# Blocking & NonBlocking

## Blocking/ NonBlocking & Synchronous/ Asynchronous

- Blocking

  자신의 작업을 진행하다가 다른 주체의 작업이 시작되면 다른 작업이 끝날 때까지 기다렸다가 자신의 작업을 시작하는 것

- NonBlocking

  다른 주체의 작업에 관련 없이 자신의 작업을 하는 것

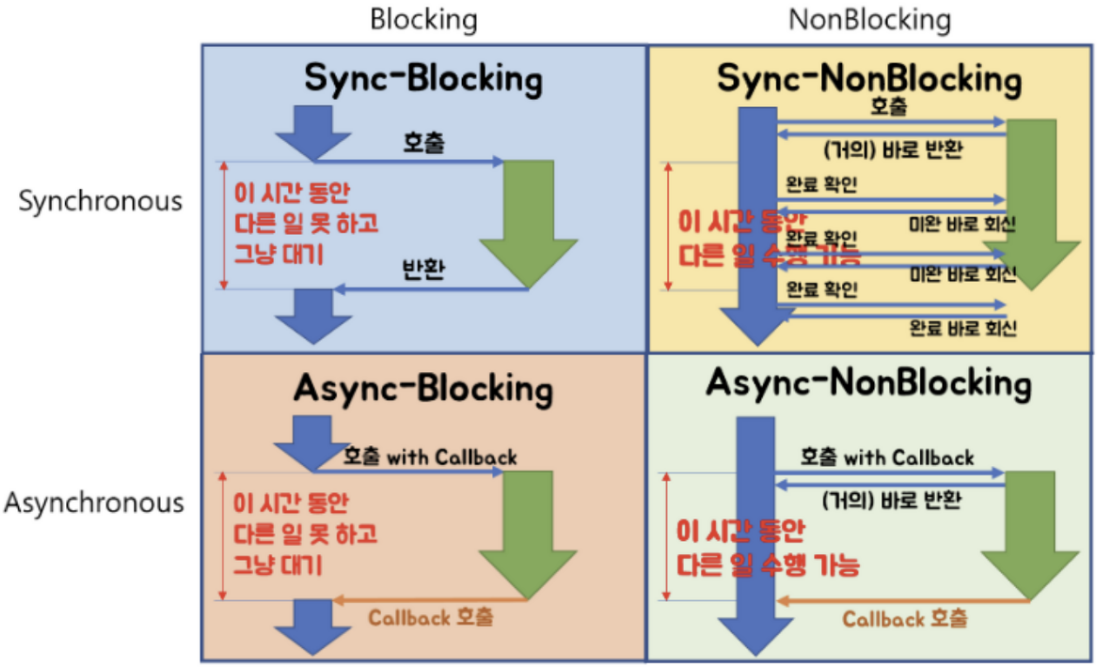
- Synchronous

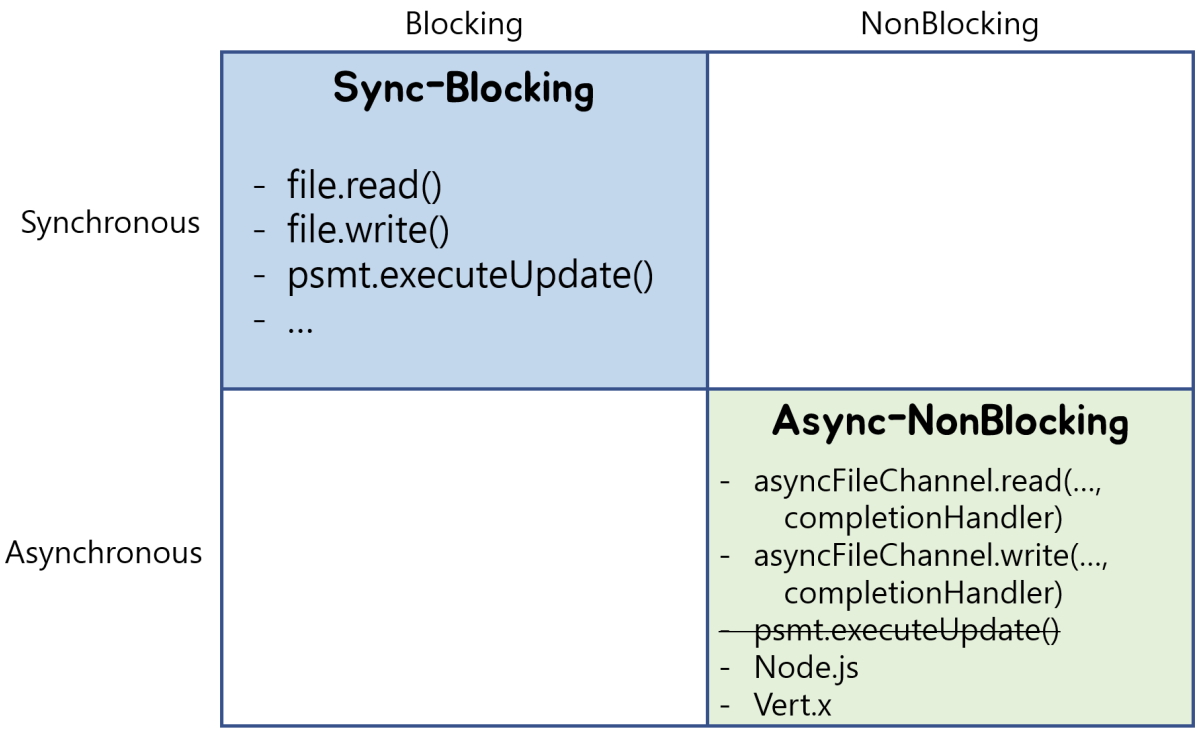
  요청이 들어온 순서에 맞게 하나씩 처리하는 것

- Asynchronous

  하나의 요청이 끝나기도 전에, 다른 요청을 동시에 처리할 수 있는 것

Synchronous와 Blocking → 무언가 기다리게 하는거 / Asynchronous와 Non-Blocking → 기다리지 않고 바로 처리된다는 점에서 유사하지만 각각 차이가 있다.






## Blocking/ NonBlocking

> 호출된 함수가 **호출한 함수에게 제어권을 건네주는 유무의 차이 (호출되는 함수가 바로 리턴을 하는 지 여부)**

