06. 파이썬 고급 주제

Total points 9/11

멀티 프로세스와 멀티 스레드 / 테스트 코드 작성 및 실행에 대해 알아보는 중규모 이상 프로젝트시 이용가능한 내용들이네요!

굉장히 굉장히 굉장히 굉장히 어렵지만 고-급 엔지니어가 되기 위해선 알아야 하는 (또는 대-IT 기업에서 이론적 내용으로 물어보는) 내용들이니 열심히 숙지해봅시다.

이해를 돕기 위해 평소보다 코드에 더 신경썼습니다 ㅠ_ㅠ (ps. 겁나 어렵네 ㅠㅠ)

✗ 다음은 프로세스와 스레드에 대한 설명이다. 옳지 않은 것은? * 0/1

- ○ 운영체제에서 실행되는 각 프로그램은 각각이 별도의 프로세스(process)다.
- ○ 각 프로세스에는 각 하나의 스레드(thread)가 있다.
- ○ 파이썬에 스레드 메커니즘이 있지만, 진정한 병렬(parallel) 실행이 지원되지는 않는다.
- ○ 멀티 프로세스는 별도의 메모리 영역을 가지며, 프로세스 간 통신(inter-process communication, IPC)와 같은 특별한 메커니즘으로만 통신할 수 있다.
- ◉ 파이썬 프로그램에는 단 하나의 메인 스레드만 존재한다. ✗
- ○ 모르겠다.

Correct answer

- ◉ 각 프로세스에는 각 하나의 스레드(thread)가 있다.

✓ 프로세스에 속한 스레드가 공유하는 메모리 영역이 아닌 것은? * 1/1

- (●) 스택(stack) 영역 ✓
- ( ) 힙(heap) 영역
- ( ) 코드(code) 영역
- ( ) 데이터(data) 영역
- ( ) 모르겠다.

✓ 다음은 subprocess 모듈을 이해하기 위한 코드이다. 출력 순서를 고르시오. * 1/1

```python
# child_process.py
import sys
import time

if __name__ == "__main__":
    print('{} subprocess...'.format(sys.argv[1]))                # n번째 프로세스인지 표시
    time.sleep(5)                                                # 5초간 대기
    print('{} subprocess done!'.format(sys.argv[1]))             # 목표메시지 출력

# child_process2.py
import sys
import time

if __name__ == "__main__":
    print('{} subprocess...'.format(sys.argv[1]))                # n번째 프로세스인지 표시
    time.sleep(3)                                                # 3초간 대기
    print('{} subprocess done!'.format(sys.argv[1]))             # 목표메시지 출력

# hello_subprocess.py
import subprocess

print("parent process...")                                       # 목표메시지 출력

childproc1 = subprocess.Popen(['python', 'child_process.py', "1"])   # 서브 프로세스 생성
childproc2 = subprocess.Popen(['python', 'child_process2.py', "2"])  # 서브 프로세스 생성
processes = [childproc1, childproc2]                             # 실행할 프로세스 담아놓기


for childproc in processes:
    childproc.communicate()                                      # process객체의 communicate 메소드로 실행

print("parent process done!")                                    # 목표메시지 출력

# on terminal
$ python hello_subprocess.py

# 아래 정답을 순서대로 고르시오. (예시 답안: 123456)
1. "parent process..."    2. "parent process done!"
3. "1 subprocess..."      4. "1 subprocess done!"
5. "2 subprocess..."      6. "2 subprocess done!"
```

