**北 京 林 业 大 学**

**20 学年— 21 学年第2 学期 操作系统 实习报告书**

专 业： 计算机科学与技术\_ 班 级： 计算机203

姓 名： 南希诺 学 号： 201002727

实习地点： 家 辅导教师： 田萱

实习题目： PintOS Project1

实习环境： Ubuntu

# 一、课程设计目标：

**PintOS的Project1包含3个任务。分别是：**

**（1）重写timer\_sleep()，避免busy waiting**

* **thread\_mlfqs为false时使用优先级调度，为true时使用多级反馈队列调度**

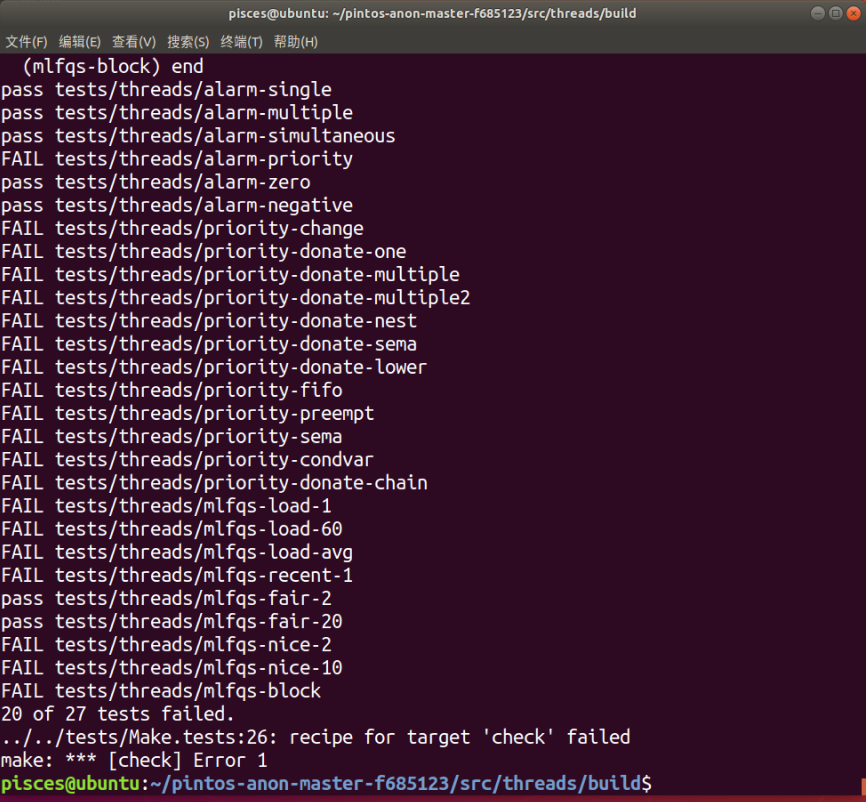
**（2）实现优先级调度**

* **schedule应选择高优先级线程**
* **高优先级线程就绪时，应抢占**
* **锁、信号量、条件变量应优先唤醒高优先级线程**
* **当前线程降低自身优先级后，应yield （即当前线程让出CPU，调度器重新调度）**
* **需考虑优先级反转，实现优先级捐赠（仅对于锁）**
* **线程不会修改其它线程的优先级**

**（3）实现多级反馈队列**

* **此时优先级仅由公式决定，不受其它因素影响**
* **实现定点数运算，实现优先级的更新**

配置好环境后，在/threads/build下进行make并make check将会出现如下结果：



# 二、线程管理的实现

以下内容参考网上资料，我能力做不动这个课设，但是我真的尽力了，这些步骤我都自己在程序里写了一遍。

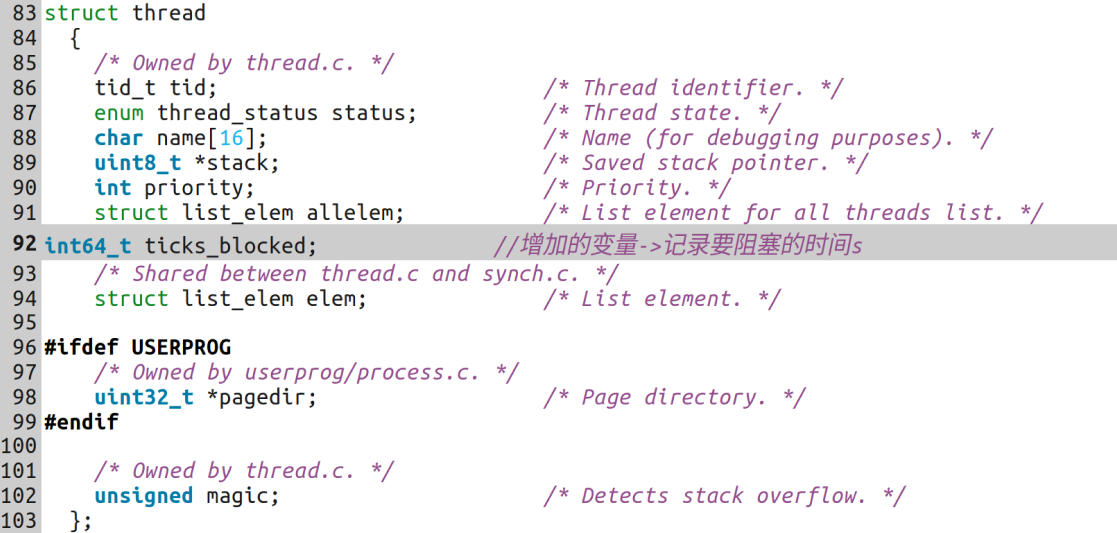
一、第一部分——timer\_sleep()函数的重新实现

1、实现思想

整体算法思想：在原来的timer\_sleep（）中，使用while循环来检测线程是否开始运行。Waitinglist用于存储时间未过期的线程。Add thread\_wait（）用于将尚未到达时间点的线程添加到waitinglist。在schedule（）中，在调用schedule（）\_List、waiting\_完成列表中的睡眠期间遍历等待，并将具有所需时间的进程放入ready\_List call thread\_wakeup（）中，以唤醒到达时间点的线程。向线程结构中添加一个变量以记录线程的睡眠时间。通过使用pintos时钟中断，每个勾号将执行一次，因此记录线程睡眠时间的变量将在每次检测期间自动减少1。当变量为0时，意味着可以唤醒线程。

