Suspending Behaviour in Real-Time Embedded Systems

Prof. Dr. Jian-Jia Chen

TU Dortmund

24,05,2016
Introduction

Suspension Models

Dynamic Suspending Task Model

Segmented Suspending Task Model

Conclusion
Outline

Introduction

Suspension Models

Dynamic Suspending Task Model

Segmented Suspending Task Model

Conclusion
Optimality of RM/DM and EDF

• For uniprocessor scheduling, if there exists a feasible schedule for ordinary sporadic real-time tasks, scheduling jobs by using EDF is also feasible.
  • EDF scheduling algorithm is optimal
Optimality of RM/DM and EDF

• For uniprocessor scheduling, if there exists a feasible schedule for ordinary sporadic real-time tasks, scheduling jobs by using EDF is also feasible.
  • EDF scheduling algorithm is optimal
• RM scheduling algorithm is optimal for fixed-priority scheduling when we consider implicit-deadline (i.e., \( T_i = D_i \)) ordinary sporadic tasks
For uniprocessor scheduling, if there exists a feasible schedule for ordinary sporadic real-time tasks, scheduling jobs by using EDF is also feasible.

- EDF scheduling algorithm is optimal

RM scheduling algorithm is optimal for fixed-priority scheduling when we consider implicit-deadline (i.e., $T_i = D_i$) ordinary sporadic tasks.

**Time Demand Analysis (TDA):** Task $\tau_k$ (with $D_i = T_i$) can be feasibly scheduled by a fixed-priority scheduling algorithm if

$$\exists t \text{ with } 0 < t \leq T_k \text{ and } C_k + \sum_{j=1}^{k-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t.$$ 

(This talk will implicitly assume $k - 1$ higher-priority tasks.)
Reasons for Suspension: Hardware Acceleration

Use FPGA in parallel (suspension aware).

Not use FPGA in parallel (busy waiting).
Reasons for Suspension: Computation Offloading

**Pseudo-code for this system**

set timer to interrupt periodically with period $T$;

at each timer interrupt do

- perform analog-to-digital conversion to get $y$;
- compute control output $u$ by using external devices;
- output $u$ and do digital-to-analog conversion;

od

```
Control System
A/D $y_k$ Control-law computation $u_k$ D/A

sensor $y(t)$ plant (The system being controlled) actuator
```

Prof. Dr. Jian-Jia Chen (TU Dortmund) 5 / 46
An Example: Unreliable Timing Channels

- Many powerful devices are timing unreliable, which are forbidden in hard real-time systems.
  - Graphics Processing Unit
  - Network Servers
  - Accelerators
Compensation Mechanism

- Based on the timing unpredictable behaviour on many components, we need a local compensation mechanism.

Receive Results

Local Compensation
Reasons for Suspension: I/O- or Memory-Intensive

- An I/O-intensive task may have to use DMA to transfer a large amount of data.
- This can take up to a few microseconds to milliseconds.
- Execution pattern of a job is as follows:
  - executes for a certain amount of time,
  - then initiates an I/O activity, and suspends itself.
  - is resumed to the ready queue to be (re)-eligible for execution once the I/O activity completes.
- Such latency can become much more dynamic and larger when we consider multicore platforms with shared memory.
A task may be parallelized such that it can be executed simultaneously on some processors to perform independent computation.

To this end, we can use a **directed acyclic graph (DAG)** to model the dependency of the subtasks in a sporadic task.

Each vertex in the DAG represents a subtask.
Reasons for Self-Suspensions: Locking Protocols

- Semaphores in uniprocessor systems: cause additional blocking due to the mutual exclusion

![Diagram showing two processes, J₁ and J₂, waiting for resource Rₖ. Process J₁ waits for semaphore Sₖ, uses resource Rₖ, and signals Sₖ. Process J₂ waits for semaphore Sₖ, uses resource Rₖ, and signals Sₖ.](image)
Reasons for Self-Suspensions: Locking Protocols

- Semaphores in uniprocessor systems: cause additional blocking due to the mutual exclusion
- Semaphores in multiprocessor systems: cause remote blocking due to the mutual exclusion
  - Suppose that $J_1$ and $J_2$ are on two different processors
  - If $J_1$ locks the semaphore, $J_2$ has to wait and the processor that runs $J_2$ may have to idle.
Reasons for Self-Suspensions: Physical Resource Sharing

