Blinder: Partition-Oblivious Hierarchical Scheduling

Man-Ki Yoon, Mengqi Liu, Hao Chen, Jung-Eun Kim, Zhong Shao

Yale University
Temporal Isolation

• Key requirement for safety-critical systems
  • Especially when integrating applications with different levels of criticality (e.g., DO-178B, ASIL)

• Time partitioning
  • Isolates potential misuse of CPU resource
  • Enables modular reasoning about temporal behavior
Real-Time Hierarchical Scheduling

- Two-level scheduling: global and local
  - Provides each application (=partition) with the illusion of exclusive CPU resource

![Diagram showing four partitions: \( \Pi_1 \), \( \Pi_2 \), \( \Pi_3 \), \( \Pi_4 \) with associated local and global schedulers. The diagram illustrates how tasks are assigned to partitions and scheduled by the global scheduler.](Image)

Selects a partition: \( \Pi(t) = \Pi_i \in \Pi \)
Real-Time Hierarchical Scheduling

- Two-level scheduling: global and local
  - Provides each application (=partition) with the illusion of exclusive CPU resource

```
Π₁  Π₂  Π₃  Π₄

Π₁-Local Scheduler
Π₂-Local Scheduler
Π₃-Local Scheduler
Π₄-Local Scheduler

Tasks

Π(t)-local scheduler selects a task from Π(t)

Selects a partition Π(t) = Πᵢ ∈ Π
```
What This Paper is About

• Temporal isolation from a security perspective
  • Algorithmic covert timing channel through hierarchical scheduling
  • Scheduling approach to making partitions oblivious to other’s varying behavior
Preliminary: Non-static Time Partitioning

- Priority-based partition scheduling
  - Real-time server algorithms (e.g., sporadic server, deferrable server)
- Budget and replenishment period
- Better CPU utilization and responsiveness than static partitioning
Algorithmic Covert Timing Channel in Hierarchical Scheduling

$\Pi_S \xrightarrow{\text{Hierarchical scheduling}} \Pi_R$

Input $X$: 0/1  
Output $Y$: 0/1
Algorithmic Covert Timing Channel in Hierarchical Scheduling

Priority(\(\Pi_S\)) > Priority(\(\Pi_R\))

Partition S

Partition R

\(\tau_{R,2}\)'s arrival

Priority(\(\tau_{R,2}\)) > Priority(\(\tau_{R,1}\))

\(t_0\) Time

\(1\) A counter shared between \(\tau_{R,1}\) and \(\tau_{R,2}\)

\(\tau_{R,1}\)

\(\tau_{R,2}\)

\(\Pi_S\) Hierarchical scheduling \(\Pi_R\)

Input X: 0/1 Output Y: 0/1

(a) \(\Pi_S\) sends bit 0

```c
int receiver1_job(void) {
    prev_c = shared_c;
    n_loops = 6000000;
    for (i=0; i<n_loops; i++)
        asm("nop");
}
```
Algorithmic Covert Timing Channel in Hierarchical Scheduling

Partition S

\[ \tau_{S,1} \]

Partition R

\[ \tau_{R,1}, \tau_{R,2} \]

Input X: 0/1

Output Y: 0/1

(a) \( \Pi_S \) sends bit 0

\[
\begin{align*}
\text{void} & \text{ sender\_job(int bit) \{ \\
& \quad \text{if (bit==1)} \\
& \quad \quad \text{n\_loops} = 1000000; \\
& \quad \text{else} \\
& \quad \quad \text{n\_loops} = 2000000; \\
& \quad \text{for (i=0; i<n\_loops; i++)} \\
& \quad \quad \text{asm("nop");} \\
& \}}
\end{align*}
\]

\[
\begin{align*}
\text{int} & \text{ receiver1\_job(void) \{ \\
& \quad \text{prev\_c} = \text{shared\_c} ; \\
& \quad \text{n\_loops} = 6000000; \\
& \quad \text{for (i=0; i<n\_loops; i++)} \\
& \quad \quad \text{asm("nop");} \\
& \}}
\end{align*}
\]

