

ZVM：基于Zephyr RTOS的 嵌入式实时虚拟机

主讲人：谢国琪

2023-01-31



湖南大学
HUNAN UNIVERSITY

嵌入式与网络计算湖南省重点实验室

CONTENTS

目录

01

01. 项目背景

02. ZVM设计

03. ZVM演示

04. ZVM规划

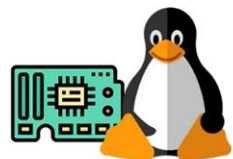
项目背景

高性能嵌入式应用场景：万物互联

- 嵌入式设备性能提升(MCU->SoC、ARM 64)
- 嵌入式系统功能丰富 (富功能+实时控制)
- 嵌入式系统开发度增加 (智能物联网时代)



面向新一代**万物互联**
的**分布式嵌入式场景**



项目背景

嵌入式OS对富功能与硬实时的混合关键部署要求

混合关键部署需求

- ◆ 当前分布式嵌入式场景存在**富功能**（数据可视化、场景可视化）与**硬实时**（精准控制）的双重需求



在单个底座上打造**双子星**
(富功能OS, 实时OS) 互
助运转的混合关键部署模式

双子星联动

富功能OS

- 智能座舱
- 数据可视化
- 场景可视化

Linux宏内核

Linux: 功能强大
但实时性不足

实时控制OS

- 智能驾驶
- 底盘控制
- 动力控制

实时微内核

QNX: 以闭源的方式
来形成技术壁垒



满足关键场景对智能化、实时性及安全性的多重要求！

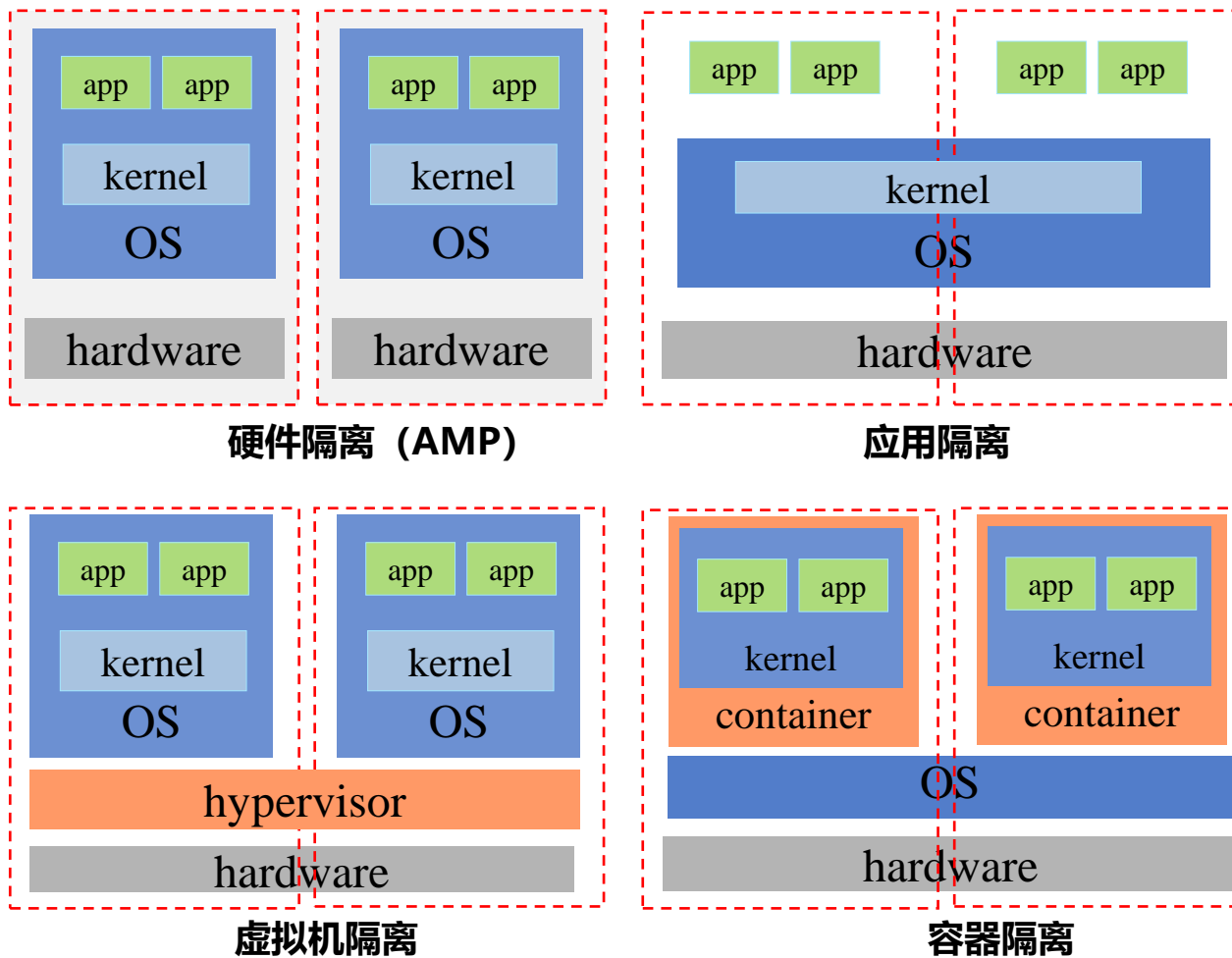
项目背景

安全隔离保护机制

◆ 解耦双子星故障联动，保证双子星彼此隔离与保护

- ① 硬件隔离：在硬件上直接为OS隔离资源
- ② 应用隔离：在操作系统级别实现应用的隔离
- ③ 虚拟机隔离：在虚拟化层实现应用隔离
- ④ 容器隔离：通过容器技术实现应用的隔离

**本项目使用虚拟化技术
(Hypervisor) 实现隔离**



项目背景

两类Hypervisor比较

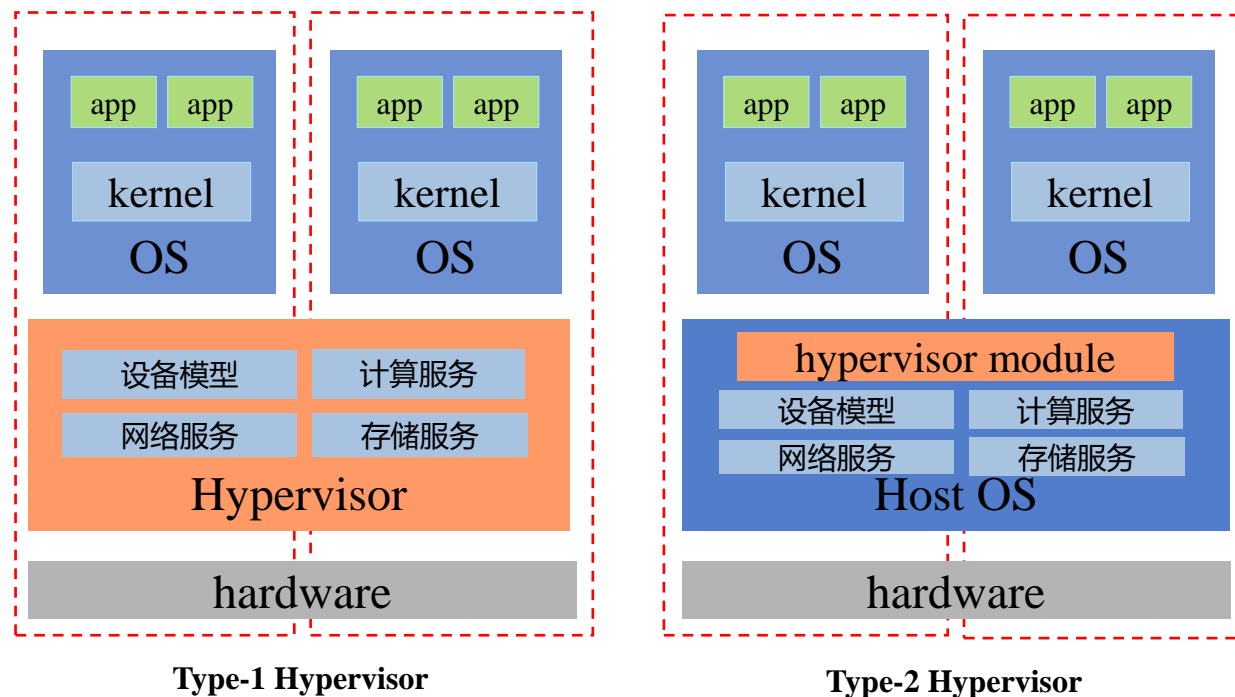
- ◆ 根据是否需要主机 (Host) 操作系统, 将其分为Type-1型和type-2型虚拟机

- ◆ Type-1特点

Hypervisor直接管理硬件: 虚拟机性能较好, 但适配性较差。(QNX Hypervisor, XEN)

- ◆ Type-2特点

Host直接硬件, Hypervisor作为Host的一部分: 虚拟机性能较差, 但适配性好。(KVM)



