AAAAA

Jian-Jia Chen, Wen-Hung Huang, and Geoffrey Nelissen
TU Dortmund University, Germany
Email: jian-jia.chen@tu-dortmund.de, wen-hung.huang@tu-dortmund.de
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Email: grrpn@isep.ipp.pt

1 Introduction

2 Task Model

This paper presents a proof to support the correctness of the schedulability test for self-suspending real-time task systems proposed by Jane W. S. Liu in her book titled "Real-Time Systems" [3, Pages 164-165]. The same concept was also implicitly used by Rajkumar, Sha, and Lehoczky [6, Page 267] for analyzing self-suspending behaviour due to synchronization protocols in multiprocessor systems.

The system model and terminologies are defined as follows: We assume a system composed of n sporadic self-suspending tasks. A sporadic task τ_i is released repeatedly, with each such invocation called a job. The j^{th} job of τ_i, denoted $\tau_{i,j}$, is released at time $r_{i,j}$ and has an absolute deadline at time $d_{i,j}$. Each job of any task τ_i is assumed to have a worst-case execution time C_i. Each job of task τ_i suspends for at most S_i time units (across all of its suspension phases). When a job suspends itself, the processor can execute another job. The response time of a job is defined as its finishing time minus its release time. Successive jobs of the same task are required to execute in sequence. Associated with each task τ_i are a period (or minimum inter-arrival time) T_i, which specifies the minimum time between two consecutive job releases of τ_i, and a relative deadline D_i, which specifies the maximum amount of time a job can take to complete its execution after its release, i.e., $d_{i,j} = r_{i,j} + D_i$. The worst-case response time R_i of a task τ_i is the maximum response time among all its jobs. The utilization of a task τ_i is defined as $U_i = C_i/T_i$.

In this paper, we focus on constrained-deadline task systems, in which $D_i \leq T_i$ for every task τ_i. We only consider preemptive fixed-priority scheduling on a single processor, in which each task is assigned with a unique priority level. We assume that the priority assignment is given.

We assume that the tasks are numbered in a decreasing priority order. That is, a task with a smaller index has higher priority than any task with a higher index, i.e., task τ_i has a higher-priority level than task τ_{i+1}. When performing the schedulability analysis of a specific task τ_k, we assume that $\tau_1, \tau_2, \ldots, \tau_{k-1}$ are already verified to meet their deadlines, i.e., that $R_i \leq D_i, \forall \tau_i \mid 1 \leq i \leq k-1$.

3 Existing Analyses

To analyze the worst-case response time (or the schedulability) of task τ_k, we usually need to quantify the worst-case interference caused by the higher-priority tasks on the execution of any job of task τ_k. In the ordinary sequential sporadic real-time task model, i.e., when $S_i = 0$ for every task τ_i, the so-called critical instant theorem by Liu and Layland [2] is commonly adopted. That is, the worst-case response time of task τ_k (if it is less than or equal to its period) happens for the first job of task τ_k when τ_k and all the higher-priority tasks release a job synchronously and the subsequent jobs are released as early as possible (i.e., with a rate equal to their period). However, as proven in [4], this definition of the critical instant does not hold for self-suspending sporadic tasks.

3.1 Model the Back-to-Back Hit as Jitter

A constrained-deadline task τ_k can be feasibly scheduled by the fixed-priority scheduling if

$$\exists t \mid 0 < t \leq D_k, \qquad C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + D_i - C_i}{T_i} \right\rceil C_i \leq t. \tag{1}$$

3.2 Model Suspension Time as Blocking Time

In [3, Pages 164-165], Jane W. S. Liu proposed a solution to study the schedulability of self-suspending tasks by modeling the *extra delay* suffered by a task τ_k due to the self-suspending behavior of the tasks as a blocking time denoted as B_k and defined as follows:

- The blocking time contributed from task τ_k is S_k.
- A higher-priority task τ_i can only block the execution of task τ_k by at most $b_i = min(C_i, S_i)$ time units.

