A Practical Comparison of Scheduling Algorithms for Mixed Criticality Systems

Authors: Carl Cristian Arlock and Edward Linderoth-Olson

The conglomeration and co-location of processes onto single systems has moved to the fore in the automotive industry. This poses the problem of interference between the many processes of varying criticality. Powerful and efficient management systems are needed to sort this problem, but which are available? And what is the performance of these systems when it comes to overhead?

What is then mixed criticality, the environment that these systems, or algorithms, are supposed to manage? It occurs when, for example, you want the control programme for a car's braking system to run on the same processor as the media player, usually in order to save money by not having to install two separate control computers. Unfortunately, we can never be entirely sure how much power a particular process needs -- we can make educated guesses but never be entirely certain.

In order to make your educated guess a bit safer you can always pad it a bit, and you can pad it more or less depending on how sure you want to be that the process will not require more processing power than the guess. This results in multiple levels of criticality, that is, different levels of how confident we can be that a process will not run into trouble because we allocated too little power to it. Multiple criticality scheduling allows us to have programmes with different levels running on the same processor without getting in each other's way.

Moving on to the actual project, an inventory of the available algorithms produced quite a lot of results. This field has been very active in the past few years and a lot of different methods of managing these systems have been produced, but after some discussion we selected three different algorithms: Adaptive Mixed Criticality (AMC), Zero-Slack Scheduling (ZSS) and Earliest Deadline First Demand Bound (EDF-DB). To put it simply, AMC was chosen as it appeared to be one of the most reliable and easy to use, EDF-DB because it was the most recently invented and advanced, and ZSS because it seemed to be completely different from any of the other ones and thus should be interesting.

In order to test how much overhead each of these algorithms generated, they were implemented into a car operating system called SmartSAR, developed at Zhejiang University in the People's Republic. Every algorithm had two specific phases, one was when it was managing the running of the programmes on the CPU, called run-time, and one before when calculations are performed to make the system ready for running.

Both ZSS and EDF-DB required large amounts of precalculations, taking several hours to complete. In EDF-DB this was compensated by the fact that it was very fast in the run-time, and also worked for many more systems than the other two. ZSS on the other hand would only work for just over half the systems we constructed, and also generated quite a lot of overhead. AMC was much quicker than either of them when it came to the pre-calculations, but slower in run-time overhead. So put simply, EDF-DB is preferable if you can afford the big off-line calculations, but AMC might be a better option if you can afford the slightly higher run-time overhead.