135642 ✓

Feedback

참고링크

https://rrbb014.tistory.co…

✓ 다음은 워커 스레드와 데몬 스레드를 이해하기 위해 제작한 queue를 이용하여 내부적으로 자원의 접근을 직렬화한 코드이다. 출력 순서를 고르시오. * 1/1

```python
# threading_with_queue.py
import queue, threading, time

q = queue.Queue()

def worker(num):
    while True:
        item = q.get()
        if item is None:
            print("스레드 {0}: None이 들어왔습니다. thread를 종료합니다.".format(num+1))
            break
        print("스레드 {0}: 시작".format(num + 1, item))
        time.sleep(1)
        print("스레드 {0}: 처리 완료".format(num+1, item))
        q.task_done()

if __name__ == "__main__":
    num_worker_threads = 2
    threads = []
    for i in range(num_worker_threads):
        t = threading.Thread(target=worker, args=(i,))
        t.start()
        threads.append(t)

    for item in range(2):
        q.put(item)

    q.join()
    print("Queue is Empty!")

    for i in range(num_worker_threads):
        q.put(None)
    for t in threads:
        t.join()
    print("Main Thread Done")

# on terminal
$ python threading_with_queue.py

# 아래 정답을 순서대로 고르시오. (예시 답안: 12345678)
1. "스레드 1: 시작"                                  2. "스레드 1: 처리 완료"
3. "스레드 2: 시작"                                  4. "스레드 2: 처리 완료"
5. "스레드 1: None이 들어왔습니다. thread를 종료합니다."  6. "스레드 2: None이 들어왔습니다. thread를 종료합니다."
7. "Queue is Empty!"                                 8. "Main Thread Done"
```

13247568 ✓

✕ 다음은 뮤텍스와 세마포어에 대한 설명이다. 옳지 않은 설명을 고르시오. * 0/1

- ○ 뮤텍스(mutex)는 공유 리소스에 한 번에 하나의 스레드만 접근 할 수 있도록 하는 상호 배제 동시성 제어 정책을 강제하기 위해 설계되었다.
- ◉ 개념적으로, 뮤텍스는 0(잠금), 1(열림) 상태로만 존재한다. ✕
- ○ 뮤텍스는 한 마디로 한 스레드, 프로세스에 의해 소유될 수 있는 Key🔑 를 기반으로 한 상호배제기법이다.
- ○ 세마포어(semaphore) 값은 곧 한 번에 자원에 접근할 수 있는 스레드의 수이다.
- ○ 개념적으로, 세마포어는 1보다 큰 수로 시작할 수 있다.
- ○ 세마포어는 한 마디로, 현재 공유자원에 접근할 수 있는 스레드, 프로세스의 수를 나타내는 값을 두어 상호배제를 달성하는 기법이다.
- ○ 위 설명이 모두 맞다.
- ○ 모르겠다.

Correct answer

- ◉ 위 설명이 모두 맞다.

Feedback

이해가 안갈 때 참고하면 좋은 사이트

https://worthpreading.tist…

✓ 다음 중 데드락(deadlock, 교착 상태)이 발생하는 상황을 고르시오. * 1/1

- ○ 상호 배제: 자원은 한 번에 한 프로세스(혹은 스레드)만 사용할 수 있다.
- ○ 점유와 대기: 한 프로세스가 자원을 가지고 있는 상태에서, 다른 프로세스가 쓰는 자원의 반납을 기다린다.
- ○ 비선점: 다른 프로세스가 이미 점유한 자원을 강제로 뺏어오지 못한다.
- ○ 순환 대기: 프로세스 A, B, C가 있다고 가정할 때, A는 B가 점유한 자원을, B는 C가 점유한 자원을, C는 A가 점유한 자원을 대기하는 상태다.
- ◉ 위의 네 상황 모두가 발생해야만 데드락이 발생한다. ✓
- ○ 모르겠다.
- ○ 옵션 2

✓ 다음 중 스핀락(spinlock)에 대한 설명 중 옳지 않은 것은? * 1/1

- ○ 고성능 컴퓨팅 상황에 유용한 바쁜 대기의 한 형태이다.
- ◉ Lock-Unlock 과정이 길 때 사용하면 유용하다. ✓
- ○ 임계 구역에 진입이 불가능할 때, 진입이 가능할 때까지 반복문을 돌면서 재시도하는 방식이다.
- ○ 컨텍스트 스위칭이 자주 일어나지 않아 cpu의 부담을 덜어준다.
- ○ 모두 다 맞는 설명이다.
- ○ 모르겠다.

Feedback

유용한 사이트

https://brownbears.tistory....

