**实验报告**

**Lab 1**

**MIT：Multithreads**

姓名：赵笑宇

班级：20信息安全

学号：20307130172

**实验报告填写要求**

**1.请在每个exercise之后简要叙述实验原理，详细描述实验过程。可以使用中文进行描述，不对语言做要求。**

**2.请将你认为的关键步骤附上必要的截图。**

**3.有需要写代码的实验，必须配有代码、注释以及对代码功能的说明。**

**4.你还可以列举包括但不局限于以下方面:实验过程中碰到的问题你是如何解决的、实验之后你还留有哪些疑问和感想。**

**5.如果实验附有练习，请在每个练习之后作答，这是实验报告评分的重要部分。**

**6.Challenge为加分选作题。每个lab可能有多个challenge,我们会根据完成情况以及难度适当加分，具体情况会在课上说明。这部分的实验过程描述应该比exercise更加详细。**

**7.切勿抄袭亦或是去互联网复制粘贴答案。**

【练习题模板】

1. Question
2. Code
3. Screenshot
4. Difficulties and solutions

（注）：以下问题均为从MIT在线网站上翻译得到

## 壹1.Question：Uthread：线程之间切换（[中等）](https://pdos.csail.mit.edu/6.S081/2022/labs/guidance.html))

在本练习中，您将为用户级线程系统设计上下文切换机制，然后实现它。为了帮助您入门，您的 xv6 有两个文件用户/uthread.c 和用户/uthread\_switch。S，以及生成文件中用于生成 uthread 程序的规则。uthread.c 包含大部分用户级线程包，以及三个简单测试线程的代码。线程包缺少一些用于创建线程和在线程之间切换的代码。

您的工作是制定一个计划来创建线程并保存/恢复寄存器以在线程之间切换，并实施该计划。完成后，“使评分”应表示你的解决方案通过了 uthread 测试。

完成后，在 xv6 上运行 uthread 时，应会看到以下输出（三个线程可能以不同的顺序启动）：

$ **make qemu**

...

$ **uthread**

thread\_a started

thread\_b started

thread\_c started

thread\_c 0

thread\_a 0

thread\_b 0

thread\_c 1

thread\_a 1

thread\_b 1

...

thread\_c 99

thread\_a 99

thread\_b 99

thread\_c: exit after 100

thread\_a: exit after 100

thread\_b: exit after 100

thread\_schedule: no runnable threads

$

此输出来自三个测试线程，每个线程都有一个循环，该循环打印一行，然后将 CPU 让给其他线程。

但是，此时，如果没有上下文切换代码，则不会看到任何输出。

您需要在用户/uthread.c 中向 thread\_create（） 和 thread\_schedule（） 添加代码，并在用户/uthread\_switch中添加thread\_switch。S. 一个目标是确保当thread\_schedule（）首次运行给定线程时，线程在其自己的堆栈上执行传递给thread\_create（）的函数。另一个目标是确保thread\_switch保存被切换的线程的寄存器，恢复被切换到的线程的寄存器，并返回到后一个线程指令中上次中断的位置。您必须决定在哪里保存/恢复寄存器;修改结构线程以保存寄存器是一个很好的计划。您需要在thread\_schedule中添加对thread\_switch的呼叫;你可以传递任何你需要thread\_switch的参数，但目的是从线程t切换到next\_thread。

一些提示：

* thread\_switch只需要保存/恢复被叫方保存寄存器。为什么？
* 您可以在用户/uthread.asm 中看到 uthread 的程序集代码，这可能便于调试。
* 要测试您的代码，使用 riscv64-linux-gnu-gdb 单步执行thread\_switch可能会有所帮助。您可以通过以下方式开始：
* (gdb) file user/\_uthread
* Reading symbols from user/\_uthread...
* (gdb) b uthread.c:60

