CPUFreq_ext： a BPF based cpufreq governor

邹贻鹏

Yipeng Zou <zouyipeng@huawei.com>

目录

CONTENT

01 背景&动机

02 可编程调频器框架

03 技术方案

04 示例展示

1

Part One

背景&动机

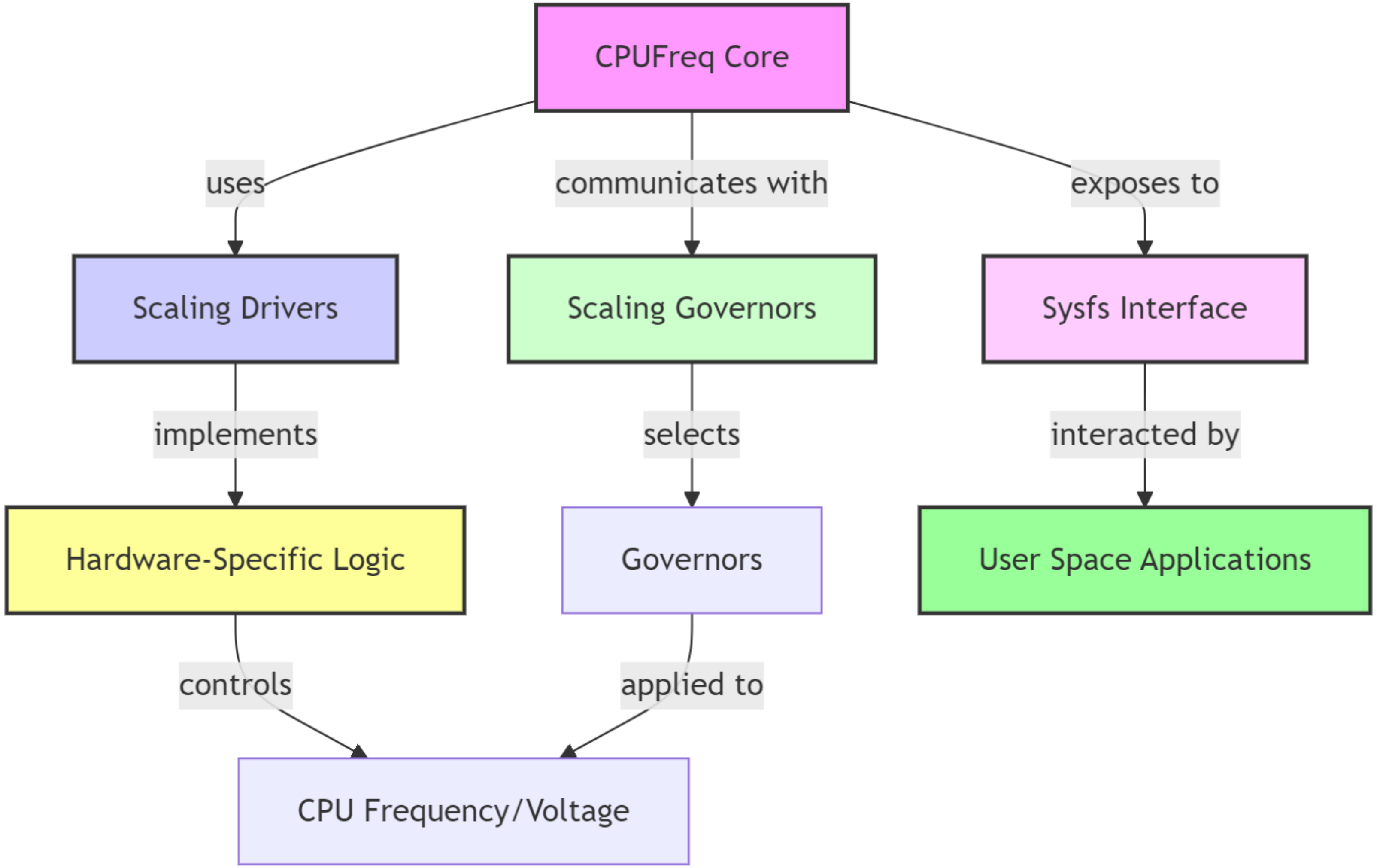
背景&动机- CPUFreq子系统

- Linux kernel中CPUFreq子系统负责动态调整CPU频率和电压
  > DVFS
  > 主要包括三部分：核心层（Core）、调频器（Governors）和驱动程序（Drivers）
  > 核心层提供了通用的代码基础设施和用户空间接口
  > 调频器实现了不同的算法来估计所需的CPU容量
  > 驱动程序则与硬件直接交互，并访问平台特定的硬件接口以改变CPU状态

- CPUFreq Governor：调频策略
  > Ondemand：在系统负载高时允许CPU工作在最大频率，在系统空闲时则降低频率
  > Conservative：类似于Ondemand，但是它在调整频率时更加平滑
  > Schedutil：通过注册到调度器的hook来响应负载变化，追踪CPU负载（PELT）进行调频
  > Userspace：允许用户程序直接设置CPU频率
  > Performance
  > Powersave

- CPUFreq Policy：调频对象
  > 可以包含若干个硬件可调频的物理CPU
  > 可以独立配置governor
  > 平台相关：调频域、调压域

背景&动机- CPUFreq子系统


CPUFreq Core

uses

communicates with

exposes to

Scaling Drivers

Scaling Governors

Sysfs Interface

implements

selects

interacted by

Hardware-Specific Logic

Governors

User Space Applications

controls

applied to

CPU Frequency/Voltage

背景&动机

- 当前在kernel中的governor都是针对通用场景的调频策略

  > conservative ondemand ：切换延时

  > userspace：提供sysfs接口供用户直接配置想要的频率，更多时候需要更多的内核信息来计算符合需求的频率

  > Schedutil：CPU Util有时无法准确体现业务特征

  > 需要定制化的调频策略：匹配特定业务的特征

- 特定场景（负载、能效比）定制化实现调频器

  > 保障关键进程QoS

  > 系统全局功耗调优：调度策略与调频策略对性能、功耗的影响巨大

- sched_ext：基于BPF的可编程内核调度类：可通过 BPF 程序来实现和调整调度策略
- cpufreq ext : 基于 BPF 的 cpufreq 调频器，可以在 BPF 程序中定制 cpufreq governor

