

Proxy Enabled Futexes

Suleiman Souhlal <suleiman@google.com>

John Stultz <jstultz@google.com>

LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



Previously...

LPC2025: [Video](#)



OSPM25: [Video](#)



LPC2024: [Video](#)



OSPM24: [Video](#)



LPC2023: [Video](#)



OSPM23: [Video](#)



OSPM19: [Video \(Juri Lelli's talk\)](#)



Why ProxyExec?

Currently

Android Suffers from Priority Inversion

Android conceptually has **FOREGROUND** and **BACKGROUND** tasks.

Want to avoid **BACKGROUND** tasks impacting **FOREGROUND** task performance.

If we use any type of restriction or throttling of **BACKGROUND** tasks, we open ourselves up to priority inversion, where **BACKGROUND** tasks take kernel locks, and then are preempted. Important **FOREGROUND** tasks then have to wait until **BACKGROUND** task can run again to release the lock. This is a common cause of UI jank.

Goal

ProxyExec solves lock based priority inversion

Generalized form of priority inheritance.

FOREGROUND tasks, when waiting on a lock held by BACKGROUND tasks will “boost” the BACKGROUND tasks, allowing them to run until they release the needed lock.

Allows utilization of various ways of restricting BACKGROUND tasks (cpu & cpuset cgroups, SCHED_IDLE, etc) so that that they don't impact FOREGROUND tasks.

Benefits

Avoids PI Caused Latency

Example case of **foreground** filesystem operation latencies, with background tasks doing the same workload.

Chart shows histogram of measured latencies.

On the left, background tasks are running unconstrained, and on the right, we are using `cpu_share` cgroups to constrain background tasks.

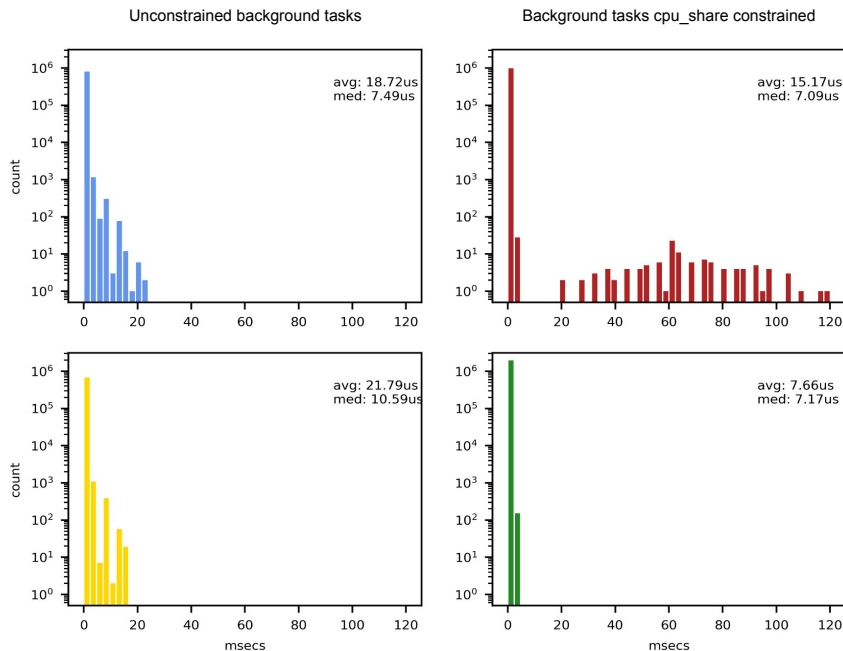
Y-axis is log-scale.

X-axis is in ***milliseconds***

Android

Histogram of time to move a file between directories with background tasks doing the same

Vanilla:



ProxyExec:

<https://github.com/johnstultz-work/priority-inversion-demo>



ProxyExec in Android



Android16-6.12

Full **base series** for **mutex** support included, but **disabled by default** (enableable via boot argument), so vendors could update Vendor Hooks that wouldn't expect the split contexts in ProxyExec



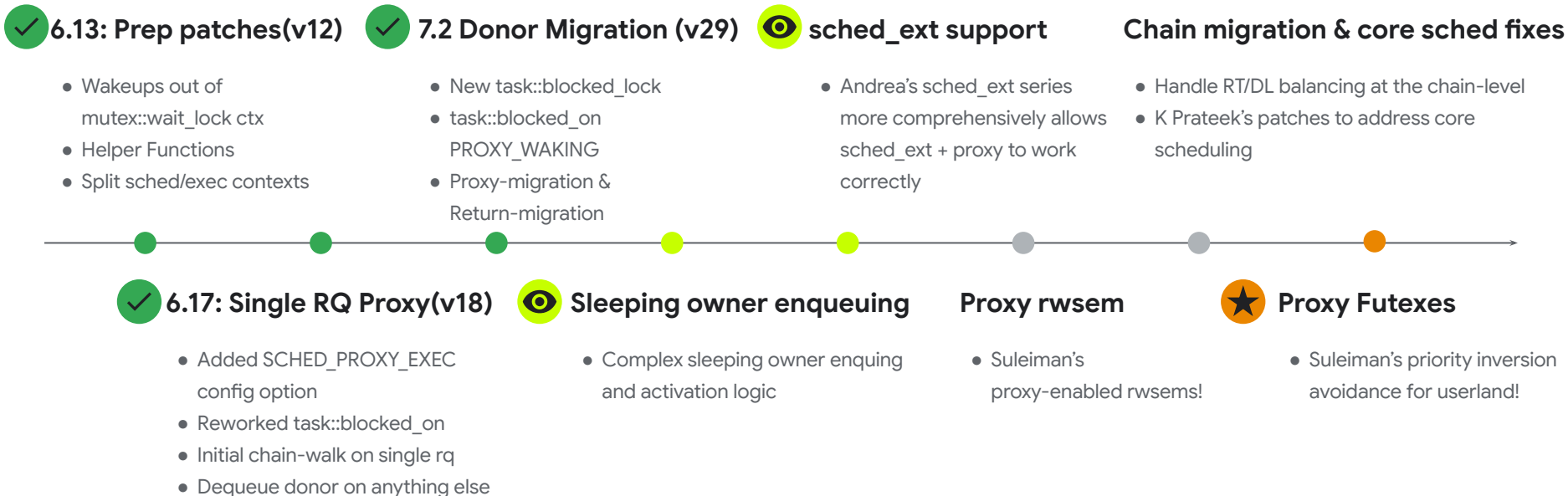
Android17-6.18

Full base series + **rwsem support** included! **Enabled by default**

Have to disable it if using sched_ext

Vendors experimenting with it, and some seeing nice results

Upstream Status & What's New



Proxy Execution is Great,

But only works for **kernel mutexes** and **rwsems**!

