Socket programming

From theory to implementation

Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

What you will know after this tutorial

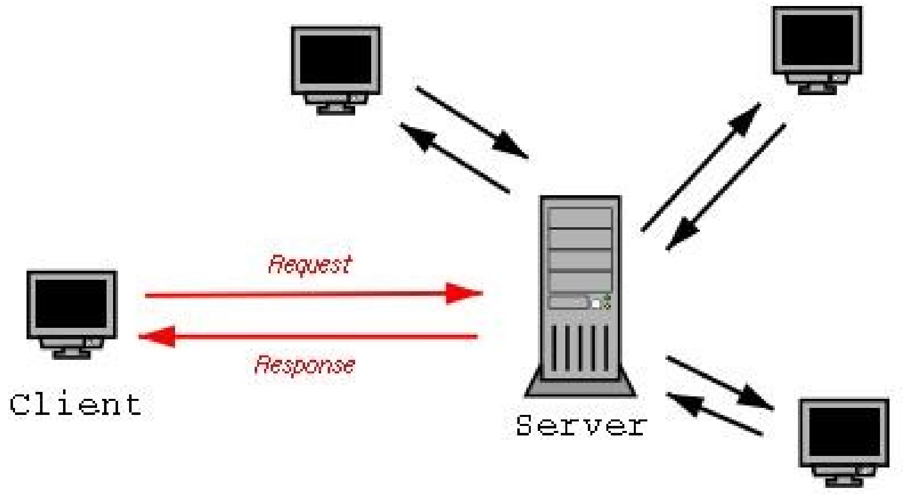
- How computers communicate on the network
  - What are sockets? Why would you use them?
- How to *practically* use sockets to enable the communication
  - How do you do this in C?

Socket programming is simple

TU/e

Client/sever model

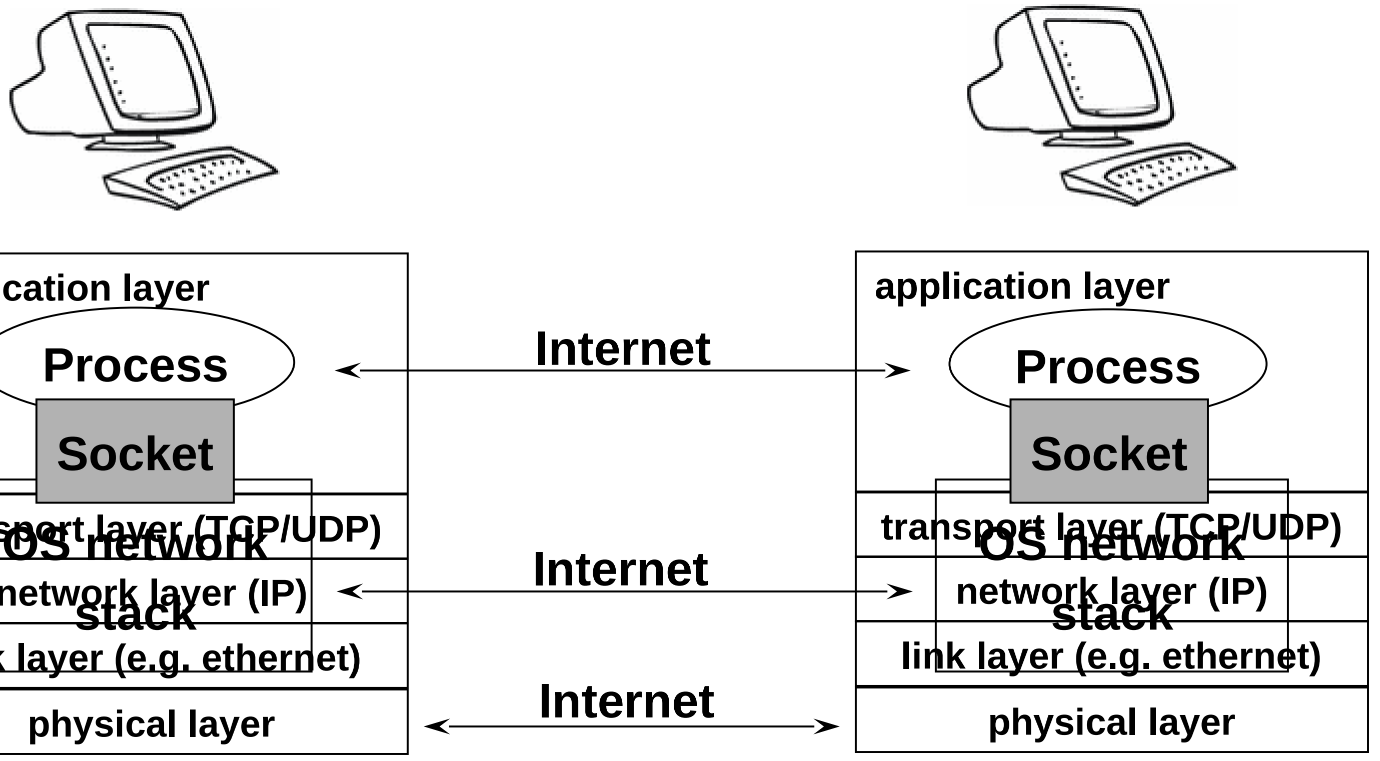
- Client asks (*request*) – server provides (*response*)
- Typically: single server - multiple clients
- The server does not need to know *anything* about the client
  - even that it exists
- The client should always now *something* about the server
  - at least where it is located



Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

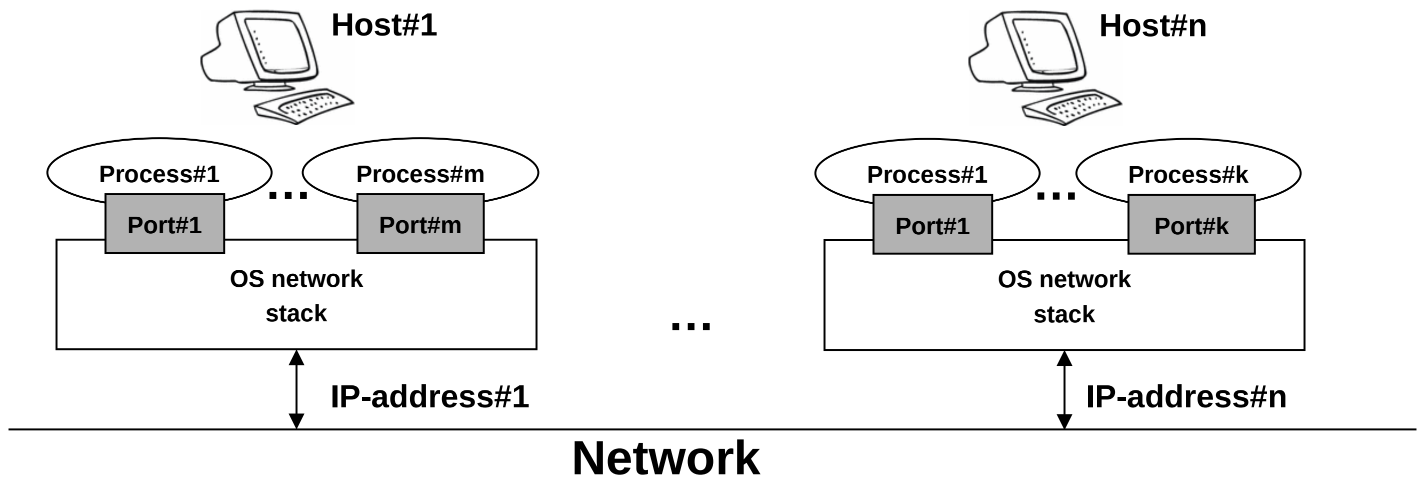
SAN

Sockets as means for inter-process communication (IPC)



Addressing server

- Address the machine on the network
  - By IP address (127.0.0.1 – localhost)
- Address the process
  - By the “port”-number
- The pair of *IP-address* + *port* – makes up a “*socket-address*”



Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

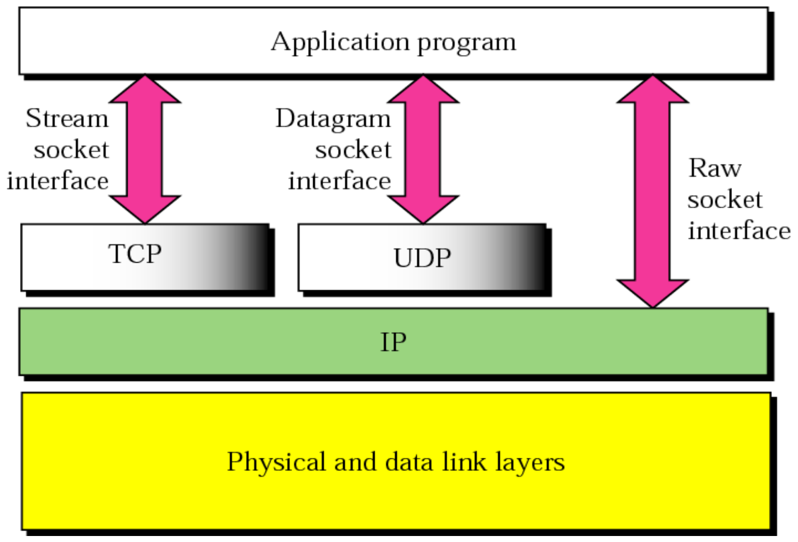
Usage of port-numbers

- Standard applications use predefined port-numbers
  - 21 - ftp
  - 23 - telnet
  - 80 - http
  - 110 - pop3 (email)
  - …
- Other applications should choose between 1024 and 65535
  - 4662 – eMule
  - …

Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

Socket types

- Datagram socket – using UDP
  - Not sequenced
  - Not reliable
  - Not unduplicated
  - Connectionless
  - Border preserving
- Stream socket – using TCP
  - Sequenced
  - Reliable
  - Unduplicated
  - Connection-oriented
  - Not border preserving
- Raw and others (extracurricular)



Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

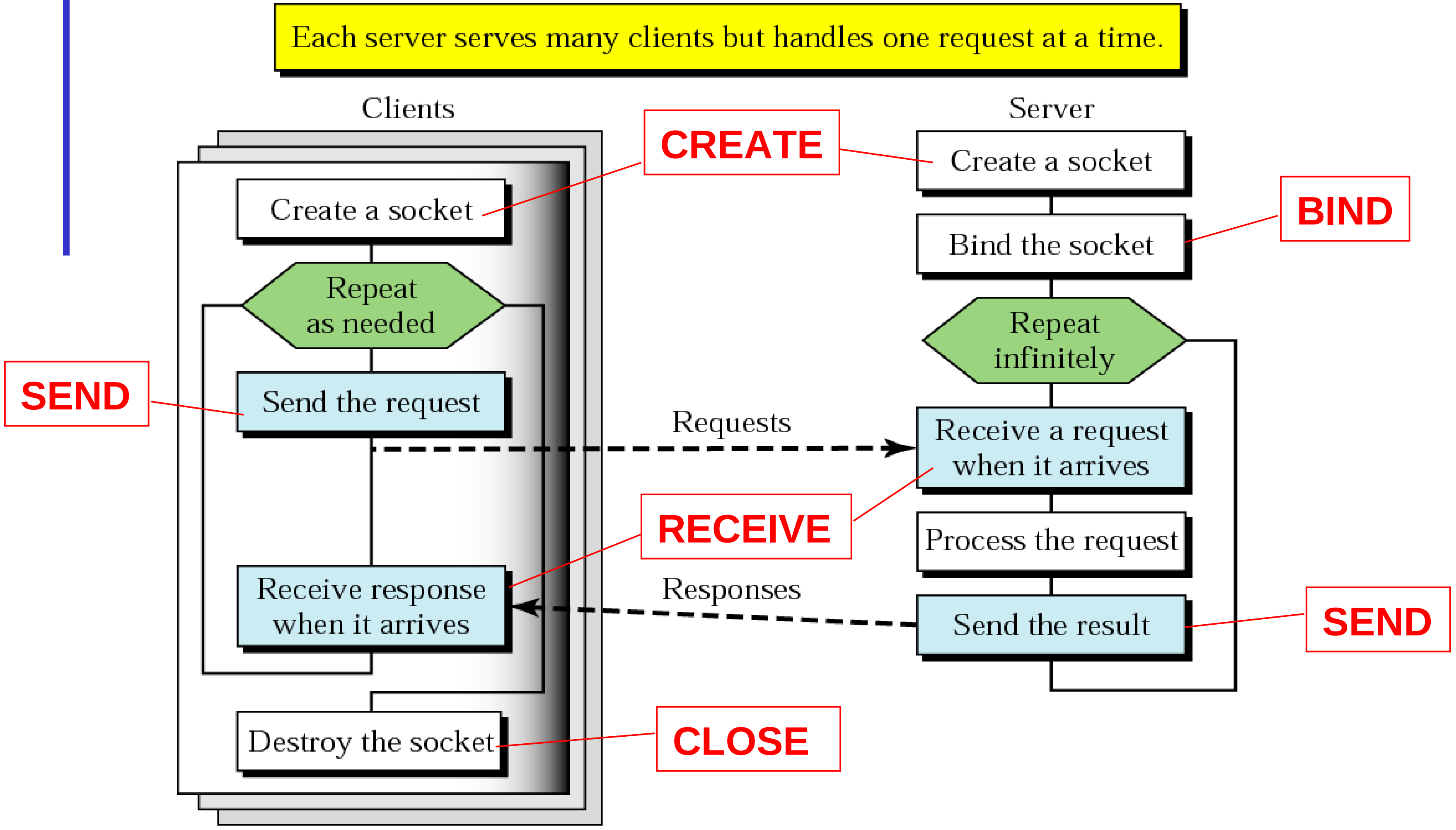
Let's summarize

- What is a *socket*?
  - A *socket* is a communication end-point between two processes (which are normally located on different machines)
- How to address a socket?
  - By a *socket address*: IP-address + port number
- Where are sockets used?
  - Any network application: ftp-applications, web-browsers, web-servers, telnet severs/terminals, email-server/clients, P2P-applications (Kazaa, eMule etc), chat-clients (ICQ, Messanger) and many more
- Which programming language do I need to program sockets?
  - Almost any language will do