这会在 uthread.c 的第 60 行设置一个断点。断点可能会（也可能不）在运行 uthread 之前触发。怎么会这样呢？

一旦你的 xv6 外壳运行，键入“uthread”，gdb 将在第 60 行断开。现在，您可以键入如下命令来检查 uthread 的状态：

(gdb) p/x \*next\_thread

使用“x”，您可以检查内存位置的内容：

(gdb) x/x next\_thread->stack

您可以跳到thread\_switch的开头，如下所示：

(gdb) b thread\_switch

(gdb) c

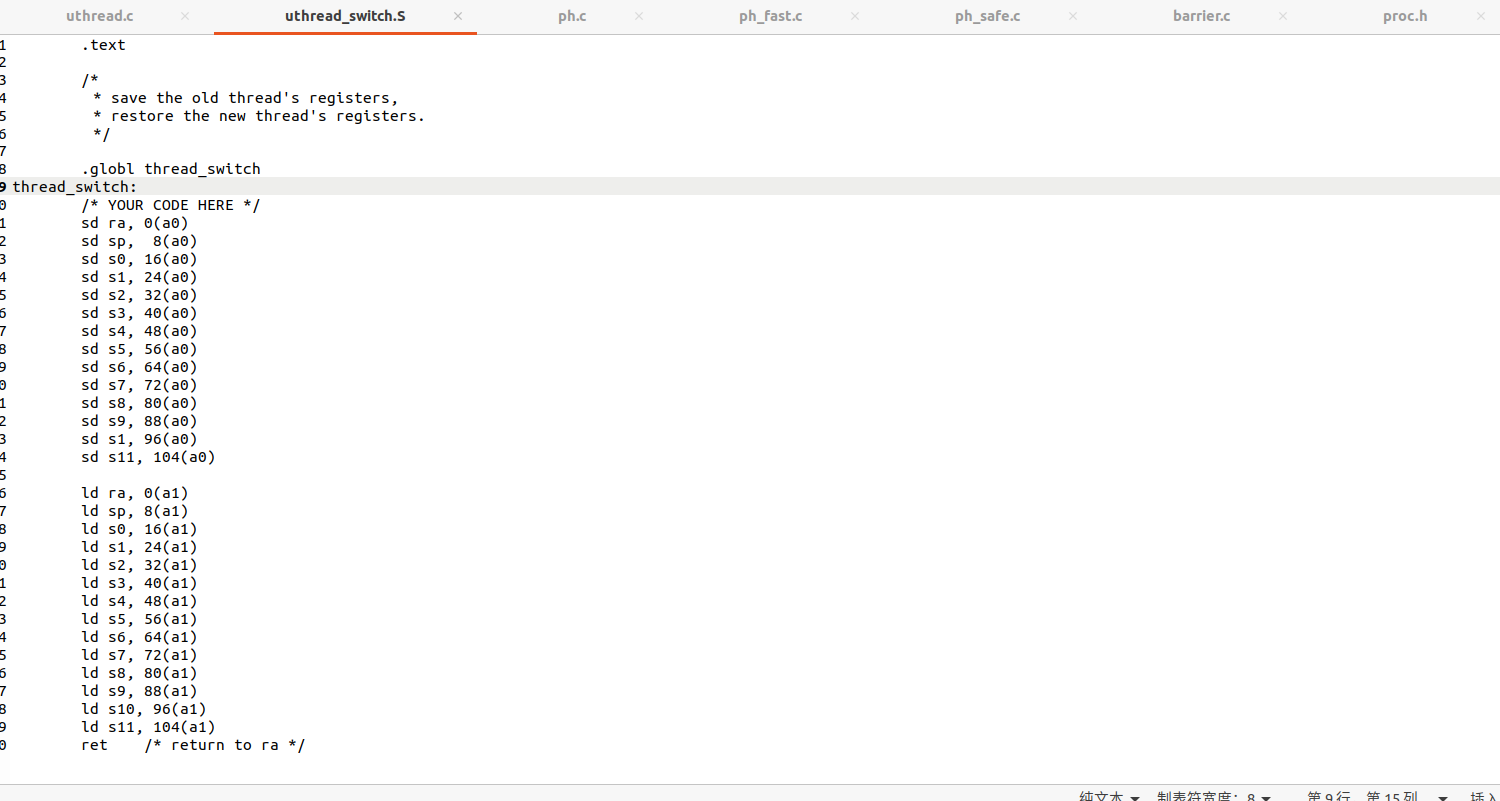
您可以使用以下命令执行单步组装说明：

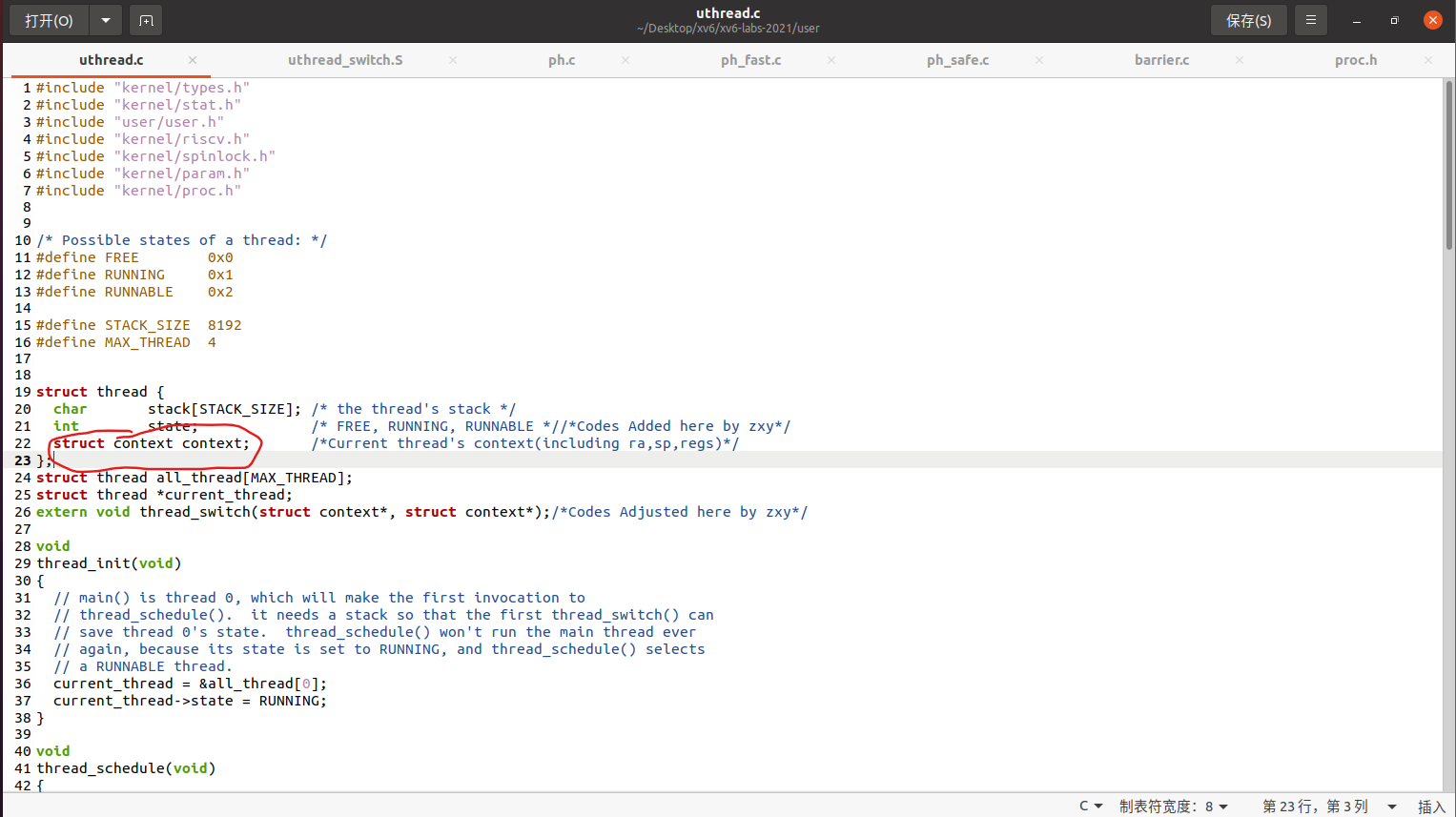
(gdb) si

gdb 的在线文档[在这里](https://sourceware.org/gdb/current/onlinedocs/gdb/)

