# 다중 접속 서버 구현 방식

appti · 2024년 3월 16일



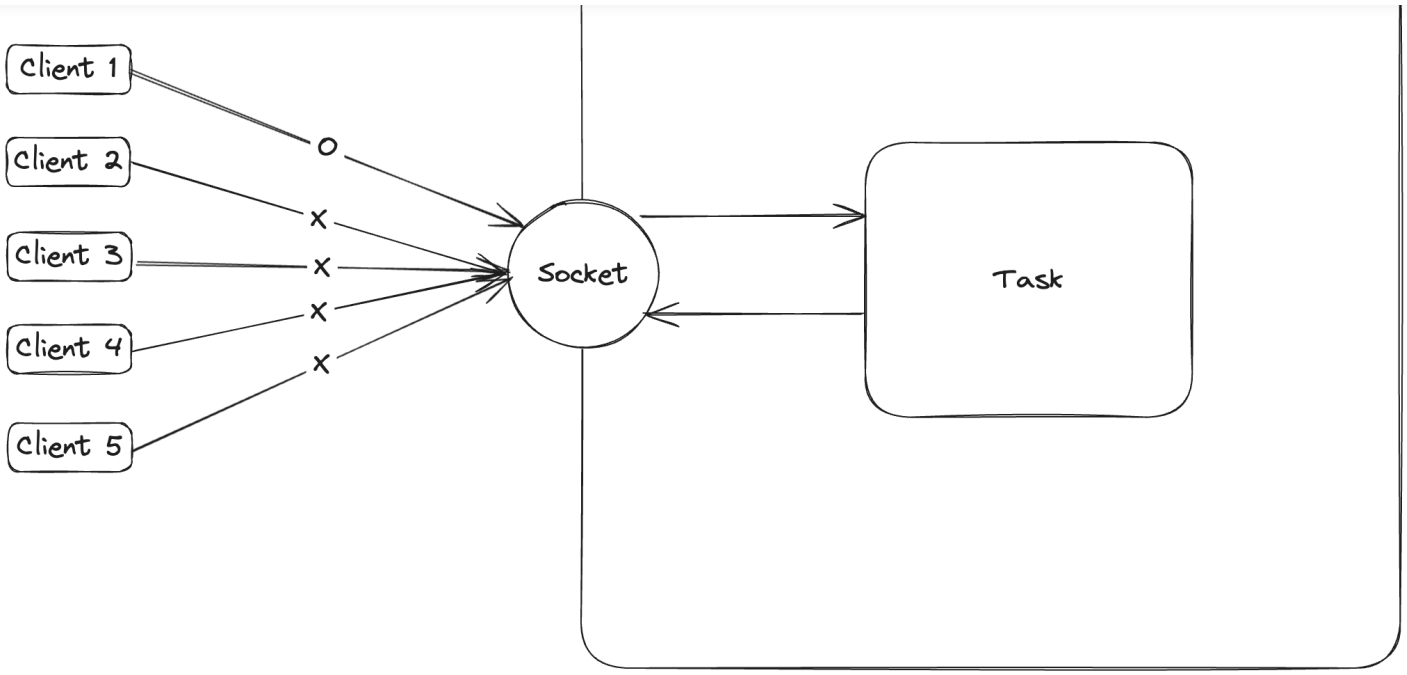
## 서론

단일 접속 서버 구현 방식과, 다중 접속 서버 구현 방식 세 가지 방식을 비교해보고자 합니다.

## 단일 접속 서버

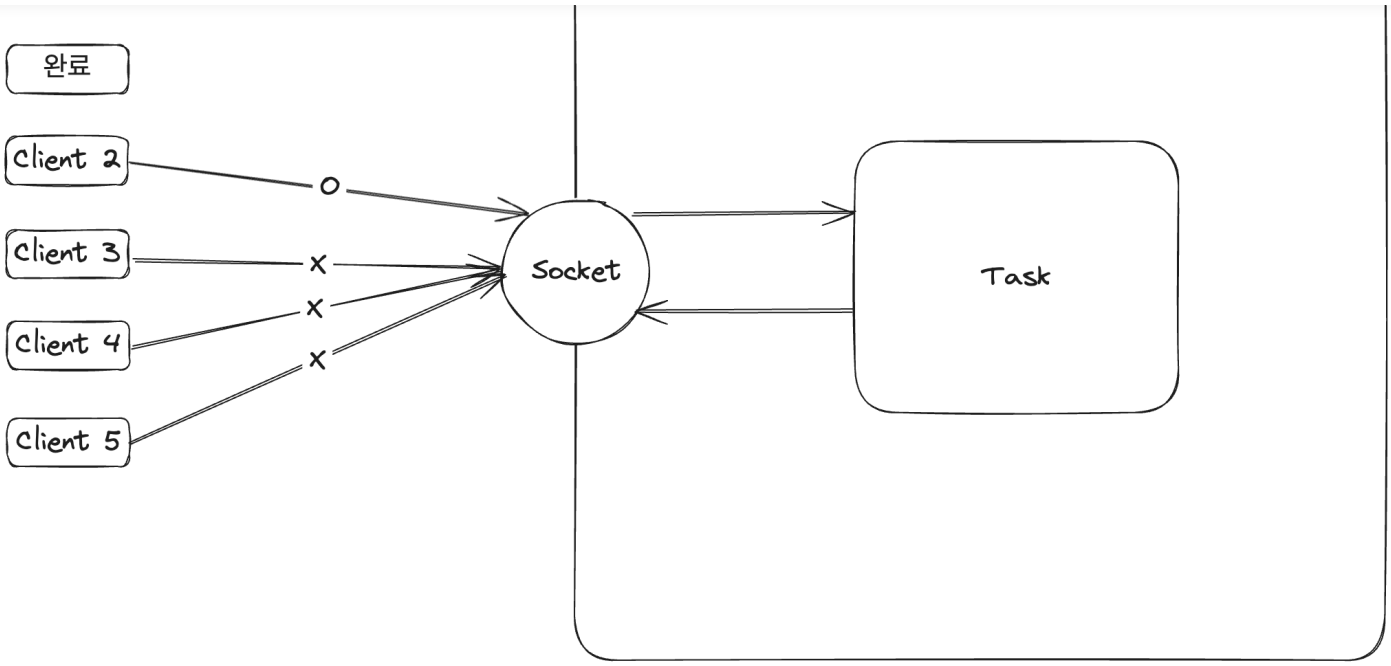
단일 접속 서버의 경우, 다음과 같이 동작합니다.

appti




1. 서버가 소켓을 열고 Listen 합니다.
2. 여러 클라이언트가 서버에 연결 요청을 합니다.
3. 서버는 클라이언트와의 연결 요청을 수락하고, 클라이언트와 연결된 소켓을 통해 로직을 수행합니다.
   3-1. 이 때 서버는 클라이언트와의 작업이 끝날 때 까지 다른 클라이언트의 소켓 연결 요청을 수행할 수 없습니다.

appti




4. 클라이언트의 요청에 따른 로직이 종료되면, 소켓 연결이 종료됩니다.
5. 서버는 다시 소켓을 열고 Listen 합니다.
6. 서버가 다시 여러 클라이언트 중 하나와 소켓 연결합니다.