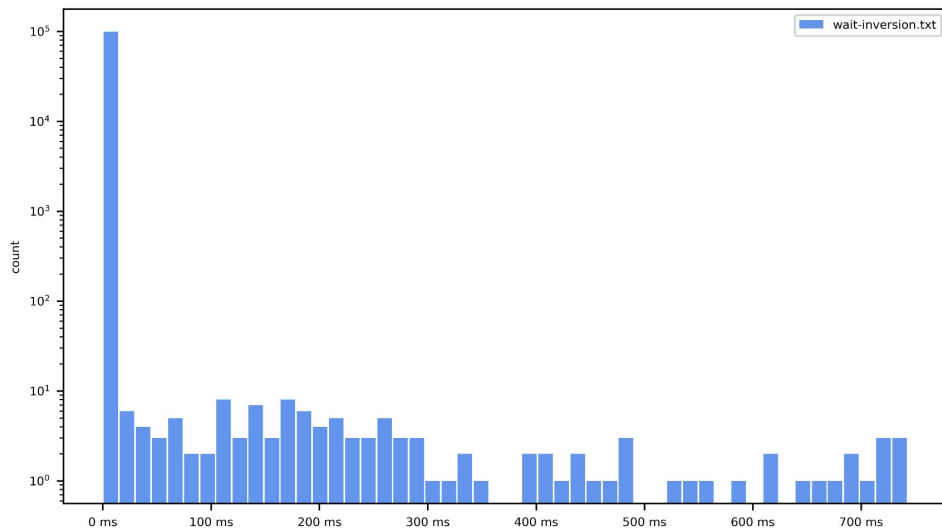
This doesn't help my application!

Futexes: (FUTEX_WAIT)

Really fast!
But inherently unfair,
and can cause
starvation even without
priority inversion

Priority inversions are also common on userland locks!

Important tasks blocked on normal FUTEX_WAIT futexes can easily see priority inversion.



Didn't we already fix this?

PI Futexes: (FUTEX_LOCK_PI)

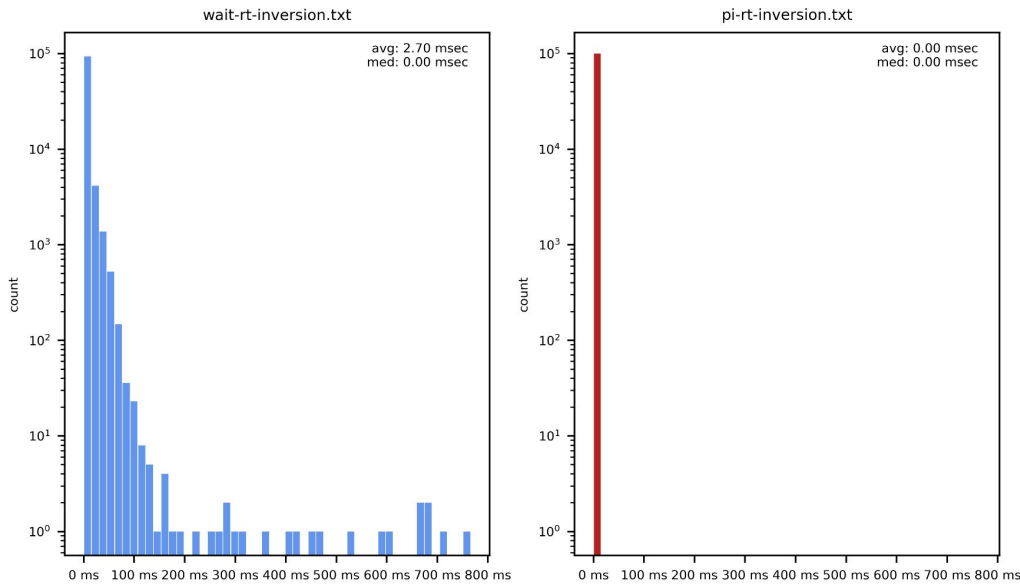
Already solved! (... for RT tasks)

Few problems:

- Only get boosting for RT (SCHED_FIFO/RR) tasks!
- Strict RT handoff semantics & no lock stealing causes performance issues

On the positive side:

- In common cases (no biasing), strict FIFO handoff order does limit worse case latencies for SCHED_NORMAL tasks as well
- Does also provide in-kernel optimistic spinning

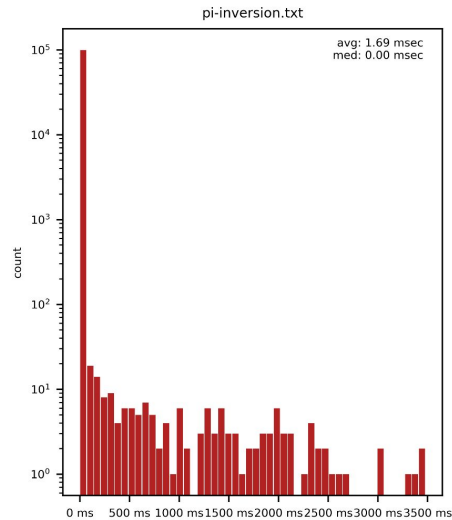
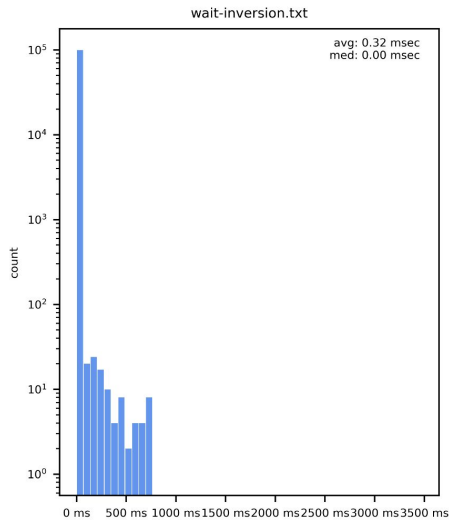


