Sprawozdanie z projektu na Oprogramowanie Systemowe

Łukasz Kołakowski

17 Stycznia 2026

1 Co udało się zrealizować

Moim zadaniem było napisanie planisty (scheduler) na system operacyjny Linux. W trakcie mojego *researchu* natrafiłem na zaimplementowanego planistę z użyciem eBPF. Skupiłem się na zrozumieniu jak działa eBPF i jak z niego korzystać, jest do tego książka [link] z której przeczytałem najważniejsze rozdziały.

Po tym czasie skończył się czas na przeszukiwanie internetu i nadszedł czas implementacji, przez to iż skupiłem się na tym jak działa eBPF, musiałem szukać więcej informacji na temat tego jak korzystać z *sched_ext*, rozszerzenia udostępnionego przez jądro linuxa, które umożliwa dodawanie własnych planistów, jeżeli polityki udostępnione standardowo nie są wystarczająco optymalne.

Udało mi się zaimplementować na początku zaimplementować minimalnego planistę, jako test czy *sched_ext* działa jak powinien. Implementuje on politykę typu *round robin* ze zmienną porcją przydzielanego czasu. Kolejka jest wspólna dla szyskich rdzeni procesora. Poczym postarałem się rozszerzyć planistę, dodatkowo zapisuje i wypisuje użytkownikowi informacje o procesach, którym przedziela czas. Zapisuje statystyki takie jak:

- przydzielony czas;
- pid;
- krótka nazwa prcesu;
- łączny czas czekania na czas procesora;
- maksymalny czas czekania.

Zobaczyłem jak zachowuje się planista, jeżeli przydzielimy bardzo kawałek czasu (sekundy, setne sekund) lub jak będzie on bardzo mały (nanosekudny).

Dodatkowo chciałem zobaczyć czy da się zagłodzić niechciane zadania, celowo nie przydzielając im czasu procesora.

2 Problemy na jakie natrafiłem

2.1 BCC czy libbpf?

BCC jest starszą biblioteką umożliwiającą pisanie progamów przy pomocy eBPF, jest on starszy …

2.2 Rozbieżność wersji

W przykładach nazwa funkcji do przełączenia zadań na procesorze nazywa się:

```
scx_bpf_consume(u64 dsq_id)
```

a w wersji jądra 6.14 nazwa została zmieniona na:

```
scx_bpf_dsq_move_to_local(u64 dsq_id)
```

by znaleźć to trzeba zajżeć do dokumentacji dokładnej wersji i zobaczyć jak zmieniła się nazwa funckji: https://github.com/torvalds/linux/blob/v6.14/kernel/sched/ext.c#L6749. Trzeba mieć na uwadzę, że jądro linuxa jest dynamicznie rozwijane i dokumentacja nie nadąża lub jest nieaktualna.

2.3 Output eBPF

Oto typowy output błędu przy kompilacji za pomocą ebpf:

```
libbpf: prog 'sched_running': BPF program load failed: -EACCES
libbpf: prog 'sched_running': -- BEGIN PROG LOAD LOG --
0: R1=ctx() R10=fp0
; int BPF_STRUCT_OPS(sched_running, struct task_struct *p) { @ selective.bpf.c:158
0: (79) r7 = *(u64 *)(r1 +0)
func 'running' arg0 has btf_id 86 type STRUCT 'task_struct'
1: R1=ctx() R7_w=trusted_ptr_task_struct()
; task_stats_ext *stats = bpf_task_storage_get(&task_storage, p, NULL, @
selective.bpf.c:160
1: (18) r1 = 0xffff8e975abf8400       ; R1_w=map_ptr(map=task_storage,ks=4,vs=88)
3: (bf) r2 = r7                       ; R2_w=trusted_ptr_task_struct()
R7_w=trusted_ptr_task_struct()
4: (b7) r3 = 0                        ; R3_w=0
5: (b7) r4 = 1                        ; R4_w=1
6: (85) call bpf_task_storage_get#156         ;
R0_w=map_value_or_null(id=1,map=task_storage,ks=4,vs=88)
7: (bf) r6 = r0                       ;
R0_w=map_value_or_null(id=1,map=task_storage,ks=4,vs=88)
R6_w=map_value_or_null(id=1,map=task_storag
e,ks=4,vs=88)
; if (!stats) { @ selective.bpf.c:163
8: (15) if r6 == 0x0 goto pc+18       ; R6_w=map_value(map=task_storage,ks=4,vs=88)
9: (b7) r1 = 23                       ; R1_w=23
; p->on_cpu = 23; @ selective.bpf.c:166
10: (63) *(u32 *)(r7 +52) = r1
processed 10 insns (limit 1000000) max_states_per_insn 0 total_states 0 peak_states 0
mark_read 0
-- END PROG LOAD LOG --
libbpf: prog 'sched_running': failed to load: -EACCES
libbpf: failed to load object 'selective_bpf'
libbpf: failed to load BPF skeleton 'selective_bpf': -EACCES
Failed to load and verify BPF skeleton
```

Cieko jest za pierwszym razem zrozumiec jaki jest błąd. Jest to błąd który wystąpi po poprawnej kompilacji programu, więc nie jest to błąd składniowy. Błąd wynika z niezrozumienia jakiejś własności programów eBPF.

2.3.1 Jak sobie z tym poradzić?

Sprawdzenie dokumentacji eBPF: [link].

