REED79

D.P. Reed and R.K. Kanodia, "Synchronization with Eventcounts and Sequencers", Communications of the ACM, February 1979, Volume 22, No. 2, pp. 115-123. The subject of this paper is a synchronizing mechanism for concurrent processes that is not based on mutual exclusion. Synchronization is achieved by observing and signalling the occurence of events with objects called eventcounts. An event count may be thought of as a nondecreasing integer variable. Three primitives are associated with a particular eventcount:

* advance(E) is used to signal the occurrence of an event with the event count E. Value of E is increased by 1.
* await(E,v) suspends the calling process until the value of the eventcount E is at least v.
* read(e) returns the current value of the eventcount E.

This paper tries to prove formally that eventcounts provide synchronization by defining timing relationships. In order to do so, particularly for distributed systems, communications time delays must be accounted for. East system observes events at different times and in particular the executions of advance, await and read primitives constitute events. Useful timing relationships are derived, like for instance:

if an await operation of the form await(E,t) is expected,
then there are at least t executions of advance(E) and
advance(E) precedes or is concurrent with await(E,t).

By making use of a set of timing relationships and an appropriate formalism, the authors are able to prove that and algorithm for the solution of the traditional producer/consumer problem, by using eventcounts and not mutual exclusion, is a valid synchronizing scheme. It is shown that the synchronization of the producer and the consumer problem is obtained from the ability of eventcounts to maintain relative ordering of events. There is an advantage in doing so, because the concurrency between producer and consumer seems to be higher than by introducing mutual exclusion. Sometimes arbitration is required in order to decide which of several events has to happen first. Eventcounts do not have this ability to discriminate among unordered events. A "sequencer" is then introduced to order events. A sequencer can be thought of as a nondecreasing integer variable initially equal to zero. A sequencer allows only an operation called ticket(S):

* ticket(S) returns a non-negative integer value as its result when applied to a sequencer. The ordering of the values returned corresponds to the time ordering of the execution of the ticket operations.

An instance of sequencer use is the case of multiple producers in the producer/consumer problem. An interesting comment about eventcounts is that they may be thought of as information channels among processes. For secure systems that try to solve confinement problems, the operations on eventcounts allow separation of signalling-only operations from observing-only operations. Advance can be thought of as a signalling-only operation, while read and await are observing-only operations. Semaphores instead have mixed functions (signal+observe) in their P operation. Unlike eventcounts, sequencers do not have pure observation and pure signalling operations. The basic result presented in this paper is a synchronizing mechanism for concurrent processes, not based on mutual exclusion. It seems to avoid unnecessary serializations of processes and from this point of view it proves to be especially important in multiprocessor or distributed systems.