Viswanathan Manickam, Alex Aravind, *If a picture is worth a thousand words, what would an animation be worth?*, Proceedings of the 16th Western Canadian Conference on Computing Education, p. 28-32, May 06-07, 2011, Prince George, British Columbia, Canada.

Cites: Dijkstra's *Solution of a problem in concurrent programming control*
Knuth's *Additional comments on a problem in concurrent programming control*
Lamport's *A new solution of Dijkstra's concurrent programming problem*

This article focuses on the teach-ability of concurrent programming through the use of visual, animated aids. Driven by the necessity to produce programmers who thoroughly understand the concept of concurrent programming, an animated program was produced to introduce students to the ideas and algorithms behind concurrent programming. The developed program illustrates thread interactivity through the exploration of mutual exclusion. A “room exclusion” problem has been designed such that multiple critical sections can be introduced. Several famous algorithms, including Dijkstra's Algorithm, Knuth's Algorithm, and Lamport's Bakery Algorithm were included in the animations so that the user would be exposed to a comprehensive set of methods of dealing with mutual exclusion. The simulator program displays an animated representation of the room simulation, as well as a description of the algorithm being used, the parameters of the problem, as well as a visualization where each thread is in its code. Finally, the simulator program itself is built to allow the user to implement their own algorithms.

This work does not challenge the validity of any concurrent programing algorithm in existence; it serves more as a supporter for an atypical approach to teaching the concepts of concurrent programming. This is as opposed to a method where students are only taught through things such as pseudo-code and text, leaving them to create their own mental model of how concurrent programming can be implemented. In creating this simulation program, the authors have filled an essential need. The concept of such things as critical sections are simple enough for anyone to understand – the physicality of the world around us eliminates the need for any sort of conscious thought about shared resources, whatever is, is. While computing, however, this physical barrier is broken. Initially, it's difficult to recognize this fact; it would seem completely unnatural that one would walk into an occupied toilet stall and attempt to simultaneously use the same toilet another is using. However, during concurrent programming, such instances could occur if there is a lack of understanding of critical sections. Thus, it is only a logical move to create a visualization of how all of these elements come together in concurrent programming. The work itself does not present anything groundbreaking or new, but it does provide new thought on the instruction of concurrent programming. Quality wise, this paper could be evaluated as good. A grant was given to the authors who were set in developing a new method of educating others in concurrent programming, and they achieved their goal. It's impact on the field of concurrent programming itself will probably be very mild, but it does provide a good foundation for the teaching of the subject.

Marcos K. Aguilera, *A pleasant stroll through the land of many creatures*, ACM SIGACT News, v.35 n.2, June 2004

Cites: Dijkstra's *Solution of a problem in concurrent programming control*
Lamport's *A new solution of Dijkstra's concurrent programming problem*

This paper discusses newer and older algorithms and how they deal with a possibly infinite number of processes. The author argues that the possibility of infinitely many threads, or rather a lack of parameterized and/or limited inputs, requires these algorithms to be more generic and elegant. An algorithm that can handle infinite processes implies that there will be no need for reconfiguration, simple crash handling, and guaranteed progress for processes. The author then moves on to consider the situations in which infinite processes will and won't occur and opts to focus on a model where a system has an infinite amount of processes and an infinite amount of processes can keep arriving forever. This model is general enough to cover all non-theoretical models of infinite process. The first issue is that of naming individual processes. In models without processes that can perpetually interrupt others, naming is simple, a name is assigned to that process when it is run. However, with the model in which a process can be stalled forever by another, this method doesn't work. This is fixed by introducing a limit on competing threads such that one certain thread is left waiting forever. A similar problem addressed is that of order. “Wait-free” algorithms work with prioritizing processes based upon their duration, such that faster processes will be given the chance to complete without having to wait on slower processes. Again, a similar algorithm with modifications can be applied to work with infinite processes models. The same problem of starvation can occur with wait-free algorithms, thus a “weak-counter” is introduced. Similar to the process naming fix for this model, only a certain number of quicker processes are allowed to pass the larger processes. We are also introduced to the idea of “snapshots” that allow the processor to look at the sizes of each process as well as “group-membership” which determines what processes are in the system at that time. Dijkstra's algorithm and Lamport's Bakery algorithm are rerouted to fit the needs of infinite processes. Memory is the next issue brought up, and the author considers a variety of solutions, examining why what works for finite systems does not work for infinite systems. The proposed solution is to add another layer of processes that decide upon how to delegate memory to the infinitely many incoming processes. Finally, the paper concludes by asking more questions about the solvability of infinite machines.

