

Bài: Đa luồng (Multithreading) trong Python

Xem bài học trên website để ủng hộ Kteam: [Đa luồng \(Multithreading\) trong Python](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

Bài toán đa luồng trong Python

Bản chất code python sẽ thực hiện tuần tự từ trên xuống và khi lệnh code trước đó hoàn thành mới tiếp tục lệnh tiếp theo.

Ví dụ bạn có một vòng lặp vô tận hoặc bạn gửi request đến bên thứ 3 và phải đợi bên đó phản hồi. Bạn muốn tiếp tục đợi nhưng có thể đồng thời thực hiện công việc khác.

Cách thức xử lý bài toán này từ đơn luồng thành đa luồng (Multithreading) là sử dụng Thread hoặc Process. Ở bài này chúng ta dùng Thread để thực hiện **đa luồng trong Python**.

Bài toán đơn luồng trong python

Ta có nhu cầu thực hiện công việc xuất ra màn hình thông tin của **class ProfileInfo** với **ProfileInfo** được định nghĩa trong file **classKteam.py** như sau:

:

```
class ProfileInfo:
    Name = ''

    def __init__(self, Name):
        self.Name = Name
```

và file **multithreading.py** có nội dung như sau:

:

```
from classKteam import *

def showInfo(profile):
    while 1==1:
        print("Profile: {}".format(profile.Name))

if __name__ == "__main__":
    showInfo(ProfileInfo('Thread 1'))
    showInfo(ProfileInfo('Thread 2'))
```

Ở đây chúng ta có thể nhận thấy tại hàm **__main__** thực hiện 2 lần hàm **showInfo**.

- Lần 1 là **profile** với **Name** là **Thread 1**
- Lần 2 là **Thread 2**.

Và kết quả khi run file **multithreading.py**

```
Profile: Thread 1
Profile: Thread 1
Profile: Thread 1
Profile: Thread 1
Profile: Thread 1
Profile: Thread 1
```

Console liên tục in ra màn hình dòng **Profile: Thread 1**.

Ồ! Thế Thread 2 đâu mất rồi? Chắc chắn Thread 2 không thể được in ra vì **showInfo** của Thread 1 chưa thực hiện xong. Và sẽ không bao giờ xong vì đây là vòng lặp vô tận:

```
:
while 1==1:
```

Bài toán đa luồng từ đơn luồng trong python

Vậy lúc này chúng ta **muốn cả Thread 1 và Thread 2 đều được in ra đồng thời thì phải làm sao?**

Ta sẽ thực hiện **import** thư viện threading ngay bên dưới dòng code **import classKteam** như sau

```
:
import threading
```

Và đoạn code để đưa một hàm vào một luồng chạy riêng biệt như sau:

```
:
p1 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 1'),))
p1.start()
```

Ở đây, chúng ta có thể thấy hàm cần thực hiện luồng riêng được đưa tên vào **target** và các tham số của hàm được đưa vào trong **args**. Các tham số cách nhau bởi dấu phẩy (",")

Toàn bộ file **multithreading.py** hiện trông như sau:

```
:
from classKteam import *
import threading

def showInfo(profile):
    while 1==1:
        print("Profile: {}".format(profile.Name))

if __name__ == "__main__":
    p1 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 1'),))
    p1.start()

    p2 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 2'),))
    p2.start()
```

Chúng ta cùng thực hiện **run script** nhé:

```
Traceback (most recent call last):
  File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10.3.10.2544.0_x64__qbz5q2kfrap@lib\threading.py", line 1016, in _bootstrap_inner
    self.run()
  File "C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10.3.10.2544.0_x64__qbz5q2kfrap@lib\threading.py", line 953, in run
    self._target(*self._args, **self._kwargs)
TypeError: self._target(*self._args, **self._kwargs) argument after * must be an iterable, not ProfileInfo

TypeError: _main._showInfo() argument after * must be an iterable, not ProfileInfo
```

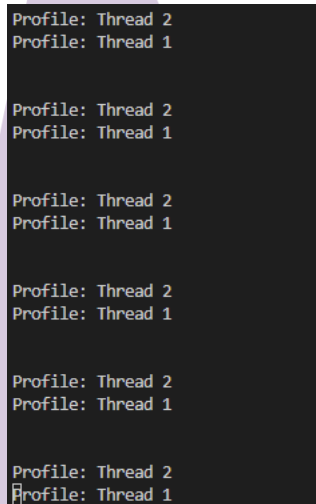
Lý do là **args** yêu cầu đầu vào phải là **iterable**. Để có thể chạy được script này ta thêm dấu phẩy (",") phía sau **parameter** đầu tiên của **args** như sau:

```
:
p1 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 1'),))
```

