北京郵電大學
Beijing University of Posts and Telecommunications

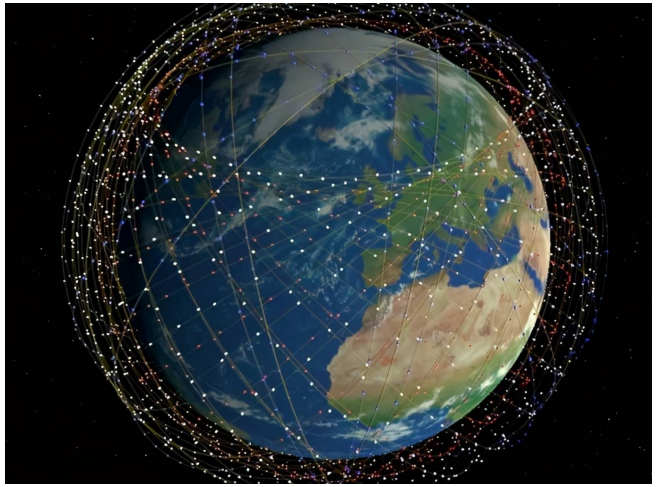
RROS：面向卫星的双内核国产操作系统

李弘宇
北京邮电大学

背景和动机

- 卫星发展势头强劲

北斗导航

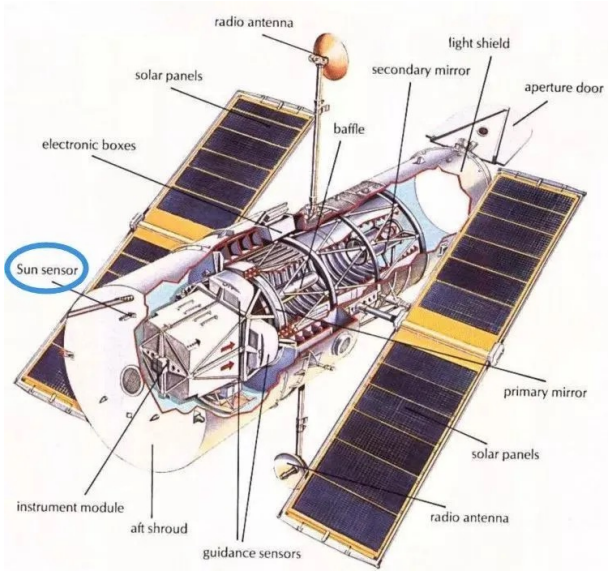


星链组网

习近平总书记指出：“必须推动空间科学、空间技术、空间应用全面发展。”

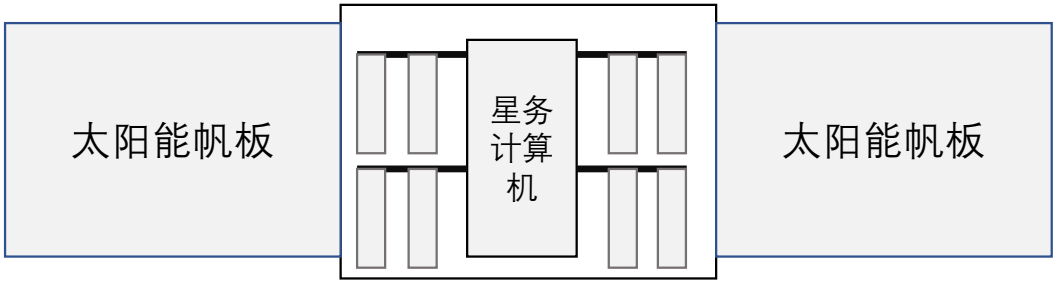
背景和动机

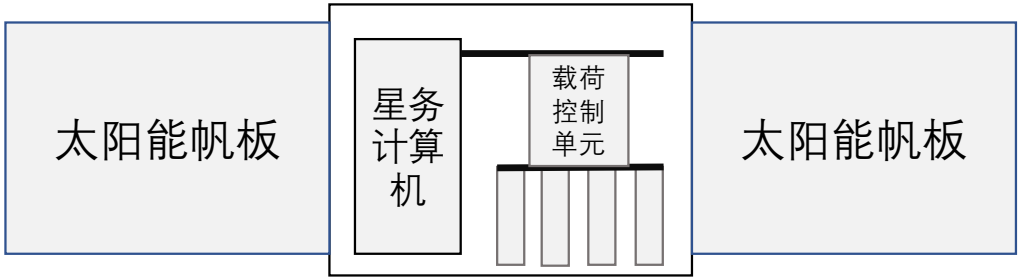
- 卫星的发展对操作系统带来了更高的要求
  - 传统的卫星任务对实时性和稳定性有较高要求
  - 新兴的卫星任务对通用性和性能有新的需求，需要良好的软件生态



