Wirth77
Wirth N.; Towards a Discipline of Real-Time Programming, CACM 20, 8 (August 1977) pp. 577-583.

In this paper, Wirth discusses the problems in the validation of concurrent and real-time programs. To this end, he outlines a discipline expressed in the context of programming in the language Modula, which when strictly adhered to greatly simplifies the complexity of reasoning about concurrency and execution time constraints.

He begins the paper with a characterization of three classes of programs: sequential, concurrent, and real-time. He notes that the notion of time first becomes important (with respect to program verification) in the domain of multiprogramming and points out that multiprograms can and should be designed with sufficient generality that they specify the computed results independently of the absolute and relative speeds of the processors employed. He observes that adherence to this discipline allows the program to be verified solely from the logical assertions on the state of the computation after each statement and signal exchange. Departure from this discipline, however, becomes necessary when attempting to interface with the outside world. Here, the program's validity depends in part on the execution speed of the utilized processors.

He next gives an example of a concurrent program - the producer/consumer - using the language Modula. He briefly describes how Modula provides mutual exclusion (i.e. Modula's monitors) and how process synchronization is achieved through the send and wait primitives. He then describes how in the hardware world, the producer might be a card reader which can not be stopped once it has started reading a card and how the consumer must execute faster than the producer in order for the program to function. He then describes how assertions involving the execution speed can be derived from the original assertions involving the synchronization primitives.

From this example, Wirth then suggests a simple recipe for producing programs in which the verification task is not unduly difficult:

1. First formulate the entire program without any reliance on execution times, explicitly providing all synchronization signals needed for the generality.
2. For each signal that is not made available by the hardware used, analytically derive the time constraints that allow the absence of the signal.
3. Check whether the constraints are met by the host system.

Wirth, next discusses signals and semaphores and points out the dangers in using signal and wait to synchronize processes, suggesting that it might be wiser to define the emission of a signal without a wait as an error rather than an empty operation.

After this, the author goes on to discuss the language requirements for real-time programming, namely, a way to describe processes, shared variables, process synchronization primitives, and a way to determine accurate time bounds for any compiled statement sequence.

For the remainder of the paper, the author discusses the topic of real-time programming in high-level languages using processor sharing. He discusses the device modules of Modula and the use of the 'doio' to interface with hardware devices. He argues that these device modules should be run with interrupts off and if there are many devices then a priority interrupt scheme should be used. After this discussion he presents his discipline of real-time programming in Modula: Time-dependent program parts are confined to device processes which are executed with interrupts off; execution time of statements in device processes is determined statically (as if there were no processor sharing); and each doio statement is assumed to be followed by a given hidden delay d, whose bounds are given by:

d <= summation over i of T(Si)

where T(Si) is the time to execute the longest statement sequence Si between any two instances of doio in the i'th device process (measured with interrupts off).

At the end of the paper, Wirth presents a final discussion on the subject of priority and whether the signaling of a process of lower priority by one of lower priority should cause it to be dispatched. He decides that it is an open question and that in the case of Modula it is not allowed, instead a more limiting set of constraints are placed on signals so that one can not get into a situation where a lower priority process is signaled and before the process is dispatched the signal postcondition has been invalidated.

, ,

This is an important paper in that it addresses the problems of verifying concurrent and real-time programs. It provides a simple and useful discipline which can make the task much more manageable yet not restrict the programmer unduly.