CHINA LINUX KERNEL
中国Linux内核开发者大会

vivo

基于eBPF支持自定义低功耗策略 cpuidle_ext框架

林义凯 （yikai.lin@vivo.com）

目录

CONTENTS

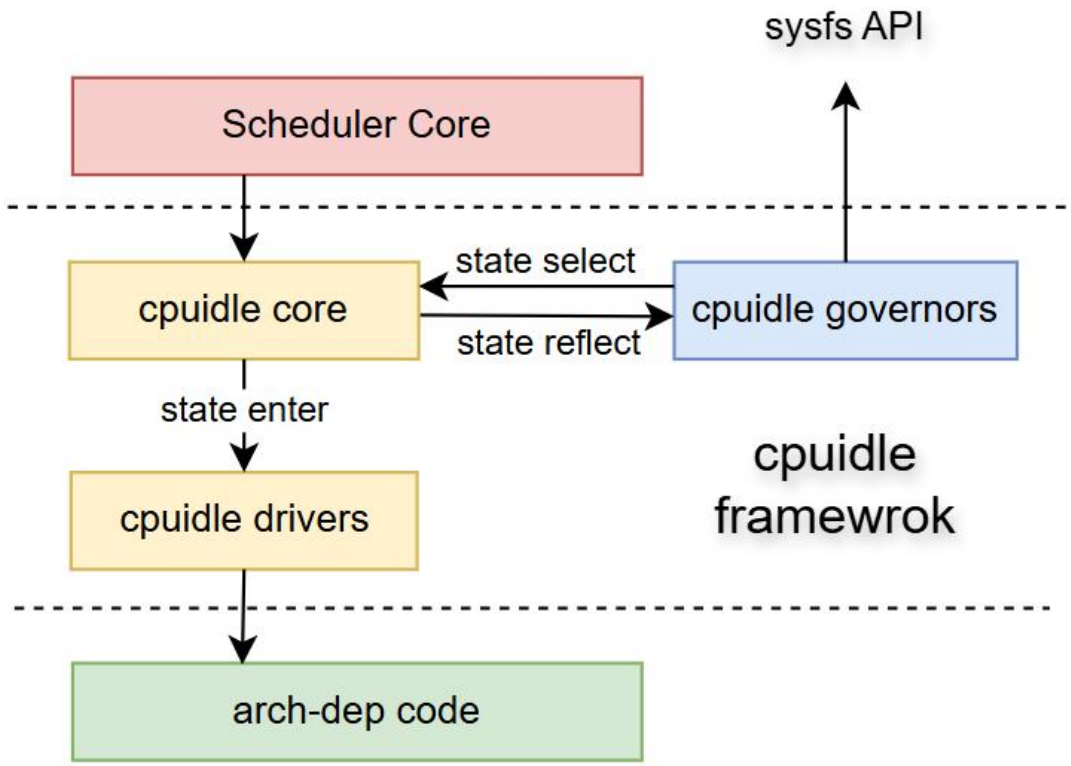
01 现状与挑战

➢ **cpuidle：控制cpu进入空闲状态，达到省电目的**

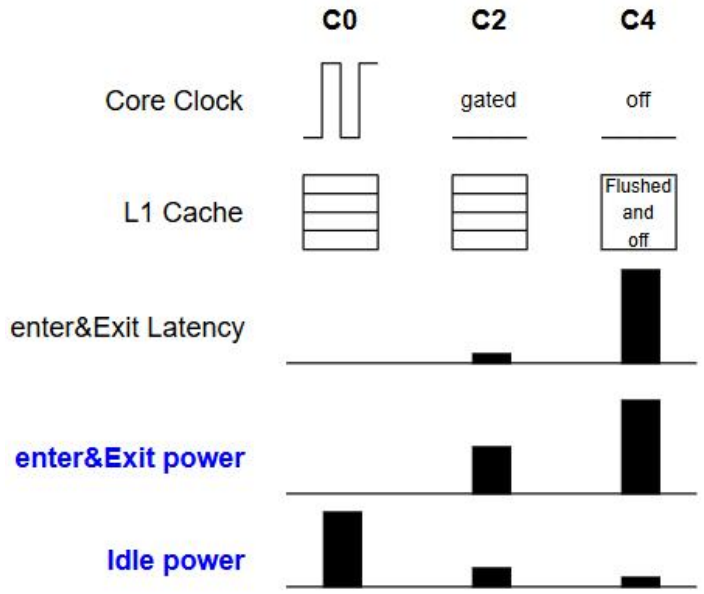
- Scheduler Core发现运行队列为空，启动空闲状态切换流程
- cpuidle governor：实现空闲状态选择算法
- cpuidle driver：执行硬件级状态切换动作（如时钟控制）