- **Blocking** : 호출된 함수는 **할 일을 다 마칠때까지 제어권을 가지고 있다**. 즉, 호출된 함수가 자신의 작업을 완료할 때까지 리턴하지 않는다.
- **NonBlocking** : 호출된 함수는 **할 일을 마지치 않았어도 제어권을 바로 넘겨준다.** 즉, 호출된 함수가 바로 리턴하며 호출한 함수에게 제어권을 바로 넘겨주어 다른 일을 할 수 있도록 한다.

## Synchronous/ Asynchronous

> **호출되는 함수의 작업 완료 여부를 누가 신경 쓰는지의 차이**

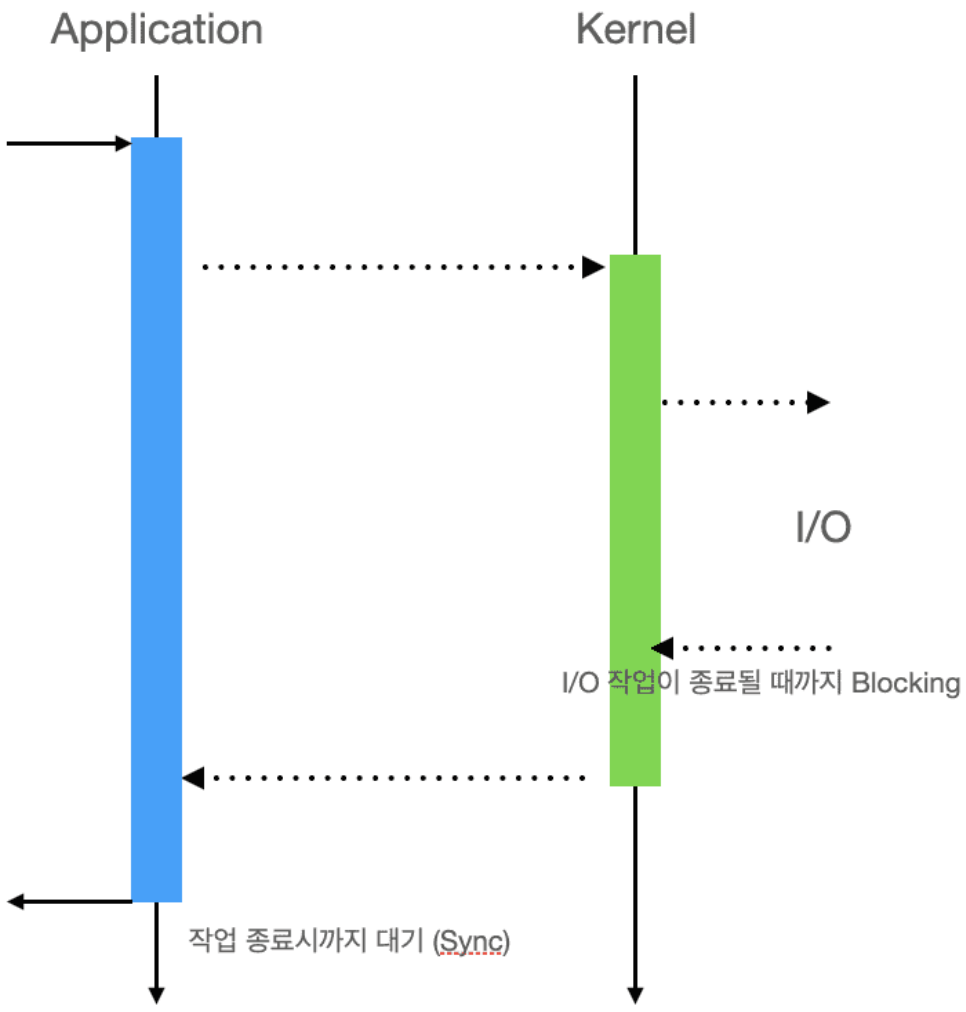
- **Synchronous** : **호출하는 함수가 작업 완료 여부를 확인한다.** 호출하는 함수는 호출된 함수의 작업 완료 여부 또는 작업 완료 후 리턴을 기다리고 있거나 주기적으로 체크한다.
- **Asynchronous** : **호출하는 함수가 작업 완료 여부를 신경 쓰지 않는다.** 호출된 함수는 수행 상태를 자신이 혼자 직접 신경 쓰면서 처리한다. 비동기는 호출 시 Callback을 전

