ANDREA RIGHI

Principal System Software Engineer @ NVIDIA

Crafting a Linux kernel scheduler in Rust



RUSTLAB The International Conference on Rust in Florence | November 9th, 2024 → November 11th, 2024

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

Why does scheduling matter?

- Performance
 - Workload
 - Topology
- Security
 - Isolation
- Energy Efficiency
 - EAS

CPU topology can be complex...

Machine (31GB total)													
Package L#0													
NUMANode L#0 P#0 (31GB)													
L3 (24MB)													
L2 (1280KB)	L2 (2048KB)				L2 (2048KB)								
L1d (48KB)	L1d (32KB)	L1d (32KB)	L1d (32KB)	L1d (32KB)									
L1i (32KB)	L1i (64KB)	L1i (64KB)	L1i (64KB)	L1i (64KB)									
Core L#0 PU L#0 P#0 PU L#1 P#1	Core L#1 PU L#2 P#2 PU L#3 P#3	Core L#2 PU L#4 P#4 PU L#5 P#5	Core L#3 PU L#6 P#6 PU L#7 P#7	Core L#4 PU L#8 P#8 PU L#9 P#9	Core L#5 PU L#10 P#10 PU L#11 P#11	Core L#6 PU L#12 P#12	Core L#7 PU L#13 P#13	Core L#8 PU L#14 P#14	Core L#9 PU L#15 P#15	Core L#10 PU L#16 P#16	Core L#11 PU L#17 P#17	Core L#12 PU L#18 P#18	Core L#13 PU L#19 P#19

Host: 9ddd680-lcelt Date: Sun 03 Nov 2024 08:49:27 AM CET

[From `lstopo` on a Dell Precision 5480 equipped with 13th Gen Intel(R) Core(TM) i7-13800H CPUs]/

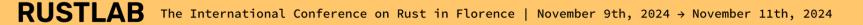
Scheduling in Linux

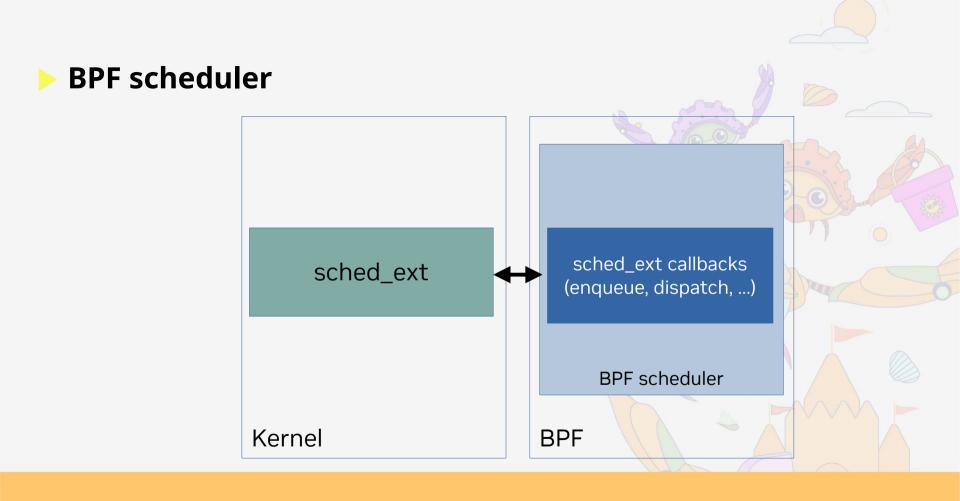
- One scheduler to "rule them all"
 - CFS < v6.6
 - EEVDF >= v6.6
- Really difficult to conduct experiments
- Really difficult to upstream changes

sched_ext

- Implement custom CPU schedulers as loadable BPF programs
- BPF guarantees safety (no kernel panic, memory bugs, ...)
- Watchdog prevents deadlock and starvation
- Available in Linux v6.12







