**实验报告**

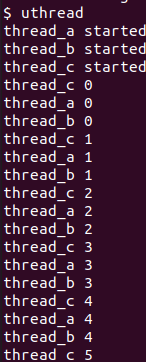
**Lab 5**

姓名：田鑫

班级：信息安全

学号：19307110353

**Part A：Uthread**

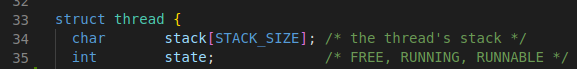
实验结果：

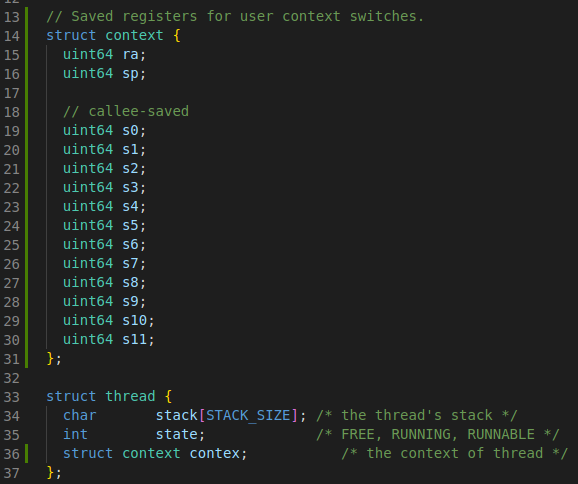
实验步骤：

1、分析确认应该在哪保存/恢复寄存器

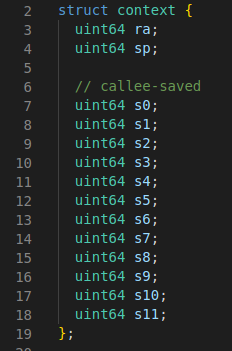
显然thread\_switch会做保存/恢复这件事，下一个问题是保存在哪？因为是用户级的线程切换，线程的创建之类都在uthread.c中完成，所以应当在uthread.c中写代码。

该文件中，此结构用于thread，所以在此基础上添加即可

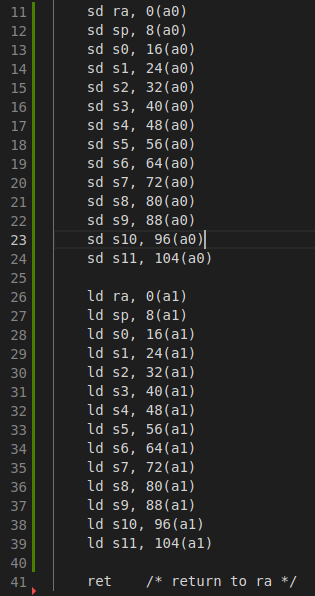




Context的内容参考proc.h当中的这个：



2、在uthread\_switch.s中实现上下文切换。（参考switch.s即可）



3、在thread\_schedule中添加调用上下文切换的代码



4、修改thread\_create完成线程的创建与初始化

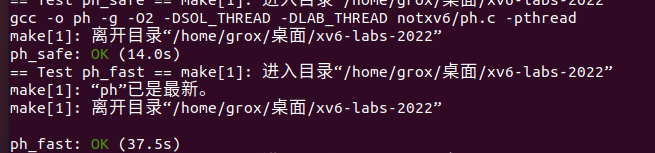


线程栈是从高位到低位，因此初始化时栈指针 sp 应该指向数组底部

返回地址 ra 直接指向该函数的地址就行，这样开始调度时，直接执行该函数（线程）

**Part B：Using threads**

实验结果：

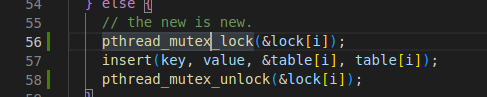


实验步骤：

1、先回答问题：

假设现在有两个线程T1和T2，两个线程都走到put函数，且假设两个线程中key%NBUCKET相等，即要插入同一个散列桶中。两个线程同时调用insert(key, value, &table[i], table[i])，insert是通过头插法实现的。如果先insert的线程还未返回另一个线程就开始insert，这里的p指向了后一个函数调用的e，前面调用的e会被忽略。

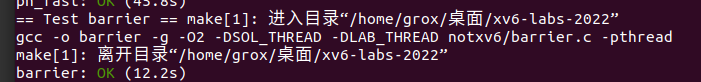
2、添加锁，此处锁的粒度还不能太大（其实也很简单）



即可

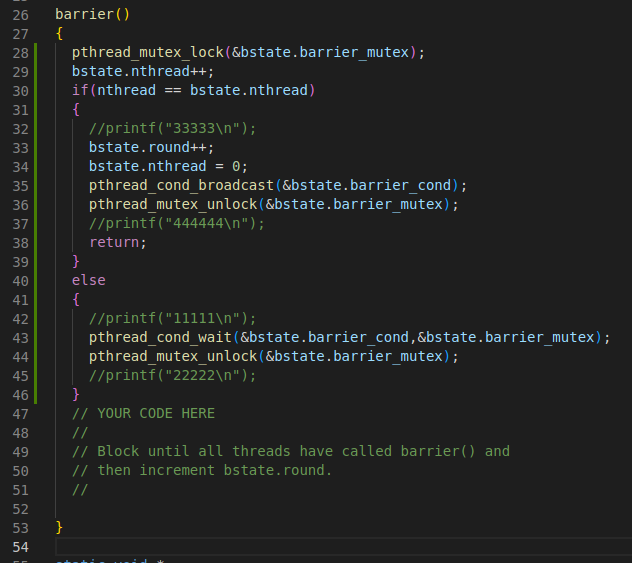
**Part C：Barrier**

实验结果：



实验步骤：

1、写barrier的代码：（比较简单）



这里是生产者消费者模式，如果还有线程没到达，就加入到队列中，等待唤起；如果最后一个线程到达了，就将轮数加一，然后唤醒所有等待这个条件变量的线程。

实验感想：

这次实验都挺简单的，离上课将多线程也过去了有段时间，只能说是复习了一下吧。