달하여 작업의 완료 여부를 호출한 함수에게 답하게 된다. Callback이 오기 전까지 호출한 함수는 신경쓰지 않고 다른 일을 할 수 있다.

---

## Sync-Blocking

> 카페에 가서 음료가 나올 때 까지 가만히 서서 기다린다



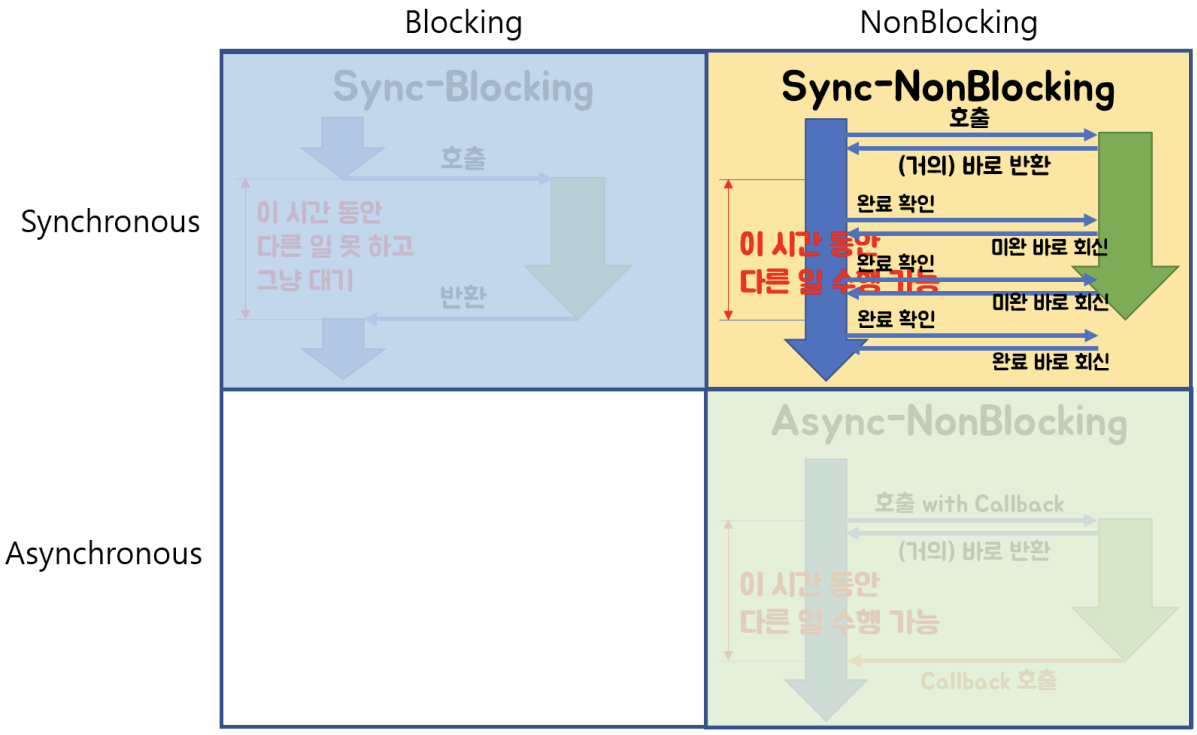

Sync-Blocking은 I/O작업이 수행되는 동안 다른 작업을 못하고 대기한다.

