A Note to Consider Self-Suspending as Blocking in Real-Time Systems*

Jian-Jia Chen[1], Wen-Hung Huang[1], and Geoffrey Nelissen[2]

[1] TU Dortmund University, Germany
Email: jian-jia.chen@tu-dortmund.de, wen-hung.huang@tu-dortmund.de
[2] CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Email: grrpn@isep.ipp.pt

1 Introduction

This report 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].

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 report, 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 $R_i \leq D_i, \forall \tau_i \mid 1 \leq i \leq k-1$. We also classify the $k-1$ higher-priority tasks into two sets: $\mathbf{T}_1$ and $\mathbf{T}_2$. A task τ_i is in $\mathbf{T}_1$ if $C_i \geq S_i$; otherwise, it is in $\mathbf{T}_2$.

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

2 Liu's Analysis

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. In [3], Jane W. S. Liu proposes 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,

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

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

$$\exists t \mid 0 < t \leq D_k, \qquad C_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t. \tag{2}$$

However, as there is no proof in [3] to support the correctness of the above tests, we present a proof in the next section of this report.

3 Proof of Liu's Analysis

This section provides the proof to support the correctness of the test in Eq. (2). First, it should be easy to see that we can convert the suspension time of task τ_k into computation. This has been proven in many previous works, e.g., Lemma 3 in [1] and Theorem 2 in [4]. Yet, it remains to formally prove that the additional interference due to the self-suspension of a higher-priority task τ_i

is upper-bounded by $b_i = min(C_i, S_i)$. The interference to be at most C_i has been provided in the literature as well, e.g., [5,1]. However, the argument about blocking task τ_k due to a higher-priority task τ_i by at most S_i amount of time is not straightforward.

From the above discussions, 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. 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, removing 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 $\tau_{i,j}$. □

We can prove the correctness of (2) by using a similar proof than for the critical instant theorem of the ordinary sporadic task model. Let R'_k be the minimum t greater than 0 such that (2) holds, i.e., $C'_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i = t$. The following lemma shows that R'_k is a safe upper bound if the worst-case response time of task τ'_k is no more than T_k.

Theorem 1. *R'_k is a safe upper bound on the worst-case response time of task τ'_k if the worst-case response time of τ'_k is not larger than T_k.*

Proof. 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 $\mathcal{S}$ be a schedule of τ' that generates the worst-case response time of τ'_k.

The proof is built upon the two following steps:

1. We discard all the jobs that do not contribute to the worst-case execution time of τ'_k in the schedule S. We follow an inductive strategy by iteratively inspecting the schedule of the higher priority tasks in $\mathcal{S}$, 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 $\mathcal{S}$. We then prove that the R'_k is indeed an upper bound on the worst-case response time of τ'_k.

Step 1: Reducing the schedule $\mathcal{S}$

During this step, we iteratively build an artificial schedule $\mathcal{S}^j$ from $\mathcal{S}^{j+1}$ (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 $\mathcal{S}^{j+1}$ (with $j = k-1, k-2, \ldots, 1$) and build $\mathcal{S}^j$ by removing all the jobs released by τ_j before t_j.

Basic step (definition of $\mathcal{S}^k$ and t_k):

Suppose that the job J_k of task τ'_k with the largest response time in $\mathcal{S}$ 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 $\mathcal{S}^k$ be a schedule identical to $\mathcal{S}$ 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 $\mathcal{S}^k$ is thus identical to the response time of J_k in $\mathcal{S}$.

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

Induction step (definition of $\mathcal{S}^j$ 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 $\mathcal{S}^{j+1}$ 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 construction of $\mathcal{S}^{j+1}$, no task with a priority lower than τ_j executes jobs before t_{j+1} in $\mathcal{S}^{j+1}$, 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) $\tau_j \in \mathbf{T}_1$, i.e., $S_j < C_j$. For such a case, t_j is set to r_j and hence $\mathcal{S}^j$ is built from $\mathcal{S}^{j+1}$ by removing all the jobs released by τ_j before r_j. Note that because by construction of $\mathcal{S}^{j+1}$ and hence $\mathcal{S}^j$ 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) $\tau_j \in \mathbf{T}_2$, i.e., $S_j \geq C_j$. For such a case, we set t_j to t_{j+1}. Let $c_j(t_j)$ be the remaining execution time for the job of task τ_j at time t_j. We know that $c_j(t_j)$ is at most C_j. Since by construction of $\mathcal{S}^j$, 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 $c_j(t_j)$ 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 $c_j(t_j)$ of τ_j at time t_j is called the *residual workload* of task τ_j in the rest of the proof.

