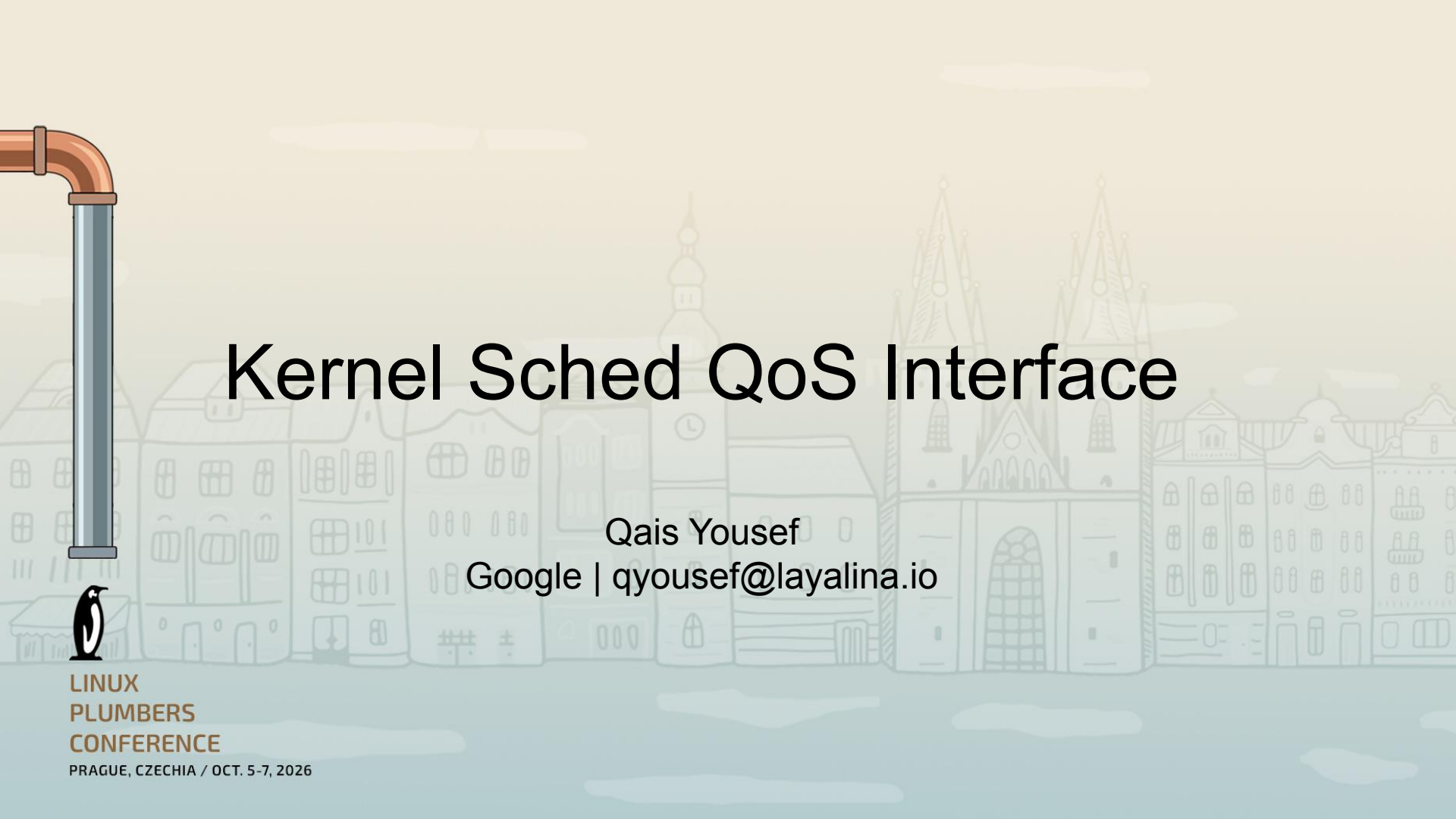




Kernel Sched QoS Interface

Qais Yousef

Google | qyousef@layalina.io



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Why do we need Kernel Sched QoS interface?

Why a new interface?

- As discussed last year, [schedqos](#) service can solve some problems but not all
- Provides an **opaque interface** to handle expanding needs without constantly impacting the syscall
- Goals: **Extendable** and **Deprecatable**
 - Change is the only constant
 - We want to easily add and remove as the code and our environment evolves and changes
 - Harder to deprecate syscall visible changes

Current Proposals

Data Dependency Tagging

Tag data-dependent tasks for **cache-aware scheduling**

DVFS & Upmigration Rampup Control

Use rampup_multiplier to manage DVFS-induced latencies

- Mitigates upmigration delays on HMP systems

Potential Other QoS

Manage performance of a pipeline - eg: game or UI pipeline



The Interface

`sched_qos_type`

Enum to tell what QoS hint to apply

`sched_qos_value`

The value of the QoS hint

`sched_qos_cookie`

Magic cookie to tag a group of tasks for which the QoS applies. If 0, the hint will apply globally system wide. If not 0, the hint will be relative to tasks that has the same cookie value only.

```
struct sched_attr {  
    __u32 size;  
    @@ -116,6 +158,10 @@ struct sched_attr {  
        __u32 sched_util_min;  
        __u32 sched_util_max;  
  
        +    __u32 sched_qos_type;  
        +    __s64 sched_qos_value;  
        +    __u32 sched_qos_cookie;  
        +};
```



