桂林电子科技大学

**实验4 进程和线程**  实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 实验名称 | **进程和线程** | | | | | | | |  | 辅导员意见：  成绩 辅导员  签 名 |
| 院 系 | 计算机与信息安全学院 | | | 专业 | | 计科专业 | | |
| 学 号 | 2100300234 | | | 姓名 | | 谢志锋 | | |
| 实验日期 | 2023 | 年 | 4 | | 月 | | 9 | 日 |
|  |  | | | | | | | |

## 一、实验目的

1. 掌握和使用进程池
2. 掌握和使用多线程
3. 掌握和使用互斥锁
4. 掌握协程的使用

## 二、实验内容

1. 使用进程池计算区间内素数个数
2. 使用多线程计算区间内合数个数
3. 互斥锁的设计和使用
4. 异步状态转换器

## 三、实验环境

在Educoder平台进行实验

## 四、实验要求

根据每个实训的每个关卡要求完成代码提交和测评

## 五、实验步骤

第四关：

定义了State1（）的异步函数。该函数接受两个参数：transition\_value和input\_value。在函数内部，将S1添加到列表中，然后根据输入值调用State2或State3。如果输入值为1，则调用State2并传递1和next(nums)作为参数。否则，调用State3并传递0和next(nums)作为参数。

定义了一个名为State2的异步函数。该函数接受两个参数：transition\_value和input\_value。在函数内部，将S2添加到列表中，然后根据输入值调用State3或State1。如果输入值为1，则调用State3并传递1和next(nums)作为参数。否则，调用State1并传递0和next(nums)作为参数。

定义State3()的异步函数，用于状态S3。在这个函数中，使用一个列表来跟踪状态转换。在这个函数中，将S3状态添加到列表中，并打印出上一个状态到S3状态的条件。然后，检查输入值是否为1。如果是，则调用EndState()函数，否则调用State1()函数。

定义EndState()的异步函数，用于结束状态S4。在这个函数中，使用一个列表来跟踪状态转换。将S4状态添加到列表中，并打印出上一个状态到S4状态的条件。

import asyncio

list = []

async def StartState(nums):

    '''

    开始状态S0

    :param nums:输入的条件

    :return:

    '''

    #你的代码在这里#

    print('Start Transition :')

    list.append('S0')

    if next(nums) == 1:

        await State1(1,next(nums))

    else:

        await State1(0,next(nums))

async def State1(transition\_value: int, input\_value):

    '''

    状态S1

    :param transition\_value:上一个状态到S1状态的条件，例如StartState转换到State1时，transition\_value为1

    :param input\_value:输入的条件nums

    :return: 当转换结束时，返回S1状态转换到其他状态的动作

    '''

    #你的代码在这里#

    i=len(list)

    list.append('S1')

    print('{0} calling {1} value = {2}'.format(list[i-1],list[i],transition\_value))

    if input\_value == 1:

        await State2(1,next(nums))

    else:

        await  State3(0,next(nums))

async def State2(transition\_value: int, input\_value):

    '''

    状态S2

    :param transition\_value:上一个状态到S2状态的条件，例如StartState转换到State2时，transition\_value为0

    :param input\_value:输入的条件nums

    :return:当转换结束时，返回S2状态转换到其他状态的动作

    '''

    #你的代码在这里#

    i=len(list)

    list.append('S2')

    print('{0} calling {1} value = {2}'.format(list[i - 1], list[i], transition\_value))

    if input\_value == 1:

        await State3(1,next(nums))

    else:

        await State1(0,next(nums))

#11001

async def State3(transition\_value: int, input\_value):

    '''

    状态S3

    :param transition\_value:上一个状态到S3状态的条件，例如State2转换到State3时，transition\_value为1

    :param input\_value:输入的条件nums

    :return:当转换结束时，返回S3状态转换到其他状态的动作

    '''

