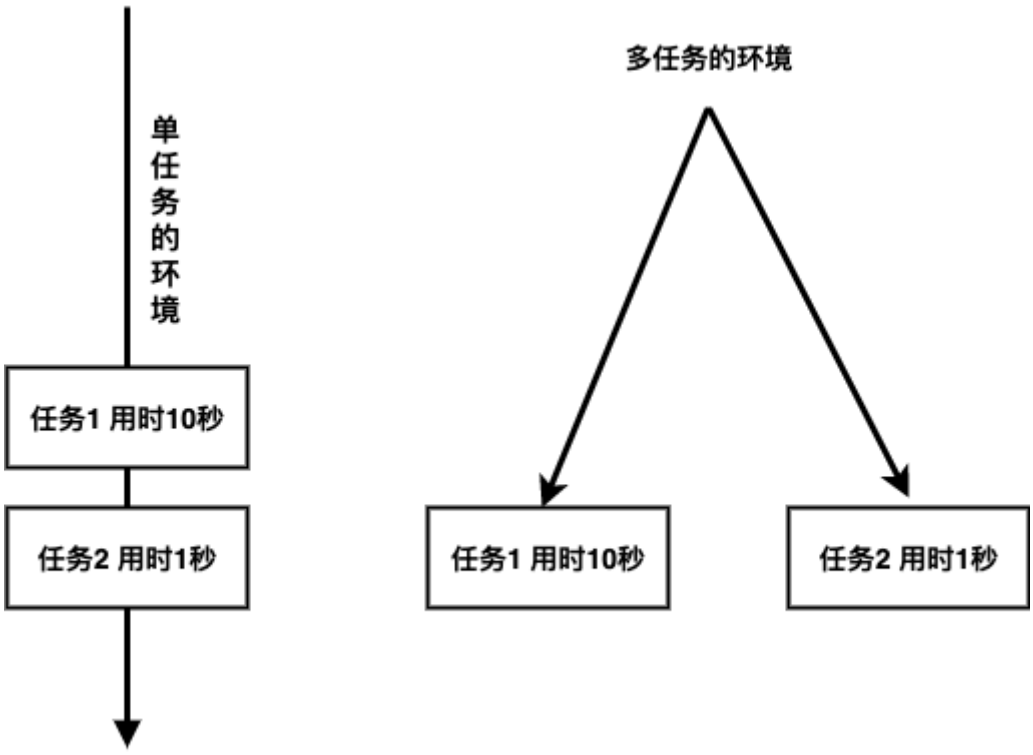
多任务

什么叫“多任务”呢？简单地说，就是操作系统可以同时运行多个任务。

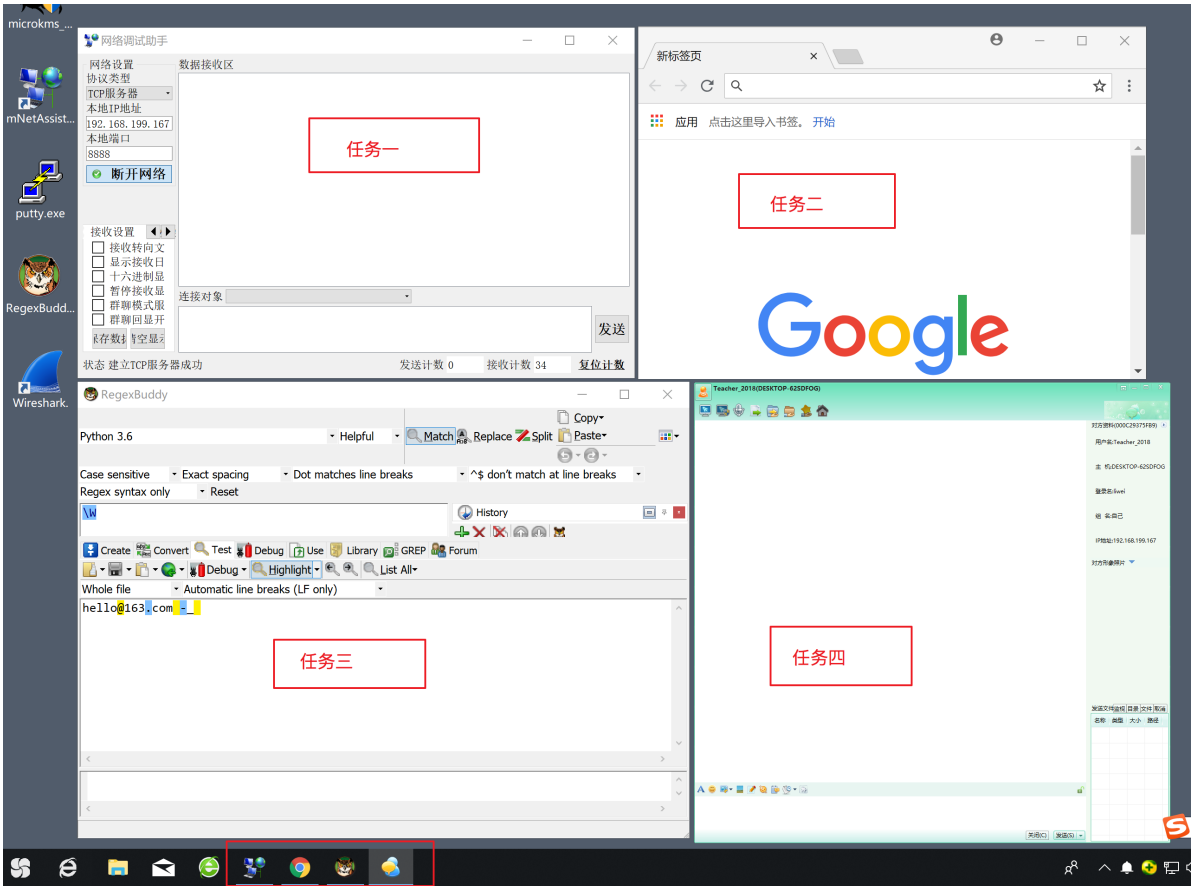
现在，多核CPU已经非常普及了，但是，即使过去的单核CPU，也可以执行 多任务。

