# 一、英文题目

Design and Implementation of a Shared Scheduler in QEMU Emulator

# 二、选题依据

## 1. 背景和研究意义

在操作系统的发展历程中，任务调度一直是一个重要的课题。在多个任务需要处理时，选择合适的处理顺序、实现高效的任务切换机制，可以提高计算机系统处理任务的性能。

随着计算机技术的发展，任务调度机制的发展呈现出以下几个趋势：一是适应硬件技术的进步，例如从单核任务调度到多核任务调度；二是调度粒度愈发精细，例如调度的对象从进程到线程，之后又出现了比操作系统线程更精简的调度对象，称为协程/轻量级线程/用户级线程（下文中统称协程）。三是调度机制的目标更加专门化，例如应用于嵌入式实时系统的任务调度机制需要尽量缩短任务的反应时间，而应用于高性能服务器的调度机制则要求高吞吐量和低时延。

这其中，以协程为单位的调度往往出现在进程和线程都无法满足性能要求的情景，因此具有比较苛刻的性能要求。同时因为其对任务的划分比较精细，通常具有较高的调度频率。因此，如果能减小每次调度时，任务选择和切换的开销，可以明显提升系统的性能。本课题的目标是实现一种基于硬件的协程调度器，它在保留了中心化调度的优势的前提下，一方面可以取消原有的调度器线程，使更多线程用于处理任务；另一方面可以利用硬件来加速任务的调度，从而提升高频率协程调度场景下的任务处理性能。

Rust语言是一门适合系统编程的新兴编程语言，其拥有高于C语言的内存安全性、更现代的语法特性和等同C语言的性能。RISC-V指令集则是一套开源、简洁、模块化的指令集。因此，基于Rust语言和RISC-V指令集的操作系统是操作系统开发的新兴方向之一。这一方向上已经出现了一些教学和科研目的的操作系统，例如rCore教学操作系统[1]。本项目设计的调度器同样适配Rust语言和RISC-V指令集，可以丰富这一方向的软硬件生态。

## 2. 国内外研究概况

好的任务调度器，需要同时满足两个要求：科学的调度策略和低开销的实现机制[2]。因此，调度策略和实现机制就称为了研究任务调度的两大方向。由于本研究注重实现机制方向，因此也重点介绍实现机制方面的研究概况。

在国际上，调度器的实现机制主要有以下研究和改进：Jupyung Lee等人改进了中断处理机制，从而降低高优先级任务的中断开销[3]。因为任务的切换多伴随着中断，因此降低中断处理开销也是实现机制的改进方法。Kostis Kaffes等人实现了一个以微秒级调度协程的调度器[2]。因为操作系统提供的机制在微秒时间尺度下开销太大，该调度器实现了自己的抢占机制和上下文切换机制。其使用中心化调度，由一个调度线程和多个工作线程组成，导致真正进行任务处理的线程比系统使用的线程少了一个。Rishabh Iyer等人在这个思路上改进，使用编译器代码插桩、二级队列等机制降低了任务切换开销和工作线程的等待时间[4]。它同时让调度线程在空闲时也处理任务，降低了使用调度线程对效率的影响，然而无法完全消除。

在国内，主要有以下研究和改进：曾素华等人将一部分满足条件的任务切换转化为函数调用，减少开销[5]。钱宏文等人引入FPGA硬件辅助CPU计算，并改进进程调度机制，将进程分配到CPU或FPGA上运行[6]。该方法创新之处在于使用FPGA作为协处理器，研究和实现了将进程在CPU和协处理器上调度的策略；不足在于将进程传递到FPGA需要进行bit流形式的传输，可能影响效率。尹震宇等人使用硬件机制调度线程，并为此建立数学模型，实现DR-EDF调度算法[7]。该方法的优势在于大幅降低了线程切换的开销，因此可以通过更频繁的调度达到更短的响应时间。不过，协程这一更细粒度、更轻量的调度单位比线程更能利用硬件机制提供的低切换开销和高调度频率。

# 三、研究目标和内容

## 1. 研究目标和主要内容

本研究拟分为原型设计和对比测试两部分。首先在QEMU模拟器中设计并实现基于硬件的共享调度器，并验证其正确性，再将其和现有的任务调度机制进行性能测试，对比分析测试结果。

本研究会在QEMU源码中增加一个新的模拟设备，使用C语言实现。其将适配采用RISC-V指令集的硬件系统，以及Rust语言提供的协程机制，以协程作为调度单元。该设备内部会维护多个队列代表不同优先级，操作系统可以向其中放入准备就绪的任务指针，并从中获得下一个运行的任务。该设备同时会承担外部中断处理的功能。接受到中断信号后，其根据中断向量表将信号对应的中断处理程序作为高优先级协程，以供操作系统在下一次切换任务时获取。

本研究同时为该硬件开发一些附属软件，以实现提供软件接口等功能。

本研究所实现的软硬件系统需要达到的目标为：可以正常工作并给出期望结果；能够发挥硬件调度的性能优势；具有可扩展性，服务多核CPU时表现较好。本研究的对比测试部分需要达到的目标为：设计科学、完善的对比实验并得出可靠结果；分析产生实验结果的原因；分析本研究实现的各个机制对结果的影响。

## 2. 拟解决的关键问题

在对协程进行调度时，若调度频率较高，软件实现的调度器会产生较大的开销。同时，软件实现的调度机制需要一个线程来运行调度器本身，这减少了可以处理工作负载的线程数。本研究拟采用硬件实现调度器的方案来解决这一关键问题，尽量降低任务切换和中断处理的CPU资源开销和时间开销，同时使更大比例的CPU资源用于处理工作负载。

