

Research description

Modern computers consist of several processing units, connected via one or more interconnects to a memory hierarchy, auxiliary processors and other devices. The traditional approach to sharing such platforms between several applications treats the whole machine as a whole and allows only a single task to execute at the same time. With the advent of multiprocessor platforms new scheduling algorithms have been devised aiming at exploiting some of the available concurrency, however, they usually abstract from the intricate dependencies involved in sharing common interconnects or memories. Many hardware platforms, especially in the embedded systems domain, provide mechanisms to manage the hardware resources individually. These mechanisms provide an opportunity for fine-grained scheduling and more efficient utilization of the available concurrency. Exploiting this concurrency while meeting real-time constraints, however, requires new scheduling techniques, accompanied by new analysis.

This thesis addresses the problem of multi-resource management in embedded real-time systems. It focuses on three research questions, which were derived from a given multimedia processing platform. The first question deals with how to map a set of tasks onto a platform, where tasks require access to several resources concurrently, e.g. processor, bus, network. The second question concentrates on how to efficiently implement such a mapping, in particular how to provide co-existing applications with guaranteed access to resources during runtime. The third question investigates how to change a given mapping during runtime.

A novel multi-resource scheduling algorithm is presented, where tasks can specify requirements for simultaneous access to several resources, akin to gang scheduling on multiprocessors. In the proposed task model, each task is represented by a Directed Acyclic Graph, where nodes represent decision points and each edge represents a set of subtasks. A subtask represents a requirement for a particular resource. Primitives are introduced which allow a task to acquire mutually exclusive access to several resources at the same time. To provide real-time guarantees the proposed synchronization primitives are implemented using an extension of the Stack Resource Policy (SRP). The extension inherits several properties from the original SRP, such as no chained-blocking, no deadlock and a single stack for tasks sharing a common resource. Furthermore, the implementation is straightforward, as shown by example in the $\mu\text{C}/\text{OS-II}$ real-time operating system.

For providing guaranteed access to resources we chose a reservation based approach. We discussed both memory and processor reservations, where timed events play an important role. Common examples are a deadline, periodic release of a task, budget replenishment and depletion. Efficient timer management is therefore essential. We investigated the overheads in traditional timer management techniques and provided a novel mechanism called Relative Timed Event Queues (RELTEQ), which provides an expressive set of primitives while at the same time reducing the performance overhead. At its core, RELTEQ manipulates event queues in which the time of each event is stored relative to the previous event in the queue. A method was presented showing how a combination of several such queues can be used to efficiently implement global and virtual timed events. RELTEQ allows to express unbounded event interarrival time between any two events in a queue, and provides an expressive set of primitives to implement higher level scheduling mechanisms. These were exploited to implement processor reservations based on several types of servers, such as a polling, periodic-idling, and deferrable server.

Finally, we investigated adapting the resource provisions to tasks during runtime, referred to as mode changes. The presented method is based on combining scalable components with fixed-priority with deferred preemption scheduling (FPDS). A scalable component can operate in one of several modes, where each mode defines a trade off between resource requirements and output quality. In this thesis we focused on memory constrained systems, with modes limited to memory requirements. During runtime the system may reallocate the resources between the components, resulting in a mode change. The latency of a mode change should satisfy an upper bound. A modeling framework was presented combining scalable components with FPDS. A quantitative analysis comparing fixed-priority preemptive scheduling with FPDS shows that our protocol based on FPDS improves the existing bounds on mode change latency.