Chapter 8
The Exokernel Operating System and Active Networks

Timothy R. Leschke
University of Maryland, Baltimore County, USA

ABSTRACT
There are two forces that are demanding a change in the traditional design of operating systems. One force requires a more flexible operating system that can accommodate the evolving requirements of new hardware and new user applications. The other force requires an operating system that is fast enough to keep pace with faster hardware and faster communication speeds. If a radical change in operating system design is not implemented soon, the traditional operating system will become the performance bottle-neck for computers in the very near future. The Exokernel Operating System, developed at the Massachusetts Institute of Technology, is an operating system that meets the needs of increased speed and increased flexibility. The Exokernel is extensible, which means that it is easily modified. The Exokernel can be easily modified to meet the requirements of the latest hardware or user applications. Ease in modification also means the Exokernel’s performance can be optimized to meet the speed requirements of faster hardware and faster communication. In this chapter, the author explores some details of the Exokernel Operating System. He also explores Active Networking, which is a technology that exploits the extensibility of the Exokernel. His investigation reveals the strengths of the Exokernel as well as some of its design concerns. He concludes his discussion by embracing the Exokernel Operating System and by encouraging more research into this approach to operating system design.

INTRODUCTION
The traditional operating system (OS) is seen as providing both management and protection of resources. As a manager, the OS controls how resources such as I/O devices, file-storage space, memory space, and CPU time get allocated. As a protector, the traditional OS controls how processes use these resources to avoid errors. Because the operating system’s role is so important, it is the one
program that is always running on a computer. The heart of the operating system is called the *kernel*.

Within a traditional computer system, the complexity of the hardware is masked behind the abstractions provided by the operating system. Although this abstraction prevents user programs from interacting with the hardware directly, user programs benefit by having a single interface that they can interact with. It is easier for a user program to interact with one operating system interface rather than developing software that must know how to interact with the different hardware components that could be present within a computer system at any one time. Because the operating system is the main interface between user programs and the raw hardware, the traditional operating system must be involved in every software-hardware interaction.

Although the traditional operating system is desirable because it provides a single interface for other software to interact with, it is ironic that this efficient interface should be the root cause of the modern computer system’s performance bottleneck. By presenting a single interface to user applications, the traditional operating system is the middle-man between all user processes and the computer hardware. By being the middle-man, the operating system tries to be all-things to all-processes. This “all-things to all-processes” approach is precisely why the design of the traditional operating system is flawed. So long as the operating system is designed to meet the minimum of a broad spectrum of operational requirements, the optimization of any one process is very unlikely. Therefore, while every application might run on a traditional operating system, few applications run well (reaches its maximum performance level) on a traditional operating system.

A team of researchers at the Massachusetts Institute of Technology (MIT) has challenged the traditional operating system design with their experimental operating system - the Exokernel Operating System. Their new approach is to separate management of resources from the protection of those resources. The Exokernel Operating System provides only protection and multiplexing of resources while allowing user processes themselves to provide the management and optimization of that resource. As Engler, Kaashoek, and O’Toole say “Applications know better than operating systems what the goal of their resource management decisions should be and therefore, they should be given as much control as possible over those decisions” (Engler, Kaashoek, & O’Toole, 1995). By separating management from protection, the abstraction provided by the traditional operating system has been eliminated. Likewise, the door to process optimization has been opened and great advances in operating system speed and flexibility have become possible.

As we investigate the Exokernel Operating System, we will discuss how it is possible to separate management from protection while still multiplexing resources within a secure environment. We will discuss select Exokernel functions such as downloading code into the kernel, reading and writing to disk memory, exception and interrupt handling, interprocess communication, tracking resource ownership, protecting and revoking resource usage, and resource management. Networking with an Exokernel will be discussed as we look at packet sending, packet receiving, the naming and routing of packets, and network error reporting. Lastly, as an example of other technologies that benefit from the Exokernel, we will briefly explore the emerging technology of Active Networking and see how the Exokernel is the ideal operating system upon which to build this new technology.

In response to the Exokernel, we will investigate why the Exokernel has not been widely accepted as the main-stream approach to Operating System design. We will investigate the potential issues with providing customer support to an extensible operating system like the Exokernel. We will argue against removing *all* management from the kernel. We will discuss
Related Content

Introduction
www.igi-global.com/chapter/introduction/98484?camid=4v1a

User Interface Design in Isolation from Underlying Code and Environment
www.igi-global.com/chapter/user-interface-design-in-isolation-from-underlying-code-and-environment/188222?camid=4v1a

Quality Metrics for Evaluating Data Provenance
www.igi-global.com/chapter/quality-metrics-evaluating-data-provenance/8245?camid=4v1a

Meta-Modeling Based Secure Software Development Processes
www.igi-global.com/article/meta-modeling-based-secure-software-development-processes/118148?camid=4v1a