

LPC 2026 - sched_ext MC

Making proxy-execution compatible with sched_ext



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Andrea Righi <arighi@nvidia.com>

John Stultz <jstultz@google.com>

What is Proxy Execution?

Classic Priority Inheritance:

- Lock owner set to blocking lock waiter's priority
- “Tricks” the scheduler to pick the lock owner to run.
- Works well for linear RT priorities, but not SCHED_DEADLINE or SCHED_FAIR, especially with complex cgroup configurations.

Proxy Execution:

- Instead of changing task priorities, leave blocked waiters on the runqueue.
- Treating the scheduler opaquely, let it pick “most important task” to run (of runnable and mutex blocked tasks).
- If selected task is blocked waiting on a mutex, run the mutex owner on behalf of the waiter (which effectively donates its cpu time and context to get the lock released).
- Mitigates priority inversion problems (we want this in sched_ext)

Proxy-Exec: Core implementation (overview)

Blocked waiter(W) --> [mutex M] <-- Runnable owner (0)

CPU0: W's original CPU

CPU1: 0's CPU

W remains on the runqueue
Scheduler selects W

0 is runnable

[W's class / priority / budget]
proxy migration

-----> rq->donor = W
rq->curr = 0
[0 executes on W's behalf]
[0 releases M]

W wakes and returns to CPU0 <----- scheduling context returned

NOTE: W is blocked & does not run on CPU1, only its scheduling context is migrated

Problem

- CONFIG_SCHED_PROXY_EXEC depends on !CONFIG_SCHED_CLASS_EXT
 - Distributions cannot ship both features in one kernel
 - sched_ext callbacks, kfuncs, slices and accounting traditionally followed rq->curr; proxy execution needs them to follow rq->donor instead
 - Proxy execution migrations can race with BPF remote-DSQ transfers and placement decisions

Where we left off at LPC 2025

- "Can ProxyExec and sched_ext get along?" - John Stultz
<https://lpc.events/event/19/contributions/2032/>
- An initial prototype showed that making the two features coexist was feasible
- The open question was who should control proxy execution: the scheduler core or the BPF scheduler?
- How to handle the N:M mapping between BPF DSQs and physical runqueues with proxy execution?

Solution

- Make proxy execution an explicit opt-in per-BPF scheduler
 - Set SCX_OPS_ENQ_BLOCKED flag to start receiving blocked donors in ops.enqueue()
 - If not set, scheduler core will not keep mutex blocked scx tasks on rq
 - Blocked donors have the SCX_ENQ_BLOCKED flag set
 - The BPF scheduler decides how to handle donors (DSQ, CPU, ordering, slice and preemption policy)
 - When the donor is selected for execution, proxy-exec core follows the owner chain transparently
 - BPF continues to observe and charge the donor (owner is hidden from its scheduling policy)

sched_ext proxy-exec workflow

W blocks on a mutex owned by 0

-> W is still current, core switches to idle first:

`ops.stopping(W) -> ops.enqueue(W, SCX_ENQ_BLOCKED) [W's CPU]`

-> BPF dispatches W; `ops.dequeue(W)`

-> Core selects W, proxy exec resolves 0, migrates W

`ops.enqueue(W, SCX_ENQ_BLOCKED) [W's CPU]`

`ops.dequeue(W);` W moved to 0's CPU (p->wake_cpu = W's CPU)

`ops.enqueue(W, SCX_ENQ_BLOCKED) [0's CPU]`

-> BPF dispatches W; `ops.dequeue(W)`

-> `ops.running(W)`

-> 0 executes using W as scheduling context

`rq->donor = W | rq->curr = 0`

Core
SCX/BPF

Still blocked when descheduled:

-> `ops.stopping(W) -> ops.enqueue(W, SCX_ENQ_BLOCKED) -> repeat`

0 unlocks and wakes W:

