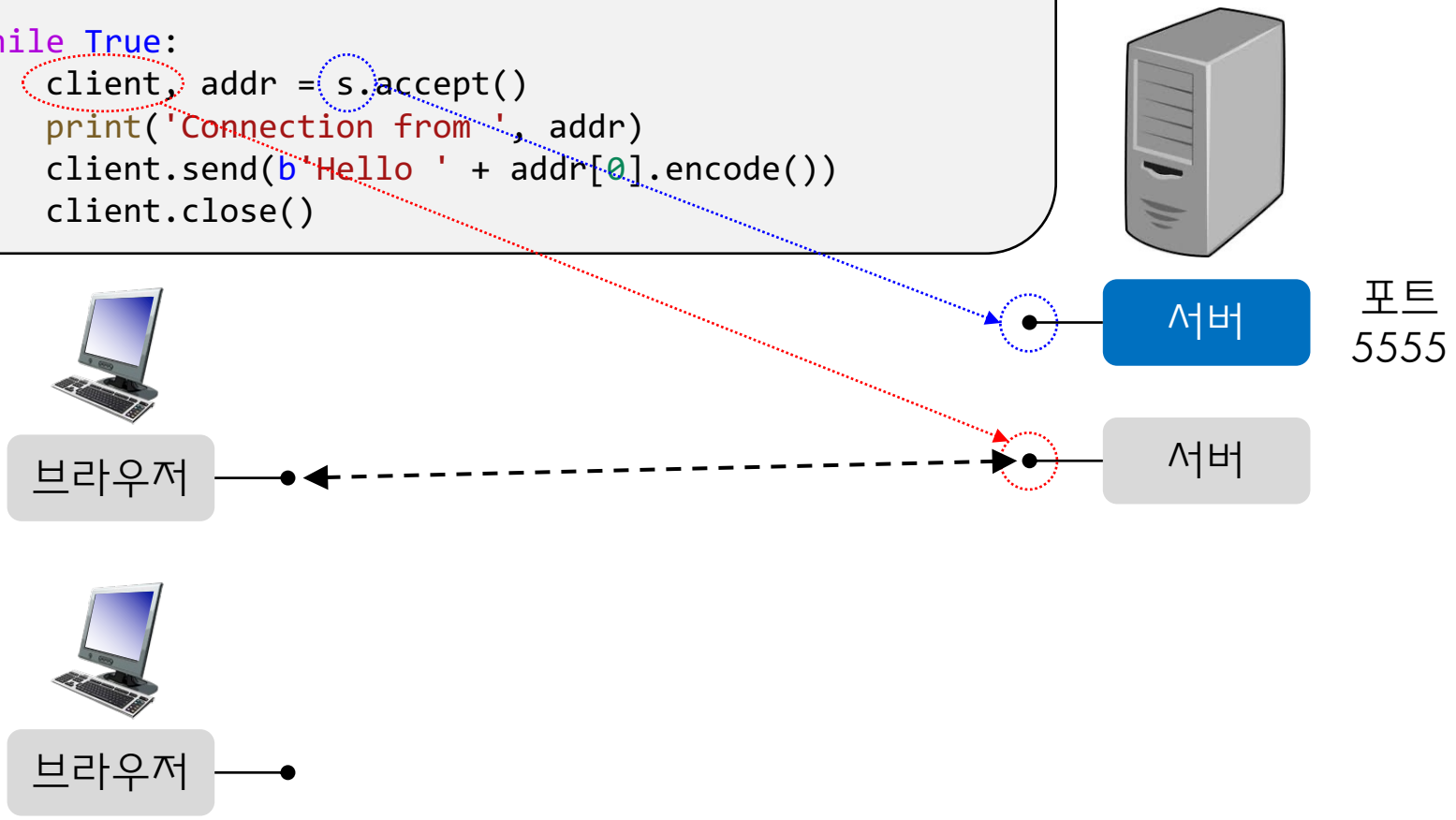
네트워크 프로그래밍
- 동시성 소켓 프로그래밍 1 -

순천향대학교 사물인터넷학과

소켓과 동시성 sockets and concurrency

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 9000))
s.listen(2)

while True:
    client, addr = s.accept()
    print('Connection from ', addr)
    client.send(b'Hello ' + addr[0].encode())
    client.close()
```



소켓과 동시성 sockets and concurrency

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 9000))
s.listen(2)

while True:
    client, addr = s.accept()
    print('Connection from ', addr)
    client.send(b'Hello ' + addr[0].encode())
    client.close()
```

브라우저

브라우저

서버

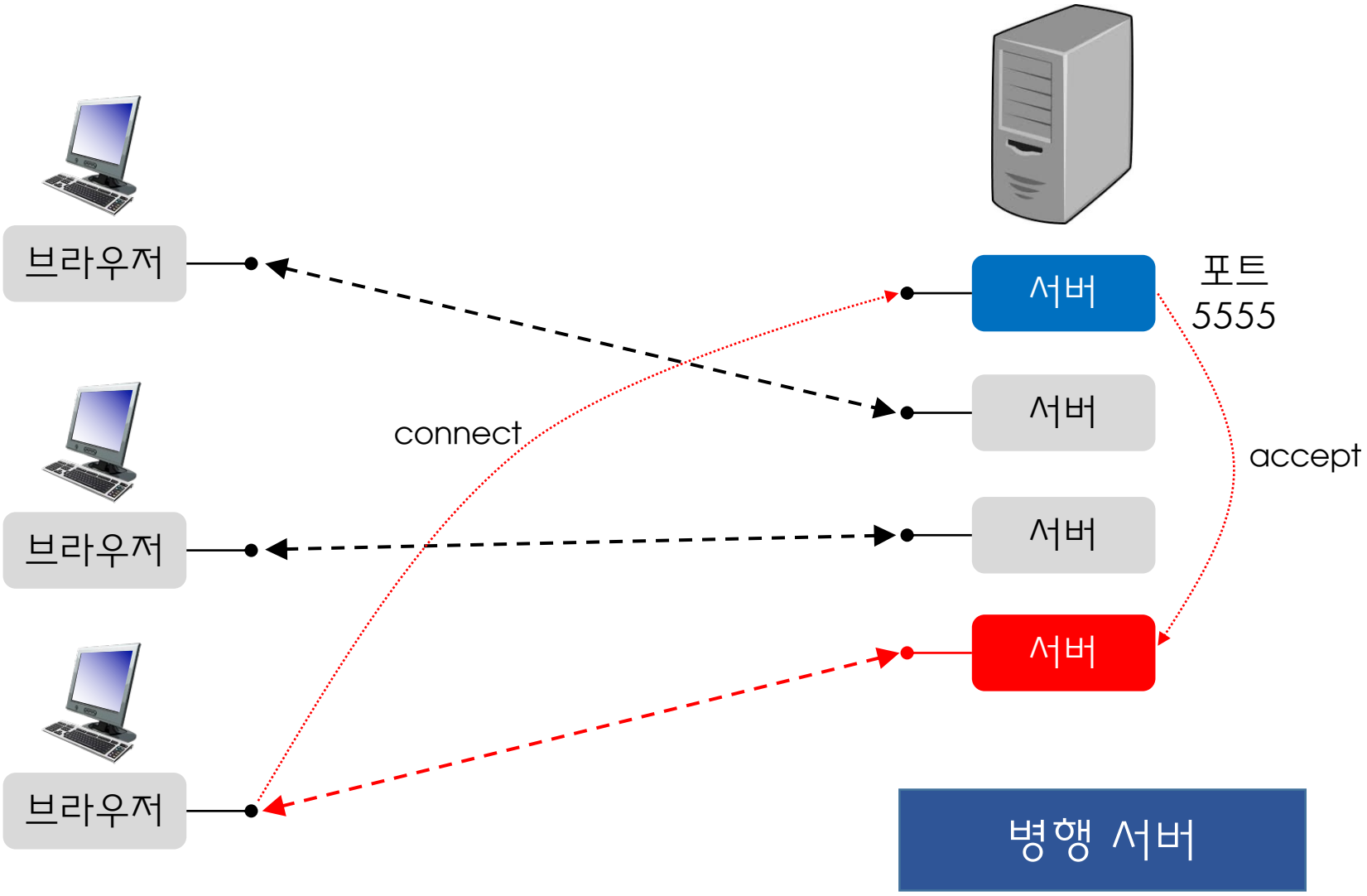
서버

서버

포트
5555

반복 서버

소켓과 동시성 sockets and concurrency



서버 종류

■ 반복 서버 iterative server

- 클라이언트의 요청을 하나씩 서비스하는 서버
- 클라이언트 A가 서비스 받고 있는 동안에 다른 클라이언트들은 대기
- A의 서비스 시간이 길어지면, 다른 클라이언트들의 대기시간이 길어짐

