操作系统课程设计

Xv6实验报告

Lab6 多线程

|  |  |
| --- | --- |
| Name | 朱从周 |
| ID | 2351893 |
| Number | 42028701 |
| major | 软件工程 |



同济大学

Tongji University

1. **环境搭建**
2. 下载VMware虚拟机，并且下载 Ubuntu 镜像并安装 Ubuntu 系统。
3. 启动Ubuntu，安装本项目所需的所有软件，运行：

$ sudo apt-get update && sudo apt-get upgrade

$ sudo apt-get install git build-essential gdb-multiarch qemu-system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu

1. 下载xv6内核源码

$ git clone git://github.com/mit-pdos/xv6-riscv.git

1. 更新镜像源

$ sudo nano /etc/apt/sources.list

$ sudo apt-get update

1. 获取源码

$ git clone git://g.csail.mit.edu/xv6-labs-2021

$ cd xv6-labs-2021

$ git checkout cow

**1. 实验目的**

本实验旨在设计并实现一个用户级线程系统的上下文切换机制。补充完成一个用户级线程的创建和切换上下文的代码。需要创建线程、保存/恢复寄存器以在线程之间切换，并且确保解决方案通过测试。通过使用线程和锁实现并行编程，以及在多线程环境下处理哈希表。学习如何使用线程库创建和管理线程，以及如何通过加锁来实现一个线程安全的哈希表，使用锁来保护共享资源，以确保多线程环境下的正确性和性能。

另外通过实现一个线程屏障（barrier），即每个线程都要在 barrier 处等待，直到所有线程到达 barrier 之后才能继续运行，加深对多线程编程中同步和互斥机制的理解。在多线程应用中，线程屏障可以用来确保多个线程在达到某一点后都等待，直到所有其他参与的线程也达到该点。通过使用pthread条件变量，我们将学习如何实现线程屏障，解决竞争条件和同步问题。

**2. 实验内容**

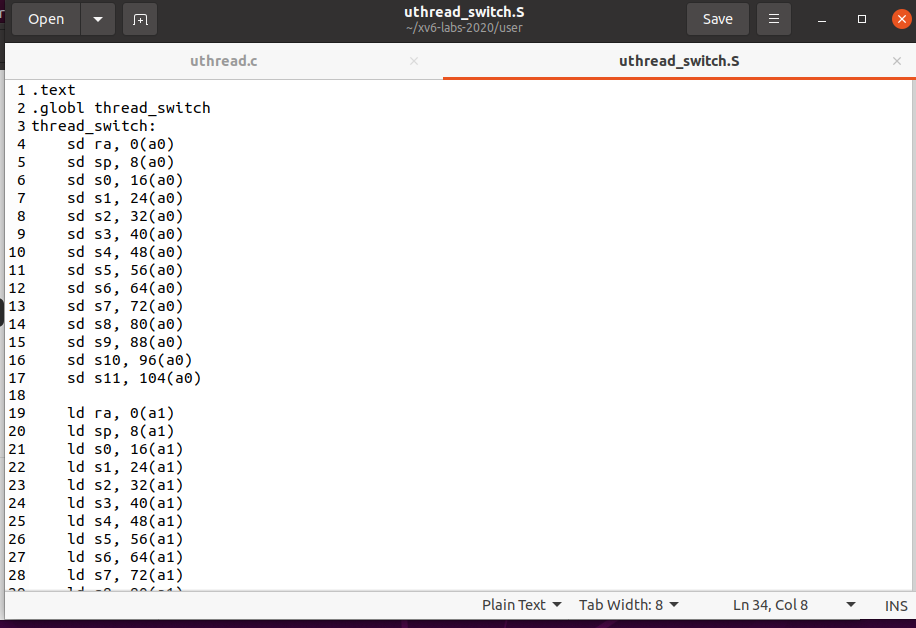
**Uthread: switching between threads (moderate)**

1、设置线程上下文结构体.