Therefore,

$$B_k = S_k + \sum_{i=1}^{k-1} b_i. \tag{2}$$

In [3], the blocking time is then used to derive a utilization-based schedulability test for rate-monotonic scheduling. Namely, it is stated that if $\frac{C_k+B_k}{T_k} + \sum_{i=1}^{k-1} U_i \leq k(2^{\frac{1}{k}} - 1)$, then task τ_k can be feasibly scheduled by using rate-monotonic scheduling if $T_i = D_i$ for every task τ_i in the given task set. If the above argument is correct, we can further prove that a constrained-deadline task τ_k can be feasibly scheduled by the fixed-priority scheduling if

$$\exists t \mid 0 < t \leq D_k, \qquad C_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t. \tag{3}$$

The same concept was also implicitly used by Rajkumar, Sha, and Lehoczky [6, Page 267] for analyzing self-suspending behaviour due to synchronization protocols in multiprocessor systems. To account for the self-suspending behaviour, it reads as follows:[1]

> *For each higher priority job J_i on the processor that suspends on global semaphores or for other reasons, add the term $min(C_i, S_i)$ to B_k, where S_i is the maximum duration that J_i can suspend itself. The sum ... yields B_k, which in turn can be used in $\frac{C_k+B_k}{T_k} + \sum_{i=1}^{k-1} U_i \leq k(2^{\frac{1}{k}} - 1)$ to determine whether the current task allocation to the processor is schedulable.*

However, there is no proof in [3], [6] to support the correctness of the above tests. We will support the correctness of the above analysis by proving a more powerful analysis framework.

4 Our General Analysis Framework

We can greedily convert the suspension time of task τ_k to its computation time. For the sake of notational brevity, let C'_k be $C_k + S_k$. We call this converted version of task τ_k as task τ'_k. Suppose that R'_k is the worst-case response time in the task system $\{\tau_1, \tau_2, \ldots, \tau_{k-1}, \tau'_k\}$. It was already shown in the previous works, e.g., Lemma 3 in [1] and Theorem 2 in [4], that R'_k is a safe upper bound on the worst-case response time of task τ_k in the original task system.

Our key result in this paper is the following theorem:

Theorem 1. *Suppose that $R'_k \leq T_k$. For any arbitrary vector assignment $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$, in which x_i is either 0 or 1, the worst-case response time R'_k is upper bounded by the minimum t (with $t > 0$) that satisfies*

$$C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq t, \tag{4}$$

where $Q_i^{\vec{x}}$ is $\sum_{j=i}^{k-1} S_j \cdot x_j$.

With Theorem 1, we can directly have the following corollary.

Corollary 1. *If there exists a vector assignment $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$, in which x_i is either 0 or 1, such that*

$$\exists t | 0 < t \leq D_k, C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq t, \tag{5}$$

where $Q_i^{\vec{x}}$ is $\sum_{j=i}^{k-1} S_j \cdot x_j$, then a constrained-deadline task τ_k can be feasibly scheduled by the fixed-priority scheduling.

4.1 An Illustrative Example and Dominance

We use an example to demonstrate how Corollary 1 can be applied. Suppose that we have three tasks

- $C_1 = 4, S_1 = 5, T_1 = D_1 = 10$,
- $C_2 = 6, S_2 = 1, T_2 = D_2 = 19$, and
- $C_3 = 4, S_3 = 0, T_3 = D_3 = 35$.

Tasks τ_1 and τ_2 can be verified to be schedulable under the fixed-priority scheduling by using Eq. (1).

[1] We rephrased the wordings and notations to be consistent with this paper.