- 通过 BPF提供更轻量级和灵活的方法来实现调频策略

  > 与可编程模块协同工作: sched_ext

  > BPF 三方工具拓展: bpftools

2

Part Two

可编程调频器框架

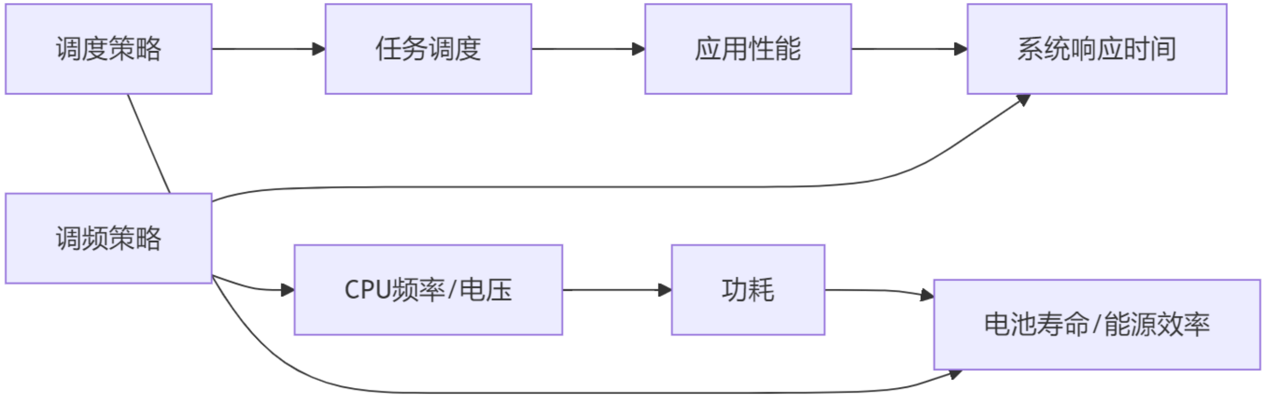
CPUFreq_ext

- 基于BPF可编程的CPU调频框架

- CPUFreq_ext旨在通过提供一个可定制的框架，该框架可以根据不同系统和应用的独特需求进行调整

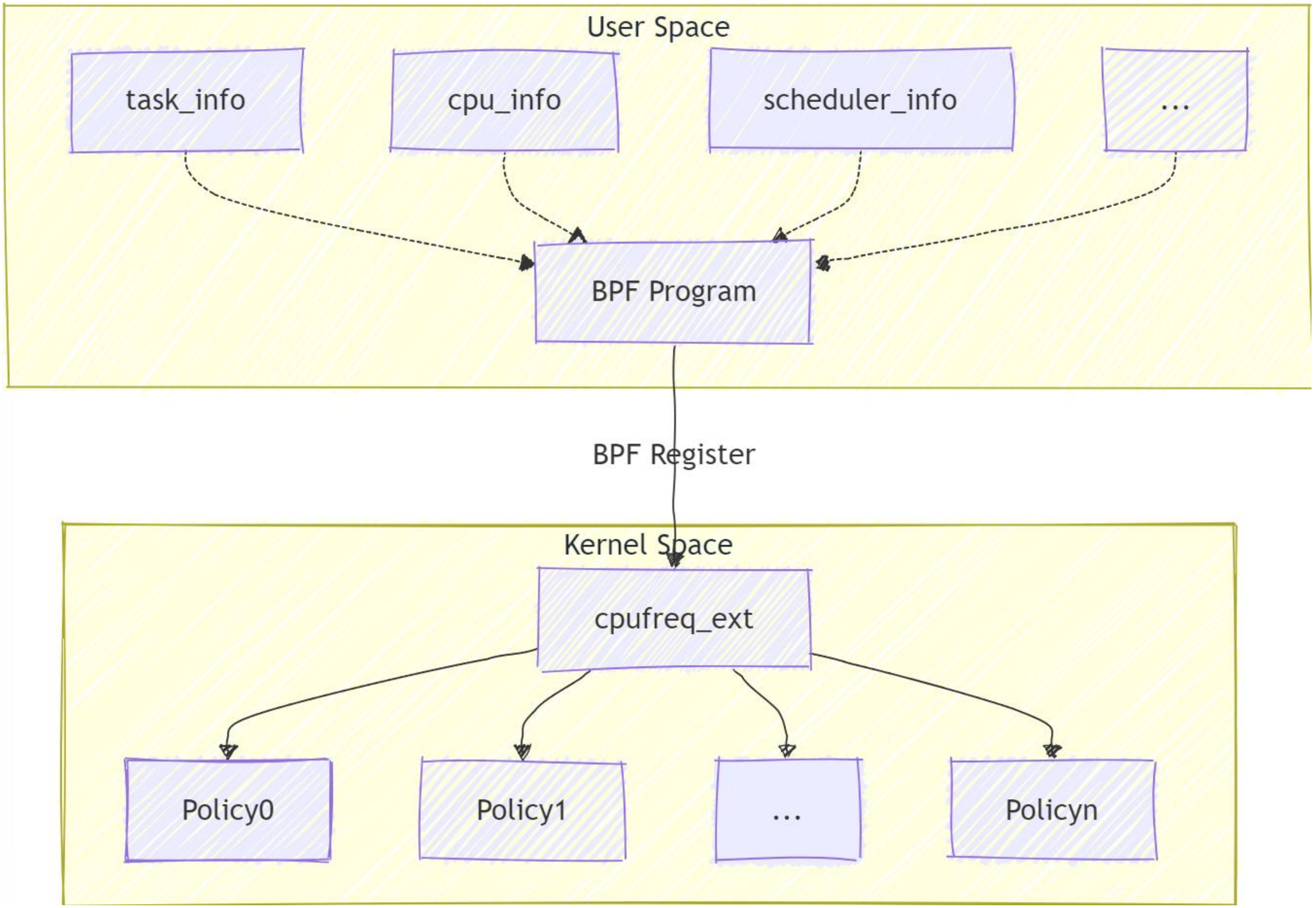
- 结合BPF 生态提供更轻量级和灵活的方法来实现调频策略

- 与sched_ext互动
  > 在BPF程序中同时定制实现CPU调度器和调频器
  > 统筹调度策略和调频策略



CPUFreq_ext

- cpufreq ext 是一种基于 BPF 的 cpufreq 调频器，我们可以在 BPF 程序中自定义实现调频器。
- 与现在CPUFreq子系统兼容，内核只新增一个governor实现
- 基于BPF Struct OPS
  > struct cpufreq_governor_ext_ops
  > 提供函数回调供BPF程序注册
- 使用BPF helper、BPF kfuncs可以在BPF governor中获取内核的各种信息



