Real-Time Operating Systems (0_KRI)
Response Time Analysis & DMPO

Ivan Cibrario Bertolotti

IEIIT-CNR / Politecnico di Torino

Academic Year 2006-2007

Outline

Criticisms of *U*-Based Tests

+ They are based on a single quantity, the utilization factor U, which is very **easy to compute**, even for large task sets.
– They are **not exact**, but provide only necessary or sufficient conditions for schedulability.
– They cannot be extended to more general process models, for example when we let $D_i \leq T_i$.

Observation

The utilization-based tests for RM are quite simple and useful, but have several significant drawbacks. A different, more sophisticated approach is needed to cope with these limitations.

Basic Principle of RTA

Response Time Analysis (Audsley et al., 1992)

To find an exact (necessary and sufficient) schedulability test for any fixed priority assignment scheme, the **exact interleaving** of higher-priority tasks must be analyzed, **individually**, for each task.

The test is performed in two stages:

1. An analytical method is used to **predict** the worst-case response time of each task.
2. These values are then **compared** with the corresponding task deadlines to assess whether all tasks meet their deadline or not.

Definition of Interference

- We still consider a fixed priority, preemptive scheduler under the basic process model, but we let $D_i \le T_i$ (instead of $D_i = T_i$).
- The preemption mechanism "grabs" the processor from a task whenever a higher-priority task is released.
- For this reason, all tasks (except the highest-priority one) suffer a certain amount of **interference** from higher-priority tasks during their execution.

The worst-case response time

According to the method proposed by Audsley et al., the worst-case response time R_i of task τ_i is computed as the sum of its computation time C_i and the **worst-case interference** I_i it experiences:

$$R_i = C_i + I_i$$

Worst-Case Interference

- Interference must be considered over **any** possible interval $[t, t + R_i[$, that is, for any t, to determine the worst case.
- The worst case occurs when all the higher-priority tasks are released at the same time as τ_i, that is, at a **critical instant**.
- Without loss of generality, it can be assumed that all tasks are released simultaneously at $t = 0$.
- The contribution of each higher-priority task to the overall worst-case interference will be analyzed individually.

Worst-Case Interference, Single Task

- Consider a single task τ_j of higher priority than τ_i.
- Within the interval $[0, R_i[$, τ_j will be released a number of times, and at least once (at $t = 0$).
- The number of releases can be computed by means of a ceiling function, as:

$$\left\lceil \frac{R_i}{T_j} \right\rceil$$

- Each release of τ_j will impose on τ_i an interference of C_j.

The **worst-case interference** imposed on τ_i by τ_j is:

$$\left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Worst-Case Interference, General Case

- Let hp(i) denote the set of task indexes with a priority higher than τ_i. These are the tasks from which τ_i will suffer interference.
- Hence, the total interference endured by τ_i is:

$$I_i = \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- This formulation is exact, but I_i cannot be computed unless we know R_i, the value being calculated.

Substituting this expression back into the defining equation of R_i, we obtain:

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

Worst-Case Response Time

Worst-Case Response Time R_i of τ_i

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

- No simple solution exists for this equation, since R_i appears on both sides, and is inside $\lceil \cdot \rceil$ on the right side.
- The equation may have more than one solution; the smallest solution is the actual worst-case response time.
- The simplest way of solving the equation is to form a **recurrence relationship**.

Recurrence Relationship

Recurrence Relationship for R_i

$$w_i^{(k+1)} = C_i + \sum_{j \in \mathrm{hp}(i)} \left\lceil \frac{w_i^{(k)}}{T_j} \right\rceil C_j$$

- Let $w_i^{(k)}$ be the k-th estimate of R_i.
- To obtain the next estimate of R_i, $w_i^{(k+1)}$ ...
- ... the right-hand side of the equation is evaluated with the current estimate, $w_i^{(k)}$.
- The succession $w_i^{(0)}, \ldots, w_i^{(k)}, \ldots$ is **monotonically nondecreasing**.

Convergence

If the initial approximation $w_i^{(0)}$ is chosen suitably, for example by letting $w_i^{(0)} = C_i$ (the smallest possible value of R_i), two cases are possible:

1. If the equation has no solutions, the succession does not converge and it will be $w_i^{(k)} > D_i$ for some k. In this case, τ_i clearly does not meet its deadline.
2. Else, the succession converges to R_i and it will be $w_i^{(k+1)} = w_i^{(k)} = R_i$ for some k. In this case, τ_i meets its deadline if and only if $R_i \leq D_i$.

What is $w_i^{(k)}$ (I)?

- $w_i^{(k)}$ is not a mere mathematical entity.
- If we consider a point of release of task τ_i, from that point and until that task instance completes, the processor will be **busy**, and will execute only tasks with the priority of τ_i or higher.
- Consider $w_i^{(k)}$ to be a **time window** that is moving down the busy period. If we let $w_i^{(0)} = C_i$, then the results of the ceiling operations will be (at least) 1. If this is indeed the case, then:

$$w_i^{(1)} = C_i + \sum_{j \in \mathrm{hp}(i)} C_j$$

- Since at $t = 0$ it is assumed that all higher-priority tasks have been released, this quantity represents the length of the busy period **unless** some of the higher-priority tasks are **released again** in the meantime.

