技 术 文 件

技术文件名称： Xvisor实现分析

技术文件编号：

版 本： V0.1

拟 制

审 核

会 签

标准化

批 准

海宁奕斯伟集成电路设计有限公司

修改记录

| 文件编号 | 版本号 | 拟制人/  修改人 | 拟制/修改日期 | 更改理由 | 主要更改内容  （写要点即可） |
| --- | --- | --- | --- | --- | --- |
|  | V0.01 | 王飞 | 2021.10.21 | 新建 | 添加xvisor简介，编译框架 |
|  | V0.02 | 王飞 | 2021.10.24 |  | 新增xvisor启动流程 |
|  | V0.03 | 王飞 | 2021.10.27 |  | 新增xvisor设计模型章节，翻译官方文档，完善启动流程 |
|  | V0.04 |  |  |  |  |
|  | V0.05 |  |  |  |  |
|  | V0.06 |  |  |  |  |
|  | V0.07 |  |  |  |  |
|  | V0.08 |  |  |  |  |
|  | V0.1 |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 注1：每次更改归档文件（指归档到事业部或公司档案室的文件）时，需填写此表。  注2：文件第一次归档时，“更改理由”、“主要更改内容”栏写“无”。 | | | | | |

**目 录**

目录

[Xvisor简介 4](#_Toc86158634)

[Xvisor代码获取 5](#_Toc86158635)

[Xvisor编译框架 6](#_Toc86158636)

[代码目录结构 6](#_Toc86158637)

[编译方法 6](#_Toc86158638)

[Xvisor总体设计模型 6](#_Toc86158639)

[1.虚拟机模型 6](#_Toc86158640)

[2.Hypervisor配置 8](#_Toc86158641)

[3. Hypervisor Timer 8](#_Toc86158642)

[4. Hypervisor Manager 9](#_Toc86158643)

[5.Hypervisor Scheduler 11](#_Toc86158644)

[6. Hypervisor 负载均衡 11](#_Toc86158645)

[7. Hypervisor Threads 12](#_Toc86158646)

[8. Device Driver Framework 12](#_Toc86158647)

[9. Device Emulation Framework 13](#_Toc86158648)

[10. Standard I/O 13](#_Toc86158649)

[11. Command Manager 14](#_Toc86158650)

[12. Storage Virtualization 15](#_Toc86158651)

[13. Network Virtualization 15](#_Toc86158652)

[14. Virtualized I/O Frameworks 16](#_Toc86158653)

[Xvisor启动流程 17](#_Toc86158654)

[启动log 17](#_Toc86158655)

[启动函数流程： 17](#_Toc86158656)

[文件系统挂载: 19](#_Toc86158657)

[Xvisor中断模块分析 22](#_Toc86158658)

[Xvisor功能模块 23](#_Toc86158659)

[关注的问题 23](#_Toc86158660)

[启动函数流程： 23](#_Toc86158661)

[参考文档： 23](#_Toc86158662)

## Xvisor简介

Xvisor®是一个开源的type-1 hypervisor，旨在提供一个单片的、轻量级的、可移植的、灵活的虚拟化解决方案。

Xvisor为ARMv5、ARMv6、ARMv7a、ARMv7a-ve、ARMv8a、x86\_64、RISC-V等CPU架构提供了高性能、低内存占用的虚拟化解决方案。与其他ARM hypervisors相比，它是为数不多的为没有ARM虚拟化扩展的ARM cpu提供支持的hypervisors之一。

在RISC-V领域，它是世界上第一个Type-1 RISC-V hypervisor。

Xvisor源代码具有高度可移植性，只要它们具有页内存管理单元(PMMU)和GNU C编译器(GCC)端口，就可以很容易地移植到大多数通用的32位或64位架构。

Xvisor主要支持全虚拟化，因此支持大量未修改的Guest操作系统。对于Xvisor来说，半虚拟化是可选的，并且将以一种独立于架构的方式(例如VirtIO PCI/MMIO设备)支持半虚拟化，以确保Guest操作系统中不更改使用半虚拟化。

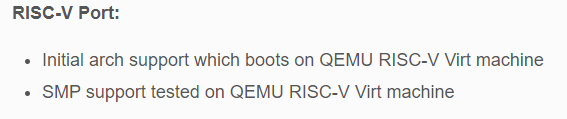
它具有现代管理程序所期望的大多数特性，例如:基于设备树的配置，无Tickless和高分辨率计时，线程框架，主机设备驱动框架，IO设备仿真框架，运行时可加载模块，直通硬件访问，动态客户创建/销毁，管理终端，网络虚拟化，输入设备虚拟化，显示设备虚拟化等等。

### Xvisor对riscv的支持情况：

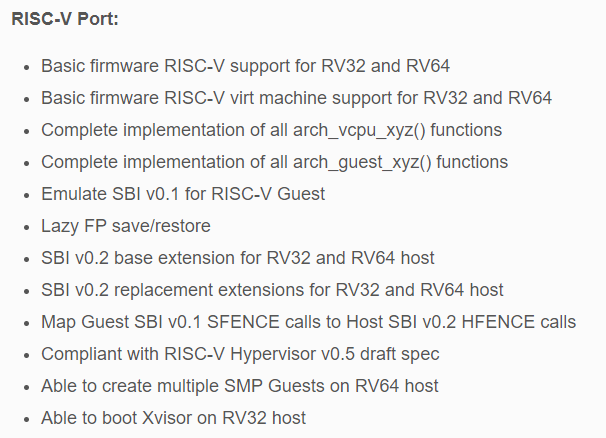
Xvisor对RISCV支持的在Xvisor发布历史上比较晚，2018-10-21发布的releaseV0.2.11上第一次支持了RISCV架构。后续在2020-1-2发布的releaseV0.3.0，完善了riscv的支持，可以在qemu上模拟运行。目前最新的release版本就是V0.3.0。xvisor对riscv的支持只存在了2个release版本，所以肯定会有很多不完善的地方。

Xvisor支持的riscv hypervisor V0.5版本的提案，对应qemu的要求是5.1.0以上（目前使用6.1.0）

Xvisor V0.2.11对riscv的release Features：



Xvisor V0.3.0对riscv的release Features：



## Xvisor代码获取

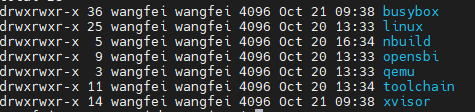
目前xvisor相关的工程代码已经合入公司gerrit。可以通过如下命令获取：

#repo init -u ssh://Gerrit账户@gerrit.eswin.cn:29418/platform/manifest -b yixi-hypervisor-f\_xvisor --repo-url=ssh://Gerrit账户@gerrit.eswin.cn:29418/tools/git-repo

#repo sync -cdj4 --no-tags

### Xvisor工程代码

工程内部包含的xvisor执行环境需要的必要资源



## Xvisor编译框架

### 代码目录结构

xvisor

├── COPYING

├── HOSTS

├── MAINTAINERS

├── Makefile

├── README.md

├── arch

├── build---临时，框架有BUG，指定O=path之后还是会在源码目录创建build目录

├── commands

├── core

├── daemons

├── docs

├── drivers

├── emulators

├── libs

├── openconf.cfg

├── tests

└── tools

### 编译方法

Xvisor目前已经创建repo工程，已支持一键编译。主要编译命令：

#cd nbuild

#make

Xvisor本身代码的编译，后续在添加，主要是vmm,bin和firmware,bin。会涉及到makefile，连接脚本，entry addr等内容。

TODO:待添加

## Xvisor总体设计模型

Xvisor包含了多个功能模块，后续需要按模块分析。TODO

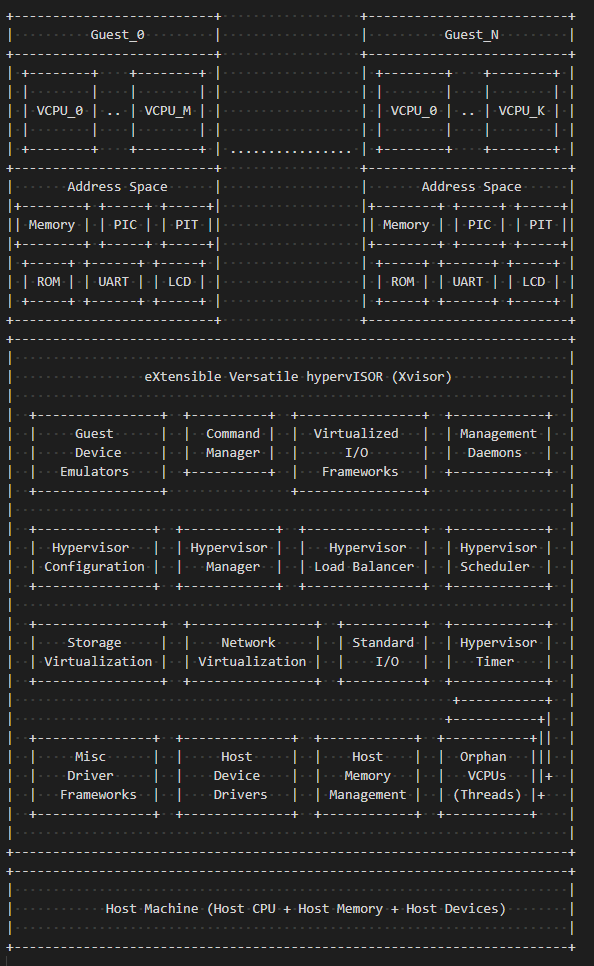
### 1.虚拟机模型

Xvisor是一个直接运行在host主机上的硬件辅助的系统虚拟化软件，它属于type-1类型的Hypervisor。

在Xvisor中，我们将系统虚拟机实例称为“Guest”实例，将系统虚拟机的虚拟cpu称为“VCPU”实例。此外，VCPU属于一个Guest称为“Normal VCPU”，不属于任何Guest的VCPU称为“Orphan VCPU”。Xvisor为各种后台处理和运行管理守护进程创建Orphan VCPU。

