LF Live

MENTORSHIP SERIES

LF live MENTORSHIP SERIES

Designing Custom Linux Schedulers with sched_ext

Andrea Righi, Principal Software Engineer, NVIDIA

Introduction and Agenda

- Scheduling
- sched_ext
- Hands-on

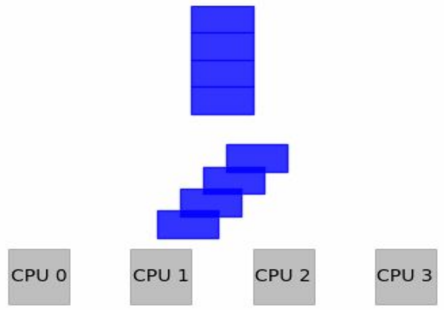
Scope and Expectations

- Become familiar with the sched_ext API
- Being able to craft you own BPF scheduler

Scheduling

What is a scheduler?

- Kernel component that determines
  - **Where** each task needs to run
  - **When** each task needs to run
  - **How long** each task needs to run



Goals

- Fairness
  - All tasks should receive a fair share of the CPU time
- Optimization
  - Make optimal use of system resources
- Low overhead
  - Scheduler should run as fast as possible
- Generalization
  - Scheduler should work on all architectures for every workload

Scheduling in Linux

- One scheduler to rule them all
  - CFS < v6.6
  - EEVDF >= v6.6
- Really difficult to conduct experiments
- Really difficult to upstream changes

sched_ext

- Technology in the Linux kernel that allows implementing scheduling policies as BPF programs
- Key features
  - Design custom schedulers
  - Rapid experimentation
  - Safety (you can’t crash the kernel + watchdog)
  - Lower the barrier of scheduling development

Proportional weight-based CPU allocation: fairness

- Each task T_i has a weight w_i
- The runtime assigned to each task T_i is proportional to its weight w_i divided by the sum of all the runnable tasks' weight

$$runtime(T_i) = \int_{t_0}^{t_1} \frac{w_i}{\sum_{j=0}^{N} w_j} dt \simeq \frac{w_i}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0)$$

Example: 2 tasks $w_0 = 1$, $w_1 = 1$

$$runtime(T_0) = \frac{w_0}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0) = \frac{1}{1+1} \cdot (t_1 - t_0) = \frac{1}{2} \cdot (t_1 - t_0)$$

$$runtime(T_1) = \frac{w_1}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0) = \frac{1}{1+1} \cdot (t_1 - t_0) = \frac{1}{2} \cdot (t_1 - t_0)$$

Example: 2 tasks $w_0 = 1$, $w_1 = 1$

50%

$$runtime(T_0) = \frac{w_0}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0) = \frac{1}{1+1} \cdot (t_1 - t_0) = \frac{1}{2} \cdot (t_1 - t_0)$$

50%

$$runtime(T_1) = \frac{w_1}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0) = \frac{1}{1+1} \cdot (t_1 - t_0) = \frac{1}{2} \cdot (t_1 - t_0)$$

Example: 2 tasks w_0 = 3, w_1 = 1

$$runtime(T_0) = \frac{w_0}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0) = \frac{3}{3+1} \cdot (t_1 - t_0) = \frac{3}{4} \cdot (t_1 - t_0)$$

$$runtime(T_1) = \frac{w_1}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0) = \frac{1}{3+1} \cdot (t_1 - t_0) = \frac{1}{4} \cdot (t_1 - t_0)$$

Example: 2 tasks $w_0 = 3$, $w_1 = 1$

75%

$$runtime(T_0) = \frac{w_0}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0) = \frac{3}{3+1} \cdot (t_1 - t_0) = \frac{3}{4} \cdot (t_1 - t_0)$$

25%

$$runtime(T_1) = \frac{w_1}{\sum_{j=0}^{N} w_j} \cdot (t_1 - t_0) = \frac{1}{3+1} \cdot (t_1 - t_0) = \frac{1}{4} \cdot (t_1 - t_0)$$

How this is implemented: virtual runtime (vruntime)

- Charge each task a runtime proportional to w_{base} and inversely proportional to its weight w_i (vruntime)
- Traditionally w_{base} = 100
- Tasks are scheduled in order of increasing vruntime

$$vruntime(T_i) = \frac{w_{base}}{w_i} \cdot (t_1 - t_0) = \frac{100}{w_i} \cdot (t_1 - t_0)$$