What is $w_i^{(k)}$ (II)?

- If this is the case, the window will need to be pushed out further, by computing a new approximation of R_i.
- As a result, the window always expands and more and more computation time falls into the window.
- If this expansion continues indefinitely, then the busy period is **unbounded** and there is no solution.
- Else, at a certain point, the window will not suffer any additional “hit” from a higher-priority task.
- In this case, the window length is the **true length** of the busy period and represents the worst-case response time R_i.

Conclusion of the Analysis

- The worst-case response time R_i is individually calculated for each task $\tau_i \in \Gamma$.
- If, at any point, either a diverging succession is encountered or $R_i > D_i$ for some i, then Γ is **not schedulable**, because τ_i misses its deadline.
- Else, Γ is **schedulable** and the worst-case response time is known for all tasks.

Observations

- With this method, it is no longer assumed that $D_i = T_i \; \forall i$.
  It can be $D_i \leq T_i$.
- The method works with **any fixed priority ordering**, and not just with the RM assignment, as long as hp(i) is defined appropriately for all i and we use a **preemptive** scheduler.

Example

Let us consider the following tasks, with $D_i = T_i$:

Task	Period T_i	Computation time C_i	Priority
τ_1	7	3	High
τ_2	12	3	Medium
τ_3	20	5	Low

- The priority assignment is RM and the CPU utilization factor U is:

$$U = \sum_{i=1}^{3} \frac{C_i}{T_i} = \frac{3}{7} + \frac{3}{12} + \frac{5}{20} \simeq 0.93$$

- The necessary schedulability test does not deny schedulability, but the sufficient test for RM is of no help in this case.

Example, τ_1

- By intuition, the highest-priority task τ_1 does not endure interference from any other task. Hence, it will have a response time equal to its computation time, that is, $R_1 = C_1$.
- Analytically, this is because $\mathrm{hp}(1) = \emptyset$ and, given $w_1^{(0)} = C_1$, we trivially have $w_1^{(1)} = C_1$.
- In this case, $C_1 = 3$, hence $R_1 = 3$ as well.

Since $R_1 = 3$ and $D_1 = 7$, then $R_1 \leq D_1$ and τ_1 meets its deadline.

Example, τ_2

- For τ_2, $\text{hp}(2) = \{1\}$ and $w_2^{(0)} = C_2 = 3$. The next approximations of R_2 are:

$$\begin{aligned} w_2^{(1)} &= 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6 \\ w_2^{(2)} &= 3 + \left\lceil \frac{6}{7} \right\rceil 3 = 6 \end{aligned}$$

- Since $w_2^{(2)} = w_2^{(1)} = 6$, then the succession converges and $R_2 = 6$. In other words, widening the time window from 3 to 6 time units did not introduce any additional interference.

Task τ_2 meets its deadline, too, because $R_2 = 6$, $D_2 = 12$, and thus $R_2 \le D_2$.

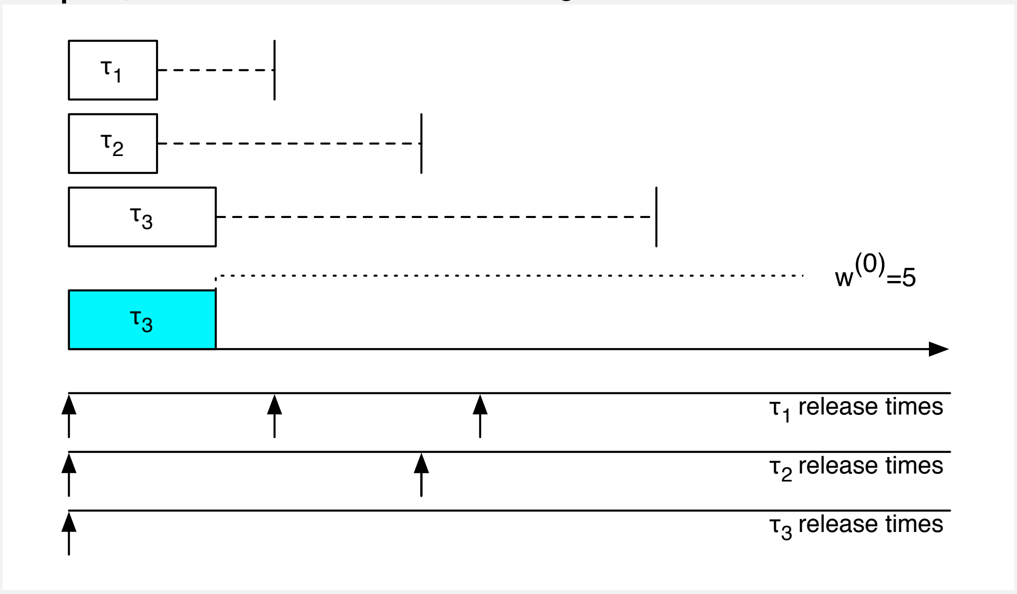
Example, τ_3

For τ_3, $\text{hp}(3) = \{1, 2\}$. It gives rise to the following calculations:

$$\begin{aligned} w_3^{(0)} &= 5 \\ w_3^{(1)} &= 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11 \\ w_3^{(2)} &= 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14 \\ w_3^{(3)} &= 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17 \\ w_3^{(4)} &= 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20 \\ w_3^{(5)} &= 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20 \end{aligned}$$

