Socket Programming

유명성

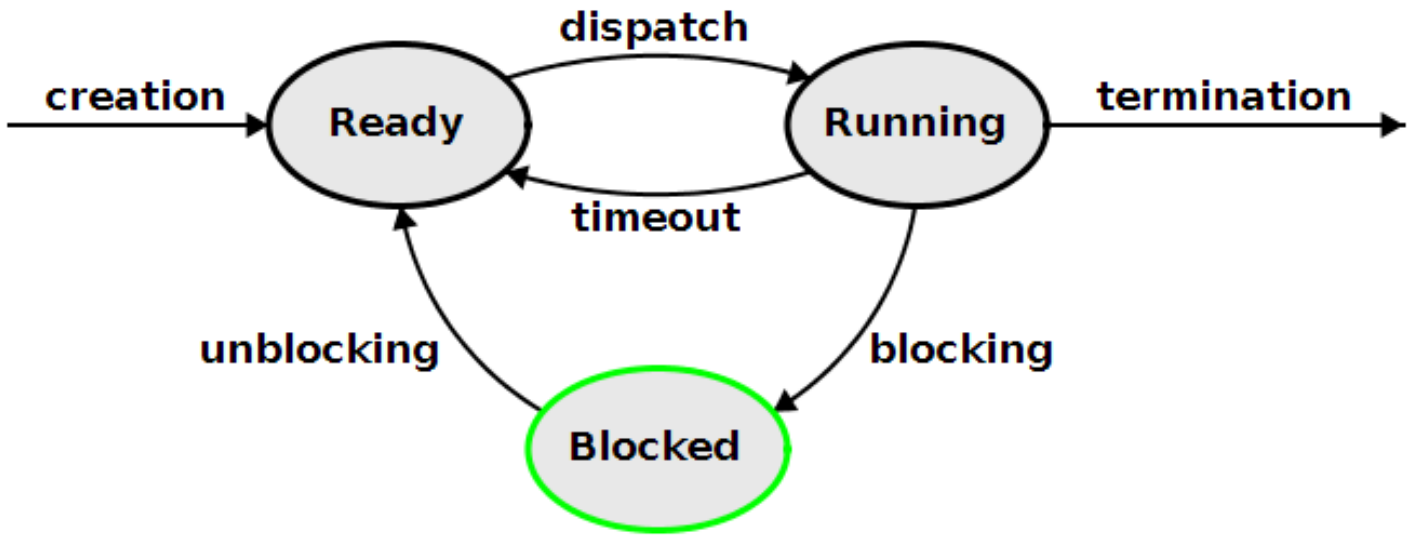
1. Blocking I/O

1. Blocking I/O

1.1 Blocking I/O

Blocking I/O

- ❖ read/write 등의 I/O 관련 시스템콜은 종료조건이 만족되지 않을 경우 호출한 스레드를 Blocking한다.
- ❖ 스레드가 Blocking될 경우 CPU는 곧바로 다른 스레드를 실행하며, Blocking된 스레드는 I/O 작업이 끝날 때까지 CPU에 의해 실행되지 못 하고 대기하게 된다.
- ❖ 프로세스 내 스레드가 1개만 있을 경우 해당 스레드가 Block되면 프로세스는 CPU에 의해 실행되지 않는다.



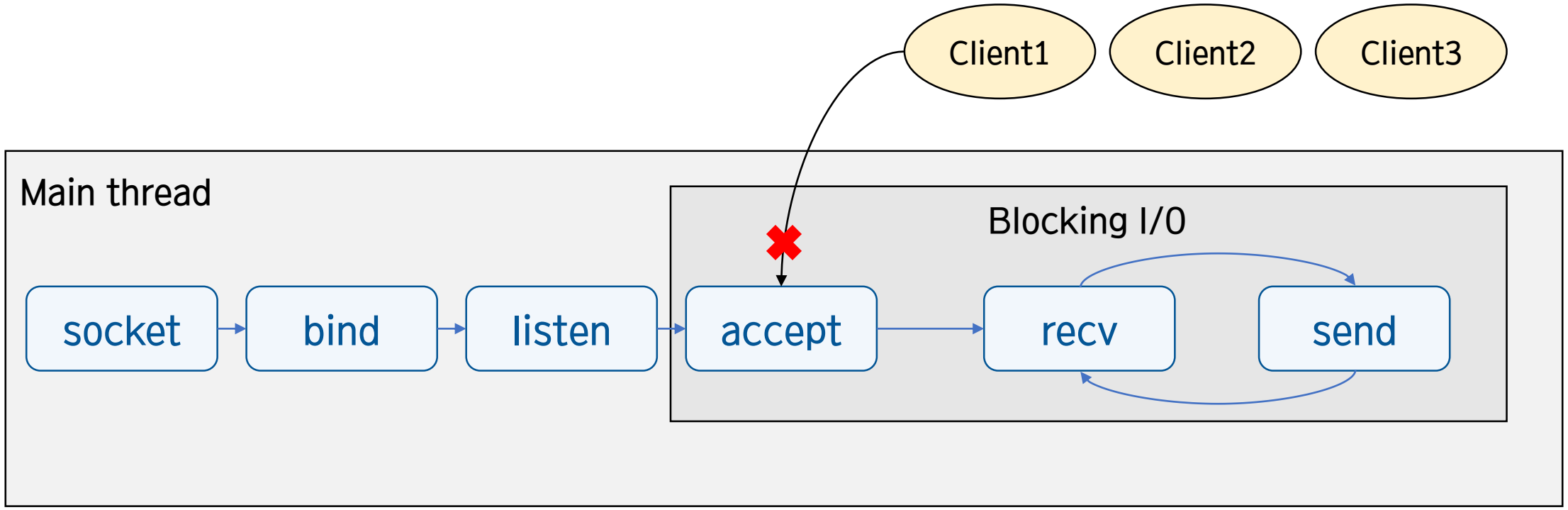
1. Blocking I/O

1.2 Blocking Socket API

accept	❖ Backlog Queue(연결 대기 Queue)에 연결요청이 없을 경우 Block
connect	❖ 서버-클라이언트간 connection이 완전하게 이루어질 때까지 Block
recv	❖ 커널의 read-buffer가 비어 있을 경우 Block ❖ recv는 read-buffer에 데이터가 1Byte라도 있을 경우 Block되지 않고 Return
send	❖ 커널의 write-buffer가 꽉 차 있을 경우 Block

