The SRP Resource Sharing Protocol for Self-Suspending Tasks

Geoffrey Nelissen and Alessandro Biondi
The SRP Resource Sharing Protocol for Self-Suspending Tasks

Geoffrey Nelissen and Alessandro Biondi
The SRP Resource Sharing Protocol for Self-Suspending Tasks (on single core platforms)

Geoffrey Nelissen and Alessandro Biondi
Motivation

Evolution of the state of practice
Motivation

Evolution of the state of practice
Motivation

Evolution of the state of practice

Parallel applications

Multicore processors
Motivation

Evolution of the state of practice

Parallel applications

Partitioned FP scheduling

Multicore processors
Motivation

Response Time Analysis of Sporadic DAG Tasks under Partitioned Scheduling

José Fonseca, Geoffrey Nelissen, Vincent Nélis and Luis Miguel Pinho

Parallel applications partitioned on multicore platforms can be modelled as sets of self-suspending tasks running on single core processors
Motivation

Parallel applications partitioned on multicore platforms can be modelled as sets of self-suspending tasks running on single core processors.

Partitioned Fixed-Priority Scheduling of Parallel Tasks Without Preemptions

Daniel Casini, Alessandro Biondi, Geoffrey Nelissen and Giorgio Buttazzo

This afternoon @ RTSS
Motivation

Parallel applications partitioned on multicore platforms can be modelled as sets of self-suspending tasks running on single core processors.

Most applications will share resources through locks.
Motivation

Parallel applications partitioned on multicore platforms can be modelled as sets of self-suspending tasks running on single core processors.

Most applications will share resources through locks.

Standards such as AUTOSAR mandates the use of the SRP for sharing resources.
The Stack Resource Policy (SRP)
The Stack Resource Policy (SRP)

Task 1

Task 2

Shared resource 1
The Stack Resource Policy (SRP)

Task 1  Prio=2

Task 2  Prio=1

Shared resource 1
The Stack Resource Policy (SRP)

Task 1  Prio=2

Task 2  Prio=1

Shared resource 1  Prio=2

priority of highest priority task accessing that resource
The Stack Resource Policy (SRP)

Task 1  Prio=2
Task 2  Prio=1
Shared resource 1  Prio=2

System ceiling

priority of the last resource locked and not unlocked
A task T1 can **preempt** another task T2 if \( \text{prio}(T1) > \text{prio}(T2) \) and \( \text{prio}(T1) > \text{system ceiling} \).
The Stack Resource Policy (SRP)

A task T1 can preempt another task T2 if 
\[ \text{prio}(T1) > \text{prio}(T2) \text{ and } \text{prio}(T1) > \text{system ceiling} \]
A task T1 can preempt another task T2 if \( \text{prio}(T1) > \text{prio}(T2) \) and \( \text{prio}(T1) > \text{system ceiling} \).
The Stack Resource Policy (SRP)

A task T1 can preempt another task T2 if
\[ \text{prio}(T1) > \text{prio}(T2) \text{ and } \text{prio}(T1) > \text{system ceiling} \]
The Stack Resource Policy (SRP)

A task T1 can **preempt** another task T2 if
\[ \text{prio}(T1) > \text{prio}(T2) \text{ and } \text{prio}(T1) > \text{system ceiling} \]
The Stack Resource Policy (SRP)

A task T1 can **preempt** another task T2 if
\[ \text{prio}(T1) > \text{prio}(T2) \text{ and } \text{prio}(T1) > \text{system ceiling} \]
The Stack Resource Policy (SRP)

A task T1 can **preempt** another task T2 if
\[ \text{prio}(T1) > \text{prio}(T2) \text{ and } \text{prio}(T1) > \text{system ceiling} \]
WCRT analysis for the SRP

\[ R_i = B_i + C_i + \sum_{\tau_j \in h_p(i)} \left\lfloor \frac{R_i}{T_j} \right\rfloor C_j \]

\[ B_i = \max_{\tau_j \in l_p(i), \ell_k \in Q} \left\{ L_{j,k} \mid \pi(\ell_k) \geq \pi_i \right\}_0 \]
WCRT analysis for the SRP

\[ R_i = B_i + C_i + \sum_{\tau_j \in hp(i)} \left\lfloor \frac{R_i}{T_j} \right\rfloor C_j \]

\[ B_i = \max_{\tau_j \in lp(i), \ell_k \in Q} \{ L_{j,k} | \pi(\ell_k) \geq \pi_i} \}

Blocking time

Critical section length whose priority \( \geq \) prio(task \( i \))
SRP and self-suspending tasks
SRP and self-suspending tasks