**2.Codes**

**这里仅展示已经修改的代码部分**

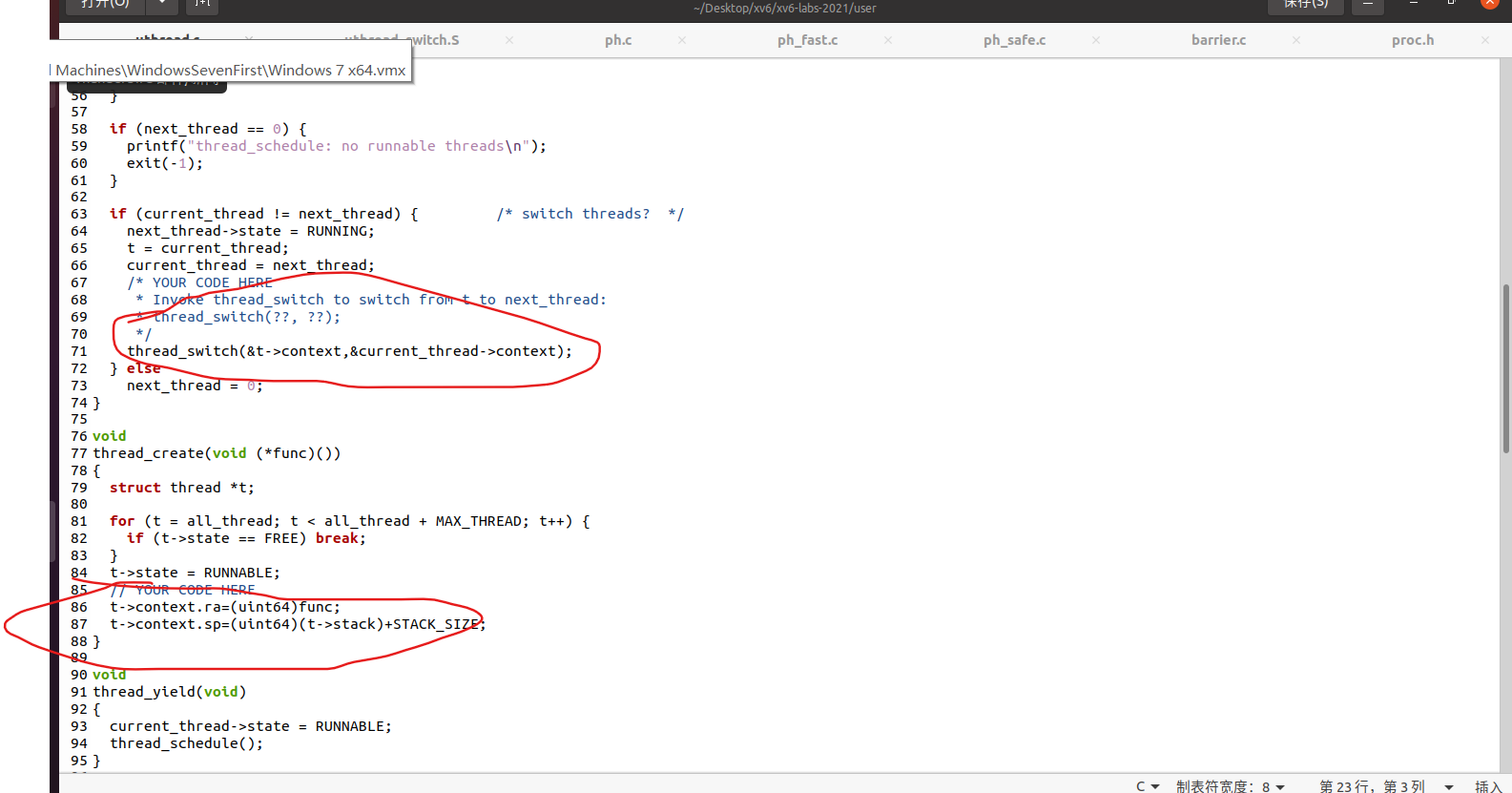


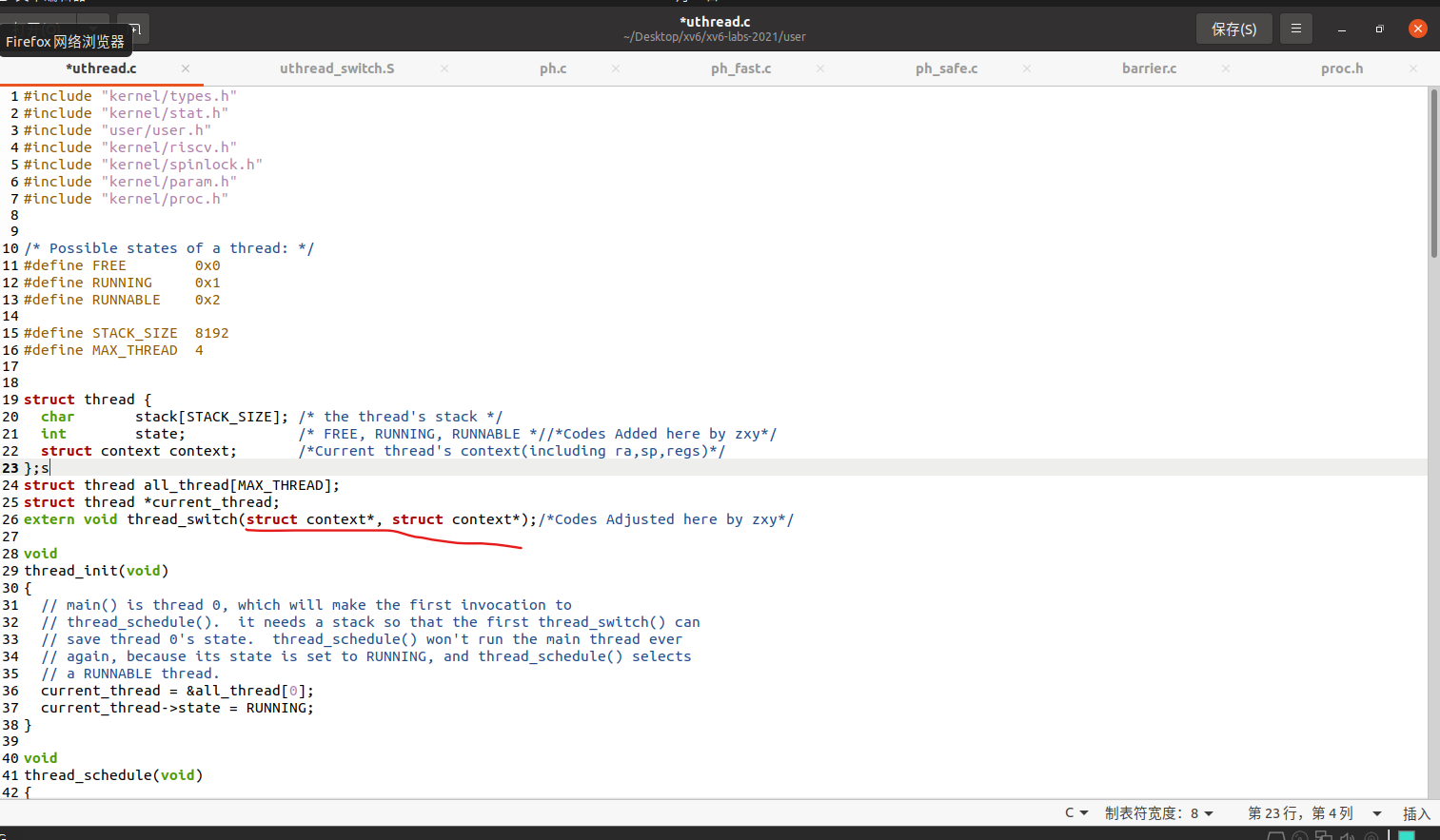


这里的context文件为proc.h文件里面所定义的

Ra与Sp寄存器外加12个寄存器总共14个寄存器

这里有大问题 待会儿在第四部分会提到

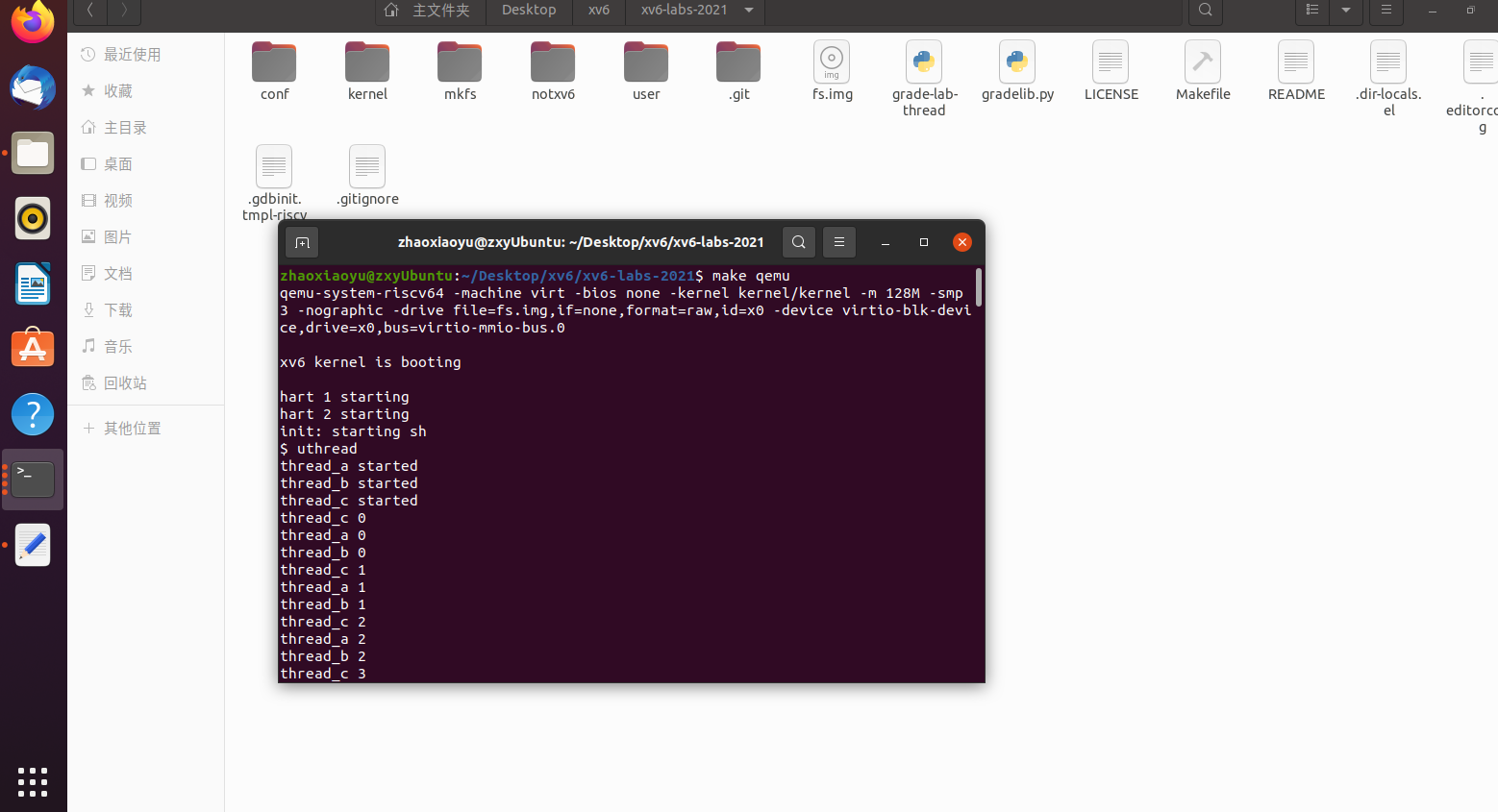


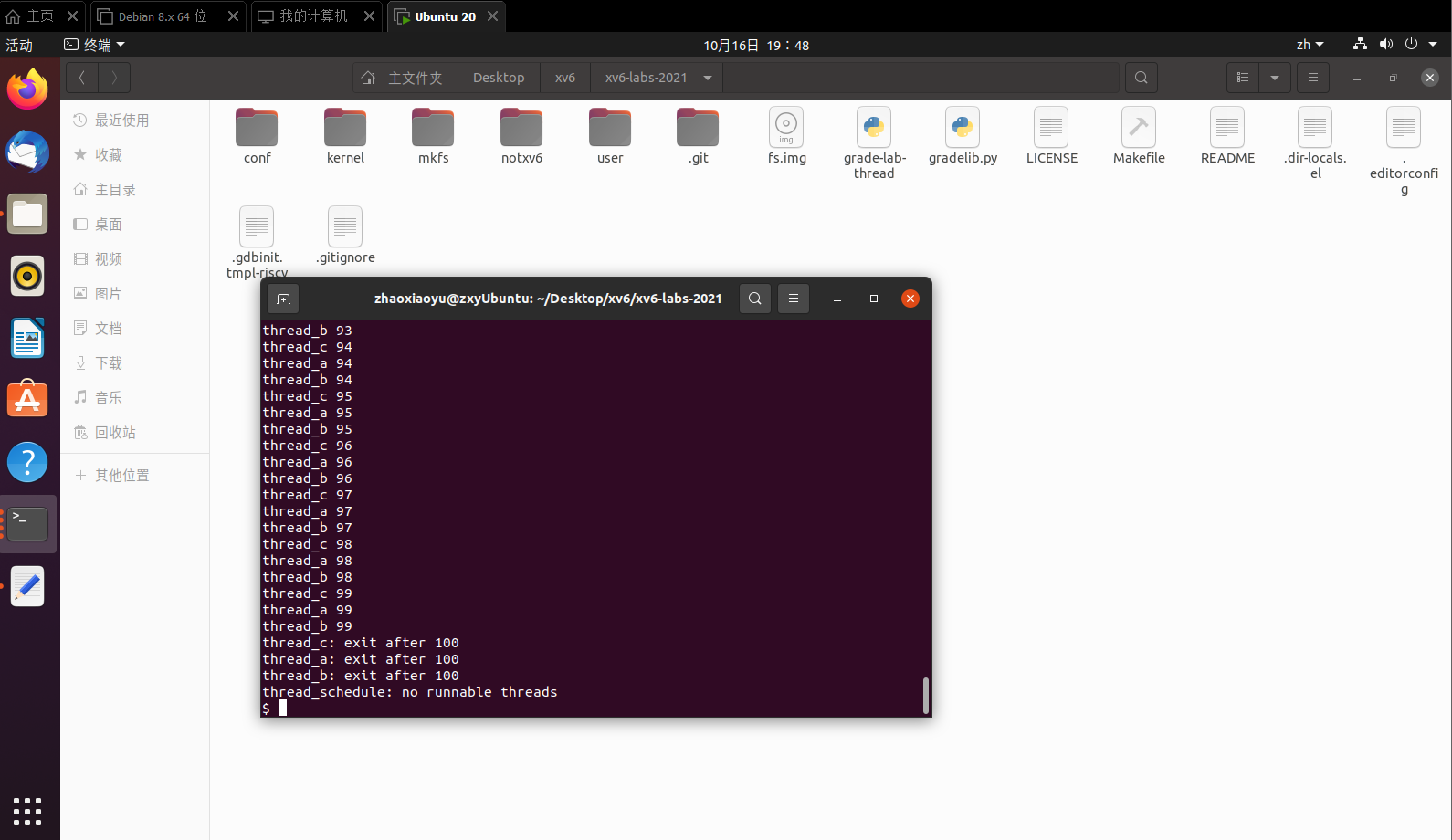