# 四、研究方案

**1. 研究方法**

本项研究在不同阶段采用的研究方法如下：

* + 在调研阶段，采用文献研究法。学习QEMU模拟器上硬件开发、Rust协程调度、RISC-V中断处理的相关知识。同时，调研现有的任务调度器的设计方案，从中学习值得借鉴的设计思路。
  + 在设计开发和对比测试阶段，采用实验研究法和定性分析法。设计对比实验，测试本项研究设计的调度器和现存几个典型调度器在不同外部条件下的性能。并且，采用定性分析的方式，根据不同调度器（特别是本研究设计的调度器）的实现机制，分析实验结果。

**2. 开发阶段的技术路线**

本次开发的调度器主要分为：在QEMU模拟器中运行的硬件部分和在虚拟机中运行的软件部分。

硬件部分，本研究对QEMU的源代码进行增量开发，在其中加入本研究实现的协程调度器。QEMU是一个开源、通用的模拟器，可以模拟与宿主机不同体系结构的计算机。本研究使用QEMU模拟外部设备的机制开发协程调度器，采用C语言进行编码，采用QEMU提供的面向对象模型、硬件模型和内存映射机制进行开发。其中，面向对象模型为C语言提供面向对象编程的支持；硬件模型在QEMU中注册本研究实现的协程调度器；内存映射机制将调度器对外可见的寄存器映射到虚拟机的虚拟内存，使得虚拟机可以通过内存地址访问共享调度器。

软件部分，使用Rust语言开发，以适配Rust语言编写的操作系统。其通过访问虚拟内存访问协程调度器，并向操作系统提供调度器的软件接口。

**3. 实验阶段的实验方案**

本研究采用的实验方法为对比实验，分别在本研究实现的调度器和现存典型调度器之间进行横向对比，以及在本研究调度器的不同版本（可能是参数不同或者某些特性的启用/关闭）之间进行纵向对比。实验需控制无关变量一致，包括：宿主机软硬件运行环境一致；均运行在版本一致的QEMU模拟器上；测量性能指标的方法一致。之后，在不同的研究变量（例如：调度周期、模拟的CPU核心数、等等）下，对比各个研究对象的性能差距。

**4. 可行性分析**

在研究正式开始前，已经进行了初步的调研：对操作系统的任务调度进行更深的理解；了解了QEMU硬件开发、Rust协程、RISC-V中断处理相关内容；也阅读了当前对任务调度器的一些研究成果。同时，对本研究的协程调度器的设计方案，目前已有初步构思，暂未遇到困难。本研究的题目已获得指导老师的认可，并有指导老师和研究方向相近的学长在理论研究、技术路线等方面提供指导。基于上述这些原因，认为该题目可行。

# 五、研究计划及进度安排

2023年11月 确定研究选题

2023年12月 进行开题工作

2023年12月~2024年1月 调研，学习相关知识；设计

2024年1月~2024年3月 开发，正确性测试

2024年3月~2024年4月 性能测试，实验

2024年4月 进行中期检查工作

2024年4月~2024年5月 撰写和完善论文

2024年5月 进行论文提交

2024年6月 进行毕业设计答辩

# 六、创新点及预期研究成果

**1. 创新点**

本研究的创新点在于以下三点：

第一，将硬件实现的调度机制与细粒度、高频率的协程调度相配合，充分利用硬件机制实现的低开销任务切换，以达到任务处理的低延迟和高吞吐量。

第二，将外部中断的处理转化为协程调度，省去了通用寄存器和控制状态寄存器的保存步骤，降低了中断处理的时间开销。

第三，该研究面向Rust语言实现和RISC-V指令集架构的计算机系统，可以一定程度上丰富该平台上的软硬件生态。

**2. 预期研究成果**

* 完成研究内容，达成研究目的。
* 设计并在QEMU中实现一个基于硬件的、以协程为单位的共享调度器。
* 对本研究设计的调度器开展性能测试和对比实验，得出实验结果。
* 提交开题报告、中期报告、毕业论文等文字成果。

# 参考文献

[1] 孙卫真,刘雪松,朱威浦,等. 基于RISC-V的计算机系统综合实验设计[J]. 计算机工程与设计,2021,42(4):1159-1165. DOI:10.16208/j.issn1000-7024.2021.04.037.

[2] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, et. al. Shinjuku: Preemptive Scheduling for μsecond-scale Tail Latency[C]. 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), 2019: 345-360, https://www.usenix.org/conference/nsdi19/presentation/kaffes

[3] Jupyung Lee and Kyu Ho Park. 2010. Interrupt handler migration and direct interrupt scheduling for rapid scheduling of interrupt-driven tasks[J]. ACM Trans. Embed. Comput. Syst. 9, 4, Article 42 (March 2010), 34 pages. https://doi.org/10.1145/1721695.1721708

[4] Rishabh Iyer, Musa Unal, Marios Kogias, and George Candea. 2023. Achieving Microsecond-Scale Tail Latency Efficiently with Approximate Optimal Scheduling[C]. In Proceedings of the 29th Symposium on Operating Systems Principles (SOSP '23). Association for Computing Machinery, New York, NY, USA, 466–481. https://doi.org/10.1145/3600006.3613136

[5] 曾素华,蒋建春. OSEK操作系统抢占式调度策略改进[J]. 计算机工程与应用, 2010, 46(34):228-231. DOI:10.3778/j.issn.1002-8331.2010.34.068.

[6] 钱宏文,张飞,吴翼虎,等. 局部动态可重构FPGA进程式调度系统设计与实现[J]. 电子技术应用,2023,49(3):114-117. DOI:10.16157/j.issn.0258-7998.222818.

[7] 尹震宇,赵海,等. 一种面向硬件线程的实时调度算法研究与设计[J]. 电子学报, 2007,35(8):1467-1471. DOI:10.3321/j.issn:0372-2112.2007.08.009.