背景和动机

- 卫星任务复杂化
  - 案例-软件定义卫星
    - 背景1：卫星的设计需要根据任务的需求重新定义，不同卫星难以复用设计，通用性不强；
    - 背景2：卫星的多个载荷之间会有重复的计算和存储功能，但是这些资源大部分处于闲置状态，未被完全利用；
    - 对于小型卫星，可以将载荷的传感器抽象出来，直连星务计算机；
    - 对于大型卫星，可以将载荷的传感器抽象出来，直连载荷控制单元；
    - 问题：星务计算机/载荷控制单元即需要处理传统实时任务，又需要处理通用任务





背景和动机

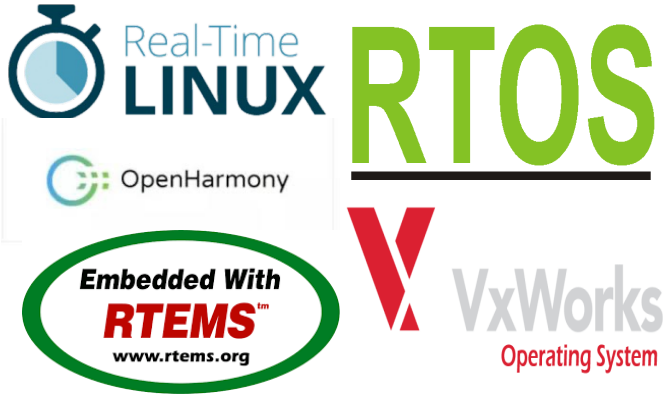
- 卫星任务复杂化
  - 案例-SpaceX 猎鹰火箭
    - 背景：传统卫星发射火箭造价昂贵，在一次卫星发生的成本中占比很高，需要可回收的火箭发射技术来降低成本；
    - SpaceX猎鹰9号火箭采用了更先进的姿态控制算法[1]，虽然已经通过算法优化将非凸优化问题转化为凸优化问题，利用凸优化的工具包求解，但是仍然需要100ms必须求解一次结果，这对于传统的实时操作系统性能提出了更高的要求；
    - 问题：卫星场景下，为了实现更精密的控制，高复杂度的算法将会逐渐增多，这类任务要求操作系统即有良好的通用性和性能，还需要在规定时间内完成求解；

[1] SpaceX火箭姿控算法：Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem

背景和动机

- 传统卫星操作系统难以同时满足通用化和实时性的需求

软件生态丰富，开发便捷灵敏



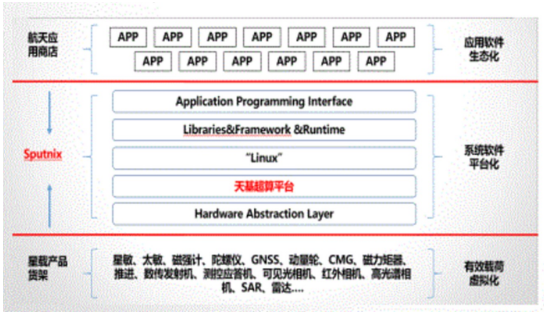
实时性保证

背景和动机

- 已有的解决思路存在缺陷
  - Real-Time Linux
    - 对Linux内核采用Preempt-RT补丁
    - 缺点：只能做到软实时
  - POSIX兼容的实时内核
    - 编写一个兼容POSIX接口的实时内核
    - 缺点：对大型的项目兼容性不够好
  - Amp/虚拟机运行实时内核与Linux
    - 利用Amp/type-1的Hypervisor同时运行Linux kernel和RTOS
    - 缺点：两个内核之间隔离相对较重，通信/性能受到限制；资源分配静态

