ABSTRACT
This research is motivated by the practical needs in the porting of embedded software over platforms and the well-known multiprocessor anomaly [2, 3]. In particular, we consider the task scheduling problem when the system configuration changes. We show that new violations of the timing constraints of tasks might occur even when a more powerful processor or device is adopted. The concept of scheduler stability and rules are then proposed to prevent scheduling anomaly from happening for task executions that might be involved with task synchronization or I/O access. Finally, we explore policies in bounding the the duration time of scheduling anomaly.

Categories and Subject Descriptors
D.4.1 [OPERATING SYSTEMS]: Process Management—Multiprocessing, multiprogramming, multitasking ; F.2.2 [ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY]: Nonnumerical Algorithms and Problems—Sequencing and scheduling

General Terms
Algorithms, Standardization, Theory

Keywords
Real-Time Task Scheduling, Process Synchronization

1. INTRODUCTION
One of the key factors in the development of successful embedded-system products is how to reuse software and hardware IP’s so that the time to the market is shortened. The issues in software-IP reuse is very different from those for hardware-IP reuse, especially when many applications today must have good response time and reliability requirements. Such requirements in system implementations often imply the needs of software portability over platforms, not just from the functionality point of view but also in terms of the timing behaviors of the target software [1]. The motivation of this research could be better illustrated by an example schedule over a uniprocessor system in which a violation of timing constraints might occur, due to the upgrading of the processor:

Figure 1: An anomaly of task executions due to hardware upgrading
As shown in Figure 1.(b), task1 now completes its execution later because the upgrading of the processor lets task2 lock a non-preemptable resource before task1 arrives.

Figure 2: An anomaly of task executions over a uniprocessor system with an I/O device
Figure 2.(b) also shows task1 completes later in a system with a more powerful processor. In addition to that, the delay on task1’s execution is more significant because of I/O access (even though parallelism is observed in this example). The scheduling anomaly should be considered the variety of embedded system products and their platforms when softwares are ported among platforms/systems with different processor/device speeds. The technical problem should be on the reasons behind the anomaly and the way to avoid it. It underlines the motivation of this research.
2. SCHEDULER STABILITY

A resource is passive if the duration for the access of the resource requires the consumption of the processor power, such as semaphores, mutex locks. A resource is active if it is not passive. In this paper, we are interested in non-preemptible active resources, such as disks, with synchronous requests. Let an access request be serviced immediately on the corresponding active resource once it is granted and available.

We denote the the j-th invocation of task τi as Ji,j (referred to as a job), where a task could be periodic or sporadic. Each job Ji,j is a sequence of sub-jobs, where a sub-job is ready when its preceding sub-job completes its execution. Ji,k denotes the k-th sub-job of Ji,j. Let C denote a system configuration. A system configuration C′ is greater than or equal to C (denoted as C′ ≥ C) if and only if the resource access duration of active/passive resources.

Given a set T of jobs of a n-task set \{τ1, τ2, ..., τn\} between a given interval P, let Π be a given real-time scheduler, and S = ΠC(T) be the schedule resulted from the scheduling of jobs in T by the scheduler Π based on a given system configuration C. Note that jobs in T have their arrival times fixed for a given T. Let \(\theta^c_{i}(J_{i,j,k})\) denote the completion time of Ji,j,k under a scheduler Π based on a system configuration C. We say that a scheduler is stable if and only if \(\forall i,j,k, \theta^c_{i}(J_{i,j,k}) \geq \theta^c_{i}(J_{i,j,k})\) for any task set T and any two system configurations C and C′ when C′ ≥ C. A real-time scheduler is unstable if it is not stable. An unstable scheduler may or may not result in an anomaly for any given system configurations C and C′ and a given task set T. An anomaly occurs for a real-time scheduler Π and any two system configurations C and C′ and a given task set T if and only if \(\exists i,j,k, \theta^c_{i}(J_{i,j,k}) > \theta^c_{i}(J_{i,j,k})\) when C′ ≥ C.

**Theorem 1.** No greedy real-time scheduler is stable with the presence of non-preemptive resources.

**Theorem 2.** Given a system configuration C, a real-time scheduler Π is not stable with respect to C only if there exists a job in some given task set T that experiences a preemption or blocking in \(S = \Pi_{C'}(T)\) but not in \(S = \Pi_{C'}(T)\), where C′ is another system configuration, and C′ ≥ C.

3. ANOMALY PREVENTION

This section presents three guidelines for real-time scheduling algorithms to prevent anomalies from happening. The first two are focused on the handling of passive resources only, and the last one extends the idea in anomaly prevention to the handling of active resources and a more flexible way in the bounding of the anomaly duration.

**IDI rule:** Given the arrival times of tasks and the duration time of each resource access, an existing scheduler, under only passive resource consideration, could be revised and be stable based on the following rule: Whenever a task τi requests a passive resource R at time t and might access R for D time units, the request is pending if \((t + D)\) is larger than the arrival time of some higher-priority task τj. We call the rule as the idle-time-insertion (IDI) rule. Note that when only a fixed set of periodic tasks are considered, the arrival time of each task could be derived based on its period and initial phase.

**OP rule:** OP rule is to maintain the resource access order of resources in S = ΠC(T) when another system configuration C′ is given. Given a schedule S = ΠC(T) for a system configuration C (either generated in an on-line or off-line fashion), we can order all of the resource requests in S in terms of their granted time, and the order is a sequential order. We define the order-preservation (OP) rule as follows: The granting of resource requests from tasks executing over another system configuration C′ must be consistent with their corresponding granting order in S. We must point out that the OP rule serves as an example idea in this paper. There always exist some smarter schedulers that do not need to preserve the granting order of resource requests for a given system configuration. However, such an approach would involve certain resource reclaiming mechanisms, such as those in [5], and the scheduling problem could become intractable easily.

**AC rule:** For utilization improving, we suppose that active and passive resources are considered separated in task synchronization. The maximum number of priority inversions for each task under PCP [4] becomes one plus the number of accesses to active resources from the task. Such observations underline the motivation of the following rule in anomaly management, referred to as the anomaly control (AC) rule: Let each table entry AC[i] denote the number of blocking tolerable to a task τi. (1) Whenever a job of τi completes its execution, AC[i] is reset to the initial setting. (2) Whenever a blocking occurs to τi, AC[i] is decremented by one. (3) Any request to a resource from τi is blocked if any AC[j] of some higher-priority task is no more than 0. (4) Active resources should be managed independent of that for passive resources. With a setting of AC[i], the maximum number of priority inversion for a task (maximum duration of anomaly) is thus bounded, given the maximum access duration of active/passive resources.

4. CONCLUSION

As more and more softwares with timing constraints are ported among different platforms, the considerations of the timing behaviors of softwares become a critical issue. This research addresses the important scheduling anomaly issues that are motivated by the practical needs in embedded-system implementations. A series of experiments was conducted to evaluate the capability of the proposed protocols, for which we have very encouraging results. As the variety of embedded systems products has made software portability an important issue, more research in this direction would be proved very rewarding.

5. REFERENCES


