JAVA SOCKET PROGRAMMING

September 2018

WHAT IS A SOCKET?

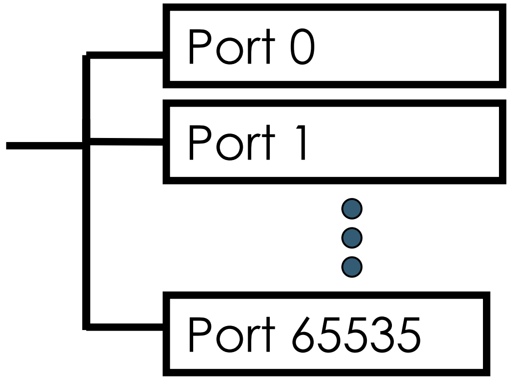
Socket

- The combination of an IP address and a port number. (RFC 793 original TCP specification)
- The name of the Berkeley-derived *application programming interfaces* (APIs) for applications using TCP/IP protocols.
- Two types
  - **Stream socket** : reliable two-way connected communication streams
  - **Datagram socket**

Socket pair

- Specified the two end points that uniquely identifies each TCP connection in an internet.
- 4-tuple: (client IP address, client port number, server IP address, server port number)

PORTS



Each host has 65,536 ports

Some ports are *reserved for specific apps*

- 20,21: FTP
- 22: ssh
- 80: HTTP
- see RFC 1700 (about 2000 ports are reserved)

A socket provides an interface to send data to/from the network through a port

CLIENT-SERVER APPLICATIONS

Implementation of a protocol standard defined in an RFC. (FTP, HTTP, SMTP…)

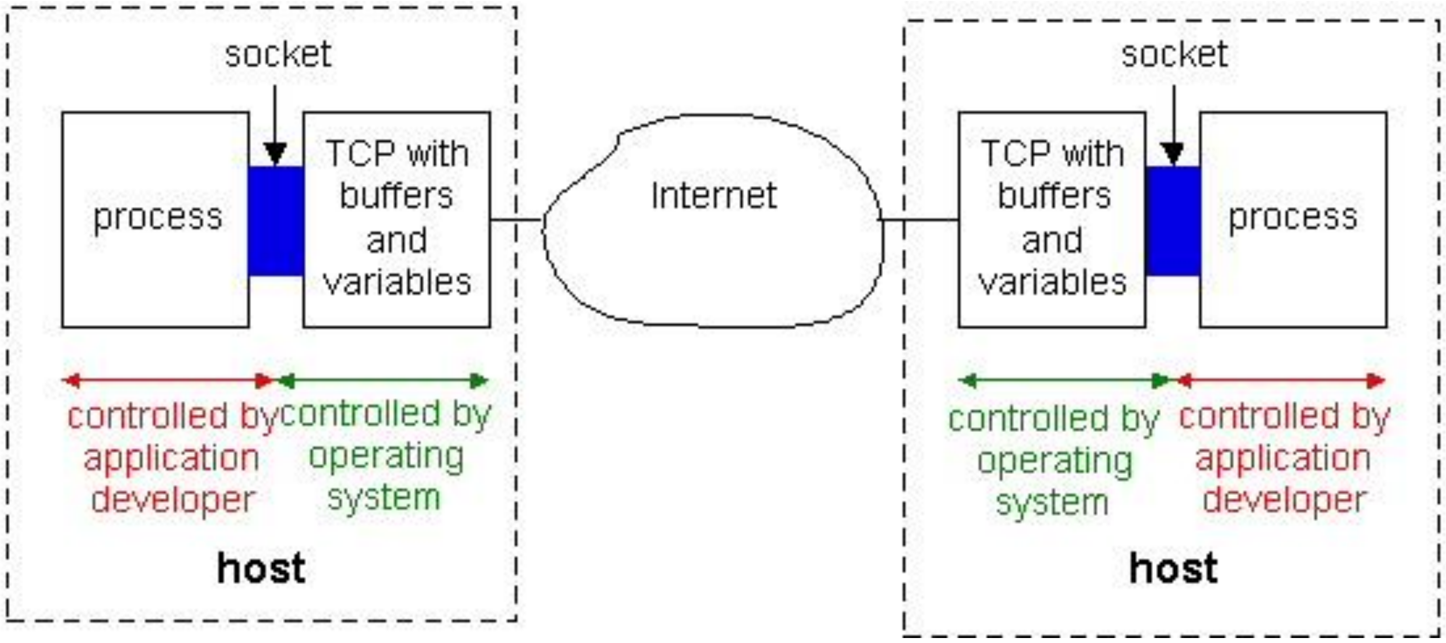
- Conform to the rules dictated by the RFC.
- Should use the port number associated with the protocol.

