SLK: The Safe Language Kernel Project

J-Kernel

Papers

People

Jigsaw/SLK (old)

The Safe Language Kernel project is developing an operating system infrastructure for extensible systems, such as customizable internet servers and application specific gateways. The primary goal of SLK is to allow users to upload custom extensions onto servers in the network in a secure yet flexible manner. Just as Java currently enables Web browsers in which users to "safely" download applets, SLK will enable "safe" Internet servers into which users can upload servlets. There are a number of reasons why this is interesting:

- internet service providers can allow users to serve more than static documents,
- it creates a software market for server plug-ins which can be installed without requiring the system administrator to fully trust the new product,
- distributed services can spread/migrate in the network in response to usage or network state, and
- mobile users can load proxy services into "nearby" servers, for example, to transform data in the network as appropriate for their displays and network connection speeds.

Current http servers actually already include some of this functionality but without any safety net.

In SLK the emphasis is on sharing among applications and the sharing between applications and the kernel is seen as a simpler special case. In traditional operating systems with a separate address space for each application the only methods of sharing are shared VM pages for data and RPC calls for code. In contrast, in SLK an application can directly export a data structure as well as code operating on the data. The protection features of the language system enforce that another application importing the data structure can only access the public fields of the data and can only invoke the public methods. Given a handle onto a single object, the importer can follow all public links to other objects and thus potentially explore and modify the data structure. The use of an imported data structure does not incur any overhead over accessing a regular private data structure: public fields are accessed at constant offsets from the object pointer and methods are invoked using a standard OO method dispatch.

SLK relies on the properties of type-safe languages in order to enforce protection boundaries between applications and the OS itself which means that all code can run in a single address space and at a single hardware privilege level. The first version of SLK is heavily Java based but a significant part of our research effort lies in understanding how to host multiple languages. For example, we plan to integrate ML into the

family of languages supported by SLK.

The most fundamental difference between the Secure Language Kernel (SLK) and a traditional operating system is the fact that the entire system runs in a single address space and at a single hardware protection level. There is no memory management hardware that prevents one application from accessing another's memory and there is no hardware privilege mode differentiating instructions executed in the kernel from those executed in an application. Instead, all protection is enforced by the language system. Languages used under SLK must be type safe and the compiler must provide enough information to the run-time system to allow protection boundaries to be enforced.

The motivation for relying on software for protection is three-fold: light weight, seamless extensibility, and flexibility in the form of fine grain sharing. Under the assumption that the language system can enforce protection, it is natural to propagate this new property through the system and eliminate redundant functionality in an attempt to reduce complexity and improve efficiency. This is also the primary technical motivation behind Sun's upcoming JavaOS [Mad], but unlike JavaOS, SLK focuses on servers and fine-grain sharing of data and code across protection boundaries. In this sense it continues a decade-long trend in OS design in moving functionality into user-level and generally blurring the user-kernel boundary. SLK removes the user-kernel boundary entirely.

The SLK project includes developments at the level of the operating system, programming languags, and the run-time system. A primary focus of the research is in the infrastructure required for large-scale collaborative information systems in which users connect with different appliances to the network and in which devices from telephones to cameras in meeting rooms are integrated into the system.

The expected benefits of our approach are higher resource efficiency, seamless system extensibility, and flexibility in the form of fine grain sharing across protection domains. We plan our research to gravitate around a small number of complex issues, in particular, the security issues involved in executing foreign code in systems, the operating system structures to support the secure execution of type-safe languages without the use of hardware protection facilities, and the efficient integration of communication and computation.

SLK fits into the current trend in operating systems research to allow applications to dynamically extend the kernel to improve application performance or extend functionality. Virtually all operating system modules in SLK are implemented as regular Java packages and are only special or privileged in that they are linked to have access to other internal data structures in the kernel or have direct access to devices.

Thorsten von Eicken

DEPARTMENT OF COMPUTER SCIENCE,

CORNELL UNIVERSITY

Last updated 02/12/98

Given the appropriate linking mechanisms and permissions, it is a simple matter to add a new module in order to extend the functionality of what other applications perceive as the "operating system".