需要修改函数定义 方便以后thread\_switch()调用

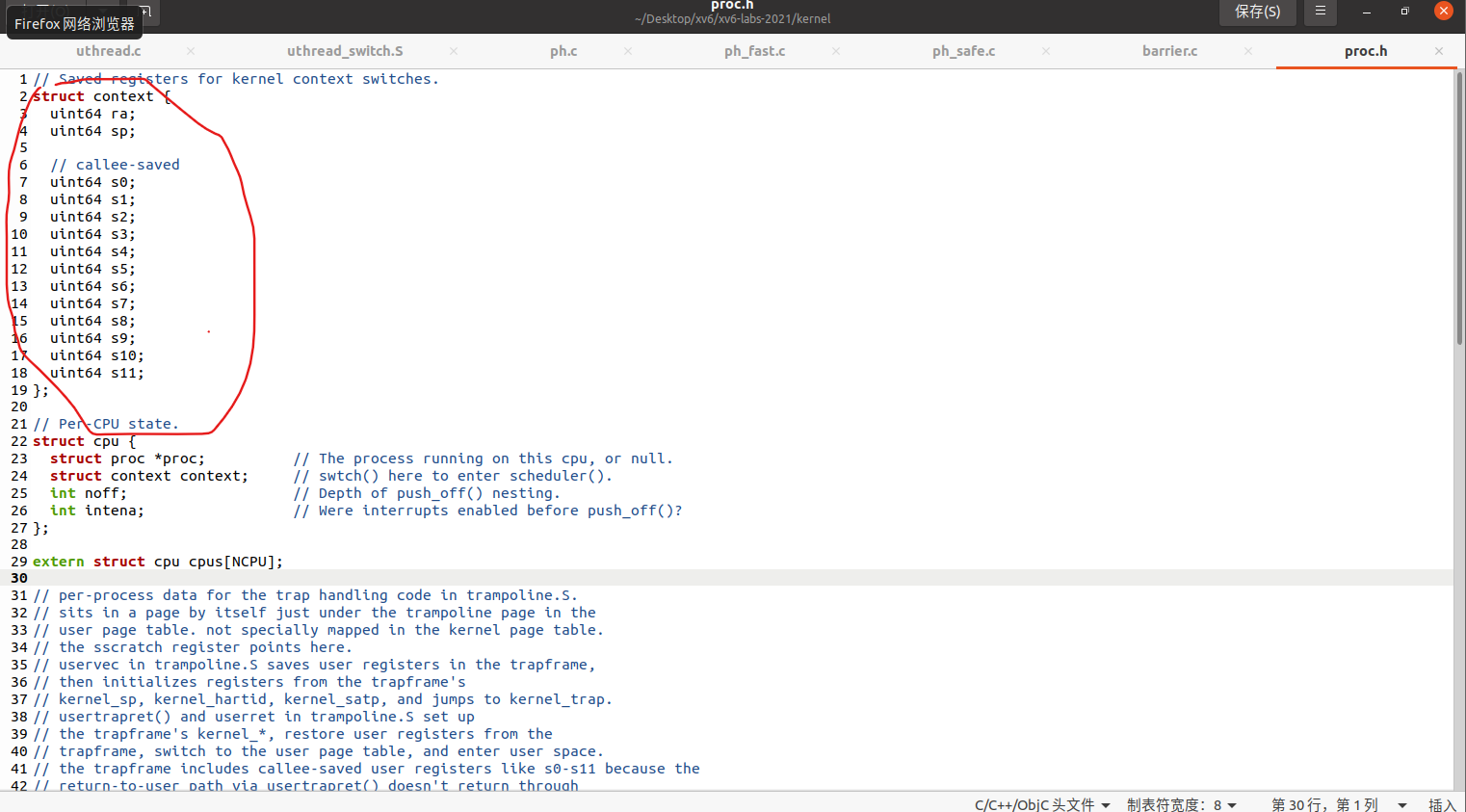
**3.ScreebShot**

运行结果如下所示





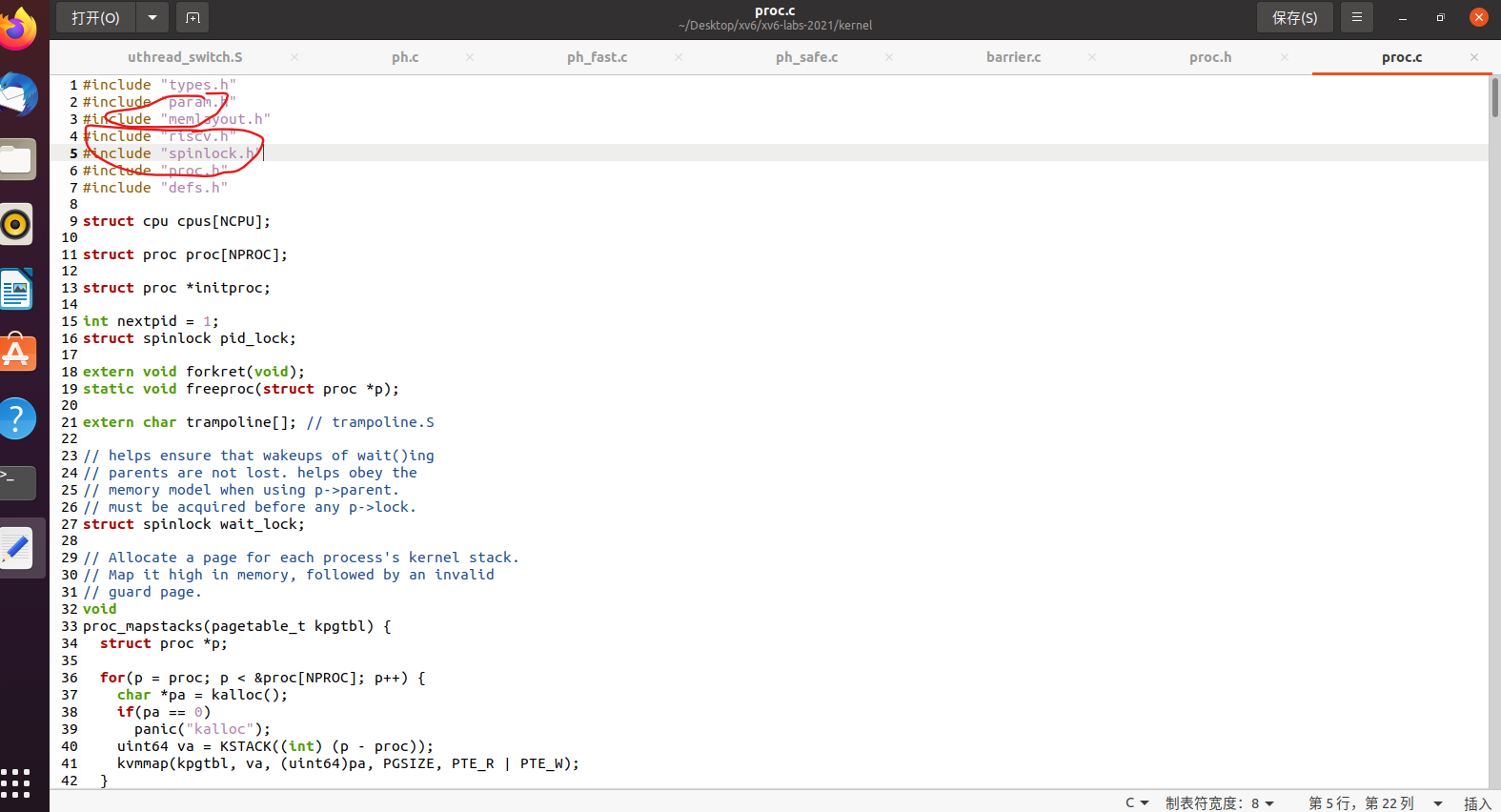
**4.Difficulties and Solutions**



由于uthread要使用struct context

他就必须include “kernel/proc.h”

