3. Extensions to the Periodic Model

UNIVERSIDAD DE CANTABRIA

3.1 Modelling task switching

3.2 Preperiod deadlines: priority assignment

3.3 Preperiod deadlines: analysis techniques

3.4 Schedulability with interrupts

3.5 Non-preemptible sections

Extensions to Basic Theory

UNIVERSIDAD DE CANTABRIA

This section extends the schedulability tests to address:

- **nonzero task switching times**
- **preperiod deadlines**
- **interrupts**
- **non-preemptible sections**

Task interactions, aperiodic tasks, and advanced issues are considered in the following sections.

Notes:

UNIVERSIDAD DE CANTABRIA

To this point we have assumed that there was no context (or task) switching overhead, that all deadlines were at the end of the period, and that tasks were scheduled according to the rate monotonic algorithm. In particular, we have assumed perfect preemption for the priority-based scheduler and we have assumed that priorities are assigned rate monotonically. Lack of preemption has the effect of delaying (or blocking) higher-priority tasks until the lower-priority task finishes its nonpreemptive execution. This situation in which a higher-priority task is ready to run, yet is prevented from running by a lower-priority task, we call *priority inversion*.

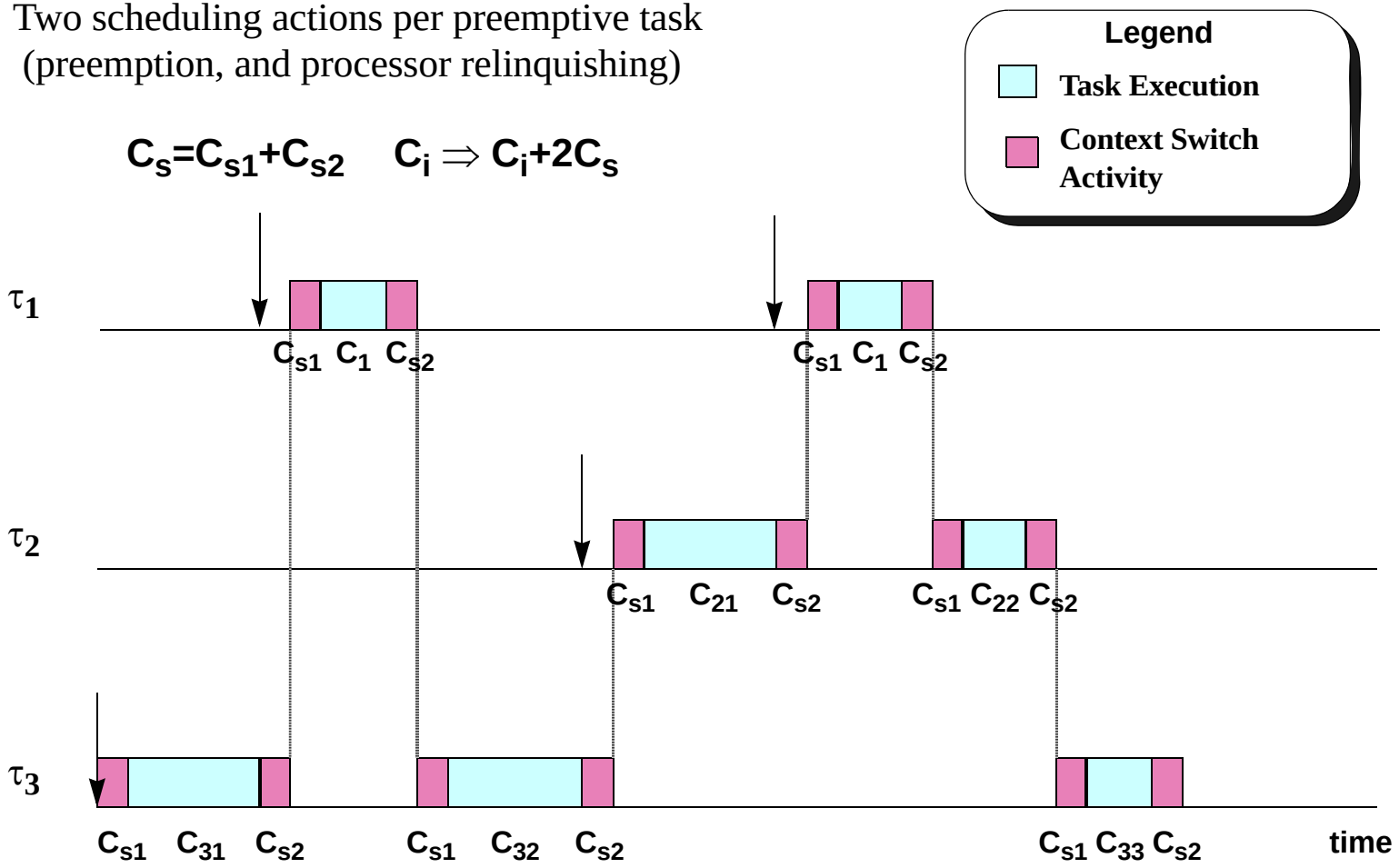
Another source of inefficiency arises when priorities are assigned to tasks in violation of the rate monotonic algorithm (and are therefore suboptimal). This may sometimes be unavoidable, such as the case of an interrupt service routine. In this situation the analysis has to take into account the negative effect.

