ELSEVIER

Microprocessors and Microsystems 24 (2001) 463–478

MICROPROCESSORS AND MICROSYSTEMS

www.elsevier.nl/locate/micpro

Testing distributed real-time systems

H. Thane*, H. Hansson

Department of Computer Engineering, Mälardalen Real-Time Research Center, Mälardalen University, P.O. Box 883, Västerås S-721 23, Sweden

Received 5 November 2000; accepted 6 November 2000

Abstract

For testing of sequential software it is usually sufficient to provide the same input (and program state) in order to reproduce the output. For real-time systems (RTS), on the other hand, we need also to control, or observe, the timing and order of the inputs. If the system additionally is multitasking, we also need to take timing and the concurrency of the executing tasks into account.

In this paper we present a method for deterministic testing of multitasking RTS, which allows explorative investigations of real-time system behavior. The method includes an analysis technique that given a set of tasks and a schedule derives all execution orderings that can occur during run-time. These orderings correspond to the possible inter-leavings of the executing tasks. The method also includes a testing strategy that using the derived execution orderings can achieve deterministic, and even reproducible, testing of RTS. Since, each execution ordering can be regarded as a sequential program, it becomes possible to use techniques for testing of sequential software in testing multitasking real-time system software. We also show how this analysis and testing strategy can be extended to encompass distributed computations, communication latencies and the effects of global clock synchronization. The number of execution orderings is an objective measure of the testability of a system since it indicates how many behaviors the system can exhibit during runtime. In the pursuit of finding errors we must thus cover all these execution orderings. The fewer the orderings the better the testability.

Keywords: Determinism; Testing; Distributed real-time systems; Reproducibility

1. Introduction

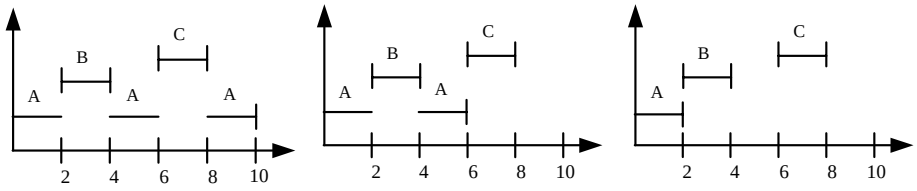
In this paper we will present a novel method for the integration testing of multitasking real-time systems (RTS) and distributed real-time systems (DRTS). This method achieves deterministic testing of RTS and DRTS by accounting for the effects of scheduling, jitter in RTS, and the inherent parallelism of DRTS applications.

A real-time system is by definition correct if it performs the correct *function* at the correct *time*. Using real-time scheduling theory we can provide guarantees that each task in the system will meet its timing requirements [1,13,27], given that the basic assumptions, e.g. task execution times and periodicity, are not violated at run-time. However, scheduling theory does not give any guarantees for the functional behavior of the system, i.e. that the computed values are correct. To assess the functional correctness other types of analysis are required. One possibility, although still immature, is to use formal methods to verify certain functional and temporal properties of a model of the system. The formally verified properties are then guaranteed to hold in the real system, as long as the model assumptions are not violated. When it comes to validating the underlying assumptions (e.g. execution times, synchronization order and the correspondence between specification and implemented code) we must use dynamic verification techniques which explore and investigate the run-time behavior of the real system. Testing [16] is the most commonly used technique, and is also the state-of-practice in functional verification.

Reproducible and deterministic testing of sequential programs can be achieved by controlling the sequence of inputs and the start conditions [14]. That is, given the same initial state and inputs, the sequential program will deterministically produce the same output on repeated executions, even in the presence of systematic faults [15]. Reproducibility is essential when performing regression testing or cyclic debugging, where the same test cases are run repeatedly with the intent to validate that either an error correction had the desired effect, or simply to make it possible to find the error when a failure has been observed [12]. However, trying to directly apply test techniques for sequential programs on DRTS is bound to lead to non-determinism and non-reproducibility, because control is only forced on the inputs, disregarding the significance of order and timing

* Corresponding author. Tel.: +46-21-101300; fax: +46-21-101544.
E-mail address: hte@mdh.se (H. Thane).