    #你的代码在这里#

    i = len(list)

    list.append('S3')

    print('{0} calling {1} value = {2}'.format(list[i - 1], list[i], transition\_value))

    if input\_value == 1:

        await EndState(1)

    else:

        await State1(0,next(nums))

async def EndState(transition\_value: int):

    '''

    结束状态S4

    :param transition\_value:上一个状态到S4状态的条件

    :return:返回转换结束时的动作

    '''

    #你的代码在这里#

    i = len(list)

    list.append('S4')

    print('{0} calling {1} value = {2}'.format(list[i - 1], list[i], transition\_value))

if \_\_name\_\_ == "\_\_main\_\_":

    nums = iter(eval(input()))

    loop = asyncio.get\_event\_loop()

    loop.run\_until\_complete(StartState(nums))

## 

第一关：

首先定义判断素数函数is\_prime()，使用concurrent.futures.ProcessPoolExecutor()模块并发地执行is\_prime()函数，对于n范围内的每个数字进行计算，然后将结果相加。

import concurrent.futures

import math

def is\_prime(m):

    """

    判断一个数m是不是素数

    :param m: 正整数

    """

    #        请在此处添加代码       #

    # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

    if m < 2:

        return False

    if m in (2, 3):

        return True

    if not m & 1:

        return False

    for i in range(3, int(m \*\* 0.5) + 1, 2):

        if m % i == 0:

            return False

    return True

    # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

def main(n):

    """

    判断0~n之间素数的个数

    :param m: 正整数

    """

    #        请在此处添加代码       #

    # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

    with concurrent.futures.ProcessPoolExecutor() as executor:

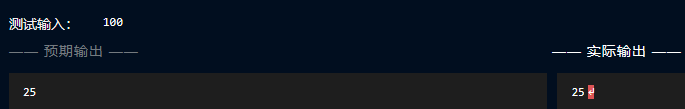
        print(sum(executor.map(is\_prime,range(n))))

    # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

if \_\_name\_\_ == '\_\_main\_\_':

    n = int(input())

    main(n)



第二关：

首先定义判断素数函数is\_prime()，定义了一个名为seprateNum()的函数，该函数将整个数字空间N分成CPU\_COUNT个部分。它返回一个列表，其中包含每个部分的起始和结束数字。在这个函数中，使用了列表推导式来生成一个包含所有部分的列表。然后，将第一个部分的起始数字设置为1，将最后一个部分的结束数字设置为N（如果它小于N）。定义how\_many函数，用for循环区间，统计区间内一共有多少个素数并返回。最后，主函数中获取一个整数N，然后将其分成32个部分。然后，它创建32个线程，并将每个线程分配给一个部分。每个线程都调用howMany()的函数，并将其部分作为参数传递。在所有线程都完成后，代码打印出N - 1 - ans的值。

import threading

import math

ans = 0

lock = threading.Lock()

import threading

import math

ans = 0

lock = threading.Lock()

def isPrime(n):

    # 判断数字是否为素数

    global ans

    if n <= 1:

        return False

    for i in range(2, int(math.sqrt(n)) + 1):

        if n % i == 0:

            return False

    return True

def howMany(T):

    # 计算给定区间含有多少个素数

    sum = 0

    for i in range(T[0], T[1] + 1):

        if isPrime(i):

            sum += 1

    lock.acquire()

    try:

        global ans

        ans += sum

    finally:

        lock.release()

def seprateNum(N, CPU\_COUNT):

    # 对整个数字空间N进行分段CPU\_COUNT

    list = [[i \* (N // CPU\_COUNT) + 1, (i + 1) \* (N // CPU\_COUNT)] for i in range(0, CPU\_COUNT)]

    list[0][0] = 1

    if list[CPU\_COUNT - 1][1] < N:

        list[CPU\_COUNT - 1][1] = N

    return list

if \_\_name\_\_ == '\_\_main\_\_':

