StarFive 赛昉科技

在昉·星光 2上运行RT-Linux的分析

版本：1.12

日期：2024/03/27

法律声明

阅读本文件前的重要法律告知。

版权注释

本文档中的说明均基于“视为正确”提供，可能包含部分错误。内容可能因产品开发而定期更新或修订。上海赛昉科技有限公司 （以下简称“赛昉科技”）保留对本协议中的任何内容进行更改的权利，恕不另行通知。

赛昉科技明确否认任何形式的担保、解释和条件，无论是明示的还是默示的，包括但不限于适销性、特定用途适用性和非侵权的担保或条件。

赛昉科技无需承担因应用或使用任何产品或电路而产生的任何责任，并明确表示无需承担任何及所有连带责任，包括但不限于间接、偶然、特殊、惩戒性或由此造成的损害。

联系我们：

地址： 浦东新区盛夏路61弄张润大厦2号楼502，上海市，201203，中国

网站： http://www.starfivetech.com

邮箱：

- sales@starfivetech.com（销售）
- support@starfivetech.com（支持）

目录

表格清单

插图清单

前言

关于本指南和技术支持信息

关于本手册

本文为赛昉科技官方文档，作为昉·惊鸿-7110项目的一部分。本文介绍了在昉·星光 2上运行RT-Linux，展示了RT-Linux在昉·星光 2上的性能。同时，它还介绍了RISC-V RT-Linux的发展。更重要的是，从kernel v6.6（LTS）版本开始，RT-Linux正式支持了RISC-V架构。一到两年后，我们将会看到RISC-V SoC在工业项目上运行。

修订历史

表 0-1 修订历史

版本	发布说明	修订
1.12	2024/03/27	更新了测量延迟 (第 19页)中的数据。
1.11	2024/01/16	更新了简介 (第 7页)中的描述。
1.1	2023/10/31	更新了附录A：实时Linux补丁 (第 25页)。
1.0	2023/06/13	首次发布。

注释和注意事项

本指南中可能会出现以下注释和注意事项：

- **提示：**
  建议如何在某个主题或步骤中应用信息。
- **注：**
  解释某个特例或阐释一个重要的点。
- **重要：**
  指出与某个主题或步骤有关的重要信息。
- **警告：**
  表明某个操作或步骤可能会导致数据丢失、安全问题或性能问题。
- **警告：**
  表明某个操作或步骤可能导致物理伤害或硬件损坏。

1. 简介

本文为赛昉科技官方文档，作为昉·惊鸿-7110项目的一部分。本文介绍了在昉·星光 2上运行RT-Linux，展示了RT-Linux在昉·星光 2上的性能。同时，它还介绍了RISC-V RT-Linux的发展。更重要的是，从kernel v6.6（LTS）版本开始，RT-Linux正式支持了RISC-V架构。一到两年后，我们将会看到RISC-V SoC在工业项目上运行。

2. Linux实时系统

当一个给定的任务必须在一定的时间限制内完成时，通常需要使用到实时系统。这样的系统必须非常准确，具有可预测性和确定性，所以这些时序要求在开发阶段就需要得到保证。因此，需要为此专门制作具有实时功能的操作系统。这与通用操作系统有着不同的设计决策和内部实现。

本章描述了实时Linux系统的不同方面，特别关注在PREEMPT_RT补丁上。首先，在实时系统 (第 8页)中，解释了实时系统的一般概念。随后，在实时扩展 (第 9页)中，提出了针对Linux内核的多个实时扩展。然后，在任务调度 (第 10页)中，介绍了的任务调度的基本原理。接下来，在内核抢占 (第 11页)中，描述了Linux中可用的不同抢占设置。最后，在调度延迟 (第 12页)中介绍了调度延迟的各种方面。

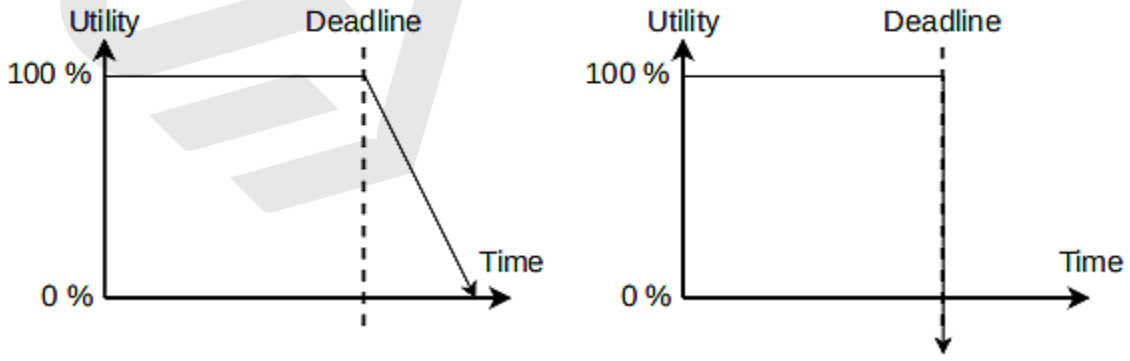
2.1. 实时系统

当给定的应用程序需要满足某些特定的定时要求时，将使用实时系统。通常这些实时要求的严格程度有着很大差异，因此我们将这些具有不同需求的实时系统划分为两个不同的类别。

- **软实时系统**：如果时限达不到要求会有一定负面影响
- **硬实时系统**：如果时限达不到要求会产生灾难性影响

这两个系统之间的效用差异如图 2-1 : 实时系统服务实用程序 (第 8页)所示。在一个软实时系统中，结果的效用是100%的，直到达到截止时间。之后，结果开始根据一些应用程序特定曲线逐渐下降，最终在某个点达到零。这种水平的实时性通常相对容易实现，因为偶尔错过最后期限可以被接受。硬实时系统的行为与软实时系统一样，直到达到最后期限，但此时，结果的效用会立即变为负值，这表示系统在当时将经历的灾难性故障的影响。设计硬实时系统是非常具有挑战性的，因为它必须保证系统始终满足定时要求。

图 2-1 实时系统服务实用程序



*(A)*软实时系统 *(B)*硬实时系统

软实时系统可以用于许多典型的非关键嵌入式应用程序，这些应用程序没有太严格的实时要求。在软实时系统中，如果错过最后时限，最坏的结果只会降低用户体验或产生不准确的结果。例如，在任何情况下，时间要求都不应是影响产品安全的关键。另一方面，硬实时系统在更关键的应用程序中都是必需的，因为它们在任何操作条件下都不能错过最后时限。如果错过这些系统的最后时限，可能会对财产造成重大损害或严重伤害。

