred 32 bytes of data from the client to the server but did nothing inside the server except process the message.

	thread_self	32-byte RPC	Ratio
Instructions	465	1317	2.83
Cycles	970	5163	5.32
Bus Cycles	218	1849	8.48
CPI	2.0	3.9	1.95

Table 2: Trap Versus RPC

Not only did the RPC require more instructions, but it had a much higher cycles per instruction metric indicating that the processor was stalling and being used less efficiently. More

detailed measurements showed that this is due to in large measure to misses on the I-cache.

Fine-Grain Objects

There has been and continues to be a significant amount of research into object-orientation in operating systems. The Workplace OS experience suggests that fine-grained objects in C++ are not appropriate for operating systems. The sheer complexity of the class structure proved to be overwhelming: operating systems are complex enough without making them even harder to write. Since C++, as we used it, effectively froze the class structure in library code with the initial version, we found the inflexibility of the implementation to be a burden. The maintenance of state in the microkernel C++ wrappers further increased the size and complexity of the system. Moreover, having a very large number of virtual method calls slowed the system down. Finally, we found that having C++ runtimes in the kernel and user space consumed considerable amounts of memory. Many of these problems were avoided in the Open Group's MK++ work [9] by carefully restricting the use of virtual methods and subclassing, combined with extensive inlining. However, MK++'s design does not provide the same level of code reuse that Taligent's did.

Conclusion

There is no good reason to believe that the function of multiple existing operating systems can be factored into a set of servers or services in such a way that they can be efficiently reconstructed by assembling parts. Moreover, operating systems whose paradigm is message passing and context switching, especially address space switching, are a poor match for the characteristics of today's processing engines which build up and maintain state internally as they execute. Liedtke [7] has been able to overcome a number of these problems at the price of making the microkernel more limited in its function than ours was and making it totally machine-dependent. However, even Liedtke's implementation of a multiserver operating system would have more overhead than a standard kernel implementation of comparable quality. Finally, the introduction of fine-grained objects and extremely complex classes has two negative effects. First, it exacerbates the performance problems. Second, it increases the complexity of the implementation. Although there will continue to be good reason to use object-orientation in operating systems, especially that of the form found in MK++, our experience suggests that simpler, coarser objects are more appropriate. In general, more modest, more targeted operating systems consume fewer resources, offer better performance and can provide the desired semantics with fewer compromises.

References

[1] Randall Dean, Michelle Dominijanni. "An RPC-Based External Memory Management Interface: Architecture." OSF Research Institute, Cambridge, MA, April, 1994.

[2] David Golub, Randall Dean, Alessandro Forin, Richard Rashid. "Unix as an Application Program." *Proceedings of the USENIX Summer Conference*, June 1990.

[3] David B. Golub, Guy G. Sotomayor, Jr., Freeman L. Rawson III. "An Architecture for Executing Device Drivers as User-Level Tasks." *Proceedings of the Third Usenix Mach Symposium*, April, 1993.

[4] David B. Golub, Ravi Manikundalam, Freeman L. Rawson III. "MVM--An Environment for Running Multiple DOS, Windows and DPMI Programs on the Microkernel." *Proceedings of the Third Usenix Mach Symposium*, April, 1993.

[5] Daniel P. Julin, Jonathan J. Chew, J. Mark Stevenson, Paulo Guedes, Paul Neves, Paul Roy. "Generalized Emulation Services for Mach 3.0: Overview, Experiences and Current Status." *Proceedings of the Usenix Mach Symposium*, November, 1991.

[6] Jochen Liedtke. "Improving IPC by Kernel Design." *1993 ACM Symposium on Operating System Principles*, 1993.

[7] Jochen Liedtke. "Toward Real Microkernels." *Communications of the ACM*, volume 39, number 9, pages 70-77, September, 1996.

[8] Gerald Malan, Richard Rashid, David Golub, Robert Baron. "DOS as a Mach 3.0 Application." *Proceedings of the Usenix Mach Symposium*, November, 1991.

[9] *MK++ Kernel High Level Design.* The Open Group Research Institute, 1996.

[10] James M. Phelan, James W. Arendt, Gary Ormsby. "An OS/2 Personality on Mach." *Proceedings of the Third Usenix Mach Symposium*, April, 1993.

[11] Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, Richard Sanzi. "Mach: A Foundation for Open Systems." *Proceedings of the Second Workshop on Workstation Operating Systems*, IEEE Computer Society, pages 109-113, September 1989.

[12] *The Power of Frameworks*, Addison-Wesley, 1995.

[13] Avadis Tevanian, *Architecture-Independent Virtual Memory Management for Parallel and Distributed Environments: the Mach Approach.* Ph.D. dissertation, Carnegie Mellon University, December, 1987.

Trademarks