软件愈加多样性对嵌入式系统的影响

Maarten Koning
技术办公室
风河
美国阿拉米达
Maarten.koning@windriver.com

摘要—*复杂嵌入式系统的软件组成，如汽车、机器人和医疗设备，正经历着巨大变化。这些设备中的应用程序越来越多地与其定制操作系统实例集成、封装在一起。这一趋势始于IT域，IT应用程序率先利用虚拟机和容器；如今，嵌入式系统领域也显现了这一趋势。但是，这种变化对嵌入式系统构建方式的动机和影响却是截然不同的。随着具有不同核心价值和需求的离散多核嵌入式系统搭载越来越多的软件，一个操作系统实例甚至有着多个实例的操作系统都无法为多应用程序运行提供最佳的运行环境。可针对特定操作系统（如Linux）或特定商用RTOS开发应用程序，且可分别与开源和认证系统一起不断发展或淘汰某些应用程序。这些应用程序也可能有多个来源，因此出于支持、许可或发布间隔等原因需要对其进行分区。本文探讨了一种全新的异构分布式系统架构，该架构应用复杂而强大的多核SoC嵌入式系统硬件；还探讨了这对于创造、交付和获取嵌入式软件价值的公司的意义与影响。*

关键词—异构系统，RTOS，多核技术，虚拟化，软件分区，Hypervisor，安全性

I. 引言

智能系统中的软件内容数量呈指数增长，抵消了摩尔定律提出的晶体管增速。对于为嵌入式系统设计的硬件而言，这些晶体管用于额外的内核处理、缓存和其他内存技术以及IO——如今都位于单个多核封装中。包含此类完整计算机实现的系统统称为系统级芯片（System-on-Chip，简称SoC）。过去，离散多核处理器更为通用，且可适配SMP操作系统，以管理、调解跨多个应用程序的硬件资源。尽管概念简单，但类似方法并不总可以扩展至如今用于嵌入式系统的异构多核复杂SoC。目前尚有许多软件工程挑战；随着专用嵌入式系统软件内容数量的增加，挑战日益棘手。

在软件开发阶段和系统运行期间，若多个程序或应用程序需要运行于单个操作系统中，使用单一SMP　OS的方法就会存在诸多挑战。在软件开发阶段，这些应用程序需要迁移到同类SMP OS 。同时，无论是文件系统、IPC机制，亦或是网络堆栈等空间，这些应用程序都需要避免在操作系统域名空间中出现冲突。系统运行期间，受操作系统调度、内存或磁盘碎片、缓存或TLB冲突以及其他资源干扰，可能会出现拒绝服务等问题。

此外，应用程序的生命周期与单一操作系统的生命周期息息相关，因为重启操作系统时，无论应用程序的重要性如何，所有应用程序也都必须重新启动。由于安全补丁或系统软件优化更新，或硬件故障（如内存故障），正常系统操作期间也可能会发生此类操作系统重启。若将此视为系统事件，则所有应用程序都与单一操作系统的整体生命周期相关，这与系统架构师的期望相悖，他们期望较小的故障域而非全局故障域。

单一操作系统方法使得信息安全与功能安全也面临挑战。若对托管信息安全和/或功能安全应用程序的操作系统有严格要求，那么这些要求会向下映射回溯至操作系统，因为不安全的操作系统本身就会导致应用程序故障以及应用程序攻击。即使操作系统的功能和信息安全级别满足应用程序的要求，代价也会更高，因为必须将操作系统信息安全和/或功能安全范围中的所有代码（甚至是功能安全或信息安全应用程序未使用的功能代码）写入相同的安全级别-。这与关注问题隔离、特权最小化和故障域最小化的设计最佳实践相背离。

[1] 若可能，将Linux中应用程序标准化为SMP OS的趋势，在某种程度上缓解了这一挑战。

性能复杂性值得关注。随着单个SMP OS可用内核数量的增加，对于单核操作系统调用性能或跨多核的并发操作系统调用，阿姆达尔定律也出现问题。这是因为SMP OS必须选择合适的锁粒度来保护共享操作系统数据结构。若SMP OS选择细粒度的锁，则由于在系统调用的内核实现过程中持有以及释放锁的数量过多，由单个线程或单个处理器核衡量的操作系统调用性能会变慢。若使用粗粒度锁实现SMP OS，则可以最快速度实现无争议的单个操作系统调用。但是，由于在其他地方锁定时间较长，因此在其他不同处理器核上线程的并发性会受到影响。