We focus on task τ_3. For task τ_3, the blocking term B_3 is $4 + 1 = 5$ by Eq. (2). The minimum t to satisfy $C_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$ happens when $t = 37$, i.e., $4+5+\left\lceil \frac{37}{10} \right\rceil \cdot 4 + \left\lceil \frac{37}{19} \right\rceil \cdot 6 = 37$. Therefore, task τ_3 cannot pass the schedulability test in Eq. (3). There are four possible vector assignments $\vec{x}$ when we consider the schedulability of task τ_3:

- Case 1 $\vec{x} = (0, 0)$: In this case, Theorem 1 states that R'_k is upper bounded by the minimum t under $0 < t \leq T_3$ that satisfies
$$4 + \left\lceil \frac{t + 6}{10} \right\rceil \cdot 4 + \left\lceil \frac{t + 13}{19} \right\rceil \cdot 6 \leq t. \tag{6}$$
Such a value t does not exist for this case.
- Case 2 $\vec{x} = (0, 1)$: In this case, Theorem 1 states that R'_k is upper bounded by the minimum t under $0 < t \leq T_3$ that satisfies
$$4 + \left\lceil \frac{t + 7}{10} \right\rceil \cdot 4 + \left\lceil \frac{t + 1}{19} \right\rceil \cdot 6 \leq t. \tag{7}$$
Therefore, $R'_k \leq 32$ due to $4+\left\lceil \frac{32+7}{10} \right\rceil \cdot 4 + \left\lceil \frac{32+1}{19} \right\rceil \cdot 6 = 32$.
- Case 3 $\vec{x} = (1, 0)$: In this case, Theorem 1 states that R'_k is upper bounded by the minimum t under $0 < t \leq T_3$ that satisfies
$$4 + \left\lceil \frac{t + 5}{10} \right\rceil \cdot 4 + \left\lceil \frac{t + 13}{19} \right\rceil \cdot 6 \leq t. \tag{8}$$
Such a value t does not exist for this case.
- Case 4 $\vec{x} = (1, 1)$: In this case, Theorem 1 states that R'_k is upper bounded by the minimum t under $0 < t \leq T_3$ that satisfies
$$4 + \left\lceil \frac{t + 6}{10} \right\rceil \cdot 4 + \left\lceil \frac{t + 1}{19} \right\rceil \cdot 6 \leq t. \tag{9}$$
Therefore, $R'_k \leq 32$ due to $4+\left\lceil \frac{32+6}{10} \right\rceil \cdot 4 + \left\lceil \frac{32+1}{19} \right\rceil \cdot 6 = 32$.

Among the above four cases, the test in Case 4, i.e., Eq. (9), is the tightest. By Corollary 1, task τ_3 is schedulable by the fixed-priority scheduling policy.

In fact, the following theorem shows that the test in Corollary 1 analytically dominates the existing tests in Eq. (1) and Eq. (3).

Theorem 2. *The schedulability test in Corollary 1 dominates the schedulability tests in Eq. (1) and Eq. (3).*

Proof: The dominance of Eq. (1) can be easily seen by considering the vector assignment $x_1 = x_2 = \cdots = x_{k-1} = 0$. The resulting test in Eq. (5) is identical to Eq. (3) for this vector assignment.

We now prove the dominance of Eq. (3) by considering the vector assignment $\vec{x}$ in which

$$x_i = \begin{cases} 1 & \text{if } S_i \leq C_i \\ 0 & \text{otherwise,} \end{cases}$$

for $i = 1, 2, \ldots, k-1$. By the fact that $Q_i^{\vec{x}} \leq Q_1^{\vec{x}}$ for $i = 1, 2, \ldots, k-1$, we know that it is more pessimistic if we test $C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{t+Q_1^{\vec{x}}+(1-x_i)(D_i-C_i)}{T_i} \right\rceil C_i \leq t$ instead of testing Eq. (5). Let θ be $t + Q_1^{\vec{x}}$. Therefore, we know that R'_k is upper bounded by the minimum $\theta - Q_1^{\vec{x}} > 0$ such that

$$C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{\theta + (1-x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq \theta - Q_1^{\vec{x}} \quad (10)$$

$$\Rightarrow\; C'_k + Q_1^{\vec{x}} + \sum_{i=1}^{k-1} \left\lceil \frac{\theta + (1-x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq \theta. \quad (11)$$

