스마트 모빌리티 프로그래밍

Ch 12. 파이썬 기반 다중 스레드 프로그래밍과 인터넷 소켓 통신

영남대학교 정보통신공학과

교수 김 영 탁

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

Outline

- 파이썬에서의 동시처리 (concurrent processing)와 병렬처리 (parallel processing)
- 파이썬 스레드 (thread) 모듈과 멀티프로세스 (multiprocess) 모듈
- 병렬/동시처리에서 공유자원의 임계구역(critical section) 관리를 위한 세마포 (semaphore) Lock()
- 파이썬 기반 인터넷 통신을 위한 Socket 모듈
- 파이썬 멀티스레드 응용
  - Socket 텍스트 통신
  - OpenCV 기반 영상통신

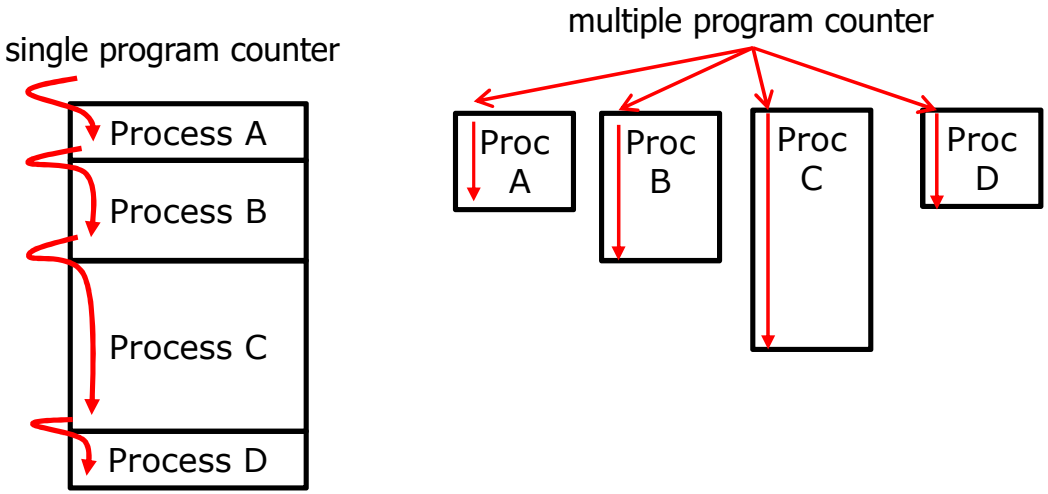
병렬/동시처리, Process/Thread

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)

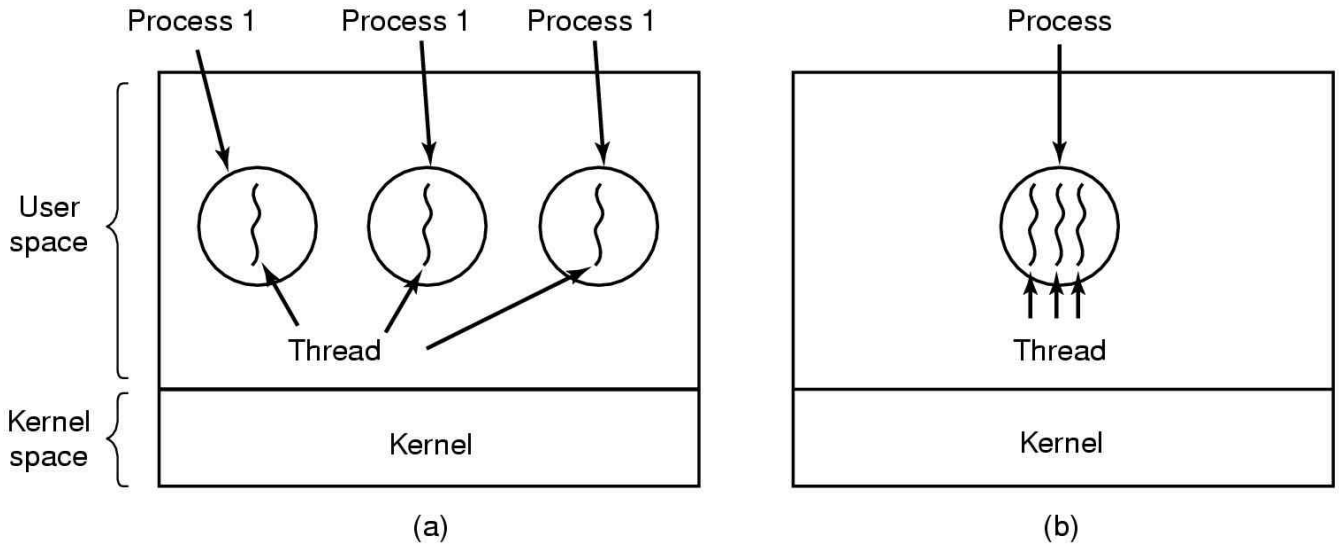
◆ 프로세스 (Process)

- 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)

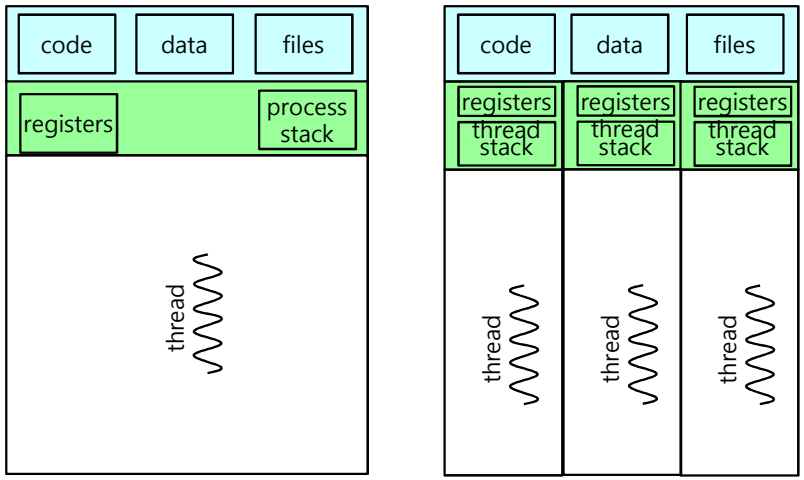
- 스레드는 하나의 프로세스 내부에 포함되는 함수들이 동시에 실행될 수 있게 한 작은 단위 프로세서 (lightweight process)
- 기본적으로 CPU를 사용하는 기본 단위
- 하나의 프로세스에 포함된 다수의 스레드 들은 프로세스의 메모리 자원들 (code section, data section, Heap 등)과 운영체제의 자원들 (예: 파일 입출력 등)을 공유함



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.

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-task A, B, C with Python Thread (1)
import threading
import time

def simple_thread(mark, max_count=10, per_line=50, delay=0.1):
    my_mark = mark
    print("Thread_{}::starts\n".format(my_mark))
    for i in range(max_count):
        for j in range(per_line):
            print(my_mark, end='')
        print()
        time.sleep(delay)
    print("Thread_{}::terminates\n".format(my_mark))

def main():
    print("main():: before creations and starts of threads")
    num_active_threads = threading.active_count()
    active_threads = threading.enumerate()
    print("main():: currently {} threads (including main) are active :".format(num_active_threads))
    print(" ", active_threads)

    thread_A = threading.Thread(target=simple_thread, name='thread_A', args=('A', 10, 50, 0.3), daemon=True)
    thread_A.start()
    thread_B = threading.Thread(target=simple_thread, name='thread_B', args=('B', 20, 50, 0.2), daemon=True)
    thread_B.start()
    thread_C = threading.Thread(target=simple_thread, name='thread_C', args=('C', 30, 50, 0.1), daemon=True)
    thread_C.start()
```

Multi-task A, B, C with Python Thread (2)

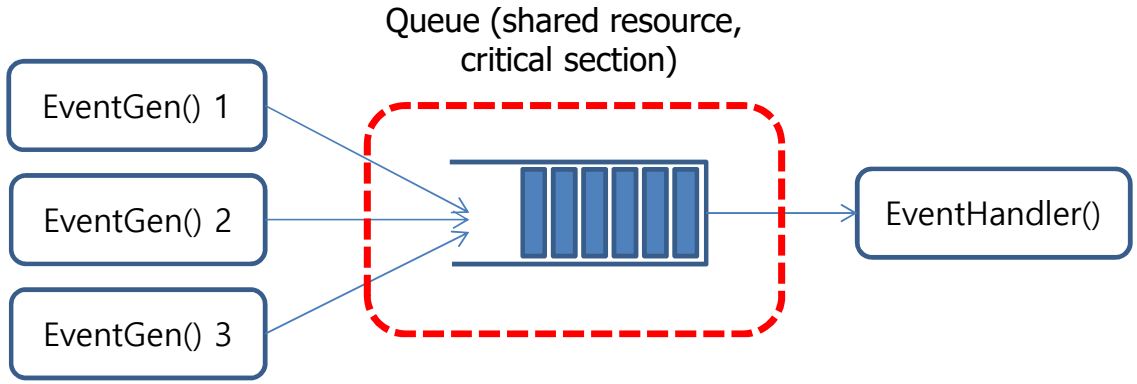
```
    threads = [thread_A, thread_B, thread_C]

    while True:
        all_threads_terminated = True
        for thr in threads:
            if thr.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():: currently {} threads (including main)\
               are active :".format(num_active_threads))
            print(" ", active_threads)
            time.sleep(1)
            continue

    for thrd in threads:
        thrd.join(timeout = 100.0)
    print("main():: all threads are terminated now !!")
    num_active_threads = threading.active_count()
    active_threads = threading.enumerate()
    print("main():: currently {} threads (including main)\
        are active :".format(num_active_threads))
    print(" ", active_threads)

if __name__ == "__main__":
    main()
```

```
main():: before creations and starts of threads
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 17996)>, <Thread(SockThread, started daemon 15984)>]
Thread_B::starts
Thread_A::starts

main():: currently 5 threads (including main) are active :Thread_C::starts



BA C BA[<_MainThread(MainThread, started 17996)>, <Thread(SockThread, started daemon 15984)>, <Thread(thr
ead_A, started daemon 11368)>, <Thread(thread_B, started daemon 17508)>, <Thread(thread_C, started daemon
15624)>]C
BACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBAC
BACBACBACBACBACBACBACBACBACBACBAC


CCCCCCCCCCBCBCBCBCBCBCBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBA
main():: currently 5 threads (including main) are active :CBA
 CBA [<_MainThread(MainThread, started 17996)>, <Thread(SockThread, started daemon 15984)>, <Thread(threa
d_A, started daemon 11368)>, <Thread(thread_B, started daemon 17508)>, <Thread(thread_C, started daemon 1
5624)>]CBA
CBACBACBACBACBACBA
BABABABABABABABABA
AAAAAAA
CCCCCCCCCCBCBCBCBCBCBCBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBAC
BACBACBA

    ▪  ▪  ▪  ▪  ▪


main():: currently 4 threads (including main) are active :
  [<_MainThread(MainThread, started 17996)>, <Thread(SockThread, started daemon 15984)>, <Thread(thread_B
, started daemon 17508)>, <Thread(thread_C, started daemon 15624)>]
CCCCCCCCCCCCCCCCCCCCCCCCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCB
BBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCB
BBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
C
main():: currently 4 threads (including main) are active :
C  C[<_MainThread(MainThread, started 17996)>, <Thread(SockThread, started daemon 15984)>, <Thread(thread
_B, started daemon 17508)>, <Thread(thread_C, started daemon 15624)>]
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CThread_B::terminates

