homework1.py

```
# 先把要复制的文件放在current文件夹下

import os, shutil, time
from multiprocessing import Pool, Manager
tot=0

# 定义了一个copy文件的方法
def copy_file(q, num, file, old_folder, new_folder):
    # while not q.empty():
    # file=q.get()
    source = os.path.join(old_folder, file)
    target = os.path.join(new_folder, file)

    file_size = (os.path.getsize(old_folder + "/" + file)) / 1024 / 1024
    cur=q.get()
    q.put(cur+1)
    blocks=int(cur/tot*50)
    print("[%s] %d%% id:%d Size: %.2f M | %s" % ('#'*blocks+'.'*(50-bloc
    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)
    global tot
    tot=len(all_file)
    start=time.time()
    cores=16
    pl=Pool(cores)
    q=Manager().Queue()
    q.put(1)

    for i,file in enumerate(all_file):
        pl.apply_async(copy_file,args=(q,i+1,file,current_folder,target_
    pl.close()
    pl.join()
    # 复制
    # for file in all_file:
    #     copy_file(0, file, current_folder, target_folder)
    end=time.time()
    print(f'\nfinished,time:{end-start}s')
if __name__ == '__main__':
    main()
```

homework2.py

运行结果:

```
Produce 1
Consume 1
Produce 2
Consume 2
Produce 3
Consume 3
```

其中每行的输出时间相差一秒
线程首先进入consumer的循环，然后执行至await语句，consumer函数被挂起，切到produce函数，执行至await语句，再挂起切到sleep函数中，produce执行结束后，切回到consumer函数，再类似地await挂起1秒，然后对其consume，以此循环，当consume了num次时，就跳出循环

homework3.py

运行结果:

```
consumer starts.
producer starts.
produce 0
produce 1
consume 0
produce 2
consume 1
produce 3
consume 2
produce 4
consume 3
consume 4
consumer ends.
producer ends.
6.010125637054443  s
```

因为在一个协程在await asyncio.sleep(1)时，整个线程并不会停止执行，而是会将这个协程挂起，执行其他的协程；因此，程序执行的大部分时间是两个协程同时在await asyncio.sleep(1)，（相当于是“并行”地“sleep”），因此总执行时间小于10s 如果将async.sleep换成IO操作，那么就可以利用一个协程进行IO的时间去执行别的协程，故理论上能将IO操作占用线程的运行时间降至最低；上下文切换效率高于多进程与多线程，同时不用担心出现线程死锁的问题