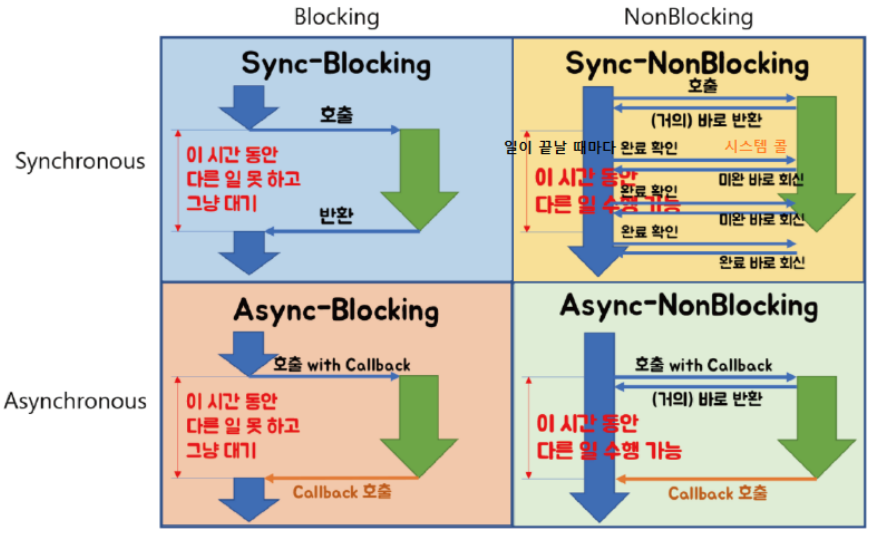

Blocking / Non-blocking

A 함수 내에서 B 함수를 호출하는 경우를 생각해보자

Blocking: A가 자신의 작업을 진행하다가 B의 작업이 시작되면 B의 작업이 끝날 때까지 기다렸다가 자신의 작업을 하는 것

Non-Blocking: B의 작업과 관련없이 자신의 작업을 하는 것

⇒ 다른 주체가 작업할 때 자신의 제어권이 있는지 없는지

Synchronous / Asynchronous

Synchronous: 동기. 작업을 동시에 수행하거나 동시에 끝나거나 끝나는 동시에 시작한다

Asynchronous: 비동기. 시작과 종료가 일치하지 않는다.

⇒ 결과를 돌려주었을 때 순서와 결과에 관심이 있는지 없는지

Blocking + Sync

B가 작업할 때 A는 제어권을 잃고 기다리다가, B가 수행 결과를 돌려주면 A는 결과를 받아 바로 처리한다.

ex) 콘솔 창 입력, 입력을 요청하면 입력이 들어올 때까지 블로킹 상태로 대기했다가 입력되는 즉시 처리한다.

Non-Blocking + Sync

B가 작업을 하는 동안 A는 자신의 제어권을 갖고 있으므로 자신의 작업을 수행할 수 있다. 하지만 B의 수행 결과를 받는 즉시 처리하기 위해서는 B의 작업 현황을 계속해서 알고 있어야 한다. A는 자신의 작업을 하면서 B의 작업이 끝났는지 계속해서 물어본다. B의 작업이 끝나면 결과를 받아 즉시 처리한다.

Blocking + Async

블로킹이기에 B가 작업하는 동안 A는 대기한다. A는 B의 작업에 관심이 없고, 결과를 받아도 바로 처리하지 않아도 된다.

Non-Blocking + Async

B가 작업하는 동안 제어권을 잃지 않고 A는 자신의 작업을 한다. B의 작업이 끝나서 결과를 받아도 A는 결과를 바로 처리하지 않아도 되기 때문에 자신의 작업을 마저 끝내고 결과를 처리할 수 있다.

IO 작업

IO 작업은 파일 입출력 뿐만 아니라 콘솔 입출력, 네트워크를 통해 데이터를 전송하고 받는 것도 IO 작업에 포함된다.

IO 작업은 유저 레벨에서 직접 수행할 수 없고, 실제 IO 작업은 커널 레벨에서 이뤄진다. User Process는 커널에게 요청하고, 커널에서 작업 후 반환되는 값을 받는다.

Blocking IO 모델

- IO 작업이 진행되는 동안 User Process는 중단(블록)된 상태로 대기한다.
- IO 작업이 CPU 자원을 거의 쓰지 않으므로 리소스 낭비가 심하다.
- 여러 클라이언트가 접속하는 서버를 블로킹 방식으로 구현하면 클라이언트 별로 별도의 쓰레드를 생성해야 한다. 접속자 수가 많아지면 쓰레드도 많아지고 이에따라 컨텍스트 스위칭 횟수도 증가하여 비효율적이다.

Non-Blocking IO 모델

- IO 작업이 진행되는 동안 User Process를 중단하지 않는다.
- 커널의 작업이 완료될 때까지 반복적인 시스템콜이 발생하기에 (sync-nonblocking 그림처럼) 리소스가 낭비된다.

IO 이벤트 통지 모델

- 수신 버퍼나 출력 버퍼의 이벤트를 통지한다.
- 입력 버터에 데이터가 수신되었다는 것을 알리거나, 출력 버퍼가 비었으니 데이터 전송이 가능하다는 것을 알리는 등