Lúc này file **multithreading.py** như sau:

```
:  
from classKteam import *  
import threading  
  
def showInfo(profile):  
    while 1==1:  
        print("Profile: {}".format(profile.Name))  
  
if __name__ == "__main__":  
    p1 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 1'),))  
    p1.start()  
  
    p2 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 2'),))  
    p2.start()
```

Khi run script:



```
Profile: Thread 2  
Profile: Thread 1  
  
Profile: Thread 2  
Profile: Thread 1  
  
Profile: Thread 2  
Profile: Thread 1  
  
Profile: Thread 2  
Profile: Thread 1  
  
Profile: Thread 2  
Profile: Thread 1  
  
Profile: Thread 2  
Profile: Thread 1
```

Thread 1 và 2 được in ra liên tục đồng đều nhau. Quá tuyệt vời phải không?

Nhưng chương trình chạy lặp liên tục và quá nhanh. Vậy nếu muốn giảm tốc độ chạy lại như kiểu có một đoạn ngủ (sleep) thì phải làm sao?

Sleep trong python

Để thực hiện sleep trong python chúng ta cần **import time**:

```
:  
import time
```

Để gọi **sleep** dùng đoạn code như sau (sleepTime được tính bằng giây):

```
:  
time.sleep(sleepTime)
```

Vậy nhu cầu sẽ là **sleep** mỗi khi in ra thông tin profile mình sẽ thêm sleep sau print như sau:

```
:
```

```
def showInfo(profile):
    while 1==1:
        print("Profile: {}".format(profile.Name))
        time.sleep(1)
```

sleep 1 là code của luồng đó sẽ tạm dừng tại vị trí đó 1 giây.

Vậy nếu mình muốn thời gian sleep này được truyền vào từ *args* thì sẽ điều chỉnh như sau:

```
:
```

```
from classKteam import *
import threading
import time

def showInfo(profile, sleepTime):
    while 1==1:
        print("Profile: {}".format(profile.Name))
        time.sleep(sleepTime)

if __name__ == "__main__":
    p1 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 1'),1))
    p1.start()

    p2 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 2'),1))
    p2.start()
```

Thêm **sleepTime** làm tham số thứ 2 cho hàm **showInfo**. Truyền biến **sleepTime** vào trong sleep và đồng thời truyền thời gian cần sleep vào trong *args* ở Thread (ở đây mình sẽ cho sleep 1 giây nên truyền vào là 1. Bạn có thể truyền vào số khác tùy ý ở mỗi luồng nhé!).

Khi **run script** bạn sẽ thấy vòng lặp vô tận được nghỉ 1 giây nên đã chạy chậm lại hẳn hơn.

Dừng (stop) luồng riêng biệt trong python

Vậy nếu mình có nhu cầu **dừng luồng số 1 khi nhấn phím 1, dừng luồng số 2 khi nhấn phím 2... Vậy thì phải làm sao?**

Chúng ta cần phải có cơ chế thấy lệnh **Stop** thì dừng, và cơ chế nhận được phím nào được bấm.

Vậy để có thể đặt lá cờ Stop cho từng luồng riêng biệt thì ta nên tạo thêm 1 biến **IsStop** vào trong **ProfileInfo**:

```
:
```

```
class ProfileInfo:
    Name = ''
    IsStop = False

    def __init__(self, Name):
        self.Name = Name
        self.IsStop = False
```

Với **IsStop = False** là luồng đó được phép chạy và ngược lại luồng đó sẽ phải dừng lại.