任何现代CPU架构都至少有两种特权模式: User mode和Supervisor Mode。User mode的权限最低，Supervisor模式的具有更高特权。Xvisor在User Mode下运行Normal VCPU，在Supervisor Mode下运行Orphan VCPU。

下面的图-1给出了由Xvisor实现的系统虚拟机模型的清晰图



### 2.Hypervisor配置

Xvisorss使用设备树来配置。

默认情况下，Xvisor将始终支持使用DTS配置设备树。它还包括一个取自Linux内核源代码的DTC编译器和一个可供体系结构使用的轻量级DTB解析库（libfdt），用于填充Xvisor设备树的特定代码（或板特定代码）

### 3. Hypervisor Timer

与任何操作系统不同，对于虚拟机监控程序来说，计时问题尤其严重，因为存在许多挑战。最明显的问题是，时间现在在主机和可能的许多guest实例之间共享。来宾操作系统永远不会得到100%的CPU执行时间，尽管事实上，它很可能会做出这样的假设。当中断源被禁用时，它可能期望它保持非常精确的界限，但实际上只有它的虚拟中断源被禁用，并且机器在任何时候都可能被抢占。这会导致问题，因为实时的通过、来宾中断的注入以及相关的时钟源不再与实时完全同步。

与任何操作系统不同，对于虚拟机监控程序来说，计时问题尤其严重，因为存在许多挑战。最明显的问题是，时间现在在主机和可能的许多来宾实例之间共享。来宾操作系统永远不会得到100%的CPU执行时间，尽管事实上，它很可能会做出这样的假设。当中断源被禁用时，它可能期望它保持非常精确的界限，但实际上只有它的虚拟中断源被禁用，并且机器在任何时候都可能被抢占。这会导致问题，因为实时的通过、来宾中断的注入以及相关的时钟源不再与实时完全同步。

有三种方法可以解决这个问题：

1.对于有单独时间源跟踪“挂钟”或“实时”的客人，可以忽略它，因为他们可能不需要对中断进行任何调整以保持适当的时间。

2.如果这还不够，可能需要向来宾系统注入额外的中断，以提高有效中断率。这种方法在极端条件下会导致复杂性，因为主机负载或客户延迟太大而无法补偿。

3.客人可能需要意识到蜱虫的损失，并在内部进行补偿。虽然在理论上很有希望，但在Linux中实现此策略非常容易出错，并且在常用的Linux系统中分布了许多错误的丢失勾号补偿变体。

从上面可以清楚地看出，系统管理程序必须以低开销、高精度和高分辨率来跟踪时间的流逝(例如，我们不能计算跟踪时间的周期性中断)。简单地说，管理程序中的计时必须是无痒且高分辨率的。此外，管理程序中的PIT仿真器可能不得不保留挂起的周期性中断的积压。

Xvisor的系统管理程序计时器子系统高度受Linux hrtimer子系统的启发，是完全无痒的。它提供了以下特性：

1。64位时间戳:时间戳表示Xvisor启动后经过的纳秒。(即Xvisor的正常运行时间以纳秒计)

2。计时器事件:我们可以创建或销毁带有相关的以纳秒为单位的过期时间的计时器事件和过期回调处理程序。时间事件是一次性事件(即，当它们到期时，它们会自动停止)，要有周期性计时器事件，我们必须从它的到期回调处理程序手动重新启动计时器事件。

hypervisor计时器需要特定于体系结构的代码来为每个主机CPU提供一个全局时钟源设备和一个时钟事件设备，以提供上述特性。

### 4. Hypervisor Manager

Xvisor中的VCPU和Guest实例由Hypervisor Manager创建和管理。它还提供了用于VCPU状态更改、VCPU统计信息和VCPU主机CPU更改的例程，这些例程构建在hypervisor调度程序例程之上。

与任何操作系统一样，Xvisor中的VCPU实例也有一个依赖于体系结构的部分和一个独立于体系结构的部分。

VCPU上下文的架构依赖部分包括:

1。Arch寄存器:仅在用户模式(或非特权模式)下由处理器更新的寄存器。这些寄存器通常是通用寄存器和由处理器自动更新的状态标志(例如比较标志、溢出标志、零标志等)。普通和孤儿vcpu都需要自己的拷贝寄存器或arch寄存器。我们将VCPU的arch寄存器称为VCPU结构中的“arch\_regs\_t \*regs”成员。

2。Arch Private:仅在管理模式(或特权模式)下由处理器更新的寄存器。当普通VCPU尝试读/写这样的寄存器时，我们会得到一个异常，我们可以返回/更新它的虚值。在大多数情况下，也有一些额外的数据结构(如MMU上下文，影子TLB，影子页表，…等等)。孤儿vcpu通常不需要arch private，只有普通vcpu才需要它们。我们将VCPU的arch private称为VCPU结构中的“void \*arch\_priv”成员。

VCPU上下文由以下内容组成：

1。ID:全局唯一标识号

2。SUBID:在父Guest中唯一的标识号。(仅适用于Normal vcpu)

3 。Name:该VCPU的名称。(仅限Orphan vcpu)

4。设备树节点:指向VCPU设备树节点的指针。(仅适用于Normal vcpu)

5。Is Normal:表示该VCPU是Normal还是Orphan的标志。

6。Is PowerOff: Guest复位后VCPU是否应该处于RESET状态的标志。(仅适用于Normal vcpu)

7。Guest:指向父Guest的指针。

8。Start PC:程序计数器的启动值。

9。Stack VA:堆栈指针的起始值。对于孤儿VCPU(或线程)，这将是运行时堆栈，而对于普通VCPU，这将是处理异常和中断的特殊堆栈。

10。Stack Size: VCPU堆栈的大小。

11。Sched Lock:用于保护动态调度器上下文的读写自旋锁。

12。主机CPU: VCPU运行的主机CPU。(使用调度锁保护)

----调度程序动态上下文----

13。主机CPU亲和性:表示VCPU可以运行的主机CPU的掩码。(使用调度锁保护)

14。State:当前VCPU的状态。(使用调度锁保护)

15。State Timestamp: VCPU进入当前状态的时间戳。(使用调度锁保护)

16。State Ready Nanosecs: VCPU处于Ready状态所花费的时间。(使用调度锁保护)

17。State Running Nanosecs: VCPU处于Running状态所花费的时间。(使用调度锁保护)

18。State Paused Nanosecs: VCPU处于暂停状态所花费的时间。(使用调度锁保护)

19。State suspended Nanosecs: VCPU处于suspended状态所花费的时间。(使用调度锁保护)

20。Reset Count: VCPU复位次数。(使用调度锁保护)

21。Reset Timestamp: VCPU处于Reset状态的时间戳。(使用调度锁保护)

22。Preempt Count: VCPU禁用抢占的次数。(使用调度锁保护)

23。Resumed: VCPU在READY或RUNNING状态下进入暂停状态前是否恢复的标志。(使用调度锁保护)

24。Private Scheduling Context:该VCPU的调度策略的私有上下文。(使用调度锁保护)

----调度程序静态上下文----

25。静态调度参数:VCPU创建时提供的调度参数，用于调度策略。(例如，priority, time\_slice, deadline，和periodicity)

26。架构特定的上下文:这个VCPU的架构特定的上下文。特定于体系结构的代码负责管理此上下文。

27。虚拟的硬中断请求优先级别信息。: VCPU虚拟中断管理信息

(例如计数assert, deassert, pending和execute)

28。Waitqueue信息。:等待队列所需的信息。

29。设备仿真上下文:指向每个VCPU的设备仿真框架所需的私有信息的指针。

VCPU可以在任何给定的timer实例中处于一种状态。下面是对所有可能状态的简要描述:

\* UNKNOWN: VCPU不属于任何Guest，不是孤儿VCPU。为了降低内存占用，我们根据最大vcpu数量预分配内存，并将它们置于此状态。

\* RESET: VCPU已初始化，正在等待某人将其踢到READY状态。为了创建新的VCPU, VCPU调度器从预分配的VCPU中获取一个处于UNKNOWN状态的VCPU并初始化它。初始化后，新创建的VCPU将处于RESET状态。

\* READY: VCPU准备在硬件上运行。

\* RUNNING: VCPU当前在硬件上运行。

\* PAUSED:VCPU已经停止，稍后可以恢复。当检测到VCPU是空闲的并且可以被调度出去时，VCPU就会在这种状态下设置(通常是由架构特定的代码)。

\* suspended: VCPU已经停止，无法恢复。当某个VCPU进行错误访问时，VCPU就会处于这种状态(通常由特定于体系结构的代码)。

VCPU状态的改变可能发生在不同的位置，比如特定于体系结构的代码、一些hypervisor线程、调度程序、一些仿真设备等。

不可能列出需要更改VCPU状态的所有可能场景的详尽列表，但是VCPU状态更改必须严格遵循由hypervisor调度程序确保的有限状态机

VCPU状态变化的有限状态机入下图所示：

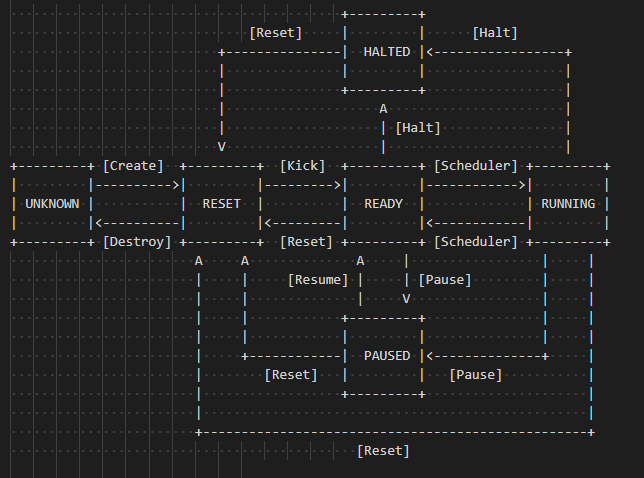


图2

### 5.Hypervisor Scheduler

Xvisor的hypervisor调度器在调度策略（或调度算法）方面是通用的和可插入的。每当从虚拟机监控程序管理器收到有关VCPU状态更改的通知时，它都会更新每个CPU就绪队列。

