컴퓨팅사고와 파이썬 프로그래밍

Ch 12. 파이썬 기반 다중 스레드 프로그래밍과 인터넷 소켓 통신

교수 김 영 탁

영남대학교 정보통신공학과

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- 파이썬에서의 동시처리 (concurrent processing)와 병렬처리 (parallel processing)
- 제네레이터 함수, 제네레이터 수식, 코루틴 (coroutine)
- 파이썬 스레드 (thread) 모듈과 멀티프로세스 (multiprocess) 모듈
- 파이썬 Socket 모듈
- 파이썬 멀티스레드 응용
  - Socket 텍스트 통신
  - OpenCV 기반 영상통신

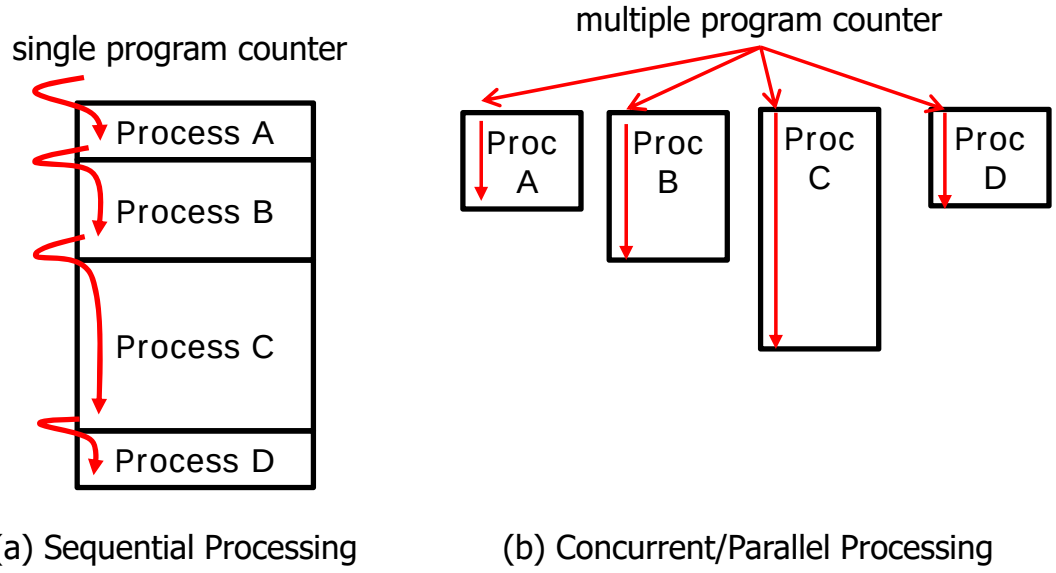
병렬/동시처리, Process/Thread

Task 수행이 병렬로 처리되어야 하는 경우

- **양방향 동시 전송이 지원되는 멀티미디어 정보통신 응용 프로그램 (application)**
  - full-duplex 실시간 전화서비스: 상대편의 음성 정보를 수신하면서, 동시에 나의 음성정보를 전송하여야 함
  - 음성정보의 입력과 출력이 동시에 처리될 수 있어야 함
  - 영상정보의 입력과 출력이 동시에 처리될 수 있어야 함
- **다수의 사건 발생을 등록하며, 우선 순위에 따라 처리하여야 하는 Event Handling/Management**
  - Event가 발생하면 이를 즉시 접수/등록
  - 접수/등록된 event를 우선 순위에 따라 처리
  - Event 처리 프로세서가 다수 사용될 수 있음
  - Event를 처리하고 있는 중에도 더 시급한 event가 발생되면 이를 처리할 수 있도록 운영

멀티태스킹

◆ Sequential Processing vs. Concurrent/Parallel Processing



(a) Sequential Processing (b) Concurrent/Parallel Processing

동시처리와 병렬처리

◆ 동시처리 (concurrent processing)

- 동시처리 (concurrent processing)은 두개 이상의 관련성이 있는 프로세스가 동시에 실행되는 것을 의미.
  예를 들어 task A가 생성되어 종료되기 이전에 task B가 생성되어 실행되며, task A와 task B가 함께 실행되는 경우를 의미.
- 동시처리를 구현하는 다양한 방법이 있으며, 다수의 CPU core가 각각 다른 프로세스를 실행하도록 하는 병렬처리 (parallel processing)과 다수의 task들을 일정한 시간 간격으로 교대시키며 실행하도록 하는 task switching이 있음

◆ 병렬처리 (parallel processing)

- 병렬처리(Parallel processing) 은 두개 이상의 프로세스가 동시에 병렬로 실행되는 것을 의미
- Multi-core CPU에서 병렬처리 수행

◆ 태스크 교체 (task switching)

- 태스크 교체 (task switching)는 다수의 task들을 일정한 시간 간격으로 교대시키며 실행하도록 하며, 하나의 CPU를 공유할 수 있게 하는 방법
- 태스크 교체에서는 실제 어떤 한 순간에는 CPU에 하나의 태스크만 실행되나, 각 테스크들이 짧은 시간동안 실행되고, 자주 교체되는 경우 전체적으로는 그 태스크들이 동시에 실행되는것 처럼 보이게 됨

프로세스 (Process)

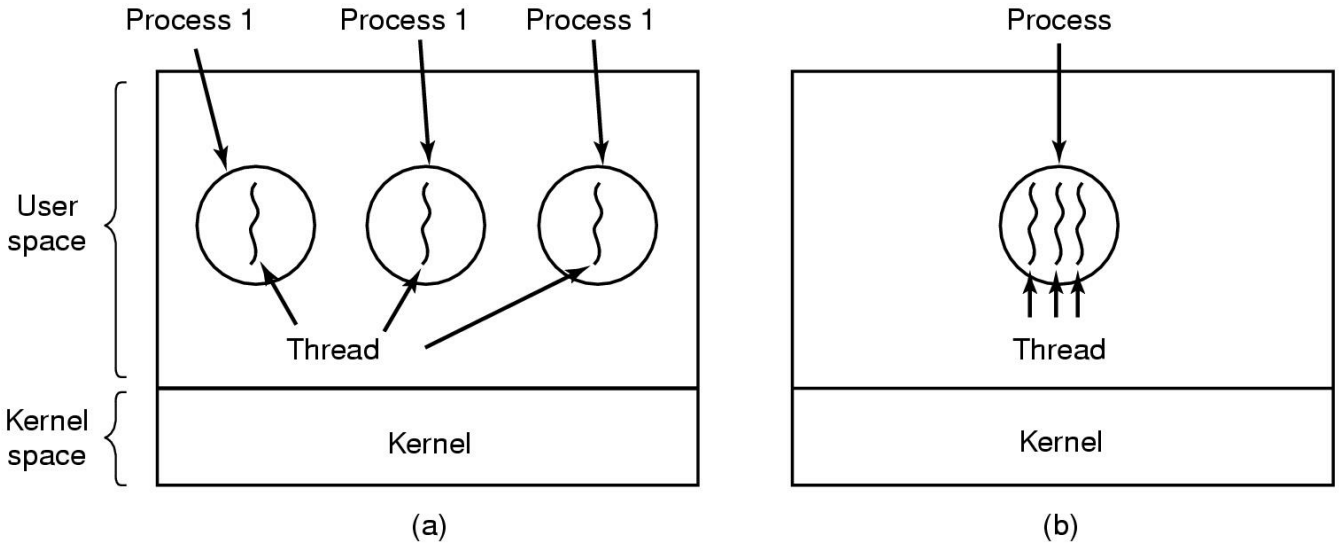
◆ 프로세스 (Process)

- 프로세스 (process)란 프로그램이 수행중인 상태 (***program in execution***)
  - 각 프로세스 마다 개별적으로 메모리가 할당 됨 (text core, initialized data (BSS), non-initialized data, heap (dynamically allocated memory), stack)
- 일반적인 PC나 대부분의 컴퓨터 환경에서 하나의 물리적인 CPU 상에 다수의 프로세스가 실행되는 ***Multi-tasking*** 이 지원되며, 운영체제가 다수의 프로세스를 일정 시간 마다 실행 기회를 가지게 하는 태스크 스케쥴링 (task scheduling)을 지원
- 하나의 프로세스가 실행을 중단하고, 다른 프로세스가 실행될 수 있게 하는 것을 컨텍스트 스위칭 (***Context switching***) 이라 하며, 운영체제의 process scheduling & switching이 프로세스간의 교체를 수행함
- 하나의 물리적인 CPU가 사용되는 시스템에서는 임의의 순간에는 하나의 프로세스만 실행되나, 일정 시간 (예: 100ms)마다 프로세스가 교체되며 실행되기 때문에 전체적으로 다수의 프로그램 들이 동시에 실행되는 것과 같이 보이게 됨

스레드 (Thread)

◆ 스레드 (Thread)

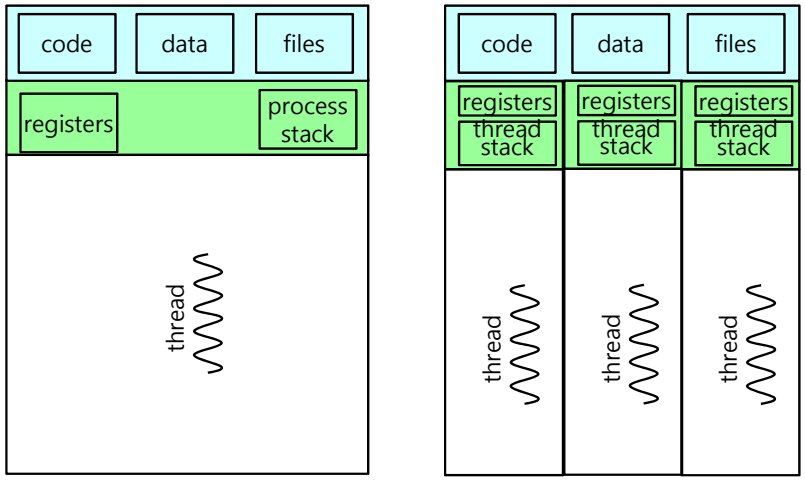
- 스레드는 하나의 프로세스 내부에 포함되는 함수들이 동시에 실행될 수 있게 한 작은 단위 프로세서 (lightweight process)
- 기본적으로 CPU를 사용하는 기본 단위
- 하나의 프로세스에 포함된 다수의 스레드 들은 프로세스의 메모리 자원들 (code section, data section, Heap 등)과 운영체제의 자원들 (예: 파일 입출력 등)을 공유함



프로세스 (Process)와 스레드 (Thread)의 차이점

◆ 멀티스레드 (Multi-thread) 란?

- 어떠한 프로그램 내에서, 특히 프로세스(process) 내에서 실행되는 흐름의 단위.
- 일반적으로 한 프로그램은 하나의 thread를 가지고 있지만, 프로그램 환경에 따라 둘 이상의 thread를 동시에 실행할 수 있다. 이를 멀티스레드(multi-thread)라 함.
- 프로세스는 각각 개별적인 code, data, file을 가지나, 스레드는 자신들이 포함된 프로세스의 code, data, file들을 공유함



(a) single-thread process (b) multi-thread process

Python에서의 병렬/동시 처리 기능

◆ Python에서의 병렬/동시 처리 기능 구현

파이썬에서의 동시처리/ 병렬처리 구현 방법	설 명
코루틴 (coroutine)	coroutine은 대화 방식으로 값을 서로 주고 받으며 작업을 수행. 진입점 (entry point)가 여러 곳일 수 있으며, 종료될 수 도 있고, 멈추었다가 다시 시작할 수 있음. Coroutine은 제네레이터 또는 async 함수로 구현할 수 있음
제네레이터 함수 (generator function)	generator 함수는 순차적으로 데이터를 생성하여 주며, 데이터를 생성하는 generator 함수는 yield를 사용하여 데이터를 전달하고, 사용자 함수에서 next() 함수를 사용하여 다음 데이터를 요청할 때 까지 기다림.
스레드 (thread)	하나의 함수에 다수의 스레드를 생성하여 세부적인 업무를 담당하게 함. 스레드는 별도의 스택과 레지스터를 가지며, 실행 코드, 데이터 및 파일은 공유함.
멀티프로세스 (multi-process)	다수의 프로세스를 구성하여 병렬처리할 수 있게 함. 하나의 CPU에 다수의 코어가 포함되는 경우 각 코어는 개별적인 프로세스를 실행시킬 수 있으며, 하나의 멀티코어 CPU가 다수의 프로세스를 병렬처리로 실행시킬 수 있음.