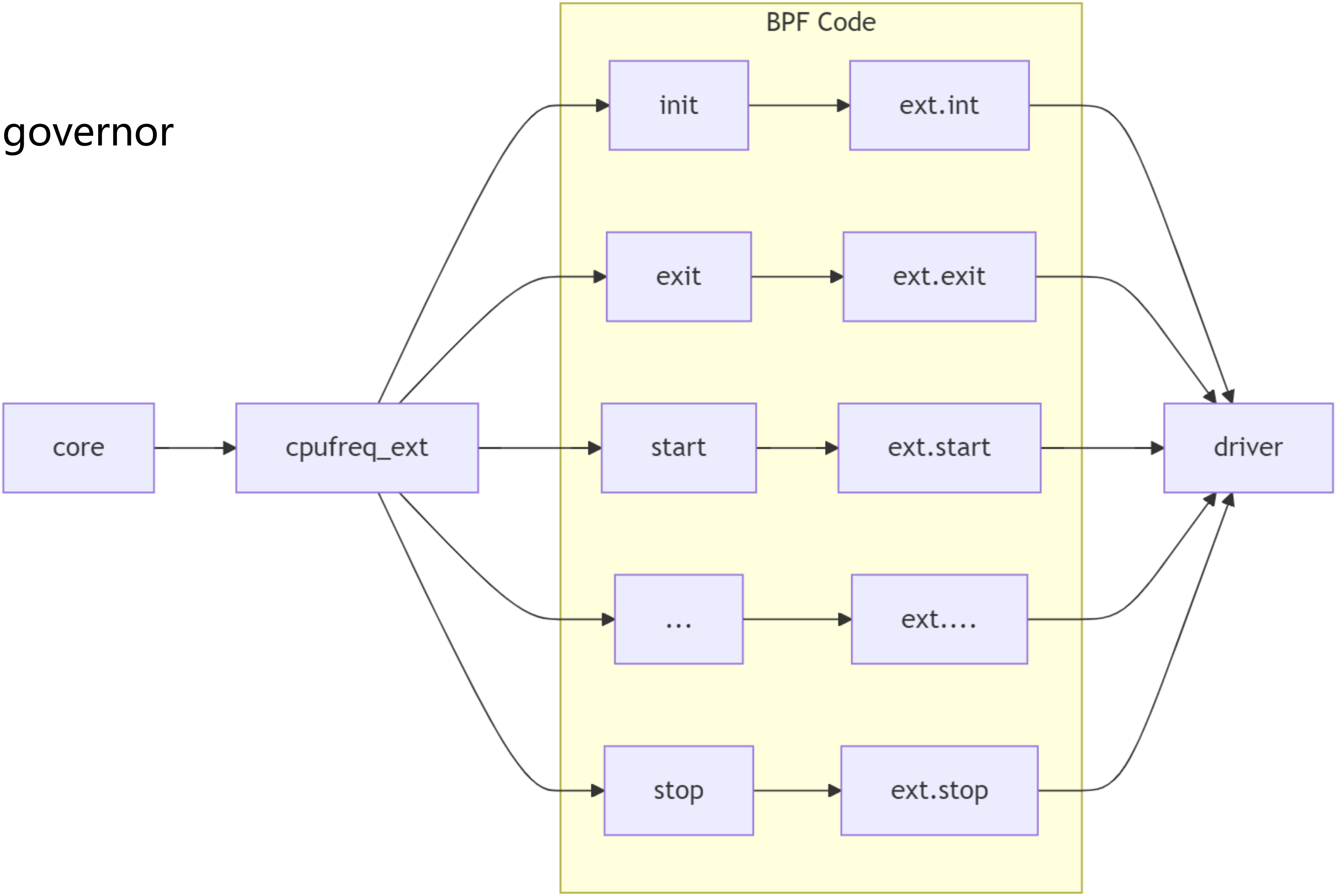
3

Part Three

技术方案

实现

- 基于CPUFreq框架实现一个通用的governor
  > cpufreq_ext
  > BPF实现的函数回调
  > 注册在cpufreq core层接口中