-> `ops.stopping(W); ops.quiescent(W, SCX_DEQ_SLEEP)`

-> `ops.select_cpu(W, prev_cpu = p->wake_cpu) -> ops.runnable(W)`

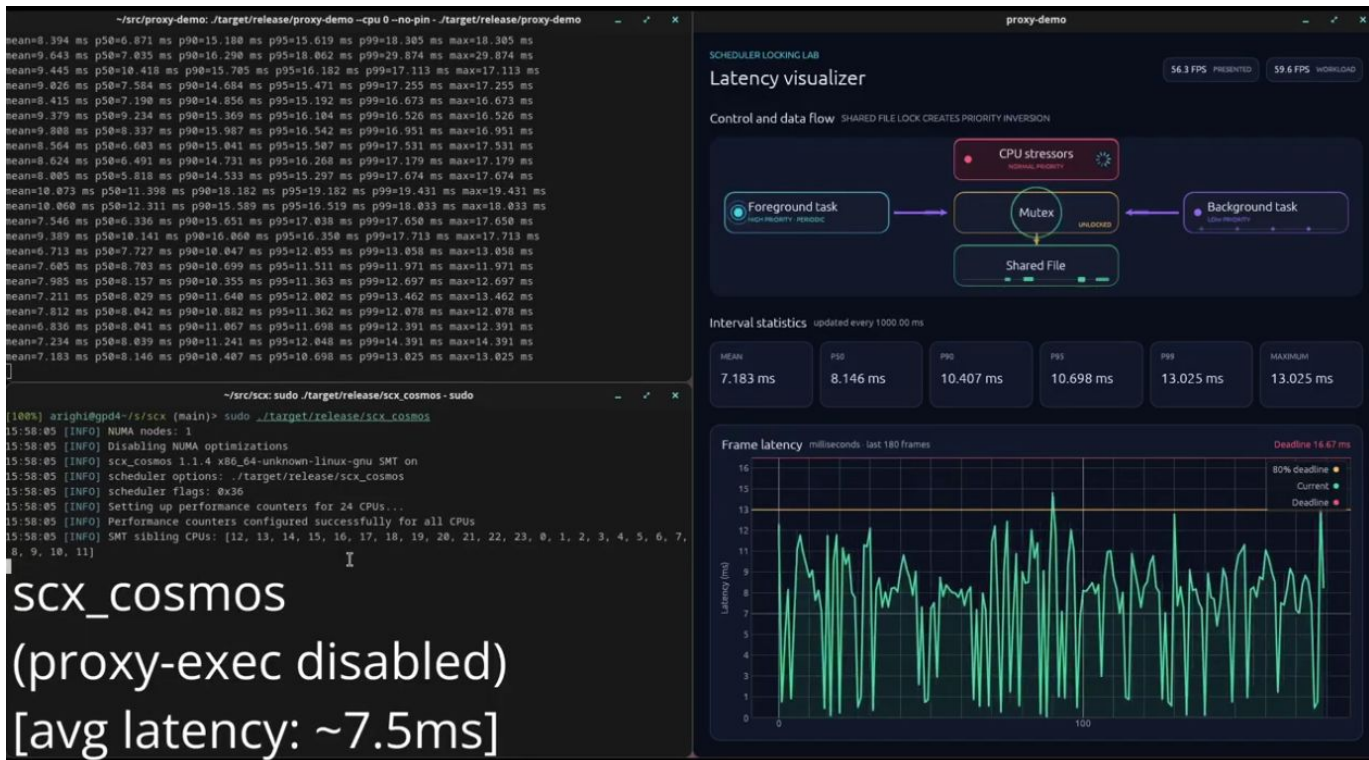
-> `ops.enqueue(W, SCX_ENQ_WAKEUP)`

Example API Usage

```
void BPF_STRUCT_OPS(enqueue, struct task_struct *p, u64 flags)
{
    /* Is @p a blocked donor? */
    if (flags & SCX_ENQ_BLOCKED)
        scx_bpf_dsq_insert(p, blocked_dsq, slice, SCX_ENQ_PREEMPT);
    else
        enqueue_normal(p, flags);
}

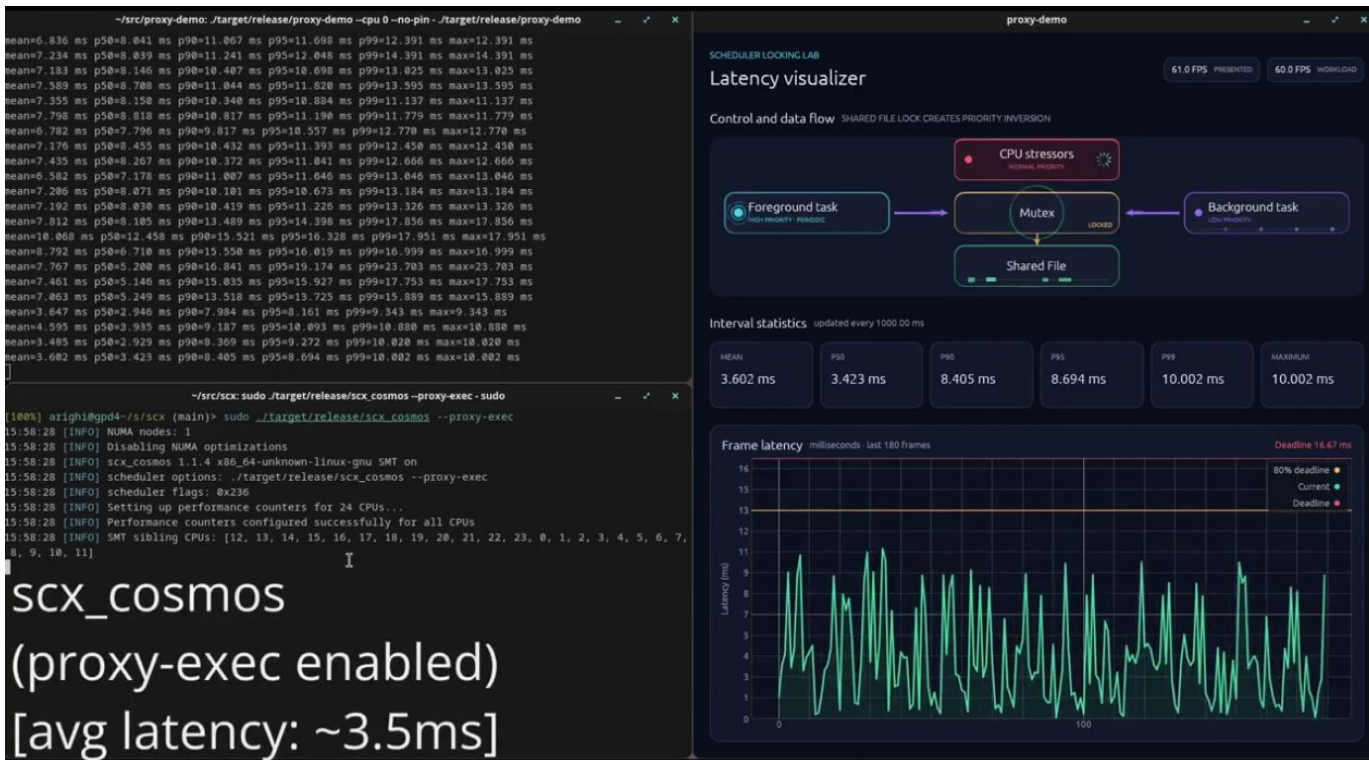
SCX_OPS_DEFINE(scheduler_ops,
    .enqueue = (void *)enqueue,
    ...
    .flags    = SCX_OPS_ENQ_BLOCKED);
```

Result: scx_cosmos without proxy-exec



<https://www.youtube.com/watch?v=gkori8b1A0k>

Result: scx_cosmos with proxy-exec



<https://www.youtube.com/watch?v=gkori8b1A0k>

Questions?

Questions?

- Is this API sufficient/complete from a BPF scheduler perspective to properly support proxy exec?
- How to handle the sub-schedulers scenario (donor managed by BPF scheduler A / cgroup A and owner managed by BPF scheduler B / cgroup B)?
 - Current contract is: the donor scheduler controls the scheduling context, while the owner may execute regardless of the owner scheduler's policy or consent.
- Open issues
 - On NOHZ_FULL CPUs, proxy execution conservatively re-enables the periodic scheduler tick while `rq->curr != rq->donor`.
 - The remote NOHZ tick currently assumes that the execution and scheduling contexts are the same, so allowing the tick to remain stopped could update or charge the wrong task. Making the remote tick donor-aware is left as a future improvement.