多任务的优势



多任务表现形式

window下打开任务管理器可以很清晰看到多个进程在同时执行任务，qq、微信等都是已进程的形式寄存在window下。大多我们在写一些控制台程序真正执行的时候都是以进程调度。



Python默认是单任务

接下来我们使用python代码来模拟“唱歌跳舞”这件事情

```python
import time


def sing():
    """唱歌函数"""
    for i in range(3):
        print("正在唱歌..")
        time.sleep(0.5)


def dance():
    """跳舞函数"""
    for i in range(3):
        print("正在跳舞..")
        time.sleep(0.5)


if __name__ == '__main__':

    sing()  # 正在唱歌
    dance() # 正在跳舞
```

!!!note

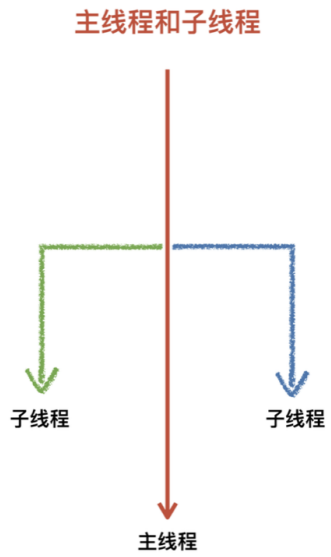
```
* 很显然刚刚的程序并没有完成唱歌和跳舞同时进行的要求
* 如果想要实现“唱歌跳舞”同时进行，那么就需要一个新的方法，叫做：多任务
```

线程

概念

线程，**可简单理解为是程序执行的一条分支**，也是程序执行流的最小单元。

线程 **是被CPU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。



主线程:

当一个程序启动时，就有一个进程被操作系统（OS）创建，与此同时一个线程也立刻运行，该线程通常叫做程序的 **主线程**.

简而言之：程序启动就会创建一个主线程。

主线程的重要性有两方面：

- 是产生其他子线程的线程；
- 通常它必须最后完成执行比如执行各种关闭动作。

子线程:

可以看做是程序执行的一条分支，当子线程启动后会和主线程一起同时执行

threading模块

python的thread模块是比较底层的模块，python的threading模块是对thread做了一些包装的，可以更加方便的被使用.