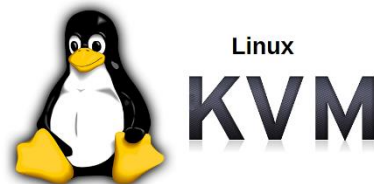
本项目偏向Type-2型虚拟机

项目动机

常见Hypervisor方案

➤ 使用广泛的开源方案

主要包括Linux KVM, XEN等为代表的开源虚拟化方案



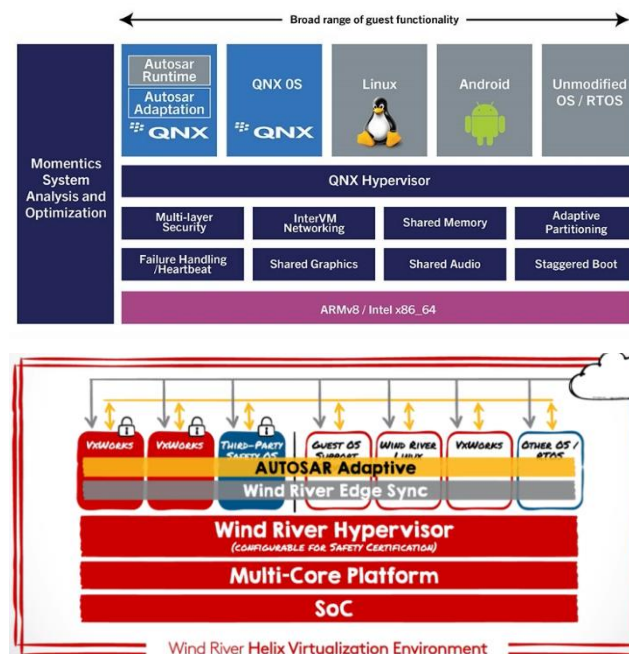
➤ 商业虚拟化方案

主要包括QNX Hypervisor, Wind River Helix及PikOS等闭源解决方案



➤ 面向嵌入式的虚拟化方案

主要为Type-1型的hypervisor平台: 如QNX Hypervisor, ACRN, Xvisor

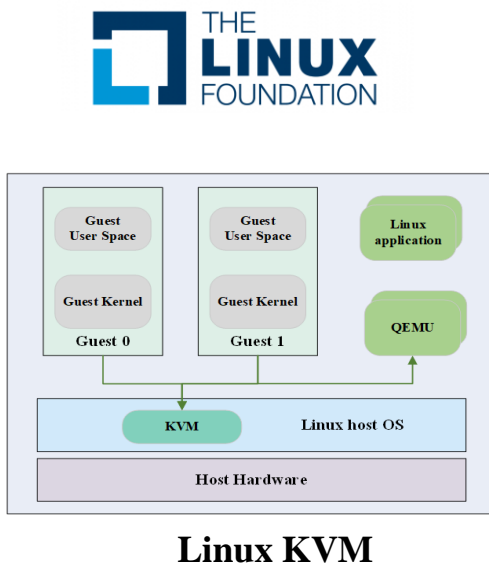


项目目标

典型Hypervisor优缺点

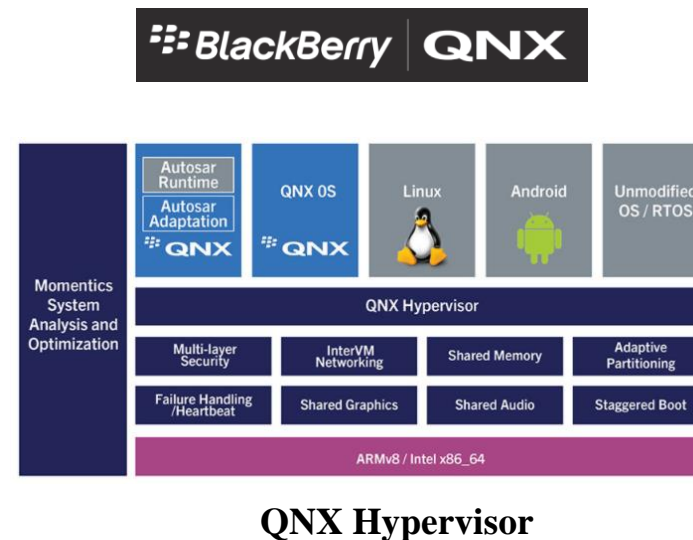
Linux/KVM

- **优势**
 - 开源，**设备适配性好**，开发人员众多
- **劣势**
 - Type-2型**性能开销较大**，**实时性不够**



QNX Hypervisor

- **优势**
 - Type-1型**性能较好**，**实时性好**
- **劣势**
 - 闭源方案，**缺少技术参考**，**设备支持相对较少**



针对嵌入式系统，开发一款**开源、硬实时、低开销、适配性好**的Hypervisor



Zephyr RTOS简介

- Zephyr是一款针对资源受限设备优化的最佳小型、可扩展的实时操作系统（RTOS）：
 - ① Zephyr项目是采用Apache 2.0协议许可，由Linux基金会托管的开源协作项目
 - ② Zephyr历史虽短但起点很高，最初的代码来自风河Rocket，其体系架构完整，中间件丰富
 - ③ 在安全设计方面有缜密的考量，Zephyr选择支持IEC61508，计划支持汽车安全标准ISO26262

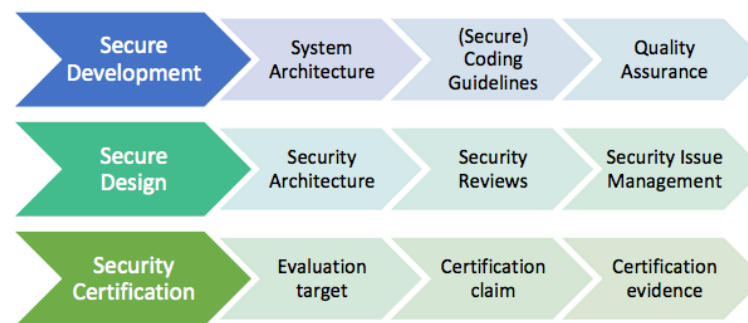




本项目选用的实时操作系统是Zephyr，它是一款开源的、面向资源受限的物联网设备的实时操作系统

Zephyr架构特性

- 与 μ COS、FreeRTOS、Contiki、Mbed OS等RTOS相比，Zephyr具有以下优势：
 - ① **支持多种硬件架构。** Zephyr能够同时支持ARM、Xtensa、Native POSIX、Intel x86、ARC(DSP 内核)、NIOS II(FPGA 软核)以及 RISC-V这些架构 (NXP、Broadcom、ST等350+板卡支持)
 - ② **Zephyr是一个产品级应用。** 可提供长期支持版本，承诺会不断更新以修补漏洞。并且，现阶段已有众多大型互联网企业如Facebook、Google、Silicon Labs、Wind River等加入了Zephyr的研发，携手打造Zephyr的生态圈



Zephyr 安全流程



Zephyr架构特性

- ③ **轻量级。** Zephyr可在RAM大小为8KB的MCU上流畅运行，经过裁剪甚至可在RAM为最小的2KB时运行
- ④ **可裁剪、配置灵活。** Zephyr为了灵活配置，在内核编译配置上借鉴了 Linux的Kconfig机制，使用该机制能够可选择性地编译功能
- ⑤ **支持多种物联网协议。** 物联网没有统一的通信协议，现阶段是多种协议并存，主流的协议有：ZigBee、Thread、6LoWPAN、BLE、Wi-Fi、NFC、3GPP等
- ⑥ Zephyr 有一个充满活力的国际开发社区，中间件更丰富，活跃度很高



部分板卡设备提供商



Zephyr Project项目部分会员

ZVM (Zephyr-based Virtual Machine)项目

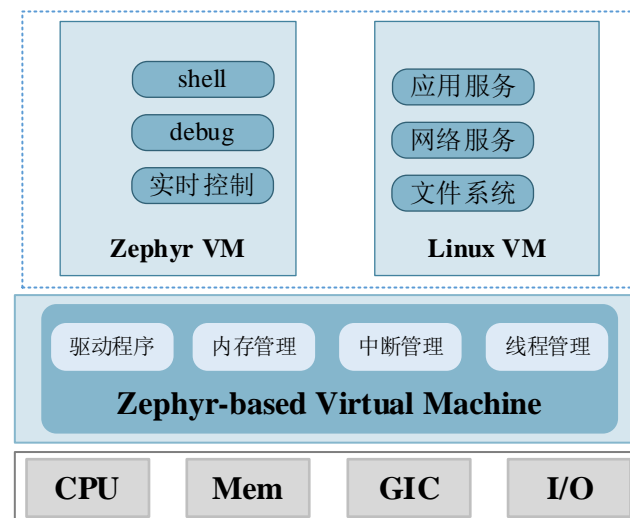
研发ZVM

➤ 整体架构:

- **底层**ARMv8硬件平台 (FVP, QEMU仿真平台, RK3568) ;
- **中间**为ZVM (Zephyr-based Virtual Machine) , 基于Zephyr RTOS 研发
- **上层**为系统支持的虚拟机, 通过ZVM启动Zephyr VM与Linux VM 2类虚拟机

➤ 实现两类VM支持:

- 第一类是Zephyr VM, 作为**实时关键系统**的控制中心
- 第二类是Linux VM (Linux5.16.12内核) , 作为**富功能系统**的控制中心



基于ZVM的混合部署架构

CONTENTS

目录

02

01. 项目背景

02. ZVM设计

03. ZVM演示

04. ZVM规划

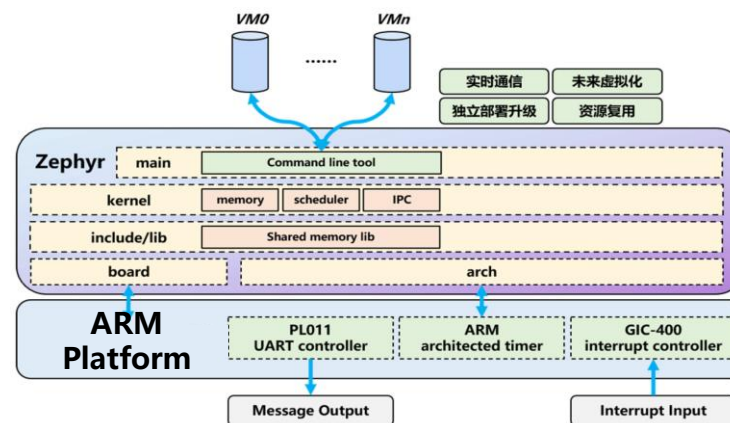
整体方案设计

五大模块设计

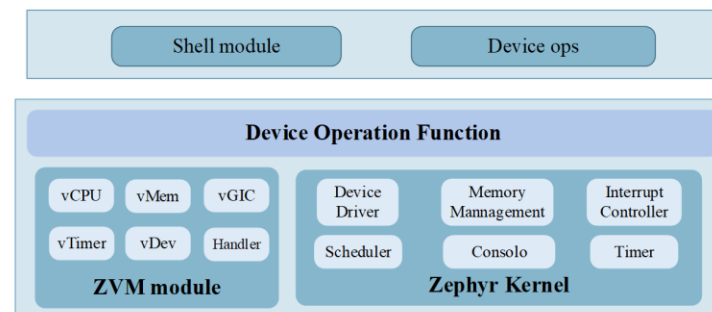
➤ 主要功能模块设计与实现：

- ① 虚拟处理器 (vCPU) 模块：核心计算资源
- ② 虚拟内存 (vMem) 模块：核心存储资源
- ③ 虚拟中断 (vGIC) 模块：物理外设中断+软件模拟中断
- ④ 虚拟设备 (vDev) 模块：完全虚拟化+直通
- ⑤ 虚拟时钟 (vTimer) 模块：Host记录VM的事件

