Real Time systems

Lecture 3

ACHIEVING PREDICTABILITY

first component that affects the predictability of the scheduling is the processor itself. The internal characteristics of the processor, such as instruction prefetch, pipelining, cache memory, and direct memory access (DMA) mechanisms, are the first cause of non-determinism. In fact, although these features improve the average performance of the processor, they introduce non-deterministic factors that prevent a precise estimation of the worst-case execution times (WCETs). **Other important components** that influence the execution of the task set are the internal characteristics of the real-time kernel, such as the scheduling algorithm, the synchronization mechanism, the types of semaphores, the memory management policy, the communication semantics, and the interrupt handling mechanism.

1.3.1 DMA

Direct memory access (DMA) is a technique used by many peripheral devices to transfer data between the device and the main memory. The purpose of DMA is to relieve the central processing unit (CPU) of the task of controlling the input/output (I/O) transfer. Since both the CPU and the I/O device share the same bus, the CPU has to be blocked when the DMA device is performing a data transfer. Several different transfer methods exist.

One of the most common methods is called *cycle stealing*, according to which the DMA device steals a CPU memory cycle in order to execute a data transfer. During the DMA operation, the I/O transfer and the CPU program execution run in parallel. However, if the CPU and the DMA device require a memory cycle at the same time, the bus is assigned to the DMA device and the CPU waits until the DMA cycle is completed. Using the cycle stealing method, there is no way of predicting how many times the CPU will have to wait for DMA during the execution of a task; hence the response time of a task cannot be precisely determined.

A possible solution to this problem is to adopt a different technique, which requires the DMA device to use the memory *time-slice method*. According to this method, each memory cycle is split into two adjacent time slots: one reserved for the CPU and the other for the DMA device. This solution is more expensive than cycle stealing but more predictable. In fact, since the

CPU andDMA device do not conflict, the response time of the tasks do not increase due to DMA operations and hence can be predicted with higher accuracy.

1.3.2 CACHE

The cache is a fast memory that is inserted as a buffer between the CPU and the random access memory (RAM) to speed up processes' execution. In real-time systems, the cache introduces some degree of non-determinism.
In preemptive systems, the cache behavior is also affected by the number of preemptions. In fact, preemption destroys program locality and heavily increases the number of cache misses due to the lines evicted by the preempting task. Moreover, the cache related preemption delay (CRPD) depends on the specific point at which preemption takes place; therefore it is very difficult to precisely estimate.

1.3.3 INTERRUPTS

Interrupts generated by I/O peripheral devices represent a big problem for the predictability of a real-time system because, if not properly handled, they can introduce unbounded delays during process execution.
In order to reduce the interference of the drivers on the application tasks and still perform I/O operations with the external world, the peripheral devices must be handled in a different way. In the following, three possible techniques are illustrated.

APPROACH A

The most radical solution to eliminate interrupt interference is to disable all external interrupts, except the one from the timer (necessary for basic system operations). In this case, all peripheral devices must be handled by the application tasks, which have direct access to the registers of the interfacing boards. Since no interrupt is generated, data transfer takes place through polling.
The main disadvantage of this solution is a low processor efficiency on I/O operations, due to the busy wait of the tasks while accessing the device registers. Another minor problem is that the application tasks must have the knowledge of all low-level details of the devices that they want to handle. However, this can be easily solved by encap sulating all device-dependent routines in a set of library functions that can be called
by the application tasks.

APPROACH B

As in the previous approach, all interrupts from external devices are disabled, except the one from the timer. Unlike the previous solution, however, the devices are not directly handled by the application tasks but are managed in turn by dedicated kernel routines, periodically activated by the timer.

This approach eliminates the unbounded delays due to the execution of interrupt drivers and confines all I/O operations to one or more periodic kernel tasks Because the interrupts are disabled, the major problem of this approach is due to the busy wait of the kernel I/O handling routines, which makes the system less efficient during the I/O operations. With respect to the previous approach, this case is characterized by a higher system overhead, due to the communication required among the application tasks and the I/O kernel routines for exchanging I/O data. Finally, since the device handling routines are part of the kernel, it has to be modified when some device is replaced or added.

APPROACH C

A third approach that can be adopted in real-time systems to deal with the I/O devices is to leave all external interrupts enabled, while reducing the drivers to the least possible size. According to this method, the only purpose of each driver is to activate a proper task that will take care of the device management. Once activated, the device manager task executes under the direct control of the operating system, and it is guaranteed and scheduled just like any other application task. In this way, the priority that can be assigned to the device handling task is completely independent from other priorities and can be set according to the application requirements. Thus, a control task can have a higher priority than a device handling task.

The major advantage of this approach with respect to the previous ones is to eliminate the busy wait during I/O operations. Moreover, compared to the traditional technique, the unbounded delays introduced by the drivers during tasks' execution are also drastically reduced (although not completely removed), so the task execution times become more predictable. As a matter of fact, a little unbounded overhead due to the execution of the small drivers still remains in the system, and it should be taken into account in the guarantee mechanism.