Real Time Systems1

ACHIEVING PREDICTABILITY

System Calls
Semaphore
Memory Management
Programming languages

Lecture 5

Achieving Predictability/ System Calls

- System predictability also depends on how the kernel primitives are implemented.
- In order to precisely evaluate the worst-case execution time of each task, all kernel calls should be characterized by a bounded execution time, used by the guaranteed mechanism while performing the schedulability analysis of the application.

Achieving Predictability/ System Calls

- In addition, in order to simplify this analysis, it is desirable that each kernel primitive be preemptable.

- In fact, any non-preemptable section could possibly delay the activation or the execution of critical activities, causing a timing fault to hard deadlines.

Achieving Predictability/ Semaphore

- The typical semaphore mechanism used in traditional operating systems is not suited for implementing real-time applications because it is subject to the priority inversion phenomenon, which occurs when a high-priority task is blocked by a low-priority task for an unbounded interval of time.

Achieving Predictability/ Semaphores

- Priority inversion must absolutely be avoided in real-time systems, since it introduces nondeterministic delays on the execution of critical tasks.

Achieving Predictability/ Memory Management

- Similarly to other kernel mechanisms, memory management techniques must not introduce nondeterministic delays during the execution of real-time activities.
- Typical solutions adopted in most real-time systems adhere to a memory segmentation rule with a fixed memory management scheme.

Achieving Predictability/ Programming Language

- Unfortunately, current programming languages are not expressive enough to prescribe certain timing behavior and hence are not suited for realizing predictable real-time applications

Achieving Predictability/ Programming Language

- New high-level languages have been proposed to support the development of hard real-time applications.
- For example, *Real-Time Euclid* Real-Time Euclid forces the programmer to specify time bounds and timeout exceptions in all loops, waits, and device accessing statements.

Achieving Predictability/ Programming Language

- Programming restrictions:
- **Absence of dynamic data structures.**
- **Absence of recursion.**
- **Time-bounded loops.**

Achieving Predictability/ Programming Language

- **Absence of dynamic data structures**. Third-generation languages normally permit the use of dynamic arrays, pointers, and arbitrarily long strings.

- In real-time languages, however, these features must be eliminated because they would prevent a correct evaluation of the time required to allocate and deallocate dynamic structures

Achieving Predictability/ Programming Language

- **Absence of recursion**. If recursive calls were permitted, the schedulability analyzer could not determine the execution time of subprograms involving recursion or how much storage will be required during execution

Achieving Predictability/ Programming Language

- **Time-bounded loops.** In order to estimate the duration of the cycles at compile time, Real-Time Euclid forces the programmer to specify for each loop construct the maximum number of iterations.

- Real-Time Euclid also allows the classification of processes as periodic or aperiodic and provides statements for specifying task timing constraints, such as activation time and period, as well as system timing parameters, such as the time resolution.