PI Futexes: (FUTEX_LOCK_PI)

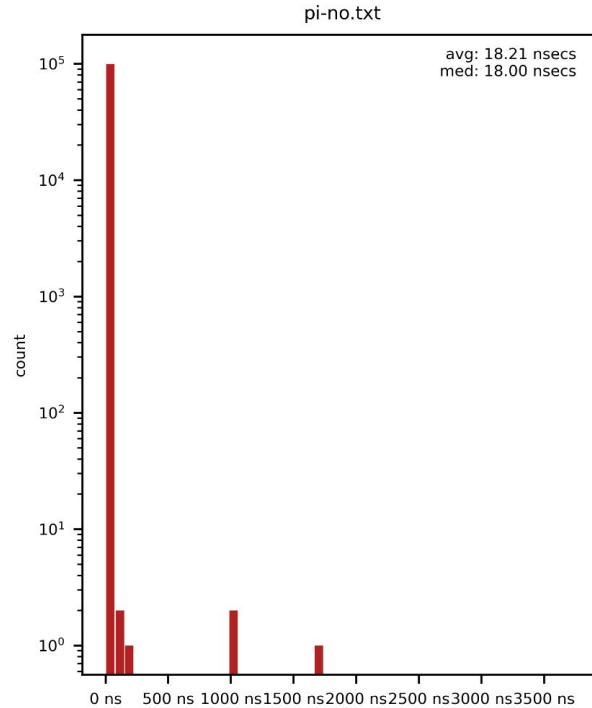
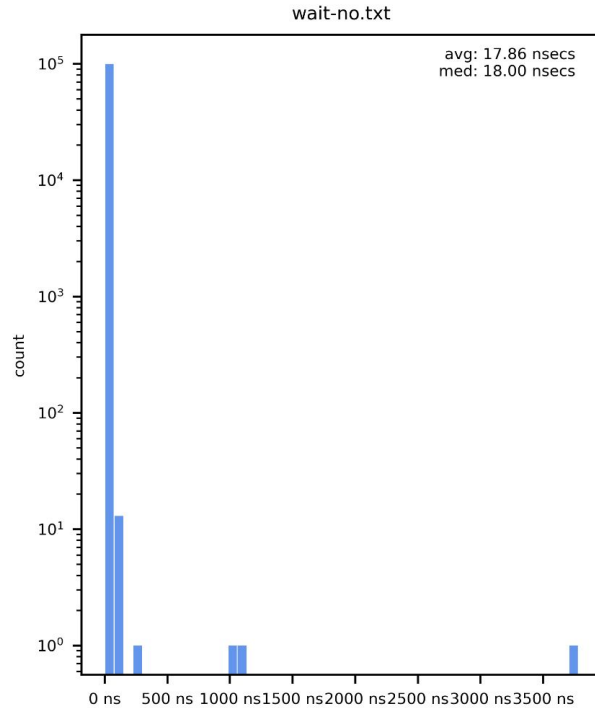
Also prevents
performance benefits
from lock stealing!

PI Futexes can be worse for SCHED_NORMAL!

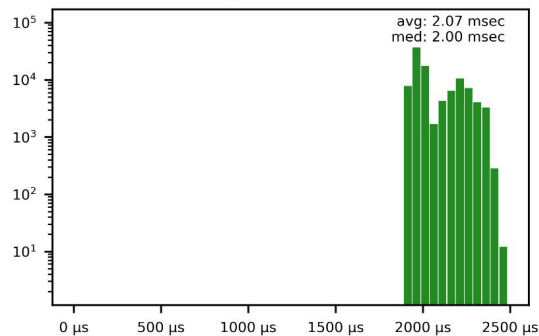
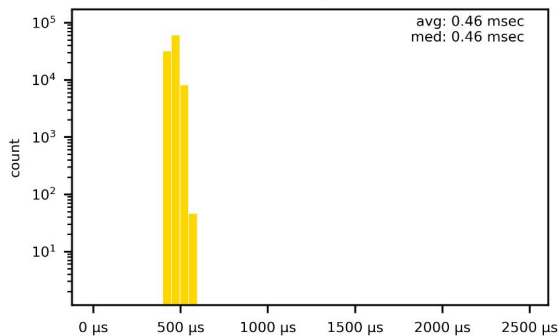
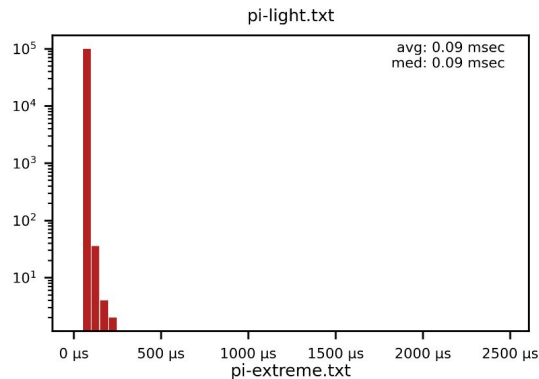
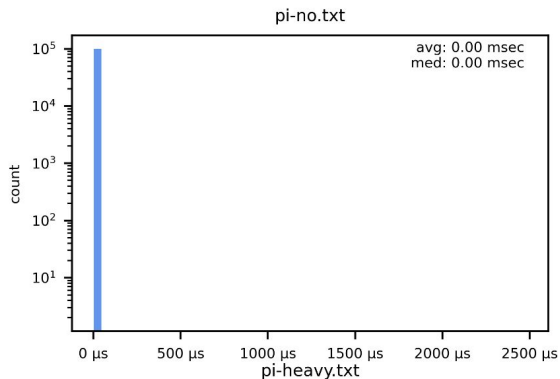
With non-RT biasing (nice, cgroup, etc), the strict FIFO order handoff means we will hand it of to a less important SCHED_NORMAL task, causing worse latencies!



