大部分嵌入式项目还需要实时操作系统吗?这个问题问得好，因为现代高性能处理器和 Linux、Windows 和其他通用操作系统 (GPOS) 的实时补丁的可用性都在飞速发展。嵌入式设备的本质道出了答案。在许多情况下，制造设备都需要几千甚至几百万个部件。哪怕设备硬件的单位成本节省 1 美元，都会为制造商带来不小的财富。换言之，设备无法承受数千兆赫兹级处理器的成本(更不用说热耗散了)。例如，在汽车远程信息处理技术市场，常见的 32 位处理器以约 600 Mhz 的速度运行——远远慢于台式机和服务器的常用处理器。在这种运行环境中，实时操作系统能得到低端硬件超快、可预测的响应，因此具有显著的节约成本的优势。

除节约成本之外，实时操作系统提供的服务还能使许多计算问题迎刃而解，特别是当多种运行争夺系统资源时。例如，试想一个用户期望(或需要)立即响应输入的系统。利用实时操作系统，软件开发人员能确保用户启动的操作会先于其他系统活动执行，除非必须先执行更重要的任务(如帮助保护用户安全的运行)。

再试想一个必须满足服务质量 (QoS) 要求的系统，例如一台可播放现场视频的设备。如果设备依靠软件播放其内容，它可能会以用户无法接受的速率出现失帧现象——从用户的角度看，该设备不可靠。但使用实时操作系统的话，软件开发人员就能精确控制软件进程的执行顺序，确保以适当和一致的媒体速率播放。

****实时操作系统并不“公平”****

对“硬”实时的需求(以及对实现该功能的实时操作系统的需求)仍然是嵌入式产品业的普遍要求。问题是，实时操作系统具备哪些通用操作系统所不具备的功能呢?适用于一些通用操作系统的实时扩展组件有多大用处呢?它们能提供和实时操作系统一样的性能吗?

让我们先从任务调度开始。在通用操作系统中，调度程序通常使用一种“公平”策略，将线程和进程分配到 CPU 中。这种策略可确保台式机和服务器的应用程序所需的较高的总吞吐量，但无法保证优先级高、时间要求严格的线程先于优先级低的线程执行。

例如，通用操作系统可能会降低分配给优先级高的线程的优先级，或按照有利于系统内其他线程的公平原则，以动态方式调整优先级。因此，优先级高的线程就可能被优先级低的线程抢占。此外，大多数通用操作系统都具有无限期的分配潜伏期：系统内的线程越多，通用操作系统调度线程执行所需的时间就越久。其中任何一种因素都能导致优先级高的线程错过最后期限，即使在速度很快的 CPU 上。

另一方面，在实时操作系统中，线程会按其优先级的顺序执行。如果优先级高的线程准备运行，它能在很短且有限的时间间隔内，从正在执行的优先级低的线程那里接管 CPU。此外，优先级高的线程还能不间断地运行直到完成任务为止——当然，除非它被优先级更高的线程抢占。这种众所周知的基于优先级的抢占式调度，可确保优先级高的线程始终如一地满足最后期限的要求，即使在其他线程争夺 CPU 时间时。

****抢占式内核****

大多数通用操作系统的操作系统内核都不是抢占式的。因此，优先级高的用户线程无法抢占内核调用，相反，它必须等待整个调用全部结束——即使是系统内优先级低的进程进行调用。此外，当驱动程序或其他系统服务(通常在内核调用中运行)以客户端线程的名义执行时，操作系统通常会丢失所有优先级信息。这种系统行为会导致无法预料的延迟，而且会妨碍关键运行按时完成。

另一方面，在实时操作系统中，内核运行是可抢占的。虽然仍有一些时间窗无法抢占，但在设计精密的实时操作系统中，这些间隔非常短暂，通常大约仅几百纳秒。另外，实时操作系统会针对抢占推迟和中断禁止的时限设置上限;这能保证软件开发人员确定情况最糟的延迟期。

为实现这一目标，实时操作系统内核必须尽可能简单、精致。实现这种简单性的最佳途径是设计一种只包含短执行路径服务的内核。通过排除内核中任务集中的运行(如进程加载)并将其分配到外部进程或线程，实时操作系统的设计人员就能保证通过内核的最长的非抢占代码路径有上限。

在一些通用操作系统，内核增加了某种程度的可抢占性。但无法抢占的时间间隔仍然比常见实时操作系统的长得多;这种抢占间隔的长度取决于通用操作系统内核中包含的最长的关键模块部分(如网络)。另外，抢占式通用操作系统内核不能解决可能的无限期延迟情形，例如因为客户端调用驱动程序或其他系统服务时丢失优先级信息。