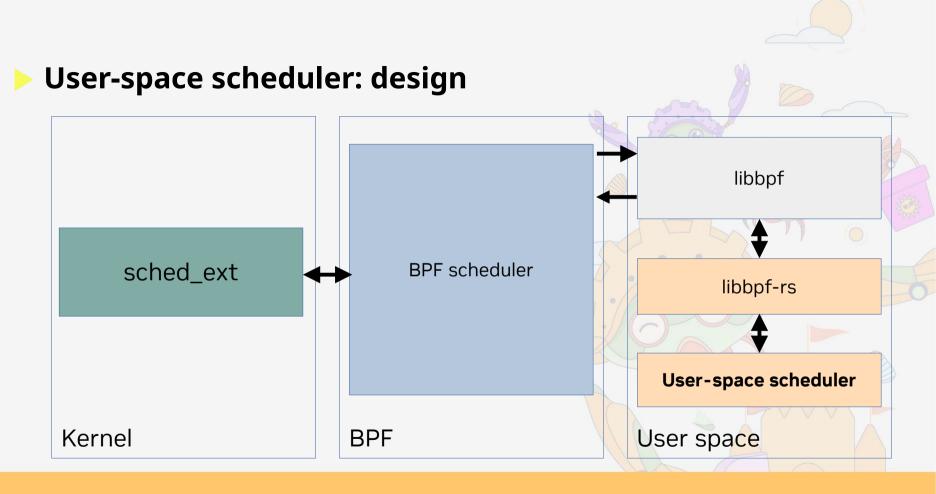
sched_ext pros / cons

- Pros
 - Ease of experimentation
 - Fast edit/compile/test iteration
 - Safety
- Cons
 - Limited programming model
 - BPF verifier complexity
 - Kernel restrictions (no user-space libs, no floating point, etc.)

User-space scheduling

Idea

- Use BPF + sched_ext to channel scheduling events to user space
- A scheduler becomes a regular user-space process
- Offload complexity to user space
- Access to user-space languages
 - Use Rust!



scx_rustland

- EDF-based scheduler
 - Deadline is evaluated as a function of the task's vruntime and the average amount of voluntary context switches
- Tasks use a variable time slice
 - Time slice inversely proportional to the amount of tasks waiting to be scheduled

Is it working?



Demo: playing Terraria while building the kernel



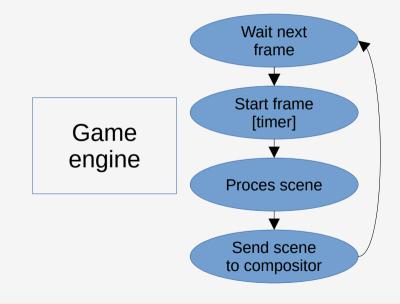


EEVDF vs scx_rustland - https://perfetto.dev

	Сри О		cl cl c clang cl cl d cl cl c clang cl d
	Сри 1		fin chi fin chi chi chi chi ha chi fin chi chi chi chi
	Cpu 2	cl. c ./Ter clang. c ./Ter. P: /Terraria.bin.x86_64 [94489] . clan. /Te. clan. c. c. clang. cl. /T. clan. clang. /Ter clang. c ./Ter. clan. clang. c ./Ter. clan. clang. c ./Ter c ./Te	lang c cla
EEVDF	Сри З		
	Сри 4		
	Сри 5		
	Сри б		
	Cpu 7		
		Nº .	
	Сри О		f fi c c c / f c c Ma
	Сри 1		C C C C C C C C C C C
	Cpu 2	a a a a a a a a a a a a a a a a a a a	
scx_rustland	Сри 3	စ္ရ ေရွက်ော္တေျမာင္းမေရ ေရေရာက္ ေရာက္ရွိရာ ေရာက္ ေ	
_	Cpu 4	c c c c c c c c c c c c c c c c c c c	
	Сри 5	e c. c. c. c. c. kw. da. d. c. c. J. c. c. c. c. c. e. J. Kw. J. c.	
	Сри б	<mark> √</mark> C C C C C C C C C	
	Cpu 7	lo c. f. lo c. c. c. d. da. d. c. d. c. c. da. c c. da. c c. d <mark>./ v. b./ d. c.</mark> c. d. c. d. c. d. d. c. d. d. c. d. d. d. d. d. c. d.	