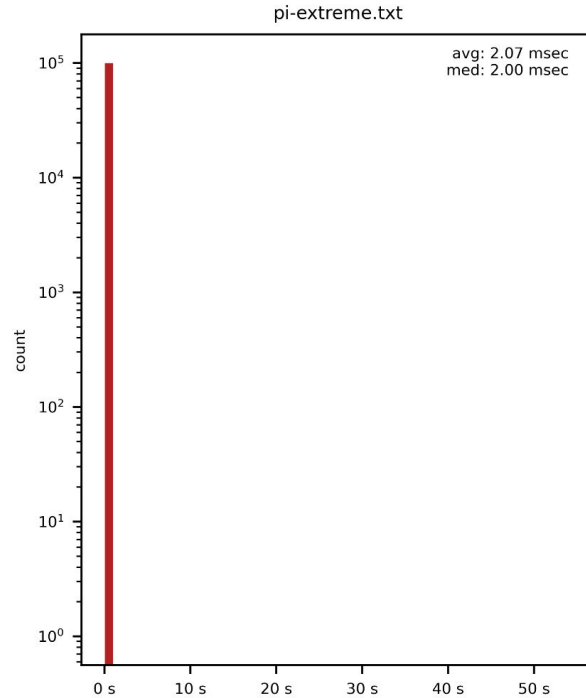
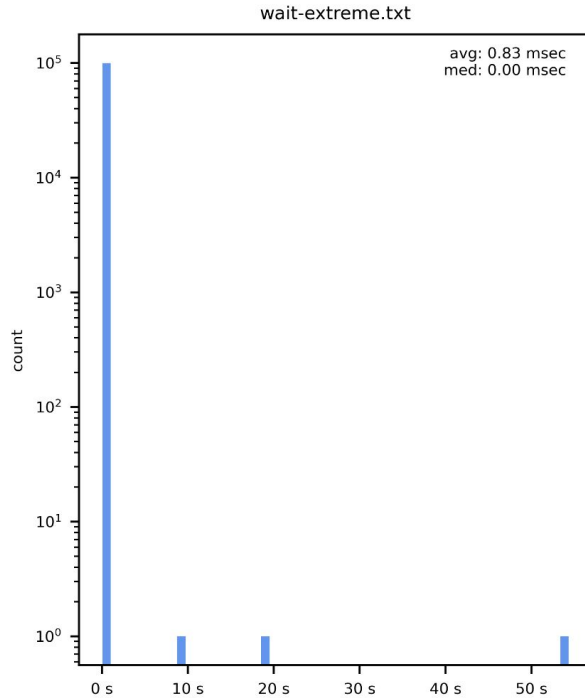
LOCK_PI: No contention



LOCK_PI: As contention grows



LOCK_PI - Extreme contention



Want LOCK_PI inheritance

With FUTEX_WAIT performance

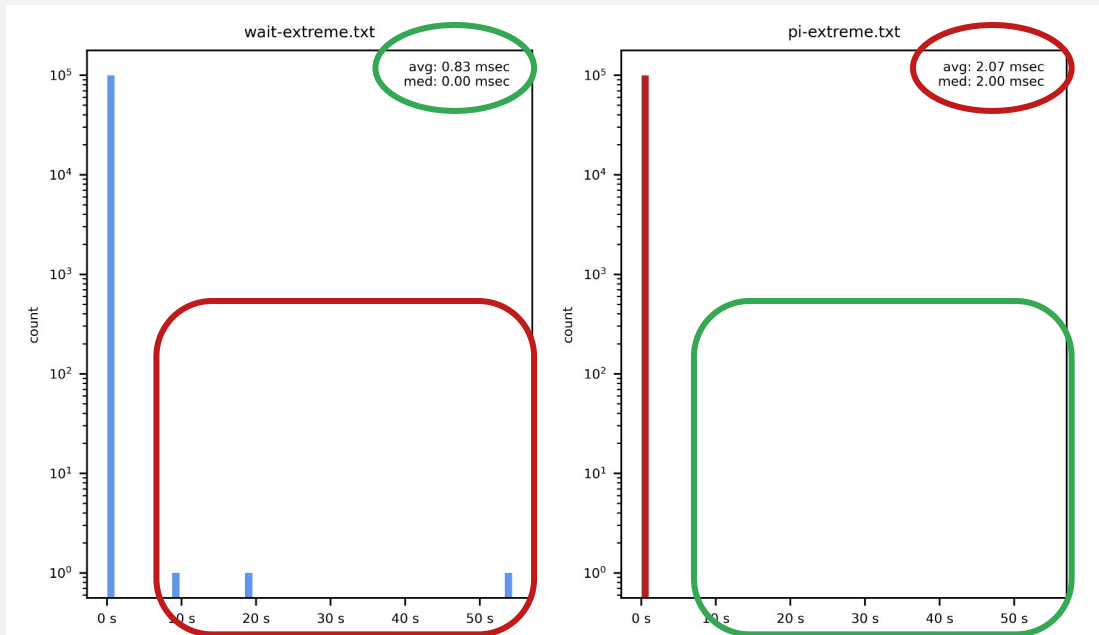
Ideally we would like to have priority inheritance while preserving the FUTEX_WAIT performance characteristics

- Also want **inheritance with SCHED_NORMAL** tasks
- Looser semantics then strict RT behavior is ok

Need to communicate the owner to the kernel via writing owner TID to lock value, similar to LOCK_PI.

So we need a new FUTEX op...

Android



Introducing:

Futex PI-NG: The Next Generation

Introducing:

Futex PI-NG: The Next Generation

(Blame Rostedt for this name!)

Ugh, that name!