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

main():: currently 3 threads (including main) are active :
  [<_MainThread(MainThread, started 17996)>, <Thread(SockThread, started daemon 15984)>, <Thread(thread_C
, started daemon 15624)>]
Thread_C::terminates

main():: all threads are terminated now !!
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 17996)>, <Thread(SockThread, started daemon 15984)>]
```

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

◆ 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

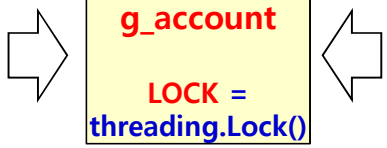
◆threading.Lock()

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.  }
```




```
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.  }
```

Python threads with Semaphore (Lock)

```
# Python Multi-thread  A, B, C with semaphore (1)

import threading
import time

def simple_thread(mark, max_count=10, per_line=50, delay=0.1, semaphore=None):
    my_mark = mark
    semaphore.acquire()
    print("Thread_{}::starts\n".format(my_mark))
    semaphore.release()
    for i in range(max_count):
        semaphore.acquire()
        for j in range(per_line):
            print(my_mark, end='')
        print()
        semaphore.release()
        time.sleep(delay)
    semaphore.acquire()
    print("Thread_{}::terminates\n".format(my_mark))
    semaphore.release()

def main():
    console_lock = threading.Lock()
    print("main():: before creations and starts of threads")
    num_active_threads = threading.active_count()
    active_threads = threading.enumerate()
    print("main():: currently {} threads (including main) are active :"\
      .format(num_active_threads))
    print(" ", active_threads)
```

```
main():: before creations and starts of threads
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>]
Thread_A::starts

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Thread_C::starts

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Thread_B::starts

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

main():: currently 5 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>, <Thread(thread_A, start
ed daemon 13096)>, <Thread(thread_B, started daemon 13832)>, <Thread(thread_C, started daemon 1380)>]
```

```
# Python Multi-thread  A, B, C with semaphore (2)

    thread_A = threading.Thread(target=simple_thread,\
        name='thread_A', args=('A', 10, 50, 0.3, console_lock),\
        daemon = True)
    thread_A.start()
    thread_B = threading.Thread(target=simple_thread,\
        name='thread_B', args=('B', 20, 50, 0.2, console_lock) ),\
        daemon = True))
    thread_B.start()
    thread_C = threading.Thread(target=simple_thread,\
        name='thread_C', args=('C', 30, 50, 0.1, console_lock) ),\
        daemon = True))
    thread_C.start()

    threads = [thread_A, thread_B, thread_C]

    while True:
        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()
            console_lock.acquire()
            print("\nmain():: currently {} threads (including main)\
               are active :".format(num_active_threads))
            print(" ", active_threads)
            console_lock.release()
            time.sleep(1)
            continue
```

```
main():: before creations and starts of threads
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>]
Thread_A::starts

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Thread_C::starts

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Thread_B::starts

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

main():: currently 5 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>, <Thread(thread_A, start
ed daemon 13096)>, <Thread(thread_B, started daemon 13832)>, <Thread(thread_C, started daemon 1380)>]
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

■  ■  ■  ■  ■

main():: currently 4 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>, <Thread(thread_B, start
ed daemon 13832)>, <Thread(thread_C, started daemon 1380)>]
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Thread_C::terminates

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

main():: currently 3 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>, <Thread(thread_B, start
ed daemon 13832)>]
Thread_B::terminates

main():: all threads are terminated now !!
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>]
```

```
# Python Multi-thread  A, B, C with semaphore (3)

    for thrd in threads:
        thrd.join(timeout = 100.0)
    print("main():: all threads are terminated now !!")
    num_active_threads = threading.active_count()
    active_threads = threading.enumerate()
    print("main():: currently {} threads (including main)\
        are active :".format(num_active_threads))
    print(" ", active_threads)

if __name__ == "__main__":
    main()
```

```
main():: before creations and starts of threads
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>]
Thread_A::starts

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Thread_C::starts

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Thread_B::starts

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

main():: currently 5 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>, <Thread(thread_A, start
ed daemon 13096)>, <Thread(thread_B, started daemon 13832)>, <Thread(thread_C, started daemon 1380)>]
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

■ ■ ■ ■ ■

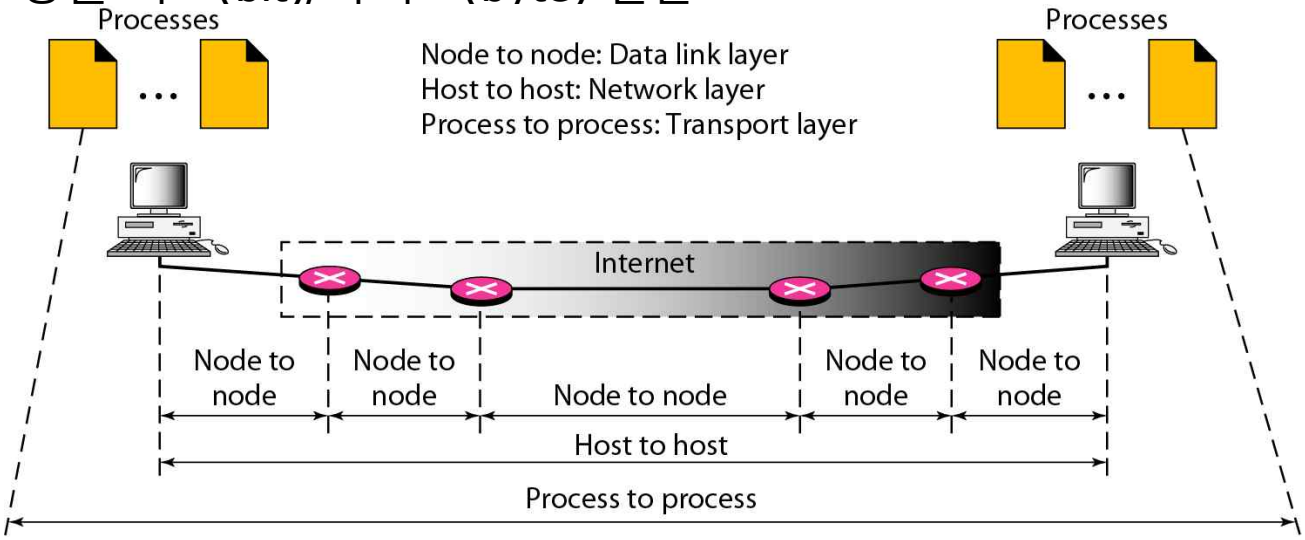
```
main():: currently 4 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>, <Thread(thread_B, start
ed daemon 13832)>, <Thread(thread_C, started daemon 1380)>]
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Thread_C::terminates

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

main():: currently 3 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>, <Thread(thread_B, start
ed daemon 13832)>]
Thread_B::terminates

