Non-Preemptive and Limited Preemptive Scheduling

Prof. Dr. Jian-Jia Chen, Georg von der Brüggen

LS 12, TU Dortmund

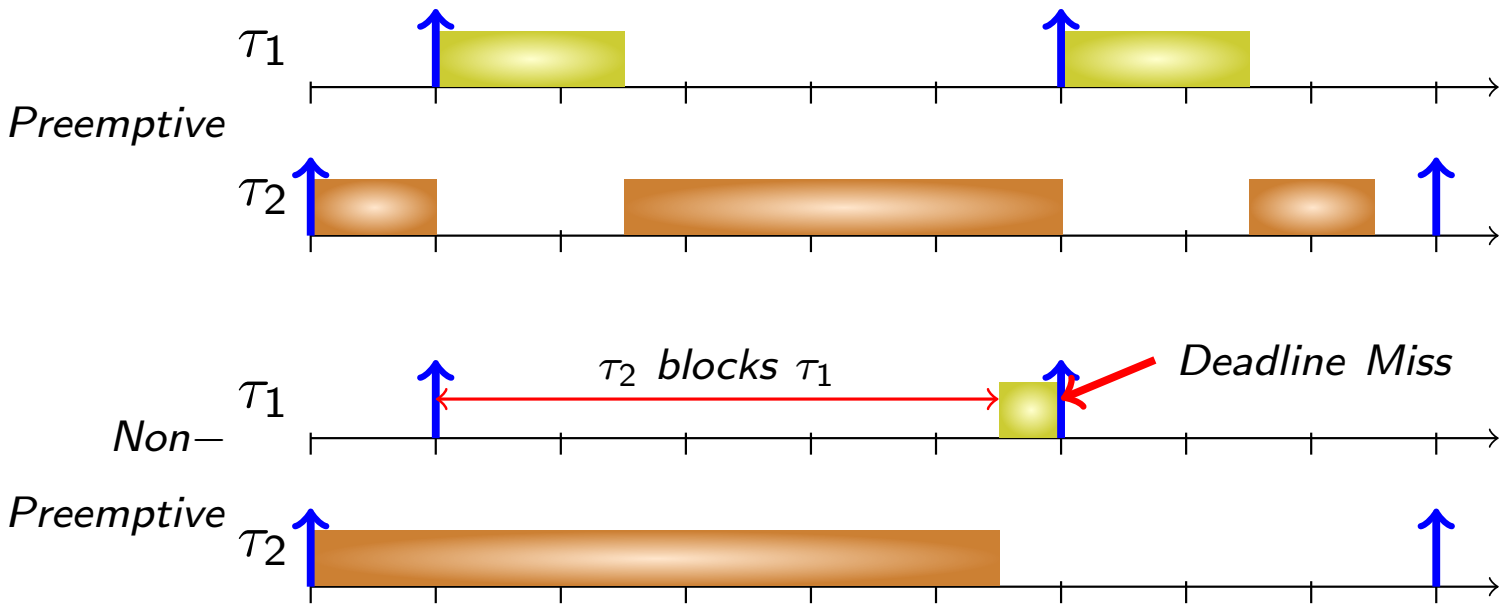
09 May 2017

Outline

Advantage of Preemption

Preemption is often seen as a key factor in real-time scheduling

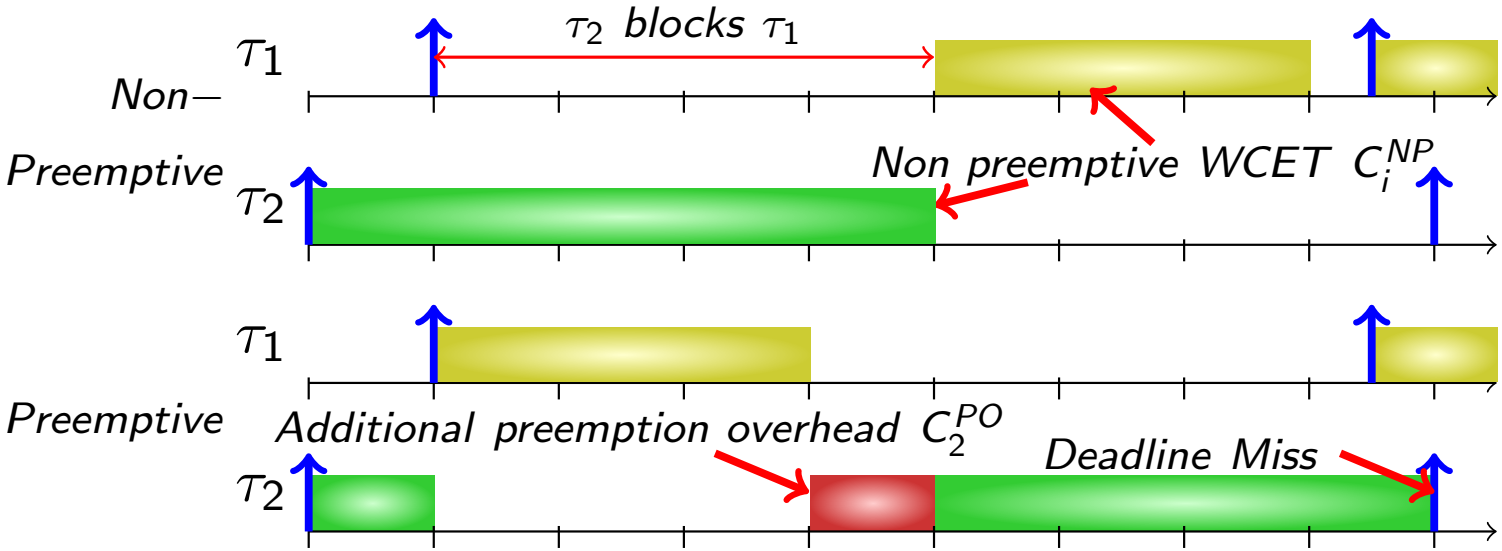
- Preemption allows to allocate the processor to incoming high priority tasks nearly immediately
- High priority tasks are not blocked by low priority tasks