Moreover, by the fact that $D_i \leq T_i$ for $i = 1, 2, \ldots, k-1$, we also have $\left\lceil \frac{\theta+(1-x_i)(D_i-C_i)}{T_i} \right\rceil C_i \leq \left\lceil \frac{\theta+(1-x_i)T_i}{T_i} \right\rceil C_i = (1 - x_i)C_i + \left\lceil \frac{\theta}{T_i} \right\rceil C_i$. Therefore, we know that R'_k is upper bounded by the minimum $\theta - Q_1^{\vec{x}} > 0$ such that

$$C_k + S_k + \sum_{i=1}^{k-1}(x_i S_i + (1-x_i)C_i) + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil C_i \leq \theta. \quad (12)$$

By the fact that B_k is defined as $S_k + \sum_{i=1}^{k-1}(x_i S_i + (1-x_i)C_i)$, and $Q_1^{\vec{x}} \geq 0$, the above test in Eq. (12) is analytically tighter than that in Eq. (3), which concludes the proof. ■

4.2 Proof of Theorem 1

We now provide the proof to support the correctness of the test in Theorem 1. Our proof strategy is to show that the worst-case response time of task τ_k can be safely upper-bounded by any assignment of $\vec{x}$ of the $k-1$ higher-priority tasks when adopting Eq. (4) as the response time analysis.

Throughout the proof, we consider any arbitrary assignment $\vec{x}$. For the sake of notational brevity, we classify the $k-1$ higher-priority tasks into two sets: $\mathbf{T}_0$ and $\mathbf{T}_1$. A task τ_i is in $\mathbf{T}_0$ if x_i is 0; otherwise, it is in $\mathbf{T}_1$.

Our analysis is also based on very simple properties and lemmas enunciated as follows:

Property 1. *In a preemptive fixed-priority schedule, the lower-priority jobs do not impact the schedule of the higher-priority jobs.*

Lemma 1. *In a preemptive fixed-priority schedule, if the worst-case response time of task τ_i is no more than its period T_i, preventing the release of a job of task τ_i does not affect the schedule of any other job of task τ_i.*

Proof: Since the worst-case response time of task τ_i is no more than its period, any job $\tau_{i,j}$ of task τ_i completes its execution before the release of the next job $\tau_{i,j+1}$. Hence, the execution of $\tau_{i,j}$ does not directly interfere with the execution of any other job of τ_i, which then depends only on the schedule of the higher priority jobs. Furthermore, as stated in Property 1, the removal of $\tau_{i,j}$ has no impact on the schedule of the higher-priority jobs, thereby implying that the other jobs of task τ_i are not affected by the removal of $\tau_{i,j}$. ■

With the above properties, we now present the proof of Theorem 1 as follows:

Proof of Theorem 1. Let us consider the task set τ' composed of $\{\tau_1, \tau_2, \ldots, \tau_{k-1}, \tau'_k, \tau_{k+1}, \ldots\}$ and let Ψ be a schedule of τ' that generates the worst-case response time of τ'_k, in which $R'_k \leq T_k$ by our assumption. The proof is built upon the two following steps:

1) We discard all the jobs that do not contribute to the worst-case response time of τ'_k in the schedule Ψ. We follow an inductive strategy by iteratively inspecting the schedule of the higher priority tasks in Ψ, starting with τ_{k-1} until the highest priority task τ_1. At each iteration, a time instant t_j is identified such that $t_j \leq t_{j+1}$ ($1 \leq j < k$). Then, all the jobs of task τ_j released before t_j are removed from the schedule and, if needed, replaced by an artificial job mimicking the interference caused by the residual workload of task τ_j at time t_j on the worst-case response time of τ'_k.
2) The final reduced schedule is analyzed so as to characterize the worst-case response time of τ'_k in Ψ. We then prove that the response time analysis in Eq. (4) is indeed an upper bound on the worst-case response time R'_k of τ'_k.

Step 1: Reducing the schedule Ψ

During this step, we iteratively build an artificial schedule Ψ^j from Ψ^{j+1} (with $1 \leq j < k$) so that the response time of τ'_k remains identical. At each iteration, we define t_j for task τ_j in the schedule Ψ^{j+1} (with $j = k-1, k-2, \ldots, 1$) and build Ψ^j by removing all the jobs released by τ_j before t_j.

