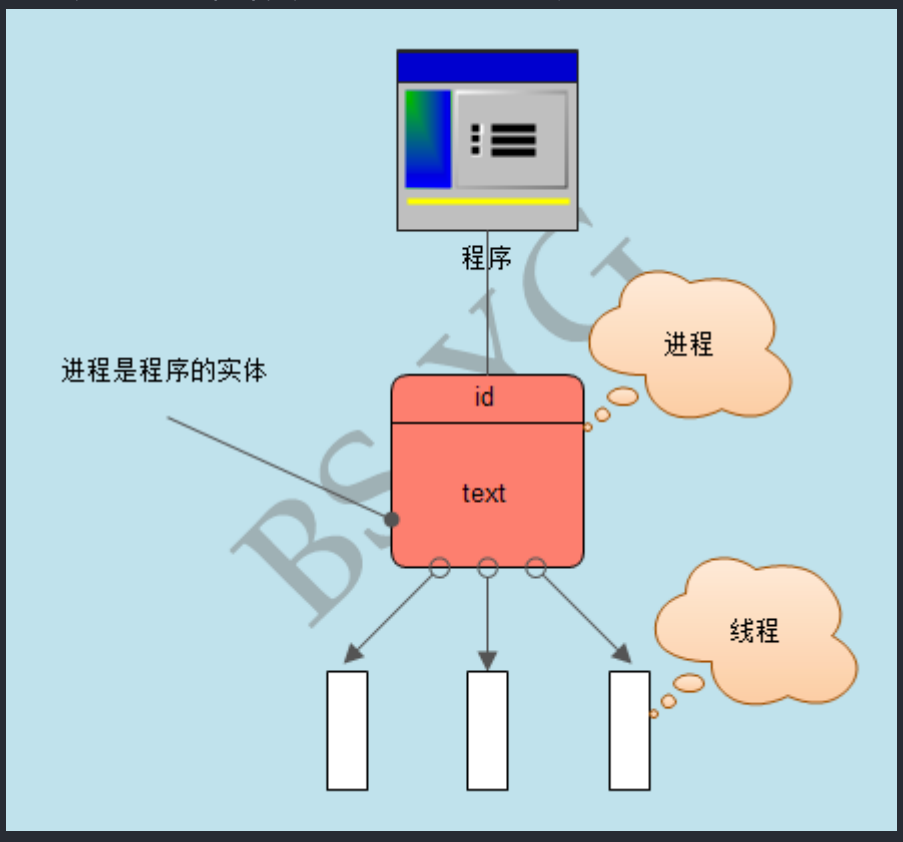
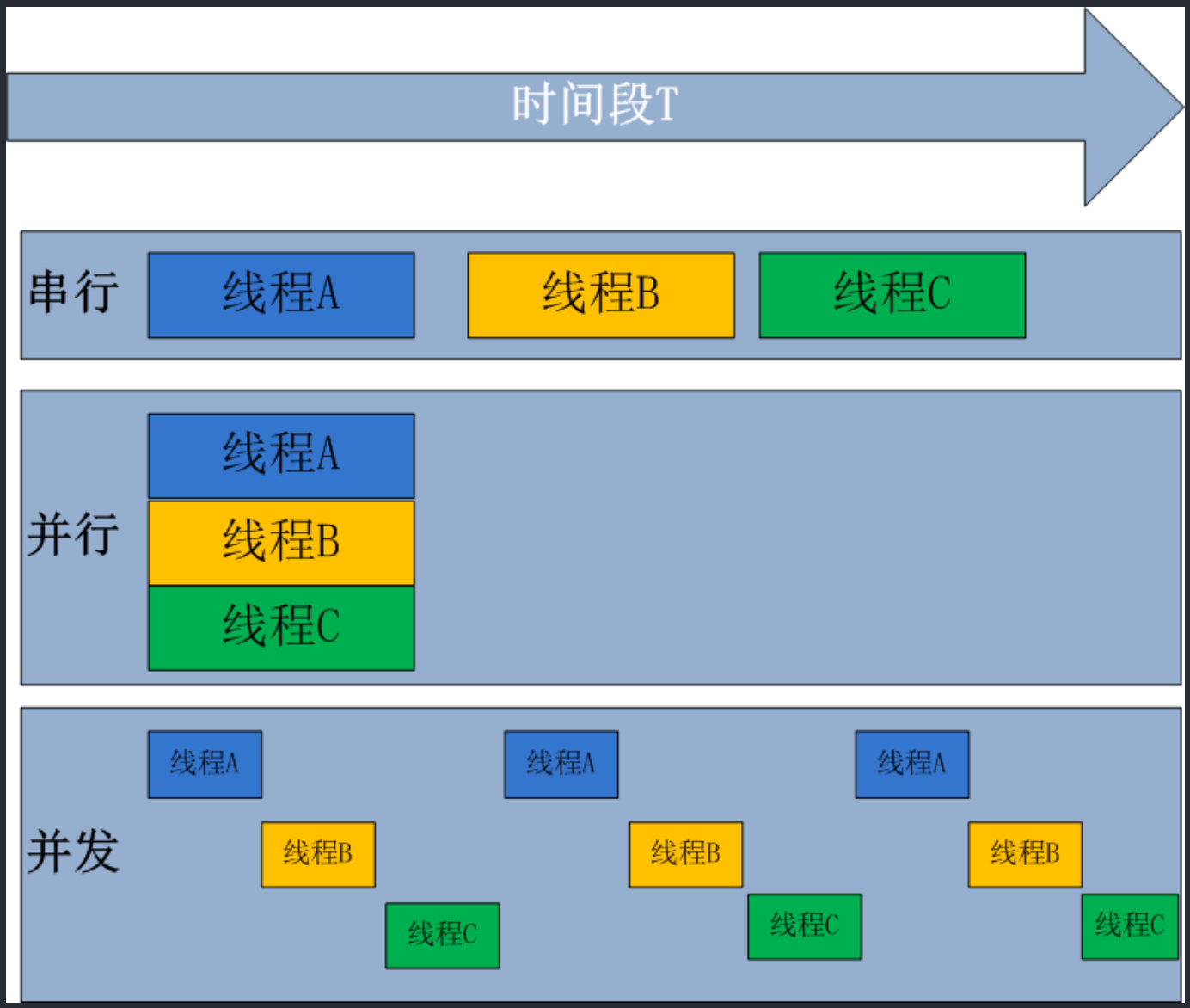
1. 进程与线程的概念

