BRINCHHANSEN75
Brinch Hansen, P.; The Programming Language Concurrent Pascal; IEEE Transactions on Software Engineering (June 1975) pp. 199-207.

This paper presents an informal description of the systems programming language Concurrent Pascal. The paper gives no implementation details of the language nor discusses the syntax of the language in any great detail. Instead, the paper restricts itself to the semantics of the concurrency features and how the can be used to aid in the implementation of operating systems.

The paper is divided into two sections. In the first section, processes and monitors are introduced as new abstract data types. A process is defined as a private data structure and a sequential program that can operate on that data. A monitor consists of a shared data structure and a set of synchronizing operations (procedures) that processes can execute to access the data in a controlled way. The virtual machine upon which the concurrent program is running handles the short term scheduling of simultaneous monitor calls while semaphores are used to perform medium delay scheduling and synchronization of processes. These semaphores are given the names delay - which blocks the running process placing it in a queue, and continue - which causes a queued process to resume execution. The queue is a simple data type and it may contain at most one process. In addition to the general monitor data type, Concurrent Pascal contains a second monitor type known as a class. The difference between a class and a regular monitor is that the exclusive access to class procedures to class variables can be guaranteed at compile time. Thus, the virtual machine does not have to perform short term scheduling of simultaneous calls to class procedures at run time so class calls are considerably faster than monitor calls.

For the remainder of the section Brinch Hansen goes on to explain how these entities are used to construct a hierarchically structured operating system and he talks a little about the scope rules for the new data abstractions and how the structures lend themselves to implementing hierarchically structured systems.

In the second section, Brinch Hansen informally introduces the language notation of Concurrent Pascal by means of a detailed example. In a second example, the author shows how the seemingly restrictive single process queues can be used to construct a multi-process queue. Thus, the semaphores of Concurrent Pascal are shown to have the same power as the multi-process constructs suggested in other language designs (i.e. ’event’ and ’condition’ queues).

This paper serves as a good introduction to process/monitor styled concurrent languages. The language itself is of particular importance being one of the first concurrent language designs and one of the first written with systems implementation in mind.