HOARE,C.A.R.74
"Monitors, An Operating System Structuring Concept", C.A.R. Hoare,
Communications of the ACM, October 1974, Volume 17, Number 10,
pp. 549-557.
Monitors have been conceived by Brinch Hansen and are further developed
in this paper by C.A.R. Hoare as modules for managing mutually exclusive access
to shared resources. A monitor consists of a collection of local variables,
used to store the resource's state, and some procedures, which implement
operations on the resource. Whenever one of these procedures is invoked,
exclusive access to the local data is provided. Moreover, monitor's
procedures should not access nonlocal variables in order to prevent
time-dependent coding errors.
Two synchronizing operations are introduced: a "wait" operation and a
"signal" operation. These two operations are defined on condition
variables. Condition variables are represented as queues of processes currently awaiting on the condition. A "wait" operation caus
to be suspended and queued on the "wait-"ed condition, whereas a "signal"
operation causes temporarily suspension of the invoker and reactivation
of the process blocked on the condition. The signalling process is
reactivated where there is no other process executing in the monitor.
Moreover, signalers are given priority over processes trying to start
execution of a monitor procedure.
In this paper a simple example is shown of a scheduling algorithm for
managing single resources using monitors. The algorithm is based on
two procedures: acquire and release. The two entries of the monitor
behave like P and V operations on binary semaphores. Monitors too can be
implemented by binary semaphores, thus showing perfect equivalence of the
two concepts.
A monitor invariant can be associated to the local (critical) data of the
monitor. The invariant must be true whenever no process is executing the
monitor. Thus a process must reassert the invariant before exiting the
monitor or before performing a "wait" or "signal" operations.
Sometimes in the design of operating systems a simple scheduling algorithm
based on first-come-first-served may not be adequate. Thus, priority
conditions are introduced and they are just like ordinary conditions, except that a wait on priority conditions must specify a non ne
A signal on a priority condition wakes up the waiting process with the lowest
parameter value.
Typical problems in concurrent software are then presented in this paper.
They are solved and coded by using monitors. They include: the "Bounded
Buffer" problem (based on a producer/ consumer type of relation), the design
of a "Buffer Allocator" (similar in design to the solution for
the Bounded Buffer problem, but with the added ability of transferring long
messages), the design of a "Disk Scheduler" (based on the idea of trying
to minimize the frequency of change of direction of movement of the heads)
and finally the handling of concurrent reading and writing using a "fair"
scheduling algorithm.
This paper represents an important milestone in the development of
concurrent programming. It is in this article that the concept of monitors
has been fully developed and the ideas expressed in this paper have been
applied most notably in the extended version of Pascal, called Concurrent
Pascal, in the CE language and in SIMULA.
Monitors allow the user to design a separate isolated monitor for each kind of resource and, as a consequence, to separate schedulin
appropriate tool for the structuring of an operating system.