장단점은 다음과 같습니다.

- 장점
  - 구현이 단순합니다.
- 단점
  - 여러 클라이언트의 요청을 동시에 처리할 수 없습니다.
    - 자신보다 먼저 소켓이 연결된 클라이언트의 동작이 끝날 때 까지 대기해야 합니다.

이러한 문제를 개선하기 위해 다중 접속 서버 개념이 등장하게 되었습니다.

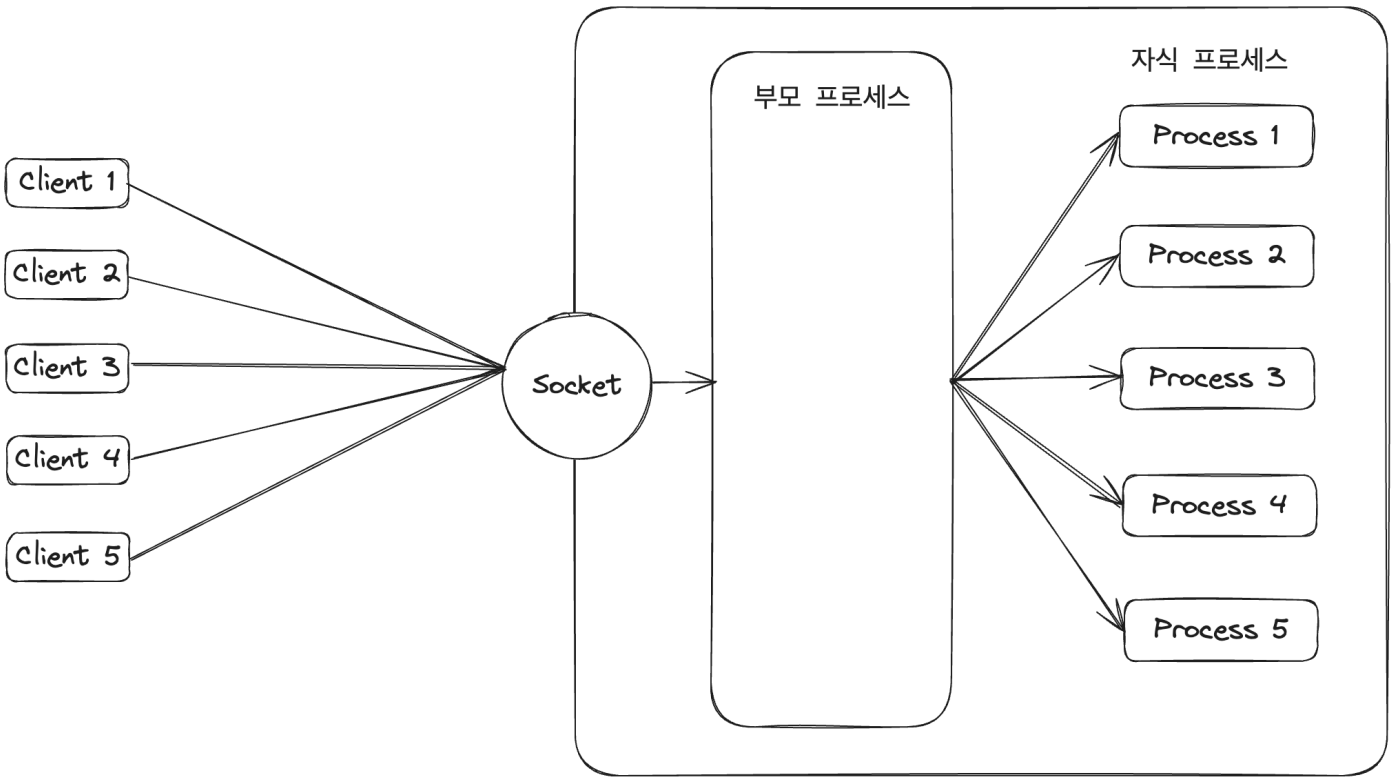
# 다중 접속 서버 동작 방식

다중 접속 서버 구현 방식에는 다음과 같이 크게 세 가지 방식이 존재합니다.

- 멀티 프로세싱 기반
- 멀티 스레딩 기반
- 멀티 플렉싱 기반

## 멀티 프로세싱 기반

멀티 프로세싱 기반의 다중 접속 서버는 다음과 같이 동작합니다.




1. 서버가 소켓을 열고 Listen 합니다.
2. 여러 클라이언트가 서버에 연결 요청을 합니다.
3. 서버는 클라이언트와의 연결 요청을 수락하고, 자식 프로세스를 생성합니다. 이후 생성한 자식 프로세스에게 클라이언트와 연결된 소켓을 전달합니다.
4. 자식 프로세스는 전달받은, 클라이언트와 연결된 소켓을 통해 동작합니다.

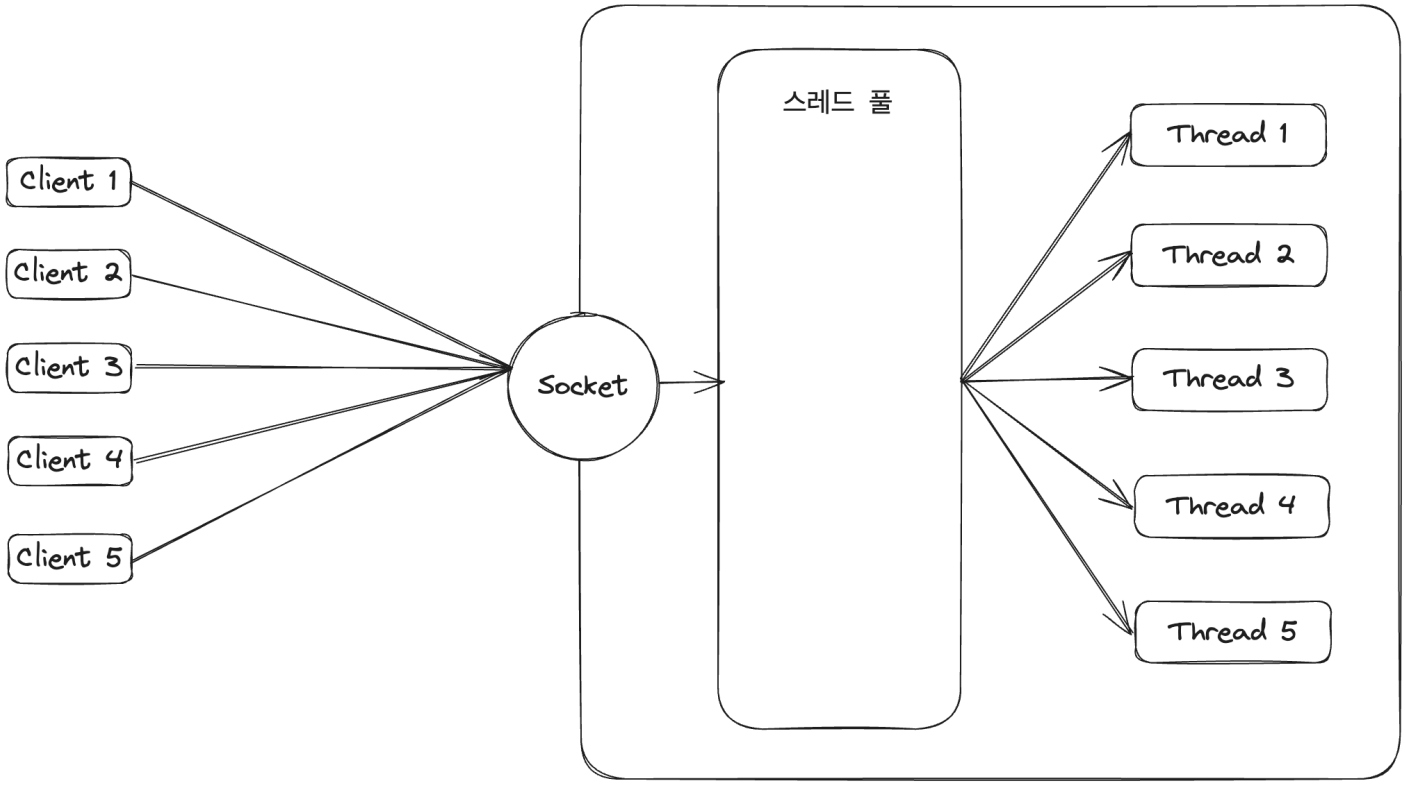
장단점은 다음과 같습니다.

