''' 并发：看起来同时运行的就可以成为并发
并行：真的意义上的同时执行

PS： 并行肯定算并发
单核计算机能使进行并发，但不能实现并行

假设单核就是一个核，不算计算机CPU内核
单核多通道技术
切换CPU分两种情况
1.当一个程序遇到IO操作时，操作系统会剥夺程序的CPU的执行权限
2.当一个程序长时间占用CPU时，操作系统也会剥夺该程序的CPU执行权限'''

''' # 进程理论
进程（Process）是什么？
是计算机中的程序关于某数据集合上的一次运行活动
是系统进行资源分配和调度的基本单位
是操作系统结构的基础

进程是正在运行的程序的实例（an instance of a computer program that is being executed）

程序与进程的区别：
程序就是一推躺在硬盘上的代码（死的）
进程表示程序正在运行的过程（是活的）

进程的特点：
动态性：进程的实质是程序在多道程序系统中的一次执行过程，进程是动态产生，动态消亡的
并发性：任何进程都可以同其他进程一起并发执行
独立性：进程是一个能独立运行的基本单位，同时也是系统分配资源和调度的独立单位
异步性：由于进程间的相互制约，使进程具有执行的间断性，按各自独立的、不可预知的速度向前推进
结构特征：进程由程序、数据和进程控制块三部分组成

多个不同的进程可以包含相同的程序：一个程序在不同的数据集里就构成不同的进程
能得到不同的结果，但是执行过程中，程序不能发生改变

进程调度
先来先服务
先来先服务（FCFS）调度算法是一种最简单的调度算法
该算法既可用于作业调度，也可用于进程调度
FCFS算法比较有利于长作业（进程），而不利于短作业（进程）
由此可知，本算法适合于CPU繁忙型作业，而不利于I/O繁忙型的作业（进程）

短作业优先算法
短作业（进程）优先调度算法（SJ/PF）是指对短作业或短进程优先调度的算法
该算法既可用于作业调度，也可用于进程调度，但其对长作业不利
不能保证紧迫性作业（进程）被及时处理
并且作业的长短只是被估算出来的

时间片轮转法+多级反馈队列
时间片轮转(Round
Robin，RR)法的基本思路是让每个进程在就绪队列中的等待时间与享受服务的时间成比例
在时间片轮转法中，需要将CPU的处理时间分成固定大小的时间片
如果一个进程在被调度选中之后用完了系统规定的时间片，但又未完成要求的任务
则它自行释放自己所占有的CPU而排到就绪队列的末尾，等待下一次调度
同时，进程调度程序又去调度当前就绪队列中的第一个进程

显然，轮转法只能用来调度分配一些可以抢占的资源
这些可以抢占的资源可以随时被剥夺，而且可以将它们再分配给别的进程

CPU是可抢占资源的一种，但打印机等资源是不可抢占的
由于作业调度是对除了CPU之外的所有系统硬件资源的分配
其中又包含有不可抢占资源，所以作业调度不使用轮转法

在轮转法中，时间片长度的选取非常重要
首先，时间片长度的选择会直接影响到系统的开销和响应时间
如果时间片长度过短，则调度程序抢占处理机的次数增多
这将使进程上下文切换次数也大大增加，从而加重系统开销
反过来，如果时间片长度选择过长
例如，一个时间片能保证就绪队列中所需执行时间最长的进程能执行完毕
则轮转法变成了先来先服务法
时间片长度的选择是根据系统对响应时间的要求和就绪队列中所允许最大的进程数来确定的

在轮转法中，加入到就绪队列的进程有3种情况：
第一种是分给它的时间片用完，但进程还未完成，回到就绪队列的末尾等待下次调度去继续执行
第二种情况是分给该进程的时间片并未用完
只是因为请求I/O或由于进程的互斥与同步关系而被阻塞。当阻塞解除之后再回到就绪队列
第三种情况就是新创建进程进入就绪队列

如果对这些进程区别对待，给予不同的优先级和时间片
从直观上看，可以进一步改善系统服务质量和效率
例如，可把就绪队列按照进程到达就绪队列的类型和进程被阻塞时的阻塞原因分成不同的就绪队列
每个队列按FCFS原则排列，各队列之间的进程享有不同的优先级，但同一队列内优先级相同
这样，当一个进程在执行完它的时间片之后、从睡眠中被唤醒、被创建之后，将进入不同的就绪队列

多级反馈队列
应设置多个就绪队列，并为各个队列赋予不同的优先级
第一个队列的优先级最高，第二个队列次之，其余各队列的优先权逐个降低
该算法赋予各个队列中进程执行时间片的大小也各不相同
在优先权愈高的队列中，为每个进程所规定的执行时间片就愈小
当一个新进程进入内存后，首先将它放入第一队列的末尾，按FCFS原则排队等待调度
当轮到该进程执行时，如它能在该时间片内完成，便可准备撤离系统
如果它在一个时间片结束时尚未完成，调度程序便将该进程转入第二队列的末尾
再同样地按FCFS原则等待调度执行
如果它在第二队列中运行一个时间片后仍未完成，再依次将它放入第三队列，……，
如此当一个长作业(进程)从第一队列依次降到第n队列后，在第n 队列便采取按时间片轮转的方式运行
仅当第一队列空闲时，调度程序才调度第二队列中的进程运行
仅当第1～(i-1)队列均空时，才会调度第i队列中的进程运行