Basic step (definition of Ψ^k and t_k):

Suppose that the job J_k of task τ'_k with the largest response time in Ψ arrives at time r_k and finishes at time f_k. We know by Property 1 that the lower priority tasks $\tau_{k+1}, \tau_{k+2}, \ldots, \tau_n$ do not impact the response time of J_k. Moreover, since we assume that the worst-case response time of task τ'_k is no more than T_k, Lemma 1 proves that removing all the jobs of task τ'_k but J_k has no impact on the schedule of J_k. Therefore, let Ψ^k be a schedule identical to Ψ but removing all the jobs released by the lower priority tasks $\tau_{k+1}, \ldots, \tau_n$ as well as all the jobs released by τ'_k at the exception of J_k. The response time of J_k in Ψ^k is thus identical to the response time of J_k in Ψ.

We define t_k as the release time of J_k (i.e., $t_k = r_k$).

Induction step (definition of Ψ^j and t_j with $1 \leq j < k$):

Let r_j be the arrival time of the last job released by τ_j before t_{j+1} in Ψ^{j+1} and let J_j denote that job. Removing all the jobs of task τ_j arrived before r_j has no impact on the schedule of any other job released by τ_j (Lemma 1) or any higher priority job released by $\tau_1, \ldots, \tau_{j-1}$ (Property 1). Moreover, because by the construction of Ψ^{j+1}, no task with a priority lower than τ_j executes jobs before t_{j+1} in Ψ^{j+1}, removing the jobs released by τ_j before t_{j+1} does not impact the schedule of the jobs of $\tau_{j+1}, \ldots, \tau_k$. Therefore, we can safely remove all the jobs of task τ_j arrived before r_j without impacting the response time of J_k. Two cases must then be considered:

(a) $\tau_j \in \mathbf{T}_0$. In this case, we analyze two different subcases:
- J_j completed its execution before or at t_{j+1}. By Lemma 1 and Property 1, removing all the jobs of task τ_j arrived before t_{j+1} has no impact on the schedule of the higher-priority jobs (jobs released by $\tau_1, \ldots, \tau_{j-1}$) and the jobs of τ_j released after or at t_{j+1}. Moreover,

because no task with lower priority than τ_j executes jobs before t_{j+1} in Ψ^{j+1}, removing the jobs released by τ_j before t_{j+1} does not impact the schedule of the jobs of $\tau_{j+1}, \ldots, \tau_k$. Therefore, t_j is set to t_{j+1} and Ψ^j is generated by removing all the jobs of task τ_j arrived before t_{j+1}. The response time of J_k in Ψ^j thus remains unchanged in comparison to its response time in Ψ^{j+1}.

- J_j did not complete its execution by t_{j+1}. For such a case, t_j is set to r_j and hence Ψ^j is built from Ψ^{j+1} by removing all the jobs released by τ_j before r_j.

Note that because by the construction of Ψ^{j+1} and hence Ψ^j there is no job with priority lower than τ_j available to be executed before t_{j+1}, the maximum amount of time during which the processor remains idle within $[t_j, t_{j+1})$ is at most S_j time units.

(b) $\tau_j \in \mathbf{T}_1$. For such a case, we set t_j to t_{j+1}. Let c_j^* be the remaining execution time for the job of task τ_j at time t_j. We know that c_j^* is at most C_j. Since by the construction of Ψ^j, all the jobs of τ_j released before t_j are removed, the job of task τ_j arrived at time r_j ($< t_j$) is replaced by a new job released at time t_j with execution time c_j^* and the same priority than τ_j. Clearly, this has no impact on the execution of any job executed after t_j and thus on the response time of J_k. The remaining execution time c_j^* of τ_j at time t_j is called the *residual workload* of task τ_j for the rest of the proof.