- **Execution time**
- **Critical section**
- **Blocking time**
- **Suspension**

Task 1
- Priority = 2
- Execution time

Task 2
- Priority = 1
- Execution time

Shared resource 1
- Priority = 2
- Execution time

System ceiling
- 0
SRP and self-suspending tasks

Task 1
Prio=2

Task 2
Prio=1

Shared resource 1
Prio=2

System ceiling

Execution time
Critical section
Blocking time
Suspension
SRP and self-suspending tasks

- Task 1: Prio=2
- Task 2: Prio=1
- Shared resource 1: Prio=2

System ceiling: 0
SRP and self-suspending tasks

Task 1
Prio = 2

Task 2
Prio = 1

Shared resource 1
Prio = 2

System ceiling
0

Execution time
Critical section
Blocking time
Suspension
SRP and self-suspending tasks

Execution time  Critical section  Blocking time  Suspension

Task 1  Prio=2

Task 2  Prio=1

Shared resource 1  Prio=2

System ceiling  0
SRP and self-suspending tasks

Execution time  Critical section  Blocking time  Suspension

Task 1  Prio=2

Task 2  Prio=1

Shared resource 1  Prio=2

System ceiling

2
SRP and self-suspending tasks

Execution time  Critical section  Blocking time  Suspension

Task 1  Prio=2
Task 2  Prio=1
Shared resource 1  Prio=2

System ceiling

0
SRP and self-suspending tasks

- Execution time
- Critical section
- Blocking time
- Suspension

Task 1: Prio=2

Task 2: Prio=1

Shared resource 1: Prio=2

System ceiling: 0
SRP and self-suspending tasks

![Diagram showing execution time, critical section, blocking time, and suspension for Task 1, Task 2, and Shared resource 1.]

Task 1: Prio=2
Task 2: Prio=1
Shared resource 1: Prio=2

System ceiling: 2
SRP and self-suspending tasks

Task 1  Prio=2
Task 2  Prio=1
Shared resource 1  Prio=2

Execution time  Critical section  Blocking time  Suspension

System ceiling

0
SRP and self-suspending tasks

A task may be blocked as many times as it self-suspends.
SRP + self-suspending tasks
Straightforward analysis

\[
R_i = (C_i + S_i) + B_i(R_i, \overline{R}) + \sum_{\tau_j \in h\rho(i)} \left[ \frac{R_i + \overline{R}_j - C_j}{T_j} \right] C_j
\]

\[
B_i(R_i, \overline{R}) = (X_i + 1) \times \max_{\tau_j \in l\rho(i), \ell_k \in Q} \{ L_{j,k} \mid \pi(\ell_k) \geq \pi_i \}
\]

Maximum number of times task \(i\) self-suspends
SRP + self-suspending tasks
Improved analysis
SRP + self-suspending tasks
Improved analysis

- Identify all the critical sections that may block task $i$ during its response time
SRP + self-suspending tasks
Improved analysis

- Identify all the critical sections that may block task $i$ during its response time
- Save their length in a multi-set $S_i(R_i)$
SRP + self-suspending tasks
Improved analysis

• Identify all the critical sections that may block task \( i \) during its response time
• Save their length in a multi-set \( S_i(R_i) \)
• The blocking \( B_i \) is bounded by the sum of the \((X_i + 1)\) longest critical sections in \( S_i(R_i) \)
SRP + self-suspending tasks
Improved analysis

- Identify all the **critical sections** that **may block** task \(i\) during its response time
- Save their length in a multi-set \(S_i(R_i)\)
- The blocking \(B_i\) is bounded by the sum of the \((X_i + 1)\) **longest** critical sections in \(S_i(R_i)\)

---

Note that the **content of** \(S_i(R_i)\) depends on the **worst-case response time of all tasks**
\(\Rightarrow \) \(S_i(R_i)\) must be **iteratively computed**
Improving the locking protocol
A task T1 can preempt another task T2 if \( \text{prio}(T1) > \text{prio}(T2) \) and \( \text{prio}(T1) > \text{system ceiling} \)
Improving the locking protocol

A task T1 can **preempt** another task T2 if \( \text{prio}(T1) > \text{prio}(T2) \) and \( \text{prio}(T1) > \text{system ceiling} \)
Improving the locking protocol

A task T1 can preempt another task T2 if \( \text{prio}(T1) > \text{prio}(T2) \) and \( \text{prio}(T1) > \text{system ceiling} \)
Improving the locking protocol

A task Ti is **eligible** to execute if
\[
prio(Ti) > \text{system priority}
\]

A task T1 can **preempt** another task T2 if
\[
prio(T1) > prio(T2) \text{ and } prio(T1) > \text{system ceiling}
\]
**Improving the locking protocol**

A task $T_i$ is *eligible* to execute if

$$\text{prio}(T_i) > \text{system priority}$$

A task $T_1$ can *preempt* another task $T_2$ if

$$\text{prio}(T_1) > \text{prio}(T_2) \text{ and } \text{prio}(T_1) > \text{system ceiling}$$
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task $T_i$ is eligible to execute if $\text{prio}(T_i) > \text{system priority}$