■ 병행 서버 concurrent server

- 클라이언트들을 동시에 서비스하는 서버
- 스레드 방식
  - ✓ 클라이언트마다 별도의 스레드 사용
    - 멀티스레드
- 프로세스 방식
  - ✓ 클라이언트마다 별도의 프로세스 사용
    - 멀티프로세스
- 이벤트 구동 방식
  - ✓ 이벤트가 발생하면 처리하는 방식
  - ✓ `select, selectors, asyncio`

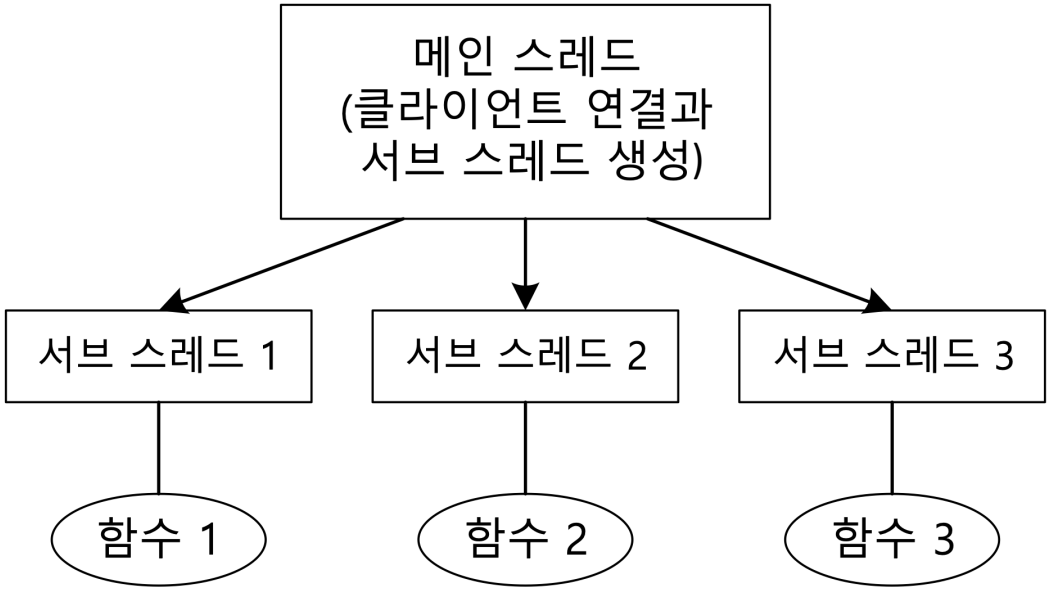
여러 개의 클라이언트를 동시에 서비스하기 위해서는

1. 서버는 항상 새로운 연결을 받아들일 준비를 하고 있어야 함 (`accept()`)
2. 각 클라이언트는 서버와 독립적으로 통신할 수 있어야 함

TCP 멀티스레드 서버

■ 스레드thread

- 운영체제에 의해 시간이 배분되고 관리되는 프로그램 실행 단위
- 코드, 데이터(전역변수 등), 파일 등을 공유
  - ✓ 스택(지역변수 등)은 공유하지 않음
- 스레드를 생성하고 스레드에게 함수의 실행을 맡기면 하나의 프로그램에서 여러 개의 스레드를 실행할 수 있음
  - ✓ 각 스레드는 동일한 작업을 수행할 수도 있고, 서로 다른 작업을 수행할 수도 있음
- (멀티스레드 서버) 메인 스레드는 클라이언트를 연결하고 해당 클라이언트와의 데이터 처리(함수)는 서브 스레드를 생성하여 맡김



TCP 멀티스레드 서버

■ 멀티스레드 구현 방법

- **threading.Thread 클래스 사용**
  - ✓ 메인 스레드에서 실행할 함수를 정의하고 threading.Thread 클래스를 사용하여 스레드 객체를 생성한 다음, start() 메소드를 사용하여 서브 스레드를 실행
  - ✓ threading.Thread로 스레드 객체를 생성할 때, 실행할 함수 이름과 함수의 인수를 지정
- **threading.Thread 파생클래스**
  - ✓ threading.Thread의 파생 클래스를 정의하고 인수와 함께 파생 클래스 객체를 생성
  - ✓ 스레드에서 처리할 내용을 run() 메소드에 정의
  - ✓ 파생 클래스 객체의 start() 메소드를 사용하여 스레드를 시작
  - ✓ 객체의 daemon 속성을 True로 지정하면 메인 스레드가 종료될 때 서브 스레드도 종료. 속성이 False이면 메일 스레드가 종료되어도 서브 스레드는 후순위로 진행

스레드 실행 방법

(1) 함수를 정의하고 스레드로 실행

```
def ftn(arguments): #메인 스레드에서 실행할 함수

t = threading.Thread(target=ftn, args=(arguments))

t.start() #스레드 시작
```

스레드가 실행할 함수 이름 / 함수로 전달할 인수

(2) threading.Thread 파생 클래스를 생성하고 run() 메소드 재정의

```
class sub_class(threading.Thread):

    def __init__(self, args):

        threading.Thread.__init__(self) # 부모 클래스의 초기화 함수를 먼저 호출

    def run(self):

t = sub_class(args) #클래스 객체를 생성하고 인수 전달

t.daemon = True #메인 스레드가 종료되면 서브 스레드도 종료
t.start() # 스레드 시작
```

파생 클래스 이름

클래스 객체를 생성할 때 전달받은 인수

스레드가 시작되면 run() 함수가 실행됨

Python 멀티스레드 구현: threading.Thread 클래스

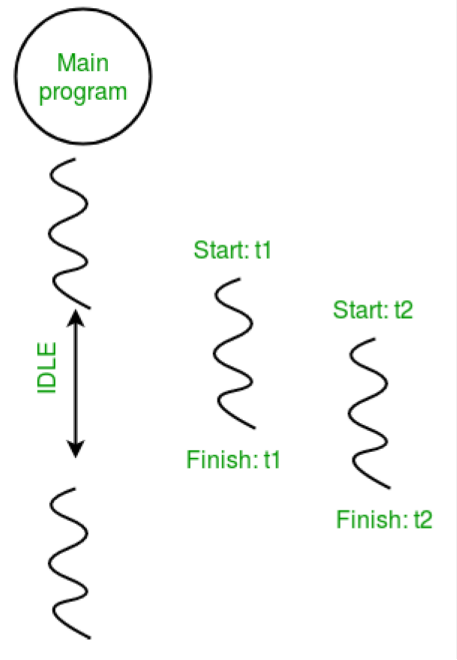
threading 모듈

- 스레드에서 수행할 함수와 클래스 정의
- threading 모듈의 Thread 클래스를 이용해 새로운 스레드 객체 생성
  - ✓ 객체 인자
    - target: 스레드에서 실행할 함수
    - args: 함수에 필요한 인자
- 스레드 시작: start() 메소드

simple_thread.py

```
import threading

def prtSquare(num):
    print("Square: {}".format(num**2))

def prtCube(num):
    print("Cube: {}".format(num**3))

t1 = threading.Thread(target=prtSquare, args=(10,))
t2 = threading.Thread(target=prtCube, args=(10,))

t1.start()  # start thread 1
t2.start()  # start thread 2

t1.join()   # wait until thread 1 is completed
t2.join()   # wait until thread 2 is completed

print('Done!')
```

```
(venv) PS C:\Users\dhkim\net_program> python simple_thread.py
Square: 100
Cube: 1000
Done!
```