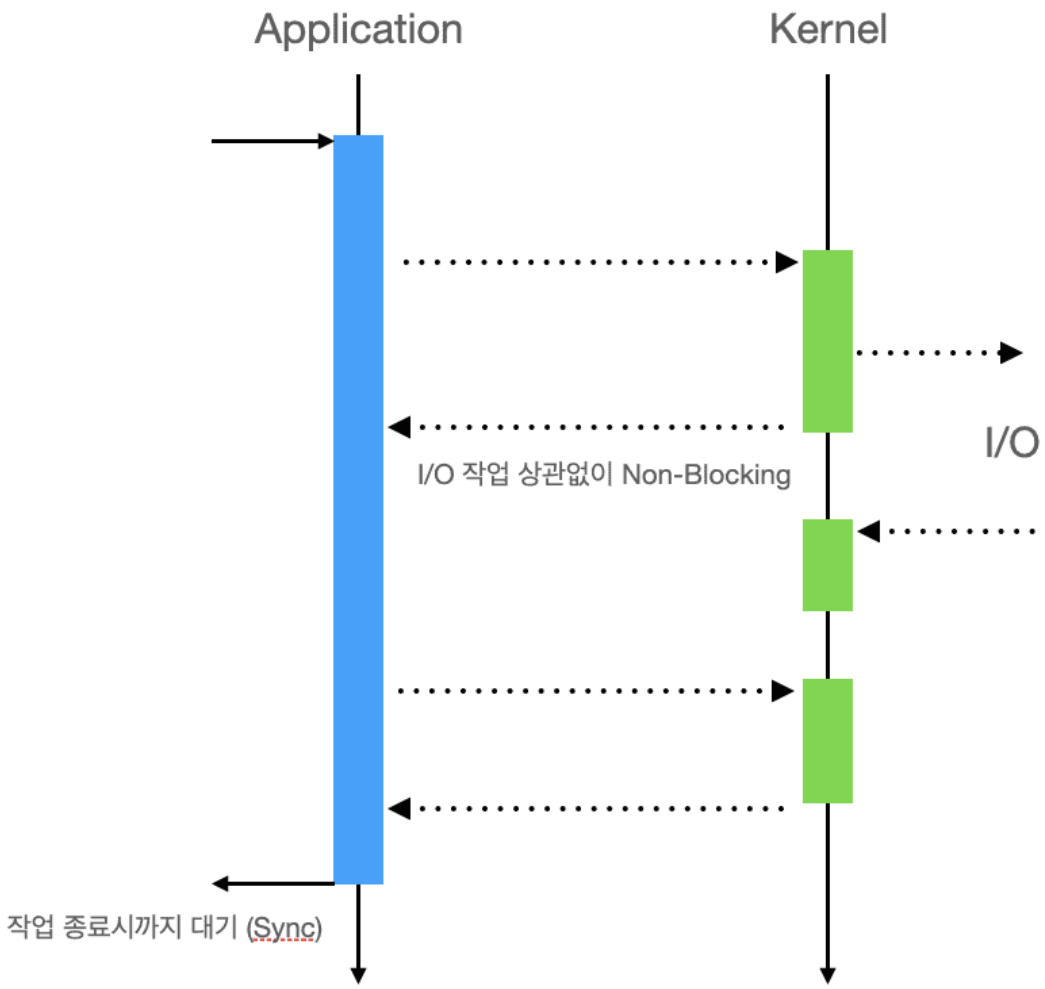
```
// 파일 입출력 함수
file.read()
file.wrtie()
```

```
System.out.print("메시지를 입력하세요 : ");
final Scanner scanner = new Scanner(System.in);
String message = scanner.nextLine();
System.out.println(message);
```

## Sync-NonBlocking

> 카페에 가서 음료를 시키고 앉아 있으면서 계속 언제 나오나 직접 체크






Sync-NonBlocking는 호출되는 함수는 바로 리턴하고, 호출하는 함수는 작업 완료 여부를 신경쓰는 것이다. 즉, NonBlocking 메서드 호출 후 바로 반환 받아서 다른 작업을 할 수 있게 되지만, 메서드 호출에 의해 수행되는 작업이 완료된 것은 아니며 호출하는 메서드가 호출되는 메서드 쪽에 작업 완료 여부를 계속 문의한다.