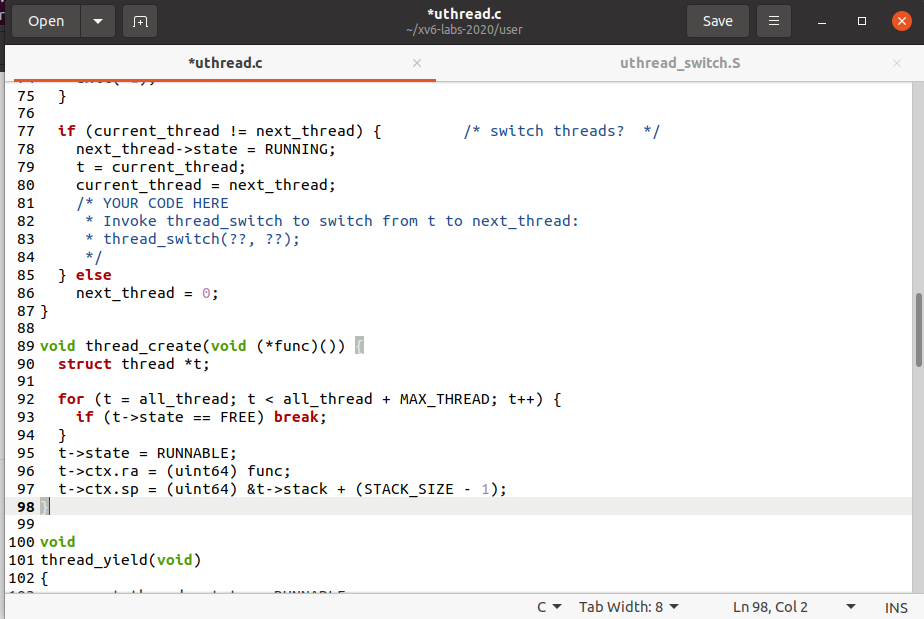
所以在 user/uthread.c 中定义一个 struct context 来保存 callee-save registers 的信息。



**实现 thread\_switch 函数**： 该函数用于保存当前线程的上下文，并恢复下一个线程的上下文。其实现与内核中的 swtch 函数类似。



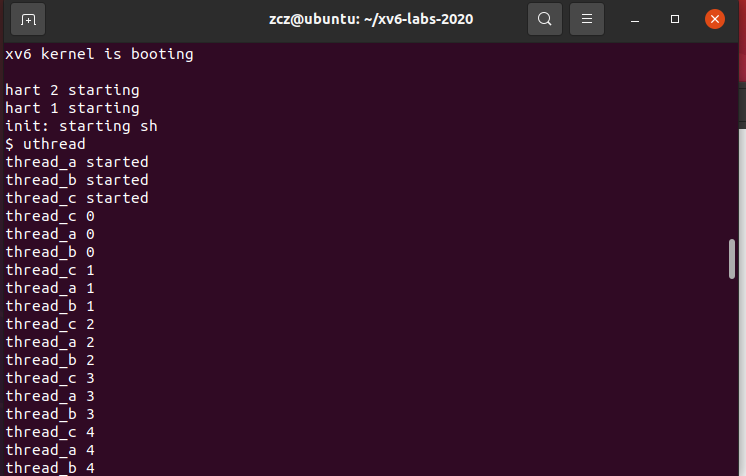
**实现 thread\_create 函数**： 该函数用于创建新的线程。为每个线程分配栈空间，并初始化线程的上下文，使其能够正确地执行用户提供的函数。

