第三届 eBPF开发者大会

www.ebpftravel.com

基于eBPF用户态调度器的Android系统性能优化与应用实践

荣耀终端 | OS Kernel Lab

古钦辉 / 肖俊

HONOR

中国·西安

1. sched_ext 框架介绍
2. Android 移动端应用启动性能分析
3. 基于 sched_ext 的优先级调度优化
4. 总结

① sched_ext 框架介绍

Linux内核调度器：

· 通用调度器（CFS、EEVDF）

· 越来越复杂的用户交互场景

· 新调度策略的实验成本和维护难度

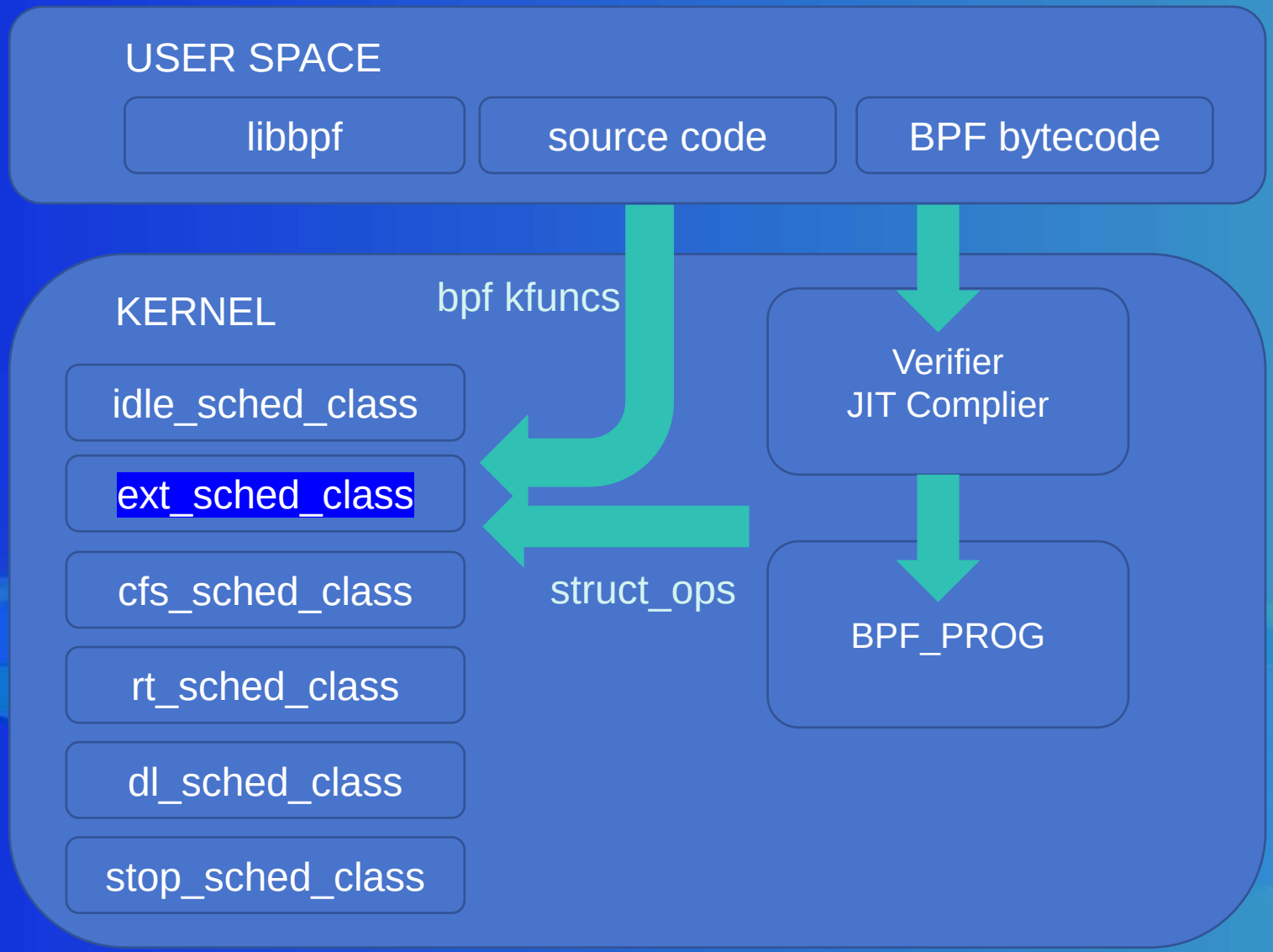