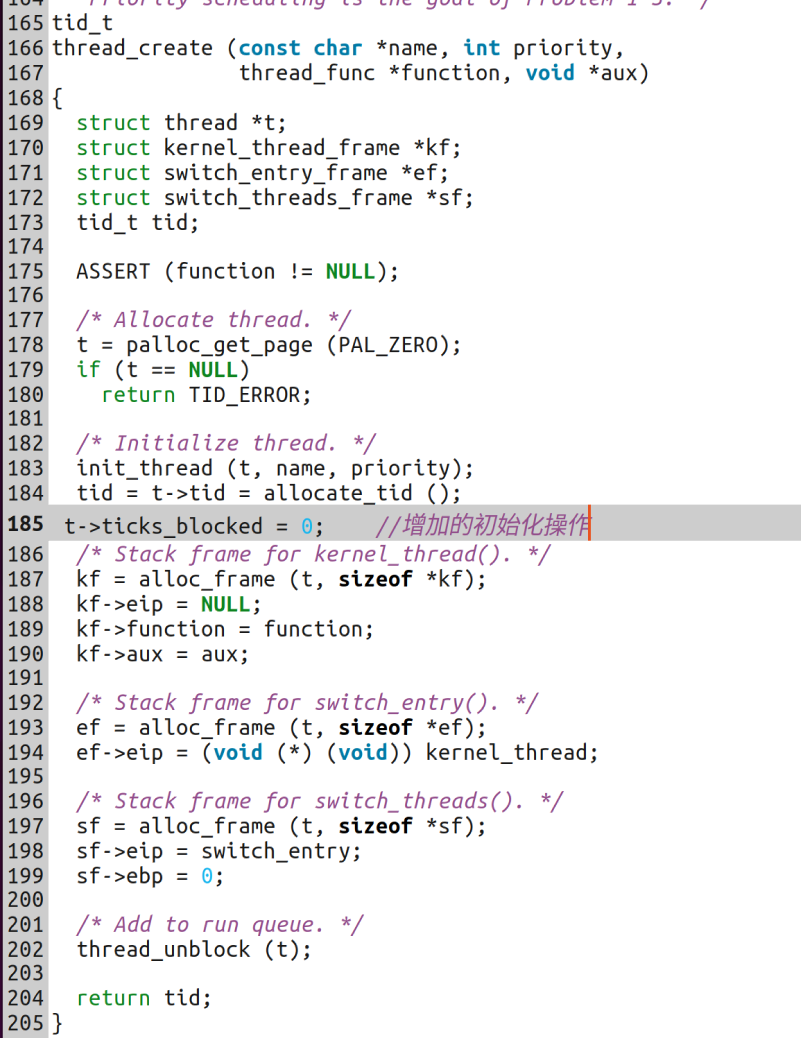
2、数据结构

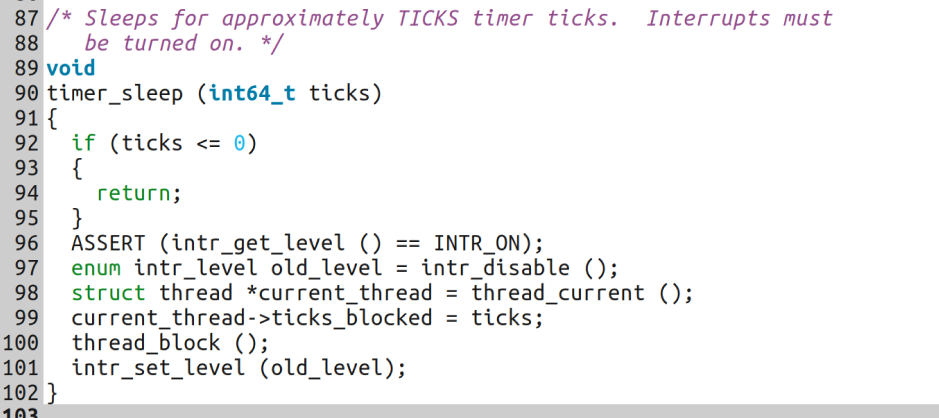
改写线程结构体，在结构体中增加记录线程睡眠时间的变量ticks\_blocked

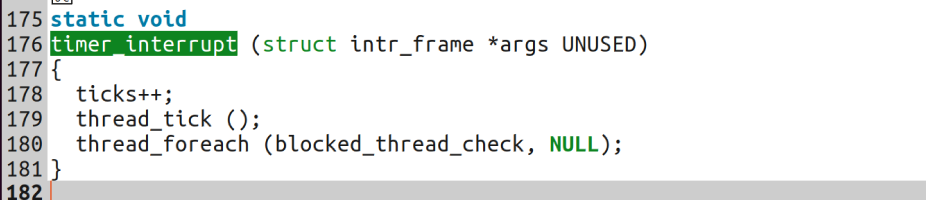


增加记录线程睡眠时间的变量的赋初值操作t->ticks\_blocked = 0;

调用Pintos的线程阻塞函数

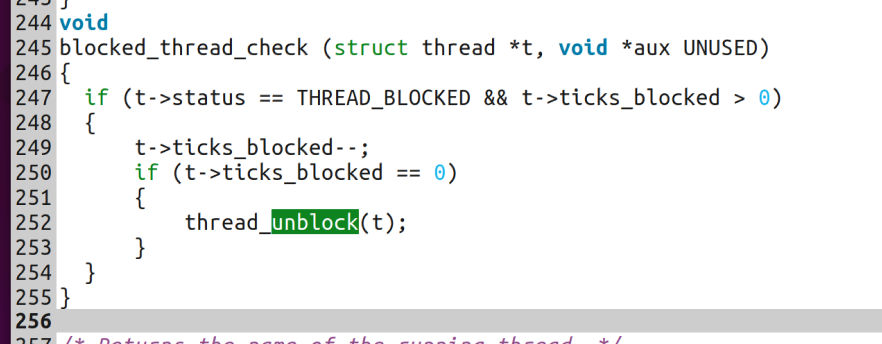


实现blocked\_thread\_check()函数





实现逻辑：如果这个线程处于阻塞态且阻塞时间还未结束，则减小其阻塞时间并判断当其不用再被阻塞，便调用Pintos函数thread\_unblock()将线程放入就绪队列并更改为就绪态。

至

二、第二部分——优先级调度的实现

1、保证插入线程至就绪队列时保持优先级队列

（1）实现思路：

由于Pintos预置函数list\_insert\_ordered()的存在，可以直接使用这个函数实现线程插入时按照优先级完成，因此只需要将涉及直接在末尾插入线程的函数中的语句进行替换即可。整体思路：一个进程执行时，但是产生了更高优先级的线程，那么该线程就会被放在就绪队列readlist中。在这个过程中必须确定该状态是否稳定。每个线程都利用sema\_is\_larger, lock\_is\_larger, thread\_is\_larger进行比较。把优先级最高的放在列首。如果碰到优先级相同的话，就用FIFO思想，谁先到，谁就先执行。