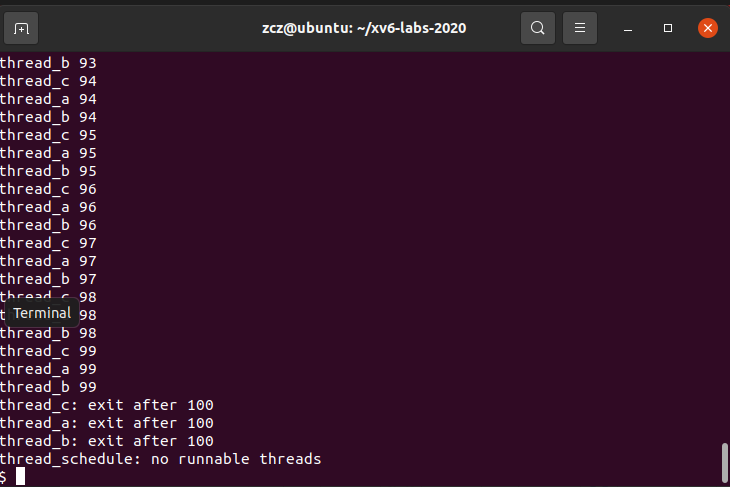


**实现 thread\_schedule 函数**： 该函数用于调度线程。遍历所有线程，找到一个 RUNNABLE 状态的线程，并切换到该线程执行。

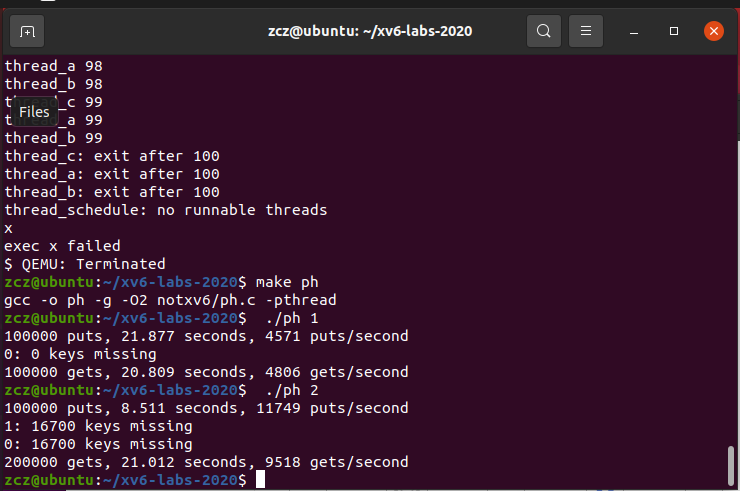


**测试结果：**

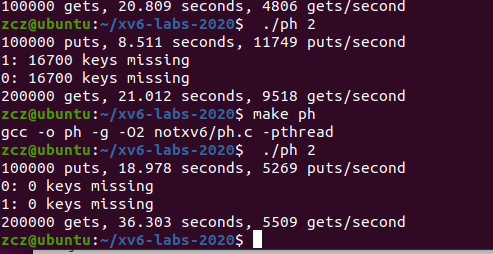




**Using threads (moderate)**

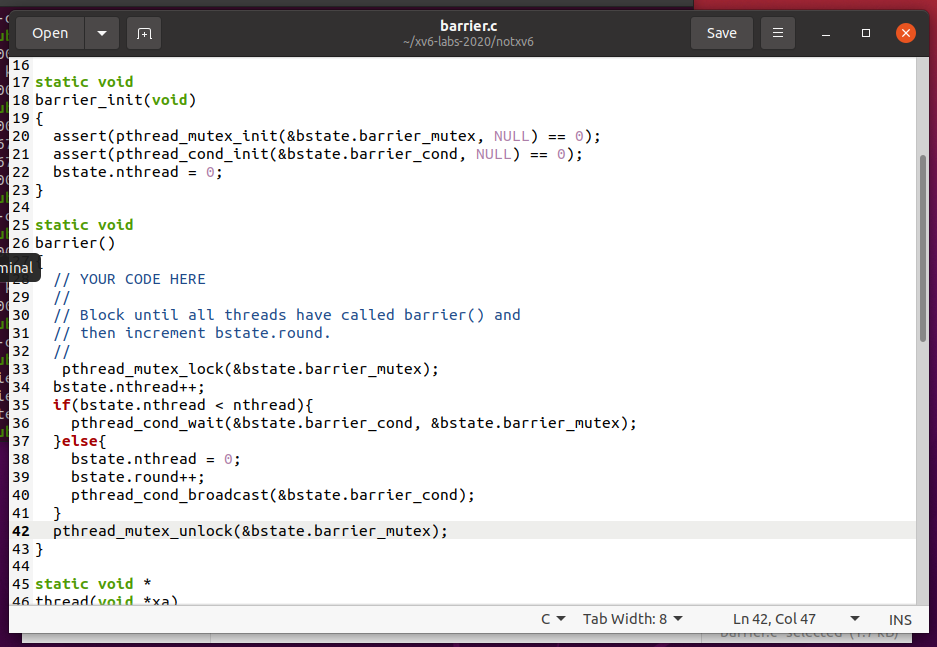


修改加锁后两个线程运行不会有 key 丢失, 同时可以看到 put 的性能相比加锁之前有所减小,

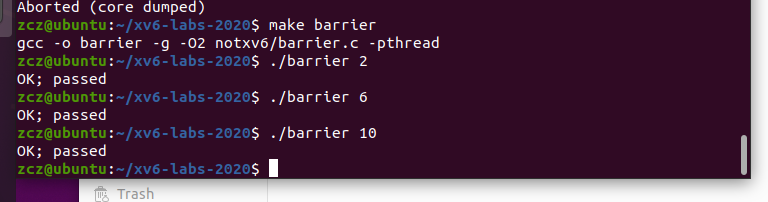


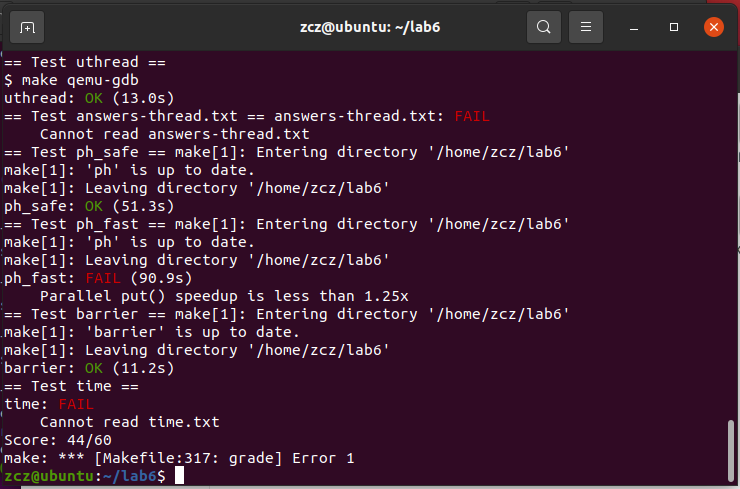
**Barrier**

**实现屏障函数**： 实现 barrier 函数，使用互斥锁和条件变量来实现同步屏障。



测试结果：