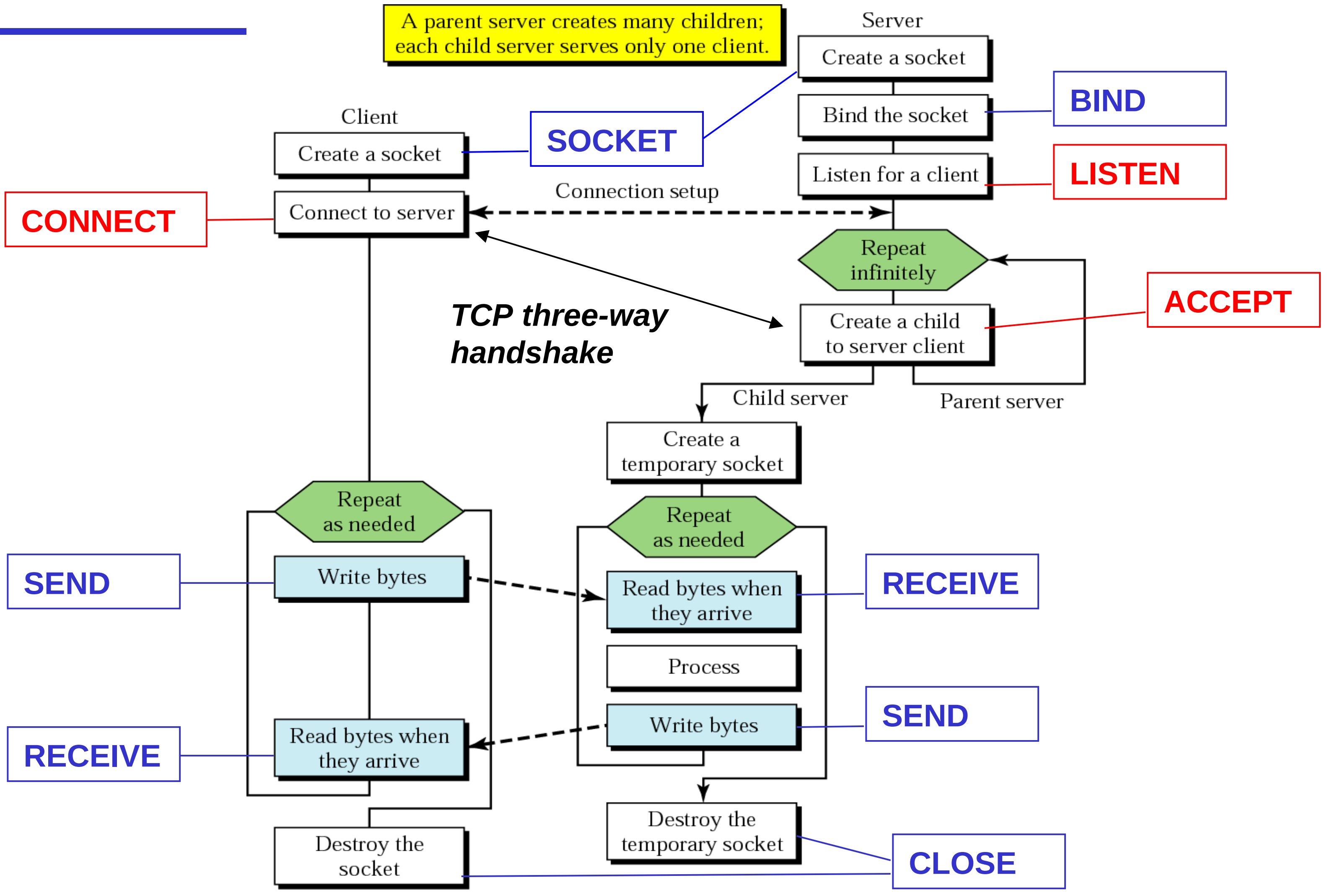
Socket primitives

SOCKET	create a new socket
BIND	attach a local address to a socket
LISTEN	announce a willingness to accept connections
ACCEPT	block the caller process until a connection attempt arrives
CONNECT	actively attempt to establish a connection
SEND	send some data over the connection
RECEIVE	receive some data from the connection
CLOSE	release the connection (the port)

Client+server: connectionless



Client+server: connection-oriented



Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

Primitives in C

- **SOCKET**: **int socket(int *domain*, int *type*, int *protocol*);**
  - *domain* := AF_INET
  - *type* := (SOCK_DATAGRAM *or* SOCK_STREAM)
  - *protocol* := 0
  - *returned*: socket descriptor (*sockfd*)

“-1” returned?

