Mixed-Criticality Support in a High-Assurance, General-Purpose Microkernel

Anna Lyons, Gernot Heiser

UNSW Australia & NICTA
Could be OS guests
Untrusted
Not critical

Trusted
Critical

seL4

Single core (for now)
Has memory management unit
Untrusted
Not critical

Trusted
Critical

seL4
Untrusted
Not critical

Shared resource

Shared resource

Trusted
Critical

seL4
seL4

- Functional Correctness [SOSP’09]
- Integrity [ITP’11]
- Timeliness (known WCET) [RTSS’11, EuroSys’12]
- Translation Correctness [PLDI’13]
- Non-interference [S&P’13]
- Fast (258 cycle IPC roundtrip on 1GHz Cortex–A9)
- Minimal TCB (~9000 SLoC)

Safety: specifically temporal properties.
Goals of this work

• Real-time scheduling support
• Temporal isolation (beyond total static partitions)
• Asymmetric temporal protection
  • support for criticality mode changes
• Bounded resource sharing
  • across criticalities
Mechanisms

1. Scheduling contexts
2. Thread criticalities
3. Temporal exceptions
This talk

1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing
1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing
seL4 design principles

- Minimality principle
- Fast
- Possible to verify
  - avoid concurrency
  - avoid unnecessary complexity
  - kernel should not require re-verification if user-level changes
What is a capability?

- unforgeable access token
- stored in the \texttt{c-space} of an app
  - threads can share c-spaces
- \texttt{invoked} by user-level to perform an action
  - no capability, no action
- can be copied, moved between c-spaces
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message.
seL4 basics: sync endpoints

Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message
Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message
seL4 basics: sync endpoints

Synchronous endpoints: essentially message ports, which senders/waiters queue on until both are present to receive a message
Async endpoints (AE): essentially message ports, which accumulate messages until a waiter is present. Waiters queue until a message is present.
Async endpoints (AEP): essentially message ports, which accumulate messages until a waiter is present. Waiters queue until a message is present.

A bound async endpoint has a special 1:1 relationship with a thread — and only the bound thread is allowed to wait a bound AEP.
seL4 Memory Model

Initial Task

1GB

512MB

4KB 4KB

4KB 4KB

seL4
seL4 Memory Model

Initial Task

1GB

512MB

4KB

4KB

4KB

seL4
Meet seL4: Summary

- capability based
- communication via endpoints
  - synchronous or asynchronous
- all resources managed at user-level
- initial task gets capabilities to everything in the system
1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing
Resource kernels*

- Timeliness of resource access
  - reservations
- Efficient resource utilisation
- Enforcement & Protection
- Access to multiple resource types

* [Rajkumar et al. 2001]
Resource kernel mechanisms

• Admission
• Scheduling
• Enforcement
• Accounting

Which mechanisms belong in a microkernel?
Resource kernel mechanisms

- Admission (policy)
- Scheduling
- Enforcement
- Accounting
Scheduling Contexts

- Implements processor "reservation"
- adapted from Fiasco [Steinberg 2010]
- Upper bound
- No priority
- Rate = e / p
- **Full** or **Partial**
- Only 1 per thread
Full reservations

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>253</th>
<th>254</th>
<th>255</th>
</tr>
</thead>
</table>

- $e = 4$
- $p = 4$
- $t_1$

- $e = 5$
- $p = 5$
- $t_2$

- $e = 4$
- $p = 4$
- $t_3$
Partial reservations

Scheduling contexts act as sporadic servers
Partial reservations

\[ e = 2 \]
\[ p = 4 \]

Release Queue

\[ t_1 \]

Scheduling contexts act as sporadic servers
Admission

- New **control** capability, `seL4_SchedControl`.
- Controls population of scheduling context parameters.
- Must take into account priorities.
Scheduling
Basic Rate Monotonic

![Diagram showing scheduling and basic rate monotonic with tasks and percentages.](image)
Scheduling
Low priority tasks in slack

\[
\begin{align*}
0 & \quad 1 & \quad 2 & \quad 3 & \quad \ldots & \quad 253 & \quad 254 & \quad 255 \\
\downarrow \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow & \quad \downarrow \\
\text{\textcolor{green}{t_3}} & \quad \text{\textcolor{green}{t_3}} & \quad \text{\textcolor{green}{t_2}} & \quad \text{\textcolor{green}{t_1}} \\
\text{e = 4} & \quad \text{e = 4} & \quad \text{e = 20} & \quad \text{e = 5} \\
p = 20 & \quad p = 20 & \quad p = 40 & \quad p = 30
\end{align*}
\]
Time as a resource: summary

• scheduling contexts
  – full or partial
  – act as upper bounds
  – disjoint from priority

• user-level admission
  – allows for mixed RT/RR scheduling
  – not full flexibility of user-level scheduling
This talk

1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing
Task model

```c
while (1) {
    /* job release */
    doJob();
    /* job completion */
    seL4_Wait(bep);
}
```

If job completion does not occur before the budget expires, send a **temporal exception** or rate-limit.