main():: all threads are terminated now !!
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 17272)>, <Thread(SockThread, started daemon 2840)>]
```

인터넷 통신에서 사용되는 소켓 (socket)

인터넷 통신 구조

◆ 인터넷의 정보 전달을 위한 통신 프로토콜 계층

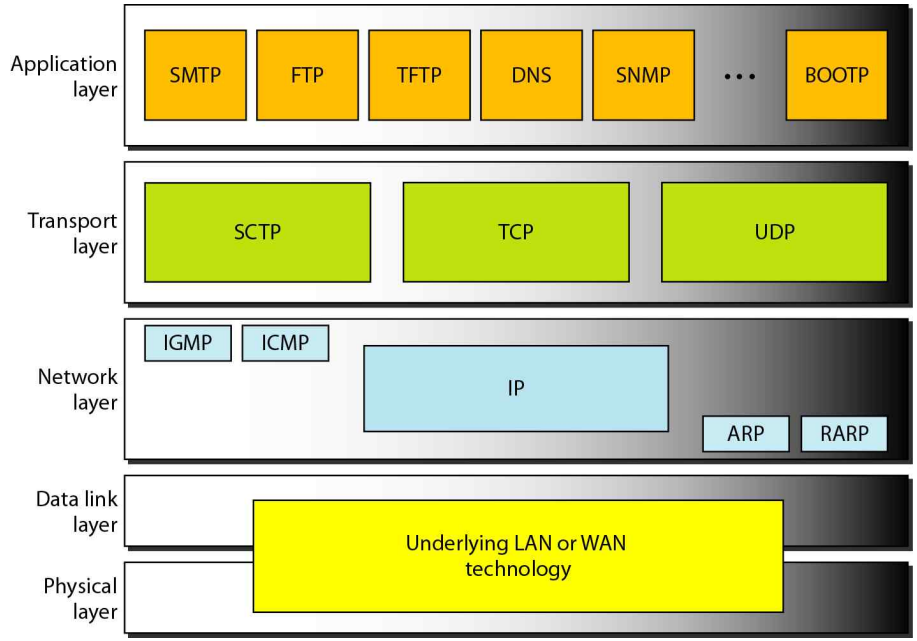
- 응용 계층 (application layer): 응용 서비스 제공을 위한 응용 프로그램/프로세스
- 수송 계층 (transport layer): 응용 데이터 (메시지)의 프로세스 간 전달
- 네트워크 계층: 호스트 컴퓨터간 패킷 (packet/datagram) 전달
- 데이터 링크 계층: 컴퓨터 장치 간의 프레임(frame) 전달
- 물리계층: 광케이블, 구리선 전송 선로, 무선 채널 등을 사용하여 전기신호, 광신호, 전파신호로 인코딩된 비트(bit)/바이트(byte) 전달



인터넷 통신 프로토콜 계층 구조

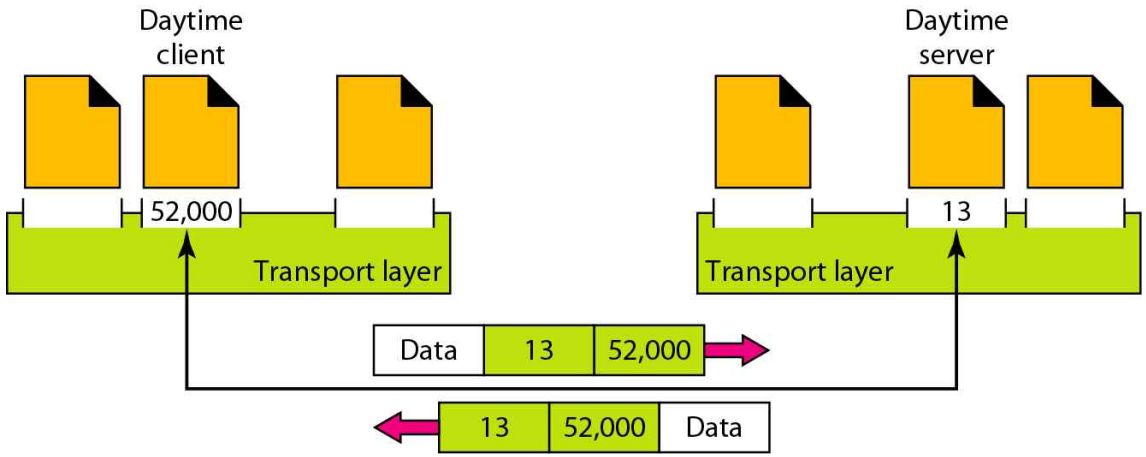
◆ 인터넷 통신 프로토콜 계층 구조

- 응용 계층: SMTP (simple mail transfer protocol), FTP (file transfer protocol)
- 수송 계층 (transport layer): UDP, TCP, SCTP
- IP (internet protocol) 계층
- 데이터 링크 계층 :
  LAN (local area network): Ethernet, WiFi
  WAN (wide area network): LTE, 5G
- 물리 계층: 광케이블, 구리선 전송 선로, 무선 채널 등을 사용하여 전기신호, 광신호, 전파신호 로 인코딩된 비트(bit)/바이트(byte) 전달



인터넷 통신 주소 (address)

- **Addressing**
  - local host
  - local process
  - remote host
  - remote process
- **Port numbers**
  - Well-known port numbers
  - ephemeral port number



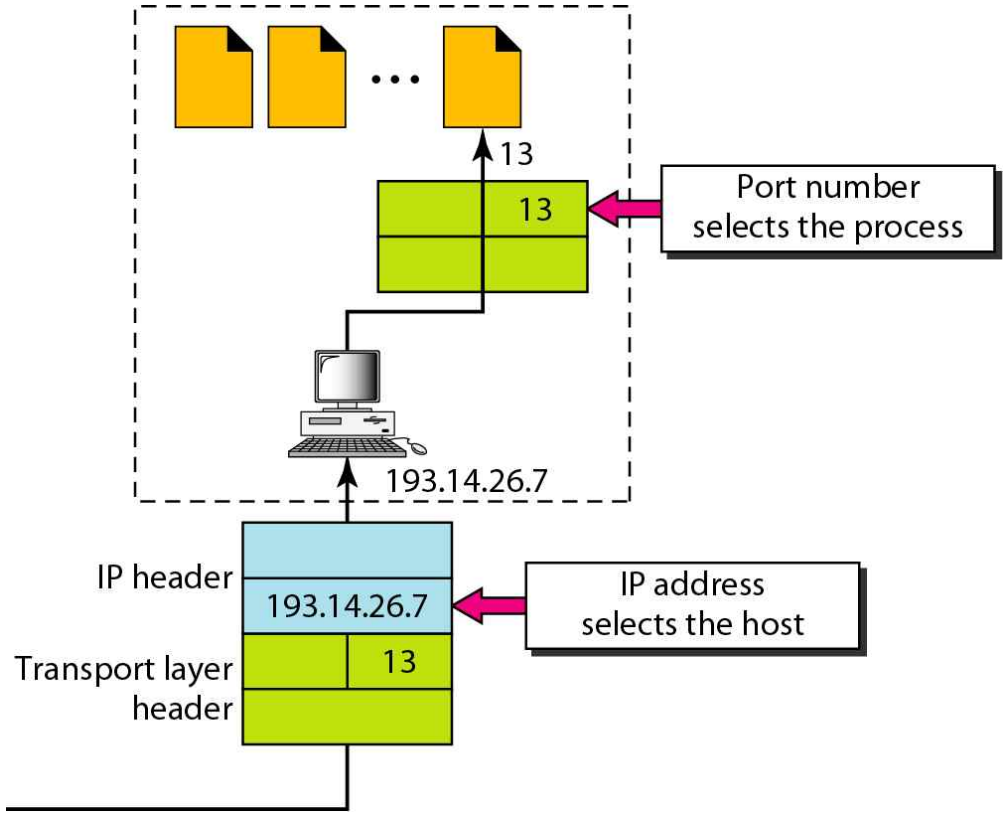
IP 주소와 포트 (port) 번호

◆ IP address

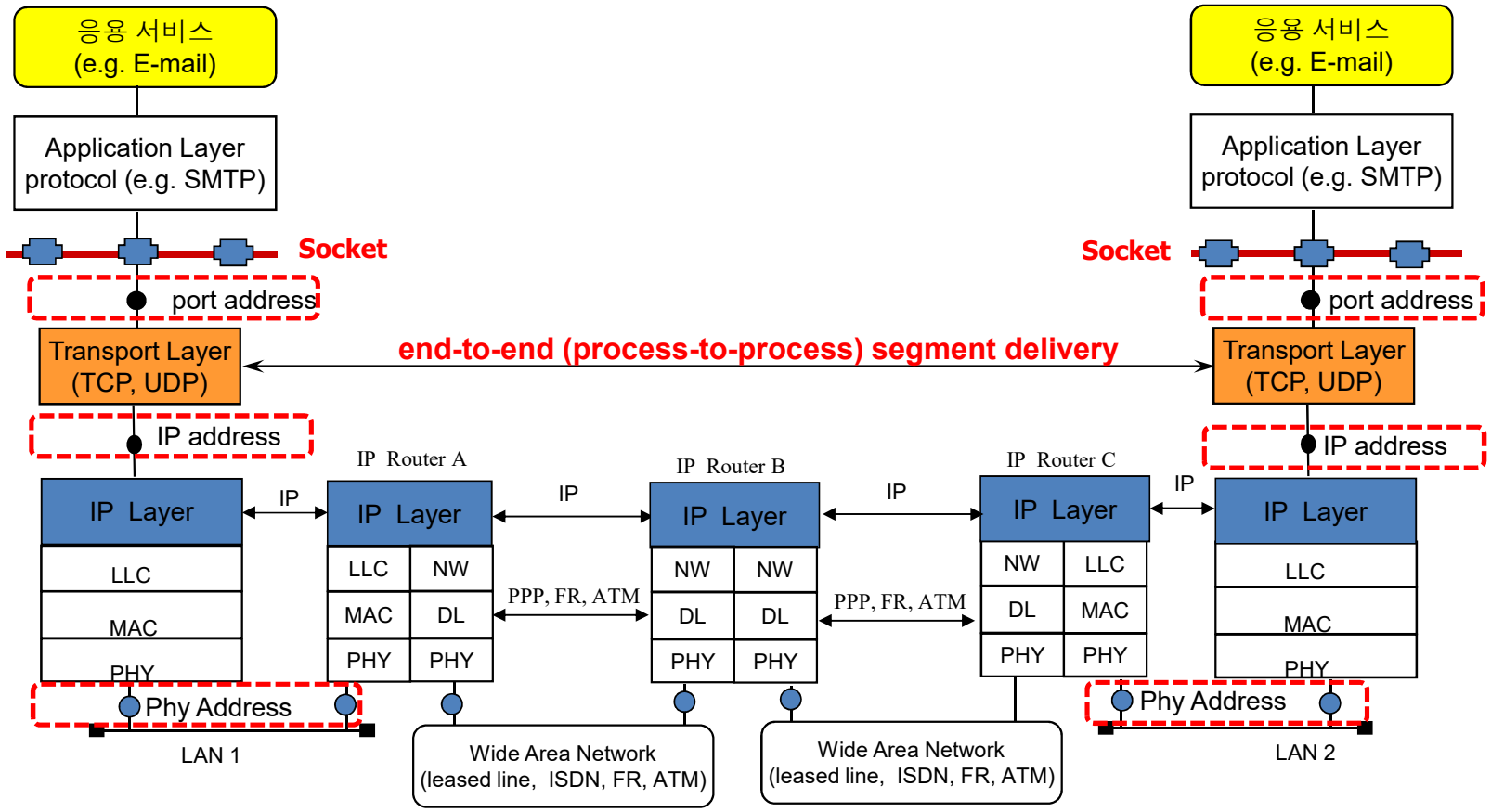
- IP address selects the host
- 인터넷 상에서 중복되지 않게 관리
- IPv4에서는 32비트 (4 바이트) 주소
- 영남대학교: 165.229.xxx.yyy

◆ Port number

- Port number selects the process
- 포트 번호는 16비트 (2바이트) 크기



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
PPP, FR, ATM
Phy Address
LAN 1
LAN 2
Wide Area Network
(leased line, ISDN, FR, ATM)

연결형 통신과 비연결형 통신

◆ 연결형 (Connection-oriented) 통신

- 정보 전달용 연결이 먼저 구성된 후 대화 (dialog)를 시작
- 일반 유선 전화
- TCP (transmission control protocol)
- 설정된 연결에 따라 흐름제어 (flow control)과 오류제어 (error control) 기능 제공으로 신뢰성이 높은 통신 기능 제공
- 연결 설정 및 흐름제어/오류제어의 지연 시간이 발생함

◆ 비연결형 (Connectionless) 통신

- 사전에 연결 설정 절차없이 데이터 그램 (datagram) 정보를 전송
- 우편 시스템을 사용한 편지와 전보 (telegram)
- UDP (user datagram protocol)
- 연결 설정 및 흐름제어/오류제어의 지연 시간이 없어 신속하게 데이터 처리
- 통신망의 상태가 안정적이지 않을 경우, 별도의 흐름제어/오류제어 기능을 추가하여야 함

User Datagram Protocol (UDP)

UDP

- connectionless, unreliable transport protocol
- a very simple protocol using a minimum of overhead
- if a process wants to send a small message and does not care much about reliability, it can use UDP

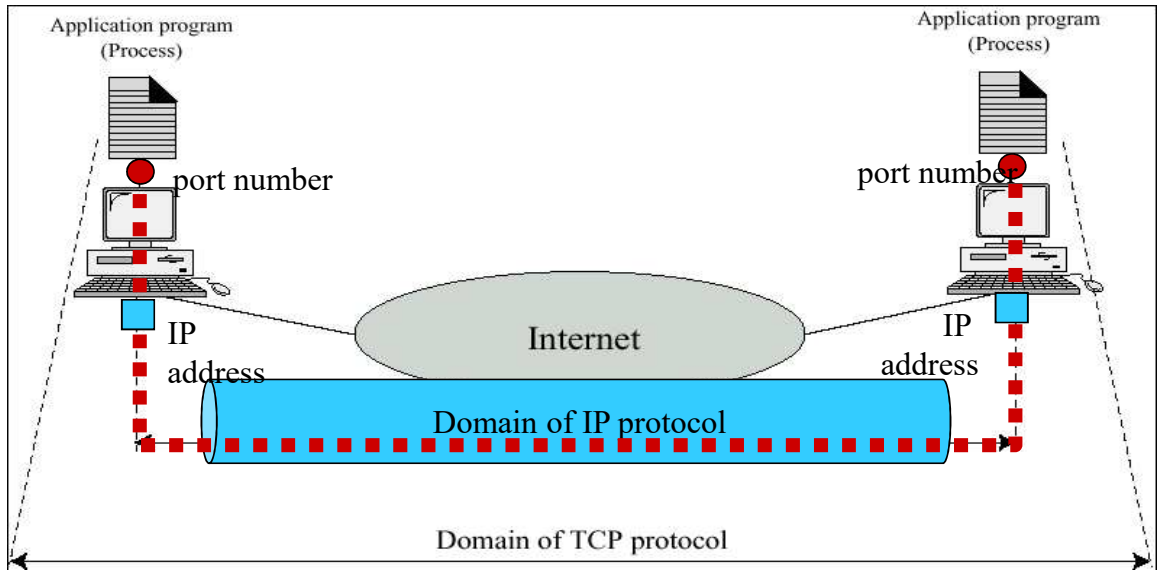
Use of UDP

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control; it is not usually used for a process such as FTP that needs to send bulk data
- UDP is suitable for a process with internal flow and error control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP)

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

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

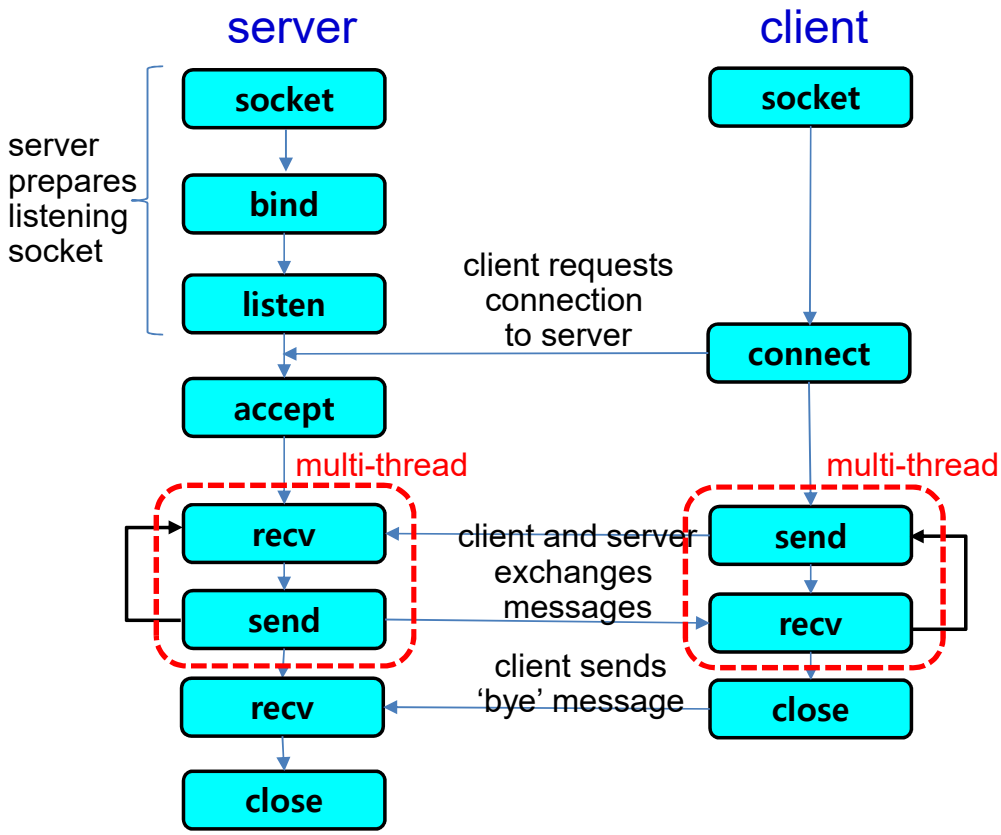
◆ Python socket module (2)

- 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()
```

