

Research Directions

Julia Lawall

1 Research Overview

The role of an operating system (OS) is to provide applications with safe and convenient access to low-level machine resources. Standard OSes, such as Linux and Windows, try to provide acceptable service to all applications with as little input from these applications as possible. Nevertheless, the emergence of highly complex, resource-demanding applications, such as multimedia applications, coupled with the increasing use of restricted environments, such as very small embedded systems, makes this general-purpose model less and less viable. What is needed is a means to allow application developers to adapt an OS to the application's particular needs. My research uses programming-language techniques in the design of frameworks for the development and deployment of adaptations of specific OS services. Such a framework is centered around a *domain-specific language* (DSL) for programming adaptations. The main challenges are to make OS adaptation easy for non-OS experts, to make the resulting adaptations safe and efficient, and to make the adaptation process portable across multiple architectures and OSes.

Creating OS adaptations requires a deep understanding of the existing OS structure, to know both what functionality an adaptation should provide and how the existing code should be modified to integrate the adaptation with the OS. The structure of typical OSes makes this task difficult for those who are not OS experts. The need to manage many different resources, and to do so as efficiently as possible, has implied that OS code typically is voluminous, tangled, and relies on many implicit invariants. The challenge is to design a framework that both guides the programmer as to what should be defined as part of an adaptation of a particular service and manages the integration of such an adaptation into the target OS. In this context, a DSL should include declarations, operator, and other constructions that guide the programmer in the structure of adaptations.

To ensure the continued good functioning of the target OS, an OS extension must not only satisfy standard safety properties, such as dereferencing only non-null pointers, but also respect OS invariants. The latter can be complex, for example, requiring specific instruction sequences, and may not be adequately documented. The challenge is to describe OS invariants formally, both to provide the adaptation programmer with appropriate documentation and so that these properties can be checked formally in the adaptation code. To address the need for safety, strategies are needed for designing DSLs that not only facilitate programming, but also make properties relevant to a particular kind of adaptation easily verifiable. A language design is even more important when considering adaptation of multiple OS services. In this case, both properties of the individual services and properties describing their interaction must be taken into account.

Efficiency is critical for OS code, as execution in the OS represents a pure overhead from the point of view of applications. Efficient coding strategies are often low level and error prone, and may require architecture-specific knowledge that is outside the scope of most application programmers. The challenge is to encapsulate this knowledge in framework support tools such as the DSL compiler. The compiler should also use information collected during the verification process to guide optimizations.

The space of appropriate adaptations is determined more and more by the affected service than by the target OS. The challenge is thus to provide a framework that can be used with multiple OSes. Some OS dependencies can be encapsulated in the DSL, with the OS specific implementation generated automatically by the compiler. When needed, OS dependencies should be exposed to the programmer as part of the formal description of OS requirements, separate from the framework itself, so that the same framework can be used across multiple OSes.

2 Current status and future plans

I am carrying out this work in collaboration with Prof. Gilles Muller of the École des Mines de Nantes. Our current work is on the Bossa framework for implementing process scheduling policies in standard OSes. This framework meets the challenges described above. The Bossa DSL simplifies the problem of expressing scheduling policies to the point that students with no previous kernel programming experience have been able to implement scheduling policies for Linux. The compiler of this DSL performs verifications based on a description of the requirements of the target OS. These verifications implied that none of the students crashed the machine in testing their schedulers. In our own experiments, we have found that the use of Bossa incurs no overhead in the execution of a variety of standard applications. Bossa has been implemented for several versions of Linux and we are currently porting it to Windows, to further test the portability of the model.

The Bossa experiment has shown the viability of the language-based approach to OS extension in the context of a single OS resource. The next step is to apply the ideas developed in the work on Bossa to the management of other resources. Our concrete goal is to develop a framework for controlling energy usage that encompasses management strategies for multiple energy-consuming devices. In this context, we will investigate how to specify and reason about management strategies for the individual resources and how to reason about the ways in which the management strategies for these resources interact.