✓ 다음은 threading 모듈 중 Condition을 이해하기 위한 코드이다. 출력 순서를 고르시오. * 1/1

```python
# threading_with_condition.py
import threading, time

def consumer(cond):
    name = threading.currentThread().getName()
    print("{0} 시작".format(name))
    with cond:
        print("{0} 대기".format(name))
        cond.wait()
        print("{0} 자원 소비".format(name))

def producer(cond):
    name = threading.currentThread().getName()
    print("{0} 시작".format(name))
    with cond:
        print("{0} 자원 생산 후 모든 소비자에게 알림".format(name))
        cond.notifyAll()

if __name__ == "__main__":
    condition = threading.Condition()
    consumer1 = threading.Thread(
        name="소비자1", target=consumer, args=(condition,))
    consumer2 = threading.Thread(
        name="소비자2", target=consumer, args=(condition,))
    producer0 = threading.Thread(
        name="생산자", target=producer, args=(condition,))

    consumer1.start()
    consumer2.start()
    time.sleep(0.5)
    producer0.start()

# # on terminal
# $ python threading_with_condition.py
#
# # 아래 정답을 순서대로 고르시오. (예시 답안: 12345678)
# 1. "소비자1 시작"        2. "소비자2 시작"
# 3. "소비자1 대기"        4. "소비자2 대기"
# 5. "소비자1 자원 소비"    6. "소비자2 자원 소비"
# 7. "생산자 시작"         8. "생산자 자원 생산 후 모든 소비자에게 알림"
```

13247856 ✓

✓ 다음은 프로파일링에 대한 내용이다. 성능을 향상 시킬 수 있는 방법이 아닌 것은? * 1/1

- ○ 읽기 전용 데이터는 리스트 대신 튜플을 사용한다.
- ◉ 반복문에서 제너레이터를 사용하여 순회하는 대신, 리스트나 튜플로 순회한다. ✓
- ○ 문자열을 연결할 때 + 연산자로 연결하는 대신, 리스트에 문자열을 추가한 후 마지막에 모든 항목을 연결한다.
- ○ 모두 성능을 향상시킬 수 있는 방법이다.
- ○ 모르겠다.

✓ 다음은 cProfile 모듈을 이용해 병목 현상을 찾는 과정을 나타낸 것이다. 다음 중 가장 크게 병목 현상을 일으키는 함수는? * 1/1

```
import cProfile

def function1():
    ls = []
    [redacted]
    print("function1:", str(ls[:5])[:-1], "...", str(ls[-5:])[1:])

def function2():
    [redacted]
    print("function2:", str(ls[:5])[:-1], "...", str(ls[-5:])[1:])

def function3():
    [redacted]
    print("function3:", str(ls[:5])[:-1], "...", str(ls[-5:])[1:])

def main():
    function1()
    function2()
    function3()

cProfile.run('main()')
```

```
cProfileTest
/Users/dewey.sudo/Document/PycharmProjects/Algorithm2/venv/bin/python /Users/dewey.sudo/Documen
function1: [0, 1, 2, 3, 4 ... 19999995, 19999996, 19999997, 19999998, 19999999]
function2: [0, 1, 2, 3, 4 ... 19999995, 19999996, 19999997, 19999998, 19999999]
function3: [0, 1, 2, 3, 4 ... 19999995, 19999996, 19999997, 19999998, 19999999]
         20000011 function calls in 5.597 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    5.597    5.597 <string>:1(<module>)
        1    0.697    0.697    0.697    0.697 cProfileTest.py:10(<listcomp>)
        1    0.472    0.472    0.472    0.472 cProfileTest.py:13(function3)
        1    0.820    0.820    5.597    5.597 cProfileTest.py:17(main)
        1    2.342    2.342    3.609    3.609 cProfileTest.py:3(function1)
        1    0.000    0.000    0.697    0.697 cProfileTest.py:9(function2)
        1    0.000    0.000    5.597    5.597 {built-in method builtins.exec}
        3    0.000    0.000    0.000    0.000 {built-in method builtins.print}
 20000000    1.266    0.000    1.266    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

- (●) function1 ✓
- ( ) function2
- ( ) function3
- ( ) builtins.exec
- ( ) builtins.print
- ( ) 모르겠다.

✓ 다음 중 단위 테스트 모듈이 아닌 것은? *

1/1

- (●) casetest ✓
- ( ) doctest
- ( ) pytest
- ( ) unittest
- ( ) 모르겠다.