Proprietary client-server application.

- A single developer(or team) creates both client and server program.
- The developer has complete control.
- Must be careful not to use one of the well-known port number defined in the RFCs.

* well-known port number : managed by the Internet Assigned Numbers Authority(IANA)

SOCKET PROGRAMMING WITH TCP


socket
process
TCP with buffers and variables
Internet
controlled by application developer
controlled by operating system
host
socket
TCP with buffers and variables
process
controlled by operating system
controlled by application developer
host

The application developer has the ability to fix a few TCP parameters, such as maximum buffer and maximum segment sizes.

SOCKETS FOR SERVER AND CLIENT

Server

- Welcoming socket
  - Welcomes some initial contact from a client.
- Connection socket
  - Is created at initial contact of client.
  - New socket that is dedicated to the particular client.

Client

- Client socket
  - Initiate a TCP connection to the server by creating a socket object. (Three-way handshake)
  - Specify the address of the server process, namely, the IP address of the server and the port number of the process.

SOCKET FUNCTIONAL CALLS

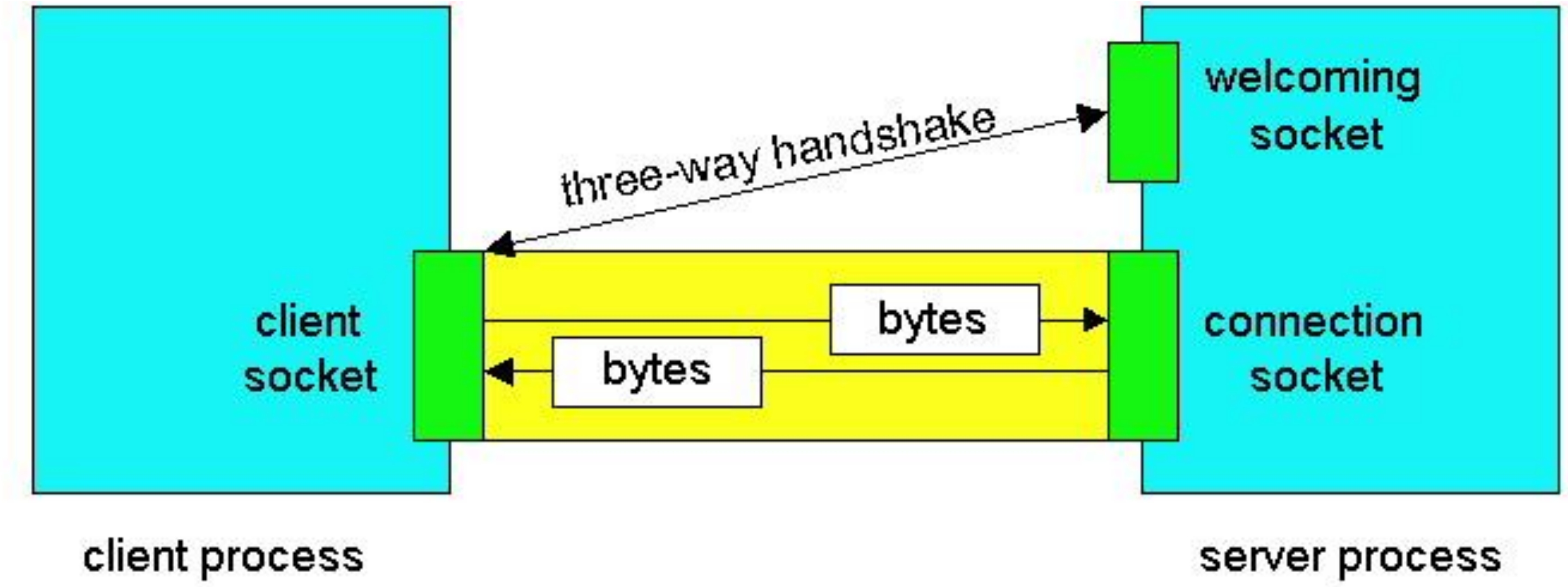
socket(): Create a socket
bind(): bind a socket to a local IP address and port #

listen(): passively waiting for connections
connect(): initiating connection to another socket
accept(): accept a new connection

write(): write data to a socket
read(): read data from a socket
sendto(): send a datagram to another UDP socket
recvfrom(): read a datagram from a UDP socket