**3.实验心得**

首先我更好地理解了寄存器的功能特性。在创建线程时，正确分配线程的堆栈空间是关键。通过理解 RISC-V 架构中寄存器的功能特性，我能够选择适当的寄存器来存储必要的信息，比如函数指针和栈顶指针。这样，当线程被调度执行时，它能够正确跳转到函数起始位置，并且在自己的独立栈上执行，避免与其他线程的干扰。

这个实验让我更深入地理解了线程的概念以及线程切换的底层实现机制。通过充分利用寄存器的特性和调试工具，我成功地解决了遇到的问题，并完成了实验任务。

当我参与使用线程进行并行编程的实验时，我深刻地了解到了多线程编程的挑战和重要性。本次实验涉及了使用线程和锁来实现一个哈希表，并探索了在多线程环境下的表现。我明白了锁在多线程环境中的关键作用。使用锁可以确保在访问共享资源时的线程安全性，防止竞争条件和数据不一致的问题。我学会了如何使用 pthread\_mutex\_t 类型的锁，并且确保在使用锁时始终遵循正确的获取和释放顺序，以避免死锁情况的发生。此外，我还了解到在锁的创建和销毁方面需要注意性能问题，因为未销毁的锁可能会影响系统的资源和性能。

其次，我在优化并发性能方面获得了一些见解。我意识到过多的锁可能会导致性能下降，因此需要在保证正确性的前提下尽量减少锁的使用。特别是，我学会了如何将锁的粒度缩小，以便最大程度地减少锁竞争，从而提高并行性和性能。我在实验中发现，为每个哈希桶添加独立的锁是一个有效的优化方法，可以显著提升多线程执行的效率。