WORKSHOP PEMROGRAMAN JARINGAN

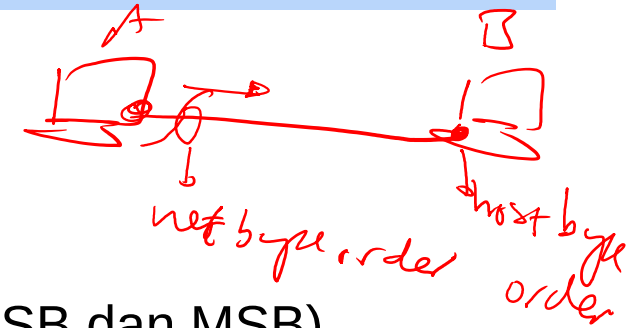
Mochammad Zen Samsono Hadi, ST. MSc. Ph.D

Teknik Telekomunikasi PENS

TOPIK PEMBAHASAN

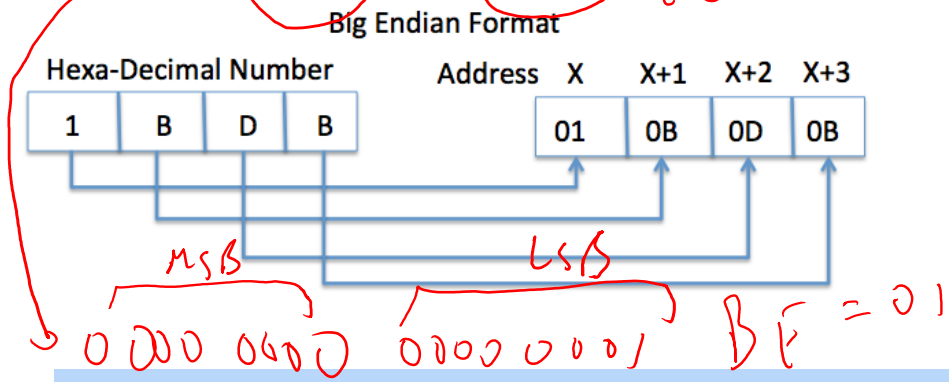
- Merubah integer / host ke network byte order
- Memodifikasi ukuran buffer Tx dan Rx
- Membuat aplikasi berbasis TCP
- Membuat aplikasi berbasis UDP

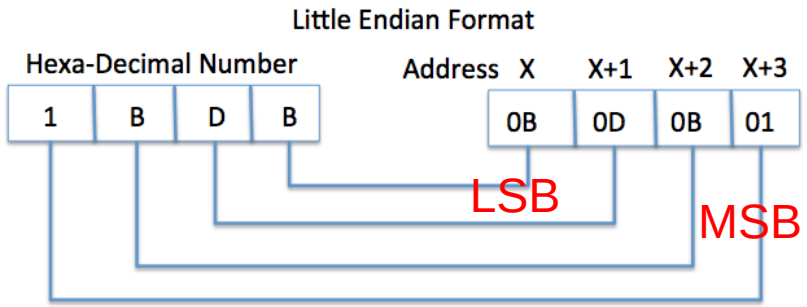
Merubah Integer / Host ke Network Byte Order



- Format data dalam jaringan komputer
  - Big Endian Format (MSB → LSB)
  - Little Endian Format (urutan terbalik antara LSB dan MSB)
- Ini untuk merubah IP address ke format yang dipahami oleh komputer.
- Misal: 01 -> Little Endian Format 16 bit menjadi 10
  - 0000 0001 0000 0000 => 256