实现

- 在调度器调频点

  > enqueue_entity

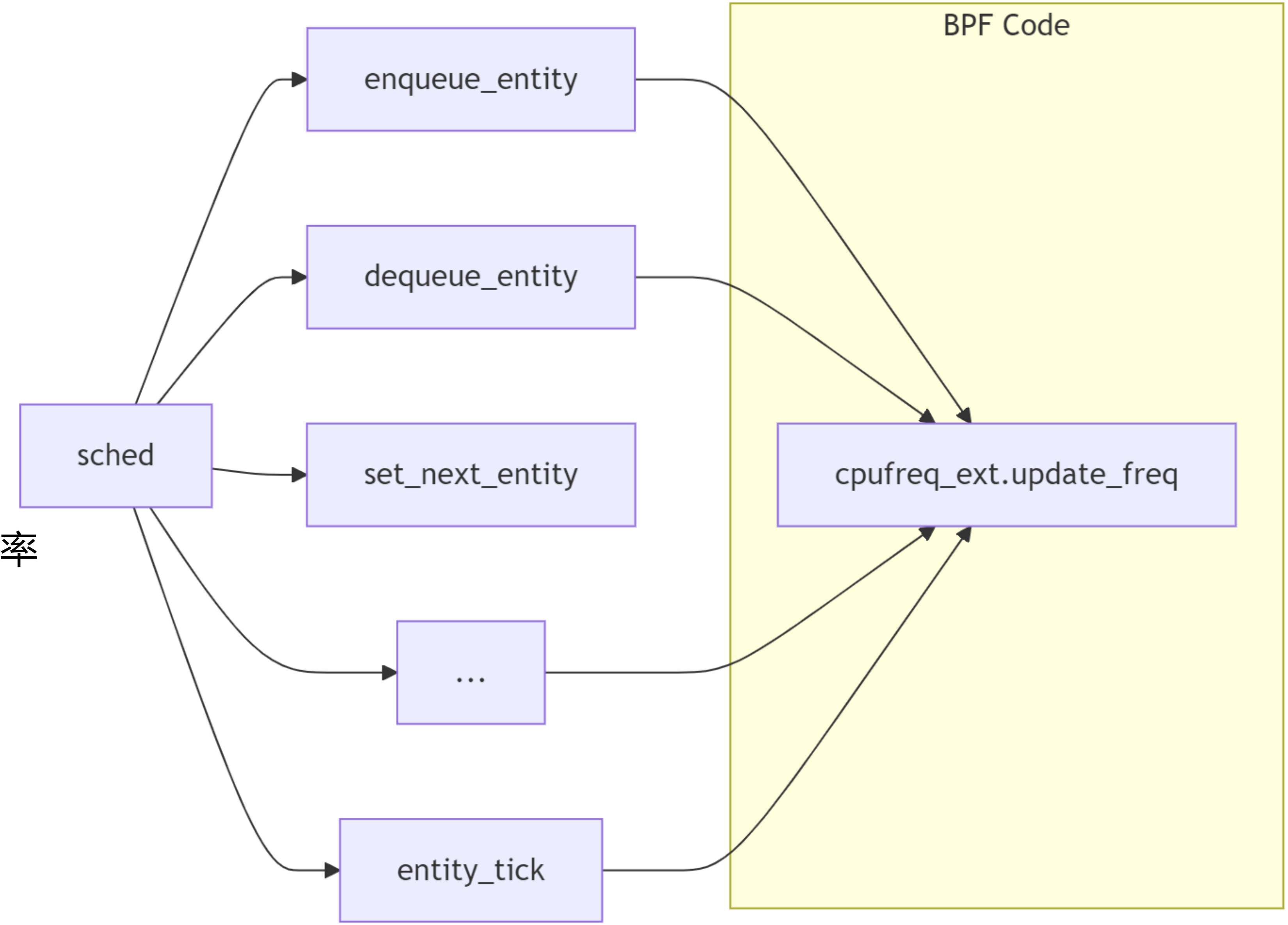
  > dequeue_entity

  > set_next_entity

  > entity_tick

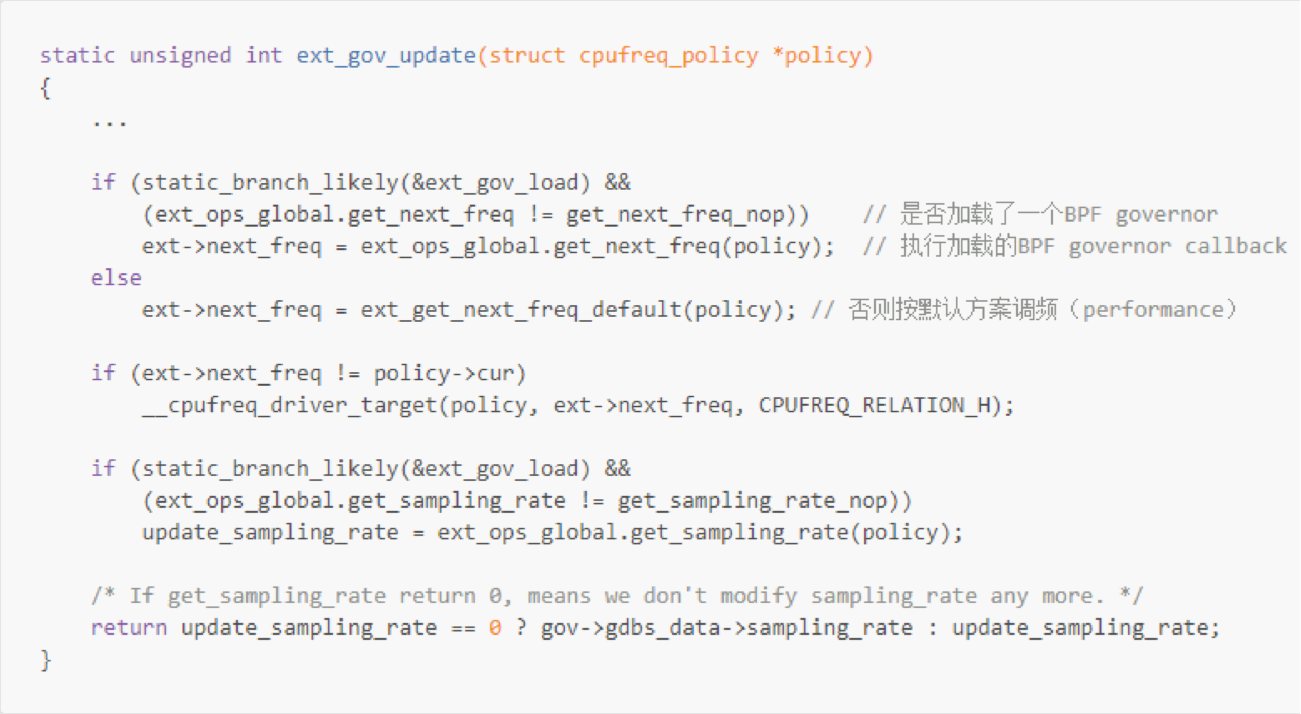
  > ...