EEVDF: Earliest Eligible Virtual Deadline First

- **Lag**: difference between the ideal runtime and the actual runtime of a task
- **Eligibility**: a task is eligible to run if its lag is >= 0
- **Virtual deadline**: vruntime + requested vruntime

$$lag_T(t_1) = V_{avg}(t_1) - V_T(t_1) \geq 0$$

$$D_T(t_1) = V_T(t_1) + \Delta t_T \cdot \frac{w_{base}}{w_i}$$

sched_ext

sched_ext implementation

- New scheduling policy (SCHED_EXT) that runs below the fair sched class
- When a BPF scheduler is loaded, all fair sched class tasks are moved to SCHED_EXT
  - Partial mode: tasks must be explicitly moved to the SCHED_EXT class via sched_setscheduler()
- sched_ext invokes BPF callbacks via struct_ops: ops.*()
- When a BPF scheduler exits, all SCHED_EXT tasks are moved back to the fair class

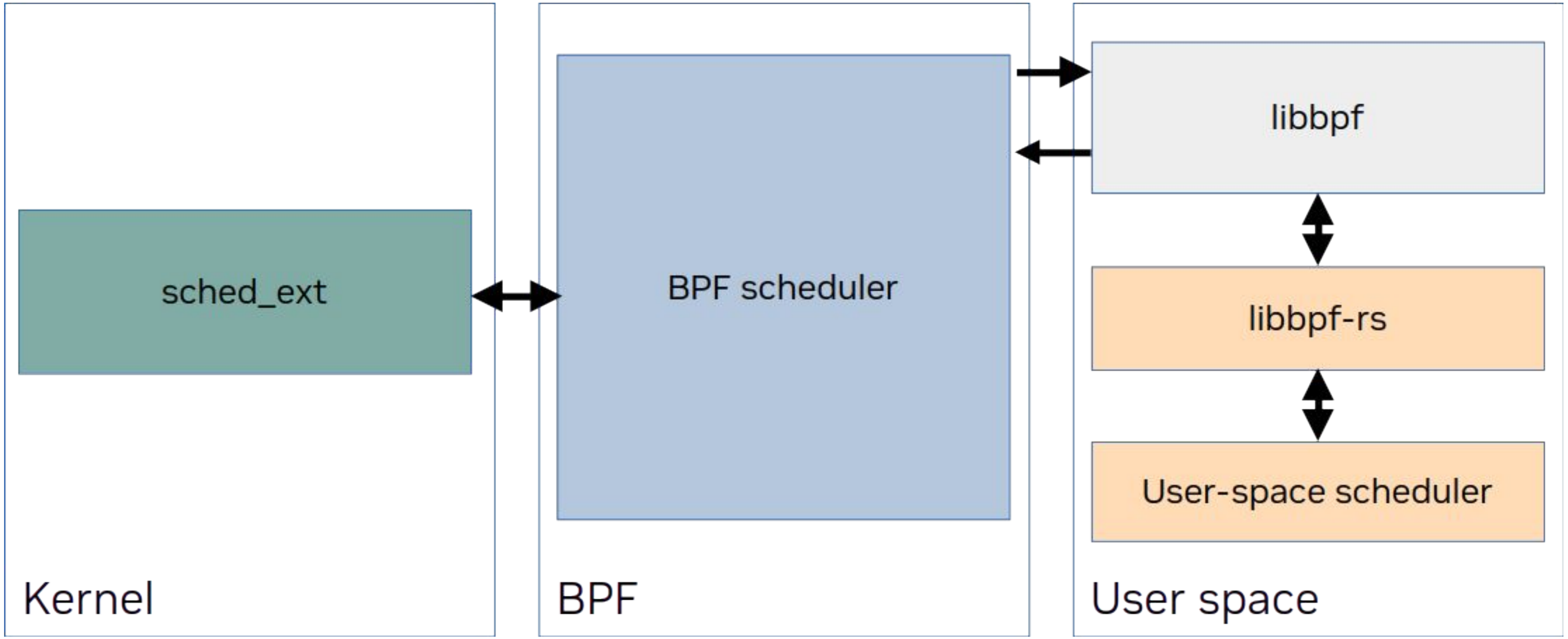
Scheduling classes and policies in Linux