尽管中等粒度锁是明显的折衷方案，但是阿姆达尔定律向我们展示了，即便锁引入的串行化极为少量，但仍是昂贵机制，它会干扰系统并发性能力——尤其是随着理论上可用的并发数量增长，这一影响更为凸显。阿姆达尔定律指出，所有计算任务都包含串行工作；根据任务中的串行化量，由于多核并发导致的任务增速量可表示为：增速= 1 /（序列化+（1-序列化）/ N），其中N是无干扰并发执行环境，序列化是串行任务的比例（即0-1（含0和1））。

上述公式表明，由于SMP OS本身的串行性，使用单个SMP操作系统运行争用操作系统资源的多个应用程序，会抵消多核并发带来的好处。另外，如前文所述，若各应用程序对于操作系统要求（尤其是功能安全或信息安全操作）不同，则单一SMP OS方案也存在问题。另一种方法是在嵌入式系统中集成多个相同或不同类型的操作系统，该系统可将一个或多个应用程序分组以共享一个操作系统实例，或拥有一个专用于该应用程序的操作系统实例。

在IT和云领域中，在单个处理器上运行多个操作系统实例，每个实例专用于一个或多个应用程序的方式很常见。有两种虚拟化方法，基于硬件的虚拟化运行虚拟机和/或基于操作系统的虚拟化运行容器。这两种方法都为运行的应用程序提供了专用操作系统实例。利用虚拟化对系统分区的重要动机是能够实现应用程序间的生命周期开解和故障分离。此外，通过虚拟机或容器，系统工程师可拥有全新的有用实体来供应资源，以确保应用程序和系统获得可接受操作所需的内存、CPU和磁盘或网络带宽。这些技术现已应用于嵌入式系统，以管理系统复杂性，提高大型系统弹性，如在运行于复杂且强大的多核SoC上的嵌入式系统。

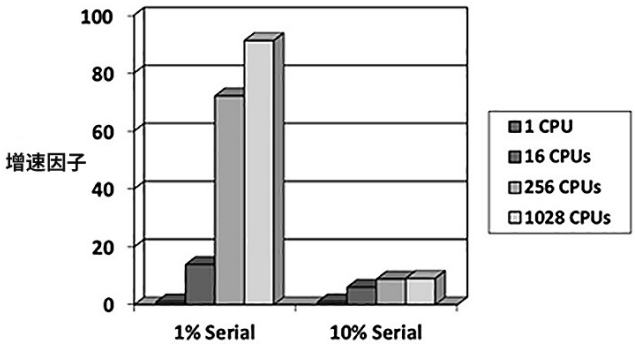

图1. 阿姆达尔定律

II. 分区

前文提及的将系统分为多个操作系统实例的动机包括：1）启用**混合关键性**系统；2）**防止故障传播**；3）**保证不同OS各自的核心优势**，如实时性或高安全认证等级要求；4）**分离特权**和硬件访问权以最小化攻击面；5）为软件模块设计**资源调配**，以及6）为各应用程序启用**实时软件更新**，这些应用程序越来越多地绑定到私有操作系统实例。

除解决上述挑战外，其他原因也推动了分区的使用：1）**工作负载合并**，对原有的运行于多个离散计算模块的软件再设计，以使其在一个多核SoC上并行运行，2）**软件重用**，应用程序及其定制操作系统实例可在新的软件负载或产品变型中按原样重新部署，3）**组织所有权**，可使组织实体拥有、负责完整执行环境的操作，这包括其他组织实体有相同所有权和责任制要求环境下的操作系统和应用程序，以及4）分离某些通用平台服务（如**网络安全功能或系统管理功能**）的机制。