Disadvantage of Preemption - Context Switch Costs

Context switch introduces overhead to the system:

- Scheduling costs: Time the scheduling Algorithm needs to suspend the running task, insert it into the ready queue, switch the context, dispatch the new incoming task
- Pipeline costs: Time to flush and refill the processor pipeline
- Cache-related costs: Time to reload the evicted cache lines
- Bus-related costs: Additional bus interference for accessing RAM at cache misses caused by preemption



Calculating The Preemption Overhead

- Essential for real-time systems: good estimation of WCETs
- WCET normally determined for non-preemptive case: C_i^{NP}
  - Time the processor needs to execute without interruptions
- Idea: add preemption costs to preempted tasks WCET
- Costs for one preemption C_i^{PO} has to be estimated
  - Summing up all context switch costs
  - Problem: especially the cache related and bus related costs can change drastically depending on the preemption point
- The number of preemptions p has to be estimated
  - Depends on the higher priority tasks
  - Ensure the estimated number is safe: $p_{est} \geq p$
  - But not much over estimation
- Preemptive WCET: $C_i^P = C_i^{NP} + p_{est} \cdot C_i^{PO}$
- Indirect preemption costs: the extra execution time also increases the number of preemptions

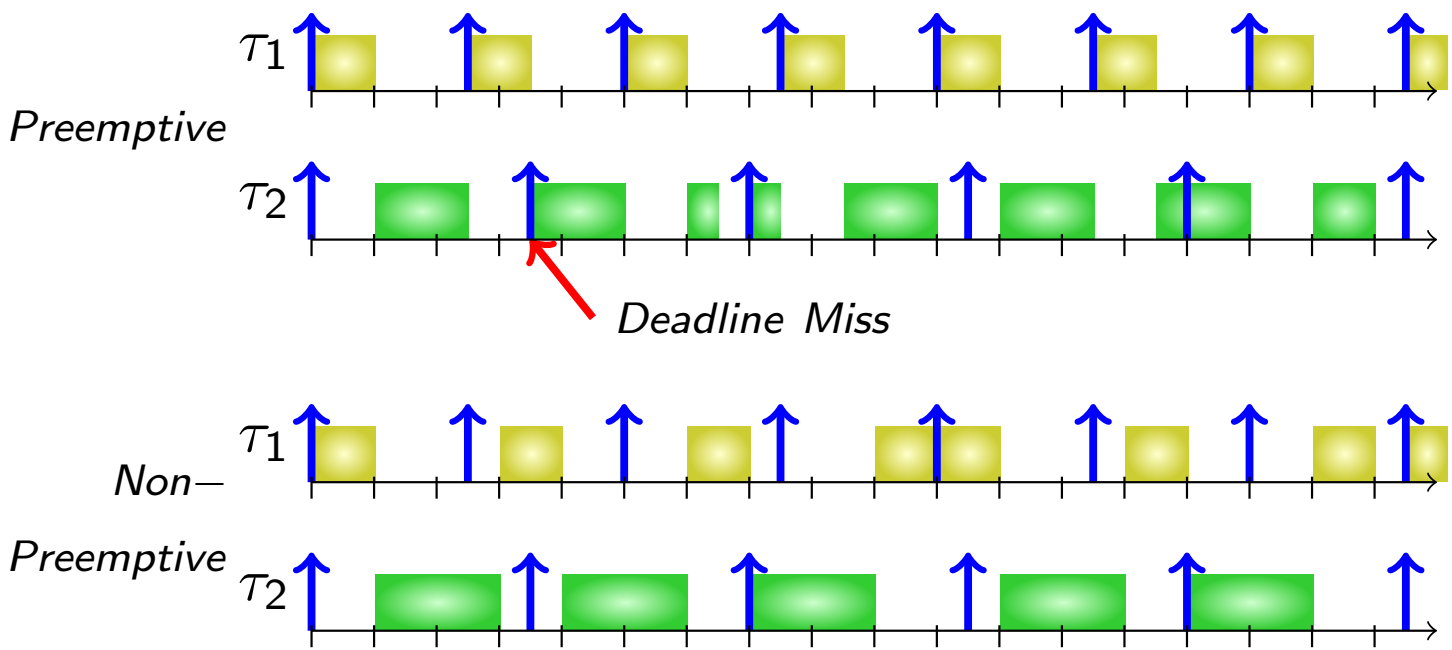
$\Rightarrow$ It is hard to determine a good WCET (safe but not much over estimation) for the preemptive case