虚拟机监控程序调度器使用每CPU虚拟机监控程序计时器事件为VCPU分配时间片。当CPU的调度程序计时器事件过期时，调度程序将使用某些调度策略（或算法）查找下一个VCPU，并为下一个VCPU配置调度程序计时器事件。

对于Xvisor而言，正常的VCPU是一个黑匣子（即任何东西都可能在VCPU上运行），异常或中断是恢复控制的唯一方法。无论何时执行Xvisor代码，我们都可能处于以下任一上下文中：

1.IRQ上下文：为主机的某个外部设备生成的中断提供服务时。

2.正常上下文：在Xvisor中模拟某些功能或指令或代表正常VCPU模拟IO时。

3.孤立上下文：当以孤立VCPU或线程的形式运行Xvisor的某些部分时（注意：虚拟机监控程序线程将在后面介绍。）

与其他虚拟机监控程序不同，Xvisor有一个称为普通上下文的特殊上下文。只有当虚拟机监控程序代表普通VCPU执行某些操作（如处理异常、模拟IO等）时，虚拟机监控程序才处于正常上下文中。

正常上下文是不可睡眠的，这意味着正常VCPU在正常上下文中时无法调度。事实上，只有当Xvisor退出IRQ上下文或正常上下文时，才会调度出正常的VCPU。这有助于Xvisor在处理异常或模拟IO时确保可预测的延迟。

调度器在特定于体系结构的异常或中断处理程序的帮助下跟踪当前执行上下文。

特定于体系结构的VCPU上下文切换所涉及的预期高级步骤如下：

1. 将堆栈中的arch寄存器（或arch regs）保存（由特定于体系结构的异常或中断处理程序保存）到当前的VCPU arch寄存器（或arch regs）。
2. 还原堆栈上下一个VCPU的arch寄存器（或arch\_regs\_t）（从异常或中断处理程序C代码返回时将还原）。
3. 切换特定于体系结构的CPU资源的上下文，如MMU、浮点子系统等。
4. 调度程序调用VCPU上下文开关的可能场景

详情如下:

1. 当分配给当前VCPU的时间片过期时，我们调用VCPU上下文开关。我们称这种情况为VCPU抢占。
2. 如果正常VCPU出现错误（即寄存器/内存访问无效），则特定于体系结构的代码可以检测到这种情况，并使用hypervisor manager中的API暂停/暂停负责的正常VCPU。
3. 孤立VCPU（或线程）选择自动暂停（即睡眠）。
4. 孤立的VCPU（或线程）选择自动生成其时间片。
5. 还可以使用虚拟机监控程序管理器API从某些其他VCPU更改VCPU状态。

我们可以从Xvisor menuconfig选项中选择不同的调度策略（或算法）。任何调度策略（或算法）将获得每个VCPU的以下调度信息：

1. 优先级：VCPU的优先级。值越高，优先级越高。
2. 时间片：VCPU计划完成后必须获得的最小时间量（以纳秒为单位）。
3. 截止日期：VCPU必须计划运行的最长时间（以纳秒为单位）。
4. 周期性：VCPU准备就绪或VCPU开始工作的速率（以纳秒为单位）。

目前，可用的调度策略（或算法）有：

1. 固定优先级循环（PRR）
2. 固定优先级最早截止日期优先（PEDF）

（注意：默认情况下，Xvisor使用固定优先级循环策略）

### 6. Hypervisor 负载均衡

除了在每个主机CPU上调度任务，通用操作系统或hypervisor的调度器还必须在SMP主机上的多个主机CPU上负载平衡任务数量。

与传统的通用操作系统不同，负载平衡不是由Xvisor的hypervisor调度器完成的。相反，Xvisor hypervisor调度器为每个主机CPU保留单独的上下文，并且只执行每个主机CPU的调度工作。在多个主机cpu上平衡vcpu(正常或孤儿)的工作是由Xvisor中称为hypervisor负载平衡器的独立实体完成的。

hypervisor负载均衡器实现为孤儿VCPU(或线程)，运行在固定的主机cpu上。它将向系统管理器提供关于在创建VCPU时为给定VCPU分配“hcpu”的提示。它还将以几秒为间隔周期性地调用，以平衡主机cpu之间的vcpu。此外，系统管理程序负载平衡器的负载平衡策略是运行时可插拔的。目前，我们有“粗略的”负载平衡策略，即基于主机CPU利用率来平衡vcpu。Xvisor的用户可以根据自己的用例或主机硬件编写自己的负载均衡策略。

综上所述，Xvisor的hypervisor负载平衡器是一个可插拔的惰性工作线程，用于在多个主机cpu之间平衡vcpu

### 7. Hypervisor Threads

Xvisor中用于管理后台线程的线程框架称为Hypervisor线程。Xvisor中的线程与孤儿VCPU没有什么不同，实际上每个线程都是孤儿VCPU的包装器。线程的最佳例子是我们的管理终端(mterm)。

要在Xvisor中创建一个线程，我们需要5个必备事项:

1。名称:分配给该线程的名称。

2。函数:线程入口函数指针。

3。Data:指向要作为参数传递给线程入口函数的任意数据的Void指针。

4。优先级:线程(或底层孤儿VCPU)的优先级。值越高，优先级越高。

5。时间片:线程(或底层孤儿VCPU)在调度后必须获得的最小时间，单位为纳秒。

我们不需要为每个线程显式地创建堆栈，因为hypervisor管理器会在创建VCPU时自动为每个孤儿VCPU(即线程)创建固定大小的堆栈。所有线程的默认堆栈大小可以在编译时使用Xvisor menuconfig选项更改。

线程ID、优先级和时间片与底层野VCPU的ID、优先级和时间片相同。

线程在任何时候都可以处于以下状态之一:

\* CREATED:线程是新创建的，它还没有运行。

\* RUNNING:线程正在运行。

\* SLEEPING:线程正在等待队列中睡觉。

\* STOPPED:线程已经停止。它要么被强行制止，要么任务已经完成。

(注意:孤儿VCPU状态可以直接映射到一个线程状态，因此，为了获得当前线程状态，我们查看底层孤儿VCPU的状态。)

对于线程间的同步，我们有以下同步原语:

1。自旋锁:通常用于较小的临界区，用于IRQ上下文、普通上下文和孤儿上下文之间的同步。

2。完成:通常在线程(或孤儿VCPU)想要等待事件(例如从主机设备中断)发生时使用。

3所示。信号量:传统的信号量锁，它允许线程(或孤儿VCPU)在锁(或资源)不可用时休眠。

4所示。互斥锁:传统的互斥锁，当锁不可用时，允许线程(或孤儿VCPU)休眠。

从上面的完成，信号量，和互斥使用Xvisor等待队列的睡眠。只有一个线程(或者孤儿VCPU)可以在一个等待队列中休眠，因为我们不能在IRQ和Normal上下文中休眠。由于Completion、Semaphore和Mutex的可休眠操作只能在孤儿上下文中完成。

### 8. Device Driver Framework

Xvisor设备驱动程序框架在抽象和可用API方面与Linux内核设备驱动程序模型非常相似。这种与Linux设备驱动程序模型的相似性有助于Xvisor为设备驱动程序移植提供Linux兼容性头。我们还有类似Linux的设备资源管理API，Xvisor使用这些API跟踪各种设备驱动程序使用的主机资源。

以下是Xvisor设备驱动程序框架定义的实体：

vmm\_bus

多个设备可以驻留的总线的逻辑表示。例如平台、usb、spi、i2c等。

vmm\_class

由集合实现的功能类的逻辑表示设备的数量。

例如，vmm\_chardev、vmm\_blockdev、vmm\_netport、vmm\_netwtich等。

vmm\_device

驻留在总线上或是类的一部分但不是两者的设备的逻辑表示。设备可以是其他设备的子设备，也可以有自己的子设备。如果设备X驻留在总线上并实现A类和B类的功能，那么我们将拥有伪设备XA和XB，它们是A类和B类的一部分，父设备为X。

vmm\_driver

驻留在总线上的设备的设备驱动程序的逻辑表示。我们只能将设备驱动程序注册到总线，而不能注册到类。

上述定义实体的实例在运行时由各种Xvisor模块注册。Xvisor中的默认总线是平台总线，这是一个伪总线，用于通过Xvisor设备树探测所有设备。vmm\_chardev是Xvisor中始终可用的默认类，因为标准IO子系统（稍后描述）和命令管理器（稍后描述）严重依赖于字符设备。

### 9. Device Emulation Framework

设备仿真框架是任何虚拟机监控程序中最关键的组件之一。它帮助管理程序为GUEST提供特定的虚拟硬件。Xvisor设备仿真框架设计灵活、重量轻、速度快。Xvisor设备仿真框架最重要的实体是：vmm\_emulator和vmm\_emudev。vmm\_emulator是由设备仿真器模块注册的设备仿真器的逻辑表示，而vmm\_emulator是任何仿真/传递设备的逻辑表示。

在guest创建时，虚拟机监控程序管理器将在设备仿真框架的帮助下为每个虚拟和直通guest区域创建一个vmm\_emudev实例。框架将尝试为每个新创建的vmm\_emudev实例探测匹配的vmm\_仿真器，如果框架未能找到匹配的vmm\_仿真器，或者如果vmm\_仿真器的探测函数返回错误，则来宾创建将失败。

