基于seL4的分区操作系统设计

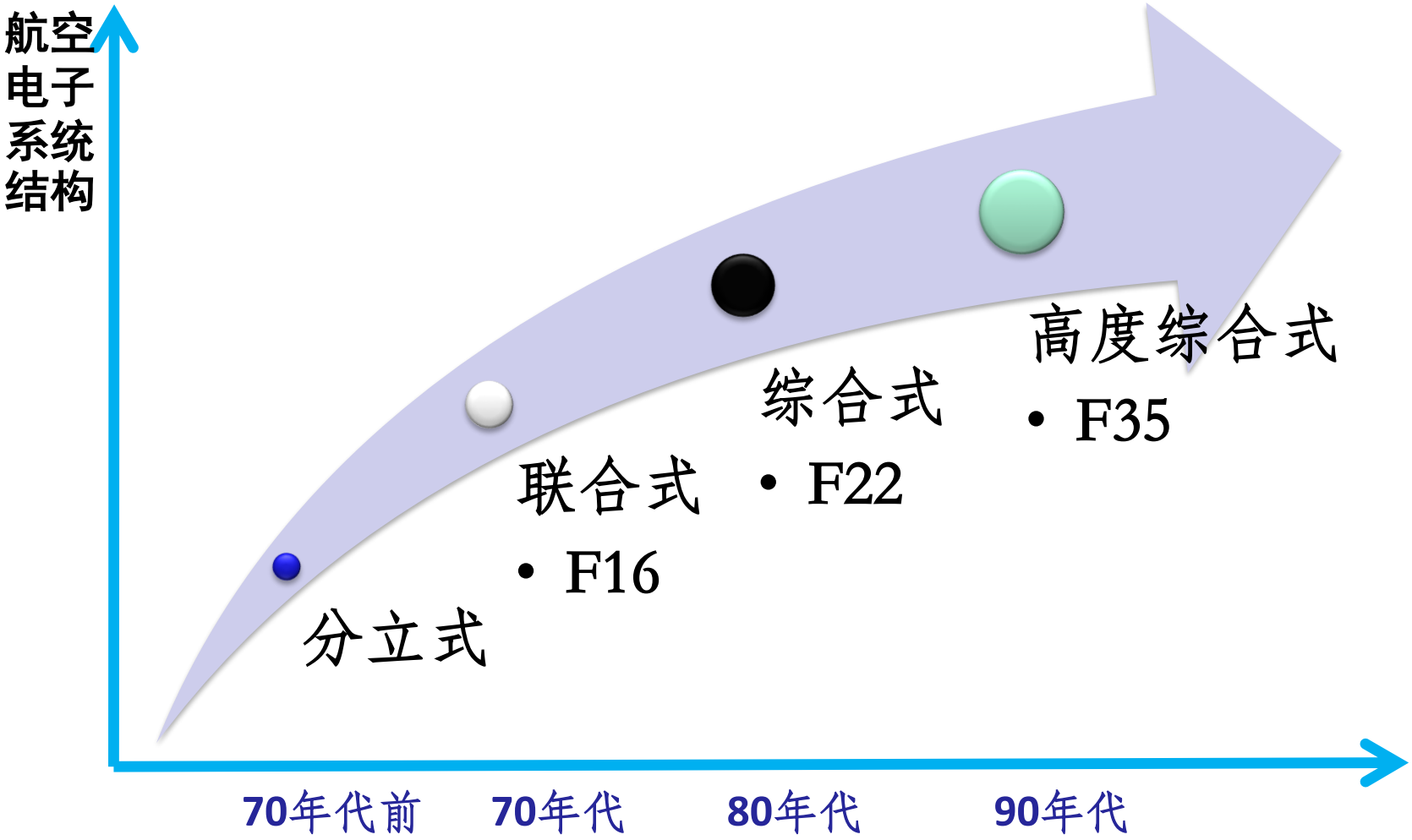
王雷

大纲

- 课题背景与现状
- seL4提供的支持
- 基于seL4的分区系统设计
  - 分区的表示
  - 分区管理方法
  - 分区调度器
  - 分区间通信
  - APEX接口
- 总结

课题背景

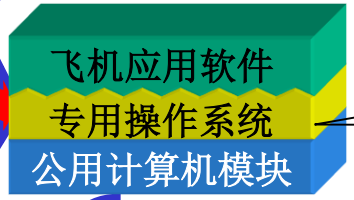
发展趋势：综合化、模块化、网络化



课题背景

没有操作系统

操作系统的雏形
管理软件



专用操作系统

标准操作系统
ARINC653接口
DO-178B验证

课题背景

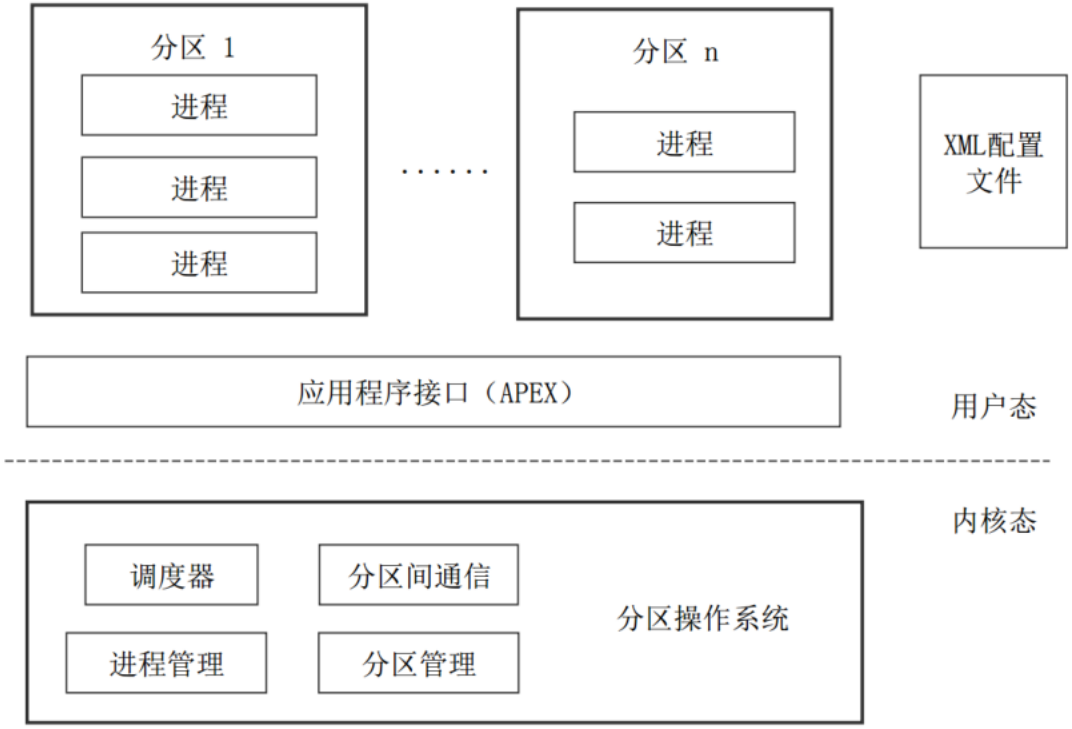
- 分区的背景：嵌入式硬件性能大幅提高
  - 能够集成更多软件——节约成本
  - 例如在航空电子环境中，集成多种软件：
    - » 高安全级：飞行高度、速度的采样程序
    - » 低安全级：影音、广播等多媒体软件
- 问题产生
  - 需要对不同安全级的软件进行保护
- 解决办法
  - ARINC 653提出由嵌入式操作系统使用“分区隔离”实施保护

课题背景

- seL4微内核
  - 2009年澳洲NICTA实验室开发的第三代微内核
  - 安全：内核8700行C代码通过了形式化验证（历史首次）
  - 性能：快速的微内核操作系统
  - 问题：
    - » 不支持ARINC 653分区
- 课题意义
  - 基于seL4实现ARINC 653标准的分区隔离机制，从而让seL4应用于航空电子领域
  - 既保留seL4的内核验证成果，又提供ARINC 653的分区隔离支持

国内外研究现状

- ARINC 653 分区管理
  - 分区、进程数量由配置文件静态规定
  - 分区之间的内存空间隔离

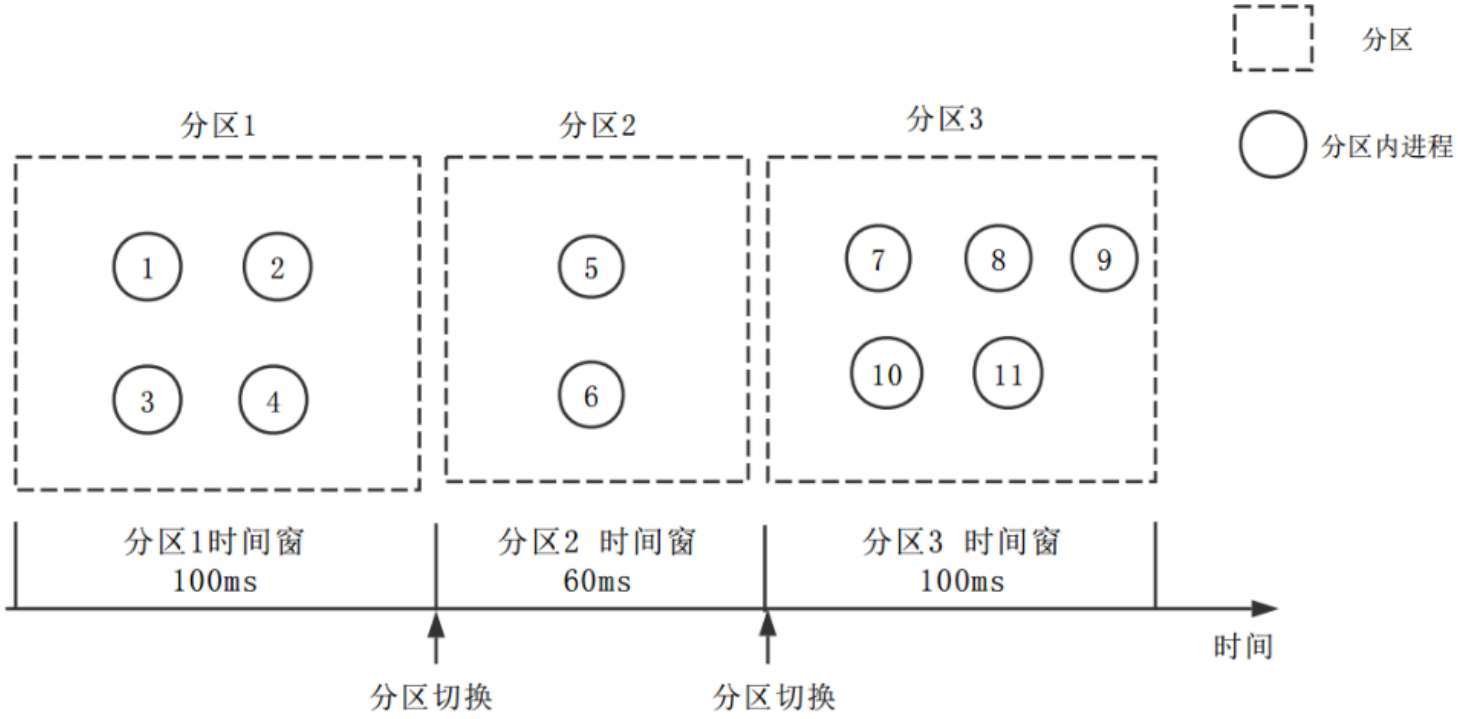


国内外研究现状

- 分区管理研究现状
  - 研究现状
    - VxWorks653，**pikeOS**，用进程表示分区，线程表示分区内进程
    - **Linux 653**，用**Linux**的进程表示分区，用**Linux**的轻量级进程（线程）表示分区内进程。
  - 总结：
    - 一般都用进程（有独立的页表）表示分区，用线程表示分区内进程。利用页表实现分区内存隔离