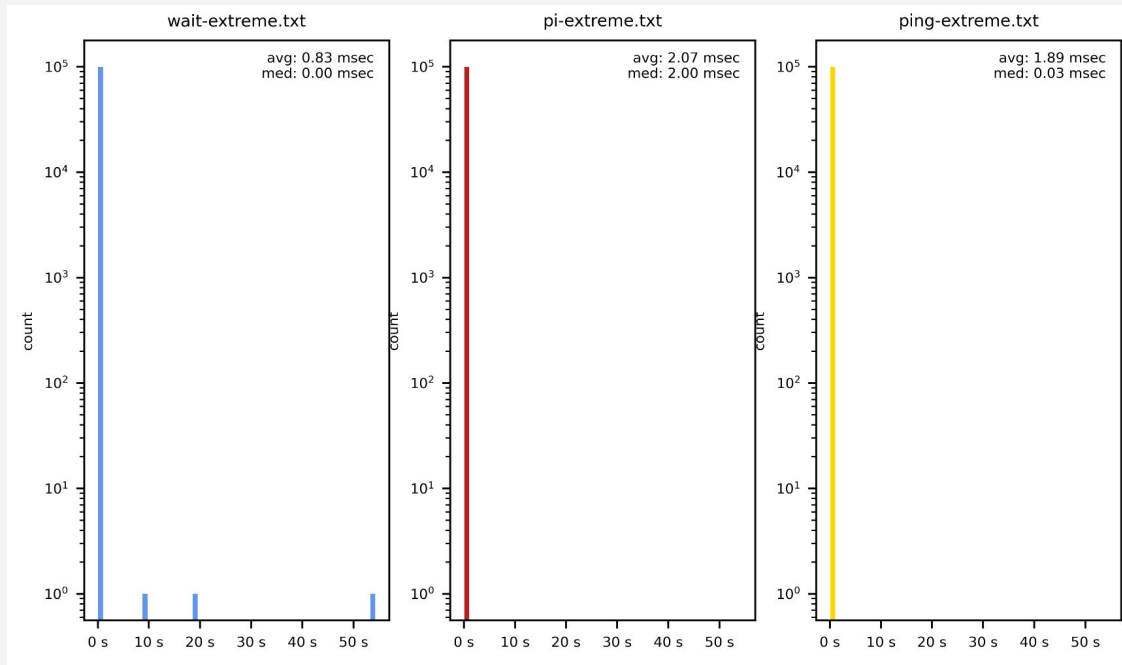
FUTEX_LOCK_PING

Similar to FUTEX_LOCK_PI

- Write TID to lock
- Has waiterbit and owner dead bits
- Supports kernel side optimistic spinning

But different as well:

- Allows for lock stealing from both kernel and userland
- Allows for waiter, with empty owner
- Uses Proxy Execution for priority boosting (instead of `rt_mutexes`), enabling priority inheritance with `SCHED_NORMAL` tasks
- Dynamic in its handoff rules to prevent starvation



User API

Same as PI futexes, but optionally allows for stealing in userspace when locking:

```
void futex_lock(uint32_t *mtx) {
    static __thread pid_t tid = getpid();
    uint32_t oldval = 0; // Unlocked with no waiters. Try to take it.

    if (atomic_compare_exchange_strong(mtx, &oldval, tid))
        return;

    oldval = FUTEX_WAITERS; // Waiters but no owner. Try to steal it.
    if (atomic_compare_exchange_strong(mtx, &oldval,
                                      tid | FUTEX_WAITERS))
        return;

    if (futex(mtx, FUTEX_LOCK_PING, 0, NULL)) != 0)
        err(1, "FUTEX_LOCK_PING");
}
```

Layout: 32 bit word

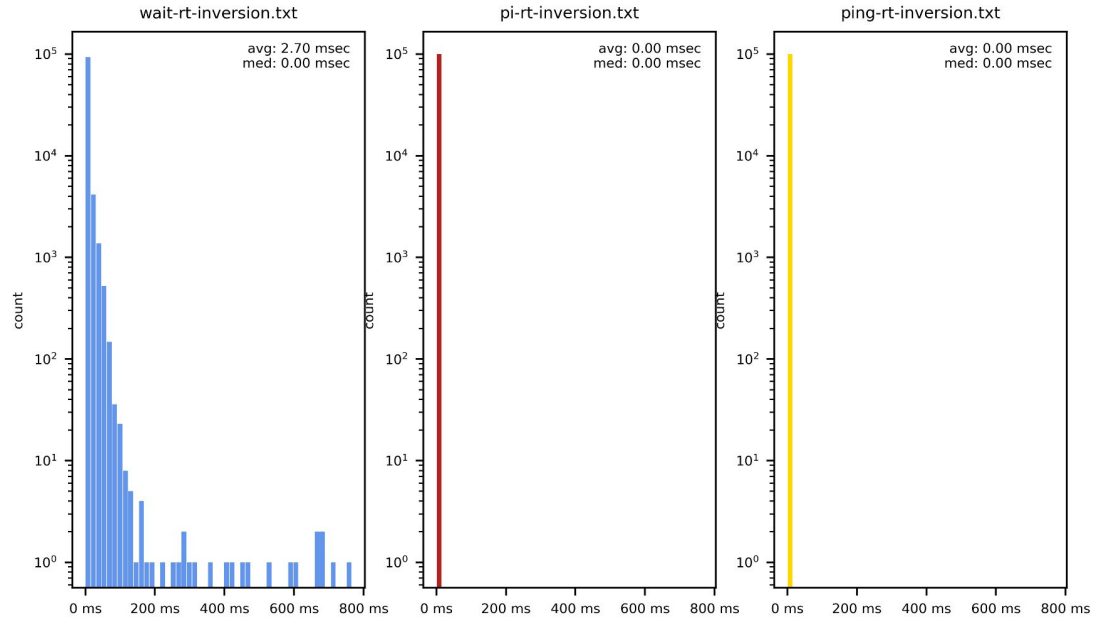
#define FUTEX_WAITERS	0x80000000
#define FUTEX_OWNER_DIED	0x40000000
#define FUTEX_TID_MASK	0x3fffffff

```
void futex_unlock(uint32_t *mtx) {
    static __thread pid_t tid = getpid();
    uint32_t oldval = tid; // No waiters. Try to unlock it.

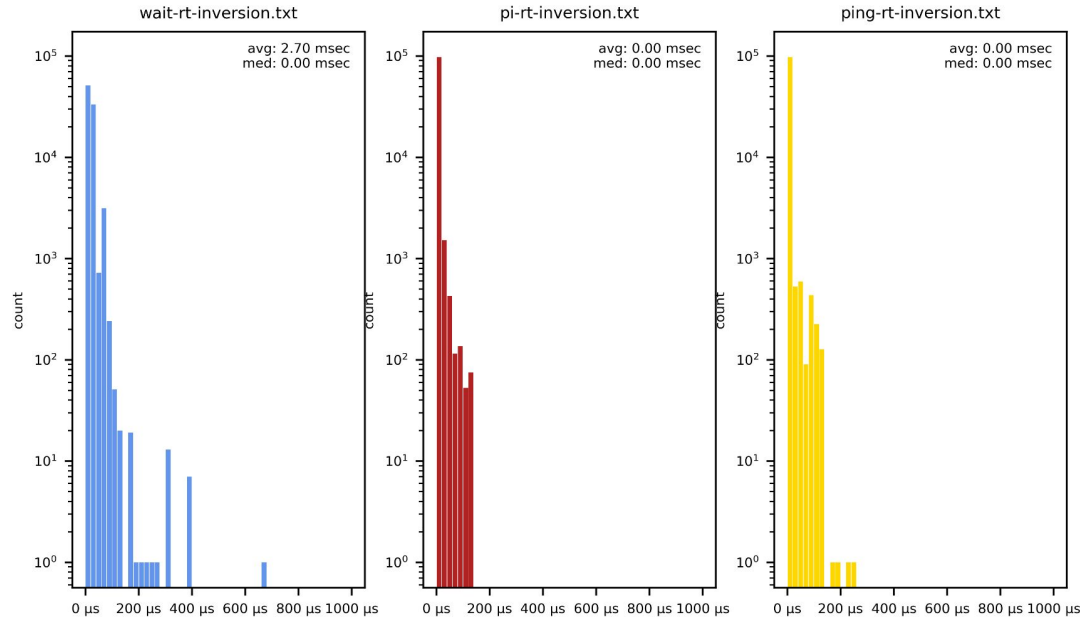
    if (atomic_compare_exchange_strong(mtx, &oldval, 0))
        return;

    if (futex(mtx, FUTEX_UNLOCK_PING, 0, NULL) != 0)
        err(1, "FUTEX_UNLOCK_PING");
}
```