背景和动机

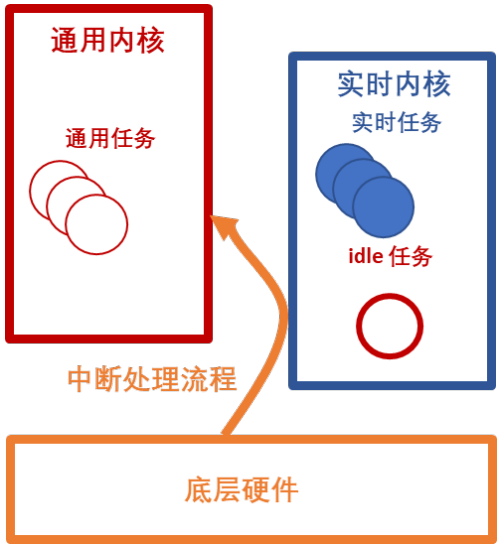
- 已有的解决思路存在缺陷
  - 国内外案例
    - SpaceX猎鹰火箭，星链卫星，dragon飞船，星链均采用COTS器件作为计算硬件，利用三模冗余提升可靠性，采用Preempt-Linux作为主要的操作系统[1]。
    - 天智卫星[2]计划采用Linux作为主要的操作系统
    - Smartsat[3]星座采用Hypervisor技术同时运行Linux和实时操作系统



[1] ELC: SpaceX lessons learned https://lwn.net/Articles/540368/
[2] https://news.lockheedmartin.com/2019-03-20-Lockheed-Martins-First-Smart-Satellites-are-Tiny-with-Big-Missions
[3] https://max.book118.com/html/2020/1220/7152012022003033.shtm

