Design a user-space framework to implement sched_ext schedulers

Andrea Righi

Linux Plumbers Conference 2024 | Vienna

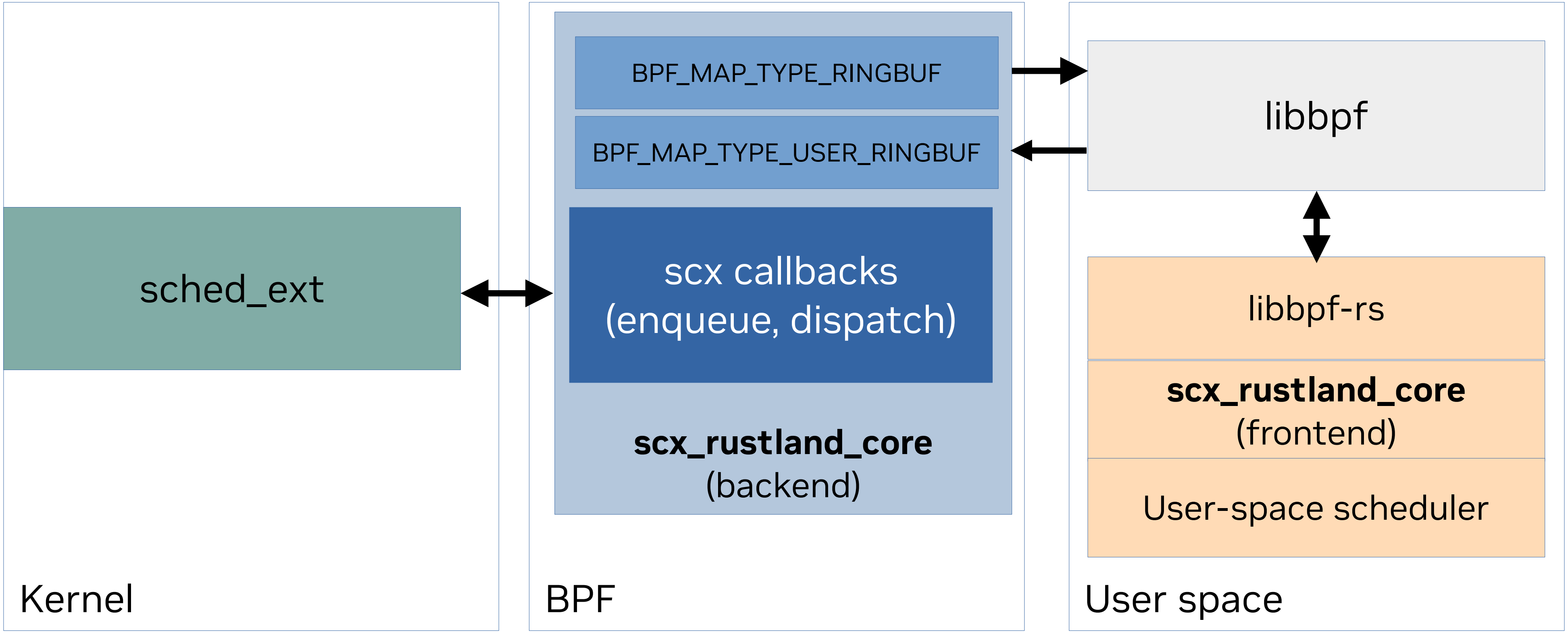
scx_rustland_core

- Abstraction layer over sched_ext
- Interface between BPF/sched_ext and user space
- Kernel scheduler is a user-space process
- Can be used in standalone Rust projects
- GPLv2 license

Goal

- User-space integration (libs, services, …)
- Better debugging and observability
- Fast edit/compile/test iterations
- Quickly prototype and test ideas
- Lower the barrier of scheduling development

Architecture



BPF_MAP_TYPE_RINGBUF
BPF_MAP_TYPE_USER_RINGBUF
libbpf
sched_ext
scx callbacks (enqueue, dispatch)
libbpf-rs
scx_rustland_core (backend)
scx_rustland_core (frontend)
User-space scheduler
Kernel
BPF
User space

NVIDIA

Workflow

- sched_ext callback intercepts tasks that want to run
- Tasks are added to a BPF_MAP_TYPE_RINGBUF
- BPF component schedules a user-space task (scheduler)
- User-space scheduler consumes tasks from the ringbuf and assigns a CPU and time slice to each one of them
- Tasks are added to a BPF_MAP_TYPE_USER_RINGBUF
- BPF component consumes tasks from the user ringbuf and dispatches

scx_rustland_core API

- struct BpfScheduler
  - **Task management**
    - dequeue_task(&mut self) -> Result<Option<QueuedTask>, i32>
      - consume a task that wants to run
    - select_cpu(&mut self, pid: i32, cpu: i32, flags: u64) -> i32
      - find an idle CPU for the task
    - dispatch_task(&mut self, task: &DispatchedTask) -> Result<(), Error>
      - dispatch a task
  - **Completion notification**
    - notify_complete(&mut self, nr_pending: u64)
      - notify BPF component that some tasks have been dispatched

Rust data types

```
struct QueuedTask {
     pub pid: i32,              // pid that uniquely identifies a task
     pub cpu: i32,              // CPU previously used by the task
     pub sum_exec_runtime: u64, // Total cpu time in nanoseconds
     pub weight: u64,           // Task priority [1..10000] (default is 100)
}

struct DispatchedTask {
     pub pid: i32,         // pid that uniquely identifies a task
     pub cpu: i32,         // target CPU selected by the scheduler
     pub flags: u64,       // special dispatch flags
     pub slice_ns: u64,    // time slice in nanoseconds assigned to the task
     pub vtime: u64,       // send task's vruntime/deadline to the BPF dispatcher
}
```

Issues

- User-space scheduler must not be blocked
- Page faults are bad
  - Custom memory allocator in Rust (mlocked arena)
  - vm.compact_unevictable_allowed=0
- Multi-threading is tricky [SOLVED]
- Overhead
  - There is some communication overhead (but it’s not that relevant)
- Less kernel visibility
  - CPU state (e.g., idle cpumasks)

Future plans

- Standardize the user-space framework APIs
- Introduce concept of scheduling domains:
  - Allocate/configure cpumask from user-space
  - Attach a task to a domain (domain ID)
- Call scx_bpf_dispatch() directly from user-space
- Achieve performance identical to BPF/hybrid schedulers

References

- scx_rust_scheduler: simple FIFO scheduler template
  - https://github.com/arighi/scx_rust_scheduler
- scx_rustland_core: main repo
  - https://github.com/sched-ext/scx/blob/main/rust/scx_rustland_core/README.md

Questions?

Andrea Righi
Linux Plumbers Conference 2024 | Vienna