Were Microkernels a good idea that did not work?
CS423 final exam – Prof. Roy H. Campbell

Fabio Kon – *f-kon@cs.uiuc.edu*

May 1, 1996

1 Introduction

The kernel is a piece of the operating system which has complete access privileges for all machine resource including memory and I/O devices.

In opposition to user-level processes which have a restricted address space and execute in the processor unprivileged mode, the kernel runs in supervisor mode and usually can access any machine register and set up processes' address spaces to protect processes from one another.

An OS server can run either as a user-level process or as a kernel process. Sometimes a user-level server can also run in supervisor mode in order to get access to an I/O device.

There are two main approaches to OS design. *Monolithic kernels* tend to run a large number of OS services as kernel processes. *Microkernels* tend to shrink the kernel running a large number of servers as user-level processes.

Systems like UNIX (which has a kernel of around 1M of code and data) are clearly monolithic while QNS, for example, is clearly a microkernel (with only 10K of code). On the other hand, there are systems like MACH 3.0 (500K of code and data) which are somewhere between the monolithic and the microkernel worlds.

2 Microkernels X Monolithic Kernels

Usually monolithic kernels are not modular which makes the job of implementing new services or modifying the existing ones much harder. Besides, the lack of modularity makes the complete understanding of the whole operation of the system something difficult to achieve.

However, that does not mean that it is impossible to build a modular monolithic OS. Choices, an Object-Oriented OS, has shown that monolithic kernels and modularity are not mutually exclusive characteristics. It has also shown that object-oriented languages like C++ can also be used in the implementation of efficient operating systems.

Anyway, an implementation error in one module of a monolithic kernel can affect other module's data structures and compromise the correctness of all system. Also, modifying one module implies rebuilding the whole kernel and requires that the system be rebooted.

The microkernel architecture, on the other hand, enforces modularity because each server has its own address space and cannot access the kernel data structures directly.

Therefore microkernels are easier to extend and modify. Only a small piece of the system must be rebuilt each time and a server can be replaced without interrupting other services and without rebooting. Some systems like CHORUS even allow the possibility of adding and removing in-kernel servers without interrupting the system functions.

The modularity also makes portability easier to implement since the hardware dependent functions can be isolated in a small number of modules.

The main problem regarding microkernels is its efficiency. Since most of the services are performed by user-level processes, the need for context switches between kernel mode and user mode is much larger than on monolithic systems. Besides, running entities like, for example, network servers in user mode requires extra data copying when sending messages from one machine to another.

In the present decade since the release of the RISC processors, the arguments against the microkernel architecture became stronger because in this kind of processor the number of registers is much larger and usually there is no way to save all of them quickly making the context switch time proportionally larger.

Microkernels usually provides at least some basic functionalities like process management, memory management and some sort of local IPC mechanism. However, even in these three topics, more complex services like schedulers, virtual memory managers and RPC mechanisms can be implemented in user-level.

The main challenge for microkernel developers is to provide the flexibility and extensibility that microkernels offer while maintaining the level of performance of monolithic systems. Some alternatives to the traditional monolithic and microkernel models have been suggested. In the following section we discuss some traditional microkernel-based systems as well as some recent systems which have proposed new kinds of architecture.

3 Implementation Critique

In this section we focus in the *kernel architecture design* of various systems. All cited systems have many other interesting features which are not addressed here for being out of the scope of this paper.

3.1 Amoeba

Designed and developed from scratch at the Vrije Universiteit in Amsterdam and with no concerns about keeping compatibility with existing systems, the Amoeba distributed system [Tan92] had as a design goal running the largest number of servers in user-level. All machines in the network run the same microkernel and their specialization is determined by the user-level servers they run.

However, Amoeba designers found running network communication servers at user-level unacceptable because of the huge degradation in performance. They decided to implement a very fast RPC protocol inside the kernel space. One interesting point is that Amoeba's designers used the argument that memory is getting cheaper in order to not implement any kind of virtual memory support.

The Amoeba microkernel also manages processes and threads, provides low-level memory management and handle low-level I/O.

The main advantage of Amoeba is its fast network support. However, the lack of compatibility to existing systems[1] makes the Amoeba user community very small. Even though, Amoeba has

[1]Just a weak form of UNIX emulation is provided in Amoeba.

been used in academia, industry, and government.

3.2 Mach

Differently from Amoeba, Mach [CDK95] was originally designed to provide Unix compatibility and complement system features with support for network services and multiprocessors.

Most of UNIX services were implemented as independent processes. However, the first major release (version 2.5) included almost all UNIX functions inside the kernel for performance reasons. So the system which was intended to be microkernel-based had a core even more complex than UNIX's.

This was fixed in version 3.0, the second major release where UNIX code was successfully removed from the kernel. Besides several kernel emulations including different UNIX flavors, OS/2, MS-DOS, and VMS were implemented or are being implemented.

A Mach user can run binaries developed for the different OS's at the same machine at the same time. This is for sure a wonderful feature but the performance of these emulations is now the major concern.

When a system call to an emulated OS occurs, Mach transparently redirects it to an emulation library which calls a user-level OS server. Exceptions are also redirected to this server.

The performance concern is particularly important for distributed applications or for local applications which access remote servers because, in opposition to Amoeba's approach, all network operations are based on user-level processes.

The interesting point is that the kernel is completely unaware of networking functions dealing only with local events. All network operations are based on a neat port-based location-independent communication model which makes much easier the implementation of distributed applications.

Implementing communication services using user-level network servers gave a great level of flexibility to Mach users and system administrators which can choose among several different network services, protocols, and interfaces to use. However, the user can be sure that none of them will be as fast as Amoeba's in-kernel communication primitives which does not need to cope with extra data copying and context switches.

