Exhaustive Testing for a Scheduler of POSIX-compliant OS

2010146 Akira Hasegawa

Nowadays, in the field of embedded systems, IoT and AI features are in high demand. As a result, POSIX-compliant operating systems (POSIX-compliant OSs), such as Linux, are used as the OSs. Since typical POSIX-compliant OSs and middleware applications are provided as open source software and readily available, there is a significant advantage that developers can make a prototype easily and quickly. Moreover, typically, POSIX-compliant OSs have much more flexibility to add additional functions than traditional real-time OSs.

However, since POSIX is originally a specification for general-purpose OSs, there are parts where strict behavior is not specified to improve the versatility. Therefore, even if OSs comply with POSIX, their behavior can be completely different. For instance, POSIX does not specify how to allocate threads to each CPU core, and it depends on the implementation of each OS. In addition, since the behavior of OS itself is more complicated than traditional OSs, testing for them is challenging.

Unlike desktop and server applications computers, embedded systems are often used when failures can enormously impact human life. For instance, recently, an application for self-driving cars has also been considered, and Adaptive AUTOSAR, an extension of POSIX, has already been created. In such a mission-critical system, it is highly possible that the introduction of bugs will severely impact society and human life, so it is crucial to ensure the quality through testing or verification.

Especially, scheduling functions are essential for mission-critical systems, and they must not have bugs. Scheduling is the process of managing the execution order of each program (threads/processes). If a bug is included here, it will easily affect the entire system, leading to malfunctions or hangs.

Scheduling-related APIs are stateful; that is, the execution results change depending on the calling order of the APIs. In this case, model-based testing (MBT) is used. To apply MBT, first, we need to create a model that represents the desired behavior using the specification such as POSIX. And secondly, create test cases using the model and execute them.

Through the exhaustive testing for the scheduling-related APIs, we can obtain confidence that the scheduling functions work as we think. However, since POSIX has a problem that did not occur in testing for traditional real-time OSs, such as OSEK/VDX-compliant OSs, the traditional MBT method cannot be directly applied to POSIX-compliant OSs. For instance, consider a situation where the limitation of parallel running threads is two and two threads with low priority, TID 1 and TID 2, are already running, and a new thread TID 3, which has higher

priority, is now created. In this case, two behaviors satisfy POSIX: (1) since TID 1 has lower priority than TID 3, TID 1 is suspended, and TID 3 starts to run instead of TID 1, and (2) since TID 2 has lower priority than TID 3, TID 2 is suspended and TID 3 starts to run instead of TID 2. In this research, we call this property that the execution results are not uniquely determined from the specification as the indeterminacy of the execution result.

Traditional real-time OSs such as OSEK/VDX-compliant OSs do not have this property, and the specification determines only one correct behavior. However, since POSIX-compliant OSs have the property, the traditional MBT is not reasonable for POSIX-compliant OSs. In this study, we have proposed an MBT method that supports the indeterminacy of execution results and achieved exhaustive testing for the scheduling-related APIs of POSIX.