如果处理机正在第i队列中为某进程服务时，又有新进程进入优先权较高的队列(第1～(i-1)中的任何一个队列)
则此时新进程将抢占正在运行进程的处理机，即由调度程序把正在运行的进程放回到第i队列的末尾
把处理机分配给新到的高优先权进程

多级反馈队列调度算法则不必事先知道各种进程所需的执行时间，而且还可以满足各种类型进程的需要
因而它是目前被公认的一种较好的进程调度算法

进程运行三状态图

#两对重要概念
同步和异步（描述任务的提交方式）：

同步：任务提交后，原地等待结果，等待过程中不做任何事（程序表现结果为卡住）
异步：任务提交之后不原地等待任务反馈结果，做其他事情
提交结果获取：任务返回结果有一个异步回调机制自动处理
阻塞和非阻塞（描述程序/进程的运行状态）：
阻塞：阻塞态
非阻塞：就绪态和运行态'''

###开启进程的两种方式

线程理论
###什么是线程？
进程：资源单位（仅仅是在内u车内空间中开辟一个独立空间）
线程：执行单位（真正的CPU执行的东西，线程指的是代码执行的过程，
执行代码所需要的资源都找所在的进程索要）

操作系统比作工厂，进程是车间，线程是流水线
每一次进程肯定自带一个线程
同一个进程下的不同线程数据共享

###为何要有线程？
开设进程：1.申请内存 2.拷贝代码（linux）
开合线程：在一个进程内开设多个线程，无需申请内存空间，消耗小

###开启线程的两种方式
```
import time
from threading import Thread

def task(name):
    print('%s is running' % name)
    time.sleep(1)
    print('%s is over' % name)
'''
开启线程不需要在main下面执行代码，直接书写
但是我们习惯性还是将启动命令写在main下面
'''
if __name__ == '__main__':
    t = Thread(target=task, args=('alex',))
    t.start()
    print('main')
```

```python
class MyThead(Thread):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def run(self) -> None:
        print('hello')
        time.sleep(3)
        print('out')

if __name__ == '__main__':
    p = MyThead('alex')
    p.start()
    print('main')
```

###GIL解释器锁

在CPython中，GIL是一把互斥锁，用来阻止同一个进程下的多线程的同时执行

主要是因为CPython中的内存管理（垃圾回收机制）不是安全的

1.应用计数

2.标记清除

3.分代回收

同一个进程下的多个线程无法利用多核优势

单任务10S（都有应用场景！！！）

单核：四个任务（计算密集型）

多进程：额外开销

多线程：节省开销

多核：四个任务（计算密集型）

多进程：10S

多线程：40S

多核：（IO密集型）

多进程：相对浪费资源

多线程：相对节省资源

1.GIL不是Python的特点而是CPython解释器的特点
2.GIL保证解释器级别的数据安全
3.阻止同一个进程下的多线程的同时执行
4.针对不同的数据，仍然需要加不同的锁处理
5.解释型语言的通病：同一个进程下的多线程不能利用多和优势

多进程和多线程都有各自优势
我们通常写项目时多进程下开设多线程
这样的话既可以利用多核也可以节省资源消耗

死锁现象（了解）
递归锁：可以连续地被acquire&release

但是只能被第一个人抢到，内部有一个计数器，被acquier一次就计数加一

释放一次就计数减一，只有在计数为0时，才可以被下一个人抢到

信号量（了解）

如果将互斥锁比作一个坑位厕所

那么信号量就相当于公共厕所

Event事件（了解）
一些进程/线程需要等待另外一些进程/线程完毕之后才能完成，类似于发射信号

线程Q，见代码

###池的概念
池是用来保证计算机安全的情况下最大限度的利用计算机
他降低了程序的效率，但是保证了计算机硬件安全，从而让你的程序可正常运行

###进程池和线程池（掌握）
无论是开设进程也好还是线程也好，都要消耗资源
无限开放，计算机硬件不够
见代码

###协程：单线程下实现并发（多道技术的应用=切换+保存状态），由程序员想象，实际上不存在
CPU切换：1.程序遇到IO
2.程序长时间调用
程序员自己在代码成面上完成切换，一旦遇到IO后，完成切换
这样CPU感觉就是程序在一直运行，没有IO
从而提升程序运行效率

切换：　　IO切换-----》提升效率
没有IO切---》降低效率

保存状态：保存上依次执行的状态，下一次接着上一次操作往后执行
yield

gevent模块（了解）

基本sql语句
针对库的增删改查（文件夹）
create database 库名;
create

针对表的增删改查（文件）

针对数据的增删改查（一行行数据）