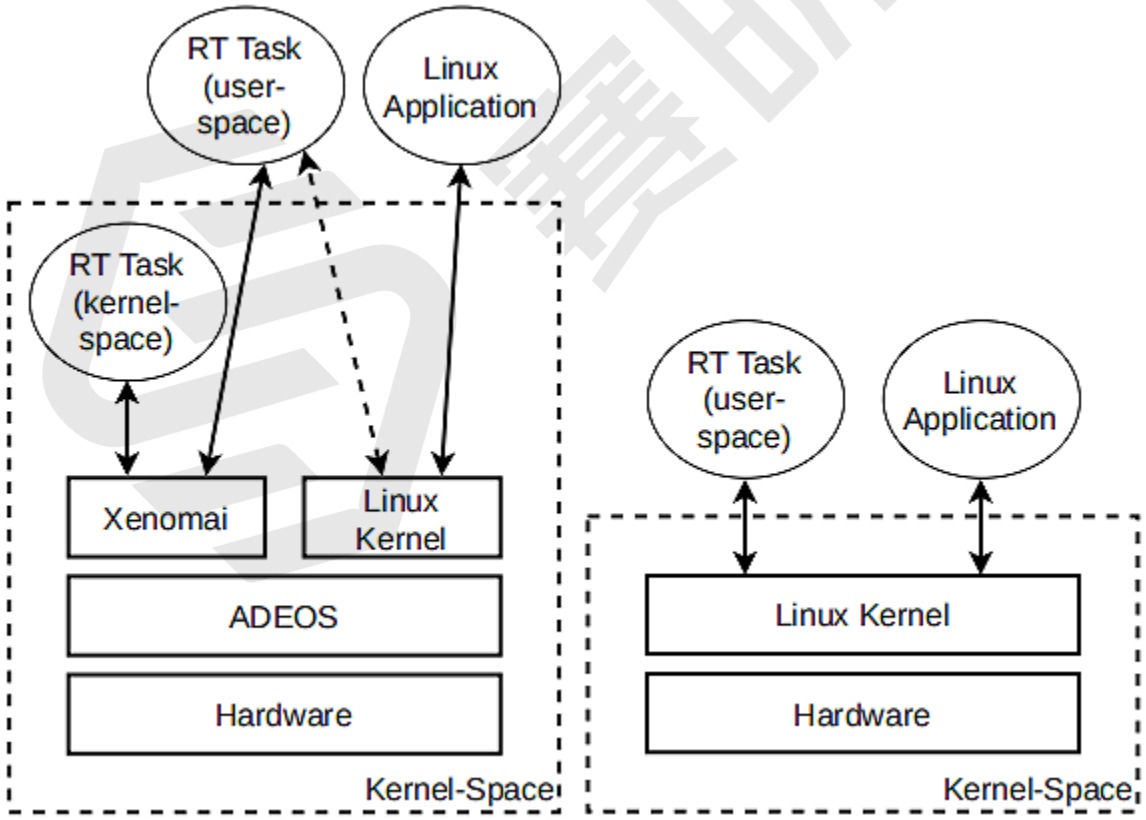
2.2. 实时扩展

与任何通用操作系统一样，linux内核主要是为吞吐量进行优化的，因为这是此类系统最重要的设计目标。这意味着在默认情况下，未修改的Linux系统将经常经历在实时系统中无法接受的显著的无限延迟。但是，可以将缺失的实时功能添加到Linux内核中去。这样，操作系统可以在提供广泛的系统服务和可接受的实时性能之间达到最佳平衡。

通常，有两种不同的方法来实现实时功能，要么通过实现基于协同内核的系统，要么通过在主线内核中打补丁，如图 2-2 : 实时Linux系统的不同体系结构 (第 9页)所示。在第一个选项中，Linux内核在一个单独的实时调度器下作为一个正常进程运行。当这个调度程序还处理实时任务和系统事件时，实时部分可以很容易地与系统的其他部分分离出来。第二种选择是不影响底层的内核体系结构，而是对内核本身引入一组更改。在此配置中，Linux内核负责同时调度在系统内执行的正常任务和实时任务。

在协核系统中，Linux内核在某种程度上与实时任务隔离。

图 2-2 实时Linux系统的不同体系结构



*(A)*协同内核 *(Xenomai) (B)*补丁*(PREEMPT_RT)*

这种系统有一个非常典型的例子：xenomai项目，它利用了硬件和Linux内核之间的中间层，有效地充当了中断调度程序和调度器。它基本上处理的是系统事件，以便处理总是为最高优先级的实时任务的优先级，以保证指定的时间要求。另一个可行的替代方案是打了PREEMPT_RT补丁的Linux，即实时Linux内核。在这种方法中，底层架构与以前保持完全相同。以前，PREEMPT_RT对内核引入了大量的更改，但现在，这些特性许多已经合并到主线内核中。这项工作仍在进行中，但最终整个PREEMPT_RT补丁会合并到主线内核。

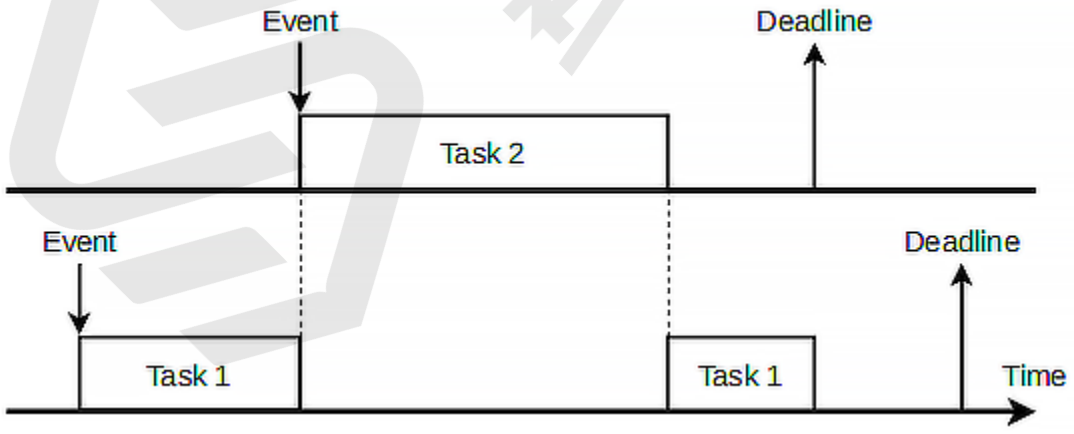
在这两种体系结构之间进行选择是取决于应用程序的。通常，利用协内核的方法可以实现略低的延迟。但另一方面，系统的复杂性增加了，需要专门为所使用的实时内核精心制作实时任务。这与PREEMPT_RT补丁的Linux内核相反，因为体系结构更简单，实时任务几乎可以像任何其他常规应用程序一样编写。此外，打补丁的Linux的整体性能平均比协同内核所能实现的更好。

