Effective Python

파이썬 코딩의 기술

Ch5. Concurrency & Parallelism
Ch6. Embedded Modules

장예훈

Ch5. 병행성과 병렬성

- 병행성(Concurrency): 컴퓨터가 여러 일을 마치 동시에 하듯이 수행

  Ex) CPU 코어가 하나인 컴퓨터가 실행 프로그램을 빠르게 변경하여

  동시에 여러 프로그램을 실행하는 것처럼 보이게 하는 것

- 병렬성(Parallelism): 실제로 여러 작업을 동시에 실행

  Ex) CPU 코어가 여러 개인 컴퓨터는 실제로 여러 프로그램을 동시에 실행,

  이 때 각 CPU 코어가 서로 다른 프로그램의 명령어(Instruction)을 실행하여 여러 프로그램을 같은 순간에 실행 시킴

1. 자식 프로세스를 관리하려면 subprocess를 사용하자

cf) 자식프로세스(child process): 부모 프로세스(parent process)에서 포크(fork)하여 생긴 프로세스

- **실전에서 단련된 자식 프로세스 실행과 관리용 라이브러리를 갖추고 있음**

  → 유틸리티 같은 다른 도구들을 연계하는 게 아주 좋은 언어

  (∵가독성과 유지보수성 확보가 좋기 때문)

- **파이썬으로 시작한 자식 프로세스는 병렬로 실행 가능**

  → CPU 코어를 모두 이용해 프로그램 처리량을 극대화 할 수 있음

  최근 자식프로세스를 관리하는 최선의 방법 = **subprocess** (과거에는 popen, pope2, os.exec 등)

- **자식 프로세스와 부모 프로세스는 파이썬 인터프리터와 독립적으로 실행**

  → 자식 프로세스의 상태는 파이썬이 다른 작업을 하는 동안 주기적으로 폴링(Polling)

1. 자식 프로세스를 관리하려면 subprocess를 사용하자

cf) 자식프로세스(child process): 부모 프로세스(parent process)에서 포크(fork)하여 생긴 프로세스

“

부모에서 자식 프로세스를 떼어낸다는 것

= 부모 프로세스가 자유롭게 여러 자식 프로세스를 병렬로 실행할 수 있음을 의미

”

▶ 예제

2. 스레드를 블로킹 I/O용으로 사용하고, 병렬화용으로는 사용하지 말자

cf) 스레드(Thread): 컴퓨터 프로그램 수행 시 프로세스 내부에 존재하는 수행 경로, 즉 일련의 실행 코드

- 파이썬은 전역 인터프리터 잠금(GIL, Global Interpreter Lock)이라는 메커니즘으로 일관성을 유지

- GIL

  : 소스코드를 바이트코드로 파싱, 컴파일, 스택 기반 인터프리터로 코드(바이트) 실행,

  프로그램 실행 시 인터프리터 지속,

  이 때 일관성 있는 상태 유지 → GIL 매커니즘

2. 스레드를 블로킹 I/O용으로 사용하고, 병렬화용으로는 사용하지 말자

cf) 스레드(Thread): 컴퓨터 프로그램 수행 시 프로세스 내부에 존재하는 수행 경로, 즉 일련의 실행 코드

- GIL은 선점형 멀티 스레드를 막음

C++이나 자바 같은 언어로 작성 된 프로그램에서 여러 스레드 사용

= 여러 CPU 코어를 사용하는 것

↕

파이썬으로 작성 된 프로그램에서 여러 스레드 사용

= 한 번에 한 스레드만 사용

병렬 연산 or 속도를 높여야 하는 상황에 부적합

▶ 예제

2. 스레드를 블로킹 I/O용으로 사용하고, 병렬화용으로는 사용하지 말자

cf) 스레드(Thread): 컴퓨터 프로그램 수행 시 프로세스 내부에 존재하는 수행 경로, 즉 일련의 실행 코드

- **멀티 스레드가 더 느리면 GIL은 왜 사용?**

  은 뒤에서 더 자세히 다루지만 대략적인 이유를 살펴보면…

1) 멀티 스레딩은 프로그램이 동시에 여러 작업을 하는 것처럼 보이게 하기에 유용
2) System Call(외부 환경과 대신 상호작용하도록 os에 요청하는 방법)할 때 일어나는 블로킹 I/O를 다루기 위해

2. 스레드를 블로킹 I/O용으로 사용하고, 병렬화용으로는 사용하지 말자

cf) 스레드(Thread): 컴퓨터 프로그램 수행 시 프로세스 내부에 존재하는 수행 경로, 즉 일련의 실행 코드

- 병렬 처리 시간이 순차적 처리 시간보다 5배나 짧음,

  하지만, 시스템 콜에서는 GIL의 부정적 영향(스레드를 하나씩만 사용하는)이 없음

- 왜?