该分区嵌入式系统的一个重要最终结果是能够水平或垂直地单独部署和扩展应用程序。包含操作系统实例的分区系统可使**体系结构清晰、操作灵活**，因为它创建了一个可以运行各应用程序的执行环境，无需考虑其核心要求和操作系统挂钩。笔者称这种软件环境为“**集成平台**”。

需集成的软件规模越来越复杂，集成平台方法的一个优势在于，它可以将大型项目细分为更易于管理整合的小项目。例如，大型汽车**OEM已经为平台开发人员、应用程序开发人员和系统集成商**实现了角色分离，并且构建了一种系统架构可直接将此类角色与元素、接口和工具实现映射，且各元素、接口和工具都明确哪些可通过集成平台模型进行定制化修改，即使应用程序开发人员为系统集成商提供了操作系统和核心价值以将其整合至产品中。

III. 分区技术

应用程序分区技术可由弱分区到强分区来阐述。传统Unix风格的进程可用于承载多个独立应用程序，每个应用程序均由多个相互不干扰的活动实体（线程和进程）组成。这是通过“设计”先验AKA“白板分区”（whiteboard partitioning）完成的。但Unix风格操作系统提供的分区技术被视是“弱分区”，因为该操作系统并不强制应用程序互不干扰，反而在很大程度上都是约定俗成的，其中包括一些通用用户ID和GroupID的特定配置。

容器技术（如Docker和LXC等）利用的Linux控制组和域名空间实现“软分区”，通过提供操作系统虚拟化的主操作系统，软分区间存在干扰通道。随着更多基于硬件的功能可在硅片中实现，诸如KVM等Type 2（基于托管操作系统）hypervisor提供的硬件虚拟化也在稳步改善。通常将其称为“硬件虚拟化”或“虚拟分区”。 Type 1 hypervisor（直接运行于硬件，可支持硬件虚拟化）可实现“强分区”。由于Type 2 hypervisor意味着通过托管操作系统在分区间存在软件诱发的干扰通道，因此Type 2 hypervisor通常不被视为“强分区”。

值得注意的是，即便在系统处理器架构上正确实现Type 1 hypervisor以支持硬件虚拟化，若在硬件实施过程中有干扰，则强分区间仍存在干扰通道。例如，若各硬件虚拟化域间有共享缓存，或一个硬件虚拟化域发起的DMA可能会扰动另一硬件虚拟化域，但却未考虑hypervisor修复的可能性，则分区间仍存在潜在干扰。某些处理器体系结构考虑了此类干扰通道，比其他处理器体系更完整地实施了修复补救措施。因此，某些处理器体系结构比其他处理器体系结构更适合混合关键嵌入式系统。最后，即使存在此类基于硬件的干扰通道，type 1 hypervisor有时也可提供消除或最小化此类干扰通道的机制。对于以上举例，若关键虚拟机必须运行且不得受到基于硬件的干扰，则可暂停激活DMA的虚拟机或尽为（best-effort）虚拟机。尽管此类功能会为提高安全性和/或确定性而牺牲性能，但对于必须通过安全认证或满足严苛实时目标的系统中的Type 1 hypervisor，它们却是必备条件。

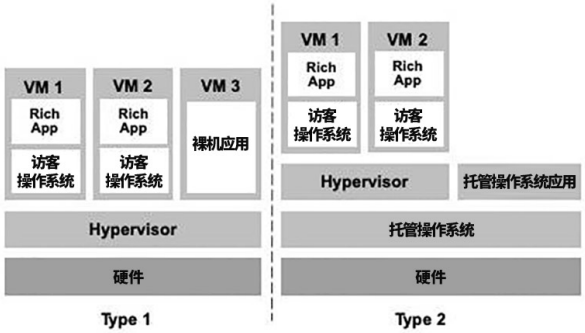

图2. Type 1与Type 2 Hypervisors对比

无论是Type 1还是Type 2 hypervisor，都有多种方法从虚拟机访问**物理设备**，包括直接访问hypervisor映射至虚拟机的设备特定的硬件寄存器。Hypervisor支持以下部分或全部功能：1）**直通设备**，其中设备被映射至虚拟机以直接访问和可选的设备中断； 2）**仿真设备**，其中物理设备接口以软件实现，因此可实现设备仲裁或设备分区等软件控制功能，以及 3）**虚拟设备**，其中可能存在也可能不存在用于实现虚拟驱动程序的物理设备。