Read \( C \)
Algorithmic Covert Timing Channel in Hierarchical Scheduling

$$\Pi_S \xrightarrow{\tau_{S,1}} \Pi_R$$

Input X: 0/1 \hspace{2cm} Output Y: 0/1

(a) $$\Pi_S$$ sends bit 0

Partition S

Partition R

$$\tau_{R,2}$$’s arrival

$$\tau_{R,1}$$

$$(a) \quad \tau_{S,1}$$

$$(b) \quad \tau_{R,2}$$

$$(c) \quad \tau_{R,1}$$

Read C \hspace{1cm} Read C

Time

void sender_job(int bit) {
  if (bit==1)
    n_loops = 10000000;
  else
    n_loops = 2000000;
  for (i=0; i<n_loops; i++)
    asm("nop");
}

int receiver1_job(void) {
  prev_c = shared_c;
  n_loops = 6000000;
  for (i=0; i<n_loops; i++)
    asm("nop");
  curr_c = shared_c;
  return curr_c - prev_c;
}
Algorithmic Covert Timing Channel in Hierarchical Scheduling

Partition S
\( \tau_{S,1} \)

Partition R
\( \tau_{R,1} \)
\( \tau_{R,2} \)
\( C \leftarrow C + 1 \)

\( t_0 \), \( t_1 \), \( t_2 \)

1. Read \( C \)
2. Read \( C \)

\( C \) remains same

void sender_job(int bit) {
    if (bit==1)
        n_loops = 1000000;
    else
        n_loops = 2000000;
    for (i=0; i<n_loops; i++)
        asm("nop");
}

int receiver1_job(void) {
    prev_c = shared_c;
    n_loops = 6000000;
    for (i=0; i<n_loops; i++)
        asm("nop");
    curr_c = shared_c;
    return curr_c - prev_c;
}

void receiver2_job(void) {
    shared_c++;
}
Algorithmic Covert Timing Channel in Hierarchical Scheduling

Input X: 0/1
Output Y: 0/1

Partition S

Partition R

W
\( C \leftarrow C + 1 \)

\( \tau_{S,1} \)

\( \tau_{R,1} \)

\( \tau_{R,2} \)

Time

\( t_0 \)

\( t_1 \)

\( t_2 \)

\( 1 \)

Read \( C \)

(b) \( \Pi_S \) sends bit 1

void sender_job(int bit) {
    if (bit==1)
        n_loops = 10000000;
    else
        n_loops = 2000000;
    for (i=0; i<n_loops; i++)
        asm("nop");
}

int receiver1_job(void) {
    prev_c = shared_c;
    n_loops = 6000000;
    for (i=0; i<n_loops; i++)
        asm("nop");
    return curr_c - prev_c;
}

void receiver2_job(void) {
    shared_c++;
}

\( W(b) \)

\( \mathcal{S} \) sends bit 1
Algorithmic Covert Timing Channel in Hierarchical Scheduling

Input \( X: 0/1 \)

Output \( Y: 0/1 \)

(b) \( \Pi_S \) sends bit 1

Partition S

 Partition R

\( t_{S,1} \)

\( t_{R,1} \)

\( t_{R,2} \)

Time

\( t_0 \)

\( t_1 \)

\( t_2 \)

\( C \leftarrow C + 1 \)

\( C \) has incremented by 1

\( \tau_{S,1} \)

\( \tau_{R,1} \)

\( \tau_{R,2} \)

\( \Pi_S \)

\( \Pi_R \)

\( W \)

\( \text{void sender_job(int bit) \{} \)

\( \text{if (bit==1) \{} \)

\( \text{n_loops = 1000000;} \)

\( \text{else} \)

\( \text{n_loops = 2000000;} \)

\( \text{for (i=0; i<n_loops; i++) asm("nop");} \)

\( \text{\}} \)

\( \text{int receiver1_job(void) \{} \)

\( \text{prev_c = shared_c; \} \)

\( \text{n_loops = 6000000;} \)

\( \text{for (i=0; i<n_loops; i++) asm("nop");} \)

\( \text{curr_c = shared_c; \} \)

\( \text{return curr_c - prev_c;} \)

\( \text{\}} \)

\( \text{int receiver2_job(void) \{} \)

\( \text{\}} \)

\( \text{\}} \)

\( \text{\}} \)

\( \text{\}} \)

\( \text{\}} \)
Blinder: Partition-Oblivious Hierarchical Scheduling

- Idea: making partition-local schedule deterministic no matter how other partitions behave
Blinder: Partition-Oblivious Hierarchical Scheduling

- Idea: making partition-local schedule deterministic no matter how other partitions behave