0141-9331/00/$ - see front matter
PII: S0141-9331(00)00099-5



(a)$WCET_A$=6 (b) $WCET_A$=4 (c) C_A=2

Fig. 1. Three different execution order scenarios.

of the executing and communicating tasks. Any intrusive observation of a distributed real-time system will, in addition, incur a temporal probe-effect [6,18] that subsequently will affect the temporal and functional behavior of the system.

The main contribution of this paper is a method for achieving deterministic testing of DRTS. We will specifically address task sets with recurring release patterns, executing in a distributed system where the scheduling on each node is handled by a priority driven preemptive scheduler. This includes statically scheduled systems that are subject to preemption [17,27], as well as strictly periodic fixed priority systems [1,13]. The method aims at transforming the non-deterministic DRTS testing problem into a set of deterministic sequential program testing problems. This is achieved by deriving all the possible execution orderings of the distributed real-time system and regarding each of them as a sequential program. A formal definition of what actually constitutes an execution order scenario will be given later in the paper. The following small example presents the underlying intuition:

Consider Fig. 1a, which depicts the execution of the tasks *A*, *B* and *C* during one instant of the repetitive pattern of executions dictated by an off-line generated static schedule, of length equal to the Least Common Multiple (LCM) of the task period times. The tasks have fixed execution times, i.e. the worst and best-case execution times coincide ($\mathrm{WCET}_i = \mathrm{BCET}_i$, for $i \in \{A,B,C\}$). A task with later release time is assigned higher priority. These non-varying execution times have the effect of only yielding one possible execution scenario during the LCM, as depicted in Fig. 1a. However, if e.g. task *A* would have a minimum execution time of 2 ($\mathrm{BCET}_A = 2$; $\mathrm{WCET}_A = 6$) we would get three possible execution scenarios, as depicted in Fig. 1a–c. In addition to the execution order scenario in Fig. 1a, there are now possibilities for *A* to complete before *C* is released (Fig. 1b), and for *A* to complete before *B* is released (Fig. 1c).

Given that these different scenarios yield different system behaviors for the same input, because of the order or timing of the produced outputs, or because of unwanted side effects via unspecified interfaces (caused by bugs), we would, by using regular testing techniques for sequential programs, get non-deterministic results. For example, assume that all tasks use a common resource *X* on which they carry out operations. Assume further that they receive input immediately when starting, and deliver output at their termination.

We would then, for the different scenarios depicted in Fig. 1a–c, get different results as follows:

- The scenario in Fig. 1a would give $A(X)$, $B(X)$ and $C(B(X))$.
- The scenario in Fig. 1b would give $A(X)$, $B(X)$ and $C(A(X))$.
- The scenario in Fig. 1c would give $A(X)$, $B(A(X))$ and $C(B(A(X)))$.

Making use of the information that the real-time system depends on the execution orderings of the involved tasks, we can achieve deterministic testing, since for the same input to the tasks and the same execution ordering, the system will deliver the same output on repeated executions.

In order to address the scenario dependent behavior we suggest the following testing strategy.

1. Identify all possible execution order scenarios for each scheduled node in the system during a single instance of the release pattern of tasks with duration *T*; typically equal to the LCM of the period times of the involved tasks.
2. Test the system using any regular testing technique of choice, and monitor for each test case which execution order scenario is run during [0,*T*], i.e. which, when and in what order jobs are started, preempted and completed. By jobs we mean single instances of the recurring tasks during *T*.
3. Map test case and output onto the correct execution ordering, based on observation.
4. Repeat 2–3 until the coverage sought is achieved.

The probe effect is avoided by making the probes part of the design and then letting them remain in the target system [24,25].

1.1. Contributions

In this paper we present a method for functional integration testing of multitasking RTS and DRTS: This method is to our knowledge the first testing method to fully encompass scheduling of DRTS. The main activities of the testing method are to:

- identify the execution order scenarios for each node in a distributed real-time system;