- 执行hook获取需要设置的CPU频率

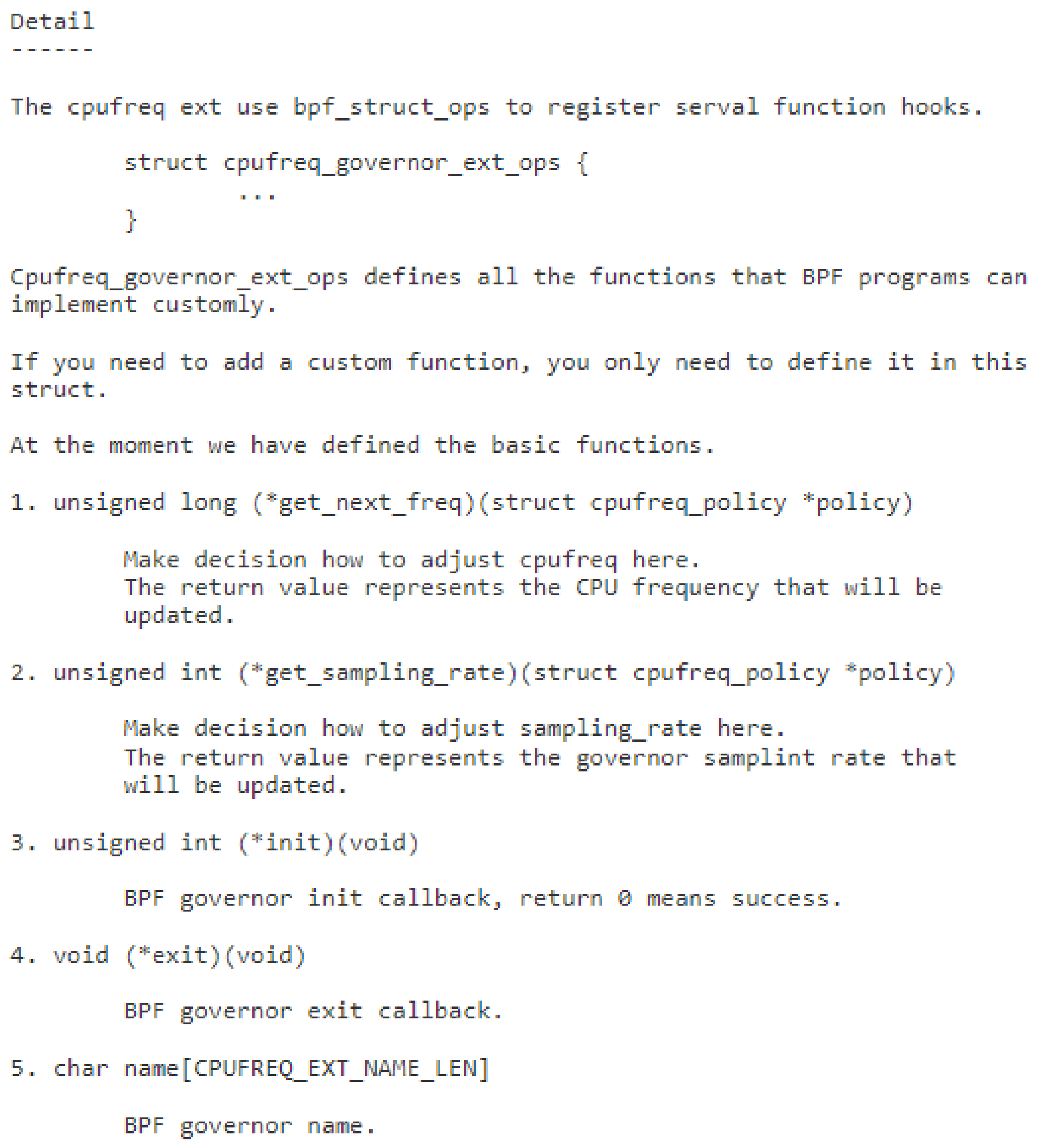


实现

- 加载一个BPF governor，注册对应的函数到cpufreq_ext中
- 在调频点调用注册的调频策略函数
- 没有调频器注册时，默认按照performance策略



```
static unsigned int ext_gov_update(struct cpufreq_policy *policy)
{
    ...

    if (static_branch_likely(&ext_gov_load) &&
        (ext_ops_global.get_next_freq != get_next_freq_nop))    // 是否加载了一个BPF governor
        ext->next_freq = ext_ops_global.get_next_freq(policy);  // 执行加载的BPF governor callback
    else
        ext->next_freq = ext_get_next_freq_default(policy); // 否则按默认方案调频（performance）

    if (ext->next_freq != policy->cur)
        __cpufreq_driver_target(policy, ext->next_freq, CPUFREQ_RELATION_H);

    if (static_branch_likely(&ext_gov_load) &&
        (ext_ops_global.get_sampling_rate != get_sampling_rate_nop))
        update_sampling_rate = ext_ops_global.get_sampling_rate(policy);

    /* If get_sampling_rate return 0, means we don't modify sampling_rate any more. */
    return update_sampling_rate == 0 ? gov->gdbs_data->sampling_rate : update_sampling_rate;
}
```



```
Detail
------

The cpufreq ext use bpf_struct_ops to register serval function hooks.

        struct cpufreq_governor_ext_ops {
                ...
        }

Cpufreq_governor_ext_ops defines all the functions that BPF programs can
implement customly.

If you need to add a custom function, you only need to define it in this
struct.

At the moment we have defined the basic functions.

1. unsigned long (*get_next_freq)(struct cpufreq_policy *policy)

        Make decision how to adjust cpufreq here.
        The return value represents the CPU frequency that will be
        updated.

2. unsigned int (*get_sampling_rate)(struct cpufreq_policy *policy)

        Make decision how to adjust sampling_rate here.
        The return value represents the governor samplint rate that
        will be updated.

3. unsigned int (*init)(void)

        BPF governor init callback, return 0 means success.

4. void (*exit)(void)

        BPF governor exit callback.

5. char name[CPUFREQ_EXT_NAME_LEN]

        BPF governor name.
```

4

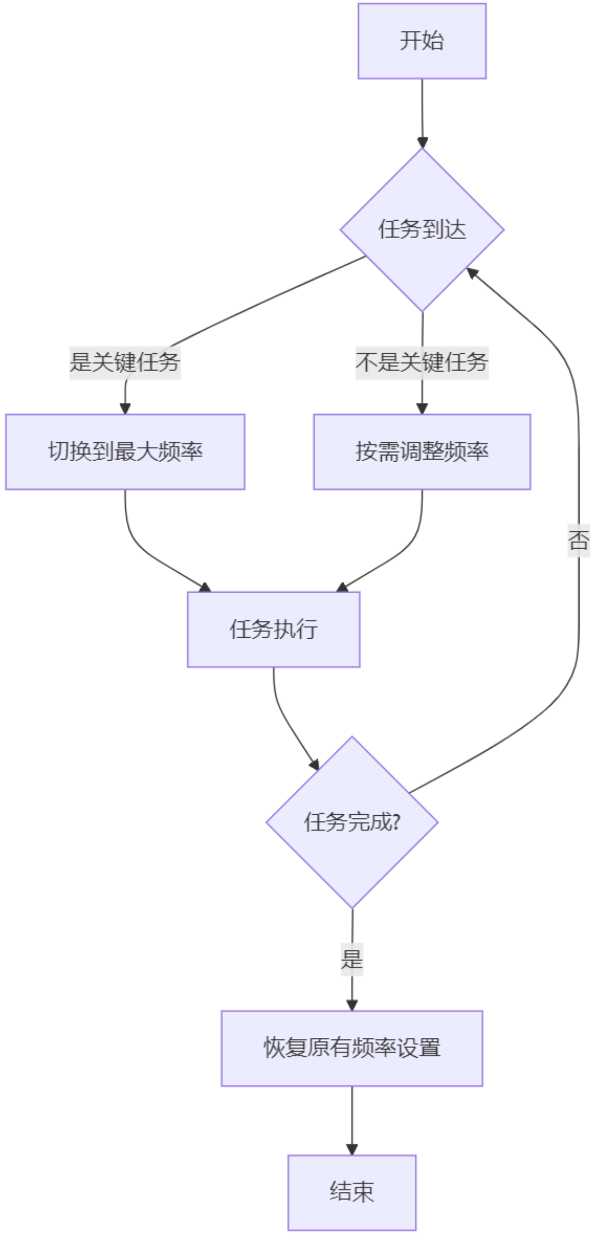
Part Three

示例展示

示例

- [RFC,2/2] cpufreq_ext: Add bpf sample

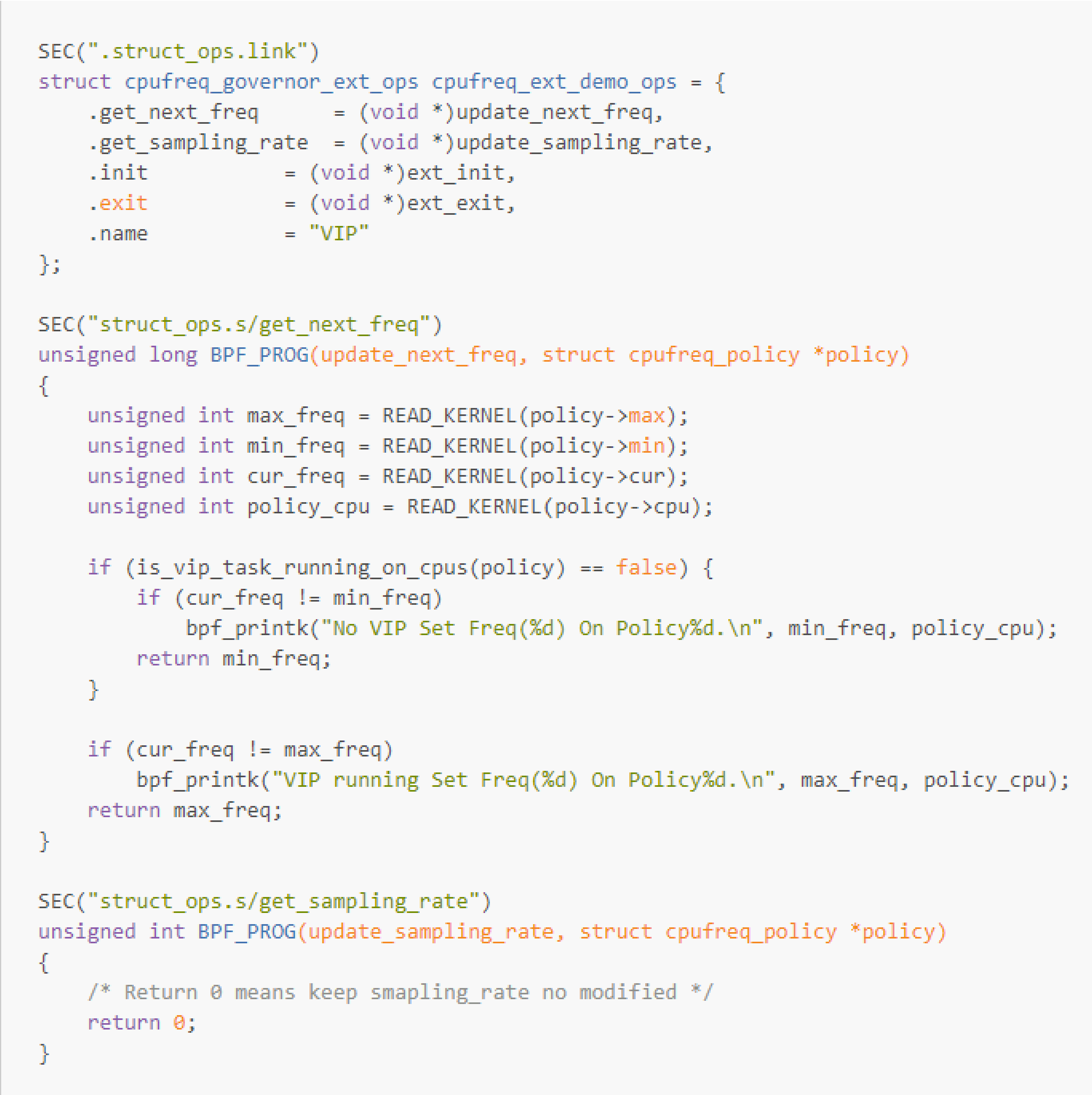
- 实现一个基于VIP任务的BPF调频器：
  > 在目标任务运行在目标CPU上时，提高频率
  > 任务完成后，恢复CPU频率



