Redes de Computadores

LEIC-A, MEIC-A

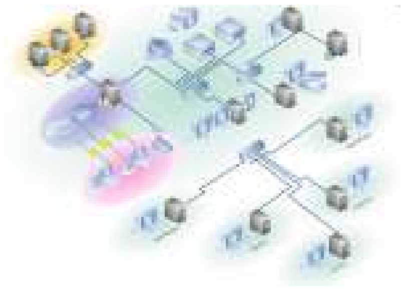
2 – Application Layer Socket Programming

Prof. Paulo Lobato Correia

IST, DEEC – Área Científica de Telecomunicações

Objectives

➔ Socket Programming with TCP and UDP



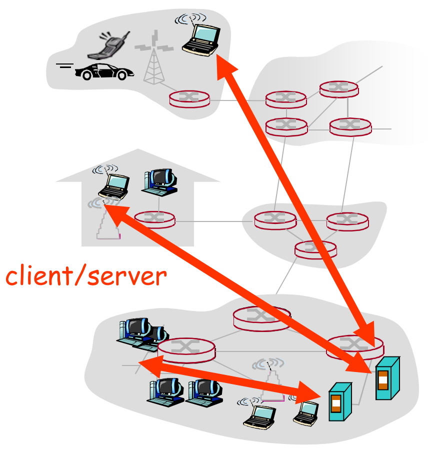
Client-Server Architecture

Server:

- ☑ Always-on host;
- ☑ Permanent IP address;
- ☑ Server farms for scaling.

Clients:

- ☑ Communicate with server;
- ☑ May be intermittently connected;
- ☑ May have dynamic IP addresses;
- ☑ Do not communicate directly with each other.



Socket Programming

Goal: Learn how to build client/server applications that communicate using sockets

Socket API:

- ➔ Introduced in BSD4.1 UNIX, 1981;
- ➔ Client/Server paradigm;
- ➔ Two types of transport service via socket API:
  - ☑ Unreliable datagram;
  - ☑ Reliable, byte stream-oriented .

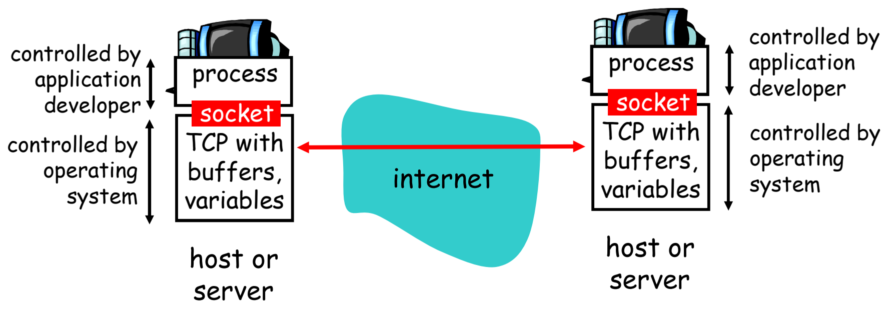
socket

A host, local, application-created, OS-controlled interface (a "door") into/from which application process can send/receive messages to/from another application process.

Socket Programming using TCP

Socket: a door between application process and the end-end-transport protocol (UDP or TCP);

TCP service: reliable transfer of **bytes** from one process to another.


controlled by application developer
controlled by operating system
process
socket
TCP with buffers, variables
host or server
internet
process
socket
TCP with buffers, variables
host or server
controlled by application developer
controlled by operating system

Socket Programming using TCP

Client must contact server:

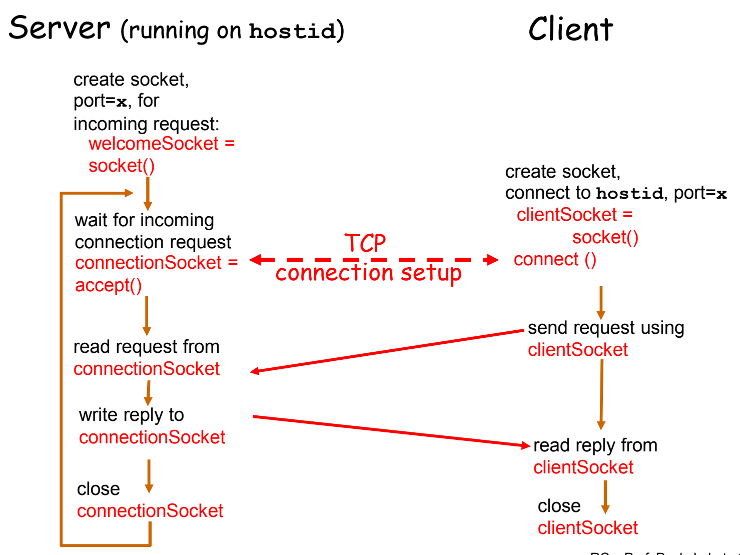
- Server process must first be running;
- Server must have created socket (door) to welcome client contacts.

Client contacts server by:

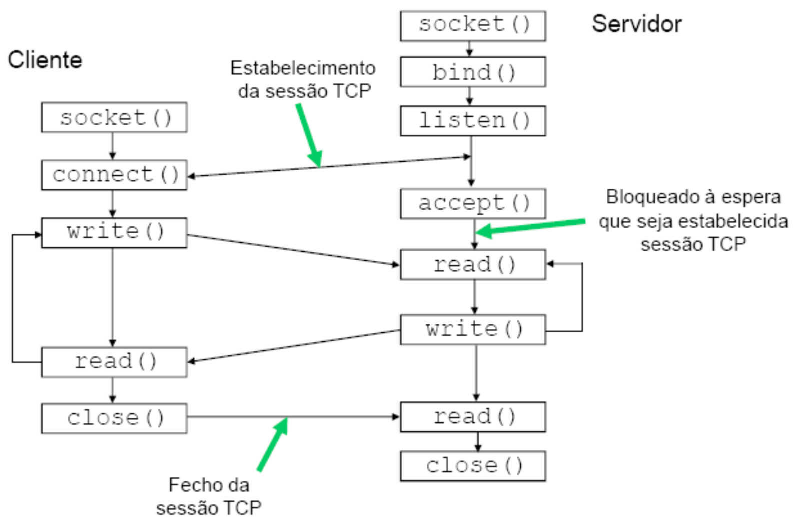
- Creating client-local TCP socket;
- Specifying IP address, port number of server process;
- When client creates socket:
  - Client TCP establishes connection to server TCP.
- When contacted by client, server TCP creates new socket for communication between server and client:
  - Allows server to talk with multiple clients;
  - Source port numbers are used to distinguish clients.

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

Client/server Socket Interaction: TCP

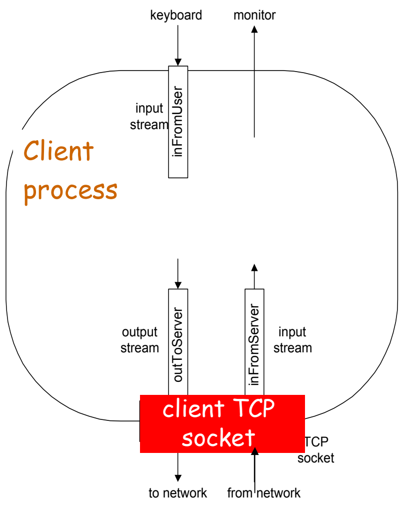

Server (running on hostid)
Client
create socket, port=x, for incoming request:
welcomeSocket = socket()
wait for incoming connection request
connectionSocket = accept()
TCP connection setup
create socket, connect to hostid, port=x
clientSocket = socket()
connect ()
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

