Flexible Event Handling for an Extensible Operating System

Przemysław Pardyak
Department of Computer Science and Engineering
University of Washington, Seattle, U.S.A.

November 8th, 1994

SPIN is an extensible operating system being built at the University of Washington. The *SPIN* microkernel allows applications to define customized operating system services. Three key mechanisms make this possible: a dynamic linker, which installs user code into the kernel at runtime, a safe programming language, which ensures that the user's extensions do not violate system integrity, and an event-based invocation system allows extension code to request, and execute in response to changes in system state.

The design of *SPIN* includes internal interfaces to the system services which expose internal working of the system. The interfaces together with the event machinery enable extensibility in *SPIN* by providing respectively the hooks where extensions may be installed and a means for communicating with them. We believe that the combination of the right interfaces that give access to the internals of the system and allow changing its inner polices together with a flexible means for integrating the extensions are critical for any flexible system.

Events are a means of indirect invocation where communicating parties do not know about each other. An event handler is activated when an event for which it is registered is raised. Handling may be conditional on a set of constraints over the arguments of the event and the system state. A single event may be handled by any number of handlers. An event that announces that a VM page fault occurred allows, e.g., to either override the default pager with card marking scheme or to add a routine that counts faults.

An event dispatcher is the module responsible for event delivery: it is invoked to raise an event and eventually it delivers the event to the right handlers. An event dispatcher for a flexible and extensible system should also be flexible so that the semantics of event handling can be extended. Therefore the design of the event dispatcher for *SPIN* includes a meta-interface that contains a set of events corresponding to the stages of event handling. The events can be used for providing extensions or alternatives to the default operations of the dispatcher. As an example consider the problem of order in which events should be delivered. The default policy to use the order of their raising may be extended with a priority scheme that will order them as a reaction to an event raised by the dispatcher to order the events.

The characteristics of the event dispatcher determine usability and performance of the system. The cost of the dispatcher operations limits granularity of extensions, its interface influences the way extensions are built, and the guarantees about event semantics determine the way events are used for interactions among extensions. The extra level of indirection introduced by events may not decrease performance of the critical parts of the system and the parts of the system that are not using flexibility should not pay for it. We assure it by optimizing for common cases and by using compiler techniques to reduce the indirection overhead. The interface to the dispatcher contains few low level operations that implement simple and efficient event handling. If the default abstractions and semantics are not adequate they may be modified by means of the dispatcher's meta-interface.