SCHED_EXT 的特点和优势：

1、灵活性，可以通过 eBPF 技术在运行时从用户态加载调度器到内核

2、降低试验新调度策略的难度，实现快速部署

3、安全性 / 防饿死

① sched_ext 框架介绍



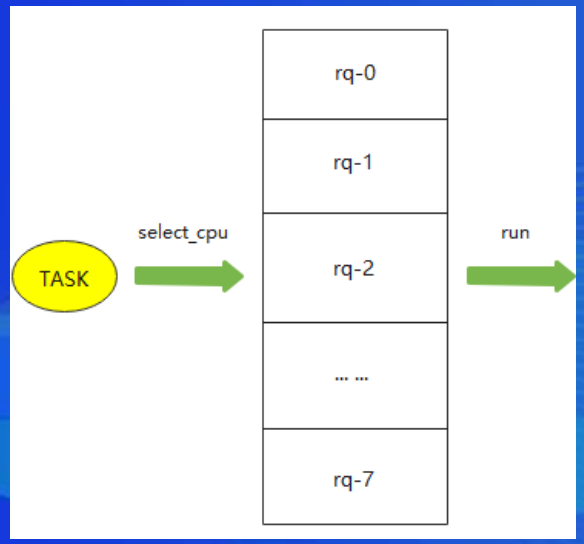
SCX 框架：

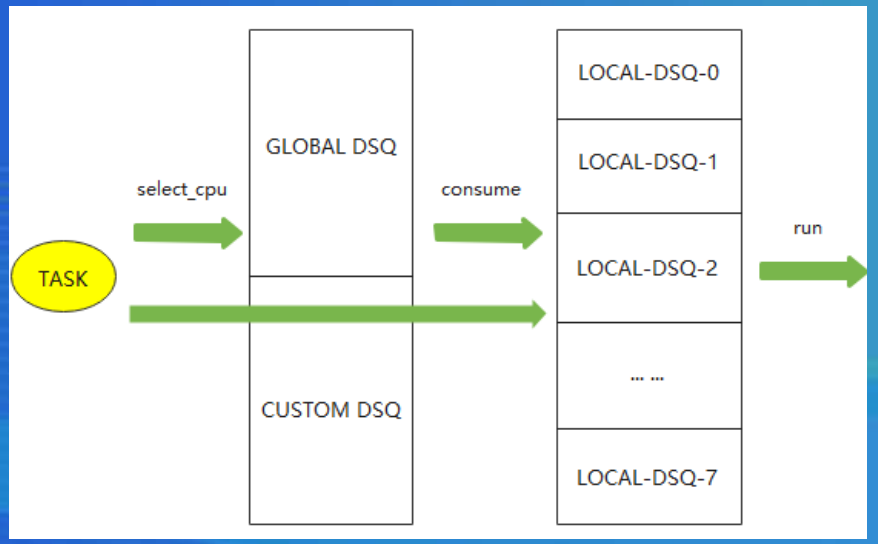
- core scheduler 负责在内核态修改调度模块的逻辑：任务队列管理 / 任务抢占 / 负载均衡 / 选核 / ...
- bpf scheduler 实现具体的任务调度策略和执行调度程序的注册加载：注入动态策略