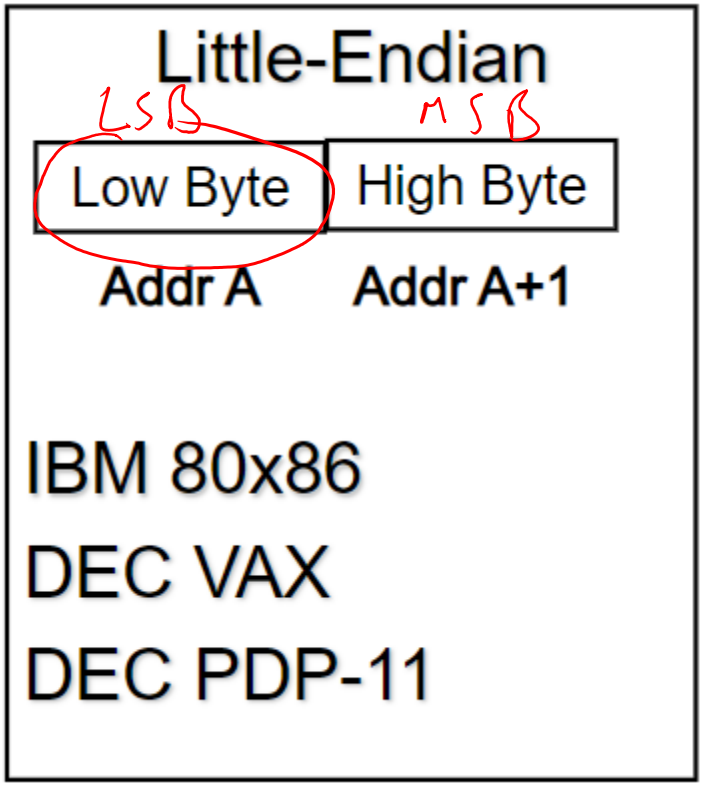
LSB MSB

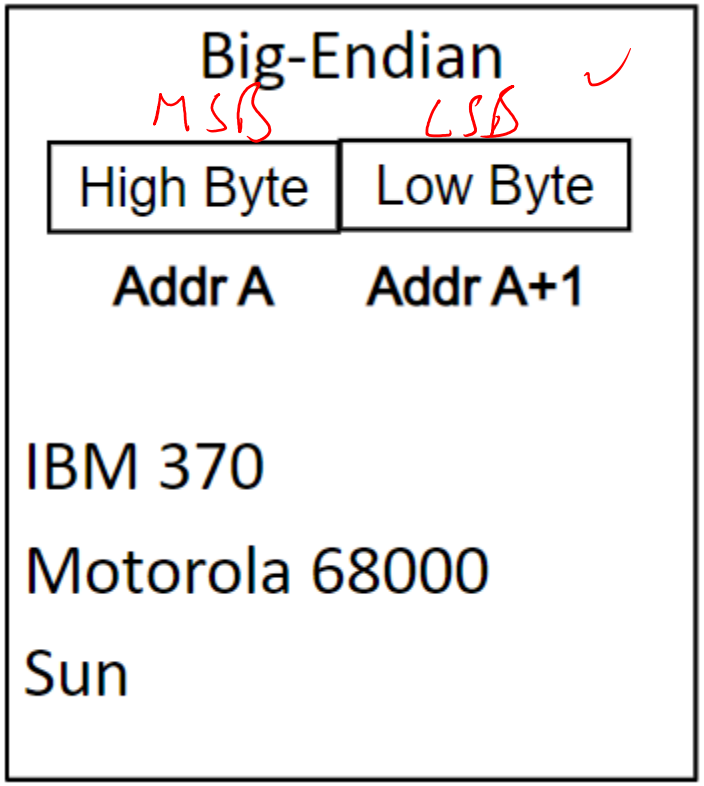




Merubah Integer / Host ke Network Byte Order

- Byte ordering





Merubah Integer / Host ke Network Byte Order

- **Network Byte Order Function**
- Network Byte Order = Big-Endian Architecture
- Ada 2 cara untuk menyimpan 2-byte data di memori:

  little-endian and big-endian => Host byte order
- Fungsi-fungsi untuk meng-konversi dari host byte order ke network byte order umumnya diawali dengan huruf:
  - ‘h’ : host byte order
  - ‘n’ : network byte order
  - ‘s’ : short (16 bit)
  - ‘l’ : long (32 bit)

Merubah Integer / Host ke Network Byte Order

- Listing 1.5

```
import socket

def convert_integer():
    data = 1234
    # 32-bit
    print ("Original: %s => Long  host byte order: %s, Network byte order:
%s" %(data, socket.ntohl(data), socket.htonl(data)))
    # 16-bit
    print ("Original: %s => Short  host byte order: %s, Network byte order:
%s" %(data, socket.ntohs(data), socket.htons(data)))

if __name__ == '__main__':
    convert_integer()
```