Python 멀티스레드 구현: threading.Thread의 파생 클래스

Thread 클래스의 파생 클래스 생성

- threading 모듈의 Thread 클래스의 파생 클래스 정의
- __init__() 메소드 오버라이드
  - ✓ 필요한 정보 저장 (예: 네트워크 프로그래밍의 경우 '소켓 객체' 등)
- run() 메소드 오버라이드
  - ✓ 스레드 내에서 실행할 코드 작성
- 파생 클래스 생성 후 스레드 시작

simple_thread_class.py

```
import threading
import datetime

class myThread(threading.Thread):
    def __init__(self, name, counter):
        super().__init__()
        self.name = name
        self.counter = counter

    def run(self):
        print('\nStarting {}[{}]'.format(self.name, self.counter))
        print_date(self.name, self.counter)
        print('\nExiting {}[{}]'.format(self.name, self.counter))
```

Python 멀티스레드 구현: Thread의 파생 클래스 이용

```
def print_date(threadName, counter):
    today = datetime.date.today()
    print('\n{}[{}]: {}'.format(threadName, counter, today))

thread1 = myThread('Th', 1)
thread2 = myThread('Th', 2)

thread1.start()
thread2.start()

thread1.join()
thread2.join()

print('\nExiting the program')
```

■ 실행 결과

```
(venv) PS C:\Users\dhkim\net_program> python simple_thread_class.py

Starting Th[1]

Starting Th[2]

Th[1]: 2022-03-27

Exiting Th[1]

Th[2]: 2022-03-27

Exiting Th[2]

Exiting the program
(venv) PS C:\Users\dhkim\net_program> python simple_thread_class.py

Starting Th[1]

Th[1]: 2022-03-27
Starting Th[2]

Th[2]: 2022-03-27

Exiting Th[2]


Exiting Th[1]

Exiting the program
```

경쟁 상태 race condition

■ 경쟁 상태

- 공유 자원에 대해 여러 개의 스레드가 동시에 접근을 시도할 때 접근의 타이밍이나 순서 등이 결과값에 영향을 줄 수 있는 상태

■ 임계 구역critical section

- 공유 자원에 접근하는 프로그램의 부분
- 여러 개의 스레드가 동시에 공유 자원을 변경시키려고 하면 경쟁 상태가 발생함
  - ✓ 예측할 수 없는 결과값이 나옴

Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

경쟁 상태

아래 코드는 파이썬3.9 이하에서만 경쟁 상태가 발생함

■ 예제: 2개의 스레드를 가지고 전역변수 'x' 증가시키기

race_cond.py

```
import threading

x = 0   # global variable shared by threads

def increment():
    global x
    x += 1

def thread_task():
    for _ in range(300000):
        increment()

def main_task():
    global x
    x = 0   # initialize x as 0

    t1 = threading.Thread(target=thread_task)
    t2 = threading.Thread(target=thread_task)

    t1.start()
    t2.start()

    t1.join()
    t2.join()

for i in range(10):
    main_task()
    print('Iteration {0}: x = {1}'.format(i, x))
```

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python race_cond.py
Iteration 0: x = 600000
Iteration 1: x = 474017
Iteration 2: x = 600000
Iteration 3: x = 512406
Iteration 4: x = 515163
Iteration 5: x = 600000
Iteration 6: x = 513733
Iteration 7: x = 600000
Iteration 8: x = 600000
Iteration 9: x = 600000
(base) dhkim@gimdaehuiui-MacBookPro NP % python race_cond.py
Iteration 0: x = 493885
Iteration 1: x = 470042
Iteration 2: x = 600000
Iteration 3: x = 600000
Iteration 4: x = 553919
Iteration 5: x = 431400
Iteration 6: x = 468280
Iteration 7: x = 430445
Iteration 8: x = 441005
Iteration 9: x = 600000
```

스레드 동기화 Thread Synchronization

■ 어떻게 경쟁 상태를 방지할 수 있는가?

- 임계구역 보호를 통한 스레드 동기화
- 특정 시간에 1개의 스레드만 임계구역에 접근할 수 있도록 함

■ Lock 클래스 사용

- `acquire()`: 임계구역 진입을 위한 'lock'을 획득
  - ✓ 다른 스레드가 'lock'을 가지고 있지 않을 경우, 'lock'을 획득하여 임계구역 진입
  - ✓ 다른 스레드가 'lock'을 가지고 있을 경우, 'lock'을 반납할 때까지 대기
    - 'lock'이 반납되면 임계구역 진입
- `release()`: 임계구역 종료 후 'lock'을 반납
  - ✓ 'lock'을 반납하여, 대기 중이던 스레드 중 1개의 스레드가 'lock'을 획득하도록 함
- 1개의 스레드가 'lock'을 획득하면, 다른 스레드들은 'lock'이 반납될 때까지 공유 자원에 접근할 수 없음

스레드 동기화

■ 예제: 2개의 스레드를 가지고 전역변수 'x' 증가시키기

race_cond_lock.py

```
import threading

x = 0   # global variable shared by threads

def increment():
    global x
    x += 1

def thread_task(lock):
    for _ in range(300000):
        lock.acquire()  # Acquire lock before accessing the shared data
        increment()
        lock.release()  # Release lock after finishing the access

def main_task():
    global x
    x = 0   # initialize x as 0

    lock = threading.Lock()     # create a lock object

    t1 = threading.Thread(target=thread_task, args=(lock,))
    t2 = threading.Thread(target=thread_task, args=(lock,))

    t1.start()
    t2.start()

    t1.join()
    t2.join()

for i in range(10):
    main_task()
    print('Iteration {0}: x = {1}'.format(i, x))
```

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python race_cond_lock.py
Iteration 0: x = 600000
Iteration 1: x = 600000
Iteration 2: x = 600000
Iteration 3: x = 600000
Iteration 4: x = 600000
Iteration 5: x = 600000
Iteration 6: x = 600000
Iteration 7: x = 600000
Iteration 8: x = 600000
Iteration 9: x = 600000
(base) dhkim@gimdaehuiui-MacBookPro NP % python race_cond_lock.py
Iteration 0: x = 600000
Iteration 1: x = 600000
Iteration 2: x = 600000
Iteration 3: x = 600000
Iteration 4: x = 600000
Iteration 5: x = 600000
Iteration 6: x = 600000
Iteration 7: x = 600000
Iteration 8: x = 600000
Iteration 9: x = 600000
```

멀티스레드 서버

멀티스레드 서버 작성 방법

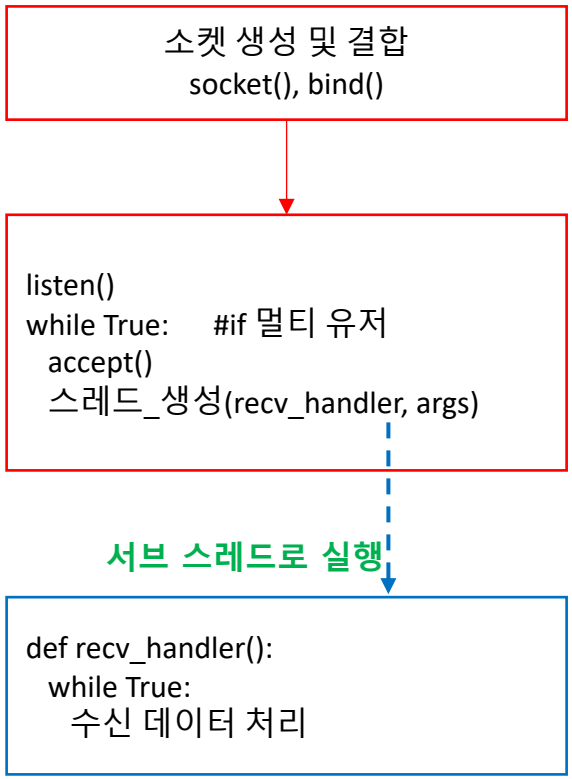
- 서버는 항상 클라이언트의 요청을 기다리고 있어야 함
- 클라이언트의 요청이 들어오면, 클라이언트와 통신할 새로운 소켓(클라이언트 소켓)을 생성하고, 새로운 스레드를 시작함
  - ✓ 새로운 스레드에 생성한 소켓을 인자로 넘겨줌
  - ✓ 스레드는 해당 소켓을 이용해 클라이언트와의 통신을 수행
- 서버는 즉시 다음 클라이언트의 요청을 기다리는 상태로 돌아감