2.3. 任务调度

在操作系统环境中，调度器只是决定在给定的时间点执行什么任务。通常，它是所有操作系统中最关键的组件之一。通常，通用操作系统和实时操作系统对调度决策有不同的偏好，因此针对这两种情况开发了多个调度器实现。在类别之间有这种区别是很重要的，因为设计目标完全不同。

在设计实时系统时，最合适的调度算法总是取决于具体情况。或许最简单但却有用的实时调度算法可以被认为是固定优先级的优先级调度算法，如图 2-3 : 实时调度原理 [6] (第10页)所示。在这个调度方案中，每个任务在设计阶段都获得一个分配给它们的静态优先级。在系统操作期间，调度器只是保证当前任务始终具有最高的优先级。由于调度是抢占式的，这意味着即使正在执行某些任务的任务也可以被暂时中止，让另一个更高优先级的任务执行。

图 2-3 实时调度原理 [6]



与前面提到的固定优先级的优先级调度算法相比，Linux调度器更复杂一些，但从根本上讲，它们处理实时任务的方式十分相似。Linux调度器实现了几种调度策略，可分为非实时调度组和实时调度组。普通应用程序通常使用其中一个非实时策略执行，并且它们的优先级始终低于使用任何实时策略运行的任务。Linux调度器支持三种不同的实时策略，它们可以分配给任何需要实时优先级的任务。

- **SCHED_FIFO**：将执行使用此策略运行的任务，直到它完成并自动抢占，或者被一个优先级更高的任务抢占。
- **SCHED_RR**：如果没有被高优先级任务抢占，任务在被抢占前只能在指定的时间片内以最高优先级运行。
- **SCHED_DEADLINE**：是一种实现基于任务执行期限的调度算法的策略。

使用任何实时策略执行的所有任务都有一个静态实时优先级。对于**SCHED_FIFO**或**SCHED_RR**策略，此优先级可以在1（低）和99（高）之间。具有**SCHED_DEADLINE**任务的有效优先级始终为100，具有最高的优先级。任何以较高优先级执行的任务都将总是优先于另一个具有较低优先级的任务。有了这些策略，就可以仔细设计一个实时系统的执行，以满足苛刻的定时要求。

2.4. 内核抢占

主线Linux内核目前有三个可用的抢占设置：服务器、桌面和低延迟桌面。随着PREEMPT_RT补丁的引入，第四个选项（实时）开始可用。通过使用这些选项， 就可以将吞吐率优 先替换为延时优先 。以下根据实时性能从低到最佳排序，描述了这些选项。

- **服务器**：传统的抢占模型，通过此选项，将执行内核代码，并禁用抢占以实现最大吞吐量。
- **桌面**：抢占模型向内核代码中添加了显式的抢占点。此选项以较低的吞吐量，提供了更好的响应性。
- **低延迟桌面**：通过使所有普通内核代码可抢占来减少延迟。此设置允许更快的响应交互式事件。
- **实时**：选项实际上使整个内核可抢占，包括最关键的部分。此选项仅在应用**PREEMPT_RT**补丁时可用。

正如抢占选项名称所建议的那样，这些设置每个都有一个适当的用例。服务器抢占可用于吞吐量是唯一最重要指标的服务器安装中。另一方面，实时抢占应该用于嵌入式系统，其中的绝对吞吐量不是最关键的，关键的是最大经验延迟。因此，Linux中不同的抢占级别允许在不同的环境中灵活使用，也就是说，相同的操作系统可以在服务器和嵌入式系统中使用。总的来说，这种协同作用对整个生态系统都非常有益，因为为服务器系统实现的改进或修复也可以自动应用于小型嵌入式设备。

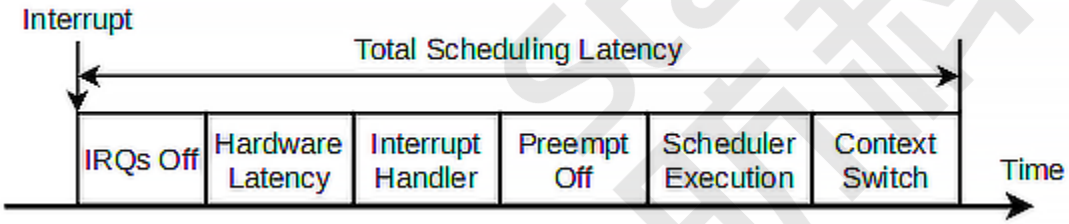
即使一些抢占设置声称减少了延迟，但在实践中，实时设置是所有实时系统的唯一可行的选项。在这种配置中，大多数自旋锁被转换为正常的睡眠锁，中断处理程序是线程化的，高分辨率计时器被用于精确计时。此外，还引入了其他一些更无关紧要的变化。有了这些改进，实际上整个内核都是完全可抢占的。只有非常低级的事件处理才能在不可抢占的环境中执行。总之，实时抢占模型显著提高了系统的响应能力，但降低了整体性能，因为每次引入的更改都会导致一些额外的负载。

2.5. 调度延迟

调度延迟衡量了从外部事件或中断到相关任务开始执行所经过的时间。通常，这意味着例如定时器中断和执行某些周期性任务之间的延迟。调度延迟通常是评估系统实时性能时最重要的值。基本上，它将所有的延迟来源组合成一个易于分析、测量和比较的单一变量。

然而，为了进行更详细的分析，通常需要单独研究调度延迟的不同部分。这些单独的元素可见图 2-4 : 调度延迟组成 (第 12页)。延迟的其中一个原因基本发生在硬件层的中断之后。通常，由硬件引起的实际延迟非常小，只有几个时钟周期，但在这个阶段，如果系统中断被完全禁用，可能会经历额外的延迟。这有时会延长更多时间，并且取决于当前正在执行的软件。造成延迟的另一个原因是即时中断处理例程。通常这部分可能包含一些额外的操作系统逻辑，但一般来说，它在实时系统中保持的时间非常短。在处理了中断之后，调度程序可以决定下一个执行的任务。在大多数时候，调度程序本身都非常快，但现在，以前运行的软件可能会导致额外的延迟。如果禁用了抢占，则先前执行的程序此时将恢复并执行，直到再次启用抢占为止。最后，在这些步骤之后，调度器可以通过执行上下文切换来调度与中断对应的任务。

图 2-4 调度延迟组成



对于延迟的这些不同部分，可以看出，除了被禁用的中断和抢占延迟之外，其他部分总是相当恒定和可预测的。因此，这两个延迟的原因是重要的，因为它们依赖于当前执行的程序，这意味着延迟可能会延长很长一段时间。在Linux内核中优化和最小化调度延迟的这两部分是**PREEMPT_RT**补丁最重要的目标。