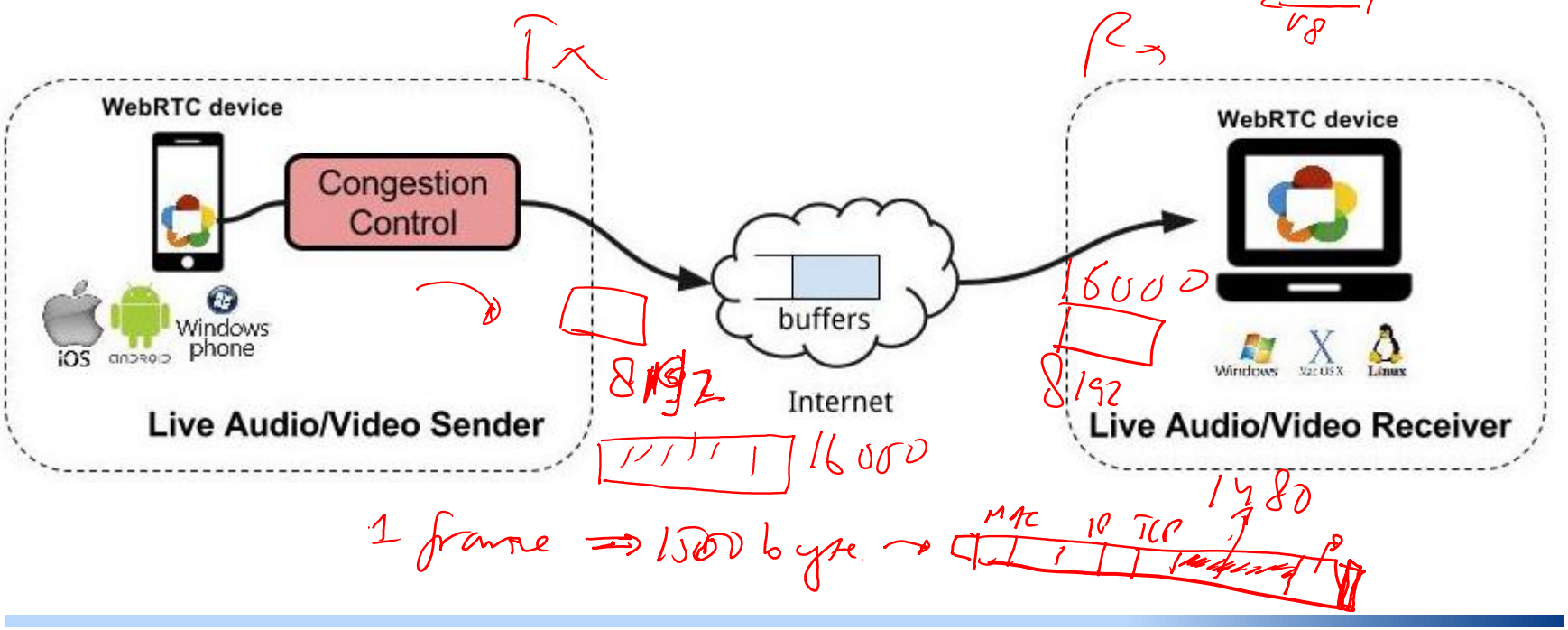
Rubah data = 1 dan lihat hasilnya untuk 16 bit

```
$ python 1_5_integer_conversion.py
Original: 1234 => Long  host byte order: 3523477504,
Network byte order: 3523477504
Original: 1234 => Short  host byte order: 53764,
Network byte order: 53764
```

LSB MSB

Memodifikasi ukuran buffer Tx dan Rx

- Tujuan buffer sebagai penyimpanan sementara adalah menghindari data congestion dari transfer (incoming/outgoing) dan membuat data dapat diambil sebagai satu kesatuan.