Generator, Co-routine

Generator, Subroutine, Co-routine

◆ 서브루틴 (Subroutine)

- 일정한 업무 단위(task unit)를 수행하는 함수
- main() 함수는 각 서브루틴을 호출하여 지정된 업무 단위를 수행하게 함
- 서브루틴은 하나의 진입점을 가지며, 일단 호출이 되면 함수의 종료 지점까지 실행되며, 결과를 return하면서 종료됨
- 서브루틴이 종료되면 프로그램의 실행 제어는 그 서브루틴을 호출한 함수로 되돌아감
- 파이썬의 일반 함수들은 서브루틴으로 구현됨

◆ 코루틴 (Co-routine)

- 코루틴은 다수의 진입점을 가질 수 있으며, 지정된 기능을 수행 후 종료되거나, 중간 단계 데이터를 전달한 후 대기 (pause) - 재가동 (resume)을 반복할 수 있음
- 코루틴은 대화형 (interactive mode)으로 실행
- 코루틴은 제네레이터 함수나 비동기 (async) 함수로 구현됨

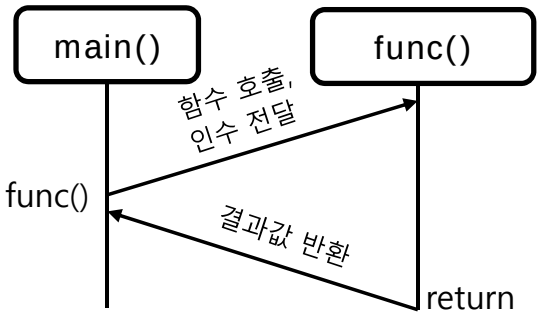
◆ 제네레이터 (Generator)

- 제네레이터는 순차적 데이터를 차례대로 생성하여 제공
- 제네레이터 함수 (generator function) 또는 제네레이터 수식 (generator expression)으로 구현 됨
- 제네레이터 함수에서는 return 대신 yield를 사용하여 중간 단계의 데이터를 전달하며, 함수를 종료하지 않고, 다음 데이터 요청을 기다림

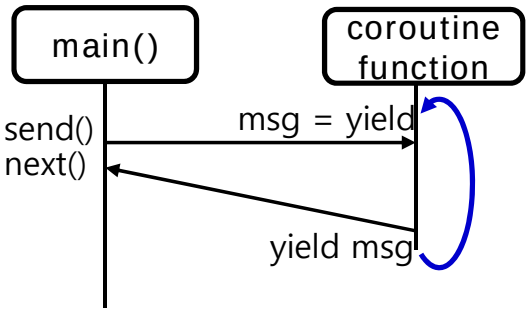
일반적인 함수 호출 (function call) vs. 코루틴 (coroutine)

◆ 일반적인 함수 호출 (subroutine)과 코루틴의 비교

- 서브루틴에서는 함수 호출에 대한 결과를 return으로 반환한 후, 함수가 종료됨
- 코루틴에서는 함수의 중간 결과를 yield로 전달한 후, 함수를 종료하지 않고 대기; 대입 연산자 오른쪽에 있는 yield를 사용하여 데이터를 전달 받음
- 코루틴을 호출하는 함수는 next()를 사용하여 데이터를 전달받으며, send()를 사용하여 데이터를 전달



(a) Subroutine



(b) Coroutine

제네레이터 함수

◆ 제네레이터 (Generator)

- 제네레이터는 데이터를 순차적으로 생성하여 전달
- 제네레이터는 제네레이터 함수 (generator function) 또는 제네레이터 수식 (generator expression)으로 구현
- 제네레이터 함수는 return 대신 yield를 사용하여 (중간) 결과값을 전달하거나, 전달 받음

```
# Generator function with yield

def myRange(n):
    i = 0
    while i < n:
        yield i #using yield instead of return
        i += 1

for i in myRange(5):
    print("returned value from myRange(n) : {:2d}".format(i))
```

```
returned value from myRange(n) :  0
returned value from myRange(n) :  1
returned value from myRange(n) :  2
returned value from myRange(n) :  3
returned value from myRange(n) :  4
```

제네레이터 함수의 기본 메소드

◆ Generator 함수의 기본 메소드

Generator method	설 명
generator.__next__()	제네레이터 함수를 시작 (start) 시키거나, yield에 의해 멈추어 있는 지점에서 실행을 재개 (resume) 시킴. 이 메소드는 for-loop 또는 내장함수 next()의 실행에서 호출됨
generator.send(value)	제네레이터 함수의 실행을 재개 시키고 value를 제네레이터 함수의 (yield)에 전달. send() 함수는 제네레이터가 yield를 사용하여 전달하는 데이터를 받음. 제네레이터의 반복자 (iterator)가 중지된 경우 StopIteration 예외상황을 발생시킴 next() 내장 함수 대신 send(None) 메소드를 사용하여 제네레이터를 시작 시킬 수 있음
generator.throw(type[, value[, traceback]])	제네레이터에 type의 예외 상황을 발생시킴
generator.close()	제네레이터 함수를 중지시키고, GeneratorExit을 발생시킴

제네레이터 함수

◆ 제네레이터 함수의 생존 기간

```
# Generator function with yield and next

def myRange(n):
    i = 0
    while i < n:
        yield i #using yield instead of return
        i += 1
MAX_RANGE = 5
itr = myRange(MAX_RANGE)
for i in myRange(MAX_RANGE + 1):
    d = next(itr)
    print("returned value from myRange(n) : {:2d}".format(d))
```

```
returned value from myRange(n) :  0
returned value from myRange(n) :  1
returned value from myRange(n) :  2
returned value from myRange(n) :  3
returned value from myRange(n) :  4
Traceback (most recent call last):
  File "C:/YTK-Progs/2019-8 Book (Pytl
enerator function/Generator function 
    d = next(itr)
StopIteration
```

제네레이터 함수

◆ 제네레이터 함수 구조의 피보나치 수열 생성

```
# Fibonacci Series with Generator Function

def genFibo():
    a, b = 0, 1
    while True:
        yield a #using yield instead of return
        a, b = b, a+b
i = 1
for f in genFibo():
    s = "{:2d}-th Fibonacci Series : {:10d}".format(i, f)
    print(s)
    i += 1
    if i > 20:
        break
```

```
 1-th Fibonacci Series :          0
 2-th Fibonacci Series :          1
 3-th Fibonacci Series :          1
 4-th Fibonacci Series :          2
 5-th Fibonacci Series :          3
 6-th Fibonacci Series :          5
 7-th Fibonacci Series :          8
 8-th Fibonacci Series :         13
 9-th Fibonacci Series :         21
10-th Fibonacci Series :         34
11-th Fibonacci Series :         55
12-th Fibonacci Series :         89
13-th Fibonacci Series :        144
14-th Fibonacci Series :        233
15-th Fibonacci Series :        377
16-th Fibonacci Series :        610
17-th Fibonacci Series :        987
18-th Fibonacci Series :       1597
19-th Fibonacci Series :       2584
20-th Fibonacci Series :       4181
```

제네레이터 함수

◆ 제네레이터 함수에서 yield from를 사용하여, 다른 제네레이터 함수의 호출

- 제네레이터 함수에서 일부분의 기능을 다른 제네레이터 함수를 사용하여 구현할 수 있으며, ***yield from***를 사용하여 다른 제네레이터 함수에서 생성된 데이터를 전달

```
# Generator Function with yield from

def genListUpDown(n):
    yield from range(n)
    yield from range(n, 0, -1)

L = list(genListUpDown(5))
print("List generated with genListUpDown(5) :\n", L)
```

```
List generated with genListUpDown(5) :
 [0, 1, 2, 3, 4, 5, 4, 3, 2, 1]
```

제네레이터 수식 (Generator Expression)

◆ 제네레이터 수식 (Generator Expression)

- 제네레이터 수식은 괄호 안에 for가 포함되어 있는 반복자 (iterator)를 반환
- 제네레이터 수식은 리스트와 동일한 기능을 제공하며,
  대괄호 "[,]" 대신 소괄호 "(,)" 로 표시
- 사전에 데이터를 생성하여 메모리에 저장하는 리스트와는 달리
  제네레이터 함수나 제네레이터 수식은 필요할 때 데이터를 생성하여 사용할 수 있게 함

```
# Generator expression

genList = (x*x for x in range(5))

print("genList : ", genList)
L = list(genList)
print("L : ", L)
```

```
genList :  <generator object <genexpr> at 0x02CAA3B0>
L :  [0, 1, 4, 9, 16]
```

yield를 사용하여 co-routine에 인수 전달

◆ yield를 사용하여 외부 데이터를 전달 받는 코루틴

```
# Co-routine that receives data from outside

def echoMsg():
    print("Starting echoMsg() co-routine ....")
    while True:
        msg = (yield) # data is obtained from yield
        print("Received message : ", msg)

genMsg = echoMsg()
print("genMsg : ", genMsg)
next(genMsg)

genMsg.send("Hello")
genMsg.send("Welcome")
genMsg.send(1)
genMsg.send(2)
genMsg.close()
genMsg.send(3)
```

```
genMsg :  <generator object echoMsg at 0x02F586B0>
Starting echoMsg() co-routine ....
Received message :  Hello
Received message :  Welcome
Received message :  1
Received message :  2
Traceback (most recent call last):
  File "C:/YTK-Progs/2019-8 Book (Python)/ch 12 Par
enerator function/EchoMsg - receives data using yie
    genMsg.send(3)
StopIteration
```

코루틴에서의 예외처리 (exception handling)

```
# Co-routine that receives data from outside with exception handling

def echoMsg():
    print("Starting echoMsg() co-routine ....")
    try:
        while True:
            try:
                msg = (yield) # data is obtained from yield
                print("Received message : ", msg)
            except TypeError as te:
                print("TypeError : ", te)
            except ValueError as ve:
                print("ValueError : ", ve)
            except IndexError as ie:
                print("IndexError : ", ie)
            except Exception as e:
                print("Exception : ", e)
    finally:
        print("Generator function is closed now")

genMsg = echoMsg()
next(genMsg)

genMsg.send("Greetings !!")
genMsg.throw(TypeError, "TypeError message !")
genMsg.throw(ValueError, "ValueError message !")
genMsg.throw(IndexError, "IndexError message !")
genMsg.throw(Exception, "Exception message !")
genMsg.send("Hello")
genMsg.close()
```

```
Starting echoMsg() co-routine ....
Received message :  Greetings !!
TypeError :  TypeError message !
ValueError :  ValueError message !
IndexError :  IndexError message !
Exception :  Exception message !
Received message :  Hello
Generator function is closed now
```