国内外研究现状

- 分区间时间隔离——两级调度
  - 分区使用固定时间窗调度
  - 分区内进程采用固定优先级调度



国内外研究现状

- 时间隔离研究现状
  - 研究现状
    - **2003，M.D.Bennett：L4/IMA**，内核态中，多个优先级队列实现实现分区调度。
    - 2009，**pikeOS**，提出在用户态进行分区调度，内核态进行进程调度（分区切换仍然由内核负责）
    - **2013**，Han：**Linux 653**，基于**Linux**的**CFS**，每个分区构建一棵调度**RB-Tree**
  - 总结：
    - 大都通过修改内核调度器实现，部分在用户态实现。

国内外研究现状

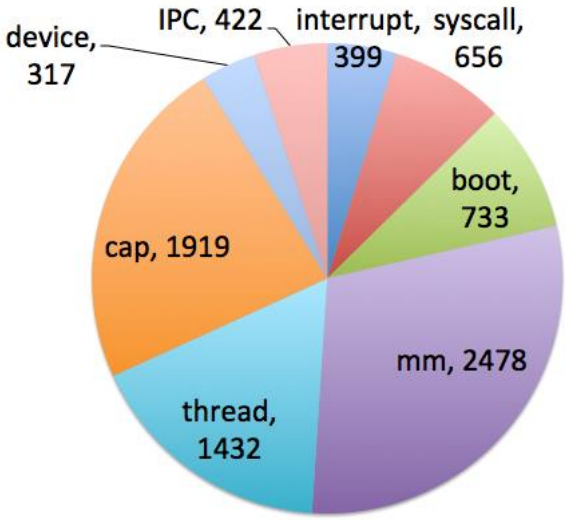
- 分区间通信
  - 研究现状
    - **2013**，Han：**Linux 653**，基于**Linux**的消息队列实现
    - **2015**，**Asberg**等人：基于**L4**的**IPC**接口实现
  - 总结
    - 在内核实现**IPC**，并通过系统调用的形式提供接口
    - 消息缓冲一般由内核提供。

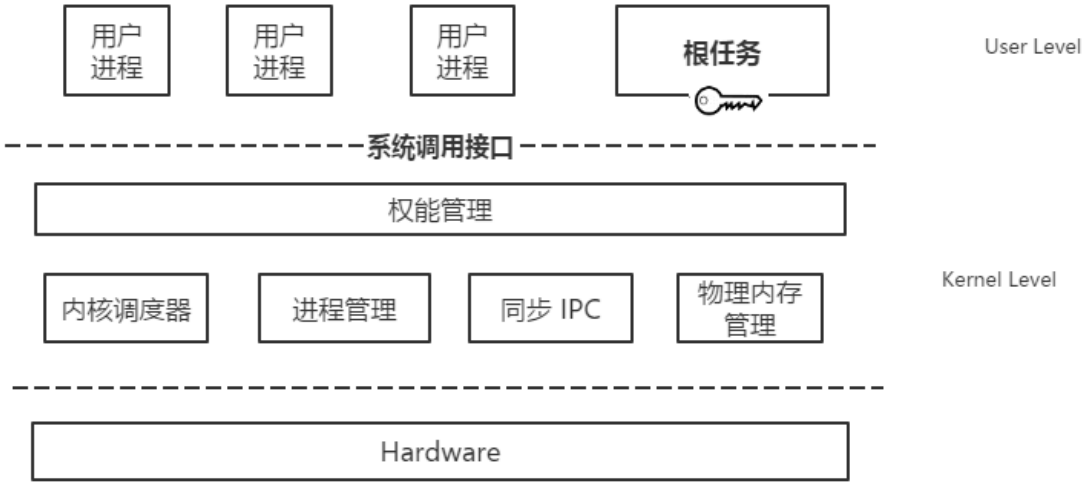
研究目标与内容

- 研究目标
  - 基于seL4微内核研究分区隔离的方法，实现对ARINC 653标准的支持，并实现原型操作系统
- 研究内容
  - 分区在seL4的表示与管理
    - » 分区在seL4中由哪一种实体表示
    - » 如何创建分区和进程、管理分区与进程的关系等
  - 基于seL4实现分区调度器
  - 基于seL4提供分区间通信

seL4的基础设施

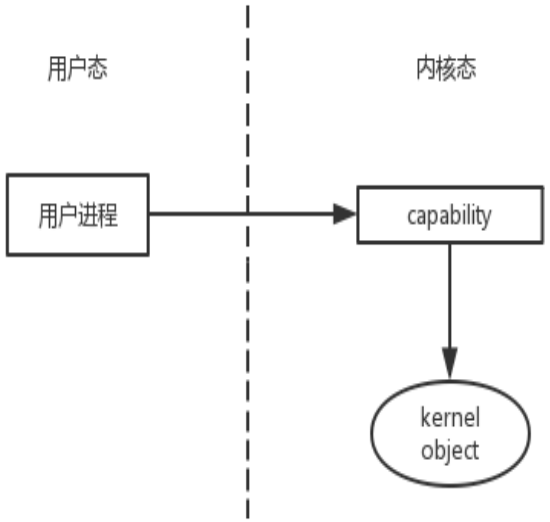
- seL4提供了什么？
  - 基本的进程模型
    - » 简单的两级调度
    - » 进程地址空间的隔离
  - IPC机制
    - » 允许进程之间进行通信
  - 权能管理
    - » 上述内核服务都通过权能提供
  - 根任务——掌握所有权能



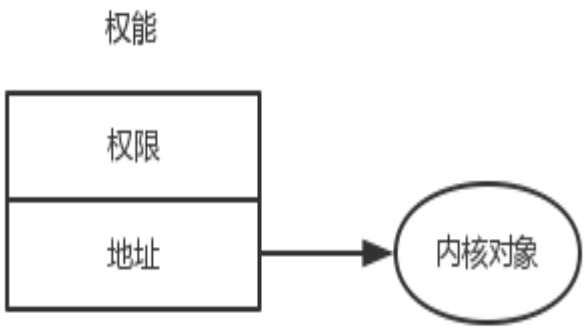


权能分析（1）

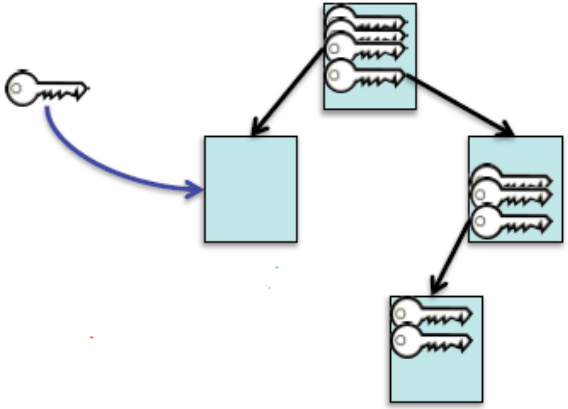
- 权能的作用
  - 是访问内核对象的钥匙
    - 一个进程要访问一个内核对象（比如TCB），这个进程须拥有相应的权能。
    - 而且权能具备访问权限，才能进行访问。
    - err = method(cap, args);



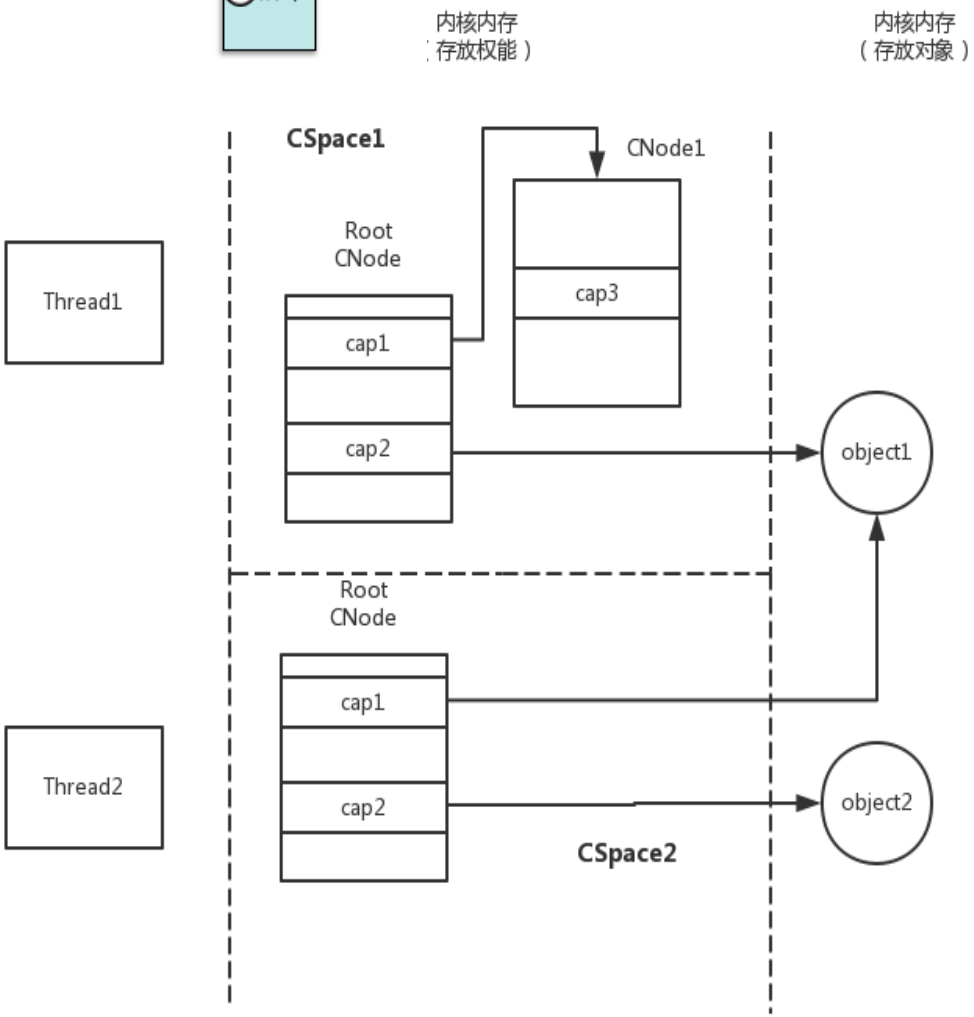
- 权能（capability）的结构
  - 权能就是一个二元组
    - 地址：指向一个对象，一个权能一定指向仅一个对象
    - 权限：能够在这个对象上做什么事情（比如读、写等）