The above construction of $\Psi^{k-1}, \Psi^{k-2}, \ldots, \Psi^1$ is repeated until producing Ψ^1. The procedures are well-defined. Therefore, it is guaranteed that Ψ^1 can be constructed. Note that after each iteration, the number of jobs considered in the schedule have been reduced, yet without affecting the response time of J_k.

Step 2: Analyzing the final reduced schedule Ψ^1

We now analyze the properties of the final schedule Ψ^1 in which all the unnecessary jobs have been removed. The proof is based on the fact that for any interval $[t_1, t)$, there is

$$\mathrm{idle}(t_1, t) + \mathrm{exec}(t_1, t) = (t - t_1) \tag{13}$$

where $\mathrm{exec}(t_1, t)$ is the amount of time during which the processor executed tasks within $[t_1, t)$, and $\mathrm{idle}(t_1, t)$ is the amount of time during which the processor remained idle within the interval $[t_1, t)$.

From case (a) of Step 1, we know that the maximum amount of time during which the processor is idle in Ψ^1 within any time interval $[t_1, t)$ such that $t_1 < t \leq t_k$, is upper bounded by

$$\mathrm{idle}(t_1, t) \leq \sum_{\tau_i : t_i < t \text{ and } \tau_i \in \mathbf{T}_1} S_i.$$

Because there is no job released by lower priority tasks than τ_k in Ψ^1, the workload released by $\tau_1, \ldots, \tau_k$ within any interval $[t_1, t)$ is an upper bound on the workload $\mathrm{exec}(t_1, t)$ executed within $[t_1, t)$. From case (a) of Step 1, we know that the maximum workload released by task τ_j in $\mathbf{T}_1$ within any time interval $[t_1, t)$ in schedule Ψ^1 such that $t_j \leq t \leq f_k$ is upper bounded by $\left\lceil \frac{t - t_j}{T_j} \right\rceil C_j$.

From case (b) of Step 1, if the residual workload c_j^* of task τ_j in $\mathbf{T}_0$ is positive, the earliest arrival time of task τ_j arriving *strictly after* t_j is at least $t_j + (T_j - D_j + c_j^*)$. Therefore, considering the fact that no job of τ_j is released before t_j in Ψ^1 ($j = 1, 2, \ldots, k$), the workload released by task τ_j (by treating the residual workload as released workload as well) in $\mathbf{T}_0$ within any time interval $[t_1, t)$ in schedule Ψ^1 such that $t_1 < t \leq f_k$ is upper bounded by $rbf_j(t_j, t)$ defined as follows:

$$rbf_j(t_j, t) = \begin{cases} c_j^* + \left\lceil \frac{t - t_j - (D_j - c_j^*)}{T_j} \right\rceil C_j & \text{if } t > t_j \\ 0 & \text{otherwise.} \end{cases}$$

Putting the execution time from the tasks in $\mathbf{T}_0$ and $\mathbf{T}_1$ together, we have $\forall t \mid t_1 \leq t < t_i$

$$\mathrm{exec}(t_1, t) \leq \sum_{j=1}^{i-1} x_j \left\lceil \frac{t - t_j}{T_j} \right\rceil C_j + (1 - x_j) rbf_j(t_j, t) \tag{14}$$

jj: here!!! : endjj

$$\forall t \mid t_1 \leq t < f_k, \quad \mathrm{exec}(t_1, t) \leq \sum_{\tau_i \in \mathbf{T}_1} C_i + \sum_{i=1}^{k} \max\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i. \tag{15}$$

Furthermore, from case (a) of Step 1, we know that the maximum amount of time during which the processor is idle in Ψ^1 within any time interval $[t_1, t)$ such that $t_1 < t \leq t_k$, is upper bounded by $\sum_{\tau_i \in \mathbf{T}_0} S_i$. That is,

$$\forall t \mid t_1 \leq t < t_k, \quad \mathrm{idle}(t_1, t) \leq \sum_{\tau_i \in \mathbf{T}_0} S_i. \tag{16}$$