If there was anything that ever needed a visualization program from the article about animating concurrent programming algorithms, it's this. The text was dense, and the pseudo-code did little to help illustrate the algorithms being explained. All of the algorithms mentioned were easy to understand after a bit of reading; the complexity of the writing is far greater than that of the algorithms themselves. The name of the article was completely misleading. The stroll was more of a hike through the woods while following a near-invisible path of tiny breadcrumbs, there was a severe lack of “creatures” (unless the author was referring to the readers scratching their heads as they perused the article), and only a true masochist would consider reading this paper pleasant. However, considering that this article was published through SIGACT, whose website boasts “Work in this field is often distinguished by its emphasis on mathematical technique and rigor,” this paper is of good quality. Both informal and solid proofs drive the point to the reader, and although dense, each bit of text effectively explains whatever point the author is trying to make.

K Alagarsamy, Some myths about famous mutual exclusion algorithms, ACM SIGACT News, v. 34 n.3, p.94-103, September 2003

Cites: Dijkstra's *Solution of a problem in concurrent programming control*

This paper discusses “myths” that surround Dijkstra's algorithm, Dekker's algorithm, and Peterson's algorithm and aims to dissolve said “myths.” Hopeful to clear up the muddle that common misinterpretations and misleading references, the author introduces each myth and then proceeds to prove them false. The first issue we run into is with a “generalization” proposed by A.J. Martin in his *A New Generalization of Dekker's Mutual Exclusion Problem.* The author first introduces Dekker's original algorithm in which two compete for a shared resource. Dekker has each vying for the resource, having the one that has accessed the shared resource least recently winning. A couple of definitions of “generalization” are included, creating emphasis on the fact that generalizations must cover all cases in that they generalize and that a generalization must produce the original outputs when original inputs are used. The author uses these definitions to prove that Martin's algorithm is not a generalization of Dekker's and thus should not be considered such. Additionally, although Dijkstra does not himself claim that his algorithms are generalizations of Dekker's, many consider them to be just that. All three supposed generalizations of the algorithm are susceptible to starvation, while Dekker's clearly never will reach that state. He proves how each can starve processes, and with that, all fail to meet the defined properties of a generalization. We are then provided with an actual generalization of Dekker's algorithm which meets all conditions of the original, and thus can be considered a generalization. This is used to dispel the myth that Dekker's algorithm can be trivially modified so solve n-process case. Again, with the examination of Dijkstra's and Martin's algorithms, we can see that simply attempting to tweak the number of processes causes starvation and cannot be considered generalizations. The given generalization had to be rebuilt to accommodate n-processes. The final myth that the author investigates is that Peterson's algorithm assures bounded bypass. After detailing the specifics of the algorithm, the author introduces an instance in which the algorithm does not guarantee bounded bypass. With three processes, a situation can occur in which one process can perpetually bypass another by continually setting certain conditions. The paper concludes noting that concurrent programming often emits such subtle errors and minor differences as exemplified by certain cited papers.

Of the three papers in this annotated bibliography, this one is of the greatest quality. All work is concise and well organized and written in clear language. The organization of this paper was well constructed – although it dealt with several different algorithms, the authors thoughts were easily traceable as well as considerately placed. Each section is highlighted by pseudo-code that is relevant and readable. Although the intended audience would be those who already have known of these myths, the thoroughness of the introduction creates accessibility for anyone with an understanding of computing basics. Each argument is well placed and effectively supported with straightforward proofs, pseudo-code, and tables.