Why is it better?

- Interactive workloads are typically cyclic (pipeline)
- Tasks release the CPU voluntarily



Generalize user-space scheduling

scx_rustland_core framework

- Abstract scx_rustland backend
- Define generic scheduling API
- Provide a Rust crate (scx_rustland_core)
- Allow to implement Linux schedulers easily as regular Rust projects

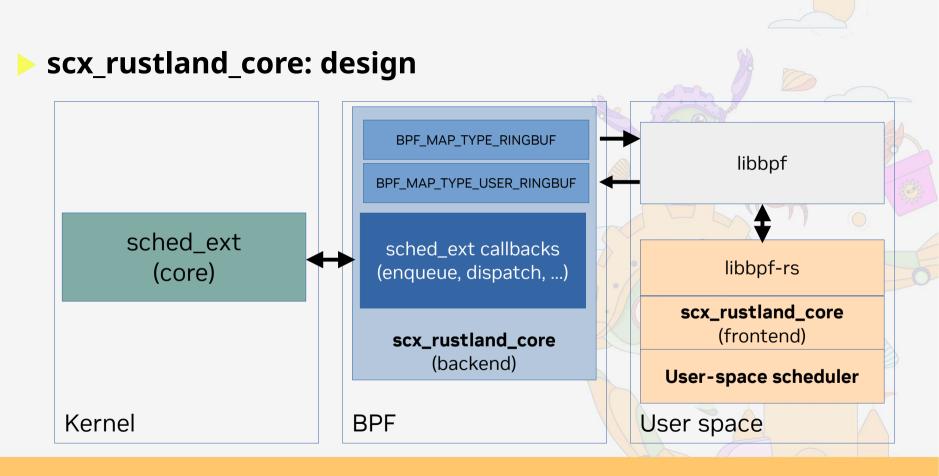
FIFO scheduler (using the scx_rustland_core crate)

```
fn schedule(&mut self) {
let nr waiting = *self.bpf.nr queued mut();
while let Ok(Some(task)) = self.bpf.dequeue task() {
    let mut dispatched task = DispatchedTask::new(&task);
    let cpu = self.bpf.select cpu(task.pid, task.cpu, 0);
    dispatched task.cpu = if cpu >= 0 { cpu } else { RL CPU ANY };
    dispatched task.slice ns = SLICE NS / (nr waiting + 1);
    self.bpf.dispatch task(&dispatched task).unwrap();
self.bpf.notify complete(0);
```

AI-generated schedulers?







Caveats

- User-space scheduler can't be blocked
 - Page faults are bad
 - Override the Rust allocator via GlobalAlloc
 - Work on a pre-allocated mlock()ed memory arena

Conclusion



Key takeaways

- scx_rustland is not a better scheduler in general
- Rust itself doesn't make the scheduling better
- Ease of experimentation is the key
 - Fast edit/compile/test cycle
 - Integration with user-space components (Rust crates)

Future idea

- What if we provide a similar technology for other kernel subsystems?
 - Drivers

. . .

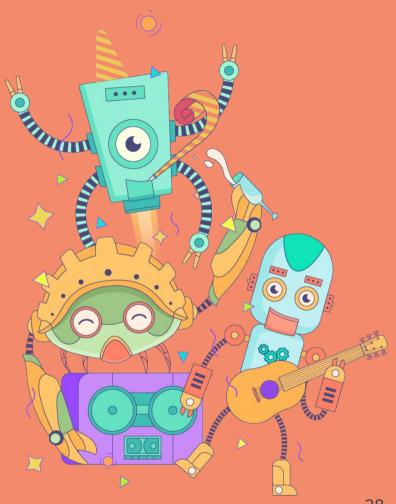
- Filesystems

 Implement more kernel subsystems in Rust (without adding Rust into the kernel)

References

- Main scx project
 - https://github.com/sched_ext/scx
- Rust scheduler template
 - https://github.com/arighi/scx_rust_ scheduler
- LWN.net sched_ext at LPC
 - https://lwn.net/Articles/991205

Questions?



ANDREA RIGHI

arighi@nvidia.com