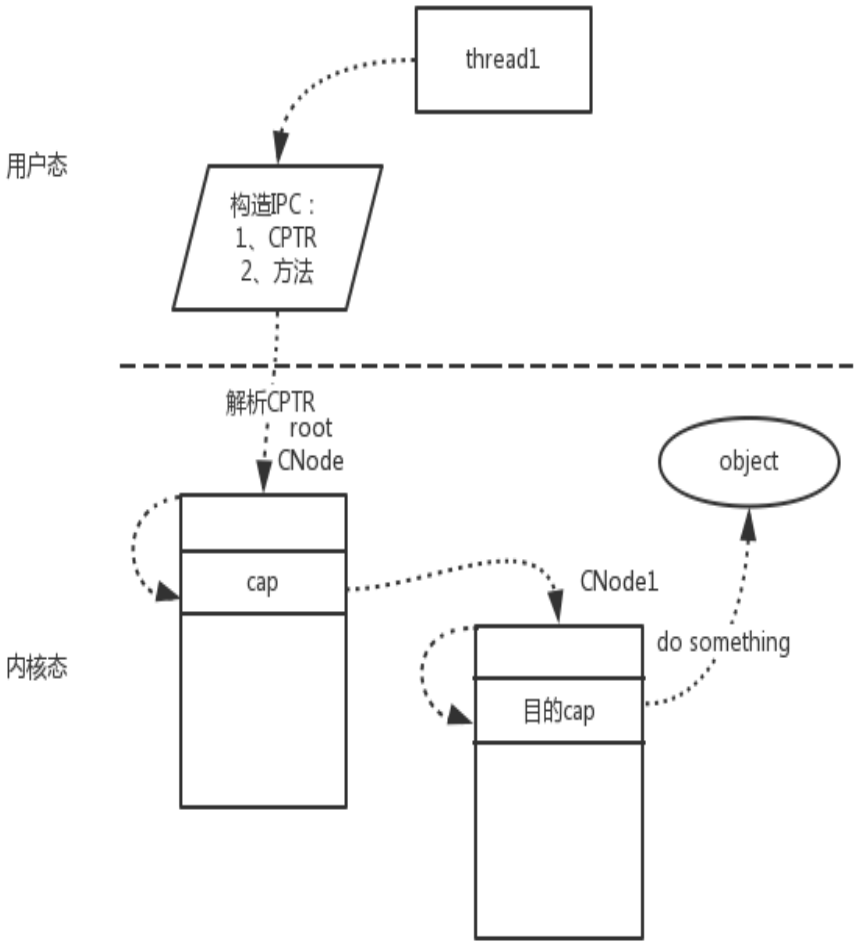
权能分析（2）



- 权能的存储
  - 权能位于Cnode中
    - Cnode可以理解为是一个权能数组。
    - Cnode位于内核空间，由内核管理
  - 权能空间Cspace
    - 一个进程有很多Cnode
    - 一个进程所有权能，构成了Cspace
    - Cnode往往成树/图状结构



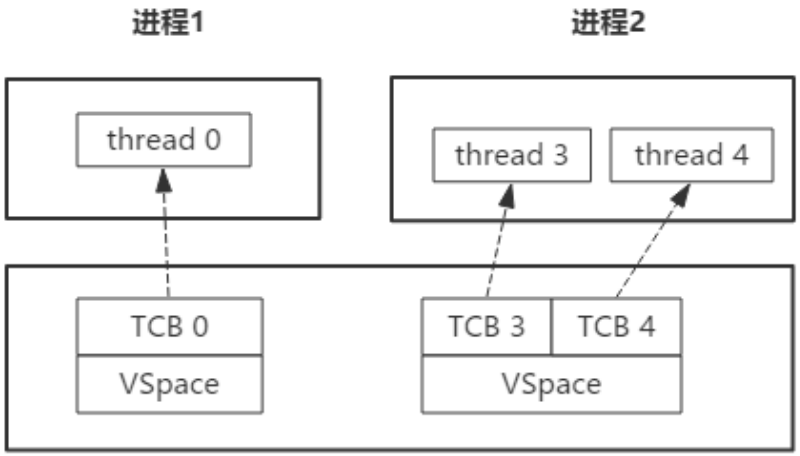
权能管理（3）



- 权能的使用
  - 权能地址（CPTR）
    - CPTR就是一个索引
    - 上层程序不能直接访问权能，只有一个CPTR
  - 利用CPTR访问对象
    - 发送一个IPC给内核，捎带上CPTR
    - 内核通过CPTR找到相应的Cnode
    - 在Cnode中找到相应的权能
    - 检查权能是不是有访问权限，执行访问操作

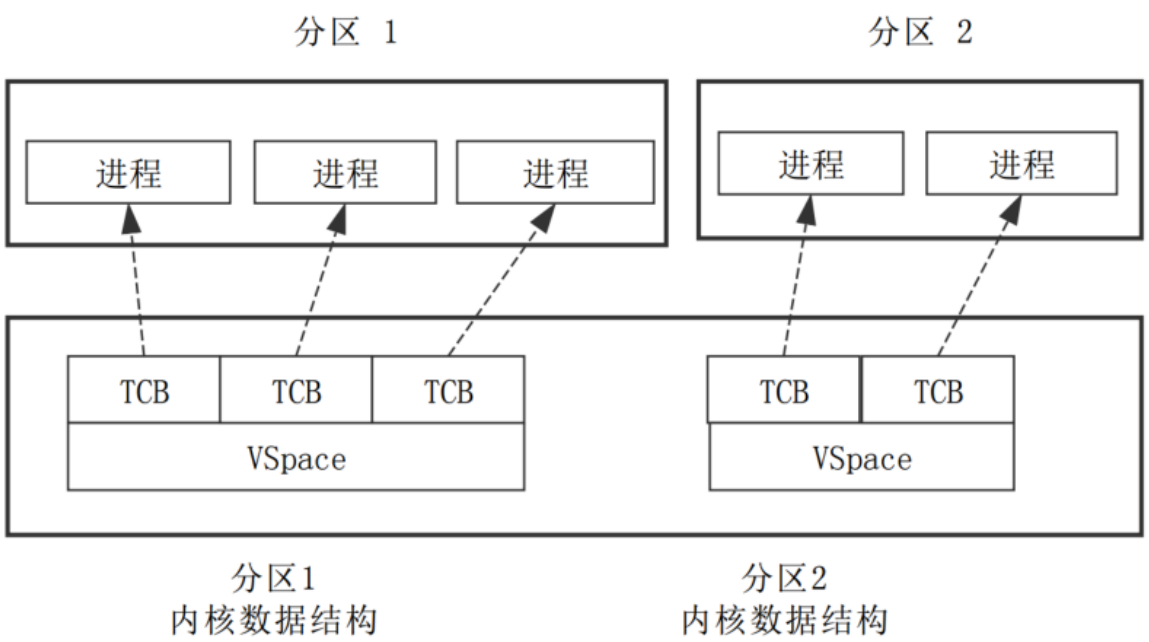
研究点1：分区在seL4中的表示

- seL4提供对任务和地址空间的抽象
  - 任务：TCB
  - 地址空间：VSpace
  - 线程——多个TCB对应一个VSpace
  - 进程——一个TCB对应一个VSpace
- seL4内核只创建一个进程——根任务
  - 其他进程或线程由根任务创建



