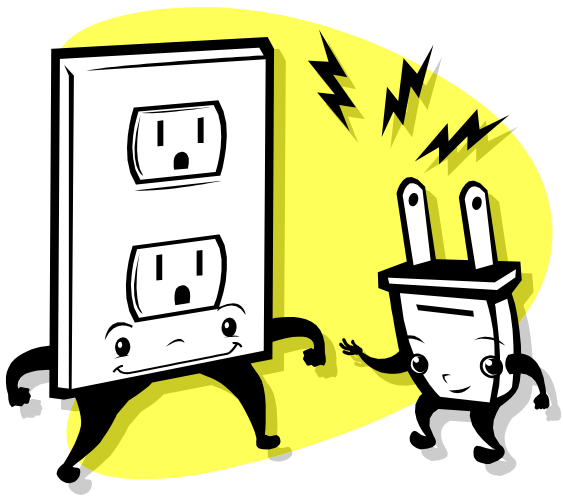
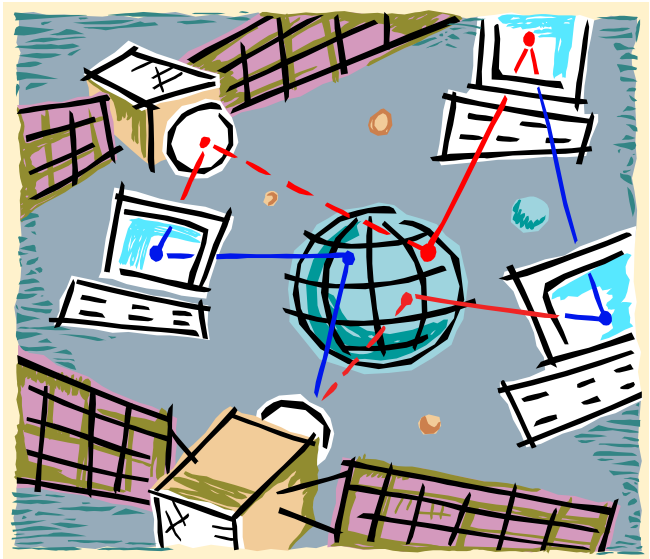
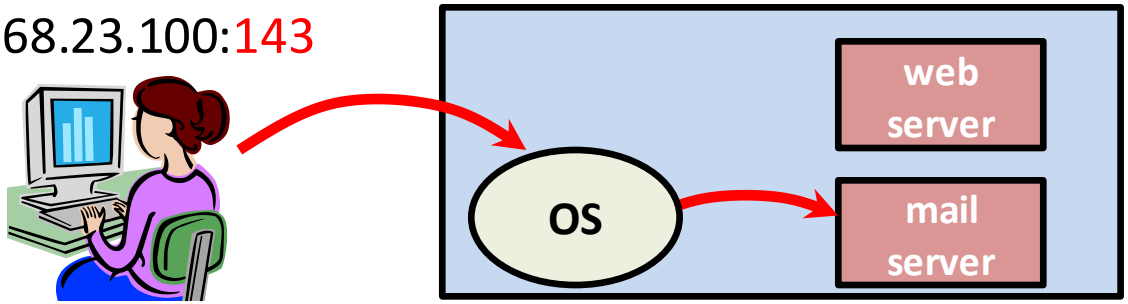
Socket programming

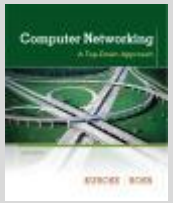




192.168.23.100:143



Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley



Overview

- Chapter 2: Application Layer
  - Many familiar services operate here
    - Web, email, Skype, P2P file sharing
  - Socket programming
- Socket programming
  - In Python
  - UDP
  - TCP

application
transport
network
link
physical

Communication - division of labor

- Network
  - Gets data to the destination host
  - Uses destination IP address
- Operating system
  - Forwards data to a given "silo" based on port #
  - E.g. All port 80 request go the web server
- Application
  - Actually reads and writes to socket
  - Implement the application specific magic