除上述内容外，Xvisor设备仿真框架还通过提供固定数量的guest irq线，为中断控制器仿真器和GPIO控制器仿真器提供了特殊支持。可以在guest创建时通过guest设备树指定guest irq行的总数。中断控制器和GPIO控制器仿真器都将提供回调函数，以监控某些来宾irq行的级别变化。中断控制器仿真器将根据guest irq线的电平变化触发guest VCPU中断，而GPIO控制器仿真器将根据guest irq线的电平变化执行某些操作。

vmm\_emulator的所有读/写回调都在正常上下文中调用，因此设备模拟器无法在读/写回调中休眠或使用可休眠锁，如果绝对需要休眠，那么设备模拟器将必须使用后台工作线程（或孤立VCPU）。这种“设备仿真环境下无睡眠”有助于确保设备仿真中可预测的延迟，这对实时系统非常重要。

### 10. Standard I/O

任何通用操作系统或实时操作系统都需要一种打印和扫描ascii文本的方法。Xvisor也不例外，所以我们在Xvisor中有标准的IO子系统，它实现了各种形式的打印和扫描api。

标准I/O子系统需要以下条件来完成其任务:

1. “defterm”功能来自于体系结构特定的代码

打印和扫描ascii文本的默认方法是使用体系结构特定代码提供的defterm函数。与体系结构相关的代码将为defterm提供init、getc和putc函数。这些“缺省”功能对于所有体系结构都是强制性的，但是体系结构特定的代码可以选择提供存根实现。

2. “defterm early”从体系结构特定的代码中打印函数

标准I/O子系统是作为启动过程的一部分初始化的。甚至在标准I/O子系统初始化之前，就已经完成了很多初始化，所以我们从架构特定的代码中确定了早期打印功能，如果Xvisor的某些部分试图在标准I/O子系统初始化之前打印ascii文本，就会调用该功能。独立于体系结构的代码包含了缺陷早期打印函数的一个弱实现，因此提供该函数对于体系结构特定的代码是完全可选的。一般来说，“defterm early”打印函数仅用于调试，在大多数架构中，默认情况下应该禁用它。

3.字符设备

标准I/O子系统可以从任何字符设备实例(例如vmm\_chardev)打印/扫描字符。只有在没有为标准I/O设置字符设备实例时，它才会使用"defterm"函数。可以使用来自管理守护进程的各种标准I/O命令或通过调用标准I/O子系统的更改设备API为标准I/O设置字符设备。对于标准I/O的所有交互，并不是必须使用相同的字符设备集。实际上，标准I/O子系统提供了api来通过特定的字符设备实例打印或扫描字符，该实例可能与已经为标准I/O设置的字符设备实例不同。

除此之外，标准I/O子系统还提供了大量的调试宏和堆栈跟踪打印api。要打印堆栈跟踪，标准I/O子系统将再次依赖于体系结构特定的代码。

### 11. Command Manager

作为一个单片hypervisor, Xvisor需要命令行接口，但是命令行接口有几种传输介质，例如:

1. 在串行端口

2. 基于VT-100的图形控制台

3.通过网络Telnet连接

4. -----更多-----

Xvisor命令管理器提供了独立于传输的方式来管理和执行命令，这样我们就可以在不同的管理守护进程之间共享Xvisor命令，而无需进行任何更改。最重要的是，它提供了api，用于在指定的字符设备上通过命令的输入-输出执行命令字符串。它还提供了用于注册、注销和管理命令的api。

有几种类型的命令可通过命令管理器，如:

1. 架构特性的命令

2. 通用命令

3. 虚拟I/O相关命令

4. 设备驱动相关命令

5. 网络命令

6. 文件系统命令

7. -----更多-----

Xvisor命令主要代码在路径：xvisor\commands

后续分析

### 12. Storage Virtualization

Xvisor中的存储虚拟化非常简单和轻量级。它有两个关键实体:vmm\_blockdev和vmm\_vdisk。

Xvisor中的块设备框架将是Xvisor中存储虚拟化最关键的部分。vmm\_blockdev是在Xvisor设备驱动框架的块类下注册的块设备实例的逻辑表示。每个vmm\_blockdev都与一个由vmm\_request\_queue实体表示的请求队列相关联。vmm\_blockdev上的所有IO都是异步的，并以vmm\_request实体的形式提交。vmm\_blockdev的分区将表示为子vmm\_blockdev实例，子vmm\_blockdev实例具有与父vmm\_blockdev相同的vmm\_request\_queue。

文件系统不是实现存储虚拟化的必需条件。事实上，Xvisor中的文件系统库(称为“VFS”)是完全可选的，仅用于加载客户映像、脚本和日志记录。

Xvisor中的磁盘控制器仿真器为来宾实例的每个虚拟磁盘创建一个vmm\_vdisk实例。vmm\_vdisk是vmm\_blockdev实例之上的逻辑包装器，必须在客户创建时或在运行时使用Xvisor命令将其附加到vmm\_blockdev实例。如果绑定到vmm\_vdisk的vmm\_blockdev在运行时未注册，那么它将自动从vmm\_vdisk卸载。如果vmm\_vdisk没有附加到任何vmm\_blockdev，所有对vmm\_vdisk的IO请求都会失败。此外，vmm\_vdisk块大小必须是vmm\_blockdev块大小的倍数。

### 13. Network Virtualization

Xvisor中的网络虚拟化以轻量级分组交换框架的形式提供。它主要提供使用某些包交换策略共享主机网络接口的抽象。对于Xvisor来说，网络堆栈(或netstack或网络套接字库)是完全可选的，因为我们不需要全面的网络堆栈来提供网络虚拟化。只有基于网络的管理守护进程需要网络套接字api，在大多数情况下，我们可以禁用这些守护进程。

总体而言，Xvisor网络支持包括四个关键组成部分:

1. 网络切换框架

(在:& lt; xvisor\_source&gt /核心/净)

Xvisor网络背后的主要思想是拥有一个快速分组交换框架。组网核心实现vmm\_mbuf、vmm\_netswitch和vmm\_netport。

vmm\_mbuf是包的类似bsd的表示形式。它是一个非常通用的包表示，事实上我们可以用Xvisor vmm\_mbuf来表示Linux sk\_buff。

vmm\_netswitch是一个模拟的网络交换机，可以有多个vmm\_netports连接到它。vmm\_netswitch可以有不同的策略，如:集线器、网桥、路由器、VLAN交换机等。目前，我们有mac级桥接策略和HUB/Repeater策略可用。可以在Xvisor引导时使用给定的策略或使用来自管理终端的命令创建vmm\_netswitch。

vmm\_netport是vmm\_netswitch和驱动程序、模拟器或netstack之间的逻辑连接。没有连接到任何vmm\_netswitch的vmm\_netport将丢弃数据包。

2. 网络设备驱动程序

(在:& lt; xvisor\_source&gt /司机/净)

主机网络设备驱动程序将创建vmm\_netport并将其连接到vmm\_netswitch。在这种情况下，vmm\_netport的MAC地址将与主机网络设备的实际MAC地址相同。用于移植网络设备驱动程序的Linux兼容api将提供:使用vmm\_netport的"struct net\_device"和使用vmm\_mbuf的"struct sk\_buff"。

我们从不给主机网络设备的vmm\_netports分配任何IP地址。这些vmm\_netports以混杂模式运行，接受任何目的IP地址的数据包。vmm\_netswitch将根据其交换策略决定转发数据包的vmm\_netports。

3.网络设备模拟器

(在:& lt; xvisor\_source&gt /模拟器/净)

网络设备仿真器还创建vmm\_netport并将其连接到vmm\_netswitch。在此vmm\_netport上接收的所有报文都将被来宾操作系统接收，来宾操作系统发送的所有报文都将通过此vmm\_netport传输。vmm\_netport的MAC地址是在客户创建时指定的，或者使用随机数生成的。vmm\_netport的IP地址是在Guest操作系统中分配的，Xvisor不知道它。

4. 可选网络堆栈(或netstack或网络套接字库)

(在:& lt; xvisor\_source&gt / libs / netstack)

如上所述，网络堆栈(或netstack或网络套接字库)是完全可选的，我们可以在Xvisor中集成任何GPLv2兼容的网络堆栈，只要它实现了在libs/include/libs/netstack.h中定义的api。

通常，网络堆栈将使用一个或多个连接到vmm\_netswitch的伪vmm\_netports来实现。伪vmm\_netports的MAC地址可以在Xvisor引导时指定，也可以使用随机数生成。目前，我们选择了lwIP作为可选的网络堆栈，将来我们可能会做一些其他的事情，但libs/netstack.h api将保持不变。

当在Xvisor中启用网络堆栈时，它还需要为每个vmm\_netports提供一个IP地址和其他网络设置，因此我们有“net”命令。我们还可以从外部或Guest操作系统“ping”分配给网络堆栈vmm\_netports的IP地址。

### 14. Virtualized I/O Frameworks

除了存储和网络虚拟化，还有许多类型的I/O需要虚拟化来更好地共享主机硬件，例如:

1. 串行端口

2. 输入设备(如键盘、鼠标、多点触控、操纵杆等)

3.显示设备(如帧缓冲器、LED等)

4. USB设备

5. CAN总线

6. 传感器设备(如GPS、温度、陀螺仪等)

7. -----更多-----

我们不可能对上述所有设备都有一个通用的虚拟化框架，所以在Xvisor中，我们有基于I/O设备类型的专门虚拟化框架。这些虚拟I/O框架充当客户模拟设备和主机设备之间的桥梁，从而支持在多个客户实例之间共享主机设备。

目前，我们有以下几个虚拟I/O框架:

**vmm\_vserial**

虚拟串口子系统由vmm\_vserial和vmm\_vserial\_receiver两个实体组成。vmm\_vserial是虚拟串口的逻辑表示。来宾串口仿真器为每个来宾串口创建一个vmm\_vserial实例。所有发送到vmm\_vserial实例的字符都由相应的guest接收，所有从guest接收的字符都由vmm\_vserial实例接收。串口捕获守护进程将vmm\_vserial\_receiver实例注册到vmm\_vserial实例，以便从vmm\_vserial实例接收字符。这些守护进程可以使用发送api将字符发送到vmm\_serial实例。