可能有很多线程等待信号，所以将等待的信号根据优先级的高低添加到sema->waiters中。

可能许多线程等待适合的环境，建立semaphore\_elem队列，按优先级大小依次将他们排好放在队列中。

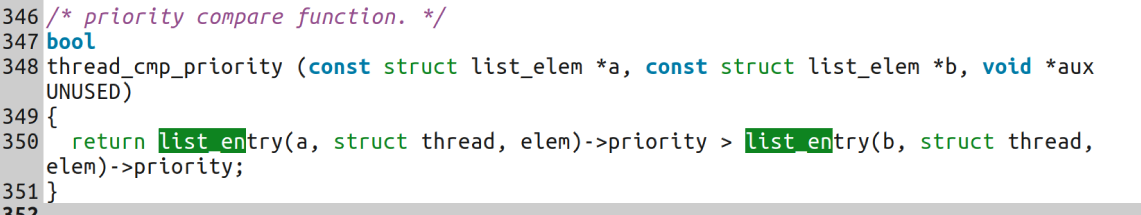
当一个线程请求加锁的时候，测试时候有线程在用其他锁。如果没有其他线程使用此锁，那么该线程就获得此锁。如果不幸有其他线程拥有此锁，但是拥有此锁的优先级低于申请锁的优先级，那么目前有此锁的线程就要放弃锁，将此锁让给优先级高的线程。在牺牲锁之前，记录线程拥有旧锁 的优先级，在牺牲后，记录线程的优先级。

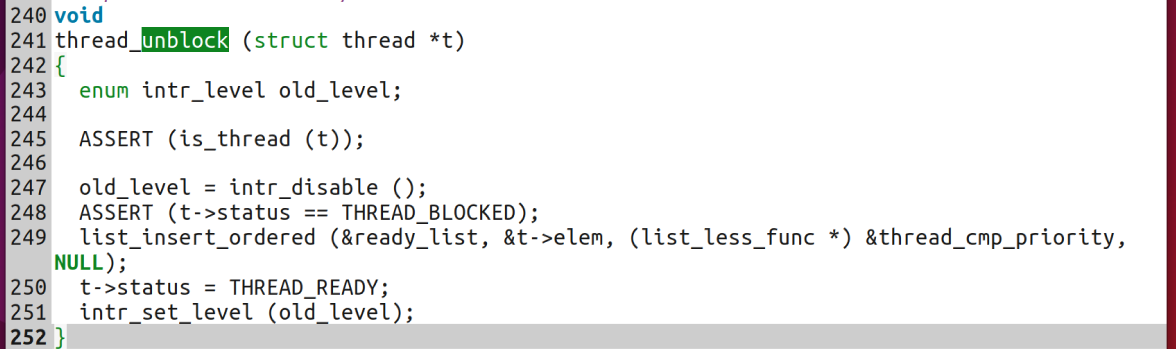
如果有的线程想要释放锁，那么首先要检查是否可以释放锁，如果可以，那么就可以释放，并将此进程放入thread\_current。如果不可以，那么就从thread\_current()->donate\_lock\_list中获得第一个锁，然后返回它的later\_priority到thread\_current中。

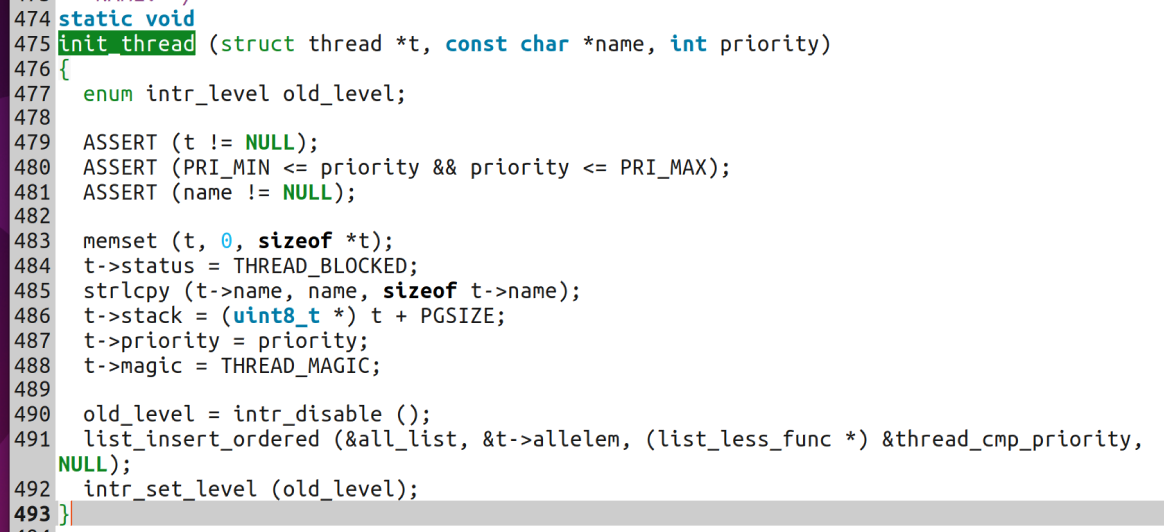
最后lock->is\_donated将为false，指向锁的指针将指向Null。

（2）数据结构：

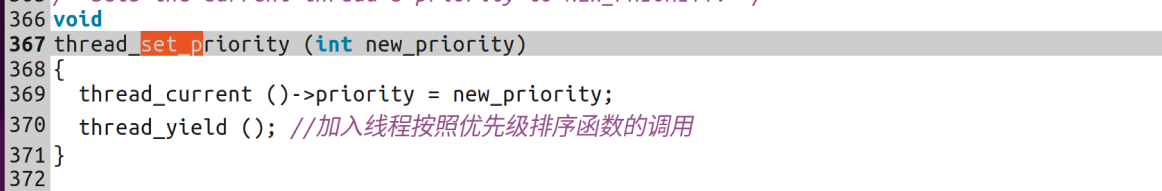
实现一个优先级比较函数thread\_cmp\_priority():



