# Network Programming

**Pemrograman sistem dan jaringan**

Henry Saptono
2019

# Server Libraries

- **SocketServer** module provides basic server features
- Subclass the **TCPServer** and **UDPServer** classes to serve specific protocols
- Subclass **BaseRequestHandler**, overriding its handle() method, to handle requests
- Mix-in classes allow asynchronous handling via **ThreadingMixIn**

# Using SocketServer Module

- Server instance created with address and handler-class as arguments: **SocketServer.UDPServer(myaddr, MyHandler)**
- Each connection/transmission creates a request handler instance by calling the **handler-class***
- Created handler instance handles a message (UDP) or a complete client session (TCP)

# Example: echo-server

```
# echo_server.py
import socketserver
class MyTCPSocketHandler(socketserver.BaseRequestHandler):
    def handle(self):
        self.data = self.request.recv(1024).strip()
        print("{} wrote:".format(self.client_address[0]))
        print(self.data)
        self.request.sendall(self.data.upper())
```

# Example: echo-server

```
if __name__ == "__main__":
    HOST, PORT = "localhost", 9999
    server = socketserver.TCPServer((HOST, PORT),
MyTCPSocketHandler)
    server.serve_forever()
```

# Example: echo-client

```
# echo_client.py
import socket, sys
HOST, PORT = "localhost", 9999
data = " ".join(sys.argv[1:])
print('data = %s' %(data) )
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    # connect to server
    sock.connect((HOST, PORT))
    # send data
    sock.sendall(data.encode())
    # receive data back from the server
    received = str(sock.recv(1024))
```

# Example: echo-client

```
finally:
    # shut down
    sock.close()

print("Sent:%s" %(data) )
print("Received: %s" %(received) )
```

# Asynchronous Request Handling

- In the previous, we used TCPServer which process requests synchronously. That means each request must be completed before the next request can be started. This isn't suitable if each request takes a long time to complete, because it requires a lot of computation, or because it returns a lot of data which the client is slow to process.
- The solution to this is to create a separate process or thread to handle each request. The **ForkingMixIn** and **ThreadingMixIn** mix-in classes can be used to support asynchronous behavior.

# asynchronous echo-server

```
# async.py

import socketserver

class ThreadedTCPRequestHandler(socketserver.BaseRequestHandler):

    def handle(self):

        data = str(self.request.recv(1024))

        response = "INI DATA BALASAN : %s"%(data)

        self.request.sendall(response.encode())

class ThreadedTCPServer(socketserver.ThreadingMixIn,
socketserver.TCPServer):

    daemon_threads = True

    allow_reuse_address = True

    def __init__(self, server_address, RequestHandlerClass):

        socketserver.TCPServer.__init__(self, server_address,
RequestHandlerClass)
```

# asynchronous echo-server

```
if __name__ == "__main__":
    # port 0 means to select an arbitrary unused port
    HOST, PORT = "localhost", 9999
    server = ThreadedTCPServer((HOST, PORT),
ThreadedTCPRequestHandler)
    server.serve_forever()
```