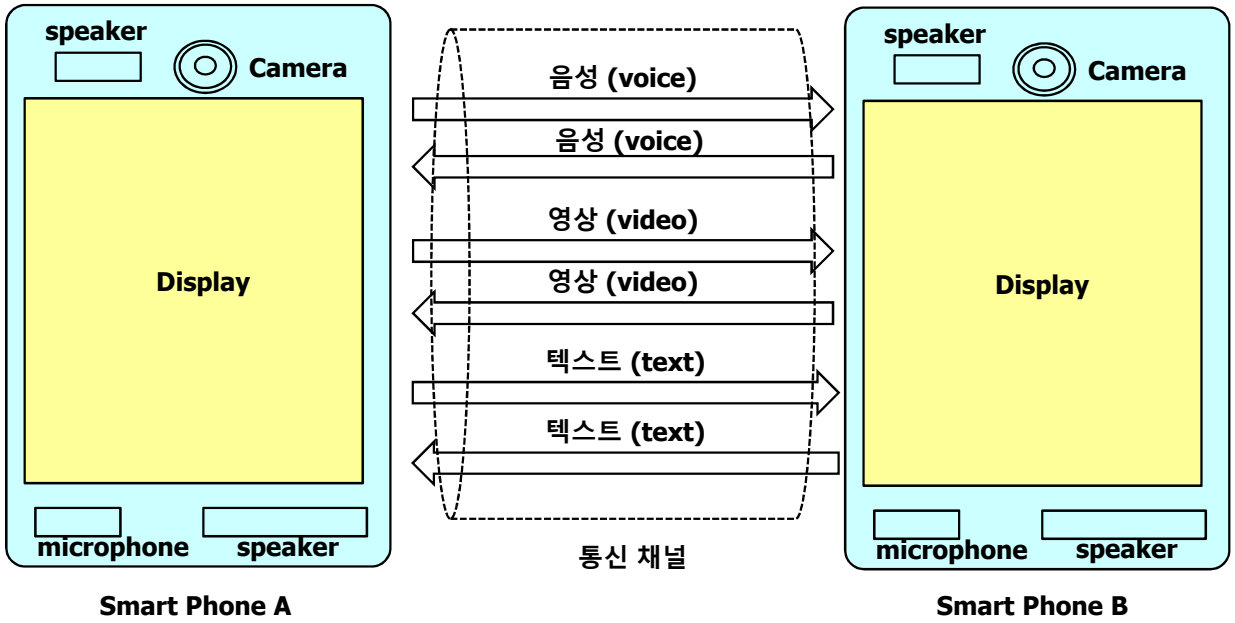
```
# Python Socket Communication - Threaded TCP Streaming Client

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

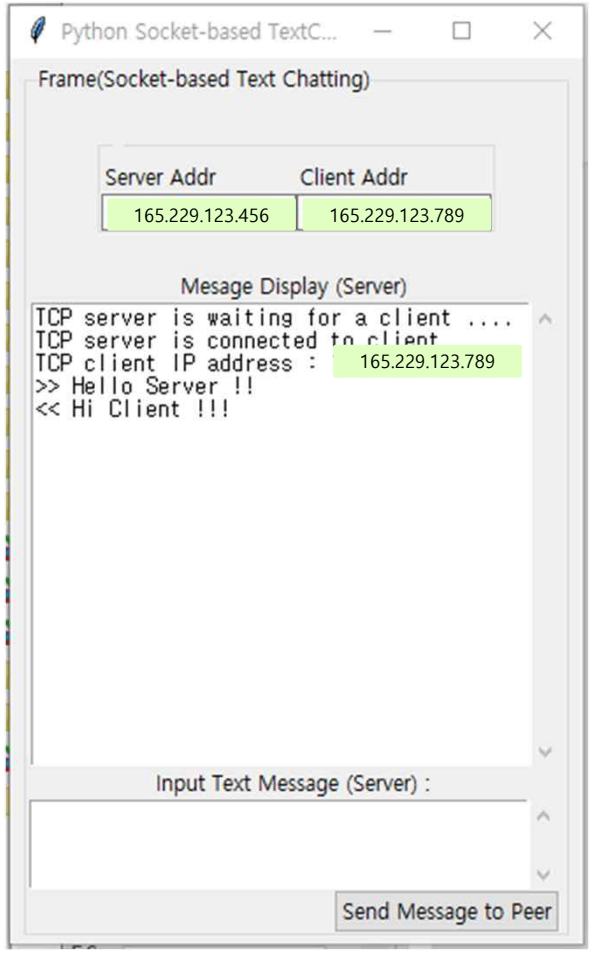
- 음성 및 영상의 양방향 전이중 (bidirectional full-duplex) 통신
- 음성/영상/텍스트/데이터를 동시에 송신 및 수신

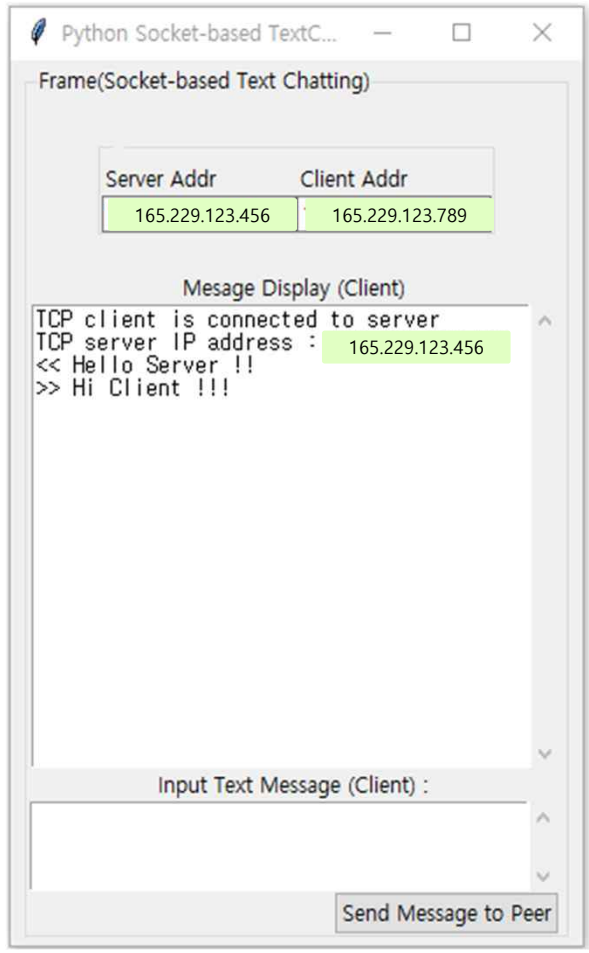


TextChat Server, TextChat Client

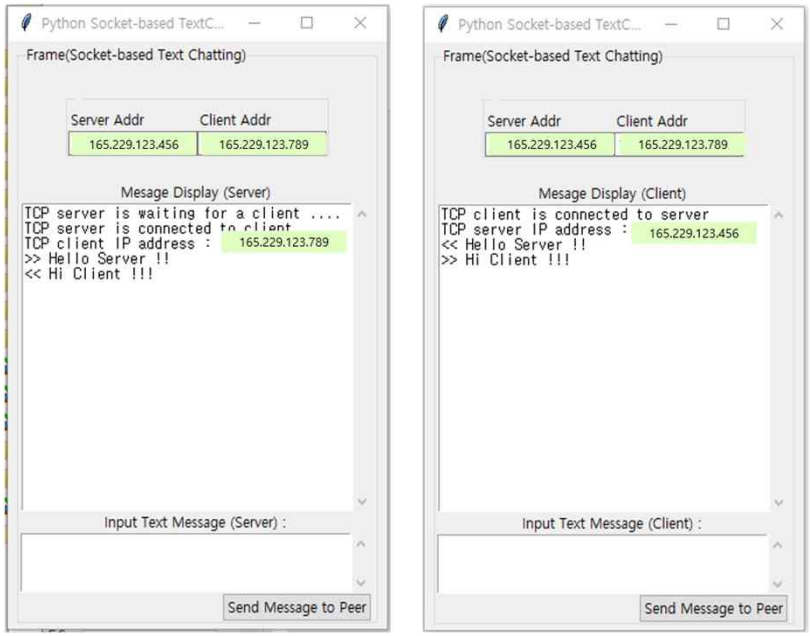
```
# TextChat Server

from Class_TextChat  import *

# Application of TextChatting as TCP Server
if __name__ == "__main__":
    print("Running TCP server")
    textChatServer = TextChat("Server")
    textChatServer.win.mainloop()
```

```
# TextChat Client

from Class_TextChat import *

# Application of TextChatting as TCP Client
if __name__ == "__main__":
    print("Running TCP Client")
    textChatClient = TextChat('Client')
    textChatClient.win.mainloop()
```



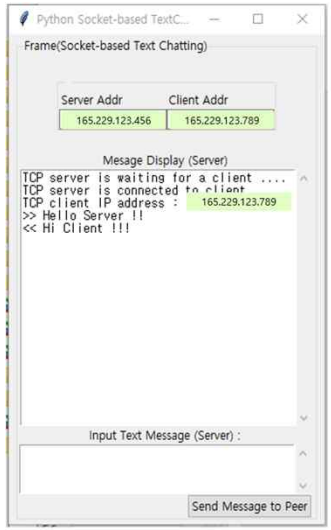


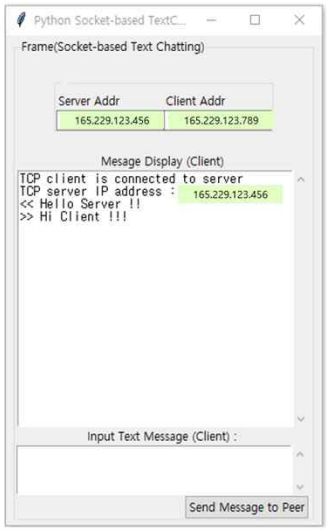
```
# User-defined module - Class_TextChat.py (1)
import socket, sys, threading
from threading import Thread # for testing multi-thread
from time import  sleep #for sleep in thread
import tkinter as tk
from tkinter import ttk, scrolledtext, END
LocalHost = "127.0.0.1"
SocketChat_PortNumber = 24000

class TextChat():
    def __init__(self, role):
        global hostAddr
        # Create instance
        self.win = tk.Tk()
        self.myRole = role # "Server" or "Client"