- 장점
  - 프로세스는 운영 체제에서 독립적인 프로그램이기 때문에 각 클라이언트의 요청을 독립적으로 처리할 수 있습니다.
- 단점
  - 프로세스 생성은 운영 체제에서 비용이 비쌉니다.
  - 각각의 프로세스는 독립적이기 때문에 데이터를 공유하기 힘듭니다.

appti

## 멀티 스레드 기반

멀티 스레드 기반의 다중 접속 서버는 다음과 같이 동작합니다.




1. 서버가 소켓을 열고 Listen 합니다.
2. 여러 클라이언트가 서버에 연결 요청을 합니다.
3. 서버는 클라이언트와의 연결 요청을 수락하고, 스레드 풀을 통해 스레드를 생성합니다. 이후 생성한 스레드에게 클라이언트와 연결된 소켓을 전달합니다.
4. 스레드는 전달받은, 클라이언트와 연결된 소켓을 통해 동작합니다.

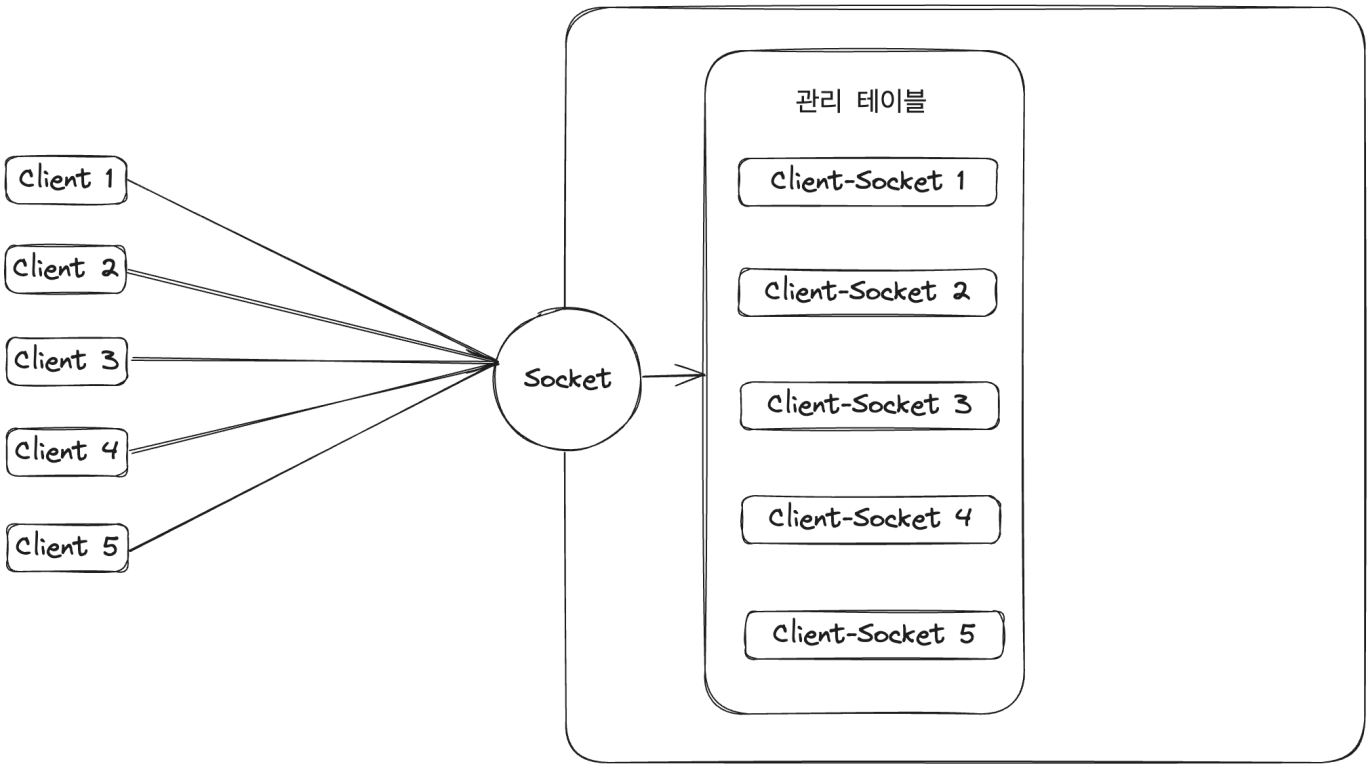
장단점은 다음과 같습니다.

- 장점
  - 스레드 생성 비용은 프로세스보다 저렴합니다.
  - 스레드 간 컨텍스트 스위칭 비용은 프로세스보다 저렴합니다.
  - 스레드의 경우 공유하는 메모리가 있기 때문에 데이터를 공유하기 쉽습니다.
- 단점