而proc.h中又包含了其他头文件中定义的东西



此时添加上其他头文件

但是这个头文件的顺序非常重要 否则老是报错

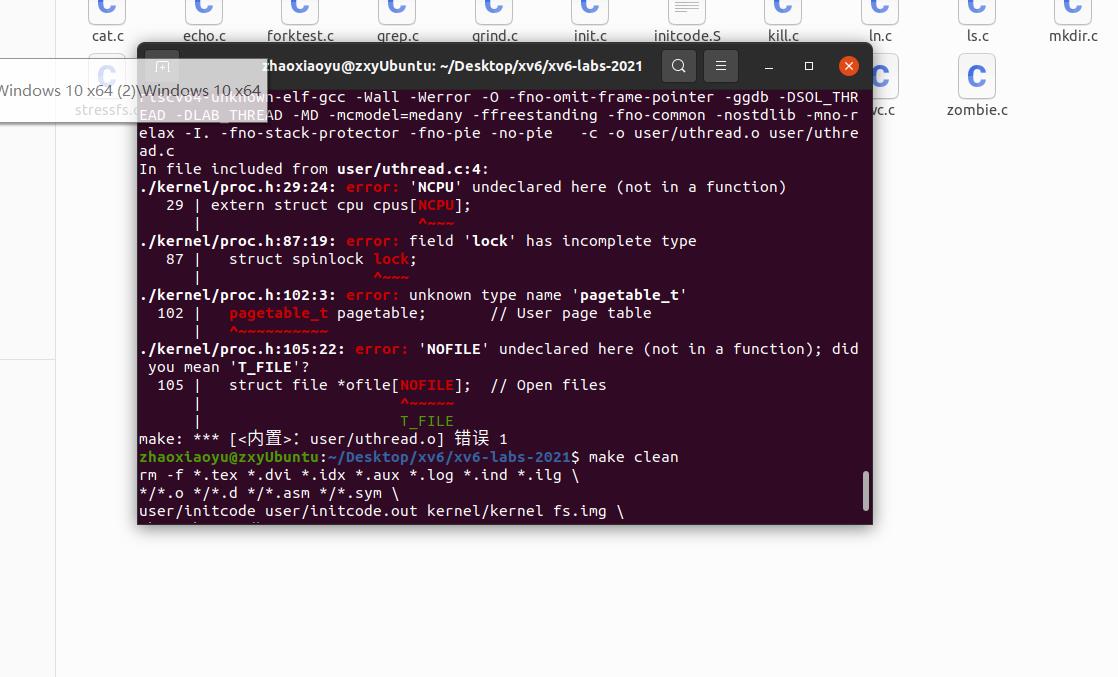
应该把proc.h放在最后

否则它就会说几个变量undefined

这一步是在make qemu过程中出错的

可以根据报错信息排查 找到定义变量的头文件引用

按照define-use原则（先定义再使用）（上一学期的逆向工程原理有所涉及



之后运行正确后即可排查

代码还是比较容易写的

万事开头难 环境往往是最难配置的

中间Ubuntu突然不能上网了 鼓捣了yihuier

参考教程

[Ubuntu突然无法上网解决\_啡咖啡咖啡咖啡咖的博客-CSDN博客](https://blog.csdn.net/qq_33680024/article/details/83239890)

中间git 环境配置

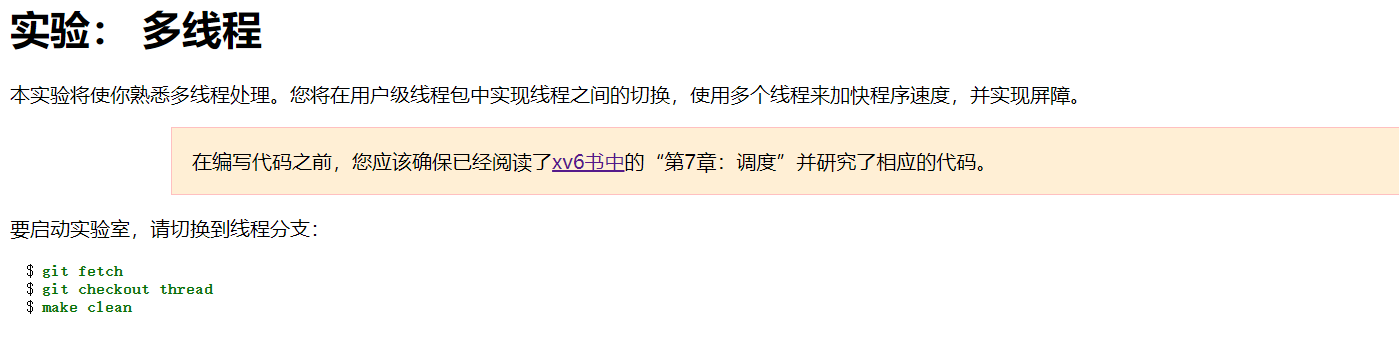
[MIT 6.S081环境搭建 - Ricardo609 - 博客园 (cnblogs.com)](https://www.cnblogs.com/Ricardo609/p/14313122.html)

然后环境配置好 git搭建完毕

Git clone的时候发现没东西

还要切换分支

全是学的不深入以及太不仔细导致



## 贰2.Question：使用线程（[中等](https://pdos.csail.mit.edu/6.S081/2022/labs/guidance.html)）)

在本作业中，您将探索使用哈希表使用线程和锁的并行编程。您应该在具有多个内核的真正 Linux 或 MacOS 计算机（不是 xv6，不是 qemu）上执行此任务。最新的笔记本电脑都有多核处理器。

此分配使用 UNIX 父线程线程库。你可以从手册页中找到有关它的信息，包括人螺纹，你可以在网上查看，例如[这里](https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_mutex_lock.html)，[这里](https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_mutex_init.html)和[这里](https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_create.html)。

文件 notxv6/ph.c 包含一个简单的哈希表，如果从单个线程使用，则该哈希表是正确的，但在从多个线程使用时不正确。在主 xv6 目录（可能是 ~/xv6 实验室-2021）中，键入以下内容：

$ make ph

$ ./ph 1

请注意，要构建ph，生成文件使用操作系统的gcc，而不是6.S081工具。ph 的参数指定对哈希表执行 put 和 get 操作的线程数。运行一会儿后，ph 1 将产生类似于以下内容的输出：

100000 puts, 3.991 seconds, 25056 puts/second

0: 0 keys missing

100000 gets, 3.981 seconds, 25118 gets/second

您看到的数字可能与此示例输出相差两倍或更多，具体取决于计算机的速度、是否具有多个内核以及是否忙于执行其他操作。

ph 运行两个基准测试。首先，它通过调用 put（） 将大量键添加到哈希表中，并以每秒的 put 为单位打印实现的速率。它使用 get（） 从哈希表中提取键。它打印由于看跌期权而应该在哈希表中但丢失的数字键（在本例中为零），并打印它每秒实现的获取次数。

您可以通过为ph提供大于1的参数来告诉ph同时使用来自多个线程的哈希表。试试ph 2：

$ ./ph 2

100000 puts, 1.885 seconds, 53044 puts/second

1: 16579 keys missing

0: 16579 keys missing

200000 gets, 4.322 seconds, 46274 gets/second

此 ph 2 输出的第一行指示当两个线程同时向哈希表添加条目时，它们的总速率达到每秒 53，044 次插入。这大约是运行ph 1的单线程速率的两倍。这是一个极好的“并行加速”，大约是2倍，是人们可能希望的（即两倍的内核，每单位时间产生两倍的工作量）。

但是，缺少 16579 键的两行表示不存在本应存在于哈希表中的大量键。也就是说，看跌期权应该将这些键添加到哈希表中，但是出了点问题。看看 notxv6/ph.c，特别是看 （） 和插入 （）。

为什么缺少 2 个线程的键，而不是 1 个线程的键？标识具有 2 个线程的事件序列，这些线程可能导致密钥丢失。在答案线程中提交您的序列并附上简短的解释.txt

要避免这种事件序列，请在 put 和 get in notxv6/ph.c 中插入锁定和解锁语句，以便两个线程中缺少的密钥数始终为 0。相关的 p 线程调用包括：

pthread\_mutex\_t lock; // declare a lock

pthread\_mutex\_init(&lock, NULL); // initialize the lock

pthread\_mutex\_lock(&lock); // acquire lock

pthread\_mutex\_unlock(&lock); // release lock

当 make grade 表示您的代码通过了ph\_safe测试时，您就完成了，这需要两个线程的零缺失键。此时，无法通过ph\_fast测试是可以的。

别忘了打电话给pthread\_mutex\_init（）。首先使用 1 个线程测试代码，然后使用 2 个线程对其进行测试。它是否正确（即您是否消除了丢失的密钥？）？与单线程版本相比，双线程版本是否实现了并行加速（即每单位时间的总工作量更多）？

在某些情况下，并发 put（） 在哈希表中读取或写入的内存中没有重叠，因此不需要锁来相互保护。您能否更改 ph.c 以利用这种情况来获得某些看跌期权的并行加速？提示：每个哈希桶的锁怎么样？