Memodifikasi ukuran buffer Tx dan Rx

Listing 1.8

```
import socket

SEND_BUF_SIZE = 4096
RECV_BUF_SIZE = 4096

def modify_buff_size():
    sock = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
    # Get the size of the socket's send buffer
    bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
    print ("Buffer size [Before]:%d" %bufsize)
    sock.setsockopt(socket.SOL_TCP,
                    socket.TCP_NODELAY, 1)
    sock.setsockopt(
            socket.SOL_SOCKET,
            socket.SO_SNDBUF,
            SEND_BUF_SIZE)
    sock.setsockopt(
            socket.SOL_SOCKET,
            socket.SO_RCVBUF,
            RECV_BUF_SIZE)
    bufsize = sock.getsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF)
    print ("Buffer size [After]:%d" %bufsize)

if __name__ == '__main__':
    modify_buff_size()
```

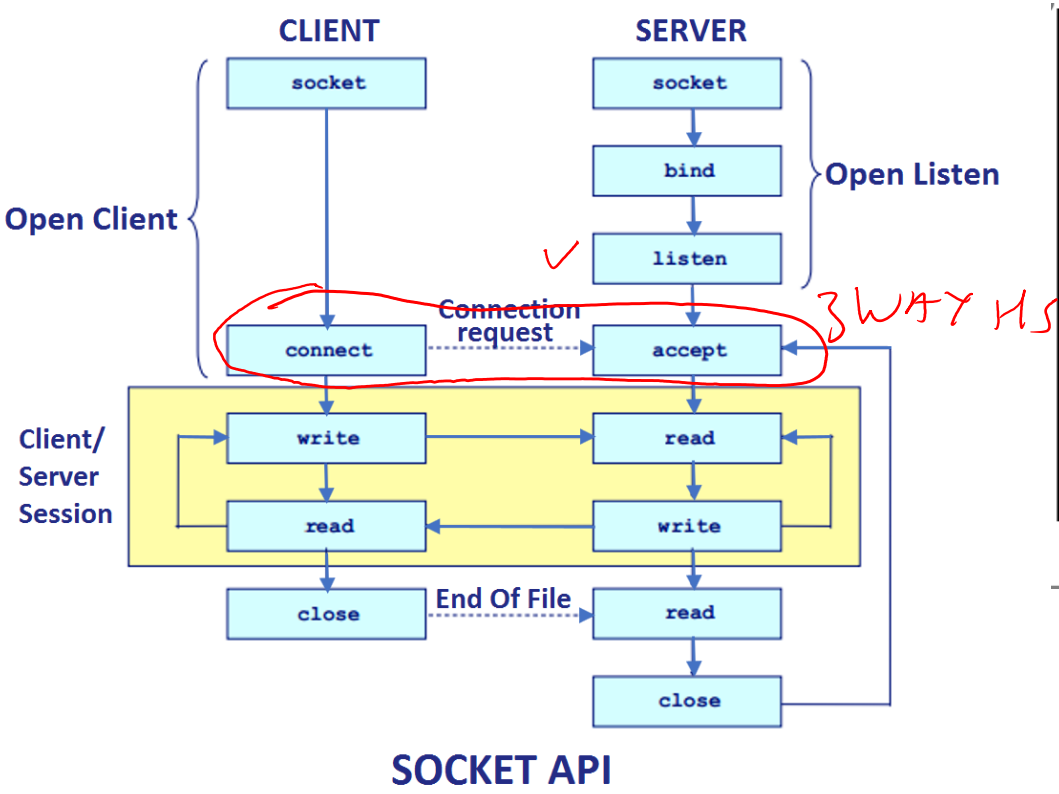
```
$ python 1_8_modify_buff_size.py
Buffer size [Before]:16384
Buffer size [After]:8192
```