파이썬의 다중 스레드 프로그래밍

Python Threading

◆ 파이썬 스레드 관련 threading 모듈의 메소드와 속성 (1)

threading 모듈 함수/메소드, 속성	설명
Thread(target, name, args)	스레드 생성 (스레드 함수는 target으로 설정, 이름은 name으로 설정, 스레드 함수에 전달되는 인수들은 args로 설정)
name	스레드의 이름
daemon	스레드가 데몬으로 실행될 것인가를 설정하며, 스레드의 start()가 실행되기 이전에 설정되어야 함 (기본 설정값은 daemon = False) 데몬 스레드는 백그라운드에서 실행되며 메인스레드가 종료할 때 함께 종료됨
start()	스레드의 동작이 시작되도록 하며 내부적으로 run() 메소드가 호출하여 줌
run()	start()에 의하여 내부적으로 호출되며 스레드를 실행시킴
join()	스레드가 종료될 때까지 join() 메소드를 호출한 메인스레드의 진행을 중지시키고 대기하게 함
is_alive()	스레드가 활성화되어 실행상태에 있는지 확인
get_ident()	현재 실행중인 스레드의 식별자 (identifier)를 반환

Python Threading

◆ 파이썬 스레드 관련 threading 모듈의 메소드와 속성 (2)

threading 모듈 함수/메소드, 속성	설명
active_count()	현재 활성화 상태로 실행중인 스레드의 개수를 파악
current_thread()	현재 실행중인 스레드의 객체를 반환
enumerate()	현재 실행중인 모든 스레드의 객체 리스트를 반환
main_thread()	메인스레드 객체를 반환
settrace()	threading 모듈에 의해 생성된 스레드들의 추적 (trace) 기능을 설정
setprofile()	threading 모듈에 의해 생성된 스레드들의 프로파일링 기능을 설정
stack_size()	새로운 스레드를 생성할 때 스택 크기를 알려줌

파이썬 스레드 생성

- **Thread(group=None, target=None, name=None, args=(), kwargs={}, *, daemon=None)**
  - *group* 는 향후 스레드 그룹 클래스에서 사용 예정. 현재는 None으로 설정.
  - *target*은 run() 메소드에서 호출되는 객체. 기본값은 None.
  - *name*은 스레드 이름. 기본값으로 "Thread-N" 형식의 중복되지 않는 이름 생성.
  - *args*는 target 메소드 호출에서 사용되는 인수 튜플 (argument tuple). 기본값은 ().
  - *kwargs* 는 target 메소드 호출에서 사용되는 키워드 인수의 딕셔너리. 기본값은 {}.
  - *daemon* 은 스레드가 데몬으로 생성되는지를 명시. 기본값은 None이며, 이 경우 데몬 속성은 이 스레드를 생성하는 현재의 스레드로 부터 상속 받음.

Multi-tasking with Python threads

```
# multi-tasking with Python threads (1)
import threading
import time
from tkinter import *

def Sample_Thread(name, max_count=10):
    count = 0
    my_name = name
    print("\nThread_{0} is activated".format(my_name))
    while True:
        print("\nThread_{0} is still running (count: {1})".format(my_name, count), end=" ")
        count += 1
        if count > max_count:
            break
        time.sleep(1)
    print("\nThread ({0}) is terminating".format(my_name))

def main():
    thread_A = threading.Thread(target = Sample_Thread, args=('A',10))
    thread_B = threading.Thread(target = Sample_Thread, args=('B',10))
    thread_C = threading.Thread(target = Sample_Thread, args=('C',10))

    threads = []
    threads.append(thread_A)
    threads.append(thread_B)
    threads.append(thread_C)
```

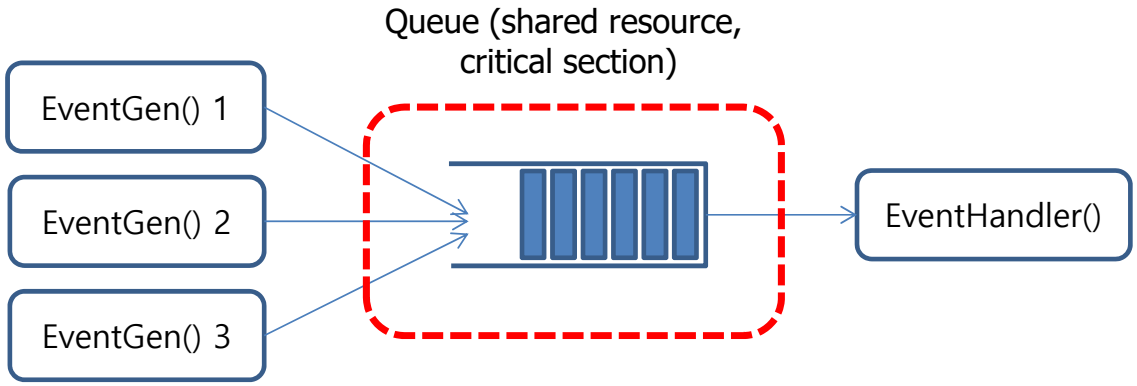
```
# multi-tasking with Python threads (1)

    for t in threads:
        print("\nActivating thread_{}".format(t.getName()))
        t.daemon = True
        t.start()
        t.join(timeout = 1.0)
    while (1):
        all_threads_terminated = True
        for t in threads:
            if t.is_alive():
                all_threads_terminated = False
        if all_threads_terminated == True:
            break
        else:
            num_active_threads = threading.active_count()
            active_threads = threading.enumerate()
            print("\nmain():: still {} threads (including main) are active :“\
                .format(num_active_threads))
            print(“ ", active_threads)
            time.sleep(1)
            continue
    print("main():: all threads are terminated now !!")
if __name__ == '__main__':
    main()
```

```
Thread_C is still running (count: 8)
Thread_B is still running (count: 9)
Thread_A is still running (count: 10)
main():: still 5 threads (including main) are active ....

Thread (A) is terminating
main():: active threads:
 [<_MainThread(MainThread, started 15976)>, <Thread(SockThread
, started daemon 16868)>, <Thread(Thread-1, stopped daemon 137
20)>, <Thread(Thread-2, started daemon 16632)>, <Thread(Thread
-3, started daemon 12476)>]

Thread_C is still running (count: 9)
Thread_B is still running (count: 10)
main():: still 4 threads (including main) are active ....

Thread (B) is terminating
main():: active threads:
 [<_MainThread(MainThread, started 15976)>, <Thread(SockThread
, started daemon 16868)>, <Thread(Thread-2, stopped daemon 166
32)>, <Thread(Thread-3, started daemon 12476)>]

Thread_C is still running (count: 10)
main():: still 3 threads (including main) are active ....

Thread (C) is terminating
main():: active threads:
 [<_MainThread(MainThread, started 15976)>, <Thread(SockThread
, started daemon 16868)>, <Thread(Thread-3, stopped daemon 124
76)>]
main():: all threads are terminated now !!
>>>
```

병렬/동시 처리에서의 공유자원 관리
- 임계구역 (critical section)과 세마포

◆ LIFO Queue를 사용한 정보 전달

- 스레드 간에 정보/메시지/신호를 전달하기 위하여 FIFO 동작을 수행하는 queue (예: Circular Queue, Priority Queue)를 사용
- Queue의 end에 정보를 추가하는 enqueue()
- Queue의 front에 있는 정보를 추출하는 dequeue()
- Queue는 다수의 스레드가 공유하는 자원 (shared resource) 이며, **임계구역** (critical section)으로 보호되어야 함



Critical Section (임계구역)

◆ Critical Section (임계구역)

- 다중 스레드 사용을 지원하는 운영체제는 프로그램 실행 중에 스레드 또는 프로세스간에 교체가 일어 날 수 있게 하여, 다수의 스레드/프로세스가 병렬로 처리될 수 있도록 관리
- Context switching이 일어나면, 현재 실행 중이던 스레드/프로세스의 중간 상태가임시 저장되고, 다른 스레드/프로세스가 실행됨
- 프로그램 실행 중에 특정 구역은 실행이 종료될 때 까지 스레드/프로세서 교체가 일어 나지 않도록 관리하여야 하는 경우가 있음
- 아래의 인터넷 은행 입금 및 출금 스레드 예에서 critical section으로 보호하여야 할 구역은 ?

```
1.  Thread_Deposit (int deposit)
2.  {
3.      // account is shared variable
4.      l_account = g_account;

5.      l_account =
            l_account + deposit;

6.      g_account = l_account;

7.      print(l_account);
8.      . . . .
9.  }
```

shared resource

은행잔고
g_account

```
1.  Thread_Withdraw (int withdraw)
2.  {
3.      // account is shared variable
4.      l_account = g_account;

5.      l_account =
            l_account - withdraw;

6.      g_account = l_account;

7.      print(l_account);
8.      . . . .
9.  }
```

공유데이터의 정상적인 사용

실행 순서	Thread_Deposit (deposit = 70)	account (g_acct = 100)	Thread_Withdraw (widthdraw = 80)
0	Thread Switching		
1	l_acct = g_acct	100	
2	l_acct = l_acct + deposit;		
3	g_acct = l_acct;	170	
4	print(l_acct);		
5	Thread Switching		
6			l_acct = g_acct;
7			l_acct = l_acct - widthdraw;
8		90	g_acct = l_acct;
9		**90**	print(l_acct);
10	Thread Switching		

임계구역이 설정없이 스레드 교체가 발생되어 공유 데이터에 문제가 발생되는 경우

실행 순서	Thread_Deposit (deposit = 70)	account (g_acct = 100)	Thread_Withdraw (widthdraw = 80)
0	Thread Switching		
1	l_acct = g_acct	100	
2	Thread Switching		
3			l_acct = g_acct;
4			l_acct = l_acct - widthdraw;
5		20	g_acct = l_acct;
6			print(l_acct);
7	Thread Switching		
8	l_acct = l_acct + deposit;		
9	g_acct = l_acct;	170	
10	print(l_acct);	**170**	
11	Thread Switching		

파이썬의 Critical Section (임계구역) 관리

◆threading.Lock()

임계구역 관련 파이썬함수	설명
import threading LOCK = threading.Lock()	임계구역 (critical section) 세마포 (semaphore) 생성
LOCK.acquire()	임계구역을 사용할 수 있도록 잠금 (lock)을 확보하도록 함. acquire() 메소드 호출에서 blocking = True로 설정되었을 때 지정된 잠금을 다른 스레드가 확보하고 있는 상태이면 그 잠금이 해제될 때 까지 대기한 후 잠금을 확보하고 True를 반환함. acquire() 메소드 호출에서 blocking = False로 설정되었을 때 지정된 잠금을 다른 스레드가 확보하고 있는 상태이면 대기하지 않고 False를 반환함.
LOCK.release()	임계구역을 관리하는 잠금 (lock)을 해제시키며, 그 잠금을 확보한 스레드와 다른 스레드가 호출할 수 있음. 만약 잠금이 locked 상태에 있으면 unlocked 상태로 변경시키며, 이 잠금을 기다리고 있는 스레드 하나를 선정하여 잠금을 확보할 수 있게 함.