Advantages of Non-Preemptive Scheduling

- It reduces context-switch overhead
  - Making WCETs
    - smaller
    - easier to calculate / more predictable
- It simplifies the access to shared resources
  - No semaphores are needed for critical sections
  - Deadlock prevention is trivial for non-preemptive scheduling
- It reduces stack size
  - Task can share the same stack, since no more than one task can be in execution
- Preemption may be very costly or forbidden for some actions anyways, e.g. I/O
  - Non-preemption allows zero I/O jitter: $C_i = f_i - a_i$ (constant)

Advantages of Non-Preemptive Scheduling (contd)

Also preemption is assumed to be a key factor for schedulability, there are some task sets that are schedulable in the non-preemptive case and not schedulable in the preemptive case, even when the preemption overhead is ignored:
RM, $\tau_1 = (2, 5),\ \tau_2 = (4, 7),\ U_{sum} = \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \approx 0.97$



(under the assumption that the arrival times of the jobs are integers)

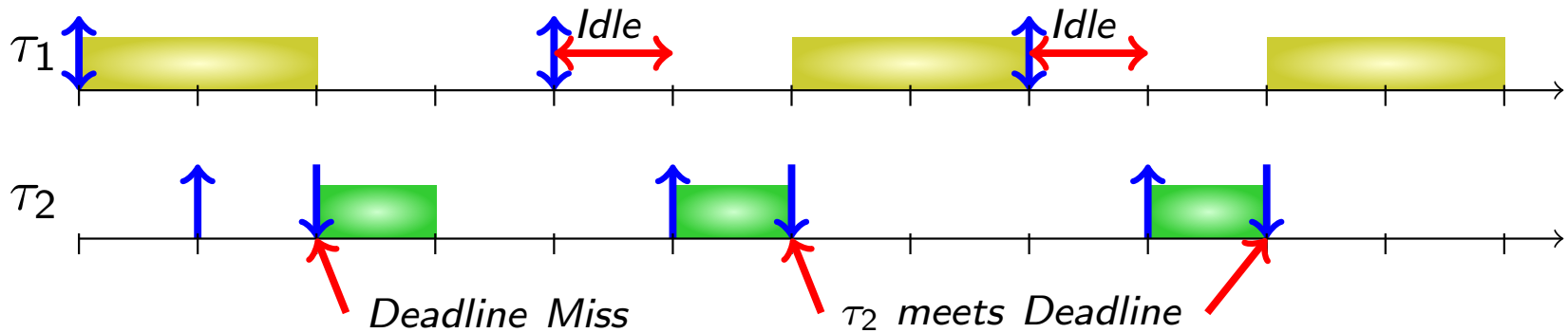
Issues with Non-Preemptive Scheduling

If preemption is not allowed, optimal schedules may have to leave the processor idle at certain times.

Assume the following periodic constrained deadline task set:

- $\tau_1: \; C_1 = 2, \; T_1 = 4, \; D_1 = 4, \; \Phi_1 = 0$
- $\tau_2: \; C_2 = 1, \; T_2 = 4, \; D_2 = 1, \; \Phi_1 = 1$

$\Rightarrow \tau_1$ is always available at $4 \cdot n$ and τ_2 always at $4 \cdot n + 1$