Client/server Socket Interaction: TCP



Sockets: Streams

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



Socket Programming with UDP

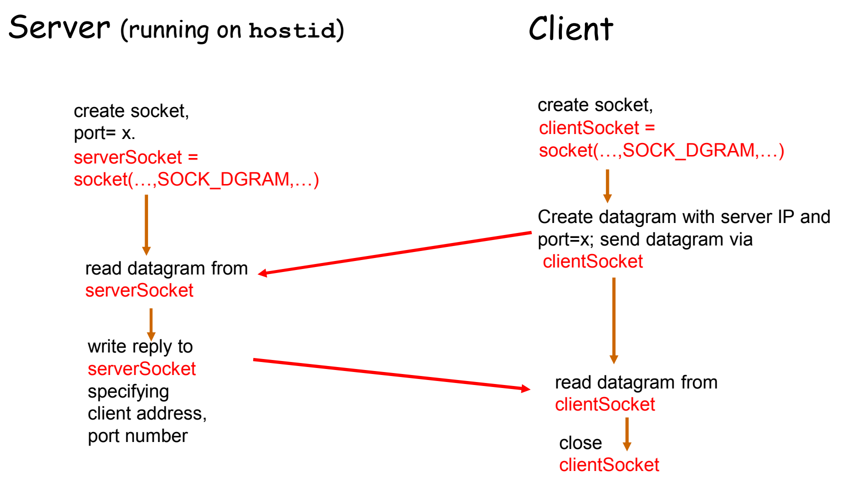
UDP – no "connection" between client and server:

- No handshaking;
- Sender explicitly includes IP address and port of destination to each packet;
- Server must extract IP address and port of sender from the received packet.

UDP – transmitted data may be received out of order, or lost!

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

Client/server Socket Interaction: UDP


Server (running on **hostid**)

Client

create socket,
port= x.
serverSocket =
socket(…,SOCK_DGRAM,…)

read datagram from
serverSocket

write reply to
serverSocket
specifying
client address,
port number

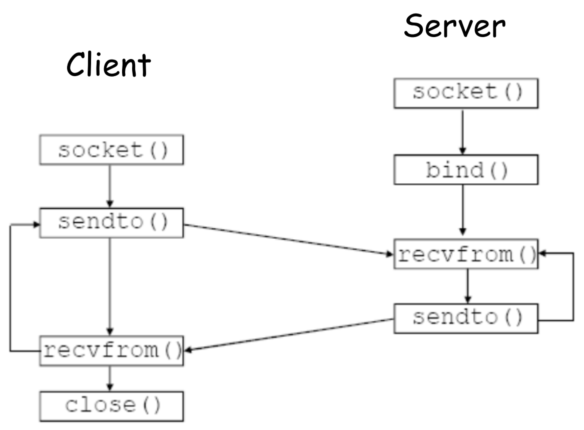
create socket,
clientSocket =
socket(…,SOCK_DGRAM,…)

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

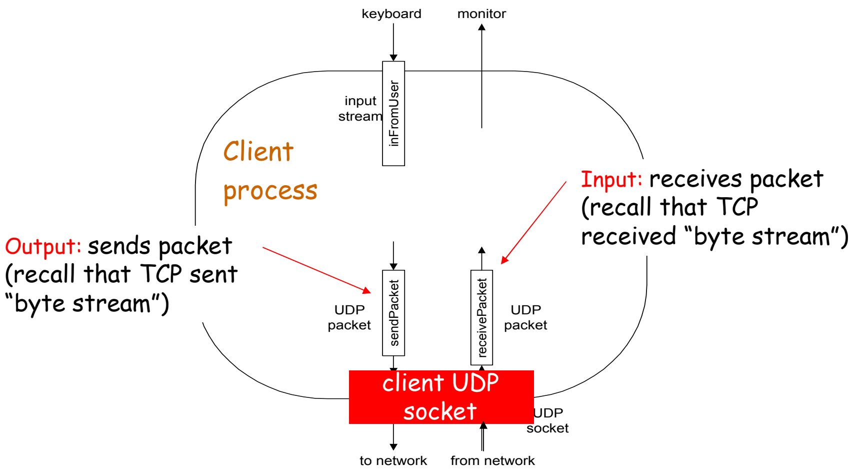
read datagram from
clientSocket

close
clientSocket

Client/server Socket Interaction: UDP



Client/server Socket Interaction: UDP



Socket Programming: TCP vs UDP

TCP:
- `read()` and `write()`;
- Byte stream (and <u>no byte is lost</u>);
- Bytes read with `read()` may correspond to several `write()`;
- Bytes written with `write()` may need to be read with several `read()`;

UDP:
- `sendto()` and `recvfrom()`;
- Preserves boundary between messages;
- Each message read with `recvfrom()` corresponds to a single `sendto()`;
- A message may be lost.