# COMMENTS ON MONITOR DEFINITION AND IMPLEMENTATION *

Gregor V. BOCHMANN
*Département d'Informatique, Université de Montréal, Montréal, Canada*

**Received** 13 May 1976, revised version received 29 July 1976

**Monitors, scheduling, synchronization, mutual exclusion, monitor implementation**

The concept of monitors has been proposed as a structuring tool for the design of systems of closely interacting processes [1,2]. In ref. [2], Hoare gives three different descriptions of the monitor concept: an informal introduction of the concept, an algorithm for interpreting monitors in terms of semaphores [3], and proof rules that can be used for verifying the corrections of a monitor. We point out in this note that the *urgent* semaphore of Hoare's interpretation algorithm is not needed for obtaining a monitor implementation that satisfies the given proof rules.

We take the view that the rules for scheduling concurrent interacting processes should be divided into two parts: (a) the rules that are sufficient for obtaining a correct operation of the whole system, and (b) the rules that increase the efficiency of the resulting schedule. In the case of monitors, the basic scheduling rules (sufficient for the correct operation of the monitor construct in general) specify in what order the monitor procedure calls of the different processes are executed. These rules are:

(a) mutual exclusion of the different calls on procedures of the same monitor,
(b) immediate resumption of processes waiting on a condition variable, as soon as the condition is signalled,
(c) otherwise *fair* scheduling of waiting processes.

More explicitly, these rules can be expressed as follows: *Mutual exclusion*: if a monitor is written such that a certain invariant assertion I about the local variables of the monitor holds (1) after initialization, (2) before returning from each monitor procedure, and (3) before each *wait* and *signal* operation within any monitor procedure, then the same assertion can be assumed to hold also (1) at the entry to each monitor procedure, and (2) after each *wait* and *signal* operation within any monitor procedure.

*Immediate resumption*: If a monitor is written such that a certain assertion *B* holds before each *signal* operation of a certain condition variable, then the same assertion can be assumed to hold also after each *wait* operation referring to the same condition variable. (The assertion *B* describes the condition(s) under which a waiting process wishes to be resumed).

These rules for mutual exclusion and immediate resumption are equivalent to Hoare's proof rules [2].

*Fair scheduling*, in the sense of Brinch-Hansen [1], means that the priority rule for selecting a delayed process for continuation must be such that no process can be delayed indefinitely in favor of more urgent processes.

Since monitors are normally used for scheduling resources, it is important that the execution of the monitor procedures are much faster than the resource they schedule (to avoid keeping the resource idle). Therefore the fair scheduling of the monitor calls has a strong influence on the system efficiency. In particular, processes that wait for entering the monitor should be given on higher priority than processes that are ready for executing code in some non-critical region.

Hoare [2] gives an interpretation of monitors in

* This work was supported in part by the National Research Council of Canada and the Ministere de l'Education du Quebec. Departement d'Informatique, Universite de Montreal, publication #233.

terms of semaphores. He uses a semaphore *mutex* for establishing mutual exclusion, a semaphore *urgent* on which those processes wait that have executed a *signal* operation, and one semaphore for each condition variable on which those processes wait that have executed a *wait* operation for the corresponding condition. Assuming fair scheduling for the processes waiting on a given semaphore, Hoare's interpretation algorithm can be shown to satisfy the above three basic monitor scheduling rules. (We note that specifying the monitor interpretation in terms of semaphores does not necessarily mean that monitors must be implemented that way. It is just a convenient way of describing how monitors function.)

The following interpretation algorithm is similar and also satisfies these rules, but it is simpler since it does not use the semaphore *urgent*. The actions to be executed at the different occasions are the following:

entry to a monitor procedure:

*P* (mutex)

exit from a monitor procedure:

*V* (mutex)

wait on a certain condition:

condcount := condcount +1;
*V* (mutex);
*P* (condsem);
condcount := condcount −1;

signal the same condition:

**if** condcount > 0 **then** {*V* (condsem); *P* (mutex)}.

If a *signal* operation is the last operation of a procedure body, it can be combined with the monitor exit as follows:

**if** condcount > 0 **then** *V* (condsem)
**else** *V* (mutex)

A similar monitor interpretation has already been described by Saxena [4]. The difference, compared to Hoare's interpretation, is that the processes that execute a *signal* operation have no priority for continuing the monitor procedure, over the processes that wait for beginning the execution of a monitor procedure call. Since the basic monitor definition given above only specifies that the scheduling must be fair, we can say that this difference is a question of efficiency only. More detailed discussions of monitor implementations and their efficiency can be found in refs. [4] and [5].

Hoare [2] mentions Dahl's suggestion that *signals* should always be the last operation of a monitor procedure. This restriction is in fact realized in the monitors of Brinch–Hansen's Concurrent Pascal [6]. There seem to be two reasons for imposing this restriction: (1) this restriction is a natural one, i.e. it is satisfied in most examples; (2) if this restriction is imposed then the semaphore *urgent* in Hoare's interpretation algorithm can be omitted, together with all operations upon it [2]. This second reason, one of efficiency, loses much of its justification in the light of the interpretation algorithm given above, which does not use the semaphore *urgent* anyway. We conclude that no obvious advantage in efficiency is obtained by restricting the signal operation in monitors to be the last operation of a procedure. Future experience will show whether occasionally a signal operation in the middle of a monitor procedure can be useful.

We have discussed, in the case of monitors, the distinction between the scheduling rules that are essential for the correct operation of a system, and the rules that only influence the efficiency of the system. We hope that, in other cases as well, this distinction can be useful for clarifying synchronization concepts.

**Acknowledgement**

I am grateful to Jean Vaucher and Pierre Desjardins for many discussions on this subject.

**References**

[1] P. Brinch-Hansen, Operating Systems Principles (Prentice-Hall, Englewood Cliffs, N.Y., 1973).
[2] C.A.R. Hoare, Monitors: an operating system structuring concept, Comm. ACM 17 (1974) 549–557.
[3] E.W. Dijkstra, Cooperating sequential processes, in Programming Languages (Ed. F. Genuys), (Academic Press, New York, 1968).
[4] A.R. Saxena, An efficient implementation of monitors and condition variables, presented at the ACM Interprocess Communication Workshop, March 1975, Santa Monica.
[5] D. Bustard, C.A.R. Hoare, R.M. McKeag, A nucleus for a multiprocessor multiprogramming system, Tech. Rep., Dep. Computer Science, The Queen's University of Belfast.
[6] P. Brinch-Hansen, The Programming Language Concurrent Pascal, IEEE Transactions on Software Eng. SE-1 (1975) 199–207.