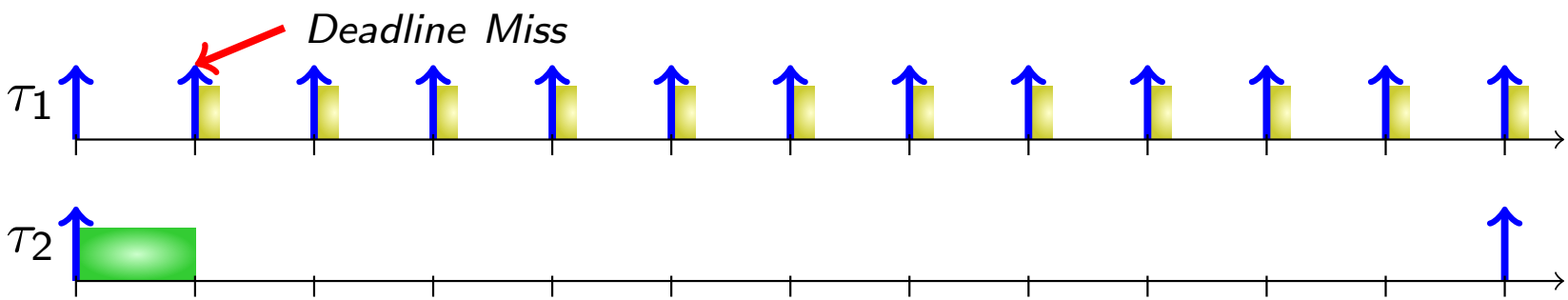
Hardness for Non-Preemptive Scheduling

- Optimal schedules may leave processor idle to finish tasks with early deadlines arriving late.
- Knowledge about the future is needed for optimal scheduling algorithms.
- No online algorithm can decide whether to keep idle or not.
- EDF is optimal among workload conserving scheduling algorithms, i.e., algorithms that due not keep the processor idle as long as there is workload to be executed.
  - Recent proof by von der Brüggen, Chen and Huang (ECRTS 2015) shows that (non-preemptive) RM and DM has a resource augmentation factor 1.76322 compared to (non-preemptive) EDF for implicit-deadline and constrained-deadline sporadic task systems.
  - Resource augmentation factors will be explained in a few weeks.
- Even if arrival times are known a priori, the scheduling problem is still NP-hard in the strong sense.

Outline

Problem - Utilization

- For preemptive RM scheduling we got easy utilization based schedulability tests (Liu & Layland and Hyperbolic Bound)
- The utilization bound under non preemptive scheduling drops to zero
- $\tau_1 = (\varepsilon,\ T_1),\ \tau_2 = (T_1,\ T_2),\ \varepsilon > 0$ but very small
- If τ_2 starts right before τ_1 arrives, τ_1 always misses its deadline
- We can make $\varepsilon > 0$ arbitrary small and T_2 arbitrary large
- $U_{sum} = \frac{\varepsilon}{T_1} + \frac{T_1}{T_2} = \frac{0}{T_1} + \frac{T_1}{\infty} = 0$

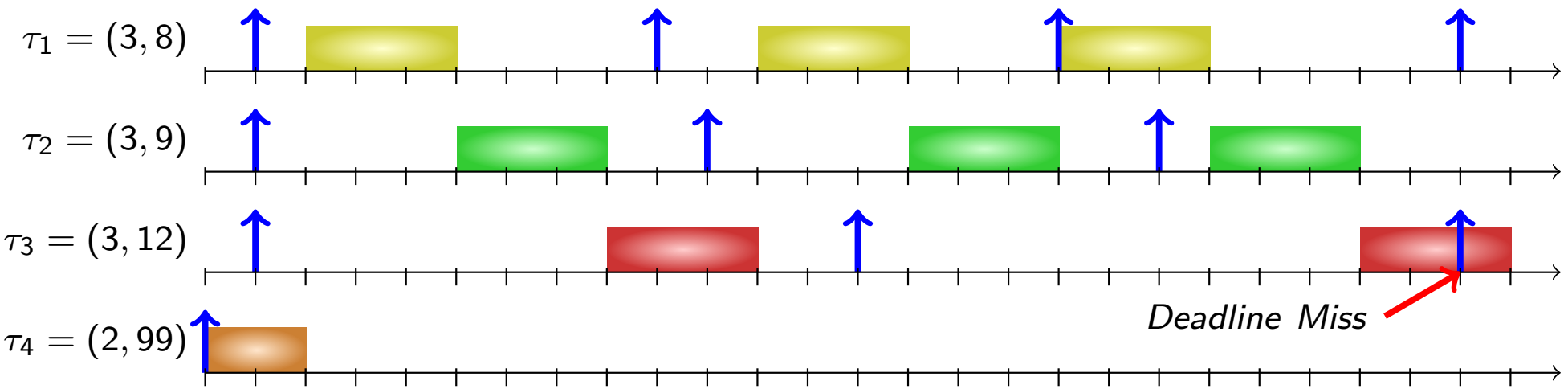


Problem - Self-Pushing Phenomenon

Analysis of non-preemptive systems more complex: largest response time may not occur in the first job after the critical instant

Definition: Self-Pushing Phenomenon

High priority jobs activated during non-preemptive execution of lower priority tasks are pushed ahead and introduce higher delays in subsequent jobs of the same task.



