作业10 进程、线程与协程

提交截止时间：2020年12月4日（周五）晚23:59

1

下面的python程序将`current`文件夹下的所有文件复制到`target`文件夹下。（见`homework1.py`）

```
import os, shutil, time
from multiprocessing import Pool, Manager


# 定义了一个copy文件的方法
def copy_file(file, old_folder, new_folder):

    source = os.path.join(old_folder, file)
    target = os.path.join(new_folder, file)

    file_size = (os.path.getsize(old_folder + "/" + file)) / 1024 / 1024
    print("Size:%.2f M | %s" % (file_size, target))

    with open(source, "rb") as fr:
        with open(target, "wb") as fw:
            while True:
                content = fr.read(1024 * 1024)
                fw.write(content)
                if not content:
                    break

def main():
    current_folder = './current'
    target_folder = './target'

    if os.path.exists(target_folder):
        shutil.rmtree(target_folder)

    os.mkdir(target_folder)

    # 获取当前目录下所有的文件名
    all_file = os.listdir(current_folder)

    # 复制
    for file in all_file:
        copy_file(file, current_folder, target_folder)

if __name__ == '__main__':
    main()
```

请你修改上面的程序，使用多线程，并显示复制的进度。你可以对函数`copy_file`进行必要的修改。

提示：

- 可以每个进程负责一个文件的复制。为了避免一次开太多进程把系统挤爆，可以考虑使用进程池。
- 使用进程池时，建议使用`multiprocess.Manager().Queue()`代替`multiprocess.Queue()`进行进程间的通信。[1]

2

协程本质上是子程序的推广，它可以在执行过程中将自身挂起，而之后可以重入之前挂起的地方继续执行。

关键字`async`会把一个函数定义为协程。关键字`await`可以把自身挂起并调用另一个awaitable的对象（如另一个协程），直到满足一定条件后才自身继续执行。

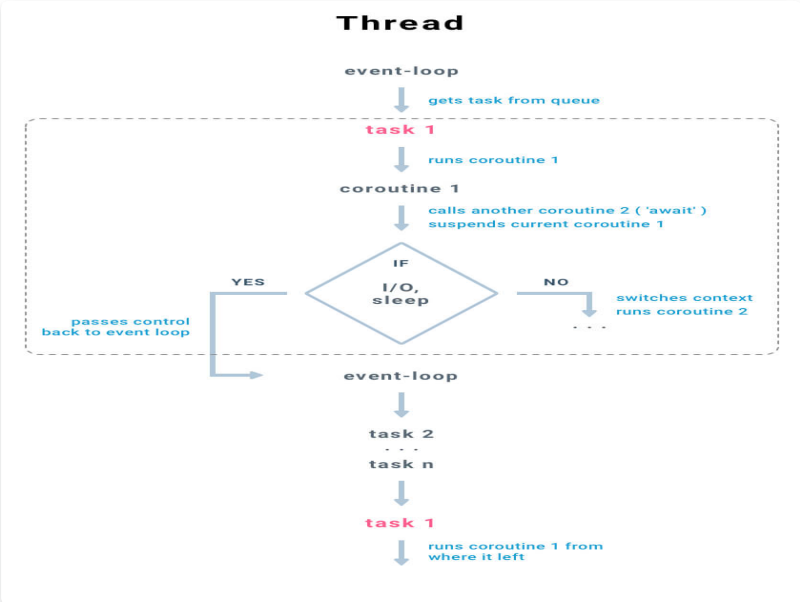
下面是简化版的生产者-消费者模型，其中生产者总是等待消费者的命令才生产一个物品，而这个物品随后马上被消费掉：

```
import random, asyncio

x = 0

async def produce():
    global x
    x += 1
    await asyncio.sleep(1)
    print("Produce %d" % x)
    return x

async def consumer(num):
    count = 0
    while True:
        item = await produce()
        await asyncio.sleep(1)
        print("Consume %d" % item)
        count += 1
        if count == num:
            break

asyncio.run(consumer(3))
```

请你运行程序给出结果，并用文字或图形解释程序的运行流程。

3

2中的程序仍然是个串行的链式结构。不妨让生产者和消费者都往队列里进行存取，并让`asyncio`提供的事件循环`event_loop`对这些协程的执行和切换进行自动的调度。事件循环会按下面的方式调度协程[2]：



`asyncio.Queue()`提供了支持异步操作的队列。下面便是课堂上展示的使用异步操作的生产者-消费者模型。

```
import asyncio, time
```

```
async def consumer(q):
    print('consumer starts.')
    while True:
        item = await q.get()
        if item is None:
            q.task_done() # Indicate that a formerly enqueued task is complete.
            break
        else:
            await asyncio.sleep(1) # take 1s to consume
            print('consume %d' % item)
            q.task_done()
    print('consumer ends.')

async def producer(q):
    print('producer starts.')
    for i in range(5):
        await asyncio.sleep(1) # take 1s to produce
        print('produce %d' % i)
        await q.put(i)
    await q.put(None)
    await q.join()  # Block until all items in the queue have been gotten and processed.
    print('producer ends.')

q = asyncio.Queue(maxsize=10)
t0 = time.time()
loop = asyncio.get_event_loop()
tasks = [producer(q), consumer(q)]
loop.run_until_complete(asyncio.wait(tasks))
loop.close()
print(time.time() - t0, " s")
```

请问：生产者、消费者只不过作为协程被碎片化调度执行了，为什么实际运行花费的时间少于10s？为什么协程十分适合进行IO密集型任务，比如网络爬虫？

> （选做）如果你愿意进行更多的探索，你可以尝试利用协程实现多个生产者消费者的模型、利用aiohttp库爬取若干网络页面等等。

---

1. 请参考https://stackoverflow.com/questions/43439194/python-multiprocessing-queue-vs-multiprocessing-manager-queue。↩
2. 图片来源：https://www.linux.com/training-tutorials/asynchronous-programming-python-asyncio-0/↩