

sched_ext

Шаго Павел Евгеньевич

Научный руководитель
к.ф.-м.н., доцент Корхов В. В.

Санкт-Петербургский государственный университет
Факультет прикладной математики – процессов управления
Кафедра моделирования электромеханических и компьютерных систем

29 апреля 2025 г.

Цель и задачи работы

Цель

- ▶ Исследование новой подсистемы планировщика в Linux

Задачи

- ▶ Изучение применимости подсистемы и её реализаций
- ▶ Составление и вывод результатов из виртуальных окружений (демо)

Что такое sched_ext?

`sched_ext` – это инфраструктура позволяющая описывать планировщик набором bpf программ (bpf scheduler)

1. `sched_ext` экспортирует полноценный (необходимый) интерфейс планирования, так что любой алгоритм можно реализовать поверх этого интерфейса
2. `sched_ext` планировщик можно в любой момент включить или выключить
3. Целостность системы сохранена вне зависимости от того что делает планировщик, в случае ошибок спускаемся до стандартного поведения, управление получает EEVDF, а от подсистемы получаем информативную dump трассу

Преимущества подхода

- ▶ Кастомизация под различное поведение
- ▶ Возможность попробовать различные эвристики
- ▶ Большая безопасность
- ▶ Поддержка со стороны крупного бизнеса (Google, Meta')

Минусы

- ▶ Поддержка только с версии Linux 6.12 (ноябрь 2024)

Пример: FIFO планировщик с sched-ext

```
1 void BPF_STRUCT_OPS(simple_enqueue, struct task_struct *p, u64 enq_flags)
1 {
2     scx_bpf_dispatch(p, SCX_DSQ_GLOBAL, SCX_SLICE_DFL, enq_flags);
3 }
4
5 void BPF_STRUCT_OPS(simple_dispatch, s32 cpu, struct task_struct *prev)
6 {
7     scx_bpf_consume(SCX_DSQ_GLOBAL);
8 }
9
10 s32 BPF_STRUCT_OPS(simple_init)
11 {
12     scx_bpf_switch_all();
13     return 0;
14 }
15
16 SEC(".struct_ops.link");
17 struct sched_ext_ops simple_ops = {
18     .enqueue   = (void *)simple_enqueue,
19     .dispatch  = (void *)simple_dispatch,
20     .init      = (void *)simple_init,
21     .name      = "simple",
22 };
```

NUMA aware (демо 1)

Commit **bc96fcd**



arighi committed on Feb 27 · ✓ 50 / 52

`scx_bpfland: NUMA-aware scheduling`

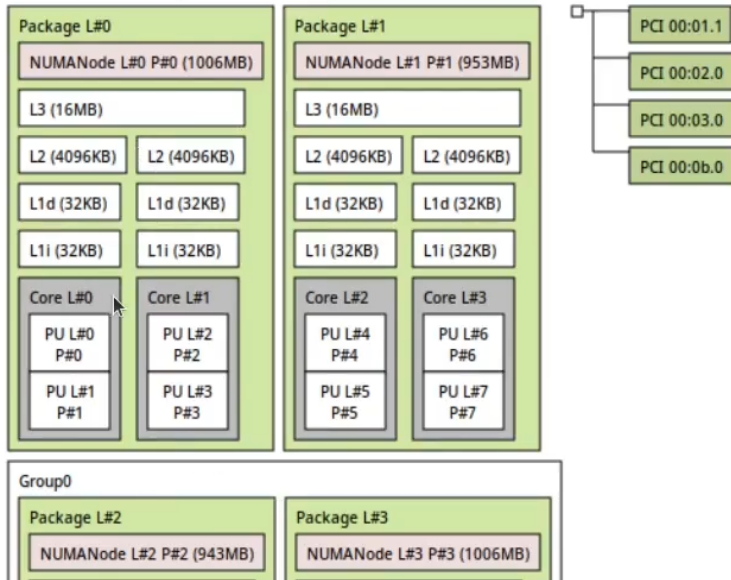
Use the new `sched_ext` node-aware APIs to account for NUMA topologies in the scheduling decisions.

