Userspace Assisted Scheduling via Sched QoS

Qais Yousef <qyousef@layalina.io>
Vincent Guittot <vincent.guittot@linaro.org>
John Stultz <jstultz@google.com>
Steven Rostedt <rostedt@goodmis.org>

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Linux supports diverse number of hardware and workloads

- Historically server users dominated Linux space
  - Typical userspace was throughput oriented
  - Many of historical biases are visible in scheduler default values
- Interactive systems are on the rise
  - Android, Desktop, Steam Deck, etc
  - Server workloads are becoming more diverse with requirements for higher interactivity similar to desktop environments
- Applications are still written with stale assumptions
  - Spawn N threads to mach N CPUs, not taking into account they share the system with other apps (and users)
  - POSIX haven't evolved to introduce mechanism to address power management, SMP, HMP additions

One scheduler to rule them all

- Managing this diversity with a single scheduler is possible
- If userspace cooperates by providing additional info/hints existing scheduler should be able to address the diverse workloads
    - It already has to manage the diverse hardware
- Why guess when we can be informed?
- A new SchedQoS library/middleware can help us abstract that
    - For the remainder of the talk, assume SchedQoS library/middleware is responsible for most of the implementation unless otherwise specified

Previous Talks

- LPC22:
  - Len Brown: Linux needs a Scheduler QOS API -- and it isn't nice(2)
  - Vincent Guittot: Latency hints for CFS task
- OSPM24:
  - Lukasz Luba & Rafael J. Wysocki: Performance QoS For Tasks
- LPC24:
  - John Stultz: QoS Hinting APIs: If we had them, what would they actually do?!

The Roadmap

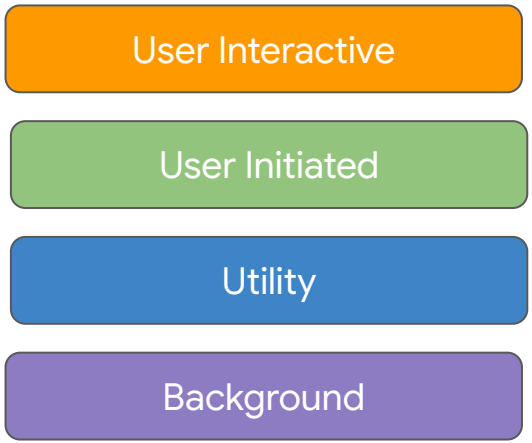
- Reaching to a fully usable user space assisted scheduling requires introducing/improving a number of areas in tandem
  a. QoS plumbing framework
  b. Multimodal wakeup and load balancer
  c. Easy adoption/deployment
  d. Scheduler default values
  e. Generalized Performance Inheritance support
  f. Access control

QoS plumbing framework

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

QoS Plumbing

- It's not about the kernel interface!
  a. It matters, but we almost got everything we need
- Applications need a high level interface to describe their workload generally
- Industry already offers one that has a proven track record
  a. Apple support for QoS has been available for years
  b. It provides 4 levels of QoS classes
  c. Existing sched attributes are mostly sufficient to describe them



The QoS mapping

- **runtime**: manages slice request size/deadline via EEVDF
  a. We should block usage of **nice** value as it will destroy determinism and generally are misused and not trustworthy
- **policy**: *NORMAL* vs *BATCH* to control wake up preemption
- **uclamp_max**: Restrict access to top performance levels that are usually expensive
  a. Helps with power and thermal
  b. Once load balancer extensions are available, can help keep them away from bigger cores
- **uclamp_min** is avoided for portability reasons
  a. Might still be useful, we shall see
- **rampup_multiplier**: manages DVFS response time
  a. Missing from upstream, proposals have been on the list

The QoS Classes

Class	Description	**Suggested** sched_attribute mapping
USER_INTERACTIVE	Requires immediate response.	runtime = 8ms policy = normal rampup_multiplier = 4
USER_INITIATED	Tolerates a short delay, but must respond quickly still.	runtime = 10ms policy = normal rampup_mulitplier = 1
UTILITY	Tolerates long delays, but not prolonged ones.	runtime = 16ms policy = batch rampup_multiplier = 0 uclamp_max = 75%
BACKGROUND	Doesn’t mind prolonged delays.	runtime = 20ms policy = batch/idle rampup_multiplier = 0 uclamp_max = 50%
DEFAULT	System default behavior	Same as UTILITY

How fast does USER_INTERACTIVE need to be?