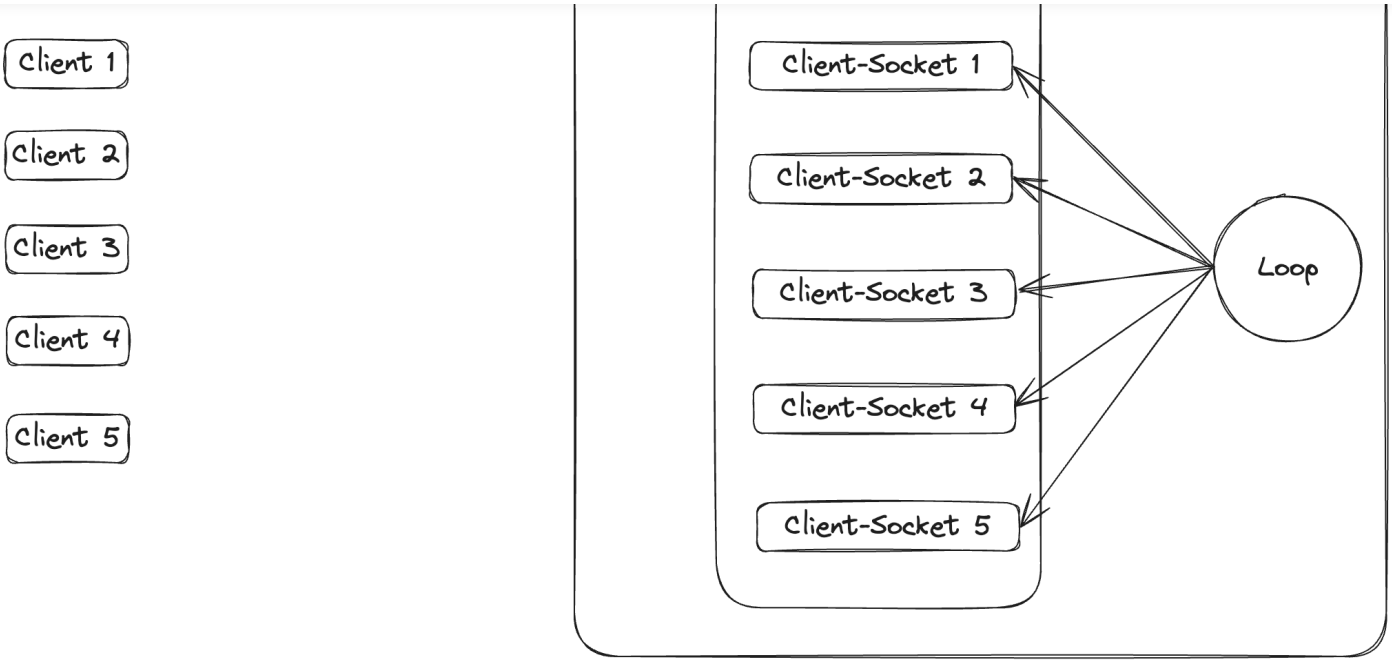
- 내용당 드래픽을 처리할 수 없습니다.

## 멀티 플렉싱 기반

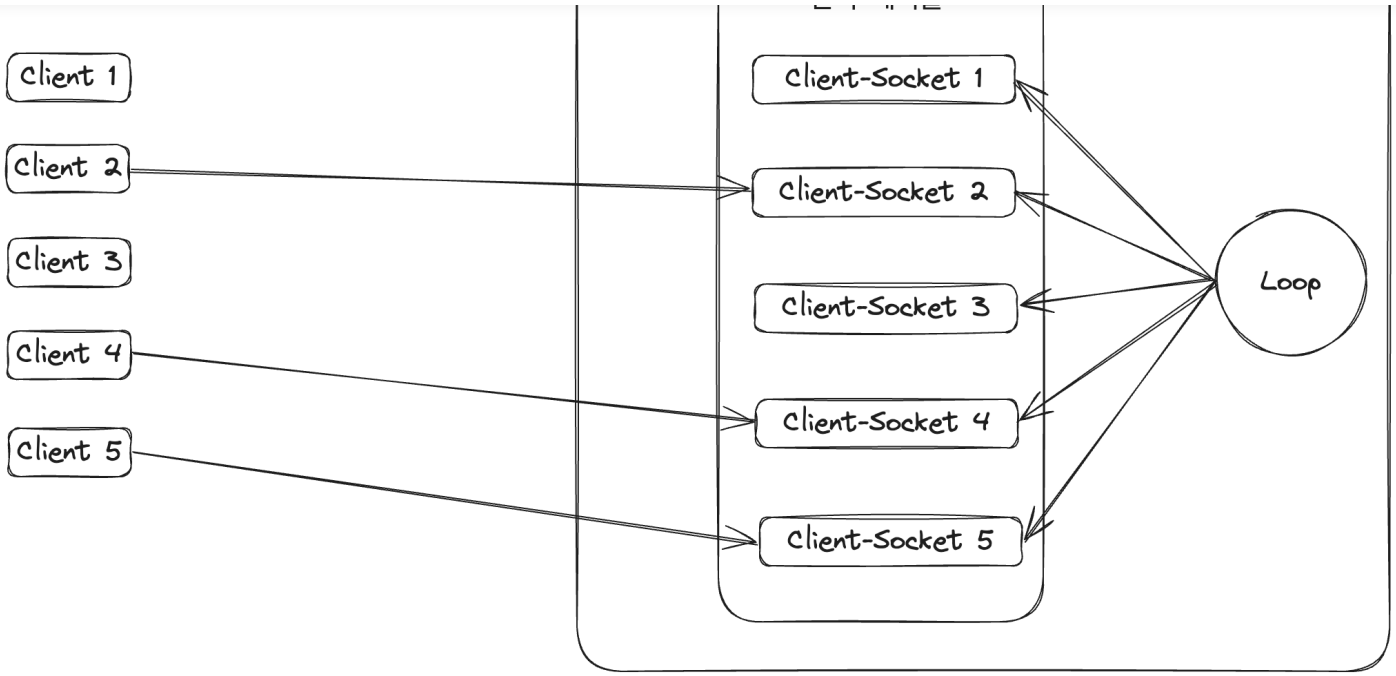
멀티 플렉싱 기반의 다중 접속 서버는 다음과 같이 동작합니다.



