线程相关

线程概念：

线程，有时被称为轻量进程(LightweightProcess，LWP），是程序执行流的最小单元。一个标准的线程由线程 ID，当前指令指针(PC），寄存器集合和堆栈组成。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。一个线程可以创建和撤消另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有就绪、阻塞和运行三种基本状态。就绪状态是指线程具备运行的所有条件，逻辑上可以运行，在等待处理机；运行状态是指线程占有处理机正在运行；阻塞状态是指线程在等待一个事件（如某个信号量），逻辑上不可执行。每一个程序都至少有一个线程若程序只有一个线程，那就是程序本身。

其中：线程是程序中一个单一的顺序控制流程。进程内有一个相对独立的、可调度的执行单元，是系统独立调度和分派 CPU的基本单位指令运行时的程序的调度单位。在单个程序中同时运行多个线程完成不同的工作，称为多线程

多线程的概念：

多线程：多线程（英语：multithreading），是指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程，进而提升整体处理性能。具有这种能力的系统包括对称多处理机、多核心处理器以及芯片级多处理（Chip-levelmultithreading）或同时多线程（Simultaneousmultithreading）处理器。[1]在一个程序中，这些独立运行的程序片段叫作“线程”（Thread），利用它编程的概念就叫作“多线程处理（Multithreading）”。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程（台湾译作 “执行绪”），进而提升整体处理性能

线程的相关模块：

_thread模块：低级模块
threading模块：高级模块，对_thread进行了封装

线程间的数据共享：

多线程和多进程最大的不同在于，多进程中，同一个全局变量，每个子进程各自有一份拷贝，互不影响。而在多线程中所有变量都由线程共享，所以任何一个变量都可以被任意线程所修改。因此，多个线程同时改变一个变量，容易把内容改乱了

例如：

```
有一个全局的变量money
def changeMoney(n):
    global money
    money = money + n
    money = money - n
```

如果有多个线程同时执行changeMoney(n)这个函数，则全局变量有可能被改变
具体去查笔记

补充：进程之间的数据共享　在创建子进程时会将主进程的资源拷贝到子进程中，子进程单独有一份主进程中的数据。相互不影响

线程锁：

问题：两个线程同时一存一取，可能造成变量值的不对，我们必须保证一个线程在修改 money的时候，其他的线程一定不能修改

解决：使用线程锁

线程锁使用注意事项：在线程中对变量上锁，但是千万要注意释放锁（自己的锁自己放），否则会造成死锁

全局锁：

是个对象：ThreadLocal对象

概念：
每个线程独立的存储空间
每个Thread对它都可以读写属性操作，但是互不影响

进程vs线程

找对应的pdf文件

定时线程：

```
threading.Timer(time, run)
```

说明：time时间后启动线程执行run函数

线程通信：

给一个想要控制的语句上锁达到控制的目的

例子：

```
for i in range(10):
    #阻塞
  event.wait()
    #重置
  event.clear()

print("--------------" +
str(i))
```

通过event事件给for语句上锁

```
for i in range(10):
    event.set()
    time.sleep(1)
```

通过event.set()控制for语句的执行

线程中的生产者与消费者

自己去看笔记

线程 调度

需求：多个线程的执行都是各自独立的，这就出现了一个问题，无法控制多个线程之间的协调按顺序执行

解决：利用线程调度，通过各个线程之间的通信来实现这个需求
详情请查看笔记