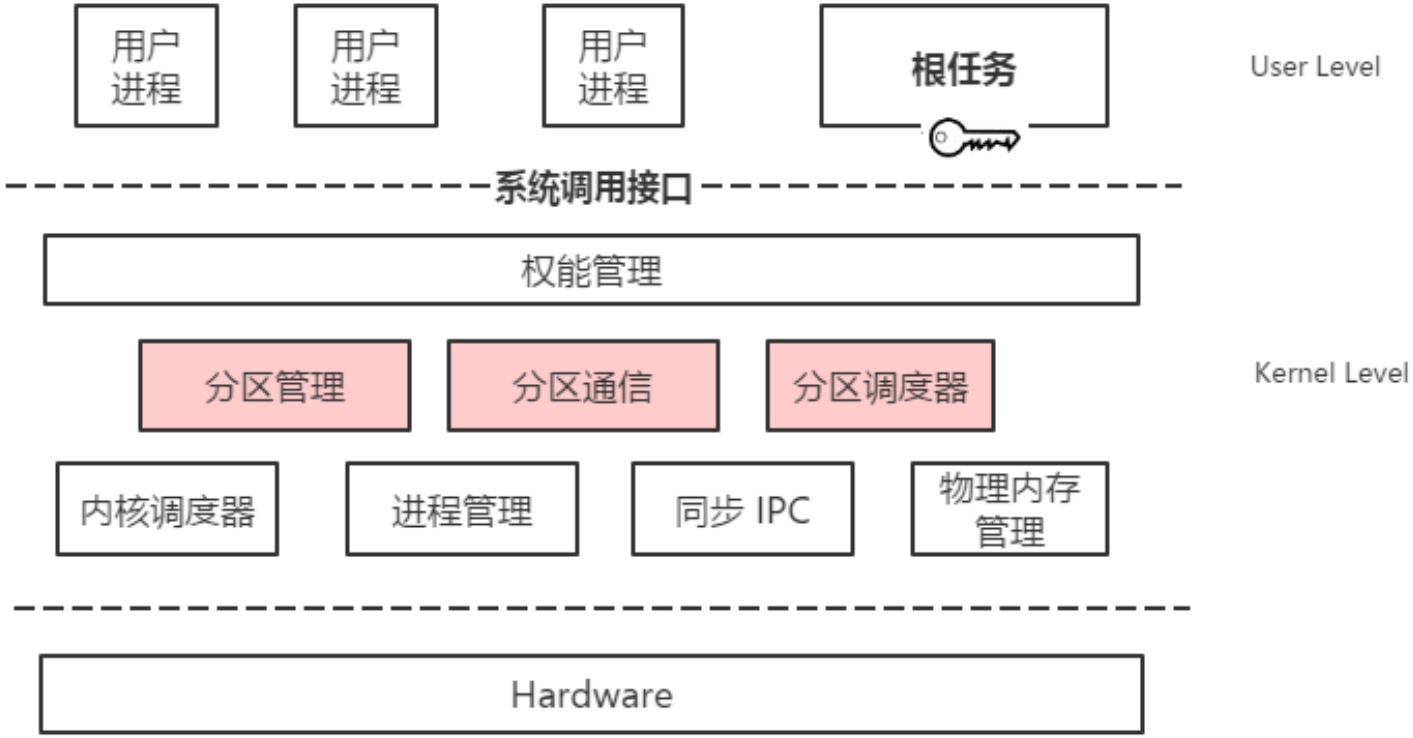
研究点1：分区在seL4中的表示

- 借鉴Linux 653的做法
  - ARINC 653分区 = seL4进程
  - ARINC 653进程 = seL4线程



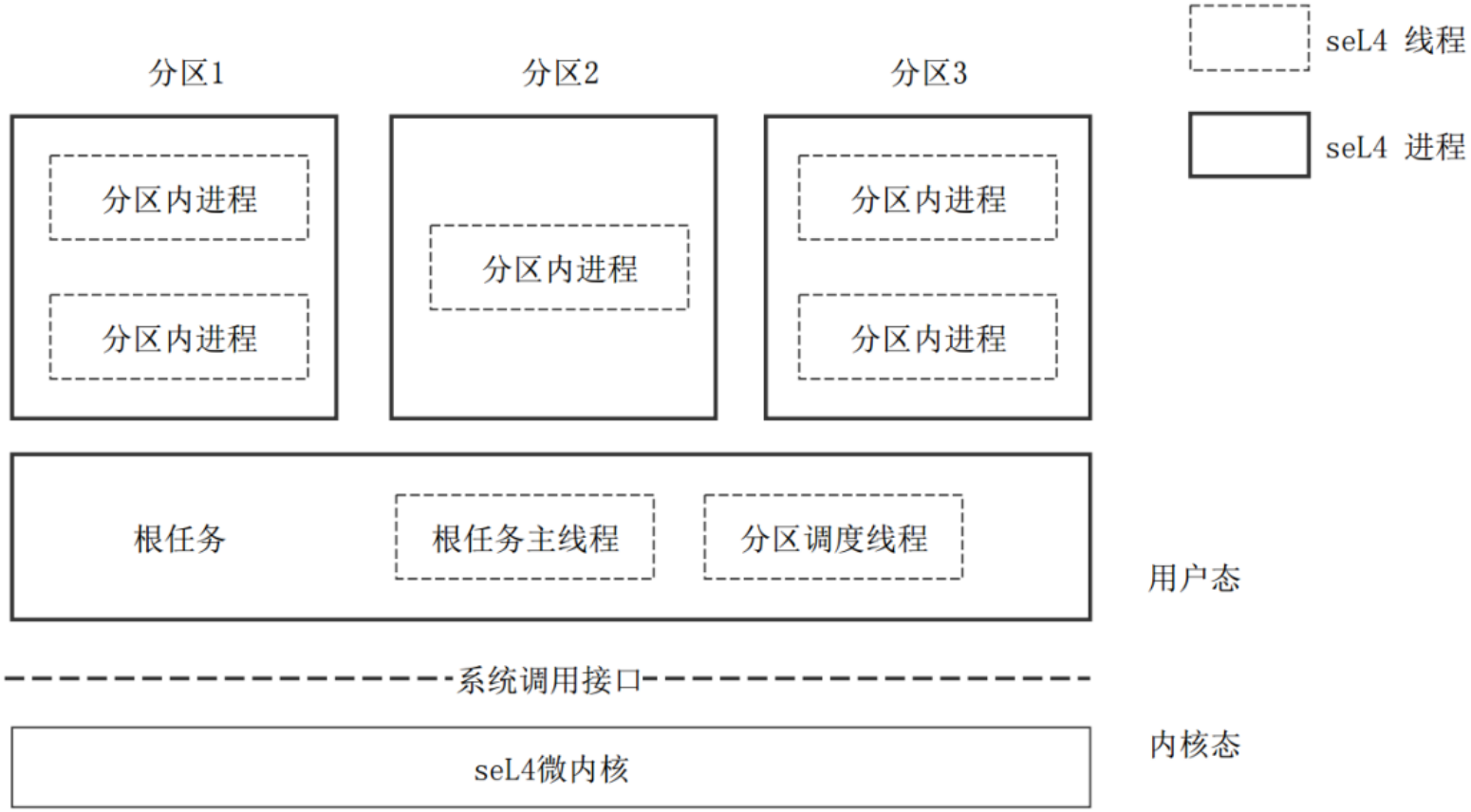
研究点2：分区管理方法

- 方案1
  - 在内核加入【分区管理】模块
  - 发现内核调度的问题
  - 破坏内核的完整性



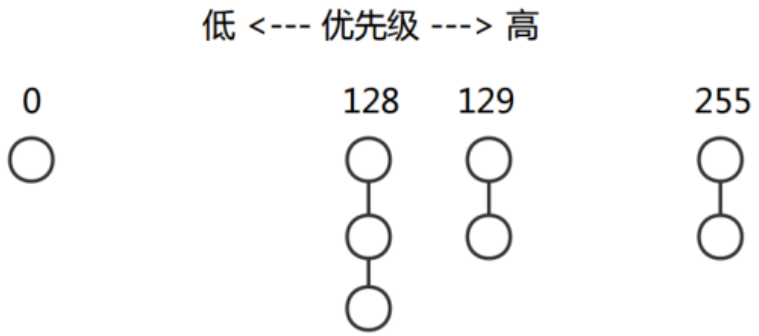
研究点2：分区管理方法

- 不能修改内核代码——在用户态实现
- 由根任务创建分区、实施分区管理



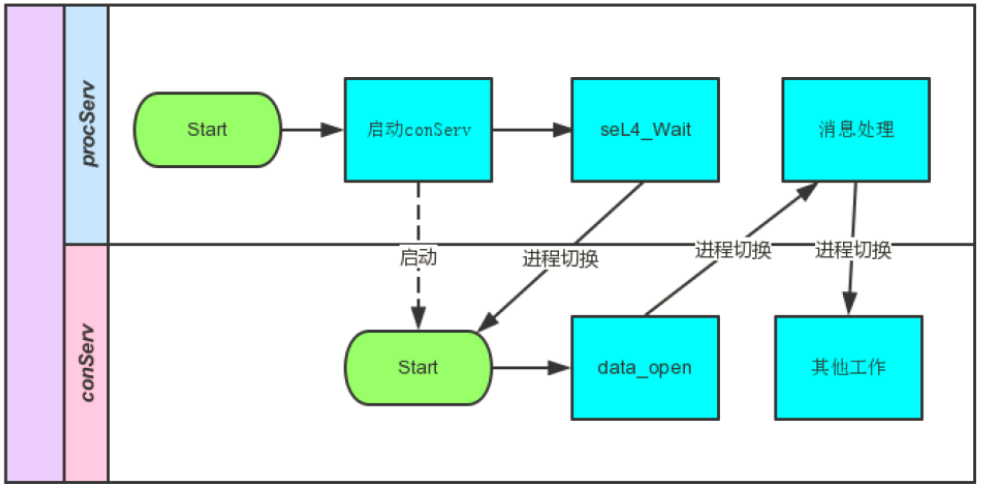
研究点3：分区调度器研究

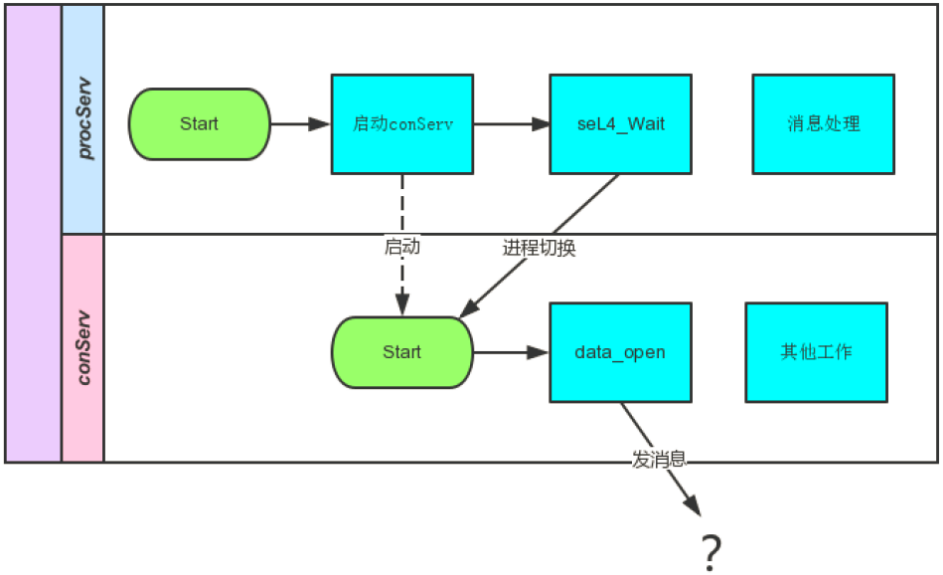
- seL4的调度模型
  - 内核中维护基于优先级的调度队列



  - 调度算法：基于优先级的可抢占调度
  - 调度对象：线程控制块TCB
  - 调度时机：中断处理/系统调用返回用户态之前
  - 两级调度：域和线程两级调度

研究点3：分区调度器研究





研究点3：分区调度器研究

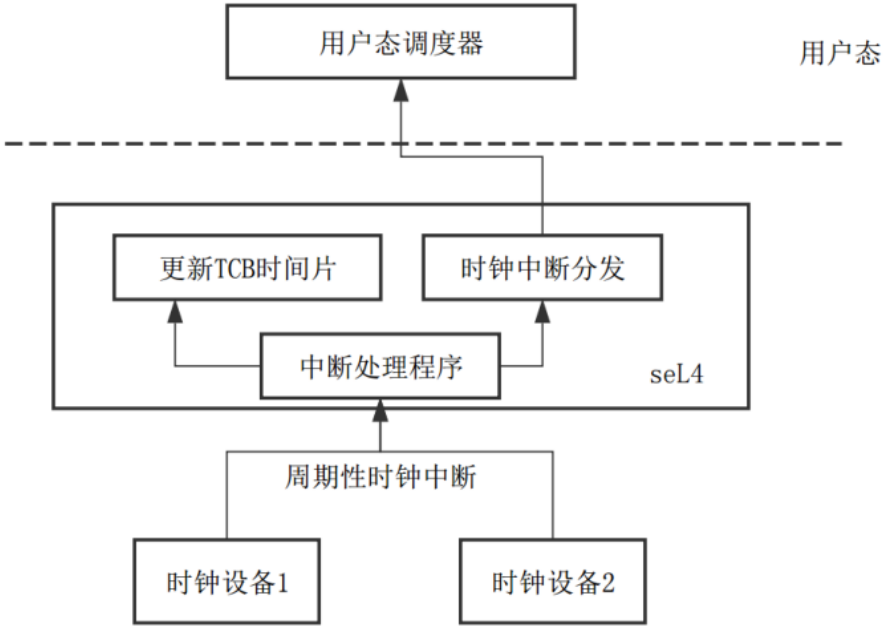
- thread.c中 possibleSwitchTo：

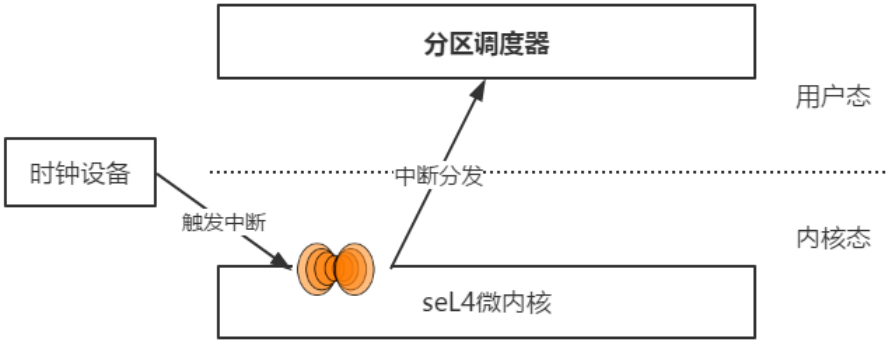
```
 if (CONFIG_NUM_DOMAINS > 1)
{
  dom_t curDom = ksCurDomain;
 dom_t targetDom = target->tcbDomain;