        # Add a title
        self.win.title("Python Socket-based TextChatt ({})".format(self.myRole))
        hostname = socket.gethostname()
        hostAddr = socket.gethostbyname(hostname)
        print("My ({}) IP address = {}".format(self.myRole, hostAddr))
        self.myAddr = hostAddr
        self.createWidgets()
```



```
# User-defined module - Class_TextChat.py (2)

        self.mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        if self.myRole == "Server":
            self.mySocket.bind((hostAddr, SocketChat_PortNumber)) # bind socket to (IP_addr(local_host), port_number)
            self.scrDisplay.insert(tk.INSERT,"TCP server is waiting for a client .... \n" )
            self.mySocket.listen(1)
            self.conn, self.cliAddr = self.mySocket.accept() # cliAddr : (IPaddr, port_no)
            print("TCP Server is connected to client ({})\n".format(self.cliAddr))
            self.scrDisplay.insert(tk.INSERT,"TCP server is connected to client\n" )
            self.scrDisplay.insert(tk.INSERT, "TCP client IP address : {}\n".format(self.cliAddr[0]))
            self.cliAddr_entry.insert(END, self.cliAddr[0])
        elif self.myRole == "Client":
            self.cliAddr = self.myAddr
            self.cliAddr_entry.insert(END, self.myAddr)
            servAddr_str = input("Server IP Addr (e.g., '127.0.0.1') = ")
            self.mySocket.connect((servAddr_str, SocketChat_PortNumber)) # send connect request to TCP server
            self.servAddr = self.mySocket.getpeername()
            print("TCP Client is connected to server ({})\n".format(self.servAddr))
            self.scrDisplay.insert(tk.INSERT,"TCP client is connected to server \n")
            self.scrDisplay.insert(tk.INSERT,"TCP server IP address : {}\n".format(self.servAddr[0]) )
            self.servAddr_entry.insert(END, self.servAddr[0])
            self.conn = self.mySocket

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

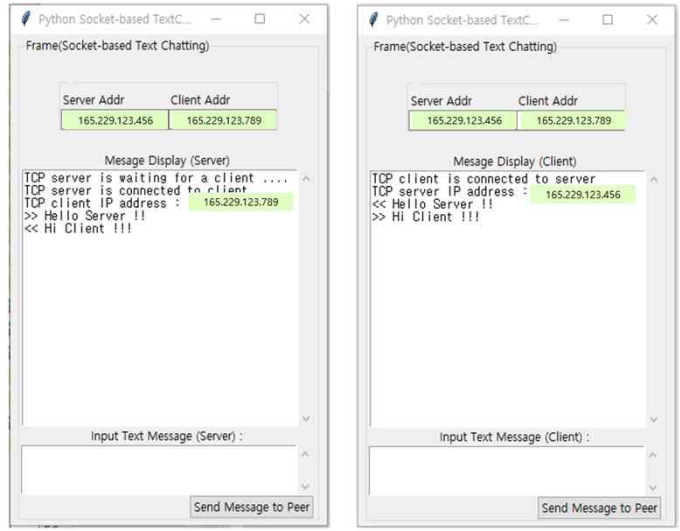
```
# User-defined module - Class_TextChat.py (3)

    def sockRecvMsg(self):
        while True:
            recvMsg = self.conn.recv(512).decode()
            if not recvMsg:
                break
            self.scrDisplay.insert(tk.INSERT,">> " + recvMsg)
        self.conn.close()

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

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



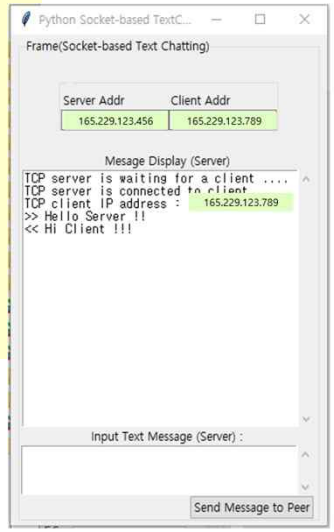


User-defined module - Class_TextChat.py (4)

```
    def createWidgets(self):
        ##########################################
        # Add a frame in self.win
        frame = ttk.LabelFrame(self.win, text="Frame(Socket-based Text 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
        servAddr_label = ttk.Label(frame_addr_connect, text="Server Addr")
        servAddr_label.grid(column=0, row=0, sticky='W') #
        cliAddr_label = ttk.Label(frame_addr_connect, text="Client Addr")
        cliAddr_label.grid(column=1, row=0, sticky='W') #

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

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

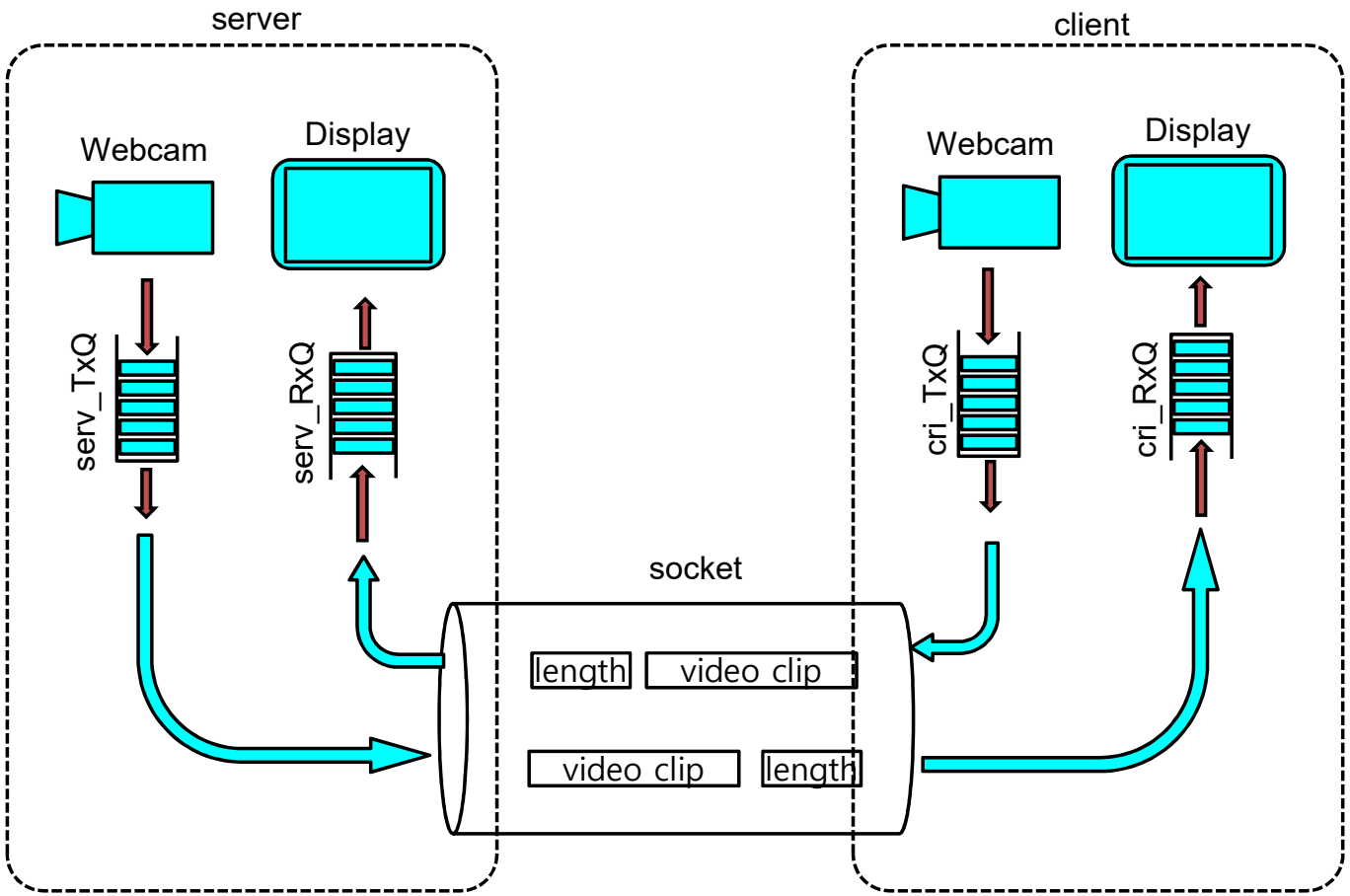


```
# User-defined module - Class_TextChat.py (5)

        # Add ScrolledText fields of display and input
        scrol_w,  scrol_h = 40, 20
        msgDisplay_label = ttk.Label(frame, text="Mesage Display ({})".format(self.myRole))
        msgDisplay_label.grid(column=0, row=1 )
        self.scrDisplay = scrolledtext.ScrolledText(frame, width=scrol_w, height=scrol_h, wrap=tk.WORD)
        self.scrDisplay.grid(column=0, row=2, sticky='E') #, columnspan=3

        msgInput_label = ttk.Label(frame, text="Input Text Message ({}) :".format(self.myRole))
        msgInput_label.grid(column=0, row=3 )

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

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

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



Python OpenCV Module 기반
영상 채팅 기능 구현

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 설치

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

```
C:₩Users₩Owner>python -m pip install --upgrade opencv-python
Collecting opencv-python
  Downloading opencv_python-4.5.4.60-cp310-cp310-win_amd64.whl (35.1 MB)
     |████████████████████████████████| 35.1 MB 6.8 MB/s
Requirement already satisfied: numpy>=1.21.2 in c:₩users₩owner₩appdata₩local₩programs₩python₩python310₩lib₩site-packages
 (from opencv-python) (1.21.4)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.5.4.60
```

◆ 또는 contribution 모듈이 포함된 OpenCV 설치