Self-pushing phenomenon for τ_3

General Response Time Analysis - Definitions

Definition: $hp(\tau_k)$, $lp(\tau_k)$:

- $hp(\tau_k)$: set of tasks with priority higher than τ_k
- $lp(\tau_k)$: set of tasks with priority lower than τ_k

Definition: Maximum Blocking Time B_k:

The maximum blocking time $B_k = \max_{\tau_i \in lp(\tau_k)} \{C_i - \varepsilon\}$

where $\varepsilon > 0$ but arbitrary small

Informal: To determine if a task τ_k is schedulable, we have to start at the critical instance and check all jobs of τ_k until the processor idles for the first time by

- summing up the interference from $\tau_i \in hp(\tau_k)$
- summing up the computation amount of previous jobs of τ_k
- add the maximum blocking time B_k from lower priority task

Formal Definition - Level-k Active Period

Definition: Level-k Pending Workload $W_k^p(t)$

The Level-k Pending Workload $W_k^p(t)$ at time t is the amount of processing that still needs to be performed at time t due to jobs with priority higher than or equal to τ_k released strictly before t

Definition: Level-k Active Period

A Level-k Active Period L_k is an Interval $[a,\ b)$ such that $W_k^p(t) > 0\ \forall t \in (a,\ b)$ and $W_k^p(t) = 0$ for $t = a$ and $t = b$

Computing the longest Level-k Active Period:
the smallest value where $L_k^{(s)} = L_k^{(s-1)}$ with

$$L_k^{(0)} = B_k + C_k$$

$$L_k^{(s)} = B_k + \sum_{\tau_i \in \{hp(\tau_k) \cup \tau_k\}} \left\lceil \frac{L_k^{(s-1)}}{T_i} \right\rceil C_i$$

Exact Schedulability Test

- Due to Self-Pushing Phenomenon: compute the response time of all jobs $\tau_{k,j}$ with $j \in [1, K_k]$ where $K_k = \left\lceil \frac{L_k}{T_k} \right\rceil$
- The start time $s_{k,j}$ of $\tau_{k,j}$ can be computed recurrently as well:

$$s_{k,j}^{(0)} = B_k + \sum_{\tau_i \in \{hp(\tau_k) \cup \tau_k\}} C_i$$

$$s_{k,j}^{(s)} = B_k + (j-1)C_k + \sum_{\tau_i \in hp(\tau_k)} \left(\left\lfloor \frac{s_{k,j}^{(s-1)}}{T_i} \right\rfloor + 1 \right) C_i$$

- As non-preemptive scheduling is used, a job always finishes once it is started $\Rightarrow \; f_{k,j} = s_{k,j} + C_k$
- Response time of τ_k: $R_k = \max_{j \in [1,K_k]} \{f_{k,j} - (j-1)T_k\}$
- A task set is feasible $\Leftrightarrow \; R_i \leq D_i \; \forall i = 1, \ldots, n$

Outline

Restricting the Analysis to the First Job

- Due to Self-Pushing Phenomenon: For an exact test we have to test all jobs in the Level-k Active Period
- We can restrict to only looking at the first job under some (not to restrictive) conditions

Theorem

[Yao, Buttazzo, and Bertogna, 2010] The worst-case response time of a non-preemptive task occurs in the first job if the task is activated at its critical instant and the following two conditions are both satisfied:

1. the task set is feasible under preemptive scheduling;
2. the relative deadlines are less than or equal to periods.

The recurrent relation to determine the start time in this case is:

$$s_{k,j}^{(s)} = B_k + C_k + \sum_{\tau_i \in hp(\tau_k)} \left(\left\lfloor \frac{s_{k,j}^{(s-1)}}{T_i} \right\rfloor + 1 \right) C_i$$

A Pessimistic Schedulability Test

- When we restrict ourselves to the first task, we still get an exact test using the recurrent computation of the start time
- This test still has pseudo-polynomial runtime
- Idea: sacrifice some precision to get a sufficient but easier test
- From the theorem by Yao, Buttazzo, and Bertogna we know we have to consider schedulability in the preemptive case
- Exact schedulability test in the preemptive case:

$$\exists t \text{ with } 0 < t \leq D_k \text{ and } C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$$

- We get a more pessimistic test by testing schedulability for preemptive and non-preemptive in one equation:

$$\exists t \text{ with } 0 < t \leq D_k \text{ and } B_k + C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$$

A Utilization Based Test

Problem with utilization based tests for NP: Blocking time has to be taken into account for every task individually

$\Rightarrow$ Every task has to be tested individually

Theorem

[Theorem 1 in von der Brüeggen, Chen, and Huang, 2015]
A task τ_k in a non-preemptive sporadic task system with constrained deadlines can be feasibly scheduled by a fixed-priority scheduling algorithm, if the schedulability for all higher priority tasks has already been ensured and the following condition holds:

$$\left(\frac{C_k + B_k}{D_k} + 1\right) \prod_{\tau_j \in hp(\tau_k)} (U_j + 1) \leq 2$$

If this holds $\forall \tau_k \in \tau$ the hole task set is schedulable

Utilization Bounds for RM-NP

- General utilization bounds are not possible
- It is possible to define a utilization bound based on the ratio of the computation time of a task and its blocking time

Theorem

[Theorem 4 in von der Brüeggen, Chen, and Huang, 2015] Suppose that $\gamma = \max_{\tau_k} \left\{ \max_{\tau_i \in lp(\tau_k)} \left\{ \frac{C_i}{C_k} \right\} \right\}$. A task set can be feasibly scheduled by RM-NP if

$$U_{sum} \leq \begin{cases} \frac{\gamma}{1+\gamma} + ln\left(\frac{2}{1+\gamma}\right) & \text{if } \gamma \leq 1 \\ \frac{1}{1+\gamma} & \text{if } \gamma > 1 \end{cases}$$