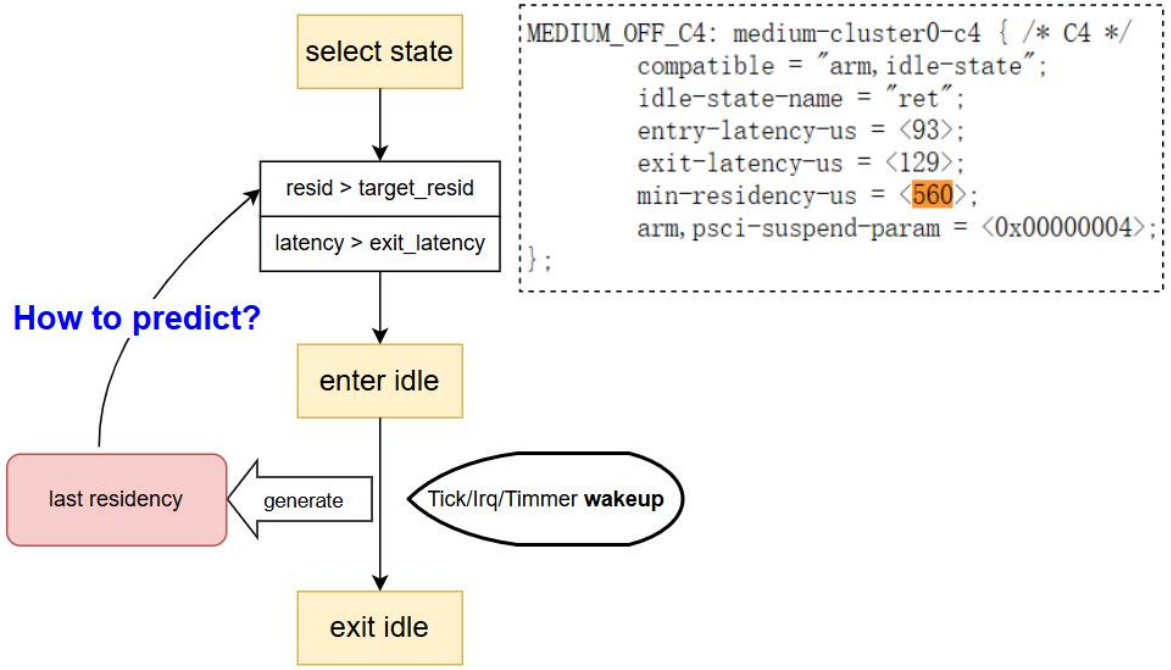
➢ **cpuidle governor**

- 在linux中包含多种策略，如：menu/teo等， 手机中也有自己的策略，如：qcom-lpm
- state_select：选择下一次要进入的idle状态
- state_reflect：接收上一次状态退出的信息，更新算法状态参数



01 现状与挑战





➢ **Idle state划分**

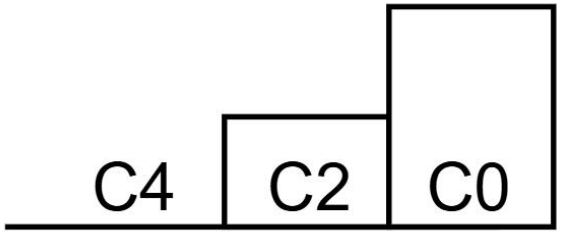
- CPU空闲状态（C-states）分为多个级别: C0~Cn，不同平台会有差异，但都通过控制clock/Ln cache等器件实现节能
- 虽然深层Idle状态更节能，但进入和退出本身需要耗能，因此始终选择深层idle并非最佳选择，需要均衡考虑

