1. Rust改写Zephyr RTOS的网络协议栈

选题背景
Zephyr是Linux基金会旗下的开源实时操作系统，广泛应用于物联网和嵌入式设备。其网络协议栈（L2/L3层）目前主要用C实现，但存在内存安全风险。使用Rust改写可结合其内存安全特性和异步编程优势，尤其适合物联网设备对稳定性和低延迟的需求。

参考仓库

- Zephyr官方仓库：https://github.com/zephyrproject-rtos/zephyr/tree/main/subsys/net
- Rust嵌入式网络协议栈参考：https://github.com/smoltcp-rs/smoltcp
  技术亮点
  可结合Rust的`async/await`机制重构ARP/IPv4/TCP协议实现，并与Zephyr的驱动层交互。

1.1 L2 数据链路层

- **任务目标**：替换现有的以太网（Ethernet）和 IEEE 802.15.4 MAC 层实现
- **具体内容**：
  - 用 Rust 重写 `net_if` 接口的发送/接收逻辑
  - 实现 `Ethernet` 帧封装/解封装（参考 smoltcp 的 `phy` 模块）
  - 支持 VLAN 标记处理
- **技术难点**：与硬件驱动的 DMA 缓冲区交互（需通过 `unsafe` 安全封装）

1.2 L3 网络层

- **IPv4/IPv6 协议栈重构**：
  - 实现基于 `no_std` 的 IP 包解析/封装
  - 重构路由表（使用 Rust `BTreeMap` 替代 C 链表）
  - 支持 ICMPv4/ICMPv6 协议（如 Ping 响应）
- **ARP/NDP 协议**：
  - 用 Rust 的 `HashMap` 缓存表项，结合 `async` 实现请求超时机制
  - 避免传统 C 实现中的竞态条件（如 ARP 缓存污染攻击）

1.3 L4 传输层

- **TCP/UDP 协议实现**：
  - 基于状态机的 TCP 连接管理（参考 rubble 的轻量级实现）
  - 滑动窗口和拥塞控制算法（如 Reno/CUBIC）
  - 零拷贝的 UDP 数据包处理（利用 Rust 的 `Cow` 类型）

---

2. Rust改写HarmonyOS分布式任务调度子系统

选题背景
HarmonyOS的分布式软总线（DSoftBus）是其核心子系统，负责跨设备任务调度。目前实现基于C++，存在复杂的线程同步问题。用Rust改写可借助其所有权模型解决竞态条件，同时利用`tokio`实现高效异步通信。

参考仓库

- OpenHarmony官方代码仓：https://gitee.com/openharmony/communication_dsoftbus
- Rust分布式框架参考：https://github.com/tokio-rs/tokio
  技术亮点
  聚焦跨设备任务调度中的消息队列和资源分配模块，尝试实现零拷贝数据传输。

2.1 设备发现与组网模块

- **任务目标**：替换原有的基于 C++ 的设备发现协议
- **具体内容**：
  - 实现 **mDNS/DNS-SD** 协议（Rust 库参考 mdns-sd）
  - 构建基于 **QUIC** 的轻量级组网通信（替代传统 TCP）
  - 设备指纹安全校验（使用 Rust 的 `ring` 加密库）

2.2. 任务迁移与资源同步

- **任务目标**：重构跨设备任务迁移的上下文保存/恢复机制
- **具体内容**：
  - **上下文序列化**：使用 `serde` + `bincode` 替代 Protocol Buffers
  - **内存热迁移**：通过 `mmap` 实现进程状态的零拷贝传输
  - **资源依赖解析**：构建 DAG（有向无环图）描述任务依赖关系

2.3. 调度策略引擎

- **任务目标**：用 Rust 类型系统强化调度策略安全性
- **具体内容**：
  - 实现 **负载均衡算法**（如一致性哈希 + 加权轮询）
  - 开发 **优先级抢占式调度器**（利用 `tokio::sync::Mutex` 替代传统信号量）
  - 动态资源感知调度（实时监控设备 CPU/RAM/网络状态）

---

3. Rust为FreeRTOS设计基于机器学习的任务预测调度器

选题背景
FreeRTOS的任务调度依赖静态优先级，难以适应动态负载。用Rust集成轻量级ML模型（如TinyML），通过`tch-rs`绑定实现运行时任务需求预测，构建自适应调度算法。

参考仓库

- FreeRTOS内核源码：https://github.com/FreeRTOS/FreeRTOS-Kernel

- Rust嵌入式ML参考：https://github.com/varunshenoy/rust-tinyml
  技术亮点
  在资源受限环境下（<64KB RAM）部署神经网络模型，利用Rust的`const generics`实现编译时张量维度校验。

3.1. 任务特征数据采集与预处理

- **目标**：实时记录任务执行特征，构建训练数据集
- **具体任务**：
  - **数据采集**：
    - 在 FreeRTOS 的 `vTaskSwitchContext` 钩子函数中记录
  - **数据存储**：
    - 使用循环缓冲区（`heapless::Vec`）存储最近 N 次任务切换记录
    - 通过 `serde` 序列化数据到外部 Flash/SD 卡（离线训练用）

3.2. 轻量级机器学习模型设计与训练

- **目标**：构建适用于嵌入式场景的预测模型
- **具体任务**：
  - **模型选型**：
    - 决策树（`linfa` 库）
    - 轻量级 LSTM（`tch-rs` + TensorFlow Lite 转换）
    - 时间序列预测（ARIMA 或 Prophet 简化版）

3.3. 调度器核心算法改造

- **目标**：将预测结果融入 FreeRTOS 调度策略
- **具体任务**：
  -
    - 在 `xPortPendSVHandler`（ARM Cortex-M 上下文切换）中插入预测逻辑
    - 动态调整 `uxPriority` 或时间片（`configTICK_RATE_HZ`）