파이썬의 Critical Section (임계구역) 관리

◆ 임계구역 (Critical section) 설정 예

```
1.   Thread_Deposit (deposit, LOCK)
2.   {
3.      // account is shared variable
4.      LOCK.acquire(False)
5.      l_account = g_account;

6.      l_account =
           l_account + deposit;

7.      g_account = l_account;
8.      print(l_account);
9.      LOCK.release()

10.     . . . .
11.  }
```

shared resource

은행잔고
g_account

LOCK = threading.Lock()

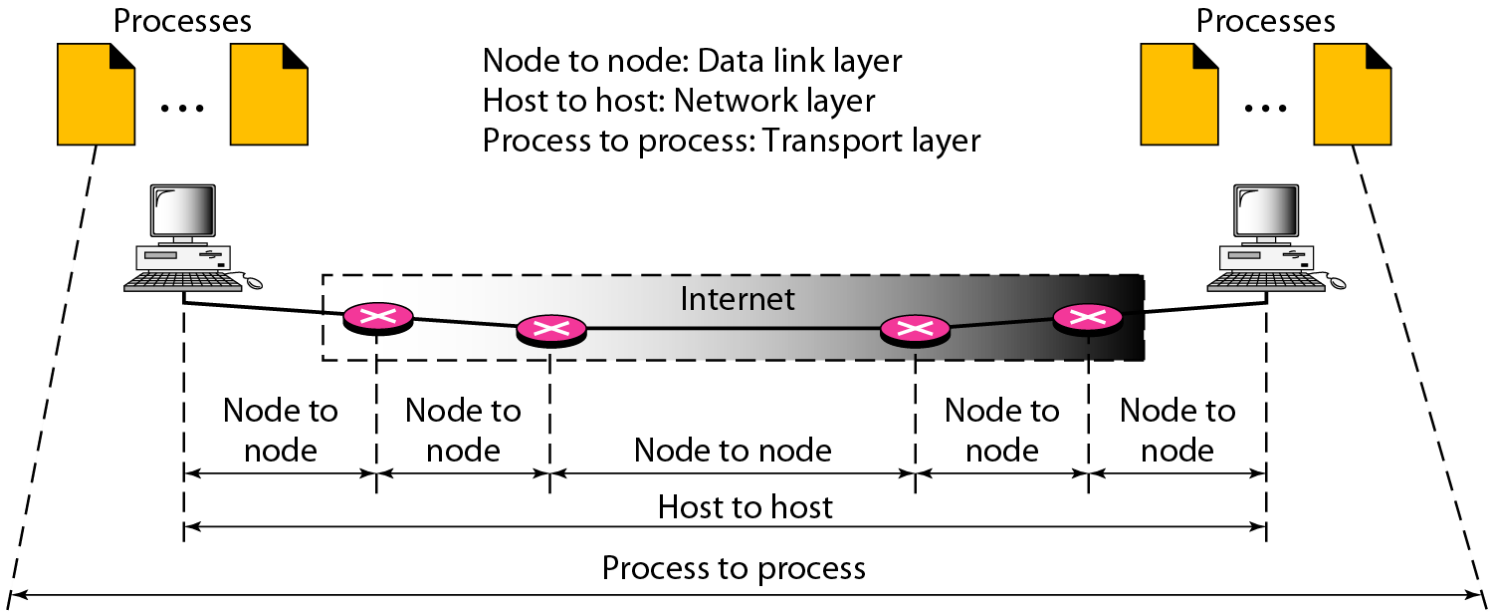
```
1.   Thread_Withdraw (withdraw, LOCK)
2.   {
3.      // account is shared variable
4.      LOCK.acquire(False)
5.      l_account = g_account;

6.      l_account =
           l_account - withdraw;

7.      g_account = l_account;
8.      print(l_account);
9.      LOCK.release()

10.     . . . .
11.  }
```

인터넷 통신에서 사용되는 소켓 (socket)

단말장치의 프로세스간 통신

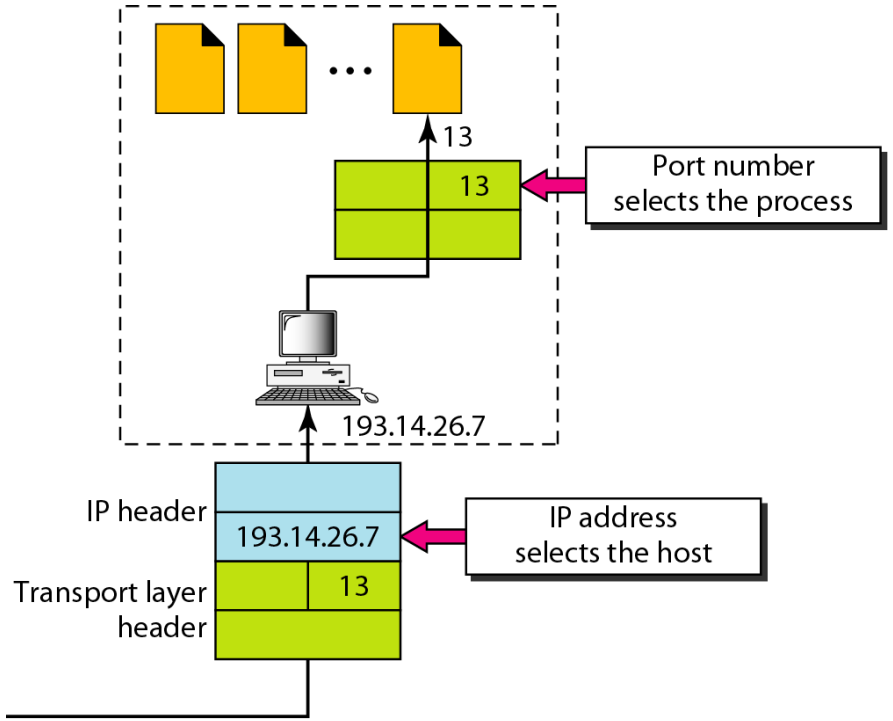
◆ 인터넷의 정보 전달을 위한 통신 계층

- 데이터 링크 계층: node-to-node delivery of frame
- 네트워크 계층: 호스트 컴퓨터간 패킷 (packet/datagram) 전달
- 트랜스포트 계층 (transport layer): 응용 데이터 (메시지)의 프로세스 간 전달



◆ 인터넷의 IP 주소와 포트 번호 (Port number)

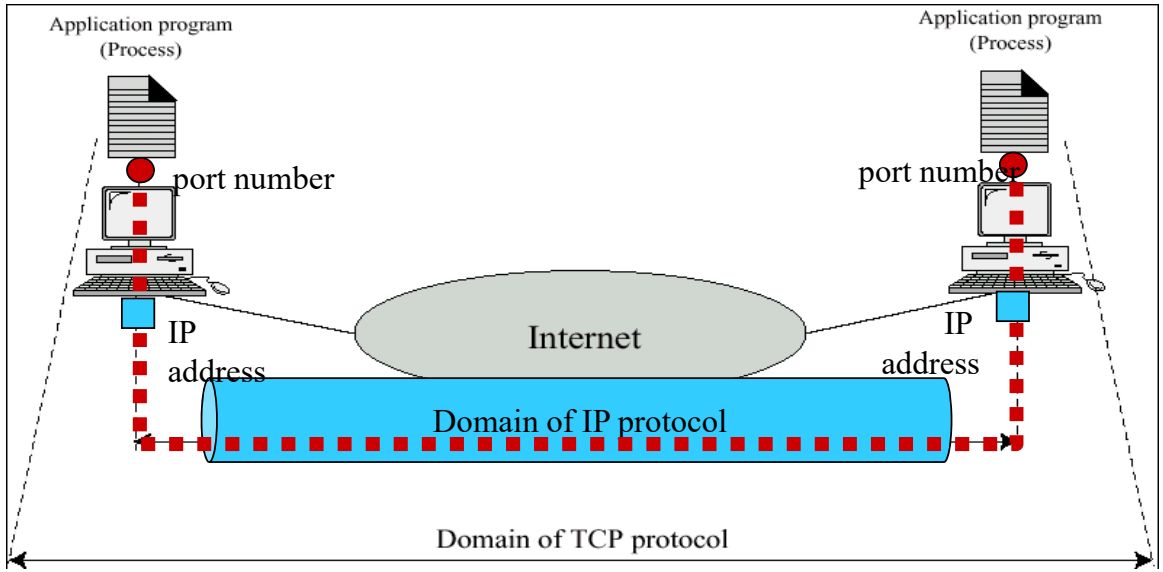
- IP (internet protocol) 주소는 호스트 컴퓨터의 네트워크 접속 (network interface)별로 설정되며, 하나의 호스트 컴퓨터에 다수의 IP 주소 설정 가능
- 포트 번호 (Port number)는 호스트 컴퓨터에서 실행되는 프로그램, 즉 프로세스별로 설정



인터넷 통신에서의
TCP (Transmission Control Protocol)

◆ 인터넷에서의 프로세스간 통신

- TCP (Transmission Control Protocol)는 연결형 통신을 위한 트랜스포트 계층 프로토콜이며, 네트워크 계층 프로토콜인 IP이 상위 계층 프로토콜
- IP 계층의 IP 주소는 호스트 컴퓨터를 지정하며, TCP 프로토콜의 포트번호 (port number)는 통신 기능을 수행하는 프로그램/프로세스를 지정



TCP 프로토콜 포트 번호

◆ TCP 프로토콜에서 설정된 대표적인 포트번호

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	CharGen	Returns a string of characters
20	FTP, data	파일 전송 프로토콜 (데이터 전송 연결)
21	FTP, control	파일 전송 프로토콜 (제어 연결)
23	Telnet	원격 터미널 접속 (Terminal Network)
25	SMTP	전자우편 전송 프로토콜 (Simple Mail Transfer Protocol)
53	DNS	웹의 도메인 이름 서버 (Domain Name Server)
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	웹 문서 전송을 위한 Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

TCP의 연결성, 양방향 전이중, 신뢰성 통신

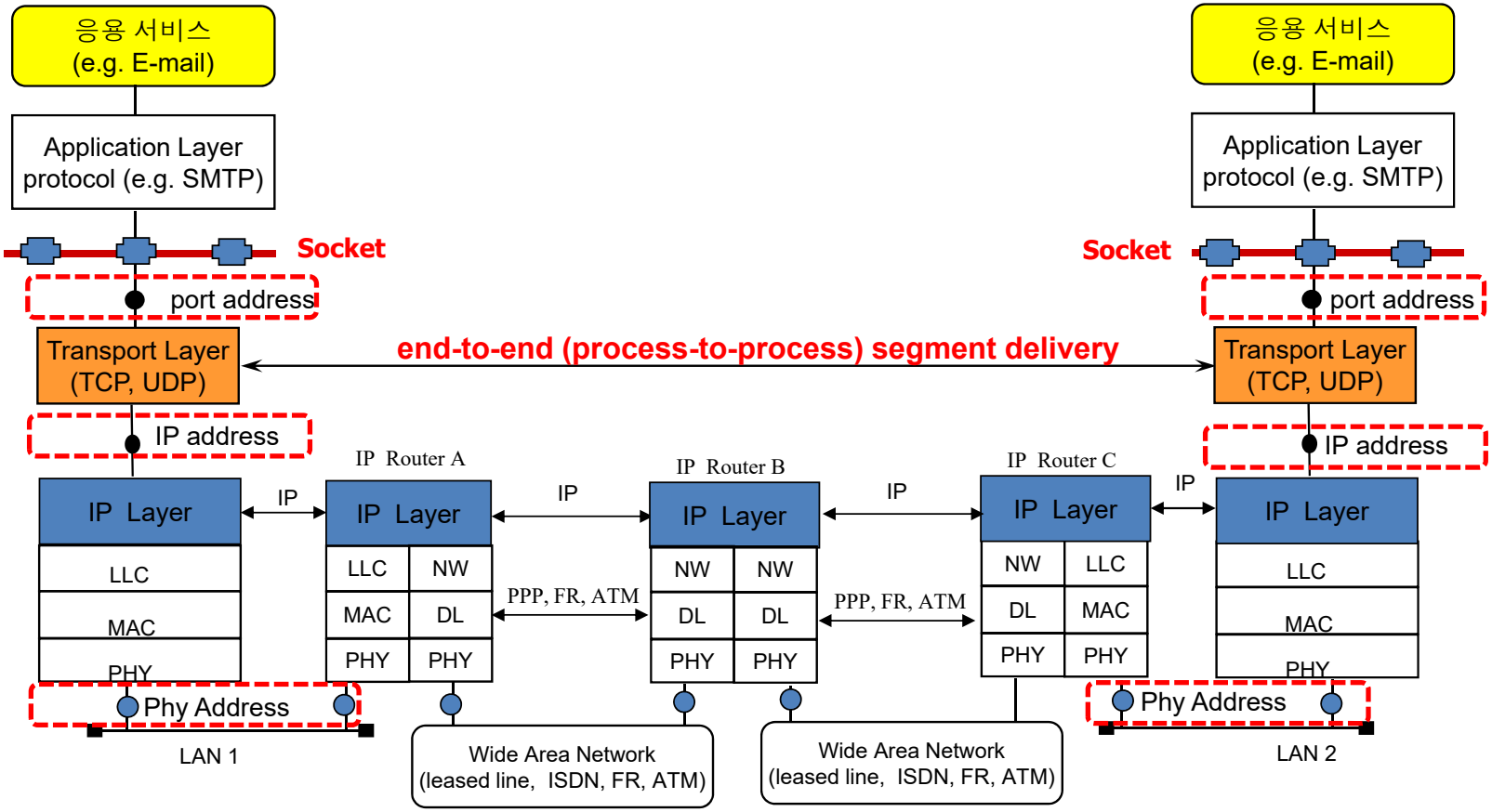
◆ 연결성 서비스

- 연결성 서비스는 데이터를 전달하기 전에 미리 데이터 전송 통로를 먼저 설정 (예: 전화 통화)
- TCP 통신 프로토콜은 양방향 연결을 설정한 후, 데이터를 전달