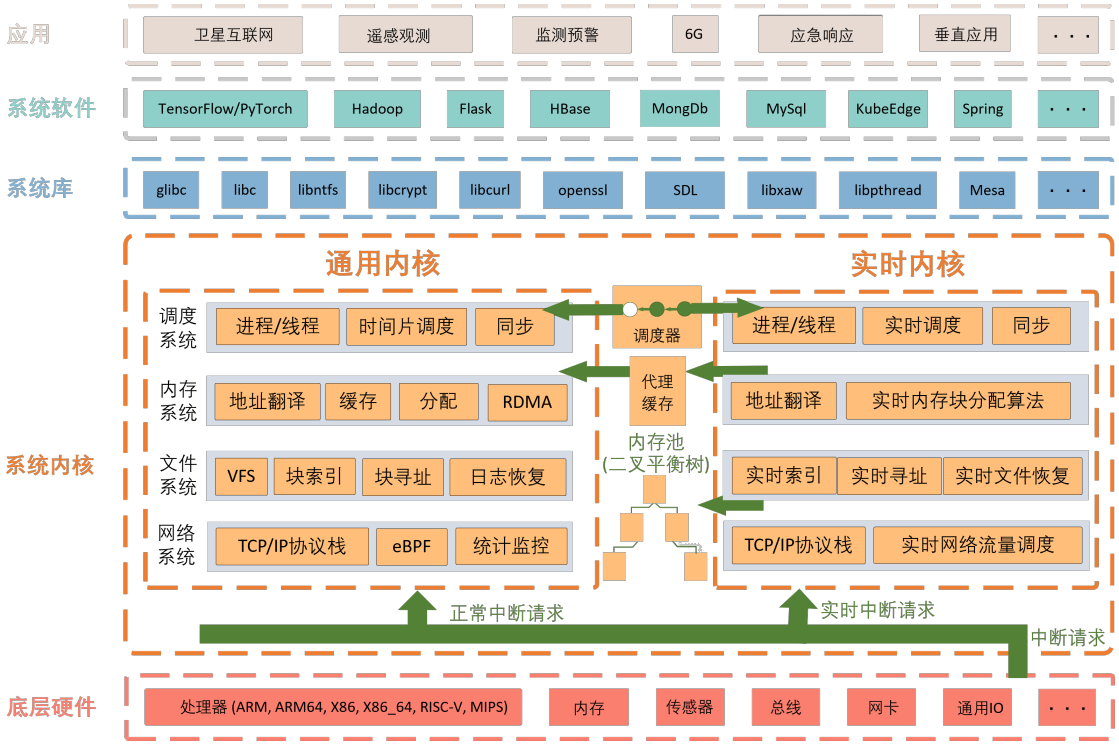
RROS整体架构

- RROS针对实时性和通用性难以兼顾的解决方案：**双内核**
  - 整体架构
    - 通用Linux内核提供通用性的支持
    - 实时内核提供硬实时保证
    - 采用中断虚拟化隔离内核，减少开销
  - 中断处理流程
    - 底层硬件产生的中断首先发送到实时内核
    - 若实时内核无对应中断处理函数再向通用内核转发
  - 任务处理流程
    - 如果实时内核有实时任务，则优先处理实时任务
    - 如果所有的实时任务都被处理完，则处理idle任务
    - 实时内核的idle任务即代表通用内核，此时处理通用内核的通用任务



RROS整体架构

- RROS详细架构：通用型内核Linux（基于C）+实时内核（基于Rust）



Rust语言优势

- 内存安全
- 高性能
- 并发编程
- 生态完善

The EVL Core： https://evlproject.org/core/

RROS整体架构

- RROS针对部署场景受限的解决方案：可伸缩性架构
  - 针对不同开发板的资源差异，可以灵活调整Linux内核的大小
    - 对于资源丰富场景，如手持终端，车载芯片：采用通用Linux
    - 对于资源一般场景，如基站网关，监控摄像：采用剪裁后的Linux
    - 对于资源紧张场景，如海面浮标，气象检测站点：采用ucLinux

RROS架构	资源需求	内核大小	目标硬件
Linux+实时内核	高	15~30MB	Zynq, NVIDIA Jetson, etc
精简Linux+实时内核	中	3~7MB	RPI, HiFive Unmatched, etc
ucLinux+实时内核	低	400~700KB	STM32, 飞凌RT1052, etc

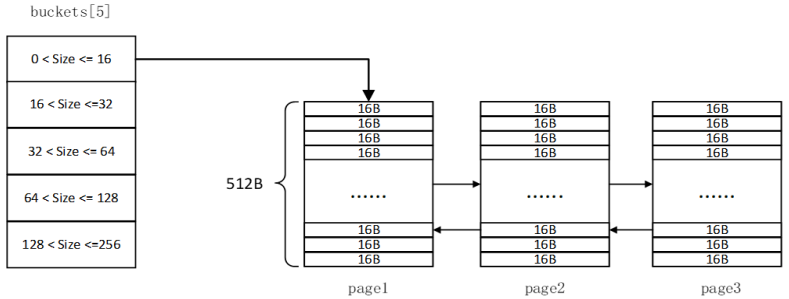
RROS整体架构

- RROS针对实时操作系统生态破碎的解决方案：提供多种实时接口
  - 由于在多种场景下均采用了Linux+实时内核的统一架构
    - 针对通用的任务，我们都可以提供Linux运行环境，提升通用性
    - 针对实时的任务，我们都可以提供freertos，uc/os或者vxworks的接口，提升实时性

RROS架构	通用接口	实时接口
Linux+实时内核	Linux接口	rros，freertos，uc/os，vxworks
精简Linux+实时内核	Linux接口	rros，freertos，uc/os，vxworks
ucLinux+实时内核	Linux接口	rros，freertos，uc/os，vxworks

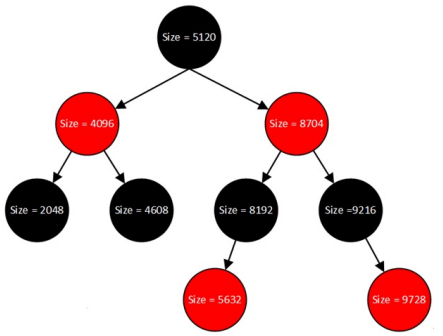
RROS实时内存管理

- 问题
  - Linux内存管理系统：不实时，动态分配
  - 传统实时内存管理系统：实时，静态分配
- RROS针对动态内存管理不实时的解决方案：



小内存管理采用快速缓冲池 分配时间复杂度为O(1)

不实时来源	解决方案
内存分配算法时间复杂度高	高效分配算法+预留空间/延迟释放
实时应用和通用应用不隔离	双内存池隔离实时和通用应用
Hard/Soft Page Fault	内核vmalloc分配
TLB/Cache miss	实时内核缓存动态保活