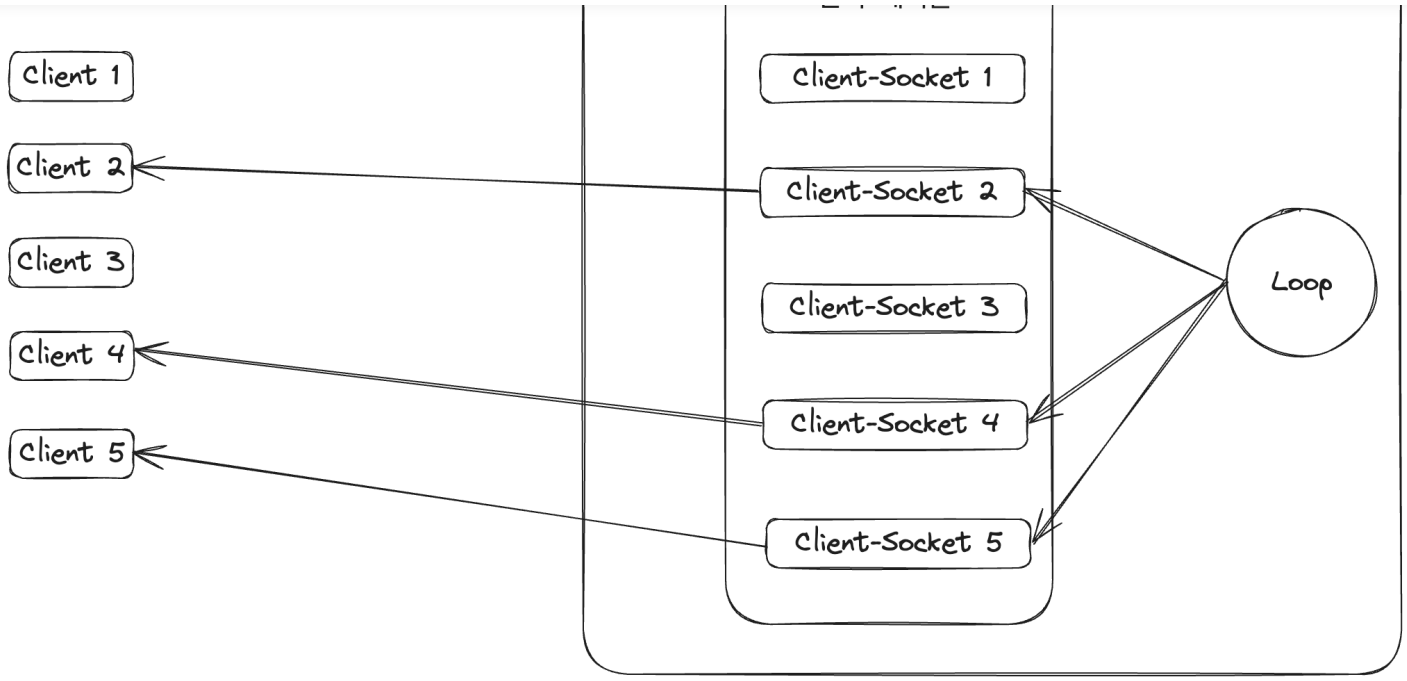

1. 서버가 소켓을 열고 Listen 합니다.
2. 여러 클라이언트가 서버에 연결 요청을 합니다.
3. 서버는 요청한 모든 클라이언트와의 연결을 수락하고, 연결된 소켓을 별도의 관리 테이블에 저장합니다.




4. 서버는 내부적으로 연결된 소켓에서 발생하는 이벤트를 무한 반복문을 통해 감시합니다.
   4-1. 이 때의 이벤트는, 클라이언트에서 데이터를 소켓에 전송해 커널 영역에 데이터가 세팅되어 서버와 연결된 소켓이 read가 가능해진 상태가 되는 이벤트를 의미합니다.
   4-2. 서버는 select, poll, epoll로 이벤트를 감지합니다.




5. 클라이언트가 연결된 소켓에게 데이터를 전송합니다.
   5-1. 데이터가 커널 영역에 저장되어 read 호출이 가능해지면 이벤트가 발생합니다.
   5-2. 서버는 계속 무한 반복문을 통해 이벤트를 감시합니다.




6. 서버에서 무한 루프를 통해 관리 테이블에 저장한 소켓의 이벤트를 감시하다가, 이벤트가 발생한 것을 감지하고 클라이언트의 요청을 처리합니다.
7. 서버에서 요청을 처리한 이후 클라이언트와 연결된 소켓을 통해 해당 클라이언트에게 응답을 반환합니다.

장단점은 다음과 같습니다.

- 장점
  - 하나의 스레드에서 여러 소켓을 처리할 수 있으므로 대용량 트래픽을 감당할 수 있습니다.
  - 별도의 스레드, 프로세스를 만들지 않기 때문에 생성 비용이나 컨텍스트 스위칭과 같은 비용이 필요하지 않습니다.
- 단점
  - 운영 체제에 따라 소켓(파일 디스크립터)의 제한이 존재합니다.
  - 발생하는 이벤트를 감지하기 위해 서버에서 계속 모든 클라이언트와 연결된 소켓을 감시해야 합니다.

# 이벤트 기반 동작 방식

멀티 플렉싱 기반 다중 서버를 사용할 때, 서버가 클라이언트와 연결된 소켓에서 발생하는 이벤트를 감지한다고 언급했었습니다.

이러한 이벤트에 대해 살펴보고, 이벤트 기반으로 동작 방식을 살펴보고자 합니다.
이 때, 동작 시점은 클라이언트의 연결 요청을 서버 소켓이 수락한 이후입니다.

위해서는 커널 영역에서 유저 영역으로 데이터의 복사가 필요합니다.

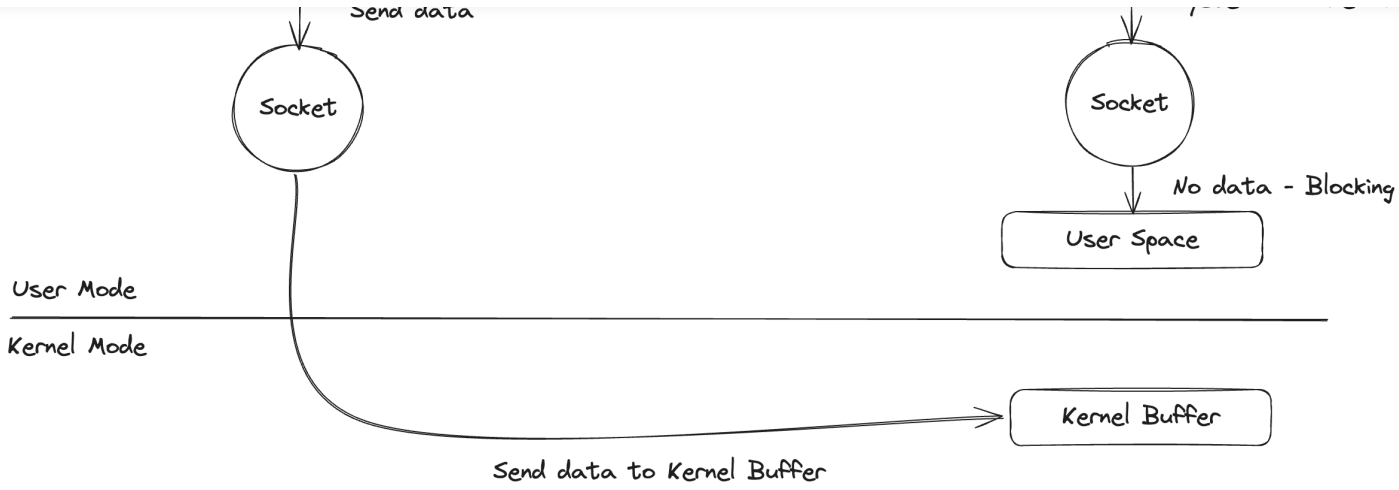
소켓은 커널 영역에서 동작하지만, 서버(프로세스)에서 소켓의 데이터를 읽을 때에는 유저 영역에서 접근하기 때문입니다.

즉, 소켓의 데이터는 커널 영역에서 관리되는데 서버는 유저 레벨로 실행되기 때문에 커널 영역의 데이터를 읽을 수 없으므로, 데이터를 읽기 위해서는 커널 영역에 있는 데이터가 유저 영역으로 복사가 되어야 한다는 것입니다.