◆ 양방향 전이중 (full duplex) 신뢰성 통신

- TCP는 양방향 전이중 (full-duplex) 통신 기능을 제공하며, 동시에 양방향 송신 및 수신 기능을 제공
- TCP는 수신 데이터 (세그멘트)에 대하여 응답을 보내며, 이를 기반으로 흐름제어 및 오류제어 기능 제공하여, 신뢰성이 있는 서비스 제공
- TCP 세그멘트가 전송될 때, 상대방으로 부터 수신된 세그멘트의 번호를 함께 *piggybacking* 전달하여 흐름제어 및 오류제어 기능 제공

인터넷 통신 구조


응용 서비스
(e.g. E-mail)
Application Layer
protocol (e.g. SMTP)
Socket
port address
Transport Layer
(TCP, UDP)
IP address
end-to-end (process-to-process) segment delivery
IP Router A
IP Router B
IP Router C
IP Layer
IP
LLC
MAC
PHY
NW
DL
Phy Address
PPP, FR, ATM
LAN 1
LAN 2
Wide Area Network
(leased line, ISDN, FR, ATM)

파이썬 소켓 통신

Python Socket Module

◆ Python socket module (1)

- https://docs.python.org/3/library/socket.html

구분	socket method	Description
server/ client	socket()	socket 객체를 생성; 전달되는 인자로 Address Family(AF), Socket Type (SOCK_STREAM, SOCK_DGRAM)
	socket.setsockopt()	setsockopt(level, optname, value) 지정된 socket option의 값을 설정 level은 option이 지정된 수준을 나타내며, SOL_SOCKET로 설정
	socket.close()	socket을 종료
server	socket.bind(addr)	생성된 socket에 고유한 host와 port를 매핑시켜 인터넷에 고유한 네트워크 접속자원 (IP 주소와 port 번호)에 매핑시켜 프로그램 인터페이스와 네트워크 자원을 연결시킴; 인자로 전달되는 addr은 host와 port의 튜플
	socket.listen()	client로 부터의 연결 설정 요청 대기
	socket.accept()	client로 부터의 연결 설정 요청에 대한 승낙 및 실제 사용될 소켓을 반환

Python Socket Module

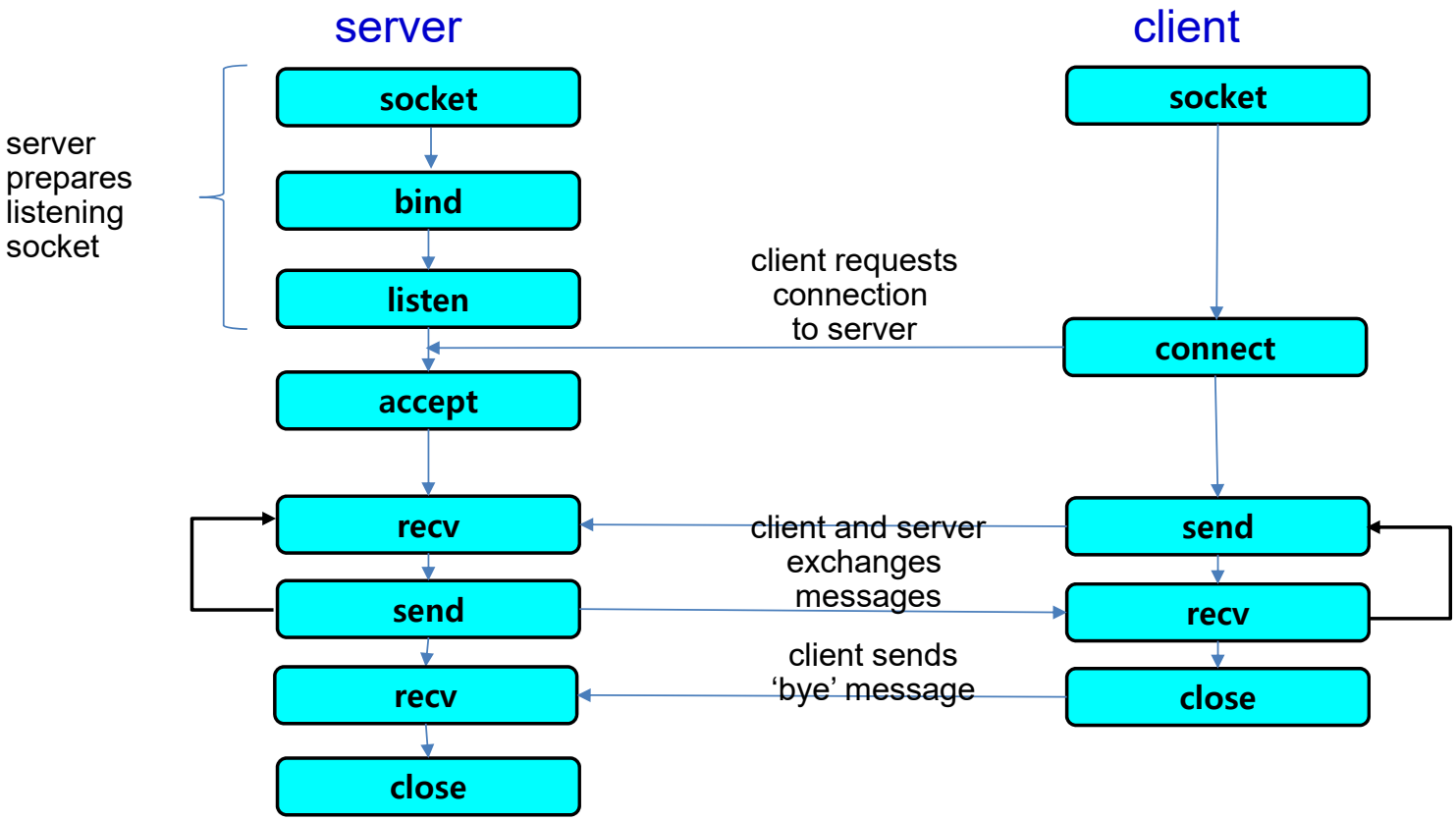
◆ Python socket module (2)

- https://docs.python.org/3/library/socket.html

구분	socket method	Description
client	socket.connect(addr)	client에서 server로 연결 요청; addr은 연결할 socket의 호스트(주소)와 port 정보의 튜플
server/client	socket.send(bytes)	server/client에서 상대방으로 데이터 송신; 인수는 bytes
	socket.sendall(bytes)	server/client에서 상대방으로 데이터 송신; 인수는 bytes
	socket.recv(bufsize)	socket을 사용하여 데이터를 수신; bufsize는 한번에 수신할 수 있는 최대 데이터 크기
	socket.sendto(bytes)	UDP 소켓 (SOCK_DGRAM)을 사용하여 데이터 송신
	socket.recvfrom(bufsize)	UDP 소켓 (SOCK_DGRAM)을 사용하여 데이터 수신

TCP Stream Socket의 연결 및 데이터 통신

◆ TCP Stream Socket 연결 및 데이터 통신 절차



```
# Python Socket Communication - Threaded TCP Streaming Server (1)
import socket
from _thread import *
import time
import threading

LOCK = threading.Lock()

def thread_Rx(client_socket, addr):
    # repeats until the client disconnects
    while True:
        try:
            # when message is received from client, just echo the received message
            rx_msg = client_socket.recv(1024)
            if not rx_msg:
                print('Server:: disconnected by client ({}:{})\n'.format(addr[0], addr[1]))
                break
            print('Rx_msg = {}'.format(repr(rx_msg.decode())))
        except ConnectionResetError as e:
            print('Server:: disconnected by client ({}:{})'.format(addr[0],addr[1]))
            break
        time.sleep(1) # for thread_switching

def thread_Tx(client_socket, addr):
    # repeats until the client disconnects
    while True:
        tx_msg = input("Tx_msg = ")
        try:
            client_socket.send(tx_msg.encode())
        except Exception as e:
            print("Server:: exception in thread_Tx - {}".format(e))
            break
        time.sleep(1) # for thread_switching
```

```
# Python Socket Communication - Threaded TCP Streaming Server (2)

HOST = '127.0.0.1'
PORT = 8089

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((HOST, PORT))
server_socket.listen()

LOCK.acquire()
print('Server:: TCP streaming server is started now')
LOCK.release()

# when a client requests connection, the server accepts and returns a new client socket
# communication is provided using the new client socket
while True:
    LOCK.acquire()
    print('Server:: waiting connection request from next client ....\n')
    LOCK.release()
    client_socket, addr = server_socket.accept()
    LOCK.acquire()
    print('Server:: connected to a client ({}:{})'.format(addr[0], addr[1]))
    print('Server:: client_socket = ', client_socket)
    LOCK.release()
    start_new_thread(thread_Rx, (client_socket, addr))
    start_new_thread(thread_Tx, (client_socket, addr))

server_socket.close()
```

