LETTERS

TERMINATE WITH EXTREME PREJUDICE

To the Editor:

I am writing to raise awareness that a pervasive software paradigm is prone to a serious performance pitfall. At least one widespread instance of the problem has been remarkably adept at evading detection.

The paradigm in question is the *work queue* at the heart of myriad programs: software repeatedly dequeues a task and performs corresponding work, which may enqueue new tasks, until the queue is empty. The performance bug arises when output attains its final state long before the work queue drains; subsequent effort to empty the queue is wasted because it does not change the output.

The obvious solution—a "stop when done" termination test—is not always obvious to algorithm designers and developers coding in the work queue paradigm. More than once, I've seen production software that descends into a tragicomic frenzy of needless toil merely to drain a work queue, with no externally observable effect whatsoever beyond raising the CPU temperature. Work queues naturally incline toward self-inflicted busywork unless thoughtfully supervised.

Unfortunately, such vigilance is itself exceptionally difficult. We might hope that a modicum of peer review would suffice to exorcise gratuitous inefficiency from queue-centric designs. One of the most widely known elementary algorithms of all time, however, shows that extensive scrutiny is not proof against this problem.

Top textbooks, such as those by Cormen et al. (*Introduction to Algorithms,* third edition) and Sedgewick and Wayne (*Algorithms*, fourth edition), have withstood decades of intense critical attention from generations of academics and practitioners. The breadth-first search algorithm presented in these and similar texts terminates when its work queue drains, which may occur long after all output is finalized.

Compared to an "efficient BFS" that terminates when the output reaches quiescence, the classic textbook BFS is like a penny in a fuse box: never better and sometimes catastrophically worse—that is, sometimes slower by a factor proportional to the number of vertices in the input graph. For detailed evaluations, see https://queue.acm.org/detail.cfm?id=3424304.

> The paradigm in question is the work queue at the heart of myriad programs: software repeatedly dequeues a task and performs corresponding work, which may enqueue new tasks, until the queue is empty.

The root cause of the "work queue run mad" antipattern is a confusion of ends versus means. Work queues are the latter. They provide reminders to be considered, not commands to be blindly obeyed, as computations unfold. Unfortunately, experience shows that work queues can distract attention from the simple fact that every program's purpose is to compute its output.

Terence Kelly
tpkelly@acm.org

Digital Object Identifier 10.1109/MC.2021.3055874
Date of current version: 30 July 2021