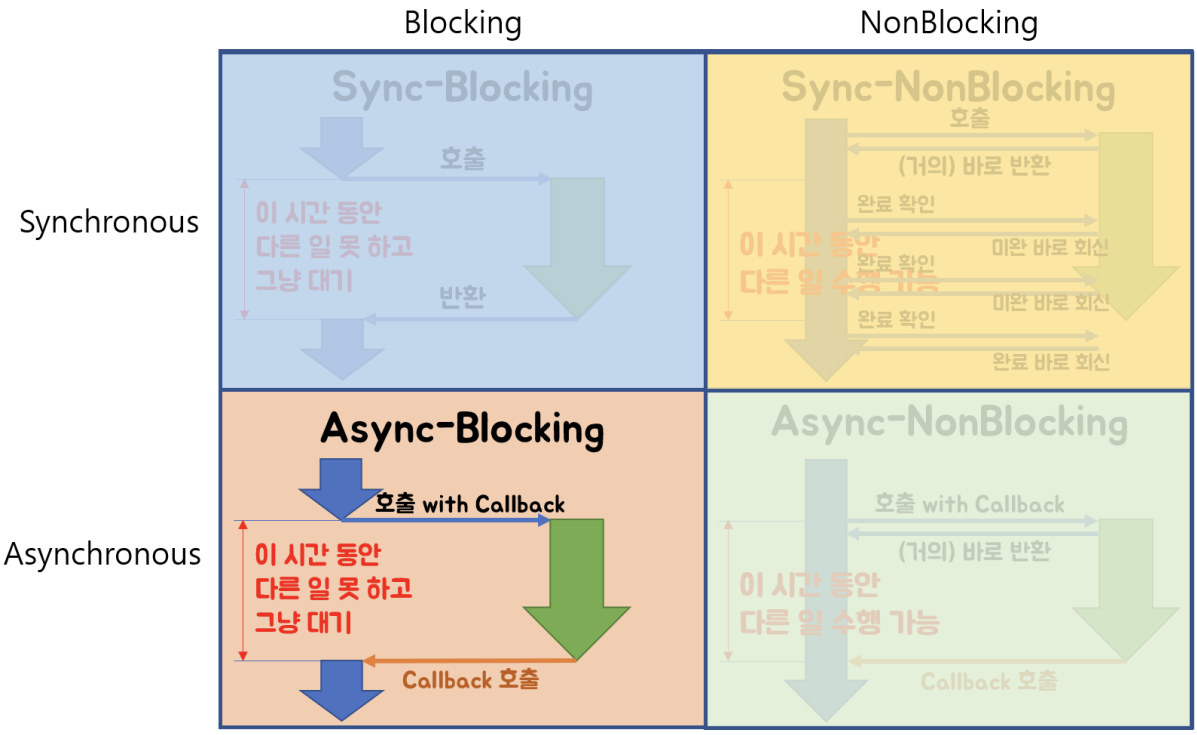
```
Future ft = asyncFileChannel.read(~~~);

while(!ft.isDone()) {
    // isDone()은 asyncChannle.read() 작업이 완료되지 않았다면 false를 바로 리턴해준다.
    // isDone()은 물어보면 대답을 해줄 뿐 작업 완료를 스스로 신경쓰지 않고,
    //     isDone()을 호출하는 쪽에서 계속 isDone()을 호출하면서 작업 완료를 신경쓴다.
    // asyncChannle.read()이 완료되지 않아도 여기에서 다른 작업 수행 가능
}

// 작업이 완료되면 작업 결과에 따른 다른 작업 처리
```

---

## Async-Blocking

> 카페에 가서 음료를 시켜서 진동벨은 주지만 딴거는 못하고 가만히 기다려야함

Async-Blocking는 호출되는 함수가 바로 리턴하지 않고 호출하는 함수는 작업 완료를 신경 쓰지 않는 것이다.




Sync-Blocking 방식과 성능적으로 별 차이가 없음. 즉, 별로 이점이 없어 일부러 사용하지는 않지만 의도지 않게 사용되는 경우도 있다. 대표적인 예가 Node.js와 MySQL의 조합이다.