To do so, modify the scheduler as following:

- split the shared DSQ into multiple a per-node DSQs,
- tasks will be always enqueued to the DSQ of their assigned NUMA node,
- prioritize picking idle CPUs within the same NUMA node,
- on wakeup, tasks have a chance to migrate to a different node if all the CPUs in their current node are busy.

NUMA aware (демо 1)

Machine (3908MB total)



NUMA aware (демо 1)

```
vng -vr ~/src/linux --cpu 16,sockets=4,cores=2,threads=2 -m 4G \  
--numa 1G,cpus=0-3 --numa 1G,cpus=4,7 --numa 1G,cpus=8-11, \  
--numa 1G,cpus=12-15 --numa-distance 0,1=51 --numa-distance 0,2=31 \  
--numa-distance 0,3=41 --numa-distance 1,2=21 --numa-distance 1,3=61 \  
--numa-distance 2,3=11
```

```
node 1 cpus: 4 5 6 7  
node 1 size: 1006 MB  
node 1 free: 979 MB  
node 2 cpus: 8 9 10 11  
node 2 size: 889 MB  
node 2 free: 866 MB  
node 3 cpus: 12 13 14 15  
node 3 size: 1005 MB  
node 3 free: 981 MB  
node distances:
```

node	0	1	2	3
0:	10	51	31	41
1:	51	10	21	61
2:	31	21	10	11
3:	41	61	11	10

CPU	NODE	SOCKET	CORE	L1d:L1i:L2:L3	ONLINE
0	0	0	0	0 0:0:0:0	yes
1	0	0	0	0 0:0:0:0	yes
2	0	0	1	1 1:1:1:0	yes
3	0	0	1	1 1:1:1:0	yes
4	1	1	2	2 2:2:2:1	yes
5	1	1	2	2 2:2:2:1	yes
6	1	1	3	3 3:3:3:1	yes
7	1	1	3	3 3:3:3:1	yes
8	2	2	4	4 4:4:4:2	yes
9	2	2	4	4 4:4:4:2	yes
10	2	2	5	5 5:5:5:2	yes
11	2	2	5	5 5:5:5:2	yes
12	3	3	6	6 6:6:6:3	yes
13	3	3	6	6 6:6:6:3	yes
14	3	3	7	7 7:7:7:3	yes
15	3	3	7	7 7:7:7:3	yes

NUMA aware (демо 1)

```
0[|||||||||||||||||||||||||||||||||||||||||100.0%] 8[| 0.7%]
1[|||||||||||||||||||||||||||||||||||||||||100.0%] 9[| 0.0%]
2[|||||||||||||||||||||||||||||||||||||||||0.0%] 10[| 0.0%]
3[|||||||||||||||||||||||||||||||||||||||||0.0%] 11[| 0.0%]
4[|||||||||||||||||||||||||||||||||||||||||0.0%] 12[| 0.0%]
5[|||||||||||||||||||||||||||||||||||||||||0.0%] 13[| 0.0%]
6[|||||||||||||||||||||||||||||||||||||||||0.0%] 14[| 0.0%]
7[|||||||||||||||||||||||||||||||||||||||||0.0%] 15[| 0.0%]

scx_bpfland 1.0.11-g58da3bc9-dirty x86_64-unknown-linux-gnu SMT on
scheduler flags: 0x76
primary CPU domain = 0xffff
cpufreq performance level: max
SMT sibling CPUs: [1, 0, 3, 2, 5, 4, 7, 6, 9, 8, 11, 10, 13, 12, 15, 14]
L3 cache ID 0: sibling CPUs: [0, 1, 2, 3]
L3 cache ID 1: sibling CPUs: [4, 5, 6, 7]
L3 cache ID 2: sibling CPUs: [10, 11, 8, 9]
L3 cache ID 3: sibling CPUs: [12, 13, 14, 15]
sched_ext: BPF scheduler "bpfland" enabled
```

```
0[|||||||||||||||||||||||||||||||||||||||||100.0%] 8[|||||||||||||||||||||||||||||||||||||100.0%]
1[|||||||||||||||||||||||||||||||||||||||||0.0%] 9[|||||||||||||||||||||||||||||||||||||0.0%]
2[|||||||||||||||||||||||||||||||||||||||||0.0%] 10[|||||||||||||||||||||||||||||||||||||0.0%]
3[|||||||||||||||||||||||||||||||||||||||||100.0%] 11[|||||||||||||||||||||||||||||||||||||100.0%]
4[|||||||||||||||||||||||||||||||||||||||||0.0%] 12[|||||||||||||||||||||||||||||||||||||0.0%]
5[|||||||||||||||||||||||||||||||||||||||||0.0%] 13[|||||||||||||||||||||||||||||||||||||1.3%]
6[|||||||||||||||||||||||||||||||||||||||||0.0%] 14[|||||||||||||||||||||||||||||||||||||0.0%]
7[|||||||||||||||||||||||||||||||||||||||||0.0%] 15[|||||||||||||||||||||||||||||||||||||0.0%]
```