Zephyr RTOS提供线程调度器、内存管理单元、中断管理模块、定时器、设备管理模块等支持



系统架构图



系统功能模块图

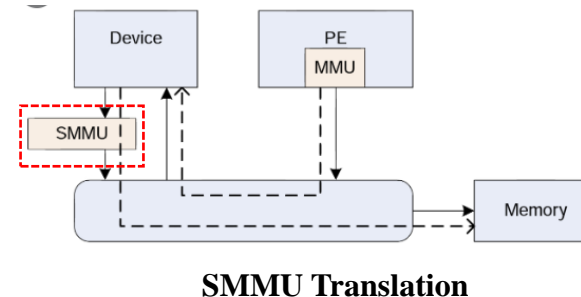
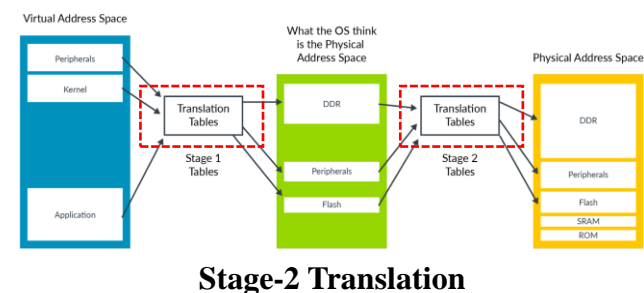
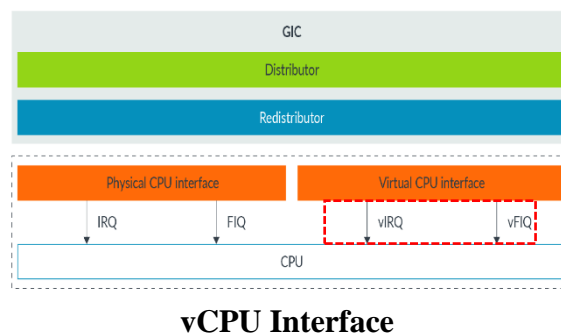
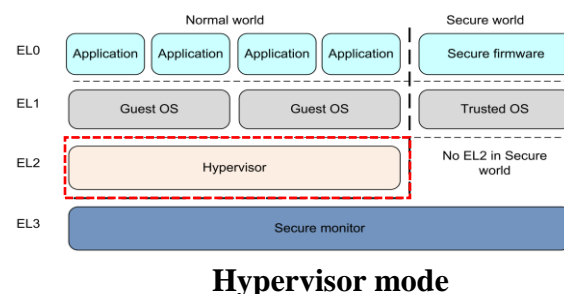
硬件辅助虚拟化

ARMv8架构硬件虚拟化支持

➤ 完善的硬件虚拟化支持

- 处理器虚拟化拓展支持，为Hypervisor设计了单独的特权级（**EL2 mode**）
- 内存虚拟化拓展支持，支持两阶段物理地址转换(**Stage-2 Translation**)
- 中断虚拟化拓展支持，提供一组物理寄存器支持虚拟中断注入(**vCPU Interface**)
- 外设虚拟化拓展支持，为DMA设备提供单独硬件地址转换机制(**SMMU**)

显著降低系统虚拟化开销

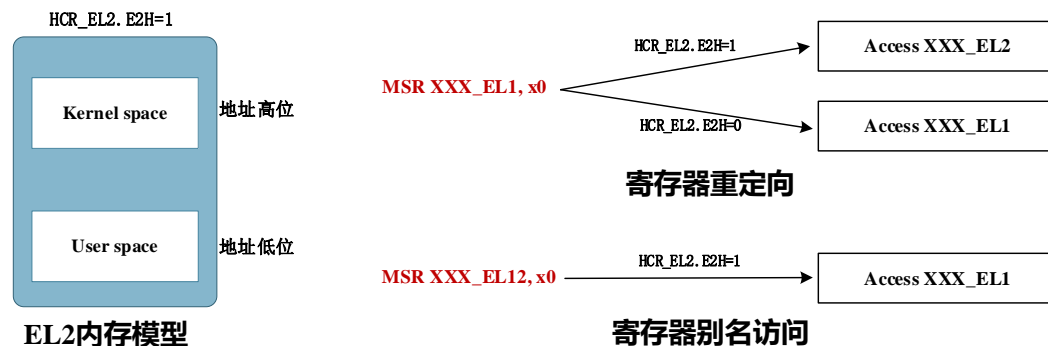


硬件辅助虚拟化

ARMv8虚拟化主机拓展支持

➤ Virtualization Host Extension (VHE)

- ① Host OS直接运行于EL2 mode，显著减少上下文切换开销
- ② EL2内存模型实现用户态/内核态内存访问
- ③ TLB中增加ASID，支持MMU快速地址转换
- ④ EL1寄存器自动重定向为EL2寄存器，避免修改主机内核
- ⑤ 增加EL1别名寄存器，支持Hypervisor控制虚拟机

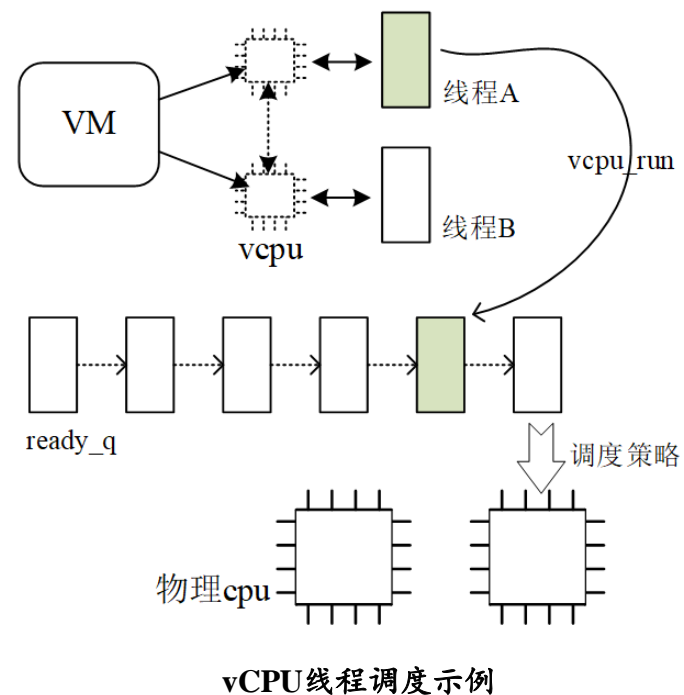


- VHE模式提供了ttbr0_el2和ttbr1_el2两个寄存器。解除了以往非VHE EL2模式下不能访问用户空间的限制。
- ASID（地址空间标识符）标记应用（减少不必要的切换）以减少开销。MMU在做地址转换时会把TLB表项里的ASID和当前进程的ASID值做比较，只有ASID值相等，MMU才认为这条表项是我需要的
- 非VHE模式下，主机如果要运行在EL2模式，需要将EL1模式下访问的以EL1后缀结尾的寄存器修改为以EL2结尾的寄存器（ARM架构以不同的后缀标识不同访问级别）。而在VHE模式下，对一些访问EL1后缀的寄存器，硬件自动变为访问EL2，所以不需要修改内核代码。
- 因为第3点的存在，原先处在Hypervisor模式下访问EL1寄存器时都变为了访问EL2寄存器。为了解决这个问题，ARM硬件新增加了一些别名寄存器，专门为VHE模式下的Hypervisor访问虚拟机寄存器而使用，非VHE模式下基本不用

虚拟处理器模块设计

vCPU线程调度

- ◆ **处理器虚拟化**：为每个虚拟机虚拟出单独隔离的上下文、程序执行及异常状态，vCPU以线程的形式存在并由Hypervisor统一调度
- **线程**为Zephyr调度基本单位，vCPU在系统中实体为单个线程，并根据配置的规则进行统一调度
- **vCPU执行点**：系统调用`z_vcpu_run`函数时，系统将会激活先前已初始化的线程，并加入`ready_q`就绪队列，等待调度
- **调度算法**：调度算法可分为单链表、红黑树和多链表；线程较少时使用单链表，否则可用红黑树和多链表，**现阶段采用单链表实现**
- **相关初始化**：初始化线程模型，初始化调度策略等