Nuances

01 / FORK BEHAVIOR

No inheritance on fork

- To prevent easy overcommit, each QoS must be intentional
- We had lessons learnt from [uclamp support](#)

02 / CGROUP POLICY

No kernel cgroup support

- Userspace based groups policy management is proposed instead. Keep the kernel simple and per task.
- We need new NETLINK events to tell userspace when a task changes group
- Userspace can use the new NETLINK events to create custom policies

03 / ACCESS CONTROL

Fully managed sched_attr

- For QoS to work, the requests has to be fully controlled for the relationship between tasks to make sense
- sched_attr must be locked down and access restricted to a **single running binary**
- **root should fail too**

psst, block affinity too



ABI Protection

01 / STRICT ACCESS RESTRICTION

Single QoS Manager Enforcement

- Restricts interface access to a single binary on any system acting on behalf of all apps. Via capabilities.
- Confines users in practice to the proposed **schedqos** userspace utility or another similar popular fork.
- No random apps breaking hopefully. Admins must set the capability to allow a binary to manage QoS.

02 / DILIGENT ERROR HANDLING

Strict Usage Validation

- Avoid repeating the TCMalloc mistake—explicitly error on all bad or unsupported usages.
- Error on all unexpected values.
- Hopefully easy given the simplicity of the interface?



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Things to think about

01 / SPEED

Syscall Overhead

- Current proposal requires a syscall per QoS set/get request
- Should we provide an array to allow batching?
- Must be answered before merging.

02 / COOKIE MANAGEMENT

Cross-Process Support

- Userspace can manage 'local' cookies for threads within a single process
- How to manage global cookies to span threads across multiple processes?
- Does it make sense to support this case?
- Must be answered before merging.

03 / DISCOVERABILITY

Feature Querying

- Should we provide dedicated interface for currently supported QoS by the running kernel or rely on error returns?
- Can be dealt post merging if necessary.



Local vs Global Cookie Description

One solution is to embed the meaning in the QoS Type

LOCAL COOKIE

```
sched_qos_type = SCHED_QOS_DATA_DEP_LOCAL;  
sched_qos_value = 0;  
// treated as local by the kernel  
sched_qos_cookie = 1;
```

GLOBAL COOKIE

```
sched_qos_type = SCHED_QOS_DATA_DEP_GLOBAL;  
sched_qos_value = 0;  
// treated as global by the kernel  
sched_qos_cookie = 1;
```

Food for thoughts: should unique global cookie be generated by the kernel or fully managed by userspace? How kernel error on mistakes done by userspace if the latter?



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Local vs Global Cookie Description

Or if value is not used, it can be used to choose the mode

LOCAL COOKIE

```
sched_qos_type = SCHED_QOS_DATA_DEP;  
sched_qos_value = MODE_LOCAL;  
// treated as local by the kernel  
sched_qos_cookie = 1;
```

GLOBAL COOKIE

```
sched_qos_type = SCHED_QOS_DATA_DEP;  
sched_qos_value = MODE_GLOBAL;  
// treated as global by the kernel  
sched_qos_cookie = 1;
```



Task behavior vs Implementation details

So this is a very specific feature. This is made possible by basically having a huge type space, allowing for throw-away hints (as per the previous email).

I suppose having these specific hints is easy, but as per always there is the discussion about describing task behaviour vs implementation details. With the argument being that task behaviour might be a more lasting / stable hint, while implementation details are far easier to actually do.

I'm missing this discussion.

How do you see the need for new QoS should be managed/handled?



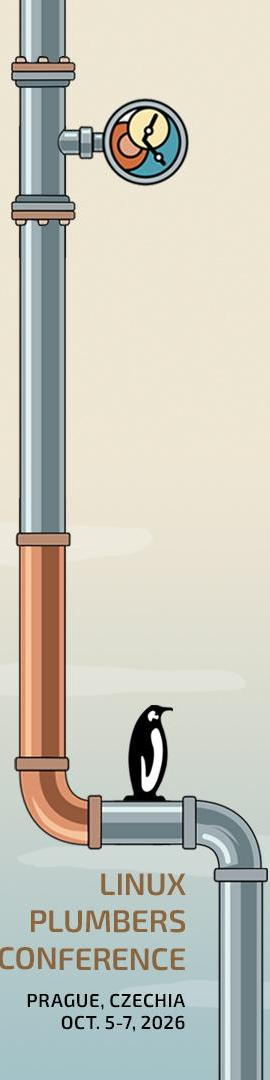
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Discuss(ion)

LKML: <https://lore.kernel.org/lkml/20260504020003.71306-9-qyousef@layalina.io/>



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026