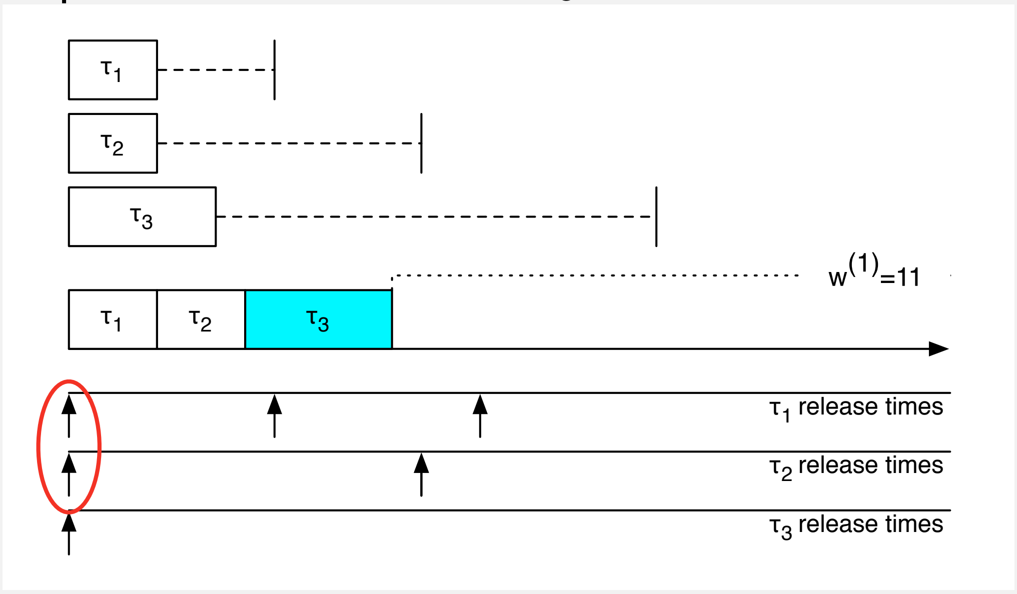
$R_3 = 20$ and $D_3 = 20$, thus $R_3 \le D_3$: τ_3 (just) meets its deadline.

Example, Time Windows for τ_3



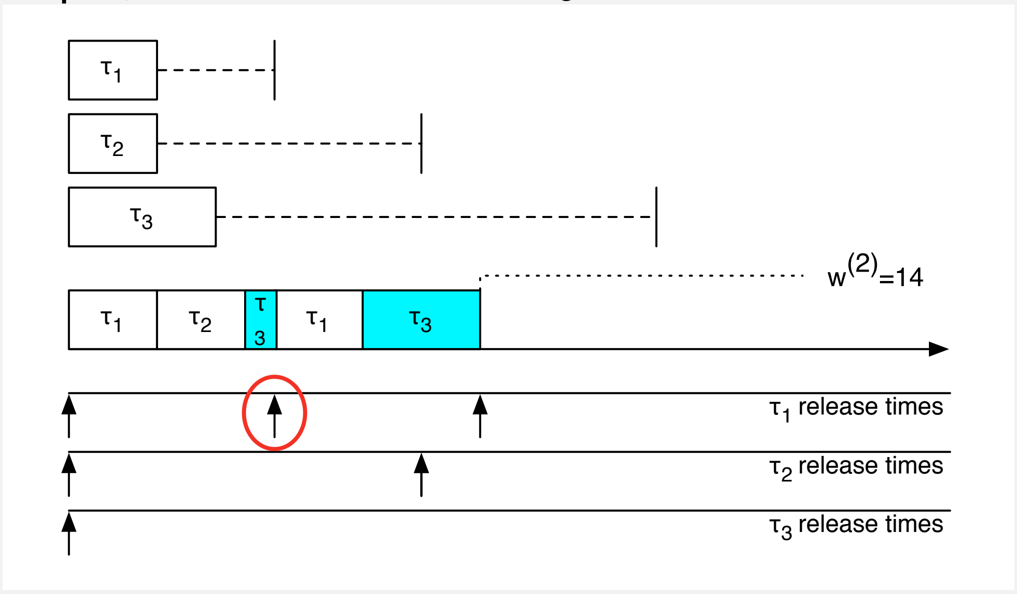
- If τ_3 were alone, its response time would be $C_3 = 5$.
- This is the initial approximation of the RTA, $w^{(0)}$.