Vì bản chất **ProfileInfo** là 1 *class* nên nó là tham chiếu. Vậy nên khi giá trị của **IsStop** thay đổi thì bên trong hàm *showInfo* cũng sẽ nhận được giá trị thay đổi đó của **IsStop**. Mình sẽ cài đặt thêm cơ chế dừng cho hàm *showInfo* như sau:

```
:
```

```

if(profile.IsStop):
    print("{} stop.\n".format(profile.Name))
    break

```

Hàm *showInfo* lúc này trông như sau:

:

```

def showInfo(profile, sleepTime):
    # function to print cube of given num
    while 1==1:
        print("Profile: {}-{} - again after {}\n".format(profile.Name, profile.IsStop, sleepTime))
        if(profile.IsStop):
            print("{} stop.\n".format(profile.Name))
            break
        time.sleep(sleepTime)
        if(profile.IsStop):
            print("{} stop.\n".format(profile.Name))
            break

```

Mình kiểm tra nếu biến **IsStop** thành true thì sẽ **break** vòng lặp vô tận ngay lập tức. Mình cho lệnh này gọi ở trước và sau khi **sleep** để đảm bảo có thể stop hàm ngay khi có thể.

Multithreading.py lúc này như sau:

:

```

from classKteam import *
import threading
import time

def showInfo(profile, sleepTime):
    while 1==1:
        print("Profile: {}\n".format(profile.Name))
        if(profile.IsStop):
            print("{} stop.\n".format(profile.Name))
            break
        time.sleep(sleepTime)
        if(profile.IsStop):
            print("{} stop.\n".format(profile.Name))
            break

if __name__ == "__main__":
    p1 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 1'),1))
    p1.start()

    p2 = threading.Thread(target=showInfo, args=(ProfileInfo('Thread 2'),1))
    p2.start()

```

Vậy để cài đặt thêm việc nhận biết phím nào được bấm thì làm sao?

Trước đó chúng ta cần phải đưa các **Thread** của mình vào một danh sách để có thể quản lý

Mình sẽ tạo một biến **Profiles** ngoài hàm và đưa các **Thread** cần chạy vào trong nó như sau:

:

```

profiles = [ProfileInfo('Thread 1'), ProfileInfo('Thread 2')]

```

Ở hàm **main** thay vì thực hiện gọi start từng thread mình sẽ đưa vào vòng lặp duyệt qua hết profile mình có:

:

```
for item in profiles:
    p = threading.Thread(target=showInfo, args=(item, 1))
    p.start()
```

Chúng ta cần cài thêm **pynput** và **import keyboard** vào code

Đầu tiên bạn cần **install pynput** bằng cách gọi lệnh sau ở **terminal**

```
:
```

```
pip install pynput
```

Khi cài đặt xong bạn import keyboard vào code của mình như sau:

```
:
```

```
from pynput import keyboard
```

Giờ chúng ta sẽ triển khai **listener** cho keyboard:

Thêm đoạn code này vào cuối hàm **__main__**

```
:
```

```
with keyboard.Listener(on_press=on_press) as listener:
    listen = listener
    listener.join()
```

và thêm biến **listen** ngoài hàm bên dưới profiles như sau:

```
:
```

```
listen = keyboard.Listener
```

Và bạn cần định nghĩa thêm hàm **on_press** như sau:

```
:
```

```
def on_press(key):
    vk = key.vk if hasattr(key, 'vk') else key.value.vk
    print('vk =', vk)
    if(vk == None):
        return
    index = vk - 48
    if(index >= 0 and index < len(profiles) and profiles[index].IsStop == False):
        print("Doing stop: {}".format(profiles[index].Name))
        profiles[index].IsStop = True
```

Ở hàm **on_press** này khi nhấn **phím số thì sẽ trừ đi 48** để ra được **vị trí tương ứng** của **ProfileInfo** trong profiles.

- Ví dụ số 0 được nhấn sẽ có mã số là 48. 48-48 = 0. Khi này **index** sẽ bằng **0**. **Profiles[index]** cũng chính là profile đầu tiên.

Chúng ta có được profile cần dừng thì thực hiện chuyển giá trị **IsStop** cho profile đó thành **True** để trong hàm **showInfo** kiểm tra thấy **IsStop == true** thì **break** vòng lặp. Lúc này hàm **showInfo** sẽ được dừng lại.