**vmm\_vinput**

虚拟输入子系统由vmm\_vkeyboard和vmm\_vmouse两个实体组成。来宾键盘仿真器将为每个来宾键盘创建一个vmm\_vkeyboard实例，而来宾鼠标仿真器将为每个来宾鼠标创建一个vmm\_vmouse实例。Display守护进程可以将按键事件和鼠标移动事件分别注入到vmm\_vkeyboard实例和vmm\_vmouse实例。vmm\_vkeyboard实例和vmm\_vmouse实例接收到的所有事件都作为虚拟按键事件和虚拟鼠标移动事件注入来宾操作系统。

**vmm\_vdisplay**

虚拟显示子系统有两个重要的实体:vmm\_vdisplay和vmm\_surface。GUI呈现守护进程(例如VNC守护进程或VScreen守护进程等等)创建vmm\_surface实例，并将其添加/绑定到vmm\_vdisplay实例。多个GUI呈现守护进程可以将它们的vmm\_surface实例添加到单个vmm\_vdisplay实例中。GUI呈现守护进程还将更新vmm\_vdisplay的api，以定期使用vmm\_vdisplay实例更新/同步vmm\_surface实例。显示仿真器创建vmm\_vdisplay实例来模拟虚拟显示。这些显示模拟器还将使用vmm\_vdisplay的曲面相关api，向vmm\_surface实例提供有关虚拟显示更改的提示。

## Xvisor启动流程

### 启动log

OpenSBI v0.9-154-g9283d50

\_\_\_\_ \_\_\_\_\_ \_\_\_\_ \_\_\_\_\_

/ \_\_ \ / \_\_\_\_| \_ \\_ \_|

| | | |\_ \_\_ \_\_\_ \_ \_\_ | (\_\_\_ | |\_) || |

| | | | '\_ \ / \_ \ '\_ \ \\_\_\_ \| \_ < | |

| |\_\_| | |\_) | \_\_/ | | |\_\_\_\_) | |\_) || |\_

\\_\_\_\_/| .\_\_/ \\_\_\_|\_| |\_|\_\_\_\_\_/|\_\_\_\_/\_\_\_\_\_|

| |

|\_|

Platform Name : riscv-virtio,qemu

Platform Features : medeleg

Platform HART Count : 1

Platform IPI Device : aclint-mswi

Platform Timer Device : aclint-mtimer @ 10000000Hz

Platform Console Device : uart8250

Platform HSM Device : ---

Platform Reboot Device : sifive\_test

Platform Shutdown Device : sifive\_test

Firmware Base : 0x80000000

Firmware Size : 248 KB

Runtime SBI Version : 0.3

Domain0 Name : root

Domain0 Boot HART : 0

Domain0 HARTs : 0\*

Domain0 Region00 : 0x0000000002000000-0x000000000200ffff (I)

Domain0 Region01 : 0x0000000080000000-0x000000008003ffff ()

Domain0 Region02 : 0x0000000000000000-0xffffffffffffffff (R,W,X)

Domain0 Next Address : 0x0000000080200000

Domain0 Next Arg1 : 0x0000000082200000

Domain0 Next Mode : S-mode

Domain0 SysReset : yes

Boot HART ID : 0

Boot HART Domain : root

Boot HART ISA : rv64imafdcsuh

Boot HART Features : scounteren,mcounteren,time

Boot HART PMP Count : 16

Boot HART PMP Granularity : 4

Boot HART PMP Address Bits: 54

Boot HART MHPM Count : 0

Boot HART MIDELEG : 0x0000000000000666

Boot HART MEDELEG : 0x0000000000f0b509

Xvisor v0.3.0-224-g217edf76 (Oct 18 2021 13:59:21)

INIT: host address space

INIT: vapool: base=0x0000000010000000 size=33554432

INIT: vapool: hkbase=0x0000000010151000 hksize=131072

INIT: ram: bank0 phys=0x0000000080000000 size=1073741824

INIT: ram: bank0 hkbase=0x0000000010171000 hksize=32768

INIT: ram\_reserve: phys=0x0000000080200000 size=856064

INIT: ram\_reserve: phys=0x0000000080351000 size=393216

INIT: ram\_reserve: phys=0x00000000802D1000 size=524288

INIT: vapool\_reserve: virt=0x0000000010000000 size=1773568

INIT: ram\_reserve: phys=0x0000000082200000 size=8192

INIT: heap management

INIT: device tree

INIT: device tree reserved-memory

INIT: ram\_reserve: phys=0x0000000080000000 size=262144

INIT: DMA heap management

INIT: CPU nascent

INIT: SBI specification v0.3 detected

INIT: SBI implementation ID=0x1 Version=0x9

INIT: SBI TIME extension detected

INIT: SBI IPI extension detected

INIT: SBI RFENCE extension detected

INIT: SBI SRST extension detected

INIT: board nascent

INIT: nascent funtions

INIT: page pool

INIT: exception table

INIT: discover secondary CPUs

INIT: using SBI SMP operations

INIT: per-CPU areas

INIT: CPU hotplug

INIT: host irq subsystem

INIT: riscv-intc: registered 64 local interrupts

INIT: plic: plic@c000000: devices=54 contexts=1/2

INIT: CPU early

INIT: early\_params: vmm.bootcmd="vfs mount initrd /; vfs run /boot.xscript;vfs cat /system/banner.txt"

INIT: riscv-sbi-ipi: registered IPI domain

INIT: board early

INIT: early funtions

INIT: standard I/O

INIT: clocksource manager

INIT: riscv-timer: registered clocksource @ 10000000Hz

INIT: clockchip manager

INIT: hypervisor timer

INIT: hypervisor soft delay

INIT: hypervisor shared memory

INIT: hypervisor manager

INIT: synchronus inter-processor interrupts

INIT: hypervisor scheduler

INIT: asynchronus inter-processor interrupts

INIT: hypervisor threads

INIT: workqueue framework

INIT: wallclock subsystem

INIT: start secondary CPUs

INIT: command manager

INIT: device driver framework

INIT: device emulation framework

INIT: character device framework

INIT: iommu framework

INIT: hypervisor modules

INIT: block device framework

INIT: network switch framework

INIT: network port framework

initrd: RBD created at 0x88200000 - 0x8a200000

INIT: CPU final

INIT: board final

INIT: final functions

INIT: CPU0 online

INIT: brought-up 1 CPUs

INIT: freeing init memory 28K

INIT: change stdio device to uart@10000000

INIT: syncup wallclock using rtc@101000

INIT: bootcmd: vfs mount initrd /

Trying: cpio ext4

Mounted initrd using ext4 at /

INIT: bootcmd: vfs run /boot.xscript

Created default shared memory

guest0: Created

guest0: Parsing /images/riscv/virt64/nor\_flash.list

guest0: Loading 0x0000000000000000 with file ./firmware.bin

guest0: Loaded 0x0000000000000000 with 73892 bytes

guest0: Loading 0x00000000000FF000 with file ./cmdlist

guest0: Loaded 0x00000000000FF000 with 167 bytes

guest0: Loading 0x0000000000100000 with file ./Image

guest0: Loaded 0x0000000000100000 with 19814912 bytes

guest0: Loading 0x0000000001F00000 with file ./virt64.dtb

guest0: Loaded 0x0000000001F00000 with 2122 bytes

guest0: Loading 0x0000000002000000 with file ./rootfs.img

guest0: Loaded 0x0000000002000000 with 1546752 bytes

guest1: Created

guest1: Parsing /images/riscv/virt64/nor\_flash.list

guest1: Loading 0x0000000000000000 with file ./firmware.bin

guest1: Loaded 0x0000000000000000 with 73892 bytes

guest1: Loading 0x00000000000FF000 with file ./cmdlist

guest1: Loaded 0x00000000000FF000 with 167 bytes

guest1: Loading 0x0000000000100000 with file ./Image

guest1: Loaded 0x0000000000100000 with 19814912 bytes

guest1: Loading 0x0000000001F00000 with file ./virt64.dtb

guest1: Loaded 0x0000000001F00000 with 2122 bytes

guest1: Loading 0x0000000002000000 with file ./rootfs.img

guest1: Loaded 0x0000000002000000 with 1546752 bytes

INIT: bootcmd: vfs cat /system/banner.txt

ooooooo ooooo oooooo oooo ooooo .oooooo..o .oooooo. ooooooooo.

`8888 d8' `888. .8' `888' d8P' `Y8 d8P' `Y8b `888 `Y88.

Y888..8P `888. .8' 888 Y88bo. 888 888 888 .d88'

`8888' `888. .8' 888 `Y8888o. 888 888 888ooo88P'

.8PY888. `888.8' 888 `Y88b 888 888 888`88b.

d8' `888b `888' 888 oo .d8P `88b d88' 888 `88b.

o888o o88888o `8' o888o 8''88888P' `Y8bood8P' o888o o888o

XVisor# guest kick guest0

guest0: Kicked

XVisor# vserial bind guest0/uart0

[guest0/uart0] RISC-V Virt64 Basic Firmware

[guest0/uart0]

[guest0/uart0] autoboot: disabled

[guest0/uart0]

[guest0/uart0] basic# autoexec

[guest0/uart0] autoexec(copy 0x80200000 0x00100000 0x1E00000)

[guest0/uart0] copy took 61533 usecs for 0x1E00000 bytes

[guest0/uart0] autoexec(copy 0x83000000 0x01F00000 0x020000)

[guest0/uart0] copy took 943 usecs for 0x020000 bytes

[guest0/uart0] autoexec(copy 0x83100000 0x02000000 0x800000)

[guest0/uart0] copy took 14180 usecs for 0x800000 bytes

[guest0/uart0] autoexec(start\_linux\_fdt 0x80200000 0x83000000 0x83100000 0x800000)

[guest0/uart0] Trying to set new bootargs: root=/dev/ram rw console=ttyS0,115200 earlycon=uart8250,mmio,0x10000000 mem=256M

[guest0/uart0] Jumping into linux ...