  시스템 콜을 만들기 전에 GIL을 풀고 작업이 끝나면 다시 GIL을 얻음

- 그런데 왜 에러?

  Window 기반 코드가 아님

  cf) 윈도우 기반 코드로 바꾸면 시간차이 별로 안 나는데 왜냐하면 시스템 콜을 했지만 코드를 수정하는 과정 중에

  바이트코드(파이썬 스레드)가 들어가 있기 때문에 결국에는 GIL의 영향을 받음

3. 스레드에서 데이터 경쟁을 막으려면 Lock을 이용하자

GIL은 우리의 생각만큼 안전하지 않다!!

예제로 확인합시다

3. 스레드에서 데이터 경쟁을 막으려면 Lock을 이용하자

- **오류가 나는 이유**

  : 파이썬은 모든 스레드가 거의 동등한 처리 시간 동안 실행하는 것을 지향

  따라서, 기존의 실행 중이던 스레드를 중단하고 차례대로 다른 스레드를 재개

!! 문제는 !!

정확히 언제 스레드를 중단 할 지 모른다는 점

3. 스레드에서 데이터 경쟁을 막으려면 Lock을 이용하자

- 데이터를 Thread-Safe 하게 만들기 위해서 Lock을 사용

  → 여러 스레드가 동시에 실행 되도 현재 클래스 값 보호 가능

▶ 예제

4. 많은 함수를 동시에 실행하려면 코루틴을 고려하자

- 스레드를 사용한 동시성은 크게 세 가지 문제점이 있음

1) 순차 일관성(Sequential Consistency)을 보장하는 데 특별한 도구(Lock etc.)가 필요

→ 이해하기 어렵고 코드 확장, 유지보수하기 어려움

2) 스레드는 메모리가 많이 필요 (스레드 당 8MB 정도)

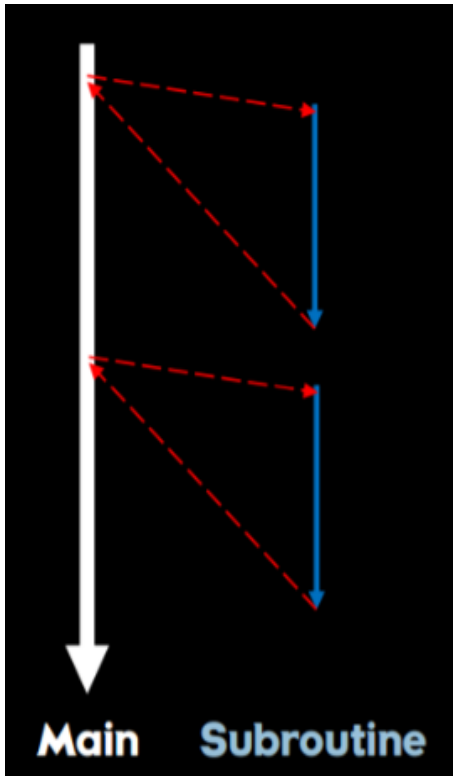
→ 수 십 개의 스레드 수준은 문제 X, but, 프로그램이 함수 수 천 개를 동시에 처리해야 할 때 문제