3. 建立测量机制

在测量和分析实时Linux系统的实时性能的过程中，使用了各种RISC-V开发平台和软件工具。选定的开发平台代表目前可用的典型的RISC-V系统。此外，所使用的测量工具在行业中很常见，在其他类似研究中广泛使用，因此它们提供了容易比较的结果。

3.1. 内核菜单配置

从2017年的4.15版内核开始，RISC-V就已经支持Linux内核架构。 赛昉科技昉·星光 2 BSP现在使用5.15.0版本的内核。因此，我们选择了内核5.15.0来应用RT-Linux补丁。由于RT-Linux在5.15.0版内核没有支持RISC-V架构，为了让PREEMPT_RT在RISC-V上发挥作用，需要增加一些修改。在RISC-V的PREEMPT_RT (第 16页)中详细描述了这些修改。这个定制的内核代码库用于所有度量和延迟分析。

最终的测量结果涉及两种不同配置的内核。内核间的唯一区别是抢占设置，除此之外使用的配置均相同。在实时内核配置中，抢占设置被设置为实时模式。而在主线内核配置中，对应的选项被设置为低延迟桌面。

3.2. 负载生成

当给定的计算机系统处于空闲状态，即什么都不做时，它通常会非常迅速地对外部事件做出反应。当负载加重时，系统将对外部事件做出响应（stress-ng 未被调用）。 但是，在许多系统上，当应用负载时，这种行为可能会带来巨大的变化。因此，在没有任何负载的情况下测量实时平台的延迟结果通常不具有代表性。理想情况下，运行实际的应用程序软件将允许度量实际系统经历的实际延迟。然而，该软件并不总是可用，或者由于某种原因，运行该软件将不实用。此外，如果程序很少执行某些特定的操作，那么捕获最长的延迟可能需要大量的时间。对于这些情况，通常最好在延迟测量期间运行一些人工负载。

现在有许多工具可用于此目的，但本次测试选择了*stress-ng*(版本0.13.0)，因为它是最灵活的实用程序之一，并且已经在行业中广泛使用。它专门设计来用于测试各种操作系统接口和计算机平台的物理子系统。*stress-ng*的最初目的是发现硬件问题，如热过载。目前，该工具可以用于产生多种压力源以实施广泛的压力测试，例如，检测内核错误并执行基准测试。但最重要的是，它还可以用来发现来自实时系统的意外延迟。

总的来说，*stress-ng*实现了超过220种不同的压力源，这使得设计一套全面的测试成为可能。本研究应用了所有类别的压力源以便对系统不同部分施加压力，尽可能的模拟实际使用情况。为了获得关于系统行为的广泛信息，我们总共选择了五种不同的压力类别进行进一步的检查。一旦执行了某个类别，四个进程都会启动，每个核一个进程。这就确保了所有的核都有相同数量的负载。

- **Idle**：系统没有任何后台工作程序执行计算。系统将能够尽快做出响应（未调用*stress-ng*）。
- **CPU**：创建可执行各种浮点算术运算负载工作线程。这将增加CPU和缓存的压力（stress-ng --matrix 4）。
- **OS**：创建能够执行各种*set*()*系统调用的负载。这将增强内核操作和数据结构的压力（stress-ng --set 4）。
- **Memory**：创建连续调用*mmap()*操作并写入已分配的内存的工作线程。这将增加内存处理压力(stress-ng --vm 4)。
- **storage**：创建可重复创建和删除目录的工作线程。这将增强文件系统和I/O压力(stress-ng --dir 4)。

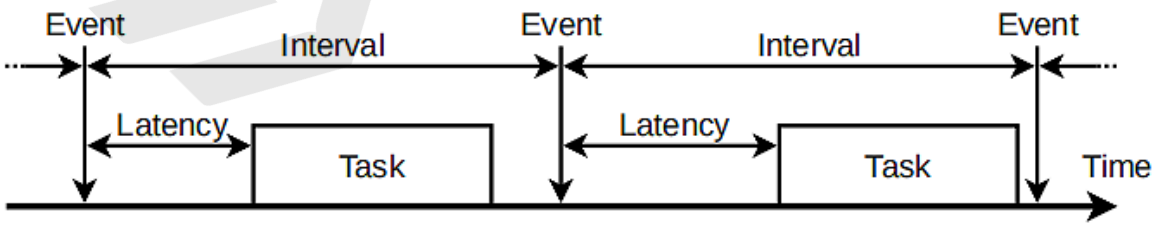
我们遵循了一个简单的原则，为每个类别选择最合适的压力源。首先，对所有可用的压力源进行简要测试，并记录测量的延迟。然后，根据观察到的最大延迟对结果进行排序，从这个列表中，可以选择最合适的压力源。最后的选择是为了让每个类别都代表嵌入式实时系统中的典型工作负载，但其方式可以发现一些有趣的结果。因此，通过一些精心设计的论点，本可以找到产生更长延迟的压力源，但由于它们不能代表实际情况，这些选择就被忽视了。

3.3. 延时测量

有多种方法可以用来定义给定系统的延迟，也有多种方法来测量它。本研究选择了一种基于调度延迟来评估给定系统实时能力的方法，既简单又被广泛使用。这模拟了许多典型控制系统的行为，这些系统必须执行精确定时的周期性任务。此外，当系统对某些外部异步事件作出反应时，也会出现同样的情况。

Cyclictest（2.20版本）被用作延迟测量的主要测量工具。它最初是由PREEMPT_RT的作者构建的，旨在通过精确测量系统延迟来帮助开发工作，如图 3-1 : Cyclictest延时测量 (第14页)所示。循环测试的工作原理是启动一个常规的主线程，它将启动几个实时测量线程。这些实时线程中的每一个都被设置为在一个定义的时间间隔之后定期唤醒。每次它们被唤醒时，预期和实际唤醒时间的时差将被记录并传递给主进程。主进程存储所有这些结果，一旦测试结束，它将输出汇总的结果。

图 3-1 Cyclictest延时测量



对于SMP系统，如昉·星光 2，cyclictest为每个核心分别提供统计数据。然而，拥有这些分开的结果对于总体延迟评估通常并不重要。因此，通过简单地将每个核的结果相加，得到整个系统的单个延迟数，简化了本文中给出的计算和图形。这使得所提供的测量结果更容易被可视化和理解。

应为每个系统和测量场景仔细配置cyclictest。最重要的是，需要根据测量情况来设置优先级和时间间隔。要测量系统延迟，优先级应该设置为高于系统上运行的负载。该间隔的最优值将略大于观察到的最大延迟。总的来说，多个选项被设置为特定的值，以获得最合适的延迟结果。