1. Blocking I/O

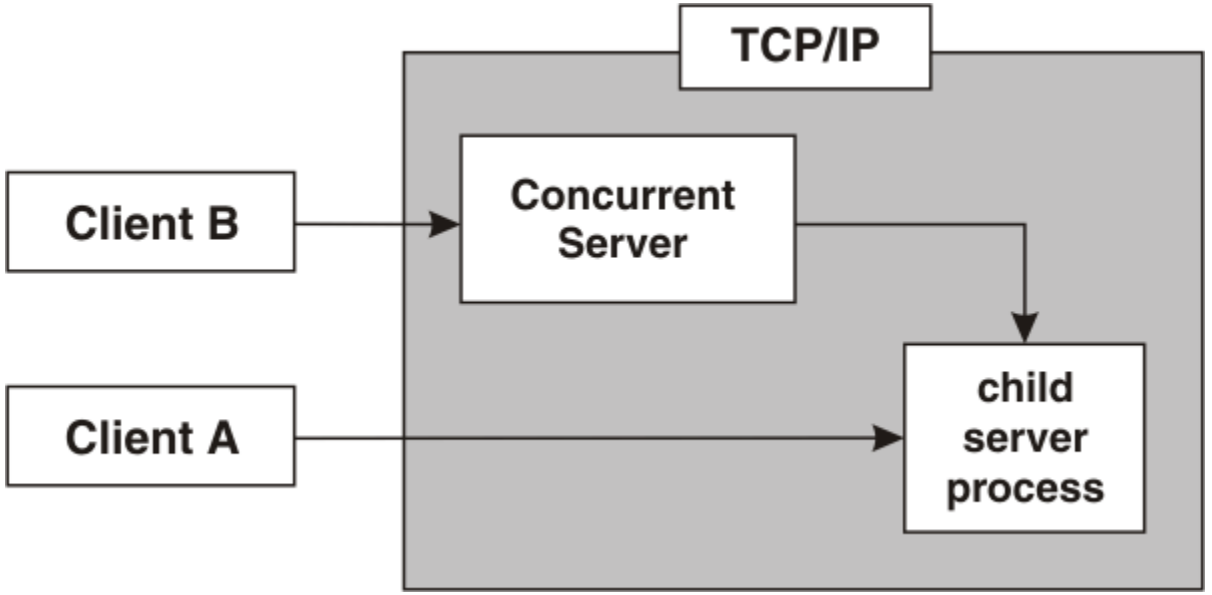
1.3 Single Thread Server with Blocking I/O



Blocking I/O에서 스레드가 1개일 경우 다른 요청을 처리할 수 없다.

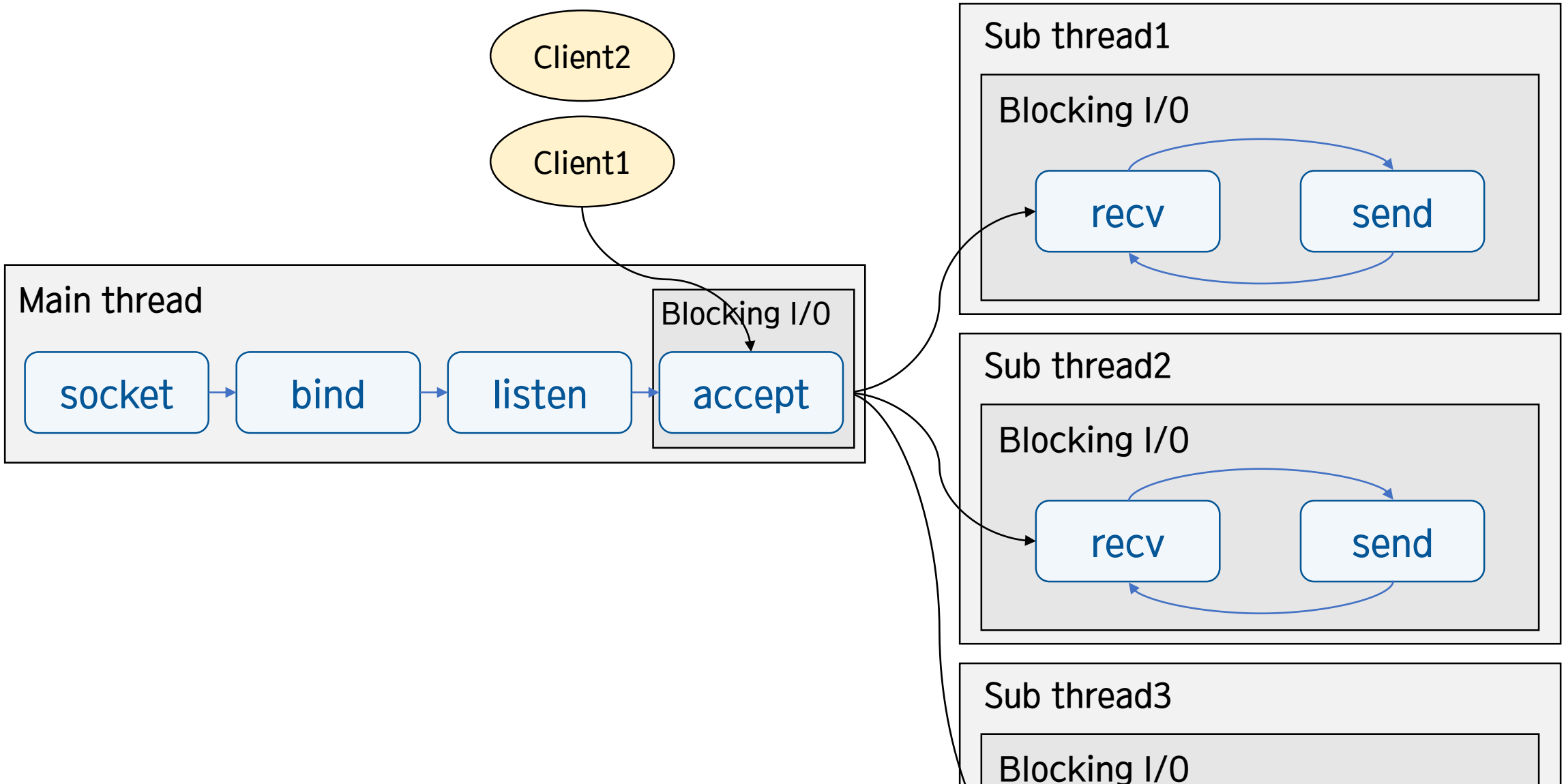
병행서버

- ❖ 다수의 스레드 혹은 프로세스를 이용해 많은 클라이언트의 요청을 병렬로 처리
- ❖ 서버와 클라이언트간 통신이 길어져도 다른 클라이언트 요청을 즉각 처리할 수 있다.
- ❖ 멀티 스레드 혹은 멀티 프로세스를 이용해 구현하기 때문에 시스템 자원 소모가 크다.
- ❖ 대표적인 예로 Apache 서버가 있으며 pre-fork 방식을 사용한다.



