Real Time, Real People

Scott Davidson

As our digital systems get involved in more of our lives, they interact with the world in more ways. Since the world waits for no IoT device, our devices must react to the world in time to do their jobs. If your car has a collision avoidance system, it had better press your brakes before, not after, you slam into the car ahead of you.

How to design and verify systems that interact with our fast-moving world is the topic of this issue of Design and Test on Time-Critical Systems. It is not an easy job.

Why? We must ensure that systems, under a variety of loads and environments, can react to stimuli in the required time. We must be able to set priorities for reacting to critical events.

Thinking in programming terms about fast response takes a different mindset from writing code in an object-oriented language. When more of us worked closer to the machines, we learned this. My undergraduate assembly language class was on the PDP-1, where the first Spacewar game was developed. Our final class assignment was to write a program to play Conway’s Game of Life, and handle the interrupts quickly enough so that the display didn’t flicker. Does anyone learn real-time programming as a basic part of their toolset anymore?

An excellent insight from several papers in this issue is that our programming languages don’t support time as a basic concept. A very few languages handle asynchronous inputs at all, let alone well. It is doubly difficult to implement a time-critical system using languages that don’t have the structures needed.

Not all real-time systems are implemented in hardware. Each of us is an example of a real-time system. We can (usually) avoid collisions while driving without computer assistance. If someone unexpectedly throws something at us, we can often react quickly enough to catch it or avoid it.

We aren’t born with this ability. If you play catch with a baby, you can see him figuring out what to do with the ball. At first, this takes so long that he reaches out his hands too late. Eventually he catches on, and reacts quickly enough to make the catch.

Several papers, in this issue, mention worst-case execution times (WCETs), which determine if a system is fast enough to accomplish a given task. Our WCETs are not built in. As we practice a task, we get better at it and react faster. Our WCETs are adaptive. We have different inherent speeds. We learn them, and stick to the tasks we can accomplish. I’ll never catch or pass a basketball as quickly as Steph Curry.

Perhaps our real-time systems can learn how to accomplish their tasks faster also, and variations in their natural speeds can lead to variations in the tasks they are expected to perform, similar to speed binning. If the system is complex enough, it can find the datapaths and resources needed to optimize the time required to do a task. This process might go faster if the first versions of a real-time system had some help in finding how to do their best.

Coaches help people get better and faster. Perhaps coaching machines is a job for the future.

Direct questions and comments about this department to Scott Davidson; davidson.scott687@gmail.com.

Digital Object Identifier 10.1109/MDAT.2018.2794402
Date of current version: 23 March 2018.