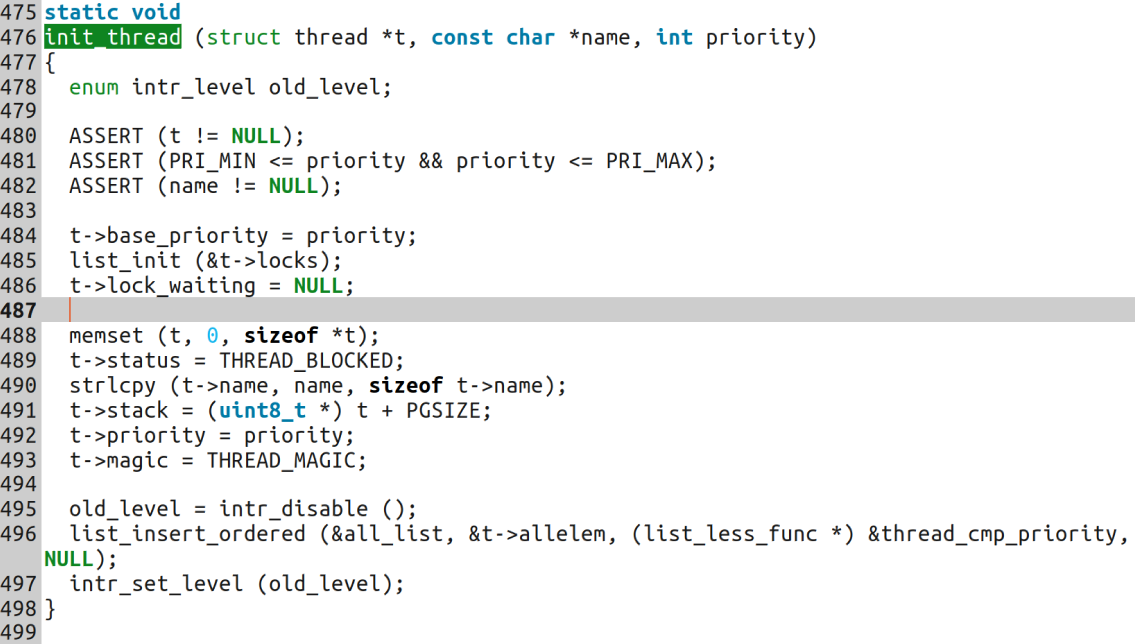


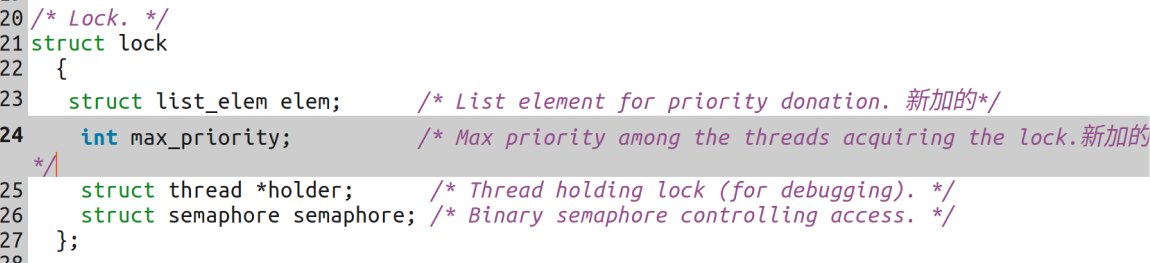


如果创建的线程优先级高于正在运行的线程的优先级，则需要将正在运行的线程加入就绪队列，并且使新建线程准备运行。

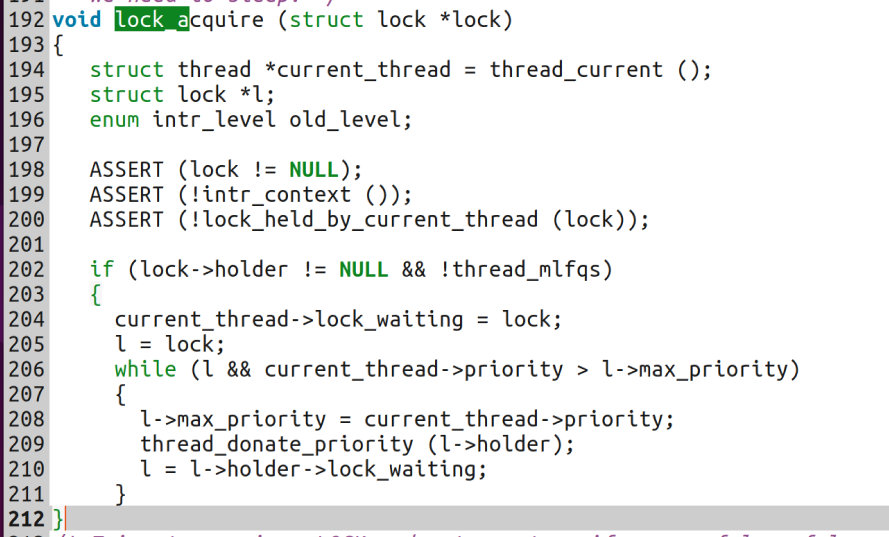


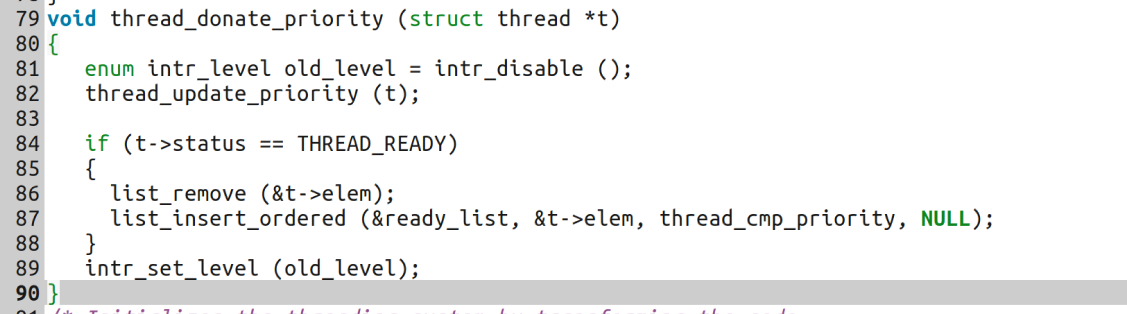


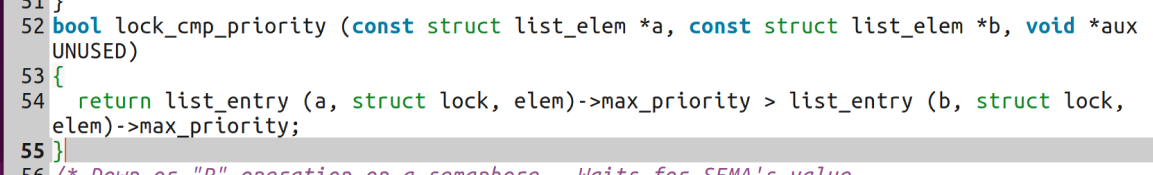


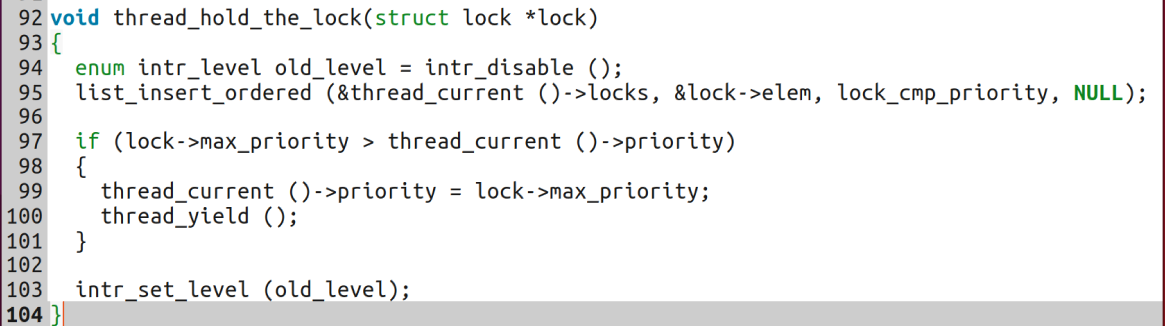


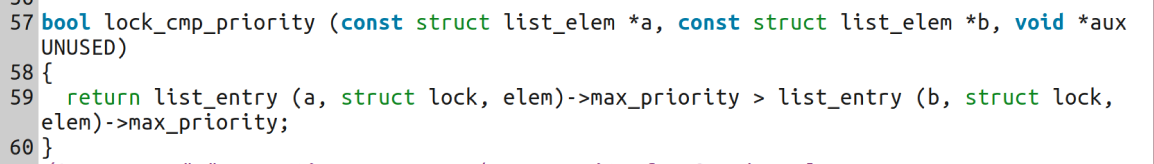
需要在设置优先级函数thread\_set\_priority()中加入thread\_yield ()函数以确保每次修改线程优先级后立刻对就绪队列的线程进行重新排序。