Defer $\tau_{i,2}$‘s release as if $\Pi_j$‘s preemption did not occur!
Blinder: Partition-Oblivious Hierarchical Scheduling

- Idea: making partition-local schedule deterministic no matter how other partitions behave.
Blinder: Partition-Oblivious Hierarchical Scheduling

- Idea: making partition-local schedule deterministic no matter how other partitions behave

\[ \tau_{i,1}, \tau_{i,2} \]

Physical time

[Diagram with arrows and labels]

\[ \Delta y \] is independent from the duration of \( \tau_{i,2} \). Thus, \( \tau_{i,1} \) always makes the same amount of progress.

\[ \Delta x + \Delta y = \text{the amount of time that } \tau_{i,1} \text{ would have executed for until } \tau_{i,2} \text{ arrives if the partition-level preemption by } \Pi_j \text{ did not happen.} \]
Blinder: Partition-Oblivious Hierarchical Scheduling

- Idea: making partition-local schedule deterministic no matter how other partitions behave
Blinder: Partition-Oblivious Hierarchical Scheduling

- Idea: making partition-local schedule deterministic no matter how other partitions behave
Blinder: Partition-Oblivious Hierarchical Scheduling

- Idea: making partition-local schedule deterministic no matter how other partitions behave

Guarantees that partition-local schedule always follows the partition’s canonical local schedule

The local schedule when no other partitions run

Canonical schedule being stretched out by higher-priority partitions
Lag-based Task Release

- Canonical local schedule cannot be statically constructed
  - Tasks may arrive at arbitrary times and have variable execution times
  - Also, these affect partition budget depletion and replenishment
- Hence, Blinder constructs canonical local schedule on the fly
Lag-based Task Release

- Canonical local schedule cannot be statically constructed
  - Tasks may arrive at arbitrary times and have variable execution times
  - Also, these affect partition budget depletion and replenishment
- Hence, Blinder constructs canonical local schedule on the fly

![Diagram showing task release modes: Normal and Deferred]
Lag-based Task Release

- Canonical local schedule cannot be statically constructed
- Tasks may arrive at arbitrary times and have variable execution times
- Also, these affect partition budget depletion and replenishment
- Hence, Blinder constructs canonical local schedule on the fly

\[
\begin{align*}
\tau_i, 1 & \quad \tau_i, 2 & \quad \tau_i, 3 \\
\end{align*}
\]

Deferred release mode

\[
\begin{align*}
\Pi_j & \quad \Pi_i \\
\Delta x & \quad \Delta y & \quad \Delta y \\
\end{align*}
\]

When the local schedule starts deviating from the canonical one.

Task release mode

Normal mode

Deferred mode
Lag-based Task Release

- Canonical local schedule cannot be statically constructed
- Tasks may arrive at arbitrary times and have variable execution times
- Also, these affect partition budget depletion and replenishment
- Hence, Blinder constructs canonical local schedule on the fly

\[ \tau_i, 1, 2, 3 \], \tau_1, 2, 3

Deferred release mode

Task release mode

Normal mode

Deferred mode

When the local schedule starts deviating from the canonical one.

When the local schedule is synchronized with the canonical one.
Lag-based Task Release

• When to release task?
  • When ‘lag’ becomes 0

\[ \tau_{i,1}, \tau_{i,2}'s \text{ arrival} \]

\[ t_0, t_1, t_2, t_3 \]

\( \Pi_j \) [Available Time]
\( \Pi_i \) [Used Time] [Lag]

Available time: the maximum amount of time that would have been available since entering into the deferred release mode

Used time: the amount of time that has actually been used since entering into the deferred release mode

Lag: available time – used time (i.e., the maximum amount of time by which the current local schedule lags behind the canonical schedule)
Lag-based Task Release

- When to release task?
  - When ‘lag’ becomes 0

\[
\tau_{i,2} \text{’s arrival}
\]

\[
\Delta y = \tau_{i,1}, \Delta y
\]

available time = \Delta y
used time = 0
lag = \Delta y

Available time: the maximum amount of time that would have been available since entering into the deferred release mode

Used time: the amount of time that has actually been used since entering into the deferred release mode

Lag: available time – used time (i.e., the maximum amount of time by which the current local schedule lags behind the canonical schedule)

Lag is reduced as long as partition runs.