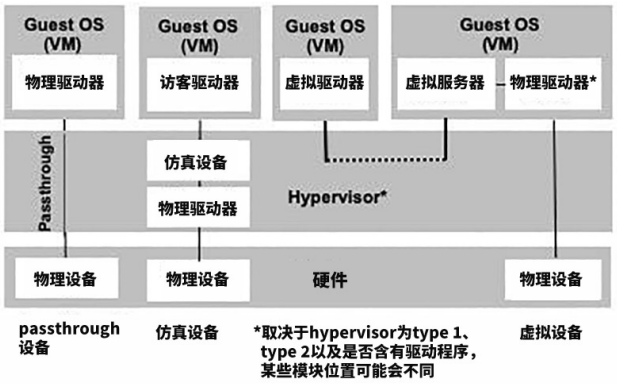

图3. 设备模型

某些 Type-1 hypervisor 提供最少的设备仿真和虚拟机调度，其主要重点是将硬件资源分区至虚拟机，强制执行这些分区，使用户基本上不知道Hypervisor的运行环境。这种主要专注于资源分区和分区执行的 Type-1 hypervisor 有时被称为 exokernels。安全可靠的多分区系统体系结构优化的一大潜力领域是将exokernel风格的hypervisor与裸机应用程序环境相结合，裸机应用程序环境包含单个地址空间和操作系统库（如RTOS或unikernel）。这种安排使各应用程序使用要求支持操作系统的功能，无需设置陷阱即可执行操作系统调用，从而实现以机器速度直接访问设备。

IV. 分布式系统

若基于多核的系统由多个分区组装而成，且每个分区包含一个或多个应用程序和一个操作系统实例，则整个运行环境是在芯片上运行的松散耦合分布式系统。该系统要求有用于各分区间通信的机制，以便在分区内运行的应用程序可彼此协作。尽管**共享内存**可行且处理速度快，但必须谨慎使用，以免在分区间引入干扰通道或故障传播路径。良好的系统架构会减少而非增加故障域和受攻击面。**单向共享内存**（共享内存区域只能由一个虚拟机写入）实现的基于共享内存的通信机制本质上比允许多个写入者的通信机制更安全，因为多个写入者间可能出现错误或故意重写。

基于共享内存分区间通信的替代方法是利用hypervisor，使用仿真设备或通过专用**超级调用（hypercall）**接口，在虚拟域间安全传递。引入基于**超级调用**的分区间通信的原因之一是针对混合关键性系统，以便将与安全相关通信软件实现的安全要求限制在安全分区和可验证的hypervisor实现。同时，非认证操作系统（如可能运行于同一系统的Linux或Windows实例）也可灵活的运行在同一系统中。

通过安全的分区间通信机制，一个分区的应用程序可利用另一分区管理的资源。此类资源包括文件系统、（如用于实时时钟或网络接口的）设备驱动程序以及操作程序服务（如服务名称和日志假脱机服务）。无论是在同一多核芯片中，还是通过总线连接的多个独立芯片中，对于在相同或不同hypervisor上运行的分区，或直接在相同或不同的CPU集群中硬件上运行的分区，都是如此。

V. 执行岛

商业多核芯片和软件环境采用术语“**安全岛**”和“**实时岛**”来描述用于安全相关和实时相关软件的硬件分区和软件分区执行环境。“**安全岛**”也用于与安全相关的专用软件硬件或软件分区，但并不常见。

硬件实现的安全岛或实时孤岛通常是一台具有内存、连接性和处理核的独立计算机，区别于较大的通用计算芯片。这些孤岛无需通用内核，即可作为不同的执行环境进行操作。有时，这样的孤岛处理器体系结构专用于通用内核，特别是当通用内核由于共享硬件引入的干扰通道而不再适用于此类用途时。

