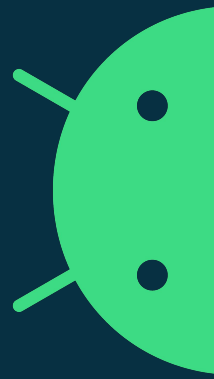
Rethinking Android's Priority Inheritance

A Binder Driver Problem

Carlos Llamas - cmllamas@google.com



android

What is Binder?

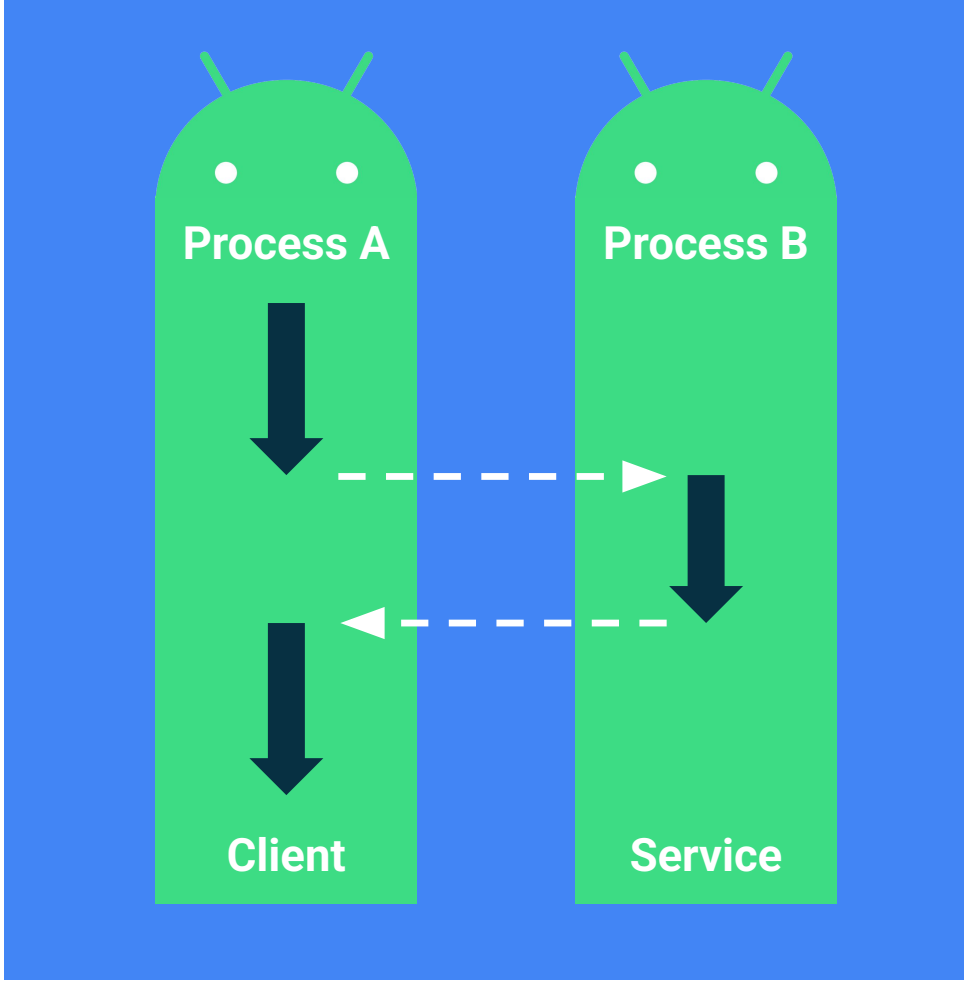
A Quick Primer

Yet another IPC/RPC

- Used **extensively** in Android.
- Lightweight and **transparent** design.