[guest0/uart0] [ 0.000000] Linux version 5.14.7 (wangfei@HP-DEV) (riscv64-unknown-linux-gnu-gcc (GCC) 10.1.0, GNU ld (GNU Binutils) 2.35) #1 SMP Mon Oct 18 13:59:35 CST 2021

[guest0/uart0] [ 0.000000] OF: fdt: Ignoring memory range 0x80000000 - 0x80200000

[guest0/uart0] [ 0.000000] Machine model: Virt64

[guest0/uart0] [ 0.000000] earlycon: uart8250 at MMIO 0x0000000010000000 (options '')

[guest0/uart0] [ 0.000000] printk: bootconsole [uart8250] enabled

[guest0/uart0] [ 0.000000] Memory limited to 256MB

[guest0/uart0] [ 0.000000] efi: UEFI not found.

[guest0/uart0] [ 0.000000] Reserved memory: created DMA memory pool at 0x0000000058000000, size 16 MiB

[guest0/uart0] [ 0.000000] OF: reserved mem: initialized node shmem@58000000, compatible id shared-dma-pool

[guest0/uart0] [ 0.000000] Zone ranges:

[guest0/uart0] [ 0.000000] DMA32 [mem 0x0000000080200000-0x000000008fffffff]

[guest0/uart0] [ 0.000000] Normal empty

[guest0/uart0] [ 0.000000] Movable zone start for each node

[guest0/uart0] [ 0.000000] Early memory node ranges

[guest0/uart0] [ 0.000000] node 0: [mem 0x0000000080200000-0x000000008fffffff]

[guest0/uart0] [ 0.000000] Initmem setup node 0 [mem 0x0000000080200000-0x000000008fffffff]

[guest0/uart0] [ 0.000000] SBI specification v0.3 detected

[guest0/uart0] [ 0.000000] SBI implementation ID=0x2 Version=0x3000

[guest0/uart0] [ 0.000000] SBI TIME extension detected

[guest0/uart0] [ 0.000000] SBI IPI extension detected

[guest0/uart0] [ 0.000000] SBI RFENCE extension detected

[guest0/uart0] [ 0.000000] SBI v0.2 HSM extension detected

[guest0/uart0] [ 0.000000] riscv: ISA extensions acdfim

[guest0/uart0] [ 0.000000] riscv: ELF capabilities acdfim

[guest0/uart0] [ 0.000000] percpu: Embedded 19 pages/cpu s39016 r8192 d30616 u77824

[guest0/uart0] [ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 64135

[guest0/uart0] [ 0.000000] Kernel command line: root=/dev/ram rw console=ttyS0,115200 earlycon=uart8250,mmio,0x10000000 mem=256M

[guest0/uart0] [ 0.000000] Dentry cache hash table entries: 32768 (order: 6, 262144 bytes, linear)

[guest0/uart0] [ 0.000000] Inode-cache hash table entries: 16384 (order: 5, 131072 bytes, linear)

[guest0/uart0] [ 0.000000] Sorting \_\_ex\_table...

[guest0/uart0] [ 0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off

[guest0/uart0] [ 0.000000] Virtual kernel memory layout:

[guest0/uart0] [ 0.000000] fixmap : 0xffffffcefee00000 - 0xffffffceff000000 (2048 kB)

[guest0/uart0] [ 0.000000] pci io : 0xffffffceff000000 - 0xffffffcf00000000 ( 16 MB)

[guest0/uart0] [ 0.000000] vmemmap : 0xffffffcf00000000 - 0xffffffcfffffffff (4095 MB)

[guest0/uart0] [ 0.000000] vmalloc : 0xffffffd000000000 - 0xffffffdfffffffff (65535 MB)

[guest0/uart0] [ 0.000000] lowmem : 0xffffffe000000000 - 0xffffffe00fe00000 ( 254 MB)

[guest0/uart0] [ 0.000000] kernel : 0xffffffff80000000 - 0xffffffffffffffff (2047 MB)

[guest0/uart0] [ 0.000000] Memory: 227260K/260096K available (7405K kernel code, 4895K rwdata, 4096K rodata, 2166K init, 307K bss, 32836K reserved, 0K cma-reserved)

[guest0/uart0] [ 0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=2, Nodes=1

[guest0/uart0] [ 0.000000] rcu: Hierarchical RCU implementation.

[guest0/uart0] [ 0.000000] rcu: RCU restricting CPUs from NR\_CPUS=8 to nr\_cpu\_ids=2.

[guest0/uart0] [ 0.000000] rcu: RCU debug extended QS entry/exit.

[guest0/uart0] [ 0.000000] Tracing variant of Tasks RCU enabled.

[guest0/uart0] [ 0.000000] rcu: RCU calculated value of scheduler-enlistment delay is 25 jiffies.

[guest0/uart0] [ 0.000000] rcu: Adjusting geometry for rcu\_fanout\_leaf=16, nr\_cpu\_ids=2

[guest0/uart0] [ 0.000000] NR\_IRQS: 64, nr\_irqs: 64, preallocated irqs: 0

[guest0/uart0] [ 0.000000] riscv-intc: 64 local interrupts mapped

[guest0/uart0] [ 0.000000] plic: interrupt-controller@c000000: mapped 32 interrupts with 2 handlers for 4 contexts.

[guest0/uart0] [ 0.000000] random: get\_random\_bytes called from start\_kernel+0x4fa/0x720 with crng\_init=0

[guest0/uart0] [ 0.000000] riscv\_timer\_init\_dt: Registering clocksource cpuid [0] hartid [0]

[guest0/uart0] [ 0.000000] clocksource: riscv\_clocksource: mask: 0xffffffffffffffff max\_cycles: 0x24e6a1710, max\_idle\_ns: 440795202120 ns

[guest0/uart0] [ 0.000084] sched\_clock: 64 bits at 10MHz, resolution 100ns, wraps every 4398046511100ns

[guest0/uart0] [ 0.009225] Console: colour dummy device 80x25

[guest0/uart0] [ 0.017552] Calibrating delay loop (skipped), value calculated using timer frequency.. 20.00 BogoMIPS (lpj=40000)

[guest0/uart0] [ 0.026518] pid\_max: default: 32768 minimum: 301

[guest0/uart0] [ 0.032297] Mount-cache hash table entries: 512 (order: 0, 4096 bytes, linear)

[guest0/uart0] [ 0.039606] Mountpoint-cache hash table entries: 512 (order: 0, 4096 bytes, linear)

[guest0/uart0] [ 0.075056] ASID allocator disabled

[guest0/uart0] [ 0.079030] rcu: Hierarchical SRCU implementation.

[guest0/uart0] [ 0.085061] EFI services will not be available.

[guest0/uart0] [ 0.091277] smp: Bringing up secondary CPUs ...

[guest0/uart0] [ 0.103634] smp: Brought up 1 node, 2 CPUs

[guest0/uart0] [ 0.121309] devtmpfs: initialized

[guest0/uart0] [ 0.132655] clocksource: jiffies: mask: 0xffffffff max\_cycles: 0xffffffff, max\_idle\_ns: 7645041785100000 ns

[guest0/uart0] [ 0.143592] futex hash table entries: 512 (order: 3, 32768 bytes, linear)

[guest0/uart0] [ 0.154812] NET: Registered PF\_NETLINK/PF\_ROUTE protocol family

[guest0/uart0] [ 0.200793] wait\_for\_initramfs() called before rootfs\_initcalls

[guest0/uart0] [ 0.228903] vgaarb: loaded

[guest0/uart0] [ 0.232772] SCSI subsystem initialized

[guest0/uart0] [ 0.238521] usbcore: registered new interface driver usbfs

[guest0/uart0] [ 0.243568] usbcore: registered new interface driver hub

[guest0/uart0] [ 0.249087] usbcore: registered new device driver usb

[guest0/uart0] [ 0.289329] clocksource: Switched to clocksource riscv\_clocksource

[guest0/uart0] [ 0.298666] simple-framebuffer 50000000.simplefb: framebuffer at 0x50000000, 0x1000000 bytes

[guest0/uart0] [ 0.306631] simple-framebuffer 50000000.simplefb: format=a8r8g8b8, mode=1600x900x32, linelength=6400

[guest0/uart0] [ 0.336956] Console: switching to colour frame buffer device 200x56

[guest0/uart0] [ 0.352923] simple-framebuffer 50000000.simplefb: fb0: simplefb registered!

[guest0/uart0] [ 0.452823] NET: Registered PF\_INET protocol family

[guest0/uart0] [ 0.458160] IP idents hash table entries: 4096 (order: 3, 32768 bytes, linear)

[guest0/uart0] [ 0.468375] tcp\_listen\_portaddr\_hash hash table entries: 128 (order: 0, 5120 bytes, linear)

[guest0/uart0] [ 0.476404] TCP established hash table entries: 2048 (order: 2, 16384 bytes, linear)

[guest0/uart0] [ 0.483447] TCP bind hash table entries: 2048 (order: 4, 65536 bytes, linear)

[guest0/uart0] [ 0.489436] TCP: Hash tables configured (established 2048 bind 2048)

[guest0/uart0] [ 0.496149] UDP hash table entries: 256 (order: 2, 24576 bytes, linear)

[guest0/uart0] [ 0.501907] UDP-Lite hash table entries: 256 (order: 2, 24576 bytes, linear)

[guest0/uart0] [ 0.509550] NET: Registered PF\_UNIX/PF\_LOCAL protocol family

[guest0/uart0] [ 0.520583] RPC: Registered named UNIX socket transport module.

[guest0/uart0] [ 0.525639] RPC: Registered udp transport module.

[guest0/uart0] [ 0.530403] RPC: Registered tcp transport module.

[guest0/uart0] [ 0.534469] RPC: Registered tcp NFSv4.1 backchannel transport module.

[guest0/uart0] [ 0.540572] PCI: CLS 0 bytes, default 64

[guest0/uart0] [ 0.548913] Unpacking initramfs...