- Multiple cores may share a bus
- The contention on the bus can be considered as a suspension problem (with respect to the bus access)
Outline

Introduction

Suspension Models

Dynamic Suspending Task Model

Segmented Suspending Task Model

Conclusion
Possible Self Suspensions

Implicit-deadline sporadic suspending task: $\tau(C, S, T)$

- 1-Segmented self-suspension: 2 computation segments separated by a suspension interval

- Segmented self-suspension: $f$ computation segments separated by $f - 1$ suspension intervals

- Dynamic self-suspension: the suspension pattern is unknown and can be arbitrary

Jobs may alternate between computation and suspension phases without any restriction on how they interleave.
Terminologies

- $C_{i,j}$ or $C^j_i$: the worst-case execution time for task $\tau_i$ in the $j$-th computation segment
- $C_i$: the worst-case execution time for task $\tau_i$
- $S_{i,j}$ or $S^j_i$: the self-suspension time for task $\tau_i$ in the $j$-th suspension interval
- $S_i$: the self-suspension time for task $\tau_i$
- $T_i$: period of task $\tau_i$
- $D_i$: relative deadline of task $\tau_i$. I will implicitly assume $T_i = D_i$, unless it is specified.
- $U_i$: utilization of task $\tau_i$, defined as $\frac{C_i}{T_i}$
Counterexample for RM and EDF

- $C_1 = 1$, $S_1 = 0$, $D_1 = 5$, $T_1 = \infty$.
- $C_{2,1} = 1$, $S_2 = 8$, $C_{2,2} = 1$, $D_2 = 10$, $T_2 = 10$.

We can easily extend to let $S_2$ to be very large.

EDF and Rate-Monotonic are in general not good.
Counterexample for RM and EDF

- $C_1 = 1, S_1 = 0, D_1 = 5, T_1 = \infty$.
- $C_{2,1} = 1, S_2 = 8, C_{2,2} = 1, D_2 = 10, T_2 = 10$.

We can easily extend to let $S_2$ to be very large.

EDF and Rate-Monotonic are in general not good.
Wait, What does this Mean?

- The gain by offloading can be completely useless
- The remote blocking and synchronization can completely destroy the feasibility
- Existing scheduling algorithms are not going to work very well
- Suspension has triggered a new dimension for designing systems
- If suspension is not handled carefully, the suspension may be harmful to the system utilization
Wait, What does this Mean?

- The gain by offloading can be completely useless
- The remote blocking and synchronization can completely destroy the feasibility
- Existing scheduling algorithms are not going to work very well
- Suspension has triggered a new dimension for designing systems
- If suspension is not handled carefully, the suspension may be harmful to the system utilization
- So, the key is to utilize and analyze the suspension impact well.
Outline

Introduction

Suspension Models

Dynamic Suspending Task Model

Segmented Suspending Task Model

Conclusion
Release the higher-priority tasks at the same time as the task (here $\tau_k$) under analysis.

The following jobs of a higher-priority task should be released then by following the period constraint:

$$\exists t \text{ with } 0 < t \leq T_k \text{ and } C_k + \sum_{j=1}^{k-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t.$$
Suspension Induces Jitter

- The response time of task $\tau_4$ becomes $27 - 5 = 22$. (It was 21 if there is no suspension.)
- Is this the worst case if only task $\tau_1$ suspends itself?
Periodic Tasks with Jitter (pjd Tasks)

A common event pattern (that is not purely periodic) can be specified by the parameter triple \((p, j, d)\), where \(p\) denotes the period, \(j\) the jitter, and \(d\) the minimum inter-arrival distance of events in the modeled stream.

courtesy slide from Lothar Thiele.
Suspension Creates Jitter (cont.)

- A self-suspending task $\tau_i$ is a PJD task
  - Period is $T_i$
  - Jitter is $S_i$
  - Minimum inter-arrival time is $C_i$ (I will not use this constraint.)
A self-suspending task $\tau_i$ is a PJD task
- Period is $T_i$
- Jitter is $S_i$
- Minimum inter-arrival time is $C_i$ (I will not use this constraint.)

