Flexible Multi-Policy Scheduling based on CPU Inheritance

Bryan A. Ford  Sai R. Susarla
Department of Computer Science
University of Utah
Salt Lake City, UT 84112
flux@cs.utah.edu
http://www.cs.utah.edu/projects/flux/

May 10, 1996

Abstract

Traditional processor scheduling mechanisms in operating systems are fairly rigid, often supporting only one fixed scheduling policy, or, at most, a few “scheduling classes” whose implementations are closely tied together in the OS kernel. This paper presents CPU inheritance scheduling, a novel processor scheduling framework in which arbitrary threads can act as schedulers for other threads. Widely different scheduling policies can be implemented under the framework, and many different policies can coexist in a single system, providing much greater scheduling flexibility. Modular, hierarchical control can be provided over the processor utilization of arbitrary administrative domains, such as processes, jobs, users, and groups, and the CPU resources consumed can be accounted for and attributed accurately. Applications as well as the OS can implement customized local scheduling policies; the framework ensures that all the different policies work together logically and predictably. As a side effect, the framework also cleanly addresses priority inversion by providing a generalized form of priority inheritance that automatically works within and among multiple diverse scheduling policies. CPU inheritance scheduling extends naturally to multiprocessors, and supports processor management techniques such as processor affinity [7] and scheduler activations [1]. Experimental results and simulations indicate that this framework can be provided with negligible overhead in typical situations, and fairly small (5-10%) performance degradation even in scheduling-intensive situations.

1 Introduction

Traditional operating systems control the sharing of the machine’s CPU resources among threads using a fixed scheduling scheme, typically based on priorities. Sometimes a few variants on the basic policy are provided, such as support for fixed-priority (non-degrading) threads [? ,? ], or several “scheduling classes” to which threads with different purposes can be assigned (e.g. real-time, interactive, background). [? ]. However, even these variants are generally hard-coded into the system implementation and cannot easily be adapted to the specific needs of individual applications.

In this paper we develop a novel processor scheduling framework based on a generalized notion of priority inheritance. In this framework, known as CPU inheritance scheduling, arbitrary threads can act as schedulers for other threads by temporarily donating their CPU time to selected other threads while waiting on events of interest such as clock/timer interrupts. The receiving threads can further donate their CPU time to other threads, and so on, forming a logical hierarchy of schedulers, as illustrated in Figure 1. Scheduler threads can be notified when the thread to which they donated their CPU time no longer needs it (e.g., because the target thread has blocked), so that they can reassign their CPU to other target threads. The basic thread dispatching mechanism necessary to implement this framework does not have any notion of thread priority, CPU usage, or clocks and timers; all of these functions, when needed, are implemented by threads acting as schedulers.

Under this framework, arbitrary scheduling policies can be implemented by ordinary threads cooperating with each

1We use the terms CPU and processor synonymously.