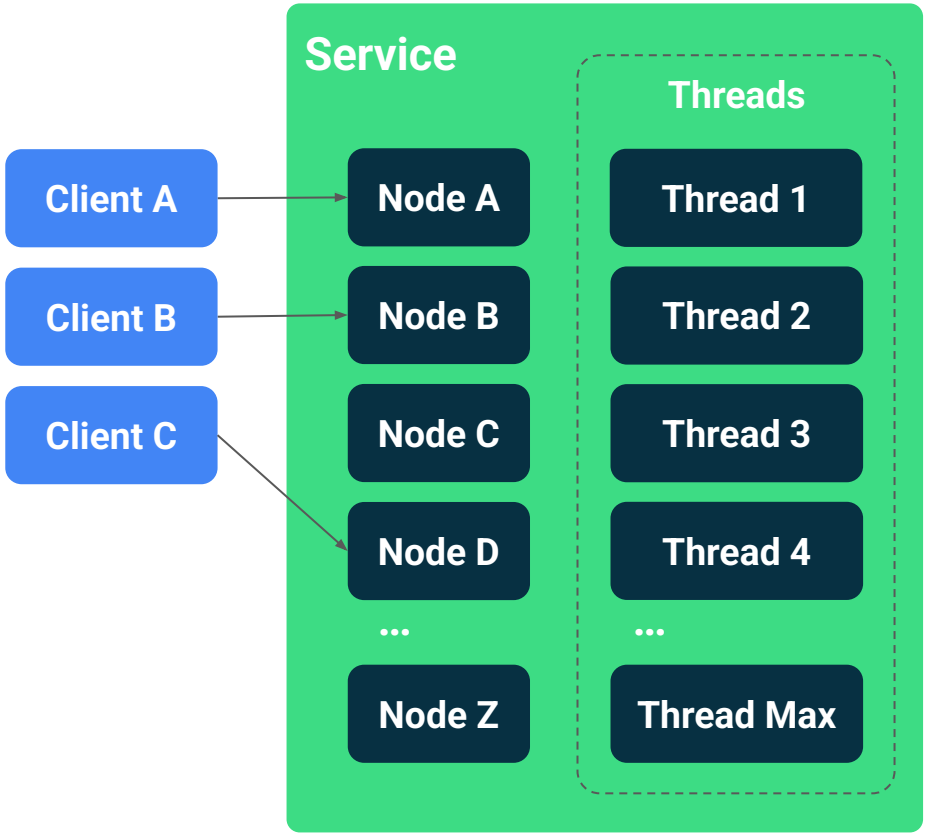
```
/* remote call */
res = proxy.foo(a, b);
```

- Services create a **nodes**.
- Each node implements an interface.
- Client obtains a **proxy** to the node.



Behind the Curtain

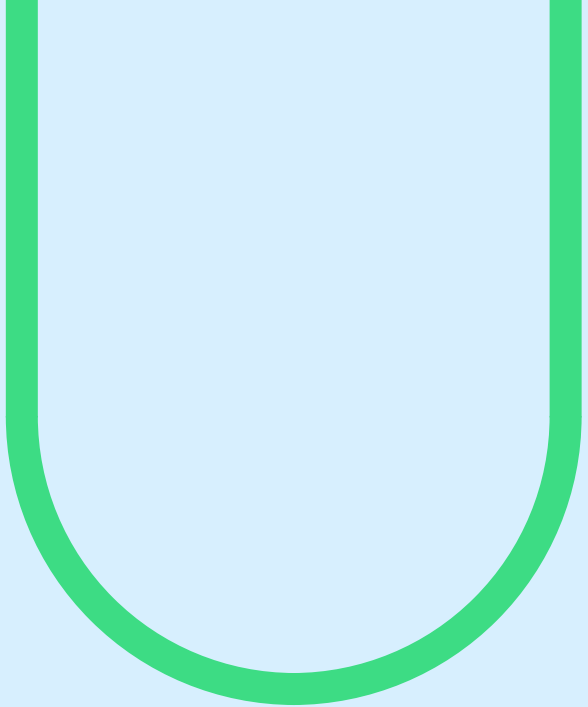
- Client invokes method via proxy.
- Driver copies data and **blocks** Client.
- Driver selects a Service thread.
- Service executes method.
- Driver returns result and **wakes** Client.

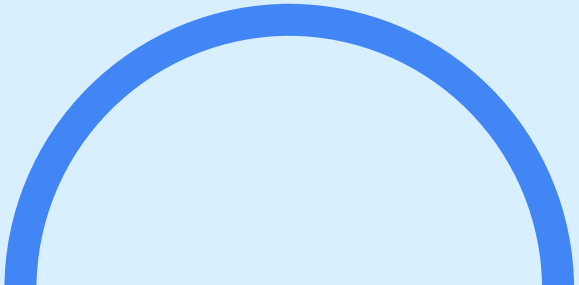


android

Priority Inheritance

The Custom Engine



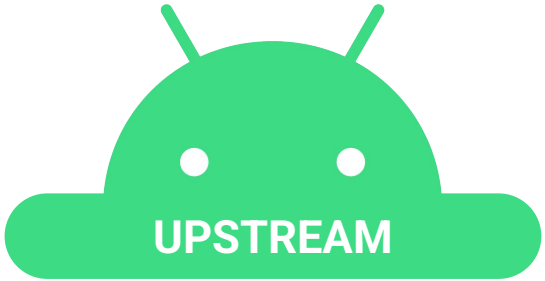


android

Why is Priority Inheritance needed?