Example, Time Windows for τ_3



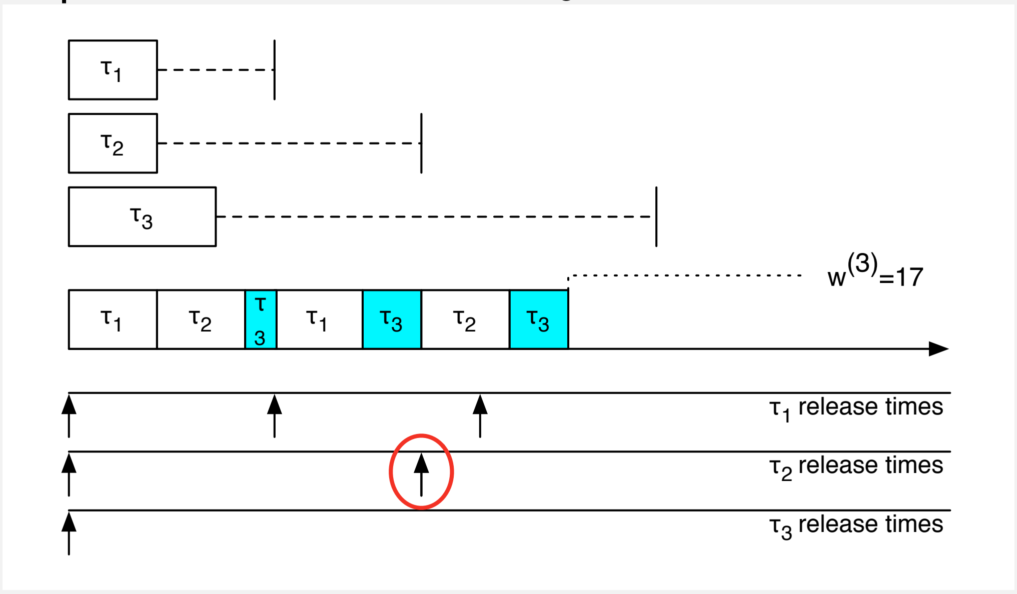
- We must consider the release of τ_1 and τ_2 at $t = 0$.
- The next approximation extends the window up to $w^{(1)} = 11$.

Example, Time Windows for τ_3



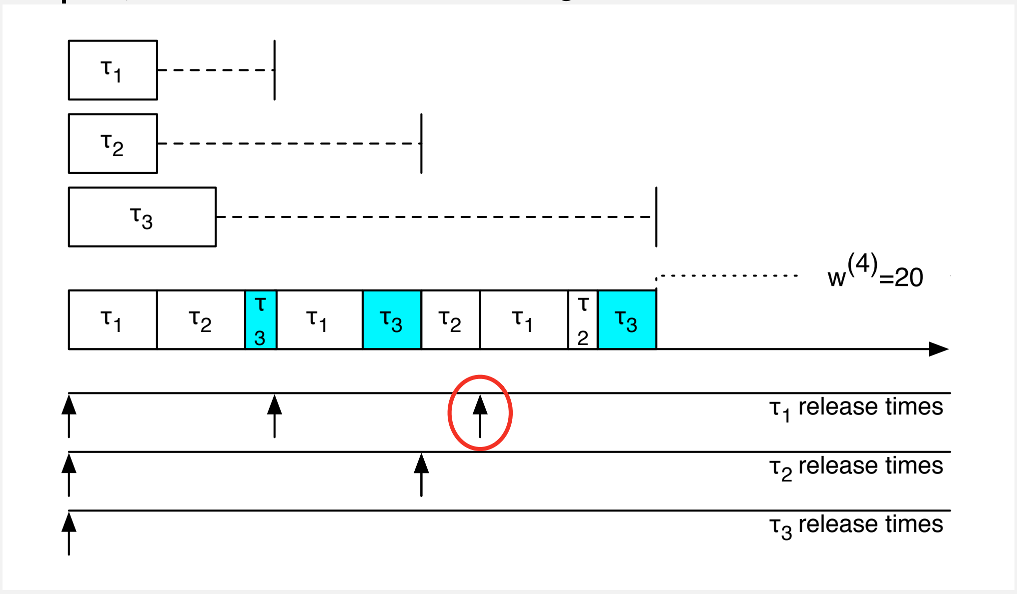
- Within this new window, τ_1 is released again, at $t = 7$.
- This release further extends the window up to $w^{(2)} = 14$.