****避免优先级反转的机制****

即使在实时操作系统中，优先级低的线程也能在无意中阻止优先级高的线程访问 CPU——这种情况被称为优先级反转。当出现无限期的优先级反转时，可能会错过关键的最后期限，进而导致系统运行异常和全面故障的结果。遗憾的是，在系统设计过程中人们往往会忽视优先级反转。有很多优先级反转的实例，包括 1997 年 7 月火星探路者项目遭受困扰的实例。

一般来说，当优先级不同的两个任务共享资源，而优先级高的任务无法从优先级低的任务那里获得资源时，就会出现优先级反转。为防止这种状况超过有限的时间间隔，实时操作系统可提供一种通用操作系统不具备的选择机制，包括优先级继承和优先级封顶模拟。我们不能单纯地评价两种机制的优劣，所以我们着重介绍优先级继承的实例。

首先，我们必须考虑任务同步如何能造成阻塞，而阻塞反过来又如何导致优先级反转。我们假设有任务 1 和任务 2 两个任务正在运行，其中任务 1 具有较高的优先级。如果任务 1 准备执行，但必须等待任务 2 完成运行，就出现阻塞的状况。同步化也会导致这种阻塞;例如，任务 1 和任务 2 共享由锁或信号量控制的资源，任务 1 等待任务 2 对资源进行解锁。或者，当任务 1 请求目前正由任务 2 使用的服务时，也会出现阻塞状况。

****Michael Barr.“优先级反转简介”****

阻塞允许任务 2 运行，直到任务 1 等待的条件出现为止(例如，任务 2 对两个任务共享的资源解锁)。此时，任务 1 可以执行。任务 1 须等待的总时间会随最少时间、平均时间和最多时间变化。这种间隔就是阻塞因数。如果任务 1 必须满足一定的时间限制，该因数就不能随任何参数变化，如线程数或系统内的输入。换句话说，必须限制阻塞因数。

现在，我们引入第三个任务(任务 3)——其优先级比任务 2 的高但比任务 1 的低(参见图 1)。当任务 2 正在运行时，任务 3 准备运行，它会抢占任务 2，而任务 2 在任务 3 被阻塞或完成前都无法运行。当然，这样会增加任务 1 的阻塞因数;也就是说，它会进一步延迟任务 1 的运行。抢占导致的总延迟就是优先级反转。

实际上，可以有多个任务以这种方式抢占任务 2，从而导致连续阻塞的结果。在这种情况下，任务 2 可能被无限期地抢占，产生无限期的优先级反转，导致任务 1 无法满足其最后期限。

这时优先级继承就会发挥作用。如果我们回到上述假设中，在同步期内使任务 2 以任务 1 的优先级运行，那么任务 3 就无法抢占任务 2，这样就能避免优先级反转的产生(参见图 2)。

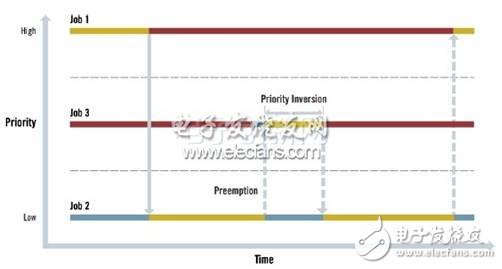
[](http://www.elecfans.com/uploads/allimg/171201/0110022461_0.jpg)

图 1——当任务 3 抢占任务 2 时，任务 1 等待任务 2 完成运行。这进一步推迟了任务 1 的运行。 4

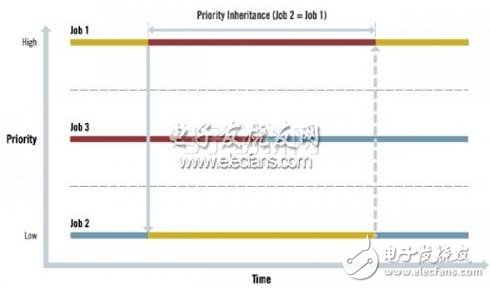
[](http://www.elecfans.com/uploads/allimg/171201/01095961T_0.jpg)