Schedulability test of task $\tau_k$:
\[
\exists t \text{ with } 0 < t \leq T_k \text{ and } C_k + S_k + \sum_{j=1}^{k-1} \left\lceil \frac{t + S_j}{T_j} \right\rceil C_j \leq t.
\]
Suspension-Aware Schedulability Analysis

The following papers are based on this observation

- Meng, RTCSA 1994
- Kim et al., RTCSA 1995
- Audsley and Bletsas, ECRTS 2004
- Audsley and Bletsas, RTAS 2004
- Lakshmanan and Rajkumar, RTSS 2009 for multiprocessor synchronization problems
- Several other papers (10+) that are based on Lakshmanan and Rajkumar in RTSS 2009.
An Example

The above analysis

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$S_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>5</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>1</td>
<td>0</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
An Example

The above analysis

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$C_i$</th>
<th>$S_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>5</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>1</td>
<td>0</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

Prof. Dr. Jian-Jia Chen (TU Dortmund)
What was the Misconception?

The above analysis is incorrect.

Too optimistic!

- The setting of jitter to $S_i$ is too optimistic.
- The impact: the following papers are based on this observation
  - Meng, RTCSA 1994 (flawed)
  - Kim et al., RTCSA 1995 (flawed)
  - Audsley and Bletsas, ECRTS 2004 (flawed)
  - Audsley and Bletsas, RTAS 2004 (flawed)
  - Lakshmanan and Rajkumar, RTSS 2009 (flawed) for multiprocessor synchronization problems
  - Several other papers (10+) that are based on Lakshmanan and Rajkumar in RTSS 2009.
How to Fix It? Be Pessimistic!

\[ \text{head} \]
\[ \text{body} \]
\[ \text{tail} \]
How to Fix It? Be Pessimistic!
How to Fix It? Be Pessimistic!

head

\[ a_k \]

\[ \tau_i \]

body

\[ \tau_i \]

\[ \tau_i \]

\[ \tau_i \]

tail

\[ d_k \]

\[ \tau_i \]

bursty

\[ a_k \]

\[ \tau_i \]

\[ \tau_i \]

\[ \tau_i \]

\[ \tau_i \]
Task $\tau_k$ is schedulable under fixed-priority scheduling in a self-suspension task set, if

$$\exists t \text{ with } 0 < t \leq T_k \text{ and } C_k + S_k + \sum_{i=1}^{k-1} W_i(t) \leq t,$$

where

$$W_i(t) = \left(\left\lceil \frac{t}{T_i} \right\rceil - 1\right) C_i + 2C_i.$$

Or, equivalently, the jitter of a higher-priority task $\tau_i$ is $T_i$. 
Time-Demand Schedulability Analysis

Task $\tau_k$ is schedulable under fixed-priority scheduling in a self-suspension task set, if

$$\exists t \text{ with } 0 < t \leq T_k \text{ and } C_k + S_k + \sum_{i=1}^{k-1} W_i(t) \leq t,$$

where

$$W_i(t) = \left(\left\lceil \frac{t}{T_i} \right\rceil - 1\right) C_i + 2C_i.$$

Or, equivalently, the jitter of a higher-priority task $\tau_i$ is $T_i$. 

bursty$=2C_i$
Time-Demand Schedulability Analysis

Task $\tau_k$ is schedulable under fixed-priority scheduling in a self-suspension task set, if

$$\exists t \text{ with } 0 < t \leq T_k \text{ and } C_k + S_k + \sum_{i=1}^{k-1} W_i(t) \leq t,$$

where

$$W_i(t) = \left(\left\lfloor \frac{t}{T_i} \right\rfloor - 1\right) C_i + 2C_i.$$

Or, equivalently, the jitter of a higher-priority task $\tau_i$ is $T_i$. 

bursty=$2C_i$

Diagram:

- $\tau_i$ tasks with $a_k, d_k$ marks.
- Bursty=$2C_i$.
Utilization-Based Analysis

Theorem (Bini et al. in ECRTS 2001)

Any sporadic task set is schedulable under RM if the following conditions hold:

\[ \forall 1 \leq k \leq n, \left[ U_k \right] \leq 1 - (1 + 1) \cdot \left( 1 - \frac{1}{\prod_{i=1}^{k-1}(U_i + 1)} \right) . \quad (1) \]

Theorem (Liu and Layland JACM 1973)