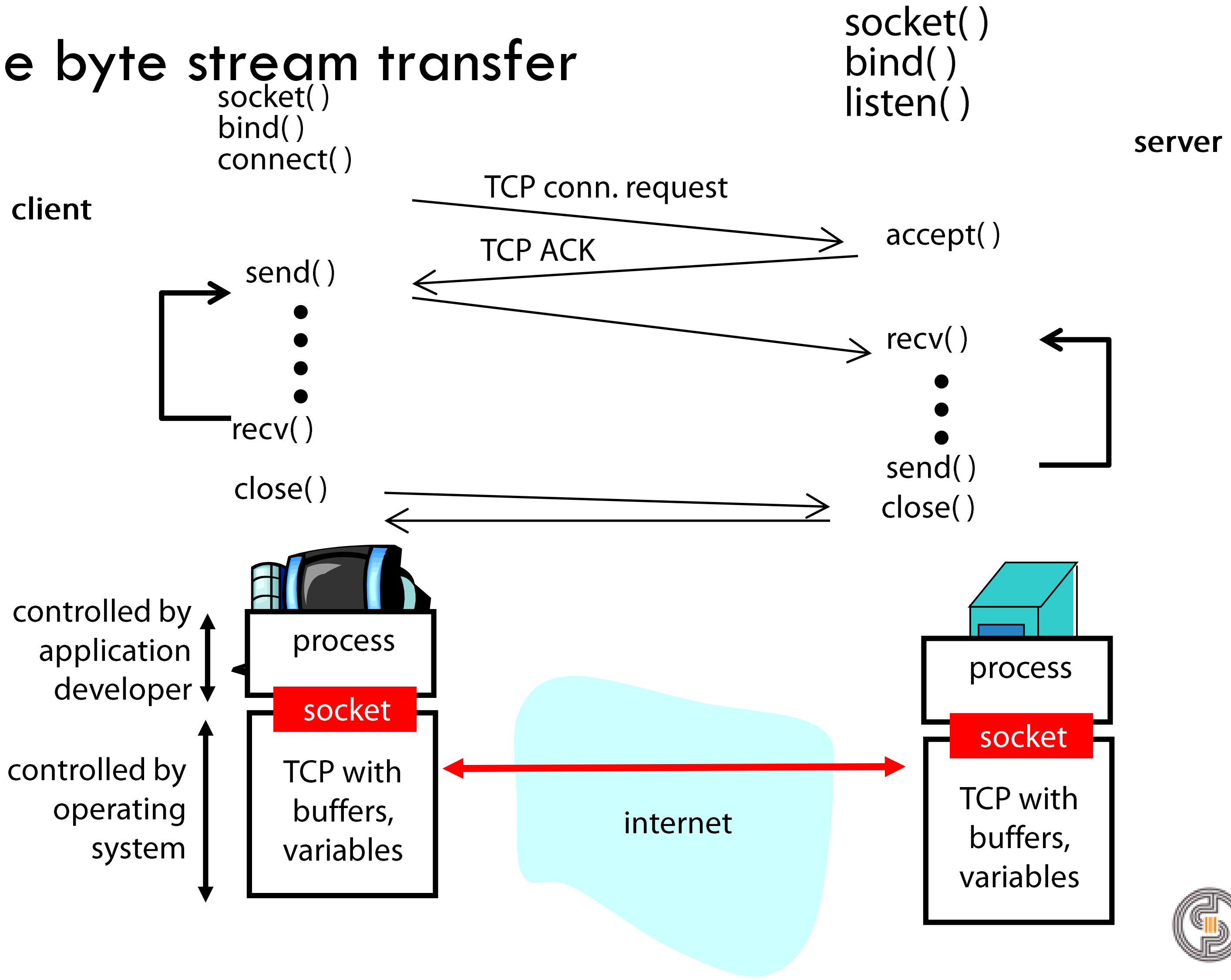
close(): close a socket

SOCKETS


welcoming
socket
three-way handshake
client
socket
bytes
bytes
connection
socket
client process
server process