Hence, injecting Eq. (15) and Eq. (16) into Eq. (13), we get

$$\forall t \mid t_1 \leq t < t_k, \qquad \sum_{\tau_i \in \mathbf{T}_0} S_i + \sum_{\tau_i \in \mathbf{T}_1} C_i + \sum_{i=1}^{k} \max\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\} \geq t$$

Since $C'_k > 0$ and $\max\{0, \left\lceil \frac{t - t_k}{T_k} \right\rceil C'_k\} = 0$ for any t smaller than t_k, it holds that

$$\forall t \mid t_1 \leq t < t_k, \qquad \sum_{\tau_i \in \mathbf{T}_0} S_i + \sum_{\tau_i \in \mathbf{T}_1} C_i + C'_k + \sum_{i=1}^{k-1} \max\left\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\right.$$

and using the definition of b_i

$$\forall t \mid t_1 \leq t < t_k, \qquad C'_k + \sum_{i=1}^{k-1} b_i + \sum_{i=1}^{k-1} \max\left\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\right\} > t - t_1. \tag{17}$$

Furthermore, because J_k is released at time t_k and does not complete its execution before f_k, it must hold that

$$\forall t \mid t_k \leq t < f_k, \qquad C'_k + \sum_{i=1}^{k-1} b_i + \sum_{i=1}^{k-1} \max\left\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\right\} > t - t_1. \tag{18}$$

Combining Eq. (17) and Eq. (18), we get

$$\forall t \mid t_1 \leq t < f_k, \qquad C'_k+\sum_{i=1}^{k-1} b_i+\sum_{i=1}^{k-1} \max\left\{0, \left\lceil \frac{t-t_i}{T_i} \right\rceil C_i\right\} > t-t_1. \tag{19}$$

Since $t_i \geq t_1$ for $i = 1, 2, \ldots, k$, there is

$$\left\lceil \frac{t-t_i}{T_i} \right\rceil \leq \left\lceil \frac{t-t_1}{T_i} \right\rceil,$$

thereby leading to

$$\forall t \mid t_1 \leq t < f_k, \qquad C'_k+\sum_{i=1}^{k-1} b_i+\sum_{i=1}^{k-1} \max\left\{0, \left\lceil \frac{t-t_1}{T_i} \right\rceil C_i\right\} > t-t_1. \tag{20}$$

By replacing $t - t_1$ with θ, Eq. (20) becomes[2]

$$\forall \theta \mid 0 < \theta < f_k - t_1, \qquad C'_k + \sum_{i=1}^{k-1} b_i + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil C_i > \theta.$$

The above inequation implies that the minimum t such that $C'_k + \sum_{i=1}^{k-1} b_i + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$ is larger than or equal to $f_k - t_1$. And because by assumption the worst-case response time of τ'_k is equal to $f_k - t_k \leq f_k - t_1$ which is obviously smaller than or equal to R'_k, it holds that R'_k is a safe upper bound on the worst-case response time of τ'_k. □

To illustrate Step 1 in the above proof, we also provide one concrete example. Consider a task system with the following 4 tasks:

- $T_1 = 6, C_1 = 1, S_1 = 1,$
- $T_2 = 10, C_2 = 1, S_2 = 6,$
- $T_3 = 18, C_3 = 4, S_3 = 1,$
- $T_4 = 20, C_4 = 5, S_4 = 0.$

Figure 1 demonstrates a schedule for the jobs of the above 4 tasks. We assume that the first job of task τ_1 arrives at time $4+\epsilon$ with a very small $\epsilon > 0$. The first job of task τ_2 suspends itself from time 0 to time $5+\epsilon$, and is blocked by task τ_1 from time $5 + \epsilon$ to time $6 + \epsilon$. After some very short computation with ϵ amount of time, the first job of task τ_2 suspends itself again from time $6+2\epsilon$ to 7. In this schedule, f_k is set to $20-\epsilon$.

We define t_4 as 7. Then, we set t_3 to 6. When considering task τ_2, since it belongs to $\mathbf{T}_1$, we greedily set t_2 to $t_3 = 6$ and the residual workload C'_2 is 1. Then, t_1 is set to $4 + \epsilon$. In the above schedule, the idle time from $4 + \epsilon$ to $20 - \epsilon$ is at most $2 = S_1 + S_3$. We have to further consider one job of task τ_2 arrived before time t_1 with execution time C_2.

