python #7

▼ 문제

- 1,000만 개의 난수를 생성한 후, 각 숫자가 소수인지 판별하는 작업을 단일 프로세스와 멀티 프로세스로 나누어 처리 시간을 비교

- 요구사항
  - random을 사용하여 1,000만 개의 1~10,000,000 사이 정수 리스트를 생성하세요.
  - 숫자가 소수인지 판별하는 함수를 구현하세요.
  - 아래 두 가지 방식으로 소수의 개수를 세고 처리 시간을 비교하세요
  (a) 단일 프로세스 방식
  (b) multiprocessing.Pool 사용 (병렬 처리)
  - 처리 시간과 소수 개수를 출력하세요.

▼ 코드

▼ 전체 코드

```
import random
import time
import math
from multiprocessing import Pool, cpu_count

# 소수 판별 함수
def is_prime(n):
    if n < 2: # 0, 1, 음수는 소수가 아니므로
        return False # False로 종료
    for i in range(2, int(math.isqrt(n)) + 1): # 나눠볼 후보 약수: 2부터 √n.
        if n % i == 0: # 나누어떨어지는 i를 찾으면 합성수이므로
            return False # 바로 False로 종료
    return True # √n까지 어떤 i로도 나누어떨어지지 않으면 소수

# 단일 프로세스 소수 세기
def count_primes_single(numbers):
    start = time.perf_counter() # 시작 시간 기록
    count = sum(1 for num in numbers if is_prime(num)) # 소수 판별 결과가 True인지 확인, True이면 더해서 count를 구함
    end = time.perf_counter() # 종료 시간 기록
    return count, end - start # 경과 시간 계산 (소수 개수, 걸린 시간초 반환)

# 멀티 프로세스 소수 세기
def count_primes_multi(numbers):
    start = time.perf_counter() # 시작 시간 기록
    with Pool(processes=cpu_count()) as pool: # CPU 코어 수만큼 프로세스를 띄워 Pool 구성
        results = pool.map(is_prime, numbers) # numbers 리스트를 프로세스들에 분할 배정해 병렬 처리: 워커 프로세스에서 is_prime 검사를 각 정수에 수행
    count = sum(results) # results 내 True(1)의 합으로 소수 개수 구하기
    end = time.perf_counter() # 종료 시간 기록
    return count, end - start # 경과 시간 계산

if __name__ == "__main__":
    N = 10_000_000
    numbers = [random.randint(1, 10_000_000) for _ in range(N)] # 1 ≤ x ≤ 10,000,000 범위에서 난수 리스
```

```
트 생성

    # 단일 프로세스
    sp_count, sp_time = count_primes_single(numbers) # numbers에 대해 단일 프로세스로 소수 판별 -> 소수 개수, 경과 시간 반환
    print(f"[단일] 소수 개수: {sp_count}, 시간: {sp_time:.2f}초") # 소수점 둘째 자리까지 출력

    # 멀티 프로세스
    mp_count, mp_time = count_primes_multi(numbers) # numbers에 대해 multi processing으로 소수 판별 -> 소수 개수, 경과 시간 반환
    print(f"[멀티] 소수 개수: {mp_count}, 시간: {mp_time:.2f}초") # 소수점 둘째 자리까지 출력
```

▼ 소수 판별 함수

```
# 소수 판별 함수
def is_prime(n):
    if n < 2: # 0, 1, 음수는 소수가 아니므로
        return False # False로 종료
    for i in range(2, int(math.isqrt(n)) + 1): # 나눠볼 후보 약수: 2부터 √n.
        if n % i == 0: # 나누어떨어지는 i를 찾으면 합성수이므로
            return False # 바로 False로 종료
    return True # √n까지 어떤 i로도 나누어떨어지지 않으면 소수
```

- n < 2
  - 0, 1, 음수는 소수가 아니므로 False
- math.isqrt(n)
  - 정수 제곱근
- n % i == 0 → return False
  - 나누어떨어지는 i를 찾으면 합성수이므로 바로 False로 종료
- return True
  - √n까지 어떤 i로도 나누어떨어지지 않으면 소수

▼ 소수 세기 - 단일 프로세스

```
# 단일 프로세스 소수 세기
def count_primes_single(numbers):
    start = time.perf_counter() # 시작 시간 기록
    count = sum(1 for num in numbers if is_prime(num)) # 소수 판별 결과가 True인지 확인, True이면 더해서 count를 구함
    end = time.perf_counter() # 종료 시간 기록
    return count, end - start # 경과 시간 계산 (소수 개수, 걸린 시간초 반환)
```