Node.js쪽에서 async하게 해도 DB작업 호출 시 MySQL에서 제공하는 드라이버를 호출 → 이 드라이버가 Blocking 방식이다. 즉, Blocking-Async는 별다른 장점이 없어서 일부러 사용할 필요는 없지만, **Async-NonBlocking 방식을 쓰는데 그 과정 중에 하나라도 Blocking으로 동작하는 놈이 포함되어 있다면 의도하지 않게 Asnyc-Blocking으로 작동할 수 있다.**

---

## Async-NonBlocking

> 카페에 가서 음료를 시키면 진동벨을 주고 자리에 앉아 있다가 진동벨이 울리면 메뉴를 받는다.

일반적인 비동기 함수의 동작 방식과 같다. 함수가 다른 함수를 호출하고 바로 제어권을 반환받는다. 제어권을 반환받은 함수는 호출된 함수의 작업 결과에 상관없이 바로 다른 작업을

할 수 있기 때문에 다음 라인으로 넘어간다. 호출된 함수는 작업을 수행하고 그 결과를 콜백 함수로 반환한다.

---

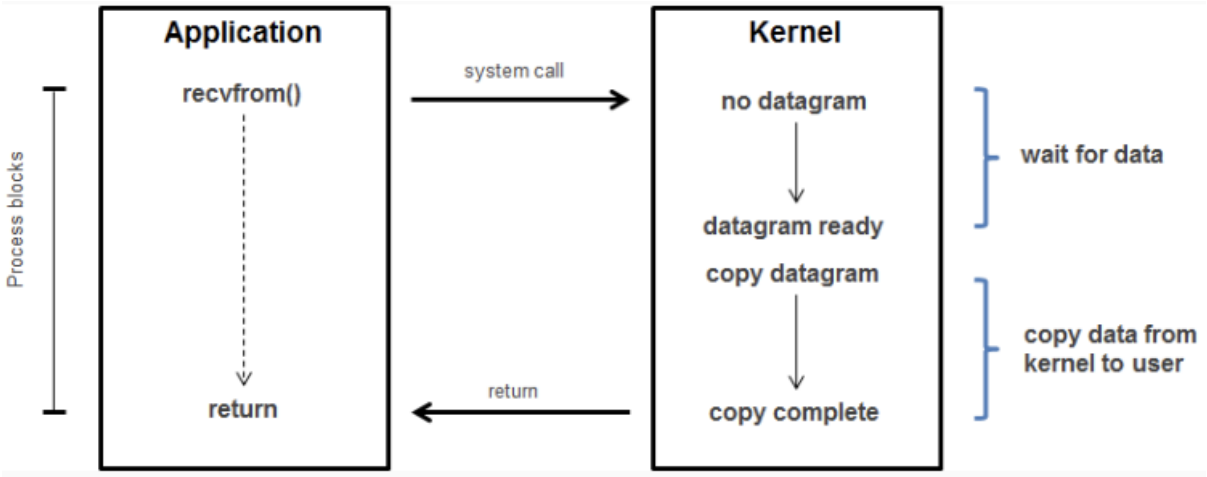
## Blocking I/O & Non-Blocking I/O

> I/O 작업은 유저 레벨에서 실행될 수 없고 커널 레벨에서만 실행 될 수 있다.

