기타 크롤링 모듈 –
이미지 크롤링 (병렬처리)

유현조

hyunjoe.yoo@mail.mcgill.ca

Intro

- 기존에는 1개씩 다운받으며 진행하는 방식
  - 먼저 시작하던 작업이 끝나야 다음 이미지를 다운받을 수 있었죠
- 기본적으로 파이썬은 프로세스가 1개로 실행
- 여러개의 프로세스로 나눠서 동시에 처리하고 싶으면 스레드 또는 멀티프로세싱 모듈을 사용
- 여러개의 프로세스(스레드)로 작업을 동시에 진행하는것을 병렬처리라고 합니다

Multithreading

- 파이썬의 멀티프로세싱이라는 모듈을 응용하여 더 빠른 크롤러를 개발할 수 있어요

```
from bs4 import BeautifulSoup
import urllib.request
import time
import os
import multiprocessing
```

- 기존의 소스코드 상단에 multiprocessing 모듈을 import 해줍니다. (파이썬 기본모듈)

Function & Process

- 하나의 함수를 정의합니다 - 총 크롤링 할 이미지 수를 프로세스의 수에 알맞게 분배하는 함수입니다.

```python
# 총 크롤링 갯수를 프로세스에 맞춰 할당량 계산
def get_count(num, p=4):
    list = []
    allocate = int(num/p) # 프로세스 4개 생성하므로
    for n in range(p):
        list.append(allocate)
    list[p-1] += num%p # 마지막 프로세스는 할당량 분배 후 남는 작업도 포함
    print("프로세스당 할당량:", list)
    return list
```


- 함수의 인자는 2개: 첫 번째 인자 num (총 이미지 수), 두 번째 인자 p (프로세스의 수)

Function & Process (cont..)

- 프로세스의 수를 명시해주지않으면
  - 기본값으로 4로 진행합니다
- 25개의 이미지를 4개의 프로세스가 나눠서 처리한다고 하면
  - 기본적으로 6개씩 진행하고, 남는 1개는 아무 프로세스에서 추가로 진행하면 됩니다

Function for Distribution

```
list[p-1] += num%p # 마지막 프로세스는 할당량 분배 후 남는 작업도 포함
```

- 위 코드 부분이 남는 작업을 마지막 프로세스에게 추가해주는 코드입니다
  - (할당량 분배 후 남는 작업 수 더해주기)

- 이제 분배해주는 함수를 작성하였으니 사용할준비가 되었습니다.

Code Explanation

- process 변수에는 생성한 프로세스들을 저장하기 위한 리스트입니다.
- 첫 번째 for문에서는 방금 만든 get_count(총 이미지 수, 프로세스 수) 함수를 호출하여 프로세스마다 얼마씩 작업을 해야하는지에 대한 리스트를 받아옵니다.

```
if __name__ == "__main__":
    num = int(input("이미지 수: "))
    start = time.time() # 크롤링 시작 시간
    process = []
    for count in get_count(num, 4):
        p = multiprocessing.Process(target=get, args=(count,))
        process.append(p)
        p.start()

    for p in process:
        p.join()
    print("크롤링 종료")
    print("크롤링 소요 시간:", round(time.time() - start, 6)) # 소수점 아래 6자리까지
```

Code Explanation (cont..)

- 만약 4개의 프로세스가 총 13개의 이미지를 다운받아야한다면 [3, 3, 3, 4] 리스트가 반환됩니다.
- for문으로 3, 3, 3, 4를 순차적으로 반복하면서 프로세스를 생성합니다.
- Process(target=함수이름, args=함수인자)

```
if __name__ == "__main__":
    num = int(input("이미지 수: "))
    start = time.time() # 크롤링 시작 시간
    process = []
    for count in get_count(num, 4):
        p = multiprocessing.Process(target=get, args=(count,))
        process.append(p)
        p.start()

    for p in process:
        p.join()
    print("크롤링 종료")
    print("크롤링 소요 시간:", round(time.time() - start, 6)) # 소수점 아래 6자리까지
```

Target Function

- 기존에 크롤링하던 함수는 get 함수였고 몇개를 크롤링할지 숫자 값을 받아왔습니다.

```
def get(max_count = 1):
    base_url = "http://10000img.com/" # 이미지 src와 조합하여 다운받을 주소
    url = "http://10000img.com/ran.php" # 접속할 URL
    count = 1
    while count <= max_count:
```


- target에는 함수 이름인 get을 전달해주고
- args에는 target 함수의 인자로 전달 할 데이터를 추가해주시면 됩니다.