We also want to consider the frequent case in which deadlines do not necessarily coincide with the end of the periods for all tasks in the system.

3.1. Modeling Task Switching

Two scheduling actions per preemptive task (preemption, and processor relinquishing)

$C_s = C_{s1} + C_{s2}$ $C_i \Rightarrow C_i + 2C_s$



Notes:

When a task preempts a lower-priority task the execution state of the lower-priority task is saved and the execution state of the higher-priority task is established. When the higher-priority task completes its processing and relinquishes the CPU to a lower-priority task, its execution state is saved and the state of the lower-priority task is reestablished. We will assume that both kinds of context switches, preemption, and processor relinquishing, have the same duration, which we call C_s. C_s is composed of two simpler actions, saving the context of the previous task and restoring the context of the new task. These simpler actions have durations C_{s1} and C_{s2}, respectively, and so $C_s=C_{s1}+C_{s2}$.

In the figure above we can see that each time a lower priority task is preempted by a higher priority task, it is delayed by two context switches: one at the time of preemption, and one at the time the preempting task relinquishes the processor. Therefore, we model context switch time by adding two context switches ($2C_s$) to the execution time of each task.

Consider that task τ_3 is preempted by higher-priority tasks, τ_1 and τ_2. Worst case is that τ_3 will be preempted by τ_1 (and τ_2) every time τ_1 (or τ_2) is ready to run. Each preemption causes two context switches; one from τ_3 to run τ_1 and one from τ_1 back to resume τ_3. These two context switch times are charged to the preempting task.

3.2 Preperiod Deadlines: Priority Assignment

UNIVERSIDAD DE CANTABRIA

The critical-instant and checking-the-first-deadline concepts are applicable.

Rate Monotonic Priorities are no longer optimum

The Utilization Bounds Test must be modified

The response time test is applicable with no modification

Notes:

UNIVERSIDAD DE CANTABRIA

It is common in many real-time applications to set a task's deadline to be prior to the end of its period, because a fast response is needed after an event arrives.

When some or all of the deadlines are before the end of the period, the two basic concepts of critical-instant and checking-the-first-deadline are still applicable. However, rate monotonic priorities are no longer optimum. We will discuss what the optimum priority assignment is in the next slides.

The utilization bounds test that was shown in is not applicable, but a different UB test exists for pre-period deadlines, which is also shown in a following slide.

Finally, the response test is completely applicable to the case with preperiod deadlines. This test is based on the critical instant concept and is applicable to any fixed-priority assignment. It allows obtaining the response time, which can then be checked against the actual deadline of each task.

Deadline Monotonic Priorities

UNIVERSIDAD DE CANTABRIA

For a set of periodic independent tasks, with deadlines within the period, the optimum priority assignment is the deadline monotonic assignment:

- **Priorities are assigned according to task deadlines.**
- **A task with a shorter deadline is assigned a higher priority**

Rate Monotonic priorities are a special case of Deadline Monotonic Priorities

Notes:

Leung and Layland proved that for a set of periodic independent tasks with deadlines within the period, the optimum priority assignment is the deadline monotonic assignment, in which priorities are assigned according to task deadlines. The task with the shortest deadline is assigned the highest priority, the task with the next shortest deadline is assigned the next highest priority, and so on.