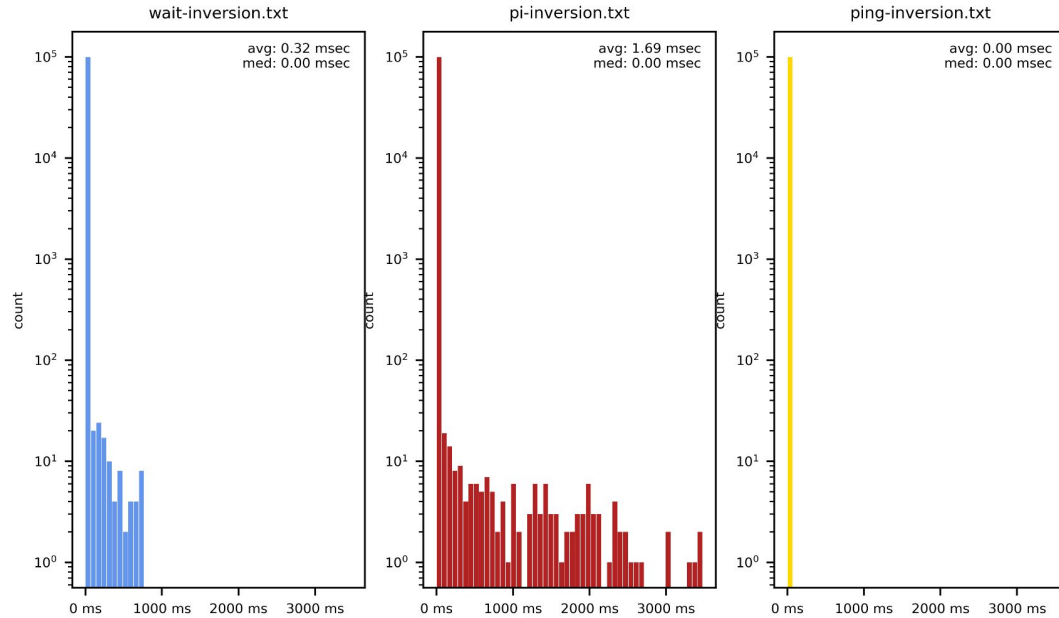
LOCK_PING: PI for RT tasks



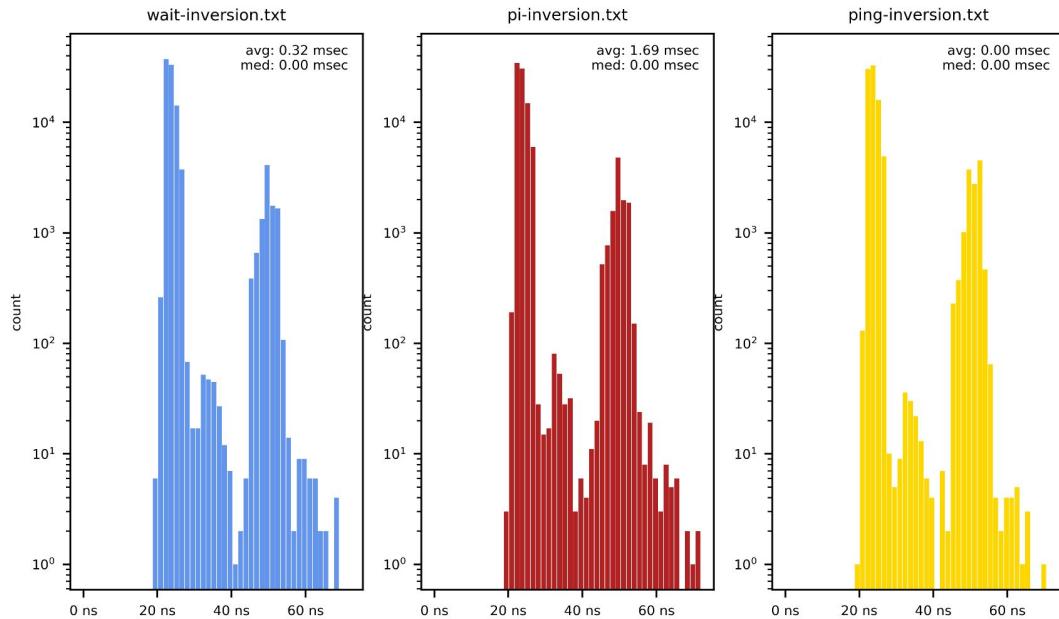
LOCK_PING: PI for RT tasks



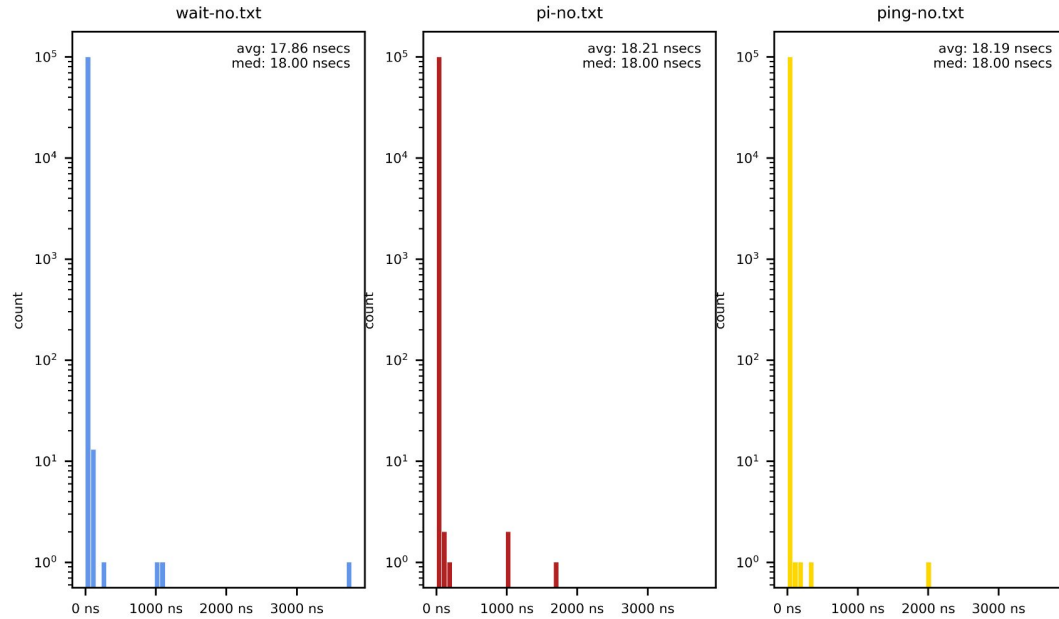
LOCK_PING: PI for NORMAL tasks