- **--priority=99**：选择了相等于99的测量线程的实时优先级。
- **--interval=200**：将测量线程的预期唤醒周期设置为200μs。
- **--duration=10m**：测量的总持续时间设置为10分钟。
- **--Histogram=200**：允许输出延迟在200μs以内的统计数据并生成直方图。
- **--smp**：为每个可用的核启动单个固定线程。
- **--mlockall**：内存分配完毕后锁定该内存页， 防止发生swap影响测试。

Cyclictest工具还会自动设置一些值得注意的参数，也可以手动设置，基于这些选择为默认值。需要注意的是，由于指定了直方图选项，测量线程唤醒时间的间隔参数被自动设置为零。此外，调度策略默认设置为**SCHED_FIFO**。总之，这些参数非常典型，非常适合测量实时应用程序将经历的延迟。

4. RISC-V的PREEMPT_RT

本章主要描述在赛昉科技昉·惊鸿-7110 SDK内核（5.15版Linux内核）应用**PREEMPT_RT**的方法（仅限于测试）。如果从工业项目落地来讲，用6.6 LTS内核更合适，因为RT-Linux已在6.6 LTS内核支持RISC-V架构，而且赛昉科技昉·惊鸿-7110的大部分驱动已在6.6内核能找到，只需在6.6版本内核打上RT-Linux补丁，并编译包含昉·星光 2的驱动程序的内核就可以使用和测试**PREEMPT_RT**。

PREEMPT_RT补丁已经支持多种架构，并且大多数重要的实时功能都是在独立于架构的内核代码中实现的。因此，可以放心地假设内核的这些部分可以正确地工作，并且RISC-V操作所需的所有更改只需要在特定于体系结构的代码中完成。在Linux内核源代码树中，这将包括*arch/riscv/*目录下的所有文件，以及*drivers/*目录中的每个RISC-V特定的驱动程序。对Linux内核所实现的更改列在附录A：实时Linux补丁 (第 *25*页)中。

本章讨论了将**PREEMPT_RT**补丁的基本功能移植到RISC-V架构时所需的不同步骤和修改。首先，在应用补丁 (第 *16*页)中，提出了将**PREEMPT_RT**补丁应用于主线内核源代码树的过程。然后，在LAZY_PREEMPT (第 *17*页)中，讨论了RT-Linux中的LAZY_PREEMPT。

4.1. 应用补丁

让**PREEMPT_RT**与RISC-V架构一起工作的第一步是将官方补丁文件应用到主线Linux内核源代码树中。通常，对于正确选择的版本，这一步十分简单，只是使用*patch*命令来应用一个补丁文件。然而，在这种情况下，需要一个非常特定的Linux内核版本来正确地支持赛昉科技昉·星光 2开发包。目前赛昉科技昉·星光 2的Linux版本是5.15.0。

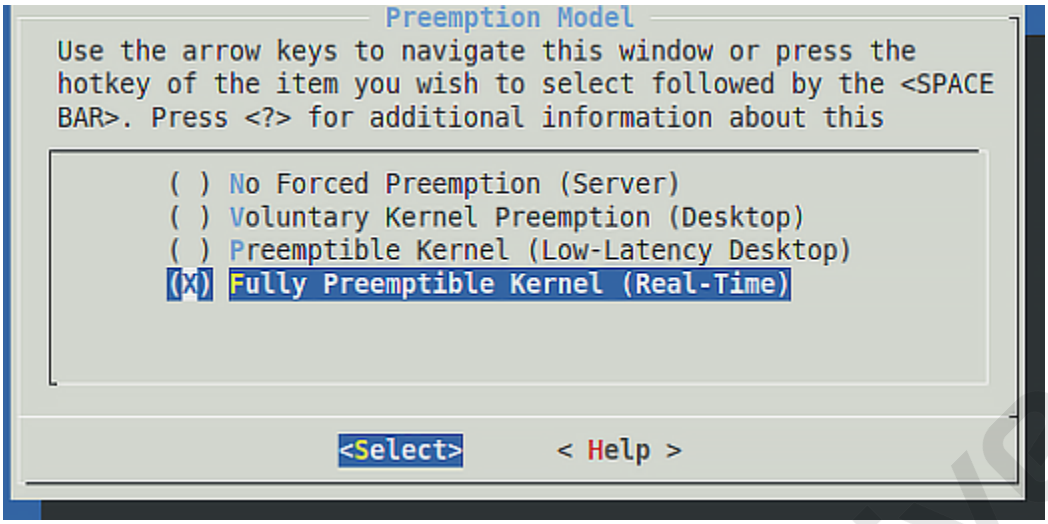
下载5.15.0-rt patch：点击下载。

下载6.6 patch：点击下载。

注：
请执行以下命令应用补丁：

```
patch -p1 < (patch)
```

在应用5.15.0 RT补丁后，应用附录A (第 *25*页)中的补丁。这些补丁包含RISC-V架构的延迟抢占配置，并为昉·星光 2开发包启用**PREEMPT_RT**配置。此外，还对`kernel/sched/cpupri.c`文件进行了关于通过补丁进行任务优先级的修改：`0004-cpupri-a-work-around-for-non-rt-test-panic`。 这个修改解决了5.15.0 RT中的一个问题，即非RT任务会在压力测试中触发“kernel panic”。但是，请注意，这是一个变通办法。



4.2. LAZY_PREEMPT

在5.15的RT补丁集中，不支持在RISC-V平台启用LAZY_PREEMPT配置，但它是RT-Linux中的一个重要特性。此补丁提高了非RT工作负载性能，因此有必要去支持它。尽管RT-Linux不支持RISC-V架构，但是我们可以使用其他的架构补丁（ARM/x86）来实现。

附录A中列出了RISC-V LAZY_PREEPMT的实现方法。该补丁已通过OpenPLC测试验证。 通过测试结果改进了延迟和抖动。

获取LAZY_PREEMPT的更多细节，请查看下面LAZY_PREEMPT补丁的commit信息：

It has become an obsession to mitigate the determinism vs. throughput loss of RT.Looking at the mainline semantics of preemption points gives a hint why RT sucks throughput wise for ordinary SCHED_OTHER tasks.One major issue is the wakeup of tasks which are right away preempting the waking task while the waking task holds a lock on which the woken task will block right after having preempted the wakee.In mainline this is prevented due to the implicit preemption disable of spin/rw_lock held regions.On RT this is not possible due to the fully preemptible nature of sleeping spinlocks.

Though for a SCHED_OTHER task preempting another SCHED_OTHER task this is really not a correctness issue.RT folks are concerned about SCHED_FIFO/RR tasks preemption and not about the purely fairness driven SCHED_OTHER preemption latencies.