 if (targetDom != curDom) {
     tcbSchedEnqueue(target);
 }
}
else
                .....
```

研究点3：分区调度器研究-用户态调度

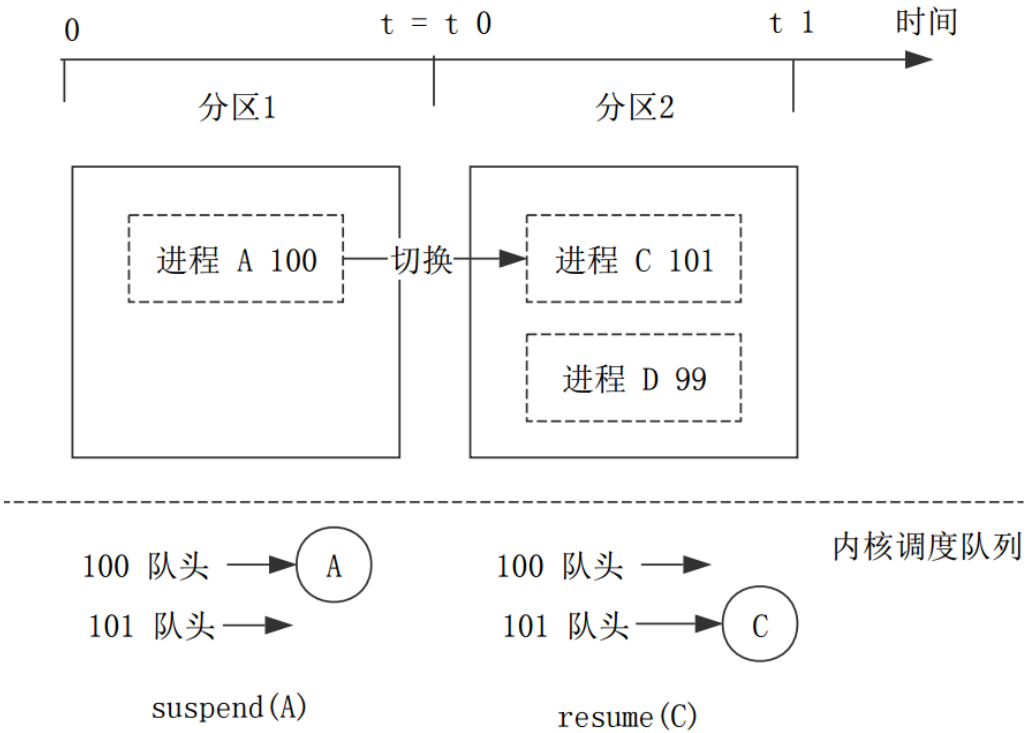
- 问题1：调度时机
  - 用户进程如何能够周期性启动?
  - 解决方案：利用seL4提供的中断分发机制
    - » 两个定时器，一个内核用，一个用户态用
    - » 图中小喇叭：seL4中的端点，可与特定的中断绑定
    - » 每次时钟中断都由调度器处理，可实现周期性启动

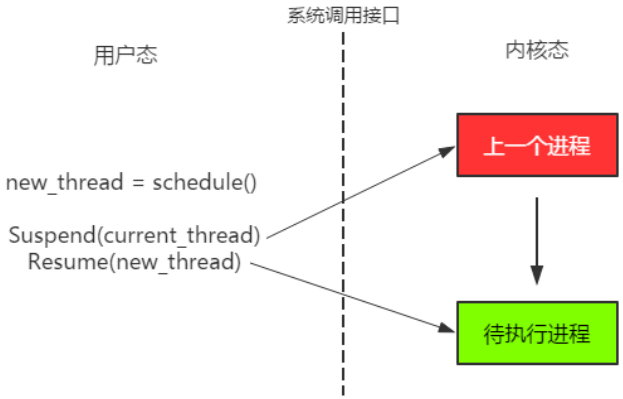




研究点3：分区调度器研究

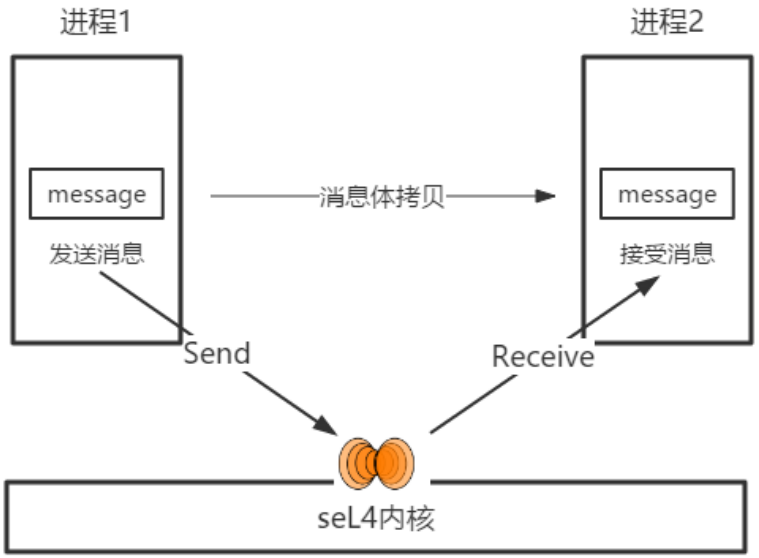
- 问题2：如何与内核调度器配合
  - 为了保持seL4内核代码不变，不得不保留内核调度器
  - 调用suspend和resume接口干预内核调度队列
  - 调度示例





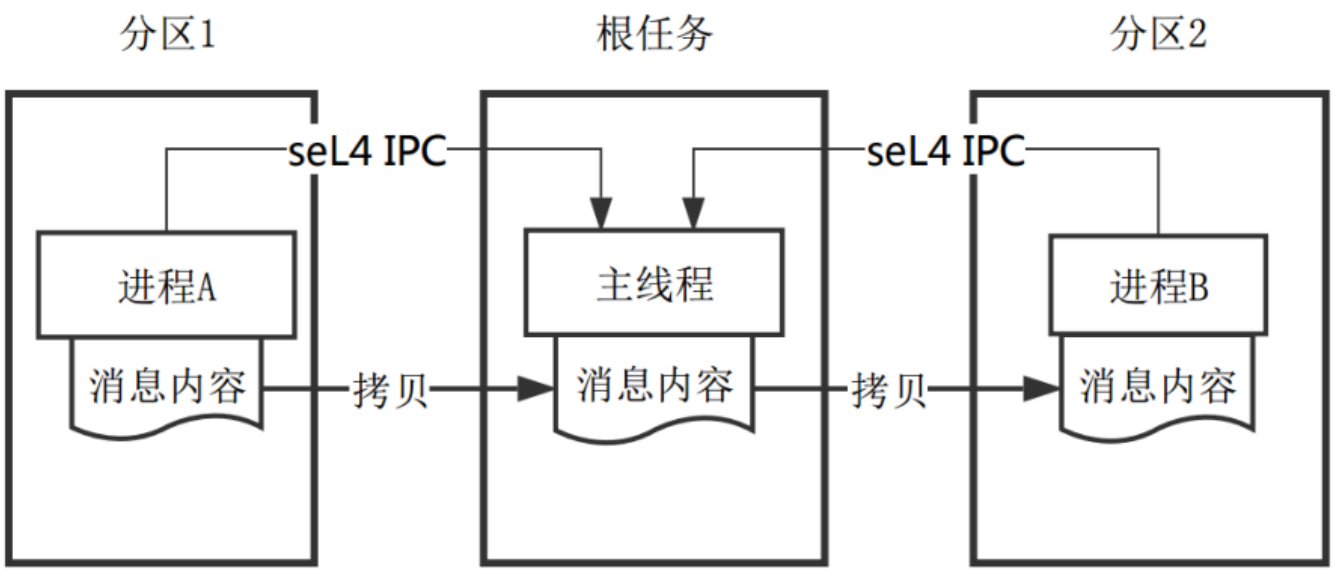
研究点4：分区间通信方法

- seL4的IPC（进程间通信）模型
  - 基于端点的同步IPC
    - » 端点：就像一个邮箱，可以向其他发消息，也可以从其中收消息。
    - » 端点也有所有人，只有有权限的进程才能使用端点。
  - 内核不提供消息缓冲区
  - IPC系统调用接口
    - » Send
    - » Receive
    - » Call
    - » Reply等
  - 与ARINC 653通信模型矛盾