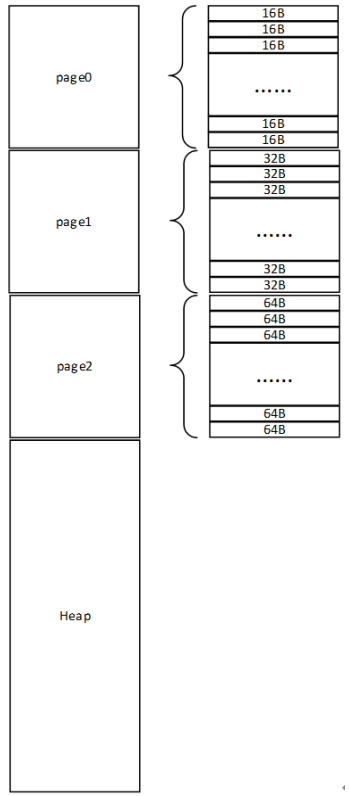


大内存管理直接从红黑树中查找 分配时间复杂度为O(logn)

RROS实时内存管理

- 问题
  - Linux内存管理系统：不实时，动态分配
  - 传统实时内存管理系统：实时，静态分配
- RROS针对动态内存管理不实时的解决方案：

不实时来源	解决方案
内存分配算法时间复杂度高	高效分配算法+预留空间/延迟释放
实时应用和通用应用不隔离	双内存池隔离实时和通用应用
Hard/Soft Page Fault	内核vmalloc分配
TLB/Cache miss	实时内核缓存动态保活

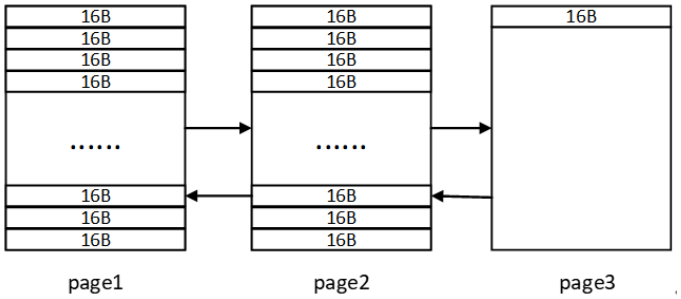


小内存管理分配时采用预留空间方法 提升实时性

RROS实时内存管理

- 问题
  - Linux内存管理系统：不实时，动态分配
  - 传统实时内存管理系统：实时，静态分配
- RROS针对动态内存管理不实时的解决方案：

不实时来源	解决方案
内存分配算法时间复杂度高	高效分配算法+预留空间/延迟释放
实时应用和通用应用不隔离	双内存池隔离实时和通用应用
Hard/Soft Page Fault	内核vmalloc分配
TLB/Cache miss	实时内核缓存动态保活

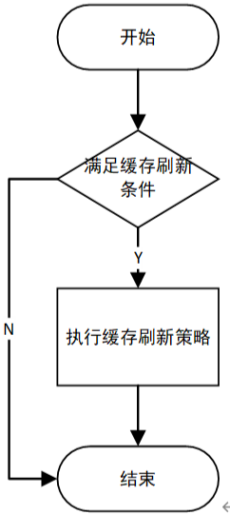


小内存管理释放时采用延迟释放方法 提升实时性

RROS实时内存管理

- 问题
  - Linux内存管理系统：不实时，动态分配
  - 传统实时内存管理系统：实时，静态分配
- RROS针对动态内存管理不实时的解决方案：

不实时来源	解决方案
内存分配算法时间复杂度高	高效分配算法+预留空间/延迟释放
实时应用和通用应用不隔离	双内存池隔离实时和通用应用
Hard/Soft Page Fault	内核vmalloc分配
TLB/Cache miss	实时内核缓存动态保活



