Lecture 15: Concurrency

Concurrency – operating system management and execution of task synchronization, communication, and resource sharing

- The primary goal of a real-time system is predictability. Achieving this goal requires all levels of the system to work in concert to provide fixed worst-case execution times.
- Predictability cannot be achieved solely through task scheduling. For concurrency, and hence predictability to be achieved, many internal components must know how to react differently.
- Scheduling is generally considered the most important aspect of a real-time system. The goal of scheduling is to determine whether a set of tasks can meet their specified timing requirements and to provide an ordering of tasks that satisfies the specified constraints.
- Concurrency is required due to parallel process execution needed to achieve a certain system performance. The system may be distributed, complex, or redundant for reliability reasons.
- Communication in concurrent programming must be considered (shared data, task synchronization).
- A simple lock construct (e.g. a semaphore) is necessary and sufficient to achieve concurrent programming. But the low-level lock must be made part of the programming language at a high-level.
- Most theoretical work on real-time scheduling deals with mutual exclusion protection with semaphores. For task synchronization, theoretical work does not exist.
- Without understanding and compensating for scheduling conflicts, starvation and deadlock can occur.
- The most important criteria for a real-time system is that a task meets its deadline. Priority inversion is the most likely source of missed deadlines (when one higher priority task is blocked by a lower priority task using a resource it needs).
- Dynamic priority scheduling can create a queue maintenance and run-time overhead problem for tasks. Also blocked tasks can suddenly have their priority level changed at any time. Hence unpredictable behavior may result in a cascade of missed deadlines.
- Fundamental real-time scheduling requires a specified knowledge of all task timing constraints. These constraints provide the ability to create predictable and schedulable systems.
- Unbounded operations are not permitted in real-time systems. Iterations and recursions must be bounded in some way; most likely a timeout mechanism for aborting the procedure is needed. This becomes an issue with task synchronization and communication.
- Timeout mechanisms can cause deadlock and system overhead.
- In many cases, an assigned task priority level has little meaning to the actual importance of the task; the priority has everything to do with the relative timing characteristics of the task.
- A scheduler must know if hard deadlines exist or soft ones (if the consequences of missing a task deadline are severe to the system's performance or not disastrous).
- Scheduling decisions are based on a variety of constraints: task period, worst-case execution time, deadline, required start-time, and finally task importance.
- Priority queue synchronization problems can occur as a result of new tasks being introduced into the system or tasks exiting the system. Task priority shifting can occur and required priority spacing between tasks must be allocated. Also priority queue data structures must be updated in a synchronized way.
- Many priority-based scheduling algorithms assume that tasks are independent whereas they are not due to shared resource allocation required in concurrent systems. Critical sections must be protected by mutual exclusion potentially causing priority inversion.
- A technique used to handle priority inversion is called priority inheritance, where a task priority is temporarily raised to the level of the resource it owns – usually assigned the highest priority task that is actively requesting the resource. However, this technique still does not prevent deadlock in multi-shared resource systems.
- To avoid deadlock with priority inheritance protocol, a task scheduler may use priority ceiling protocol where the task that has highest priority to run is the task that owns the highest priority resource.
- Priority disinheritance is used to shift a task's priority back to its previous level due to the release of a shared resource, or the priority level of the highest priority resource it still owns.