1. 进程：
    1. 定义：进程就是一个程序在一个数据集上的一次动态执行过程
    2. 组成：由程序、数据集、进程控制块三部分组成
        - 程序用来描述进程要完成哪些功能以及如何完成
        - 数据集则是程序在执行过程中所需要使用的资源
        - 进程控制块用来记录进程的外部特征，描述进程的执行变化过程，系统可以利用它来控制和管理进程，它是系统感知进程存在的唯一标志
    3. 进程与程序的区别：
        - 程序仅仅只是一堆代码而已，而进程指的是程序的运行过程
2. 线程：
    1. 定义：线程也叫轻量级进程，它是一个基本的CPU执行单元，也是程序执行过程中的最小单元，由线程ID、程序计数器、寄存器集合和堆栈共同组成
        - 线程的引入减小了程序并发执行时的开销，提高了操作系统的并发性能。线程没有自己的系统资源
3. 进程与线程的关系
    1. 进程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。或者说进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位
    2. 线程则是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位。
    3. 进程和线程的关系:
        1. 一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。
        2. 资源分配给进程，同一进程的所有线程共享该进程的所有资源。
        3. CPU分给线程，即真正在CPU上运行的是线程。



2. 并行和并发

1. 并行处理
    1. 是计算机系统中能同时执行两个或更多个处理的一种计算方法
    2. 可同时工作于同一程序的不同方面，并行处理的主要目的是节省大型和复杂问题的解决时间
2. 并发处理
    1. 指一个时间段中有几个程序都处于已启动运行到运行完毕之间，且这几个程序都是在同一个处理机(CPU)上运行，但任一个时刻点上只有一个程序在处理机(CPU)上运行
3. 区别：
    1. 并发的关键是你有处理多个任务的能力，不一定要同时。
    2. 并行的关键是你有同时处理多个任务的能力。



3. 同步与异步

1. 同步：就是在发出一个功能调用时，在没有得到结果之前，该调用就不会返回
2. 异步：当一个异步功能调用发出后，调用者不能立刻得到结果。当该异步功能完成后，通过状态、通知或回调来通知调用者
3. 注意：如果异步功能用状态来通知，那么调用者就需要每隔一定时间检查一次，效率就很低（有些初学多线程编程的人，总喜欢用一个循环去检查某个变量的值，这其实是一 种很严重的错误）。如果是使用通知的方式，效率则很高，因为异步功能几乎不需要做额外的操作。至于回调函数，其实和通知没太多区别