SOCKET-PROGRAMMING USING TCP

reliable byte stream transfer



SOCKET PROGRAMMING WITH TCP

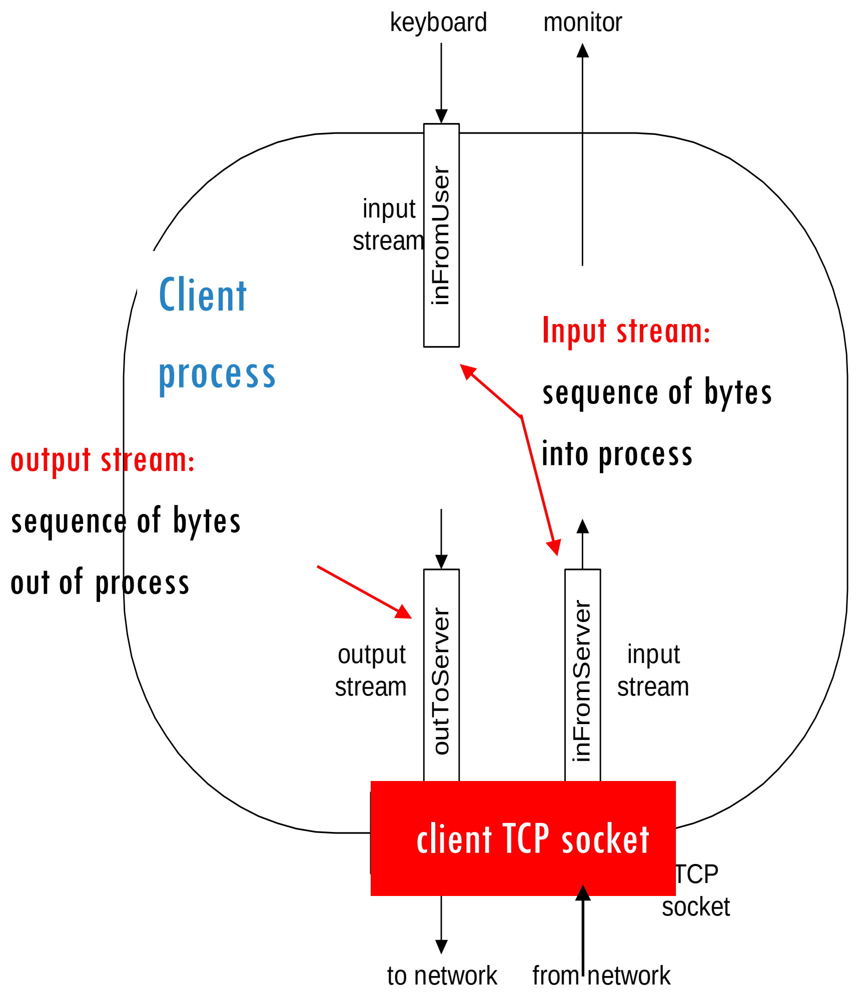
Example client-server app:

client reads line from standard input, sends to server via socket

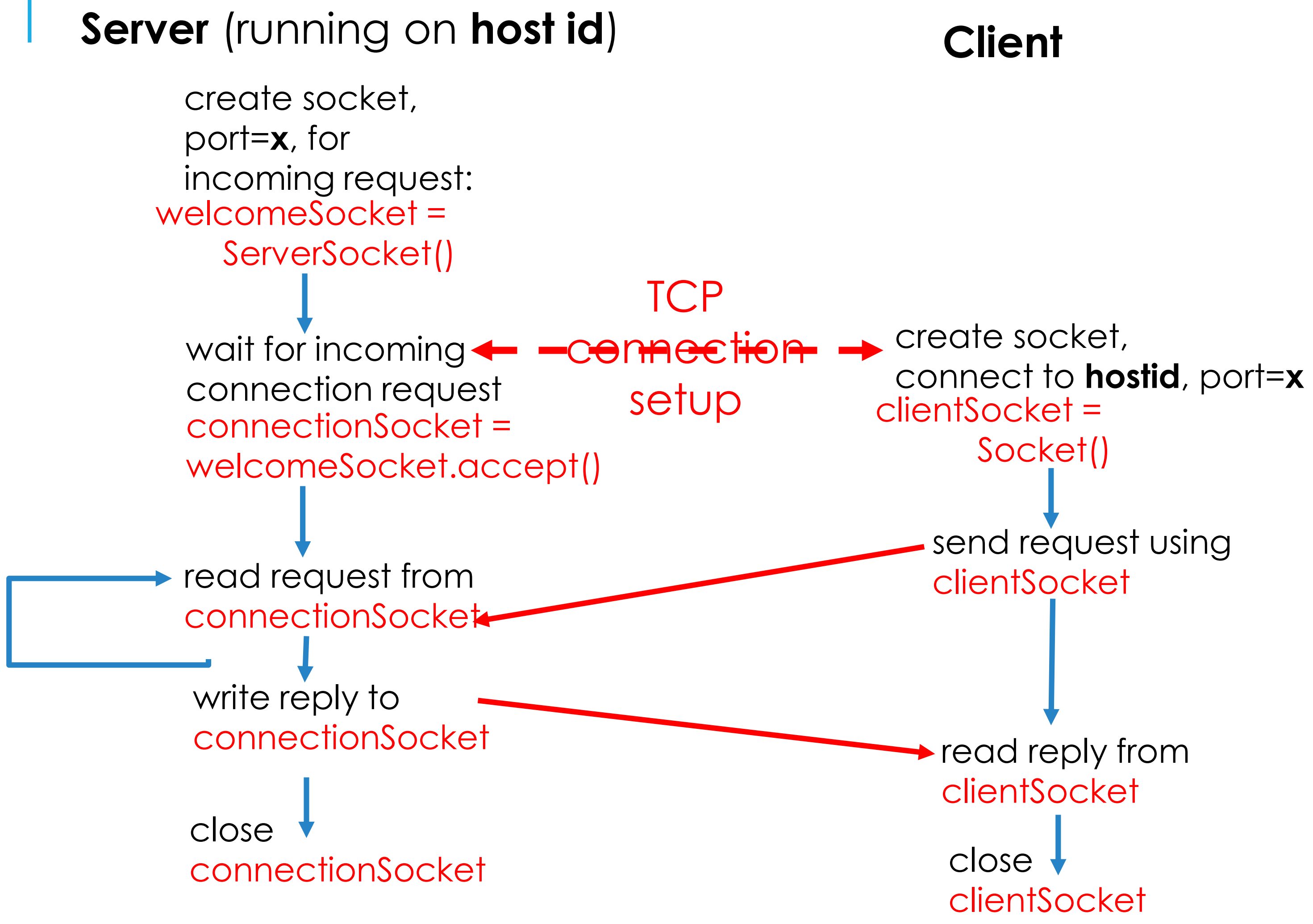
server reads line from socket

server converts line to uppercase, sends back to client

client reads, prints modified line from socket



CLIENT/SERVER SOCKET INTERACTION: TCP


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

JAVA SOCKETS PROGRAMMING

The package java.net provides support for sockets programming (and more).

Typically you import everything defined in this package with:

```
import java.net.*;
```

DIFFERENT CLASSES

`InetAddress`

`Socket`

`ServerSocket`

`DatagramSocket`

`DatagramPacket`

INETADDRESS CLASS

static methods you can use to create new `InetAddress` objects.

- `getByName(String host)`
- `getAllByName(String host)`
- `getLocalHost()`

```
InetAddress x = InetAddress.getByName(
                    “upssitech.eu”);
```

throws **`UnknownHostException`**

```
try {

  InetAddress a = InetAddress.getByName(hostname);

  System.out.println(hostname + ":" +
      a.getHostAddress());

} catch (UnknownHostException e) {

  System.out.println("No address found for " +
      hostname);

}
```

SOCKET CLASS

Corresponds to active TCP sockets only!

