Multiprocessing and Multithreading in python

파이썬에서 병렬처리를 구현하는 방식은 두가지로 멀티 쓰레드를 사용하거나 멀티 프로세스를 사용하는 것이다. 쓰레드는 가볍지만 GIL로 인해 계산 처리를 하는 작업은 한번에 하나의 쓰레드에서만 작동하여 cpu 작업이 적고 I/O 작업이 많은 병렬 처리 프로그램에서 효과를 볼 수 있다.

프로세스는 각자가 고유한 메모리 영역을 가지기 때문에 더 많은 메모리를 필요로 하지만, 각각 프로세스에서 병렬로 cpu 작업을 할 수 있고 이를 이용해 여러 머신에서 동작하는 분산 처리 프로그래밍도 구현할 수 있다.

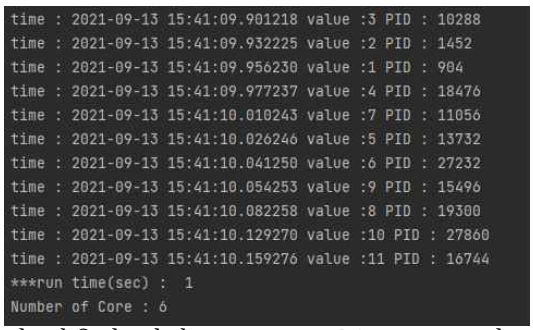
(프로그램 : 코드 덩어리 파일, 그 프로그램을 실행한 게 프로세스

스레드 : 프로세스의 코드에 정의된 절차에 따라 실행되는 특정한 수행 경로)

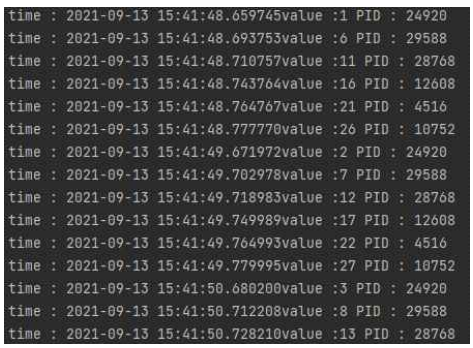
멀티프로세스- (POOL과 PROCESS가 대표적)

Pool과 Process 차이점은, Pool의 경우 실행되어야 할 작업이 코어수 만큼 분할되고 각 코어수 만큼 프로세스가 생성되어 힐당받은 작업을 처리하는데, Process의 경우 각 작업마다 새로운 프로세스가 할당되어 작업을 처리한다.

1.Process 파일을 실행시켜 결과를 살펴보면 각 실행마다 다른 프로세스(PID가 각각 다름)에서 실행된것을 알 수 있다. (pool.py) PID : Process Identification Number

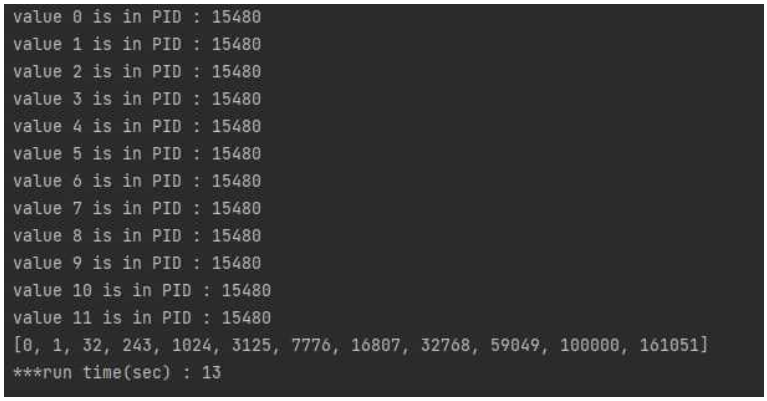


2.Pool 코드를 실행하면 다음과 같다. non multiprocess 라고 되어 있는 결과는 for문을 사용한 싱글 프로세스이며, 한 개의 프로세스만 사용되었으며, 순차적으로 실행되어 11초의 시간이 소요된 것을 확인할 수 있다. Multiprocess의 경우 동시에 6개의 프로세스(본인 테스트 컴퓨터는 6core이다.)가 할당되었으며 각 PID가 다른것을 확인할 수 있다. 99까지 도달하는데 20초 걸렸으므로 싱글코어의 11까지 도달한것과 상대적 비교시 약 2초정도 걸렸으므로 매우 속도가 빨라졌다. (process.py : 주석 달리하며 비교할 것)**(PID: 24920 주기적으로 반복확인)**



