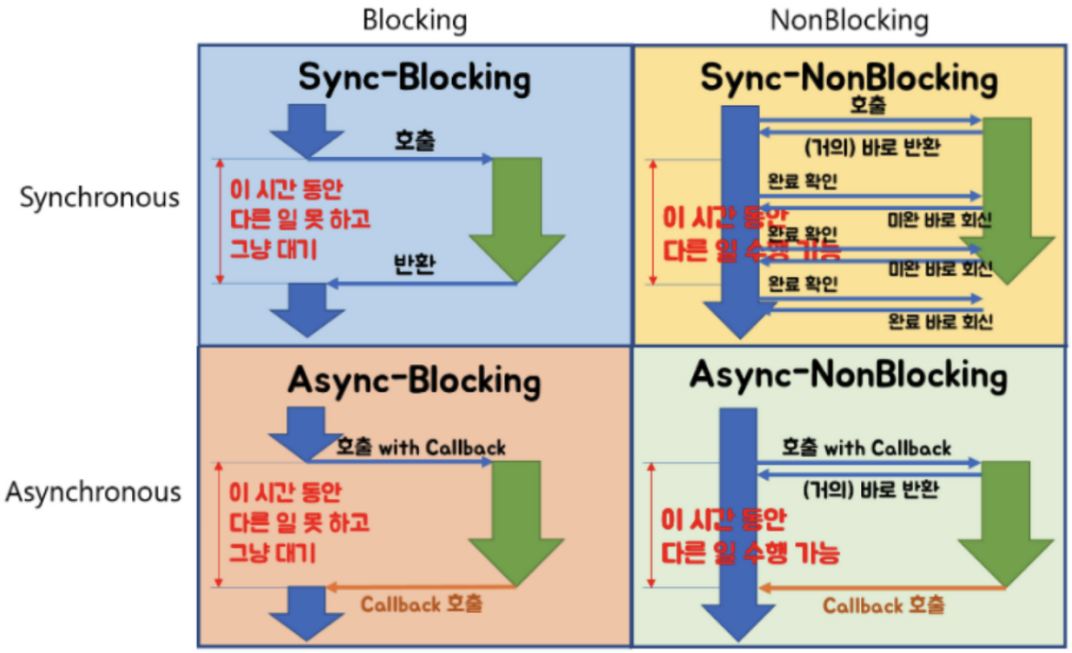
blocking, synchronous IO



block/ nonblock은 caller가 제어권을 갖는 시점과 관련이 있다.

blocking

caller가 callee가 일을 마치 때까지 제어권을 빼앗기고 대기하는 상황.

nonblocking

caller 가 callee가 일을 마치지 않았더라도 제어권을 들고있는 상황

caller가 다른 일을 할 기회가 있다.

Synchronous/Asynchronous는 호출되는 함수의 `작업 완료 여부`를 누가 신경쓰냐가 관심사
다.

synchronous

caller가 callee를 신경쓰면서 기다리는 상황. 블로킹의 경우 반환될 때까지 기다리고 있고, 논블로킹의 경우 완료 여부를 체크하면서 다른 일을 한다. callee는 결과의 순서에 관심이 있다.

```
//블록
callAndAwaitToDone()
// 관련된 일

//논블록
while (!other.isDone()){
  // 관련없는 다른일
}
// 호출결과랑 관련된 일
```

asynchronous

callee가 callback을 호출하여 결과를 통지하게된다.

callee는 작업 완료 여부에 관심이 없다.

---

block + sync

caller는 제어권을 넘겨준 상태로 대기하다가 callee의 결과를 직접 바로 처리

block + async

caller 는 제어권을 넘겨준 상태로 기다리긴 하는데, callee의 결과를 바로 처리하지 않아도 된다.

일부러 쓸 필요가 없는 타입. nonblock + async를 쓰려고 하다가 block하는 코드가 있어 어쩔 수 없이 block되는 경우에 나타날 수 있다고 한다.

nonblock + sync

caller 는 제어권을 잃지 않는다. 다만, callee의 결과를 바로 처리하기 위해 작업 완료 여부를 계속 체크한다.

- callee는 불릴때마다 계속 반환한다.

nonblock + async

caller 는 제어권을 잃지 않는다. B의 작업이 끝나도 바로 처리하지 않아도 상관 없다.

IO와 블로킹

- io시 블록하는 경우, 프로세스 내지는 스레드가 중단된다. 따라서 리소스 낭비가 심하다. 특히, 클라이언트가 여럿 접속하는 서버의 경우 클라이언트 별도 스레드가 필요하므로 context 스위칭이 증가한다.
- 논블록의 경우, 커널 작업이 완료될 때까지 프로세스 내지는 스레드가 작업을 하면서, 확인 시스템콜을 반복적으로 호출한다. 여러번 시스템콜을 호출하므로 context 스위치 비용이 든다.