- dl_sched_class
  - SCHED_DEADLINE
- rt_sched_class
  - SCHED_FIFO, SCHED_RR
- fair_sched_class
  - SCHED_NORMAL, SCHED_BATCH
- ext_sched_class
  - SCHED_EXT **<= Your BPF program runs here!**
- Idle_sched_class:
  - SCHED_IDLE

Kernel ⇔ kernel ABI

- No UAPI restrictions
  - Linux can’t break user space => ABI restrictions
  - BPF is not user space => kernel-to-kernel ABI
  - BPF struct_ops => trigger callbacks in BPF
- BPF program shares address space with user space
  - User space task can access BPF address space
  - Use BPF to communicate between BPF and user-space
- BPF program **must** be GPLv2

User-space scheduling (Rust, Java, …)



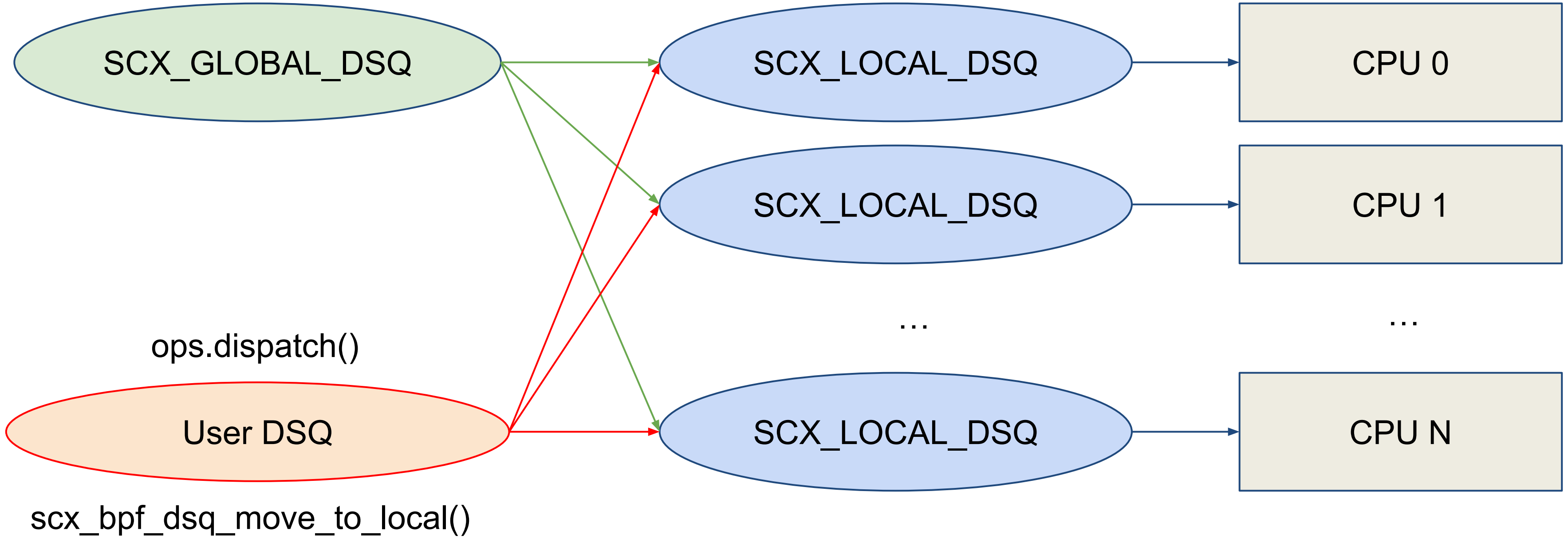
BPF scheduler implementation

```
SCX_OPS_DEFINE(vder_ops,
               .select_cpu                 = (void *)vder_select_cpu,
               .enqueue                    = (void *)vder_enqueue,
               .dispatch                   = (void *)vder_dispatch,
               .runnable                   = (void *)vder_runnable,
               .running                    = (void *)vder_running,
               .stopping                   = (void *)vder_stopping,
               .quiescent                  = (void *)vder_quiescent,
               .enable                     = (void *)vder_enable,
               .init_task                  = (void *)vder_init_task,
               .cpu_release                = (void *)vder_cpu_release,
               .init                       = (void *)vder_init,
               .exit                       = (void *)vder_exit,
               .timeout_ms                 = 5000,
               .name                       = "vder");
```