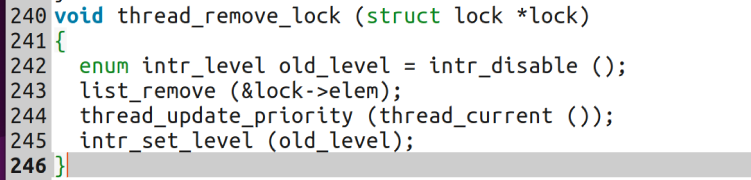


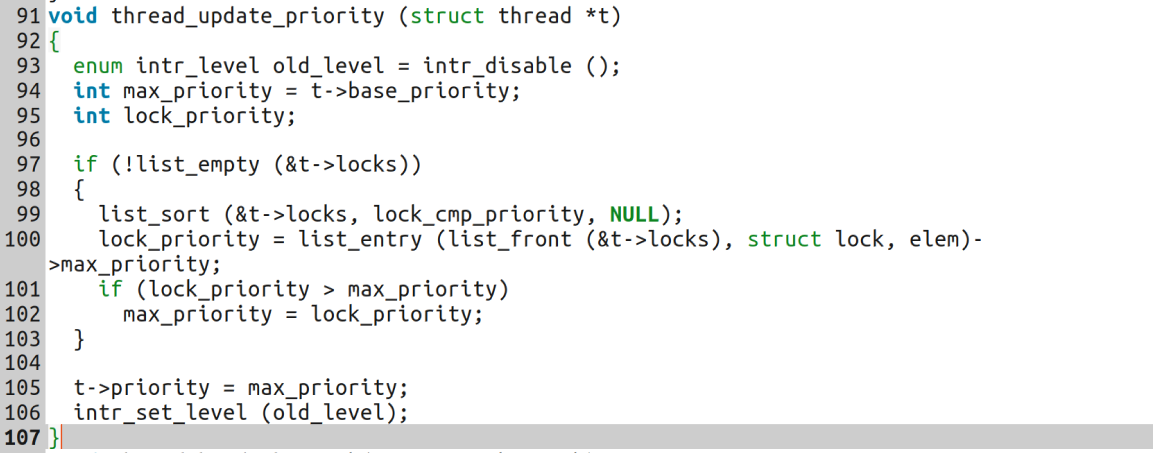


thread\_update\_priority函数更新优先级来实现优先级捐赠。

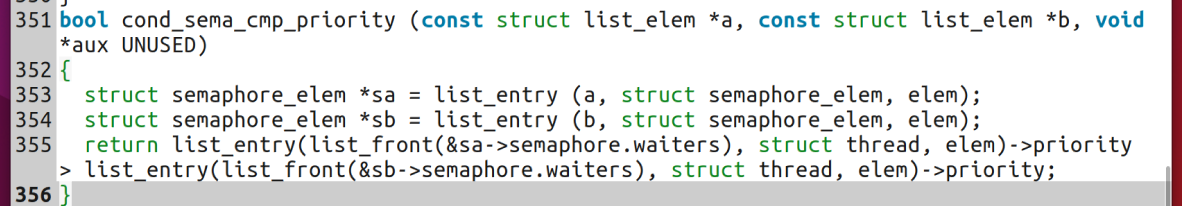
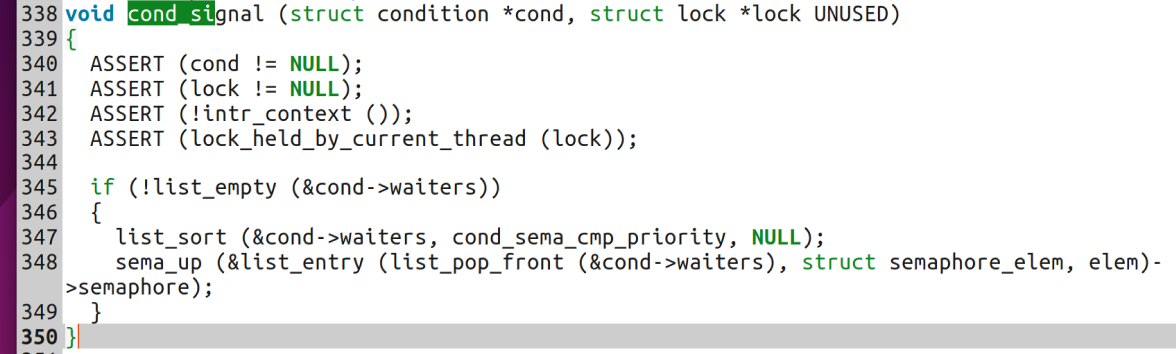


实现lock\_cmp\_priority，以达到对线程优先级的更新和在队列中位置的重新排布: 

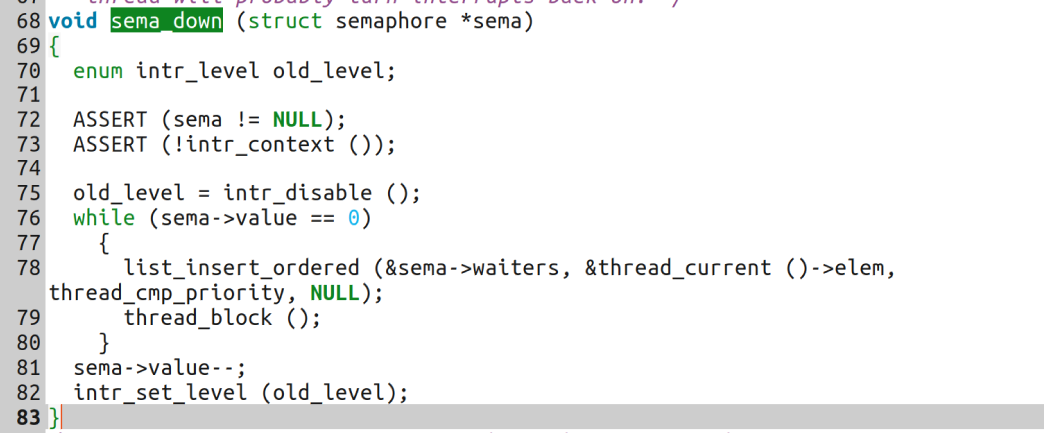




根据锁记录的线程最大优先级更新当前线程的优先级并重新调度：



修改thread\_set\_priority函数，完成对新优先级的变换



三、第三部分——多级反馈调度的实现

1、实现思路

thread初始化的值为0。

每4个tick更新recent\_cpu

priority = PRI\_MAX - (recent\_cpu / 4) - (nice \* 2)