For the simplicity of presentation, we assume that Ψ is a schedule of τ' that generates the worst-case response time of τ'_k in the proof of Theorem 1. This can be relaxed to start from an arbitrary job J_k in any fixed-priority schedule by using the same proof flow with similar arguments.

5 Improved Analysis

5.1 Direct Implication

We can enumerate all possible configurations of $\vec{x}$.

[2] We take $0 < \theta$ instead of $0 \leq \theta$ since C'_k is assumed to be positive.

5.2 Utilization Bounds

Suppose that $S_i \leq \gamma C_i$ for every task $\tau_i \in hp(\tau_k)$. We will present the utilization bounds in this subsection.

a) Simple Test: We start from the analysis by Liu, which considers the self-suspension time as blocking time for such cases. By using the k2U framework, task τ_k in an implicit deadline system is schedulable by using RM scheduling if

$$(\frac{C_k + S_k}{T_k} + 1 + \gamma) \prod_{i=1}^{k-1} (1 + U_i) \leq 2 + \gamma.$$

That is, $0 < \alpha_i \leq 1+\gamma$ and $0 < \beta_i \leq 1$ for $i = 1, 2, \ldots, k-1$. This gives the immediate utilization bound to find the infimum $\sum_{i=1}^{k} U_k$ such that

$$\begin{aligned} &(1 + \gamma) * (1 + U_k) \prod_{i=1}^{k-1} (1 + U_i) \\ &\geq (\frac{C_k + S_k}{T_k} + 1 + \gamma) \prod_{i=1}^{k-1} (1 + U_i) > 2 + \gamma. \\ &\Rightarrow \prod_{i=1}^{k} (1 + U_i) > \frac{2 + \gamma}{1 + \gamma}. \end{aligned}$$

Therefore, the utilization bound for a given $0 \leq \gamma \leq 1$ is $\ln(\frac{2+\gamma}{1+\gamma})$.

b) Tighter Analysis: This is much more involved and requires deeper knowledge of the k2U framework. Now, suppose that $\sum_{i=1}^{k-1} C_i =$

6 Conclusion

Acknowledgement: This paper is supported by DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/).

References

[1] C. Liu and J.-J. Chen. Bursty-Interference Analysis Techniques for Analyzing Complex Real-Time Task Models. In *2014 IEEE Real-Time Systems Symposium*. Institute of Electrical & Electronics Engineers (IEEE), dec 2014.

[2] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, jan 1973.

[3] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[4] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2015.

[5] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Proceedings.,10th International Conference on Distributed Computing Systems*. Institute of Electrical & Electronics Engineers (IEEE), 1990.

[6] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*, pages 259–269, 1988.

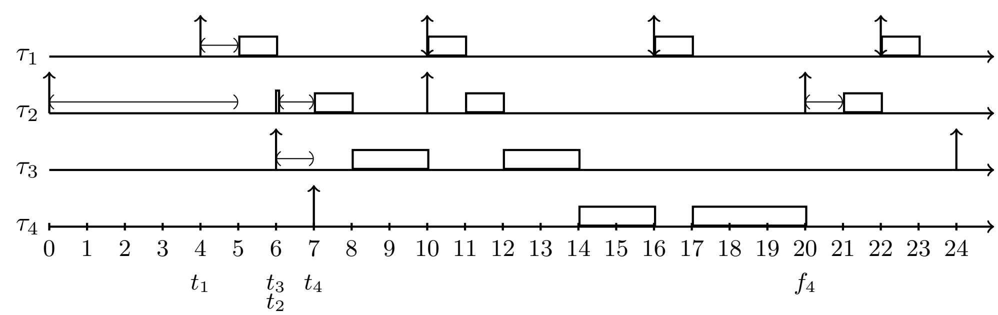

Fig. 1: An illustrative example of Step 1 in the proof of Theorem 1.