Joseph Paris
CS 443

Application Performance and Flexibility on Exokernel Systems

Kaashoek et al.

This paper describes an Exokernel system that allows specialized applications to achieve high performance without sacrificing the performance of unmodified UNIX programs. The Exokernel architecture strives to safely provide untrusted software efficient control over hardware and software resources by separating management from protection. The work stems from the traditional methodology of only allowing privileged servers and the kernel to manage system resources, which forces untrusted applications to be restricted to the interfaces and implementations of the privileged software. The contention is that this organization if incorrect because application demands vary widely. This is to say that an interface designed to accommodate every application must anticipate all possible needs, which is impossible to attain. The solution that Exokernel provides is to protect the resources but delegate management of those resources to applications. This is achieved by linking an application with a library operating system (libOS) which provides an implementation of traditional operating system abstractions. Like with microkernels, the thought here is to allow specialized treatment of traditionally kernel level operations at a user level in hopes of increasing application performance (and to make the OS more modular).

One of the interesting aspects of their design is the use of UDF's for translation of metadata information in a storage system. The problem exists where we want to create multiple library file systems (libFS) that libOS's can use to interact with the storage system. For similar reasons why you would want a libOS, multiple libFS's allow us to interact with the underlying storage system in any number of application defined ways. Creating a new file should be simple and lightweight, meaning it should not require any special privileges. In addition, the protection substrate should allow multiple libFS's to safely share files at the raw disk block and metadata level. The solution is to employ UDF's . UDF's are metadata translation functions specific to each file type. This is used to analyze metadata and translate it into a simple form the kernel understands. A libFS developer can install UDFs to introduce new on-disk metadata formats. This allows an a libFS developer to write any number of UDF's that allow for correct interpretation of the metadata information stored in each file. This greatly simplifies the problems of sharing different storage structure types across varying applications.

As with other system of this type of design (microkernels, etc). Some amount of work must be put in to get the required performance increase for an application. In addition, it would be useful to find out what percentage of all applications would be able to benefit from such a structure. The nice thing about the Exokernel infrastructure is that you don't have to use the low level Exokernel implementations of the various abstractions allowing all applications to benefit from such a configuration.

A more modular operating system, as described in Exokernel, can definitely be a direction to head in. While most might not take advantages of these systems, high performance applications can thrive in such an environment.