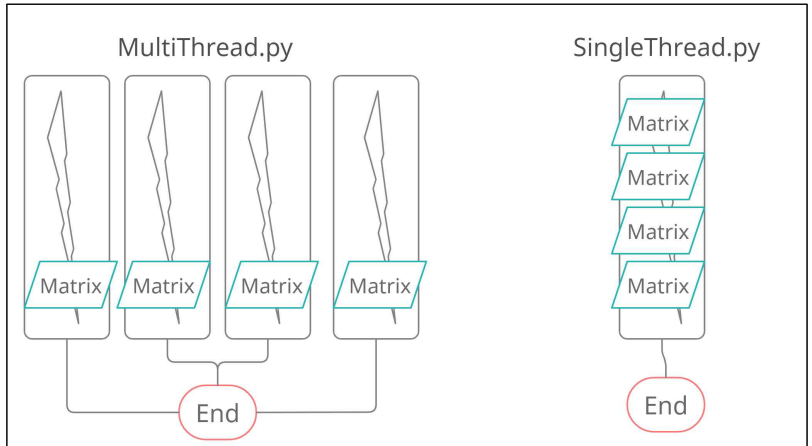
[OS 과제2] thread로 행렬 곱 분할 계산

1. 과제 목표
 - 프로세스가 수행할 작업을 thread별로 독립적으로 수행하는 개념을 습득한다.

2. 과제
 1)과제 내용

난수를 통해 행렬을 생성한 뒤 행렬 곱셈을 수행한다.
이때,
①단일 스레드로 수행한 경우
②다중 스레드로 수행한 경우
 의 실행시간을 비교하여 다중스레드로 실행한 경우 실행 시간 단축 효과가 있는지 확인해 본다.

2) 문제 해결 방법
과제 수행 중 Unix 상에서 행렬 연산에 필요한 모듈인 numpy가 import되지 않았다. 보통 리눅스 계열인 우분투에서도 pip install numpy하면 다운로드가 되어 import한 뒤 수행할 수 있지만 위 Unix상에서는 안됐기에 다른 대체방안으로 윈도우 상의 pycharm 환경으로 과제를 대체하였다. 이전 과제에서도 잘 수행했듯 다중 파일과 make 유틸리티 사용법을 알지만 현재 상황상 그러지 못한점을 감안해 주셨으면 좋겠습니다.



[설명] 멀티스레드에서는 각각의 스레드가 하나의 행렬 연산을 한다. 하지만 단일스레드에서는 한번에 네 개의 행렬 연산을 한다. 따라서 최종 결과는 행렬 연산이 네 개가 나오는 것은 동일하나. 두 파이썬 파일의 연산 시간을 측정해 무엇이 더 빠른지 비교한다.

3) 프로그램 구조

첫 번째. 단일 스레드 필요한 모듈 imort

```
from threading import Thread
import time, os
import numpy as np
```

[설명] 필요한 쓰레드와 time, numpy 등의 모듈을 import 한다.

두 번째. 행렬 연산 함수 작성

```
def matrix(num,row,column):
    X = np.random.randint(num, size=(row, column))
    Y = np.random.randint(num, size=(row, column))
    sol = [[sum(a * b for a, b in zip(X_row, Y_col)) for Y_col in zip(*Y)] for X_row in X]
    for r in sol:
        print(r)
    X = np.random.randint(num, size=(row, column))
    Y = np.random.randint(num, size=(row, column))
    sol = [[sum(a * b for a, b in zip(X_row, Y_col)) for Y_col in zip(*Y)] for X_row in X]
    for r in sol:
        print(r)
    X = np.random.randint(num, size=(row, column))
    Y = np.random.randint(num, size=(row, column))
    sol = [[sum(a * b for a, b in zip(X_row, Y_col)) for Y_col in zip(*Y)] for X_row in X]
    for r in sol:
        print(r)
    X = np.random.randint(num, size=(row, column))
    Y = np.random.randint(num, size=(row, column))
    sol = [[sum(a * b for a, b in zip(X_row, Y_col)) for Y_col in zip(*Y)] for X_row in X]
    for r in sol:
        print(r)
    return
```

[설명] 단일 스레드에서 수행할 함수에 행렬 연산을 4번 수행한다.

