Supporting Low-Latency, Low-Criticality Tasks in a Certified Mixed-Criticality OS

Manohar Vanga
Björn Brandenburg
Andrea Bastoni
Henrik Theiling
Mixed-Criticality Systems

- Increasing SWaP constraints
- Consolidation of components with different criticalities onto shared hardware platforms
- Mixed-Criticality Systems
Mixed-Criticality Systems

Key Challenges

- Ensuring isolation between tasks of different criticalities.
- Ensuring that throughput and latency requirements of all tasks are met.
What is Criticality?
What is Criticality?

Level of **failure assurance** a task is certified for.

- e.g. **DAL levels** in DO-178B/C
Criticality vs. Importance

What is Criticality?
Level of **failure assurance** a task is certified for.
e.g. **DAL levels** in DO-178B/C

What is "importance"?
Any feature that is crucial to the **commercial success** of a product.
e.g. **touch GUI** in cars
Criticality vs. Importance

**What is Criticality?**

Level of *failure assurance* a task is certified for.

*Example:* DAL levels in DO-178B/C

**What is "importance"?**

Any feature that is crucial to the *commercial success* of a product.

*Example:* touch GUI in cars

**Key point:** a task may be of low criticality but still important!
A Case Study

RTOS consisting of a hypervisor-based separation microkernel designed for the highest levels of safety and security.

Deployed across many safety-critical domains including avionics, automotive, and transportation applications.

Certified on a wide range of projects using various certification standards, including DO-178B/C, IEC 61508, EN 50128.
A Case Study

PikeOS

Strong, battle-tested support for high-criticality tasks.

the highest levels of safety and security.

Deployed across domains and transportation applications.

Certified on a wide range of projects using various including DO-178B/C, IEC 61508, EN 50128.
A Case Study

Strong, battle-tested support for **high-criticality tasks**.

Deployed across domains and transportation applications.

Certified on a wide range of projects using various standards including DO-178B/C, IEC 61508, EN 50128.

How can we integrate support for **low-criticality tasks**?
Our Paper: A Summary

Identify **deficiencies in low-criticality support** in PikeOS.

Highlight **key design constraints** required in a commercial context, and **typically not addressed in academic designs**.

Present a **minimally-invasive extension of the PikeOS scheduler** to address the determined deficiencies.

Design and implementation of a **prototype in PikeOS**, with results from a **freely-shareable re-implementation in LITMUS\textsuperscript{RT}**.
Rest of this Talk

I  The problem of low-criticality, low-latency tasks

II Working within real-world design constraints

III Our proposed scheduler extensions
The problem of low-criticality, low-latency tasks

Working within real-world design constraints

Our proposed scheduler extensions
Application Tasks are Assigned to Time Partitions

Time partitioned scheduling

Every task must be certified to the same level of assurance as the highest criticality task it interferes with.
Fixed-Priority Scheduling Within Time Partitions

Per-Time-Partition Structures

FIFO Queue
Application Tasks are Assigned to Time Partitions
A Static Scheduling Table is Generated
System Tasks are Placed in Always-Eligible TP0

TP0 tasks are prioritized over app tasks at the same priority.

High criticality

Always eligible to run.
System Tasks are Placed in Always-Eligible TP0

Rest of the talk: TPx
# Criticality and Latency Requirements

<table>
<thead>
<tr>
<th></th>
<th>Low Latency Required</th>
<th>High Latency Acceptable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High Criticality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Low Criticality</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
High-Criticality Tasks + Low Latency

Examples

Safety-critical event handlers
High-rate, sensor-data retrieval tasks
High-Criticality Tasks + High Latency

Examples

Computation-heavy, mission-critical planning tasks
High-Criticality Tasks + High Latency

Choice depends on tradeoff between acceptable latency bound and system performance.

