G(IP)$^2$C

Temporally Isolated IPC with Server-To-Server Invocations

Cédric Courtaud, Björn Brandenburg

23/05/12, RTAS’23 San Antonio, TX
An Alternative to Locking
Suitable for Mixed-Criticality Systems

Locking protocol

Acquire lock

Release lock

Prepare food

Locking requires trust!

An untrusted client can

ignore the lock

Leave the resource in an inconsistent state

“Forget” to release the lock

Ruin everybody’s meal
A Better Approach: Resource Servers

Delegating critical section provides **logical** fault isolation.

What about **temporal** isolation?
What About Temporal Isolation?

Can we satisfy time-critical clients even if

- Some clients try to monopolize the server?
- The total number of clients is not known a priori?
Preventing Resource Server Flooding
With Reservation-Based Scheduling

Each client is encapsulated in a reservation with a CPU time budget and a priority.

A reservation is active if one of its clients has a pending job.

A reservation can be scheduled if it is active and has non-zero budget.

A scheduled reservation drains budget at unit rate and ceases to be scheduled when its budget is depleted.

Clients cannot monopolize the processor.
What About Temporal Isolation?

Resource Server

Server must reply in bounded time

Can we satisfy time-critical clients even if

Yes (with reservation-based scheduling)

Some clients try to monopolize the server?

How to provision the budget of time-critical clients?

The total number of clients is not known a priori?
Prior Work: MC-IPC protocol

"A synchronous IPC protocol for predictable access to shared resources in mixed-criticality systems." RTSS 2014

Resource servers **migrate** in the reservation of their clients.

Queuing structure inspired by real-time locking

Clients embedded in **reservations** with processor time budget

Combine techniques from

- **Reservation-Based Scheduling**
- **Real-Time Locking**
- **Mixed-Criticality Microkernels**

**Isolation Properties**

**Bounded Interference**
Bound on budget drained for IPC calls
That is **independent** of the number of clients

**Independence Preservation**
No interference from unrelated requests

**Main limitation**

Time-critical clients in reservations can be provisioned with enough budget to satisfy their requests

No support for server-to-server (S2S) requests
Nesting is everywhere
A Motivational Example
Real-Time Secure Database

1. Query database
2. Fetch file
3. Fetch block
4. Decipher block
5. Sign result
Nesting Is Painful

Servers can be called **individually**

**Nested** requests can be **delayed**

Can lead to **exponential** budget drain
Our Contribution

The **Group-Independence-Preserving IPC Protocol**

- **Supports S2S requests**
- **Bounded Interference**
- **Group-Independence Preservation**
- **Deadlock-free**

Based on techniques from:

- **Reservation-based scheduling**
- **Nested Real-time Locking**
- **Mixed-Criticality Microkernels**
- **Budgeting**
- **RNLP**
- **GIPP**
- **Passive Servers**
Paper Content

Extensions
- Background tasks support
- Muti-occupancy reservations

Abortion Rules
- Handling budget exhaustion during IPC

Progress Rules
- Scheduling context transfer

Sequencing Rules
- Concurrent request ordering

System Model
- Main entities involved in the protocol

This talk
System Model

Each server belongs to a server group.

- Limitation lifted in the paper.

- m processors
- n reservations
- Partitioned scheduling
- 1 client per reservation*
- q resource servers
Synchronous IPC API

Clients can invoke any server

Servers can invoke other servers if they belong to the same group

C2S request emits S2S request

Clients can invoke any server

Forbidden inter-group S2S requests
**C2S Request Lifecycle**

- **Emitted C2S request**
  - Aborted if client exhausts its reservation budget

- **Committed C2S request**
  - Cannot be aborted anymore

- **Completed C2S request**
  - Possible S2S requests

- gip_invoke
- gip_wait
- gip_reply
C2S Request Representation

- C2S requests progress tracked by **IPC contexts**
- IPC context contains a **request stack** and **tracking metadata**
- Servers pick requests **from the request stack**
C2S Requests
Sequencing
1. Each server group has a **global group queue** of IPC contexts

2. IPC contexts enqueued in FIFO order in the group queue.

3. Resource servers traverse the group queue and commit requests as soon as possible
The Problem with the Straw-Man Approach

How to bound the group queue? How to avoid interference from later requests?

1. Group queue length bounded by unknown number of clients.

2. IPC contexts enqueued in FIFO order in the group queue.

3. Resource servers traverse the group queue and commit requests as soon as possible.
Revised Architecture
An RNLP Based Approach

How to bound the group queue? How to avoid interference from later requests?

1. Each server group has a **global group queue** of IPC contexts

2. Client **acquires** an IPC context.

3. Once acquired, the IPC context is enqueued in FIFO order in the group queue

4. Resource servers traverse the queue and commit requests **that cannot interfere with earlier ones.** Need to predict the future!
Server Tickets

Present and Future Resource Server Invocation Tracking

Server tickets are server-specific tokens stored in IPC contexts.

A ticket is needed to invoke a resource server.

The handled ticket is consumed when the server replies.

An IPC context with a ticket for S₁ in its ticket multiset has a request for S₁ or will have one in the future.
A request is committed only if there is no preceding IPC context in the group queue with overlapping ticket multiset.

Key property
Once committed, a C2S request is never delayed.
What budget is needed to satisfy a request?

\[ B = (2m + 1) \cdot L_g^{\text{max}} \]

Does **not** depend on the number of clients!

\[ L_g^{\text{max}} \]

Max required budget by a request in group \( g \)

\[ (m + 1) \cdot L_g^{\text{max}} + (m - 1) \cdot L_g^{\text{max}} + 1 \cdot L_g^{\text{max}} \]

\( L_g^{\text{max}} \) allows for **compositional budget provisioning** 😊
What budget is drained in practice?
Experimental Setup

Prototype implemented in LITMUS<sup>RT</sup>

4-core i5-4590 Intel evaluation target

**Evaluation Benchmark**

- Chain of $q$ servers
- On average 1ms of budget required per server

$L_{\text{server}}^{\text{max}} \approx 1\text{ms}$

$q \in [1,8]$

**Invocation patterns**

- **Sequential**
  - Adversaries invoke the first server

- **Random**
  - Servers randomly picked by adversaries

- **Distributed**
  - Servers evenly assigned to adversaries
Our Theoretical Bound is Verified in Practice

Bound is tight

In-group parallelism

pattern
- Sequential
- Random
- Distributed

Bound

9 \cdot L_{\text{max}}^{\text{link}}
Thank you for your attention!

**Contribution**

G(IP)$^2$C: The first synchronous IPC protocol with temporal isolation for S2S invocations.

**Key properties**

- Bounded Interference: $(2m + 1) \cdot L_g^{max}$
- Group-Independence Preserving
- Deadlock Free

**Scope**

Multiprocessor systems under partitioned reservation-based scheduling

**There is more in the paper!**

- Full proof
- Progress rules
- Abortion Rules

**Limitations**

- Requires careful resource partitioning
- Higher Runtime Overhead than simpler protocols

**Extensions**

- Support for background jobs
- Multi-occupancy reservations