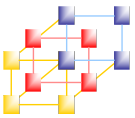
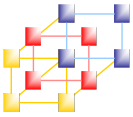
Unit 10
Network Programming Using Python



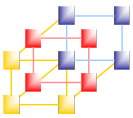
Sockets

- The endpoints of a network connection
- Each host has a unique IP address
- Each service runs on a specific port
- Each connection is maintained on both ends by a socket
- Sockets API allows us to send and receive data
  - Programming Languages provide modules and classes to use this API



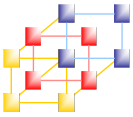
Socket Types

- Stream Sockets (SOCK_STREAM)
  - Connection-oriented
  - Use TCP
- Datagram Sockets (SOCK_DGRAM)
  - Connectionless
  - Use UDP
- Other types
  - E.g. Raw Sockets

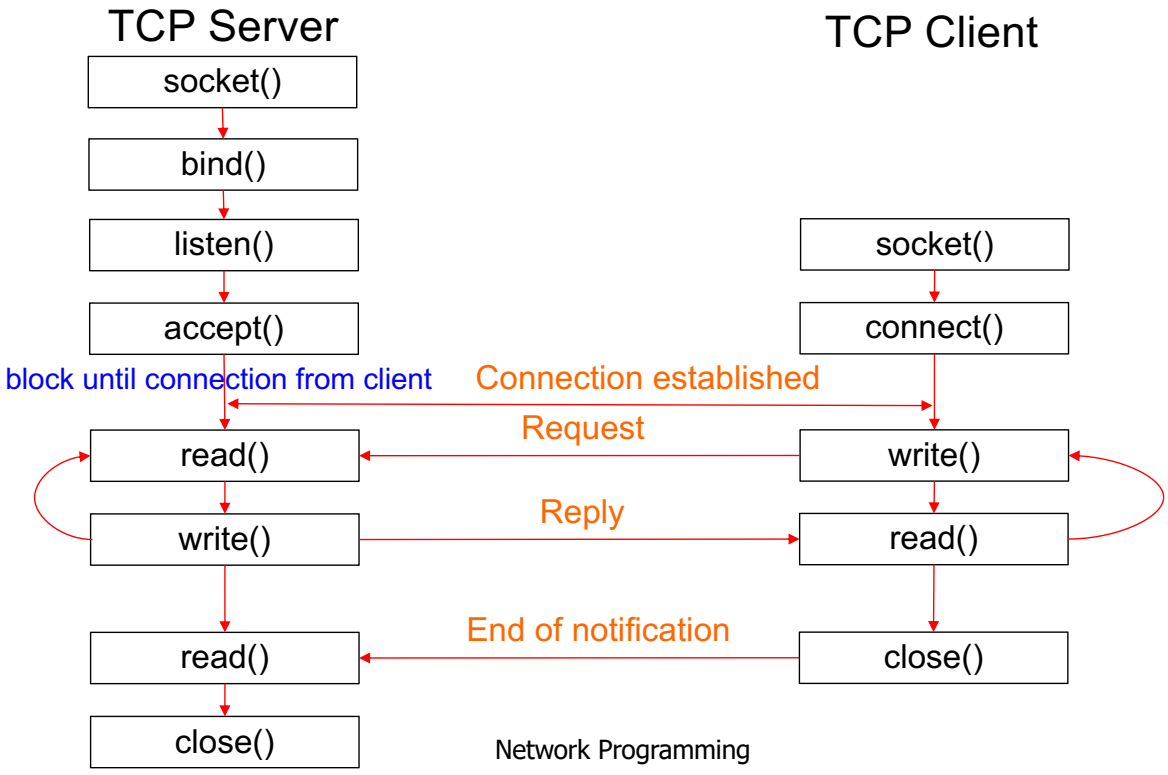


Creating sockets

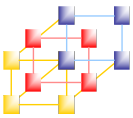
- socket.socket(family, type, proto, fileno)
  - Create a new socket using the given address family, socket type and protocol number
- Address family
  - AF_INET (the default), AF_INET6, AF_UNIX, ...
- Socket type
  - SOCK_STREAM (the default), SOCK_DGRAM, SOCK_RAW
- The protocol number is usually zero and may be omitted
- Default fileno is None



TCP 網路應用程式模型

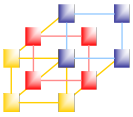


1-TCPServer.py, 1-TCPClient.py



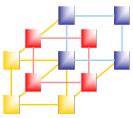
Stream Sockets Functions (1/3)