Hence, the release of \(\tau_{i,2}\) is deferred until \(\Pi_i\) will have executed for \(\Delta y\).
Available Time Under Budget Constraint

- Available time can be limited by budget constraint
  - Assuming all budgets are used up as soon as they become available

\[
\begin{align*}
\text{Remaining budget until the next replenishment or } t & = a_1 + a_2 + a_3 \\

a_1 &= \min\left[B_{\text{def}}, (t - t_{\text{def}}), (t_{\text{rep}} - t_{\text{def}})\right] \\

\text{Full-budget replenishments} & \quad a_2 = \left[\frac{t - t_{\text{rep}}}{T_i}\right]B_i \\

\text{Last replenishment} & \quad a_3 = \min\left[B_i, (t - t_{\text{rep}}) - \left[\frac{t - t_{\text{rep}}}{T_i}\right]T_i\right]
\end{align*}
\]
Real-time Schedulability

• Will real-time tasks meet their deadlines even with Blinder?

• If partition is schedulable*, Blinder does not increase the worst-case response time of tasks
  • That is, tasks were schedulable without Blinder → they are still schedulable with Blinder
  • See the proof for Theorem 2 in the paper

*Schedulable iff worst-case response time <= deadline

* Partition $\Pi_i$ is said to be schedulable if it is guaranteed to execute for $B_i$ over every replenishment period $T_i$
Implementation and Use Case

- 1/10th-scale self-driving car
  - OS: LITMUS\textsuperscript{RT}
    - Ubuntu 16.04 with kernel version 4.9.30
    - Global: Sporadic-polling server, Partition-local: fixed-priority preemptive scheduling
  - Platform: Intel NUC mini PC
    - Intel Core i5-7260U processor (@ 2.20 GHz) + 8 GB main memory
Evaluation – Impact on Responsiveness

- Synthetic partition set

<table>
<thead>
<tr>
<th></th>
<th>$\tau_{i,1}$</th>
<th>$\tau_{i,2}$</th>
<th>$\tau_{i,3}$</th>
<th>$\tau_{i,4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Pi_1$ (20,4$\alpha$)</td>
<td>(40,2$\beta$)</td>
<td>(80,4$\beta$)</td>
<td>(160,8$\beta$)</td>
<td>(320,16$\beta$)</td>
</tr>
<tr>
<td>$\Pi_2$ (30,6$\alpha$)</td>
<td>(60,3$\beta$)</td>
<td>(120,6$\beta$)</td>
<td>(240,12$\beta$)</td>
<td>(480,24$\beta$)</td>
</tr>
<tr>
<td>$\Pi_3$ (40,8$\alpha$)</td>
<td>(80,4$\beta$)</td>
<td>(160,8$\beta$)</td>
<td>(320,16$\beta$)</td>
<td>(640,32$\beta$)</td>
</tr>
<tr>
<td>$\Pi_4$ (50,10$\alpha$)</td>
<td>(100,5$\beta$)</td>
<td>(200,10$\beta$)</td>
<td>(400,20$\beta$)</td>
<td>(800,40$\beta$)</td>
</tr>
</tbody>
</table>

- Various rate groups
- Task inter-arrival times are allowed to vary by 20%
- Compare
  - T: Static partitioning (TDMA),
  - P: Non-static partitioning without Blinder
  - N: Non-static partitioning with Blinder
- Collected task response times for 10 hours/config

System load=80% when $\alpha = \beta = 1$ (Replenishment period, Budget) → (Min inter-arrival time, Exec time)
Evaluation – Impact on Responsiveness

- Blinder does not increase the WCRTs

\[ \tau_{i,1}, \tau_{i,2}, \tau_{i,3}, \tau_{i,4} \]

- \( \tau \): static, \( \Pi \): non-static w/o Blinder, \( N \): non-static w/ Blinder
Evaluation – Impact on Responsiveness

- Blinder does not increase the WCRTs
- But, Blinder may increase the average-case response times
  - In particular of high-priority tasks
  - Due to the lag-based deferred release
- Impact on low-priority tasks is small
  - Delayed by higher-priority ones anyway
Summary

• Blinder closes algorithmic covert timing channel over hierarchical scheduling
  • By making partition-local schedules deterministic (i.e., lag-based task release)

• Blinder achieves
  • Flexible CPU resource usage (that non-static partitioning achieves)
  • Partition obliviousness (that static partitioning achieves)

