三：多任务编程

- 并发与并行
  - 并发：在一段时间内交替进行多个任务
  - 并行：在一段时间内同时一起执行多个任务
- 进程的介绍
  - 概念：进程是资源分配的最小单位，它是操作系统进行资源分配和条度运行的基本单位，通俗理解，一个正在运行的程序就是一个进城
  - 作用：提升程序运行效率
- 多进程完成多任务
  - 进程的创建步骤
    - 1：导入进程包
    - 2：通过进程创建进程对象
    - 3：启动进程

```
import multiprocessing
import time
def coding(num):
    for i in range(num):
        print('coding...')
        time.sleep(0.2)

def music(count):
    for i in range(count):
        print('music...')
        time.sleep(0.2)

if __name__=='__main__':
    coding_process=multiprocessing.Process(target=
coding,args=(3,))
    music_process=multiprocessing.Process(target=m
usic,kwargs={'count':2})
    coding_process.start()
    music_process.start()
```

    - 注意：字典中的key值必须和函数中的参数名一致，元组传参顺序一定要一致
- 获取进程编号
  - 作用：为了方便管理每个进城都是有自己的编号的，通过获取进程编号就可以快速区分不同的进城
  - 获取进程编号
    - 1.获取当前进程编号：getpid()
    - 2.获取当前父进程编号：getppid()
    -

```python
import multiprocessing
import time
import os
def coding(num):
    print('coding主进程：%d'%os.getppid())
    for i in range(num):
        print('coding>>>%d'%os.getpid())
        print('coding...')
        time.sleep(0.2)

def music(count):
    print('music主进程：%d'%os.getppid())
    for i in range(count):
        print('music>>>%d'%os.getpid())
        print('music...')
        time.sleep(0.2)

if __name__=='__main__':
    coding_process=multiprocessing.Process(target=
coding,args=(3,))
    music_process=multiprocessing.Process(target=m
usic,kwargs={'count':2})
    coding_process.start()
    music_process.start()
```

```
~ $ vim g.py
~ $ vim g.py
~ $ python g.py
coding主进程：30341
music主进程：30341
coding>>>30342
music>>>30343
coding...
music...
music>>>30343
music...
coding>>>30342
coding...
coding>>>30342
coding...
~ $
```

- 进程间不共享全局变量
  - 只是名字一样

```python
import multiprocessing
import time
my_list=[]
def write_data():
    for i in range(3):
        my_list.append(i)

def read_data():
    print(my_list)

if __name__=='__main__':
    write_process=multiprocessing.Process(target=w
rite_data)
    write_process.start()
    read_process=multiprocessing.Process(target=re
ad_data)
    read_process.start()
```

```
~ $ vim gg.py
~ $ python gg.py
[]
~ $
```

- 主进程子进程结束顺序
  - 主进程会等待子进程结束再结束
  - workprocess.daemon()守护主进程
  - workprocess .terminate()结束子进程
- 线程介绍
  - 多线程是python程序中实现多任务的一种方式
  - 线程是程序执行的最小单位
  - 同属一个进程的多个线程共享进程所有资源
- 多线程完成多任务
  - 线程的创建步骤
    - 导入线程模块
    - 通过线程类创建线程对象
    - 启动线程执行命令

```python
import threading
import time
def work():
    for i in range(3):
        print('working...')
        time.sleep(0.2)

def music():
    for i in range(3):
        print('music...')
        time.sleep(0.2)

if __name__=='__main__':
    coding_thread=threading.Thread(target=work)
    music_thread=threading.Thread(target=music)
    coding_thread.start()
    music_thread.start()
```

```
~ $ python g.py
working...
music...
working...
music...
working...
music...
~ $
```

  - 线程传参

```
import threading
import time
def work(num):
    for i in range(num):
        print('working...')
        time.sleep(0.2)

def music(count):
    for i in range(count):
        print('music...')
        time.sleep(0.2)

if __name__=='__main__':
    coding_thread=threading.Thread(target=work,arg
s=(3,))
    music_thread=threading.Thread(target=music,kwa
rgs={'count':3})
    coding_thread.start()
    music_thread.start()
```

- 主线程和子线程的结束顺序
  - 主线程会等待子线程完毕再结束
  - 设置守护主线程
    - music_thread=threading.Thread(target=music,daemon=True)
    - music_thread.setDaemon(True)

- 线程间执行顺序
  - 线程间执行是无序的

```
import threading
import time
def work(num):
    current_thread=threading.current_thread()
    print(current_thread)
    for i in range(num):
        print('working...')
        time.sleep(0.2)

def music(count):
    current_thread=threading.current_thread()
    print(current_thread)
    for i in range(count):
        print('music...')
        time.sleep(0.2)

if __name__=='__main__':
    coding_thread=threading.Thread(target=work,arg
s=(3,))
    music_thread=threading.Thread(target=music,kwa
rgs={'count':3})
    coding_thread.start()
    music_thread.start()
```

```
~ $ python g.py
<Thread(Thread-1 (work), started 489235260656)>
working...
<Thread(Thread-2 (music), started 489232004336)>
music...
working...
music...
working...
music...
~ $
```

- 线程间共享全局变量
  - 容易资源出错（来不及的问题）
  - 解决方法：同步，就是协同步调，按预定的先后次序执行，互斥锁
- 互斥锁的使用
  - 互斥锁介绍：对共享数据进行锁定，保证同一时刻只有一个线程操作。注意，互斥锁是多个线程一起抢，抢到锁的线程先执行，等锁使用完释放后，其它等待的线程再去执行
  - 使用：
    - 1.互斥锁创建
    - 2.上锁
    - 3.释放锁

```python
import threading as th
import time

g_num=0

def sum1():
    #上锁
    mutex.acquire()
    global g_num
    for i in range(1000000):
        g_num+=1
    #解锁
    mutex.release()
    print(g_num)

def sum2():
    #上锁
    mutex.acquire()
    global g_num
    for i in range(1000000):
        g_num+=1
    #解锁
    mutex.release()
    print(g_num)

if __name__=='__main__':
    #创建锁
    mutex=th.Lock()
    #线程创建
    sum1_thread=th.Thread(target=sum1)
    sum2_thread=th.Thread(target=sum2)
    #线程开始
    sum1_thread.start()
    sum2_thread.start()
```

- 死锁
  - 没有释放锁，造成程序无法完成
- 进程和线程对比
  - 1.关系对比
    - 线程是依附于进程，没有进程就没有线程

    - 一个进程默认提供一个线程，进程可以创建多个线程
- 2.区别对比
    - 进程之间不共享全局变量
    - 线程之间可以共享全局变量
    - 创建进程的资源开销比创建进程大
    - 进程是操作系统资源分配的基本单位，线程是cpu调度的基本单位
- 优缺点对比
    - 进程优缺点
        - 优点：可以用多核
        - 缺点：资源开销大
    - 线程优缺点
        - 优点：资源开销小
        - 缺点：不能使用多核