Interface and Execution Models in the Fluke Kernel

Bryan Ford Mike Hibler Jay Lepreau Roland McGrath Patrick Tullmann

Department of Computer Science
University of Utah

Technical Report UUCS-98-013
August, 1998

Abstract

We have defined and implemented a new kernel API that makes every exported operation either fully interruptible and restartable, thereby appearing atomic to the user. To achieve interruptibility, all possible states in which a thread may become blocked for a "long" time are completely representable as valid kernel API calls, without needing to retain any kernel internal state.

This API provides important functionality. Since all kernel operations appear atomic, services such as transparent checkpointing and process migration that need access to the complete and consistent state of a process can be implemented by ordinary user-mode processes. Atomic operations also enable applications to provide reliability in a more straightforward manner.

This API also allows novel kernel implementation techniques and evaluation of existing techniques, which we explore in this paper. Our new kernel's single source implements either the "process" or the "interrupt" execution model on both uni- and multiprocessors, depending only on a configuration option affecting a small amount of code. Our kernel structure avoids the major complexities of traditional implementations of the interrupt model, neither requiring ad hoc saving of state, nor limiting the operations (such as demand-paged memory) that can be handled by the kernel. Finally, our interrupt model configuration can support the process model for selected components, with the attendant flexibility benefits.

We report preliminary measurements comparing fully, partially and non-preemptible configurations of both process and interrupt model implementations. We find that the interrupt model has a modest speed edge in some benchmarks, maximum latency varies nearly three orders of magnitude, average latency varies by a factor of six, and memory use favors the interrupt model as expected, but not by a large amount. We find that the overhead for restarting the most costly kernel operation ranges from 2–8%.

This research was supported in part by the Defense Advanced Research Projects Agency, monitored by the Department of the Army under contract number DABT63–94–C–0058, and the Air Force Research Laboratory, Rome Research Site, USAF, under agreement number F30602–96–2–0269.

Contact information: lepreau@cs.utah.edu. Dept. of Computer Science, 50 S. Central Campus Drive, Rm. 3190, University of Utah, SLC, UT 84112-9205. http://www.cs.utah.edu/projects/flux/.

1 Introduction

This paper attempts to bring to light an important and useful control-flow property of OS kernel interface semantics that has been neglected in prevailing systems, and to distinguish this *interface* property from the control-flow properties of an OS kernel *implementation*. An essential issue of operating system design and implementation is when and how one thread can block and relinquish control to another, and how the state of a thread suspended by blocking or preemption is represented in the system. This crucially affects both the kernel interface that represents these states to user code, and the fundamental internal organization of the kernel implementation. A central aspect of this internal structure is the execution model in which the kernel handles processor traps, hardware interrupts, and system calls. In the *process model*, which is used by traditional monolithic kernels such as BSD, Linux, and Windows NT, each thread of control in the system has its own kernel stack. In the *interrupt model*, used by systems such as V [7], QNX [14], and Aegis [12], the kernel uses only one kernel stack per *processor*—for typical uniprocessor kernels, just one kernel stack, period. A thread in a process-model kernel retains its kernel stack state when it sleeps, whereas in an interrupt-model kernel threads must manually save any important kernel state before sleeping. This saved kernel state is often known as a *continuation* [10], since it allows the thread to "continue" where it left off.

In this paper we draw attention to the distinction between an interrupt-model *kernel implementation*, which is a kernel that uses only one kernel stack per processor by manually saving implicit kernel state for sleeping threads, and an "atomic" *kernel API*, which is an API designed so that sleeping threads *need* no such implicit kernel state at all. These two kernel properties are related but fall on or-