Port numbers

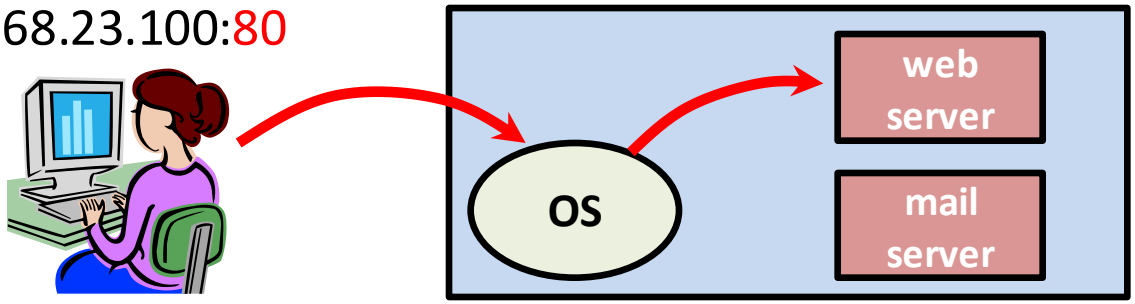
- Popular applications have known ports
  - Server uses a well-known port, 0 - 1023
  - Client uses a free temporary port, 1024 - 65535
    - Assigned by the operating system

Port	Service
21	File transfer protocol (FTP)
22	Secure shell (SSH)
23	Telenet
25	Simple mail transfer protocol (SMTP)
53	Domain name system (DNS)
80	Hypertext transfer protocol (HTTP)
110	Post office protocol (POP)
143	Internet message access protocol (IMAP)
443	HTTP secure (HTTPS)