```
C:₩Users₩Owner>pip install --upgrade opencv-contrib-python
Collecting opencv-contrib-python
  Downloading opencv_contrib_python-4.5.4.60-cp310-cp310-win_amd64.whl (42.0 MB)
     |████████████████████████████████| 42.0 MB 6.4 MB/s
Requirement already satisfied: numpy>=1.21.2 in c:₩users₩owner₩appdata₩local₩programs₩python₩python310₩lib₩site-packages
 (from opencv-contrib-python) (1.21.4)
Installing collected packages: opencv-contrib-python
Successfully installed opencv-contrib-python-4.5.4.60
```

OpenCV 메소드 (1)

OpenCV method		설명
영상/ 이미지 입출력	cv2.imshow()	cv2.imshow("title", image) : 인수로 전달된 image를 화면에 출력하며, 영상 출력 화면의 제목은 title로 설정
	video_dev = cv2.VideoCapture() check, frame = video_dev.read() video_dev.release()	video 캡처 장치의 객체 생성 video 캡처 장치로 부터 영상 입력, 입력된 영상은 frame에 저장 video 캡처 장치를 해제
도형 및 텍스트 출력	cv2.rectangle()	지정된 위치에 지정된 크기의 사각형 도형 그리기
	cv2.circle()	지정된 위치에 지정된 크기의 원 그리기
	cv2.line()	지정된 두 지점 간에 선 그리기
	cv2.putText()	지정된 위치에 텍스트 출력
영상/ 이미지 변경	cv2.resize()	지정된 크기로 조정
	cv2.flip()	이미지를 지정된 축을 기준으로 뒤집기
	cv2.cvtColor()	이미지 색상을 변경
	matrix = cv2.getRotationMatrix2D (center, angle, scale)	영상 회전을 위한 2차원 행렬 생성
영상 편집	cv2.split()	이미지 분할
	cv2.merge()	이미지 병합

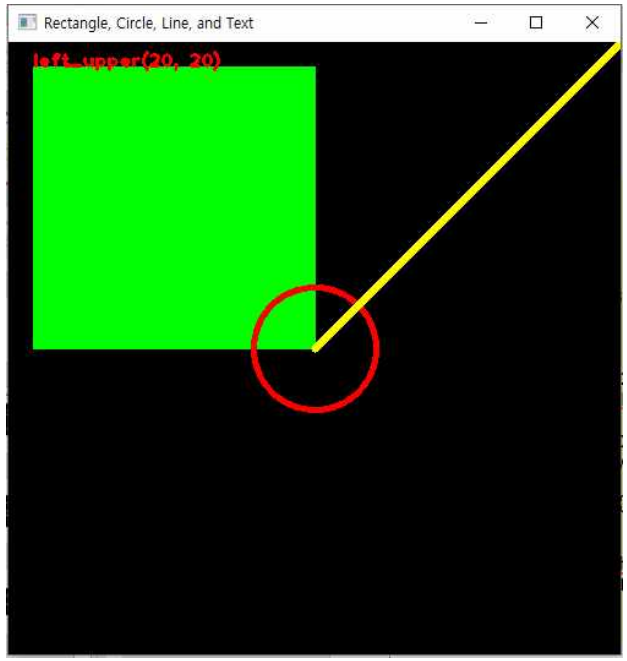
OpenCV 메소드 (2)

OpenCV method		설명
영상 처리	cv2.blur()	이미지를 흐릿하게 처리
	cv2.medianBlur()	이미지를 흐릿하게 처리
	cv2.GaussianBlur()	이미지를 흐릿하게 처리
	cv2.Canny()	경계 검출
	cv2.findContours()	경계 검출
	cv2.CascadeClassifier()	객체 검출 (object detection)
	cv2.threshold()	threshold
	cv2.bilateralFilter()	잡음 제거를 위한 필터링
기타	cv2.waitKey(delay)	ms단위로 설정된 delay 동안 키 입력 대기. delay가 0이면 무한대로 대기.
	cv2.destroyAllWindows()	현재 생성되어 있는 모든 영상출력용 윈도우 들을 삭제

(OpenCV: https://docs.opencv.org/4.x/index.html)

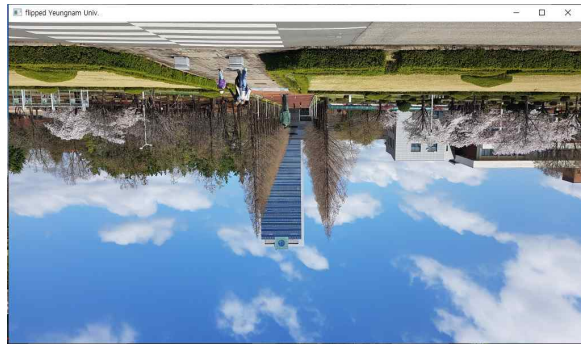
OpenCV - rectangle(), circle()

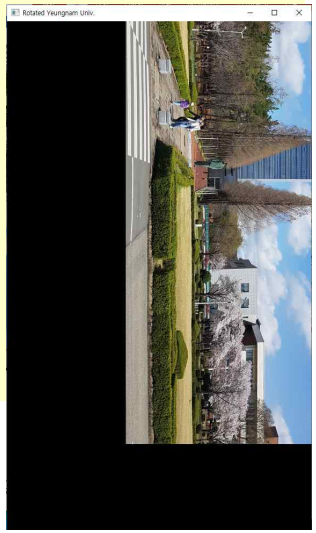
```
# Testing OpenCV

import cv2 as cv
import numpy as np

blank = np.zeros((500, 500, 3), dtype='uint8')
# blank[200:300, 300:400] = 255, 0, 0 # B, G, R
# cv.imshow('Blue', blank)

# cv.rectangle(blank, (0, 0), (250, 250), (0, 0, 255), thickness=cv.FILLED)
cv.rectangle(blank, (20, 20), (blank.shape[0]//2, blank.shape[1]//2),\
    (0, 255, 0), thickness=cv.FILLED)
cv.circle(blank, (blank.shape[0]//2, blank.shape[1]//2), 50, (0, 0, 255), thickness=3)
cv.line(blank, (blank.shape[0]//2, blank.shape[1]//2), (blank.shape[0], 0),\
  (0, 255, 255), thickness=5)
cv.putText(blank, 'left_upper({}, {})'.format(20, 20), (20, 20), cv.FONT_HERSHEY_PLAIN,\
  1.0, (0, 0, 255), 2)
#cv.putText(blank, 'center({}, {})'.format(blank.shape[0]//2, blank.shape[1]//2),\
# (blank.shape[0]//2, blank.shape[1]//2), cv.FONT_HERSHEY_PLAIN, 1.0, (0, 0, 255), 2)
cv.imshow('Rectangle, Circle, Line, and Text', blank)
cv.waitKey(0)
```



OpenCV - flip(), rotation

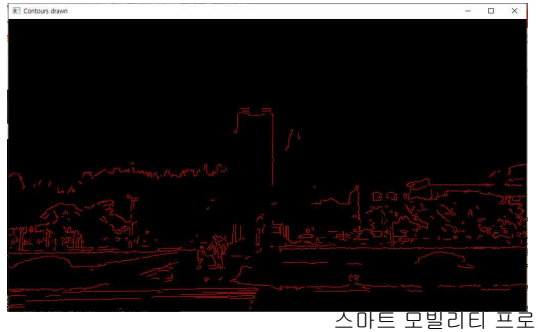
```
import cv2 as cv

yu_img_org = cv.imread("./images/yu_img.jpg")
height, width, channel = yu_img_org.shape
print("yu_img_org shapes: height({}), width({}), channel({})".format(height, width, channel))
yu_img = cv.resize(yu_img_org, dsize=(width//4, height//4), interpolation=cv.INTER_AREA)
cv.imshow("Yeungnam Univ. ({}, {})".format(width//4, height//4), yu_img)

flipped_yu_img = cv.flip(yu_img, 1)
cv.imshow("flipped Yeungnam Univ.", flipped_yu_img)

center = (yu_img.shape[1]//2, yu_img.shape[0]//2)
angle = -90
scale = 1
rot_mtrx = cv.getRotationMatrix2D(center, angle, scale)
rotated_yu_img = cv.warpAffine(yu_img, rot_mtrx,₩
  (yu_img.shape[0], yu_img.shape[1]))
cv.imshow("Rotated Yeungnam Univ.", rotated_yu_img)

cv.waitKey(0)
```





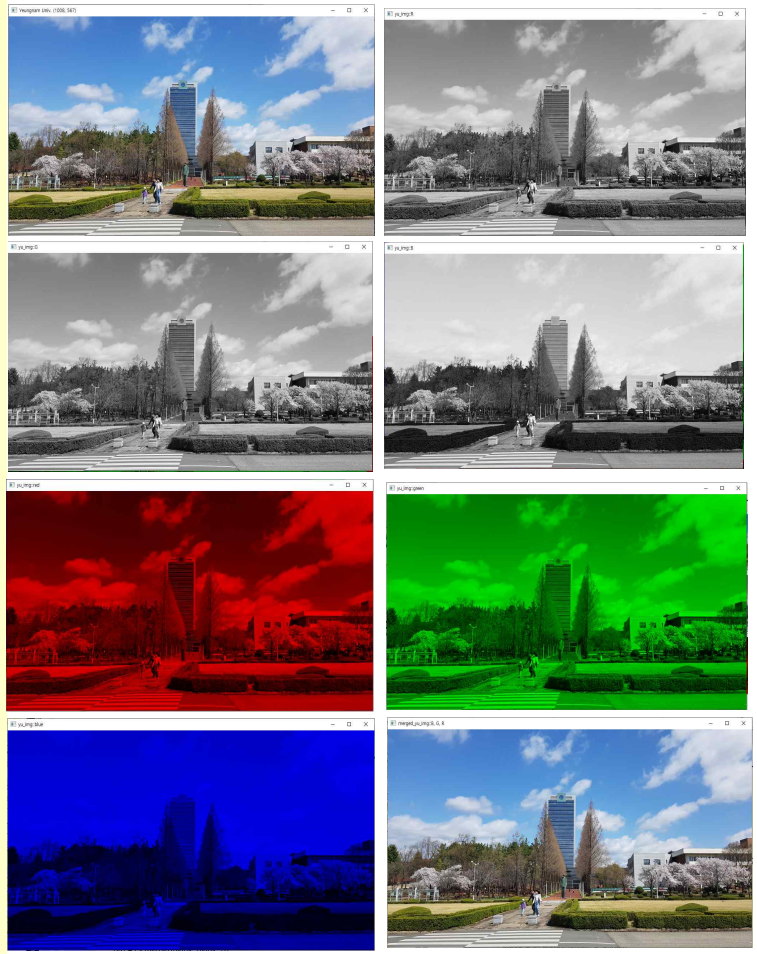
OpenCV 기반 경계선 검출

```
# OpenCV, gray, contour, thresh
import cv2 as cv
import numpy as np

yu_img_org = cv.imread("./images/yu_img.jpg")
height, width, channel = yu_img_org.shape
print("yu_img_org shapes: height({}), width({}), channel({})".format(height, width, channel))
yu_img = cv.resize(yu_img_org, dsize=(width//4, height//4), interpolation=cv.INTER_AREA)
cv.imshow("Yeungnam Univ. ({}, {})".format(width//4, height//4), yu_img)
blank = np.zeros(yu_img.shape, dtype='uint8')

gray_yu_img = cv.cvtColor(yu_img, cv.COLOR_BGR2GRAY)
cv.imshow("Gray YU Img", gray_yu_img)

blurred_yu_img = cv.GaussianBlur(gray_yu_img, (5, 5), cv.BORDER_DEFAULT)
cv.imshow("Gray YU Img", blurred_yu_img)

canny_blurred_yu_img = cv.Canny(blurred_yu_img, 125, 175)
cv.imshow("Canny Blurred YU Img", canny_blurred_yu_img)

contours, hierarchies = cv.findContours(canny_blurred_yu_img, cv.RETR_LIST, cv.CHAIN_APPROX_NONE)
print("after blurring, {} contours were found !".format(len(contours)))

# ret, thresh_yu_img = cv.threshold(blurred_yu_img, 125, 255, cv.THRESH_BINARY)
# cv.imshow("Threshold YU Img", thresh_yu_img)
# contours, hierarchies = cv.findContours(thresh_yu_img, cv.RETR_LIST, cv.CHAIN_APPROX_NONE)
# print("with blurring and threshold, {} contours were found !".format(len(contours)))

cv.drawContours(blank, contours, -1, (0,0,255), 1)
cv.imshow("Contours drawn", blank)
cv.waitKey(0)
```



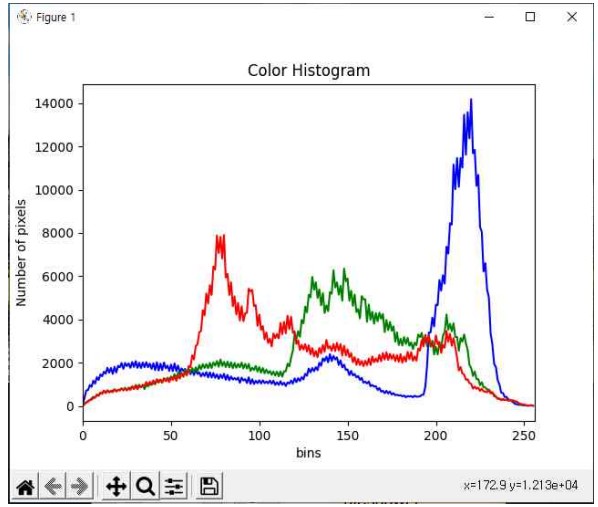
Splitting Color Channels - split(), merge()

