The Non-Preemptable Sections Latency

Luca Abeni
luca.abeni@unitn.it

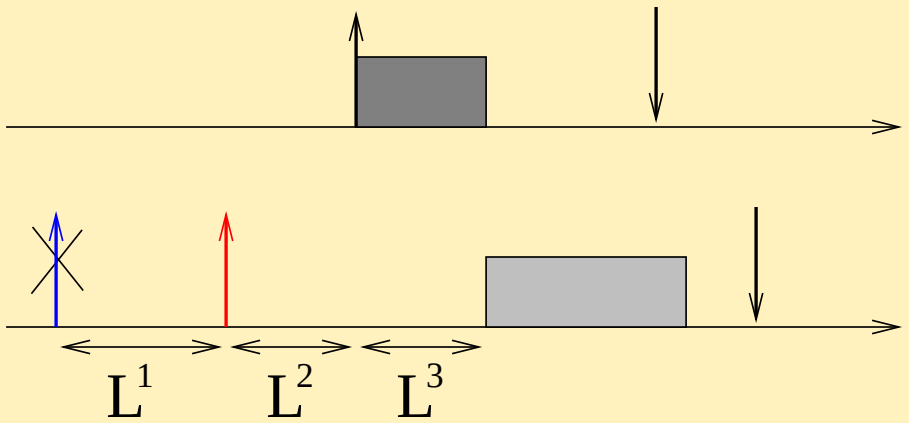
December 1, 2014

Latency

- Latency: measure of the difference between the theoretical and actual schedule
  - Task τ expects to be scheduled at time t ...
  - ... but is scheduled at time t'
  - $\Rightarrow$ Latency $L = t' - t$
- The latency L can be modelled as a blocking time $\Rightarrow$ affects the guarantee test
- If L is too high, only few task sets result to be schedulable
  - The latency must be *bounded*: $\exists L^{max} : L < L^{max}$
  - The latency bound L^{max} cannot be too high

Sources of Latency

- A task τ_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 the sum of various components:
    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 *scheduler latency*
  - Interference from higher priority tasks
  - Already accounted by the guarantee tests $\rightarrow$ let's not consider it
- L^2 is the *non-preemptable section latency*, called L^{np}
  - Due to non-preemptable sections in the kernel, which delays the response to hardware interrupts
  - It is composed by various parts: *interrupt disabling*, *bottom halves delaying*, ...
- L^1 is due to the delayed interrupt generation

Interrupt Generation Latency

- Hardware interrupts are generated by external devices
- Sometimes, a device must generate an interrupt at time t ...
- ... but actually generates it 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 L^{timer} can often be much larger than the non-preemptable section latency L^{np}

The Timer Resolution Latency

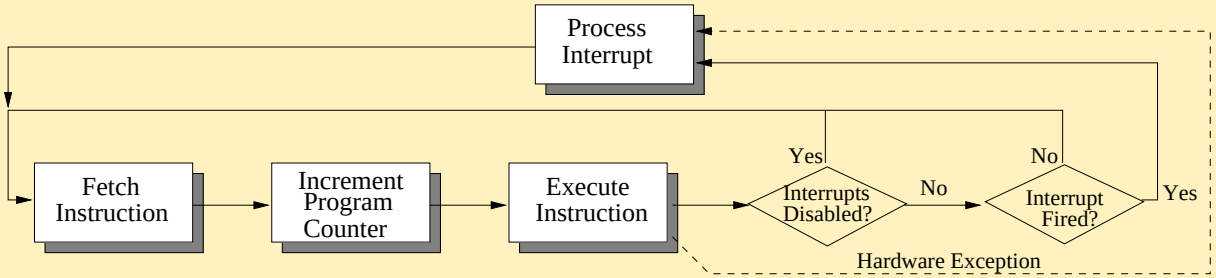
- Kernel timers are generally implemented by using a hardware device that produces periodic interrupts
- Periodic timer interrupt $\rightarrow$ tick
- Example: periodic task (`setitimer()`, Posix timers, `clock_nanosleep()`, ...) τ_i with period T_i
- At the end of each job, τ_i sleeps for the next activation
- Activations are triggered by the periodic interrupt
  - Periodic tick interrupt, with period T^{tick}
  - Every T^{tick}, the kernel checks if the task must be woken up
  - If T_i is not multiple of T^{tick}, τ_i experiences a timer resolution latency

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

Disabling Interrupts

- Remember? Before checking if an interrupt fired, the CPU checks if interrupts are enabled...



- Every CPU has some *protected* instructions (`STI`/`CLI` on x86) for enabling/disabling interrupts
  - 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

Delayed Interrupt Service - 1

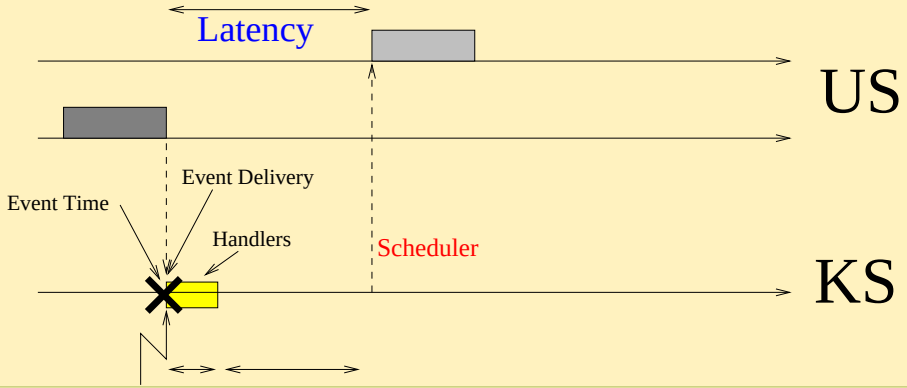
- When the interrupt fire, the ISR is executed, but the kernel can delay interrupt service some more...
  - ISRs are generally small, and do only few things
  - An ISR can set some kind of *software flag*, to notify that the interrupt fired
  - Later, the kernel can check such flag and run a larger (and more complex) interrupt handler
- Some sort of "software interrupts"...
- Advantages:
  - ISRs generally run with interrupts disabled
  - But software interrupt handlers can re-enable hardware interrupts
  - Enabling/Disabling software interrupt handlers is simpler / cheaper