DSQ (Dispatch queue)

- **SCX_DSQ_LOCAL**: one on each CPU
- **SCX_DSQ_LOCAL_ON | cpu**: local DSQ of a target CPU
- **SCX_DSQ_GLOBAL**: global DSQ
- **User-defined DSQs**: custom DSQs

DSQs


SCX_GLOBAL_DSQ
SCX_LOCAL_DSQ
SCX_LOCAL_DSQ
SCX_LOCAL_DSQ
CPU 0
CPU 1
CPU N
...
...
ops.dispatch()
User DSQ
scx_bpf_dsq_move_to_local()

DSQ workflow

- Tasks are always executed in FIFO order from the local DSQ of each CPU
- Every time resched_curr() is called and the current task's time slice is zero, a new task is consumed:
  - If the local DSQ is empty, the first task from the built-in global DSQ is moved to the local DSQ
  - Otherwise, ops.dispatch() is called and we can consume tasks from user DSQs via scx_bpf_dsq_move_to_local()

FIFO vs priority DSQ

- Built-in DSQs can only be used as FIFO
- User-defined DSQs can be used either as FIFO or priority (in a mutually exclusive way)

DSQ methods

- **scx_bpf_dsq_insert()**: insert a task to a DSQ (used as FIFO)
- **scx_bpf_dsq_insert_vtime()**: insert a task to a DSQ (used as priority)
- **scx_bpf_dsq_nr_queued()**: return the number of tasks in the DSQ
- **scx_bpf_dsq_move_to_local()**: pop the first task from a user DSQ and move it to the current CPU's local DSQ
- **scx_bpf_create_dsq()**: create a new user DSQ

Task lifecycle

- **ops.init_task()**: a new task is created
- **ops.enable()**: enable BPF scheduling for a task
- **ops.select_cpu()**: called on task wakeups (optimization)
- **ops.runnable()**: task becomes ready to run
- **ops.enqueue()**: task is added to the scheduler (skipped if already dispatched)
- **ops.running()**: task starts running on its assigned CPU
- **ops.tick()**: called every 1/HZ seconds
- **ops.stopping()**: task stops running on its assigned CPU
- **ops.quiescent()**: task releases its assigned CPU (wait)
- **ops.disable()**: disable BPF scheduling for a task
- **ops.exit_task()**: a task is destroyed

```
ops.init_task()
ops.enable_task()

while (is_sched_ext_task(p)) {
    if (task_can_migrate(p))
        ops.select_cpu();

    ops.runnable();
    while (!is_task_blocked()) {
        if (!is_task_dispatched())
            ops.enqueue();
        ops.running();
        ops.tick();
        ops.stopping();
    }
    ops.quiescent();
}

ops.disable_task(()
ops.exit_task()
```

← **Task workflow**

Task properties

- Mutable attributes:
  - **p->scx.slice**: task time slice in ns, decremented every tick, if zero resched_curr() is called
  - **p->scx.dsq_vtime**: defines the position of a task in a priority DSQ
  - **p->scx.disallow**: exclude the task from SCHED_EXT
- Immutable attributes
  - **p->scx.weight**: task's weight (priority) in the range [1 … 10000], where 100 is the default

CPU callbacks

- **ops.dispatch()**: a CPU is ready to dispatch a task
  - If a task is moved to the local DSQ, it will be executed on the CPU
  - If the current task time slice is refilled, the task will continue to run on the CPU
  - If no action is done, the CPU goes idle
- **ops.update_idle()**: called when a CPU is entering or exiting the idle state

CPU methods

- **scx_bpf_select_cpu_dfl()**: return the optimal idle CPU for a target task
- **scx_bpf_pick_idle_cpu()**: return an idle CPU usable by a target task (if present)
- **scx_bpf_pick_any_cpu()**: return any CPU usable by a targe task
- **scx_bpf_kick_cpu()**: wakeup an idle CPU or trigger a reschedule event

cpufreq integration (schedutil)