So I introduced a lazy preemption mechanism which only applies to SCHED_OTHER tasks preempting another SCHED_OTHER task.Aside of the existing preempt_count each tasks sports now a preempt_lazy_count which is manipulated on lock acquiry and release.This is slightly incorrect as for lazyness reasons I coupled this on migrate_disable/enable so some other mechanisms get the same treatment (e.g. get_cpu_light).

Now on the scheduler side instead of setting NEED_RESCHED this sets NEED_RESCHED_LAZY in case of a SCHED_OTHER/SCHED_OTHER preemption and therefor allows to exit the waking task the lock held region before the woken task

preempts.That also works better for cross CPU wakeups as the other side can stay in the adaptive spinning loop.

For RT class preemption there is no change.This simply sets NEED_RESCHED and forgoes the lazy preemption counter.

5. Cyclist测试结果

Linux内核是一个非常复杂的软件，因此要证明给定系统的确切延迟特性并不容易。通常，延迟也高度依赖于工作负载和选定的内核配置。这些参数和许多其他参数的组合是无穷无尽的，但实际的测量可以很好地了解延迟，这在实际应用中也会遇到。

本章介绍了本文的延迟测量结果。首先，在一般功能 (第 19页)中，讨论了测量系统的一般功能。然后，在测量延迟 (第 19页)中，从两个内核中给出了测量的延迟表和相关的特征数。

5.1. 一般功能

目前没有任何定量的测量来评估系统的一般功能和正确性。然而，正常使用、全面测试和延迟度量已经对整个系统功能提供了很好的预估。一个错误的工作系统会导致明显的问题，如错误的计算结果或内核panic消息。虽然在测试和开发过程中不断观察系统的功能，可证明一般系统工作非常可靠，但并没有执行其他更彻底的实验来评估系统的功能。

在所有的延迟测量中，该系统是完全稳定和可用的，没有任何出现问题的迹象。即使在不同的区域被施加大量负载时，内核也没有显示出任何不稳定性。在所有的测试中，没有观察到系统的错误行为，这是一个很好的迹象，即PREEMPT_RT补丁的Linux在RISC-V架构上没有遇到任何严重的问题。

5.2. 测量延迟

在本地开发机上编译适当的内核镜像、文件系统、引导加载程序和其他必要文件后，开始进行延迟测量。测试使用的两种内核都按照前面在负载生成 (第 13页)中所述进行了配置。一旦所有文件都烧录到SD卡并成功启动系统后，在进行任何测试之前，系统将出现3分钟的空闲。在此期间，内核有时间初始化内部数据结构，如随机数生成器，以完全完成启动过程。这样，任何内部内核初始化都不会影响测量结果。

在这个过程之后，内核已经准备好进行实际的测量了，因为昉·星光 2包含DVFS驱动程序，在测试中可以改变CPU的频率。要获得精度测试结果，请将CPU频率设置为最高值（1.5 GHz）。

1. 设置最高的CPU频率：

   echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

2. 设置不同的负载（CPU、OS、Memory或Storage）：

   cpu: stress-ng --matrix 4

   os:stress-ng --set 4

   memory: stress-ng --vm 4

   storage:stress-ng --dir 4

3. 运行cyclitest：

cyclictest -m -S -p 99 -i 200 -q -D 10m -H 200

（cyclictest参数细节可以看延时测量 (第 14页)一节）

每次测试延迟持续10分钟，产生大约120万个延迟样本。使用主线和PREEMPT_RT补丁内核对每个负载类别重复相同的过程，没有遇到任何问题。全部加起来，一共执行了10种不同的测量组合。

表 5-1 : 测量延时结果 (第 20页)中显示了测试结果数据。使用表格格式在不同类别和内核之间进行比较更容易。最重要的是，该表包含了每个测量的绝对最大观测延迟。其他重要的指标是平均延迟和各自的标准偏差。此外，为了保证完整性，还给出了最小的观察延迟。循环测试工具报告测量到的实际时钟分辨率为1μs，所以提出的所有计算和测量都是四舍五入到最近的微秒。

stdev是延迟的标准偏差。它反映了延迟的抖动。stdev的值越低，实时系统就越好。附录B中列出了计算方法和一个实例。

表 5-1 测量延时结果

压力	实时				主线			
	平均值（μs）	Stdev（μs	最小值（μs）	最大值（μs）	平均值（μs）	Stdev（μs）	最小值（μs）	最大值（μs）
Idle	6	2	5	37	8	4	6	66
CPU	8	4	6	39	10	6	6	88
OS	9	8	6	46	20	16	6	9162
Mem	12	6	6	95	12	14	6	14125
Storage	10	7	7	49	16	14	7	3025

就主线内核而言，idle条件下测得的最大延迟最低，这在意料之中。此外，在这种特殊情况下，平均延迟和标准偏差也最小。当系统引入 CPU 负载时，会导致延迟上升，但最大延迟保持在 100 μs 以下。然而，在其他各种负载下，延迟时间明显延长。

在主线内核中进行的操作系统和内存压力测试似乎比实时内核要差得多。在这种类型的负载下，最大延迟以毫秒计，并且平均延迟与实时系统相比也要高得多，标准偏差较显著。

对于实时内核，在所有不同的类别中，观察到的延迟彼此非常相似。除误差类别外，所有测量的最大延迟都远低于200μs。此外，在整个测试过程中，观察到的标准偏差始终很小，这表明大多数经历的延迟都接近于平均数字。因此，向系统应用任何选定的负载都不会对系统的响应造成太大影响。

6. 分析

与当前的主线版本相比，PREEMPT_RT补丁改进了RISC-V中的最坏情况延迟数。这些数据表明，该系统已经满足了实时操作的最低要求。此外，所研究的RISC-V系统似乎能够实现与其他一些已经得到官方支持的架构非常相似的延迟数据。这一观察结果证实了RISC-V在未来具有很大的潜力，但要使这种设置在生产级使用中可行，仍有大量的工作和验证要做。

本章分析了前面第四章Cyclist测试结果 (第 19页)测试结果中提出的测量结果。首先，在延时分析 (第 21页)中讨论了影响延迟性能的不同方面。然后，在实时应用的适应性 (第 22页)中详细分析了RISC-V实时Linux在实际应用中的适用性。最后，在未来的工作 (第 22页)中讨论与此设置相关的未来工作。

6.1. 延时分析

正如预期的那样，添加PREEMPT_RT补丁后，在所有压力情况下，最坏情况下产生的系统延迟都显著减少了。这在一些要求最苛刻的类别中尤为明显，在这些类别中，实时内核中的延迟接近不施加负载的工况。在这些相同的测试案例中，主线内核的延迟非常大，是不能被实时系统接受的。最值得注意的是，出现内存溢出错误情况下的延迟非常显著，并且在主线内核的毫秒区域内达到了很好的延迟。但实时内核的情况并非如此，其延迟时间仍与其他类别相当。根据这些测量结果，实时内核的可靠性和确定性在调度延迟方面得到了证实。