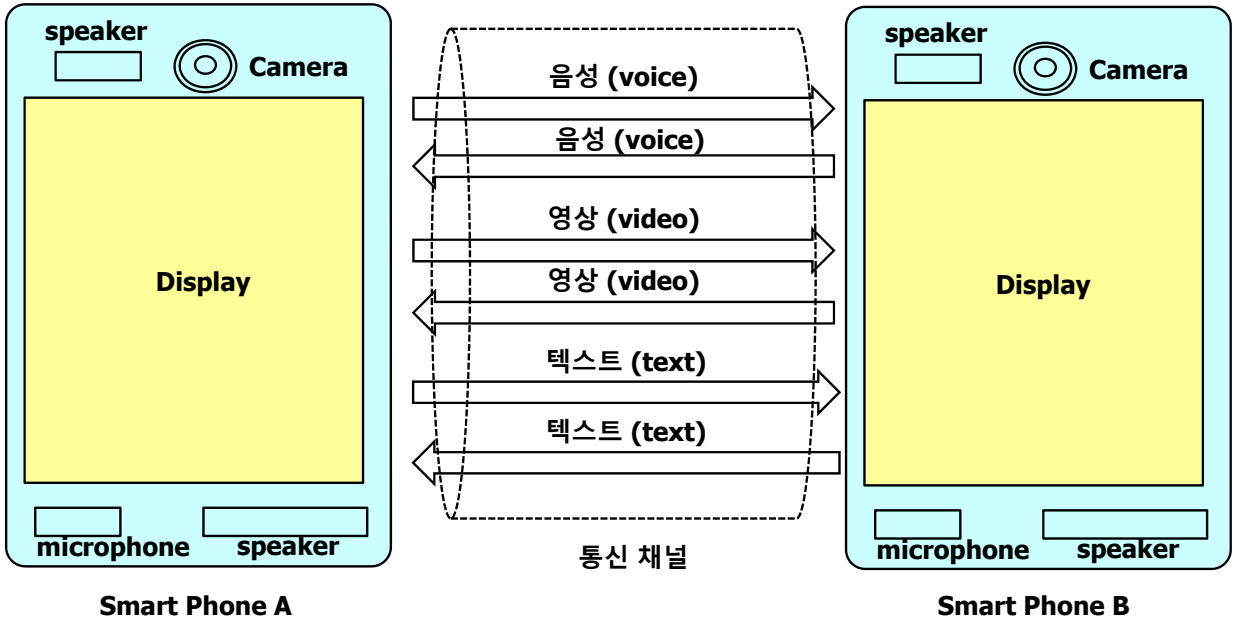
```
# Python Socket Communication - Threaded TCP Streaming Client

import socket
import time
import threading

HOST = "127.0.0.1"
PORT = 8089
serv_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serv_sock.connect((HOST, PORT))

def thread_Tx():
    while True:
        tx_msg = input("Tx_msg = ")
        tx_data = bytes(tx_msg, "utf-8")
        try:
            serv_sock.send(tx_data)
        except Exception as e:
            print("Client:: exception in thread_Tx - {}".format(e))
            serv_sock.close()
            break
        time.sleep(1) # for thread_switching

def thread_Rx():
    while True:
        try:
            rx_data = serv_sock.recv(1024)
        except Exception as e:
            print("Client:: exception in thread_Rx - {}".format(e))
            serv_sock.close()
            break
        rx_msg = repr(rx_data.decode())
        print("Rx_msg = ", rx_msg)
        time.sleep(1) # for thread_switching

threading._start_new_thread(thread_Tx,())
threading._start_new_thread(thread_Rx,())

while True:
    pass
```

파이썬 다중 스레드와 소켓 기반
양방향 전이중 텍스트 채팅

스마트폰의 양방향 전이중 멀티미디어 통신 (Bidirectional Full Duplex Communications)

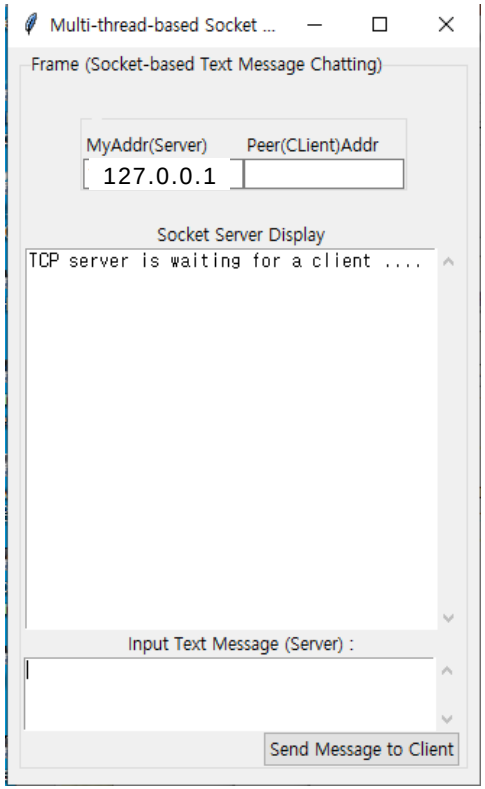
◆ 양방향 전이중 멀티미디어 통신

- 음성 및 영상의 양방향 전이중 (bidirectional full-duplex) 통신
- 음성/영상/텍스트/데이터를 동시에 송신 및 수신



Multi-thread, GUI, Socket **Server** Role (1)

```
import socket, sys, threading
from threading import Thread # for testing multi-thread
from time import  sleep #for sleep in thread
import tkinter as tk
from tkinter import ttk, scrolledtext, END
LocalHost = "127.0.0.1"
SocketChat_PortNumber = 24000

class SocketChatting:
    def __init__(self, mode):
        global hostAddr
        # Create instance
        self.win = tk.Tk()
        self.mode = mode # server or client

        self.win.title("Multi-thread-based Socket Chatting (TCP Server)")
        hostname = socket.gethostname()
        hostAddr = socket.gethostbyname(hostname)
        print("My (server) IP address = {}".format(hostAddr))
        self.myAddr = hostAddr
        self.createWidgets()

        # Start TCP/IP server in its own thread
        serv_thread = Thread(target=self.TCPServer, daemon=True)
        serv_thread.start()
```

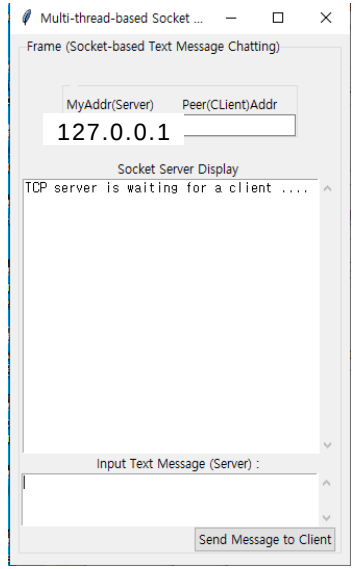


Multi-thread, GUI, Socket Server Role (2)

```
    # TCP server
    def TCPServer(self):
        self.servSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.servSock.bind((hostAddr, SocketChat_PortNumber))
          # bind socket to (IP_addr(local_host), port_number)
        self.scr_servDisplay.insert(tk.INSERT,"TCP server is waiting for a client .... \n" )
        self.servSock.listen(1)
        self.conn, self.cliAddr = self.servSock.accept() # cliAddr : (IPaddr, port_no)
        print("TCP Server is connected to client ({})\n".format(self.cliAddr))
        self.scr_servDisplay.insert(tk.INSERT,"TCP server is connected to client\n" )
        self.scr_servDisplay.insert(tk.INSERT, "TCP client IP address : {}\n".format(self.cliAddr[0]))
        self.peerAddr_entry.insert(END, self.cliAddr[0])
        while True:
          servRecvMsg = self.conn.recv(512).decode()
          if not servRecvMsg:
              break
          self.scr_servDisplay.insert(tk.INSERT,">> " + servRecvMsg)
        self.conn.close()

    #Exit GUI cleanly; definition of quit()
    def _quit(self):
        self.win.quit()
        self.win.destroy()
        exit()

    def serv_send(self): # from server send message to client
        msgToCli = str(self.scr_servInput.get(1.0, END))
        self.scr_servDisplay.insert(tk.INSERT,"<< " + msgToCli)
        self.conn.send(bytes(msgToCli.encode()))
        self.scr_servInput.delete('1.0', END) #clear scr_msgInput scrolltext
```

Multi-thread, GUI, Socket Server Role (3)

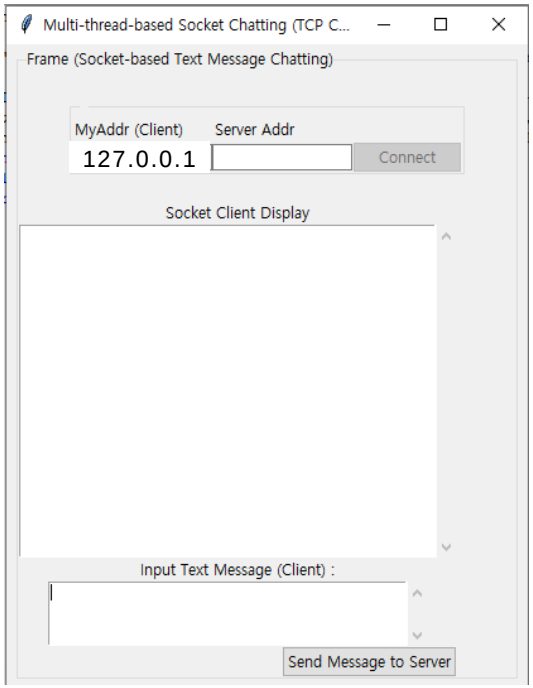
```
    def createWidgets(self):
        # Add a frame in self.win
        frame = ttk.LabelFrame(self.win, text="Frame (Socket-based Text Message Chatting)")
        frame.grid(column=0, row=0, padx=8, pady=4)

        #Add a LabelFrame of myAddr, peerAddr, Connect Button in frame
        frame_addr_connect = ttk.LabelFrame(frame, text="")
        frame_addr_connect.grid(column=0, row=0, padx=40, pady=20, columnspan=2)

        # Add labels (myAddr, peerAddr) in the frame_addr_connect
        myAddr_label = ttk.Label(frame_addr_connect, text="MyAddr(Server)")
        myAddr_label.grid(column=0, row=0, sticky='W') #
        peerAddr_label = ttk.Label(frame_addr_connect, text="Peer(CLient)Addr")
        peerAddr_label.grid(column=1, row=0, sticky='W') #

        # Add a Textbox Entry widgets (myAddr, peerAddr) in the frame_addr_connect
        self.myAddr = tk.StringVar()
        self.myAddr_entry = ttk.Entry(frame_addr_connect, width=15,\
            textvariable=self.myAddr)
        self.myAddr_entry.insert(END, hostAddr)
        self.myAddr_entry.grid(column=0, row=1, sticky='W')

        self.peerAddr = tk.StringVar()
        self.peerAddr_entry = ttk.Entry(frame_addr_connect, width=15, textvariable="")
        #self.peerAddr_entry.insert(END, LocalHost)
        self.peerAddr_entry.grid(column=1, row=1, sticky='W')
```



```
# Multi-thread, GUI, Socket Server Role (4)

        # Add ScrolledText fields of display and input
        scrol_w,  scrol_h = 40, 20
        servDisplay_label = ttk.Label(frame, text="Socket Server Display")
        servDisplay_label.grid(column=0, row=1 )
        self.scr_servDisplay = scrolledtext.ScrolledText(frame, width=scrol_w, height=scrol_h, wrap=tk.WORD)
        self.scr_servDisplay.grid(column=0, row=2, sticky='E') #, columnspan=3

        servInput_label = ttk.Label(frame, text="Input Text Message (Server) :")
        servInput_label.grid(column=0, row=3 )

        self.scr_servInput = scrolledtext.ScrolledText(frame, width=40, height=3, wrap=tk.WORD)
        self.scr_servInput.grid(column=0, row=4) #, columnspan=3

        # Add Buttons (cli_send, serv_send)
        serv_send_button = ttk.Button(frame, text="Send Message to Client", command=self.serv_send)
        serv_send_button.grid(column=0, row=5, sticky='E')

        #Place cursor into the message input scrolled text
        self.scr_servInput.focus()


#======================
# Start GUI
#======================
print("Running TCP server")
sockChat = SocketChatting("server")
sockChat.win.mainloop()
```

Multi-thread, GUI, Socket **Client** Role (1)

