Scheduling predictability

Date	@October 30, 2024

Predictable hardware support

Predictability refers to the ability of the OS to guarantee that all critical timing constraints are met and it requires determinism.

A deterministic algorithm is an algorithm which, given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states.

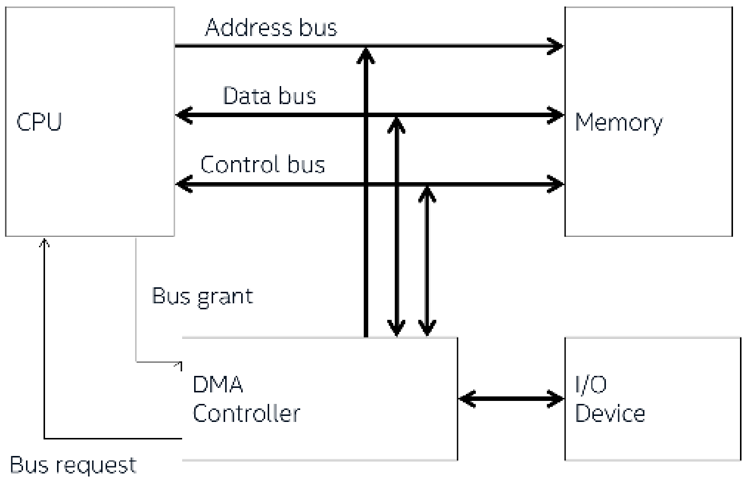
Determinism is influenced by:

- architectural features of the hardware
- scheduling algorithm
- IPC mechanism
- others

HW/SW sources of nondeterminism

Direct memory access (DMA)

DMA is used by may peripheral devices to transfer data between the device and the main memory. It relieves the CPU of the task of controlling the the I/O transfer. The CPU and the I/O device share the same bus and the CPU is blocked when the DMA device is performing data transfer.



There are different data transfer mode:

- burst: an entire block of data is transferred in one contiguous sequence and the CPU remains inactive for relatively long period of time (until the whole transfer is completed)
- cycle stealing: DMA transfers byte of data and then release the bus returning control to the CPU; it continually issues requests, transferring one byte of data per request, until it has transferred the entire block of data and it takes longer to transfer data, meaning the CPU is blocked for less time
- transparent: DMA transfers data when the CPU is performing operations that do not use the system buses

DMA makes difficult to predict task response time. The time the task remains blocked depends on:

- the size of the block of data
- the transfer mode

In real-time oriented transfer mode is used the time sliced method: each memory cycle is split into two adjacent time slots, one reserved for the CPU and the other for the DMA device; it is more time expensive but more predictable and CPU and DMA never interfere with each other.

Cache memory

It is fast memory inserted as a buffer between the CPU and the RAM to speed up task execution. At each memory access the hardware checks whether the requested information is stored in the cache:

- cache hit: data are read from the cache (very fast)
- cache miss: data are read from the RAM and is copied in the cache along with a set of adjacent locations (very slow)

Cache memory exploit the program locality concept.

In cache memory, hit ratio is a source of nondeterminism , which improves the performance on average and is affected by the number of preemption in preemptive systems.

Solutions are:

- disable the cache memory, which impacts the performance but leads to more deterministic behavior
- overestimate the WCET, which does not impact the performance but uses only a fraction of the available CPU time

Interrupts

Interrupts are event generated by I/O devices to get attention of the CPU when a data is available. The arrival of an interrupt causes the execution of an interrupt service routine, which is the fragment of code dedicated to the management of its associated I/O device.

For example, to get data from an I/O device, each task must:

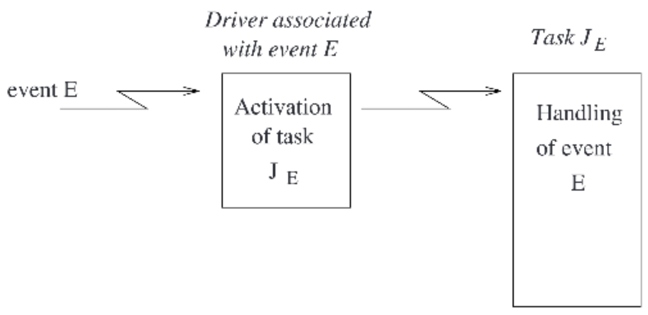
- enable the hardware to generate interrupts
- wait for the interrupt
- read the data from a memory buffer shared with the driver

In general purpose OS interrupts are served using a fixed priority scheme and each driver is scheduled based on a static priority, higher than process priority.

In real-time OS tasks could be more urgent than an interrupt service routine and it is very difficult to bound a priori the number of interrupts that a task may experience. The delay introduced by interrupts on tasks execution is unpredictable.

There are many solutions:

- solution 1: disable all interrupts but timer interrupt, which are needed for scheduling tasks, and handle I/O devices using polling, so that the application task is responsible for checking I/O devices periodically for available data; this solution is deterministic but requires busy waiting for communicating with I/O device.
- solution 2: disable all interrupts but timer interrupt and treat the I/o devices as aperiodic tasks managed through servers, so that slow devices are multiplexed and served by a single server running at a low rate while fast devices are served by dedicated servers running at higher frequencies; in this solution servers are implemented by the OS but still requires busy waiting in the server; servers are scheduled accordingly to their priority, independently from those of application tasks
- solution 3: all interrupts are enabled and the interrupt handling is performed using a driver, whose only purpose is to activate the task associated to the I/O device, and a task, which is responsible for handling the I/O device; in this solution the driver execution time is very limited (so it has a negligible impact on the tasks WCET) and the I/O task is scheduled by the OS based on its priority, independent from those of application tasks; in this solution there is no busy waiting



System calls

The execution time of system calls must be characterized and considered when evaluating if the scheduling is feasible. System calls must be pre-emptible because any non-pre-emptible section could delay the activation or the execution of critical tasks.

Sempahores

Traditional semaphores are not suitable for real-time systems and this may produce the priority inversion problem. The OS must implement into semaphores the solutions to priority inversion.

Memory management

Memory management techniques must not introduce nondeterministic delays during the execution of real-time activities..

Programming language

Restrictions that may be needed for real-time are:

- no dynamic data structures: dynamic arrays must be eliminated because they would prevent a correct evaluation of the time required to allocate and deallocate memory
- no 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
- time-bounded loops: in order to estimate the duration of the cycles at compile time, programmer should specify for each loop construct the maximum number of iterations