3) 스레드 시작 비용 많이 듦

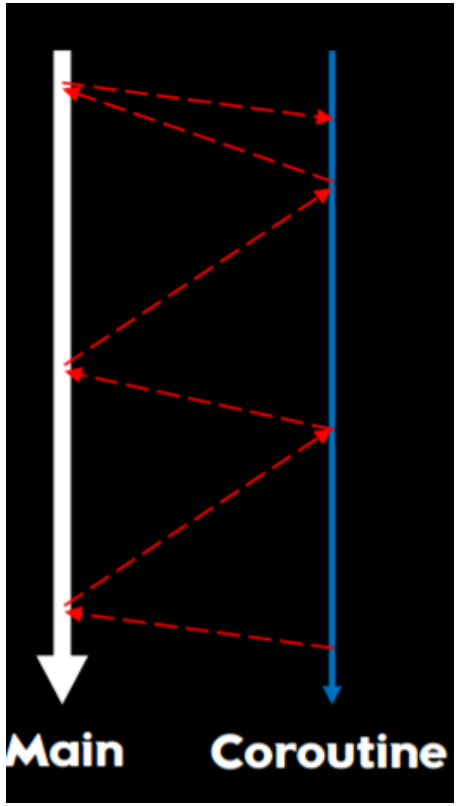
→ 스레드 스택을 위한 메모리 할당, 컨텍스트 생성 비용

4. 많은 함수를 동시에 실행하려면 코루틴을 고려하자

cf) 서브루틴과 코루틴의 차이점



〈서브루틴〉
특정 목적을 지닌 작업을 처리하는 코드의 모음



〈코루틴〉
메인 루틴과 코루틴 사이를 번갈아가며 실행

4. 많은 함수를 동시에 실행하려면 코루틴을 고려하자

- 코루틴이 제너레이터와 다른점

  → send 함수를 통해서 제너레이터 함수에 값을 전달 할 수 있음

- 코루틴은 프로그램의 핵심 로직 주변 환경과

  상호 작용하는 코드로부터 분리할 수 있는 강력한 도구

5. 진정한 병렬성을 실현하려면 concurrent.futures를 고려하자

- **파이썬 표준 구현 프로그램은 GIL로 인해 스레드 병렬화가 불가능**

  → 성능이 중요한 상황에서는 C언어로 모듈을 작성 가능

  but, 이미 파이썬으로 작성된 프로그램을 다시 C언어로 재작성 시 비용이 많이 들고,

  오류 발생 여부 불확실

- **문제점 해결**

  → multiprocessing: 자식 프로세스에서 추가적인 인터프리터를 실행하여 여러 CPU 코어를 병렬로 활용할 수 있음

▶ 예제

5. 진정한 병렬성을 실현하려면 concurrent.futures를 고려하자

<복잡한 과정>

1. 입력 데이터에서 map으로 각 아이템을 가져온다.
2. pickle 모듈을 사용하여 바이너리 데이터로 직렬화한다
3. 주 인터프리터 프로세스에서 직렬화한 데이터를 지역 소켓을 통해 자식 인터프리터 프로세스로 복사한다.
4. 자식 인터프리터에서는 pickle을 사용해서 데이터를 파이썬 객체로 역 직렬화한다.
5. gcd 함수가 들어 있는 파이썬 모듈을 임포트한다.
6. 다른 자식 프로세스를 사용하여 병렬로 입력 데이터를 처리한다.
7. 결과를 다시 바이트로 직렬화한다.
8. 소켓을 통해 바이트를 다시 복사한다.
9. 바이트를 부모 프로세스에 있는 파이썬 객체로 역 직렬화한다.
10. 여러 자식에 있는 결과를 하나의 결과로 합친다.

Ch6. 내장 모듈

파이썬은 '필요한 기능을 갖추는' 방법을 취함

Tip 정도의 내장 모듈을 설명하겠습니다

1. fuctools.wraps로 함수 데코레이터를 정의하자

- **데코레이터(Decorator)**

  : 감싸고 있는 함수를 호출하기 전이나 후에 추가로 코드를 실행하는 기능

  → 시맨틱 강조, 디버깅, 함수 등록 상황에서 유용

-
```
@mydecorator
def funcA():
```

- 데코레이터를 사용하면 디버거와 같이 객체 내부를 조사하는 도구가 이상하게 동작

  → 따라서 직접 데코레이터를 정의할 때 이런 문제를 피하려면 내장 모듈 functools의 warps데코레이터를 사용하면 됨

2. 재사용 가능한 try/finally 동작을 만들려면 contextilb와 with문을 고려하자

- Try/finally 구문에 상응하는 with문을 작성 가능

▶ 예제

- With 타깃 사용하기: ex) 파일에 쓰기를 수행한 후 해당 파일을 올바르게 닫는 코드를 짤 때

▶ 예제

3. 지역시간은 time이 아닌 datetime으로 표현하자

- **파이썬은 두 가지 시간대 변환 방법을 제공**

1) UTC(유닉스 기원 이후로 지나간 초로 시간을 표현

ex) UTC 15:00 – 7시

2) 지역시간

ex) 정오, 오전 8시 등

▶ 예제

4. 정밀도가 중요할 때는 Decimal을 사용하자

- **파이썬은 숫자 데이터를 다루는 코드를 작성하기에 아주 뛰어난 언어**

  → 정수타입은 현실적인 크기의 값을 모두 표현 할 수 있고, 허수 값을 표현하는 표준 복소수 타입도 제공

▶ 예제

4. 정밀도가 중요할 때는 Decimal을 사용하자

- 그러나 앞의 예제는 합리적이지 않음

  → 해결책은 바로 파이썬 내장 모듈 중 Decimal 클래스를 사용하는 것

- Decimal 클래스로 계산하면 근삿값이 아닌 정확한 값이 출력

▶ 예제

THANK
Y O U