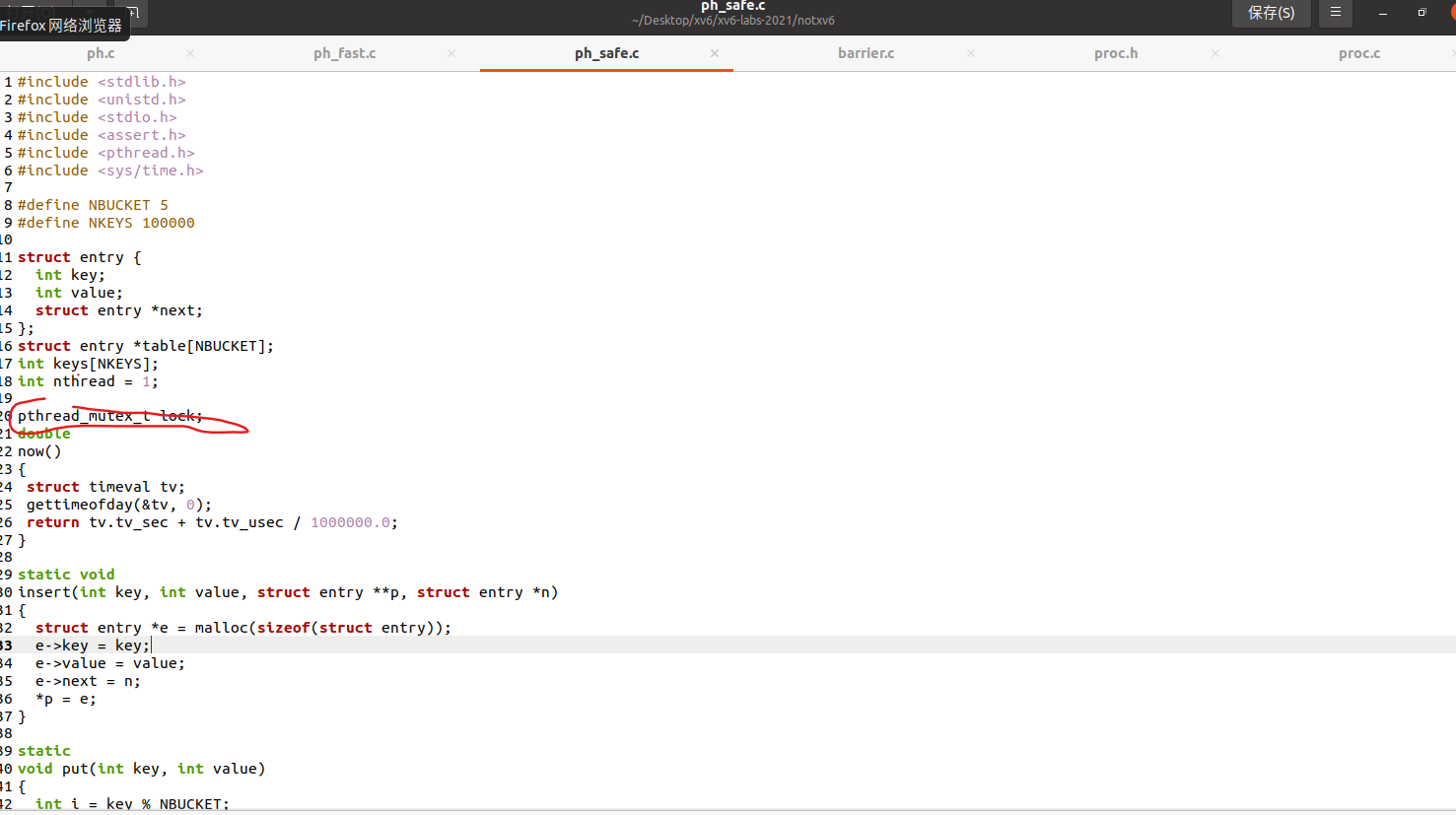
修改代码，以便某些 put 操作并行运行，同时保持正确性。当 make grade 表示您的代码通过了ph\_safe和ph\_fast测试时，您就完成了。ph\_fast测试要求两个线程的吞吐量至少是一个线程的 1.25 倍/秒。

**2.Code**

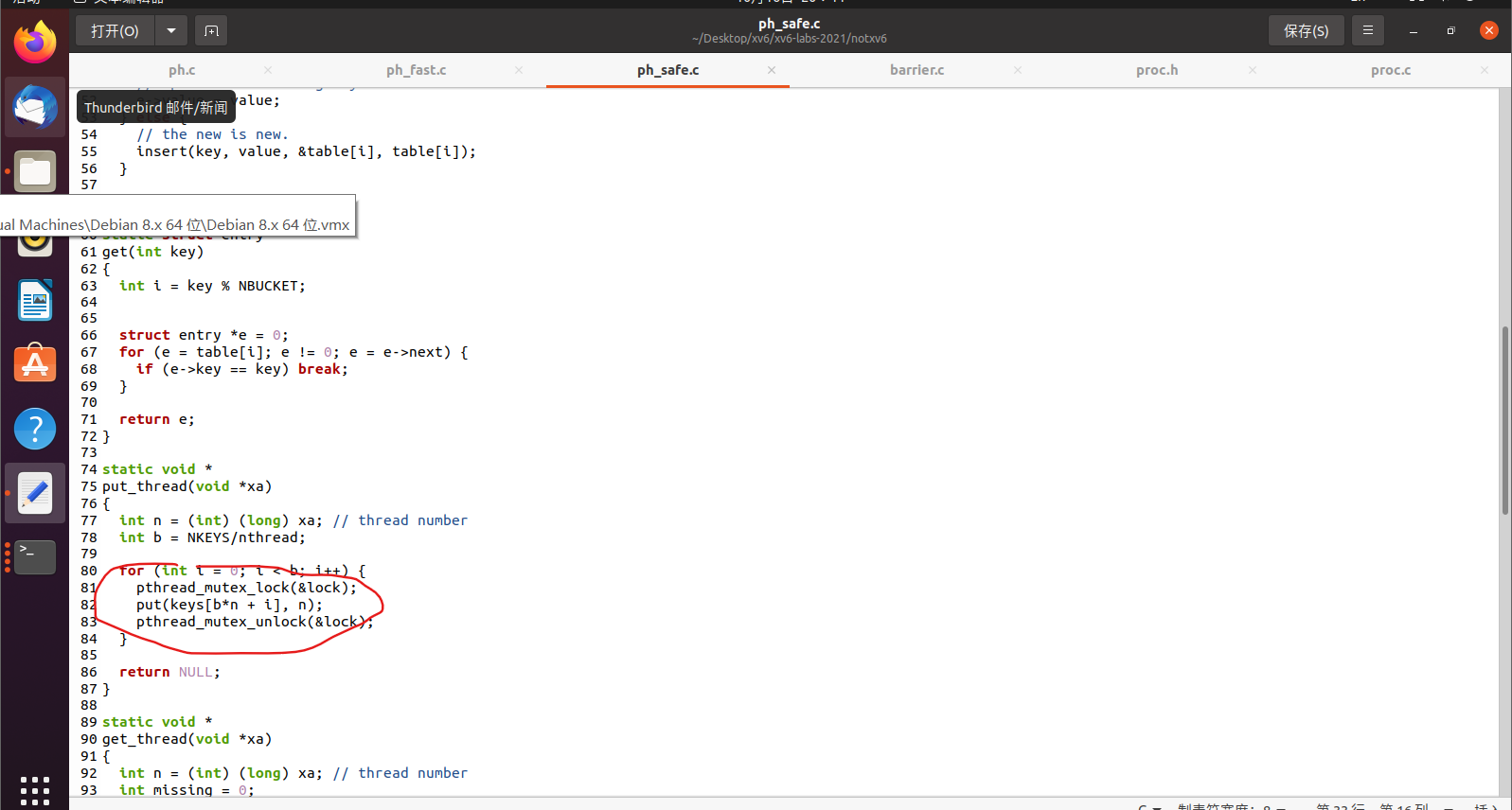
其中哈希表的线程安全问题是: 多个线程同时调用 put() 对同一个 bucket 进行数据插入时, 可能会使得先插入的 entry 丢失.

