Keyboard Debouncing

When writing code to debounce a keyboard, it was found that the magic number was 20 msec (twenty milliseconds).

If a keystroke took longer than 20 milliseconds to be recognized, the keyboard "feels slow" to the human operator.

If the software sampled the keystrokes too quickly (e.g. 1 msec), then the sampler found that the key switch actually, physically, bounced (1 to 0 to 1 to 0, etc.) and settled down to a final, steady value before the magic number of 20 msec.

I don't know how this magic number has changed with respect to more modern keyboards based on newer technologies.

Music Software

Music software - DAWs (Digital Audio Workstations) - need to worry about sluggishness and latency.

Piano-style keyboards are *polyphonic* - the musician can press more than one key at once (or in sequence) and expects to hear notes "right away".

Playback of multiple tracks is/was also a problem. When multiple tracks are played by the DAW, the musician expects to hear them all "in sync". If the number of tracks (and effects processors contained in the track - e.g. reverb) exceeds the ability of the processor to keep up, the music will sound sluggish and "out of sync".

Faster hardware has ameliorated this problem. The same software runs faster and can process more tracks before maxing out. I expect that software built with

preemptive multitasking (i.e. the norm) will max out sooner than software optimized to handle keyboard reading.

Game Industry

I expect, but don't have the direct experience, that this problem also affects the game industry. Production Engineering of software can save money when shipping a zillion xboxes.

IoT

I expect that IoT will run into this problem. Using Linux on a small processor is (usually) over-kill for what is actually needed, and, Linux could be production engineered away if economy of scale requires it.

Concurrency (and parallelism) can be built less expensively than by using Linux.[1]

One needs to make a trade-off assessment - how many $'s can by saved in hardware by spending $'s on software optimization? And, reliability enters the equation, too - preemptive multitasking is notoriously hard to get right (http://www.cs.cornell.edu/courses/cs614/1999sp/papers/pathfinder.html).[2]

---

[1] See my various essays, e.g. "Box-and-Arrow for Concurrency".

[2] This description claims that the problem was due to lack of correctness proof. The real problem, IMO, is that memory sharing and full preemption were used. Understanding that full-preemption is only one of the possible solutions to the problem (an over-kill, over-generalized solution), might have generated a different solution and the bug might have been detected earlier.