示例

- 在BPF程序中实现struct_ops {…}

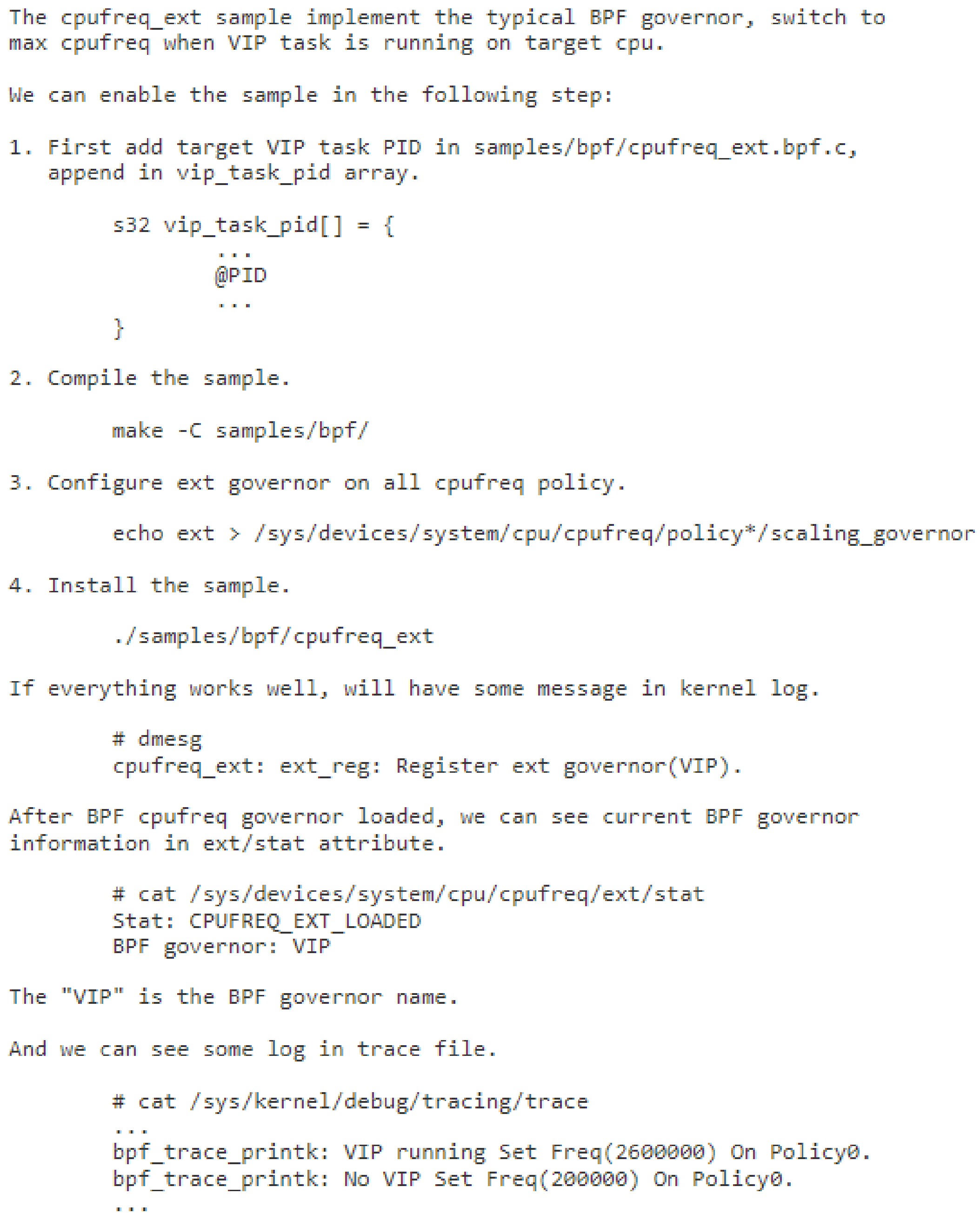
- 在BPF程序中实现调频策略



```
SEC(".struct_ops.link")
struct cpufreq_governor_ext_ops cpufreq_ext_demo_ops = {
    .get_next_freq      = (void *)update_next_freq,
    .get_sampling_rate  = (void *)update_sampling_rate,
    .init           = (void *)ext_init,
    .exit           = (void *)ext_exit,
    .name           = "VIP"
};

SEC("struct_ops.s/get_next_freq")
unsigned long BPF_PROG(update_next_freq, struct cpufreq_policy *policy)
{
    unsigned int max_freq = READ_KERNEL(policy->max);
    unsigned int min_freq = READ_KERNEL(policy->min);
    unsigned int cur_freq = READ_KERNEL(policy->cur);
    unsigned int policy_cpu = READ_KERNEL(policy->cpu);

    if (is_vip_task_running_on_cpus(policy) == false) {
        if (cur_freq != min_freq)
            bpf_printk("No VIP Set Freq(%d) On Policy%d.\n", min_freq, policy_cpu);
        return min_freq;
    }

    if (cur_freq != max_freq)
        bpf_printk("VIP running Set Freq(%d) On Policy%d.\n", max_freq, policy_cpu);
    return max_freq;
}

SEC("struct_ops.s/get_sampling_rate")
unsigned int BPF_PROG(update_sampling_rate, struct cpufreq_policy *policy)
{
    /* Return 0 means keep smapling_rate no modified */
    return 0;
}
```