分别按照实验要求实现了两种 ph\_safe 与ph\_fast

ph\_safe 全局只有一把锁



在put\_thread中调用put的时候让其为原子操作（太过小心

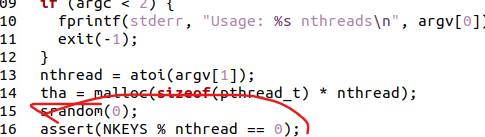


ph\_fast

这个哈希表的构建过程如下所述

首先线程的个数必须是NKEYS（100000）的因子

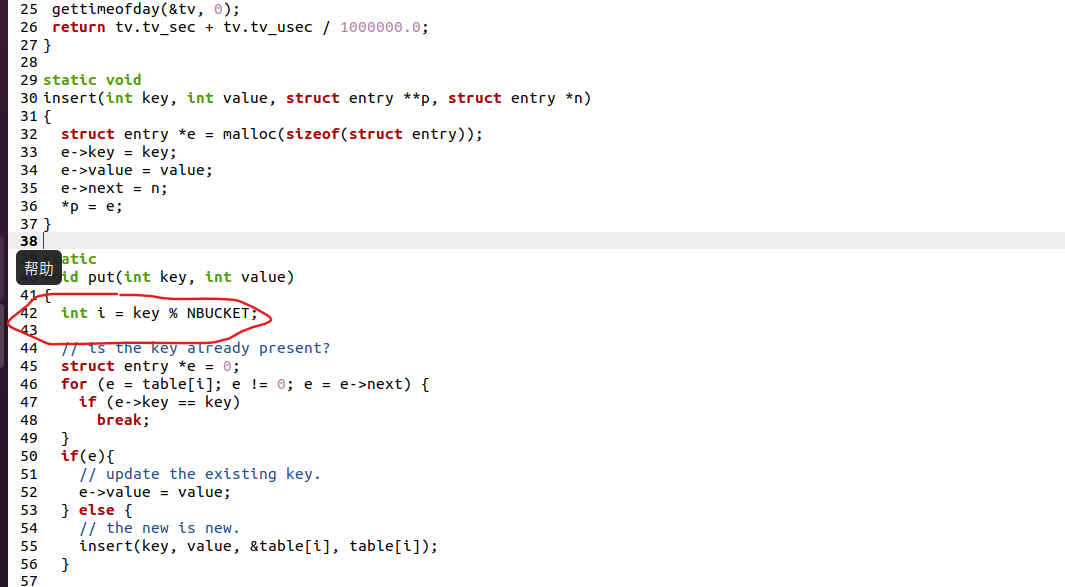
有指令声明这一限制



然后将100000个内存地址调用random后也即生成了100000个随机数

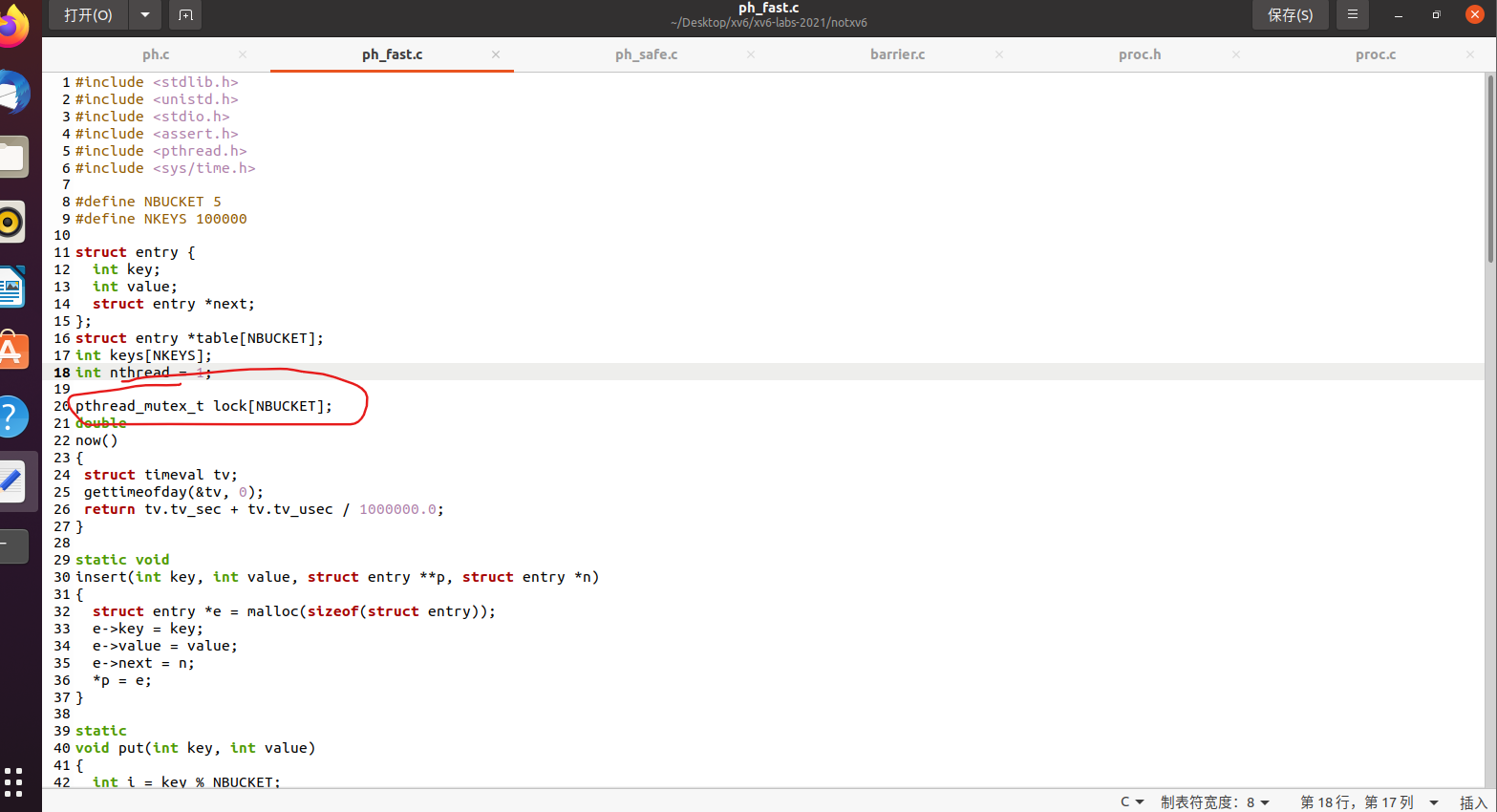
将这100000个随机数的插入任务平均分给所有线程

但是插入的时候



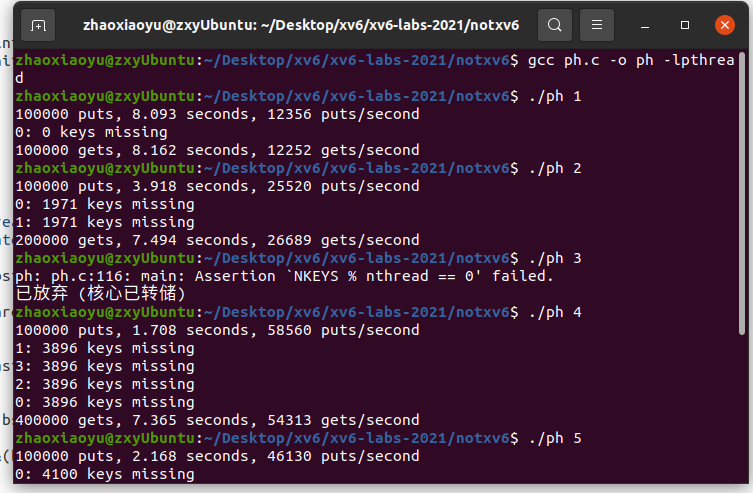
因此对于每一个Bucket上一个锁即可

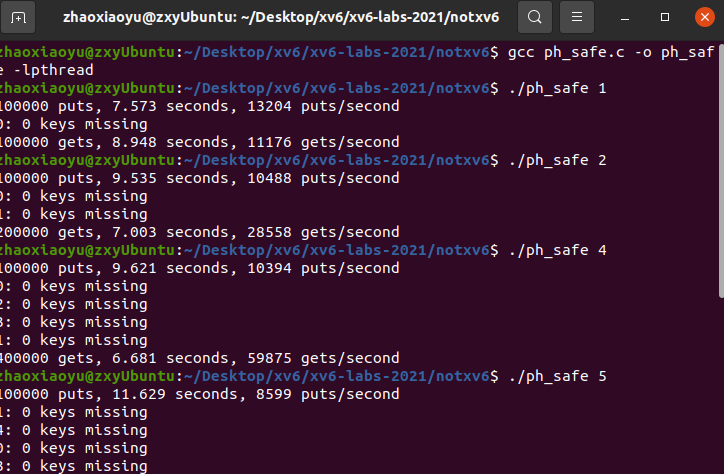
代码如下

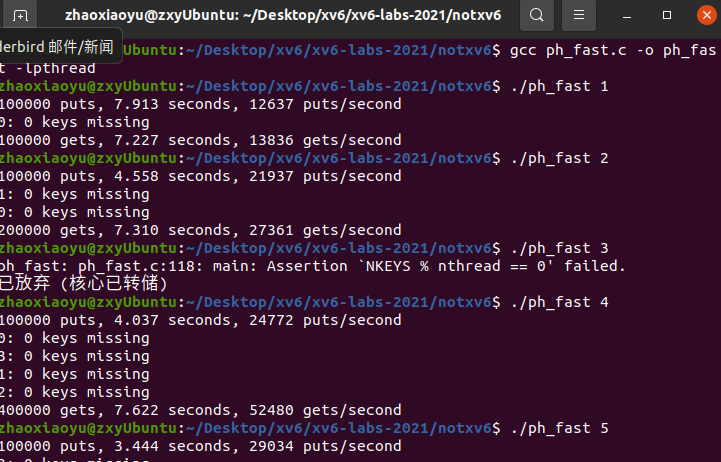




3.Screenshot







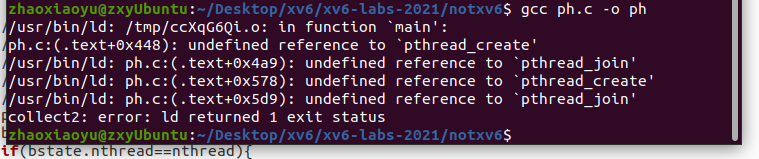
**4.Difficulties and Solutions**

主要是对于为什么多个线程对哈希表进行插入时 会有一定的疑惑

但是仔细想想过程还是有解决办法的

结合代码具体来讲, 假设有 A 和 B 两个线程同时 put(), 由于该哈希表的桶数 NBUCKET为 5, 哈希函数为 key%NBUCKET, 而插入的 key 为 keys[b\*n+i], 而 b=NKEYS/nthread=100000/2=50000, 而 b%NBUCKET==0, 因此对于 A 和 B 两个线程, 在 i 相同时实际上会在同一个 bucket 插入数据. 假设 A 和 B 都运行到 put() 函数的 insert() 处, 还未进入该函数内部, 这就会导致两个线程 insert() 的后两个参数是相同的, 都是当前 bucket 的链表头, 如若线程 A 调用 insert() 插入完 entry 后, 切换到线程 B 再调用 insert() 插入 entry, 则会导致线程 A 刚刚插入的 entry 丢失.