4. multiprocessing模块介绍

- 模块介绍：
  1. multiprocessing模块用来开启子进程，并在子进程中执行我们定制的任务（比如函数），该模块与多线程模块threading的编程接口类似
  2. multiprocessing模块的功能众多：支持子进程、通信和共享数据、执行不同形式的同步，提供了Process、Queue、Pipe、Lock等组件
  3. 与线程不同，进程没有任何共享状态，进程修改的数据，改动仅限于该进程内

5. Process类的介绍

1. 创建进程的类

```
Process([group [, target [, name [, args [, kwargs]]]]])，由该类实例化得到的对象，表示一个子进程中的任务（尚未启动）

强调：
1. 需要使用关键字的方式来指定参数
2. args指定的为传给target函数的位置参数，是一个元组形式，必须有逗号
```

2. 参数介绍

```
group参数未使用，值始终为None
target表示调用对象，即子进程要执行的任务
args表示调用对象的位置参数元组，args=(1,2,'egon',)
kwargs表示调用对象的字典,kwargs={'name':'egon','age':18}
name为子进程的名称
```

3. 方法介绍

```
p.start()：启动进程，并调用该子进程中的p.run()
p.run():进程启动时运行的方法，正是它去调用target指定的函数，我们自定义类的类中一定要实现该方法


p.terminate():强制终止进程p，不会进行任何清理操作，如果p创建了子进程，该子进程就成了僵尸进程，使用该方法需要特别小心这种情况。如果p还保存了一个锁那么也将不会被释放，进而导致死锁
p.is_alive():如果p仍然运行，返回True

p.join([timeout]):主线程等待p终止（强调：是主线程处于等的状态，而p是处于运行的状态）。timeout是可选的超时时间，需要强调的是，p.join只能join住start开启的进程，而不能join住run开启的进程
```

4. 属性介绍

```
p.daemon：默认值为False，如果设为True，代表p为后台运行的守护进程，
当p的父进程终止时，p也随之终止，并且设定为True后，p不能创建自己的新进程，必须在p.start()之前设置

p.name:进程的名称

```

p.pid：进程的pid

p.exitcode:进程在运行时为None、如果为-N，表示被信号N结束(了解即可)

p.authkey:进程的身份验证键,默认是由os.urandom()随机生成的32字符的字符串。

这个键的用途是为涉及网络连接的底层进程间通信提供安全性，这类连接只有在具有相同的身份验证键时才能成功（了解即可）

6. Process类的使用

1. 注意：在windows中Process()必须放到# if **name** == '**main**':下

```
if __name__ == "__main__"
since statements inside this if-statement will not get called upon import.
由于Windows没有fork，多处理模块启动一个新的Python进程并导入调用模块。
如果在导入时调用Process（），那么这将启动无限继承的新进程（或直到机器耗尽资源）。
这是隐藏对Process（）内部调用的原，使用if __name__ == “__main __”，这个if语句中的语句将不会在导入时被调用。
```

2. Process 类的调用开启进程

```
import time
from multiprocessing import Process

def piao(name):
    print('%s piaoing'% name)
    time.sleep(2)
    print('%s piao end'% name)

if __name__ == '__main__':

    p1 = Process(target=piao, args=('egon',))
    p2 = Process(target=piao, args=('alex',))
    p3 = Process(target=piao, args=('wupeiqi',))
    p4 = Process(target=piao, args=('yuanhao',))

    p1.start()
    p2.start()
    p3.start()
    p4.start()

    print('主线程')

# 主线程
# egon piaoing
# wupeiqi piaoing
# alex piaoing
# yuanhao piaoing
# egon piao end
# alex piao end
```

```
# wupeiqi piao end
# yuanhao piao end
```

3. Process 类的继承开启进程

```
import time
from multiprocessing import Process

class piao(Process):
    def __init__(self, name):
        super().__init__()
        self.name = name
    def run(self):
        print('%s piaoing'%self.name)
        time.sleep(2)
        print('%s piao end'%self.name)

if __name__ == '__main__':

    p1=piao('egon')
    p2=piao('alex')
    p3=piao('wupeiqi')
    p4=piao('yuanhao')

    p1.start()
    p2.start()
    p3.start()
    p4.start()

    print('主线程')

# 主线程
# egon piaoing
# alex piaoing
# yuanhao piaoing
# wupeiqi piaoing
# egon piao end
# alex piao end
# yuanhao piao end
# wupeiqi piao end
```