- Synchronous calls **block** and lead to Priority Inversion.
- Higher transaction **latency** and lower **throughput.**
- Translates to **frustrated** users (e.g. display janks).
- **Priority Inheritance** is the in-kernel solution.
- The target **temporarily** inherits the priority of the caller.
- The original priority is restored after the call is serviced.

Current Status: The Two Binders



- CFS Inheritance

- CFS Inheritance
- **RT** Inheritance (**OOT**)

What is the problem?

The Good, the Bad and the Ugly

android

Problem 1: CFS Inheritance Hack

- Niceness inheritance:

  "They are inheriting nice values and that is plain wrong."

- **Dynamic** vs **Static** Priorities.
- Does it work in WFQ?
- Breaks the fairness of the CFS algorithm.
- Task-level boost is helpless with **cgroup quotas.**

Solution 1: CFS Inheritance

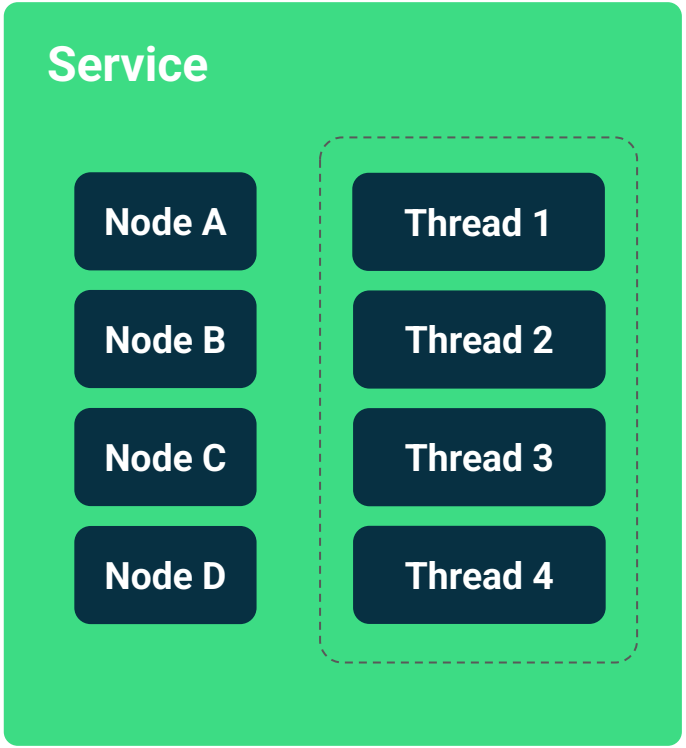
- Don't.

RT Priority Inheritance

- Makes more sense (static).
- Issues associated with an **OOT** patch.
- Custom **open-coded PI** implementation.
- Unexpected custom behavior:
  - Priority Demoted
  - Default Priority
  - **Minimum Node Priority** ← What is this?