核心方法

- threading.Thread(target=函数名) threading模块的Thread类 创建子线程对象
- 线程对象.start() 启动子线程

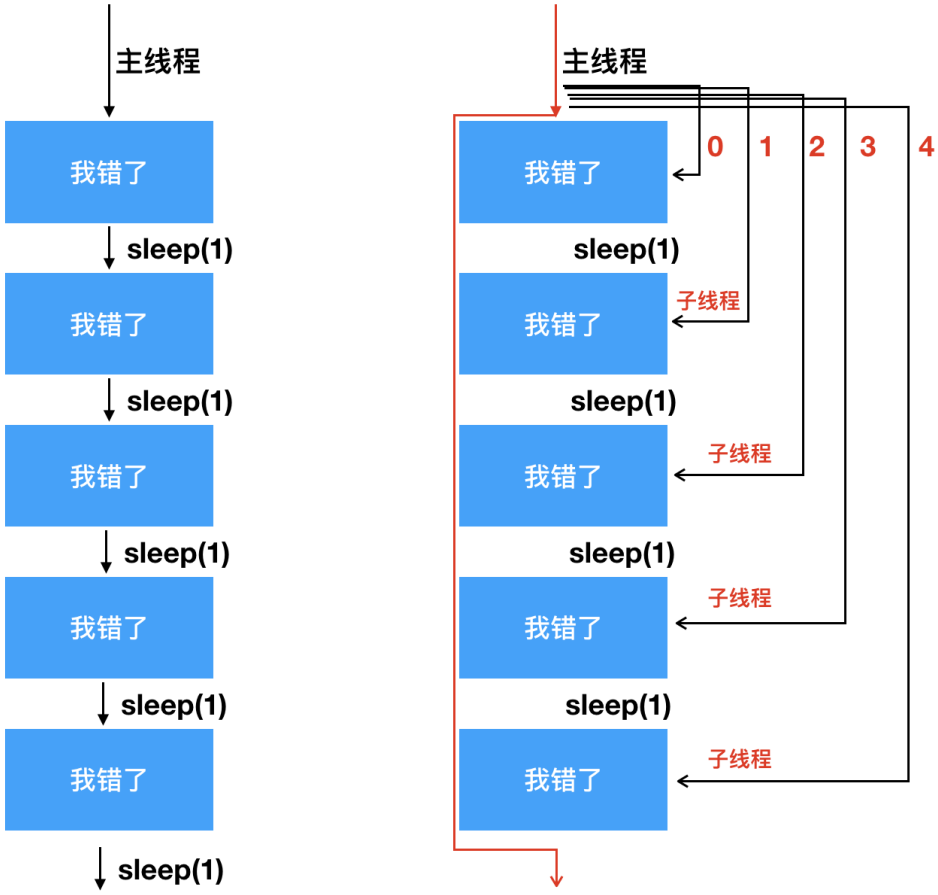
代码实现

```python
import time
import threading

def say_sorry():
    print("亲爱的，我错了，我能吃饭了吗？")
    time.sleep(1)


if __name__ == '__main__':

    for i in range(5):
        # 创建子线程的的方法
        # 1、导入 threading 模块
        # 2、使用 threading.Tread() 方法，创建子线程对象
        # threading.Thread(target=函数名，args=(参数列表，元组))
        t1 = threading.Thread(target=say_sorry)
        # 3、使用子线程对象调用start() 方法，可以开启子线程（子线程将同时执行）
        t1.start()

    print("我是主线程")
```

运行结果:

说明

- 可以明显看出使用了多线程并发的操作，花费时间要短很多
- 当调用start()时，才会真正的创建线程，并且开始执行
- 每个线程都有一个唯一标示符，来区分线程中的主次关系
- 主线程:mainThread,Main函数或者程序主入口，都可以称为主线程
- 子线程:Thread-x 使用 threading.Thread() 创建出来的都是子线程
- 线程数量：主线程数 + 子线程数


主线程
我错了
sleep(1)
我错了
sleep(1)
我错了
sleep(1)
我错了
sleep(1)
我错了
主线程
我错了
0
1
2
3
4
sleep(1)
我错了
子线程
sleep(1)
我错了
子线程
sleep(1)
我错了
子线程
sleep(1)
我错了
子线程

GUI实现


下位机控制
LED控制
打开LED
关闭LED
开关LED
电机控制
发送
转速显示
转速(圈/秒) 0.0
蜂鸣器控制
打开蜂鸣器
关闭蜂鸣器
开关蜂鸣器
OLED控制
发送

```
def __init_third(self, layout):
        rpm_group = QGroupBox('转速显示')
        rpm_layout = QFormLayout(rpm_group)
        layout.addWidget(rpm_group)

        self.__init_rpm_ui(rpm_layout)

    def __init_rpm_ui(self, layout):
        self.__lb_rpm = QLabel()

        layout.addRow('转速(圈/秒)', self.__lb_rpm)
```