The Non-Preemptable
Section Latency

Real Time Operating Systems and Middleware

Luca Abeni
luca.abeni@unitn.it
Latency

- Latency: measure of the difference between the theoretical and actual schedule
  - Task $\tau$ expects to be scheduled at time $t \ldots$
  - $\ldots$ but is actually scheduled at time $t'$
  - $\Rightarrow$ Latency $L = t' - t$

- The latency $L$ can be modelled as a blocking time
  $\Rightarrow$ affects the guarantee test
  - Similar to what done for shared resources
  - Blocking time due to latency, not to priority inversion
Effects of the Latency

- Upper bound for $L$? If not known, no schedulability tests!!!
  
  - The latency must be \textit{bounded}: \( \exists L_{\text{max}} : L < L_{\text{max}} \)

- If $L_{\text{max}}$ is too high, only few task sets result to be schedulable
  
  - Large blocking time \textit{experienced by all tasks}!
  
  - The worst-case latency $L_{\text{max}}$ cannot be too high
Sources of Latency

- A task $\tau_i$ is a stream of jobs $J_{i,j}$ arriving at time $r_{i,j}$
- Job $J_{i,j}$ is scheduled at time $t' > r_{i,j}$

$t' - r_{i,j}$ is given by:

1. $J_{i,j}$'s arrival is signalled at time $r_{i,j} + L^1$
2. Such event is served at time $r_{i,j} + L^1 + L^2$
3. $J_{i,j}$ is actually scheduled at $r_{i,j} + L^1 + L^2 + L^3$
Analysis of the Various Sources

\[ L = L^1 + L^2 + L^3 \]

- \( L^3 \) is the \textit{scheduler latency}
  - Interference from higher priority tasks
  - Already accounted by the guarantee tests → let’s not consider it

- \( L^2 \) is the \textit{non-preemptable section latency} \( (L^{np}) \)

- \( L^1 \) is due to the delayed interrupt generation
Non-Preemptable Section Latency

- Delay between time when an event is generated and when the kernel handles it
  - Due to non-preemptable sections in the kernel, which delay the response to hardware interrupts
  - Composed by various parts: interrupt disabling, bottom halves delaying, . . .

- Depends on how the kernel handles the various events...

- Will talk about it later!
Interrupt Generation Latency

- Hardware interrupts: generated by devices
- Sometimes, an interrupt should be generated at time \( t \) . . .
- . . . but it is actually generated at time \( t' = t + L^{int} \)
- \( L^{int} \) is the Interrupt Generation Latency
  - It is due to hardware issues
  - It is generally small compared to \( L^{np} \)
  - Exception: if the device is a timer device, the interrupt generation latency can be quite high
    - Timer Resolution Latency \( L^{timer} \)
The Timer Resolution Latency

- Interrupt generation latency for a hw timer device
- $L_{timer}$ can often be much larger than the non-preemptable section latency $L_{np}$

Where does it come from?

- Kernel timers are generally implemented by using a hardware device that produces periodic interrupts

Can we do anything about it?
Non-Preemptable Section Latency

- The *non-preemptable section latency* $L_{np}$ is given by the sum of different components
  1. Interrupt disabling
  2. Delayed interrupt service
  3. Delayed scheduler invocation

- The first two are mechanisms used by the kernel to guarantee the consistency of internal structures

- The third mechanism is sometimes used to reduce the number of preemptions and increase the system throughput
● Remember? Before checking if an interrupt fired, the CPU checks if interrupts are enabled...

![Flowchart diagram]

● Every CPU has some *protected* instructions ($\text{STI/CLI}$ on x86) for enabling/disabling interrupts
Interrupts and Latency

- In modern system, only the kernel (or code running in KS) can enable/disable interrupts
- Interrupts disabled for a time $T_{cli} \rightarrow L_{np} \geq T_{cli}$
- Interrupt disabling is used to enforce mutual exclusion between sections of the kernel and ISRs
When the interrupt fire, the ISR is ran, but the kernel can delay interrupt service some more...

- ISRs are generally small, and do only few things
- An ISR can set some kind of *software flag*, to notify that the interrupt fired
- Later, the kernel can check such flag and run a larger (and more complex) interrupt handler

- Hard IRQ handlers (ISRs) vs “Soft IRQ handlers”
Advantages of “soft IRQ handlers”:

- ISRs generally run with interrupts disabled,
- Soft IRQ handlers can re-enable hardware interrupts
- Enabling/Disabling soft handlers is simpler/cheaper

Disadvantages:

- Increase NP latency: $L_{np} \gg T_{cli}$
- “Soft IRQ handlers” are often non-preemptable increasing the latency for other tasks too...
Deferred Scheduling

- Scheduler invoked when returning from KS to US
- Sometimes, return to US after a lot of activities
  - Try to reduce the number of KS ↔ US switches
  - Reduce the number of context switches
  - Throughput vs low latency
- ISR executed at the correct time, soft IRQ handler ran immediately, but scheduler invoked too late

![Diagram showing event time, latency, and scheduler invocation]
Latency in the Standard Kernel

Real-Time Operating Systems and Middleware

Non-Preemptable Section Latency
$L^{np}$ depends on some different factors

In general, no hw reasons $\rightarrow$ it almost entirely depends on the *kernel structure*

Non-preemptable section latency is generally the result of the strategy used by the kernel for ensuring mutual exclusion on its internal data structures
To analyze / reduce $L^{np}$, we need to understand such strategies.

Different kernels, based on different structures, work in different ways.

Some activities causing $L^{np}$:

- Interrupt Handling (Device Drivers)
- Management of the parallelism
Example: Data Structures Consistency

- HW interrupt: *breaks* the regular execution flow
  - If the CPU is executing in US, switch to KS

- If execution is already in KS, possible problems:
  1. The kernel is updating a linked list
  2. IRQ While the list is in an inconsistent state
  3. Jump to the ISR, that needs to access the list...

- Must *disable interrupts* while updating the list!

- Similar interrupt disabling is also used in spinlocks and mutex implementations...