➢ **影响Idle state_select关键因素**

- entry(exit)-latency：进入/退出当前idle state的延迟
- min-residency: 最小停留时长，节省功耗 VS 进入/退出消耗功耗的临界值

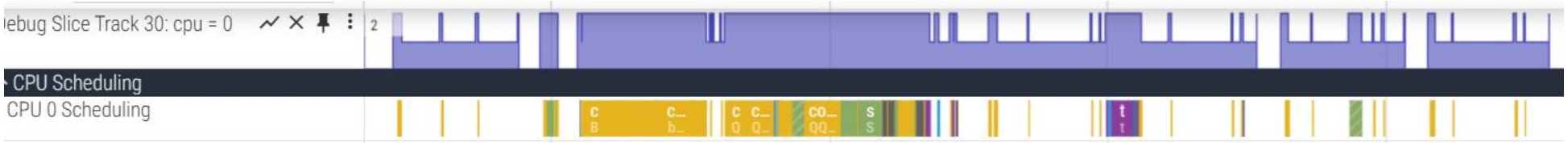
我们以状态跃迁图来表示state切换，如右图

- C4：深层idle，C2：浅层idle
- C0：cpu running



实际CPU trace表现如下图，task运行时处于C0状态，task没有运行时在C2/C4之间切换

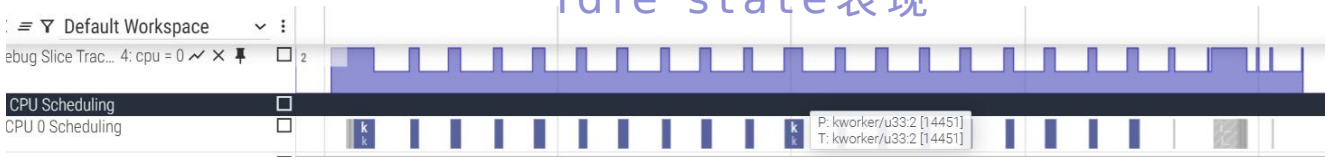
Idle state表现



task running表现

01 现状与挑战