① sched_ext 框架介绍

s32 (*select_cpu)(struct task_struct *p, s32 prev_cpu, u64 wake_flags);
// 选核：选择cpu

void (*enqueue)(struct task_struct *p, u64 enq_flags);
// 入队：分发任务



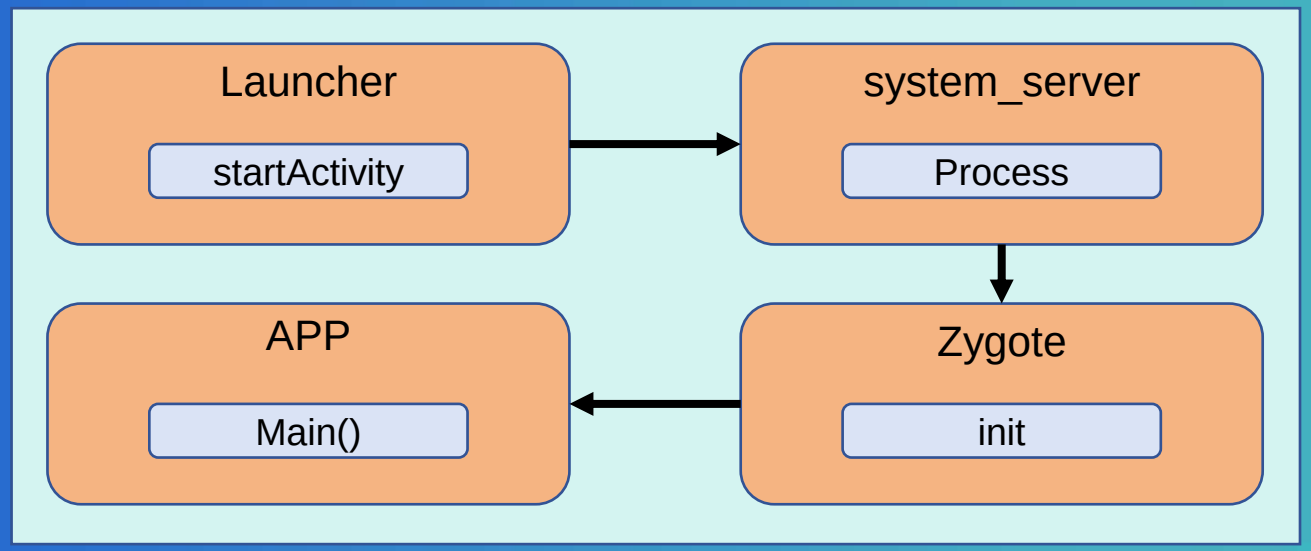



```
SEC(".struct_ops")
struct sched_ext_ops hndemo_ops = {
    /* core func */
    .select_cpu     = (void *)hndemo_select_cpu,
    .enqueue        = (void *)hndemo_enqueue,
    .dispatch       = (void *)hndemo_dispatch,
    .set_cpumask    = (void *)hndemo_set_cpumask,
    /* task state */
    .init_task      = (void *)hndemo_init_task,
    .running        = (void *)hndemo_running,
    .runnable       = (void *)hndemo_runnable,
    .stopping       = (void *)hndemo_stopping,
    .enable         = (void *)hndemo_enable,
    .set_weight     = (void *)hndemo_set_weight,
    /* scheduler state */
    .init           = (void *)hndemo_init,
    .exit           = (void *)hndemo_exit,
    // ...
}
```