세 번째. main 문 작성

```
if __name__ == "__main__":
    start = time.time()
    num=100
    row=100
    column=100
    th1 = Thread(target=matrix, args=(num, row, column))


    th1.start()
    th1.join()

#print(f"Result: {sum(result)}")
print("***run time(sec) :", time.time() - start)
```

[설명] main문에서 단일 스레드에 타겟을 2번의 함수로 지정한뒤 필요한 argument 값을 넣는다.

네 번째. 실행속도 측정

```
[211892, 237956, 247678, 208632, 200703, 225732, 219747, 234287, 237923, 240817, 209435, 225636, 215527, 230
[220759, 235142, 217550, 210671, 193177, 218688, 216694, 251691, 223994, 229344, 222908, 226228, 219146, 234
[237582, 250018, 260732, 224084, 236703, 251108, 230161, 262050, 247782, 251702, 221582, 225861, 226274, 246
[244624, 258437, 272107, 244404, 229694, 272016, 229425, 283171, 278831, 247091, 231284, 250271, 252211, 257
[221491, 247509, 229614, 233246, 208698, 238932, 216951, 256640, 246573, 244229, 222616, 237406, 238899, 243
[264178, 288149, 302570, 268947, 256018, 288242, 270080, 316687, 302542, 287832, 261741, 272253, 280228, 297
***run time(sec) : 1.9464380741119385
```

[설명] 싱글 스레드의 경우 연산시간 약 1.94초 소요되었다.

다섯 번째. 멀티스레드 필요한 모듈 imort

```
from threading import Thread
import time, os
import numpy as np
```

[설명] 멀티스레디 연산에 필요한 Thread와 time, numpy 등의 모듈을 import 한다.

여섯 번째. 행렬 연산 함수 작성

```
def matrix(num,row,column):
    X = np.random.randint(num, size=(row, column))
    Y = np.random.randint(num, size=(row, column))
    sol = [[sum(a * b for a, b in zip(X_row, Y_col)) for Y_col in zip(*Y)] for X_row in X]
    for r in sol:
        print(r)

    return
```

[설명] 멀티 스레드에서 각각의 스레드가 연산할 하나의 행렬연산이 담긴 함수를 작성한다.

일곱 번째. main문 생성

```
if __name__ == "__main__":
    start = time.time()

    num = 100
    row = 100
    column = 100

    th1 = Thread(target=matrix, args=(num, row, column))
    th2 = Thread(target=matrix, args=(num, row, column))
    th3 = Thread(target=matrix, args=(num, row, column))
    th4 = Thread(target=matrix, args=(num, row, column))

    th1.start()
    th2.start()
    th3.start()
    th4.start()
    th1.join()
    th2.join()
    th3.join()
    th4.join()
```

[설명] main문에서 멀티 스레드에 타겟을 2번의 함수로 각각 지정한뒤 필요한 argument 값을 넣는다.

여덟 번째. 실행속도 측정

```
[262421, 261260, 256878, 239640, 230995, 276723, 259208, 222118, 235385, 232906, 251660, 221059, 260531, 244
[232713, 241617, 243649, 220628, 219537, 279318, 246373, 215544, 204288, 211234, 245169, 225476, 235789, 227
[245314, 244437, 242865, 223492, 224753, 258226, 227265, 202531, 220130, 207035, 217379, 210172, 240868, 227
[248509, 244640, 229506, 222646, 206309, 256716, 258144, 226467, 206525, 210451, 240215, 220067, 249251, 232
[268267, 265397, 263533, 245283, 247322, 282937, 266187, 230045, 239536, 268597, 268652, 245216, 258452, 272
[267879, 289776, 258933, 254775, 236219, 281334, 252490, 228897, 232781, 250146, 265730, 217622, 267952, 260
***run time(sec) : 1.5933582782745361
```

[설명] 멀티 스레드의 경우 연산시간 약 1.59초 소요되었다.

4) 결론

단일스레드와 멀티스레드 각각의 100x100 행렬연산을 수행한뒤 시간측정을 하였습니다. 단일 스레드의 경우 1.94초, 멀티 스레드의 경우 1.59초가 수행되었습니다. 따라서 멀티 스레드의 경우 연산속도가 더 빠른 것을 알 수 있습니다.

5) 부록 및 참고문헌

유투브 :

https://www.programiz.com/python-programming/examples/multiply-matrix