**作业三：采用Python语言创建多进程：**

利用Python内置工具包multiprocessing实现：

**代码部分：**

方法一：向Process中直接传入要运行的方法

def foo(i):

print('say hi', i)

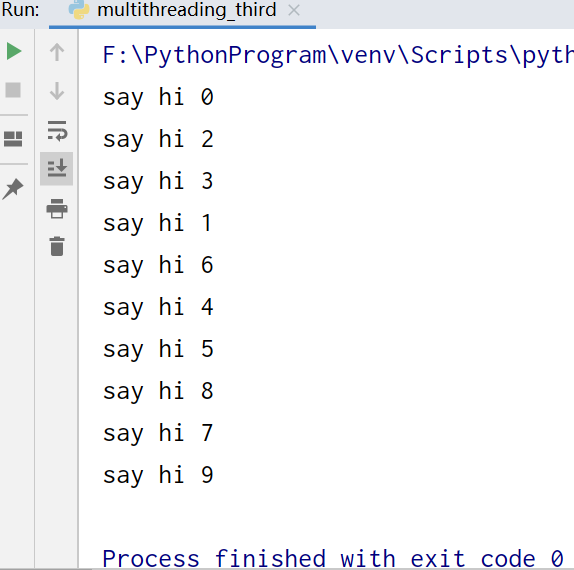
if \_\_name\_\_ == '\_\_main\_\_':

for i in range(10):

p = Process(target=foo, args=(i,))

p.start()

**运行结果：**

方法二：

创建子类，继承父类（Process），重写Run方法，实例化子类对象时，通过对象start开启子进程，子进程所执行的内容，就是在重写的run方法中的内容。在.start()方法时，会自动调用run方法。

class MyProcess(Process):

def \_\_init\_\_(self, arg):

super(MyProcess, self).\_\_init\_\_()

self.arg = arg

def run(self):

print('say hi', self.arg)

time.sleep(1)

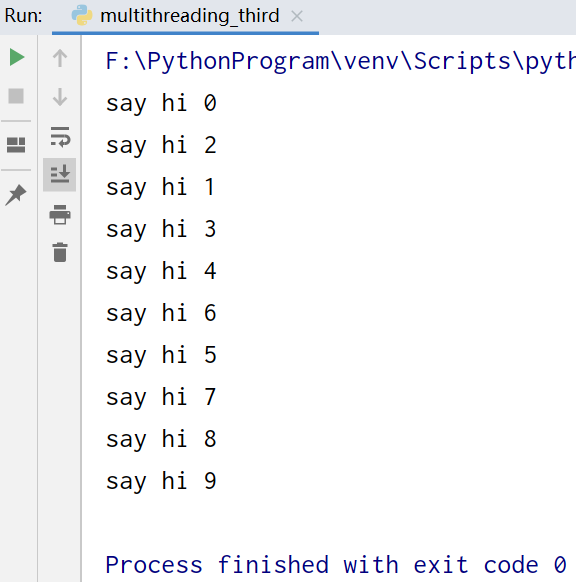
if \_\_name\_\_ == '\_\_main\_\_':

for i in range(10):

p = MyProcess(i)

p.start()

**运行结果：**



方法三：进程池（非阻塞式）-Pool方法实现

当需要创建进程的数量巨大时，就可以使用multiprocessing模块中的Pool方法。而用multiprocessing模块中的Process方法手动生成进程工作量会很大。

**运行结果：**

def task(task\_name):

print('开始做任务啦!',task\_name)

start = time.time()

# 使用sleep

time.sleep(random() \* 2)

end = time.time()

print('完成任务:{}! 耗时:{} ,进程ID:{}'.format(task\_name,(end-start),os.getpid()))

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

pool = Pool(5)

tasks = ['1','2','3','4','5','6','7']

for t in tasks:

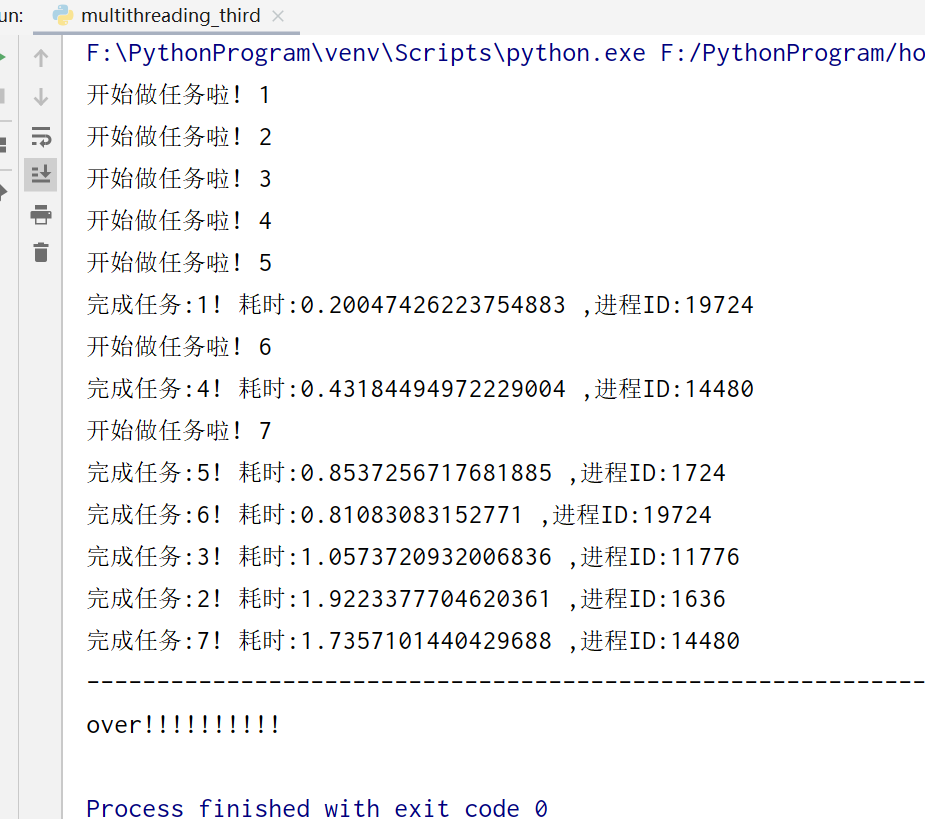
pool.apply\_async(task, args=(t,))

pool.close() # 关闭进程池，不允许继续添加进程

pool.join() # 等待进程池中的所有进程结束

print('------'\*10)

print('over!!!!!!!!!!')



**实验心得：**

本次实验通过调用python的multiprocessing内置工具包实现多进程的创建，实验中采用了Process和Pool的方法实现多进程创建，在Process开启的子进程中，会等待所有的子进程结束后，才会结束主进程，实现了程序的并发执行（可以发现进程的执行的顺序是随机的）；使用Pool类时，进程池内部维护一个进程序列，当使用时，则去进程池中获取一个进程，如果进程池序列中没有可供使用的进进程，那么程序就会等待，直到进程池中有可用进程为止。通过网络资料的查询和实验的实践过程，熟悉和掌握了Process类和Pool类的构造方法、具体的实例方法和各种属性的意义。