图 2——任务 2 继承了任务 1 的优先级，因而阻止了任务 3 抢占任务 2。任务 3 不再推迟任务 1 的运行。

****提供有保证的 CPU 可用性的分区调度****

保证资源的可用性对许多系统而言都至关重要。如果某个关键子系统(如 CPU 周期)丢失，用户就无法获取该子系统提供的服务。例如，在拒绝服务 (DoS) 攻击中，恶意用户会利用需要优先级高的进程处理的请求攻击系统。该进程会使 CPU 过载并导致其他进程的 CPU 周期匮乏，从而使用户无法使用系统。

安全漏洞并非是导致进程匮乏的唯一原因。在许多情况下，增加系统的软件功能都会导致系统“濒临危险边缘”，导致现有应用程序的 CPU 时间匮乏。及时运行的应用程序或服务不再按预期或要求的那样迅速响应。从历史角度，解决这一问题的唯一途径是更新硬件或对软件进行重新编码(或重新设计)——这两种方法都无法让人满意。

为解决这些问题，系统设计人员需要一种可通过硬件或软件执行 CPU 预算的分区计划，以阻止进程或线程独占其他进程或线程所需的 CPU 周期。实时操作系统正是执行 CPU 分区预算的最佳选择，因为实时操作系统已经提供了对 CPU、内存和其他计算资源的集中访问。

某些实时操作系统提供了固定分区调度算法。系统设计人员能利用这种调度算法对任务进行分组或分区，然后为每个分区分配一定比例的 CPU 时间。利用这种方法，任何既定分区内的任务消耗的 CPU 时间都不会超过该分区静态确定的比例。例如，我们假设为分区分配了 30% 的 CPU。如果该分区内的进程随后成为拒绝服务攻击的目标，它会消耗不超过 30% 的 CPU 时间。这种分配限制确保了其他进程保持各自的可用性;例如，它能保证可访问的用户界面(如远程终端)。因此，操作人员能访问系统并解决问题——无需按动复位开关。 5

但是，这种方法也存在问题。由于调度算法是固定的，因此一个分区无法使用分配到另一个分区的 CPU 周期，即使这些分区未使用其分配的周期。这种方法会浪费 CPU 周期并阻止系统处理高峰需求。因此，系统设计人员必须使用更昂贵的处理器应对运行缓慢的系统，或限制系统能支持的功能数量。

****自适应分区****

另一种被称作自适应分区的方法提供了更灵活的动态调度算法，从而有效克服了这些缺陷。与静态分区一样，自适应分区允许系统设计人员为单独进程或一组进程预留 CPU 周期。因此，系统设计人员能保证一个子系统或分区上的负载不会影响其他子系统的可用性。而且，与静态分区不同的是，自适应分区还能将空闲分区中的 CPU 周期以动态方式重新分配到可受益于额外处理时间的分区——只有当 CPU 满载时，分区预算才会执行。因此，系统就能处理高峰需求并使利用率达到 100%，同时又能继续享受资源保证带来的收益。

同样重要的是，自适应分区还能直接应用于先有系统，而无需重新设计代码或修改代码。例如，在 QNX Neutrino 中，系统设计人员只需启动分区内已有的基于 POSIX 应用程序，实时操作系统的调度程序就会确保每个分区都会收到分配的预算。在每个分区内，会继续根据基于优先级的抢占式调度算法对每项任务进行调度——应用程序不必更改其调度运行。此外，系统设计人员还能以动态方式重新配置分区，以精确调节系统实现最佳性能。

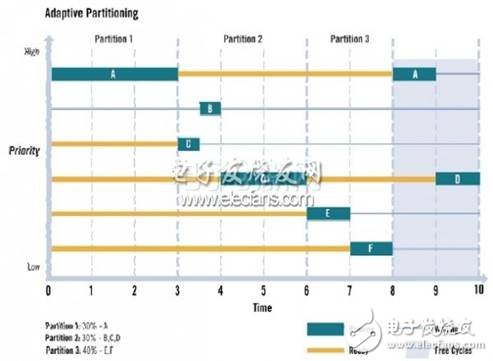
[](http://www.elecfans.com/uploads/allimg/171201/01092CO5_0.jpg)

图 3——自适应分区可阻止优先级高的任务消耗超过其分配的规定比例的 CPU，除非系统包含未使用的 CPU 周期。例如，任务 A 和任务 D 可在分配至分区 3 的时间内运行，因为任务 E 和任务 F 不需要剩余的 CPU 周期预算。 6