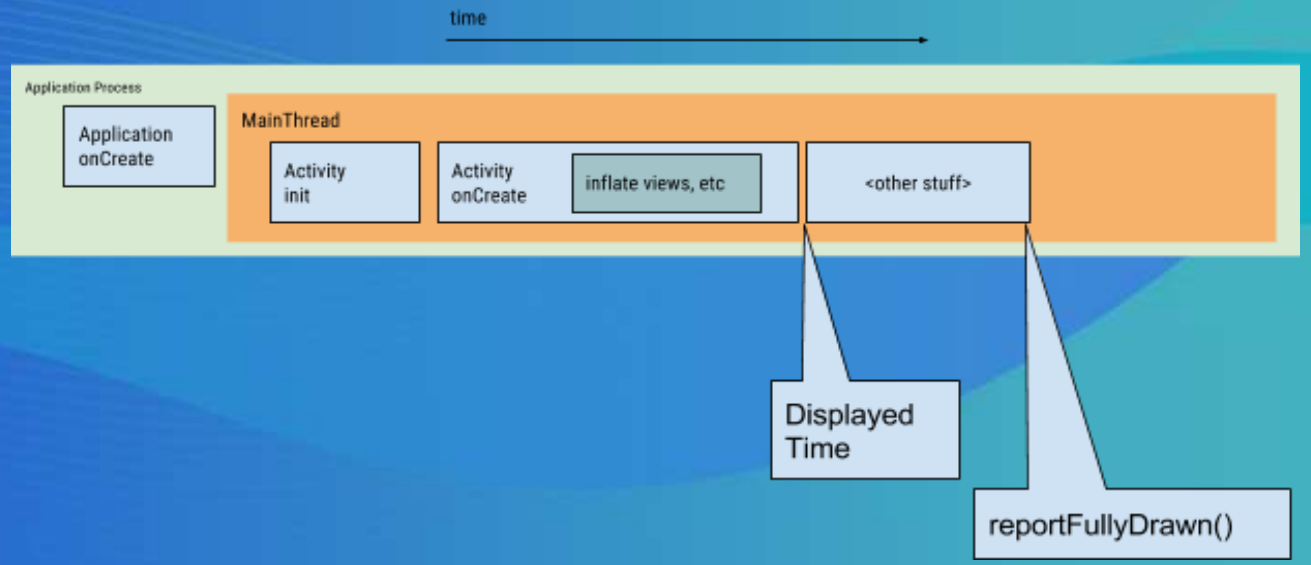
void (*dispatch)(s32 cpu, struct task_struct *prev);
// 选择任务：cpu 从指定队列消费任务

② Android应用启动性能分析

应用启动关键流程：
1、创建app对象
2、启动main thread
3、创建main activity
4、创建布局 Inflate views
5、首帧绘制 Initial draw



Android Vitals 会认为您的应用存在启动时间过长的情况： （from Google Develop）
•冷启动用了 5 秒或更长时间。
•温启动用了 2 秒或更长时间。
•热启动用了 1.5 秒或更长时间。
(对性能的指标随着业务而变化)



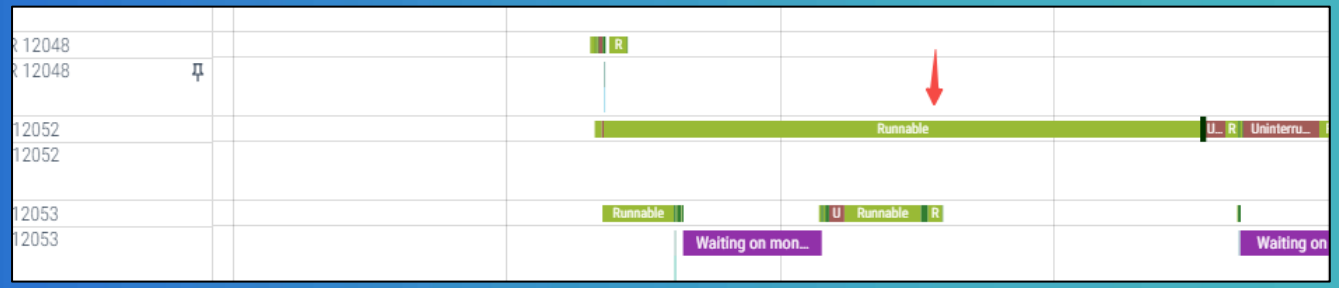
主要业务进程：
Launcher
MainThread
RenderThread
Binder事务
...