The rate monotonic priority assignment is a special case of the deadline monotonic assignment, because when deadlines coincide with the end of the periods there is no difference between the two priority assignments. Still, the theory is sometimes called “Rate Monotonic Analysis” in part of the literature, for historical reasons.

3.3 Preperiod Deadlines: Analysis Techniques

UNIVERSIDAD DE CANTABRIA

Utilization Bounds Test for Pre-Period Deadlines

For the analysis of τ_i create the following two sets:

- **H1, singly preemptive tasks:**
  - **preemptive tasks with periods $\geq D_i$**
- **Hn, multiply preemptive tasks:**
  - **preemptive tasks with periods $< D_i$**

The effective utilization for task τ_i is:

$$f_i = \sum_{j \in Hn} \frac{C_j}{T_j} + \sum_{k \in H1} \frac{C_k}{T_i} + \frac{C_i}{T_i}$$

Notes:

UNIVERSIDAD DE CANTABRIA

This utilization bound test can be applied to sets of periodic independent tasks with deadlines within the period. When analyzing a given task τ_i, only that task and higher priority tasks need to be considered. The higher priority tasks are classified into two sets:

- The singly preemptive tasks*: the H1* set consists of higher priority tasks that preempt task τ_i only once before its deadline at D_i.
- The *multiply preemptive tasks: the Hn* set consists of higher priority tasks that may preempt task τ_i multiple times before the deadline at D_i.

This test treats event sequences in *H1* different than event sequences in *Hn*. The utilization term for tasks in *Hn* is C_j/T_j for each task τ_j (the denominator is the period of τ_j). The utilization term for tasks in *H1* is C_k/T_i for each task τ_k (the denominator is the period of τ_i).

Utilization Bounds (cont.)

The utilization bound for task τ_i is:

$$n = num(Hn) + 1$$

$$\Delta_i = \frac{D_i}{T_i} \leq 1$$

$$U(n, \Delta_i) = \begin{cases} n((2\Delta_i)^{1/n} - 1) + 1 - \Delta_i & , 0.5 \leq \Delta_i \leq 1 \\ \Delta_i & , 0 \leq \Delta_i \leq 0.5 \end{cases}$$

And the schedulability test is: $f_i \leq U(n, \Delta_i)$

Notes:

UNIVERSIDAD DE CANTABRIA

The number *n* is the number of multiply preemptive tasks (those in *Hn*) plus one.

Notice that in this utilization bound test, the effective utilization must be calculated individually for each task and also, the utilization bound is different for each task.

As with the utilization bound test for tasks with deadlines coincident with the end of the period, the test is only a sufficient condition for schedulability. If the result of the test is negative, the task set may still be schedulable. To determine a necessary and sufficient condition for schedulability, the response time test can be applied.

Sample Prob.: Task Switching and Preperiod Deadline

UNIVERSIDAD DE CANTABRIA

Applying the UB Test to the sample problem, with D_2=130:

τ_1 $$\frac{(C_1 + 2C_s)}{T_1} \leq U(1,1) = U(1) = 1$$

τ_2 $$\frac{(C_1 + 2C_s)}{T_1} + \frac{(C_2 + 2C_s)}{T_2} \leq U(2, 130/150) = 0.766$$

τ_3 $$\frac{(C_1 + 2C_s)}{T_1} + \frac{(C_2 + 2C_s)}{T_2} + \frac{(C_3 + 2C_s)}{T_3} \leq U(3) = 0.779$$

Results

UNIVERSIDAD DE CANTABRIA

Substituting the numbers into the schedulability formulas, we see all tasks are schedulable:

Task	Utilization
τ_1	$\frac{21}{100} = 0{,}210 < 1{,}000$
τ_2	$\frac{21}{100} + \frac{41}{150} = 0{,}484 < 0{,}766$
τ_3	$\frac{21}{100} + \frac{41}{150} + \frac{101}{350} = 0{,}772 < 0{,}779$