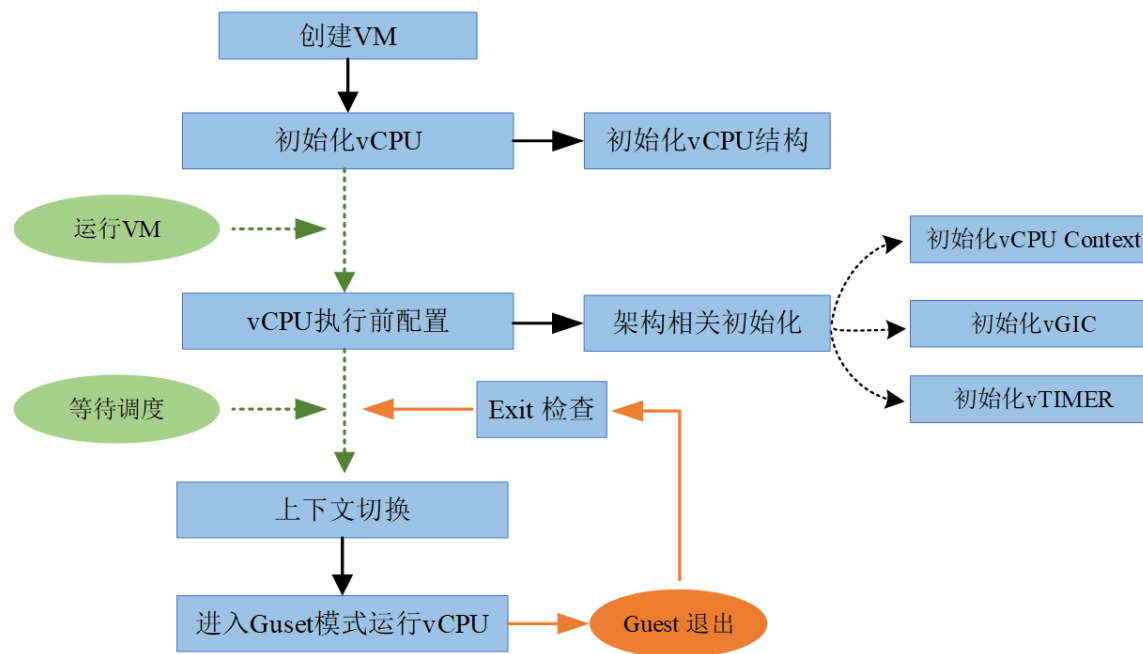
处理器虚拟化的2个关键技术： 1) vCPU的构造； 2) vCPU的分配

虚拟处理器模块设计

vCPU模块执行逻辑

➤ 此模块主要分为如下三个部分：

- 创建VM时根据VM信息**创建vCPU**结构体，并创建vCPU线程加入等待队列
- 运行VM时根据OS信息**初始化vCPU**，并检查系统状态，判断是否执行
- 在Guest Entry或Guest Exit时进行上下文切换并在 Guest Exit时进行处理



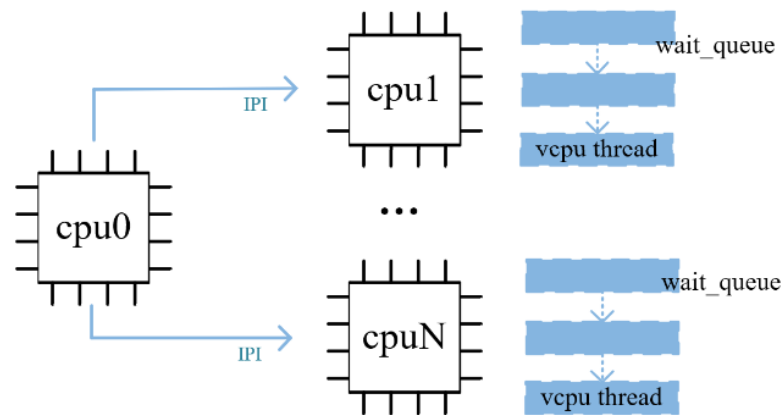
vCPU执行流图

虚拟处理器模块设计

vCPU分配方法

➤ 采用主从核的模式：

- **cpu0负责主机的任务调度**，如shell输入产生的中断将路由至cpu0进行处理，执行控制指令。
- cpu0 与 cpuN 间的通信通过核间中断(Inter-Processor Interrupt,IPI)方式实现，cpu0通过IPI通知cpuN执行任务。
- **vCPU线程在初始化过程中绑定一个物理CPU**，并在启动时部署到指定cpu上执行，直到异常发生。

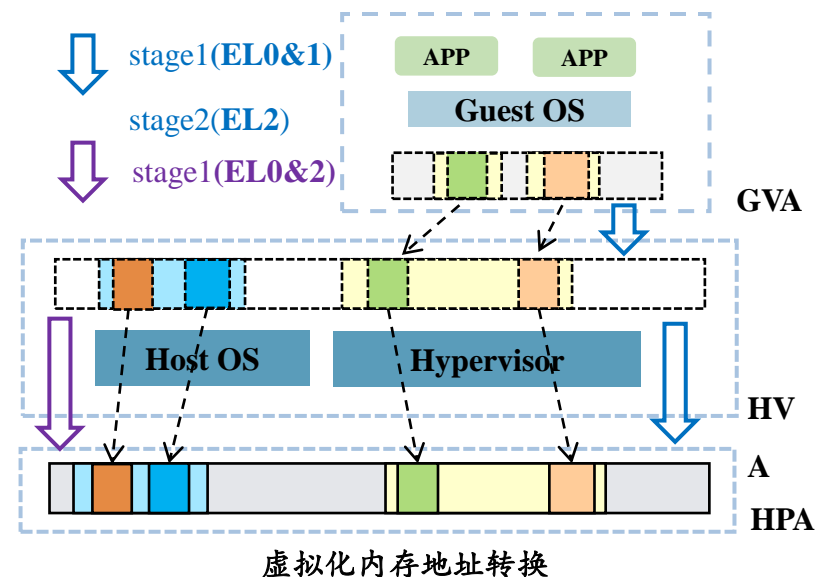


SMP系统调度流程

虚拟内存模块设计

内存地址转换

- ◆ **内存虚拟化**：实现VM内存地址的隔离，并监控VM对实际物理内存的访问，以实现主机物理内存的保护
- **主机操作系统stage-1转换**：主机操作系统经stage-1地址转换，直接由HVA经过一次地址转换找到HPA
- **客户机操作系统stage-2转换**
 - 客户机操作系统通过自身地址映射，实现stage-1地址转换，由GVA找到HVA
 - Hypervisor通过stage-2地址转换机制，由HVA转换为真实物理地址HPA



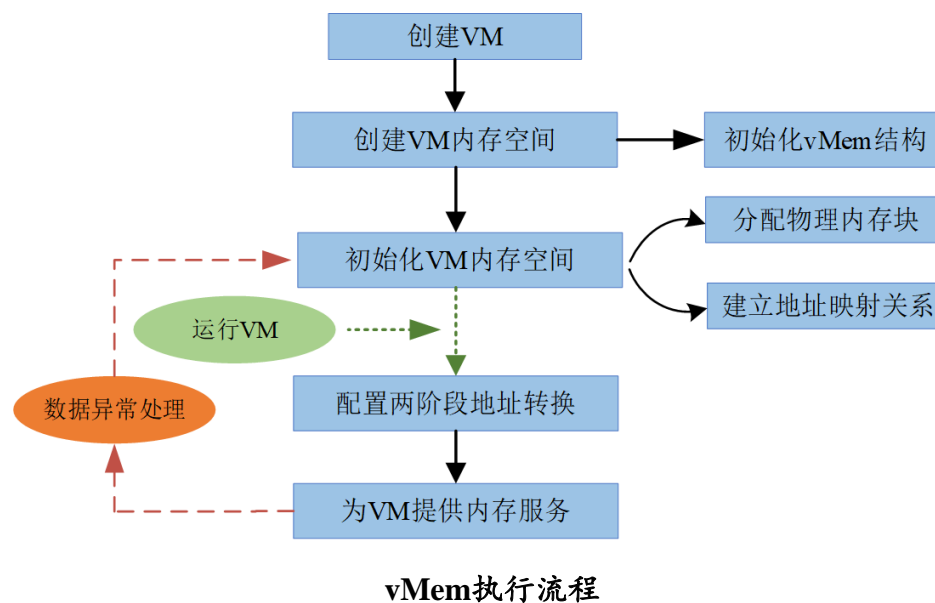
内存虚拟化的2个关键技术： 1) vMem的构造； 2) vMem地址映射

虚拟内存模块设计

vMem模块实现

➤ 此模块主要分为如下两个部分：

- 创建VM时根据参数**初始化vMem**结构体，将虚拟地址空间以Memory Area进行划分；所有Area空间组成的链表，共同构成**VM内存空间**。
- 为Memory Area分配物理内存并加载OS镜像文件，加载完成后**建立地址映射关系**，以配置两阶段的地址转换能力，最终为VM提供内存服务。



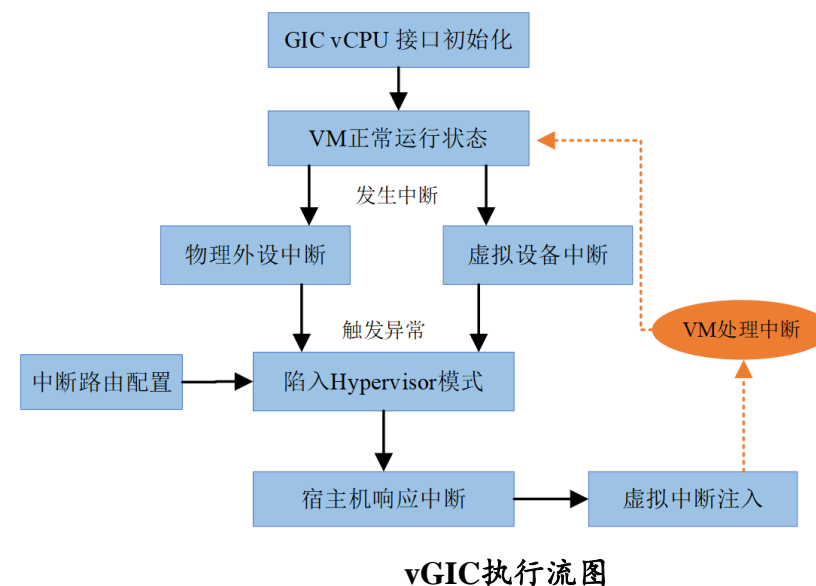
虚拟中断模块设计

vGIC模块实现

- ◆ 中断虚拟化：使用ARM GIC设备，并基于其实现虚拟中断的配置；通过GIC中的Virtual CPU interface和List Register（物理CPU中有一些List register专门用作存储虚拟中断信息）来实现**虚拟中断的注入**。vGIC就是为VM模拟了一个GIC控制器，然后Hypervisor监控VM对物理GIC的操作。

➤ 此模块主要分为如下两个部分：

- **配置vGIC接口**，并将需发送给VM的虚拟中断统一存储到Pend_list链表中，构成当前VM的需处理虚拟中断。list register的数量有限（最多16个，现在平台上默认只有4个，而真实系统中irq号可能有几十上百个），通过Pend_list作为等待list
- **中断注入**：在VM Entry前，根据Pend_list中的中断数据配置相应虚拟中断接口实现中断注入，进入VM即可响应并处理中断。