동작

- 각 클라이언트와 (서버의) 스레드 간 통신은 독립적으로 병행 실행됨

주의사항

- 공유 자원에 접근하는 경우, 스레드 동기화 필요



멀티스레드 에코 서버

■ 멀티스레드 에코 서버

- 특정 클라이언트로부터 메시지를 받고, 응답하는 부분을 별도 스레드로 구현
- 메인 스레드는 클라이언트로부터 연결요청을 받은 후, 서브 스레드를 생성하는 역할을 수행
  - ✓ 새로운 스레드를 생성한 후, 다음 연결요청을 기다림
- 서브 스레드는 해당 클라이언트와 통신을 수행

```
(venv) PS C:\Users\dhkim\net_program> python thread_class_echo_server.py
connected by 127.0.0.1 54549
connected by 127.0.0.1 54550
connected by 127.0.0.1 54551
Received message: Hi~
Received message: Hello!
Received message: IoT?
Received message: Who are you?
Received message: I am Daehee.
Received message: Bye!
```

```
(venv) PS C:\Users\dhkim\net_program> pyt
hon echo_client.py
Port No: 2500
Message to send: Hi~
Received message: Hi~
Message to send: Bye!
Received message: Bye!
Message to send:
```

```
(venv) PS C:\Users\dhkim\net_program> pyth
on echo_client.py
Port No: 2500
Message to send: Hello!
Received message: Hello!
Message to send: I am Daehee.
Received message: I am Daehee.
Message to send:
```

```
(venv) PS C:\Users\dhkim\net_program> pyt
hon echo_client.py
Port No: 2500
Message to send: IoT?
Received message: IoT?
Message to send: Who are you?
Received message: Who are you?
Message to send:
```

멀티스레드 에코 서버: threading.Thread 클래스

threading_echo_server.py

```
from socket import *
import threading

port = 2500
BUFSIZE = 1024

def echoTask(sock):
    while True:
        data = sock.recv(BUFSIZE)
        if not data:
            break
        print('Received message:', data.decode())
        sock.send(data)

    sock.close()

sock = socket(AF_INET, SOCK_STREAM)
sock.bind(('', port))
sock.listen(5)

while True:
    conn, (remotehost, remoteport) = sock.accept()
    print('connected by', remotehost, remoteport)
    th = threading.Thread(target=echoTask, args=(conn,))
    th.start()
```

멀티스레드 에코 서버: threading.Thread의 파생 클래스 이용

thread_class_echo_server.py

```
from socket import *
import threading

port = 2500
BUFSIZE = 1024

class ClientThread(threading.Thread):
    def __init__(self, sock):
        threading.Thread.__init__(self)
        self.sock = sock

    def run(self):
        while True:
            data = self.sock.recv(BUFSIZE)
            if not data:
                break
            print('Received message:', data.decode())
            self.sock.send(data)

        self.sock.close()

sock = socket(AF_INET, SOCK_STREAM)
sock.bind(('', port))
sock.listen(5)

while True:
    conn, (remotehost, remoteport) = sock.accept()
    print('connected by', remotehost, remoteport)
    th = ClientThread(conn)
    th.start()
```

멀티스레드 TCP 채팅 프로그램 만들기

- 기존 채팅 프로그램의 문제점
  - 한 번씩 번갈아 가면서 채팅을 해야 함
  - 수신자는 메시지 수신을 위해 recv() 함수를 호출하여 블로킹 되어 있음
  - 따라서, 사용자 입력을 받아서 전송할 수 없음

```
(venv) PS C:\Users\dhkim\net_program> python udp_chat_server.py
<-  Hello
-> Hi, IoT
<-  This is my first chatting program.
-> Me, too. Good luck.
```

```
(venv) PS C:\Users\dhkim\net_program> python udp_chat_client.py
-> Hello
<-  Hi, IoT
-> This is my first chatting program.
<-  Me, too. Good luck.
->
```

- 해결 방법
  - 사용자 입력을 받아 전송하는 부분과, 메시지를 수신하는 부분을 별도 스레드로 구현

멀티스레드 TCP 채팅 프로그램 만들기: 채팅 서버

tcp_thread_chat_server.py

```
from socket import *
import threading

port = 3333
BUFFSIZE = 1024

def sendTask(sock):
    while True:
        resp = input()
        print('->', resp)
        sock.send(resp.encode())

s = socket(AF_INET, SOCK_STREAM)
s.bind(('', port))
s.listen(1)
conn, addr = s.accept()

th = threading.Thread(target=sendTask, args=(conn,))
th.start()

while True:
    data = conn.recv(BUFFSIZE)
    print('<-', data.decode())
```

멀티스레드 TCP 채팅 프로그램 만들기: 채팅 클라이언트

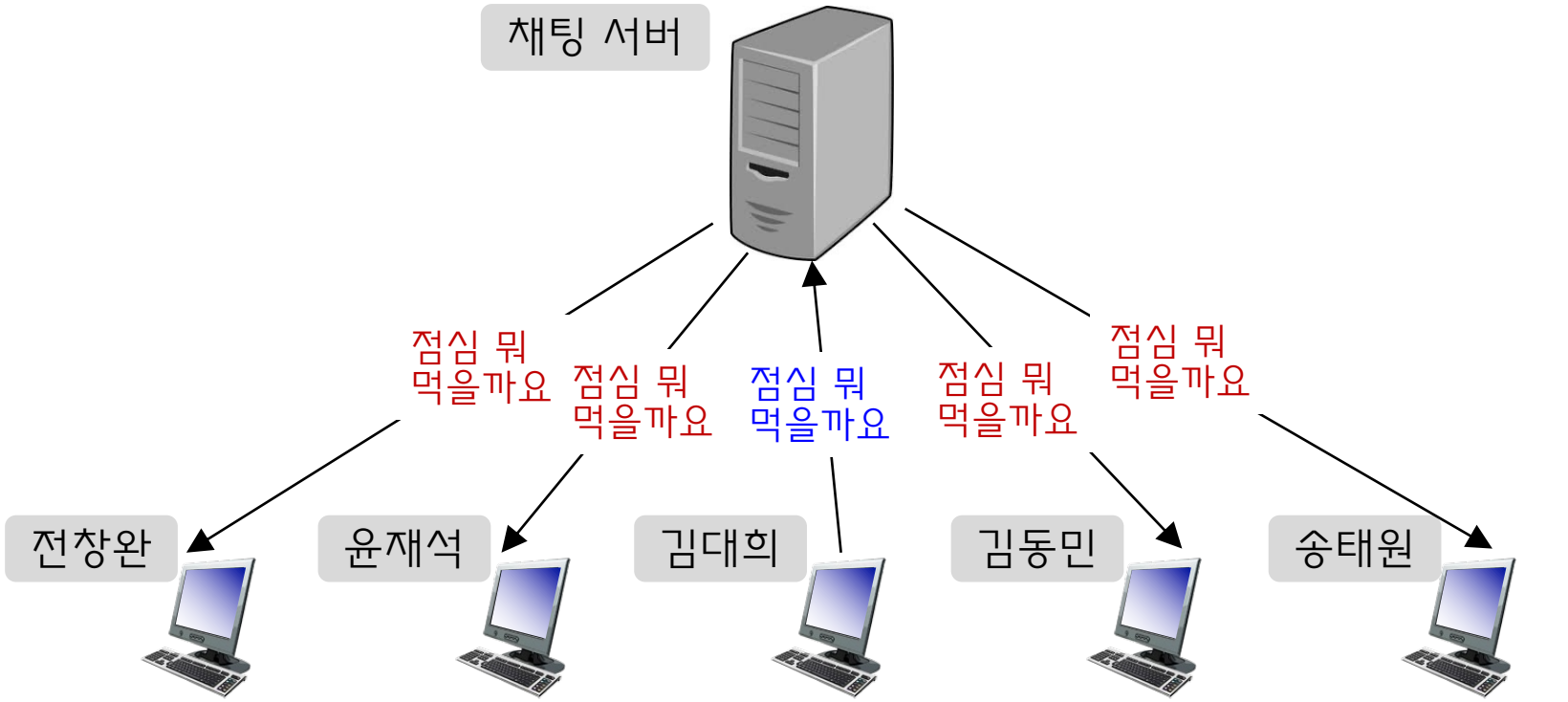
tcp_thread_chat_client.py

```
from socket import *
import threading

port = 3333
BUFFSIZE = 1024

def recvTask(sock):
    while True:
        data = sock.recv(BUFFSIZE)
        print('<-', data.decode())

sock = socket(AF_INET, SOCK_STREAM)
sock.connect(('localhost', port))

th = threading.Thread(target=recvTask, args=(sock,))
th.start()

while True:
    msg = input()
    print('->', msg)
    sock.send(msg.encode())
```

```
(venv) PS C:\Users\dhkim\net_program> python tcp_thread_chat_s
erver.py
Hi
-> Hi
<- Hello
Nice to meet you.
-> Nice to meet you.
<- What is your concern these days?
There are a lot of homeworks!
-> There are a lot of homeworks!
I am so tired.
-> I am so tired.
```

```
(venv) PS C:\Users\dhkim\net_program> python tcp_thread_chat_c
lient.py
<- Hi
Hello
-> Hello
<- Nice to meet you.
What is your concern these days?
-> What is your concern these days?
<- There are a lot of homeworks!
<- I am so tired.
```