- client sockets
- socket returned by `accept`();

Passive sockets are supported by a different class:

- `ServerSocket`

UDP sockets are supported by

- `DatagramSocket`

SOCKET PROGRAMMING WITH UDP

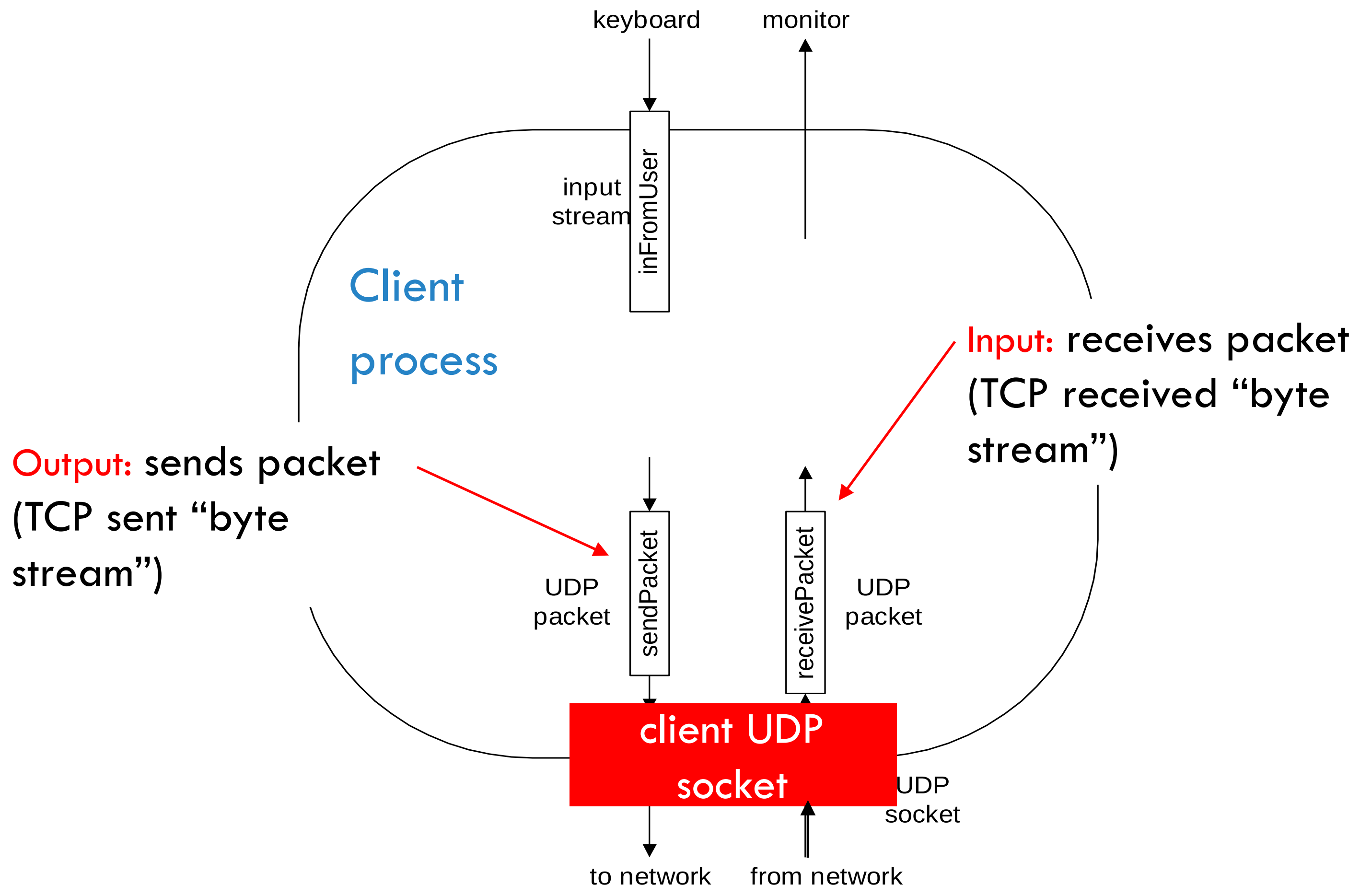
UDP

- Connectionless and unreliable service.
- There isn’t an initial handshaking phase.
- Doesn’t have a pipe.
- transmitted data may be received out of order, or lost

Socket Programming with UDP

- No need for a welcoming socket.
- No streams are attached to the sockets.
- the sending hosts creates “packets” by attaching the IP destination address and port number to each batch of bytes.
- The receiving process must unravel to received packet to obtain the packet’s information bytes.

EXAMPLE: JAVA CLIENT (UDP)



CONCURRENT SERVER

Servers usually need to handle a new connection request while processing previous requests.

- Most TCP servers are designed to be **concurrent**.

When a new connection request arrives at a server, the server accepts and invokes a new process (with a thread) to handle the new client.