Examples

Computation-heavy, mission-critical planning tasks
Low-Criticality Tasks + High Latency

<table>
<thead>
<tr>
<th></th>
<th>Low Latency Required</th>
<th>High Latency Acceptable</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Criticality</td>
<td>$TP_0$</td>
<td>$TP_0$ or high-criticality $TP_x$</td>
</tr>
<tr>
<td>Low Criticality</td>
<td></td>
<td>Low-criticality $TP_x$</td>
</tr>
</tbody>
</table>

**Examples**

Navigation or route planning tasks
Low Latency + Low-Criticality Tasks (L3C Tasks)

Examples
Low-latency, user interface tasks
Experiment: Latency vs. System Performance

Static Schedule

- UDP Echo Server
- Matrix Multiplication Task

**L3C task**

**Hard real-time task**
Experiment: Latency vs. System Performance

Static Schedule

- **L3C task**
- **Hard real-time task**

Varying slot sizes

- UDP Echo Server
- Matrix Multiplication Task

We measured: **UDP echo latency**

We measured: **LLC misses**
Evaluation Setup

Raspberry Pi 3b

- Broadcom BCM2837.
- 64-bit ARM Cortex-A53 @ 1.2GHz.
- 1GB RAM

LITMUS$RT$ 2017.1 with Linux 4.9.30
Experiment: Latency vs. System Performance

![Graph showing latency vs. system performance. The x-axis represents slot size (in ms) ranging from 0 to 100, and the y-axis represents UDP echo latency (in ms) ranging from 0 to 100. The line on the graph indicates a linear increase in latency with increasing slot size.]
Experiment: Latency vs. System Performance

UDP echo latency reduces with smaller slot size
Tradeoff: Latency vs. System Performance

Large Slots

| T1 | T2 | T1 | T2 |

Small Slots

| T1 | T2 | T1 | T2 | T1 | T2 | T1 | T2 |
Tradeoff: Latency vs. System Performance

Large Slots

Small Slots

Latency incurred by tasks reduces with smaller slot sizes
Experiment: Latency vs. System Performance

Graph showing the relationship between slot size (in ms) and HRT L1 misses/sec, as well as UDP echo latency (in ms).
L3C tasks requiring < 10ms latency cannot be placed within a TPx without affecting the rest of the system.
Tradeoff: Latency vs. System Performance

Large Slots

Small Slots

Number of preemptions increases with smaller slot sizes
(loss of cache affinity, more scheduler invocations)
Other Possibilities?
Other Possibilities?

L3C tasks cannot be placed at a high priority in TP0 without causing potentially unbounded interference on high-criticality tasks.
Other Possibilities?

L3C tasks cannot be placed at a high priority in TP0 without causing potentially unbounded interference on high-criticality tasks.

L3C tasks cannot be placed at a low priority in TP0 (criticality-monotic priority assignment) without itself incurring significant interference.
Rest of this Talk

1. The problem of low-criticality, low-latency tasks
2. Working within real-world design constraints
3. Our proposed scheduler extensions
Design Constraints

We present nine key design constraints in the paper!
Design Constraints

We present nine key design constraints in the paper!

• Strictly opt-in for OS customers.
• Minimally intrusive for the OS vendor.
• Optionally strict freedom-from-interference.
Design Constraints

We present nine key design constraints in the paper!

• Strictly opt-in for OS customers.
• Minimally intrusive for the OS vendor.
• Optionally strict freedom-from-interference.
Strictly Opt-In for Customers

Customers bear significant costs

- **Design and implementation** of product components.
- **Configuration, testing, and certification** of production systems.
- Adoption of specific workflows and tools.
Strictly Opt-In for Customers

Customers bear significant costs

- Design and implementation of product components.
- Configuration, testing, and certification of production systems.
- Adoption of specific workflows and tools.

By default, customers should require no changes. All L3C support features should be strictly opt-in.
Strictly Opt-In for Customers

Customers bear significant costs

- **Design and implementation** of product components.
- **Configuration, testing, and certification** of production systems.
- Adoption of specific workflows and tools.

By default, customers should require no changes. All L3C support features should be strictly opt-in.

It should be easy to opt-in incrementally to new L3C-support features. That is, without significant changes to designs and workflows.
Design Constraints

- Strictly opt-in for OS customers.
- **Minimally intrusive for the OS vendor.**
- Optionally strict freedom-from-interference.
Minimally Intrusive for the OS Vendor

RTOS vendors bear significant costs too!