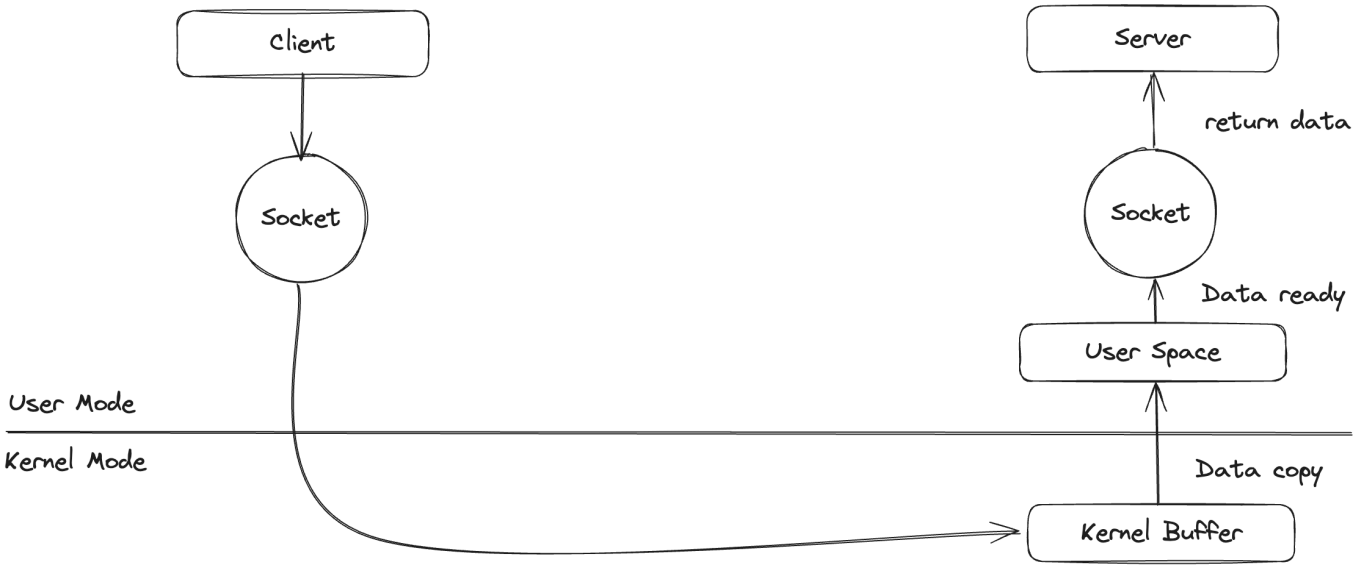
## 멀티 프로세서 기반 & 멀티 스레드 기반

이를 기반으로 동작하는 멀티 프로세서 기반 & 멀티 스레드 기반의 동작 과정은 다음과 같습니다.


1. 서버가 소켓에 시스템 콜인 read를 호출합니다.
2. 소켓은 유저 영역에서 읽을 수 있는 데이터를 확인합니다. 혀재에는 데이터가 없으므로 데이터를 준비될 때 까지 대기합니다.

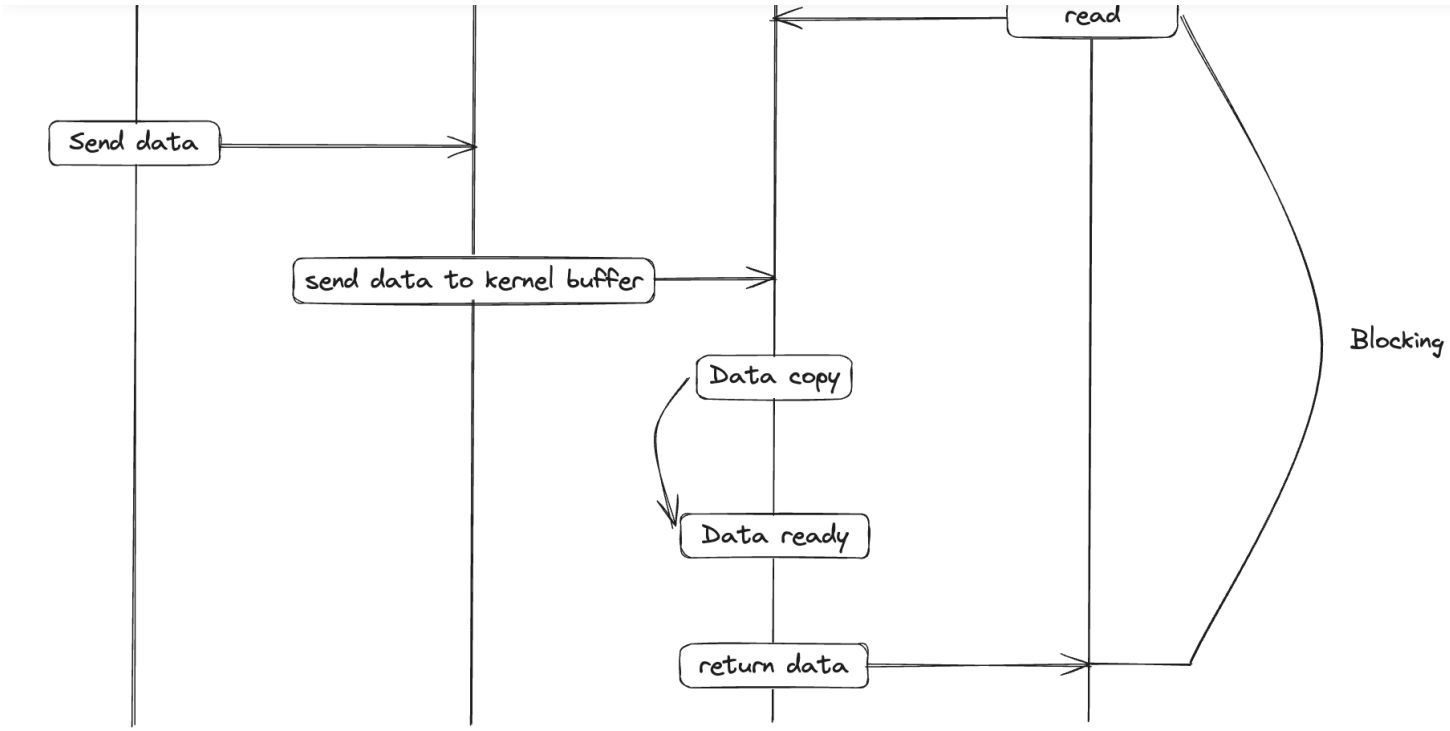



3. 클라이언트가 소켓에게 데이터를 전송합니다.
4. 소켓은 서버의 Kernel Buffer에 데이터를 전송합니다.




5. Kernel Buffer에 데이터를 모두 보냈으면 유저 영역에 데이터를 복사합니다.
6. 유저 영역에 데이터를 복사했다면 read를 수행할 수 있는 상태가 된 것이므로, 서버가 호출한 read의 결과로 복사한 데이터를 반환합니다.

이 동작 과정을 표현하면 다음과 같습니다.