네트워크 에서의 소켓 recv/send가 I/O에 해당. 컴퓨터끼리 통신할 때 이루어지는 데이터의 입력과 출력을 받는 과정

### 1. Blocking I/O 모델

> I/O 작업을 요청한 유저 프로세스는 **작업을 중단한 채 반환을 기다리면서 대기**하는 방식



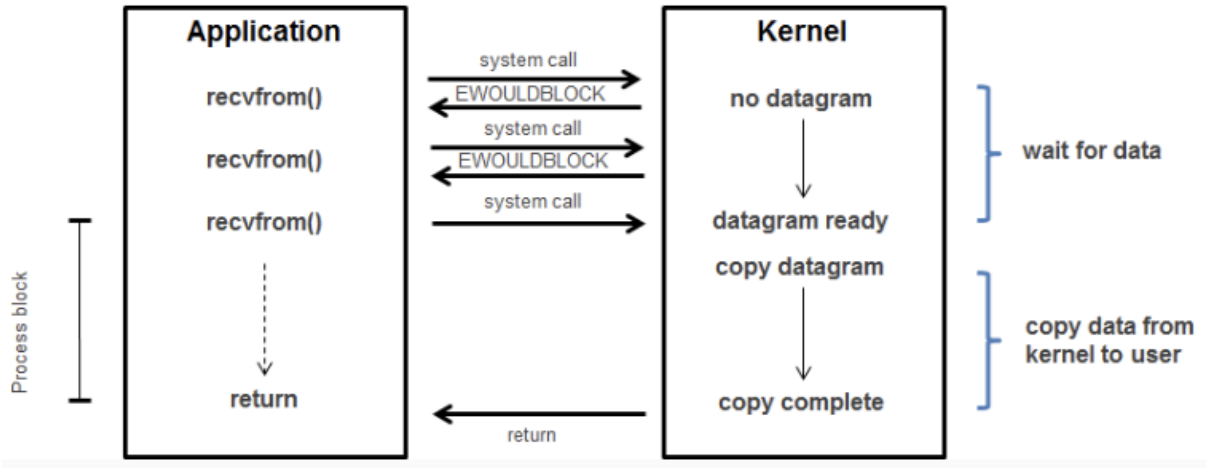

1. 유저가 커널에 작업을 요청하고 데이터가 입력될 때까지 대기한다.
2. 커널이 작업을 완료하면 작업 결과 받아옴

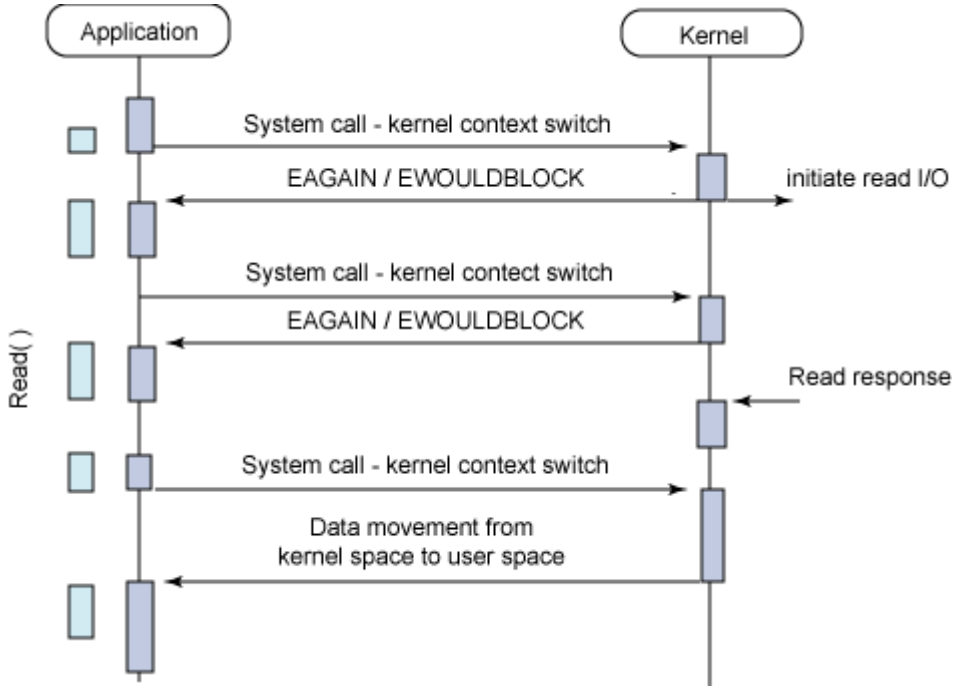
- I/O 작업이 진행되는 동안 유저 프로세스는 자신의 작업을 중단한 채 대기
- 리소스 낭비가 심함 (I/O 작업이 CPU 자원 거의 사용 X)