实时内核性能优越的最大原因是将自旋锁换为了睡眠锁，这也实现了优先级继承功能。因为内核内部的不可抢占区域由于这种变化而显著地减少了。此外，将大多数中断处理程序线程化，从而以较低的优先级进行调度，这可能会稍微解释实时内核为何能够测量到的更好的延迟。在这种特定的内核配置下，这种变化可能与转换自旋锁的关系不大，原因是启用的中断源并不太多。与其他延迟来源相比，那些需要启用的中断似乎都能很快完成。最后，与主线内核的性能相比，PREEMPT_RT的延迟改进可能并没有那么大。这主要是因为以往由PREEMPT_RT引入的其他重大更改已经合并，因此它们也有利于主线内核。

即使实时内核的表现比主线更好，它仍然有一段连续产生的超过150µs的延迟。基于对内核内部内容的跟踪，我们对这些延迟有了一些深入了解。最长的延迟似乎完全是由计时器中断（始终在硬中断上下文中执行）和偶尔需要的原始自旋锁造成的。延迟直方图中的其他显著峰值表明，也有一些额外的原因导致较小但重要的延迟。然而，通过跟踪来可靠地分析这些是困难的，并且如果仍然出现较长的延迟，则可能没有必要。此外，不同类别之间的一些延迟差异可能可以由硬件特定的问题来解释。例如，对系统要求更高的负载可能会导致额外的缓存丢失，等等。而PREEMPT_RT补丁程序并不能完全解决这一点。

总之，本文的研究结果意义重大，因为本文表明，RISC-V上的PREEMPT_RT补丁无需太多额外工作即可运行。在启用了实时抢占模型后，测量的延迟是完全可预测的，并且通常在实时系统可接受的范围内。此外，在RISC-V体系结构上运行PREEMPT_RT似乎没有任何问题。但是，如果从配置中启用了一些更高级的内核特性，那么当前的RISC-V特定实现可能会出现一些基本问题。至少，如果从内核中启用了更多的特性，延迟性能可能会变得更糟。

尽管在Linux系统上执行精确的测量是具有挑战性的，但得到的测试结果是可复用且合理的。然而，由于中断导致的系统异步，延迟测量总是有一些不确定性。通过运行更长时间的延

迟测量，比如长达几天的时间，就可以观察到更精确的结果。不过，较短的测试持续时间仍然可以捕捉到延迟响应的总体变化。所提供结果中可能出现误差的一个实际原因是所使用的PREEMPT_RT补丁版本，因为最接近的版本是6.3内核。这意味着，在这个版本正式发布之前，有可能会添加一些PREEMPT_RT无法正确处理的新特性。然而，这种情况不太可能发生，而且追踪结果表明，在这方面不应该有任何问题。无论如何，使用本文中给出的结果作为一个保守估计是很好的。最后，在系统配置中有可能仍然存在一些疏忽。否则，所有其他误差源都应得到包容，并在测量中予以考虑。

6.2. 实时应用的适应性

所提出的系统是否适合实际应用，取决于其应用情况。这些测量结果总体上为各种实际应用（以不同方式强调系统）提供了很好的参考。然而，略微不同的配置，或选择的压力源不同可能会给出不同的结果。同样，应该注意的是，*cyclictest*可以给出一些乐观的数据，因此在参考这些数据以供实际使用时，需要仔细考虑这一点。即使是*cyclistest*程序本身也会干扰延迟测量。此外，所使用的最小实时内核不能代表实际应用程序，因此更改配置可能会带来额外的挑战。

无论如何，在这一点上，如果没有任何重要的特定于架构的延迟优化，系统似乎就已经实现了可接受的延迟。目前的系统可以被认为是软的或固定的，但肯定不具有硬的实时能力。这意味着，在理论上，所提出的系统将适用于工业项目，但在实践中，将需要由先前有丰富经验的PREEMPT_RT提供的官方支持。从kernel v6.6起，RT-Linux正式支持RISC-V架构。

当然，与工业应用的其他架构相比，RISC-V的其他特性提供了一些吸引人的优势。它是一个完全开放的生态系统，例如，开发完全的自定义指令级扩展更容易。RISC-V生态系统也非常现代，因为在系统上没有任何遗留负担。所有这些方面都很可能使一个在RISC-V平台上运行的实时Linux成为未来某些用例的理想系统。

6.3. 未来的工作

本文只介绍了在RISC-V架构上评估实时Linux的第一步。由于其他相关研究几乎不存在，因此有大量的可能性进一步研究这一主题。未来的工作可能包括，例如，测试一些额外内核选项的效果，评估更多的压力类别，以及研究一些内核优化，如与多核操作相关的事情。此外，来自不同硬件平台的实验、关于完全不同度量的测量以及实际的PREEMPT_RT开发工作在未来将是有价值的。进一步研究这些主题将为RISC-V架构在实际的工业实时应用中的可能性提供重要信息。

如前所述，本文中使用的内核配置是非常简单的，而实际应用的系统肯定需要对Linux内核的一些附加特性进行启用。因此，通过启用一些最重要的附加内核配置选项并测试它们如何影响延迟性能，进行进一步的研究将是有益的。排除明显的调试选项外，正常的实时Linux内核不会受到它的显著影响。然而，这些选项的某些部分可能会引入大量的延迟，因此需要进行一些优化工作，才能被认为可以在实时配置中使用。此外，同时甚至可能发现并修复与锁定和一般实时行为相关的其他错误。

由于Linux系统非常复杂，除了可以测量的延迟之外，还有许多其他有趣的测量标准。从内存使用、功耗和系统吞吐量等方面获得更多统计数据，这些都将成为在RISC-V上使用实时Linux其他功能非常有价值的信息。此外，由于本研究完全忽略了网络或其他导致大量中断的来源等影响，因此可以测试和评估更广泛的压力类别。设计这些额外的压力类别必须与启用额外的内核特性一起完成。

当前的系统有可能对本研究中使用的设置进行一些额外的优化，并实现更低的延迟。例如，通过设置CPU亲和位来将某些中断限制在某些核上，可能会略微改善其他核上所经历的延迟。此外，一些代码级的优化也是可能的。这种优化工作应该针对前面讨论的延迟最长的区域。有了这些变化，延迟可能会比本文中提出的还要小。此外，RISC-V体系结构本身正在不断开发中，并有定期提出和批准的新特性。例如，目前正在努力设计和实现一个更好、更灵活的中断控制器硬件。这个关于CLIC（核-本地中断控制器）的建议旨在为当前基于PLIC的架构添加全新的特性。重大的改进将包括支持中断抢占和选择向量控制等。这些特性对于某些对时间敏感的应用程序可能是有用的，甚至是至关重要的，因此在未来，RISC-V将更能够处理不同的实时工作负载。然而，应该注意的是，这些细节可能会发生变化，因为CLIC规范仍处于早期开发阶段。未来肯定还会有其他类似的新硬件块的开发。