UDP를 이용한 단체 채팅 프로그램 만들기



UDP를 이용한 단체 채팅 프로그램 만들기: 서버

■ 채팅 서버

- 채팅 서버는 수신한 메시지를 발신자를 제외한 다른 클라이언트에게 전송
- 새로운 클라이언트가 들어오면, 클라이언트 목록에 저장
- 채팅 클라이언트가 'quit'를 전송하면 해당 클라이언트를 목록에서 삭제

udp_multi_chat_server.py

```
import socket
import time

clients = []    # 클라이언트 목록

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('', 2500))

print('Server Started')
```

UDP를 이용한 단체 채팅 프로그램 만들기: 서버

```
while True:
    data, addr = s.recvfrom(1024)
    # 'quit'을 수신하면 해당 클라이언트를 목록에서 삭제
    if 'quit' in data.decode():
        if addr in clients:
            print(addr, 'exited')
            clients.remove(addr)
            continue

    # 새로운 클라이언트이면 목록에 추가
    if addr not in clients:
        print('new client', addr)
        clients.append(addr)

    print(time.asctime() + str(addr) + ':' + data.decode())

    # 모든 클라이언트에게 전송
    for client in clients:
        if client != addr:
            s.sendto(data, client)
```

UDP를 이용한 단체 채팅 프로그램 만들기: 클라이언트

■ 채팅 클라이언트

- 최초 실행 시, 'ID'를 입력받아 서버로 전송
- 서브 스레드는 채팅 서버로부터 메시지를 수신하여 화면에 출력
- 메인 스레드는 사용자의 입력을 받아 서버로 전송

udp_multi_chat_client.py

```
import socket
import threading

def handler(sock):
    while True:
        msg, addr = sock.recvfrom(1024)
        print(msg.decode())

svr_addr = ('localhost', 2500)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

my_id = input('ID를 입력하세요: ')
sock.sendto(('['+my_id+']').encode(), svr_addr)

th = threading.Thread(target=handler, args=(sock,))
th.daemon = True
th.start()

while True:
    msg = '[' + my_id + '] ' + input()
    sock.sendto(msg.encode(), svr_addr)
```

UDP를 이용한 단체 채팅 프로그램 만들기

```
(venv) PS C:\Users\dhkim\net_program> python udp_multi_chat_server.py
Server Started
new client ('127.0.0.1', 52601)
Mon Mar 28 10:50:11 2022('127.0.0.1', 52601):[전창완]
new client ('127.0.0.1', 52602)
Mon Mar 28 10:50:30 2022('127.0.0.1', 52602):[윤재석]
new client ('127.0.0.1', 52603)
Mon Mar 28 10:50:46 2022('127.0.0.1', 52603):[김대희]
Mon Mar 28 10:51:14 2022('127.0.0.1', 52603):[김대희] 점심 뭐 드실까요?
Mon Mar 28 10:51:33 2022('127.0.0.1', 52603):[윤재석] 맘스터치 어떠세요?
Mon Mar 28 10:51:49 2022('127.0.0.1', 52603):[전창완] 청와삼대 가시죠
Mon Mar 28 10:52:02 2022('127.0.0.1', 52603):[윤재석] 좋습니다. 가시죠
Mon Mar 28 10:52:10 2022('127.0.0.1', 52603):[김대희] 가시죠
```

채팅 서버

```
(venv) PS C:\Users\dhkim\net_program> python udp_multi_chat_client.py
ID를 입력하세요: 전창완
[윤재석]
[김대희]
[김대희] 점심 뭐 드실까요?
[윤재석] 맘스터치 어떠세요?
청와삼대 가시죠
[윤재석] 좋습니다. 가시죠
[김대희] 가시죠
```

전창완

```
(venv) PS C:\Users\dhkim\net_program> python udp_multi_chat_client.py
ID를 입력하세요: 윤재석
[김대희]
[김대희] 점심 뭐 드실까요?
맘스터치 어떠세요?
[전창완] 청와삼대 가시죠
좋습니다. 가시죠
[김대희] 가시죠
```

윤재석

```
(venv) PS C:\Users\dhkim\net_program> python udp_multi_chat_client.py
ID를 입력하세요: 김대희
점심 뭐 드실까요?
[윤재석] 맘스터치 어떠세요?
[전창완] 청와삼대 가시죠
[윤재석] 좋습니다. 가시죠
가시죠
```

김대희

과제8: TCP를 이용한 단체 채팅 프로그램 만들기(멀티스레드)

■ TCP를 이용한 단체 채팅 프로그램 만들기

- 동작은 '슬라이드 22~26'의 UDP 프로그램과 동일
- 서버
  - ✓ 채팅 서버를 '멀티 스레드'로 작성하여야 함
  - ✓ 각 클라이언트가 접속하면 해당 클라이언트를 처리할 스레드를 생성하고, 해당 소켓을 리스트에 저장해야 함
    - 리스트는 현재 접속한 모든 클라이언트들의 소켓을 저장하고 있으며, 모든 클라이언트들에게 메시지를 전송하기 위해 사용됨
  - ✓ 생성된 스레드는 '슬라이드 24'의 동작을 수행
- 클라이언트
  - ✓ 채팅 클라이언트는 '슬라이드 25'의 동작을 수행
  - ✓ TCP이므로 `connect()` 실행 후, `send()/recv()` 동작 수행하면 됨

과제8: TCP를 이용한 단체 채팅 프로그램 만들기(멀티스레드)

■ 과제8

- 서버, 클라이언트 각각 1개의 소스 코드(.py)로 저장
- 실행화면 캡처 파일 (서버 1개, 클라이언트 3개)

■ 소스 코드

- GitHub에 hw8 폴더 생성 후, 소스 코드 업로드
- GitHub 화면 캡처
  - ✓ 폴더 이름과 파일 이름이 보이도록 캡처할 것

■ 제출

- 과제 공지 후 1주일, eClass 제출
- 제출물
  - ✓ 실행화면 캡처 파일(4개)
  - ✓ GitHub 화면 캡처 파일 (1개)

공유 자원을 사용하는 프로그램

파이썬3.9 이하에서만
경쟁 상태가 발생함

서버

- 클라이언트 접속 시, 별도 스레드를 생성하여 처리함
- 스레드 내에서 공유자원(sharedData)을 1씩 10,000,000번 증가시킴

클라이언트

- 서버에 접속 후, 공유자원의 값을 수신하여 출력하는 프로그램

시나리오

- 2개의 클라이언트가 동시에 접속
- 서버는 각 클라이언트를 별도 스레드로 처리
- 각 스레드는 sharedData를 동시에 접근함
- 경쟁상태 발생!!!

