****



**MIT xv6实验报告**

**——lab7 : thread**

**学生姓名 胡轶然**

**学 号 3019244355**

1. **实验目的**
2. 实现线程切换、多线程加速和barrier（线程同步）。
3. 熟悉xv6的线程工作原理。
4. **前期准备**
5. 切换git分支。
6. 阅读指导书第7章，了解与多线程相关的知识。
7. 阅读相关代码。
8. **实验内容及实现步骤**

**任务1 Uthread: switching between threads**

1. **问题描述**

本任务要求实现用户级线程之间的切换。

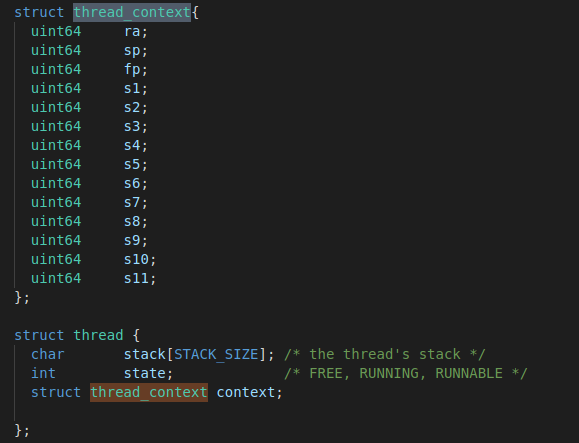
1. **思路与实现步骤**

本任务的主要工作是切换线程时对寄存器的保存和恢复。两个线程间切换时，需要保存被切换线程的callee saved寄存器，同时将切入cpu的线程中保存的callee saved寄存器的值存入cpu寄存器中。

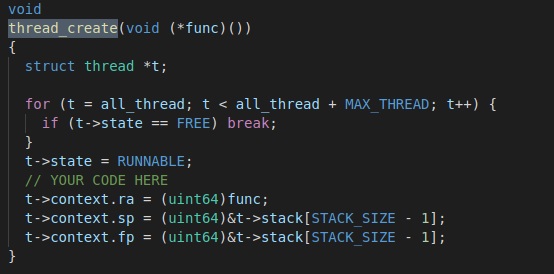
可将任务分为3部分：定义存储寄存器的结构体、在创建线程时初始化该结构体和使用汇编码实现线程切换时，寄存器的保存和恢复。

具体修改内容如下：

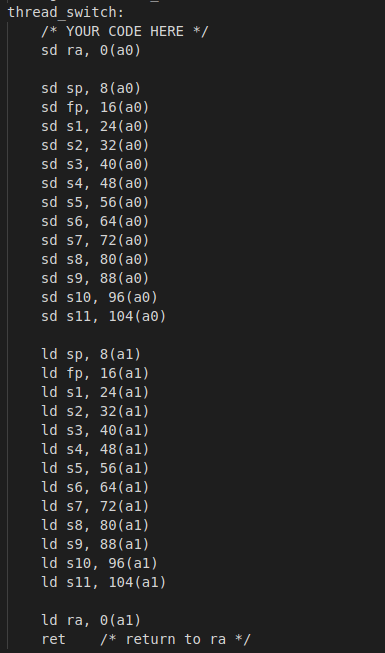
1. 在thread结构体中保存callee saved寄存器。定义thread\_context结构体，并添加为thread结构体的元素。



1. 编写thread\_create函数，在创建线程时初始化context内容。将context中的ra寄存器指向返回地址，sp、fp寄存器指向栈底。



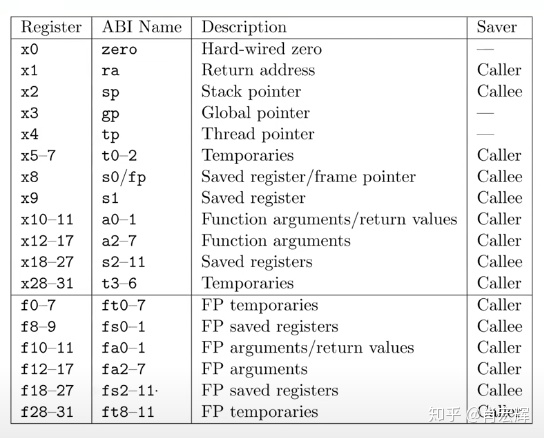
1. 参考kernel/swtch.S文件，在[uthread\_switch.S](https://github.com/weijunji/xv6-6.S081/commit/34ea60fd693130b0f9817e0f3a6dcd4b1938d604" \l "diff-b9a53a58306c9e71e0b16c68c1d0aac2a44295d12bfd20c9936b1960ed28b15d" \o "user/uthread_switch.S)文件中，编写thread\_switch函数，使用sd指令将cpu中callee saved寄存器中的值存入被切换线程的thread\_context结构体中。
2. 参考kernel/swtch.S文件，在[uthread\_switch.S](https://github.com/weijunji/xv6-6.S081/commit/34ea60fd693130b0f9817e0f3a6dcd4b1938d604" \l "diff-b9a53a58306c9e71e0b16c68c1d0aac2a44295d12bfd20c9936b1960ed28b15d" \o "user/uthread_switch.S)文件中，编写thread\_switch函数，使用ld指令将切入线程中保存的callee save存器的值存入cpu对应寄存器中。



