Многопроцессорность в Python. Библиотека multiprocessing

Лукьянов Павел

МГУ им. М.В.Ломоносова

7 декабря 2018

Мультипроцессорность

Мультипроцессорность - это выполнение множественных параллельных программных процессов в системе в противоположность выполнению одного процесса в любой момент времени.
Многопроцессная многозадачность реализована в Python благодаря модулю multiprocessing

class Process

В multiprocessing процессы порождаются созданием объекта класса Process и последующим вызовом метода start()

```
import os
from multiprocessing import Process
def f(n):
    proc = os.getpid()
    print(n % 2)
if __name__ == '__main__':
    s = [1, 10, 15, 23, 46]
    procs = []
    for x in s:
        proc = Process(target=f, args=(x,))
        procs.append(proc)
        proc.start()
    for proc in procs:
        proc.join()
```

class Process

Если вам нужно остановить процесс, вы можете вызвать метод terminate().
Если же мы хотим получить информацию о текущем процессе, то нужно воспользоваться

```
current_process from multiprocessing
```

Например, если нас интересует имя процесса, то используем

```
current_process().name
```

Lock

Модуль multiprocessing поддерживает замки

```
from multiprocessing import Process, Lock

def f(lock, i):
    lock.acquire()
    print(i)
    lock.release()

if __name__ == '__main__':
    lock = Lock()
    for i in range(5):
        proc = Process(target = f, args=(lock, i))
        proc.start()
```

Exchanging objects between processes

multiprocessing поддерживает два типа канала связи между процессами: Queue и Pipe
Queue:

```
from multiprocessing import Process, Queue

def f(q):
    q.put(['u', 0, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target = f , args = (q,))
    p.start()
    print(q.get())
    p.join ()
```

will print to standard output:
['u', 0, 'hello']

Pipe

Два объекта соединения, возвращаемые Pipe() представляют собой два конца трубы. Каждый объект соединения имеет методы send() и recv() (среди прочих). Обратите внимание, что данные в канале могут стать поврежденными, если два процесса пытаются читать или записывать на один и тот же конец канала одновременно

Pipe

```
from multiprocessing import Process, Pipe

def f(conn):
    conn.send([23, None, 'child'])
    conn.close()

if __name__ == '__main__':
    p_conn, c_conn = Pipe()
    p = Process(target=f, args=(c_conn,))
    p.start()
    print(p_conn.recv())
    p.join()
```

will print to standard output:
[23, None, 'child']

Pool

class Pool предлагает удобное средство для параллелизации выполнения функции через несколько входных значений, распределение входных данных через процессы.

```
from multiprocessing import Pool
def deg2(n):
    return n ** 2
if __name__ == '__main__':
    ns = [5, 10, 20]
    pool = Pool(processes=3)
    print(pool.map(deg2, ns))
```

will print to standard output:
[25, 100, 400]

Заключение

Итак, мы познакомились с библиотекой multiprocessing. Также для взаимодействия процессов в библиотеке присутствуют такие механизмы, как Shared memory и Server process.