研究点4：分区间通信方法

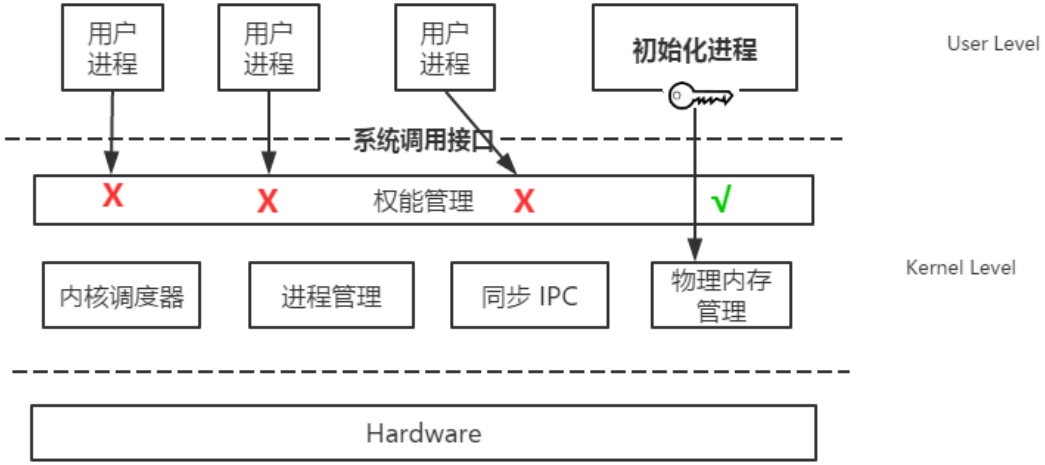
- 解决办法：由一个进程专门负责消息发送，并提供消息缓冲。
  - 分析：
    - » 优点是避免内核提供缓冲区，提高了安全性
    - » 缺点：需要执行两次消息拷贝，降低了效率



研究点5：APEX接口研究

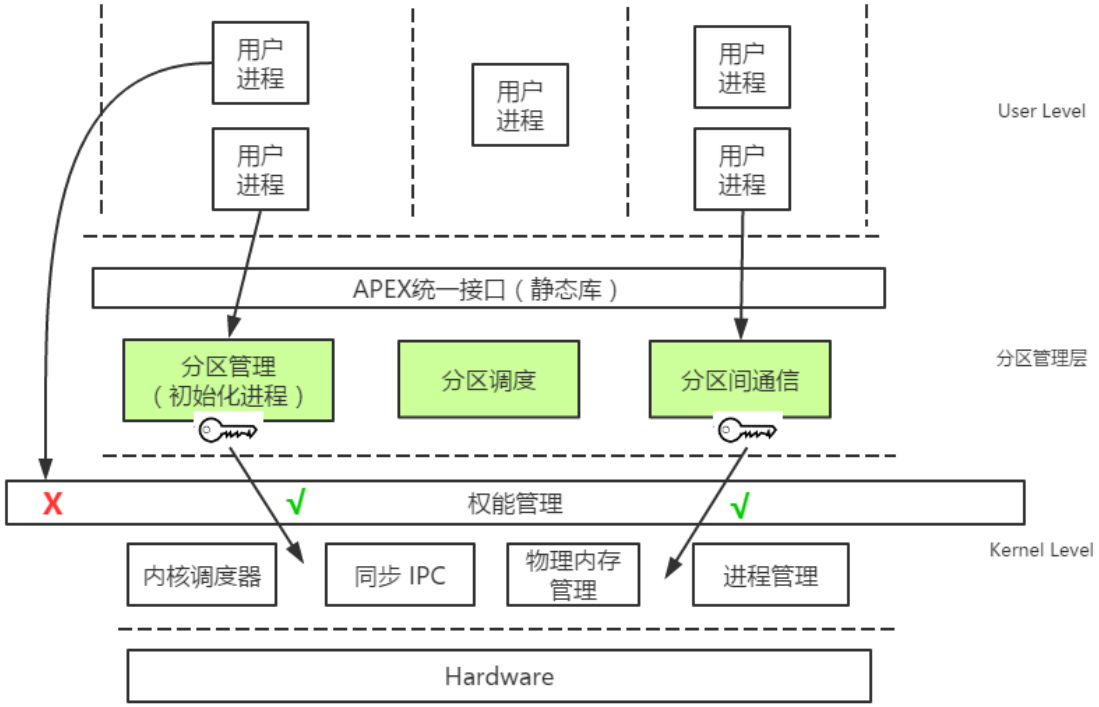
- seL4的系统调用
  - 调用系统调用需要提供权能（capability）
  - 举例：
    - » 权能解释
      - 访问内核的“钥匙”
      - seL4内核启动后把所有权能交给第一个进程（初始化进程）

```
static inline int
seL4_TCB_SetPriority(seL4_TCB service, seL4_Uint8 priority)
```



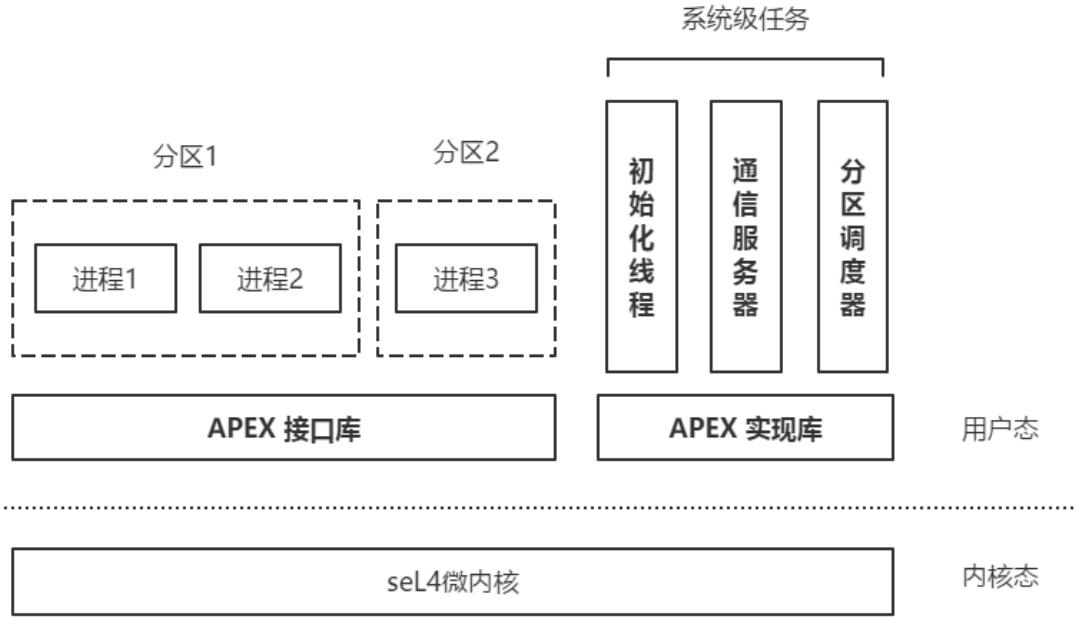
研究点5：APEX接口研究

- 问题5：如何提供操作系统统一接口，并避免普通程序调用系统调用？
  - 普通进程通过IPC请求根任务执行统一接口
  - 由根任务实现统一接口
  - 根任务提供端点，允许普通进程发送请求
  - 根任务调用seL4系统调用



原型系统实现

- 在内核与用户态之间新加一个层，用于实施分区隔离
  - 分区管理器：初始化进程+APEX实现
  - 分区调度器：在用户态实施分区调度
  - 通信服务器：在用户态实施分区间通信+APEX实现
  - APEX接口库



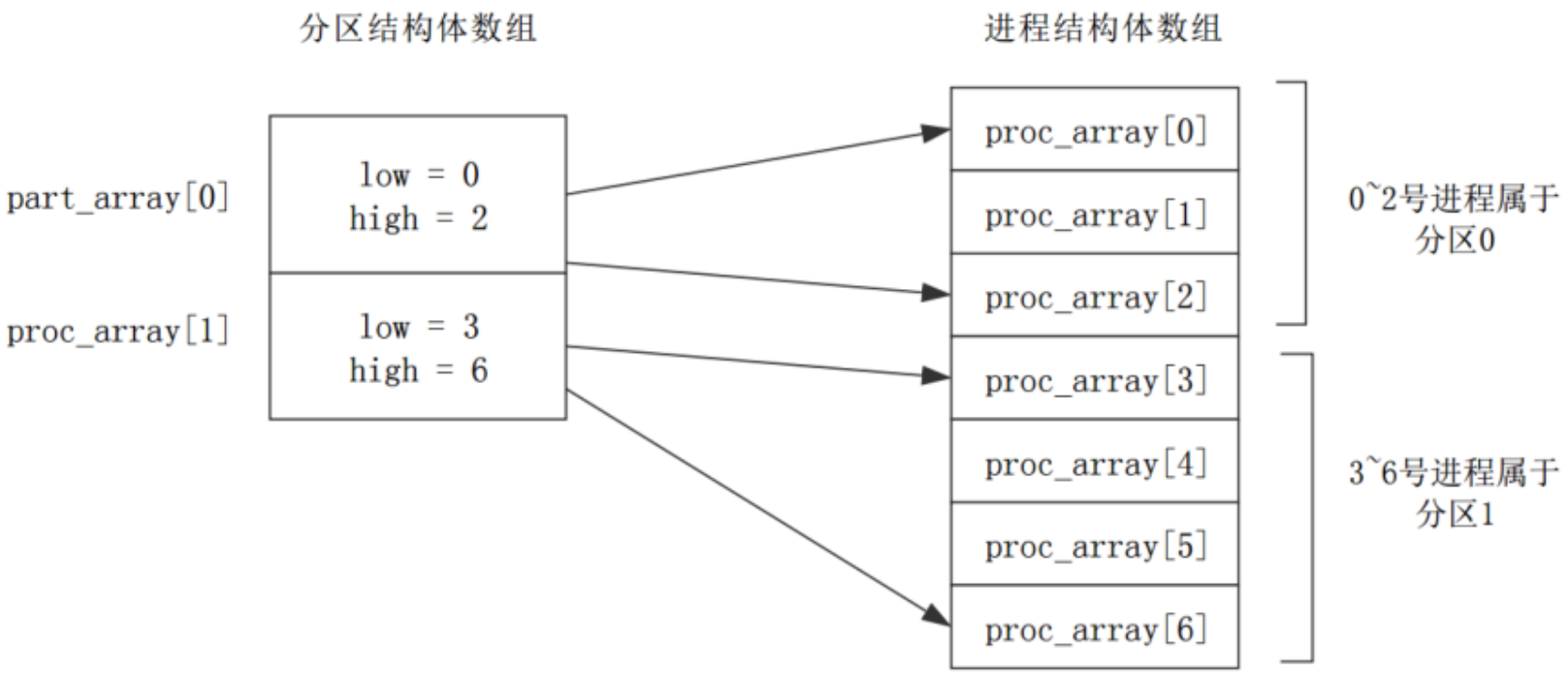
原型系统实现

- 系统配置信息
  - .h文件定义

配置类型	配置项	说明	配置值示例
基本配置	PARTITION_NB	分区数量	2
	PROCESS_NB	分区内进程总数量	5
	PARTITION_PROCS	各个分区的进程数量	{2,3}
	STACK_SIZE	进程栈大小	{4096} * 5
	MEM_SIZE	分区内存大小	{100 * 4096} * 5
	PART_NAME	分区镜像文件名称	{"part1.elf", "part2.elf"}
调度配置	SCHED_SLOTS	分区时间窗大小（毫秒）	{2000,3000}
	MAJOR_SLOT	主时间窗大小（毫秒）	{5000}
端口配置	PORT_NB_BY_PART	各个分区的端口 ID	{0,1}
	PORT_NAME	端口名称	{"in","out"}
	PORT_ID	端口的 ID	{0,1}
	PORT_NB_DESTS	端口目的地数量	{1,0}
	PORT_DESTS	各个端口的目的地	{{1,},{}}
	PORT_KIND	端口类型	{"SAMPLING", "SAMPLING"}