1. Blocking I/O

1.4 Concurrent Server



1. Blocking I/O

1.5 Assignment#6

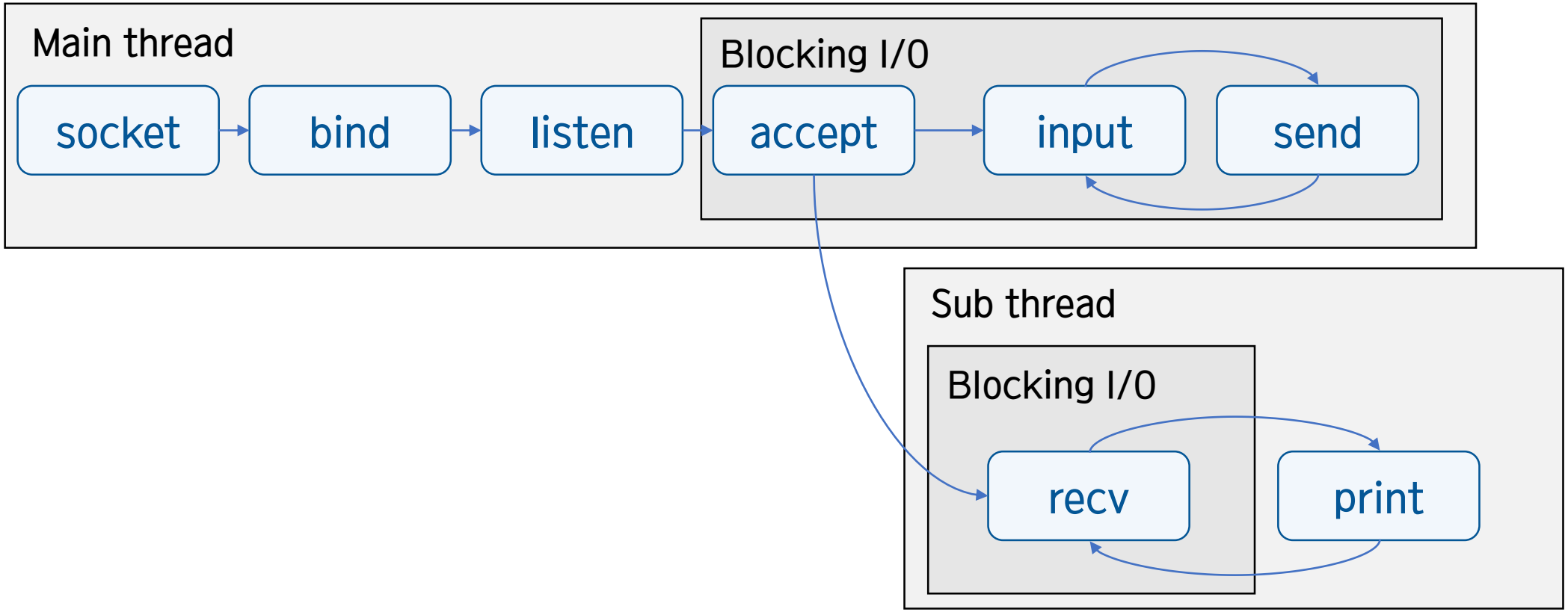
서버에서 단순히 클라이언트가 보낸 문자열을 뒤집어 전송하는게 아니라,
지속적으로 서버와 클라이언트가 메시지를 주고 받으려면?

1. Blocking I/O

1.6 Chatting Server API Sequence

Server

❖ Server는 Accept 호출 후 recv와 print를 반복 수행할 스레드를 생성 후 input/send를 반복 수행
❖ 1개의 클라이언트만 처리하기 때문에 accept까지 진행하는 스레드를 분리할 필요가 없다.