Utilization Bound for RM-NP

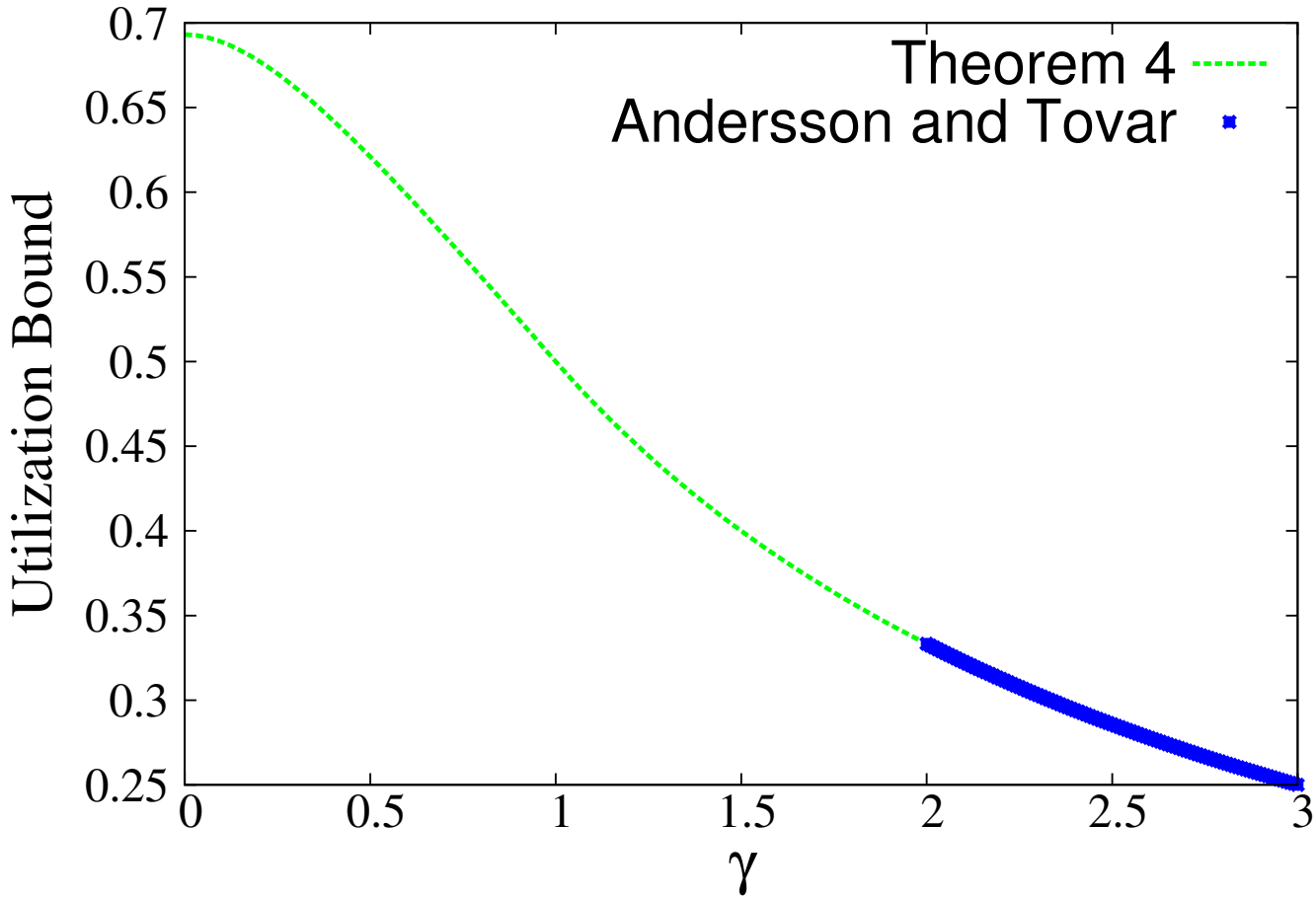


Figure: Comparison of the total utilization bound of RM-NP with respect to $\gamma = \max_{\tau_k} \left\{ \max_{\tau_i \in lp(\tau_k)} \left\{ \frac{C_i}{C_k} \right\} \right\}$ provided by Theorem 4 in [von der Brüeggen, Chen, and Huang, 2015] with previously known results (Andersson and Tovar, 2009).

A Tighter Utilization Based Test

Theorem

[Yao, Buttazzo, and Bertogna, 2010] The worst-case response time of a non-preemptive task occurs in the first job if the task is activated at its critical instant and the following two conditions are both satisfied:

1. the task set is feasible under preemptive scheduling;
2. the relative deadlines are less than or equal to periods.