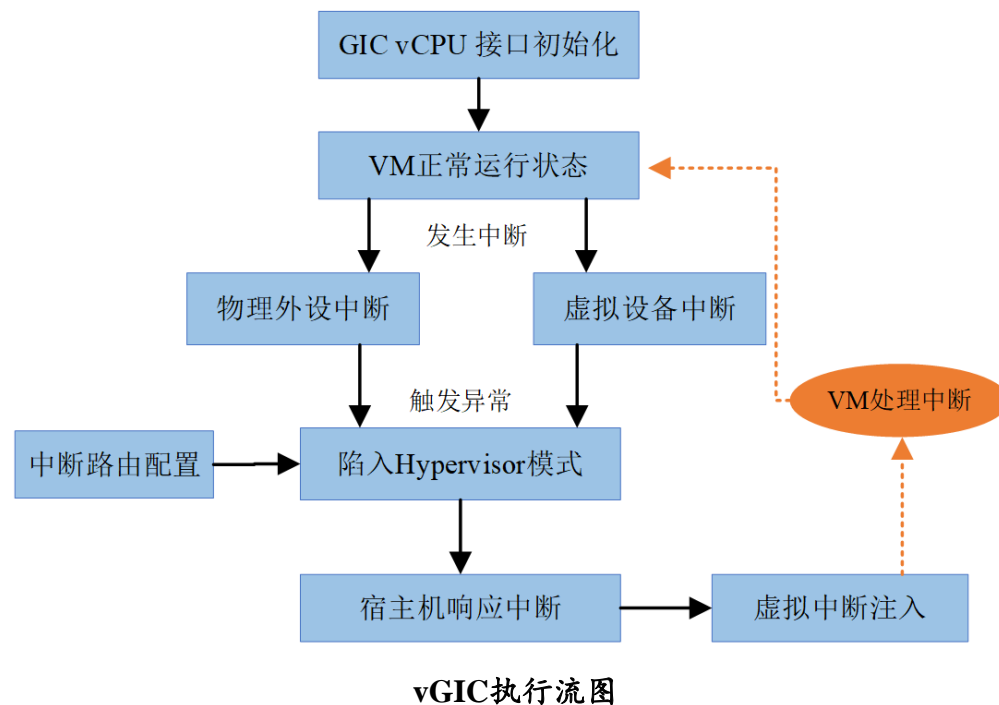


虚拟中断模块设计

实现的中断类型

➤ 现阶段支持的中断处理类型有两种：

- (1) **物理外设中断**（如串口产生的中断）。这里我们将其统一路由到EL2特权级并由Hypervisor进行处理。
- (2) **软件模拟中断**（vCPU间的核间中断）。这类中断用于模拟核间中断IPI（Inter-Processor Interrupt），并由此使得主机能够控制VM；IPI中断首先仍然也是由Hypervisor接管，并根据中断处理函数进行相应处理（因为现阶段VM是单核环境，所以通过主机CPU给VM vCPU发送核间中断）。



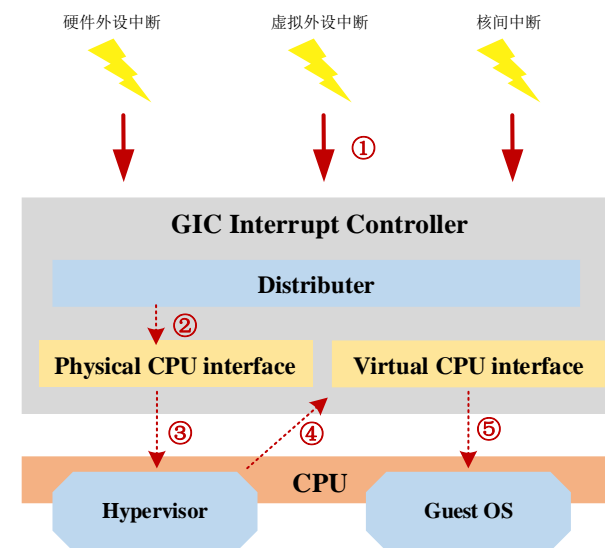
中断虚拟化的2个关键技术： 1) vGIC的构造； 2) 中断处理流程的实现

虚拟中断模块设计

虚拟中断处理流程

- ① 外设产生一个中断发送到Distributor
- ② Distributor把这个中断发送给Physical CPU interface
- ③ Physical CPU interface告诉Hypervisor去处理这个中断
- ④ Hypervisor对这个中断进行检查，发现这个中断是送给VM处理的，它会设置 一个对应的虚拟中断，把这个虚拟中断加入到Virtual CPU interface
- ⑤ Virtual CPU interface会根据Hypervisor加入的虚拟中断， 向List register写入内容（用作存储虚拟中断信息），然后VM就可以感知中断是给它的
- ⑥ VM通过virtual CPU interface发来的中断进行处理(在EL1模式)，处理后返回
- ⑦ Virtual CPU interface发现这个虚拟中断来自于一个物理中断，就会在Distributor上清除这个物理中断（表示处理完毕），整个虚拟中断处理过程结束

中断控制器是模拟的控制器，中断处理是先由主机处理，再转发给vCPU处理



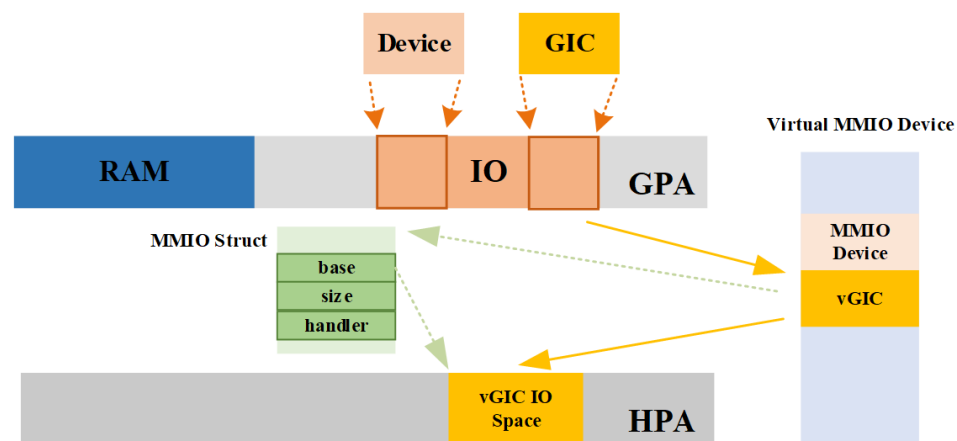
虚拟中断处理流程

虚拟外设模块设计与实现

虚拟外设映射

➤ MMIO虚拟设备映射框架：

- 当虚拟机访问一个虚拟的I/O设备时，可以通过内存中提前配置好的Virtual MMIO Device来实现设备访问。
- MMIO struct保存虚拟设备信息，其中主要通过回调函数进行自定义处理。如图中虚拟机进行GIC设备的操作时，会触发异常并调用handler回调函数，以访问到Virtual MMIO Device，实现设备的虚拟化。



虚拟外设MMIO框架

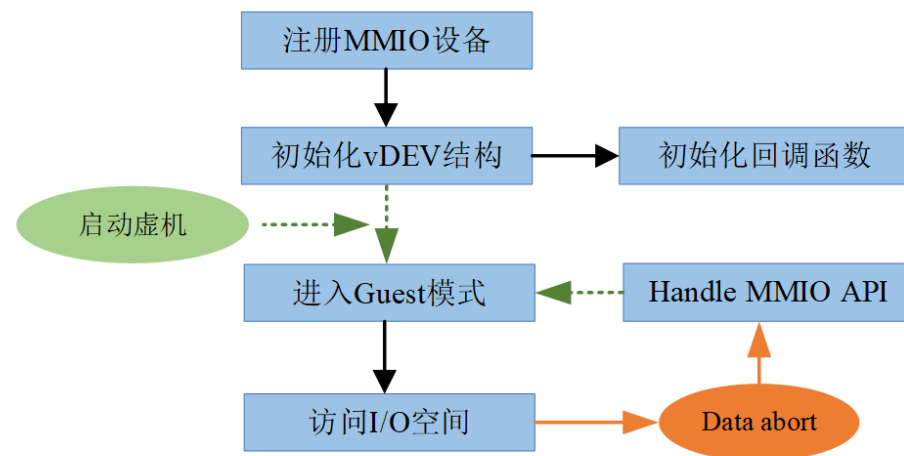
设备虚拟化的2个关键技术：vDec设备的构造，vCPU如何访问vDec

虚拟外设模块设计

虚拟外设模块设计

➤ 此模块主要分为如下两个部分：

- 在VM创建过程中，需要创建一些**Virtual MMIO** (Memor mapped I/O,将设备地址映射到物理内存地址空间进行访**设备**，为VM访问相应设备构造好地址空间和处理机制)
- 在VM执行过程中，当访问到I/O空间时，将触发地址访问：误（vCPU无法直接访问**Virtual MMIO device**地址）Hypervisor调用回调函数来模拟device访问操作。而这**Virtual MMIO device**地址通过MMIO映射到物理内存空间，从而vCPU可以通过Hypervisor访问到Virtual MMIO Device



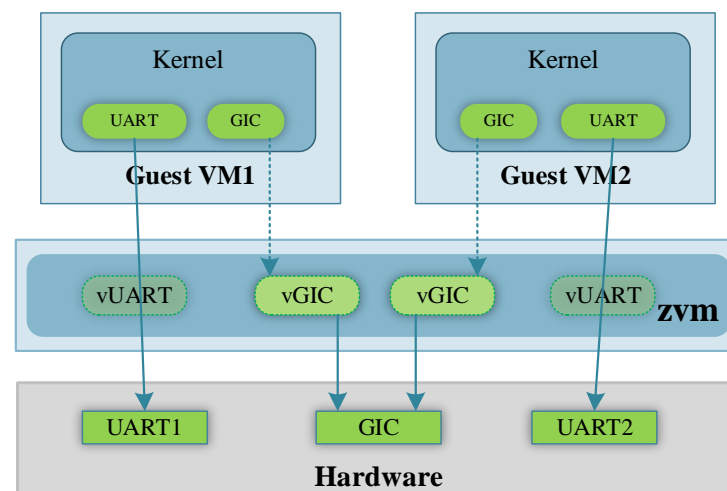
虚拟外设执行流图

设备是模拟的设备，设备访问是代理式访问

虚拟外设模块设计

完全设备虚拟化

- **完全虚拟化设备**：由软件模拟，兼容性好，开销大，性能差。用于**不可单独分配设备**（例如中断控制器GIC）。
- 为每个VM提供了一个完全虚拟化的GIC设备，并为其在内存中分配一段地址，模拟GIC的IO地址空间，并存储当前vGIC的配置信息；当执行VM时，将配置信息通过Hypervisor控制写入物理GIC地址当中或者完全通过软件模拟GIC访问操作，以实现GIC设备的虚拟化



