Java Operating Systems: Design and Implementation

Godmar Back  Patrick Tullmann  Leigh Stoller  Wilson C. Hsieh  Jay Lepreau
Department of Computer Science
University of Utah

Technical Report UUCS-98-015
August, 1998

Abstract

Language-based extensible systems such as Java use type safety to provide memory safety in a single address space. Memory safety alone, however, is not sufficient to protect different applications from each other. Such systems must support a process model that enables the control and management of computational resources. In particular, language-based extensible systems must support resource control mechanisms analogous to those in standard operating systems. They must support the separation of processes and limit their use of resources, but still support safe and efficient interprocess communication.

We demonstrate how this challenge can be addressed in Java operating systems. First, we describe the technical issues that arise when implementing a process model in Java. In particular, we lay out the design choices for managing resources. Second, we describe the solutions that we are exploring in two complementary projects, Alta and GVM. GVM is similar to a traditional monolithic kernel, whereas Alta closely models the Fluke operating system. Features of our prototypes include flexible control of processor time using CPU inheritance scheduling, per-process memory controls, fair allocation of network bandwidth, and execution directly on hardware using the OSKit. Finally, we compare our prototypes with other language-based operating systems and explore the tradeoffs between the various designs.

1 Introduction

Language-based extensible systems in the form of Java virtual machines are used to implement execution environments for applets in browsers, servlets in servers, and mobile agents. All of these environments share the property that they run multiple applications at the same time. For example, a user may load applets from different Web sites into a browser; a server may run servlets from different sources; and an agent server may run agents from across the Internet. Given the necessity of supporting multiple applications, a language-based extensible system must be able to isolate applications from one another because they may be buggy or even malicious.

Conventional operating systems provide the abstraction of a process, which encapsulates the execution of a program. A process model defines what a process is and what it may do. The following features are necessary in any process model for safe, extensible systems:

- **Protection.** A process must not be able to manipulate or destroy another process’s data in an uncontrolled manner. For example, an unprivileged process must not be able to deliberately (or accidentally) interfere with another process’s forward progress.

- **Resource Management.** Resources allocated to a process must be separable from those allocated to other processes. An unprivileged or untrusted process must not be able to starve other processes by denying them resources.

- **Communication.** Since applications may consist of multiple cooperating processes, processes must be able to communicate with each other. The communication channels must be safe and should be efficient.

These requirements on processes form one of the primary tradeoffs in building operating systems, as illustrated in Figure 1. On the right-hand side, processes can be protected from each other most easily if they are on completely separate machines. In addition, managing computational resources is much simpler, since the resources are completely separate. Unfortunately, communication is more expensive between processes on different machines. On the left-hand side, communication is much cheaper, since processes can share memory directly. As a result, though, protection and accurate resource accounting become more difficult.

Operating systems research has spanned the entire range of these systems, with a primary focus on systems in the middle. Research in distributed systems and networking