Real Time systems

Lecture 2

LIMITS OF CURRENT REAL-TIME SYSTEMS

Most of the real-time computing systems used to support control applications are based on kernels, which are modified versions of timesharing operating systems. As a consequence, they have the same basic features found in timesharing systems, which are not suited to support real-time activities. The main characteristics of such real-time systems include the following:

Multitasking. A support for concurrent programming is provided through a set of system calls for process management (such as *create*, *activate*, *terminate*, *delay*, *suspend*, and *resume*). Many of these primitives do not take time into account and, even worse, introduce unbounded delays on tasks' execution time that may cause hard tasks to miss their deadlines in an unpredictable way.

Priority-based scheduling. This scheduling mechanism is quite flexible, since it allows the implementation of several strategies for process management just by changing the rule for assigning priorities to tasks. Nevertheless, when application tasks have explicit time requirements, mapping timing constraints into a set of priorities may not be simple, especially in dynamic environments. The major problem comes from the fact that these kernels have a limited number of priority levels (typically 128 or 256), whereas task deadlines can vary in a much wider range. Moreover, in dynamic environments, the arrival of a new task may require the remapping of the entire set of priorities.

Ability to quickly respond to external interrupts. This feature is usually obtained by setting interrupt priorities higher than process priorities and by reducing the portions of code executed with interrupts disabled. Note that, although this approach increases the reactivity of the system to external events, it introduces unbounded delays on processes' execution. In fact, an application process will be always interrupted by a driver, even though it is more important than the device that is going to be served. Moreover, in the general case, the number of interrupts that a process can experience during its execution cannot be bounded in advance, since it depends on the particular environmental conditions.

Basic mechanisms for process communication and synchronization. Binary semaphores are typically used to synchronize tasks and achieve

mutual exclusion on shared resources. However, if no access protocols are used to enter critical sections, classical semaphores can cause a number of undesirable phenomena, such as priority inversion, chained blocking, and deadlock, which again introduce unbounded delays on real-time activities.

Small kernel and fast context switch. This feature reduces system overhead, thus improving the average response time of the task set. However, a small average response time on the task set does not provide any guarantee on the individual deadlines of the tasks. On the other hand, a small kernel implies limited functionality, which affects the predictability of the system.

Support of a real-time clock as an internal time reference. This is an essential feature for any real-time kernel that handles time-critical activities that interact with the environment. Nevertheless, in most commercial kernels this is the only mechanism for time management. In many cases, there are no primitives for explicitly specifying timing constraints (such as deadlines) on tasks, and there is no mechanism for automatic activation of periodic tasks.

1.2.3 DESIRABLE FEATURES OF REAL-TIME SYSTEMS

there are some very important basic properties that real-time systems must have to support critical applications. They include the following:

Timeliness. Results have to be correct not only in their value but also in the time domain. As a consequence, the operating system must provide specific kernel mechanisms for time management and for handling tasks with explicit timing constraints and different criticality.

Predictability. To achieve a desired level of performance, the system must be analyzable to predict the consequences of any scheduling decision. In safety critical applications, all timing requirements should be guaranteed off line, before putting system in operation. If some task cannot be guaranteed within its time constraints, the system must notify this fact in advance, so that alternative actions can be planned to handle the exception.

Efficiency. Most of real-time systems are embedded into small devices with severe constraints in terms of space, weight, energy, memory, and computational power. In these systems, an efficient management of the available resources by the operating system is essential for achieving a desired performance

Robustness. Real-time systems must not collapse when they are subject to peak load conditions, so they must be designed to manage all anticipated load scenarios.
Overload management and adaptation behavior are essential features to handle systems with variable resource needs and high load variations.

Fault tolerance. Single hardware and software failures should not cause the system to crash. Therefore, critical components of the real-time system have to be designed to be fault tolerant.

Maintainability. The architecture of a real-time system should be designed according to a modular structure to ensure that possible system modifications are easy to perform .