细节放大后图示如下：



Idle state表现

task running表现

- 对于部分有规律的内核线程，采用常见的方法：取一组历史标准差小的residency的平均值，作为预测值，是一个好的选择

Idle state表现

task running表现

- 对于部分无规律应用进程，历史均值算法难免导致较大偏差

当前问题点：cpuidle算法实现目前位于kernel，无法感知上层应用表现，难以适配差异化能效场景

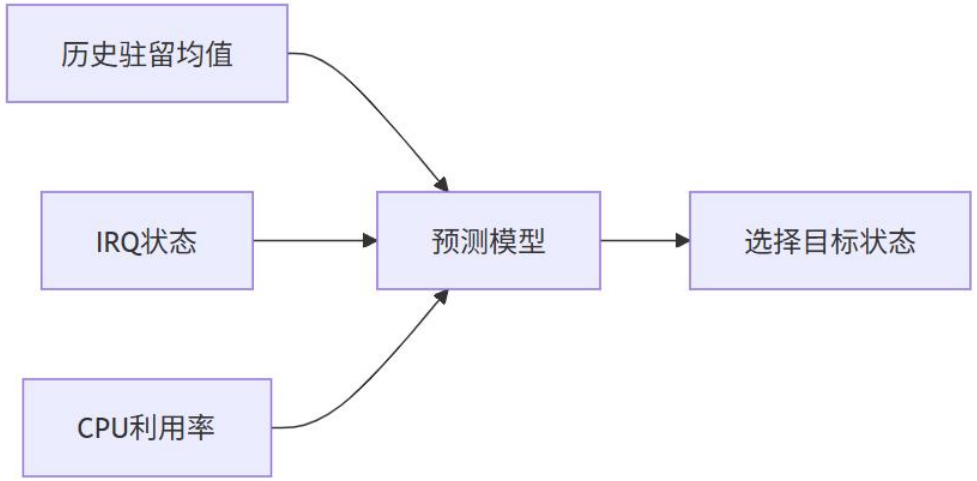
01 现状与挑战

➢ 当前linux中是针对通用场景的cpuidle策略：

- 预测维度局限：仅依赖cpu上历史idle均值+负载/中断（IRQ/CPU util），缺乏用户场景感知能力