- Testing preemptive and non-preemptive case schedulability is necessary to restrict testing to the first job
- Theorem 1 performs these in one single test
- We get tighter by doing two separated tests
  1. non-preemptive case:
     $$\exists t \in (0, D_k - C_k] \text{ with } B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$$
  2. preemptive case:
     $$\exists t \in (0, D_k] \text{ with } C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$$

A Tighter Utilization Based Test (contd)

Theorem

[Theorem 6 in von der Brüeggen, Chen, and Huang, 2015]
A task τ_k is schedulable by a fixed priority non-preemptive scheduling algorithm A^{NP} if all higher priority tasks are schedulable and the following two conditions hold:

$$\left(\frac{B_k + \sum_{\tau_i \in hp_2^{NP}(\tau_k)} C_i}{D_k - C_k} + 1 \right) \prod_{\tau_j \in hp_1^{NP}(\tau_k)} (U_j + 1) \leq 2$$

$$\left(\frac{C_k + \sum_{\tau_i \in hp_2^{P}(\tau_k)} C_i}{D_k} + 1 \right) \prod_{\tau_j \in hp_1^{P}(\tau_k)} (U_j + 1) \leq 2$$

A Tighter Utilization Bound for RM-NP

Using Theorem 6 the utilization bound for Rate Monotonic Non-Preemptive Scheduling can be made a bit tighter.

Theorem

[Theorem 9 in von der Brüeggen, Chen, and Huang, 2015]
Suppose that $\gamma = \max_{\tau_k} \left\{ \max_{\tau_i \in lp(\tau_k)} \left\{ \frac{C_i}{C_k} \right\} \right\}$. A task set can be feasibly scheduled by RM-NP if

$$U_{sum} \leq \begin{cases} ln(2) \approx 0.693 & \text{if } \gamma \leq \frac{1-ln(2)}{ln(2)} \\ \frac{1}{1+\gamma} & \text{if } \gamma > \frac{1-ln(2)}{ln(2)} \end{cases}$$