Example, Time Windows for τ_3



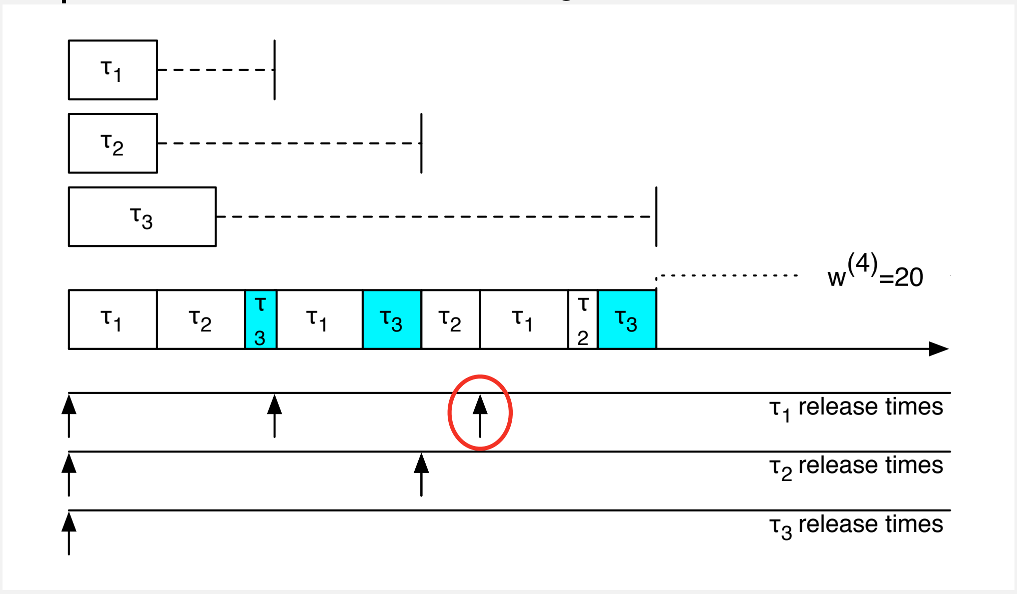
- Now we have to consider a further release of τ_2, at $t = 12$.
- The window is thus extended up to $w^{(3)} = 17$.

Example, Time Windows for τ_3



- The window now encompasses yet another release of τ_1, taking place at $t = 14$.
- Hence, it must be extended up to $w^{(4)} = 20$.

Example, Time Windows for τ_3



- The last extension does not encounter any further release of any other task.
- The worst-case response time of τ_3 is $R_3 = 20$.

Example, Summary

In summary, we have:

Task	Period T_i	Comp. time C_i	Worst-case resp. time R_i
τ_1	7	3	3
τ_2	12	3	6
τ_3	20	5	20

- The response time analysis guarantees that all tasks meet their deadline in this case.
- Unlike the U-based tests, it also provides the worst-case response time of each task.

Worst-Case Execution Time (Outline)

Observation

In all the scheduling analysis approaches described so far, it is assumed that the worst-case execution time of each task, C_i, is known. But, **how** is it determined?

The worst-case execution time can be obtained by two distinct methods, often used in combination:

1. **measurement**, and/or
2. **analysis**.

Measurement vs. Analysis

Measurement

+ Often **easy** to perform.
– It may be difficult to be sure that the worst case has **actually** been observed: for example, the task execution time can depend on its input data.

Analysis

+ It produces a tight estimate of the worst-case execution time.
– More **difficult** to perform: it requires an effective **model** of the processor (including pipelines, caches, memory, . . .) and sophisticated **code analysis** techniques.

Abstract of the Analysis Techniques

Most analysis techniques involve several distinct activities:

1. Decompose the code of the task into a directed graph of **basic blocks**. Each basic block is a “straight” segment of code (without tests, loops, . . .).
2. Consider each basic block and, by means of the processor model, determine its worst-case execution time.
3. Collapse the graph by means of the available semantic information about the program and, possibly, additional **annotations** provided by either the programmer or the compiler.

Collapsing the Code Graph

- For example, the graph of an `if P then` S_1 `else` S_2 statement can naively be collapsed into a single block whose worst-case execution time is equal to the maximum of the worst-case execution times of blocks S_1 and S_2.
- If that statement is enclosed in a loop to be executed 10 times it may be possible to deduce, by means of more sophisticated analysis techniques, that the predicate `P` can be true on at most 3 occasions.
- In this case, it becomes possible to compute the overall worst-case execution time for the whole loop more accurately than simply taking the maximum of the worst-case execution times of S_1 and S_2, and multiplying it by 10.
- Often, some **restrictions** must be placed on the **structure of the code** in order to perform such an accurate analysis.

The Biggest Challenge

The biggest **challenge** facing both the measurement and the analysis of the worst-case execution time comes from several **hardware components** commonly found in modern processors, for example caches, translation lookaside buffers, branch predictors, and even pipelines.

+ These devices aim to reduce the **average** execution time, and perform very well in this respect. As a consequence, ignoring them makes the analysis very pessimistic.
– On the other hand, their impact on the **worst-case** execution time can be hard to predict.

Effect of Caches

- On a cache **hit** most processors need only 1 clock cycle to access a data word.
- Instead, they often need more than 10 clock cycles on a cache **miss**.
- Even if it is possible to satisfactorily assess the performance of a cache from the **statistical** point of view, the behavior of a cache with respect to a **specific** data access is often hard to predict.
- Moreover, cache behavior depends in part on events external to the task under analysis, for example **preemptions**. This is true also for translation lookaside buffers and branch predictors.

