Lab: Multithreading

Uthread: switching between threads

- 在本练习中，你将为用户级线程系统设计上下文切换机制，然后实现它。xv6 有两个文件 user/uthread.c 和 user/uthread_switch.S，以及makefile中用于构建 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。
- 你的任务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。

Uthread: switching between threads

- 当你完成后应当看到如下输出：

```
$ 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
$
```

Uthread: switching between threads

- 你需要将代码添加到 user/uthread.c 中的 thread_create（） 和 thread_schedule（），以及 user/uthread_switch.S 中的 thread_switch。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，恢复正在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。你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。你需要在thread_schedule中添加对thread_switch的调用;您可以传递需要thread_switch的任何参数，但目的是从线程 t 切换到next_thread。

Using threads

- 在本作业中，你将探索使用哈希表的线程和锁的并行编程。你应该在具有多个内核的真实 Linux 或 MacOS 计算机（不是 xv6，不是 qemu）上完成。
- 该作业使用 UNIX 线程库pthreads。你可以从手册页找到关于它的信息，用 man pthreads命令，或者查看如下参考资料：
- https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_mutex_lock.html
- https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_mutex_init.html
- https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_create.html

Using threads

- 文件 notxv6/ph.c 包含一个简单的哈希表，如果从单个线程使用，则正确，但在从多个线程使用时不正确

```
$ make ph
$ ./ph 1
100000 puts, 3.991 seconds, 25056 puts/second
0: 0 keys missing
100000 gets, 3.981 seconds, 25118 gets/second
```

```
$ ./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
```

- 查看notxv6/ph.c，特别是put（）和insert（）

Using threads

- 请注意，要build ph.c，Makefile使用自己操作系统的gcc，而不是6.S081的工具。
- 不要忘记调用 pthread_mutex_init（）。首先使用 1 个线程测试代码，然后使用 2 个线程进行测试。

Barrier

- 在此作业中，你将实现一个barrier：应用程序中的某个点，在该点上，所有参与线程都必须等待，直到所有其他参与线程也到达该点。你将使用 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。
- 应该在真实计算机上执行此操作（不是 xv6，不是 qemu）。
- 文件notxv6/barrier.c 包含一个有问题的barrier.

Barrier

```
$ make barrier
$ ./barrier 2
barrier: notxv6/barrier.c:42: thread: Assertion `i == t' failed.
```

- 参数指定在屏障上同步的线程数（barrier.c 中的 nthread）。每个线程执行一个循环。在每次循环迭代中，线程调用 barrier（），然后休眠随机微秒。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在 barrier（） 中阻塞，直到它们的所有 n 个线程都调用了 barrier（）。
- 您的目标是实现所需的barrier行为。
- 参考如下：
- https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_cond_wait.html
- https://pubs.opengroup.org/onlinepubs/007908799/xsh/pthread_cond_broadcast.html

实验提交

在各位同学提交实验之前，务必运行 make grade命令以检验自己是否通过了所有的测试，并且**保证**xv6的其他模块没有被你新增的代码所损坏（详见上页PPT）。

1. 实验代码文件夹（提交前请make clean）
2. 实验报告（中文即可，PDF格式）
   - 实现思路，实现代码，测试结果。
   - 本实验中有问题回答环节，请勿遗漏
   - 实验中遇到的问题，如何思考并解决
   - 实验总结
3. 实验报告提交截止日期：2022年12月5日24:00点

1、2放入一个文件夹，命名为：学号-姓名-oslab5.zip

注意：请各位同学独立完成实验，参考代码需注明

实验报告（**PDF**）提交：

【腾讯文档】操作系统2022课程lab5

https://docs.qq.com/form/page/DV0xlbXBWeGd4WG5i

实验压缩包提交：

教室	对应邮箱
302	22210240347@m.fudan.edu.cn
204 205	michael_chen22@126.com
202	21210240021@m.fudan.edu.cn

请各位同学，务必提交到**链接**和**邮箱**。