SYNCHRONIZATION WITH EVENTCOUNTS AND SEQUENCERS (Extended Abstract)

David P. Reed
Rajendra K. Kanodia*

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

The design of computer systems to be concurrently used by multiple, independent users requires a mechanism that allows programs to synchronize their use of shared resources. Many such mechanisms have been developed and used in practical applications. Most of the currently favored mechanisms, such as semaphores and monitors are based on the concept of mutual exclusion.

In this paper, we describe an alternative synchronization mechanism that is not based on the concept of mutual exclusion, but rather on observing the sequencing of significant events in the course of an asynchronous computation. Two kinds of objects are described, an eventcount, which is a communication path for signalling and observing the progress of concurrent computations, and a sequencer, which assigns an order to events occurring in the system.

Eventcounts and sequencers are a more natural mechanism for controlling the sequence of execution of processes that do not need mutual exclusion. Examples of these applications are monitoring state changes of operating system variables, and broadcasting the occurrence of an event to any number of interested processes.

In applications where mutual exclusion mechanisms are explicitly prohibited, such as physically distributed systems and systems that need to solve the confinement problem, eventcounts and sequencers can be used to solve synchronization problems in a very natural way.

An eventcount is an object that keeps a count of the number of events in a particular class that have occurred so far in the execution of the system. An eventcount can be thought of as a non-decreasing integer variable. We define an advance primitive to signal the occurrence of an event associated with a particular eventcount and two primitives, await and read, that obtain the "value" of an eventcount.

Some synchronization problems require arbitration: a decision based on which of several events happens first. Eventcounts alone do not have this ability to discriminate between two events that happen. Consequently, we provide another kind of non-decreasing integer object, called a sequencer, that can be used to dynamically order the events in a given class.

Eventcounts and sequencers are a primitive synchronization mechanism of approximately the same level of abstraction as semaphores. We show in the paper how P and V semaphore operations can be expressed in terms of eventcount operations, and how a simultaneous-P operation on several semaphores is just as easily expressed. On the other hand, eventcounts are more easily used in expressing synchronization requirements directly in the program because they directly encode the history of events in the system.

We show in the paper how the eventcount and sequencer primitives can be axiomatized in terms of a partial ordering of the events that occur within the system. From the axioms defining eventcounts and sequencers, one can directly obtain the ordering constraints enforced in a particular set of programs.

Synchronization mechanisms provided by operating systems present a thorny obstacle to solving the "confinement problem" defined by Lampson. Eventcounts and sequencers provide some help in controlling the flow of information, by providing a distinction between those synchronization primitives that can be used to send information (advance and ticket) and those that can only receive (read and await). In the paper we show how to solve a synchronization problem called the "secure readers-writers problem" while preventing unnecessary leaks of information.

This research was performed in the Computer Systems Research Division of the M.I.T. Laboratory for Computer Science. It was sponsored in part by Honeywell Information Systems Inc., and in part by the Air Force Information Systems Technology Applications Office (ISTAO), and by the Advanced Research Projects Agency (ARPA) of the Department of Defense under ARPA order no. 2641, which was monitored by ISTAO under contract No. F19628-74-C-0193.

* Present affiliation of R. K. Kanodia: Bell Laboratories, Holmdel, N.J.