Position Summary: Secure OS Extensibility Needn't Cost an Arm and a Leg

Antony Edwards and Gernot Heiser

University of NSW, Sydney 2052, Australia
{antonye,gernot}@cse.unsw.edu.au

Abstract

This position paper makes the claim that secure extensibility of operating systems is not only desirable but also achievable. We claim that OS extensibility should be done at user-level to avoid the security problems inherent in other approaches. We furthermore claim (backed up by some initial results) that user-level extensibility is possible at a performance that is similar to in-kernel extensions. Finally, user-level extensions allow the use of modern software engineering techniques.

Extensibility is a way to build operating systems that are highly adaptable to specific application domains. This allows, for example, the use of subsystems that are highly tuned to a particular usage patterns, and thus should be able to outperform more generic systems.

In the past, user-level extensibility in systems like Mach and Chorus has lead to poor performance. This has triggered approaches like loadable kernel modules in Linux, which require complete trust in extensions, or secure extensible systems like Spin or Vino, which use trusted compilers or in-kernel protection domains to achieve security. We believe that secure extensibility is possible, with good performance, at user level.

We think that extensibility will only work if they are secure, minimal restrictions are imposed, performance is not degraded, and modern software engineering techniques are supported.

We have developed an extension system based on *components* [2] for our Mungi single-address-space operating system. The component model provides interfaces based on CORBA, and supports modularisation and reuse to make is suitable for building large systems. It supports dynamic binding of extensions, and independent customisation (different users can invoke different, even mutually incompatible extensions).

The single address space helps to achieve performance goals, as it minimises the payload sizes and the amount of marshaling required for component invocations (data is usually passed by reference). In combination with an appropriate protection model, it also makes it easy to expose system resources, to make them accessible to extensions.

The security of the extension model is ensured by a protection system that combines discretionary access control (via password capabilities), with mandatory access control. The former supports *least privilege* while the latter is used to enforce system-wide security policies. These security policies are defined by user-level security objects that are themselves extensions. Both aspects of the protection model are used to restrict the data the extensions can access, as well as who can access the extensions. Mandatory security supports the confinement of extensions, to prevent them from leaking data, even between different clients invoking the same extension.

Mungi	Spin	Vino	COM	omniORB	ORBacus
100	101	885	1993	768	9319

The table compares invocation costs (microseconds) various extensible architectures. These are to be taken with a grain of salt, as they have been measured on different hardware and normalised according to SPECint-95 ratings. However, these results clearly show that Mungi's performance is superior to existing component architectures, and at least equivalent to existing extensible operating systems. This is being achieved while providing full protection, and without relying on type-safe languages.

For more information see [1].

References

[1] A. Edwards and G. Heiser. A component architecture for system extensibility. Technical Report UNSW-CSE-TR-0103, School Comp. Sci. & Engin., University NSW, Sydney 2052, Australia, Mar 2001. URL ftp://ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/0103.pdf.

[2] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley/ACM Press, Essex, England, 1997.