3.4 Schedulability with Interrupts

UNIVERSIDAD DE CANTABRIA

Interrupt processing can be inconsistent with rate monotonic priority assignment:

- interrupt handler executes with high priority despite its period
- interrupt processing may delay execution of tasks with shorter periods

Effects of interrupt processing must be taken into account in schedulability model:

- The response time test works fine for arbitrary priorities
- The UB test for deadlines within the period also works for arbitrary priorities
- The UB test for deadlines equal to periods must be modified

Notes:

Interrupt processing is common in real-time applications and often violates the optimum priority assignment. Typically, interrupt processing is higher in priority than application-level processing. So although it should be assigned a lower priority than an application-level task (i.e., when it has a longer period), it will have a higher *runtime* priority and will preempt the application-level tasks.

The response time test is capable of taking into account the effect of a non optimal priority assignment. In fact, the test works for any priority assignment.

However, the utilization bounds test must be modified to address the issue of non rate monotonic priorities. We will now explore how the basic rate monotonic schedulability model can be extended to account for interrupt processing.

To clarify some definitions;

- Runtime priority - the priority actually assigned to a task.
- Rate monotonic priority - the priority of a task, if assigned according to its rate.
- Runtime preemption - seizing of the CPU by a task with a higher runtime priority.

Example: Determining Schedulability with Interrupts

	C	T	U
Task τ_3:	60	200	0.300
Task τ_1:	20	100	0.200
Task τ_2:	40	150	0.267
Task τ_4:	40	350	0.115