[guest0/uart0] [ 0.575574] workingset: timestamp\_bits=62 max\_order=16 bucket\_order=0

[guest0/uart0] [ 0.613969] NFS: Registering the id\_resolver key type

[guest0/uart0] [ 0.619394] Key type id\_resolver registered

[guest0/uart0] [ 0.633073] Key type id\_legacy registered

[guest0/uart0] [ 0.637060] nfs4filelayout\_init: NFSv4 File Layout Driver Registering...

[guest0/uart0] [ 0.653055] nfs4flexfilelayout\_init: NFSv4 Flexfile Layout Driver Registering...

[guest0/uart0] [ 0.671130] 9p: Installing v9fs 9p2000 file system support

[guest0/uart0] [ 0.678546] NET: Registered PF\_ALG protocol family

[guest0/uart0] [ 0.693612] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 251)

[guest0/uart0] [ 0.699834] io scheduler mq-deadline registered

[guest0/uart0] [ 0.714156] io scheduler kyber registered

[guest0/uart0] [ 0.901294] Freeing initrd memory: 8192K

[guest0/uart0] [ 0.922379] virtio\_rpmsg\_bus virtio3: rpmsg host is online

[guest0/uart0] [ 0.939328] input: Xvisor VirtIO Input as /devices/platform/smb/20004000.virtio\_input/virtio4/input/input0

[guest0/uart0] [ 1.454056] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled

[guest0/uart0] [ 1.471241] printk: console [ttyS0] disabled

[guest0/uart0] [ 1.476628] 10000000.uart: ttyS0 at MMIO 0x10000000 (irq = 1, base\_baud = 230400) is a 16550A

[guest0/uart0] [ 1.486408] printk: console [ttyS0] enabled

[guest0/uart0] [ 1.486408] printk: console [ttyS0] enabled

[guest0/uart0] [ 1.495003] printk: bootconsole [uart8250] disabled

[guest0/uart0] [ 1.495003] printk: bootconsole [uart8250] disabled

[guest0/uart0] [ 1.511991] [drm] radeon kernel modesetting enabled.

[guest0/uart0] [ 1.546516] loop: module loaded

[guest0/uart0] [ 1.553343] virtio\_blk virtio1: [vda] 0 512-byte logical blocks (0 B/0 B)

[guest0/uart0] [ 1.567214] libphy: Fixed MDIO Bus: probed

[guest0/uart0] [ 1.582090] e1000e: Intel(R) PRO/1000 Network Driver

[guest0/uart0] [ 1.588073] e1000e: Copyright(c) 1999 - 2015 Intel Corporation.

[guest0/uart0] [ 1.593688] ehci\_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver

[guest0/uart0] [ 1.599833] ehci-pci: EHCI PCI platform driver

[guest0/uart0] [ 1.604115] ehci-platform: EHCI generic platform driver

[guest0/uart0] [ 1.609378] ohci\_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver

[guest0/uart0] [ 1.615064] ohci-pci: OHCI PCI platform driver

[guest0/uart0] [ 1.619483] ohci-platform: OHCI generic platform driver

[guest0/uart0] [ 1.625511] usbcore: registered new interface driver uas

[guest0/uart0] [ 1.631255] usbcore: registered new interface driver usb-storage

[guest0/uart0] [ 1.649715] mousedev: PS/2 mouse device common for all mice

[guest0/uart0] [ 1.659423] goldfish\_rtc 10003000.rtc: registered as rtc0

[guest0/uart0] [ 1.669486] goldfish\_rtc 10003000.rtc: setting system clock to 2021-10-20T00:42:46 UTC (1634690566)

[guest0/uart0] [ 1.677954] sdhci: Secure Digital Host Controller Interface driver

[guest0/uart0] [ 1.683448] sdhci: Copyright(c) Pierre Ossman

[guest0/uart0] [ 1.687317] sdhci-pltfm: SDHCI platform and OF driver helper

[guest0/uart0] [ 1.693183] usbcore: registered new interface driver usbhid

[guest0/uart0] [ 1.697965] usbhid: USB HID core driver

[guest0/uart0] [ 1.703648] NET: Registered PF\_INET6 protocol family

[guest0/uart0] [ 1.716386] Segment Routing with IPv6

[guest0/uart0] [ 1.720042] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver

[guest0/uart0] [ 1.728201] NET: Registered PF\_PACKET protocol family

[guest0/uart0] [ 1.733693] 9pnet: Installing 9P2000 support

[guest0/uart0] [ 1.738305] Key type dns\_resolver registered

[guest0/uart0] [ 1.744570] debug\_vm\_pgtable: [debug\_vm\_pgtable ]: Validating architecture page table helpers

[guest0/uart0] [ 1.785342] Freeing unused kernel image (initmem) memory: 2164K

[guest0/uart0] [ 1.804512] Run /init as init process

[guest0/uart0] \_ \_

[guest0/uart0] | ||\_|

[guest0/uart0] | | \_ \_\_\_\_ \_ \_ \_ \_

[guest0/uart0] | || | \_ \| | | |\ \/ /

[guest0/uart0] | || | | | | |\_| |/ \

[guest0/uart0] |\_||\_|\_| |\_|\\_\_\_\_|\\_/\\_/

[guest0/uart0]

[guest0/uart0] Busybox Rootfs

[guest0/uart0]

[guest0/uart0] Please press Enter to activate this console.

[guest0/uart0] / #

[guest0/uart0] / #

### 启动函数流程：

Vmm.bin

链接脚本路径xvisor\arch\riscv\cpu\generic\linker.ld

程序入口ENTRY(\_start)，\_start定义在xvisor\arch\riscv\cpu\generic\cpu\_entry.S

函数主调用关系：

\_start E:\code\xvisor-next\arch\riscv\cpu\generic\cpu\_entry.S

cpu\_init E:\code\xvisor-next\arch\riscv\cpu\generic\cpu\_init.c

vmm\_init E:\code\xvisor-next\core\vmm\_main.c

vmm\_smp\_set\_bootcpu

vmm\_init\_done

vmm\_smp\_is\_bootcpu

init\_bootcpu

vmm\_set\_cpu\_possible

vmm\_set\_cpu\_present

vmm\_host\_aspace\_init

vmm\_heap\_init

vmm\_devtree\_init

vmm\_devtree\_reserved\_memory\_init

vmm\_dma\_heap\_init

arch\_cpu\_nascent\_init

arch\_board\_nascent\_init

vmm\_initfn\_nascent

vmm\_pagepool\_init

vmm\_extable\_init

arch\_smp\_init\_cpus

arch\_smp\_prepare\_cpus

vmm\_percpu\_init

vmm\_cpuhp\_init

vmm\_cpuhp\_set\_state(VMM\_CPUHP\_STATE\_ONLINE)

vmm\_devtree\_getnode(VMM\_DEVTREE\_PATH\_SEPARATOR\_STRING

VMM\_DEVTREE\_GUESTINFO\_NODE\_NAME);

vmm\_devtree\_addnode(NULL, VMM\_DEVTREE\_GUESTINFO\_NODE\_NAME);

vmm\_devtree\_getnode(VMM\_DEVTREE\_PATH\_SEPARATOR\_STRING

VMM\_DEVTREE\_VMMINFO\_NODE\_NAME);

vmm\_devtree\_addnode(NULL, VMM\_DEVTREE\_VMMINFO\_NODE\_NAME);

vmm\_host\_irq\_init();

arch\_cpu\_early\_init();

arch\_board\_early\_init

vmm\_initfn\_early

vmm\_stdio\_init

vmm\_clocksource\_init

vmm\_clockchip\_init

vmm\_timer\_init

vmm\_delay\_init

vmm\_shmem\_init

vmm\_manager\_init

vmm\_smp\_sync\_ipi\_init

vmm\_scheduler\_init

vmm\_smp\_async\_ipi\_init

vmm\_threads\_init

vmm\_workqueue\_init

@INIT\_WORK(&sys\_init, &system\_init\_work);

@vmm\_workqueue\_schedule\_work(NULL, &sys\_init);

vmm\_wallclock\_init

@vmm\_smp\_bootcpu\_id

@arch\_smp\_start\_cpu

vmm\_cmdmgr\_init

vmm\_devdrv\_init

vmm\_devemu\_init

vmm\_chardev\_init

vmm\_cpu\_online 循环allcpu

vmm\_iommu\_init

vmm\_modules\_init

arch\_cpu\_final\_init

arch\_board\_final\_init

vmm\_initfn\_final

@INIT\_WORK(&sys\_postinit, &system\_postinit\_work);

@vmm\_workqueue\_schedule\_work(NULL, &sys\_postinit);

在system\_postinit\_work中

vmm\_host\_free\_initmem

vmm\_devtree\_getnode(VMM\_DEVTREE\_PATH\_SEPARATOR\_STRING

VMM\_DEVTREE\_CHOSEN\_NODE\_NAME);

console\_param\_process(console\_param);

vmm\_stdio\_change\_device(cdev);// "change stdio device to %s\n"

rtcdev\_param\_process(rtcdev\_param);