同样地，软件实现的安全性或实时孤岛也是有专用内存、连接性和处理核的独立计算环境。但软件管理程序或实时孤岛通常由hypervisor配置，使用处理器的硬件虚拟化功能，将通用内核中的软件实现的安全性或实时岛配置为不同的硬件分区。 实际上，仅由Type 1 hypervisor配置的安全和实时孤岛才有用；因为若由Type 2 hypervisor配置，则其安全性或实时要求，包括安全孤岛认证或实时孤岛的确定性 ，将沦为Type 2 hypervisor的标准，无法令人满意。当然情况可能会随时间而变化，有可能此类系统能够获得适度安全性和/或实时性，例如，Linux™在这些领域眼光长远，能力储备深厚，如 Linux Foundation的ELISA项目。

VI. 结语

现代强大的多核SoC硬件功能强大，已成为具有不同核心价值的软件集成平台，包括与特定操作系统变体挂钩的FOSS应用程序，与信息安全、功能安全或实时操作系统挂钩的软件，或裸机应用程序。由于集成了各种应用程序的多操作系统实例，因此这些应用程序需要基于软件或硬件的分区技术。若存在安全或实时要求，则可使用专用硬件孤岛或Type 1 hypervisor或两者组合实现。 对于共享内存总线的内核，hypervisor提供了更高的灵活性，因为它可用于例示任意数量的通用运行分区以及所有孤岛类别。

这样的体系结构进一步提升了**体系结构灵活性**，从而可更轻松地在产品版本和产品变型间更新和重用设计元素。由于分区的生命周期各不相同，因而还引入了**操作灵活性**这一概念；凭借松散的耦合和分布式系统体系结构，生命周期各不相同的分区可实现**连续集成**和**连续部署**，以及动态**水平和垂直扩展**。

尽管在应用程序+操作系统运行堆栈中引入hypervisor增加了第三操作层，但设一单独层专用于资源分区和分区实施的确将焦点集中于每一层的作用，并将通用计算、安全性和实时性分离出来，以便优化每类运行。尽管hypervisor会给系统带来中断延迟，但现代type 1虚拟化硬件可实现**直接中断**，并将其直接传递给客户，因而在系统架构中，hypervisor的引入并不会造成中断延迟。此外，hypervisor有**调试功能**，因而开发人员可远程调试操作系统分区和裸机应用程序。

其中有一点很重要，随着复杂应用程序（大多源于FOSS社区）中软件数量的增加，以及该软件现带有其操作系统假设，将这种软件移植到统一操作系统中（如SMP　OS或运行于所有核的传统微内核（microkernel））已经越来越不常见了；因为具有硬件支持的hypervisor可使此类软件在其本机环境中运行，同时确保实时性以及所需的安全性。该架构可在同一硬件平台上轻松运行具有多核心价值的软件，更具架构和操作灵活性；此外，通过增加对稳定软件分区的重复使用，该架构还可利用以前一次性工程费用的沉没成本。因而，笔者认为这种架构更加省时高效，更具成本效益，有益于系统架构师和嵌入式系统公司。

基于以上几点，为混合关键型系统选择Type 1 hypervisor时，要考虑以下属性：

1. 硬实时客户虚拟机支持（即本身须为硬实时）。
2. Hypervisor安全认证标准的可用性。
3. 裸机应用程序或客户操作系统中驱动程序支持的设备型号。
4. 访客和CPU集群间资源共享。
5. 强分区，最大程度减少硬件干扰通道。
6. 可扩展至更大数量的内核（即使用无锁技术实现hypervisor，避免使用N级算法（order N algorithms））。
7. 直接中断，中断可绕过hypervisor直接进入虚拟机以达到裸机中断处理性能。
8. CPU SKU和处理器体系结构的独立性（可移植性）
9. 支持UP和SMP访客操作系统。
10. 支持部分核调度（在一核上运行多个虚拟机）。
11. 支持虚拟机抢占（如RTOS抢占通用操作系统，如同一内核上的Linux）
12. 支持未修改的客户OS，如安卓、 Linux，Windows 等。
13. 支持单个设备访问（如每个虚拟机都将PCI设备映射至虚拟机）。
14. 通过hypervisor调试功能实现丰富的开发和工具体验。