Utilization Bounds for RM-NP

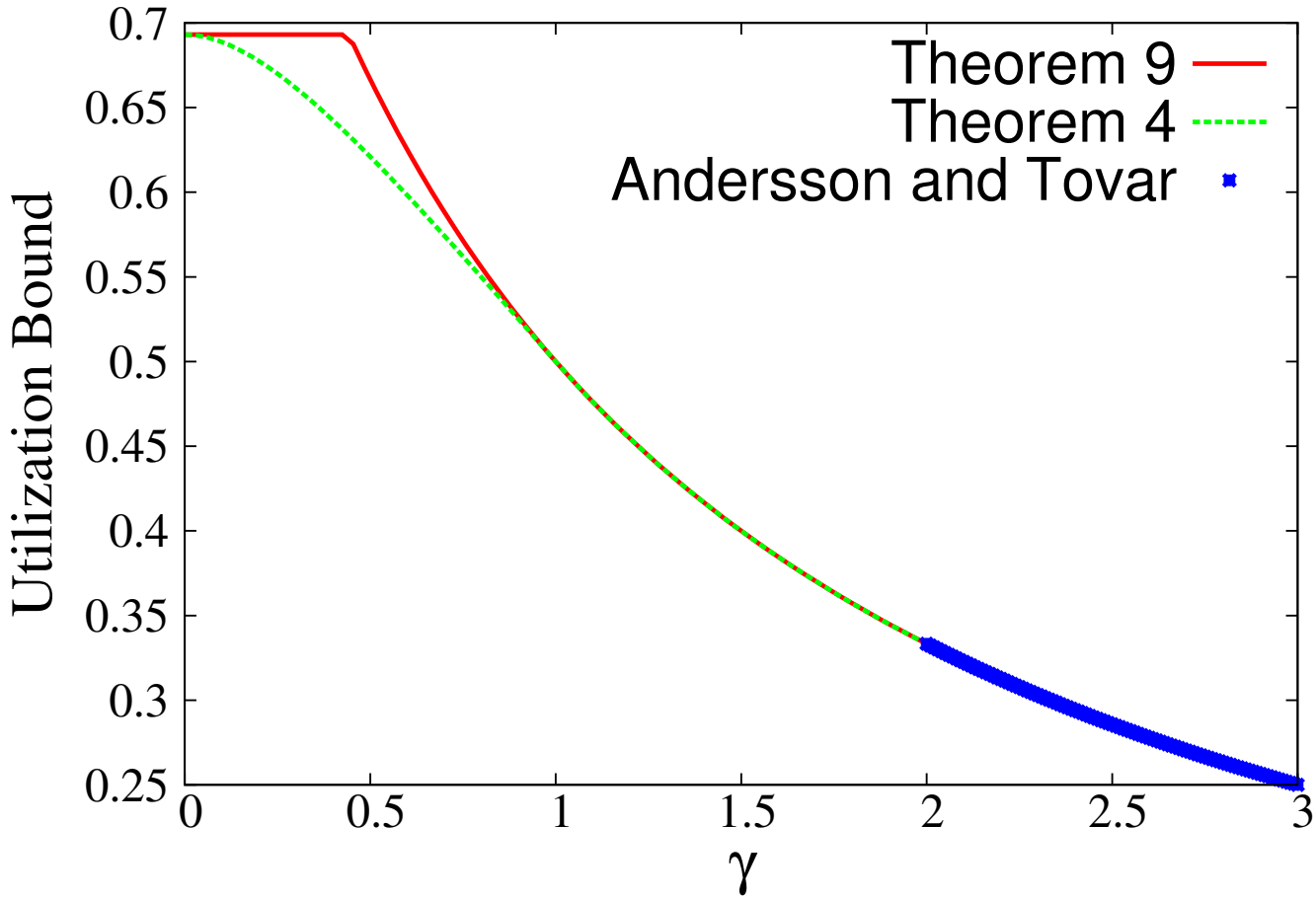


Figure: Comparison of the total utilization bound of RM-NP with respect to $\gamma = \max_{\tau_k} \left\{ \max_{\tau_i \in lp(\tau_k)} \left\{ \frac{C_i}{C_k} \right\} \right\}$ provided by Theorem 4 and Theorem 9 in [von der Brüeggen, Chen, and Huang, 2015] with previously known results (Andersson and Tovar, 2009).

Outline

Types of Limited Preemption

Problem: We want preemption to ensure high priority tasks meet their deadline but we want as few preemptions as possible $\Rightarrow$ Combine the advantages and disadvantages of preemptive and non-preemptive scheduling into limited preemptive scheduling

- **Preemption Thresholds:** Each task can be preempted only by tasks with priority higher than a specified threshold.
- **Deferred Preemptions:** Each task can defer its preemption up to a specified interval of time.
- **Fixed Preemption Points:** Each task can be preempted only at predefined points specified in the code by the programmer.

Preemption Thresholds

- Each task has two priorities
  - P_k: Nominal priority used to enqueue the task in the ready queue and to preempt other tasks
  - Θ_k: threshold priority used while task is execution. τ_k can be preempted by τ_i only if $P_i > \Theta_k$
- Analysis has to be done in the longest Level-k busy period
- Response time analysis has to be done in two phases:
  - Blocking time by tasks $\tau_i \in lp(\tau_k)$ with priority less than τ_k but preemption threshold larger than τ_k combined with Interference of higher priority tasks with $P_i > P_k$ until τ_k starts to determine the start time
    $$S_{k,j} = B_k + (j-1)C_k + \sum_{\tau_i \in hp(\tau_l)} \left(\left\lfloor \frac{S_{k,j}}{T_i} \right\rfloor + 1 \right) C_i$$
  - Preemption of tasks with priority larger than the preemption threshold ($P_h > \Theta_i$) after the task started to determine the finish time
    $$f_{k,j} = s_{k,j} + C_k + \sum_{\tau_i : P_i > \Theta_k} \left(\left\lceil \frac{R_i}{T_i} \right\rceil - \left(\left\lfloor \frac{S_{k,j}}{T_i} \right\rfloor + 1 \right) \right) C_i$$

Deferred Preemptions

- Each task can defer preemption up to q_i if a task τ_k with $P_k > P_i$ wants to preempt $\tau_i \Rightarrow \; B_k = \max_{\tau_i \in lp(\tau_k)} \{q_i\}$
- Interesting problem: given a preemptively feasible task set, find the longest non-preemptive interval Q_i for each task that still preserves schedulability
- High priority tasks often have $Q_i = C_i$, meaning that they can execute fully non preemptively
- To compute Q_i, we need to find the maximum blocking time that can be tolerated by a task, called blocking tolerance β_i
- Q_i can be used to divide a task into non preemptive chunks of length no larger than Q_i
- If all critical regions can be completely included in those non-preemptive chunks the access to shared resources is trivial

Fixed Preemption Points

- Each task τ_i is divided in m_i chunks $q_{i,1}, \ldots, q_{1,m}$
- The task can only be preempted between chunks
- $B_k = \max_{\tau_i \in lp(\tau_k)}\{q_i^{max}\}$
- Analysis must be carried out up the busy period of each task
- Preemption points are assumed to be given by the programmer
- If preemption points are chosen carefully
  - preemption in critical region will not occur
  - preemption overhead due to cache misses can be reduced, e.g. as preemption points will be placed outside loops
  - stack size can be reduced

General Remarks

- **Preemption Thresholds** are easy to specify, but it is difficult to predict the number of preemptions and where they occur
  - still possibly large preemption overhead
- **Deferred Preemption** allows bounding the number of preemptions but it is difficult to predict where they occur
  - number of preemptions bounded
  - overhead per preemption may still be high
- **Fixed Preemption Points** allow more control on preemptions and can be selected on purpose
  - number of preemptions bounded
  - overhead per preemption can be bounded if chosen carefully
  - longest non preemptive interval Q_i can be used to get an upper bound on the length of the non-preemptive chunks
  - preemption in critical region will not occur if the preemption points are chosen carefully