```
# OpenCV, gray, contour, thresh
import cv2 as cv
import numpy as np

yu_img_org = cv.imread("./images/yu_img.jpg")
height, width, channel = yu_img_org.shape
print("yu_img_org shapes: height({}), width({}), channel({})".format(height, width, channel))
yu_img = cv.resize(yu_img_org, dsize=(width//4, height//4), interpolation=cv.INTER_AREA)
cv.imshow("Yeungnam Univ. ({}, {})".format(width//4, height//4), yu_img)
blank = np.zeros(yu_img.shape[:2], dtype='uint8')
b, g, r = cv.split(yu_img)
cv.imshow("yu_img::B", b)
cv.imshow("yu_img::G", g)
cv.imshow("yu_img::R", g)

merged_yu_img = cv.merge([b, g, r])
cv.imshow("merged_yu_img::B, G, R", merged_yu_img)

blue = cv.merge([b, blank, blank])
green = cv.merge([blank, g, blank])
red = cv.merge([blank, blank, r])

cv.imshow("yu_img::blue", blue)
cv.imshow("yu_img::green", green)
cv.imshow("yu_img::red", red)

cv.waitKey(0)
```



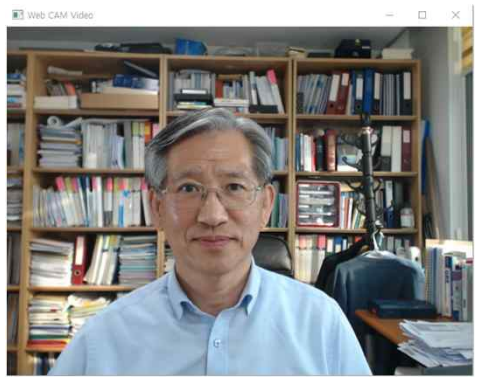
OpenCV - histogram

```
# OpenCV - histogram
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

yu_img_org = cv.imread("./images/yu_img.jpg")
height, width, channel = yu_img_org.shape
print("yu_img_org shapes: height({}), width({}), channel({})".format(height, width, channel))
yu_img = cv.resize(yu_img_org, dsize=(width//4, height//4), interpolation=cv.INTER_AREA)
cv.imshow("Yeungnam Univ. ({}, {})".format(width//4, height//4), yu_img)

# gray_yu_img = cv.cvtColor(yu_img, cv.COLOR_BGR2GRAY)
# cv.imshow("Grayscale yu_img", gray_yu_img)
colors = ['b', 'g', 'r']
plt.figure()
plt.title("Color Histogram")
plt.xlabel("bins")
plt.ylabel("Number of pixels")
for i, col in enumerate(colors):
    hist = cv.calcHist([yu_img], [i], None, [256], [0, 256])
    plt.plot(hist, color=col)
    plt.xlim([0, 256])

plt.show()

cv.waitKey(0)
```



Web CAM Video Capture, Display

```
# Web_cam Video Capture and Show

import cv2

WEBCAM_NO = 0
webcam = cv2.VideoCapture(WEBCAM_NO)
while True:
    ret, frame = webcam.read()
    if ret == False:
        continue
    cv2.imshow('Web CAM Video', frame)
    key = cv2.waitKey(1)
    if key == 27: # if ESC key is input, then exit
        break
```

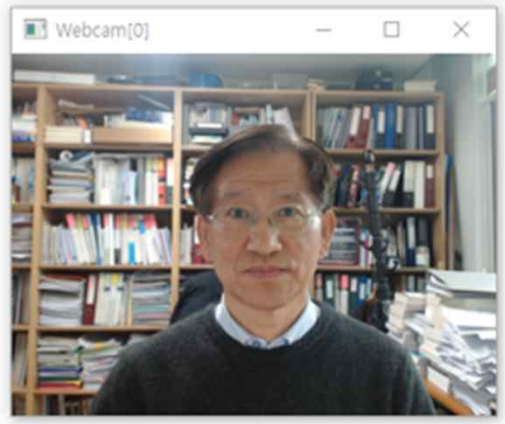


Python OpenCV Module 기반
양방향 화상 통신

```
# OpenCV, WebcamConfig test (1)
import cv2
import numpy
import threading
import time
from _thread import *

def webcam_capture_display(webcam_dev):
    print("Preparing webcam[{}] . . . . ".format(webcam_dev))
    webcam = cv2.VideoCapture(webcam_dev)
    # check webcam
    ret, vframe = webcam.read()
    if ret == False:
        print("Error in reading webcam[{}]".format(webcam_dev))
        return
    else:
        print("Webcam[{}] is correctly working now ...".format(webcam_dev))

        fr_width, fr_height = webcam.get(3), webcam.get(4)
        fps = webcam.get(cv2.CAP_PROP_FPS)
        print("Webcam cam frame width ({}), height({})".format(fr_width, fr_height))
        print(" frame-per-sec = {}".format(fps))
        print("Enter ESC to quit")
    while True:
        ret, vframe = webcam.read()
        if ret == False:
            continue
        resized_fr = cv2.resize(vframe, (int(vframe.shape[1]/2), int(vframe.shape[0]/2)))
        cv2.imshow('Webcam[{}]'.format(webcam_dev), resized_fr)
        key = cv2.waitKey(1)
        if key == 27: # if ESC key is input, then exit
            break
        time.sleep(0.1)
```

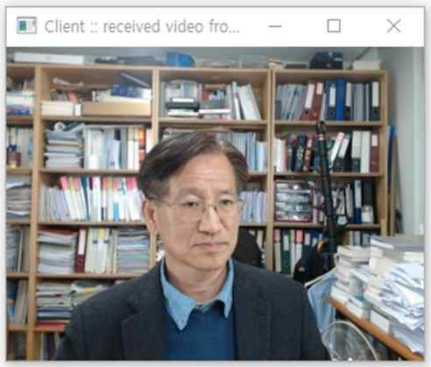
```
# OpenCV, WebcamConfig test (2)

if __name__ == "__main__":
    webcam_no = int(input("Input webcam No (0 or 1) : "))
    print('Starting webcam_capture_dispaly')
    thrd_webcamCapDisp = threading.Thread(target=webcam_capture_display, args=(webcam_no,))
    thrd_webcamCapDisp.start()

    thrd_webcamCapDisp.join()
```

```
Input webcam No (0 or 1) : 0
Starting webcam_capture_dispaly
Preparing webcam[0] . . . .
Webcam[0] is correctly working now ...
Webcam cam frame width (640.0), height(480.0)
 frame-per-sec = 30.0
Enter ESC to quit
```



```
# VideoChatting Server - class Video-Chatting Server with OpenCV and socket

from Class_VideoChat import *

if __name__ == "__main__":
    videoChatt_server = VideoChat("Server")
    videoChatt_server.run()
```

```
# VideoChatting Client - class Video-Chatting client with OpenCV and socket

from Class_VideoChatting import *

if __name__ == "__main__":
    videoChatt_client = VideoChat("Client")
    videoChatt_client.run()
```



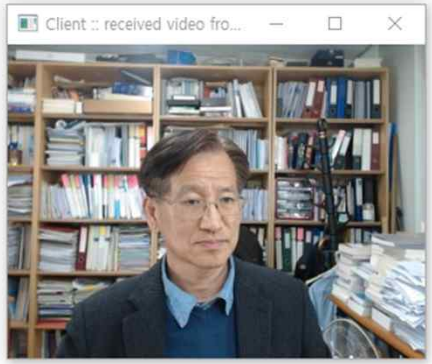

```
# User0defined module - Class_VideoChat (1)
import socket, cv2, time
import numpy as np
import threading
from queue import Queue
from _thread import *

PORT = 9999

class VideoChat():
    def __init__(self, role):
        self.myRole = role
        print("VideoChat initiated as {} ...".format(role))
        hostname = socket.gethostname()
        self.myAddr = socket.gethostbyname(hostname)
        print("My IP address = {}".format(self.myAddr))
        if self.myRole == "Server":
            self.myWebCam = 0 # SERVER_WEBCAM = 0
        else:
            self.myWebCam = 1  # CLIENT_WEBCAM = 1
        self.op_state = "RUN"
```

```
# User0defined module - Class_VideoChat (2)

    def run(self):
        if self.myRole == "Server":
            self.mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.mySocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            self.mySocket.bind((self.myAddr, PORT))
            self.mySocket.listen()
            print('Server::Video chatting server started')
            print('Server::Waiting for client .... ')
            self.peerSocket, self.peerAddr = self.mySocket.accept()
            print('Server::connected to client ({} : {})'.format(self.peerSocket, self.peerAddr))
        elif self.myRole == "Client":
            self.peerAddr = input("Input server IP address = ")
            print('Client::Connecting to Server')
            self.mySocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            self.mySocket.connect((self.peerAddr, PORT))
            print('Client::Connected to Server({}:{})'.format(self.peerAddr, PORT))
            self.peerSocket = self.mySocket

        self.queue = Queue()
        thrd_CaptureVideo = threading.Thread(target= self.captureVideo, args=(self.queue,))
        thrd_CaptureVideo.start()
        thrd_TxVideo = threading.Thread(target=self.txVideo, args=(self.peerSocket, self.peerAddr, self.queue,))
        thrd_TxVideo.start()
        thrd_RxVideo = threading.Thread(target=self.rxVideo, args=(self.peerSocket,))
        thrd_RxVideo.start()

        thrd_TxVideo.join()
        thrd_RxVideo.join()
        thrd_CaptureVideo.join()
        print("VideoChatt( {}) is closing socket and quit video chatt".format(self.myRole))
        self.mySocket.close()
```

```
# User0defined module - Class_VideoChat (3)

    def recvall(self, sock, count):
        if count == 0 or count == None:
            return None
        buf = b''
        while count:
            try:
                newbuf = sock.recv(count)
            except:
                self.op_state = "QUIT"
                break
            if not newbuf:
                return None
            buf += newbuf
            count -= len(newbuf)
        return buf

    def txVideo(self, peerSocket, addr, queue):
        while True:
            if self.op_state == "QUIT":
                break
            if queue.empty ():
                #print("{}::queue is empty, self.op_state = {}".format(self.myRole, self.op_state))
                time.sleep(0.1)
            try:
                stringData = queue.get()
                peerSocket.send(str(len(stringData)).ljust(16).encode())
                peerSocket.send(stringData)
            except: # ConnectionResetError, ConnectionAbortedError
                self.op_state = "QUIT"
                break
            time.sleep(0.1)
        print("{}:: closing peerSocket() ...".format(self.myRole))
        peerSocket.close()
        print("{}:: terminating thread_txVideo() ...".format(self.myRole))
```