Any sporadic task set is schedulable under RM if the following conditions hold:

\[ \forall 1 \leq k \leq n, U_k + \sum_{i=1}^{k-1} U_i \leq k \left( \left( \frac{1}{1 + 1} \right)^{\frac{1}{k}} - 1 \right) \]  \quad (2)
Utilization-Based Analysis

Theorem (Liu and Chen in RTSS 2014)

Any sporadic self-suspending task set is schedulable under RM if the following conditions hold:

\[ \forall 1 \leq k \leq n, \quad U_k + \frac{S_k}{T_k} \leq 1 - \left( \frac{2}{k} + 1 \right) \cdot \left( 1 - \frac{1}{\prod_{i=1}^{k-1}(U_i + 1)} \right) \]  

(1)

Theorem (Liu and Chen in RTSS 2014)

Any sporadic self-suspending task set is schedulable under RM if the following conditions hold:

\[ \forall 1 \leq k \leq n, \quad U_k + \frac{S_k}{T_k} + \sum_{i=1}^{k-1} U_i \leq k \left( \left( \frac{2}{2} + 1 \right)^{\frac{1}{k}} - 1 \right) \]  

(2)
Calculating Suspension Time Can Be Also Tricky

- The original analysis in distributed priority ceiling protocol (DPCP by Rajkumar in ICDCS 1990)

- Non-nested critical sections
- Critical sections guarded by one semaphore are always executed on one dedicated processor
- Three tasks, each of them assigned on one processor, using one binary semaphore on Proc₀.

<table>
<thead>
<tr>
<th>τᵢ</th>
<th>Proc(τᵢ)</th>
<th>Cᵢ</th>
<th>Tᵢ (= Dᵢ)</th>
<th>Nₖ</th>
<th>Lᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁</td>
<td>Proc₁</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>τ₂</td>
<td>Proc₂</td>
<td>11</td>
<td>18</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>τ₃</td>
<td>Proc₃</td>
<td>8</td>
<td>20</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

- Cᵢ: worst-case execution time (including the critical section length)
- Tᵢ: the period
- Nᵢ: the number of critical sections per job invocation
- Lᵢ: the worst-case critical section length (per critical section).
Calculating Suspension Time Can Be Tricky

<table>
<thead>
<tr>
<th>( \tau_i )</th>
<th>( \text{Proc}(\tau_i) )</th>
<th>( C_i )</th>
<th>( T_i (= D_i) )</th>
<th>( N_k )</th>
<th>( L_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>( \text{Proc}_1 )</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>( \text{Proc}_2 )</td>
<td>11</td>
<td>18</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>( \text{Proc}_3 )</td>
<td>8</td>
<td>20</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

- Multi-tasking only takes place on \( \text{Proc}_0 \)
- The original analysis argues that the additional delay \( B_k \) due to DPCP on \( \text{Proc}_0 \) for task \( \tau_k \) is upper bounded by \( B_k \leq N_k \cdot (\max_{j \neq k} L_j) + \sum_{i=1}^{k-1} \left\lceil \frac{T_k}{T_i} \right\rceil L_i N_i \).
  - The first term is due to the fact that each critical section access can be blocked by a lower-priority task.
  - The second term is due to the interference from the higher-priority tasks under the critical instant theorem.

Therefore,
- \( B_1 \) is upper bounded by 4,
- \( B_2 \) is upper bounded by \( 1 + 2 \cdot 2 = 5 \), and
- \( B_3 \) is upper bounded by \( 0 + 2 \cdot 2 + 4 \cdot 2 = 12 \).
Something Went Wrong

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$\text{Proc}(\tau_i)$</th>
<th>$C_i$</th>
<th>$T_i (= D_i)$</th>
<th>$N_k$</th>
<th>$L_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>$\text{Proc}_1$</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>$\text{Proc}_2$</td>
<td>11</td>
<td>18</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>$\text{Proc}_3$</td>
<td>8</td>
<td>20</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

A job of task $\tau_3$: run 0.5 time unit on $\text{Proc}_3$, critical section 1 time unit, run 1 time unit on $\text{Proc}_3$, access the critical section for 1 time unit, run 3.5 time units on $\text{Proc}_3$, and access the critical section for 1 time unit.
Impact

• This wrong quantification of suspension time was used by
  • R. Rajkumar, in ICDCS 1990.
  • Lakshmanan and Rajkumar, RTSS 2009
  • P. Hsiu, D. Lee, and T. Kuo, in EMSOFT 2011.
  • F. Nemati, M. Behnam, and T. Nolte, in ECRTS 2011.