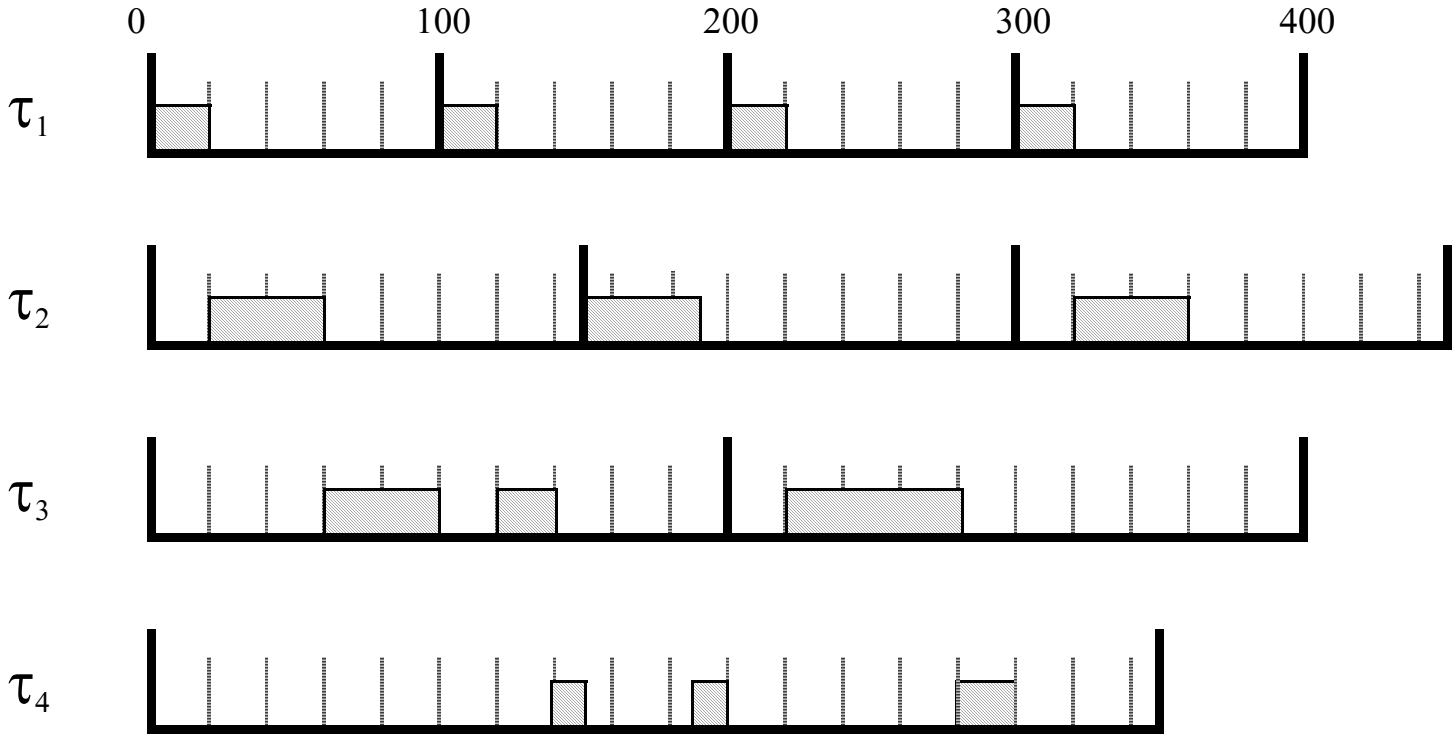
τ_3 is the interrupt handler

Notes:

We will illustrate the way we address a non optimal priority assignment with the above example.

Note that in this example, the interrupt should have a lower priority than tasks τ_1 and τ_2.

Example: Execution with Rate Monotonic Priorities



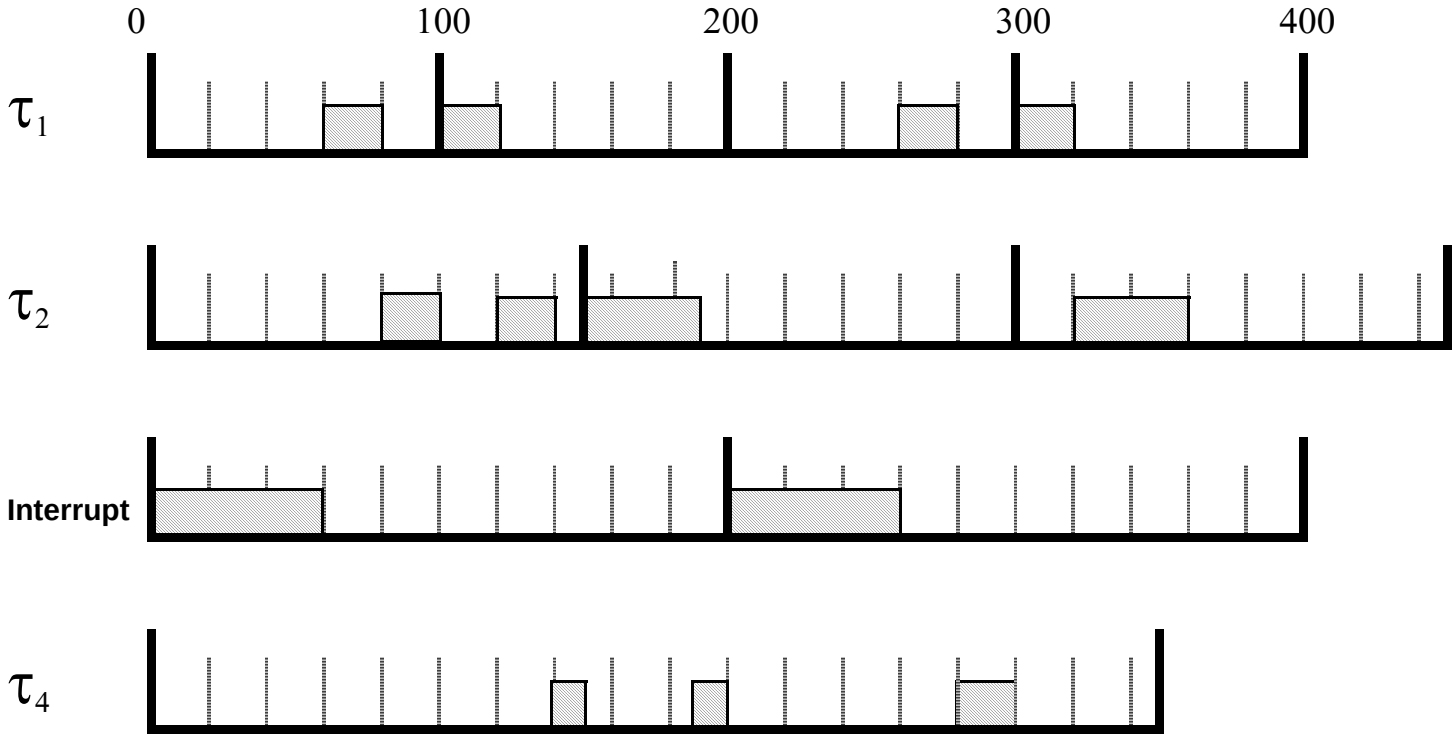
Notes:

