第三周 (3C)

多进程同步

os.fork() (Linux/Mac)
multiprocessing.Process()

↓

```
p = Process(target=f,
            args=('john',))
p.start()
p.join()
```

p.join() → 子进程结束后父才结束

调试
multiprocessing.active_children()
multiprocessing.cpu_count()

类方法创建多进程

```
class NewProcess(Process):
  def __init__(self, name):
      ...
  def run(self):   //与target=xxx一样
      ...

for i in range(2):
    p=NewProcess(i)
    p.start()
```

多进程通信.（3C）

① 变量无法共享
② 资源竞争.

global num ⊗ (不可以访问到的)

① 队列.

multiprocessing.Queue

进程安全的
可同时读写

```
q = Queue()
p = Process(target=f, args=(q,))
q.put(...)
q.get()
```

② 管道. multiprocessing.Pips

```
parent_conn, child_conn = Pipe()
parent_conn.recv()
child_conn.send()
```

③共享内存.

mutiprocess. Array .Value.

```
num = Value('d', 0.0)
arr = Array('i', range(10))
p=Process(target=f, args=(num,arr))
```

锁机制:

```
def f(l):
    l.acquire()
    ...
    l.release()

l=mp.Lock()
mp.Process(target=f,args=(l,))
```

进程池

mp.pool.Pool.

```
p=Pool(4)
for i in range(10):
    p.apply_async(run,args=(i,))
p.close()
p.join()
p.trminate()
```

```
with Pool(processes=4) as pool:
    result=pool.apply_async(f,(10,))
    print(result.got(timeout=1))
```

```
with ......                :
    pool.map(f,range(10))
    it=pool.imap(f,range(10)).
```

多线程

多线程 → 内存空间可共享.

(发起方) ⟷ (被发起方)

阻塞 ╳ 同步

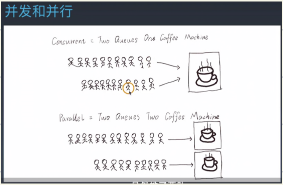
非阻塞 ╳ 异步

{ 多线程 → 同cpu运行

多进程 → 多cpu.

协程：

进程切换的调度.



并发和并行

Concurrent = Two Queues One Coffee Machine

Parallel = Two Queues Two Coffee Machine

```
t = threading.Thread(target=run, args=("x",))
t.start()
```

```
class MyThread(threading.Thread):
    def __init__(self, n):
        super().__init__()
        self.n = n
    def run(self):
        ...
```

```
thre1. is_alive()
     .getName()
     .join()
     .setDeamon(True)
```

线程锁：

```
l = threading.Lock()
l.aquire(1):
l.release()
```

threading.RLock() → 可嵌套

线程队列.

```
q = queue.Queue(5)
q.put(...)
q.get()
```

q.task_done() → 提示q.join()是否停止阻塞.

```
q.qsize()
q.empty() q.full()
```

```
queue.PriorityQueue()
 q.put((1, "work")).
```

线程池

```
from concurrent.features import
                    ThreadPoolExecutor
```

~~pool = ThreadPool(4)~~

~~pool.map(requests.get, urls)~~

```
with ThreadPoolExecutor(3) as executor:
   executor.submit(func, seed, 3)
           .map(func, seed)
```

全局解释锁.

GIL(Global Interpreter Lock)

一个进程有一个GIL锁.

拿到GIL锁才可以使用CPU.

CPython是伪并发,同时只有一个线程运行.

(CPU密集型应运效率与单线程没区别

IO密集型多线程更快)