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python tcp_threaded_shared_number_server.py
connected by ('127.0.0.1', 61163)
connected by ('127.0.0.1', 61164)
10791062
19376061
```

공유 자원을 사용하는 프로그램: 서버

파이썬3.9 이하에서만
경쟁 상태가 발생함

tcp_threaded_shared_number_server.py

```
from socket import *
import threading

port = 2500
BUFSIZE = 1024

sharedData = 0

def thread_handler(sock):
    global sharedData
    for _ in range(10000000):
        sharedData += 1
    print(sharedData)
    sock.send(str(sharedData).encode())
    sock.close()

s = socket(AF_INET, SOCK_STREAM)
s.bind(('', port))
s.listen(5)

while True:
    client, addr = s.accept()
    print('connected by', addr)
    th = threading.Thread(target=thread_handler, args=(client,))
    th.start()

s.close()
```

공유 자원을 사용하는 프로그램: 클라이언트

파이썬3.9 이하에서만 경쟁 상태가 발생함

tcp_threaded_shared_number_client.py

```
from socket import *

port = 2500
BUFSIZE = 1024

s = socket(AF_INET, SOCK_STREAM)
s.connect(('localhost', port))

print(int(s.recv(BUFSIZE).decode()))

s.close()
```

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python tcp_threaded_shared_number_server.py
connected by ('127.0.0.1', 61163)
connected by ('127.0.0.1', 61164)
10791062
19376061
```

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python tcp_threaded_shared_number_client.py
10791062
```

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python tcp_threaded_shared_number_client.py
19376061
```

공유 자원을 사용하는 프로그램: 개선된 서버

fixed_tcp_shared_number_server.py

```
from socket import *
import threading

port = 2500
BUFSIZE = 1024

sharedData = 0

def thread_handler(sock):
    global sharedData, lock
    lock.acquire()  # 임계영역 보호
    for _ in range(10000000):
        sharedData += 1
    lock.release()
    print(sharedData)
    sock.send(str(sharedData).encode())
    sock.close()

s = socket(AF_INET, SOCK_STREAM)
s.bind(('', port))
s.listen(5)

lock = threading.Lock()

while True:
    client, addr = s.accept()
    print('connected by', addr)
    th = threading.Thread(target=thread_handler, args=(client,))
    th.start()

s.close()
```

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python fixed_tcp_shared_number_server.py
connected by ('127.0.0.1', 61301)
connected by ('127.0.0.1', 61302)
10000000
20000000
connected by ('127.0.0.1', 61303)
connected by ('127.0.0.1', 61304)
30000000
40000000
```

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python tcp_threaded_shared_number_client.py
10000000
(base) dhkim@gimdaehuiui-MacBookPro NP % python tcp_threaded_shared_number_client.py
30000000
```

```
(base) dhkim@gimdaehuiui-MacBookPro NP % python tcp_threaded_shared_number_client.py
20000000
(base) dhkim@gimdaehuiui-MacBookPro NP % python tcp_threaded_shared_number_client.py
40000000
```

Blocking/Non-Blocking, Synchronous/Asynchronous IO

Blocking/Non-Blocking (블로킹/논블로킹)

- '호출되는 함수가 바로 리턴하는냐 마느냐'가 관심
  - ✓ 바로 리턴하지 않으면 Blocking
  - ✓ 바로 리턴하면 Non-Blocking

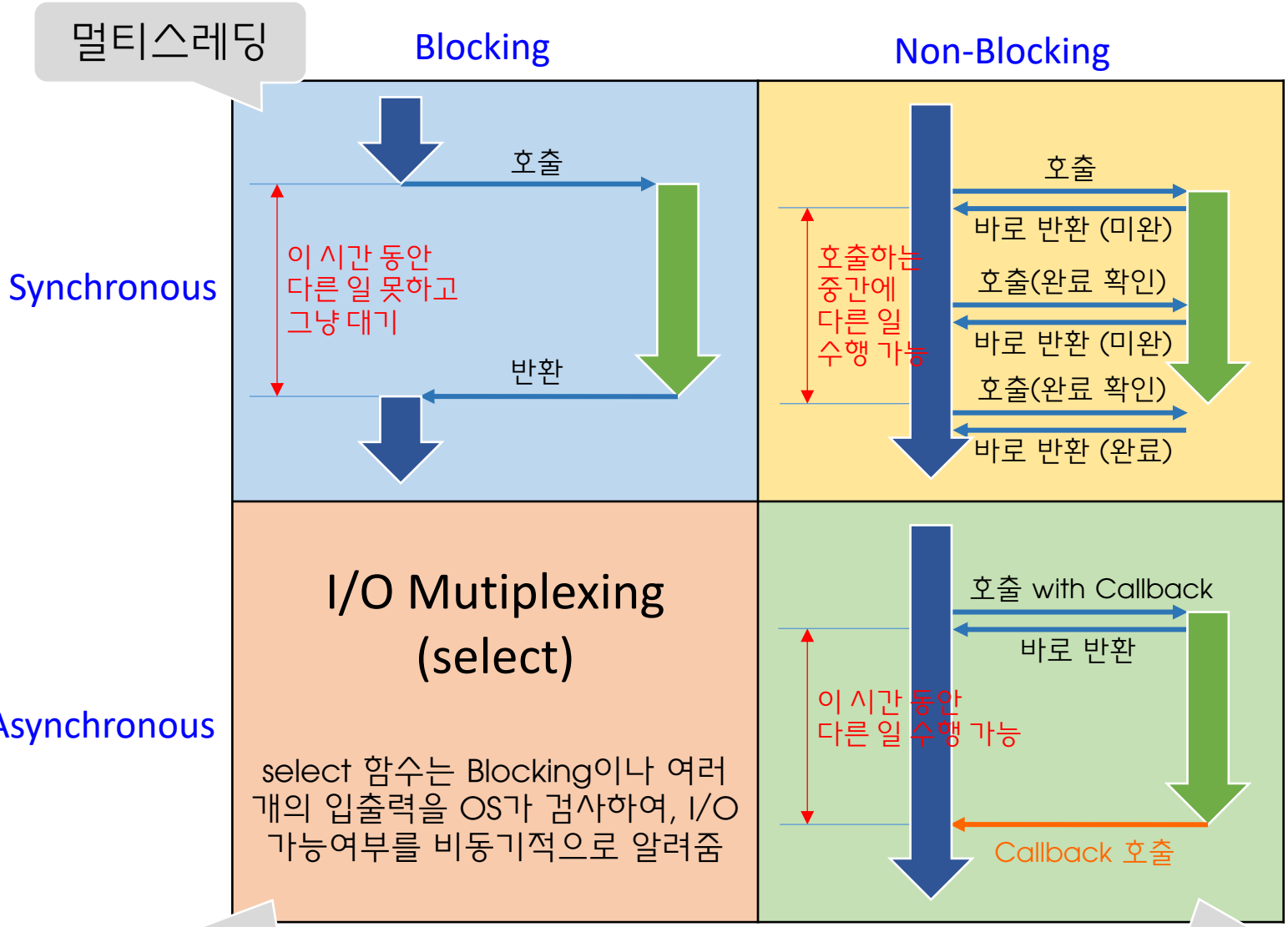
Synchronous/Asynchronous (동기/비동기)

- '호출되는 함수의 작업 완료 여부를 누가 신경 쓰느냐'가 관심사
  - ✓ 호출되는 함수의 작업 완료를 호출한 함수가 신경 쓰면 Synchronous
  - ✓ 호출되는 함수의 작업 완료를 호출된 함수가 신경 쓰면 Asynchronous

	Blocking	Non-Blocking
Synchronous	send/recv	send/recv (O_NONBLOCK)
Asynchronous	I/O Mutiplexing (select)	asyncio

Blocking/Non-Blocking, Synchronous/Asynchronous IO

멀티스레딩



select/selectors

asyncio

select() 함수

I/O(입출력)라고 하면 일반적으로 파일(장치 포함)이나 소켓에 읽고 쓰는 것을 의미함.
리눅스에서 select()는 파일과 소켓에 모두 사용 가능하지만, 윈도우에서 select()는 소켓만 지원함

■ 문제점

- 입출력(파일 또는 소켓)은 비동기적으로 이루어지므로, 응답이 올 때까지 기다리는 블로킹 입출력은 비효율적임
- 수백, 수천 개의 클라이언트가 동시에 접속하는 대규모 서버에서는 클라이언트마다 스레드나 프로세스를 따로 할당하는 작업이 불가능할 수 있음