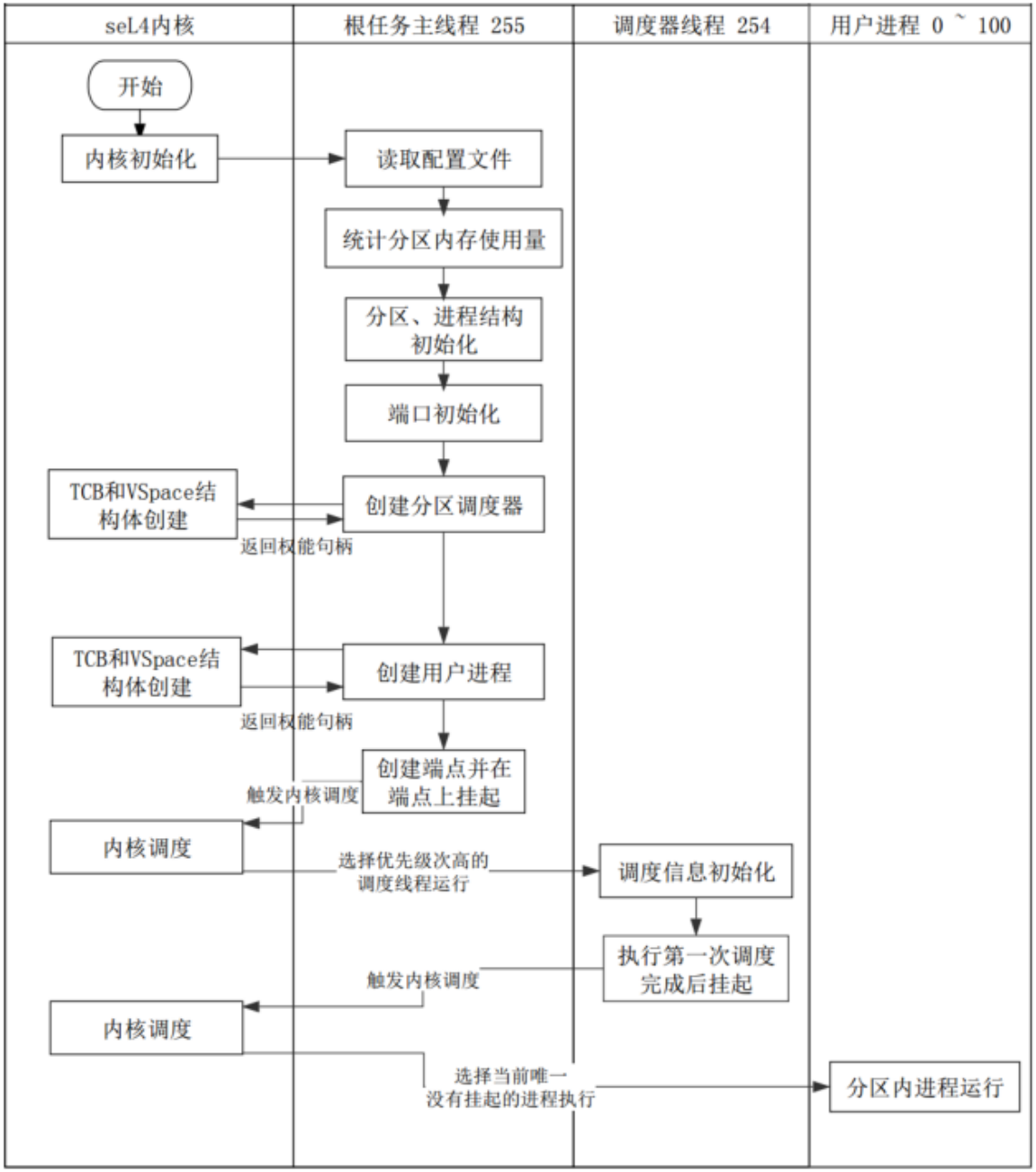
原型系统实现

- 数据结构设计
  - 分区和进程结构体以全局变量表示
  - 分区中[low,high]范围内的进程为分区内进程



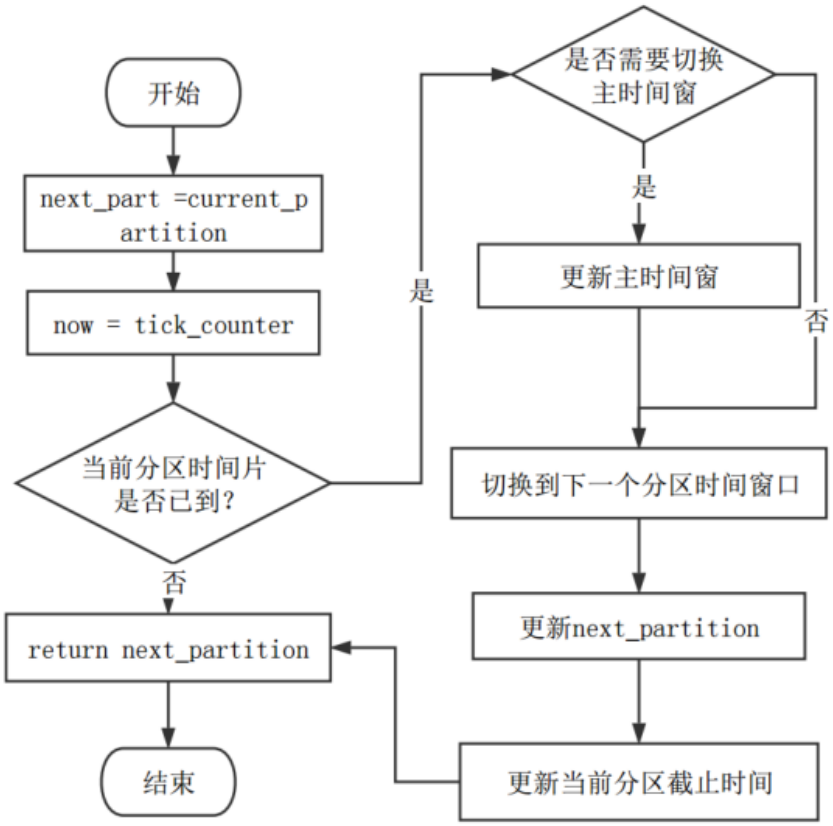
原型系统实现

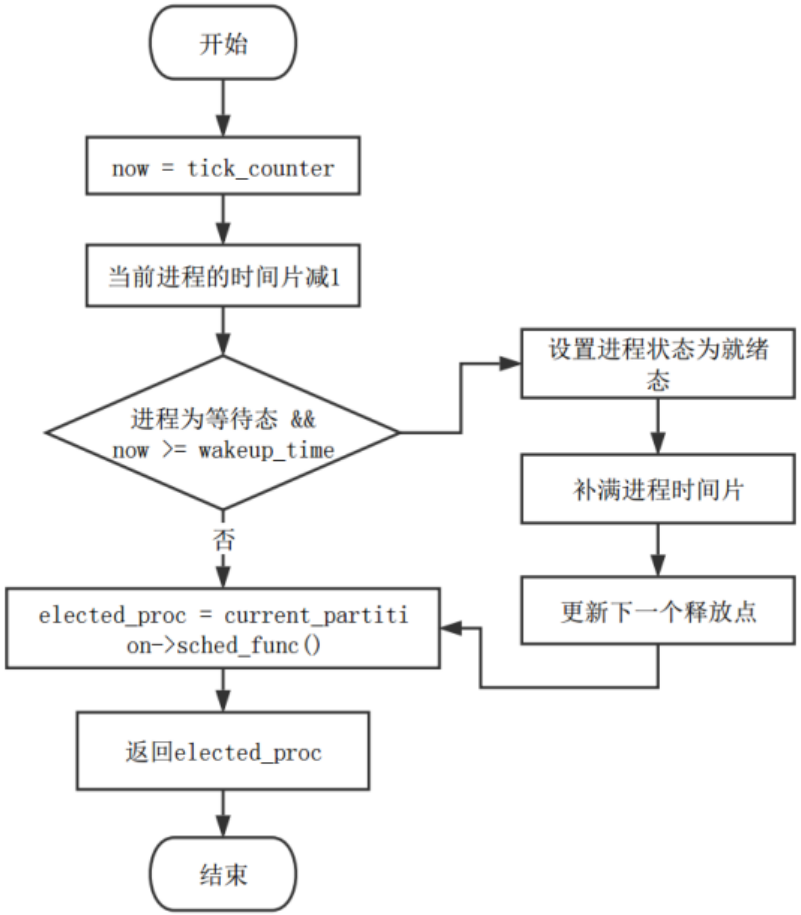
- 系统初始化



原型系统实现

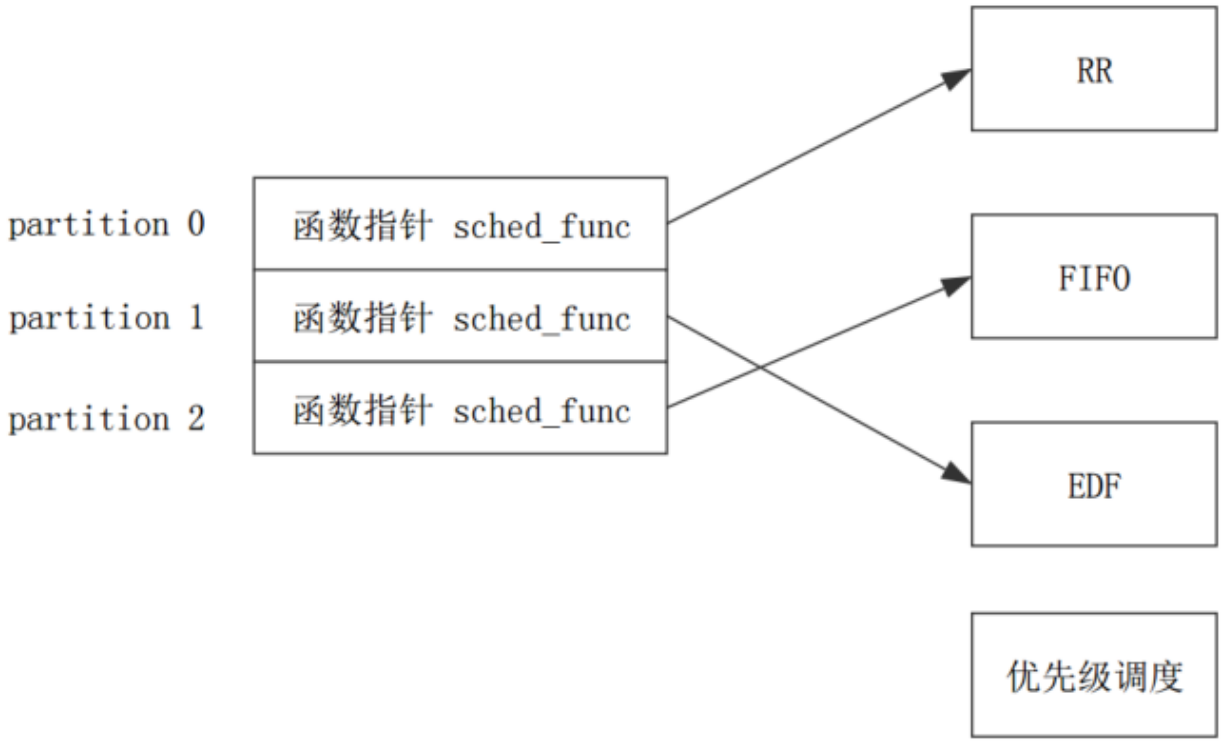
- schedule的实现
  - elect_partition选择分区
  - elect_process选择进程





原型系统实现

- schedule的实现
  - 分区内进程调度器的可配置性
  - 不同分区可以选择不同算法



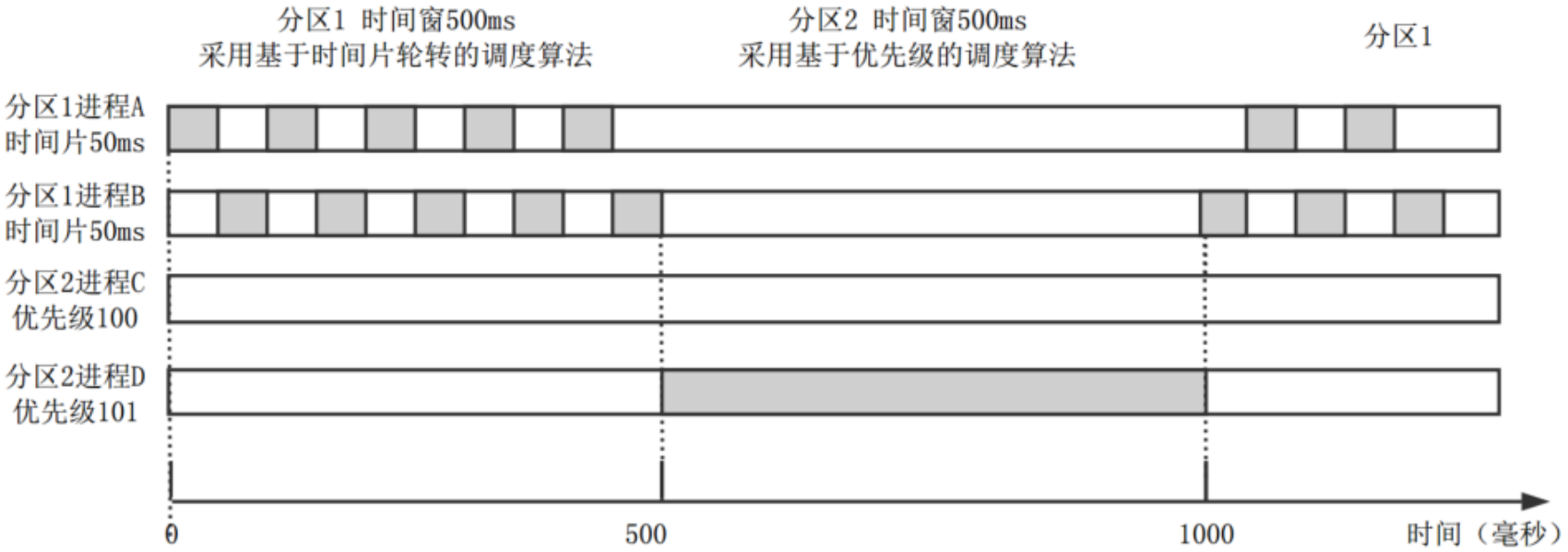
系统评测

- 测试方案：分5个部分进行

测试类型	测试内容	测试方案
功能测试	APEX 符合性测试	使用文献 [43] 给出的标准测试用例进行测试
	时间隔离测试	配置两个分区，其中同时包含周期性任务和非周期性任务，跟踪观察执行结果。
	通信隔离测试	配置三个分区，测试没有通信信道的分区之间能否通过端口获取消息。
性能测试	APEX 接口性能测试	针对每个 APEX 接口的开销进行测试，并作对比分析
	分区调度性能测试	测试分区调度与进程调度的的开销时间，并作对比分析

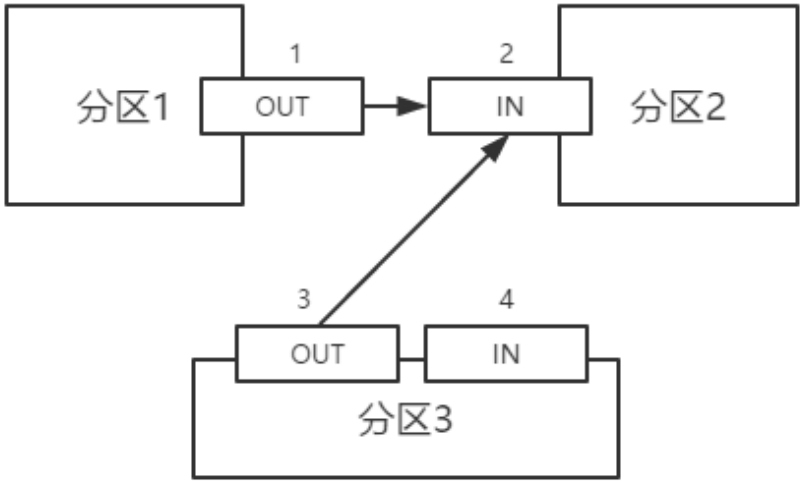
系统评测

- 功能测试
  - ARINC 653标准定义的测试用例
  - 分区隔离效果