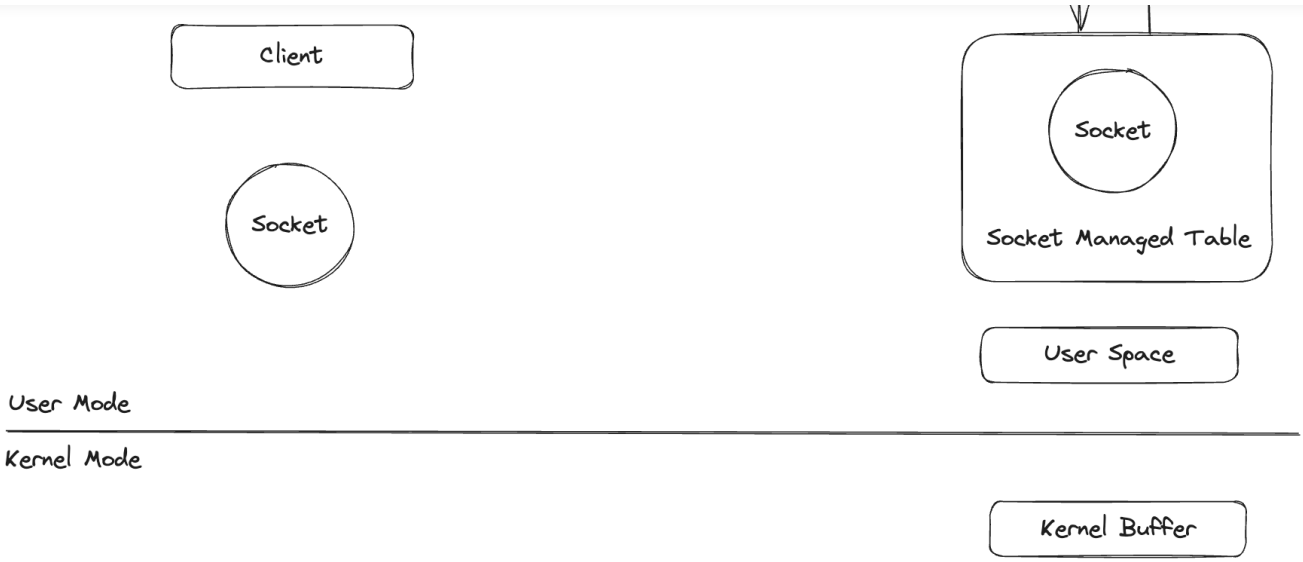



서버(애플리케이션)이 read를 호출했다면, 클라이언트에서 데이터를 보내 이 데이터가 유저 영역에 데이터를 복사해 결과 값을 반환할 때 까지 Blocking이 됩니다.

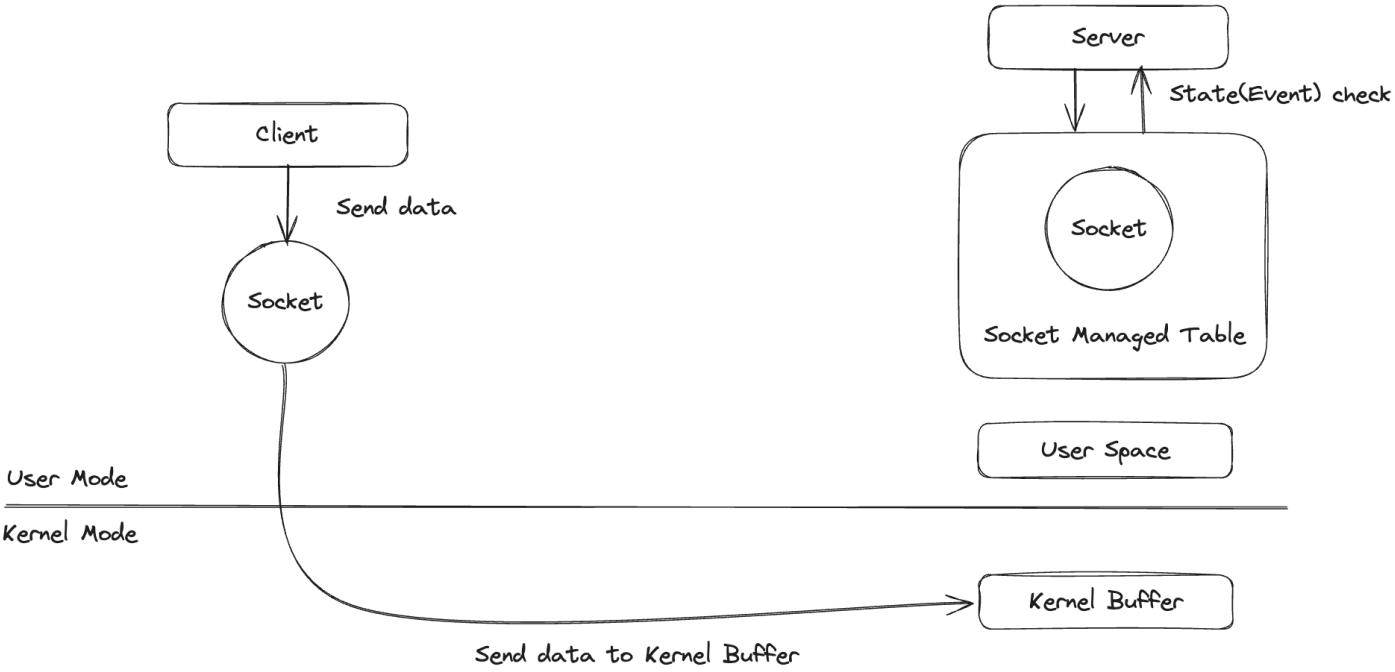
이러한 동작 방식으로 인해 멀티 프로세서 기반, 멀티 스레드 기반 다중 서버가 하나의 프로세스, 스레드에서 하나의 클라이언트만을 처리할 수 있었던 것입니다.

## 멀티 플렉싱 기반

멀티 플렉싱 기반 다중 서버는 다음과 같이 동작합니다.

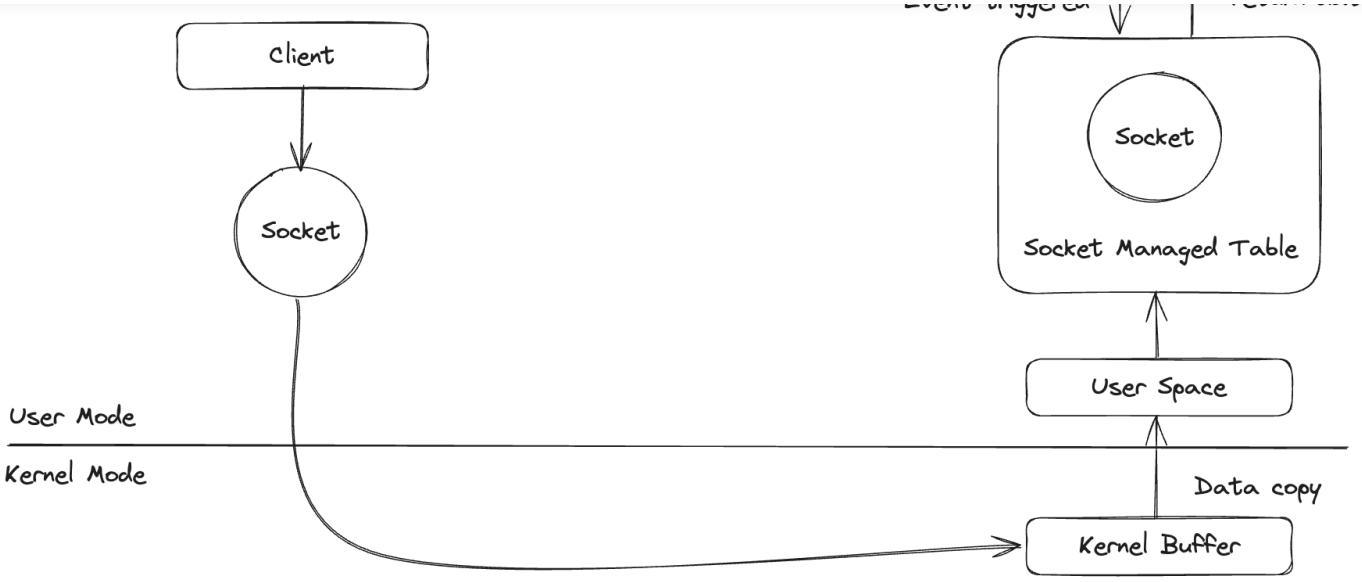



1. 서버는 클라이언트와 연결된 소켓을 별도의 관리 테이블로 관리합니다.
2. 서버는 무한 반복을 통해 관리 테이블에 존재하는 모든 소켓의 상태를 확인합니다.