****“组合”内核****

通用操作系统(包括 Linux、Windows 和各种类型的 Unix 系统)通常都缺少我们目前介绍的实时机制。但供应商已开发出许多实时扩展组件和补丁，试图填补这一空白。例如，有一种双内核方法，其中的通用操作系统能以任务的形式在专用实时内核上运行(参见图 4)。任何需要确定性调度的任务都会在该内核中运行，但其优先级要高于通用操作系统的优先级。因此，只要这些任务需要运行，它们就能抢占 Linux，而且只有在其工作完成后，才会向 Linux 释放 CPU。

遗憾的是，实时内核中运行的任务只能有限地利用通用操作系统中现有的系统服务，如文件系统、网络服务等。

实际上，如果实时任务向通用操作系统请求任何服务，它都会面临同样的抢占问题，这会阻止通用操作系统以确定方式运行。因此，必须针对实时内核创建新的驱动程序和系统服务，即使通用操作系统中已存在相同的服务。但是，多数通用操作系统为正常的非实时进程提供的受内存管理单元 (MMU) 保护的可靠环境却无法惠及在实时内核中运行的任务。相反，它们会在无保护的内核空间运行。因此，任何含有公共编码错误(如损坏的 C 指针)的实时任务都会轻易导致内核出现严重故障。这的确是一个问题，因为需要实时功能的大部分系统同样需要极高的可靠性。

双内核方法的不同实现使用不同的应用程序接口 (API) 使问题进一步复杂。在多数情况下，为通用操作系统编写的服务无法轻易移植到实时内核中，而且针对一个供应商的实时扩展组件编写的任务未必能在另一个供应商的实时扩展组件上运行。

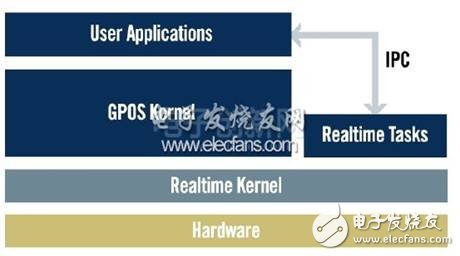
[](http://www.elecfans.com/uploads/allimg/171201/010UKD5_0.jpg)

图 4——在常见的双内核实现中，通用操作系统会作为优先级最低的任务在单独的实时内核中运行。

这种解决方案指出了使通用操作系统支持实时运行的实际困难和复杂性。但这并非是关于实时操作系统和通用操作系统孰优孰劣的问题。通用操作系统(如 Linux、Windows 和各种 Unix 系统)都能以台式机或服务器操作系统的形式正常运行。但是，当进入不符合其设计用途的确定运行环境中(如车载远程信息处理装置、医疗器械、实时控制系统和连续媒体应用)时，它们就会凸显不足。 7

****令调试和定制更轻松的源代码****

使用通用操作系统仍有一些好处，例如在 Linux 的开源模式下支持广泛使用的应用程序接口 (API)。利用开源模式，软件开发人员能针对应用程序的具体需求定制操作系统组件，从而能节约大量用于故障排解的时间。为保持这些优势，实时操作系统的供应商应根据友好的商业授权许可条款，确保其源代码易于访问。

实时操作系统的架构也发挥了重要作用。例如，基于微内核设计的实时操作系统能从根本上简化操作系统的定制过程。在微内核实时操作系统中，只有少数核心基本对象(如信号、计时器、调度程序)才会存在于内核中。所有其他组件(驱动程序、文件系统、协议栈、应用程序)都会以单独的受内存保护的进程在内核外部运行;参见图 5。因此，开发定制的驱动程序和其他与应用程序有关的操作系统扩展组件无需专门的内核调试程序或内核专家。实际上，作为用户空间程序，开发这种操作系统扩展组件和开发标准应用程序一样容易，因为开发人员都能使用标准、源代码级工具对它们进行调试。

图 5——在微内核实时操作系统中，系统服务作为标准的用户空间进程运行，因而简化了操作系统的定制任务。

例如，如果设备驱动程序试图访问其进程空间之外的内存，操作系统就能识别相关进程，指出故障位置并创建可使用源代码级调试工具查看的进程转储文件。转储文件可包括调试程序确定导致故障的源代码行所需的所有信息以及诊断信息(如数据项的内容和函数调用的历史)。 8

这种架构还提供了绝佳的故障隔离和恢复功能：如果驱动程序、协议栈或其他系统服务出现故障，它可在不影响其他服务或损坏操作系统内核的情况下完成这种操作。实际上，“软件监视程序”会持续监测此类事件，并能以动态方式重启出现故障的服务，无需重启整个系统或以任何方式干扰用户。同样，驱动程序和其他服务都能以动态方式停止、启动或升级，无需关闭系统。