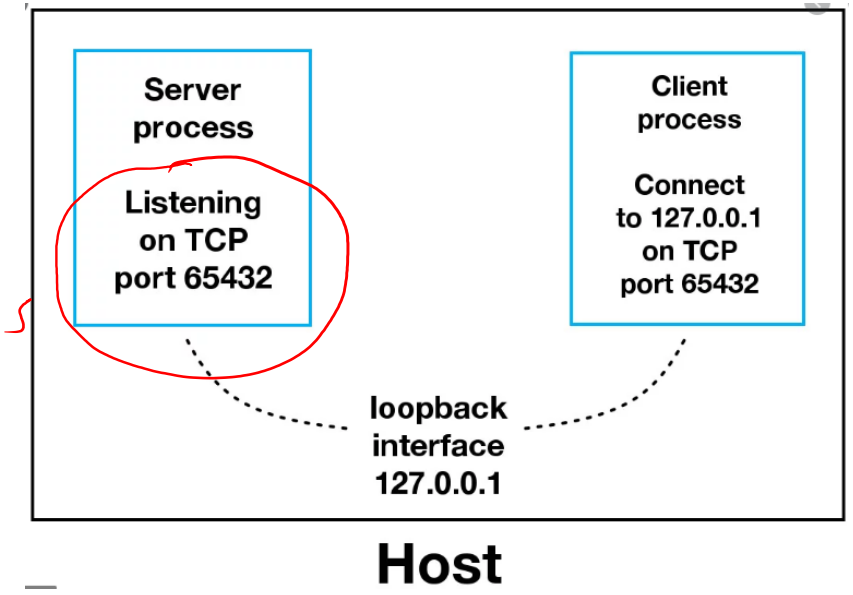
Membuat aplikasi berbasis TCP

- Koneksi client – server
  - Harus mengetahui IP dan port tujuan
  - Aplikasi tersebut berbasis TCP / UDP

Membuat aplikasi berbasis TCP

- Pembuatan socket API berbasis python

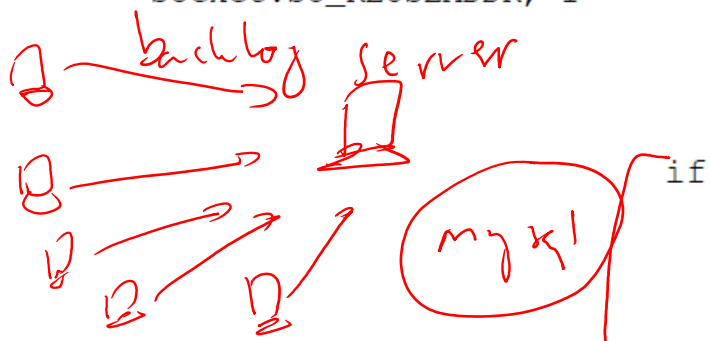




Dilakukan pada satu komputer dengan 2 terminal command prompt

Server (Listing 1-13)

```
import socket
import sys
import argparse
host = 'localhost'
data_payload = 2048
backlog = 5
def echo_server(port):
    """ A simple echo server """
    # Create a TCP socket
    sock = socket.socket(socket.AF_INET,
                         socket.SOCK_STREAM
    # Enable reuse address/port
    sock.setsockopt(socket.SOL_SOCKET,
                    socket.SO_REUSEADDR, 1
```



data_payload: max jml data yg diterima dalam satu koneksi.
backlog: jml antrian yg diijinkan

```
    # Bind the socket to the port
    server_address = (host, port)
    print ("Starting up echo server  on %s
                port %s" % server_address)
    sock.bind(server_address)
    # Listen to clients, backlog argument
      specifies the max no.
      of queued connections
    sock.listen(backlog)
    while True:
        print ("Waiting to receive message
                from client")
        client, address = sock.accept()
        data = client.recv(data_payload)
        if data:
            print ("Data: %s" %data)
            client.send(data)
            print ("sent %s bytes back
                    to %s" % (data, address))
        # end connection
        client.close()
if __name__ == '__main__':
    parser = argparse.ArgumentParser
    (description='Socket Server Example')
    parser.add_argument('--port',
    action="store", dest="port", type=int,
                        required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

Client

```
host = 'localhost'

def echo_client(port):
    """ A simple echo client """
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect the socket to the server
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address)
    # Send data
    try:
        # Send data
        message = "Test message. This will be
                   echoed"
        print ("Sending %s" % message)
        sock.sendall(message.encode('utf-8'))
        # Look for the response
        amount_received = 0
        amount_expected = len(message)
        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print ("Received: %s" % data)
    except socket.error as e:
        print ("Socket error: %s" %str(e))
    except Exception as e:
        print ("Other exception: %s" %str(e))
    finally:
        print ("Closing connection to the server")
        sock.close()
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser
             (description='Socket Server Example')
    parser.add_argument('--port', action="store",
dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

Perhatikan sock.sendall, format data harus diubah ke universal code 8 bit (ascii)