Use of port number

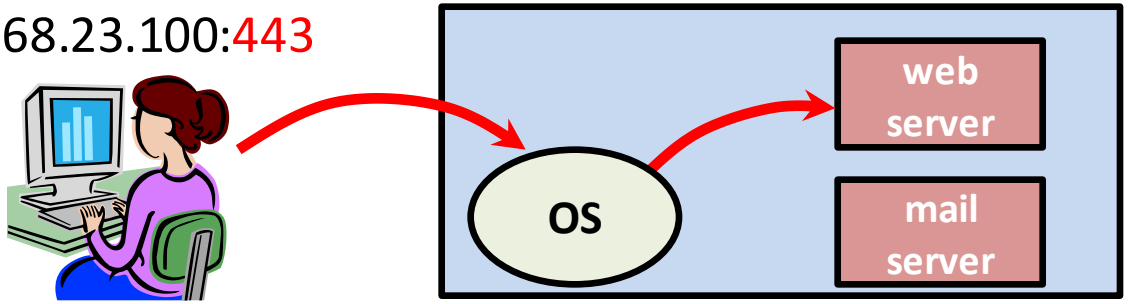
Requesting a non-secure web page

192.168.23.100:80



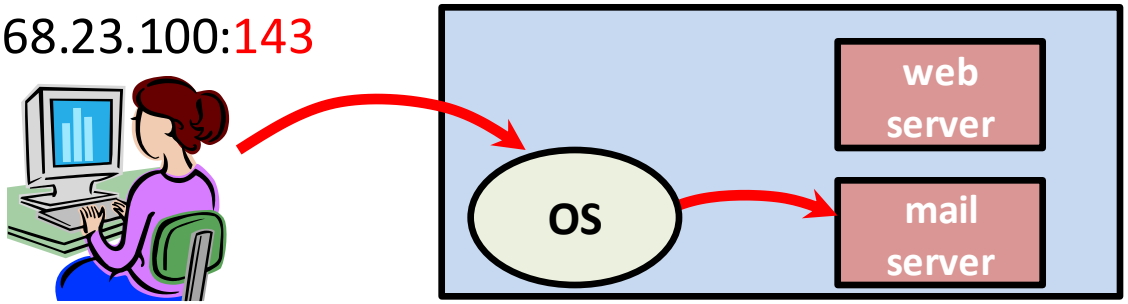
Requesting a secure web page

192.168.23.100:443

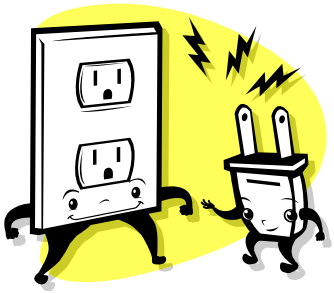


Requesting new email messages

192.168.23.100:143



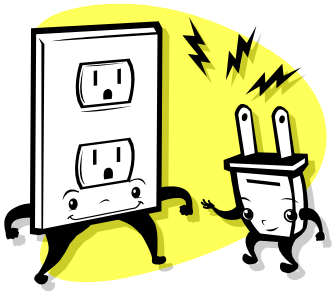
Sockets



- Socket API (applications programming interface)
  - Originally in Berkeley Unix
    - Thus: Berkeley sockets, BSD sockets
  - De facto standard in all operating systems
  - C :
    - `socket(), bind(), connect(), listen(), accept(), send(), recv(), sendto(), recvfrom(), close()`
  - Python:
    - `socket(), bind(), connect(), listen(), accept(), send(), recv(), sentto(), recvfrom(), close()`

Sockets

- Socket API (applications programming interface)
  - Java classes
    - `Socket`
    - `ServerSocket`
    - `InputStreamReader`
    - `BufferedReader`
    - `PrintWriter`



High-level process

```
// Fire up connection
// to the server
socket()
connect()

// Exchange data
while (!done)
{
  send()
  recv()
}

// Shutdown
close()
```

Client program

```
// Initial socket setup
socket()
bind()
listen()
while (1)
{
  // Wait for new caller
  accept()

  // Exchange data
  while (!done)
  {
    recv()
    send()
  }
  // Disconnect
  close()
}
```

Server program

Socket programming: UDP

UDP: no "connection" between client & server

- No handshaking before sending data
- Sender explicitly attaches IP destination address and port # to each packet
- Receiver extracts sender IP address and port # from received packet

UDP: data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Socket interaction: UDP

server (running on serverIP)

create socket, port = x:

```
serverSocket =
socket(AF_INET,SOCK_DGRAM)
```

read datagram from `serverSocket`

write reply to `serverSocket` specifying client address, port number

client

create socket:

```
clientSocket =
socket(AF_INET,SOCK_DGRAM)
```

Create datagram with server IP and port=x; send datagram via `clientSocket`

read datagram from `clientSocket`

close `clientSocket`

Python UDP client

```
from socket import *

serverName = 'hostname'
serverPort = 12000

clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)

message = raw_input('Input lowercase sentence:')

clientSocket.sendto(message, (serverName, serverPort))

modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

print modifiedMessage
clientSocket.close()
```

Include the socket library

IPv4

UDP transport

Send off data to server+port

Wait for response

Python UDP server

```
from socket import *

serverPort = 12000

serverSocket = socket(AF_INET, SOCK_DGRAM)

serverSocket.bind(('', serverPort))

print "The server is ready to receive"

while 1:
    message, clientAddress = serverSocket.recvfrom(2048)

    modifiedMessage = message.upper()

    serverSocket.sendto(modifiedMessage, clientAddress)
```

Create IPv4 UDP socket

Listen on specific port #

Blocks until message arrives
Gets data as well as who sent it

Send processed data back

TCP socket programming

Client must contact server

- Server process must first be running
- Server must have created socket (door) that welcomes client's contact

Client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *When client creates socket:* client TCP establishes connection to server TCP