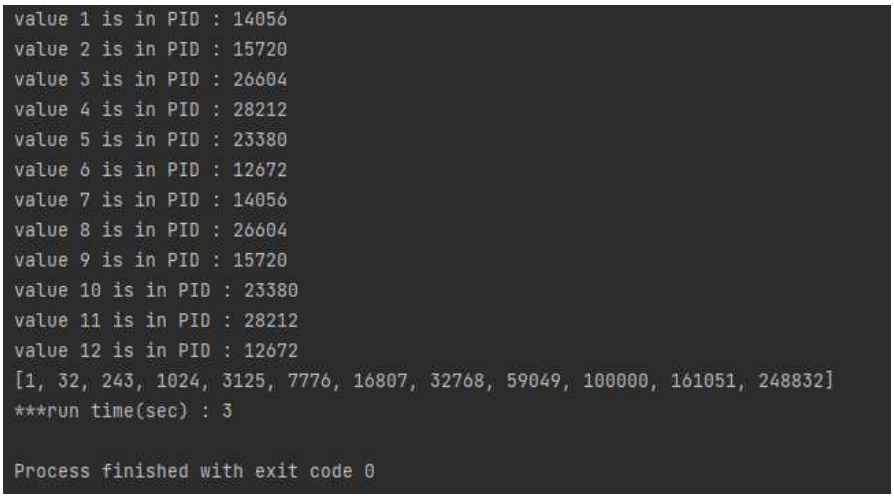
1.Pool examples : 멀티 프로세스의 경우 각기 다른 PID를 사용해 연산속도가 빠르다.
해당 코드는 1~13까지 각각 5승을 연산

single_pool.py : 싱글 프로세스 사용한 연산 13초 소요



```
value 0 is in PID : 15480
value 1 is in PID : 15480
value 2 is in PID : 15480
value 3 is in PID : 15480
value 4 is in PID : 15480
value 5 is in PID : 15480
value 6 is in PID : 15480
value 7 is in PID : 15480
value 8 is in PID : 15480
value 9 is in PID : 15480
value 10 is in PID : 15480
value 11 is in PID : 15480
[0, 1, 32, 243, 1024, 3125, 7776, 16807, 32768, 59049, 100000, 161051]
***run time(sec) : 13
```

multi_pool.py : 멀티 프로세스 사용한 연산 3초 소요



```
value 1 is in PID : 14056
value 2 is in PID : 15720
value 3 is in PID : 26604
value 4 is in PID : 28212
value 5 is in PID : 23380
value 6 is in PID : 12672
value 7 is in PID : 14056
value 8 is in PID : 26604
value 9 is in PID : 15720
value 10 is in PID : 23380
value 11 is in PID : 28212
value 12 is in PID : 12672
[1, 32, 243, 1024, 3125, 7776, 16807, 32768, 59049, 100000, 161051, 248832]
***run time(sec) : 3

Process finished with exit code 0
```

Pool안의 값으로 job을 할당받을 Process의 수를 의미한다. 위와 같이 6개의 Process에 할당하고 동일하게 map함수를 전달해본 결과, 6개의 피드(PID가 12개의 수를 6개씩 할당받아 작업을 수행했고 수행시간도 3초로 감소했음을 알 수 있다.

test.py

본인의 로컬 컴퓨터에서 활용할 수 있는 CPU 확인

2. Process example (Square 연산 예제)

process_square1.py, process_square2.py

```
Process ID: 29276
Process Name: Process-1
The number 1 squares to 1.
Process ID: 3232
Process Name: Process-2
The number 2 squares to 4.
Process ID: 26388
Process Name: Process-3
The number 3 squares to 9.
Process ID: 2200
Process Name: Process-4
The number 4 squares to 16.
Process ID: 18864
Process Name: Process-5
The number 5 squares to 25.
```

3. 멀티 스레드 예제

multi_thread.py

```
Result: 4999999950000000
***run time(sec) : 4

Process finished with exit code 0
```

4. 단일 스레드 예제

single_thread.py

```
Result: 4999999950000000
***run time(sec) : 5

Process finished with exit code 0
```

큰 차이가 없다. 그 이유는 위에 설명했듯이 쓰레드는 가볍지만 GIL로 인해 계산 처리를 하는 작업은 한번에 하나의 쓰레드에서만 작동하여 cpu 작업이 적고 I/O 작업이 많은 병렬 처리 프로그램에서 효과를 볼 수 있기 때문이다.

GIL:Global Interpreter Lock

CPython에서의 GIL은 Python 코드(bytecode)를 실행할 때에 여러 thread를 사용할 경우, 단 하나의 thread만이 Python object에 접근할 수 있도록 제한하는 mutex 이다. 그리고 이 lock이 필요한 이유는 CPython이 메모리를 관리하는 방법이 thread-safe하지 않기 때문이다.