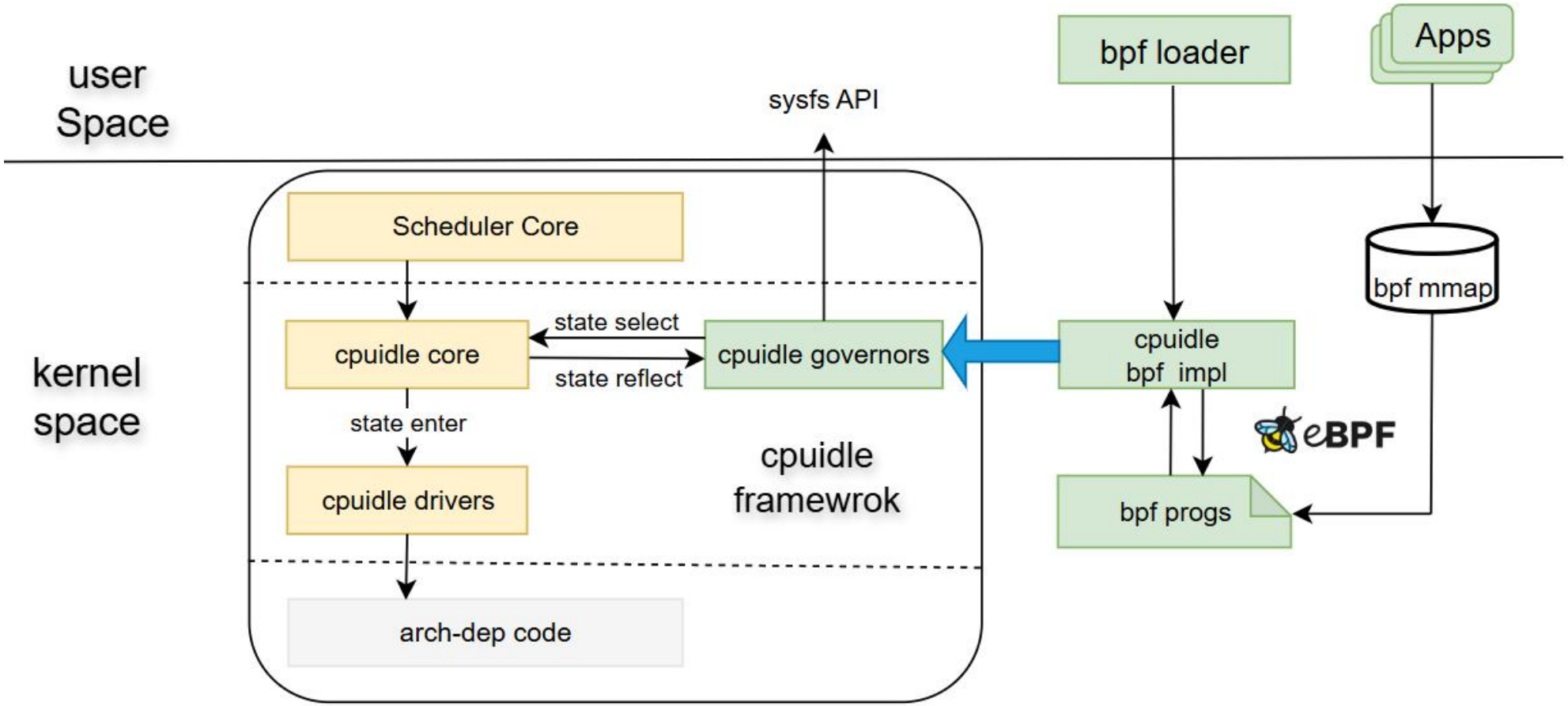
（如Apps特征/灭屏/游戏模式）

- sysfs方式策略切换延迟高
- sysfs接口无法根据多样化的用户场景进行实时动态调整



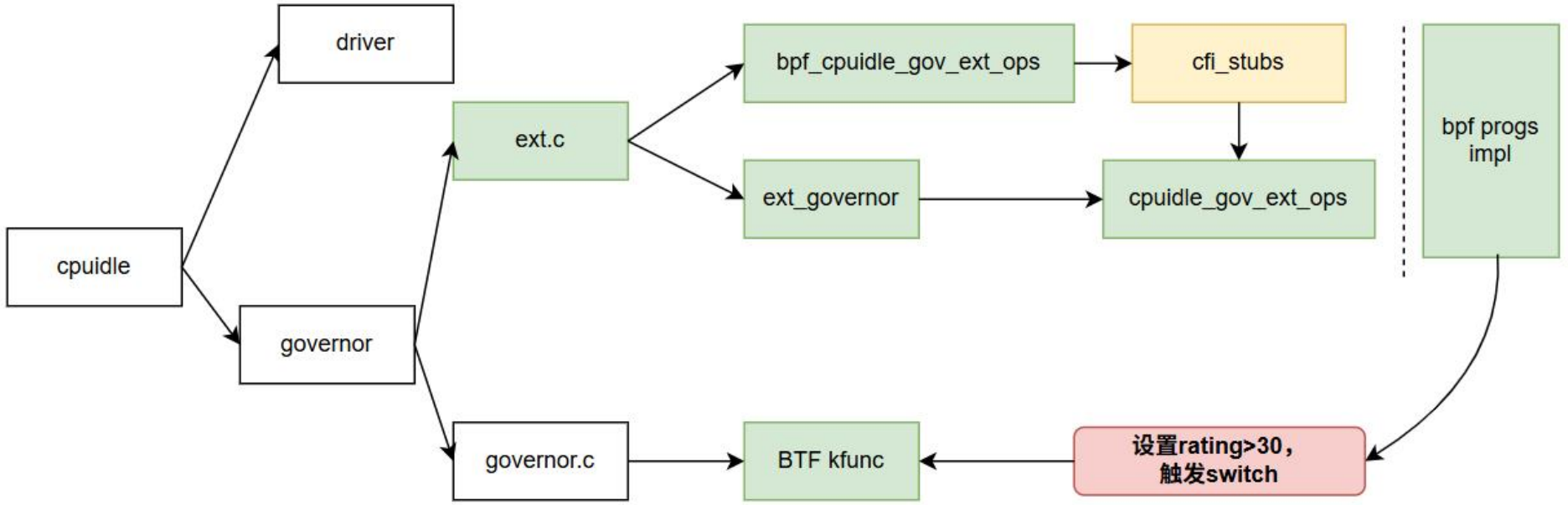
02 方案框架设计

- 依托ebpf struct_ops能力，实现cpuidle的用户态可编程框架
- 通过bpf map建立应用和内核信息的双向感知，使Apps实时行为特征参与空闲状态决策



02 方案框架设计

- 在cpuidle中内置默认ext_governor
- 通过cpuidle_gov_ext_ops提供用户态可编程接口，实现向内核动态注入idle策略
- 封装关键内核状态KFUNC，赋能bpf progs获取内核关键信息