- get 함수에 [3, 3, 3, 4]를 하나씩 반복하면서 해당 값을 전달해줍니다.
  - 각각의 프로세스에서 3, 3, 3, 4의 데이터를 get 함수에 전달해주며 각각 알아서 진행하게 됩니다.

```
process = []
for count in get_count(num, 4):
    p = multiprocessing.Process(target=get, args=(count,))
    process.append(p)
    p.start()
```

- p에 생성한 프로세스를 임시로 저장하고 process라는 리스트에 생성한 프로세스를 추가하는 모습입니다.
- 마지막으로는 start() 함수를 호출하여 프로세스를 실행시키는 모습입니다.

Join Function

- 프로세스를 실행한 후 바로 join() 함수를 호출하는 모습입니다.

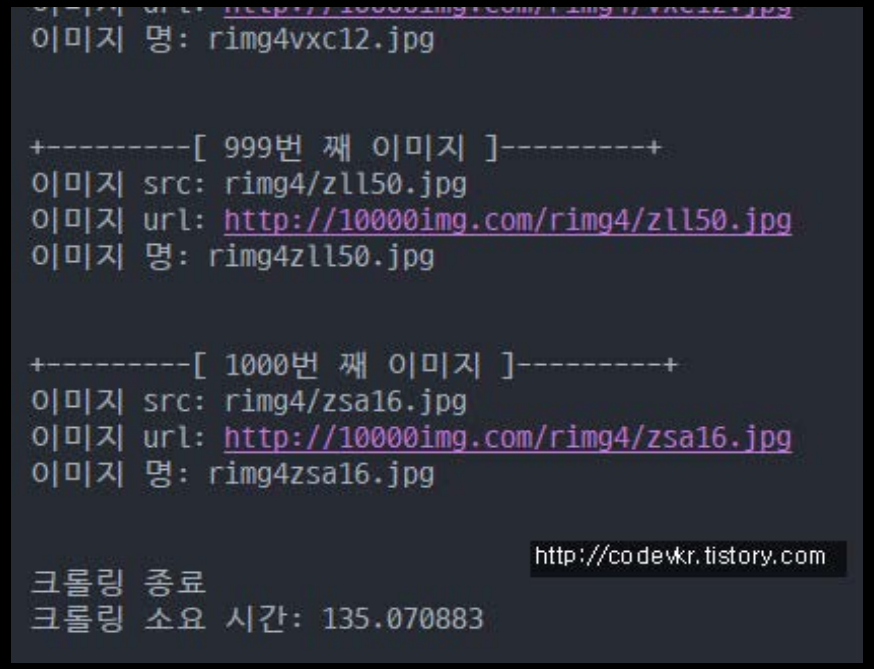
```
for p in process:
    p.join()
```

- join() 함수는 프로세스가 종료될 때 까지 대기하도록 하는 함수입니다.

- process 리스트에 생성한 모든 프로세스들이 존재하므로 모든 프로세스가 완료되어야 다음 코드로 진행할 수 있습니다.

Results

- 아래 두 사진은 1000개의 이미지 크롤링 소요시간입니다.
  - (프로세스 1개 - 병렬처리 X)
  - (프로세스 4개 - 병렬처리 O)
- 135초와 81초 → 차이가 나는 결과값 입니다



Conclusion

- 프로세스가 많을수록 좋은것도 아닙니다.
  - 한꺼번에 많은 프로세스가 네트워크에 접속하여 크롤링 할 경우 인터넷 속도 저하로 인해 응답시간 초과와 같은 문제가 발생할 수 있기 때문에
- 적당한 프로세스 수를 찾으시는게 중요합니다

감사합니다

참고자료: https://codevkr.tistory.com/38?category=705611