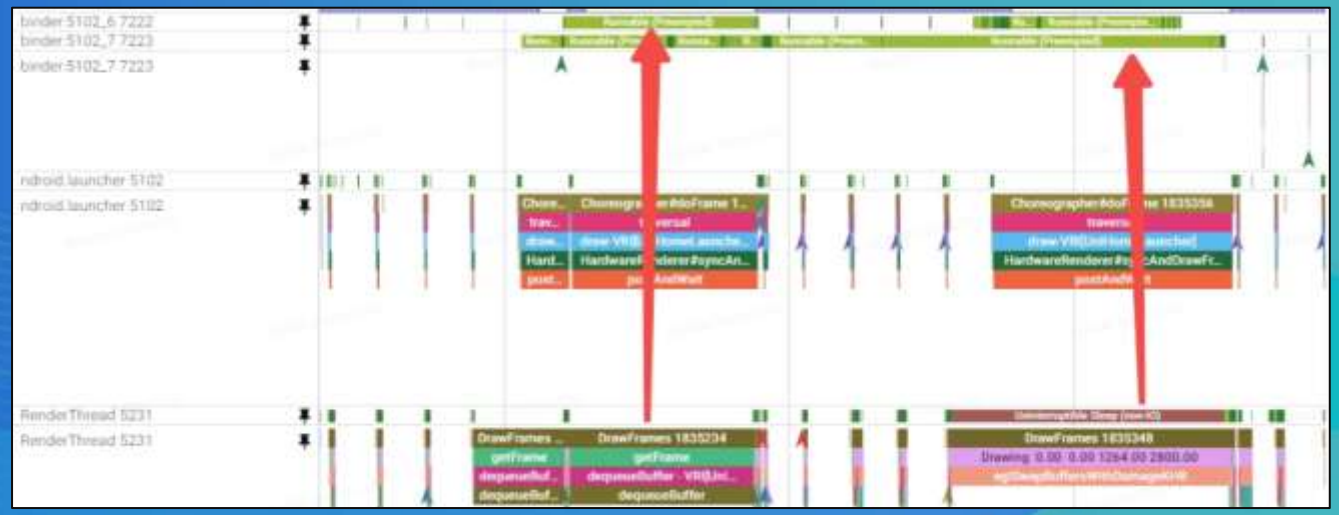
主要调度类：
RT 实时调度
CFS 公平调度

② Android应用启动性能分析

问题：

- 关键业务进程等待子线程runnable
- Binder 等待时间长
- UI、渲染任务调度不及时
- 特定业务调度不及时



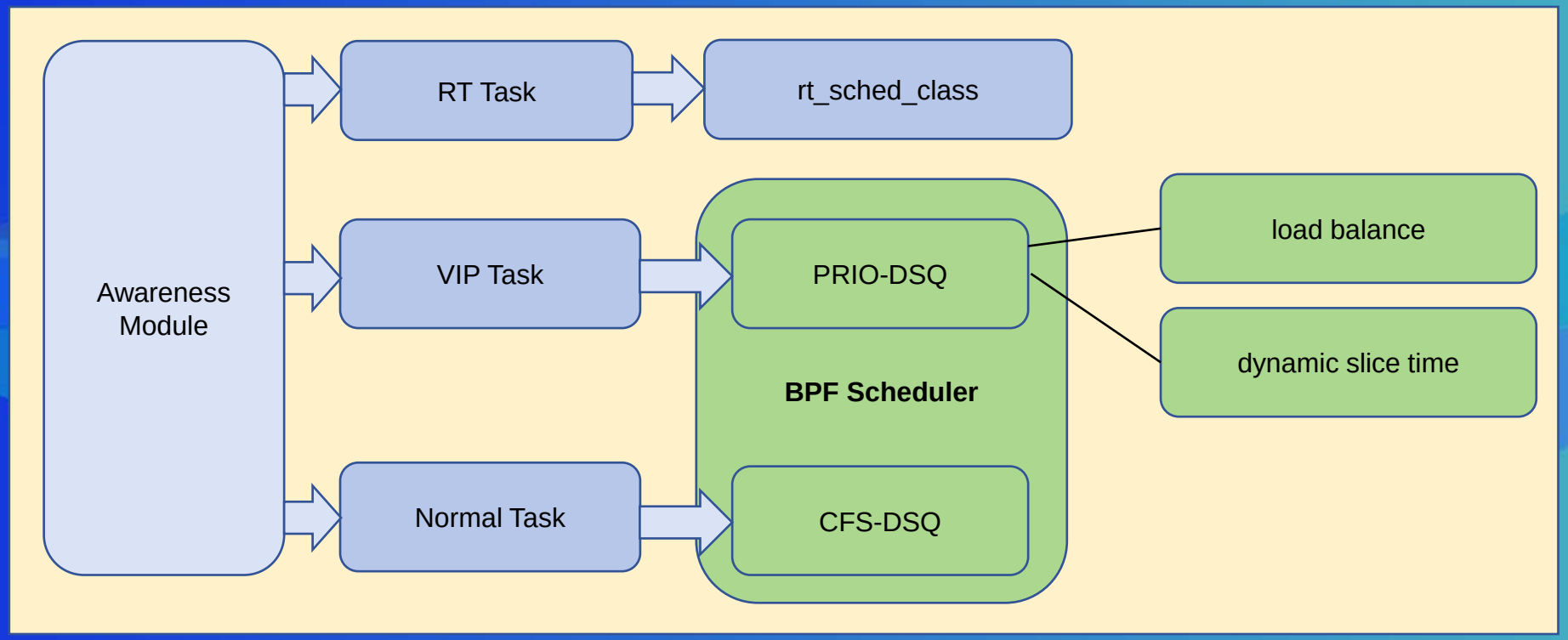


优化实践目标：

- 通过 scx 调度器实现调度策略拓展
- 识别关键任务链，精准优化
- 覆盖多个用户场景

③ 基于sched_ext的优先级调度

- 高优先级任务派发队列 PRIO-DSQ：load balance / idle first / dynamic cpu affinity
- 默认任务派发队列 CFS-DSQ
- 动态时间片分配：细粒度优化
- 依赖识别模块和优先级传递：关键任务标签 / 前台 / 动效、UX / 系统服务 / ...



③ 基于sched_ext的优先级调度

测试数据:

(基于高通骁龙8Gen3 / Kernel 6.1 / Android 14)

1. 上下文切换耗时
2. APP应用启动时间
3. APP进程任务的平均Runnable时间

上下文切换耗时	内核调度器	BPF调度器	Diff
Avg Time	X1	X2	<1%

APP S1	CFS	SCX	Improvement
Avg Lnch Time	588ms	433ms	26.3%
Runnable (0-4us)	4.88%	7.48%	\
Runnable (4-16us)	22.43%	32.84%	\
Runnable (16-128us)	31.52%	25.04%	\
Runnable (128-1024us)	26.13%	20.27%	\
Runnable (1ms-4ms)	10.01%	9.49%	\
Runnable (>4ms)	5.01%	4.88%	\

APP S2	CFS	SCX	Improvement
Avg Lnch Time	446ms	293ms	34.3%
Runnable (0-4us)	3.82%	5.40%	\
Runnable (4-16us)	25.61%	36.41%	\
Runnable (16-128us)	26.39%	27.79%	\
Runnable (128-1024us)	26.63%	19.89%	\
Runnable (1ms-4ms)	11.90%	7.85%	\
Runnable (>4ms)	5.66%	2.66%	\

④ 总结

- sched_ext 在移动端的部署和实践，适配移动端复杂的业务场景
- 运行时加载，灵活调度，快速部署，可作为快速探索不同技术方向能力的工具
- BPF验证器的强约束编程会给复杂逻辑的实现带来新的挑战
- 拓展其他领域 / 和 AI 的结合探索

THANK YOU