- a problem!

- **BIND**: **int bind(int *sockfd*, struct sockaddr **my_addr*, int *addrlen*);**
  - *sockfd* - socket descriptor (returned from socket())
  - *my_addr:* socket address
  - *addrlen* := sizeof(struct sockaddr)

```
my_addr.sin_family = AF_INET;
my_addr.sin_port = 0; // choose an unused port at random
my_addr.sin_addr.s_addr = INADDR_ANY; // use my IP address
memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct
```

Primitives in C (continued)

- **LISTEN: int listen(int *sockfd*, int *backlog*);**
  - *backlog*: how many connections we want to queue
- **ACCEPT: int accept(int *sockfd*, void **addr*, int **addrlen*);**
  - *addr*: here the socket-address of the caller will be written (use *struct sockaddr*)
  - *returned:* a new socket descriptor (for the temporal socket)
- **CONNECT: int connect(int *sockfd*, struct sockaddr **serv_addr*, int *addrlen*);**
  - parameters are same as for bind()
- **SEND: int send(int *sockfd*, const void **msg*, int *len*, int *flags*);**
  - *msg*: message you want to send
  - *len:* length of the message
  - *flags* := 0
  - *returned:* the number of bytes actually sent
- **RECEIVE: int recv(int *sockfd*, void **buf*, int *len*, unsigned int *flags*);**
  - *buf:* buffer to receive the message
  - *len:* length of the buffer (“don’t give me more!”)
  - *flags* := 0
  - *returned:* the number of bytes received

Primitives in C (continued)

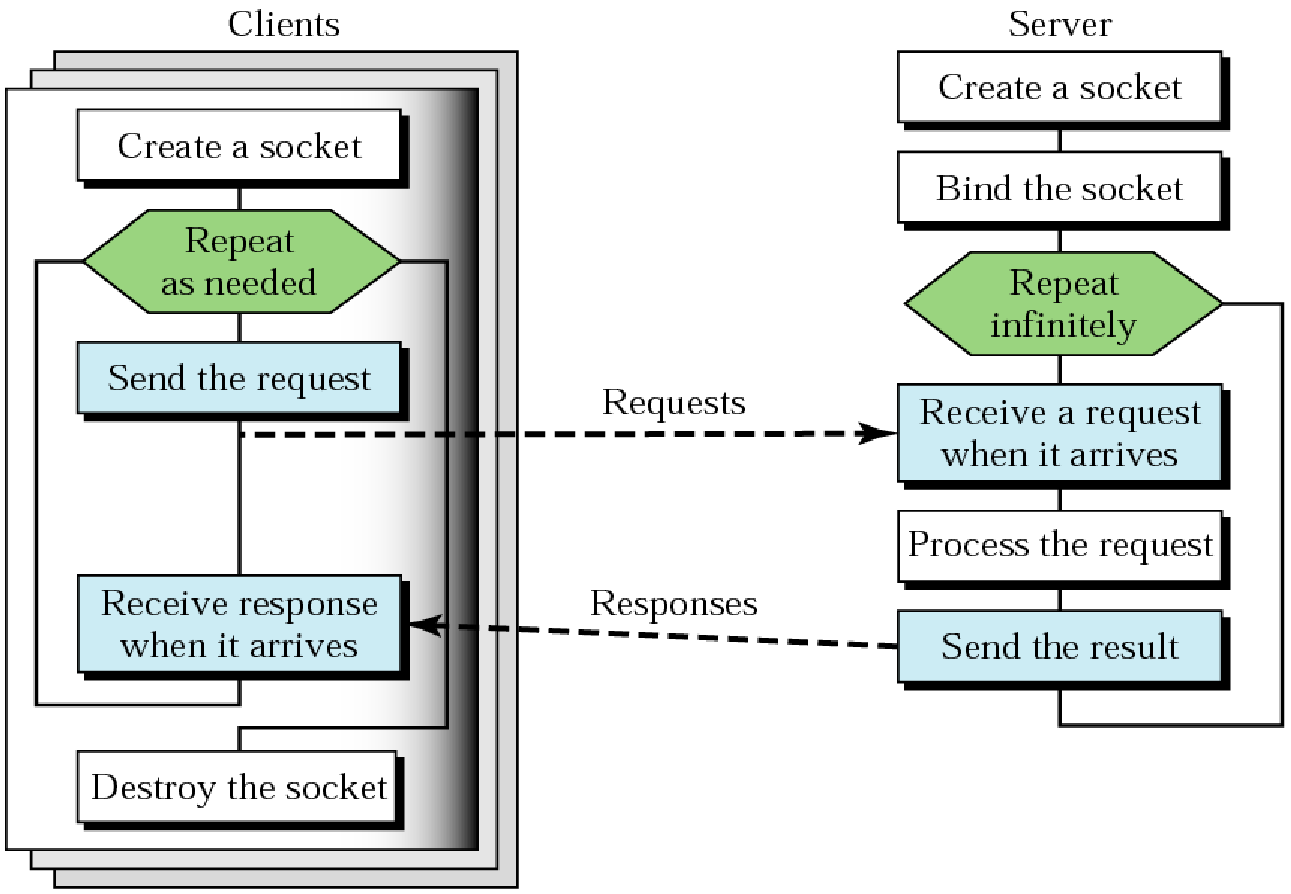
- **SEND** (DGRAM-style): **int sendto(int *sockfd*, const void **msg*, int *len*, int *flags,* const struct sockaddr **to*, int *tolen*);**
  - *msg*: message you want to send
  - *len:* length of the message
  - *flags* := 0
  - *to:* socket address of the remote process
  - *tolen*: = sizeof(struct sockaddr)
  - *returned:* the number of bytes actually sent