中断设备是模拟的设备，中断设备访问是代理式访问

虚拟外设模块设计

设备直通

- **设备直通**：使用原有的硬件驱动，不增加新驱动，性能和主机上直接使用该物理硬件非常接近。用于**可单独分配设备**（调试串口等独占外设）
- 如果在VM访问前，系统提前建立了二阶段地址映射的话，vCPU就可以直接访问真实设备
- 直通如果涉及DMA操作的话就需要SMMU（将虚拟机物理地址空间内的GPA翻译成HPA）支持，因为DMA（外设可以通过DMA，将数据批量传输到物理内存）必须使用真实的物理地址，使用虚拟地址会报错。
- 直通在VM初始化过程中分配给指定的VM，实现VM对该设备的直接使用，而**Hypervisor在此过程中只需要记录设备分配给了哪个VM**，不需要进行具体设备功能的模拟，减小了系统的开销。

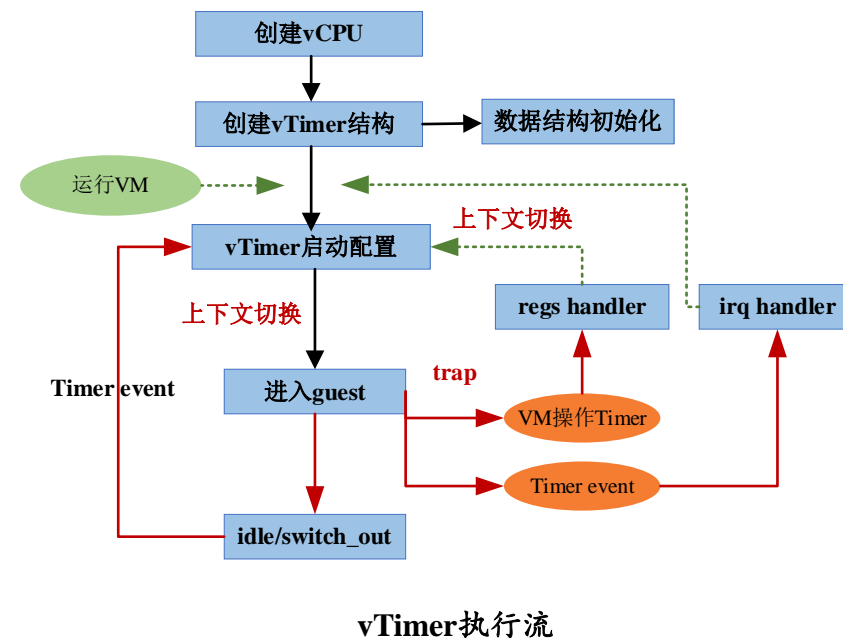
设备是真实的设备，设备访问是直接式访问

虚拟时钟模块设计

vTimer模块简介

◆ 时钟虚拟化:

- vTimer用于给vCPU提供**虚拟定时器**，满足VM中需要定时器的服务（比如VM需要在1秒后使用某个设备）。
- 每次vTimer发生计数器中断后需要Hypervisor处理,一样是硬件中断，只不过它用的是专门的面向虚拟时钟的寄存器，每个CPU定义了一组虚拟时钟寄存器，它们单独计数并在预定的时间过后抛出中断，并由主机转发至VM



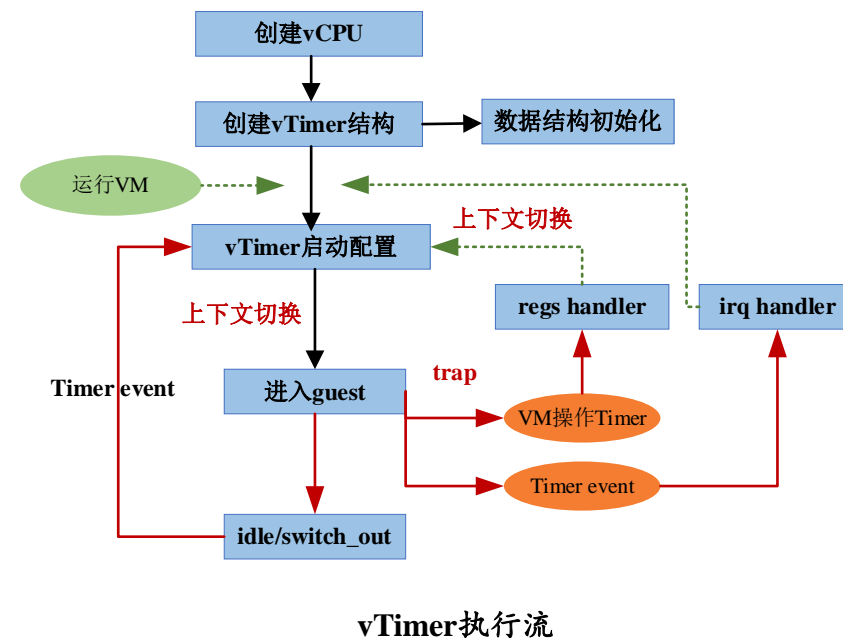
时钟虚拟化关键技术： 1) vTimer的构造； 2) vTimer定时事件如何被vCPU感知到

虚拟时钟模块设计

vTimer模块执行逻辑

➤ vTimer :

- vCPU初始化时初始化vTimer并配置中断服务程序ISR的异常处理函数及timeout事件处理函数。
- 通过配置Compare值, 实现vTimer中断, 以支持调度功能。
- vCPU空闲或者退出处理器时, 使用timeout事件处理函数注入中断并调度vCPU。



vTimer定时事件是一个硬件中断, 触发事件之后就走中断流程

CONTENTS

目录

03

01. 项目背景

02. ZVM设计

03. ZVM演示

04. ZVM规划

软硬件平台

软硬件平台概览:

◆ 硬件仿真平台 (AArch64 架构处理器)

- FVP Cortex-a55 Platform(DS2021.2内置仿真器)
- QEMU virt (max) platform(修改的QEMU6.20, 增加多串口支持, 方便演示)

◆ Host操作系统

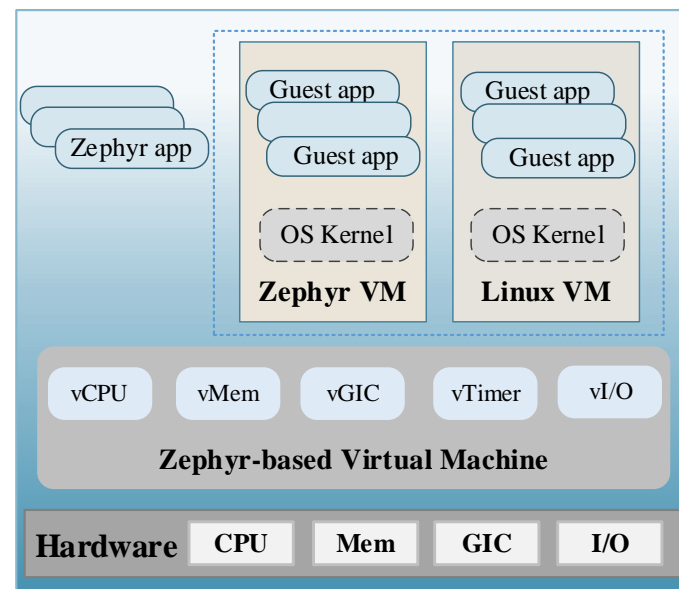
- Zephyr OS(version 2.7.99)
- 轻量级实时嵌入式操作系统

◆ 具体模块设计与实现

- 包括vCPU、vMem、vTimer、vGIC、vDev

◆ 虚拟机支持

- Zephyr VM 和 Linux VM.



ZVM部署架构

演示内容一

虚拟机环境下的基本操作

◆ 演示平台

- FVP Cortex-A55 Platform

◆ 支持外设

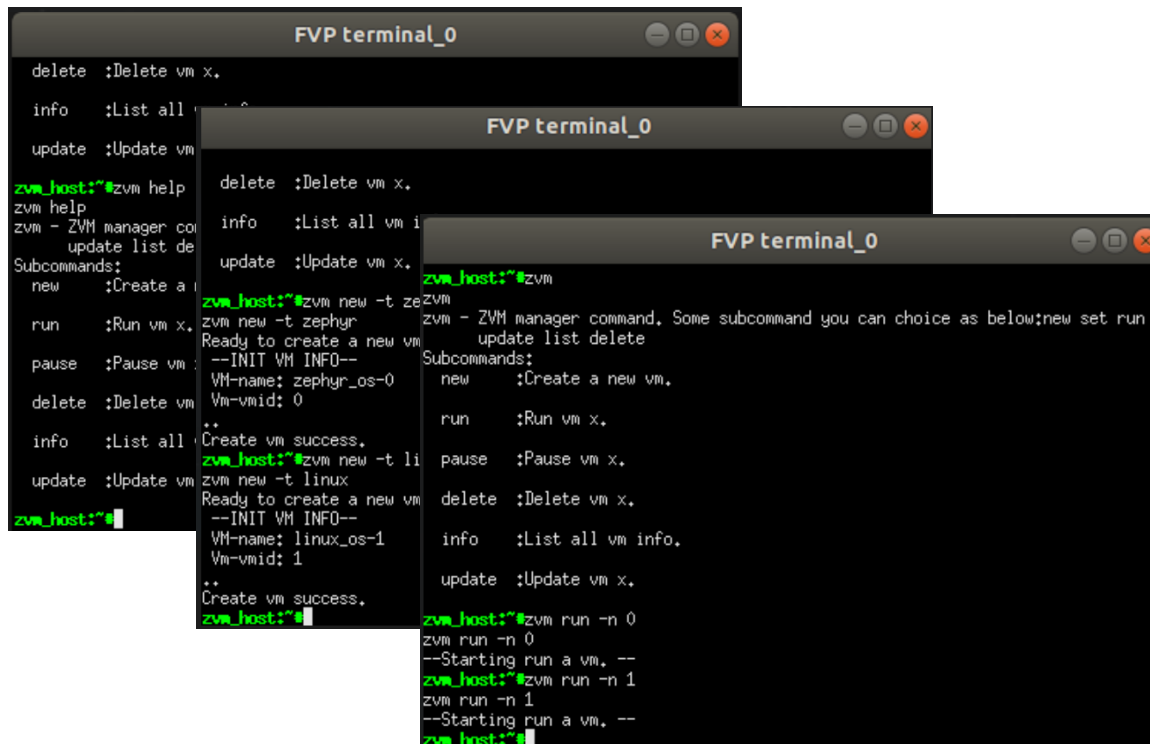
- 中断控制器 (GICv3)
- 调试串口 (PL011)

