STOTTS82
Stotts Jr., P.D.; A Comparative Survey of Concurrent Programming Languages; ACM SIGPLAN Notices 17, 9 (September 1982).

SYNOPSIS

In this paper the author surveys the design and language features found in concurrent languages. He notes that concurrent languages are usually heavily influenced by the architectural support available in the machine that will implement the language. He then identifies two major types of machine.
1) SIMD (single instruction stream, multiple data stream), for example array machines like ILLIAC IV. Explicit synchronization primitives are not present and the hardware maintains data consistency.
2) MIMD (multiple instruction stream, multiple data stream). Here the language designer must provide the synchronization primitives between processes.

This paper deals with languages designed with the second type of machine in mind. The author then surveys that various language features that concurrent languages exhibit.

1) synchronization and communication mechanisms: These include path expressions, semaphores, monitors, coroutines, message passing and ports.

2) program modularity: for example applying the procedure/subroutine concept to processes. Other features mentioned include classes, modules and separate compilation facilities.

3) process creation: Is the number of processes fixed at compile time (static) or can processes be 'spawned' or 'forked' at run time (dynamically)?

4) process network topology specification: Again there are static topologies where the communication paths between processes are fixed at compile time and the programmer must specify the topology (e.g. a tree or a ring network) or dynamic topologies which are created and modified during execution. Note that static process creation does not necessarily imply a static topology.

5) process scheduling: How are processes scheduled? How are the problems of deadlock and indefinite postponement dealt with? Does the programmer have explicit control of the scheduler? Can processes override the normal scheduling strategy?

6) processor binding: When are logical processes bound to physical processors? When the program is written, automatically by the compiler or at run time?

7) process termination: In languages where processes can spawn child processes, do the child processes terminate when the parent does?

8) real time support: This is closely related to some of the topics above. For example explicit process scheduling with a priority scheme and preemption is usually needed. Other features usually included are explicit timeouts and exception handling facilities.

9) other features: including nondeterminate execution, exception handling, verification issues and human factors (i.e. ease of use).

The article concludes with a table comparing 13 concurrent languages using the criteria outlined above. The languages compared are Ada, CSP, Concurrent Pascal, DP, Edison, Gypsy, Mesa, Modula, Modula-2, Parlance, Path Pascal, PLITS and PL/1.

COMMENTS

A useful survey article with a large list of references (27) to the languages considered. However the title is a bit misleading since the article deals with concurrency 'in the large' (ie; processes which are relatively large chunks of code) and not with concurrency that can be exploited in vector or array machines or data flow languages.