- **RECEIVE** (DGRAM-style): **int recvfrom(int *sockfd*, void **buf*, int *len*, unsigned int *flags,* struct sockaddr **from*, int **fromlen*);**
  - *buf:* buffer to receive the message
  - *len:* length of the buffer (“don’t give me more!”)
  - *from*: socket address of the process that sent the data
  - *fromlen*:= sizeof(struct sockaddr)
  - *flags* := 0
  - *returned:* the number of bytes received

- **CLOSE**: **close (*socketfd*);**

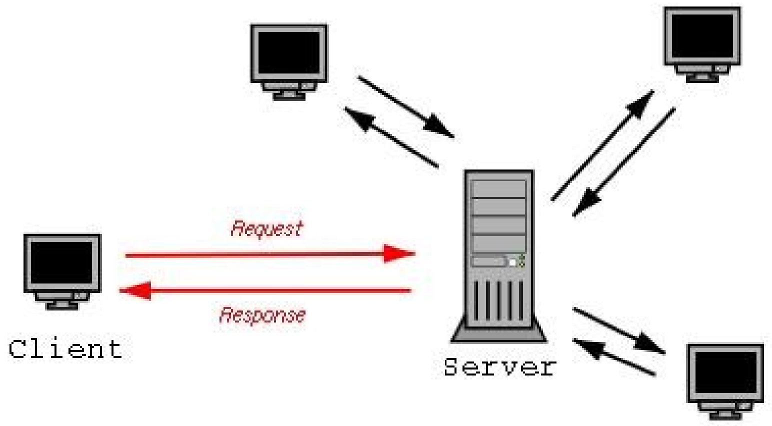
Client+server: connectionless (repeated)

Each server serves many clients but handles one request at a time.



Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

Example application: “Echo” (*in ANSI-C*)



- Client sends a message to the server
- Server echos this massage back to the client

- *We will use datagram sockets*

EchoClient.c – #include’s and #define’s

```
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), sendto(), and recvfrom() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define ECHOMAX 255     /* Longest string to echo */

void DieWithError(char *errorMessage);  /* External error handling function */
```

EchoClient.c – variable declarations

```
int main(int argc, char *argv[])
{
    int sock;                        /* Socket descriptor */
    struct sockaddr_in echoServAddr; /* Echo server address */
    struct sockaddr_in fromAddr;     /* Source address of echo */
    unsigned short echoServPort;     /* Echo server port */
    unsigned int fromSize;           /* In-out of address size for recvfrom() */
    char *servIP;                    /* IP address of server */
    char *echoString;                /* String to send to echo server */
    char echoBuffer[ECHOMAX+1];      /* Buffer for receiving echoed string */
    int echoStringLen;               /* Length of string to echo */
    int respStringLen;               /* Length of received response */

    if ((argc < 3) || (argc > 4))    /* Test for correct number of arguments */
    {
        fprintf(stderr,"Usage: %s <Server IP> <Echo Word> [<Echo Port>]\n", argv[0]);
        exit(1);
    }
```

Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

EchoClient.c – parsing the arguments

```
servIP = argv[1];           /* First arg: server IP address (dotted quad) */
echoString = argv[2];       /* Second arg: string to echo */

if ((echoStringLen = strlen(echoString)) > ECHOMAX)  /* Check input
   length */
        DieWithError("Echo word too long");

if (argc == 4)
        echoServPort = atoi(argv[3]);  /* Use given port, if any */
else
        echoServPort = 7;  /* 7 is the well-known port for the echo service */
```

