## 实验准备

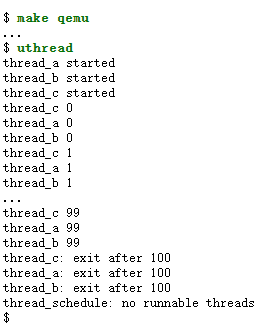
本实验将带你熟悉多线程。你将实现线程（用户级线程包）切换，使用多线程加速程序，实现一个barrier。在写代码前，你应该确定你已经读过xv6 book的章节7：scheduling，并且读过相关代码。

## Uthread：switching between threads

在本练习中，你将为用户级线程系统设计context切换机制，然后实现它。开始前，xv6有两个文件user/uthread.c和user/uthread\_switch.S，和一个在Makefile中的规则，来构建一个uthread程序。uthread.c包含绝大多数用户级线程包，以及3个简单的测试线程代码。线程包缺少一些代码来创建一个线程和在线程之间切换。

你的工作是提出一个计划：创建线程、存储/恢复寄存器来进行多线程切换，然后实现该计划。当你做完时，make grade应该说你的方案通过uthread test。

一旦你已经结束了，当你在xv6上执行uthread时，你应该看到下面输出（3个线程可能以不同顺序启动）：



这个输出来自3个测试线程，每个有一个循环（打印一行然后让出CPU到其他线程）。

基于这点，如果没有context切换代码，你将看不到输出。

你将需要添加代码到user/uthread.c中的thread\_creat()和thread\_schedule()，user/uthread\_switch.S的thread\_switch。目标一是确保当thread\_schedule()首次运行一个给定线程时，线程执行传到thread\_create()中的函数，在它自己的栈上。另外目标是确保thread\_switch保存切换前线程的寄存器，恢复要切换线程的寄存器，返回到切换后的线程上次离开时的指令。你将不得不决定哪里存储/恢复寄存器；更改struct thread来保存寄存器是一个好计划。你将需要在thread\_schedule添加thread\_switch调用；你能传递任何需要的参数到thread\_switch，但目的是从一个线程切换到下个线程。

一些提示：

thread\_switch仅需要存储/恢复callee-save寄存器。为什么？

你可以在user/uthread.asm中看到uthread汇编代码，这可能对调试有帮助。

对于测试你的代码，使用riscv64-linux-gnu-gdb单步thread\_switch可能是有帮助的。你可以用这种方式启动：

(gdb) file user/\_uthread

Reading symbols from user/\_uthread...

(gdb) b uthread.c:60

这会在uthread.c第60行设置一个断点。这个断点可能会（也可能不会）在你运行uthread之前触发。这如何发生？

一旦你的xv6 shell运行，输入”uthread”，gdb将在60行停住。现在你可以如下输入命令来检测uthread的状态：

(gdb) p/x \*next\_thread

用”x”，你可以检测内存位置的内容：

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

你可以跳到thread\_switch开始处：

(gdb) b thread\_switch

(gdb) c

你可以使用si来单步汇编指令：

(gdb) si