- The mapping can be further tweaked by adding another dimension, **PERIOD**
- Can help the SchedQos library/middleware tweak the mapping
  a. The info can be passed to the kernel via **sched_attr::period** for additional in kernel help
- We can easily scale to deal with **240Hz** refresh rate vs **30Hz** ones

240Hz

Class	Suggested sched_attribute mapping
USER_INTERACTIVE	runtime = **2ms** policy = normal rampup_multiplier = **8**
USER_INITIATED	runtime = **5ms** policy = normal rampup_mulitplier = **2**
UTILITY	runtime = **8ms** policy = batch rampup_multiplier = 0 uclamp_max = 75%
BACKGROUND	runtime = 20ms policy = batch rampup_multiplier = 0 uclamp_max = 50%

30Hz

Class	Suggested sched_attribute mapping
USER_INTERACTIVE	runtime = **12ms** policy = normal rampup_multiplier = **1**
USER_INITIATED	runtime = **16ms** policy = normal rampup_mulitplier = 1
UTILITY	runtime = 20ms policy = batch rampup_multiplier = 0 uclamp_max = 75%
BACKGROUND	runtime = 30ms policy = batch rampup_multiplier = 0 uclamp_max = 50%

Multimodal wakeup and load balancer

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

A unified task placement decision tree

- We currently do task placement based on 4 criteria that are mutually exclusive
  a. Load
  b. Load + Energy
  c. Load + NUMA
  d. Core scheduling
- Ideally this decision tree shouldn't be mutually exclusive, but hierarchical
  a. Task placement is a very important decision
  b. Modern systems and workloads are hurt by a **wrong fast decision** more than a **slow right decision**
  c. Unifying the decision tree so that all aspects are taken into account is key required change to the wake up path
  d. Multimodal means it can overlap multiple decision criteria for best task placement
     - NUMA can be replaced with generalized better handling of memory bound tasks
     - Latency is one new aspect that is required to be added to the mix, beside load and energy

It’s not only about load

- Current wake up path performs task placement based on load only
  a. EEVDF is per-cpu algorithm, it can’t help with task placement best on best latency globally
  b. Power management (like idle states) introduces extra layer of latency that might need to be taken into account
  c. Scheduler assumes all tasks are CPU bound; it poorly handles memory bounded tasks on system with non symmetric caching properties. Task placement decision need to take memory requirements into account.
- Load balancer is only based on load
  a. Breaks EAS already today
  b. Beside it is slow to react
- We need to generalize wake up path to be multimodal and ensure load balancer follows the same decisions tree made at wake up

What can we do?

- Current Periodic load balance
  a. Not really suitable for a number of cases : EAS, pull on an idle CPU
  b. Often too big or too slow
- Add per task periodic load balance
  a. push callback mechanism
- Use same select function for per task wake up and periodic balance.
  a. consolidated decisions
- Shorter slice runs first
  a. Add slice as an hint for CPU selection
- User estimate of per task utilization of CPU
- Profiling task for better CPU utilization tracking

Easy adoption/deployment

API is not the only way

- APIs have a long cycle to reach end users
  a. API must be created and released
  b. Developers must decide to opt in to the new release and release a new version that take advantage of the new API
  c. Distro must decide to opt in to the newly released software
  d. User has to update
- The cycle overall from the availability of the API until the end user sees it can be 12-18months if not more

Zero API adoption mechanism - config based hinting (1)

- A new mechanism to deploy QoS hinting without requiring API adoption is possible
- A config file can describe the workload attributes and SchedQoS middleware can take care of the rest
- NETLINK already provides an interface for userspace to track tasks creation and name changes to implement a service to tag tasks from a daemon
- Example:

```
{
  "/usr/bin/chrome": {
    "version": "0",
    “period”: “16ms”,
    "thread_qos": {
      "RenderThread": "QOS_USER_INTERACTIVE",
      “NetworkThread”: “QOS_USER_INITIATED”,
    }
}
```

Zero API adoption mechanism - config based hinting (2)

- Only SchedQoS library/middleware needs to be released
  a. Might rely on kernel version if it starts using newly added kernel mechanisms
- Users don't have to wait for app developers to opt-in to QoS hints
  a. Anyone can profile any application and write the config file
- ABI is less of a concern
  a. No binary dependency outside of SchedQoS to translate the QoS into meaningful kernel hints
  b. Rolling/Unrolling becomes order of magnitude simpler
     - Kernel QoS experiments are easier to try out
- In Android we found applications tend to abuse performance APIs provided, need ability to override by system at per-app level.

APIs might still be required, but better avoided

- The **PERIOD** for example can dynamically change in some application
  a. A single static description might not suffice
  b. But can be easily changed by the end user to match their preferred setup
     - e.g: A game can tell libschedqos about PERIOD
     - **OR** a user can **change** the config file to **match** the PERIOD to their FPS setup

Scheduler default values

Historical baggage

- Scheduler default behavior has been historically tuned towards server market (throughput)
- Runtime tunable to allow modify these default values to suit the different needs should be easy
  a. **Throughput**
     - Long slice value
     - Less frequent load balancer, enable load balancer back off
     - HRTICK disabled
  b. **Interactive**
     - Short slice value
     - More frequent load balancer, disable load balancer backoff
     - HRTICK enabled, more frequent context switch to keep responsiveness
  c. **QoS Based**
     - Everything is ‘utility’ by default
       - Simplifies giving better performance for USER_INTERACTIVE and USER_INITIATED tasks by reducing system noise level by default
     - More frequent load balancer, disable load balancer backoff
     - HRTICK enabled, runtime requests would not be precise otherwise leading to losing determinism
     - Disable affinity and nice usage since mostly abused
       - Disallow general modification of sched_attr for non privileged users as they are ‘managed’ in this configuration
     - libc/toolchain default to PI locks