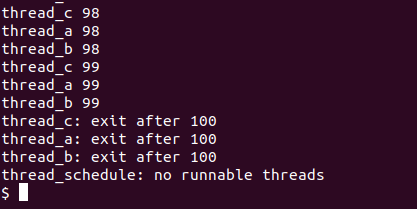
1. 在thread\_schedule函数中添加对thread\_switch函数的调用。（图略）
2. **问题与解决方法**

**问题一**：为什么不用保存caller saved寄存器？

**解决方法**：如下图所示，**caller saved寄存器是由被调用者保存的寄存器，在调用完成后，被调用者会负责恢复caller saved寄存器的值**，所以不用在调用者线程的数据结构中保存caller saved寄存器。



1. **结果**



**任务2 Using threads**

1. **问题描述**

本任务要求修改原代码对哈希表（字典）的实现方法，使之能在多线程环境下安全、高效地运行。

1. **思路与实现步骤**

原代码**使用无锁的哈希链表实现哈希表**，当多个线程同时操作哈希表时，不能保证操作的正确性和安全性。

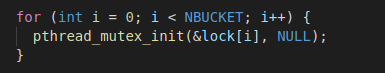
为增强安全性，只需为每个链表定义一个锁，并为字典的查找和插入操作加锁即可。

具体实现方法如下：

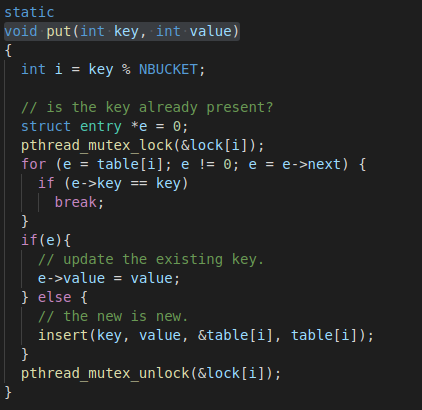
1. 为每个链表定义一个锁。



1. 在ph.c文件的main函数中初始化所有的锁。



1. 在put函数中，为更新链表元素和向链表中插入新元素的操作**加锁**，从而保证并行操作哈希表的安全性。



1. **问题与解决方法**

**问题一**：最初没有为每个链表分配一个锁，而是令**所有链表共用一个锁，导致了较严重的锁竞争现象**，无法达到题目对速度的要求，不能通过ph\_fast测试。

**解决方法**：请教同学后，**为每个链表分配一个锁，保证了链表之间的隔离性，使得线程能够并行地操作多个链表。**

1. **结果**

见make grade

**任务3：Barrier**

1. **问题描述**

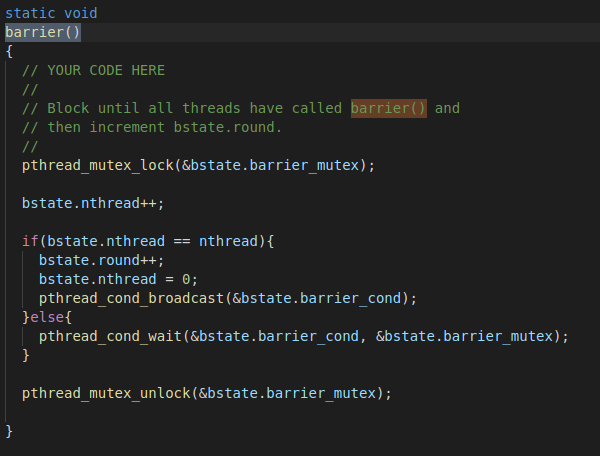
本程序要求添加一个barrier函数，从而实现线程进度的同步。

1. **思路与实现步骤**

本任务与上学期并行计算课程的内容相似，只需实现一个新函数，统计遇到barrier的线程数量，当计数达到线程总数后通过广播通知所有线程停止等待，继续运行即可。

具体实现如下：

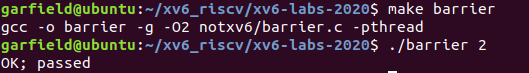
1. 定义一个barrier函数，当有线程调用barrier函数时，计数器加一。
2. 若加一后计数未达到线程总数，说明还有线程未执行到barrier函数，则调用wait函数，令当前线程进入等待状态。
3. 加一后，若计数达到线程总数，说明所有线程都已遇到barrier，此时广播所有线程停止等待，继续运行。



1. **困难与解决方案**

本任务较为简单，只需调用参考书中提供的函数即可，上学期的并行计算课程已经介绍了各函数的功能。

1. **结果**



1. **实验结果**

通过make grade测试。