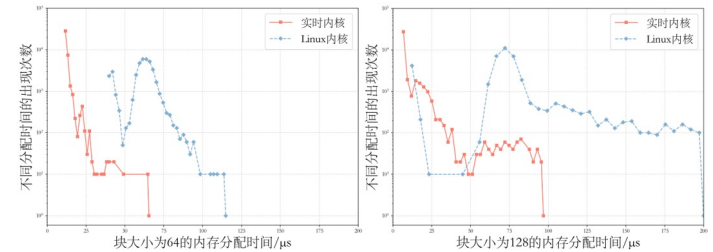
在通用内核执行任务时 对实时内核缓存保活

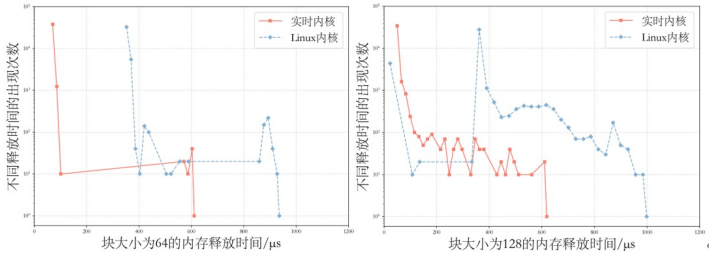
1. 对TLB刷新全部缓存，对Cache刷新部分缓存
2. 定期访问页表项提升计数
3. 编译期插入预读取指令 __builtin_prefetch，使数据常驻Cache

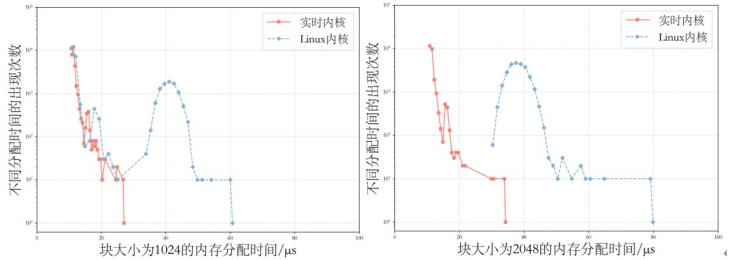
RROS实时内存管理

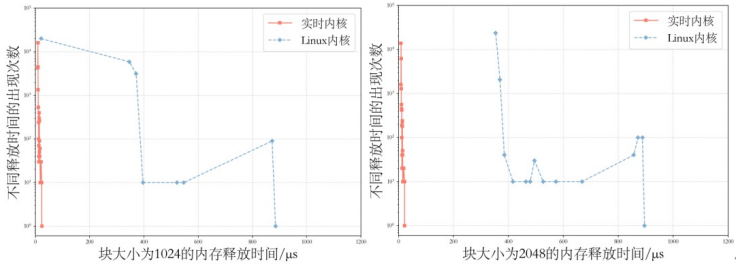
- RROS动态内存管理方案性能测试
  - 小内存分配/释放性能测试
  - 大内存分配/释放性能测试
  - 随机大小分配/释放性能测试

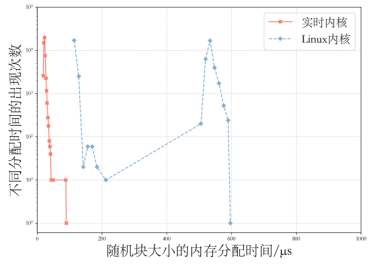
策略	优化前T_{max}/μs	优化后T_{max}/μs	优化降低
预留空间	6.12	1.67	72.71%
延迟释放	548.41	287.74	47.54%
TLB优化	32.12	12.61	60.75%
Cache优化	28.15	8.12	71.16%

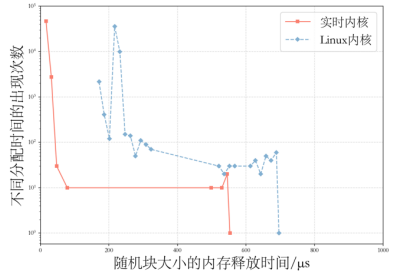












RROS整体架构

- 优势
  - **实时性强**
  - 内核达到硬实时要求，并且兼容现有的rtos的接口，应用的迁移、维护和开发更为简单
  - **通用性好**
  - 借助了Linux丰富的生态，可以直接部署tensorflow/k8s等复杂应用
  - **稳定性高**
  - 基于Rust开发增强了内核的稳定性
  - **隔离较轻**
  - 实时内核和Linux内核的交互简单，隔离相对较小
  - **适配性好**
  - 采用可伸缩架构适配多种场景，同时提供多种实时接口便于迁移应用