Later research [GD91] has attempted to move the virtual memory manager out of the kernel using user-level processes called pagers which allow Mach to use different VM policies and implementations.

Some special care is needed when implementing VM managers and file system servers at user-level. Most of the memory pages containing the FS server must be kept in physical memory because the VM manager is a FS user. If the FS pages are swapped out, it would be impossible to swap anything in again.

Unfortunately – even the Mach developers do not saying that – analyzing the published results one can see that the first experiments on moving the VMM out of Mach's kernel have shown a considerable performance degradation which turned out to be unacceptable for many applications.

3.3 Chorus

The most interesting feature of Chorus [RAA+88] is its capability of dynamically loading and unloading servers as need arises. These servers can be loaded either into the kernel address space and run as a privileged process or can be loaded as user-level processes.

The system administrator may choose for each OS server if it will run in user-space or inside the kernel. In both cases the communication between user applications and OS servers is done in the same way using a common message-based interface.

Alternatively, a server may run in user-level while its code is being developed and later moved into the kernel when a reliable version is achieved.

This architecture seems to offer the best of the two worlds (microkernels and monolithic kernels). The system can provide high performance while keeping its flexibility and extensibility. Besides, at any particular moment, the loaded OS can be minimal for the required services.

3.4 SPIN

SPIN[BSea95] is currently being developed at the University of Washington and, similarly to Chorus, provide a way of dynamically loading pieces of software into the kernel. SPIN designers call these pieces of software *extensions* in opposition to Chorus's servers.

The most interesting point of SPIN is that it relies on MODULA-3, a type safe language, to assure that kernel extensions will not damage other kernel processes' data structures. Although the extensions run inside the kernel space, they cannot access memory or execute privileged instructions unless they have been given explicit access to them through a MODULA-3 interface. This, however, implies that any kernel extension must be written in MODULA-3.

First SPIN measurements have shown extraordinary performance improvements compared to both Mach 3.0 and DEC UNIX[2].

Some successful experiments have been made where extensions designed for particular applications provided a very good performance improvement. This was one of the main ideas that led to the development of the Exokernel model described in the following section.

3.5 Beyond Microkernels: The Exokernels

Exokernel is a new kernel architecture model proposed at MIT [EKJ95]. The parallel and distributed operating system group at MIT claim that the OS should only securely multiplex available hardware resources. All higher-level abstractions including IPC and VM should be implemented at application level by "untrusted" software.

Application programmers should link their code to libraries which would implement OS abstractions running in the application user-space. One direct advantage of this approach is that some system calls can be fulfilled without crossing the kernel space boundaries.

The main difference with the traditional microkernel approach is the fact that their interface is much closer to the hardware making possible the development of more efficient applications.

Listing a series of real applications that suffer large performance degradations due to inconvenient OS policies, MIT researchers claim that each application should choose its own OS policies. On a exokernel this would be possible by implementing optimal OS policies for each application and linking them to the application programs.

Measuring the execution time of some specific applications and some specific OS operations under their exokernel implementation, MIT researchers found that the performance could be improved by a factor of 10 to 100 comparing to Ultrix, an existing UNIX implementation.

However, not counting the case of some few applications that require very high performance, it seems unlikely that programmers will write a new OS library for each new program and even that they will choose different libraries for each program. What will probably happen is that most of application programmers will be using standard libraries like, e.g., a UNIX emulation library. The

[2]These results can be found at http://www.cs.washington.edu/research/projects/spin/www/external/bench/latest/index.html.

still unanswered question is how these applications' performance will be affected by the exokernel design if they only use generic libraries.

4 Conclusions

While one can say that the Microkernel model was a good idea that did not work because of the performance degradation, nobody can deny that in the last 10 years it has been one of the main topics for discussion in the OS research community. Systems like Mach and Amoeba brought very important contributions to the research in this field.

Nowadays it seems completely unreasonable the idea of creating a new operating system using the old monolithic architecture model. The benefits in flexibility, extensibility, and even in the ease of code writing provided by the microkernel approach are embedded in all serious operating systems currently being developed.

Alternative ways of solving most of the performance problems brought by the microkernel architecture are being found. Chorus and SPIN provide a way of dynamically loading servers into the kernel in order to provide critical services efficiently without losing the benefits of a traditional microkernel architecture. Choices has shown that it is possible to write a monolithic but still very modular OS.

Recently, a new model called Exokernel has been proposed as the next generation of OS kernel design. On a exokernel-based system the kernel is only a secure hardware multiplexer being even simpler than traditional microkernels.

The near future of operating system kernel design is not yet determined but one can be sure that they will still be highly influenced by the microkernel model. On the other hand, it is impossible to foresee how operating systems will be in 15 years from now since future hardware changes can strongly affect these systems' design.

References

[BSea95] Brian N. Bershad, Stefan Savage, and et al. Extensibility, safety and performance in the spin operating system. In *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, December 1995.

[CDK95] George Coulouris, Jean Dollimore, and Tin Kindberg. *Distributed Systems – Concepts and Design.* Addison-Wesley, 1995.

[EKJ95] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr. Exokernel: An operating system architecture for application-level resource management. In *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, December 1995.

[GD91] David B. Golub and Richard P. Draves. Moving the default memory manager out of the mach kernel. In *Proceedings of the Usenix Mach Symposium*, November 1991.

[RAA+88] Marc Rozier, Vadim Abrossimov, Francois Armand, Ivan Boule, and Michel Gien. Chorus distributed operating systems. *Computing Systems Journal*, 1:305–70, December 1988.

[Tan92] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1992.