以及用本身的OS（也即ubuntu自带的gcc进行编译时会出现的问题



搜索相关问题后



加上 -lpthread 即可解决

性能分析

## 叁1.Question屏障（[中等](https://pdos.csail.mit.edu/6.S081/2022/labs/guidance.html)）)

在此分配中，您将实现一个[屏障](http://en.wikipedia.org/wiki/Barrier_(computer_science))：应用程序中的一个点，在该点上，所有参与线程都必须等待所有其他参与线程也到达该点。您将使用 pthread 条件变量，这是一种类似于 xv6 的睡眠和唤醒的序列协调技术。

您应该在真实计算机上执行此作业（不是 xv6，不是 qemu）。

文件 notxv6/屏障.c 包含一个损坏的屏障。

$ make barrier

$ ./barrier 2

barrier: notxv6/barrier.c:42: thread: Assertion `i == t' failed.

2 指定在屏障上同步的线程数（屏障中的 n 线程）。每个线程执行一个循环。在每次循环迭代中，线程调用 barrier（），然后休眠一个随机数微秒。断言触发，因为一个线程在另一个线程到达屏障之前离开屏障。期望的行为是每个线程都阻塞在屏障（）中，直到它们的所有 n 线程都调用了屏障（）。

您的目标是实现所需的屏障行为。除了您在 ph 赋值中看到的锁基元之外，您还需要以下新的 pthread 基元;查看[此处](https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_cond_wait.html)和[此处](https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_cond_broadcast.html)了解详细信息。

pthread\_cond\_wait(&cond, &mutex); // go to sleep on cond, releasing lock mutex, acquiring upon wake up

pthread\_cond\_broadcast(&cond); // wake up every thread sleeping on cond

确保您的解决方案通过等级的阻隔测试。

pthread\_cond\_wait在调用时释放互斥体，并在返回之前重新获取互斥体。

我们给了你barrier\_init（）。你的工作是实现屏障（），这样就不会发生恐慌。我们为您定义了结构屏障;其字段供您使用。

有两个问题使您的任务复杂化：

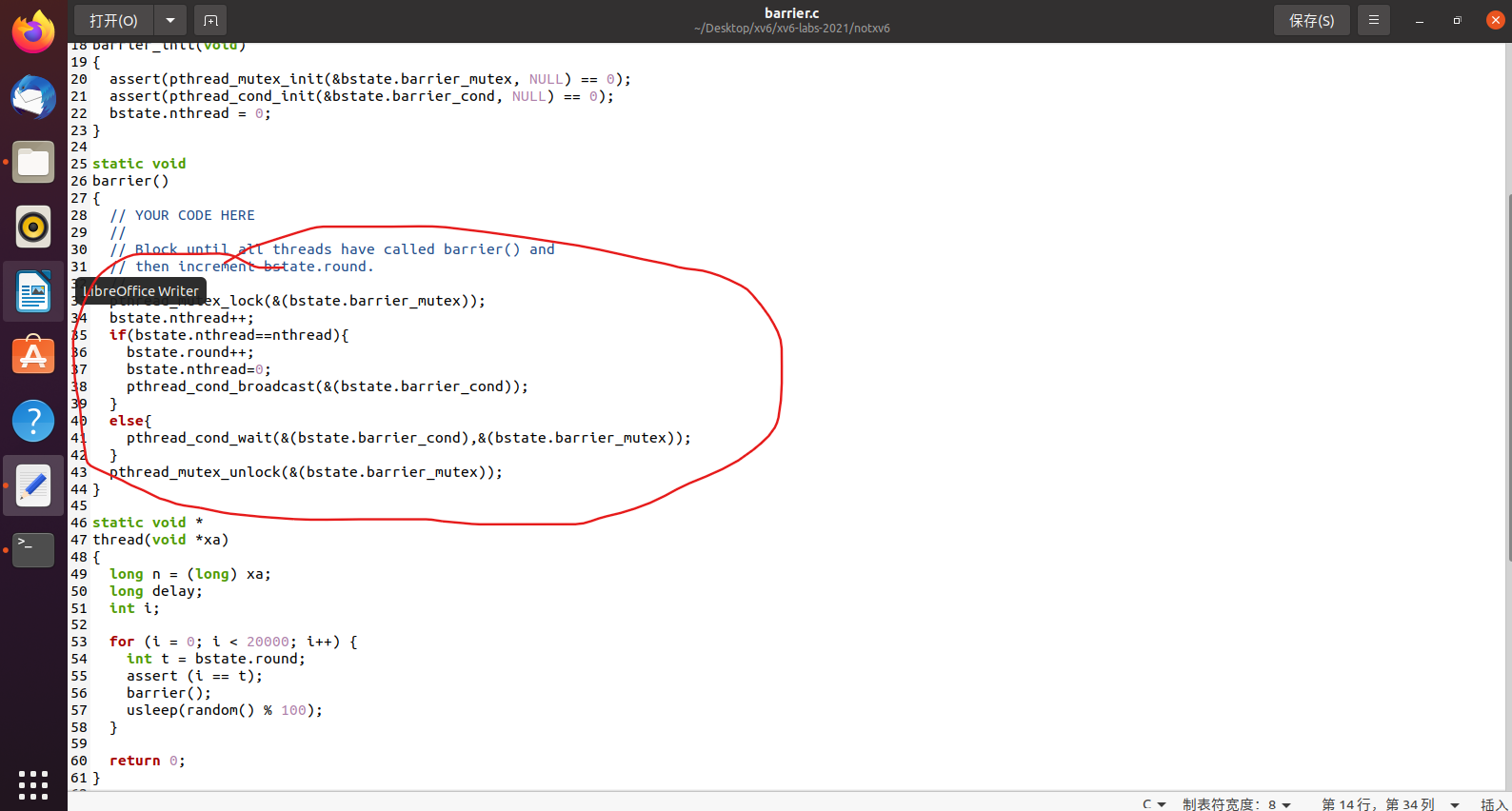
* 你必须处理一连串的障碍呼叫，每个呼叫我们都会称为一轮。b状态.回合记录当前回合。每次所有线程都达到障碍时，您都应该递增 bstate.round。
* 您必须处理这样一种情况，即一个线程在其他线程退出障碍之前绕着环路运行。特别是，您正在从一个轮次到下一个轮次重用 bstate.nthread 变量。确保离开障碍并在循环中比赛的线程不会增加bstate.nthread，而前一轮仍在使用它。

使用一个、两个和两个以上的线程测试代码。

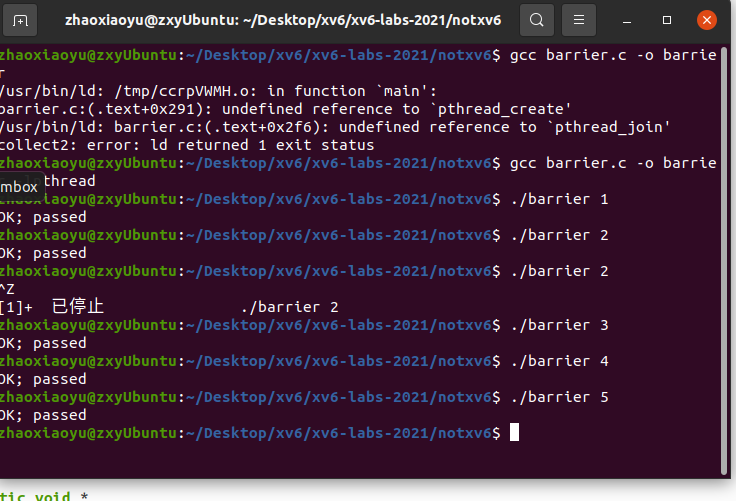
**2.Codes**

仅展示已经修改的地方

后面会make clean后上传到git仓库

**3.Screenshot**

结果如下：

**4.Difficulties and Solutions**

主要是逻辑理解

当某个线程执行到barrier

会对全局的到达该地方的线程计数器+1操作

接着检查计数器是否到了所有线程数量

如果到了 那么重置计数器 并且完成的轮数+1 并且发信号 我是最后一个 大家都醒来吧

如果没到 那么就wait最后一个到达的发的信号量 并且释放锁 即为条件变量操作