A task $T_1$ can preempt another task $T_2$ if $\text{prio}(T_1) > \text{prio}(T_2)$ and $\text{prio}(T_1) > \text{system ceiling}$

Task 1
- Prio=3
- $\pi^{ss} = 1$

Task 2
- Prio=2
- $\pi^{ss} = 1$

Task 3
- Prio=1
- $\pi^{ss} = 0$

Shared resource 1
- Prio=3

System ceiling
- 0

System Priority
- 0
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task $T_i$ is **eligible** to execute if
$$\text{prio}(T_i) > \text{system priority}$$

A task $T_1$ can **preempt** another task $T_2$ if
$$\text{prio}(T_1) > \text{prio}(T_2) \text{ and } \text{prio}(T_1) > \text{system ceiling}$$

Maximum $\pi^{ss}$ among all active tasks, i.e., started and not completed

| Task 1 | Prio=3  
|--------|--------|
| $\pi^{ss} = 1$  
| Task 2 | Prio=2  
| $\pi^{ss} = 1$  
| Task 3 | Prio=1  
| $\pi^{ss} = 0$  
| Shared resource 1 | Prio=3  
| Execution time | Critical section | Blocking time | Suspension |

System ceiling | System Priority

0 | 0
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

- **Execution time**
- **Critical section**
- **Blocking time**
- **Suspension**

- **Task 1**
  - Prio=3
  - $\pi^{ss} = 1$
- **Task 2**
  - Prio=2
  - $\pi^{ss} = 1$
- **Task 3**
  - Prio=1
  - $\pi^{ss} = 0$
- **Shared resource 1**
  - Prio=3

**System ceiling**

**System Priority**

A task $T_i$ is **eligible** to execute if $\text{prio}(T_i) > \text{system priority}$

A task $T_1$ can **preempt** another task $T_2$ if $\text{prio}(T_1) > \text{prio}(T_2)$ and $\text{prio}(T_1) > \text{system ceiling}$

Maximum $\pi^{ss}$ among all active tasks, i.e., started and not completed
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task $T_i$ is *eligible* to execute if $\text{prio}(T_i) > \text{system priority}$

A task $T_1$ can *preempt* another task $T_2$ if $\text{prio}(T_1) > \text{prio}(T_2)$ and $\text{prio}(T_1) > \text{system ceiling}$

- **Execution time**
- **Critical section**
- **Blocking time**
- **Suspension**

Task 1
- Prio=3
- $\pi^{ss} = 1$

Task 2
- Prio=2
- $\pi^{ss} = 1$

Task 3
- Prio=1
- $\pi^{ss} = 0$

Shared resource 1
- Prio=3

System ceiling
- 3

System Priority
- 0

Maximum $\pi^{ss}$ among all active tasks, i.e., started and not completed.
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task Ti is **eligible** to execute if
\[ \text{prio}(Ti) > \text{system priority} \]

A task T1 can **preempt** another task T2 if
\[ \text{prio}(T1) > \text{prio}(T2) \text{ and } \text{prio}(T1) > \text{system ceiling} \]

Maximum $\pi^{ss}$ among all active tasks, i.e., started and not completed.
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task $T_i$ is \textit{eligible} to execute if $\text{prio}(T_i) > \text{system priority}$

A task $T_1$ can \textit{preempt} another task $T_2$ if $\text{prio}(T_1) > \text{prio}(T_2)$ and $\text{prio}(T_1) > \text{system ceiling}$
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

<table>
<thead>
<tr>
<th>Task</th>
<th>Priority</th>
<th>$\pi^{ss}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Task 2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Task 3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Shared resource 1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

A task $T_i$ is **eligible** to execute if $\text{prio}(T_i) > \text{system priority}$

A task $T_1$ can **preempt** another task $T_2$ if $\text{prio}(T_1) > \text{prio}(T_2)$ and $\text{prio}(T_1) > \text{system ceiling}$
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task Ti is **eligible** to execute if
\[ \text{prio}(T_i) > \text{system priority} \]

A task T1 can **preempt** another task T2 if
\[ \text{prio}(T_1) > \text{prio}(T_2) \text{ and } \text{prio}(T_1) > \text{system ceiling} \]

Maximum \( \pi^{ss} \) among all active tasks, i.e., started and not completed
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task Ti is eligible to execute if
\[ \text{prio}(Ti) > \text{system priority} \]

A task T1 can preempt another task T2 if
\[ \text{prio}(T1) > \text{prio}(T2) \text{ and } \text{prio}(T1) > \text{system ceiling} \]