Trong hàm **showInfo** mình điều chỉnh lệnh print lại một chút để dễ kiểm tra được thông số của **ProfileInfo**:

```
:
```

```
print("Profile: {}-{} - again after {}\n".format(profile.Name, profile.IsStop, sleepTime))
```

file **multithreading.py** như sau:

:

```
import time
import threading
from classKteam import *

from pynput import keyboard

profiles = [ProfileInfo('Thread 1'), ProfileInfo('Thread 2')]
listen = keyboard.Listener

def on_press(key):
    vk = key.vk if hasattr(key, 'vk') else key.value.vk
    print('vk =', vk)
    if(vk == None):
        return
    index = vk - 48
    if(index >= 0 and index < len(profiles) and profiles[index].IsStop == False):
        print("Doing stop: {}".format(profiles[index].Name))
        profiles[index].IsStop = True

def showInfo(profile, sleepTime):
    while 1==1:
        print("Profile: {}-{} - again after {}".format(profile.Name, profile.IsStop, sleepTime))
        if(profile.IsStop):
            print("{} stop.\n".format(profile.Name))
            break
        time.sleep(sleepTime)
        if(profile.IsStop):
            print("{} stop.\n".format(profile.Name))
            break

if __name__ == "__main__":

    for item in profiles:
        p = threading.Thread(target=showInfo, args=(item, 3))
        p.start()

    with keyboard.Listener(on_press=on_press) as listener:
        listen = listener
        listener.join()
```

Khi này bạn **run script** sẽ thấy script chạy bình thường. Thử nhấn phím 0 thì sẽ thấy **Thread 1** được dừng. Nhấn thêm phím 1 thì **Thread 2** sẽ dừng. Bạn tiếp tục nhấn phím sẽ thấy terminal vẫn tiếp tục hiển thị giá trị của phím bạn bấm. Lý do là vì **listener** chưa dừng lại.

Để có thể hoàn toàn dừng chương trình khi tất cả **Thread** đã dừng chúng ta thực hiện như sau:

Thêm đoạn code này ở cuối của hàm *showInfo*

:

```
totalRunningThread = any(x.IsStop == False for x in profiles)
print("Total: {}".format(totalRunningThread))
if(totalRunningThread == False):
    listen.stop()
```

Chúng ta tạo một biến kiểm tra tất cả **Profile** mà **IsStop == False** tức là kiểm tra xem có profile nào đang chạy hay không. Nếu không có profile nào đang chạy thì cho **stop listen** bằng lệnh **listen.stop()**

Khi này bạn chạy thử lại sẽ thấy khi tất cả luồng được dừng thì script của chúng ta cũng dừng lại.

Chúng ta thêm 1 dòng này ở hàm **main** bên trong vòng lặp để đảm bảo profile trước khi start **IsStop** luôn bằng **false**.

:

```
item.IsStop = False
```

Lúc này file **multithread.py** hoàn thiện s4 như sau:

:

```
import time
import threading
from classKteam import *

from pynput import keyboard

profiles = [ProfileInfo('Thread 1'), ProfileInfo('Thread 2')]
listen = keyboard.Listener

def on_press(key):
    vk = key.vk if hasattr(key, 'vk') else key.value.vk
    print('vk =', vk)
    if(vk == None):
        return
    index = vk - 48
    if(index >= 0 and index < len(profiles) and profiles[index].IsStop == False):
        print("Doing stop: {}".format(profiles[index].Name))
        profiles[index].IsStop = True

def showInfo(profile, sleepTime):
    while 1==1:
        print("Profile: {}-{} - again after {}\n".format(profile.Name, profile.IsStop, sleepTime))
        if(profile.IsStop):
            print("{} stop.\n".format(profile.Name))
            break
        time.sleep(sleepTime)
        if(profile.IsStop):
            print("{} stop.\n".format(profile.Name))
            break
    totalRunningThread = any(x.IsStop == False for x in profiles)
    print("Total: {}\n".format(totalRunningThread))
    if(totalRunningThread == False):
        listen.stop()

if __name__ == "__main__":
    for item in profiles:
        item.IsStop = False
        p = threading.Thread(target=showInfo, args=(item, 3))
        p.start()

    with keyboard.Listener(on_press=on_press) as listener:
        listen = listener
        listener.join()
```

Kết luận

Vậy là chúng ta đã tìm hiểu qua và nắm được cách vận dụng đa luồng một cách hiệu quả. Chúc các bạn có thể vận dụng kiến thức vào trong thực tế công việc hằng ngày.

© Bài viết được đăng tải tại website Howkteam.vn và video hướng dẫn được lưu trữ tại channel youtube: [Kteam](https://www.youtube.com/channel/UCkteam). Nhớ like, subscribe và nhấn chuông thông báo để không bỏ lỡ video mới nhất nhé.