02 方案框架设计

cpuidle_gov_ext框架由三部分组成：

- cpuidle_gov_ext_ops中额外增加set_stop_tick()来实现tick控制
- btf kfunc：可获取qos latency、next tick duration等内核关键信息
- bpf prog调用update kfunc更改当前governor rating值，进行policy switch

```
OverView
----------
The BPF cpuidle ext governor registers at postcore_initcall()
but remains disabled by default due to its low priority "rating" with value "1".
Activation requires adjust higer "rating" than other governors within BPF.

Core Components:
1.**struct cpuidle_gov_ext_ops** - BPF-overridable operations:
- ops.enable()/ops.disable(): enable or disable callback
- ops.select(): cpu Idle-state selection logic
- ops.set_stop_tick(): Scheduler tick management after state selection
- ops.reflect(): feedback info about previous idle state.
- ops.init()/ops.deinit(): Initialization or cleanup.

2.**Critical kfuncs for kernel state access**:
- bpf_cpuidle_ext_gov_update_rating():
  Activate ext governor by raising rating must be called from "ops.init()"
- bpf_cpuidle_ext_gov_latency_req(): get idle-state latency constraints
- bpf_tick_nohz_get_sleep_length(): get CPU sleep duration in tickless mode
```

```
+/* Initialize the BPF cpuidle governor */
+SEC("struct_ops.s/init")
+int BPF_PROG(bpf_cpuidle_init)
+{
+       return bpf_cpuidle_ext_gov_update_rating(60);
+}
+
+/* Cleanup after the BPF cpuidle governor */
+SEC("struct_ops.s/exit")
+void BPF_PROG(bpf_cpuidle_exit) { }
+
+/* Struct_ops linkage for cpuidle governor */
+SEC(".struct_ops.link")
+struct cpuidle_gov_ext_ops ops = {
+       .enable  = (void *)bpf_cpuidle_enable,
+       .disable = (void *)bpf_cpuidle_disable,
+       .select  = (void *)bpf_cpuidle_select,
+       .set_stop_tick = (void *)bpf_cpuidle_set_stop_tick,
+       .reflect = (void *)bpf_cpuidle_reflect,
+       .init    = (void *)bpf_cpuidle_init,
+       .exit    = (void *)bpf_cpuidle_exit,
+       .name    = "BPF_cpuidle_gov"
+};
--
```

03 示例展示

图1: 通过bpf trace cpu_idle统计结果分析：C2时长（57%）明显高于C4（21%），有一定优化空间

```
在不同Cx的停留时长百分比:(trace_cpu_idle)
CPUId      percent_running(%)     percent_0(c2)(%)     percent_1(c4)(%)
0                      21.18%               57.17%               21.65%
1                       5.66%                4.72%               89.61%
2                       0.09%                0.95%               98.96%
3                       0.06%                0.95%               98.99%
4                       5.33%                4.70%               89.97%
5                       4.87%                4.57%               90.56%
6                       0.61%                0.02%               99.37%
7                       0.16%                0.01%               99.84%
```

图2: 通过trace cpuidle停留时长的结果分析：

cpu idle停留时长超C4阈值(按600us计)的百分比55%，远超图一cpu C4实际时长占比21%

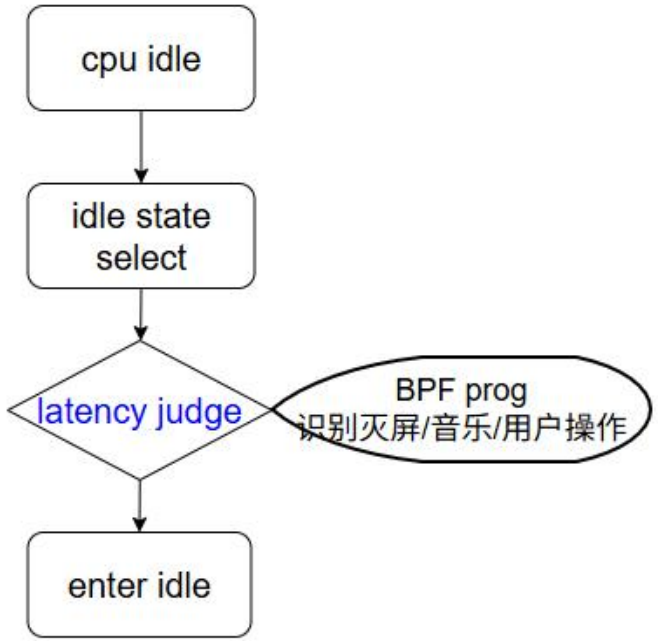
结论：C2->C4 预计最大的优化空间34%

```
cpu上swapper时长分段占比------(ns/):(sched_switch)
CPU    000~001us 001~100us  100~600us  600~...us
0              0.01%        13.95%        30.18%        55.87%
1              0.05%         3.51%         8.75%        87.69%
2              8.33%         0.00%         0.00%        91.67%
3              5.26%         0.00%         0.00%        94.74%
4              0.04%         3.84%        11.78%        84.34%
5              0.00%         2.68%         9.91%        87.41%
```