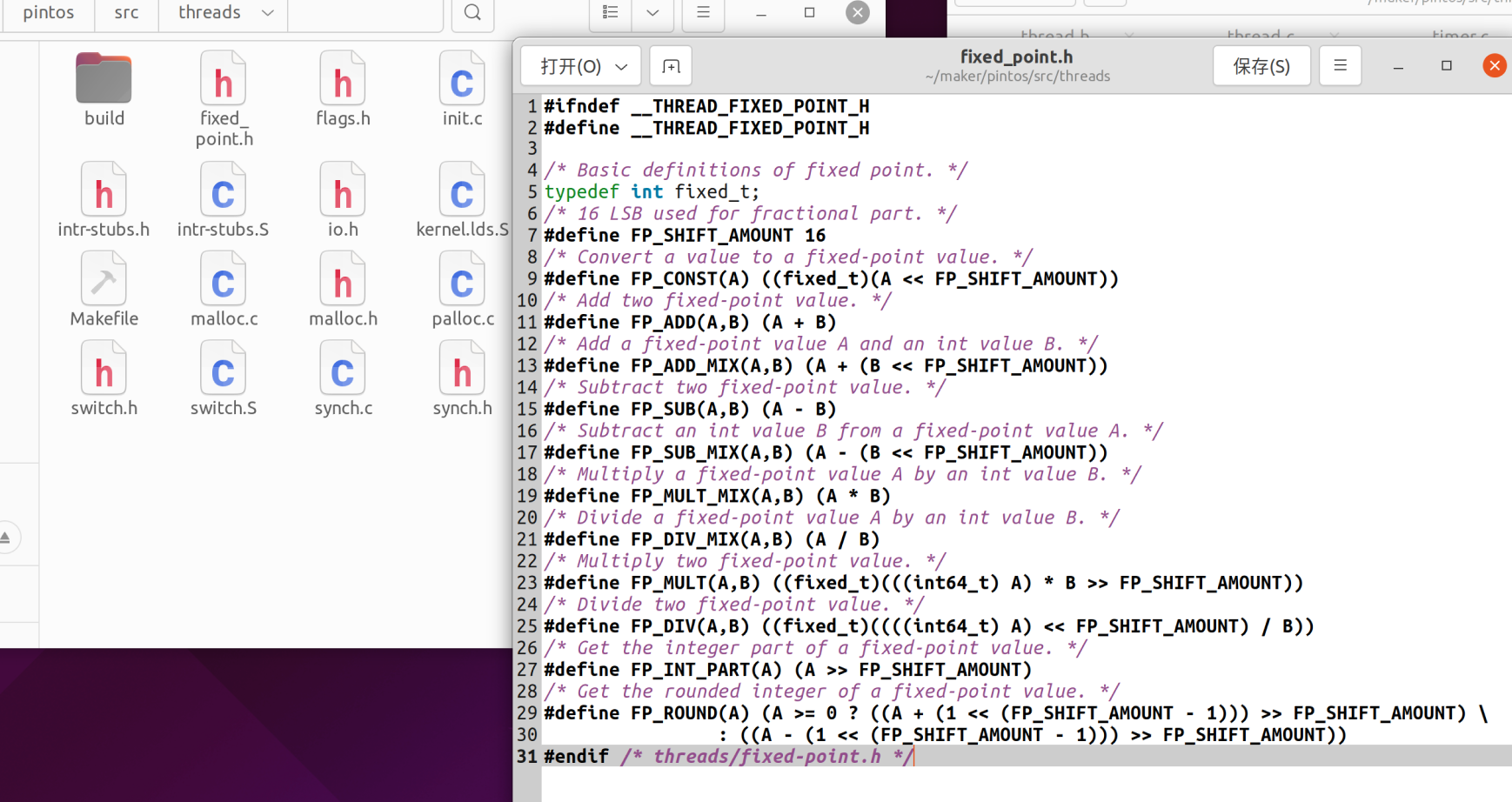
每100个tick更新 load\_avg和priority。

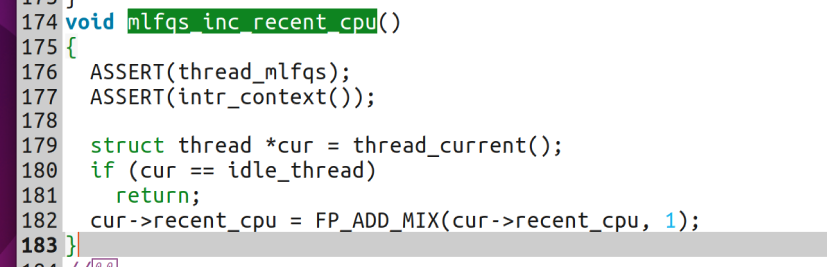
recent\_cpu = (2\*load\_avg)/(2\*load\_avg + 1) \* recent\_cpu + nice