EchoClient.c – creating the socket and sending

```
/* Create a datagram/UDP socket */
if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) DieWithError("socket()
    failed");

/* Construct the server address structure */
memset(&echoServAddr, 0, sizeof(echoServAddr)); /* Zero out structure */
echoServAddr.sin_family = AF_INET; /* Internet addr family */
echoServAddr.sin_addr.s_addr = inet_addr(servIP); /* Server IP address */
echoServAddr.sin_port = htons(echoServPort); /* Server port */

/* Send the string to the server */
if (sendto(sock, echoString, echoStringLen, 0, (struct sockaddr *)
    &echoServAddr, sizeof(echoServAddr)) != echoStringLen)
    DieWithError("sendto() sent a different number of bytes than expected");
```

EchoClient.c – receiving, printing

```
/* Recv a response */
fromSize = sizeof(fromAddr);
if ((respStringLen = recvfrom(sock, echoBuffer, ECHOMAX, 0, (struct
    sockaddr *) &fromAddr, &fromSize)) != echoStringLen)
    DieWithError("recvfrom() failed");
if (echoServAddr.sin_addr.s_addr != fromAddr.sin_addr.s_addr)
    { fprintf(stderr,"Error: received a packet from unknown source.\n");
exit(1); }

/* null-terminate the received data */
echoBuffer[respStringLen] = '\0';
printf("Received: %s\n", echoBuffer); /* Print the echoed arg */
close(sock);
exit(0);
} /* end of main () */
```

Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

EchoServer.c – #include’s and #define’s

- #include <stdio.h> /* for printf() and fprintf() */
- #include <sys/socket.h> /* for socket() and bind() */
- #include <arpa/inet.h> /* for sockaddr_in and inet_ntoa() */
- #include <stdlib.h> /* for atoi() and exit() */
- #include <string.h> /* for memset() */
- #include <unistd.h> /* for close() */

- #define ECHOMAX 255 /* Longest string to echo */

- void DieWithError(char *errorMessage); /* External error handling function */

EchoServer.c – variable declarations and arguments parsing

```
int main(int argc, char *argv[])
{
    int sock;                          /* Socket */
    struct sockaddr_in echoServAddr; /* Local address */
    struct sockaddr_in echoClntAddr; /* Client address */
    unsigned int cliAddrLen;         /* Length of incoming message */
    char echoBuffer[ECHOMAX];        /* Buffer for echo string */
    unsigned short echoServPort;     /* Server port */
    int recvMsgSize;                 /* Size of received message */

    if (argc != 2)         /* Test for correct number of parameters */
    {
        fprintf(stderr,"Usage:  %s <UDP SERVER PORT>\n", argv[0]);
        exit(1);
    }

    echoServPort = atoi(argv[1]); /* First arg: local port */
```

EchoServer.c – creating and binding socket

```
/* Create socket for sending/receiving datagrams */
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
     DieWithError("socket() failed");

  /* Construct local address structure */
  memset(&echoServAddr, 0, sizeof(echoServAddr));   /* Zero out
   structure */
  echoServAddr.sin_family = AF_INET;                /* Internet address
   family */
  echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Any
   incoming interface */
  echoServAddr.sin_port = htons(echoServPort);      /* Local port */

  /* Bind to the local address */
  if (bind(sock, (struct sockaddr *) &echoServAddr,
   sizeof(echoServAddr)) < 0)
     DieWithError("bind() failed");
```