◆ 主机支持命令

- new, run, pause, delete, info

◆ 虚拟机支持

- 调试串口打印系统或内核信息



```
FVP terminal_0
delete :Delete vm x.
info :List all vm info.
update :Update vm x.

zvm_host:~#zvm help
zvm help
zvm - ZVM manager command. Some subcommand you can choice as below:new set run
update list delete
Subcommands:
new :Create a new vm.
run :Run vm x.
pause :Pause vm x.
delete :Delete vm x.
info :List all vm info.
update :Update vm x.

zvm_host:~#zvm new -t zephyr
zvm new -t zephyr
Ready to create a new vm
--INIT VM INFO--
VM-name: zephyr_os-0
Vm-vmid: 0
**
Create vm success.
zvm_host:~#zvm new -t linux
zvm new -t linux
Ready to create a new vm
--INIT VM INFO--
VM-name: linux_os-1
Vm-vmid: 1
**
Create vm success.
zvm_host:~#

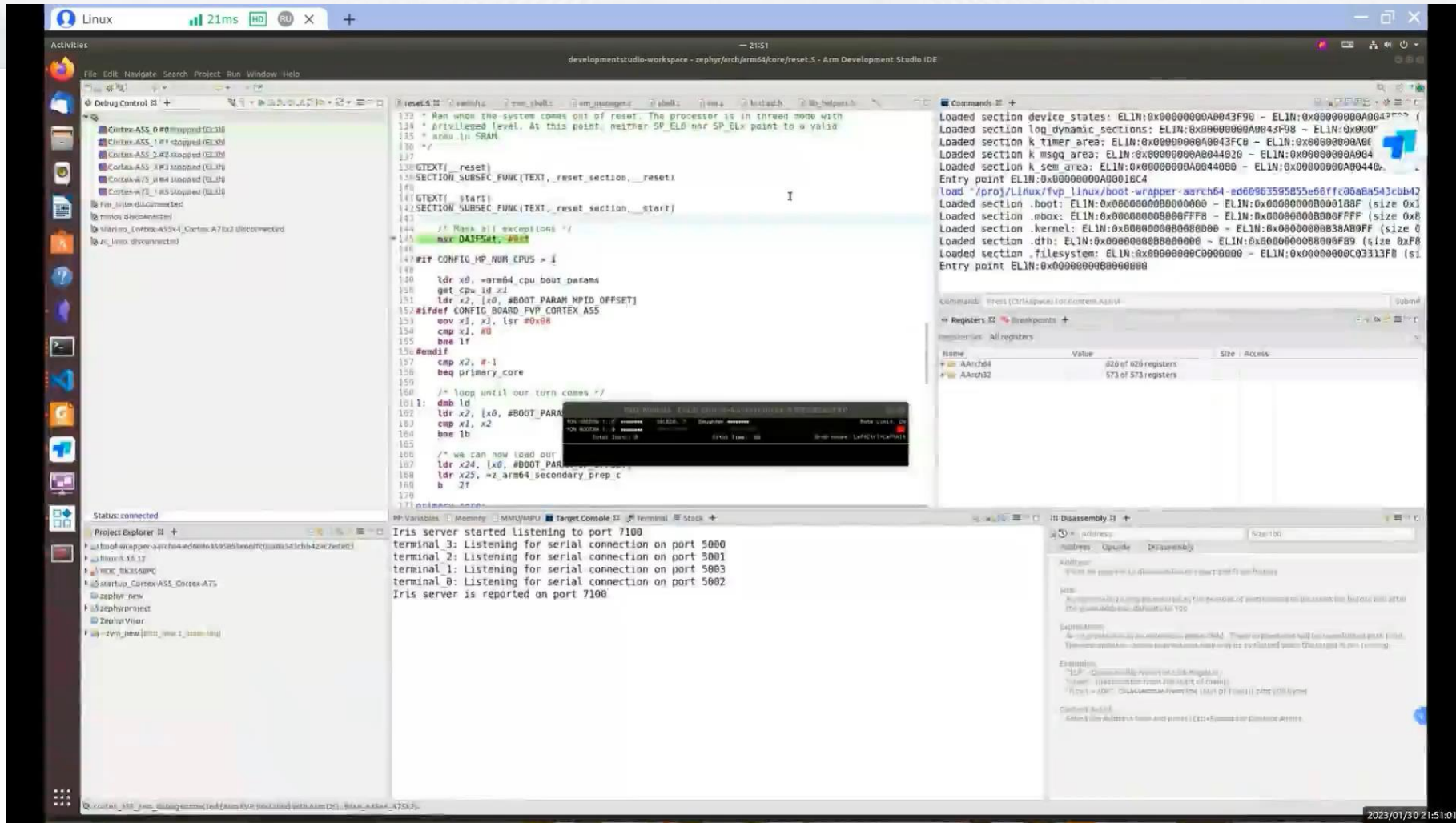
FVP terminal_0
delete :Delete vm x.
info :List all vm info.
update :Update vm x.

zvm_host:~#zvm
zvm - ZVM manager command. Some subcommand you can choice as below:new set run
update list delete
Subcommands:
new :Create a new vm.
run :Run vm x.
pause :Pause vm x.
delete :Delete vm x.
info :List all vm info.
update :Update vm x.

zvm_host:~#zvm run -n 0
zvm run -n 0
--Starting run a vm. --
zvm_host:~#zvm run -n 1
zvm run -n 1
--Starting run a vm. --
zvm_host:~#
```

虚拟机操作演示示例

演示内容一：虚拟机基本操作



演示内容二

虚拟化环境下的时延测试

◆ 演示平台

- QEMU virt(max) platform(修改的QEMU6.20)

◆ 虚拟机版本

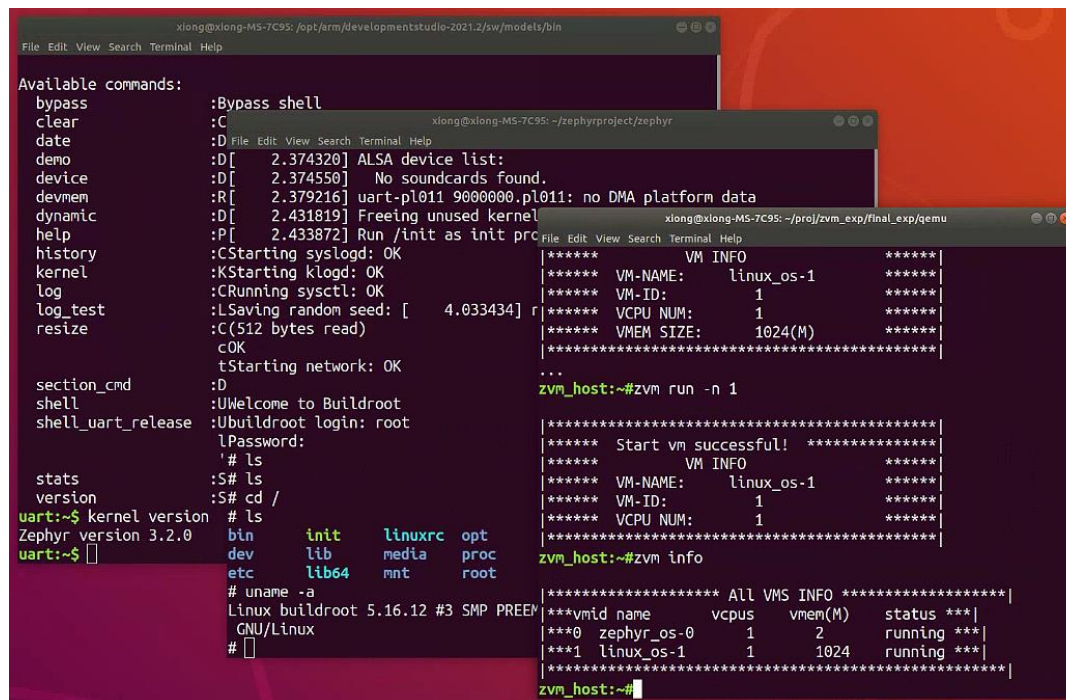
- Zephyr 3.2.0 (单核)

◆ 支持外设

- 中断控制器 (GICv3)
- 调试串口 (PL011)