Bound async endpoint where device interrupts, async messages or kernel timer trigger job release.
Criticality

- New thread field
- Range set at compile time
- `seL4_SetCriticality`
  - invokes `sched_control` cap
- HI $\rightarrow$ LO is lazy
- LO $\rightarrow$ HI is immediate, and $O(n)$
Criticality mode change

• Assumptions:
  • infrequent (if they occur at all)
  • short in duration

• Kernel provides ability to
  • change params of excepting thread
  • postpone all lower criticality threads
  • alter priorities of threads
Asymmetric Protection

Low Criticality  High Criticality

SchedControl_Extend()
SchedControl_SetCriticality()
Asymmetric Protection

Low Criticality  High Criticality

0  1  2  3  ...  252  253  254  255

Restores **criticality** when system is idle
Criticality: Summary

- **Temporal exceptions**
  - optional (not required for rate-based threads)
  - handler must have own budget

- New thread field: `criticality`

- New kernel invocation: `set criticality`
  - although `temporal exception` handler can take other actions
This talk

1) seL4 concepts
2) Time as a resource
3) Mode switch support
4) Resource sharing
Resource Sharing

seL4_Call  seL4_Wait

seL4_ReplyWait

Thread  Resource Server
NCP vs. PIP vs HLP vs PCP

Priority Inversion Bound

Implementation Complexity

Priority Ceiling Protocol

Priority Inheritance Protocol

Highest Lockers Protocol

Non-preemptive Critical Sections

Priority Inversion Bound
Resource Sharing

Thread

seL4_Call  seL4_Wait

Resource Server

seL4_ReplyWait
Resource Sharing

Thread

\texttt{seL4\_ReplyWait}

Resource Server
Resource Sharing

seL4_Call

???

e

seL4_ReplyWait

Resource Server

Thread
NCP vs. PIP vs HLP vs PCP

- Priority Ceiling Protocol
- Priority Inheritance Protocol
- Highest Lockers Protocol
- Non-preemptive Critical Sections
Active Servers (no temporal isolation)
Active Servers (no temporal isolation)

seL4_Call

Server
Active Servers (no temporal isolation) seL4_ReplyWait
Active Servers (no temporal isolation)

seL4_Call
Scheduling context donation

· **seL4_Call**
  - where server is passive, donate scheduling context to server, otherwise do nothing
  - Must *trust* the server (use async for untrusted)

· **seL4_ReplyWait**
  - donates it back
  - reply cap represents a guarantee that the scheduling context will be returned
Scheduling context donation

Server

seL4_Wait

e

A

B
Scheduling context donation

seL4_Call

A

B

Server
Scheduling context: donation

A

B

Server
Scheduling context **donation**

\[\text{seL4\_ReplyWait}\]

- A
- B
- Server
Summary: Resource sharing (so far)

- **Scheduling context donation**
  - only on Synchronous IPC with atomic send/recv operation

- **Active and passive servers**
  - Passive servers must always be trusted
Budget Expiry

Server
Budget Expiry

A

B

e

Server
Alternatives for budget expiry

- Multithreaded servers
  - COMPOSITE [Parmer 2010]
  - possible with our impl.
- Bandwidth Inheritance + helping
  - Fiasco [Steinberg et.al. 2010]
  - we avoid this to avoid dependency trees/chains
- Temporal exceptions!
Exception + Rollback

Server

Temporal fault handler
Exception + Rollback

A

B

Server

Temporal fault handler
Criticality change

B (LO criticality)

A (HI criticality)

Server (HI criticality)

Temporal fault handler
Exception + rollback

- Other actions possible on exception
  - like emergency reservation
- Rollback propagates to handle chains:
  - if a reply transfers an empty scheduling context, another temporal exception is raised
- User must implement rollback
  - middleware layer can do this
Summary: Resource sharing

- Multithreaded servers possible
- Budget expiry triggers temporal exceptions – which can be used to rollback or help a server
- So does criticality change – if lower criticality thread using server
Endgame

- Temporal isolation, asymmetric protection, safe bounded resource sharing achieved through scheduling contexts, criticality, temporal exceptions.

sel4
References + Credits
References


References


• Fiasco. [http://os.inf.tu-dresden.de/fiasco/overview.html](http://os.inf.tu-dresden.de/fiasco/overview.html)

Image + Font Credits

- Fonts sourced from [Font squirrel](#)
- All other images are in the public domain (mostly from openclipart)