Fx86: Functional Management of Imperative Virtual Machines

Bryan Ford
M.I.T.

August 28, 2003

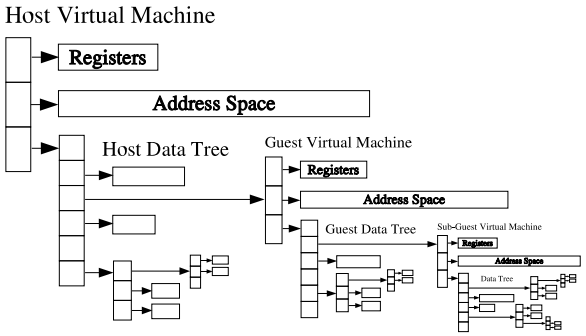
1 Introduction

Fx86 is a virtual machine architecture based on the ubiquitous x86 instruction set, extended to allow user-mode applications to create efficient, fine-grained recursive virtual machines in which to run plug-ins or other extensions in confined environments. Guest virtual machines appear to the host application as purely functional computations on passive, immutable data. This novel blend of the functional and imperative paradigms ensures that guest computations are strictly deterministic, enabling the host application to exert precise control over its guests while permitting native instruction execution at all levels. Hosts can efficiently snapshot guests at arbitrary points, roll computations forward or backward for debugging, replay computations and check the results for monitoring or verification purposes, and distribute, replicate, or migrate computations across heterogeneous processors. The architecture is fully recursive [2], enabling guests to act as hosts to further (sub-)guests without substantial performance penalty.

2 State Management

The Fx86 architecture augments a process's normal register state and virtual address space with a private *data tree*, which acts as a simple process-private file system holding bulk data exclusively owned and controlled by the process. By architecturally separating process-private bulk storage from conventional shared OS-level storage services, applications and guest computations minimize false interactions with their environment, enabling virtual machine state to be isolated and controlled more effectively.

The entire state comprising a guest computation, including the guest's data tree, forms a subtree of its host's data tree. To run a guest computation the host simply executes a special Fx86 instruction, which acts as a functional transformer taking a data subtree representing the guest's starting state and yielding a new subtree representing its deterministic result, after executing a predetermined number of guest instructions. Fx86 implementations guarantee that copied but unchanged portions of data trees are only stored once in memory or on disk (copy-on-write optimization), allowing hosts to store snapshots of guest computations efficiently at any desired frequency.



3 Implementation

Fx86 environments can be implemented to close approximation as an extension to existing operating systems or virtual machine monitors; a Linux-based implementation is in progress. Implementation of a strict, completely deterministic Fx86 environment currently requires instruction emulation or rewriting, but two minor enhancements to the x86 architecture would enable strict native implementations on future processors: the ability to disable a few well-known non-virtualizable x86 instructions, and a "recovery counter" that can trigger a trap after executing a specific number of instructions [1].

References

[1] *PA-RISC 1.1 Architecture and Instruction Set Reference Manual*. Hewlett-Packard, third edition, February 1994.

[2] Bryan Ford, Mike Hibler, Jay Lepreau, Patrick Tullmann, Godmar Back, and Stephen Clawson. Microkernels meet recursive virtual machines. In *Operating Systems Design and Implementation*, pages 137–151, 1996.