03 示例展示

- 传统手机idle策略中，为了提高性能，避免频繁进出idle state，Task调度/调频等时机，在满足一定条件时设置延迟，在此延迟期间内强制进入C2而非C4；
  - 过度防护：CPU0上延迟阈值（8.5ms）远超C4状态切换耗时（0.56ms），产生无效等待能耗
  - 场景失配：无法感知用户操作模式（如灭屏），持续高延迟策略加剧损耗

- 基于cpuidle_gov_ext可编程框架，实时捕获用户操作事件（触屏/灭屏），经BPF Map将场景数据注入空闲策略，实时调整延迟阈值



调整前，Cpu0上，enter idle latency为8.5ms，C0占比57%

在不同Cx的停留时长百分比:(trace_cpu_idle)

CPUId	percent_running(%)	percent_0(c2)(%)	percent_1(c4)(%)
0	21.18%	57.17%	21.65%
1	5.66%	4.72%	89.61%
2	0.09%	0.95%	98.96%
3	0.06%	0.95%	98.99%
4	5.33%	4.70%	89.97%
5	4.87%	4.57%	90.56%
6	0.61%	0.02%	99.37%
7	0.16%	0.01%	99.84%

调整后：Cpu0上，0.6~1ms lantency，C2占比10%，降低40%

在不同Cx的停留时长百分比:(trace_cpu_idle)

CPUId	percent_running(%)	percent_0(c2)(%)	percent_1(c4)(%)
0	17.57%	9.82%	72.61%
1	4.59%	1.57%	93.84%
2	0.09%	0.73%	99.18%
3	0.07%	0.72%	99.21%
4	3.89%	1.56%	94.56%
5	3.57%	1.51%	94.92%
6	0.06%	0.01%	99.94%
7	1.80%	0.05%	98.15%

04 总结与展望

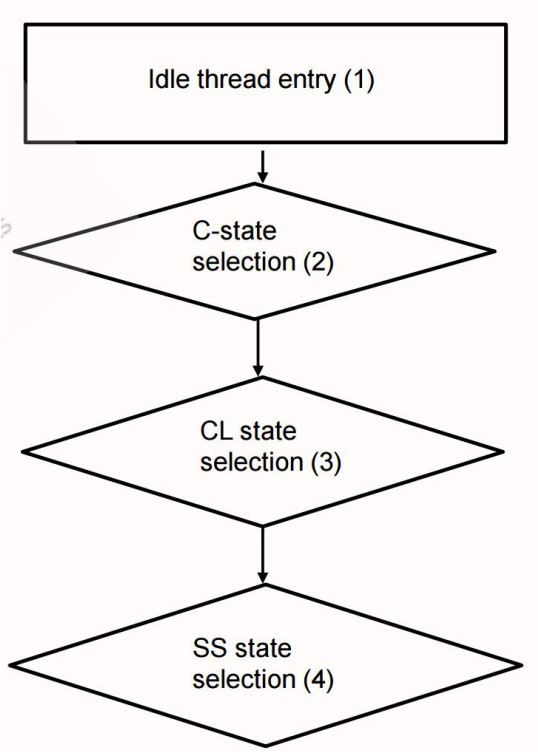
➢ 总结

- 基于cpuidle_governor_ext框架，通过BPF struct_ops扩展，

可以在用户态基于用户场景的动态变化（灭屏、音视频、应用状态），自由定制空闲策略

➢ 展望

- 某些cpu区分多种状态层级（如右图）：CPU Core→Cluster→Subsystem，需持续挖掘更优的多层级管理策略
- Idle状态的选择，同时受sched影响，借力bpf sched_ext实现调度-能效联合优化
- 综合考虑bpf arena和bpf map，实现用户-内核数据零延迟传输和低内存占用



THANKS