비동기(Asynchronous)와 동기(Synchronous)의 정의 및 차이점

-동기(Synchronous) : 작업이 순차적으로 실행되며, 이전 작업이 완료될 때까지 다음 작업이 시작되지 않는 방식입니다. 예를들어 함수A가 함수B를 호출하면 함수B가 완료될 때까지 함수A는 대기하는 상태를 말함

-비동기(Asynchronous) : 작업이 병렬로 실행되며, 이전 작업이 완료될 때까지 기다리지 않고 다음 작업을 진행. 호출자는 작업의 완료 여부와 상관없이 다음 코드를 실행하는 것

차이점 : 호출된 함수의 작업 완료 여부(call back)를 신경쓰는지의 여부 차이

Blocking 와 Non-blocking의 정의 및 차이점
-Blocking : 호출된 작업이 완료될 때가지 호출자가 대기하는 방식. 이 시간동안 호출자는 다른 작업을 수행불가 또한 호출자는 제어권을 호출된 작업으로 넘겨 제어권이 없다 호출된 작업이 다 끝나면 제어권을 돌려 받는 것
-Non-blokcing : 호출된 작업이 즉시 결과를 반환하며, 호출자는 작업의 완료 여부와 상관없이 다른 작업을 계속 수행할 수 있음. 제어권은 호출자가 계속 가지고 있음 호출된 작업으로 제어권이 넘어가지 않음

차이점 : 제어권이 어디에 있느냐 차이

시스템 설계 및 성능에 미치는 영향 분석
동기-블로킹 : 단순하지만 느린 작업으로 시스템이 비효율적
비동기-논블로킹: 높은 성능과 효율성을 제공, 그러나 복잡한 에러 처리와 상태관리 필요
동기-논블로킹 : 실행 순서 유지하면서도 일부 작업에서 대기를 피할 수 있음
비동기-블로킹 : 동기-블로킹과 성능차이 나지 않음, 다만 안티패턴 발생

각 접근 방식의 장단점 및 적합한 사용 시나리오
동기+블로킹 : 은행창구 – 순서대로 작업 수행
장점 : 프로세스가 단순하고 예측하기 쉬움
단점 : 대기 시간이 길어질 수 있어 효율성이 떨어짐

비동기 – 논블로킹 : 카카오톡

내가 카톡을 전송을 누르면 전송하는 동안 다른 작업을 할 수 있음

장점 : 자원을 효율적으로 사용하며, 대기 시간을 유용하게 활용할 수 있음

단점 : 프로세스 관리가 복잡

동기-논블로킹 : 셀프 주유소에서 결제를 진행하면서 주유 시작할 수 있음

장점 : 프로세스가 순서를 유지하면서 효율성을 높임

단점 : 일부 단계에서만 논블로킹이 가능하여 제한적임

비동기 – 블로킹 : 온라인 쇼핑몰에서 주문 후 화면을 계속 새로고침하여 배송상태 확인

장점 : 특정 잡업의 완료를 반드시 확인해야 할 때 유용

단점 : 자원이 비효율적이며 대기시간이 길어짐

비동기-논블록 :

```python
import asyncio

async def async_non_blocking_task(name):
    print(f"작업 {name} 시작 (대기 시간 2 초)")
    await asyncio.sleep(2)
    print(f"작업 {name} 종료")

async def main():
    tasks = [async_non_blocking_task(i) for i in range(3)]
    await asyncio.gather(*tasks)

if __name__ == "__main__":
    asyncio.run(main())
```

```
작업 0 시작 (대기 시간 2초)
작업 1 시작 (대기 시간 2초)
작업 2 시작 (대기 시간 2초)
작업 0 종료
작업 1 종료
작업 2 종료
```

비동기 – 블로킹 :

```python
import asyncio

async def async_blocking_task(name):
    print(f"작업 {name} 시작 (대기 시간 2 초)")
    await asyncio.sleep(2)
    print(f"작업 {name} 종료")

async def main():
    await async_blocking_task(0)
    await async_blocking_task(1)
    await async_blocking_task(2)

if __name__ == "__main__":
    asyncio.run(main())
```

```
작업 0 시작 (대기 시간 2초)
작업 0 종료
작업 1 시작 (대기 시간 2초)
작업 1 종료
작업 2 시작 (대기 시간 2초)
작업 2 종료
```

동기 – 블로킹:

```python
import time

def sync_blocking_task(name):
    print(f"작업 {name} 시작 (대기 시간 2 초)")
    time.sleep(2)
    print(f"작업 {name} 종료")

def main():
    sync_blocking_task(0)
    sync_blocking_task(1)
    sync_blocking_task(2)

if __name__ == "__main__":
    main()
```

```
작업 0 시작 (대기 시간 2초)
작업 0 종료
작업 1 시작 (대기 시간 2초)
작업 1 종료
작업 2 시작 (대기 시간 2초)
작업 2 종료
```

