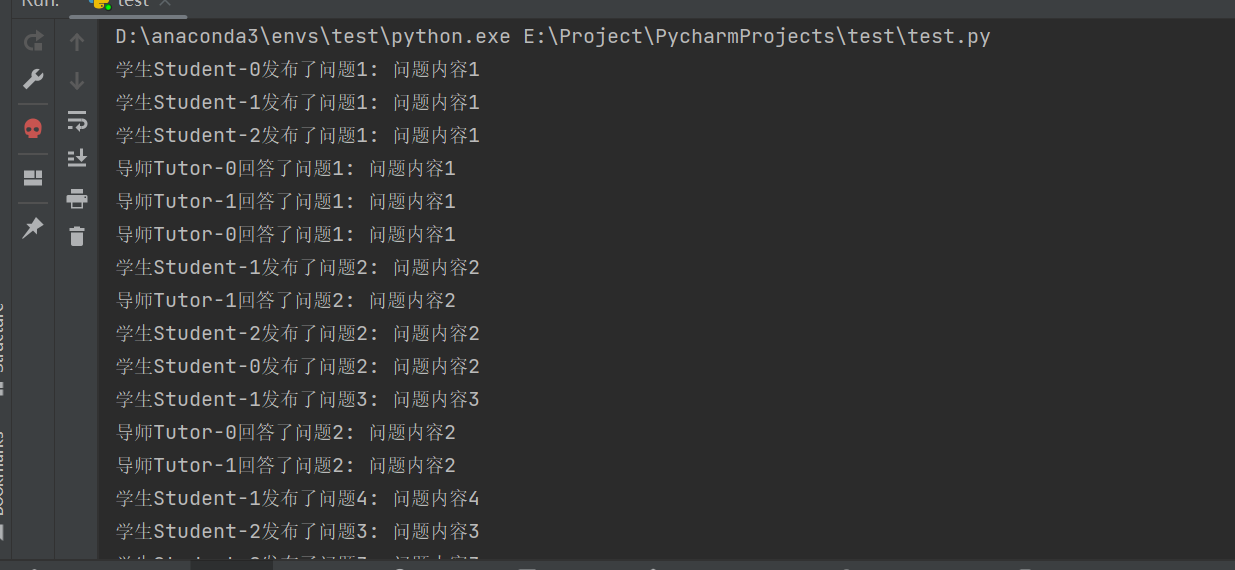
**120L022109 李世轩**

**源代码**

import threading  
import queue  
import random  
import time  
  
# 全局共享队列  
question\_queue = queue.Queue()  
  
# 互斥锁  
mutex = threading.Lock()  
  
# 条件变量  
student\_condition = threading.Condition(mutex)  
tutor\_condition = threading.Condition(mutex)  
  
  
class Question:  
 def \_\_init\_\_(self, id, content):  
 self.id = id  
 self.content = content  
  
  
def student():  
 question\_id = 1  
 while True:  
 # 生成新问题  
 content = f"问题内容{question\_id}"  
 new\_question = Question(question\_id, content)  
  
 with student\_condition:  
 # 将新问题加入问题队列  
 question\_queue.put(new\_question)  
 print(f"学生{threading.current\_thread().name}发布了问题{new\_question.id}: {new\_question.content}")  
  
 # 唤醒等待的导师线程  
 tutor\_condition.notify\_all()  
  
 # 等待随机时间，模拟学生提问过程  
 time.sleep(random.uniform(0.5, 3))  
  
 question\_id += 1  
  
  
def tutor():  
 while True:  
 with tutor\_condition:  
 # 等待问题出现在队列中  
 while question\_queue.empty():  
 tutor\_condition.wait()  
  
 # 回答队列中的问题  
 question = question\_queue.get()  
 print(f"导师{threading.current\_thread().name}回答了问题{question.id}: {question.content}")  
  
 # 等待随机时间，模拟回答问题过程  
 time.sleep(random.uniform(1, 3))  
  
  
def main():  
 # 创建学生和导师线程  
 student\_threads = [  
 threading.Thread(target=student, name=f"Student-{i}")  
 for i in range(3)  
 ]  
 tutor\_threads = [  
 threading.Thread(target=tutor, name=f"Tutor-{i}")  
 for i in range(2)  
 ]  
  
 # 启动线程  
 for t in student\_threads:  
 t.start()  
 for t in tutor\_threads:  
 t.start()  
  
 # 等待线程结束  
 for t in student\_threads:  
 t.join()  
 for t in tutor\_threads:  
 t.join()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**效果截图**



**功能设计说明**

在这个应用中，我们将创建一个简单的在线教育平台。

这里我们有两类用户：学生和导师。

学生可以发布问题，而导师可以回答这些问题。

该程序使用Python中的线程和队列，模拟了问题发布和回答的过程。在创建多个学生线程和导师线程后，学生线程会随机生成一个新问题，并将其加入到共享的问题队列中。导师线程等待着从队列中获取问题，并回答问题。

程序通过互斥锁和条件变量来保证线程之间的同步和安全性。互斥锁用于修改共享数据时的临界区互斥访问，条件变量则用于在线程之间通信，使得线程能够在某些特定条件下进行等待和唤醒。

操作系统进程

在操作系统中，一个进程代表了运行中的程序，它包括了所需的资源（如内存空间、文件句柄等）及其运行所需的代码和数据。一个进程也经常被视为操作系统分配给应用程序的最小单位。 进程的创建由操作系统完成，并分配给每个进程独立的地址空间，以确保进程之间的内存隔离和安全性。此外，操作系统通过一些机制来管理进程的状态，监控其资源使用情况及其与其他进程的交互。

操作系统线程

线程是进程的子集，表示程序执行的一条路径，或称为代码执行的单元。线程共享该进程所拥有的全部资源，可以在不同线程之间传递数据。线程比进程更轻量级，因此可以同时创建和执行更多线程。但是，与进程相反的是，线程没有单独的地址空间，而是通过共享调用者/父线程的地址空间来实现内存访问。

IPC

IPC（Inter-process communication，即进程间通信）指的是在操作系统中实现进程间消息交换和共享资源的技术和机制。这些资源可能是文件、管道、队列、共享内存等，它们通常用于跨进程提供同步和通信。

常见的IPC机制有：

1. 命名管道：用于实现两个本地进程的通信。
2. 消息队列：被用于在进程间发送消息。
3. 共享内存：允许将一个缓冲区或一块物理内存看成是所有进程的私有地址空间，进程可以直接读取和写入共享内存，从而实现数据共享。
4. 信号量：是一个整数计数器，可以用于线程同步，也可用于进程通信中。