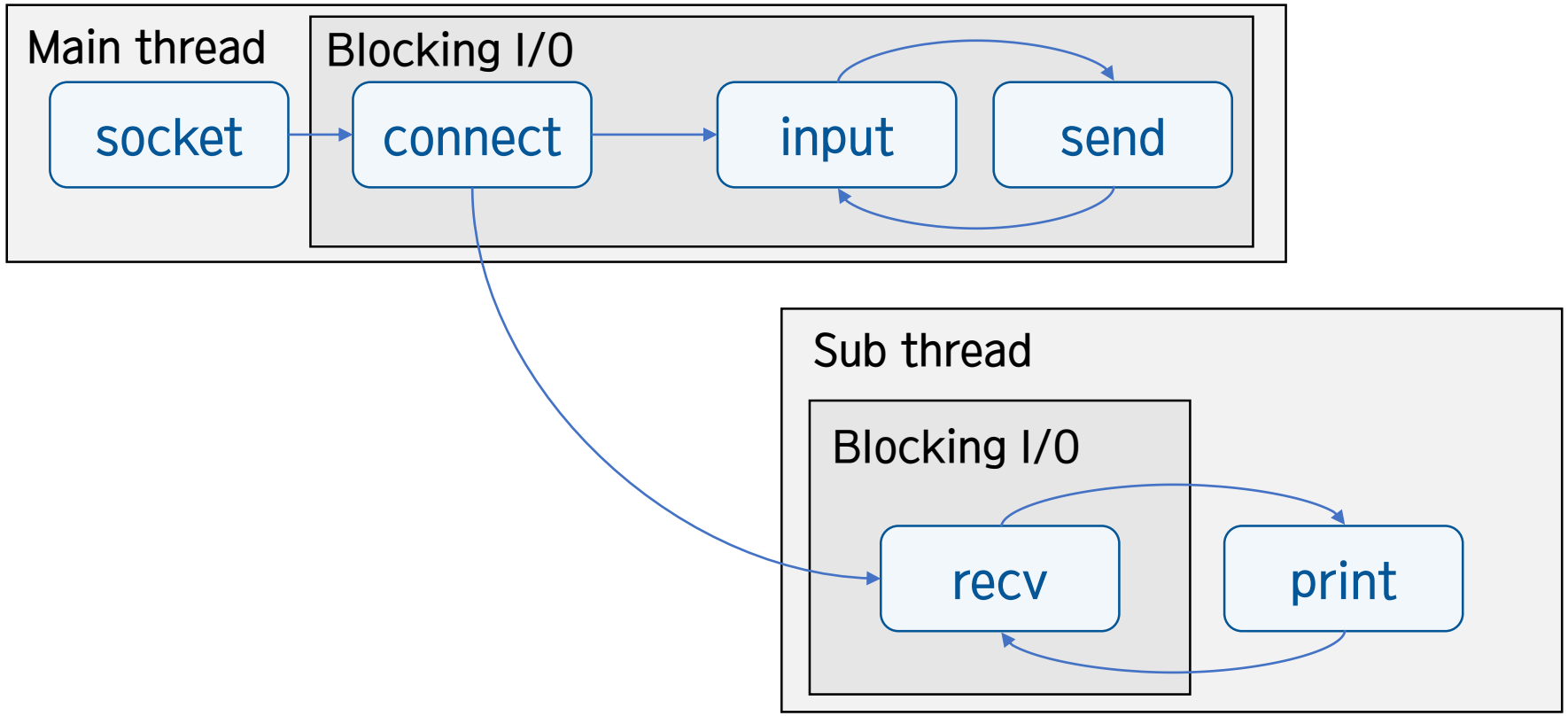


1. Blocking I/O

1.7 Chatting Client API Sequence

Client

❖ Client는 connect 수행 후 recv/print를 반복 수행할 스레드 생성 후 input/send를 반복 수행



2. Non-Blocking I/O

Non-Blocking I/O

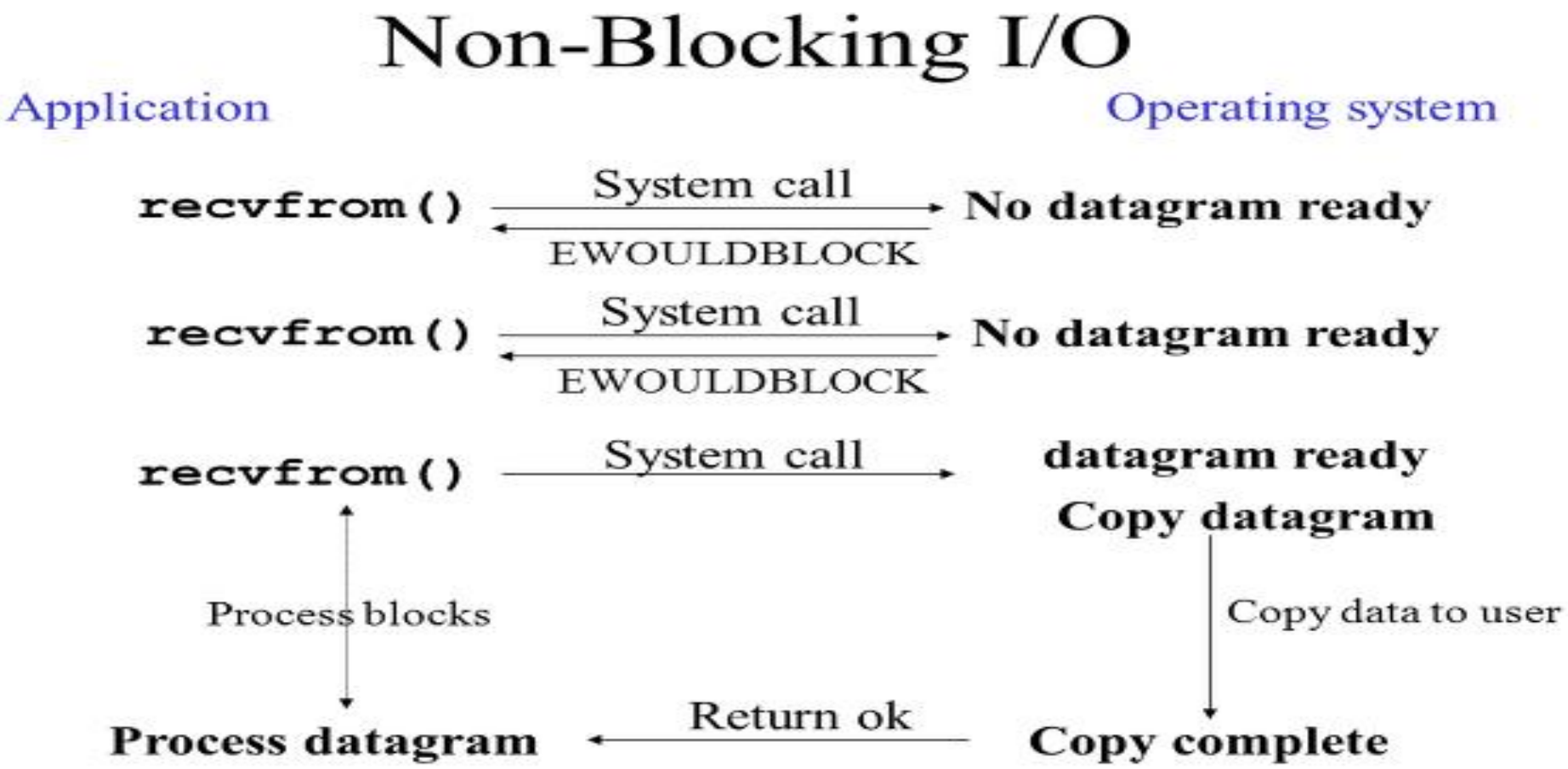
- API 호출 후 조건에 관계없이 바로 return하는 모드.
- 조건이 만족되었다면 결과를 리턴하며, 만족되지 않은 경우 EWOULDBLOCK(EAGAIN) 등 에러를 발생시킨다.
- 일반적으로 어떤 시스템 콜이 완료되었는지 보기 위해 루프를 돌며 확인하는 Polling을 사용해야 한다.

- 장점
  - I/O작업 시간동안 스레드를 Block하지 않고 다른 일을 처리해 CPU를 효율적으로 사용할 수 있다.
  - 멀티 스레드에서 Blocking을 사용 하는 것에 비해 컨텍스트 스위칭, 스레드 생성 등의 오버헤드가 없다.

- 단점
  - 프로그램이 복잡 해진다.
  - 멀티 스레드 Blocking 모델에 비해 자원(Memory, CPU)를 효율적으로 사용하지만, 응답속도가 떨어진다.