Documentation efforts for certification process.

Adoption of specific workflows and tools.
Minimally Intrusive for the OS Vendor

RTOS vendors bear significant costs too!

- **Documentation efforts** for certification process.
- Adoption of specific **workflows and tools**.

**Cannot radically re-design** core parts of the system triggering an expensive re-certification process.
Minimally Intrusive for the OS Vendor

RTOS vendors bear significant costs too!

Documentation efforts for certification process.

Adoption of specific workflows and tools.

Cannot radically re-design core parts of the system triggering an expensive re-certification process.

Need to provide continued legacy support for existing customers with products out in the market.
Minimally Intrusive for the OS Vendor

RTOS vendors bear significant costs too!

- **Documentation efforts** for certification process.
- Adoption of specific **workflows and tools**.

**Cannot radically re-design** core parts of the system triggering an expensive re-certification process.

Need to provide continued legacy support for **existing customers** with products out in the market.

Any introduced changes **must minimize re-certification costs** if they hope to be adopted.
Design Constraints

- Strictly opt-in for OS customers.
- Minimally intrusive for the OS vendor.
- Optionally strict freedom-from-interference.
Optional Strict Freedom-from-Interference

It must be possible to achieve strict freedom-from-interference.

Freedom from unbounded interference is acceptable if bounds are known and enforced by OS (and enforcement mechanism validated at a high level of assurance).
Optional Strict Freedom-from-Interference

It must be possible to achieve strict freedom-from-interference.

Freedom from unbounded interference is acceptable if bounds are known and enforced by OS (and enforcement mechanism validated at a high level of assurance).

For certain maximum-importance tasks, it must be possible to achieve **strict freedom from (all) interference**.
Optional Strict Freedom-from-Interference

It must be possible to achieve strict freedom-from-interference.

**Freedom from unbounded interference is acceptable** if bounds are known and enforced by OS (and enforcement mechanism validated at a high level of assurance).

For certain maximum-importance tasks, it must be possible to achieve **strict freedom from (all) interference**.

Even when opting in to L3C support features, for certain tasks, it **should be possible to selectively opt-out**.
Optional Strict Freedom-from-Interference

It must be possible to achieve strict freedom-from-interference.

Freedom from unbounded interference is acceptable if bounds are known and enforced by the OS and validated at a high level of assurance.

Example
EDF-VD requires all tasks in the system to be under the EDF-VD scheduler.

For certain maximum-importance tasks, it must be possible to achieve strict freedom from (all) interference.

Even when opting in to L3C support features, for certain tasks, it should be possible to selectively opt-out.
Rest of this Talk

I The problem of low-criticality, low-latency tasks

II Working within real-world design constraints

III Our proposed scheduler extensions
Key Problem

Ideally: want to place L3C tasks at a **high priority in TP0**
Key Problem

Ideally: want to place L3C tasks at a **high priority in TP0**

**Tasks achieve low latency**
(TP0 always eligible to run, immune even from higher-criticality interference)
Ideally: want to place L3C tasks at a **high priority in TP0**

- **Tasks achieve low latency**
  - (TP0 always eligible to run, immune even from higher-criticality interference)

- Can cause **potentially unbounded interference** on high-criticality tasks
  - (both in TP0 and other TPx's)
Key Problem

Ideally: want to place L3C tasks at a high priority in TP0

Tasks achieve low latency
(TP0 always eligible to run, immune even from higher-criticality interference)

Can cause potentially unbounded interference on high-criticality tasks
(both in TP0 and other TPx's)

Solution: bound interference from L3C tasks via reservations.
Fixed Priorities in TP0

Per-priority ready queues:

- 255
- 254
- ...
- 1
- 0

Connections:
- 255 → NULL
- 254 → Task
- 1 → Task
- 0 → NULL
Adopted Solution

1. Designate certain priorities in TP0 as "EDF bands"
Adopted Solution

1. Designate certain priorities in TP0 as "EDF bands"

   Per-priority ready-queues
   
   | 255 | 254 | 253 (EDF) | ... | 1 | 0 |
   | NULL | Task | L3C Task | Hard CBS server | NULL | Task |

2. Contain each L3C task within reservation

   Deferrable server
Adopted Solution

1. Designate certain priorities in TP0 as "EDF bands"
2. Contain each L3C task within reservation
3. Dispatch L3C tasks via EDF instead of FIFO
Designation of priorities in TP0 as EDF bands is strictly optional.
Adherence to Design Constraints

Strict freedom-from-interference from L3C tasks.