Effect of TLBs and Branch Predictors

- **TLB**
  - On computers equipped with an MMU, the translation lookaside buffer (TLB) dispenses the system from performing a page table walk for most memory accesses. . .
  - . . . but it is usually hard to predict whether a particular memory access will give rise to a TLB **hit** or to a **miss**.
  - In the latter case, the memory access time may grow by an order of magnitude.
- **Branch predictors**
  - It is usually hard to predict whether a specific prediction will be right or wrong.
  - Also in this case, a prediction error has an adverse impact on the task execution time.

A Combined Approach

- To model accurately the temporal behavior of a modern processor is difficult, and may require proprietary information that can be hard to obtain.
- Hence, the choice is between either adopting a simpler (and less accurate) processor model, or using a simpler (and less powerful) processor architecture, or put more effort into measurement.

Given that most real-time systems will be subject to considerable testing anyway, e.g. for safety reasons, a combined approach which combines **testing and measurement** for basic blocks and **path analysis** for complete components can often be appropriate.

Aperiodic and Sporadic Tasks

Definitions

- A **periodic task** consists of an infinite sequence of identical activities called **instances**, or **jobs**, that are regularly **released**, or **activated**, at a constant rate.
- An **aperiodic task** consists of an infinite sequence of identical jobs. However, unlike periodic tasks, their release does **not** take place at a regular rate.
- An aperiodic task for which it is possible to determine a **minimum** inter-arrival time interval is called a **sporadic task**.

Why $D_i \leq T_i$ (I)?

- One simple way of expanding the basic process model to include **sporadic tasks** is to interpret to value T_i as the **minimum** inter-arrival time interval.
- For example, a sporadic task τ_i with $T_i = 20$ ms is guaranteed not to arrive more frequently than once in any 20 ms interval.
- Actually, it may arrive much less frequently, but a suitable schedulability analysis test will ensure (if passed) that the **maximum** rate can be sustained.
- For these tasks, assuming $D_i = T_i$ is unreasonable, because they usually encapsulate **error handlers** or respond to **alarms**.
- The fault model of the system may state that the error routine will be invoked rarely but, when it is, it has a **very short** deadline.
- For many periodic tasks it is useful to define a deadline shorter than the period, too.

Why $D_i \leq T_i$ (II)?

1. The response time analysis method just described is adequate for use with the extended process model just introduced, that is, when $D_i \leq T_i$.
2. The method works with **any fixed priority ordering**, and not just with the RM assignment, as long as hp(i) is defined appropriately for all i and we use a **preemptive** scheduler.

Observation

The second property is especially important to make the technique applicable also in this case. In fact, even if RM was shown to be an optimal fixed priority assignment scheme when $D_i = T_i$, this is **no longer true for** $D_i \leq T_i$.

The Deadline Monotonic Priority Order

Theorem (Leung and Whitehead, 1982)

*The deadline monotonic priority order (**DMPO**) — the priority assignment in which each task has a **fixed** priority **inversely proportional** to its **deadline** — is optimum for a preemptive scheduler under the basic process model extended to let $D_i \leq T_i$.*

- This assignment is optimum in the same sense as Liu and Leyland's: if **any** task set Γ can be scheduled using a preemptive, fixed-priority scheduling algorithm A...
- ...then the same task set can also be scheduled using the DPMO.
- The proof of optimality will involve transforming the priorities of Γ (as assigned by A), until the priority ordering is DM. Each transformation step will (of course) preserve schedulability.

Proof

- Let τ_i and τ_j be two tasks in Γ, with **adjacent priorities**, that are "in the wrong order" for DMPO under A.
- That is, let $P_i > P_j$ and $D_i > D_j$ under A, where P_i (P_j) denotes the priority of τ_i (τ_j).
- Define a new priority assignment scheme A' to be identical to A, except that τ_i and τ_j are **swapped**.
- Consider the schedulability of Γ under A', and prove that Γ is **still schedulable under** A'.

Effects of the Swap

- All tasks with a priority higher than P_i (the maximum priority of the tasks being swapped) will be unaffected by the swap.
- All tasks with priorities lower than P_j (the minimum priority of the tasks being swapped) will be unaffected by the swap, because the amount of interference they experience from τ_i and τ_j is **the same** before and after the swap.
- Task τ_j has an **higher priority after the swap than before** and was schedulable, by hypothesis, under A. After the swap it suffers either the same or less interference (due to the priority change), and hence it must be schedulable under A', too.

The most interesting step is to show that task τ_i, which was schedulable under A and has had its priority **lowered**, is still schedulable under A'.

What Happens to τ_i (I)?

Once the tasks have been switched, the **new** worst-case response time of τ_i becomes equal to the **old** response time of τ_j, that is, $R'_i = R_j$.

- Under A:
  - $R_j \leq D_j$ (schedulability)
  - $D_j < D_i$ (hypothesis)
  - $D_i \leq T_i$ (hypothesis)

  Hence, τ_i only interferes **once** during the execution of τ_j, because $R_j < T_i$.
- Under **both** priority orderings, $C_i + C_j$ amount of computation time is completed with the same amount of interference from higher-priority processes.