**bootcmd\_param\_process(bootcmd\_param, bootcmd\_param\_len);**

vmm\_timer\_start

### 文件系统

bootcmd\_param\_process(bootcmd\_param, bootcmd\_param\_len);执行bootcmd

#### Qemu启动的命令：

qemu-system-riscv64 -cpu rv64,x-h=true -M virt -m 1G -nographic -bios <opensbi\_source\_directory>/build/platform/generic/firmware/fw\_jump.bin -**kernel ./build/vmm.bin** **-initrd ./build/disk.img** -append 'vmm.bootcmd="**vfs mount initrd /;vfs run /boot.xscript;vfs cat /system/banner.txt**"'

vmm.bin 是xvisor编译的bin

disk.img是xvisor挂载的根文件系统，它包含如下内容：

disk/

├── boot.xscript----------disk挂载后运行的脚本，通过vmm.bootcmd指定

├── images

│   └── riscv

│   ├── virt64

│   │   ├── Image-----guest linux内核

│   │   ├── cmdlist----进入basic firmware后atuoexec缩写命令执行的命令列表

│   │   ├── firmware.bin---xvisor引导guest os的固件

│   │   ├── nor\_flash.list----虚拟存储的映射文件

│   │   ├── rootfs.img-----guest linux根文件系统

│   │   └── virt64.dtb------xvisor自身的设备树

│   └── virt64-guest.dtb-----guest linux设备树

├── system

│   ├── banner.txt

│   └── logo.ppm

└── tmp

#### 文件系统挂载和脚本执行

在xvisor启动过程中，先运行了boot.xscript脚本,为后续xvisor启动guest linux os做了准备。

boot.xscript主要包含4条指令：

# Create default shared memory

**shmem create default 0x1000000 21**

# Load guest0 device tree from file

**vfs guest\_fdt\_load guest0 /images/riscv/virt64-guest.dtb 2 mem0,physical\_size,physsize,0x10000000 net0,switch,string,br0 shmem0,shared\_mem,string,default**

# Create guest0

**guest create guest0**

# Load guest0 images

**vfs guest\_load\_list guest0 /images/riscv/virt64/nor\_flash.list**

nor\_flash.list文件内容：

0x00000000 ./firmware.bin

0x000FF000 ./cmdlist

0x00100000 ./Image

0x01F00000 ./virt64.dtb

0x02000000 ./rootfs.img

（上面涉及到了hypervisor配置，manager，command manager，存储虚拟化等模块，后续实现需要进一步分析）

Xvisor启动guest os还需要在xvisor console中执行几个命令：

[15. Kick Guest0 to start Basic Firmware]

XVisor# **guest kick guest0**

[16. Bind to virtual UART]

XVisor# **vserial bind guest0/uart0**

[17. Copy linux from NOR flash to RAM and start linux booting from RAM] [guest0/uart0] **basic# autoexec**

注释：autoexec 是一个缩写命令，它会执行cmdlist中的一系列命令,主要目的是将虚拟flash中的镜像拷贝到内存。

Cmdlist文件内容为：

copy 0x80200000 0x00100000 0x1E00000----拷贝Image

copy 0x83000000 0x01F00000 0x020000-----拷贝设备树

copy 0x83100000 0x02000000 0x800000-----拷贝根文件系统

start\_linux\_fdt 0x80200000 0x83000000 0x83100000 0x800000—启动guest linux

（疑问：0x80200000不是物理地址，2个guset os可以拷贝到同一地址，后续要重点研究下存储虚拟化，内存虚拟化的实现）

Ps:Guest 命令行源码路径：E:\code\xvisor-next\commands\cmd\_guest.c

cmd\_guest\_create

疑问：

每个命令都执行什么操作？如何就启动linux了？guest os运行在哪个特权模式，xvisor运行在哪个特权模式，切换的时候特权模式如何切换的？

Firmware.bin相关代码：xvisor \tests目录

wangfei@HP-DEV:~/githypercisor/nbuild/work/xvisor/tests$ tree

.└── riscv

└── virt64

└── basic

├── arch\_board.o

├── arch\_cache.o

├── arch\_entry.o

├── arch\_irq.o

├── arch\_linux.o

├── arch\_mmu.o

├── basic\_heap.o

├── basic\_irq.o

├── basic\_main.o

├── basic\_stdio.o

├── basic\_string.o

├── dhry\_1.o

├── dhry\_2.o

├── dhry\_port.o

├── display

│   └── simplefb.o

├── firmware.bin---包含在xvisor disk.img中，用于启动guest linux

├── firmware.elf

├── firmware.lnk

├── libfdt-----设备树相关

│   ├── fdt.o

│   ├── fdt\_ro.o

│   ├── fdt\_rw.o

│   ├── fdt\_strerror.o

│   ├── fdt\_support.o

│   ├── fdt\_sw.o

│   └── fdt\_wip.o

├── pic

│   └── riscv\_intc.o

├── serial

│   └── uart8250.o

├── sys

│   └── vminfo.o

└── timer

└── riscv\_timer.o

firmware.bin用于引导guest linux。那它是如何运行的？

#### Basic firmware命令行

[guest0/uart0] basic# help

[guest0/uart0] help - List commands and their usage

[guest0/uart0]

[guest0/uart0] hi - Say hi to basic firmware

[guest0/uart0]

[guest0/uart0] hello - Say hello to basic firmware

[guest0/uart0]

[guest0/uart0] **wfi\_test** - Run wait for irq instruction test for basic firmware

[guest0/uart0] Usage: wfi\_test [<msecs>]

[guest0/uart0] <msecs> = delay in milliseconds to wait for

[guest0/uart0]

[guest0/uart0] **mmu\_setup** - Setup MMU for basic firmware

[guest0/uart0]

[guest0/uart0] **mmu\_state** - MMU is enabled/disabled for basic firmware

[guest0/uart0]

[guest0/uart0] **mmu\_test**  - Run MMU test suite for basic firmware

[guest0/uart0]

[guest0/uart0] **mmu\_cleanup** - Cleanup MMU for basic firmware

[guest0/uart0]

[guest0/uart0] **timer** - Display timer information

[guest0/uart0]

[guest0/uart0] **dhrystone** - Dhrystone 2.1 benchmark

[guest0/uart0] Usage: dhrystone [<iterations>]

[guest0/uart0]

[guest0/uart0] **hexdump** - Dump memory contents in hex format

[guest0/uart0] Usage: hexdump <addr> <count>

[guest0/uart0] <addr> = memory address in hex

[guest0/uart0] <count> = byte count in hex

[guest0/uart0]

[guest0/uart0] **copy** - Copy to target memory from source memory

[guest0/uart0] Usage: copy <dest> <src> <count>

[guest0/uart0] <dest> = destination address in hex

[guest0/uart0] <src> = source address in hex

[guest0/uart0] <count> = byte count in hex

[guest0/uart0]

[guest0/uart0] **start\_linux\_fdt** - Start linux kernel (device-tree mechanism)

[guest0/uart0] Usage: start\_linux\_fdt <kernel\_addr> <fdt\_addr> [<initrd\_addr>] [<initrd\_size>]

[guest0/uart0] <kernel\_addr> = kernel load address

[guest0/uart0] <fdt\_addr> = fdt blob address

[guest0/uart0] <initrd\_addr> = initrd load address (optional)

[guest0/uart0] <initrd\_size> = initrd size (optional)

[guest0/uart0]

[guest0/uart0] **fdt\_override\_u32** - Overrides an integer property in the device tree

[guest0/uart0] Usage: fdt\_override\_u32 <fdt\_addr> </path/to/property> <value>

[guest0/uart0]

[guest0/uart0] **fdt\_print\_tree** - Print device tree

[guest0/uart0] Usage: fdt\_print\_tree <fdt\_addr> </path/to/node>

[guest0/uart0]

[guest0/uart0] **linux\_cmdline** - Show/Update linux command line

[guest0/uart0] Usage: linux\_cmdline [<new\_linux\_cmdline>]

[guest0/uart0] <new\_linux\_cmdline> = linux command line

[guest0/uart0]

[guest0/uart0] **linux\_memory\_size** - Show/Update linux memory size

[guest0/uart0] Usage: linux\_memory\_size [<memory\_size>]

[guest0/uart0] <memory\_size> = memory size in hex

[guest0/uart0]

[guest0/uart0] **autoexec** - autoexec command list from flash

[guest0/uart0] Usage: autoexec

[guest0/uart0]

[guest0/uart0] **go** - Jump to a given address

[guest0/uart0] Usage: go <addr>

[guest0/uart0] <addr> = jump address in hex

[guest0/uart0]

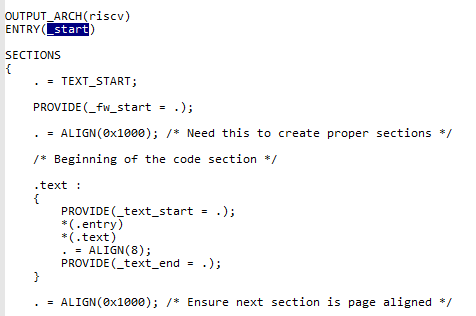
[guest0/uart0] **reset** - Reset the system

[guest0/uart0]

[guest0/uart0] basic#

#### Basic firmware 编译和连接

链接脚本：xvisor \tests\riscv\common\basic\firmware.ld



Firmware.bin的入库地址\_start，在汇报文件

xvisor \tests\riscv\common\basic\arch\_entry.S中定义

#### Basic firmware启动流程

\_start

basic\_init

basic\_main

## Xvisor中断模块分析

相关代码

E:\code\xvisor-next\core\vmm\_host\_irq.c

E:\code\xvisor-next\core\vmm\_host\_irqdomain.c

E:\code\xvisor-next\core\vmm\_host\_irqext.c

E:\code\xvisor-next\arch\riscv\cpu\generic\cpu\_entry.S

E:\code\xvisor-next\arch\riscv\cpu\generic\cpu\_exception.c

E:\code\xvisor-next\arch\riscv\cpu\generic\cpu\_proc.S

---多核相关处理arch\_smp\_id和proc\_setup\_smp\_id，没看明白

E:\code\xvisor-next\arch\riscv\cpu\generic\cpu\_smp\_ops.c---多核处理，还没分析

## 关注的问题

1.理解xvisor启动顺序（包含xvisor，x86，arm）

2.虚拟机guest的切换机制，guest切换xvisor，guest0和1如何切换，在寄存器要做的事

3.多核中guest的分配方法，调度算法

4.外设虚拟化，其他各种外设

5.2级内存的映射机制（os到实际物理地址）

6.中断虚拟化，riscv规范未说详细，关注xvisor实现

7.iommu，外设dma的支持和实现

## 参考文档：

1. Xvisor的官方网址：<http://www.xhypervisor.org/>
2. < xvisor code path> /docs /riscv/riscv64-qemu.txt
3. < xvisor code path> /docs/DesignDoc