◆ 测试软件

- Zephyr自带的Benchmark测试应用



```
xiong@xiong-M5-7C95: /opt/arm/developmentstudio-2021.2/sw/models/bin
File Edit View Search Terminal Help

Available commands:
bypass      :Bypass shell
clear       :C
date        :D File Edit View Search Terminal Help
deno        :D[ 2.374320] ALSA device list:
device      :D[ 2.374550] No soundcards found.
devmem      :R[ 2.379216] uart-pl011 9000000.pl011: no DMA platform data
dynamic     :D[ 2.431819] Freeing unused kernel
help        :P[ 2.433872] Run /init as init proc
history     :CStarting syslogd: OK
kernel      :KStarting klogd: OK
log         :CRunning sysctl: OK
log_test    :LSaving random seed: [ 4.033434] r
resize      :C(512 bytes read)
            :COK
            :tStarting network: OK

section_cmd :D
shell       :UWelcome to Buildroot
shell_uart_release :Ubuildroot login: root
            :lPassword:
            :# ls
            :S# ls
            :S# cd /
            :# ls
            :bin      init      linuxrc  opt
            :dev      lib      media   proc
            :etc      lib64   mnt     root

uart~$ kernel version
Zephyr version 3.2.0
uart~$

# uname -a
Linux buildroot 5.16.12 #3 SMP PREEM
GNU/Linux
#

***** VM INFO *****
***** VM-NAME: linux_os-1 *****
***** VM-ID: 1 *****
***** VCPU NUM: 1 *****
***** VMEM SIZE: 1024(M) *****
*****

zvm_host:~#zvm run -n 1

***** Start vm successful! *****
***** VM INFO *****
***** VM-NAME: linux_os-1 *****
***** VM-ID: 1 *****
***** VCPU NUM: 1 *****
*****

zvm_host:~#zvm info

***** All VMS INFO *****
*****vmid name vcpus vmem(M) status ***|
***0 zephyr_os-0 1 2 running ***|
***1 linux_os-1 1 1024 running ***|
*****

zvm_host:~#
```

QEMU平台演示

演示内容二:虚拟化环境下的时延测试

各类调用的时延测试

实验平台:

- qemu virt (max): Zephyr部署到qemu上, 逻辑上相当于裸机;
- Qemu virt(max-zvm): Zephyr部署到基于qemu的zvm之上, 相当于裸机上再加虚拟机。

对比平台 qemu virt(max)

```
*** Booting Zephyr OS build v3.2.0-rc3-38-gc7fc9575ebd0 ***
START - Time Measurement
Timing results: Clock frequency: 62 MHz
Average thread context switch using yield : 345 cycles , 5521 ns
Average context switch time between threads (coop) : 107 cycles , 1724 ns
Switch from ISR back to interrupted thread : 3812 cycles , 60992 ns
Time from ISR to executing a different thread : 11540 cycles , 184640 ns
Time to create a thread (without start) : 4866 cycles , 77856 ns
Time to start a thread : 3740 cycles , 59840 ns
Time to suspend a thread : 20916 cycles , 334656 ns
Time to resume a thread : 2355 cycles , 37680 ns
Time to abort a thread (not running) : 6130 cycles , 98080 ns
Average semaphore signal time : 35 cycles , 560 ns
Average semaphore test time : 34 cycles , 556 ns
Semaphore take time (context switch) : 3513 cycles , 56208 ns
Semaphore give time (context switch) : 3139 cycles , 50224 ns
Average time to lock a mutex : 25 cycles , 405 ns
Average time to unlock a mutex : 13 cycles , 211 ns
Average time for heap malloc : 751 cycles , 12031 ns
Average time for heap free : 635 cycles , 10160 ns
```

测试平台 qemu virt(max-zvm)

```
*** Booting Zephyr OS build v3.2.0-rc3-38-gc7fc9575ebd0 ***
START - Time Measurement
Timing results: Clock frequency: 62 MHz
Average thread context switch using yield : 339 cycles , 5427 ns
Average context switch time between threads (coop) : 143 cycles , 2291 ns
Switch from ISR back to interrupted thread : 4233 cycles , 67728 ns
Time from ISR to executing a different thread : 11317 cycles , 181072 ns
Time to create a thread (without start) : 9404 cycles , 150464 ns
Time to start a thread : 5684 cycles , 90944 ns
Time to suspend a thread : 15913 cycles , 254608 ns
Time to resume a thread : 1439 cycles , 23024 ns
Time to abort a thread (not running) : 6955 cycles , 111280 ns
Average semaphore signal time : 33 cycles , 539 ns
Average semaphore test time : 15 cycles , 252 ns
Semaphore take time (context switch) : 2608 cycles , 41728 ns
Semaphore give time (context switch) : 3577 cycles , 57232 ns
Average time to lock a mutex : 551 cycles , 8827 ns
Average time to unlock a mutex : 20 cycles , 320 ns
Average time for heap malloc : 587 cycles , 9403 ns
Average time for heap free : 421 cycles , 6741 ns
```

演示内容二:虚拟化环境下的时延测试

各类调用的时延测试

数据对比: 包含线程上下文切换时延, 中断时延以及线程状态切换时延 (Timing results: Clock frequency: 62 MHz)

测试内容	qemu virt(max)	qemu virt(max-zvm)	差值	波动
Average thread context switch using yield	5,521ns	5,427ns	-6ns	1.7%
Average context switch time between threads	1,724ns	2,291ns	+567ns	24.7%
Switch from ISR back to interrupted thread	60,992ns	67,728ns	+6,736ns	9.9%
Time from ISR to executing a different thread	184,640ns	181,072ns	-3,568ns	1.9%
Time to create a thread(without start)	77,856ns	150,464ns	+72,608ns	48.2%
Time to start a thread	59,840ns	90,944ns	+33,104ns	34.2%
Time to abort a thread	98,080ns	111,280ns	+13,200ns	11.8%

相较于直接在qemu平台运行, 在zvm平台上创新线程与启动线程有较大时延增加

硬件平台（正在进行）

硬件平台适配与测试

◆ 硬件平台

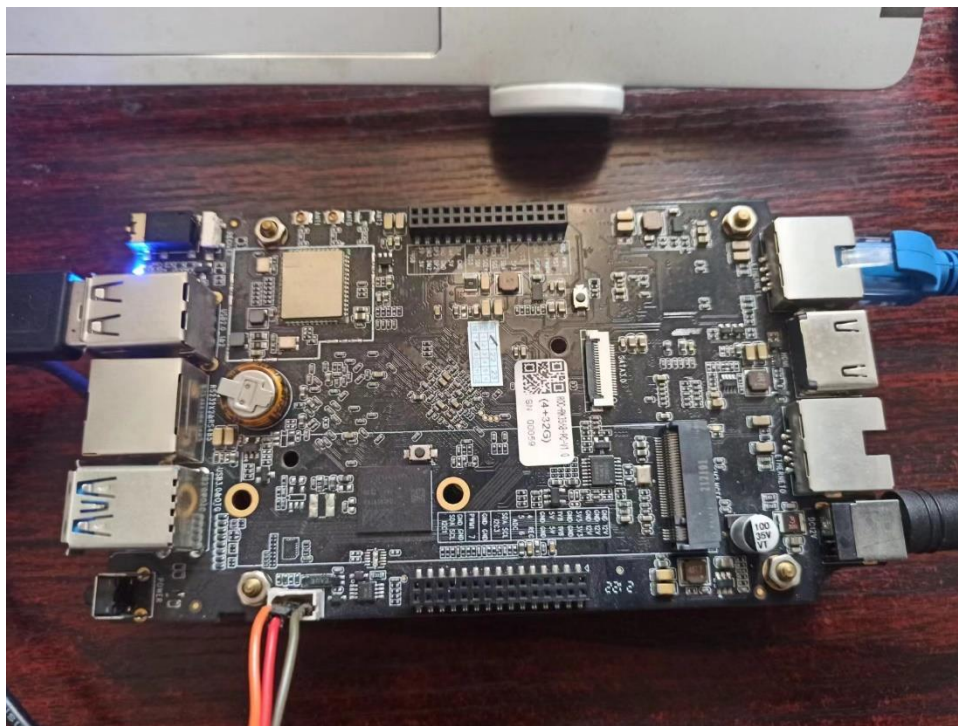
- RK3568 SOC (Cortex-A55)

◆ 测试Zephyr版本

- Zephyr 3.2.0 (实现基础应用功能)

◆ 测试ZVM

- 添加ZVM EL2特权级的Hypervisor层初始化代码后，硬件初始化失败（需迭代探索寄存器配置顺序与依赖关系等），无法正常进入ZVM系统。
- 预计近期解决该问题，实现硬件适配，并进行时延测试。



真实硬件平台

CONTENTS

目录

04

01. 项目背景

02. ZVM设计

03. ZVM演示

04. ZVM规划

项目优势

◆ ZVM实时性优势

现有大规模部署的Linux KVM方案中存在**系统控制精度不够，实时需求无法满足**等问题；ZVM将有效提高系统**实时性**，实现对系统的**精准控制**，以满足关键系统实时性场景下的**应用需求**。

◆ ZVM轻量级设计优势

现有Linux KVM实现方案较为厚重，ZVM的轻量级设计在确保**系统稳定性**的前提下，在**性能开销与部署成本**方面将优于Linux KVM。



应用领域：5G通信、工控、汽车等领域

◆ ZVM开源优势

基于发展迅速的产品级**开源RTOS**：Zephyr，实现了业界首款**基于RTOS的开源虚拟机管理器（ZVM）**，以开源带动行业技术发展，打破闭源商业模式。

ZVM规划

◆ ZVM会在openeuler SIG-Zephyr下开源、孵化、运作

- 2023年2月:在sig-zephyr下建立仓库,代码开源
- 2023年3月/4月: 在openuler Developer Day分享
- 2023年6月: 在zephyr developer summit上分享
- 2023年9月: openeuler Embedded和ZVM整合

◆ ZVM能够回馈到zephyr社区, 走向世界

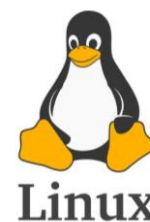
- 利用Zephyr的开源生态
- Apache2.0协议方便使用

◆ 应用规划

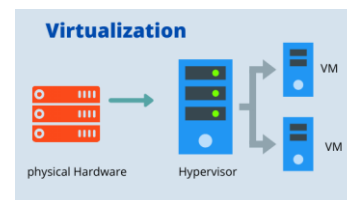
- ZVM将支持拓展支持多种架构 (RISC-V, X86)
- 实时嵌入式领域领先解决方案 (汽车、5G设备等)



arm



RTOS



THANKS FOR ALL



嵌入式与网络计算湖南省重点实验室

<http://esnl.hnu.edu.cn/>