Maximum \( \pi^{ss} \) among all active tasks, i.e., started and not completed
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task Ti is **eligible** to execute if
\[ \text{prio}(T_i) > \text{system priority} \]

A task T1 can **preempt** another task T2 if
\[ \text{prio}(T_1) > \text{prio}(T_2) \text{ and } \text{prio}(T_1) > \text{system ceiling} \]

Maximum \( \pi^{ss} \) among all active tasks, i.e., started and not completed.
The Stack Resource Policy for Self-Suspending tasks (SRP-SS)

A task $T_i$ is *eligible* to execute if $\text{prio}(T_i) > \text{system priority}$

A task $T_1$ can *preempt* another task $T_2$ if $\text{prio}(T_1) > \text{prio}(T_2)$ and $\text{prio}(T_1) > \text{system ceiling}$

Task 1 is not blocked by Task 3 anymore, but Task 3 may be blocked by Task 1 ➔ we avoided the priority inversion anymore

Maximum $\pi^{ss}$ among all active tasks, i.e., started and not completed

A task $T_i$ can *preempt* another task $T_2$ if $\text{prio}(T_1) > \text{prio}(T_2)$ and $\text{prio}(T_1) > \text{system ceiling}$

Task 1 is not blocked by Task 3 anymore, but Task 3 may be blocked by Task 1 ➔ we avoided the priority inversion anymore

Maximum $\pi^{ss}$ among all active tasks, i.e., started and not completed
Configuring SRP-SS
Configuring SRP-SS

• New challenge: How do we assign values to the parameters $\pi^{SS}$ in order to balance PI-blocking and blocking by self-suspensions?
Configuring SRP-SS

• New challenge: How do we assign values to the parameters $\pi^{ss}$ in order to balance PI-blocking and blocking by self-suspensions?

• A schedulability test and an iterative configuration algorithm is presented in the paper
  • Effective but not optimal
Configuring SRP-SS

• New challenge:
  How do we assign values to the parameters $\pi^{SS}$ in order to balance PI-blocking and blocking by self-suspensions?

• A schedulability test and an iterative configuration algorithm is presented in the paper
  • Effective but not optimal

• If all $\pi^{SS} = 0$, then SRP-SS is identical to SRP
Configuring SRP-SS

• New challenge: How do we assign values to the parameters $\pi^{SS}$ in order to balance PI-blocking and blocking by self-suspensions?

• A schedulability test and an iterative configuration algorithm is presented in the paper
  • Effective but not optimal

• If all $\pi^{SS}=0$, then SRP-SS is identical to SRP
  • $\Rightarrow$ SRP-SS dominates SRP
Results
Results

\[ n = 10, \ n_r = 4, \ N^{max} = 3, L^{min} = 10, L^{max} = 500, \]
\[ \sigma^{min} = 0, \ \sigma^{max} = 0.05, \ X^{min} = 2, \ X^{max} = 5 \]

Ideal locking protocol

SRP simple analysis
Results

\[ n = 10, n_r = 4, N^{max} = 3, L^{min} = 10, L^{max} = 500, \]
\[ \sigma^{min} = 0, \sigma^{max} = 0.05, X^{min} = 2, X^{max} = 5 \]

Ideal locking protocol

SRP simple analysis

SRP improved analysis
Results

$n = 10, \ n_r = 4, \ N^{max} = 3, L^{min} = 10, \ L^{max} = 500,$
$\sigma^{min} = 0, \ \sigma^{max} = 0.05, \ X^{min} = 2, \ X^{max} = 5$

Ideal locking protocol

SRP
Simple analysis

SRP-SS

SRP
Improved analysis
Results

$n = 15, n_r = 8, N^{max} = 1, L^{min} = 10, L^{max} = 200,$
$s^{min} = 0, s^{max} = 0.1, X^{min} = 4, X^{max} = 8$

Ideal locking protocol
SRP-SS
SRP simple analysis
SRP improved analysis
Conclusion
Conclusion

• We developed a new response time analysis for self-suspending tasks using SRP
Conclusion

• We developed a new response time analysis for self-suspending tasks using SRP

• We extended SRP to limit the number of priority inversions self-suspending tasks may suffer
  • Adding one task parameter and system priority
Conclusion

• We developed a new response time analysis for self-suspending tasks using SRP
• We extended SRP to limit the number of priority inversions self-suspending tasks may suffer
  • Adding one task parameter and system priority
• The SRP-SS dominates the SRP
Conclusion

• We developed a new **response time analysis** for self-suspending tasks using SRP
• We **extended SRP** to **limit** the number of **priority inversions** self-suspending tasks may suffer
  • Adding one task parameter and system priority
• The SRP-SS **dominates** the SRP
• We propose a **configuration algorithm** of the SRP-SS to **balance PI-blocking and interference** suffered by low priority tasks