RROS能力

- 实时操作系统接口方面

实时操作系统子系统	支持能力
线程	1. 支持fifo/rr/tp等多种实时调度算法 2. 提供优先级继承，天花板算法解决优先级反转问题
内存	支持动态分配内存的实时算法，保证灵活性和系统的实时性
文件	使用实时文件系统，提升数据保存的实时性
网络	适配udp协议栈，可以对udp报文实时处理
驱动	可以通过gpio等接口实时驱动外部设备，保证结果的正确、实时性

RROS能力

- 生态建设方面
  - 对于Linux应用可以在无修改的情况下迁移适配
  - 对于实时应用，提供freertos，uc/os，vxworks等实时接口
  - 适配x86/x86_64/arm/arm64/Risc-V等多种架构
  - 适配树莓派，瑞芯微不同型号等多种开发板
  - 内核适配范围广，支持内核剪裁到150kb，可以在2MB存储，256KB内存的MCU上运行

RROS能力

- 性能指标方面
  - 硬件平台：树莓派4b+，软件平台：Linux kernel 5.13
  - 测试应用：Tensorflow 2.12，Kubernetes 1.22.15

对比项目		RROS	PREEMPT-Linux
Timer Latency		5ms	30ms
Network ping Latency		2ms	5ms
TensorFlow	Mobilenet v2	0.25s/img	0.25s/img
	Resnet101	1.45s/img	1.37s/img
	Vgg11	1.27s/img	1.17s/img
	Vit_b_16	1.48s/img	1.64s/img
Kubeedge Cloudcore setup		13s	12s

RROS与已有操作系统比较

对比项目	RROS	Xenomai	Real-Time Linux	Vxworks	uc/os-III	QNX	Free RTOS	Nuttx	Harm onyOS
实时性	好	好	一般	好	好	好	好	好	一般
应用生态	好	好	好	一般	一般	较好	较好	好	较好
稳定性	极高	高	高	极高	极高	极高	高	高	高
性能	好	好	好	一般	一般	一般	好	一般	一般
用户态实时	是	是	是	是	否	是	是	是	是
内核态实时	是	是	否	是	是	是	是	是	是
源码开放	是	是	是	否	否	否	是	是	是
国产系统	是	否	否	否	否	否	否	否	是
适配性	好	一般	一般	一般	一般	一般	一般	好	好

RROS与已有操作系统比较

对比项目	RROS	Xenomai	Real-Time Linux	Vxworks	uc/os-III	QNX	Free RTOS	Nuttx	HarmonyOS
上下文切换开销(μs)			13.6(ARM CortexM3)	11.0(ARM CortexM3)	5.6(ARM CortexM4)	7.2(Micronics LX30WB)	4.2(ARM CortexM4)	暂无	2.0(白皮书)

RROS使用

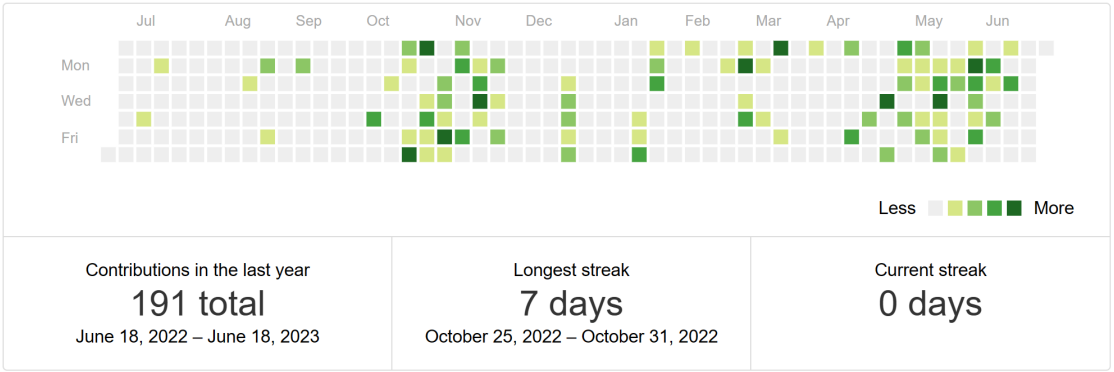
- 开发者如何使用
  - 根据应用场景的需求和资源，选择RROS架构
  - 对于双内核架构
    - 编译RROS
      - 在标准的Linux代码树中打dovetail与RROS的补丁，并编译实时内核
      - 打补丁后，RROS的kernel Image增长小于10MB
    - 开发应用
      - 内核态应用
        - 利用RROS的内核态接口编写实时应用
        - 编译内核并运行
      - 用户态应用
        - 对于需要实时能力的需求，利用RROS的系统库librros编写实时应用
        - 对于需要应用生态的需求，调用Linux下的系统库glibc编写非实时应用
        - 在用户态编译程序并运行