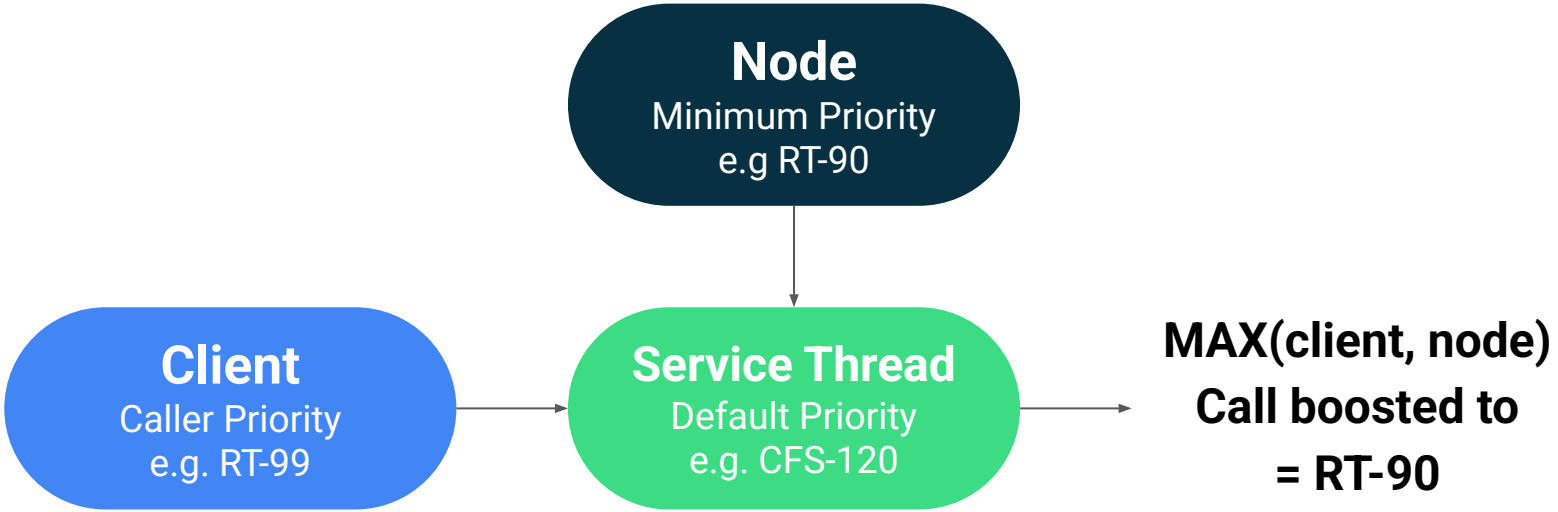
Problem 2: Minimum Node Priority

- A single service has **multiple nodes.**
- Nodes are serviced by the **same thread-pool.**
- Certain nodes are more **critical** than others.
- User assigns a **minimum priority** per node.
- Userspace policy implemented in the kernel.



Problem 2: Minimum Node Priority

Driver boosts the Service thread at the MAX between the Client and Node priorities.



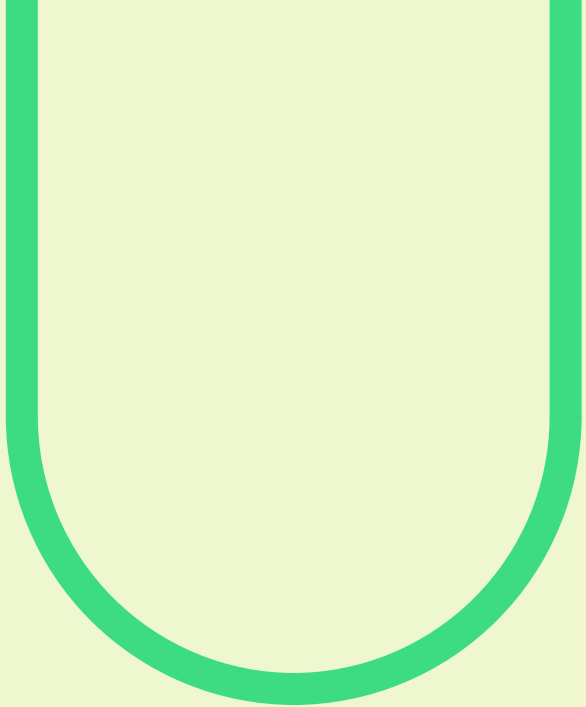
android

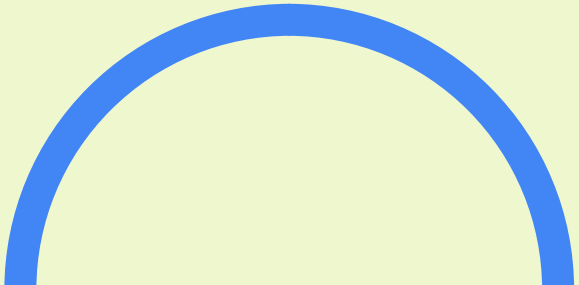
Solution 2: Use a Different Threading Model

- This should not be **implemented** in the kernel.
- Services need **dedicated** thread pools.
- **Nodes** should be allocated to the appropriate thread pool.
- How not to break userspace? I don't know.

RT Proposals

A Brighter Future





android

The rt_mutex() Approach

- Replace custom PI logic with rt_mutex()
- It needs to fit the binder wake-and-wait model.

```
thread = binder_select_avail_thread(server);
rt_mutex_init_proxy_locked(rtmutex, thread->task);
rt_mutex_start_proxy_locked(rtmutex, waiter, current);
wake_up_interruptible(&thread->wait);
[...]
```

The rt_mutex() Approach

- What about the no available threads scenario?

```
from = binder_get_transaction_from(t);
rt_mutex_init_proxy_locked(rtmutex, current);
rt_mutex_start_proxy_lock(rtmutex, waiter, from);
[...]
```

Long Term: Proxy Execution

- Allows a task to **donate** its **turn** when blocked.
- Preferred and theoretically the **ideal** solution.
- It doesn't quite **fit** the wake-and-wait model.
- How to express the dependency?

Let's Talk

Binder's PI is a 10+ year-old issue and we want to fix it.

Let's discuss how.

Carlos Llamas | cmllamas@google.com