- socket.bind(*address*)
  - Bind the socket to *address*
  - *Ex: socket.bind(('192.168.1.1', 80)), '' is any interface*
- socket.listen([*backlog*])
  - Enable a server to accept connections.
  - If *backlog* is specified, it must be at least 0; it specifies the number of unaccepted connections that the system will allow before refusing new connections
- socket.close()
  - Mark the socket closed



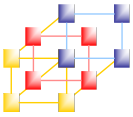
Stream Sockets Functions (2/3)

- socket.accept()
  - Accept a connection
  - The return value is a pair (conn, address)
    - *conn* is a *new* socket object usable to send and receive data on the connection
    - *address* is the address bound to the socket on the other end of the connection
- socket.connect(*address*)
  - Connect to a remote socket at *address*
  - e.g. socket.connect((host, port))



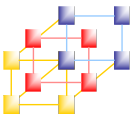
Stream Sockets Functions (3/3)

- socket.recv(*bufsize*[, *flags*])
  - Receive data from the socket
  - The return value is a bytes object representing the data received
  - The maximum amount of data to be received at once is specified by *bufsize*.
  - optional argument *flags*; it defaults to zero
- socket.send(*bytes*[, *flags*])
  - Send data to the socket
  - The optional *flags* argument has the same meaning as for recv() above
  - Returns the number of bytes sent
  - e.g. socket.send(b'Hello, World!')



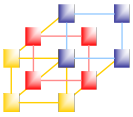
Datagram Sockets Functions

- socket.recvfrom(*bufsize*[, *flags*])
  - Receive data from the socket
  - The return value is a pair (bytes, address)
    - *bytes* is a bytes object representing the data received
    - *address* is the address of the socket sending the data.
- socket.sendto(*bytes*, *address*)
  - Send data to the socket.
  - The socket should not be connected to a remote socket, since the destination socket is specified by *address*



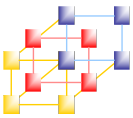
Other Socket Functions(1/2)

- socket.gethostname()
  - returns a string containing host name of the machine
- socket.gethostbyname(hostname)
  - Translates hostname to ip address
- socket.gethostbyaddr(ip_address)
  - Translates ip address to host name
- socket.getpeername()
  - Return the remote address to which the socket is connected
- socket.getsockname()
  - Return the socket's own address



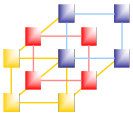
Other Socket Functions(2/2)

- socket.htonl(*x*), socket.htons(*x*)
  - Convert 32-bit / 16-bit positive integers from host to network byte order
- socket.ntohl(*x*), socket.ntohs(*x*)
  - Convert 32-bit / 16-bit positive integers from network to host byte order
- socket.inet_aton(ip_string)
  - Convert an IPv4 address from dotted-quad string format (for example, '123.45.67.89') to 32-bit packed binary format, as a bytes object four characters in length
- socket.inet_ntoa(packed_ip)
  - Convert a 32-bit packed IPv4 address (a bytes-like object four bytes in length) to its standard dotted-quad string representation (for example, '123.45.67.89')



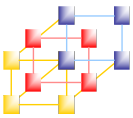
Non-blocking Socket

- Instead of timeouts, can set non-blocking
  - `>>> s.setblocking(False)`
- Future send(),recv() operations will raise an exception if the operation would have blocked



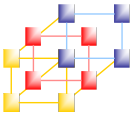
select — Waiting for I/O completion

- r, w, x = select.select(*rlist*, *wlist*, *xlist*[, *timeout*])
  - rlist: wait until ready for reading
  - wlist: wait until ready for writing
  - xlist: wait for an "exceptional condition"
  - Empty iterables are allowed
  - The optional *timeout* argument specifies a time-out as a floating point number in seconds.
    - When the *timeout* argument is omitted the function blocks until at least one file descriptor is ready
    - A time-out value of zero specifies a poll and never blocks.
  - The return value is a triple of lists of objects that are ready: subsets of the first three arguments.



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
- Server instance created with address and handler-class as arguments:
  `socketserver.TCPServer(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)



Writing a handle() Method

- `self.request` gives client access
  - (string, socket) for UDP servers
  - Connected socket for TCP servers
- `self.client_address` is remote address
- `self.server` is server instance
- TCP servers should handle a complete client session