    N = int(input())

    threadNum = 32

 #        请在此处添加代码       #

    # \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

    t = []

    sepList = seprateNum(N, threadNum)

    for i in range(0, threadNum):

        t.append(threading.Thread(target = howMany, args = (sepList[i], )))

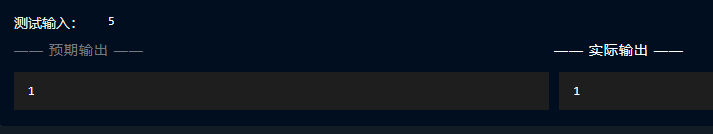
        t[i].start()

    for i in range(0, threadNum):

        t[i].join()

    print(N - 1 - ans, end = '')

    # \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#



第三关:

主函数中先定义3个互斥锁lockfoo，lockbar和lockpython，以及3个线程t1，t2和t3。每个线程都调用一个名为showfoo()，showbar()或showpython()的函数，并传递一个整数参数。使用互斥锁来确保线程按照正确的顺序打印输出。先锁住lockpython和lockbar锁，以确保先打印出foo。定义showpython()的函数，使用一个互斥锁来确保线程按照正确的顺序打印输出。使用一个for循环来打印出python字符串。在每次循环中，先锁住了lockpython锁，然后打印出python字符串。最后，释放lockfoo锁。定义了showbar()的函数，使用一个互斥锁来确保线程按照正确的顺序打印输出。使用一个for循环来打印出bar字符串。在每次循环中，先锁住了lockbar锁，然后打印出bar字符串。最后，释放了lockpython锁。

import threading

import sys

import time

def showfoo(n):

    '''

    :param n: 要输出foobarpython的次数

    :return: 无返回，可直接输出

    '''

    # 请在此处添加代码       #

# \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

    for i in range(n):

      lockfoo.acquire()

      print("foo",end="")

      lockbar.release()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

def showbar(n):

    '''

      :param n: 要输出foobarpython的次数

      :return: 无返回，可直接输出

      '''

    # 请在此处添加代码       #

# \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

    for i in range(n):

      lockbar.acquire()

      print("bar",end="")

      lockpython.release()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

def showpython(n):

    '''

      :param n: 要输出foobarpython的次数

      :return: 无返回，可直接输出

      '''

     # 请在此处添加代码       #

# \*\*\*\*\*\*\*\*\*\*\*\*\*begin\*\*\*\*\*\*\*\*\*\*\*\*#

    for i in range(n):

      lockpython.acquire()

      print("python",end="")

      lockfoo.release()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*#

if \_\_name\_\_ == '\_\_main\_\_':

    lockfoo = threading.Lock()  # 定义3个互斥锁

    lockbar = threading.Lock()

    lockpython =threading.Lock()

    n = int(input())

    t1 = threading.Thread(target=showfoo,args=[n])  # 定义3个线程

    t2 = threading.Thread(target=showbar,args=[n])

    t3 = threading.Thread(target=showpython,args=[n])

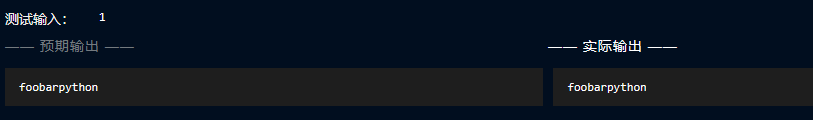
    lockpython.acquire()  # 先锁住foo,bar锁，保证先打印foo

    lockbar.acquire()

    t1.start()

    t2.start()

    t3.start()



## 六、问题记录和实验总结（必写）

在这一次实验中，我学会了运用python进程和线程解决实际问题，在质数处理中，多进程明显比单线程在时间上更具有优势，但由于代码复杂或者说是实现繁琐问题，我认为应该在处理大型进程时才采用多进程优化效率。python的进程和线程是python语言的重要组成部分，python的进程和线程方法的掌握，对于我未来的前沿专业知识学习和自身创新创业能力的培养以及学术能力的塑造都有着重要意义。