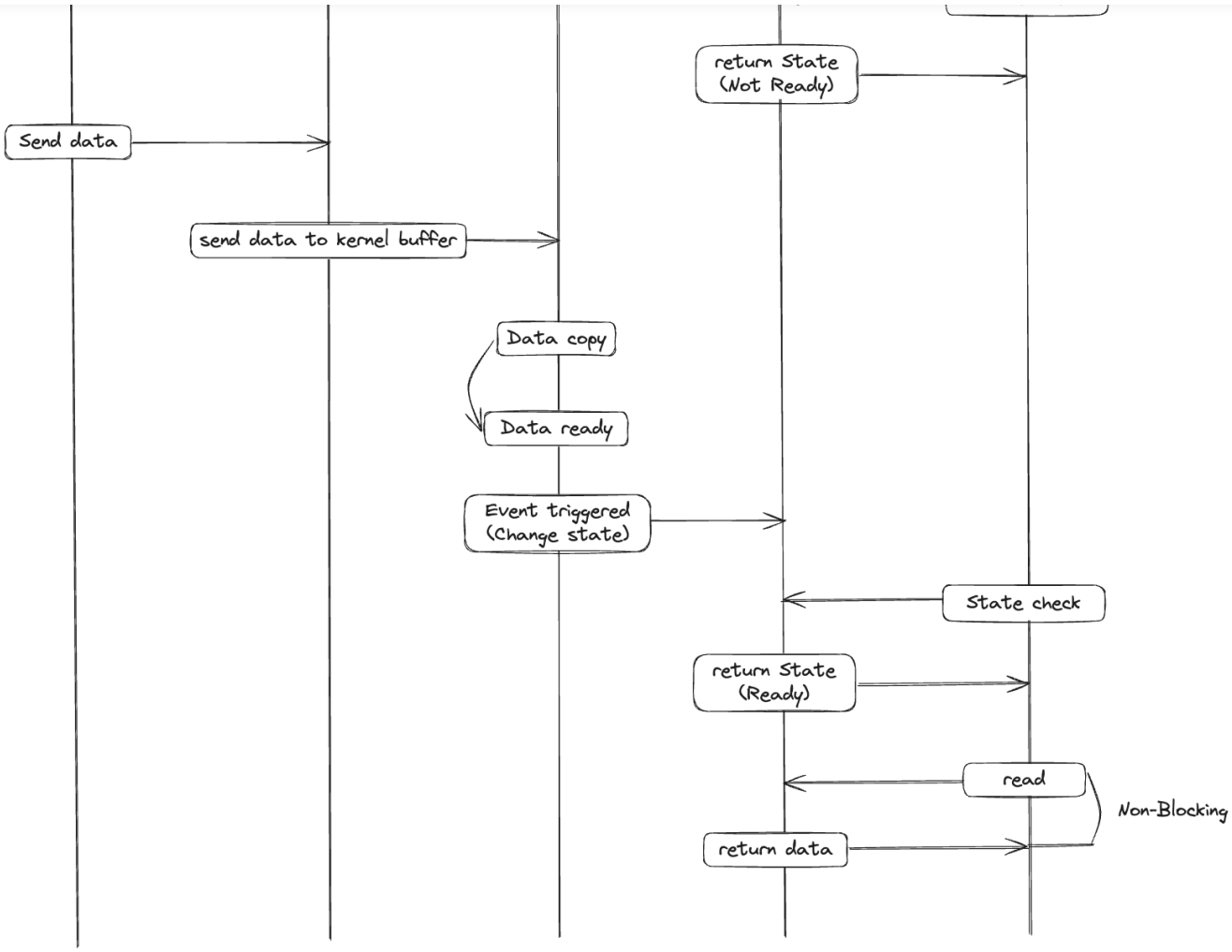

3. 클라이언트가 소켓에게 데이터를 전송합니다.
4. 소켓은 서버의 Kernel Buffer에 데이터를 전송합니다.

appti




5. Kernel Buffer에 데이터를 모두 보냈으면 유저 영역에 데이터를 복사합니다.
6. 관리 테이블에 등록된 모든 소켓의 상태를 확인하던 서버가 소켓의 상태가 변경되었음(read가 가능한 상태)을 감지합니다.
7. 이미 데이터가 준비되었기 때문에, 별도의 Blocking 없이 바로 소켓에서 데이터를 read합니다.

이 동작 과정을 표현하면 다음과 같습니다.




멀티 프로세서, 멀티 스레드, 멀티 플렉싱 모두 Blocking되는 소켓의 read를 호출합니다.

멀티 프로세서와 멀티 스레드는 소켓이 준비되었는지와는 무관하게 read를 호출하기 때문에 클라이언트가 데이터를 보낼 때 까지는 Blocking이 됩니다.

멀티 플렉싱은 소켓에서 데이터를 읽을 수 있는 상태일 때만 read를 호출하기 때문에 Blocking 되지 않습니다.

이러한 차이로 인해 멀티 프로세서와 멀티 스레드 기반은 하나의 프로세스 & 스레드가 하나의 소켓만 처리가 가능했지만, 멀티 플렉싱 기반은 하나의 스레드에서 여러 소켓을 관리할 수 있는 것입니다.

대기하는 시간 또한, 멀티 프로세서와 멀티 스레드 기반은 클라이언트에서 데이터를 보낼 때 까지 무한정 대기해야 하지만 멀티 플렉싱 기반은 바로 read를 수행할 수 있습니다.
관리 테이블에 소켓이 많다면 준비된 소켓을 찾기까지 시간이 걸리겠지만, 이는 다른 방식에 비해서는 매우 짧을 것입니다.

- 다중 접속 서버 구현 방식에는 멀티 프로세서 기반, 멀티 스레드 기반, 멀티 플렉싱 기반의 방식이 있습니다.
- 소켓이 준비될 때 까지 대기하느냐, 준비된 소켓을 찾느냐에 따라 스레드가 Blocking 유무가 결정됩니다.
- 이러한 차이점으로 인해 멀티 플렉싱 기반의 방식은 다른 방식과는 다르게 하나의 스레드에서 여러 소켓을 처리할 수 있습니다.

팔로우

**appti**

안녕하세요

다음 포스트
java.nio

이전 포스트
ApplicationContext Refresh - afterRefresh(), 정리

## 0개의 댓글

댓글을 작성하세요

댓글 작성