Delayed Interrupt Service - 2

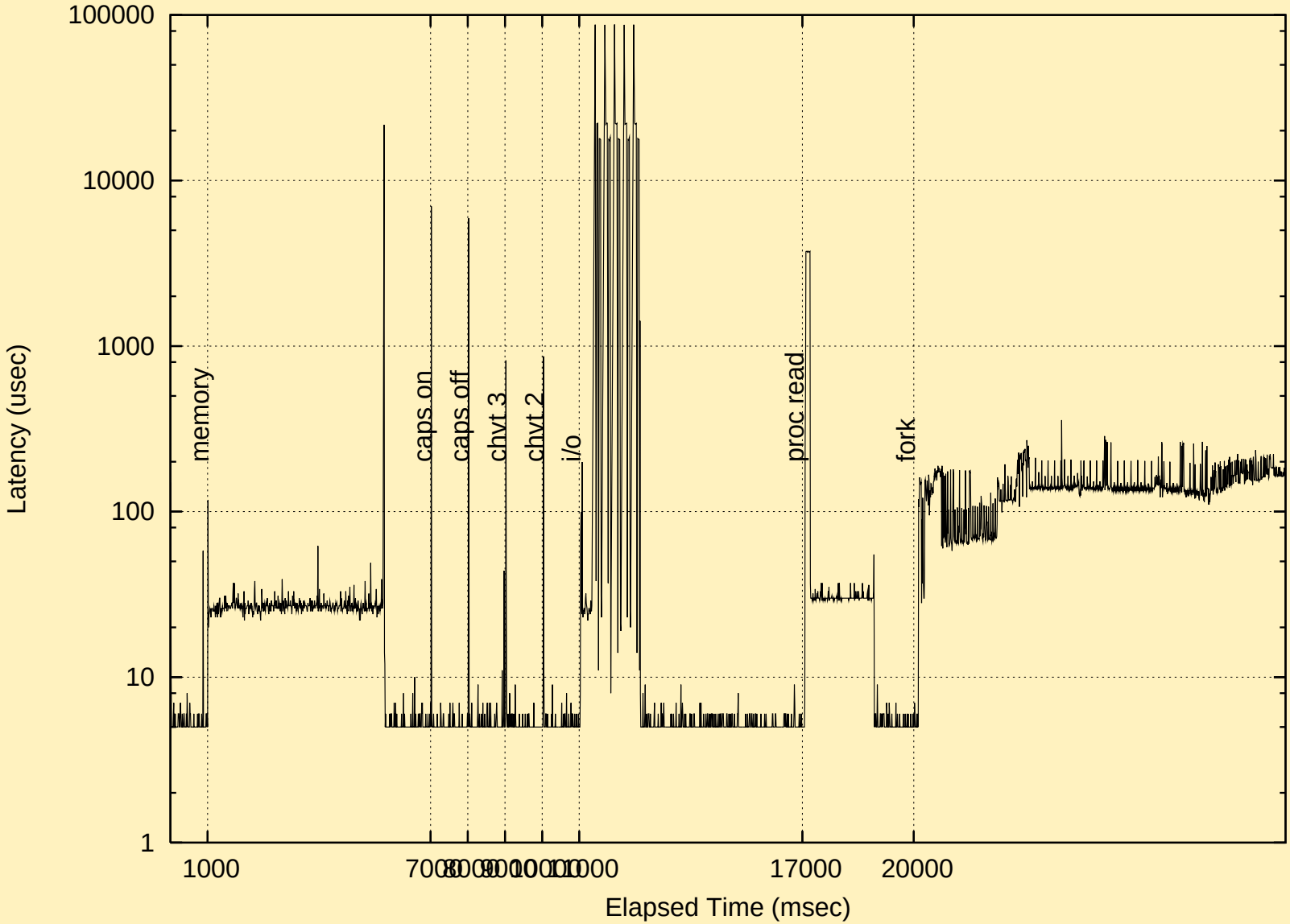
- Software interrupt handlers are good for throughput...
- ...But can be bad for real-time / latency:
  - Interrupt response latency is increased: $L^{np} >> T^{cli}$
  - Software interrupt handlers are often non-preemptable increasing the latency for other tasks too...
- Large, non-schedulable, active entities executing inside the kernel...

Deferred Scheduling

- Scheduler: invoked only when returning from KS to US
- For efficiency reasons, the kernel might want to return to user tasks only after performing a lot of activities
  - Try to reduce the number of KS $\leftrightarrow$ US switches
  - Reduce the number of context switches
  - Throughput vs low latency: opposite requirements
- So, maybe the ISR runs at the correct time, the delayed interrupt handler is executed immediately, but the scheduler is invoked after some time...



Latency in the Standard Kernel



Summing Up

- 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 of the problems:
  - Interrupt Handling (Device Drivers)
  - Management of the parallelism

Data Structures Consistency

- Hardware interrupt: *breaks* the regular execution flow
  - If the CPU is executing in US, switch to KS
  - If execution is already in KS, possible problems
- Example:
  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...
- The kernel must *disable the interrupts* while updating the list!
- Similar interrupt disabling is also used in spinlocks and mutex implementations...