• Correct settings of jitter can solve this problem
Can We Do Better? Suspension as Blocking

• In the textbook ”Real-Time Systems” by Jane W. S. Liu, she proposed to model the extra delay as blocking denoted as $B_k$:

  • The blocking time contributed from task $\tau_k$ is $S_k$.
  • A higher-priority task $\tau_i$ can only block the execution of task $\tau_k$ by at most $\min(C_i, S_i)$.

  $$B_k = S_k + \sum_{i=1}^{k-1} \min(C_i, S_i).$$
Can We Do Better? Suspension as Blocking

- In the textbook ”Real-Time Systems” by Jane W. S. Liu, she proposed to model the extra delay as blocking denoted as $B_k$:
  - The blocking time contributed from task $\tau_k$ is $S_k$.
  - A higher-priority task $\tau_i$ can only block the execution of task $\tau_k$ by at most $\min(C_i, S_i)$.

  $$B_k = S_k + \sum_{i=1}^{k-1} \min(C_i, S_i).$$

If the argument is correct, we can revise the analysis:

$$\exists t \mid 0 < t \leq T_k, \quad C_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t.$$
Can We Do Better? Suspension as Blocking

• In the textbook ”Real-Time Systems” by Jane W. S. Liu, she proposed to model the extra delay as blocking denoted as $B_k$:
  
  • The blocking time contributed from task $\tau_k$ is $S_k$.
  • A higher-priority task $\tau_i$ can only block the execution of task $\tau_k$ by at most $\min(C_i, S_i)$.

  $$B_k = S_k + \sum_{i=1}^{k-1} \min(C_i, S_i).$$

If the argument is correct, we can revise the analysis:

$$\exists t \mid 0 < t \leq T_k, \quad C_k + B_k + \sum_{i=1}^{k-1} \left\lfloor \frac{t}{T_i} \right\rfloor C_i \leq t.$$  

This was also used by Rajkumar et al. in RTSS 1988 and ICDCS 1990.
Can We Do Better? Suspension as Blocking

- In the textbook ”Real-Time Systems” by Jane W. S. Liu, she proposed to model the extra delay as blocking denoted as $B_k$:
  
  - The blocking time contributed from task $\tau_k$ is $S_k$.
  - A higher-priority task $\tau_i$ can only block the execution of task $\tau_k$ by at most $\min(C_i, S_i)$.

$$B_k = S_k + \sum_{i=1}^{k-1} \min(C_i, S_i).$$

If the argument is correct, we can revise the analysis:

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

The analysis is correct!

Outline

Introduction

Suspension Models

Dynamic Suspending Task Model

Segmented Suspending Task Model

Conclusion
Segmented Suspension

- Arbitrary suspension model provides an easy way to specify suspending systems
  - suffers from the poor schedulability
  - using arbitrary suspension blindly is too pessimistic
- When the suspension patterns are known (or are specified with certain guarantees), it is better to use segmented suspensions.
Period Enforcer

- Rajkumar in 1991 proposed the *period enforcer* algorithm.
- It is a technique to control the processor demand.
- The key idea: artificially delay the execution of computation segments if a job resumes *too soon*.
- The period enforcer algorithm determines for each computation segment an *eligibility time*.
- If a segment resumes before its eligibility time, the execution of the segment is delayed until the eligibility time is reached.
Period Enforcer

- Rajkumar in 1991 proposed the *period enforcer* algorithm.
- It is a technique to control the processor demand.
- The key idea: artificially delay the execution of computation segments if a job resumes *too soon*.
- The period enforcer algorithm determines for each computation segment an *eligibility time*.
- If a segment resumes before its eligibility time, the execution of the segment is delayed until the eligibility time is reached.
- You can imagine that this is like a sporadic server.
Period Enforcer: An Example

\[
\begin{array}{|c|c|c|}
\hline
(C_i^1, S_i^1, C_i^2) & D_i = T_i \\
\hline
\tau_1 & (3, 0, 0) & 10 \\
\tau_2 & (1, 4, 2) & 10 \\
\tau_3 & (3, 0, 0) & 10 \\
\hline
\end{array}
\]
Period Enforcement Can Induce Deadline Misses

\[
\begin{array}{|c|c|c|}
\hline
(C_i^1, S_i^1, C_i^2) & D_i = T_i \\
\hline
\tau_1 & (2, 0, 0) & 10 \\
\tau_2 & (1, 6, 1) & 11 \\
\hline
\end{array}
\]

Prof. Dr. Jian-Jia Chen (TU Dortmund)
Critical Instant?