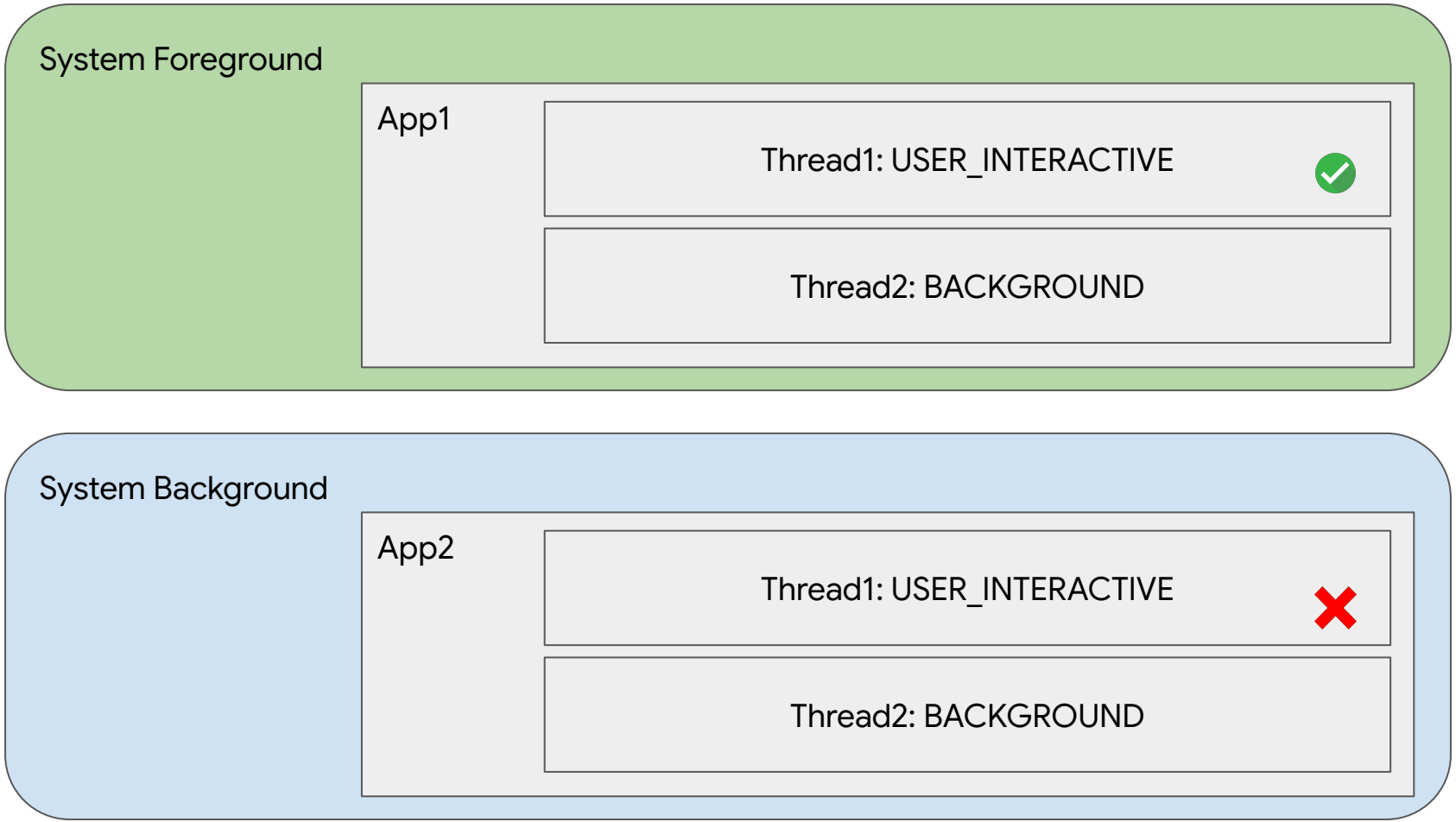
Generalized Performance Inheritance support

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Proxy Execution is the first step only

- We want user space to opt-in by default to use futex_pi
- Inheritance of all sched attributes is important
- Non blocking inheritance is important too, but hairy to implement
    a. Binder
    b. Condition variables
- Discussion in toolchains track on how to make PI the default behavior

Access control


System Foreground
App1
Thread1: USER_INTERACTIVE
Thread2: BACKGROUND
System Background
App2
Thread1: USER_INTERACTIVE
Thread2: BACKGROUND

QoS access control

- Sys Admin (could be the windowing manager like in Android) might want to enforce QoS policies
  a. e.g: background tasks are not allowed to use USER_INTERACTIVE QoS
  b. How to translate this into sched_attr control?
- If we allow APIs, what is the access control if both config based hinting and APIs requests are being made?
  a. A hierarchical QoS application is required
  b. Describing them at the same higher level, not kernel level, is required
     - e.g: **background_group::allow_user_interactive = false/true**

Questions