2. Non-Blocking I/O

2.1 Non-Blocking I/O



2. Non-Blocking I/O

2.2 Blocking Socket API

accept	❖ Backlog Queue(연결 대기 Queue)에 연결요청이 없을 경우 Block
connect	❖ 서버-클라이언트간 connection이 완전하게 이루어질 때까지 Block
recv	❖ 커널의 read-buffer가 비어 있을 경우 Block ❖ recv는 read-buffer에 데이터가 1Byte라도 있을 경우 Block되지 않고 Return
send	❖ 커널의 write-buffer가 꽉 차 있을 경우 Block

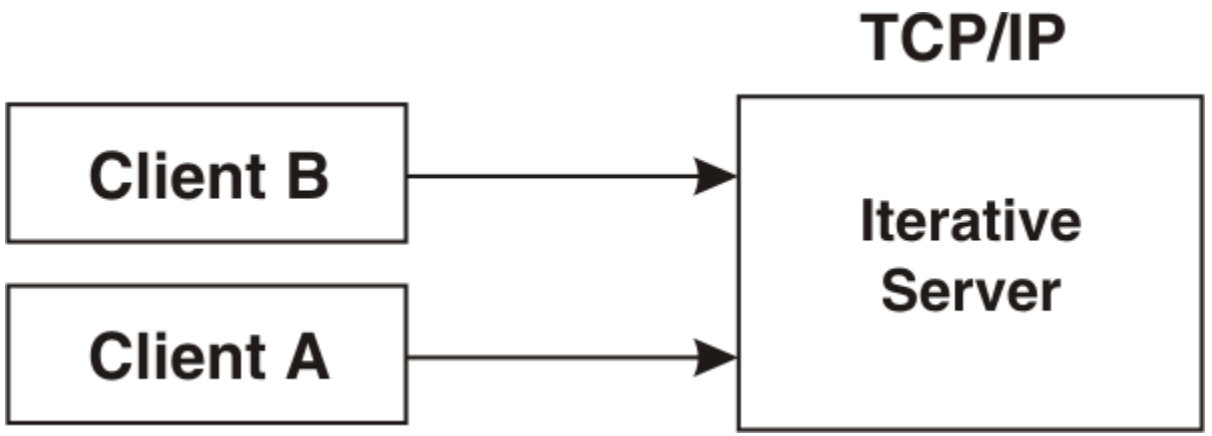
Non-Blocking Socket을 사용할 경우 Block되지 않고 EWOULDBLCOK/EAGAIN 에러 발생

2. Non-Blocking I/O

2.3 Iterative Server

반복서버

- ❖ 접속한 클라이언트를 하나씩 차례대로 처리한다.
- ❖ 하나의 스레드로 모든 요청을 처리하기 때문에 시스템 자원 소모가 적다.
- ❖ 서버와 클라이언트의 통신이 길어지면 그 시간만큼 다른 클라이언트의 요청을 처리할 수 없다.
- ❖ 주로 이벤트 기반 Non-blocking 비동기처리를 통해 구현한다. 대표적으로 node.js 서버가 있다.



2. Non-Blocking I/O

2.4 Non-Blocking Socket

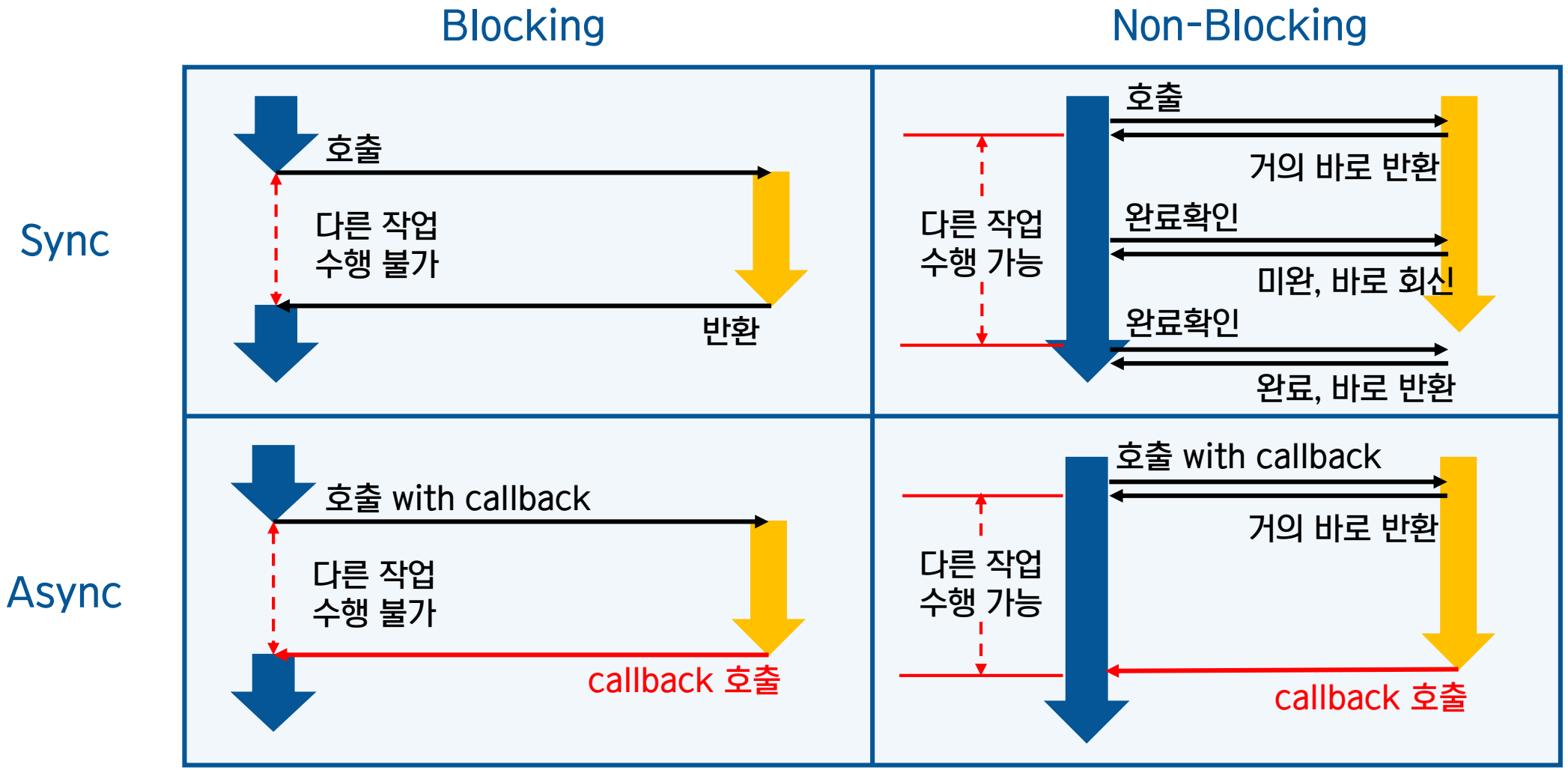
```
conn.setblocking(0) # socket을 blocking에서 non-blocking으로 설정

while True:
    try:
        data = conn.recv(BSIZE)
        # 데이터 수신 후 처리될 로직
    except socket.error as e:
        if e.args[0] == errno.EWOULDBLOCK: # errno는 에러 번호를 정의한 클래스이며,
                                           # EWWOULDBLOCK는 데이터가 없다는 에러이다.

            # 데이터가 도착하지 않을 경우 처리될 로직
        else:
            # 다른 에러 발생시 처리될 로직
```