- **scx_bpf_cpuperf_cap()**: return the maximum relative capacity of a CPU in relation to the most performant CPU in the system
- **scx_bpf_cpuperf_cur()**: return the current performance level of a CPU
- **scx_bpf_cpuperf_set()**: set the performance target of a CPU

- Performance level is in the range **[1 .. SCX_CPUPERF_ONE]**

sched_ext error handling

- If the BPF scheduler triggers an error, it is automatically terminated and a trace is returned to the user-space program that loaded it
- A kernel watchdog monitors whether all tasks are scheduled at least once within a configurable time; if any task fails to run within this time period, the scheduler triggers a stall error

sched_ext error trace

```
CPU 6   : nr_run=1 flags=0x1 cpu_rel=0 ops_qseq=54 pnt_seq=11515
          curr=yes[298] class=rt_sched_class

  R kworker/6:1[155] -7168ms
      scx_state/flags=3/0x9 dsq_flags=0x0 ops_state/qseq=0/0
      sticky/holding_cpu=-1/-1 dsq_id=0x8000000000000002 dsq_vtime=1054861 slice=20000000
      cpus=40

    kthread+0xdf/0x110
    ret_from_fork+0x31/0x50
    ret_from_fork_asm+0x1a/0x30
```

sched_ext error trace

```
CPU 1   : nr_run=2 flags=0x1 cpu_rel=0 ops_qseq=1648 pnt_seq=10281
          curr=dxvk-shader-n[20288] class=ext_sched_class

 *R dxvk-shader-n[20288] -1ms
      scx_state/flags=3/0xd dsq_flags=0x0 ops_state/qseq=0/0
      sticky/holding_cpu=-1/-1 dsq_id=(n/a) dsq_vtime=101176519145
      cpus=3f

  R kworker/1:0[29] -5056ms
      scx_state/flags=3/0x9 dsq_flags=0x1 ops_state/qseq=0/0
      sticky/holding_cpu=-1/-1 dsq_id=0x0 dsq_vtime=0
      cpus=02

    kthread+0x14a/0x170
    ret_from_fork+0x37/0x50
    ret_from_fork_asm+0x1a/0x30
```

LF Live MENTORSHIP SERIES

Hands-on

How to get started with sched_ext

- Make sure your kernel has **CONFIG_SCHED_CLASS_EXT=y**
- Clone the scx git repository
  - https://github.com/sched-ext/scx
- Demo branch
  - https://github.com/sched-ext/scx/tree/demo
- Kernel experimentation (hint: use virtme-ng)
  - https://git.kernel.org/pub/scm/linux/kernel/git/tj/sched_ext.git

Future plans

- Improve schedulers composability
- Allow to load multiple scx schedulers at the same time
- More in-kernel statistics
- Improve built-in functionalities (i.e., built-in idle selection policy)
- Paravirtualization / cooperative scheduling
- Device / IRQ affinity

References

- scx git repository
  - https://github.com/sched-ext/scx
- sched_ext kernel repository
  - https://git.kernel.org/pub/scm/linux/kernel/git/tj/sched_ext.git
- sched_ext slack channel
  - https://bit.ly/scx_slack
- Earliest Eligible Virtual Deadline First (EEVDF): A flexible and Accurate Mechanism for Proportional Share Resource Allocation
  - https://citeseerx.ist.psu.edu/document?doi=805acf7726282721504c8f00575d91ebfd750564
- The Linux Scheduler: a Decade of Wasted Cores
  - https://people.ece.ubc.ca/sasha/papers/eurosys16-final29.pdf
- Rust user-space scheduler template
  - https://github.com/arighi/scx_rust_scheduler

LF live MENTORSHIP SERIES

Thank you for joining us today!

We hope it will be helpful in your journey to learning more about effective and productive participation in open source projects. We will leave you with a few additional resources for your continued learning:

- The LF Mentoring Program is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- Outreachy remote internships program supports diversity in open source and free software
- Linux Foundation Training offers a wide range of free courses, webinars, tutorials and publications to help you explore the open source technology landscape.
- Linux Foundation Events also provide educational content across a range of skill levels and topics, as well as the chance to meet others in the community, to collaborate, exchange ideas, expand job opportunities and more. You can find all events at events.linuxfoundation.org.