What Happens to τ_i (II)?

- Under A', since τ_j was released only once during R_j, it interferes only **once** during the execution of τ_i. Therefore, we have:
  - $R'_i = R_j$ (just proved)
  - $R_j \leq D_j$ (schedulability under A)
  - $D_j < D_i$ (hypothesis)
- Hence, $R'_i < D_i$ and it can be concluded that τ_i is **still schedulable** after the switch.

Conclusion

The DM priority assignment can be obtained from any other priority assignment by a **sequence of pairwise priority reorderings** as above. Each such reordering step preserves schedulability.

$\square$

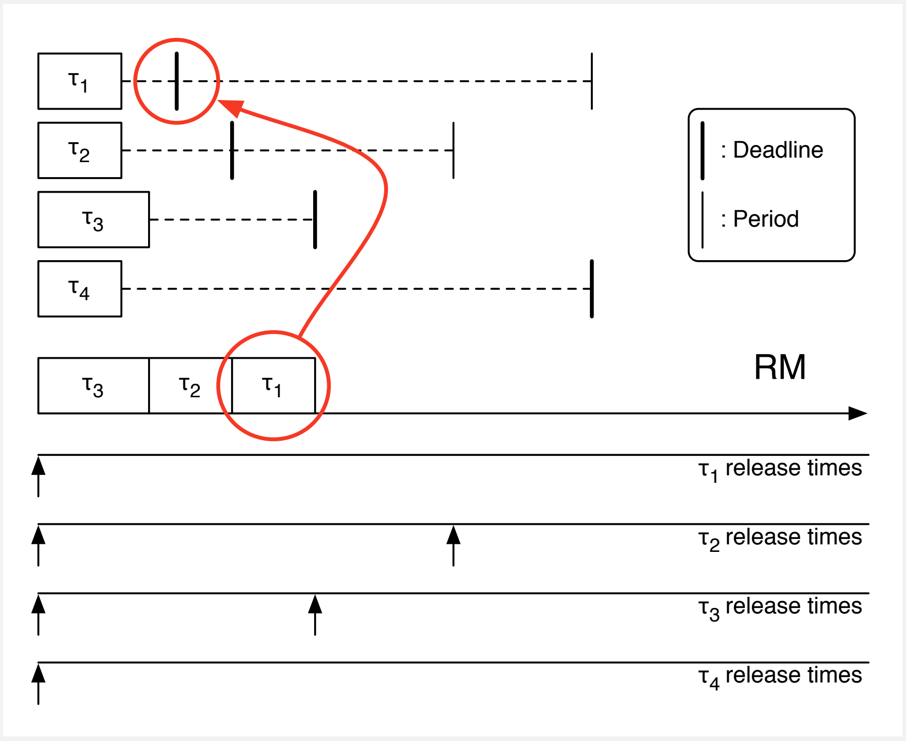
Example

Let us consider the following tasks, with $D_i \leq T_i$:

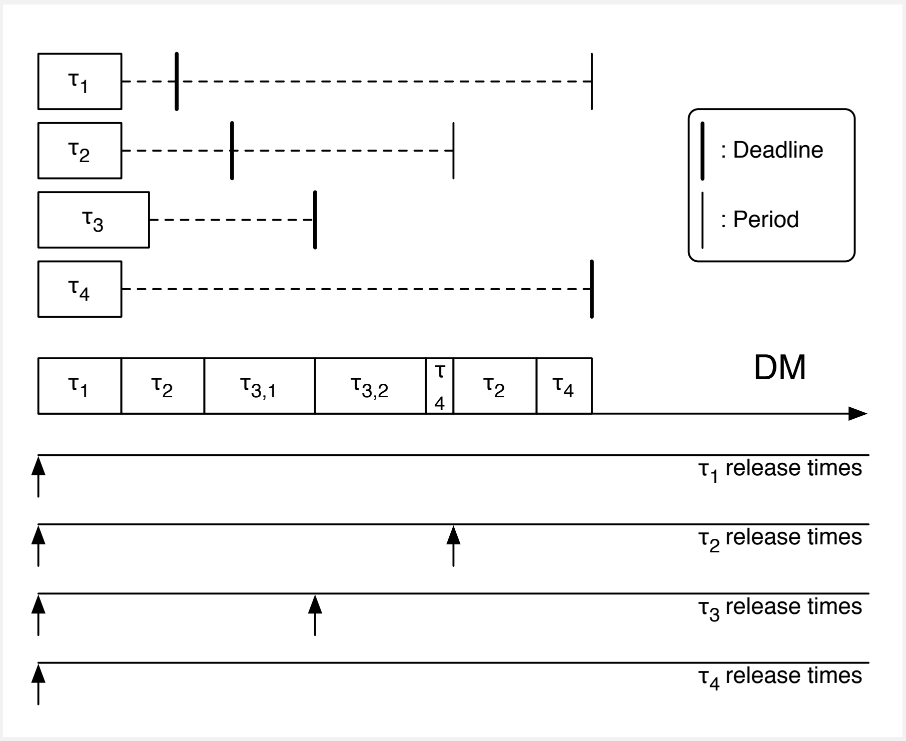
Task parameters				Priority	
Task	T_i	D_i	C_i	RM	DM
τ_1	20	5	3	Low	High
τ_2	15	7	3	Medium	Medium
τ_3	10	10	4	High	Low
τ_4	20	20	3	Low	Very low

- The RM and DM priority assignments **differ** for some tasks.
- The behaviors of RM and DM for this task set will be examined and **compared**.