- start = time.perf_counter()
  - 시작 시간 기록
- if is_prime(num)
  - is_prime: 소수 판별 함수.
  - 소수 판별 결과가 True인지 확인
- count = sum(1 for num in numbers)
  - True이면 더해서 count를 구함
- end = time.perf_counter()

    - 종료 시간 기록
- return count, end - start
    - (소수 개수, 걸린 시간초) 반환

▼ 소수 세기 - multiprocessing.Pool 사용

```
# 멀티 프로세스 소수 세기
def count_primes_multi(numbers):
    start = time.perf_counter() # 시작 시간 기록
    with Pool(processes=cpu_count()) as pool: # CPU 코어 수만큼 프로세스를 띄워 Pool 구성
        results = pool.map(is_prime, numbers) # numbers 리스트를 프로세스들에 분할 배정해 병렬 처리: 워커 프로세스에서 is_prime 검사를 각 정수에 수행
    count = sum(results) # results 내 True(1)의 합으로 소수 개수 구하기
    end = time.perf_counter() # 종료 시간 기록
    return count, end - start # 경과 시간 계산
```

- start = time.perf_counter()
    - 시작 시간 기록
- Pool(processes=cpu_count()) as pool
    - CPU 코어 수만큼 프로세스를 띄워 Pool 구성
- pool.map(is_prime, numbers)
    - numbers 리스트를 프로세스들에 분할 배정해 병렬 처리
    - is_prime: 워커 프로세스에서 is_prime 검사를 각 정수에 수행
    - results
        - is_prime의 반환값(True 또는 False)로 구성된 리스트
- count = sum(results)
    - (True는 1이므로) True의 합으로 소수 개수 구하기
- end = time.perf_counter()
    - 종료 시간 기록
- return count, end - start
    - 경과 시간 계산

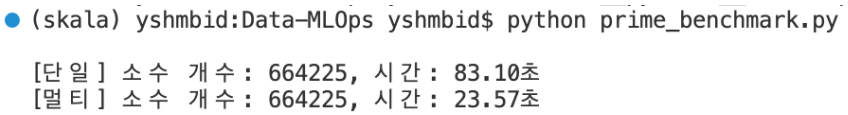
▼ __main__

```
if __name__ == "__main__":
    N = 10_000_000
    numbers = [random.randint(1, 10_000_000) for _ in range(N)] # 1 ≤ x ≤ 10,000,000 범위에서 난수 리스트 생성

    # 단일 프로세스
    sp_count, sp_time = count_primes_single(numbers) # numbers에 대해 단일 프로세스로 소수 판별 -> 소수 개수, 경과 시간 반환
    print(f"[단일] 소수 개수: {sp_count}, 시간: {sp_time:.2f}초") # 소수점 둘째 자리까지 출력

    # 멀티 프로세스
    mp_count, mp_time = count_primes_multi(numbers) # numbers에 대해 multi processing으로 소수 판별 -> 소수 개수, 경과 시간 반환
    print(f"[멀티] 소수 개수: {mp_count}, 시간: {mp_time:.2f}초") # 소수점 둘째 자리까지 출력
```

- random.randint(a, b)

    - 1 ≤ x ≤ 10,000,000 범위에서 난수 생성
- count_primes_single(numbers)
    - is_prime()으로 numbers에 대해 소수 판별
    - sp_count, sp_time
        - 소수 개수, 경과 시간 반환
- :.2f
    - 소수점 둘째 자리까지
- count_primes_multi(numbers)
    - Pool(processes=cpu_count())로 **CPU 코어 수**만큼 프로세스를 생성하여 is_prime()을 병렬로 실행해서 numbers에 대해 소수 판별
    - sp_count, sp_time
        - 소수 개수, 경과 시간 반환

▼ **실행 결과**



▼ **결과 정리**

- 단일 프로세스 count_primes_single(numbers)
    - 리스트를 순차로 검사
    - is_prime이 True인 경우만 1을 더해서 → 총 소수 개수와 경과 시간을 반환
- 멀티 프로세스 count_primes_multi(numbers)
    - CPU 코어 수만큼 워커 프로세스를 띄우고 pool.map(is_prime, numbers)로 작업을 분할해 동시에 처리
    - 워커의 True/False 결과 리스트를 sum으로 합쳐 소수 개수를 구해서 → 총 소수 개수와 경과 시간을 반환
- 결과
    - 단일: 소수 개수 664225, 시간 83.10초
    - 멀티: 소수 개수 664225, 시간 23.57초
    - 두 방식의 소수 개수는 일치하고, 멀티프로세스가 23.57초로 단일 프로세스인 83.10초보다 빨랐습니다.