2. Non-Blocking I/O

2.5 I/O Model



2. Non-Blocking I/O

2.5 I/O Model

	Blocking	Non-Blocking
Sync	Read/Send	Read/Send O_NONBLOCK
Async	I/O Multiplexing 구현체에 따라 상이	AIO

3. I/O Multipexing

Select Module

- Python에서 I/O 멀티플랙싱을 제공해주는 모듈로 Unix의 select, Linux의 e-poll 등 I/O 멀티플랙싱 관련된 다양한 API를 제공한다.
- 하나의 스레드로 다수의 FD(File Descriptor)를 검사할 수 있게 해준다.
- Windows에서는 Socket에 대해서만 사용할 수 있다.(타 OS는 File에도 사용가능)
- import select를 통해 사용할 수 있다.

select()

❖ Python에서 I/O 멀티플랙싱을 제공해주는 모듈로 인자로 전달된 FD 목록들을 살피며, FD에 특정 이벤트가 발생하면 이벤트가 발생한 FD 목록을 리턴한다.

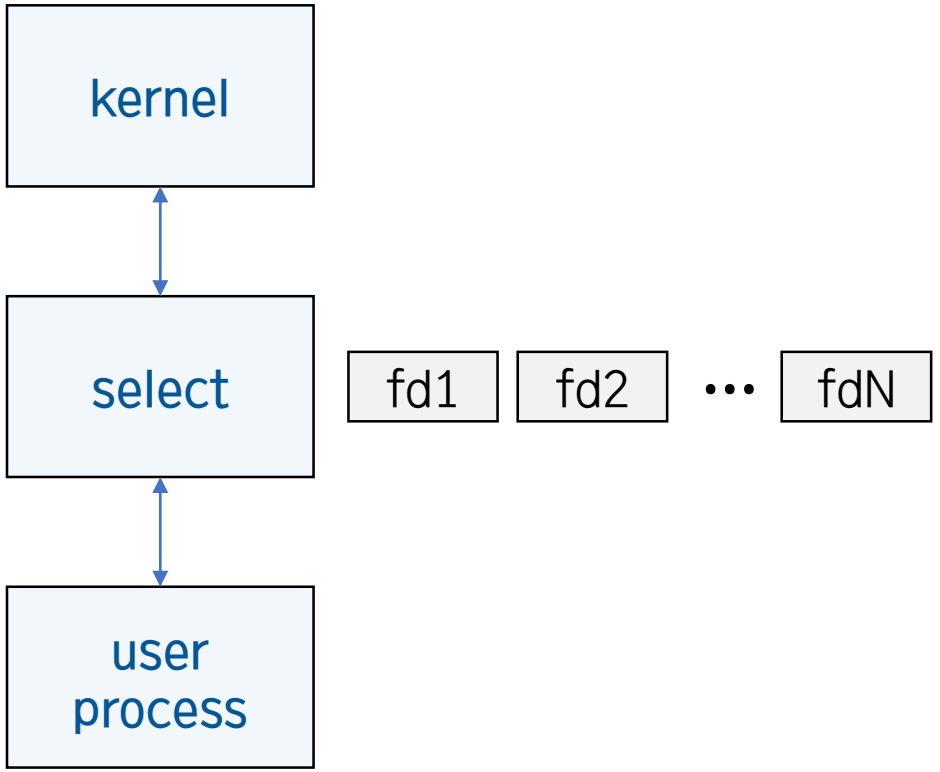
❖ 읽기 가능 FD 목록, 쓰기 가능 FD 목록, 에러 발생 FD 목록을 각각 리턴
- 읽기 가능 : 데이터가 도착해 버퍼에 복사할 준비가 됨
- 쓰기 가능 : 커널 버퍼에 전송할 데이터를 복사할 준비가 됨

```
select.select( rlist , wlist , xlist [ , timeout ] )
```

- rlist : 읽기 준비가 될 때까지 기다릴 FD 목록
- wlist : 쓰기 준비가 될 때까지 기다릴 FD 목록
- xlist : 예외상황이 발생할 때까지 기다릴 FD 목록
- timeout : select 호출 후 kernel에 대기할 최대시간(0 : 대기 안함)

3. I/O Multiplexing

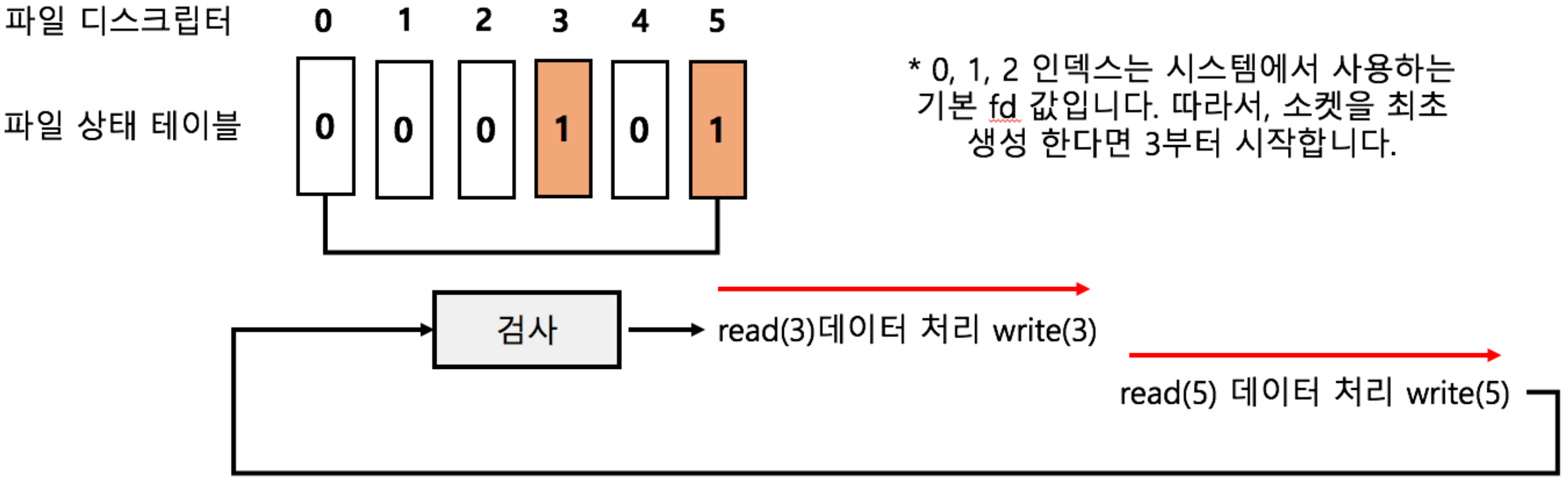
3.2 select



1. select 함수는 목록의 fd들이 각각 read나 write가 가능한지 커널에 묻는다.
2. 커널은 해당 fd의 read/write 여부를 select에게 알려준다.
3. select 함수는 read/write가 가능한 fd 목록을 리턴한다.