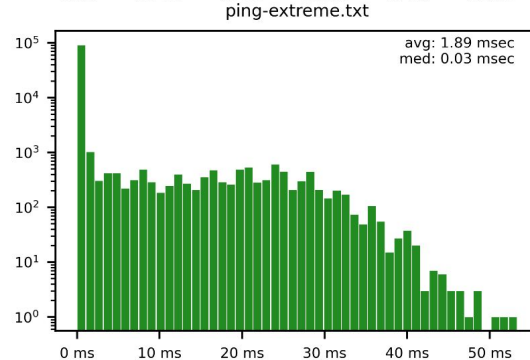
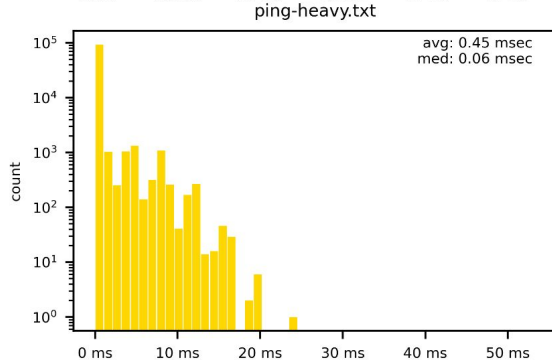
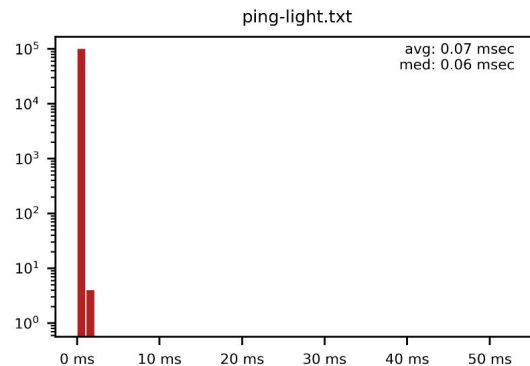
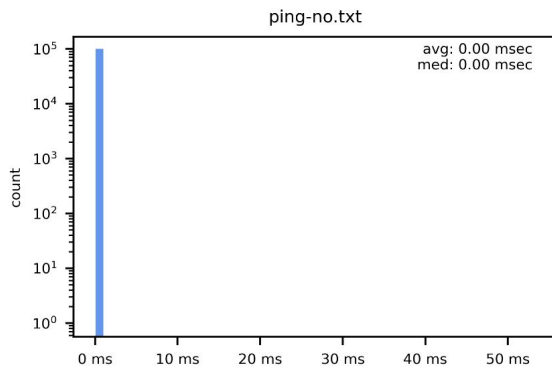
LOCK_PING: PI for NORMAL tasks