UNIVERSIDAD DE CANTABRIA

This timeline shows how the tasks would execute under worst-case phasing **if** they ran in rate monotonic order. All tasks are clearly schedulable according to the “visual” RTA test.

Unfortunately, this timeline is **not** how the tasks execute. Since τ_3 is an interrupt handler, it executes before the other tasks, causing additional delays to τ_1 and τ_2.

Example: Execution with an Interrupt Priority



Notes:

UNIVERSIDAD DE CANTABRIA

This timeline shows the actual execution order, where τ_3 is an interrupt handler and executes with a runtime priority that is higher than the other three tasks, in spite of its longer period. Notice the impact that the interrupt has on the response time of the other three tasks. Both τ_1 and τ_2 are severely affected, being delayed by interrupt processing. However, τ_4 is not affected.

Execution is clearly inconsistent with the rate monotonic scheduling policy.

UB Test for Arbitrary Priorities

For the analysis of τ_i we can use the same test that is used for tasks with pre-period deadlines

Recall that we need to create the following two sets:

- ***H1*: singly preemptive tasks, with periods $\geq D_i$**
- ***Hn*: multiply preemptive tasks, with periods $< D_i$**

The effective utilization for task τ_i is:

$$f_i = \sum_{j \in Hn} \frac{C_j}{T_j} + \sum_{k \in H1} \frac{C_k}{T_i} + \frac{C_i}{T_i}$$

The effective utilization is then checked against the utilization bound

Notes:

UNIVERSIDAD DE CANTABRIA

This utilization bound test can be applied to sets of periodic independent tasks with deadlines at the end of the period, and arbitrary priorities. When analyzing a given task τ_i, only that task and higher priority tasks need to be considered. The higher priority tasks are classified into two sets:

- The singly preemptive tasks set, *H1,* consists of higher priority tasks that preempt task τ_i only once before its deadline at $D_i = T_i$
- The multiply preemptive tasks set, *Hn,* consists of higher priority tasks that may preempt task τ_i multiple times before the deadline at $D_i = T_i$

This test treats event sequences in *H1* different than event sequences in *Hn*. The utilization term for tasks in *Hn* is C_j/T_j for each task τ_j (the denominator is the period of τ_j). The utilization term for tasks in *H1* is C_k/T_i for each task τ_k (the denominator is the period of τ_i).

The utilization bound used is the utilization bound for rate monotonic priorities:

$$U(n) = n(2^{1/n} - 1)$$

If the deadline is smaller than the period, the bound for pre-period deadlines must be used instead.

3.5 Non-Preemptible Sections: Priority Inversion

UNIVERSIDAD DE CANTABRIA

Delay to a task's execution caused by interference from lower-priority tasks is known as *priority inversion*.

Priority inversion is modeled by blocking time.

Identifying, modeling, and reducing sources of priority inversion is central to schedulability analysis.

Sources of priority inversion:

- **Non-preemptible regions of code**
- **FIFO (first-in-first-out) queues**
- **Synchronization and mutual exclusion**

Notes:

Any delay to a task caused by a task with a lower priority is a priority inversion.

We model priority inversion by adding *blocking time* to the schedulable tests. By adding blocking time, we are taking into account the worst-case execution delays due to priority inversion.

Experience with RMA has shown priority inversion to be the most common cause for a task to miss its deadline.

There are many potential sources of priority inversion. This slide lists several sources of priority inversion:

- An example of priority inversion is when a higher-priority task is prevented from preempting a lower-priority task.
- Queues are also potential sources of priority inversion. If a high-priority task is waiting because a request that it made (such as an I/O call) is pending in a FIFO queue, the task may have to wait for lower-priority requests by lower-priority tasks to be satisfied.
- Synchronization caused by resource sharing also is a potential source of priority inversion. If a lower-priority task has reserved a resource that is needed by a higher-priority task, the higher-priority task is forced to wait.

