CS 372 Lecture #15

Socket programming

- Writing application layer protocols
- *sockets* API

Note: Many of the lecture slides are based on presentations that accompany *Computer Networking: A Top Down Approach,* 6th edition, by Jim Kurose & Keith Ross, Addison-Wesley, 2013.

Socket

- OS-controlled interface (a "door")
- A logical port (implemented in software)
- Created by and associated with an application on local host

- An application process uses a socket to send / receive messages to / from another application process

Socket programming

Two socket types for two transport services:

- *UDP:* "unreliable" datagram
- *TCP:* reliable, byte stream

Application Example:

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

Socket programming with *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: transmitted data may be lost or received out-of-order

Application viewpoint:

UDP provides unreliable transfer of groups of bytes (“datagrams”) between client and server

Example application: UDP client

Python UDPClient

include Python's socket library

```
from socket import *
serverName = 'hostname'
serverPort = 12000
```

create UDP socket for server

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

get user keyboard input

```
message = raw_input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket

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

read reply characters from socket into string

```
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
```

print out received string and close socket

```
print modifiedMessage
clientSocket.close()
```

Example application: UDP server

Python UDPServer

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

```
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)
```

Socket programming with *TCP*

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

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

Example application: TCP client

Python TCPClient

create TCP socket for server, remote port 12000

No need to attach server name, port

```
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()
```

Example application: TCP server

Python TCPServer

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but noaddress as in UDP)

close connection to this client (but *not* welcoming socket)

```
 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()
```

Project #1

- See definition on course website.
- Programming using Socket API
  - Implemented in *C* or *Python* or *C++* or *Java.*
    - See references in the project description
  - **Well-modularized and well-documented**.
  - Run on an OSU *engr* server.
    - Specify your testing machine in the program documentation.
  - Don't hard-code any directories, since they might be inaccessible to the graders.
  - Cite any references and credit any collaborators.

Summary Lecture #15

- Transport layer
  - UDP
  - TCP

- Socket programming
  - SOCK_DGRAM for UDP
    - bind, sendto, recvfrom, close
  - SOCK_STREAM for TCP
    - connect, bind, listen, accept, send, recv, close