Let’s consider the simplest case under fixed-priority scheduling:

- $\tau_k$ is the lowest priority task
- all the higher priority tasks are sporadic and non-self-suspending

Lakshmanan and Rajkumar (in RTAS 2010) proved that the critical instant of task $\tau_k$ is as follows:

- every task releases a job simultaneously with $\tau_k$;
Critical Instant?

Let’s consider the simplest case under fixed-priority scheduling:

- $\tau_k$ is the lowest priority task
- all the higher priority tasks are sporadic and non-self-suspending

Lakshmanan and Rajkumar (in RTAS 2010) proved that the critical instant of task $\tau_k$ is as follows:

- every task releases a job simultaneously with $\tau_k$;
- the jobs of higher priority tasks that are eligible to be released during the self-suspension interval of $\tau_k$ are delayed to be aligned with the release of the subsequent computation segment of $\tau_k$; and
Critical Instant?

Let’s consider the simplest case under fixed-priority scheduling:

- $\tau_k$ is the lowest priority task
- all the higher priority tasks are sporadic and non-self-suspending

Lakshmanan and Rajkumar (in RTAS 2010) proved that the critical instant of task $\tau_k$ is as follows:

- every task releases a job simultaneously with $\tau_k$;
- the jobs of higher priority tasks that are eligible to be released during the self-suspension interval of $\tau_k$ are delayed to be aligned with the release of the subsequent computation segment of $\tau_k$; and
- all the remaining jobs of the higher priority tasks are released with their minimum inter-arrival time.
An Example

\[
\begin{array}{|c|c|c|}
\hline
\tau_i & (C^1_i, S^1_i, C^2_i) & D_i = T_i \\
\hline
\tau_1 & (1, 0, 0) & 4 \\
\tau_2 & (1, 0, 0) & 9 \\
\tau_3 & (1, 2, 3) & 9 \\
\hline
\end{array}
\]

(a) Lakshmannan’s Critical Instant.
An Example

<table>
<thead>
<tr>
<th>$\tau_i$</th>
<th>$(C^1_i, S^1_i, C^2_i)$</th>
<th>$D_i = T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>(1, 0, 0)</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>(1, 0, 0)</td>
<td>9</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>(1, 2, 3)</td>
<td>9</td>
</tr>
</tbody>
</table>

(a) Lakshmannan’s Critical Instant.  
(b) Do not release jobs synchronously.

Counterexample provided by Nelissen et al. in ECRTS 2015.
Fixed-Relative-Deadline (FRD) Approaches

- When a job of task $\tau_i$ arrives at time $t$,
  - the absolute deadline of the job in the first computation phase is set to $t + D_{i,1}$
  - the suspension has to be finished before $t + D_{i,1} + S_i$,
  - the release time of the second subjob (the second computation phase) is $t + D_{i,1} + S_i$
  - the absolute deadline of the second subjob is $t + T_i$
Proportional Fixed-Relative Deadline Assignments

Liu et al. in DAC 2014 for only one suspension interval per task.

- \( D_{i,1} = \frac{C_{i,1}}{C_{i,1} + C_{i,2}} (T_i - S_i) \)
- \( D_{i,2} = \frac{C_{i,2}}{C_{i,1} + C_{i,2}} (T_i - S_i) \)
- Therefore, we have \( \frac{C_{i,1}}{D_{i,1}} = \frac{C_{i,2}}{D_{i,2}} = \frac{C_{i,1} + C_{i,2}}{T_i - S_i} \)

- Is Proportional FRD Good?
  - It can be proved that this does not yield good analytical bounds.
Equal-Deadline Assignment (EDA)