Aplikasi TCP

Client

Server

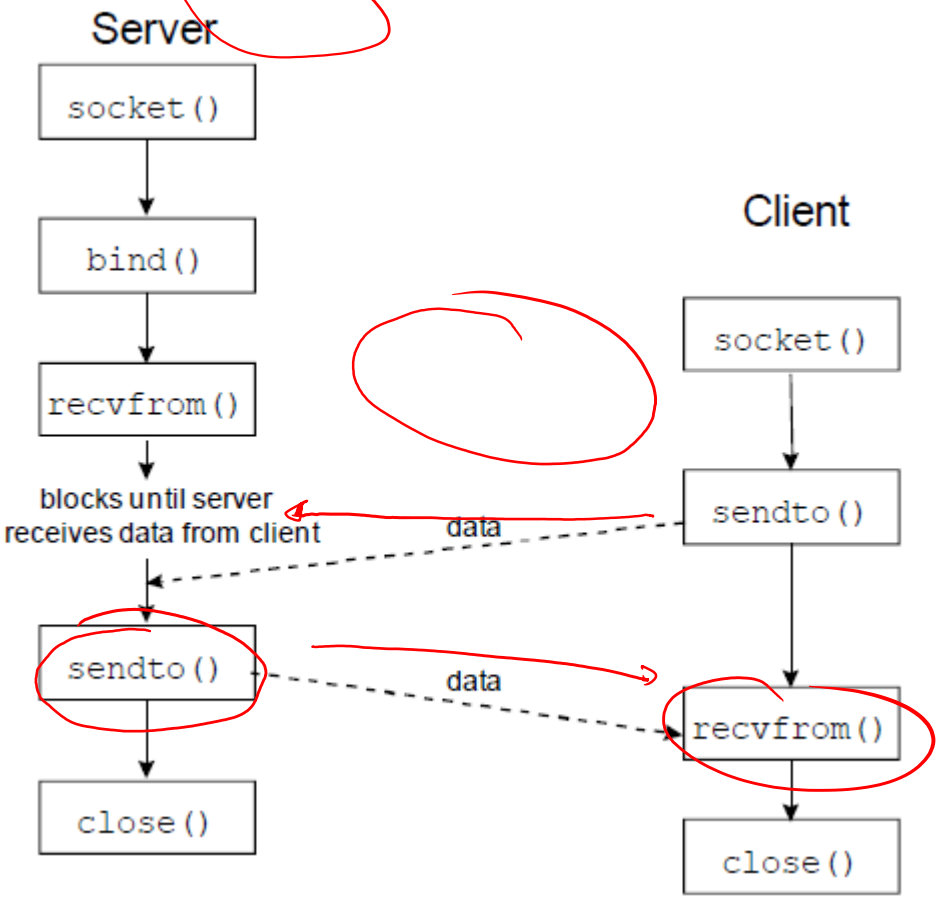
```
$ python 1_13a_echo_server.py --port=9900
Starting up echo server  on localhost port 9900
Waiting to receive message from client
```

```
$ python 1_13b_echo_client.py --port=9900
Connecting to localhost port 9900
Sending Test message. This will be echoed
Received: Test message. Th
Received: is will be echoe
Received: d
Closing connection to the server
```

```
Data: Test message. This will be echoed
sent Test message. This will be echoed
bytes back to ('127.0.0.1', 42961)
Waiting to receive message from client
```

Aplikasi UDP

- Dengan cara yang hampir sama, buat aplikasi berbasis UDP pada listing 1-14



TUGAS 1

- Buatlah aplikasi chatting berbasis TCP antara 2 komputer yang berjauhan (berbeda lokasi)
- Cobalah dahulu dengan menggunakan localhost seperti contoh sebelumnya

Client

IP: 172.10.3.4

Masukkan IP Server: 200.1.3.4
Masukkan port server: 9900
Koneksi berhasil
Client: Halo Server
Msg dari server: ini dari server
Client: Awas corona
Msg dari server: siap
…..dst

Server

IP: 200.1.3.4
Port: 9900

Menunggu koneksi

Msg dari client: Halo Server
Server: ini dari server
Msg dari client: Awas corona
Server: siap
….dst

TUGAS 2 & 3

- 2. Dari tugas 1, rubahlah backlog menjadi 1.
  Lakukan koneksi lebih dari 1 client ke server dan amati yang terjadi.
- 3. Dari tugas 1, rubahlah data_payload menjadi 8
  Amati proses pengiriman data bila:
  a. Data kurang dari 8
  b. Data lebih dari 8
  Lakukan hal yang sama bila data_payload menjadi 32