■ select() 함수

- 여러 개의 소켓(또는 파일)에 대해 비동기 I/O를 지원하는 함수

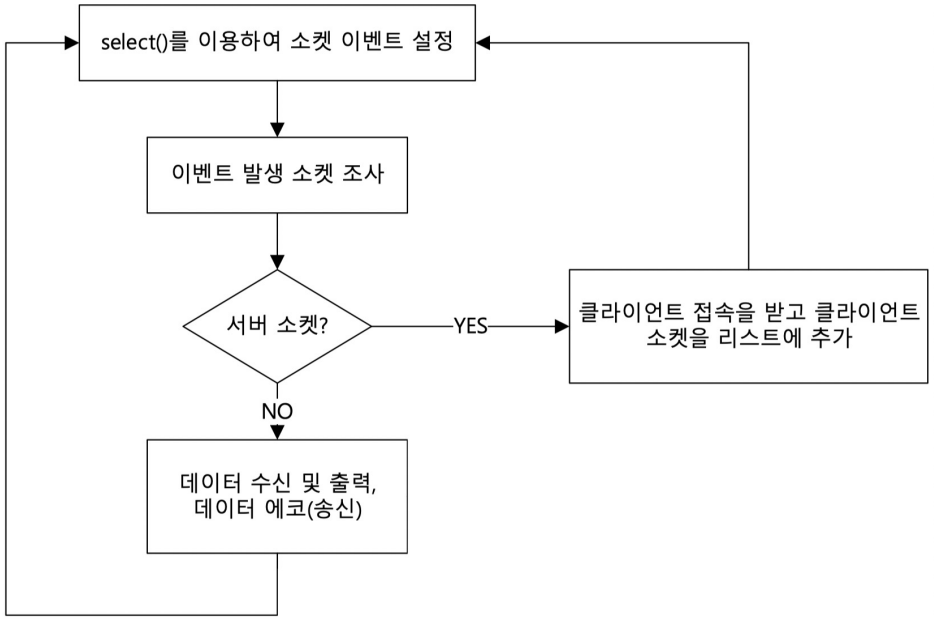
```
import select
r_sock, w_sock, e_sock = select.select(rlist, wlist, xlist[, timeout])
```

- 입력
  - ✓ rlist: 읽기 가능여부를 검사할 소켓 리스트
  - ✓ wlist: 쓰기 가능여부를 검사할 소켓 리스트
  - ✓ xlist: 예외 발생여부를 검사할 소켓 리스트
  - ✓ timeout: 함수 반환할 때까지 기다리는 시간
    - 기본값은 'None'으로 이벤트 발생 때까지 반환하지 않음 (블로킹 모드)
  - ✓ 검사하고 싶지 않은 소켓 리스트의 경우, 빈 리스트([])를 함수 인자로 사용하면 됨

select() 함수

- 출력
  - ✓ r_sock: 읽기 동작이 가능해진 소켓의 리스트
  - ✓ w_sock: 쓰기 동작이 가능해진 소켓의 리스트
  - ✓ e_sock: 예외 발생한 소켓의 리스트
  - ✓ 위의 상황이 발생하면 select() 함수는 반환됨
  - ✓ select() 함수가 반환될 때, 여러 개의 상황이 동시에 발생했을 수 있음

- 동작 예제 (에코 서버 프로그램)
  - ✓ 서버 소켓 생성
    - 서버 소켓: 클라이언트의 연결을 기다리는 소켓
  - ✓ '서버 소켓'을 읽기 가능 소켓 리스트에 추가하고 select() 호출
  - ✓ 클라이언트의 연결이 들어오면, select() 함수가 반환됨
  - ✓ 해당 클라이언트의 소켓을 읽기 가능 소켓 리스트에 추가하고, select() 함수 호출
  - ✓ 새로운 클라이언트의 요청이 있거나, 기존 클라이언트의 요청이 들어오면, select() 함수가 반환
    - 각 소켓을 검사하여 해당 동작을 수행



select()를 이용한 에코 서버

select_server.py

```
import socket, select

socks = []  # 소켓 리스트
BUFFER = 1024
PORT = 2500

s_sock = socket.socket()    # TCP 소켓
s_sock.bind(('', PORT))
s_sock.listen(5)

socks.append(s_sock)    # 소켓 리스트에 서버 소켓을 추가
print(str(PORT) + '에서 접속 대기 중')

while True:
    # 읽기 이벤트(연결요청 및 데이터수신) 대기
    r_sock, w_sock, e_sock = select.select(socks, [], [])

    for s in r_sock:    # 수신(읽기 가능한) 소켓 리스트 검사
        if s == s_sock: # 새로운 클라이언트의 연결 요청 이벤트 발생
            c_sock, addr = s_sock.accept()
            socks.append(c_sock)    # 연결된 클라이언트 소켓을 소켓 리스트에 추가
            print('Client ({}) connected'.format(addr))
        else:           # 기존 클라이언트의 데이터 수신 이벤트 발생
            data = s.recv(BUFFER)
            if not data:
                s.close()
                socks.remove(s)     # 연결 종료된 클라이언트 소켓을 소켓 리스트에서 제거
                continue
            print('Received:', data.decode())
            s.send(data)
```

select()를 이용한 에코 서버

■ 실행 결과

```
(venv) PS C:\Users\dhkim\net_program> python select_server.py
2500에서 접속 대기 중
Client (('127.0.0.1', 60626)) connected
Client (('127.0.0.1', 60627)) connected
Client (('127.0.0.1', 60628)) connected
Received: 숙제가 너무 많아
Received: 화가 난다
Received: 워워~ 릴렉스
Received: 쉬엄쉬엄 해라.
Received: I don't like Chinese Food.
Received: I want Korean Food!
```

서버

```
(venv) PS C:\Users\dhkim\net_program> python echo_client.py
Port No: 2500
Message to send: 숙제가 너무 많아
Received message: 숙제가 너무 많아
Message to send: 화가 난다
Received message: 화가 난다
Message to send:
```

클라이언트

```
(venv) PS C:\Users\dhkim\net_program> python echo_client.py
Port No: 2500
Message to send: 워워~ 릴렉스
Received message: 워워~ 릴렉스
Message to send: 쉬엄쉬엄 해라.
Received message: 쉬엄쉬엄 해라.
Message to send:
```

클라이언트

```
(venv) PS C:\Users\dhkim\net_program> python echo_client.py
Port No: 2500
Message to send: I don't like Chinese Food.
Received message: I don't like Chinese Food.
Message to send: I want Korean Food!
Received message: I want Korean Food!
Message to send:
```

클라이언트

과제9: TCP를 이용한 단체 채팅 프로그램 만들기(select)

■ TCP를 이용한 단체 채팅 프로그램 만들기

- 동작은 '슬라이드 22~26'의 UDP 프로그램과 동일

- 서버
  - ✓ 채팅 서버를 `select()` 함수를 이용하여 작성하여야 함
  - ✓ 새로운 클라이언트가 접속할 때마다, 소켓 리스트에 추가
    - 소켓 리스트는 현재 접속한 모든 클라이언트들의 소켓을 저장하고 있으며, 모든 클라이언트들에게 메시지를 전송하기 위해 사용됨
  - ✓ 메시지 수신 시마다 '슬라이드 24'의 동작을 수행

- 클라이언트
  - ✓ 채팅 클라이언트는 기존대로 멀티스레드로 작성 (과제 8에서 변경사항 없음)

과제9: TCP를 이용한 단체 채팅 프로그램 만들기(select)

■ 과제9

- 서버, 클라이언트 각각 1개의 소스 코드(.py)로 저장
- 실행화면 캡처 파일 (서버 1개, 클라이언트 3개)

■ 소스 코드

- GitHub에 hw9 폴더 생성 후, 소스 코드 업로드
- GitHub 화면 캡처
  - ✓ 폴더 이름과 파일 이름이 보이도록 캡처할 것

■ 제출

- 과제 공지 후 1주일, eClass 제출
- 제출물
  - ✓ 실행화면 캡처 파일(4개)
  - ✓ GitHub 화면 캡처 파일 (1개)

selectors 모듈

selectors 모듈

- 고수준의 효율적인 I/O 다중화를 위한 이벤트 관리 모듈
  - ✓ select: 저수준의 I/O 다중화 모듈
  - ✓ select와 동작은 거의 동일함. 약간의 사용 방법에 차이가 있음
- 특정 객체(소켓)에서 발생하는 이벤트와, 이벤트 발생 시 실행될 callback 함수를 등록
- 이벤트 발생 여부를 확인하여, 이벤트 발생 시 해당 callback 함수를 호출

사용방법

- I/O 다중화 처리를 위한 이벤트 처리기(셀렉터) 객체 생성

```
import selectors
sel = selectors.DefaultSelector()
```