동기 – 논블록 :

```python
import threading
import time

def sync_non_blocking_task(name):
    print(f"작업 {name} 시작 (대기 시간 2 초)")
    time.sleep(2)
    print(f"작업 {name} 종료")

def main():
    threads = [threading.Thread(target=sync_non_blocking_task, args=(i,)) for
i in range(3)]
    for thread in threads:
        thread.start()
    for thread in threads:
        thread.join()

if __name__ == "__main__":
    main()
```

```
작업 0 시작 (대기 시간 2초)
작업 1 시작 (대기 시간 2초)
작업 2 시작 (대기 시간 2초)
작업 0 종료
작업 2 종료작업 1 종료
```

asyncio: 비동기 프로그래밍 라이브러리

비동기 작업을 효율적으로 관리하는 라이브러리다. 이벤트 루프라는 구조랄 사용하여 작업이 완료될 때까지 기다리지 않고 다른 작업을 실행하여 Non-blocking 방식으로 작동

threading: 동시성 프로그래밍을 위한 스레드 라이브러리

멀티스레딩을 통해 동시성을 지원하는 라이브러리다. 스레드는 독립적으로 작업을 수행하며, 여러 스레드가 동시에 실행되도록 함. OS에 의해 관리되는 독립적인 실행 흐름으로 한 스레드가 블로킹되더라도 다른 스레드가 동시에 잡업을 수행 함

동기- 블로킹

```python
def process_input():
    num = int(input("숫자를 입력하세요: "))
    result = num ** 2
    print(f"제곱 결과: {result}")

process_input()
```

```
숫자를 입력하세요: 10
제곱 결과: 100
```

도식화 :

시작 - 입력대기 – 입력 받은 후 계산 – 출력 – 종료

비동기 – 논블로킹

kk.py > ...

C:\bm\bongmins'cpp\과제 • 강조 표시한 항목 포함

```python
async def process_input():
    loop = asyncio.get_event_loop()
    num = await loop.run_in_executor(None, input, "숫자를 입력하세요: ")
    num = int(num)
    result = num ** 2
    print(f"제곱 결과: {result}")

asyncio.run(process_input())
```

```
숫자를 입력하세요: 10
제곱 결과: 100
```

도식화 :

시작 – 입력 대기중 다른 작업 수행가능 – 입력 완료시 계산 – 출력 – 종료

비동기 – 블로킹

```python
import asyncio

async def process_input():
    num = int(input("숫자를 입력하세요: "))
    result = num ** 2
    print(f"제곱 결과: {result}")

asyncio.run(process_input())
```

도식화 :

시작 – 이벤트 루프 시작 – 입력 이벤트 루프 블로킹 – 입력 후 계산 – 출력 – 종료

동기 – 논블로킹

```python
import sys
import select
import time

def process_input():
    print("숫자를 입력하세요: ", end='', flush=True)
    while True:
        if sys.stdin in select.select([sys.stdin], [], [], 0)[0]:
            num = sys.stdin.readline().strip()
            num = int(num)
            result = num ** 2
            print(f"\n제곱 결과: {result}")
            break
        else:
            print(".", end='', flush=True)
            time.sleep(0.5)

process_input()
```

도식화 :

시작 – 입력 여부 확인 – 없으면 다른 작업 수행 – 입력 받으면 계산 – 출력 – 종료

사용자 경험 및 시스템 리소스 사용 측면에서의 차이점 분석

동기 및 Blocking: 사용자 입력을 받을 때까지 프로그램이 완전히 멈춤 간단하지만 응답성이 떨어짐

비동기 및 Non-blocking: 입력 대기 중에도 다른 작업을 수행할 수 있어 응답성이 높다. 하지만 구현이 복잡함

동기 및 Non-blocking: 입력 대기 중 주기적으로 다른 작업을 수행하지만, CPU 사용률이 높아질 수 있음

비동기 및 Blocking: 비동기 코드 내에서 블로킹 호출로 인해 전체 프로그램이 멈춤 비효율적임

참고 문헌/블로그

Inpa Dev, 2023.05.02
https://inpa.tistory.com/entry/%F0%9F%91%A9%E2%80%8D%F0%9F%92%BB-%EB%8F%99%EA%B8%B0%EB%B9%84%EB%8F%99%EA%B8%B0-%EB%B8%94%EB%A1%9C%ED%82%B9%EB%85%BC%EB%B8%94%EB%A1%9C%ED%82%B9-%EA%B0%9C%EB%85%90-%EC%A0%95%EB%A6%AC