4. 进程之间的内存空间是隔离的

```
from multiprocessing import Process

global n
n=100

def foo():
    n=200
    print('子程序',n)

```

```python
if __name__ == '__main__':
    p1=Process(target=foo)
    p1.start()
    print('主程序',n)

# 主程序 100
# 子程序 200
```

7. 多进程实现并发形式的socket通信

1. server端

```python
import socket
from multiprocessing import Process

server=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
ip_port=('127.0.0.1',9000)
BUFSIZE=1024

server.bind(ip_port)
server.listen(5)

def talk(conn,addr):
    while True:
        try:
            msg=conn.recv(BUFSIZE)
            print(msg.decode('utf-8'))
            conn.send(msg.upper())
        except Exception:
            break

    conn.close


if __name__ == '__main__':
    while True:
        conn,addr=server.accept()
        t=Process(target=talk,args=(conn,addr))
        t.start()

    server.close
```

2. client端

```python
import socket

client=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
ip_port=('127.0.0.1',9000)
BUFSIZE=1024

```

```
client.connect(ip_port)

while True:
    msg=input('>>>: ').strip()
    if not msg:
        continue
    client.send(msg.encode('utf-8'))
    msg_recv=client.recv(BUFSIZE)
    print(msg_recv.decode('utf-8'))

client.close
```

8. Process对象的join方法

1. 示例1

```
import time
from multiprocessing import Process

class Piao(Process):

    def __init__(self,name):
        super().__init__()
        self.name = name

    def run(self):
        print('%s is piaoing'%self.name)
        time.sleep(2)
        print('%s piao end'%self.name)

if __name__ == '__main__':
    p=Piao('egon')
    p.start()
    p.join(1)
    print('主程序')
```

2. 示例2

```
import time
from multiprocessing import Process

def piao(name):
    print('%s piaoing'% name)
    time.sleep(3)
    print('%s piao end'% name)


if __name__ == '__main__':

    p1 = Process(target=piao, args=('egon',))
    p2 = Process(target=piao, args=('alex',))
```

```
    p3 = Process(target=piao, args=('wupeiqi',))
    p4 = Process(target=piao, args=('yuanhao',))

    p1.start()
    p2.start()
    p3.start()
    p4.start()

    p1.join()
    p2.join()
    p3.join()
    p4.join()

    #或可以改写
    p_l=[p1, p2, p3, p4]
    for p in p_l:
        p.start()

    for p in p_l:
        p.join()
    #
    print('主线程')
```

9. Process补充(terminate/is_alive/name/id)

1. terminate与is_alive

```
from multiprocessing import Process
import time

class Piao(Process):

    def __init__(self, name):
        super().__init__()
        self.name = name

    def run(self):
        print('%s is piaoing'% self.name)
        time.sleep(2)
        print('%s piao end'% self.name)
if __name__ == '__main__':
    p=Piao('egon')
    p.start()

    p.terminate()       #关闭进程,不会立即关闭,所以is_alive立刻查看的结果可能还是存活
    print(p.is_alive()) #结果为Ture

    time.sleep(1)

    print(p.is_alive()) #结果为Flase
```

2. name与pid

```python
from multiprocessing import Process
import time

class Piao(Process):

    def __init__(self,name):
        self.name = name
        super().__init__()

    def run(self):
        print('%s is piaoing'% self.name)
        time.sleep(2)
        print('%s piao end'%self.name)

if __name__ == '__main__':

    p=Piao('egon')
    p.start()
    print('PID',p.pid)

# PID 57104
# Piao-1 is piaoing
# Piao-1 piao end
```

10. 守护进程

1. 主进程创建守护进程
    1. 守护进程会在主进程代码执行结束后就终止
    2. 守护进程内无法再开启子进程,否则抛出异常：AssertionError: daemonic processes are not allowed to have children
    3. 进程之间是互相独立的，主进程代码运行结束，守护进程随即终止
2. 示例代码

```python
from multiprocessing import Process
import time

class Piao(Process):

    def __init__(self, name):
        super().__init__()
        self.name=name

    def run(self):
        print('%s is piaoing'% self.name)
        time.sleep(2)
        print('%s piao end'% self.name)

if __name__ == '__main__':
    p=Piao('egon')
```

```
    p.daemon=True
    p.start()
    print('主程序')
```