```
# User0defined module - Class_VideoChat (4)

    def captureVideo(self, queue):
        server_webcam = cv2.VideoCapture(self.myWebCam)
        server_webcam.set(cv2.CAP_PROP_FPS, 8) # change FPS from 30 to 8
        fr_width, fr_height, fps = server_webcam.get(3), server_webcam.get(4), server_webcam.get(cv2.CAP_PROP_FPS)
        print("{}_webcam frame width ({}), height({}), fps({})".format(self.myRole, fr_width, fr_height, fps))
        while True:
            if self.op_state == "QUIT":
                break
            ret, serv_frame = server_webcam.read()
            if ret == False:
                continue
            resized_svrfr = cv2.resize(serv_frame, (int(serv_frame.shape[1]/2), int(serv_frame.shape[0]/2)))
            encode_param=[int(cv2.IMWRITE_JPEG_QUALITY),90]
            result, imgencode = cv2.imencode('.jpg', resized_svrfr, encode_param)
            img_data = np.array(imgencode)
            stringData = img_data.tobytes()
            queue.put(stringData)
            #cv2.imshow('Server:: Resized_Server_Video', resized_svrfr)
            key = cv2.waitKey(1)
            if key == 27: # if ESC key is input, then exit
                print("{} :: ESC key pressed => exit".format(self.myRole))
                self.op_state = "QUIT"
                break
            time.sleep(0.05)
        print("{}:: terminating thread_captureVideo() ...".format(self.myRole))
```

```
# User0defined module - Class_VideoChat (4)

    def rxVideo (self, peerSocket):
        while True:
            if self.op_state == "QUIT":
                break
            length = self.recvall (peerSocket, 16)
            if length == 0 or length == None or length == b":
                self.op_state = "QUIT"
                break
            stringData = self.recvall(peerSocket, int(length))
            data = np.frombuffer(stringData, dtype='uint8')
            decimg=cv2.imdecode(data,1)
            cv2.imshow('{} :: received video from peer'.format(self.myRole),decimg)
            key = cv2.waitKey(1)
            if key == 27: # if ESC key is input, then exit
                print("{} :: ESC key pressed => exit".format(self.myRole))
                self.op_state = "QUIT"
                break
            time.sleep(0.05)
        print("{}:: terminating thread_rxVideo() ...".format(self.myRole))
```




```
VideoChatt initiated as Server ...
My IP address = 165.229.229.50
Server::Video chatting server started
Server::Waiting for client ....
Server::connected to client (<socket.socket fd=1384, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0,
 laddr=('165.229.229.50', 9999), raddr=('165.229.229.50', 5409)> : ('165.229.229.50', 5409))
Server_webcam frame width (640.0), height(480.0), fps(7.500001875000469)


VideoChatt initiated as Client ...
My IP address = 165.229.229.50
Input server IP address = 165.229.229.50
Client::Connecting to Server
Client::Connected to Server(165.229.229.50:9999)
Client_webcam frame width (640.0), height(480.0), fps(7.500001875000469)
```

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 간단한 파이썬 스레드 프로그램

- 한 줄에 per_line (예: 50)개의 지정된 마커 (marker) (예: 'A')를 각각 지정된 횟수 max_count 만큼 출력하는 함수 printMarker(marker, max_count, per_line, delay)를 구현하라. 한 줄을 출력한 후, 초 단위 delay 만큼 sleep하도록 할 것.
- printMarker() 함수를 파이썬 스레드로 구성하고, 인수로 각각 ('A', 40, 50, 0.2), ('B', 30, 50, 0.3), ('C', 20, 50, 0.4), ('D', 10, 50, 0.5)로 설정하며, daemon 스레드로 생성되도록 하라.
- (실행 예시)

```
main():: before creations and starts of threads
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 20876)>, <Thread(SockThread, started daemon 17448)>]
Thread_A::starts
Thread_B::starts
Thread_C::starts
Thread_D::starts

main():: currently 6 threads (including main) are active :



ABCD  ABCD[<_MainThread(MainThread, started 20876)>, <Thread(SockThread, started daemon 17448)>, <Thread(thread_A, started daemon 10368)>, <Thread(thread_B, star
ted daemon 18368)>, <Thread(thread_C, started daemon 6816)>, <Thread(thread_D, started daemon 10172)>]
ABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDABCDA
BCDABCDABCDABCDABCDABCDABCDABCD


AAAAAAAAAAAAAAAAAAAAAAAAAAAAAABABABABABABABABABABABABABABABABABABABAB
BBBBBBBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBC
```

▪ ▪ ▪ ▪

```
main():: currently 4 threads (including main) are active :B
 B [<_MainThread(MainThread, started 20876)>, <Thread(SockThread, started daemon 17448)>, <Thread(thread_A, started daemon 10368)>, <Thread(thread_B, started dae
mon 18368)>]B
BBBBBBBBBBBBBBBBBBBBBBBThread_A::terminates

BBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Thread_B::terminates

main():: all threads are terminated now !!
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 20876)>, <Thread(SockThread, started daemon 17448)>]
```

Homework 12.2

12.2 공유자원 관리 기능이 포함된 파이썬 스레드 프로그램

- 10.1에서 구현하였던 printMarker() 함수에 인수 semaphore를 추가하여 printMarker(marker, max_count, per_line, delay, semaphore)로 구현하라. 추가된 semaphore는 화면 출력을 제어하며, 한 줄의 출력이 완료될 때 까지 다른 스레드가 방해하지 않도록 임계구역을 설정하도록 할 것.
- 화면 출력 기능에 대한 임계 구역 설정/관리를 위하여 세마포 console_lock을 생성하라.
- printMarker() 함수를 파이썬 스레드로 구성하고, 인수로 각각 ('A', 40, 50, 0.2, console_lock), ('B', 30, 50, 0.3, console_lock), ('C', 20, 50, 0.4, console_lock), ('D', 10, 50, 0.5, console_lock)로 설정하며, daemon 스레드로 생성되도록 하라.
- (실행 예시)

```
main():: before creations and starts of threads
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 18952)>, <Thread(SockThread, started daemon 18572)>]
Thread_A::starts

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Thread_C::starts

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

main():: currently 6 threads (including main) are active :
  [<_MainThread(MainThread, started 18952)>, <Thread(SockThread, started daemon 18572)>, <Thread(th
read_A, started daemon 16888)>, <Thread(thread_B, started daemon 20528)>, <Thread(thread_C, started
daemon 18028)>, <Thread(thread_D, started daemon 17952)>]
Thread_B::starts

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Thread_D::starts

DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

▪ ▪ ▪ ▪

```
main():: currently 4 threads (including main) are active :
  [<_MainThread(MainThread, started 18952)>, <Thread(SockThread, started daemon 18572)>, <Thread(th
read_A, started daemon 16888)>, <Thread(thread_B, started daemon 20528)>]
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

main():: currently 4 threads (including main) are active :
  [<_MainThread(MainThread, started 18952)>, <Thread(SockThread, started daemon 18572)>, <Thread(th
read_A, started daemon 16888)>, <Thread(thread_B, started daemon 20528)>]
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Thread_A::terminates

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Thread_B::terminates

main():: all threads are terminated now !!
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 18952)>, <Thread(SockThread, started daemon 18572)>]
```

Homework 12.3

12.3 공유자원 관리 기능과 순서 제어 기능이 포함된 파이썬 스레드 프로그램

- 10.2에서 구현하였던 printMarker( ) 함수를 확장하여 printMarker(marker, turn, max_count, per_line, delay, semaphore)로 구현하라. 추가된 turn은 순서를 제어하며, 자신의 순서가 도달할 때 까지 대기하도록 하며, 한 줄을 출력한 후, 다음 marker 순서 (A -> B -> C -> D -> A)가 되도록 관리할 것. 즉, 동일한 스레드가 연속하여 두 줄 이상을 출력하지 않게 할 것.
- printMarker() 함수를 파이썬 스레드로 구성하고, 인수로 각각 ('A', turn, 50, 50, 0.1, console_lock), ('B', turn, 50, 50, 0.1, console_lock), ('C', turn, 50, 50, 0.1, console_lock), ('D', turn, 50, 50, 0.1, console_lock)로 설정하며, daemon 스레드로 생성되도록 하라.
- (실행 예시)

```
main():: before creations and starts of threads
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 20828)>, <Thread(SockThread, started daemon
21408)>]
Thread_A::starts

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Thread_C::starts

Thread_D::starts


main():: currently 6 threads (including main) are active :
  [<_MainThread(MainThread, started 20828)>, <Thread(SockThread, started daemon
21408)>, <Thread(thread_A, started daemon 20276)>, <Thread(thread_B, started dae
mon 8876)>, <Thread(thread_C, started daemon 20412)>, <Thread(thread_D, started
daemon 17924)>]
Thread_B::starts

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

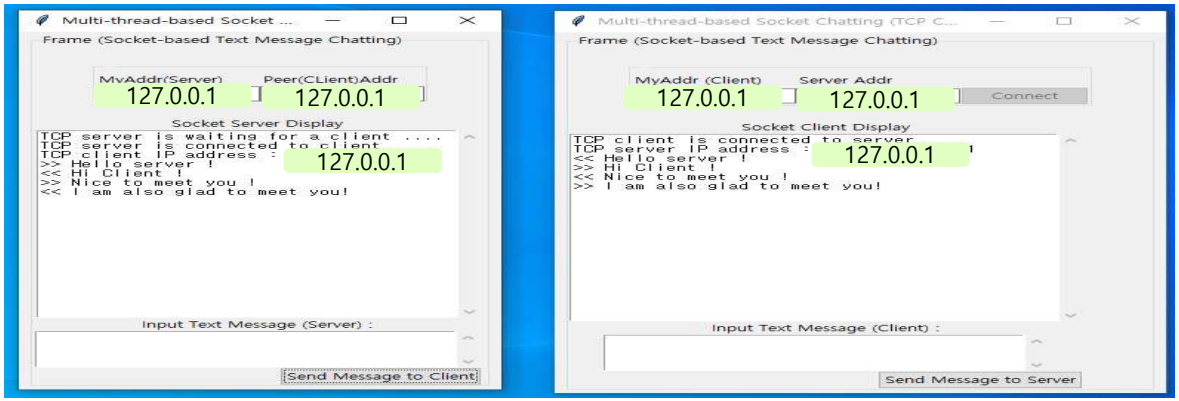
. . . .

```
main():: currently 6 threads (including main) are active :
  [<_MainThread(MainThread, started 20828)>, <Thread(SockThread, started daemon
21408)>, <Thread(thread_A, started daemon 20276)>, <Thread(thread_B, started dae
mon 8876)>, <Thread(thread_C, started daemon 20412)>, <Thread(thread_D, started
daemon 17924)>]
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Thread_A::terminates

DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Thread_B::terminates

Thread_C::terminates

Thread_D::terminates

main():: all threads are terminated now !!
main():: currently 2 threads (including main) are active :
  [<_MainThread(MainThread, started 20828)>, <Thread(SockThread, started daemon
21408)>]
```

Homework 12.4

12.4 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 12.5

12.5 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.
- TCP Socket 기반의 Video Chatting을 위한 **class VideoChat**을 사용자 정의 모듈 Class_VideoChat.py로 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.
- VideoChat 서버와 VideoChat 클라이언트는 반복적으로 webcam으로 부터 OpenCV를 사용하여 비디오 이미지를 입력받고, 이를 상대방으로 전송하며, 상대방으로 부터 전송 받은 비디오 이미지를 화면으로 출력할 것.
- Class_VideoChat을 기반으로 영상 채팅 server와 client를 각각 구현하며, 서로 다른 컴퓨터 간에 영상 채팅 프로그램을 설치하여 실행 결과를 capture하여 파이썬 소스코드와 함께 제출하라.
- 만약 다른 컴퓨터가 없는 경우, 별도의 Python shell에서 server와 client를 각각 실행하도록 할 것.