绝不可小看这些优势——因为破坏实时性能的最大威胁就是不定期的系统重启!甚至软件更新中包含的以可控制的方式进行的预定重启都会干扰系统运行。为保证一直满足最后期限的要求，软件开发人员必须使用即使在出现软件故障或进行服务升级时，也能保持连续可用的操作系统。

战略决策

实时操作系统能使复杂的应用程序变得可预测而且可靠;实际上，实时操作系统实现的精确时间控制，就是通用操作系统无法提供的可靠功能。(如果基于通用操作系统的系统由于计时错误而无法正常运行，我们可以合理认为该系统不可靠。)选择正确的实时操作系统本身仍然是一项复杂的任务。实时操作系统的底层架构是重要标准，而其他因素也不例外。具体包括：

• 调度算法的灵活选择——实时操作系统支持调度算法选择(先入先出 [FIFO] 调度、循环调度、偶发调度等)吗?您能根据每个线程分配这些算法吗?或者实时操作系统要求您针对系统中的所有线程分配一种算法吗?

• 时间分区——实时操作系统支持能为进程提供有保证的 CPU 周期预算的时间分区吗?这种保证简化了对多个开发团队或供应商的子系统进行集成的工作。它们还能确保关键任务在系统遭到拒绝服务 (DoS) 攻击和其他恶意攻击时，保持可用并满足其最后期限的要求。

• 支持多核处理器——向多核处理器迁移的功能已成为各种高性能设计的基本要求。实时操作系统是否支持多重处理模式(对称多重处理、不对称多重处理、混合多重处理)的选择，并帮助您最大限度地利用多核硬件?允许您诊断和优化多核系统性能的系统跟踪工具支持实时操作系统吗?如果没有能显示资源竞争、线程的过度迁移和其他多核设计常见问题的工具，那么对多核系统进行优化就会变得异常麻烦和费时。

• 远程诊断工具——由于许多嵌入式系统都不允许有故障时间，因此实时操作系统供应商应提供能在不中断系统服务的情况下分析系统运行的诊断工具。寻找能提供运行分析工具(用于系统剖析、应用程序剖析和内存分析的)的供应商。

• 开放的开发平台——实时操作系统供应商能否提供基于开放平台(如 Eclipse)的开发环境，允许您“插入”您最喜欢的第三方工具以便进行建模、版本控制等工作?或者，开发环境是基于专有技术的吗?

• 图形用户界面——实时操作系统使用基元图形库吗?或者它提供了高级图形处理功能(如多层界面、基于 Flash 的人机界面、多目标显示、3D 加速渲染和真正的窗口系统)吗?您能轻松地自定义图形用户界面的视觉效果吗?图形用户界面能同时显示和输入多种语言(中文、朝鲜语、日语、英语、俄语等)吗?2D(如 Flash)和 3D(如 OpenGL ES)应用程序能轻松共享同一屏幕吗? 9

• 标准应用程序接口——实时操作系统限制您使用专有应用程序接口吗?或者它提供了对标准应用程序接口(如 POSIX 和 OpenGL ES)的认证支持，使各种运行环境之间的代码移植变得更容易?实时操作系统还提供对应用程序接口的综合支持吗?还是只支持一小部分已定义接口的子集?

• 源代码——实时操作系统供应商允许您轻松访问源代码以简化调试过程，并帮助您根据具体需求定制实时操作系统服务吗?

• 面向数字媒体的中间件——对数字媒体的灵活支持已成为各种嵌入式系统(如车载收音机、医疗设备、工业控制系统、媒体服务器以及消费电子产品)的设计要求。系统可能需要处理多种媒体资源(如 USB 闪存、MP3 播放器、网络流媒体、蓝牙电话等)，分辨多种数据格式(如 MP3、WMA、AAC、MPEG-2、MPEG-4 等)并支持各种数字版权管理 (DRM) 方案。通过为数字媒体提供设计精密的中间件，实时操作系统供应商能避免连接多媒体资源、组织数据以及初始化适当的数据处理路径所需的大量软件工作。另外，设计完善的中间件解决方案还能灵活支持新的数据源(如下一代 iPod)，无需修改用户界面或其他软件组件。

选择实时操作系统是项目团队的战略性决策。如果实时系统供应商针对以上问题给出了清楚的答案，那么您离现在和将来选择到完全适用的实时操作系统又近了一步。