RM is Unable to Schedule the Task Set



But DM Succeeds



RTA for the RM Schedule (I)

- For τ_3, hp(3) $= \emptyset$.
- Hence, $R_3 = C_3 = 4$ and τ_3 (trivially) meets its deadline.
- For τ_2, hp(2) $= \{3\}$.

$$\begin{aligned} w_2^{(0)} &= 3 \\ w_2^{(1)} &= 3 + \left\lceil \frac{3}{10} \right\rceil 4 = 7 \\ w_2^{(2)} &= 3 + \left\lceil \frac{7}{10} \right\rceil 4 = 7 \quad = R_2 \end{aligned}$$

Since $R_2 = 7$ and $D_2 = 7$, τ_2 meets its deadline.

RTA for the RM Schedule (II)

- For τ_1, hp(1) $= \{3, 2\}$.

$$\begin{aligned} w_1^{(0)} &= 3 \\ w_1^{(1)} &= 3 + \left\lceil \frac{3}{10} \right\rceil 4 + \left\lceil \frac{3}{15} \right\rceil 3 = 10 \\ w_1^{(2)} &= 3 + \left\lceil \frac{10}{10} \right\rceil 4 + \left\lceil \frac{10}{15} \right\rceil 3 = 10 \quad = R_1 \end{aligned}$$

Since $R_1 = 10$ and $D_1 = 5$, τ_1 **misses its deadline**: RM is unable to schedule this task set.

RTA for the DM Schedule (I)

- For τ_1, hp(1) $= \emptyset$.
- Hence, $R_1 = C_1 = 3$ and τ_1 (trivially) meets its deadline.
- For τ_2, hp(2) $= \{1\}$.

$$\begin{aligned} w_2^{(0)} &= 3 \\ w_2^{(1)} &= 3 + \left\lceil \frac{3}{20} \right\rceil 3 = 6 \\ w_2^{(2)} &= 3 + \left\lceil \frac{6}{20} \right\rceil 3 = 6 \quad = R_2 \end{aligned}$$

Since $R_2 = 6$ and $D_2 = 7$, τ_2 meets its deadline.

RTA for the DM Schedule (II)

- For τ_3, hp(3) $= \{1, 2\}$.

$$\begin{aligned} w_3^{(0)} &= 4 \\ w_3^{(1)} &= 4 + \left\lceil \frac{4}{20} \right\rceil 3 + \left\lceil \frac{4}{15} \right\rceil 3 = 10 \\ w_3^{(2)} &= 4 + \left\lceil \frac{10}{20} \right\rceil 3 + \left\lceil \frac{10}{15} \right\rceil 3 = 10 \quad = R_3 \end{aligned}$$

Since $R_3 = 10$ and $D_2 = 10$, τ_3 meets its deadline, too.

RTA for the DM Schedule (III)

- For τ_4, hp(4) $= \{1, 2, 3\}$.

$$\begin{aligned}
w_4^{(0)} &= 3 \\
w_4^{(1)} &= 3 + \left\lceil \frac{3}{20} \right\rceil 3 + \left\lceil \frac{3}{15} \right\rceil 3 + \left\lceil \frac{3}{10} \right\rceil 4 = 13 \\
w_4^{(2)} &= 3 + \left\lceil \frac{13}{20} \right\rceil 3 + \left\lceil \frac{13}{15} \right\rceil 3 + \left\lceil \frac{13}{10} \right\rceil 4 = 17 \\
w_4^{(3)} &= 3 + \left\lceil \frac{17}{20} \right\rceil 3 + \left\lceil \frac{17}{15} \right\rceil 3 + \left\lceil \frac{17}{10} \right\rceil 4 = 20 \\
w_4^{(3)} &= 3 + \left\lceil \frac{20}{20} \right\rceil 3 + \left\lceil \frac{20}{15} \right\rceil 3 + \left\lceil \frac{20}{10} \right\rceil 4 = 20 \quad = R_4
\end{aligned}$$

Finally, also τ_4 meets its deadline, because $R_4 = 20$ and $D_4 = 20$.

Final Remarks

- Preemptive, fixed-priority scheduling, with the DM priority ordering, can adequately deal with process systems in which $D_i \leq D_i$.
- For EDF, the simple U-based necessary and sufficient schedulability test is **no longer valid** in this case.
- Moreover, the response time analysis method can be applied to EDF, but is considerably more complex (and beyond the scope of this course) that it is for FPS.
- The response time analysis method for FPS has proved its effectiveness when dealing with an extended process model, hence it has been used as a starting point for **further extensions**.