Some Issues in the Design of an Extensible Operating System

Stefan Savage and Brian N. Bershad
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{*savage,bershad*}*@cs.washington.edu*

Extensible operating systems are designed around the principle that the service and performance requirements of all applications cannot be met in advance by any operating system. Consequently, an extensible system strives to provide a dynamic software infrastructure which allows system interfaces and implementations to be adapted or replaced to best serve application demands. However, realizing this goal poses a number of unique problems which must be addressed. Foremost among these are:

- Incrementality. Small changes to the system's behavior can be affected with small amounts of code.
- Correctness. An extension provided by an application should not compromise the overall integrity of the system.
- Efficiency. The potential for an extension should incur no overhead. The use of an extension should have an overhead which is determined by the extension's code, and not the infrastructure that has enabled the extension.

A system's *incrementality* determines the ease of introducing a system extension. Systems with high degrees of incrementality have few policy decisions embedded in the implementation and provide fine grain "hooks" to allow extensions to interpose on, or replace, existing code.

High degrees of incrementality make a system flexible but they also present greater opportunities for system corruption. Traditional systems have a relatively easy job maintaining integrity because they have few trust relationships to enforce. Such systems reply on almost complete trust between different components of the operating system kernel and frequently share data directly. Untrusted clients are isolated through a combination of memory protection and run time verification of arguments passed through a small number of external interfaces. System extensions such as dynamically loadable device drivers or file systems are usually trusted in their entirely.

Extensible systems, by contrast, export potentially large numbers of interfaces to untrusted clients. These untrusted extensions must be isolated from misusing these interfaces in any way that might compromise the integrity of the overall system. Moreover, it is not possible to provide this isolation mechanically through memory protection. While reference isolation mechanisms can ensure that storage is not corrupted, they are not sufficient for enforcing more complicated restrictions, such as invariants about liveness or synchronization. For example, an extension may enter an infinite loop, or fail to follow a locking protocol on a shared resource, or invoke an operation with invalid parameters.

These invariants must be guaranteed using a combination of *a priori* verification, run-time protection mechanisms, and safe interface design techniques. We believe that much of this work is well suited to current compiler technology which allows for both the automation of such isolation mechanisms as well as opportunities to optimize their implementation based on high level information.

We are designing an extensible system called SPIN, which attempts to solve these issues using a combination of language and compiler support for the safe manipulation of objects and the design of careful resilient interfaces to share system resources.