Accounting for Priority Inversion

UNIVERSIDAD DE CANTABRIA

Recall that task schedulability is affected by:

- **preemption by higher priority tasks**
- **self execution**
- **blocking: delays caused by lower priority tasks**

The schedulability formulas are modified to add a “blocking” or “priority inversion” term to account for inversion effects.

Notes:

UNIVERSIDAD DE CANTABRIA

The schedulability tests assume that tasks are numbered in priority order. To determine whether a given task meets its deadline, one must consider three factors:

- preemption - execution time consumed by higher-priority tasks.
- execution time - from time consumed by the task itself.
- blocking - delays from execution by lower-priority tasks.

UB Test with Blocking

The general schedulability test for task τ_i is the same as before, with the addition of the blocking delay effect:

$$f_i = \sum_{j \in Hn} \frac{C_j}{T_j} + \sum_{k \in H1} \frac{C_k}{T_i} + \frac{C_i}{T_i} + \frac{B_i}{T_i} \leq U(n), n = num(Hn) + 1$$

This accounts for the following four effects:

- **multiply preemptive tasks**
- **singly preemptive tasks**
- **self execution**
- **blocking**

The UB test with blocking is used to construct a *schedulability model* of a set of tasks.

Notes:

The formula breaks down into the following types of terms:

1. Execution time consumed by higher-priority multiple preemptive tasks:

$$\sum_{j \in Hn} \frac{C_j}{T_j} = preemption(multiple)$$

2. Execution time consumed by higher-priority multiple preemptive tasks:

$$\sum_{k \in H1} \frac{C_k}{T_i} = preemption(single)$$

3. A task‘s own compute time:

$$\frac{C_i}{T_i} = execution$$

4. The sum of all blocking effects from lower-rate tasks:

$$\frac{B_i}{T_i} = blocking$$

Example with Interrupt and Non-Preemptible Section

	C	T	B
Task τ_1:	20	100	10
Task τ_2:	40	150	10
Task τ_3:	60	200	10
Task τ_4:	40	350	0

Task τ_3 is an interrupt service routine

Task τ_4 has a non-preemptible non-interruptible section of 10 ms.

Notes:

In this example we have three regular tasks and an interrupt service routine that executes at the highest priority in the system. Besides, the lowest priority task has a non-preemptible and non-interruptible section of code whose worst-case execution time is 10 ms. If one of the higher priority tasks wants to execute when task τ_4 is in its non-preemptible section, the higher priority task gets delayed. This represents a priority inversion that must be taken into account in the analysis.

In this case, the blocking term B_i for the three higher priority tasks is 10 ms., because this is the worst-case delay caused by a lower priority task.

Schedulability Model for Example

$\frac{C_1}{T_1} + \frac{C_{int}}{T_1} + \frac{B_1}{T_1} \leq U(1)$	$\frac{20}{100} + \frac{60}{100} + \frac{10}{100} = 0{,}9 \leq 1{,}0$
$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_{int}}{T_2} + \frac{B_2}{T_2} \leq U(2)$	$\frac{20}{100} + \frac{40}{150} + \frac{60}{150} + \frac{10}{150} = 0{,}934 > 0{,}828$
$\frac{C_{int}}{T_{int}} + \frac{B_3}{T_{int}} \leq U(1)$	$\frac{60}{200} + \frac{10}{200} = 0{,}350 < 1{,}00$
$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_{int}}{T_{int}} + \frac{C_4}{T_4} \leq U(4)$	$\frac{20}{100} + \frac{40}{150} + \frac{60}{200} + \frac{40}{350} = 0{,}882 > 0{,}756$

Notes:

UNIVERSIDAD DE CANTABRIA

The UB test takes the same approach and considers preemption, execution, and blocking. This slide shows how the UB test is modified to address blocking, as in the example.

Remember that in order to meet its deadline, τ_1 must accommodate its own execution, the preemption from the interrupt, and the blocking by the non preemptible section. Notice the denominator of each term. Because both τ_1's blocking and preemption from the interrupt occur only once during τ_1's period, the denominator is T_1 for both terms.