2.4 Pomyłki w przykładach?

W przykładach pokazane jest zrobienie wspólnej kolejki na procesy (*schared_dispatch_queue*) z podaną linijką:

```
u64 slice = 5000000u / scx_bpf_dsq_nr_queued(SHARED_DSQ_ID);
scx_bpf_dispatch(p, SHARED_DSQ_ID, slice, enq_flags);
```

scx_bpf_dsq_nr_queued, zwraca liczbę procesów aktualnie znajdujących się w kolejce do procesorów. Problemem jest to, że kolejka może być pusta, co powodowałoby dzielenie przez 0. Dzielenie przez 0 w programach eBPF nie powoduje wyrzucenie błędu, lecz poprostu wynikiem

jest 0. Powoduje to, że procja czasu dostarczona do procesu wynosi 0. Nie oznacza to, że proces nie dostanie czasu procesora, a poprostu zostanie wywłaszczony przy najbliższej mozliwej okacji. Ten błąd jednak nie powoduje rzadnych problemów, w trakcie trwania planisty, ale jest to raczej nie zamierzane. Danie bardzo małego kawałku czasu jest nie efektywne, ale nie powoduje utraty responsywności komputera, dopóki liczba proecesów wymagająca stale czasu procesora nie jest dużo większa od liczby rdzeni precesora.

Znalazłem to przez przypadek realizując mój projekt. Wypisując dane, które przypisuje scheduler zauważyłem że slice wynosi 0. Aktualnie nie wiem czy to jest błąd czy slice 0 ma specjalne własności. Ale trzeba wiedzieć że dokumentacja, czy przykłady mogą mieć błędy, lub być nie optymalne.

2.5 Skąd wziąć nazwy funkcji?

Jest dokumentacja dotycząca funkcji eBPF od tym linkiem: https://docs.ebpf.io/linux/program-type/BPF_PROG_TYPE_STRUCT_OPS/sched_ext_ops/. Nie jest zbiór funkcji jądra, jednak nie mam tutaj najważniejszych funkcji, które faktycznie umożliwają komunkację z jądrem w celu przydzielenia czasu procesora. By się tego dowiedzieć trzeba zobaczyć faktyczną implementację: https://github.com/torvalds/linux/blob/v6.14/kernel/sched/ext.c. Funkcje są tam dobrze opisane i faktycznie tłumaczą jakie mają zastosowanie

3 Lists and Points

Typst makes lists very intuitive. You do not need complex commands, just symbols.

3.1 Bullet Points

Use a hyphen - for bullets. Indent for nested lists.

- This is the first point.
- This is the second point.
  - This is a sub-point (indented by 2 spaces).
  - Another sub-point.
- Back to the main level.

3.2 Numbered Lists

Use a plus sign + for automatic numbering.

1. First step of the process.
2. Second step.
3. Third step.
   1. You can nest numbered lists too.
   2. It handles the hierarchy automatically (1, 2, a, b, etc.).

4 Math Arrays and Matrices

We use the `mat` function inside math mode (`$ ... $`). You use commas `,` to separate columns and semi-colons `;` to separate rows.

4.1 Standard Matrix

A standard 2x2 matrix:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

4.2 Vectors and Delimiters

You can change the delimiters (brackets, bars, etc.) using `delim:`.

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

A determinant (using bars):

$$\det(M) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

5 Figures and Images

In academic reports, images should be wrapped in a `#figure` so they can have a caption and be referenced later.

5.1 Inserting an Image

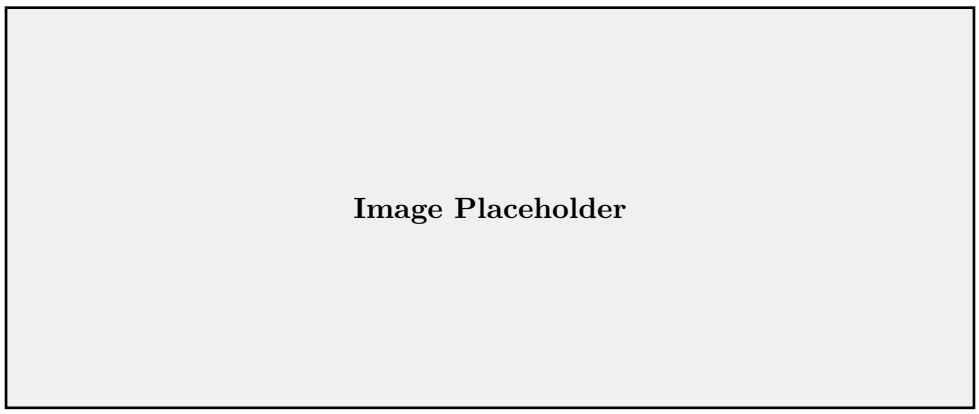

Figure 1: This is a figure caption. Note how it is numbered automatically.

As seen in Figure 1, the placeholder represents where your data visualization would go. Using the `@label` syntax lets you cross-reference figures easily.

6 Complex Equation Example

Here is a complex equation combining arrays and sums:

$$\sigma^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2$$

And a system of equations using a case block:

$$f(x) = \begin{cases} x^2 \text{ if } x > 0 \\ 0 \text{ otherwise} \end{cases}$$

ID	Experiment Description	Result (Unit)
001	Initial baseline test under standard conditions	0.45
002	Stress test with variable load	1.20
003	Thermal expansion measurement	0.99
004	Vibration analysis at 50Hz	0.05
005	Final structural integrity check	Pass

Table 1: Summary of Experimental Results

Metric	Observation	Status
Alpha	Initial read ok	Pass
Beta	Slight variance	Check
Gamma	Heat nominal	Pass
Delta	Pressure high	Warn
Epsilon	Stable	Pass

Col A	Col B	Col C
Data 1	Data 2	Data 3
Data 4	Data 5	Data 6
Data 7	Data 8	Data 9
Data 10	Data 11	Data 12
Data 13	Data 14	Data 15

ID	Description	Value
001	System check complete	98%
002	Loading sequence	45%
003	Waiting for input	0%
004	Processing data	12%
005	Exporting results	100%