  - ✓ DefaultSelector
    - 현재의 OS 플랫폼에서 사용할 수 있는 가장 효율적인 구현을 사용하는 기본 셀렉터 클래스. 대부분 사용자는 기본적으로 이것을 선택하면 됨

selectors 모듈

- 특정 객체에서 처리할 이벤트와 callback 함수 등록

```
sel.register(fileobj, events, data=None)
```

  - ✓ fileobj: 감시할 객체(소켓)
  - ✓ events: 감시할 이벤트 (읽기 또는 쓰기 가능 여부 감시)
    - selectors.EVENT_READ: 읽기 가능 여부 감시
    - selectors.EVENT_WRITE: 쓰기 가능 여부 감시
    - selectors.EVENT_READ | selectors.EVENT_WRITE: 읽기/쓰기 모두 감시
  - ✓ data: 이벤트 발생 시 반환할 데이터 값. 일반적으로 callback 함수를 등록함

```
예) sel.register(sock, selectors.EVENT_READ, accept)
```

  - ✓ sock 소켓에서 읽기 가능 이벤트(데이터 수신)이 발생하는 것을 감시하고, 이벤트 발생 시 accept를 반환하도록 이벤트 처리기에 등록함

- 이벤트 발생 여부 검사

```
events = sel.select([timeout=None])
```

  - ✓ timeout: 최대 대기 기간(단위: 초)
    - 기본값은 None이고 감시 이벤트가 발생할 때까지 블로킹됨
  - ✓ events: 발생한 이벤트 당 (key, mask) 튜플의 리스트를 반환
    - key: (fileobj, data) 튜플
      - fileobj: 등록해 놓은 객체(소켓), data: 이벤트 발생 시 반환하는 값
    - mask: 발생한 이벤트의 종류 (selectors.EVENT_READ / selectors.EVENT_WRITE)

selectors 모듈

■ 기타 함수

```
sel.unregister(fileobj)
```

- ✓ 이벤트 처리기에 등록된 객체(소켓)을 등록 해지하고, 감시에서 삭제함
- ✓ 소켓을 닫기 전에 반드시 삭제

```
sel.modify(fileobj, events, data=None)
```

- ✓ 등록된 파일 객체의 감시되는 이벤트나 첨부된 데이터를 변경
- ✓ 함수 인자는 register()와 동일
- ✓ unregister(), register()를 수행하는 것과 동일

```
sel.close()
```

- ✓ 이벤트 처리기를 닫음
- ✓ 이벤트 처리기와 관련된 모든 자원을 해제

selectors를 이용한 에코 서버

selectors_server.py

```
import selectors
import socket

sel = selectors.DefaultSelector()          # 이벤트 처리기(셀렉터) 생성

def accept(sock, mask):                    # 새로운 클라이언트로부터 연결을 처리하는 함수
    conn, addr = sock.accept()
    print( ' connected from', addr)
    sel.register(conn, selectors.EVENT_READ, read) # 클라이언트 소켓을 이벤트 처리기에 등록

def read(conn, mask):                      # 기존 클라이언트로부터 수신한 데이터를 처리하는 함수
    data = conn.recv(1024)
    if not data:
        sel.unregister(conn)               # 소켓 연결 종료 시, 이벤트 처리기에서 등록 해제
        conn.close()
        return
    print('received data:', data.decode())
    conn.send(data)

sock = socket.socket()
sock.bind(('', 2500))
sock.listen(5)

# 서버 소켓(신규 클라이언트 연결을 처리하는 소켓)을 이벤트 처리기에 등록
sel.register(sock, selectors.EVENT_READ, accept)
while True:
    events = sel.select()                  # 등록된 객체에 대한 이벤트 감시 시작
    for key, mask in events:               # 발생한 이벤트를 모두 검사
        callback = key.data                # key.data: 이벤트 처리기에 등록한 callback 함수
        callback(key.fileobj, mask)        # callback 함수 호출
```

selectors를 이용한 에코 서버

■ 실행화면

```
(venv) PS C:\Users\dhkim\net_program> python selectors_server.py
connected from ('127.0.0.1', 63723)
connected from ('127.0.0.1', 63724)
connected from ('127.0.0.1', 63725)
received data: 여러분 퇴근 시간입니다.
received data: 다들 퇴근하세요.
received data: 네, 부장님. 먼저 들어가겠습니다.
received data: 그래요. 저는 일이 있어서 조금 있다 가겠습니다.
received data: 아, 저도 남은 일이 있습니다.
received data: 조금 더 있다가 퇴근하겠습.
received data: 니다.
```

서버

```
(venv) PS C:\Users\dhkim\net_program> python echo_client.py
Port No: 2500
Message to send: 여러분 퇴근 시간입니다.
Received message: 여러분 퇴근 시간입니다.
Message to send: 다들 퇴근하세요.
Received message: 다들 퇴근하세요.
Message to send: 그래요. 저는 일이 있어서 조금 있다 가겠습니다.
Received message: 그래요. 저는 일이 있어서 조금 있다 가겠습니다.
Message to send:
```

클라이언트

```
(venv) PS C:\Users\dhkim\net_program> python echo_client.py
Port No: 2500
Message to send: 네, 부장님. 먼저 들어가겠습니다.
Received message: 네, 부장님. 먼저 들어가겠습니다.
Message to send:
```

클라이언트

```
(venv) PS C:\Users\dhkim\net_program> python echo_client.py
Port No: 2500
Message to send: 아, 저도 남은 일이 있습니다.
Received message: 아, 저도 남은 일이 있습니다.
Message to send: 조금 더 있다가 퇴근하겠습.
Received message: 조금 더 있다가 퇴근하겠습.
Message to send: 니다.
Received message: 니다.
Message to send:
```

클라이언트

과제10: 2개의 IoT 디바이스로부터 데이터 수집하기(selectors)

■ 디바이스 1: 온도, 습도, 조도를 측정 제공

- 사용자로부터 'Register' 메시지를 수신하면 온도, 습도, 조도를 3초마다 주기적으로 전송
  - ✓ 온도: 0~40 사이의 임의의 정수를 반환
  - ✓ 습도: 0~100 사이의 임의의 정수를 반환
  - ✓ 조도: 70~150 사이의 임의의 정수를 반환
- 온도, 습도, 조도 전송 방법은 자유롭게 선택

■ 디바이스 2: 심박수, 걸음수, 소모칼로리를 측정 제공

- 사용자로부터 'Register' 메시지를 수신하면 심박수, 걸음수, 소모 칼로리를 5초마다 주기적으로 전송
  - ✓ 심박수: 40~140 사이의 임의의 정수를 반환
  - ✓ 걸음수: 2000~6000 사이의 임의의 정수를 반환
  - ✓ 소모칼로리: 1000~4000 사이의 임의의 정수를 반환
- 심박수, 걸음수, 소모칼로리 전송 방법은 자유롭게 선택

과제10: 2개의 IoT 디바이스로부터 데이터 수집하기(selectors)

- 사용자 (반드시 'selectors' 모듈 사용)
  - 2개의 IoT 디바이스와 TCP 연결
    - ✓ 프로그램이 시작되면, '디바이스 1'과 '디바이스 2'로 'Register' 메시지를 전송
  - 수집한 데이터는 시간정보를 추가하여 파일에 저장 (파일이름: data.txt)
    - ✓ 저장방법: 1줄에 1개의 Device의 데이터 저장(아래 저장 형식 사용)
      - Fri Mar 18 22:55:13 2021: Device1: Temp=20, Humid=50, lilum=100
      - Fri Mar 18 22:57:40 2021: Device2: Heartbeat=80, Steps=2500, Cal=2100
    - ✓ 데이터 수집 파일은 10개(디바이스 당 5개) 이상의 결과를 포함하고 있어야 함

과제10: 2개의 IoT 디바이스로부터 데이터 수집하기(selectors)

■ 과제10

- 디바이스1, 디바이스2, 사용자 각각 1개의 소스 코드(.py)로 저장
- data.txt 파일에 수집한 데이터 저장

■ 소스 코드

- GitHub에 hw10 폴더 생성 후, 소스 코드 업로드
- GitHub 화면 캡처
  - ✓ 폴더 이름과 파일 이름이 보이도록 캡처할 것

■ 제출

- 과제 공지 후 1주일, eClass 제출
- 제출물
  - ✓ data.txt 파일 (1개)
  - ✓ GitHub 화면 캡처 파일 (1개)