示例

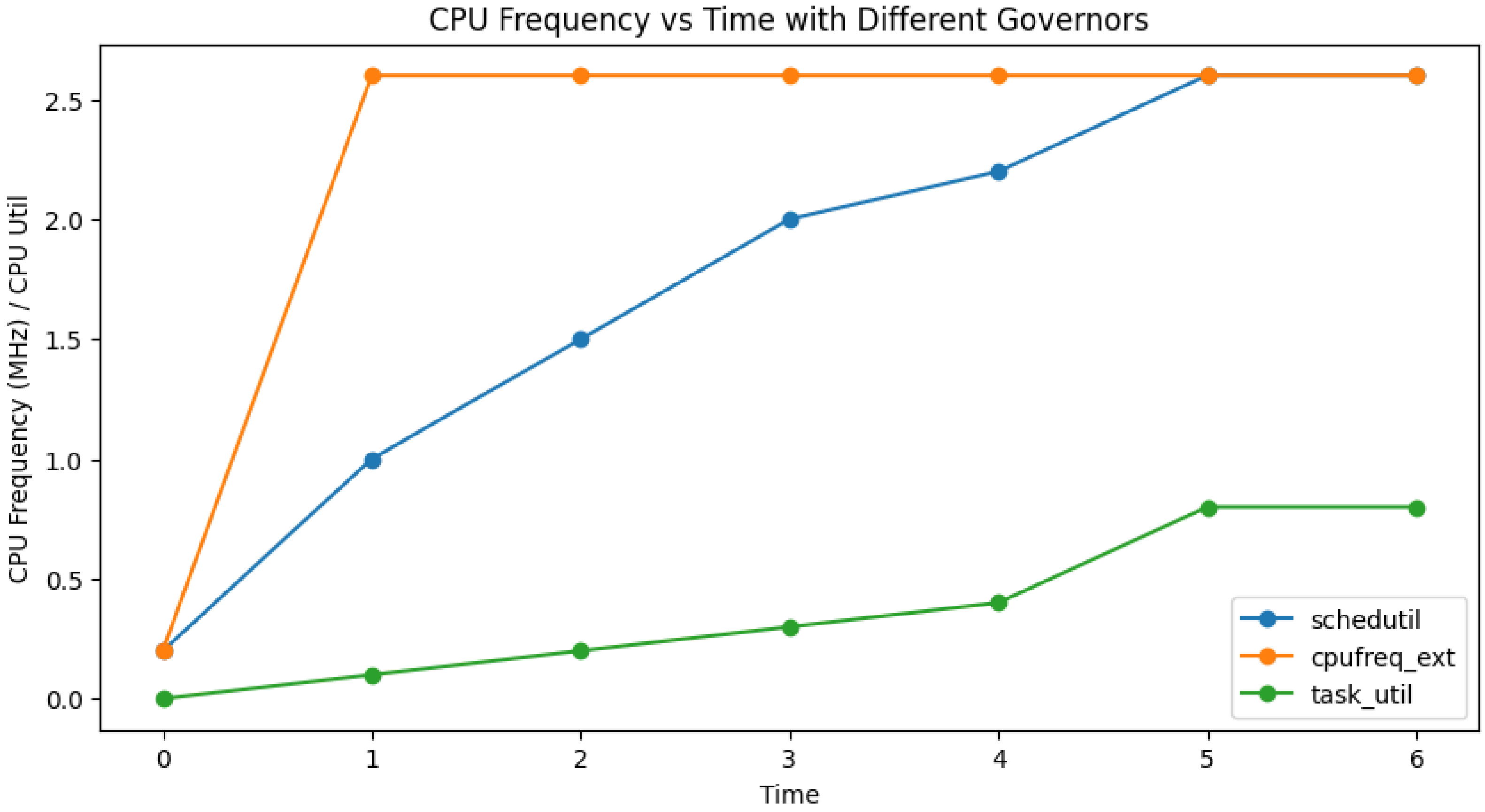
- Step1: 配置cpufreq_ext
- Step2: 加载BPF governor
- Step3: 观察频率变化



```
The cpufreq_ext sample implement the typical BPF governor, switch to
max cpufreq when VIP task is running on target cpu.

We can enable the sample in the following step:

1. First add target VIP task PID in samples/bpf/cpufreq_ext.bpf.c,
   append in vip_task_pid array.

        s32 vip_task_pid[] = {
                ...
                @PID
                ...
        }

2. Compile the sample.

        make -C samples/bpf/

3. Configure ext governor on all cpufreq policy.

        echo ext > /sys/devices/system/cpu/cpufreq/policy*/scaling_governor

4. Install the sample.

        ./samples/bpf/cpufreq_ext

If everything works well, will have some message in kernel log.

        # dmesg
        cpufreq_ext: ext_reg: Register ext governor(VIP).

After BPF cpufreq governor loaded, we can see current BPF governor
information in ext/stat attribute.

        # cat /sys/devices/system/cpu/cpufreq/ext/stat
        Stat: CPUFREQ_EXT_LOADED
        BPF governor: VIP

The "VIP" is the BPF governor name.

And we can see some log in trace file.

        # cat /sys/kernel/debug/tracing/trace
        ...
        bpf_trace_printk: VIP running Set Freq(2600000) On Policy0.
        bpf_trace_printk: No VIP Set Freq(200000) On Policy0.
        ...
```