Chen and Liu in RTSS 2014

\[ D_{i,1} = D_{i,2} = \frac{T_i - S_i}{2}. \]
Equal-Deadline Assignment (EDA)

Chen and Liu in RTSS 2014

\[ D_{i,1} = D_{i,2} = \frac{T_i - S_i}{2}. \]

Remarks

sounds very pessimistic, but the first sound method (with approximation/speedup guarantee). Originally proposed only for dynamic-priority scheduling.
Equal-Deadline Assignment (EDA)

Chen and Liu in RTSS 2014

\[ D_{i,1} = D_{i,2} = \frac{T_i - S_i}{2}. \]

Remarks

sounds very pessimistic, but the first sound method (with approximation/speedup guarantee). Originally proposed only for dynamic-priority scheduling.

Remarks

Huang and Chen (DATE 2016): extended to fixed-priority scheduling and multiple suspension intervals.
Different Priority per Computation Segment

<table>
<thead>
<tr>
<th></th>
<th>( (C_{i,1}, S_{i,1}, C_{i,2}) )</th>
<th>( D_i = T_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>( (10, 0, 0) )</td>
<td>30</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>( (5, 5, 16) )</td>
<td>40</td>
</tr>
</tbody>
</table>

- Priority level: \( C_1 - C_2 \)
- One may conclude that the worst-case response time of \( C_1 \) is 5 and the worst-case response time of \( C_2 \) is \( 16 + 10 = 26 \).
- Since \( 5 + 5 + 26 = 36 \leq 40 \), the lowest-priority segment can meet the deadline.
Different Priority per Computation Segment

<table>
<thead>
<tr>
<th>$C_i, S_i, C_i, D_i = T_i$</th>
<th>$D_i = T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$ (10, 0, 0)</td>
<td>30</td>
</tr>
<tr>
<td>$\tau_2$ (5, 5, 16)</td>
<td>40</td>
</tr>
</tbody>
</table>

- Priority level: $C_2^1 - C_1^1 - C_2^2$
- One may conclude that the worst-case response time of $C_2^1$ is 5 and the worst-case response time of $C_2^2$ is $16 + 10 = 26$.
- Since $5 + 5 + 26 = 36 \leq 40$, the lowest-priority segment can meet the deadline.
Different Priority per Computation Segment

<table>
<thead>
<tr>
<th>$(C_{i,1}, S_{i,1}, C_{i,2})$</th>
<th>$D_i = T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>$(10, 0, 0)$</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>$(5, 5, 16)$</td>
</tr>
</tbody>
</table>

- Priority level: $C_2^1 - C_1^1 - C_2^2$

- One may conclude that the worst-case response time of $C_2^1$ is 5 and the worst-case response time of $C_2^2$ is $16 + 10 = 26$.

- Since $5 + 5 + 26 = 36 \leq 40$, the lowest-priority segment can meet the deadline.

- Yes, possible, but pay attention

- This was used by Kim et al. RTSS 2013, and Ding et al. in IEICE Transactions 2009.
Outline

Introduction

Suspension Models

Dynamic Suspending Task Model

Segmented Suspending Task Model

Conclusion
Conclusion

• Suspension can be very harmful if it is not treated well
• Suspension relates to important features in the era of multicore systems and cyber-physical systems
  • Computation offloading
  • Shared memory and bus in multicore systems
  • Virtual shared resources (like semaphores) in multicore systems
  • GPU/FPGA acceleration
  • etc.

• This is a non-trivial problem
  • Studied already early in 90’s but with quite a few misconceptions
  • Broken literature
Positive Results

- Wen-Hung Huang and Jian-Jia Chen. Self-Suspension Real-Time Tasks under Fixed-Relative-Deadline Fixed-Priority Scheduling. in DATE, 2016
- Jian-Jia Chen, Cong Liu: Fixed-Relative-Deadline Scheduling of Hard Real-Time Tasks with Self-Suspensions. in RTSS 2014
- Cong Liu, Jian-Jia Chen: Bursty-Interference Analysis Techniques for Analyzing Complex Real-Time Task Models. in RTSS 2014
- Wei Liu, Jian-Jia Chen, Anas Toma, Tei-Wei Kuo, Qingxu Deng: Computation Offloading by Using Timing Unreliable Components in Real-Time Systems. in DAC 2014