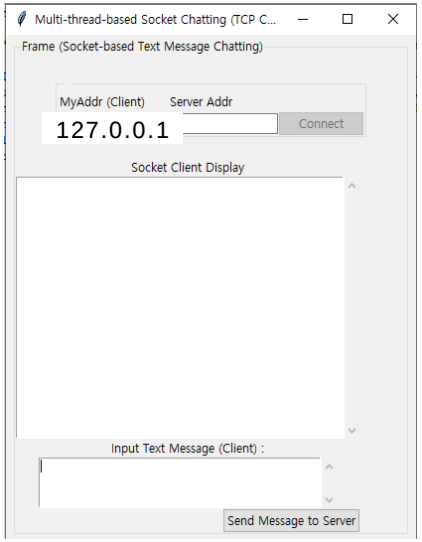
```
import socket, sys, threading
from threading import Thread # for testing multi-thread
from time import  sleep #for sleep in thread
import tkinter as tk
from tkinter import ttk, scrolledtext, END
LocalHost = "127.0.0.1"
SocketChat_PortNumber = 24000

class SocketChatting:
    def __init__(self, mode):
        global hostAddr
        # Create instance
        self.win = tk.Tk()
        self.mode = mode # server or client

        # Add a title
        self.win.title("Multi-thread-based Socket Chatting (TCP Client)")
        hostname = socket.gethostname()
        hostAddr = socket.gethostbyname(hostname)
        print("My (client) IP address = {}".format(hostAddr))
        self.myAddr = hostAddr
        self.createWidgets()

        cli_thread = Thread(target=self.TCPClient, daemon=True)
        cli_thread.start()
```



```
# Multi-thread, GUI, Socket Client Role (2)

    # TCP client
    def TCPClient(self):
        self.cliSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        servAddr_str = input("Server IP Addr (e.g., '127.0.0.1') = ")
        self.cliSock.connect((servAddr_str, SocketChat_PortNumber))
            # send connect request to TCP server
        servAddr = self.cliSock.getpeername()
        print("TCP Client is connected to server ({})\n".format(servAddr))
        self.scr_cliDisplay.insert(tk.INSERT,"TCP client is connected to server \n")
        self.scr_cliDisplay.insert(tk.INSERT,"TCP server IP address : {}\n".format(servAddr[0]) )
        self.servAddr_entry.insert(END, servAddr[0])
        while True:
            cliRecvMsg = self.cliSock.recv(8192).decode()
            if not cliRecvMsg:
                break
            self.scr_cliDisplay.insert(tk.INSERT,">> " + cliRecvMsg)
        self.cliSock.close()

    #Exit GUI cleanly; definition of quit()
    def _quit(self):
        self.win.quit()
        self.win.destroy()
        exit()
```

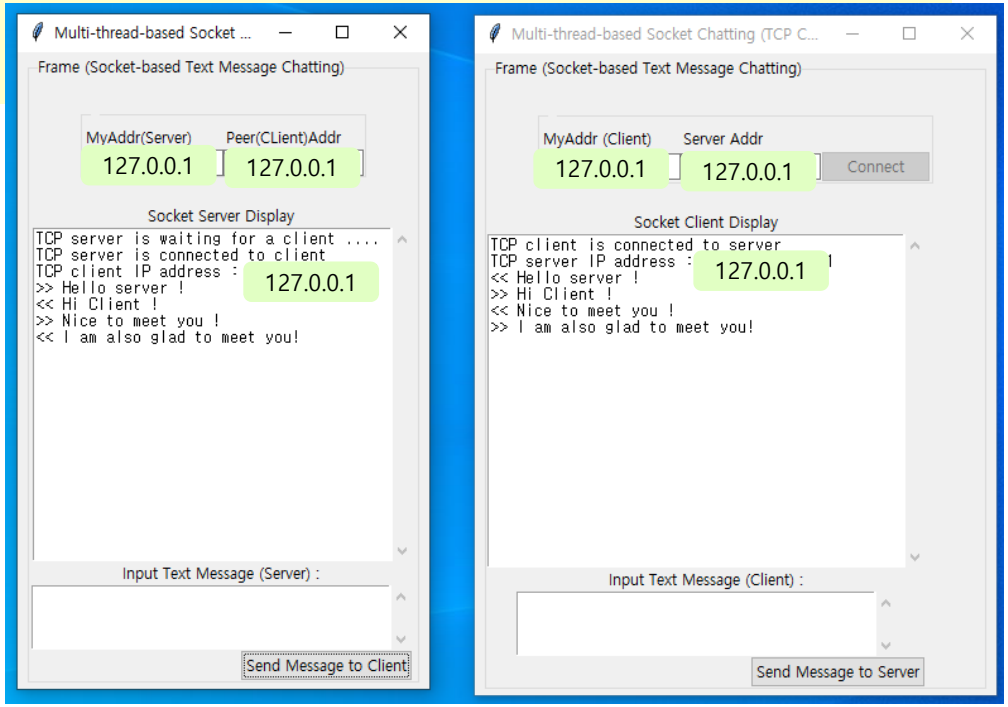
```
# Multi-thread, GUI, Socket Client Role (3)

    # Modified Button Click Function
    def connect_server(self):
        self.scr_cliDisplay.insert(tk.INSERT,"Connecting to server ....")
        self.myIpAddr = self.myAddr.get()
        self.peerIpAddr = self.peerAddr.get()
        self.scr_cliDisplay.insert(tk.INSERT, "My IP Address : " + self.myIpAddr + '\n')
        self.scr_cliDisplay.insert(tk.INSERT, "Server's IP Address : " + self.peerIpAddr + '\n')
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((self.peerIpAddr, SocketChat_PortNumber))
            # send connect request to TCP server

    #define callback for myMsg_enter()
    def cli_send(self): #from mySelf to peer/server
        msgToServer = str(self.scr_cliInput.get(1.0, END))
        self.scr_cliDisplay.insert(tk.INSERT,"<< " + msgToServer)
        self.cliSock.send(bytes(msgToServer.encode()))
        self.scr_cliInput.delete('1.0', END) #clear scr_msgInput scrolltext

    def createWidgets(self):
        # Add a frame in self.win
        frame = ttk.LabelFrame(self.win, text="Frame (Socket-based Text Message Chatting)")
        frame.grid(column=0, row=0, padx=8, pady=4)

        #Add a LabelFrame of myAddr, peerAddr, Connect Button in frame
        frame_addr_connect = ttk.LabelFrame(frame, text="")
        frame_addr_connect.grid(column=0, row=0, padx=40, pady=20, columnspan=2)

        # Add labels (myAddr, peerAddr) in the frame_addr_connect
        myAddr_label = ttk.Label(frame_addr_connect, text="MyAddr (Client)")
        myAddr_label.grid(column=0, row=0, sticky='W') #
        peerAddr_label = ttk.Label(frame_addr_connect, text="Server Addr")
        peerAddr_label.grid(column=1, row=0, sticky='W') #
```

```
# Multi-thread, GUI, Socket Client Role (4)

        # Add a Textbox Entry widgets (myAddr, peerAddr) in the frame_addr_connect
        self.myAddr = tk.StringVar()
        self.myAddr_entry = ttk.Entry(frame_addr_connect, width=15, textvariable=self.myAddr)
        self.myAddr_entry.insert(END, hostAddr)
        self.myAddr_entry.grid(column=0, row=1, sticky='W')

        self.servAddr = tk.StringVar()
        self.servAddr_entry = ttk.Entry(frame_addr_connect, width=15, textvariable="")
        #self.servAddr_entry.insert(END, LocalHost)
        self.servAddr_entry.grid(column=1, row=1, sticky='W')

        # Add a connect_button
        connect_button = ttk.Button(frame_addr_connect, text="Connect",\
            command=self.connect_server)
        connect_button.grid(column=3, row=1)
        connect_button.configure(state='disabled') # for local

        # Add ScrolledText fields of display and input
        cliDisplay_label = ttk.Label(frame, text="Socket Client Display")
        cliDisplay_label.grid(column=0, row=1 )
        scrol_w,  scrol_h = 40, 20
        self.scr_cliDisplay = scrolledtext.ScrolledText(frame, width=scrol_w,\
            height=scrol_h, wrap=tk.WORD)
        self.scr_cliDisplay.grid(column=0, row=2, sticky='WE') #, columnspan=3

        cliInput_label = ttk.Label(frame, text="Input Text Message (Client) :")
        cliInput_label.grid(column=0, row=3 )

        self.scr_cliInput = scrolledtext.ScrolledText(frame, width=40, height=3, wrap=tk.WORD)
        self.scr_cliInput.grid(column=0, row=4) #, columnspan=3
```



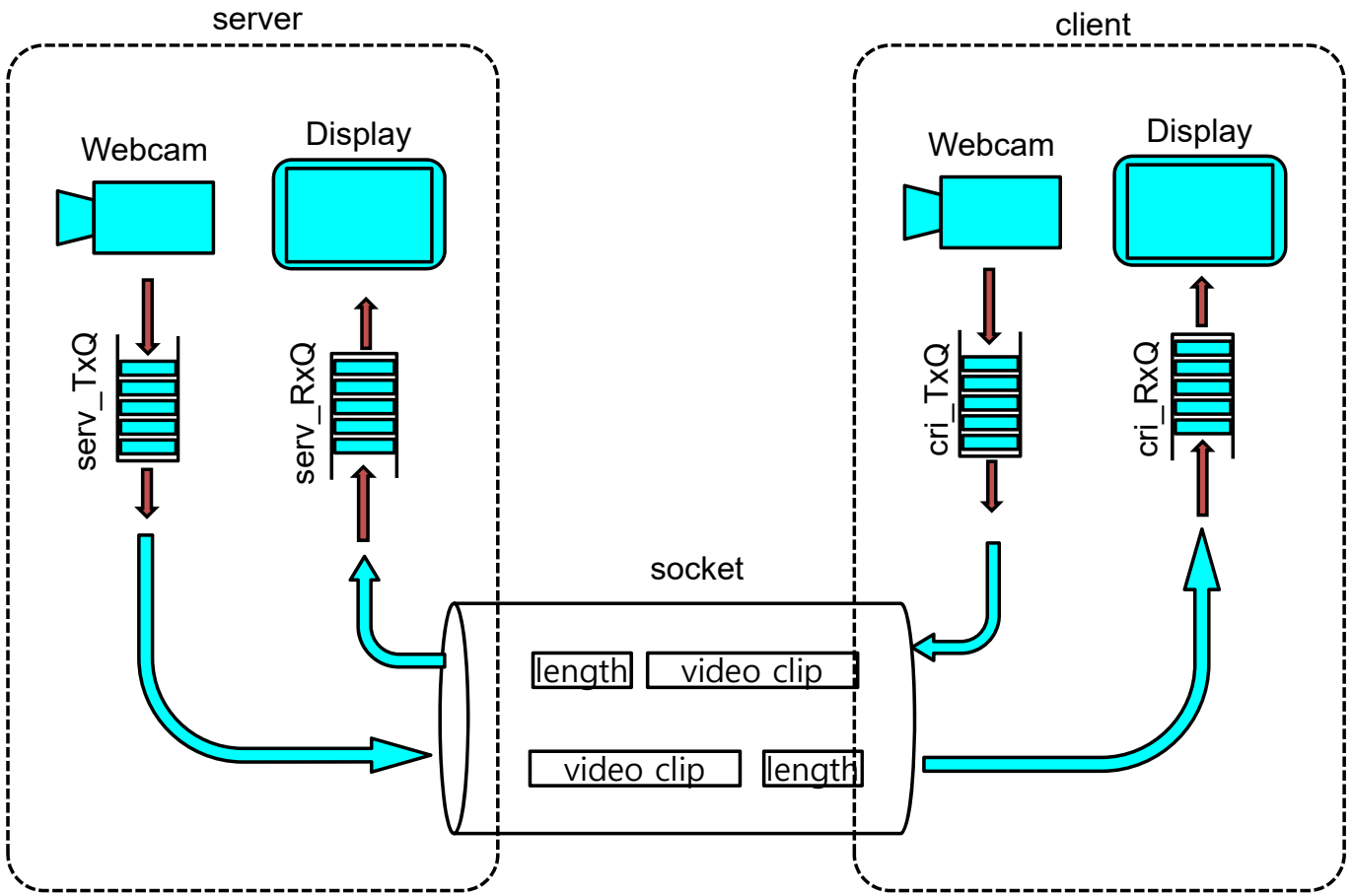
Multi-thread, GUI, Socket Client Role (5)

```
        # Add Buttons (cli_send, serv_send)
        cli_send_button = ttk.Button(frame, text="Send Message to Server", command=self.cli_send)
        cli_send_button.grid(column=0, row=5, sticky='E')

        #Place cursor into the message input scrolled text
        self.scr_cliInput.focus()

#======================
# Start GUI
#======================
print("Running TCP Client")
sockChat = SocketChatting('client')
sockChat.win.mainloop()
```