Как уже используется sched_ext

Цель

- ▶ Оптимизировать производительность продакшн-нагрузок в Meta', где ключевой метрикой является 99-й перцентиль задержки ответов. Загрузка CPU обычно 40%, но все равно рассматривается тюнинг планировщика.

Основные направления экспериментов

- ▶ Work-conservation – не простаивают ли CPU, когда есть работа?
- ▶ Idle CPU selection – как правильно выбирать CPU при пробуждении потока?
- ▶ Soft-affinity – стоит ли закрепить второстепенные задачи за конкретными CPU?
- ▶ Custom policies – использовать ли разные политики для разных типов потоков?

Как уже используется sched_ext

scx_simple

- ▶ Поддерживает work-conservation: все CPU делят один dsq
- ▶ Использует `scx_select_cpu_dfl()` —отдаёт предпочтение полностью свободным ядрам, не стремясь к локальности кешей
- ▶ Результат: 3.5% прирост на 1000 машинах
- ▶ Подтверждено: приоритет полностью свободным ядрам —ключевой фактор прироста

scx_layered

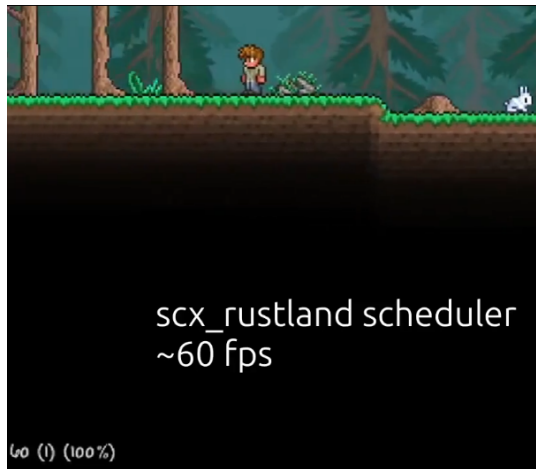
- ▶ Позволяет разделять потоки на слои с разными политиками
- ▶ Высокоприоритетные потоки могут прерывать любые другие
- ▶ В итоге выбран в качестве базы

Учет перспективы (демо 2)






```
CC lib/zlib_inflate/infutil.o
AR fs/notify/inotify/built-in.a
AR fs/notify/fanotify/built-in.a
CC fs/notify/fsnotify.o
CC drivers/video/fbdev/core/fbmem.o
CC lib/crypto/sha1.o
CC arch/x86/mm/init_64.o
CC init/version.o
CC arch/x86/kernel/acpi/boot.o
CC arch/x86/kernel/cpu/mce/genpool.o
CC lib/zlib_inflate/inftrees.o
CC arch/x86/entry/vdso/vgetcpu.o
CC arch/x86/pci/direct.o
CC arch/x86/kernel/fpu/signal.o
AR drivers/pci/msi/built-in.a
```



Учет перспективы (демо 2)



Список использованных источников

-  [arighi_linux](#)
-  Исходный код `scx`, `scx_bpfland` & `scx_rustland`
-  Исходный код ядра Linux в т. ч. документация к ядру Linux
-  Подписка на почтовую рассылку `linux-kernel@vger.kernel.org`
-  Andrew S. Tanenbaum, Herbert Bos MODERN OPERATING SYSTEMS Pearson Education 2024