

Homework: Barriers

刘恒星 2022229044

April 9, 2023

1 Part One: 实验要求

在这个作业中，将探讨如何使用 pthread 库提供的条件变量来实现一个 barrier。barrier 是一个应用程序中的一个点，在这个点上，所有线程都必须等待，直到所有其他线程也到达这个点。条件变量是一种序列协调技术，类似于 xv6 的 sleep 和 wakeup。每个线程都调用一个循环，在每个循环迭代中，一个线程调用 barrier()，然后睡眠一些随机数量的微秒。断言触发了，因为一个线程在另一个线程到达屏障之前就离开了屏障。我们期望的行为是，所有线程都应该阻塞，直到有 n 个线程调用 barrier。

2 Part Two: 实验步骤

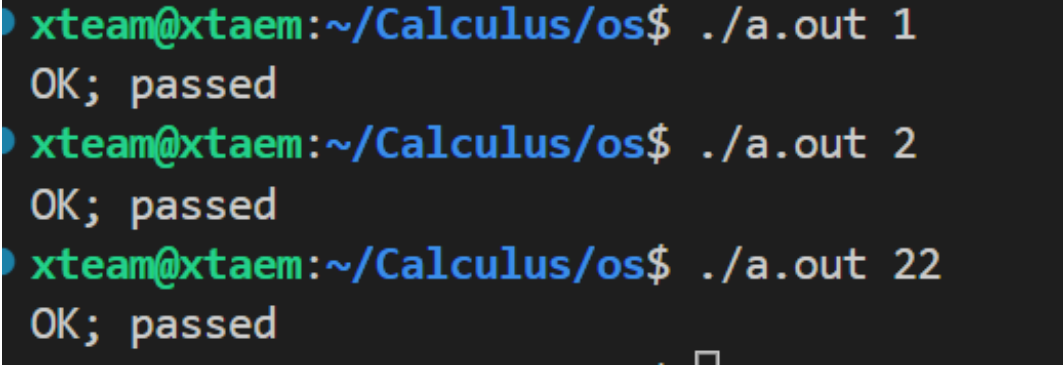
看到关键的函数

```
1  static void
2  barrier()
3  {
4      bstate.round ++;
5  }
6
7  static void *
8  thread(void *xa)
9  {
10     long n = (long)xa;
11     long delay;
12     int i;
13
14     for (i = 0; i < 20000; i++)
15     {
16         int t = bstate.round;
17         // printf("%d %d\n", i, t);
18         assert(i == t);
19         barrier();
20         usleep(random() % 100);
21     }
22 }
```

我们可以看见，因为 i 和 t 不同步，所以触发断言。其根本原因就是不同线程触发了 barrier，导致一轮循环之后每一个线程都增加了 round。如果我们想要达到所有线程调用之后才增加一个 round，那我们就需要让所有线程进入 barrier 函数之后，阻塞并等待所有的线程都到达函数中，再给 round 加一。可以使用条件变量来阻塞和释放，代码如下：

```
1  static void
2  barrier()
3  {
4      pthread_mutex_lock(&bstate.barrier_mutex);
5      bstate.nthread++;
6      if(bstate.nthread == nthread)
7      {
8          bstate.round ++;
9          bstate.nthread = 0;
10         pthread_cond_broadcast(&bstate.barrier_cond);
11     }
12     else
13     {
14         pthread_cond_wait(&bstate.barrier_cond, &bstate.barrier_mutex);
15     }
16     pthread_mutex_unlock(&bstate.barrier_mutex);
17 }
```

效果如下：



```
xteam@xtaem:~/Calculus/os$ ./a.out 1
OK; passed
xteam@xtaem:~/Calculus/os$ ./a.out 2
OK; passed
xteam@xtaem:~/Calculus/os$ ./a.out 22
OK; passed
```