3. I/O Multiplexing

3.2 select



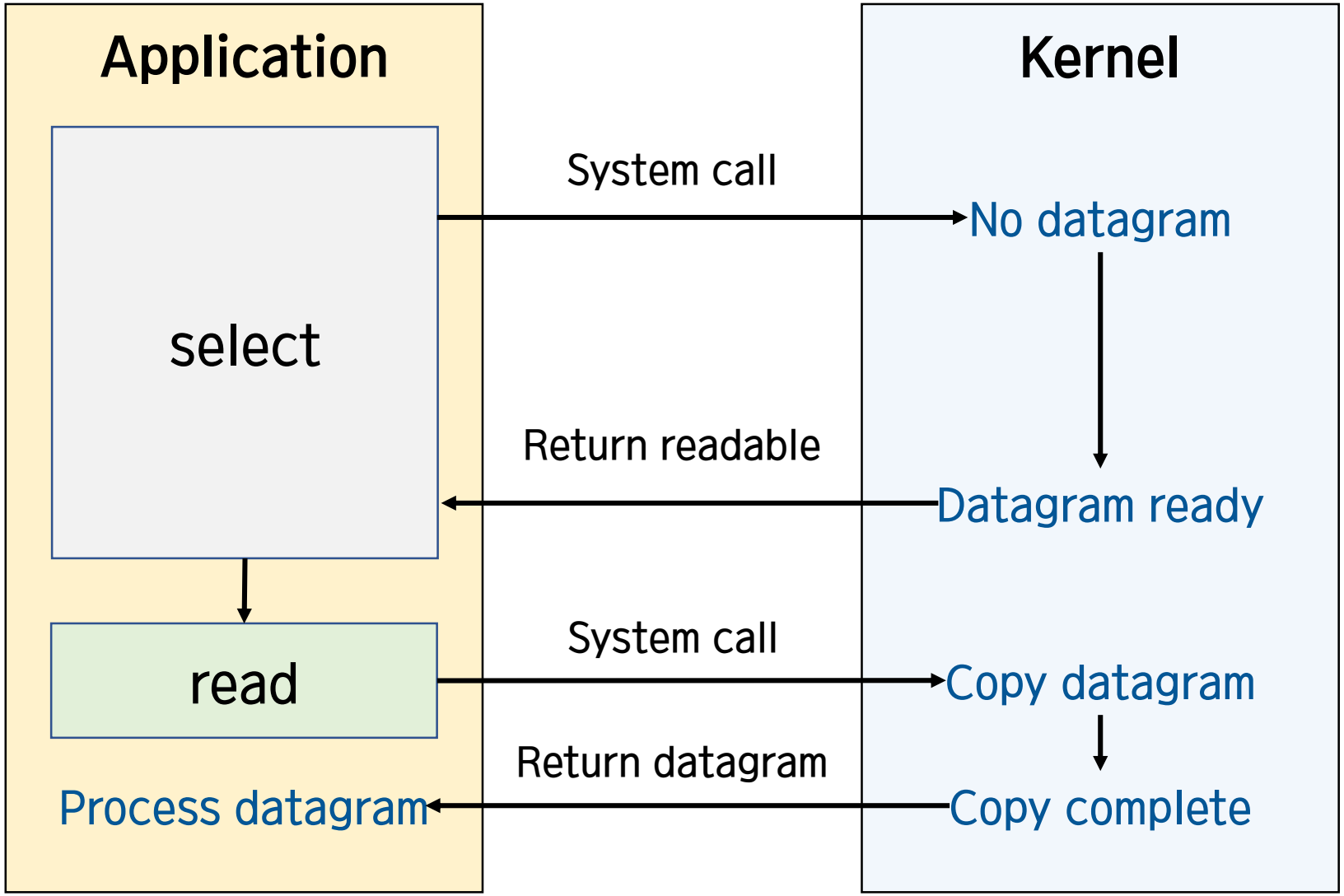
* 0, 1, 2 인덱스는 시스템에서 사용하는 기본 fd 값입니다. 따라서, 소켓을 최초 생성 한다면 3부터 시작합니다.

3. I/O Multiplexing

3.2 select

select가 감시하는 소켓 중 하나 이상이 readable할 때까지 프로세스가 Block된다.

kernel read-buffer의 데이터를 복사해올 때 까지 프로세스가 Block 된다.



3. I/O Multiplexing

3.3 select server

```
7   server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8   server.bind((host, port))
9   server.listen(5)
10  inputs = [server]
11
12  while True:
13      insds, outsds, errsds = select.select(inputs, inputs, [])   # select를 통해 서버 소켓의 R/W를 감시
14      for sds in insds:   # 소켓 목록 중 읽기가 가능한 소켓이 있다면
15          if sds is server:   # 서버 소켓이 읽기가 가능한 경우 -> 새로운 클라이언트 접속 요청
16              clientsock, clienaddr = sds.accept()
17              inputs.append(clientsock)   # 감시할 소켓 목록에 클라이언트 소켓을 추가
18          else:   # 클라이언트 소켓이 읽기가 가능한 경우 -> 새로운 데이터가 온 경우
19              buf = sds.recv(1024)
20              if len(buf) == 0:
21                  inputs.remove(sds)
22              else:
23                  print("Receive data({A}) : {D}\n".format(A=sds.getpeername(), D=buf.decode()))
24                  sds.sendall(buf[::-1])  # 받은 문자열을 뒤집어서 전송
```