Maximum-importance tasks may still be placed at higher priorities to avoid all L3C interference.
Adherence to Design Constraints

PikeOS Architecture

PikeOS Core

Fixed-Priority Scheduler
Adherence to Design Constraints

PikeOS Architecture

- PikeOS Core
  - Fixed-Priority Scheduler

PikeOS with EDF

- PikeOS Core
  - Fixed-Priority Scheduler
  - Plugin Interface
Minimally intrusive for OS vendors.

Can certify the plugin interface as part of the core kernel and amortize the cost across multiple customers.
Adherence to Design Constraints

PikeOS Architecture

- PikeOS Core
  - Fixed-Priority Scheduler

PikeOS with EDF

- PikeOS Core
  - Fixed-Priority Scheduler
  - Plugin Interface
  - EDF scheduling plugin
Adherence to Design Constraints

PikeOS Architecture

- PikeOS Core
  - Fixed-Priority Scheduler

PikeOS with EDF

- PikeOS Core
  - Fixed-Priority Scheduler
  - Plugin Interface
  - EDF scheduling plugin
  - Reservation plugins
Adherence to Design Constraints

PikeOS Architecture

PikeOS Core

Fixed-Priority Scheduler

PikeOS with EDF

PikeOS Core

Fixed-Priority Scheduler

Plugin Interface

EDF scheduling plugin

Reservation plugins

Can be changed without impacting certifiability of the core kernel.

Can even be implemented in a separate address space or as a user-space thread.
Individual L3C tasks don't need to be certified. Certifying the enforcement mechanism is sufficient.
System Configuration

No changes required for maximum-importance tasks
Need to **inflate budgets of lower- or equal-priority TP0 and TPx tasks (and change partition table)**.
Does it Work?
## Experimental Setup

<table>
<thead>
<tr>
<th>TPx</th>
<th>Hard real-time task</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(lowest priority)</td>
</tr>
</tbody>
</table>

| TP0                | UDP server (L3C task, high priority) | Hard real-time task (medium priority) |

- Measure **response time of HRT tasks** over time
- ...while **varying request load** on UDP server.
Does it Work?

L3C task at high priority in TP0

No load (zero requests per second)

Response time of TP0 task

Response time of TPx task
Does it Work?

L3C task at high priority in TP0

Normal load: 10 req/sec

Response time of TP0 task

Response time of TPx task
Does it Work?

L3C task at high priority in TP0

Normal load: 10 req/sec

Response time of TP0 task

Response time of TPx task

No undue interference on TP0 and TPx tasks under normal load.
Does it Work?

L3C task at high priority in TP0

10 req/sec + bursts

Response time of TP0 task

Response time of TPx task
Does it Work?

L3C task at high priority in TP0

10 req/sec + bursts

L3C task causes **response time spikes** in lower-priority tasks across the system.
Does it Work?

L3C task at high priority in TP0 within reservation
(budget 1ms, period 50ms, determined empirically)

10 req/sec + bursts

Response time of TP0 task

Response time of TPx task
Does it Work?

L3C task at high priority in TP0 within reservation (budget 1ms, period 50ms, determined empirically)

10 req/sec + bursts

The encapsulating reservation prevents bursts from affecting other parts of the system.
A Case Study

Identified **a deficiency in L3C support** in PikeOS and showed why they are difficult to support under the existing scheduler.

Highlighted **key design constraints** required in a commercial context, **typically not addressed in academic designs**.

Showed that careful integration of **reservation-based scheduling** best suited the constraints of an existing high-criticality RTOS.

Implemented a **prototype in PikeOS**, and presented results from a **freely-shareable re-implementation in LITMUS\textsuperscript{RT}**.

Thanks!

https://litmus-rt.org