Considering τ_2, we remember that τ_1 can preempt more than once during τ_2's period. So for τ_2's schedulability test, the denominator for C_1 is T_1. The denominator for C_2, C_{int}, and B_2 is T_2.

The interrupt must accommodate only its own execution time and the blocking effect, so there are only two terms in its UB test. For τ_4, there are three sources of preemption, plus its own execution. So each term is the execution time of each preemptor divided by its own period.

The last thing to notice is the utilization bound on the right side of each inequality. The rule of thumb for each task is: the bound is based on the number of multiply preemptors, plus itself (in other words, the number of different denominators used).

RTA Test with Blocking

In the presence of priority inversion, the RTA test for τ_i must include the blocking delay:

$$a_0 = C_1 + C_2 + \ldots + C_i$$

$$a_{k+1} = W_i(a_k) = \left\lceil \frac{a_k}{T_1} \right\rceil C_1 + \ldots + \left\lceil \frac{a_k}{T_{i-1}} \right\rceil C_{i-1} + C_i + B_i$$

Notes:

The RTA test, like the UB test, may be modified to include blocking, by adding a blocking term to the formula.

In this case, no distinction is needed between multiply preemptive and singly preemptive tasks.

Example: RTA Test for τ_2

$$a_1 = \left\lceil \frac{a_0}{T_1} \right\rceil C_1 + \left\lceil \frac{a_0}{T_{int}} \right\rceil C_{int} + C_2 + B_2 = \left\lceil \frac{120}{100} \right\rceil 20 + \left\lceil \frac{120}{200} \right\rceil 60 + 40 + 10 = 150$$

$$a_2 = \left\lceil \frac{150}{100} \right\rceil 20 + \left\lceil \frac{150}{200} \right\rceil 60 + 40 + 10 = 150 \Rightarrow \text{Done}$$

Schedulable, since response time just meets the deadline

$$R_2 = 150 \leq T_2 = 150$$

Example: RTA Test for τ_4

Response time for τ_4

$$W_4(1) = B_4 + C_4 + \sum_{j<4} \left\lceil \frac{W_4(0)}{T_j} \right\rceil C_j = 0 + C_4 + \left\lceil \frac{160}{T_1} \right\rceil C_1 + \left\lceil \frac{160}{T_2} \right\rceil C_2 + \left\lceil \frac{160}{T_{int}} \right\rceil C_{int} = 220$$

$$W_4(2) = 40 + \left\lceil \frac{220}{100} \right\rceil 20 + \left\lceil \frac{220}{150} \right\rceil 40 + \left\lceil \frac{220}{200} \right\rceil 60 = 40 + (3)20 + (2)40 + (2)60 = 300$$

$$W_4(3) = 40 + \left\lceil \frac{300}{100} \right\rceil 20 + \left\lceil \frac{300}{150} \right\rceil 40 + \left\lceil \frac{300}{200} \right\rceil 60 = 40 + (3)20 + (2)40 + (2)60 = 300$$

The response time for τ_4 is W_4=300 which is less than the deadline T_4=350.

Therefore, τ_4 is schedulable using the RTA test.

Basic Theory: Where Are We?

UNIVERSIDAD DE CANTABRIA

We have shown how to handle:

- task context switching time: include $2C_s$ within C
- preperiod deadlines: RTA Test or modified UB Test
- priority inversion: include as blocking B
- arbitrary priorities

We still need to address:

- task interactions
- aperiodic tasks
- arbitrary deadlines
- multiprocessors
- operating system effects (other than context switch)

Notes:

UNIVERSIDAD DE CANTABRIA

By adding blocking, as well as context switching, we have extended the model to address the issues we will cover in this tutorial. We still have to cover task interactions through synchronization, which does not change the schedulability model but gives additional sources of blocking effects that must be included. We will also add aperiodic tasks, which are handled by making them pseudo-periodic. Finally, we will discuss multiprocessor issues and the effects of the operating system.

Remember that we still assume that priority-based scheduling is used, and that tasks do not suspend themselves. However, later we will study a case in which priority-based scheduling is not used and look at its schedulability.