


Jitter



So far we have assumed the following model for our scheduler.


Slide 1

- A task is either runnable or not runnable, every task has a priority.
- When a task i passes its period T_i it becomes runnable.
- The run-time system ensures that the highest priority runnable task is running.

We have ignored some issues.

What if there is some delay between the time when a task becomes and when it is actually executed?


What is Jitter



Slide 2

- From dictionary.com: Random variation in the timing of a signal, especially a clock.
- In our context, jitter is the delay between the invocation (or arrival) of a task, and its release (when it actually starts to execute).

Examples



Slide 3

- Large distributed system: A sensor at site A triggers a task at site B. The message is sent over a complex network, and may take a varying amount of time to arrive. The difference between the earliest and latest possible release times of the task at site B is jitter.
- Operating systems: Say that the kernel preempts user processes at regular intervals (10 ms), and that you want a task to run every 23 ms. The OS scheduler will not be able to satisfy your request exactly. Theoretically, your task should run at times 0, 23, 46... In practice, it will run at 0, 30, 50... The variation of the actual release times is jitter (0, 7, 4, ... in this example).

Important



Slide 4 Jitter may have different causes. However, when you draw schedules, you should keep in mind the following points.

- Jitter does not preempt tasks, since jitter is not a task.
- Jitter does not get preempted, since it is not a task.

Example

Consider the task set:

Slide 5

Task Name	T	D	C	R
H	30	20	10	10
L	1000	25	15	25

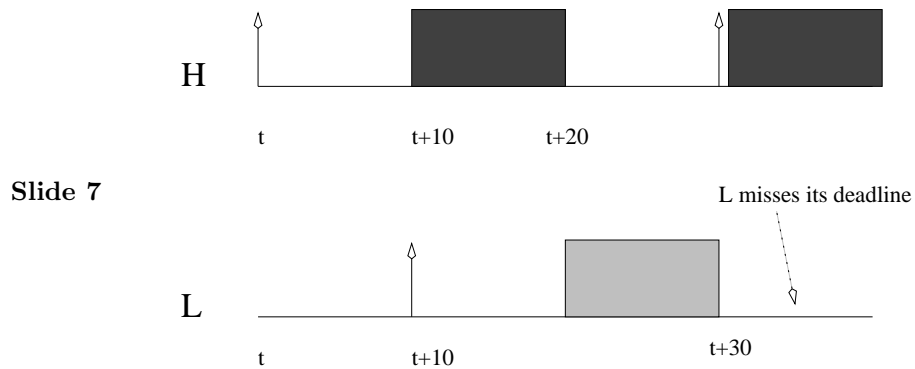
Using the response time formula we know that we can schedule the tasks.

Jitter

What happens if there is a difference between invocation time and running time? Suppose the following happens:

Slide 6

- A time t task H is invoked, but there is a delay 10ms before task H is run. So task H is released at time $t = 10$.
- At time $t + 10$ task L is released but it is preempted because task H is of a higher priority.
- At time $t + 30$ A is invoked again, but this time the delay between the invocation and its release is just 1ms. So at time $t + 41$ task H finished executing so task L misses its deadline.



Definition of Jitter

Slide 8

Release *jitter* is the difference between the earliest and the latest time a task could be released relative to the invocation of the task.

Jitter can occur for many reasons. For example the run-time system might be running on a different processor. Or there might be some implementation effects with the priority queue system implementing the runnable process list.

Analysing Jitter

Slide 9 The term jitter is used for release delays that are not constant so we can't simply change our basic equation to something like:

$$R_i = C_i + J_i + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

We have to be a bit more clever.

Defining J_i

Given a task i define

$$J_i^{\text{biggest}}$$

Slide 10 to be the longest delay from invocation of task i to the release of task i and

$$J_i^{\text{smallest}}$$

to be the smallest delay between invocation and release.

Then define:

$$J_i = J_i^{\text{biggest}} - J_i^{\text{smallest}}$$

Jitter Continued

Back to our example we have that $J_H = 10 - 9$. Task L can be preempted twice if $R_L > 21$.

So we can say that two preemption of task L may occur if:

Slide 11

$$R_L > T_H - J_H$$

Task L may be preempted three times if:

$$R_L > 2T_H - J_H$$

Task L may be preempted four times if:

$$R_L > 3T_H - J_H$$

Jitter Continued

Then L can be preempted n times if:

$$R_L > (n - 1)T_H - J_H$$

Slide 12

To work out the maximum number of preemption of a task i we need to find the maximum value of n , given R_L , T_H and J_H . So

$$\frac{R_L + J_H}{T_H} > n - 1$$

The largest value of $n - 1$ which satisfies the above equation is given by the ceiling function:

$$n = \left\lceil \frac{R_L + J_H}{T_H} \right\rceil$$

Modifying the Basic equation

Slide 13 It would seem we simple have to modify our basic equation to be:

$$R_i = C_i + B_i + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j$$

But where do we measure the response time?

When to measuring the response time

Slide 14 It would seem sensible to measure it relative to the invocation of the task. Define w_i to be the time taken for a task to complete once it has been released So the worst case response time for the task now becomes:

$$w_i = C_i + B_i + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j$$

The notation w is used because the time the task is running is a time window, during which higher priority tasks can be released and preempt.

Response time with Jitter

Slide 15 We still have to work out the response time.

In the worst possible scenario if a task is released at time t , then higher priority tasks can interfere between $t + J_i^{biggest}$ and $t + R_i$.

So the worst case response time is given by:

$$R_i = J_i^{biggest} + w_i$$

Priority Assignment with Jitter

Slide 16 It can be now shown that deadline monotonic assignment is not optimal for tasks with jitter. We have to order the priority on

$$D_i - J_i^{biggest}$$

so tasks with bigger jitter which contribute significantly to the deadline are given a higher priority.