The above construction of $\mathcal{S}^{k-1}, \mathcal{S}^{k-2}, \ldots, \mathcal{S}^1$ is repeated until producing $\mathcal{S}^1$. 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 $\mathcal{S}^1$

We now analyze the properties of the final schedule $\mathcal{S}^1$ in which all the unnecessary jobs have been removed.

From case (b) of Step 1, the total residual workload that must be considered in $\mathcal{S}^1$ is upper bounded by $\sum_{\tau_i \in \mathbf{T}_2} C_i$. Therefore, considering the fact that no job of τ_j is released before t_j in $\mathcal{S}^1$ ($j = 1, 2, \ldots, k$), the workload released by the tasks within any time interval $[t_1, t)$ such that $t_1 < t \leq f_k$ is upper bounded by

$$\sum_{i=1}^{k} \left(c_j(t_j) + \max\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\} \right) \leq \sum_{\tau_i \in \mathbf{T}_2} C_i + \sum_{i=1}^{k} \max\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\}$$

Furthermore, from case (a) of Step 1, we know that the maximum amount of time during which the processor is idle in $\mathcal{S}^1$ within any time interval $[t_1, t)$ such that $t_1 < t \leq t_k$, is upper bounded by $\sum_{\tau_i \in \mathbf{T}_1} S_i$. Hence, adding that time to the maximum workload executed by the tasks within any time interval $[t_1, t)$ such that $t_1 < t \leq t_k$, it holds that

$$\forall t \mid t_1 \leq t < t_k, \qquad \sum_{\tau_i \in \mathbf{T}_1} S_i + \sum_{\tau_i \in \mathbf{T}_2} C_i + \sum_{i=1}^{k} \max\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\} \geq t - t_1$$

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, we get that

$$\forall t \mid t_1 \leq t < t_k, \qquad \sum_{\tau_i \in \mathbf{T}_1} S_i + \sum_{\tau_i \in \mathbf{T}_2} C_i + C'_k + \sum_{i=1}^{k-1} \max\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\} > t - t_1$$

and using the definition of b_i

$$\forall t \mid t_1 \leq t < t_k, \qquad C'_k + \sum_{i=1}^{k-1} b_i + \sum_{i=1}^{k-1} \max\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\} > t - t_1 \tag{3}$$

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

$$\forall t \mid t_k \leq t < f_k, \qquad C'_k + \sum_{i=1}^{k-1} b_i + \sum_{i=1}^{k-1} \max\{0, \left\lceil \frac{t - t_i}{T_i} \right\rceil C_i\} > t - t_1. \tag{4}$$

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

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

and without any loss of generality, by arbitrarily assuming $t_1 = 0$, (3) and (4) become

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

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. And because by assumption the worst-case response time of τ'_k is equal to $f_k - t_k$ which is obviously smaller than or equal to f_k, it holds that R'_k is a safe upper bound on the worst-case response time of τ'_k. □

Corollary 1 *R'_k is a safe upper bound on the worst-case response time of task τ'_k if R'_k is not larger than T_k.*

Proof. Directly follows from Theorem 1. □

Corollary 2 *R'_k is a safe upper bound on the worst-case response time of task τ_k if R'_k is not larger than T_k.*

Proof. Since, as proven in [5,1], the worst-case response time of τ'_k is always larger than or equal to the worst-case response time of τ_k, this corollary directly follows from Corollary 1. □

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 $\mathbf{T}_2$, 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.

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.

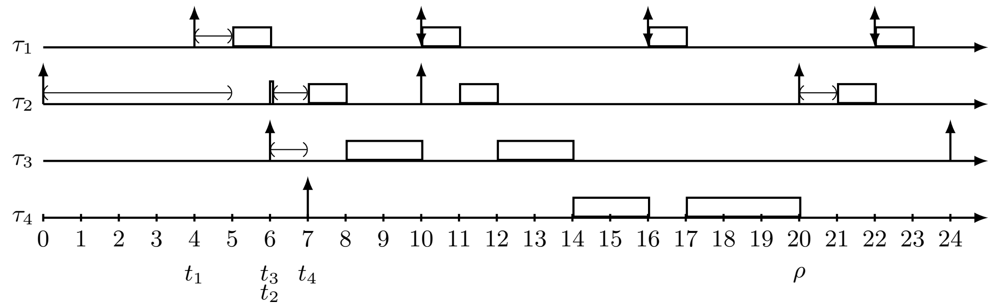

Fig. 1. An illustrative example of the proof.

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.