ex) 메시지를 보냈는데 응답이 올 때 까지 대기하며 메시지 사용 불가

### 2. Non-Blocking I/O Model

## I/O 작업이 진행되는 동안 **유저 프로세스의 작업을 중단하지 않음**



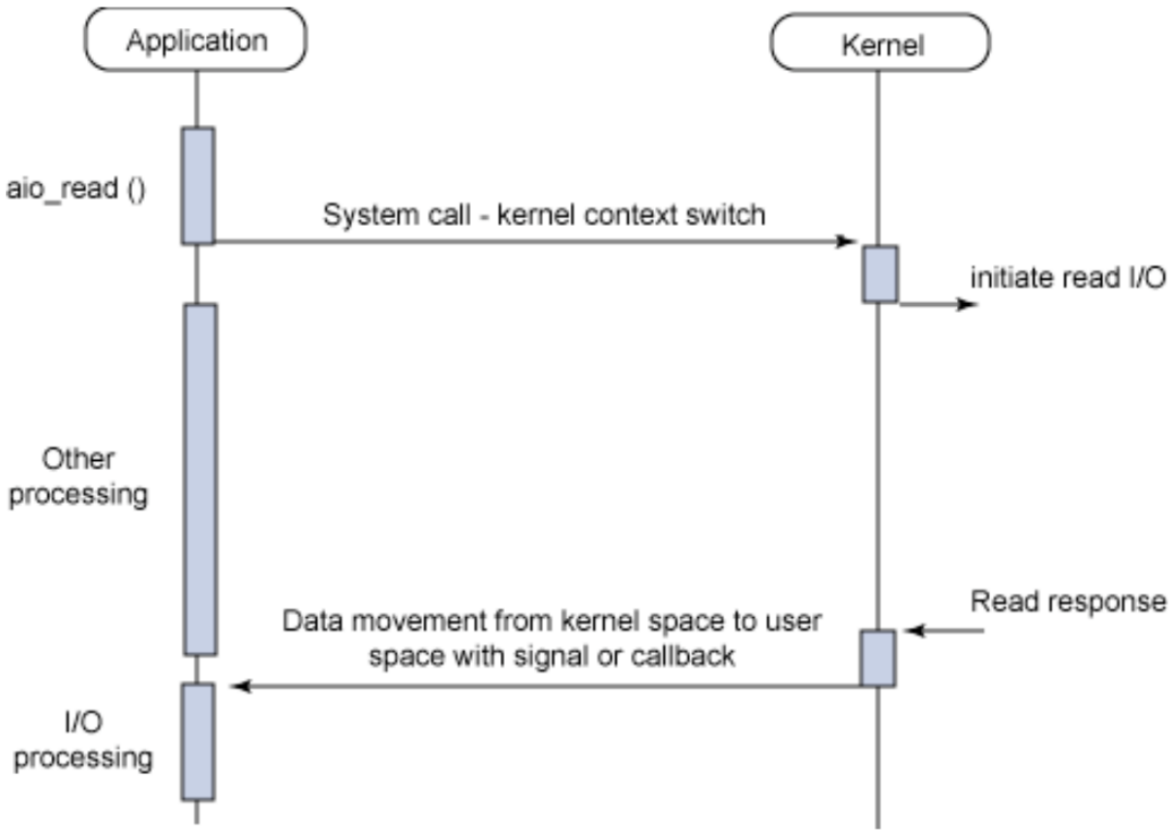




1. 유저 프로세스가 recvfrom 함수를 호출한다. 즉, 커널에게 해당 Socket으로 부터 data를 받고 싶다고 요청한다.
2. Kernel은 이 요청에 대해서 곧바로 recvBuffer를 채워서 보내지 못하므로 “EWOULDBLOCK”을 리턴함
3. Blocking 방식과 달리 유저 프로세스는 다른 작업을 진행할 수 있다.
4. recvBuffer에 user가 받을 수 있는 데이터가 있는 경우, 버퍼로 부터 데이터를 복사하여 받아온다.
5. recvfrom함수는 빠른 속도로 data를 복사한 후, 복사한 데이터의 길이와 함께 반환한다.

**I/O의 진행시간과 관계가 없기 때문에 애플리케이션은 중지 없이도 I/O 작업을 진행할 수 있다. 하지만 반복적으로 시스템 호출이 발생되기 때문에 불필요한 경우에도 자원이 쓰이므로 낭비가 발생한다.**

## 3. I/O 이벤트 통지 모델

> 수신 버퍼나 출력 버퍼의 이벤트를 통지한다




수신 버퍼의 이벤트는 말 그대로 **입력 버퍼에 데이터가 수신되었다는 것을 알려주는 것**이고, 반대로 출력 버퍼의 이벤트는 **출력 버퍼가 비었으니 데이터 전송이 가능하다는 상황을 알려주는 것**을 의미한다.

I/O 이벤트 통지 방식에 따라 동기 모델, 비동기 모델로 나눌 수 있다.

비동기 모델에서 유저 프로세스는 I/O 작업이 진행되는 동안에는 관심이 없고 자신의 일을 하다가 **이벤트 핸들러에 의해 알림이 오면 처리하는 방식**이다. Sync 방식과 반대로 커널이 주체적으로 담당하여 진행하고, 유저 프로세스는 **수동적인 입장에서 통지가 들어오면 그때 I/O 작업을 진행**한다.