系统评测

- 功能测试
  - 通信隔离测试
    - 通信权限配置



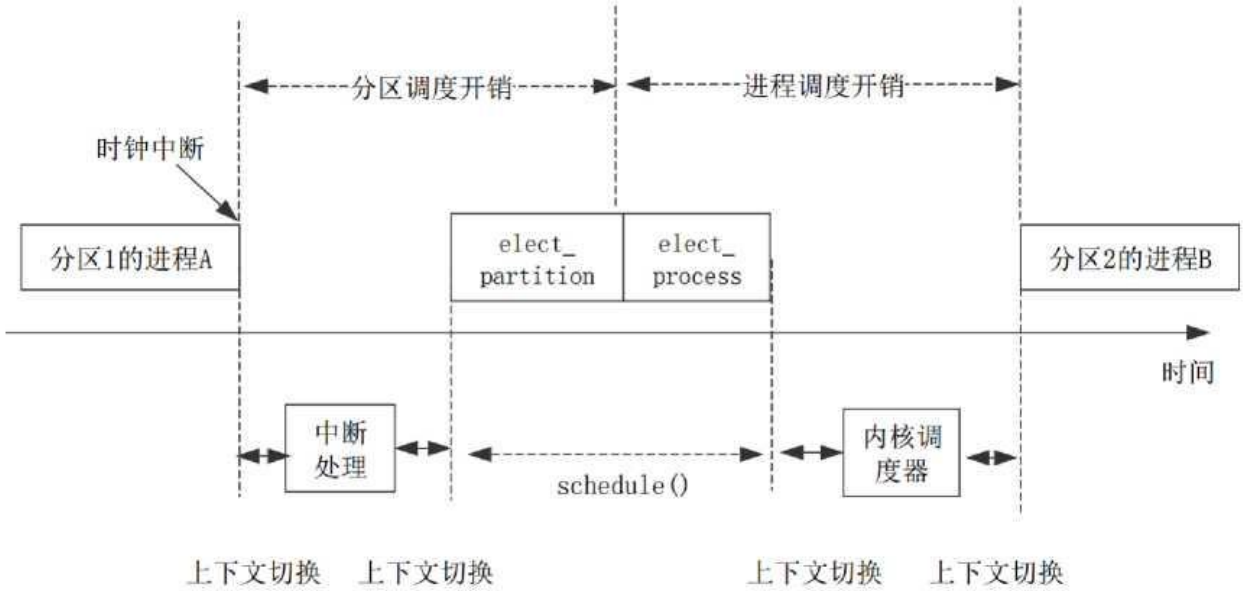
系统评测

- 性能测试
  - APEX接口性能
  - 可以在用户态完成的性能好，涉及内核的性能差

APEX 接口名称	Linux 653 的测试结果[7]	本系统测试结果	相对性能提升
GET_PARTITION_STATUS	5.2	4.0	23%
SET_PARTITION_MODE	2.9	11.5	-390%
GET_PROCESS_ID	0.8	0.4	50%
GET_PROCESS_STATUS	3.0	2.7	10%
CREATE_PROCESS	9.3	21.9	-135%
SUSPEND	3.4	3.1	8%
RESUME	2.8	3.0	-7%
GET_MY_ID	1.2	0.4	66.7%

系统评测

- 性能测试
  - 调度器性能



  - 分区、进程调度的开销与Linux的分区系统比较

测试项	Linux 653 用户态调度器[7]	Linux 653 内核态调度器[7]	Asberg 的 L4 分区调度器[44]	本系统调度器
分区调度开销	44.83	5.2	213	7.9
进程调度开销	42.5	40.4	-	40.0

总结

- 基于capability的空间分区
  - 效果很好
- 用户态分区调度
  - 修改调度策略、更换调度器更为灵活
  - 保持内核完整性
  - 性能可接受、但由于通信开销，有些操作耗时
- 基于微内核的系统设计问题
  - 通过IPC来构造，上层系统设计复杂
  - 8700行内核，13000行上层库和更多应用程序

谢谢！

敬请大家指正！