除了RISC-V硬件之外，基础的软件支持也将继续改进。最重要的软件部分，例如编译器、调试工具和Linux内核已经对RISC-V提供了很大的支持，当然，它仍然落后于其他更老和更成熟的架构。例如，当使用RISC-V架构时，尚未完全支持Linux内核的每个特性。但可以肯定的是，不断增长的社区和商业利益将解决其中的某些问题。总的来说，整个RISC-V生态系统一直都在持续改善。

最后，在RISC-V体系结构上使用实时Linux的最重要的事情将是官方支持。这将在一定程度上保证其功能，因为到那时，使用该产品的社区将会更多。从Kernel 6.6 LTS起，RISC-V已成为RT-Linux官方支持，而赛昉科技的昉·惊鸿-7110大部分驱动代码已被Kernel 6.6接受，这对于昉·惊鸿-7110 SoC进入RT-Linux商业工业项目十分有利。基于本文，RISC-V体系结构应该完全适合用于要求更高的工业应用程序。

7. 结论

随着实时系统变得更加复杂，并且需要更高级的特性，使用完整的通用操作系统来帮助更快速地实现其中一些需求的好处变得越发具有吸引力。如果实时需求不太严格，那么使用Linux内核和众多官方支持的架构中的PREEMPT_RT补丁是目前实现这一目标最可行的方法之一。与此同时，RISC-V已经在工业领域得到了大量的关注，并且在未来，它可能会成为嵌入式设备领域的一个重要参与者。一旦商业RISC-V解决方案的可用性提高，Linux内核对RISC-V体系结构的支持也有所进展，它们就带来了一个非常有趣的平台，可以与其他体系结构竞争。不可避免的是，在某个时候，对运行实时Linux的高级RISC-V系统将会有真正的需求和支持。

通过本文中提出的一些小调整，RISC-V上的PREEMPT_RT补丁的核心看起来已经完全功能化了。最重要的是，它似乎不包含任何导致内核panic、死锁或其他重大问题的根本问题代码部分。此外，延迟测量显示了在实时使用场景上可能实现的结果。这种运行在RISC-V平台上的未经优化的实时Linux已经可以实现与其他官方支持的架构类似的结果。然而，评估的系统非常简单，因此还不一定代表一个实际的系统。

使用PREEMPT_RT后，即使在高负载的情况下，最大观察延迟也保持在150µs以下。在系统内存不足的严重错误情况下，最大延迟远低于250µs。与主线内核相比，这明显更低，主线内核在毫秒区域内经历了良好的延迟。总的来说，观察到的最大延迟似乎是非常确定的，因此它们不取决于系统负载，而系统负载是所有实时系统运行的唯一最重要的要求。这些数据令人印象深刻，因为目前还没有完成任何RISC-V架构或特定于驱动程序的延迟优化。导致PREEMPT_RT内核中仍然存在当前延迟的主要因素似乎与计时器中断以及偶发的自旋锁有关，优化其中一些部分可以进一步改善经历的延迟。

基于延迟测量，RISC-V上的PREEMPT_RT可能已经适用于一些不需要比200µs更好的响应时间的情况。没有任何错误的情况下，当前系统应该能够在该时间限制内可靠地响应。此外，RT-Linux还没有实际的应用，因为对于一些严重的问题，都需要有对LAZY_PREEMPT_RT补丁中的RISC-V架构的官方支持。可能除此之外，还需要有一些可靠的跟踪记录，以给予在工业项目中使用这样的系统足够的信心。我们至少还需要几年的时间才能考虑到这一点。同样值得注意的是，向内核中添加更多的选项可能会导致一些更长的延迟。这一问题今后还需要进行进一步的研究，以了解可能导致问题的不同选择。

总体来说，运行在RISC-V架构上的具有实时能力的Linux系统的未来可能会非常有前景。Linux的不同实时扩展将比以往任何时候都更加突出，而PREEMPT_RT将在该开发中发挥重要作用。将PREEMPT_RT补丁完全维护到Linux内核源代码只会加速开发，因为它会出现在一个更大的社区中。从Kernel 6.6起，RISC-V正式支持Linux系统。在官方的支持下，能够更好跟踪解决RT-Linux内核问题。此外，主线内核中的RISC-V架构代码，以及PREEMPT_RT本身与体系结构无关的部分，也将不断发展。甚至可能为Linux内核实现一些全新的特性，在未来实现更低的延迟。这项工作很大程度上将有助于未来采用RISC-V作为其他现有架构的可行替代方案。

8. 附录A：实时Linux补丁

1. RISC-V RT补丁下载链接：0001-riscv-allow-riscv-preempt_rt-config.patch
2. Config_Add_PREEMPT补丁下载链接：0002-enable-full-preempt-rt-config.patch
3. LAZY_PREEMPT补丁下载链接：0003-riscv-rt-add-riscv-lazy-preempt-support.patch
4. CPUPRI_workaround_RT_test补丁下载链接：0004-cpupri-a-work-around-for-non-rt-test-panic.patch

注：
请按上述顺序应用补丁。

若要应用以上补丁，请执行以下命令：

- 有git信息：

```
git am (patch)
```

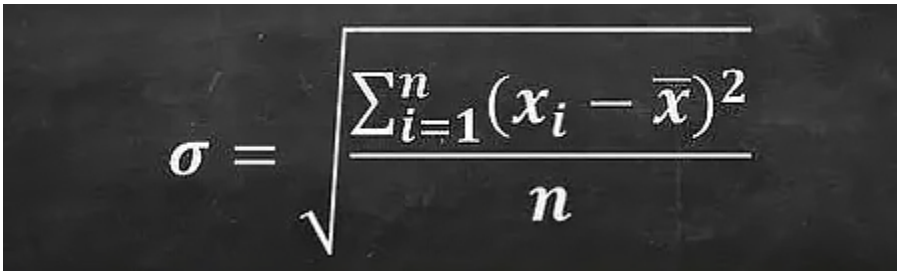
- 无git信息：

```
patch -p1 < (patch)
```

9. 附录B：标准差

标准差公式如下：

图 9-1 标准差计算公式



n为采样数，$\overline{x}$为采样数的平均值。

以下是一个标准偏差计算的例子。

共有4个延迟计数，分别是4、4、6、6。n为4，平均值为5。标准差值为1。