From Monitors to Monitors: an Early History of Concurrency Primitives

Troy K. Astarte
troy.astarte@ncl.ac.uk

June 4, 2021

Abstract (199)

As computers became multi-component systems in the 1950s, handling the speed differentials efficiently was identified as a major challenge. The desire for better understanding and control of 'concurrency' spread into hardware, software, and formalism. This work-in-progress talk traces some early attempts to find primitives for concurrency.

Initially, system programs called 'monitors' were used for directly managing synchronisation. Attempts to reframe synchronisation at a higher level led to a series of algorithms; Dijkstra's semaphores were a reaction to the algorithms' complexity. Towards the end of the 1960s, new desires for clearer 'structured programming' created a milieu in which Dijkstra, Hoare, and Brinch Hansen (among others) aimed for a concurrency primitive which embodied the new view of programming. Via conditional critical regions and Dijkstra's 'secretaries', the monitor appeared to provide the desired encapsulation. A few programming languages adopted the construct: we finish by considering Modula and Concurrent Pascal.

This story shows the effects of grappling with the power brought by concurrency while trying to manage greater conceptual (and textual) complexity. The actors involved sought a balance between abstraction and tolerable implementation and their work demonstrates changing priorities in programming and computer science.

From Monitors to Monitors: an Early History of Concurrency Primitives
Extended abstract

This talk is an early output from the author's current project on the history of concurrency and sketches the early search for programming primitives. Starting by looking briefly at the management of peripherals that drove initial work, the talk covers algorithms, semaphores, conditional critical regions, and monitors.

Different parts of computers operate at different speeds. As soon as computer systems were sufficiently complex, managing these differentials efficiently became a serious concern. This was tackled as far back as 1955: Rochester wrote about 'multiprogramming', referring to the ability of a central calculator to share focus between polling I/O and processing data [Roc55]. Gill used the terms 'parallel programming' and 'timesharing' in the context of interference between multiple CPUs, or multiple programs sharing a CPU [Gil58]. The basic challenge was to handle the spread of processing across space or time and it was realised these could be treated as the same problem [Con63].

An early approach was to use the system controlling the computer to manage synchronisation directly [Cod62]. This 'monitor' (as Brinch Hansen called it) could be hard to use and understand, with its workings hidden away inside assembly programs. By Codd's 1962 summary paper key concepts such as threading, shared variables, critical sections, and mutual exclusion were already identified. The prime challenge: preventing the interaction of critical resources, and unexpected outcomes from shared memory.

The early 60s was the era of algorithms and many appeared for managing concurrent processes. The core idea here is conditional progress: Dekker's algorithm (reported in [Dij62]) arbitrates between competing processes by alternating priority in turn. The algorithm did not generalise easily to multiple processes; Lamport's Bakery algorithm was more successful but the complexity of algorithms and proofs thereof drove a search for something simpler.

Taking a metaphor from railways, Dijkstra proposed a special kind of shared integer called a 'semaphore' [Dij62]. These synchronisation primitives controlled access to critical sections using two operations, P and V (the full names he gave varied). Semaphores allowed more concise programming, and put control into the hands of programmers. However, deadlocks loomed for certain problems (for example the classic dining philosophers).

From the early 1970s, Dijkstra, Hoare, and Brinch Hansen were all searching for ways to improve programming for concurrency. The three fed on each others' work and discussions, making it difficult (and unhelpful) to apportion individual credit. This was the age of structured programming and Pascal: 'elegance' became not just an aesthetic imperative but a correctness one. Working towards a concept, Hoare proposed 'critical regions': areas of code marked off for mutual

exclusion over a shared resource, sometimes with conditions for entry [Hoa72]. The title of this paper ('Towards a theory of parallel programming') showed Hoare's desire to generalise from a practical synchronisation primitive: part of that time period's fascination with theorising computation.

Both Brinch Hansen and Dijkstra provided ideas for improving this notion; Dijkstra's 'secretary' is worth noting for its 1970s attitude towards workplace gender roles [Dij71]. Simula-67's class concept provided the key: encapsulation. Shared resources were grouped together with the operations that could access them in a construct with mutual exclusion baked in, called a 'monitor' [Hoa74]. Inbuilt queues and signalling operations controlled waiting. Monitors took the responsibility for correctly managing synchronisation away from individual processes and made a system more reliable overall—they also obeyed the hallowed principles of structured programming.

Monitors became the sole construct for managing concurrency in a few programming languages released in the 1970s; Concurrent Pascal and Modula are of particular note. These demonstrated the workability of the monitor as a realistic synchronisation primitive; unfortunately implementation exposed underlying complexity. Recursive or interfering monitor calls were difficult to get right and a proper understanding of the queueing system was essential. The monitor idea declined in popularity towards into the 80s as communication became the new focus for concurrency. However, the extensive remarks at the end of [Han96] show that monitors had a serious impact on concurrency and their principles continue to influence object-oriented programming to this day.

The search for a useful concurrency primitive demonstrates an example of collaborative ideation, with ideas bouncing between groups and individuals. The changing scope and focus also reflects changing agendas for computing: from systems and assembly programming towards high-level languages, abstraction, and the lofty ideals of structured programming. The work paved the way for increased focus on formalisation and theory-building.

1 Acknowledgements

This work is supported by Leverhulme Trust grant RPG-2019-020. Thanks to Cliff Jones for providing comments.

References

[Cod62] EF Codd. "Multiprogramming". In: *Advances in Computers*. Ed. by F. Alt and M. Rubinoff. Vol. 3. Elsevier, 1962, pp. 77–153.

[Con63] Melvin E Conway. "A multiprocessor system design". In: *Proceedings of the November 12-14, 1963, Fall Joint Computer Conference*. 1963, pp. 139–146.

[Dij62] E. W. Dijkstra. "Over de sequetialiteit van procesbeschrijvingen [On the sequentiality of process descriptions]". Circulated privately, available in Texas Archive. EWD35. Date inferred. 1962. URL: https://www.cs.utexas.edu/users/EWD/translations/EWD35-English.html.

[Dij71] E. W. Dijkstra. "Hierarchical Ordering of Sequential Processes". In: *Acta Informatica* 1 (1971), pp. 115–138.

[Gil58] Stanley Gill. "Parallel programming". In: *The computer journal* 1.1 (1958), pp. 2–10.

[Han96] Per Brinch Hansen. "Monitors and Concurrent Pascal: a personal history". In: *History of programming languages—II*. Ed. by Thomas J. Bergin and Richard G. Gibson. New York, NY, USA: ACM Press, 1996, pp. 121–172. ISBN: 0-201-89502-1.

[Hoa72] Charles Antony Richard Hoare. "Towards a theory of parallel programming". In: *The origin of concurrent programming*. Springer-Verlag, 1972, pp. 231–244.

[Hoa74] C. A. R. Hoare. "Monitors: An Operating System Structuring Concept". In: *Communications of the ACM* 17.10 (1974), pp. 549–557.

[Roc55] Nathaniel Rochester. "The computer and its peripheral equipment". In: *Proceedings of the Eastern Joint AIEE-IRE Computer Conference: Computers in business and industrial systems*. 1955, pp. 64–69.