load\_avg = (59/60)\*load\_avg + (1/60)\*ready\_threads

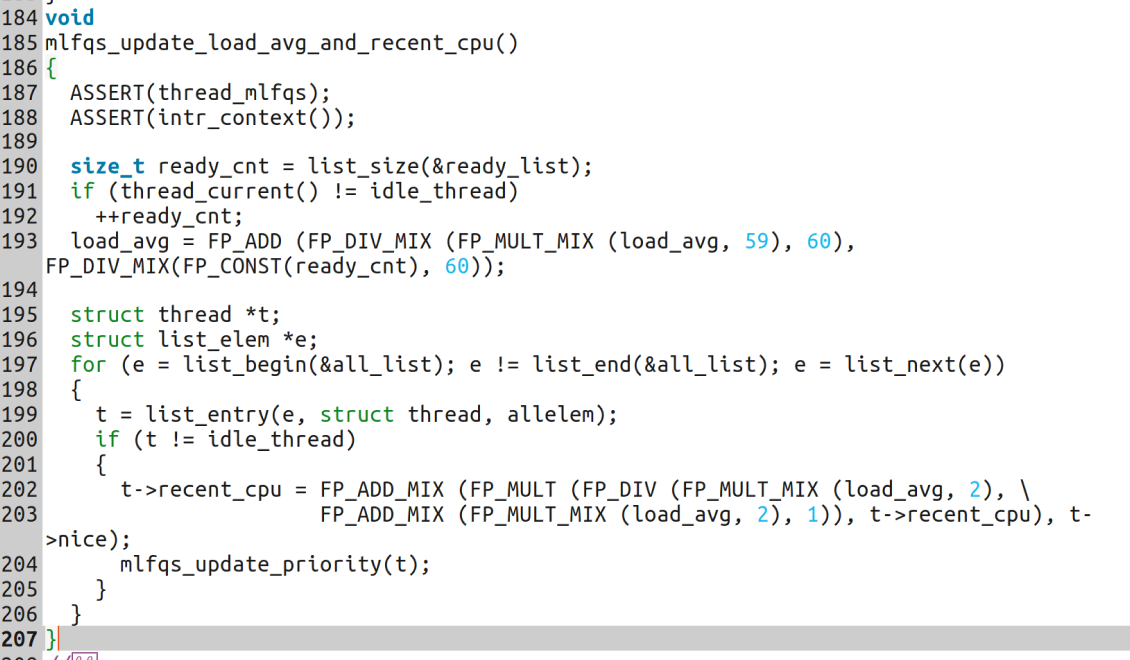
根据pintos PDF后的浮点数运算方法来实现浮点数。

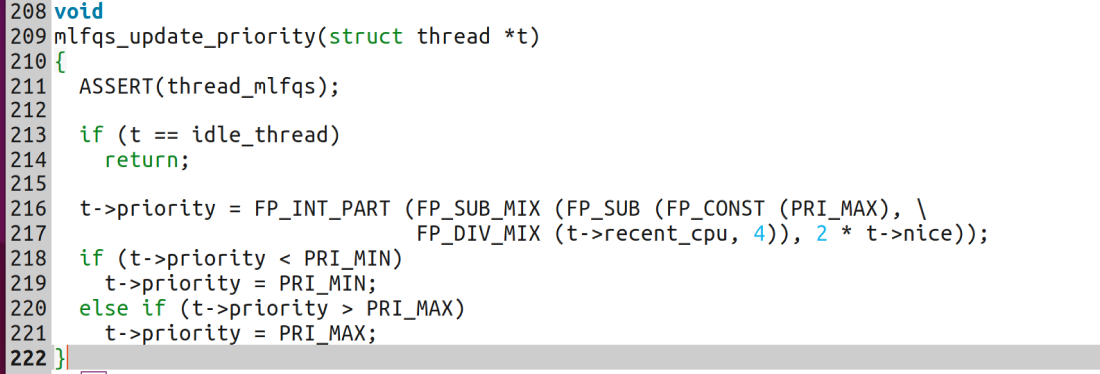
2、数据结构：



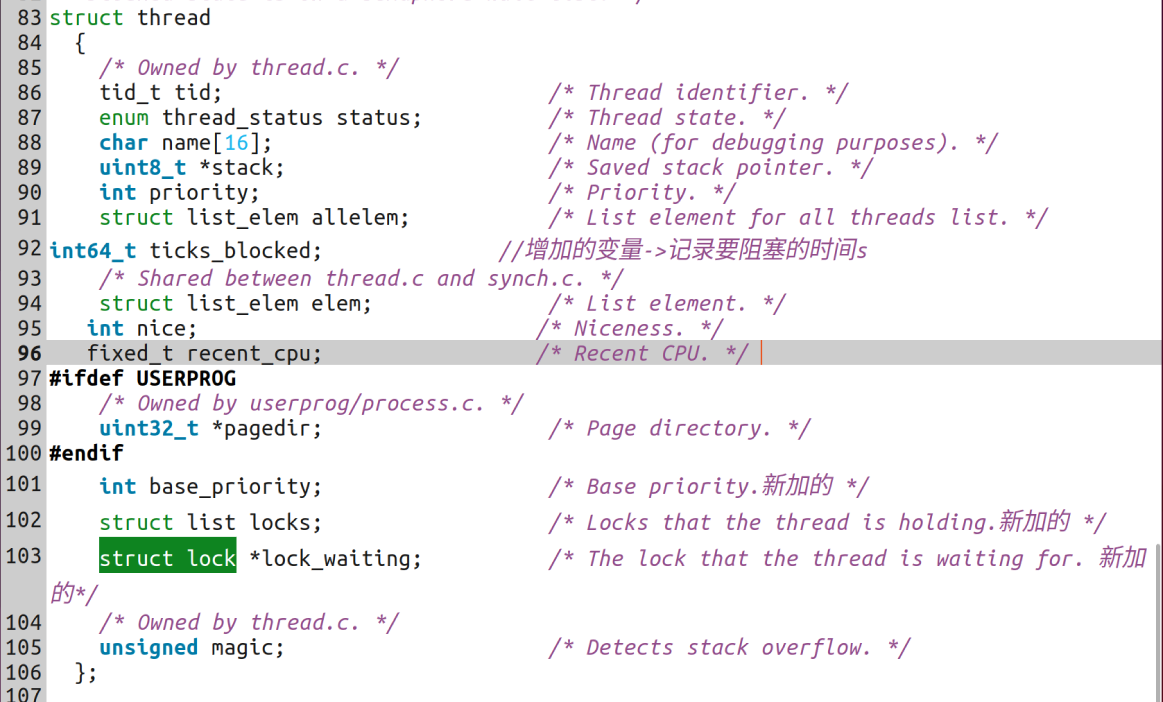


实现mlfqs\_update\_load\_avg\_and\_recent\_cpu函数

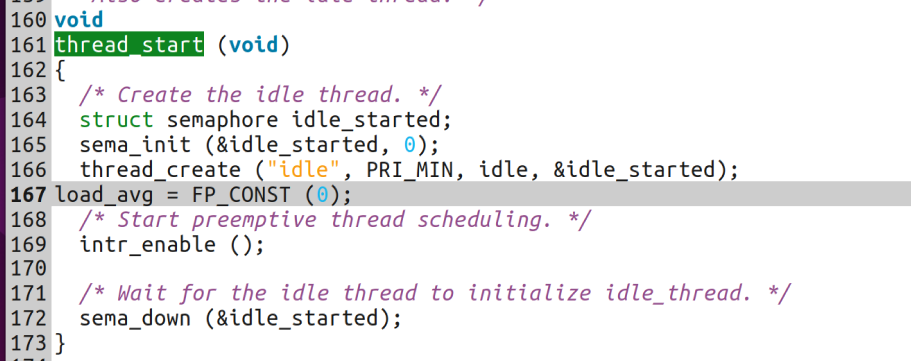




在thread结构体中加入成员并在init\_thread中初始化：

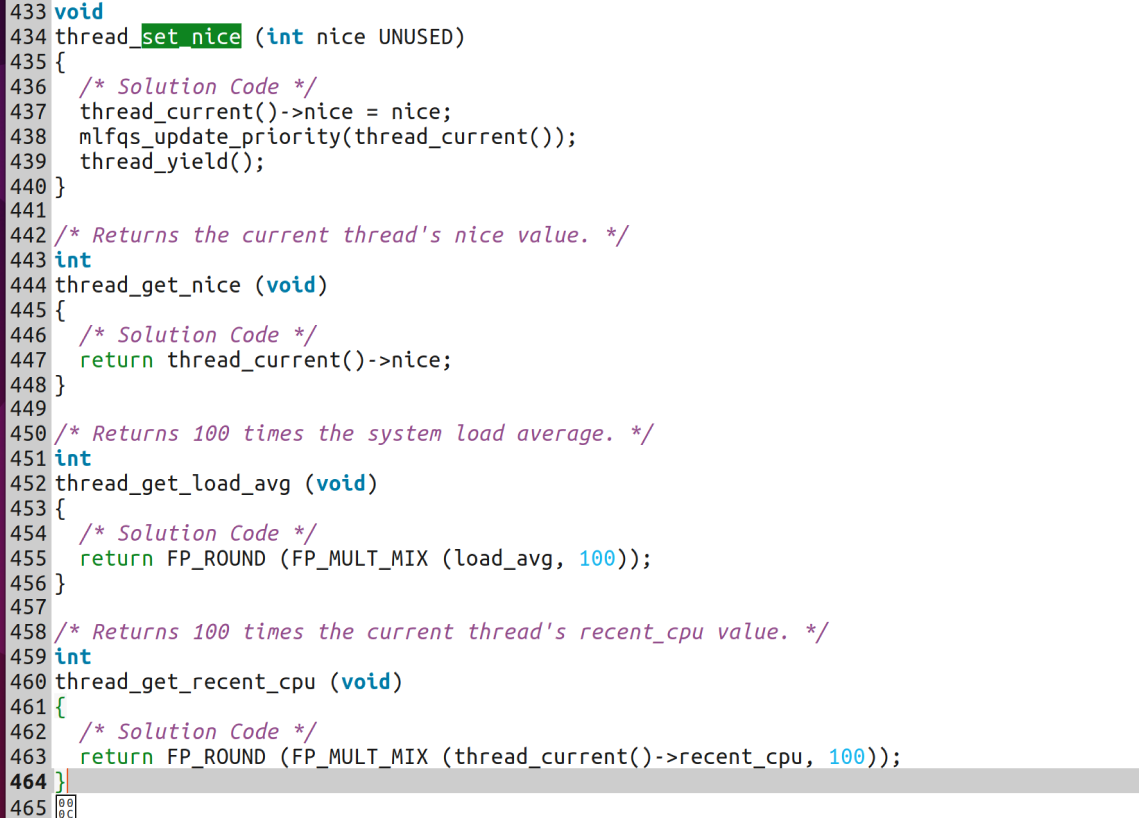


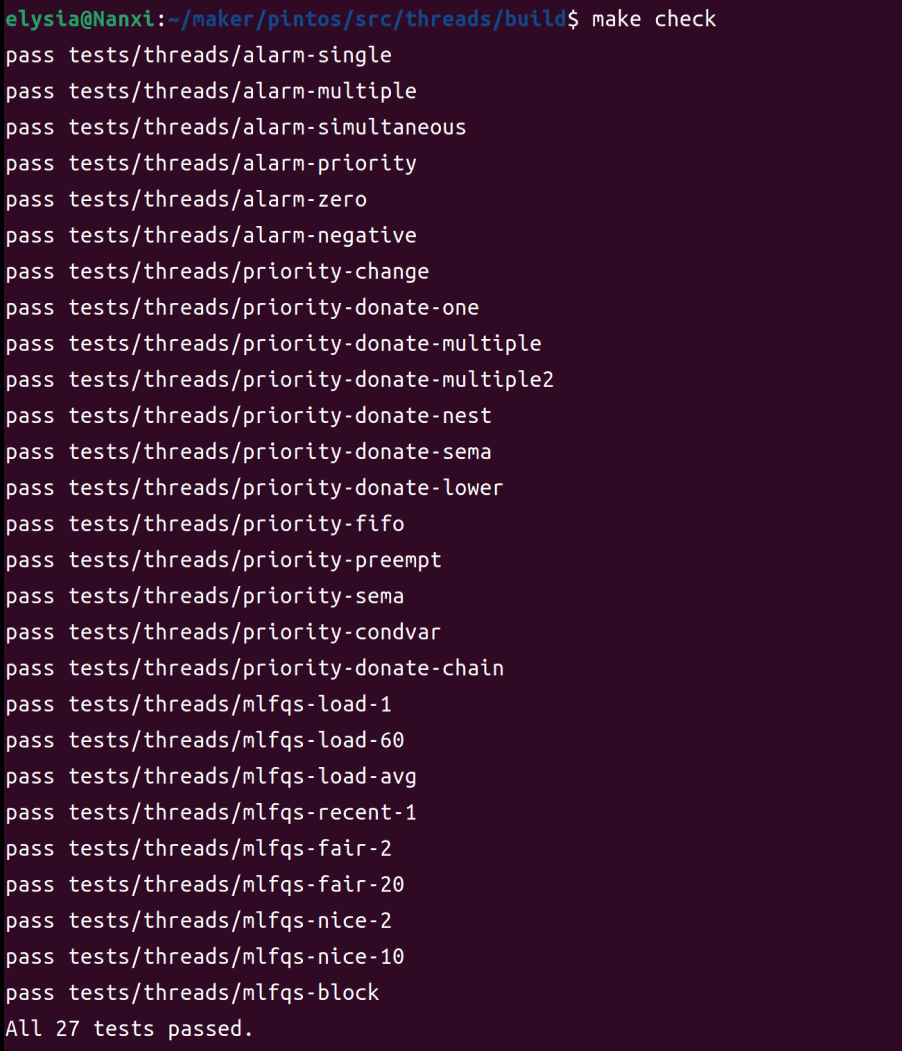




在thread.c中修改thread\_set\_nice、thread\_get\_nice、thread\_get\_load\_avg、thread\_get\_recent\_cpu函数：



3、实现结果：



错误和收获：

1、stropts.h找不到



在usr/include中新建一个以stropts.h为名的空文件，同时把 main 函数 里 关于isastream的288-293行语句注释掉即可。



Ubuntu使用过程中出现卡顿现象：

因为同时使用bochs启动了pintos，两个终端在一起模拟端口就会占用大量资源导致系统严重卡顿，pintos配置完，验证环境配置正确后关闭即可，不用再在虚拟机中继续保留这个进程，实验后续的内容都是在虚拟机上进行的。

ubuntu 在 /usr 下面添加删除文件时没有权限：

用户不是以root权限登陆的，命令行中使用sudo nautilus可以以root的身份打开并操作系统内的文件。

在网上找资料的时候，忽略了每个作者之间虚拟机的版本差异，导致跟着做的但是最后却报错，或者是因为环境不同导致同一种命令对应的具体指令也不同，比如yum是Linux的指令，对应的下载功能在Ubuntu中却是sudo apt install，在学习资料的时候应该多加辨别，不能一股脑照抄。