Python OpenCV Module 기반
양방향 화상 통신

Full-duplex Video Chatting



OpenCV

◆ OpenCV

- 실시간 컴퓨터 비전을 목적으로 한 프로그래밍 라이브러리이며, 원래는 인텔이 개발
- **실시간** 이미지 프로세싱에 중점을 둔 라이브러리
- C/C++ 프로그래밍 언어로 개발 되었으며 파이썬 , 자바 및 매트랩 / OCTAVE에 바인딩 되어 프로그래머에게 개발 환경을 지원

◆ OpenCV의 주요 알고리즘

- 이진화(binarization), 노이즈 제거, 외곽선 검출(edge detection)
- 패턴인식, 기계학습(machine learning)
- ROI(Region Of Interest) 설정
- 이미지 변환(image warping)
- 하드웨어 가속

OpenCV

◆ OpenCV (ver. 4.4.0.46) 설치

- python –m pip install --upgrade opencv-python

```
C:₩Users₩Owner>python -m pip install --upgrade opencv-python
Collecting opencv-python
  Downloading opencv_python-4.4.0.46-cp38-cp38-win32.whl (24.6 MB)
     |████████████████████████████████| 24.6 MB 6.4 MB/s
Requirement already satisfied, skipping upgrade: numpy>=1.17.3 in c:₩users₩owner₩appdata₩local₩programs₩python₩python38-
32₩lib₩site-packages (from opencv-python) (1.18.1)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.4.0.46
```

Webcam - server (1)

```
# Video-Chatting Server (1)
import socket
import cv2
import numpy
from queue import Queue
from _thread import *
SERVER_WEBCAM = 0

def recvall(sock, count):
    buf = b''
    while count:
        newbuf = sock.recv(count)
        if not newbuf: return None
        buf += newbuf
        count -= len(newbuf)
    return buf

def video_sendto_client(client_socket, addr, queue):
    print('Server::connected to ({} : {})'.format(addr[0], addr[1]))
    while True:
        try:
            if not queue.empty ():
                stringData = queue.get()
                client_socket.send(str(len(stringData)).ljust(16).encode())
                client_socket.send(stringData)
        except ConnectionResetError as e:
            break
    client_socket.close()
```

```
# Video-Chatting Server (2)

def video_chat_server(queue):
    server_webcam = cv2.VideoCapture(SERVER_WEBCAM)
    while True:
        ret, serv_frame = server_webcam.read()
        if ret == False:
            continue
        encode_param=[int(cv2.IMWRITE_JPEG_QUALITY),90]
        result, imgencode = cv2.imencode('.jpg', serv_frame, encode_param)
        img_data = numpy.array(imgencode)
        stringData = img_data.tobytes()
        queue.put(stringData)
        #cv2.imshow('Server:: Server_Video', serv_frame)
        key = cv2.waitKey(1)
        if key == 27: # if ESC key is input, then exit
            break

def video_recvfrom_client (client_socket):
    while True:
        length = recvall (client_socket, 16)
        stringData = recvall(client_socket, int(length))
        data = numpy.frombuffer(stringData, dtype='uint8')
        decimg=cv2.imdecode(data,1)
        cv2.imshow('Server:: Received from Client',decimg)
        key = cv2.waitKey(1)
        if key == 27: # if ESC key is input, then exit
            break
```

Video-Chatting Server (3)

```
if __name__ == "__main__":
    enclosure_queue = Queue()
    serverAddr = '127.0.0.1'
    PORT = 9999
    hostname = socket.gethostname()
    serverAddr = socket.gethostbyname(hostname)
    print("Server IP address = {}".format(serverAddr))
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    #server_socket.bind((HOST, PORT))
    server_socket.bind((serverAddr, PORT))
    server_socket.listen()
    print('Server::Video chatting server started')
    start_new_thread(video_chat_server, (enclosure_queue,))
    print('Server::Waitint for client .... ')
    client_socket, addr = server_socket.accept()
    print('Server::connected to ({} : {})'.format(client_socket, addr))
    start_new_thread(video_sendto_client, (client_socket, addr, enclosure_queue,))
    start_new_thread(video_recvfrom_client, (client_socket,))
    server_socket.close()
```

Webcam - Client (1)

```
# Video-chatting Client (1)
import socket
import numpy as np
import cv2
from queue import Queue
from _thread import *

CLIENT_WEBCAM = 1   # when two webcams are used 0, 1
def recvall(sock, count):
    buf = b''
    while count:
        newbuf = sock.recv(count)
        if not newbuf: return None
        buf += newbuf
        count -= len(newbuf)
    return buf

def video_sendto_server(client_socket, queue):
    while True:
        try:
            if not queue.empty ():
                stringData = queue.get()
                client_socket.send(str(len(stringData)).ljust(16).encode())
                client_socket.send(stringData)
        except ConnectionResetError as e:
            break
    client_socket.close()
```

```
# Video-chatting Client (2)

def video_chat_client(queue):
    client_webcam = cv2.VideoCapture(CLIENT_WEBCAM)
    while True:
        ret, frame = client_webcam.read()
        if ret == False:
            continue
        encode_param=[int(cv2.IMWRITE_JPEG_QUALITY),90]
        result, imgencode = cv2.imencode('.jpg', frame, encode_param)
        img_data = np.array(imgencode)
        stringData = img_data.tobytes()
        queue.put(stringData)
        #cv2.imshow('Client:: Client_Video', frame)
        key = cv2.waitKey(1)
        if key == 27: # if ESC key is input, then exit
            break
```

```
# Video-chatting Client (3)

if __name__ == "__main__":
    serverAddr = '127.0.0.1'     # default local host address
    PORT = 9999
    enclosure_queue = Queue()
    serverAddr = input("Input server IP address = ")
    print('Client::Connecting to Server')
    client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    client_socket.connect((serverAddr, PORT))
    print('Client::Connected to Server({}:{})'.format(serverAddr, PORT))
    start_new_thread(video_chat_client, (enclosure_queue,))
    start_new_thread(video_sendto_server, (client_socket, enclosure_queue,))
    #start_new_thread(video_recvfrom_server, (client_socket,))
    while True:
        length = recvall (client_socket, 16)
        stringData = recvall(client_socket, int(length))
        data = np.frombuffer(stringData, dtype='uint8')
        decimg=cv2.imdecode(data,1)
        cv2.imshow('Client:: Received from Server',decimg)
        key = cv2.waitKey(1)
        if key == 27: # if ESC key is input, then exit
            break

    client_socket.close()
```

OpenCV-based Video Chatting

◆ Webcam_Server와 Webcam_Client

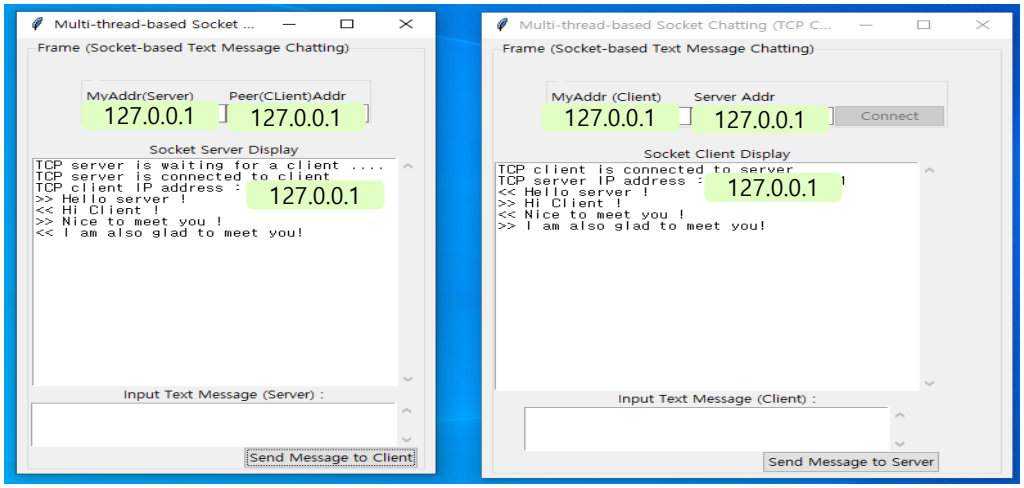
- 서로 다른 Python Shell에서 실행
- 만약 동일한 Python Shell에서 실행하는 경우, server를 실행한 후 client를 실행할 때 이전의 서버 스크립트가 닫히면서 오류가 발생함

Homework 12

Homework 12.1

12.1 파이썬 스레드 기반의 채팅 프로그램 구현

- 파이썬 socket과 threading 모듈을 사용하는 socket 기반 채팅 프로그램을 구현
- TCP/IP 통신 프로토콜을 사용하는 streaming socket (SOCK_STREAM)을 사용하도록 하며, TCP_Server와 TCP_Client 기능을 담당하는 파이썬 프로그램을 구현
- TCP_Server와 TCP_Client는 각자의 IP 주소를 파악하여 GUI 창에 표시
- TCP_Server는 TCP_Client로 부터의 연결 요청을 받을 수 있도록 준비
- TCP_Client는 TCP_Server의 IP 주소를 사용하여 연결 설정 요청
- TCP_Server와 TCP_Client는 수신된 text 메시지 및 프로그램 진행 상황을 출력할 수 있는 출력창을 scrolledtext 모듈의 ScrolledText()로 구현
- TCP_Server와 TCP_Client는 파이썬 tkinter 기반의 GUI를 제공하며, 채팅 문자를 입력받는 text 입력창을 scrolledtext 모듈의 ScrolledText()로 구현
- TCP_Server와 TCP_Client는 입력된 text를 상대편으로 보내기 위한 버튼을 구현



Homework 12.2

12.2 파이썬 멀티스레드, TCP 소켓과 OpenCV를 사용한 양방향 전이중 영상 채팅 구현

- 파이썬 멀티스레드, TCP 소켓과 OpenCV를 사용하여 양방향 전이중 (bidirectional full-duplex) 영상 채팅 프로그램을 구현하라.
- 파이썬 스트리밍 소켓 (SOCK_STREAM)을 사용하도록 하고, Video-Chat 서버와 Video-Chat 클라이언트로 기능을 나누어 구현할 것. 서버는 클라이언트로 부터의 연결 요청을 대기하도록 하고, 클라이언트로 부터 연결 설정 요청이 오면 이를 승인하여 연결을 설정하도록 할 것.
- Video-Chat 서버는 자신의 IP 주소를 확인하여 출력하도록 하고, Video-Chat 클라이언트가 연결 설정 요청을 할 때 사용할 수 있게 할 것.
- Video-Chat 클라이언트는 Video-Chat 서버 주소를 입력하여 스트리밍 소켓을 사용하여 연결을 설정하고, 영상 정보를 양방향으로 전달 할 수 있게 할 것. 포트 번호 (port number)는 9999 를 사용할 것.
- Video-Chat 서버와 Video-Chat 클라이언트는 반복적으로 webcam으로 부터 비디오 이미지를 입력받고, 이를 상대방으로 전송하며, 상대방으로 부터 전송받은 비디오 이미지를 화면으로 출력할 것.