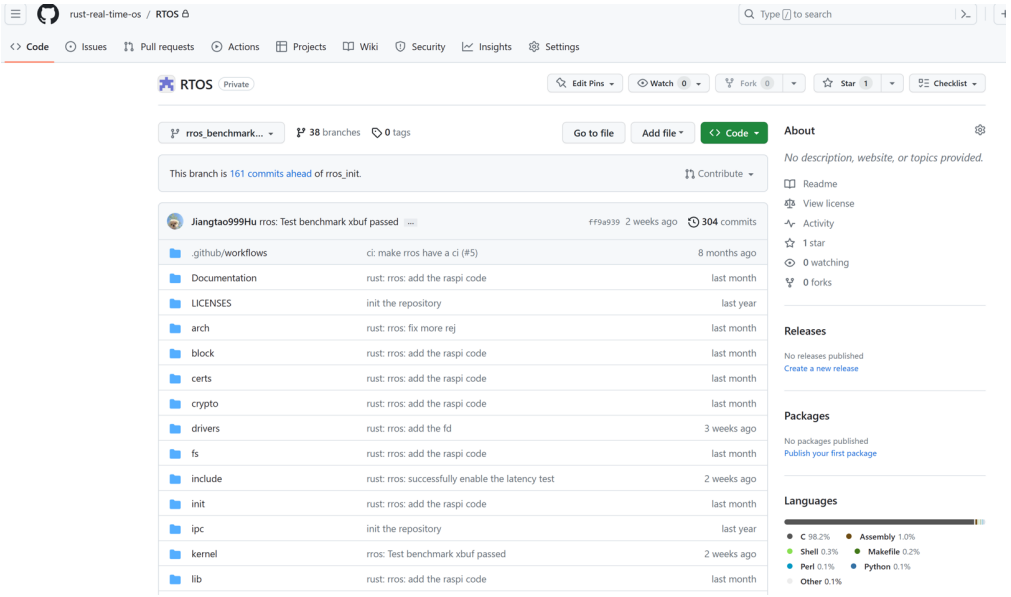
RROS演示

- 实时线程功能演示
- 实时内存分配演示
- 高精度定时器演示
- 实时文件系统演示
- 实时网络功能演示

开发现状

- 目前代码树上有38个分支。以当前主线rros_benchmark_stable为例：
  - 合计304个commit
  - Kernel/rros（实时内核）下
    - 63个rust文件
    - 15200行代码
    - 3979行注释





课程建设

- 团队依托于《操作系统课程设计》课程的需求开发了4个相关的lab。其主题分别为：Rust双链表的实现，RROS的编译和运行，RROS中的实时调度算法，实时内存分配算法四个部分。
  - Lab分为两个难度级别：基础难度用于课程教学，进阶难度用于考核实习生。
- 目前已经有50多名学生完成基础版本的lab，2名实习生完成了进阶版本的lab。另外有不少同学正在参与积极改进和贡献lab的文档、实验设计。
- 此外，我们还搭建了一个用于检查学生完成情况的在线平台，目前搭建在学校内网。学生将完成好的作业的patch提交到云端；服务器会启动一个容器运行和检查学生的完成情况

Lab链接:rust-real-time-os/os_lab: 操作系统课程设计 (github.com)

团队建设

- 目前团队有稳定成员10人（硕博士），另有多人正在考核期。
- 项目已在校内论坛发起招募。我们计划在正式开源后吸引更多感兴趣的开发者加入我们。
- 团队内部气氛活跃，技术氛围浓厚，成员关系和睦，定期开展聚餐等活动。