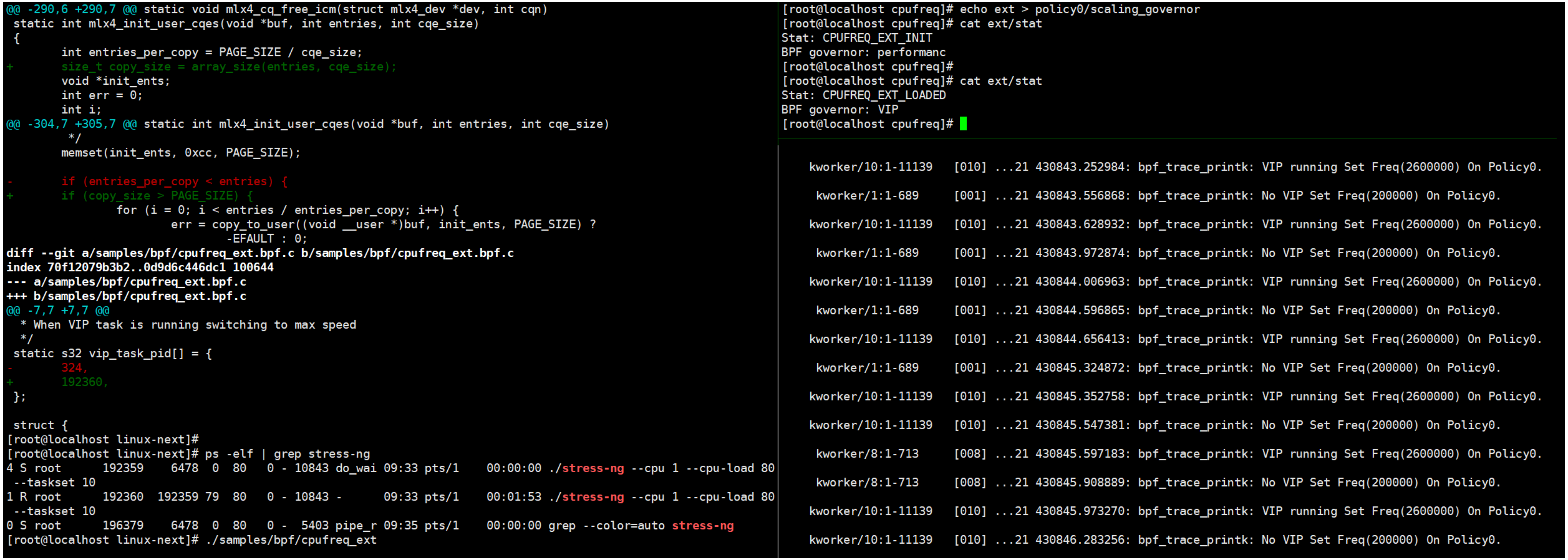
示例

- Sample在task enqueue时识别关键任务进行调频相比较schedutil调频器，可以快速响应关键任务的性能需求



示例

- 加载BPF governor后执行效果如图
  > 在bpf trace打印当前VIP任务执行、调频情况



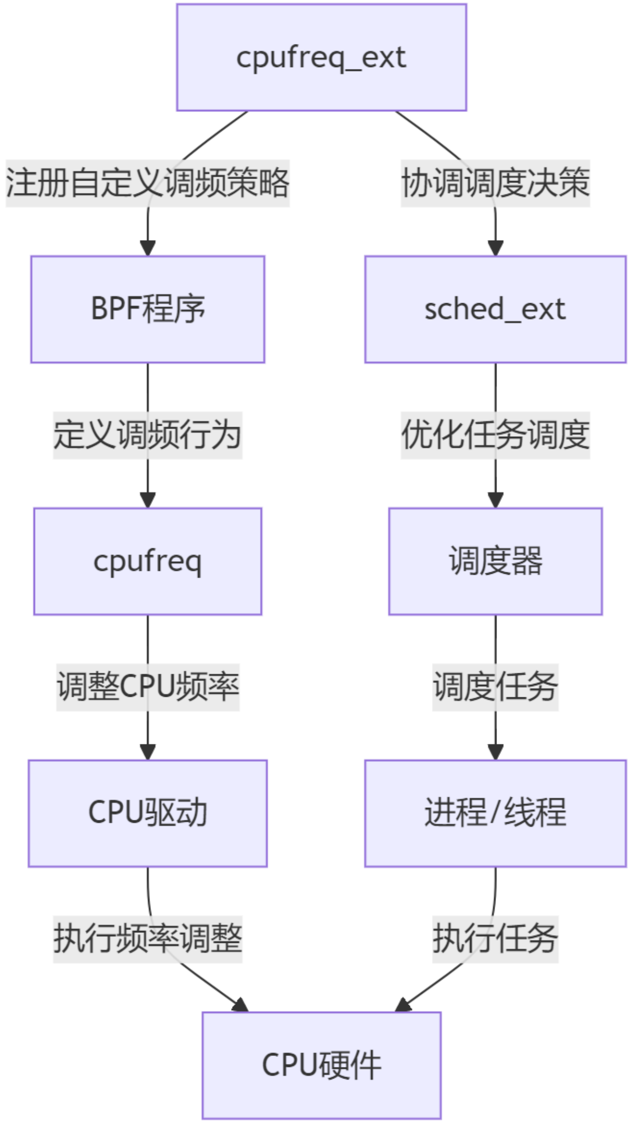
```
@@ -290,6 +290,7 @@ static void mlx4_cq_free_icm(struct mlx4_dev *dev, int cqn)
 static int mlx4_init_user_cqes(void *buf, int entries, int cqe_size)
 {
        int entries_per_copy = PAGE_SIZE / cqe_size;
+       size_t copy_size = array_size(entries, cqe_size);
        void *init_ents;
        int err = 0;
        int i;
@@ -304,7 +305,7 @@ static int mlx4_init_user_cqes(void *buf, int entries, int cqe_size)
         */
        memset(init_ents, 0xcc, PAGE_SIZE);

-       if (entries_per_copy < entries) {
+       if (copy_size > PAGE_SIZE) {
                for (i = 0; i < entries / entries_per_copy; i++) {
                        err = copy_to_user((void __user *)buf, init_ents, PAGE_SIZE) ?
                                -EFAULT : 0;
diff --git a/samples/bpf/cpufreq_ext.bpf.c b/samples/bpf/cpufreq_ext.bpf.c
index 70f12079b3b2..0d9d6c446dc1 100644
--- a/samples/bpf/cpufreq_ext.bpf.c
+++ b/samples/bpf/cpufreq_ext.bpf.c
@@ -7,7 +7,7 @@
   * When VIP task is running switching to max speed
   */
 static s32 vip_task_pid[] = {
-       324,
+       192360,
 };

 struct {
[root@localhost linux-next]#
[root@localhost linux-next]# ps -elf | grep stress-ng
4 S root      192359    6478  0  80   0 - 10843 do_wai 09:33 pts/1    00:00:00 ./stress-ng --cpu 1 --cpu-load 80
 --taskset 10
1 R root      192360  192359 79  80   0 - 10843 -      09:33 pts/1    00:01:53 ./stress-ng --cpu 1 --cpu-load 80
 --taskset 10
0 S root      196379    6478  0  80   0 -  5403 pipe_r 09:35 pts/1    00:00:00 grep --color=auto stress-ng
[root@localhost linux-next]# ./samples/bpf/cpufreq_ext
```

```
[root@localhost cpufreq]# echo ext > policy0/scaling_governor
[root@localhost cpufreq]# cat ext/stat
Stat: CPUFREQ_EXT_INIT
BPF governor: performanc
[root@localhost cpufreq]#
[root@localhost cpufreq]# cat ext/stat
Stat: CPUFREQ_EXT_LOADED
BPF governor: VIP
[root@localhost cpufreq]#
```

```
    kworker/10:1-11139   [010] ...21 430843.252984: bpf_trace_printk: VIP running Set Freq(2600000) On Policy0.

     kworker/1:1-689     [001] ...21 430843.556868: bpf_trace_printk: No VIP Set Freq(200000) On Policy0.

    kworker/10:1-11139   [010] ...21 430843.628932: bpf_trace_printk: VIP running Set Freq(2600000) On Policy0.

     kworker/1:1-689     [001] ...21 430843.972874: bpf_trace_printk: No VIP Set Freq(200000) On Policy0.

    kworker/10:1-11139   [010] ...21 430844.006963: bpf_trace_printk: VIP running Set Freq(2600000) On Policy0.

     kworker/1:1-689     [001] ...21 430844.596865: bpf_trace_printk: No VIP Set Freq(200000) On Policy0.

    kworker/10:1-11139   [010] ...21 430844.656413: bpf_trace_printk: VIP running Set Freq(2600000) On Policy0.

     kworker/1:1-689     [001] ...21 430845.324872: bpf_trace_printk: No VIP Set Freq(200000) On Policy0.

    kworker/10:1-11139   [010] ...21 430845.352758: bpf_trace_printk: VIP running Set Freq(2600000) On Policy0.

    kworker/10:1-11139   [010] ...21 430845.547381: bpf_trace_printk: No VIP Set Freq(200000) On Policy0.

     kworker/8:1-713     [008] ...21 430845.597183: bpf_trace_printk: VIP running Set Freq(2600000) On Policy0.

     kworker/8:1-713     [008] ...21 430845.908889: bpf_trace_printk: No VIP Set Freq(200000) On Policy0.

    kworker/10:1-11139   [010] ...21 430845.973270: bpf_trace_printk: VIP running Set Freq(2600000) On Policy0.

    kworker/10:1-11139   [010] ...21 430846.283256: bpf_trace_printk: No VIP Set Freq(200000) On Policy0.
```
]

TODO

- sched_ext协同优化全局性能、功耗
  > sched_ext中自定义调频点
- 完善CPUFreq_ext功能
- 收集意见、反馈



群聊：CLK_2024_华为技术分享群

该二维码7天内(10月28日前)有效，重新进入将更新

交流讨论、建议：

https://patchwork.kernel.org/project/linux-pm/cover/20240927101342.3240263-1-zouyipeng@huawei.com/