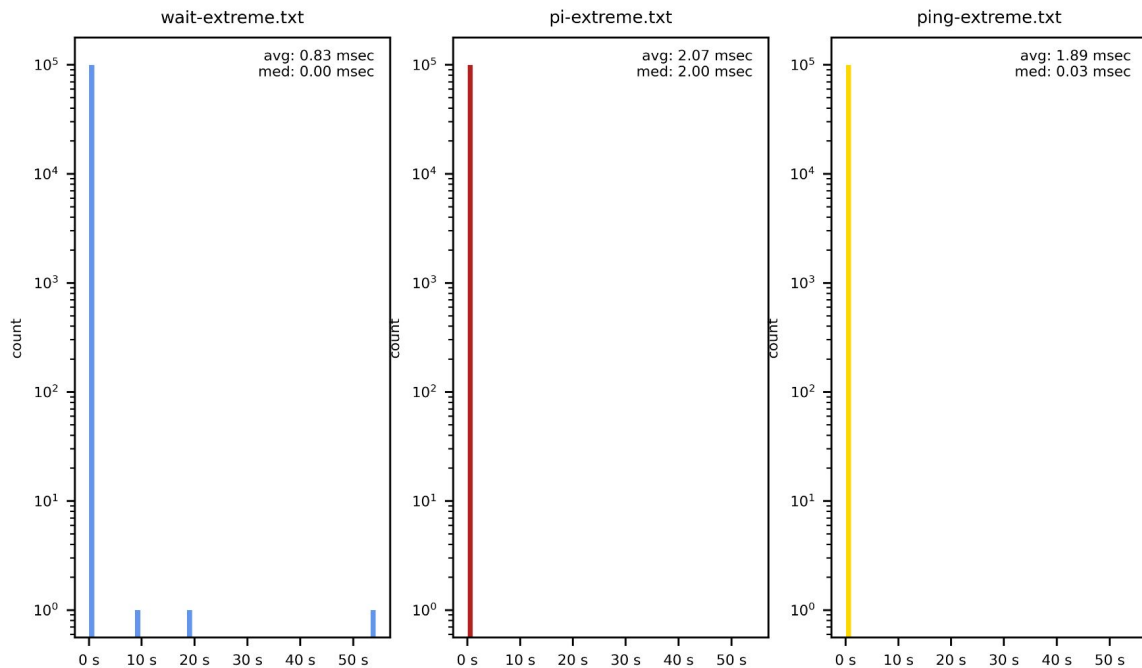
LOCK_PING: No contention



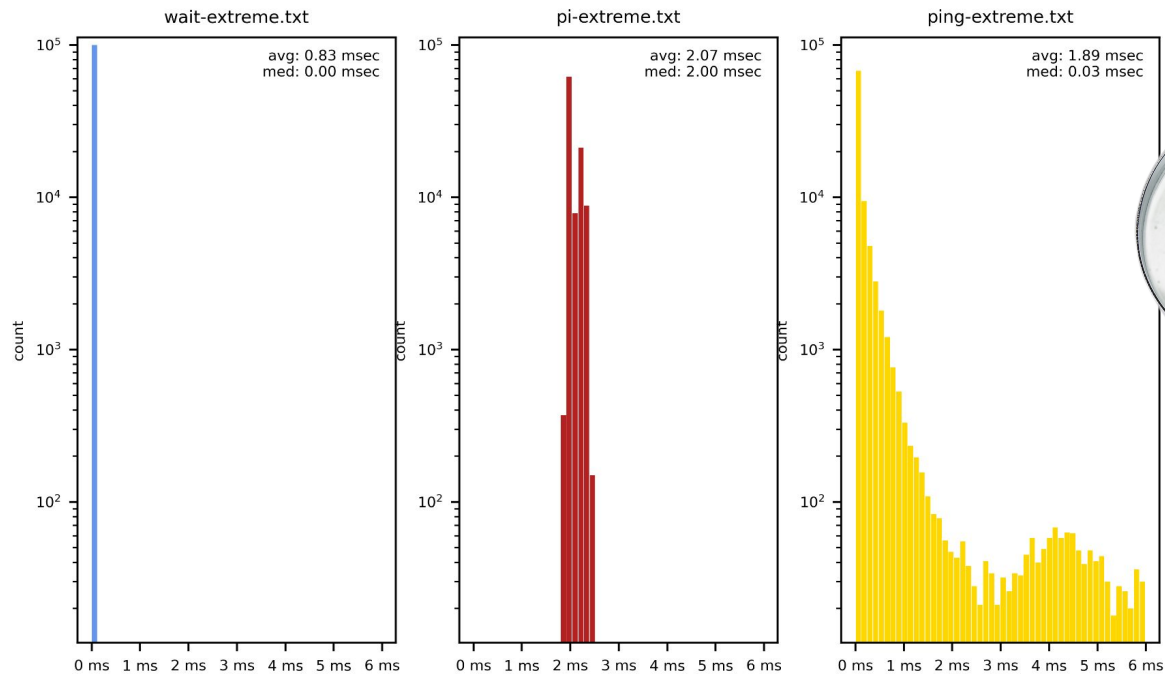
LOCK_PING: As contention grows



LOCK_PING: Extreme contention



LOCK_PING: Extreme contention



ping_bench

Our tool to
benchmark futexes

Created a tool to experiment and measure futex lock design

<https://lore.kernel.org/lkml/20260917043339.2093426-13-suleiman@google.com/>

Can generate metrics across a number of configurations

- f**: Type of futex (FUTEX_WAIT, FUTEX_LOCK_PI, FUTEX_LOCK_PING)
- t**: Number of threads acquiring/releasing the lock
- n**: Number of iterations the threads will take the lock
- s**: How frequently a thread tries to take the lock
- w**: Lock hold time
- b**: Number of “busy” cpu spinner threads
- r**: Number of threads using RT prio
- p**: Use nice () biasing (foreground thread gets more cpu time, background threads get less - allows for priority inversions)

TODOs

Optimize!

Tune handoff and opt spinning

Forced handoff, donor handoff and optimistic spinning rules need to be experimented with and tuned to better balance avg vs worst case latencies

Upstream!

Iterate Submitting Upstream

Push both the new FUTEX_LOCK_PING op and Proxy enablement along with the Proxy-rwsem support (both use the same proxy-lock abstraction)

Also [proxy-chain cycle avoidance](#) will be necessary as we can't enforce no-lock-cycles rule for userland

Actual Users!

Integrate into Android

Need to get some form of support into LTS-next based Android kernel

Likely some backporting as well, since partners will not be on LTS-next kernel very soon

As some may still disable ProxyExec, we will have to have some dynamic infrastructure to enable its use where possible

Discussion: (1/2)

- Concerns around FUTEX_LOCK/UNLOCK_PING op?
 - Better name? Please!
 - Any other add-on features needed?
- Waiman Long's [TP Futex](#) work
 - Very similar goals
 - Like the FUTEX_LOCK/FUTEX_UNLOCK name!
 - Peter wanted mutex backing similar to LOCK_PI's rt_mutex backing.

Discussion: (2/2)

- Performance impact of various approaches (donor handoff, forced handoff, osq spinning, etc)
- Reworking PI Futex code vs new PING implementation?
 - Not using `rt_mutexes` internally
 - Separate `ping_state` in hash bucket
 - Issues: `rt_mutex` needs major rework for proxy

Thank you

Particularly to folks who worked on the Proxy Exec patch series before me:

Watkins, Straub, Niehaus, Peter Zijlstra, Juri Lelli, Valentin Schneider, Connor O'Brien

Also for their review and feedback & contributions:

Andrea Righi, Joel Fernandes, Qais Yousef, Metin Kaya, K Prateek Nayak, Steven Rostedt, Suleiman Souhlal and many others!



Get in touch

suleiman@google.com

jstultz@google.com