Assignment #7

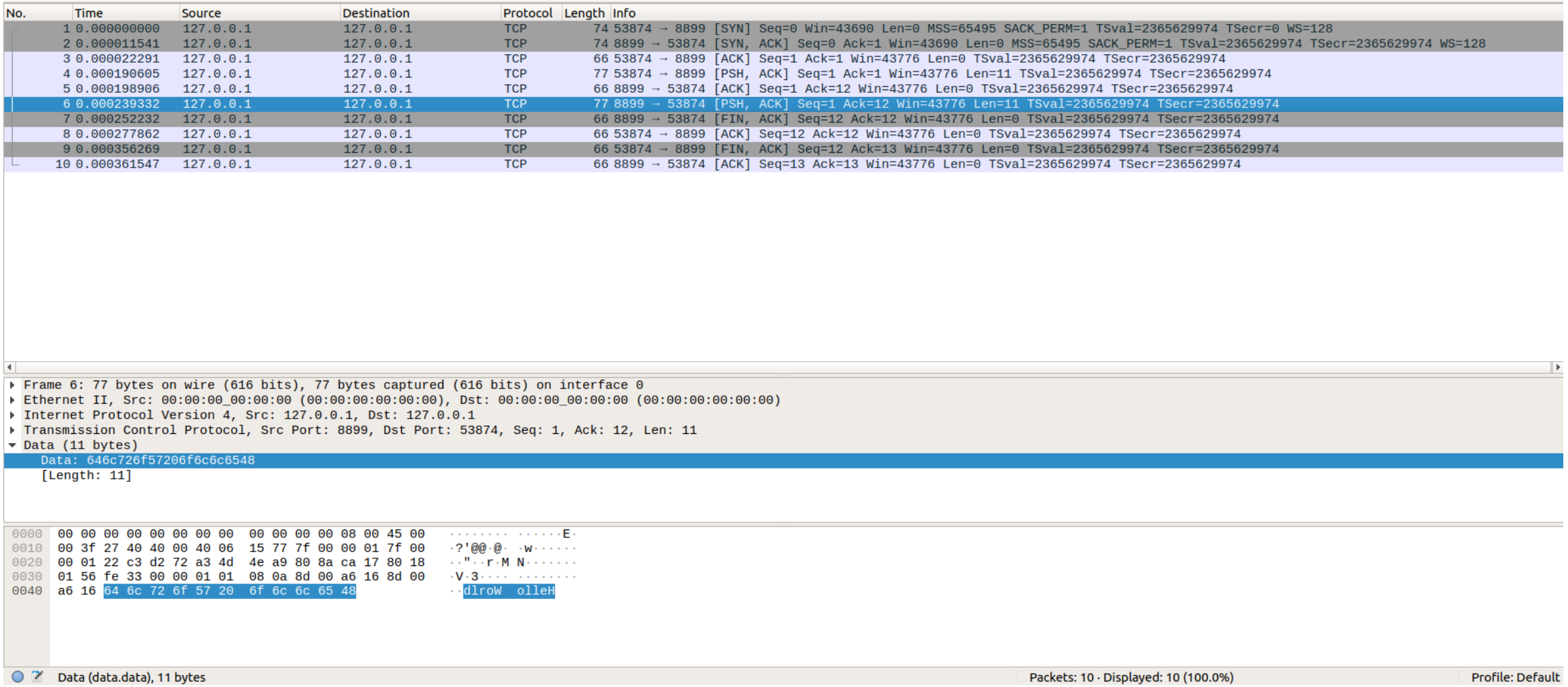
- wireshark 프로그램에 대해서 조사한 뒤 보고서 작성
  - wireshark 프로그램이란?
  - wireshark는 어떠한 라이브러리를 사용하는가?(Linux, Windows)
  - wireshark로 Assignment#2(문자열 거꾸로 전송)가 실행되면서 서버-클라이언트간 주고받은 TCP 패킷을 캡쳐해서 사진 첨부(문자열은 팀 이름을 전달)
  - 보고서는 3장 내로 작성(1~2장 : wireshark 조사, 3장 : 패킷 캡쳐 사진 및 설명)
- 팀 대표가 barcel@naver.com으로 제출 (4.30일까지)
  - Title : [컴퓨터네트워크][학번][이름][과제_N]
  - Content : github repo url

    팀명 : 길동이네

    팀원 : 홍길동(학번), 고길동(학번)

3. I/O Multiplexing

3.4 Assignment 6



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	53874 → 8899 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2365629974 TSecr=0 WS=128
2	0.000011541	127.0.0.1	127.0.0.1	TCP	74	8899 → 53874 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2365629974 TSecr=2365629974 WS=128
3	0.000022291	127.0.0.1	127.0.0.1	TCP	66	53874 → 8899 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=2365629974 TSecr=2365629974
4	0.000190605	127.0.0.1	127.0.0.1	TCP	77	53874 → 8899 [PSH, ACK] Seq=1 Ack=1 Win=43776 Len=11 TSval=2365629974 TSecr=2365629974
5	0.000198906	127.0.0.1	127.0.0.1	TCP	66	8899 → 53874 [ACK] Seq=1 Ack=12 Win=43776 Len=0 TSval=2365629974 TSecr=2365629974
6	0.000239332	127.0.0.1	127.0.0.1	TCP	77	8899 → 53874 [PSH, ACK] Seq=1 Ack=12 Win=43776 Len=11 TSval=2365629974 TSecr=2365629974
7	0.000252232	127.0.0.1	127.0.0.1	TCP	66	8899 → 53874 [FIN, ACK] Seq=12 Ack=12 Win=43776 Len=0 TSval=2365629974 TSecr=2365629974
8	0.000277862	127.0.0.1	127.0.0.1	TCP	66	53874 → 8899 [ACK] Seq=12 Ack=12 Win=43776 Len=0 TSval=2365629974 TSecr=2365629974
9	0.000356269	127.0.0.1	127.0.0.1	TCP	66	53874 → 8899 [FIN, ACK] Seq=12 Ack=13 Win=43776 Len=0 TSval=2365629974 TSecr=2365629974
10	0.000361547	127.0.0.1	127.0.0.1	TCP	66	8899 → 53874 [ACK] Seq=13 Ack=13 Win=43776 Len=0 TSval=2365629974 TSecr=2365629974

```
▸ Frame 6: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 8899, Dst Port: 53874, Seq: 1, Ack: 12, Len: 11
▾ Data (11 bytes)
    Data: 646c726f57206f6c6c6548
    [Length: 11]
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
0010  00 3f 27 40 40 00 40 06  15 77 7f 00 00 01 7f 00   ·?'@@·@· ·w·····
0020  00 01 22 c3 d2 72 a3 4d  4e a9 80 8a ca 17 80 18   ··"··r·M N·······
0030  01 56 fe 33 00 00 01 01  08 0a 8d 00 a6 16 8d 00   ·V·3···· ········
0040  a6 16 64 6c 72 6f 57 20  6f 6c 6c 65 48            ··dlroW  olleH
```

Data (data.data), 11 bytes | Packets: 10 · Displayed: 10 (100.0%) | Profile: Default

F.Y.I

Midterm

1. Socket이란 무엇인가?
2. TCP/UDP Socket API Call Sequence
3. TCP/UDP 소켓의 차이는?
4. 다수의 클라이언트 요청을 처리하기 위해 어떻게 하는가?
    1. 병행서버
    2. 반복서버
5. Blocking/Non-Blocking의 차이는?