• Blinder is backward-compatible and minimally-intrusive
  • No modification is required to global and local scheduling policies
  • Advantageous to existing safety-critical systems that require high re-certification costs

✉️ man-ki.yoon@yale.edu
Appendix
Algorithmic Covert Timing Channel in Hierarchical Scheduling

- Multi-bit channel
  - $\tau_{R,2}$ acts as a regular tick counter
  - Message = length of preemption

![Diagram showing τ_{R,1} and τ_{R,2} with time intervals and counter changes](image)

- \(\tau_{R,1}\) and \(\tau_{R,2}\) with time intervals showing counter changes
- Partition S and Partition R with time intervals
- (a) $\Delta$counter = 6
- (b) $\Delta$counter = 9
Blinder: Partition-Oblivious Hierarchical Scheduling

- Idea: making partition-local schedule deterministic no matter how other partitions behave

Guarantees that tasks are released at **deterministic partition-local times**

**Canonical schedule**

**Actual schedule**
Example Schedule Trace (1)

• From LITMUS\textsuperscript{RT}
• $T_L = 10 \text{ ms}, B_L = 7 \text{ ms}$

\begin{itemize}
  \item Normal mode $\rightarrow$ Deferred mode
    \[ B_{\text{def}} = 7 - 5 = 2 \text{ ms} \]
  \item available\textsubscript{L}(t = 21) = 2 + 1 \text{ ms}, used\textsubscript{L} = 0 \text{ ms}
    \[ \rightarrow \text{lag}_{L,2} = 3 \text{ ms} \]
  \item lag\textsubscript{L,2} becomes 0 $\rightarrow$ $\tau_{L,2}$ is released
\end{itemize}
Example Schedule Trace (2)
Example Schedule Trace (2)

Long Preemption

Short Preemption

Without Blinder

With Blinder
Evaluation – Impact on Responsiveness

- TDMA (non-static partitioning) degrades the average responsiveness

\[ \tau_{i,1}, \tau_{i,2}, \tau_{i,3}, \tau_{i,4} \]

\[ T: \text{static, P: non-static w/o Blinder, N: non-static w/ Blinder} \]
Evaluation – Impact on Responsiveness

- The highest-priority partition ($\Pi_1$) is not affected by Blinder
  - Because it never enters the deferred release mode
Evaluation – Impact on Responsiveness

- When partition is not schedulable, tasks may experience longer WCRT
  - System is ill-configure

Probability distribution of $\tau_{4,1}$’s response times

(a) $\Pi_4$ is schedulable

(b) $\Pi_4$ is not schedulable

($=\Pi_4$ cannot guarantee $B_4$ over every $T_4$)
Evaluation – Scheduling Overhead

\[ \tau_{H,1} \]

\[ \tau_{L,1} \quad \tau_{L,2} \quad \cdots \quad \tau_{L,#LA+1} \] (lower-priority tasks arrive)

\[ \Pi_H \]

\[ \Pi_L \]

10 14 44 45 47

Time

Normal release mode
Deferred release mode

Impacts of
1) Number of partition-level preemptions
2) Size of arrival queue
Evaluation – Scheduling Overhead

- Number of partition-level preemptions
  - More preemptions = more frequent timer updates

<table>
<thead>
<tr>
<th>#HP</th>
<th>12</th>
<th>18</th>
<th>24</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LA</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Preemptions of 1 ms every 2 ms (#HP=12,18,24,30)

- Normal release mode

- Deferred release mode

Average response time of $\tau_{L,1}$ (in ms) and stdev
Evaluation – Scheduling Overhead

- Size of arrival queue
  - More tasks in arrival queue = more frequent lag update

<table>
<thead>
<tr>
<th>#HP</th>
<th>12</th>
<th>18</th>
<th>24</th>
<th>30</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>#LA</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w/o Blinder</td>
<td>96.15 (0.04)</td>
<td>96.17 (0.06)</td>
<td>96.19 (0.07)</td>
<td>96.22 (0.09)</td>
<td>96.22 (0.08)</td>
</tr>
<tr>
<td>w/ Blinder</td>
<td>96.17 (0.04)</td>
<td>96.19 (0.05)</td>
<td>96.22 (0.06)</td>
<td>96.24 (0.08)</td>
<td>96.25 (0.07)</td>
</tr>
</tbody>
</table>

Average response time of $\tau_{L,1}$ (in ms) and stdev