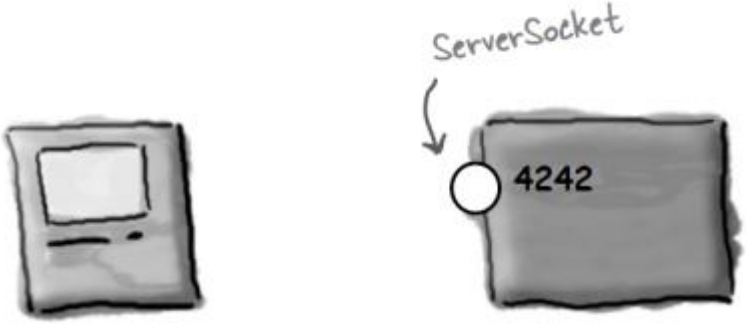
- When contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - Allows server to talk with multiple clients
  - Source port numbers used to distinguish clients (more in Ch. 3)

Application viewpoint:

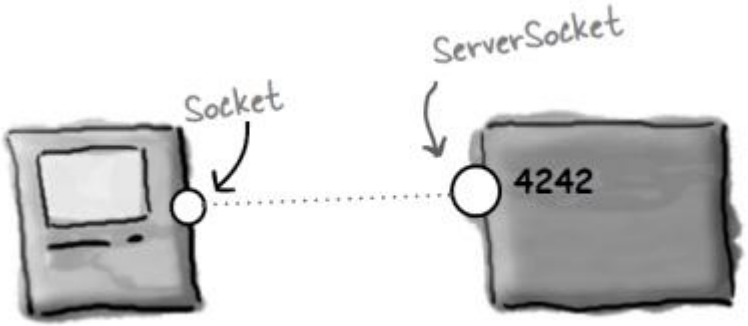
TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

TCP: connection process

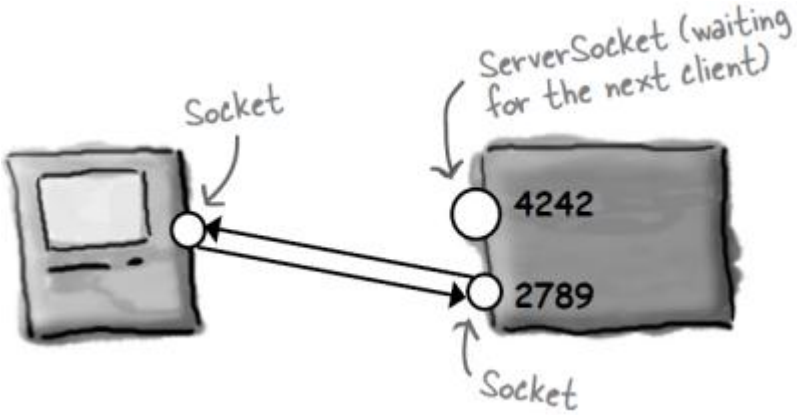
1. Server program starts up.
2. Starts listening on port 4242.
3. OS sends all inbound connection requests to 4242 to the server program.



4. Client program starts up
5. Requests connection to server IP address on port 4242.



6. Server establishes a socket connection to client, but on different port (2789)
7. Server can listen for new clients on the 4242 port number.



Python TCP client

```
from socket import *

serverName = 'servername'
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))

sentence = raw_input('Input lowercase sentence:')

clientSocket.send(sentence)

modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence

clientSocket.close()
```

TCP transport

Make the connection

Send some data (no need to say to who!)

Close the TCP connection

Python TCP server

```
from socket import *
serverPort = 12000

serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))

serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
     connectionSocket, addr = serverSocket.accept()

     sentence = connectionSocket.recv(1024)
     capitalizedSentence = sentence.upper()
     connectionSocket.send(capitalizedSentence)

     connectionSocket.close()
```

Create TCP "welcoming" socket

Begin listens for incoming TCP connections

Block until somebody comes knocking, socket created on return

Close the client's TCP connection, not the welcoming socket

Summary

- Socket programming
  - Berkley socket API
  - UDP
    - Just fire off messages towards destinations
    - They may not get there
    - They may arrive out of order
  - TCP
    - Establish a reliable byte-stream between two hosts
    - Data always arrives
    - Data arrives in order