BRINCHHANSEN78
Brinch Hansen, P.; Distributed Processes: A Concurrent Programming Concept; CACM 21, 11 (November 1978).

In this paper Brinch Hansen provides the base specifications for a language which generalizes most concurrent programming constructs.

The language is intended for real-time multiple microprocessor applications. Each microprocessor has only its own local memory, and so distributed processes do not use shared variables. This environment criterion is extended by requiring each distributed process to reside on its own processor. Real-time response is guaranteed by the programmer's allocation of one distributed process to each critical function.

A distributed process contains a specification of procedures that can be invoked by other processes, and of an initial block. It begins by executing its initial block. This continues until the initial block is over or a process must wait for a condition to become true. At this point it may accept requests from other processes to execute one of its externally visible procedures. It may then resume its own instruction thread or execute another external request.

As well, the language contains mechanisms to implement non-determinism through guarded commands and guarded regions. A guarded command non-deterministically selects one of a set of statements to execute if one or more of the guard conditions is found to be true. A guarded region continually checks to see if a region of code can be executed under control of guard conditions.

The paper then proceeds to provide a series of examples showing how distributed processes can easily implement semaphores, buffers, monitors, resource schedulers, process arrays and coroutines. Each example is followed by a short explanation and a suggestion for a programming exercise. These examples show that distributed processes are a powerful and elegant concurrency construct. The guarded regions and commands implement process synchronization and the process structure ensures mutual exclusion. Distributed processes are in the same class as Hoare's 'communicating sequential processes' as they are primarily tools for theoretical investigation into concurrency.

Brinch Hansen considers some of the implementation problems but as he readily admits, the details are excluded. For example, he suggests that parameter passing between distributed processes can be handled by a single input and output operation, but he gives no details as to how mutual exclusion among several calling processes might be implemented.

While the reader is thus left with many unanswered details, one of the fundamental goals of language research - that of finding simple powerful language constructs - is definitely achieved. The paper is easily read and formulates the basis for more study into a promising concurrency construct.