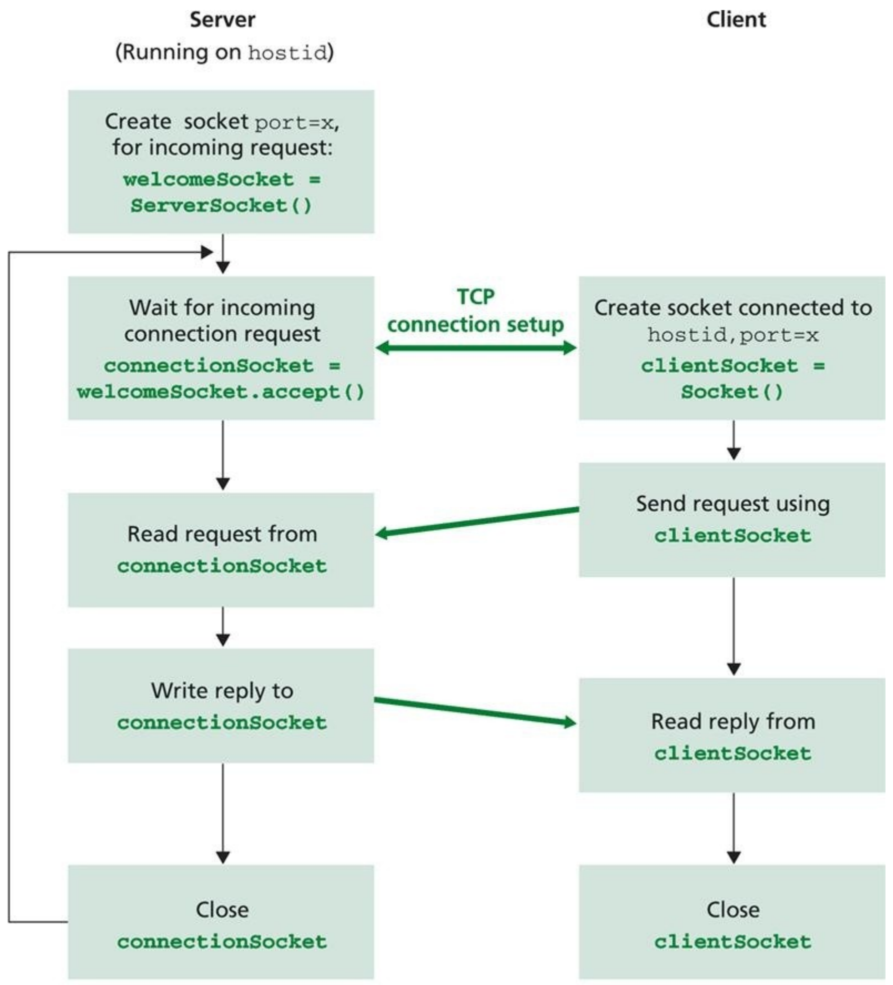
EchoServer.c – do echoing

```
for (;;) /* Run forever */
  {
     /* Set the size of the in-out parameter */
     cliAddrLen = sizeof(echoClntAddr);

     /* Block until receive message from a client */
     if ((recvMsgSize = recvfrom(sock, echoBuffer, ECHOMAX, 0,
        (struct sockaddr *) &echoClntAddr, &cliAddrLen)) < 0)
        DieWithError("recvfrom() failed");

     printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));

     /* Send received datagram back to the client */
     if (sendto(sock, echoBuffer, recvMsgSize, 0,
         (struct sockaddr *) &echoClntAddr, sizeof(echoClntAddr)) != recvMsgSize)
        DieWithError("sendto() sent a different number of bytes than expected");
  }
  /* NOT REACHED */
} /* end of main () */
```

An Example Client/Server in Java

- A client reads a line from its **standard input** (keyboard) and sends the line out its socket to the server
- The server reads a line from its connection socket
- The server converts the line to uppercase
- The server sends the modified line out its connection socket to the client
- The client reads the modified line from its socket and prints the line on its **standard output** (monitor)

Main activity between client and server


Server
(Running on hostid)
Client
Create socket port=x, for incoming request:
welcomeSocket = ServerSocket()
Wait for incoming connection request
connectionSocket = welcomeSocket.accept()
TCP connection setup
Create socket connected 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

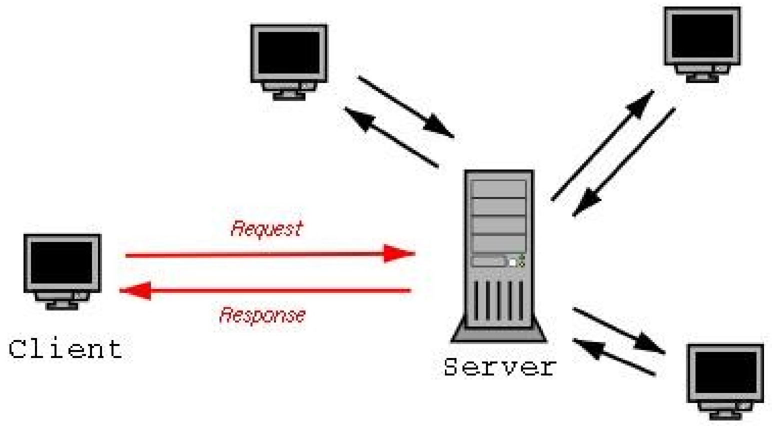
TCPClient.java

```
import java.io.*;
import java.net.*;
class TCPClient {
        public static void main (String argv[]) throws Exception
        {
                        String sentence;
                        String modifiedSentence;
                        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
                        Socket clientSocket = new Socket(“hostname”, 6789); // SOCKET, BIND, CONNECT
                        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
                        BufferedReader inFromServer = new BufferedReader(new
                                InputStreamReader(clientSocket.getInputStream()));
                        sentence = inFromUser.readLine();
                        outToServer.writeBytes(sentence + ‘\n’);
                        modifiedSentence = inFromServer.readLine();
                        System.out.println(“FROM SERVER: “ + modifiedSentence);
                        clientSocket.close();
        }
}
```

TCPServer.java

```
import java.io.*;
import java.net.*;
class TCPClient {
        public static void main (String argv[]) throws Exception
        {
                        String clientSentence;
                        String capitalizedSentence;
                        ServerSocket welcomeSocket = new ServerSocket(6789); //SOCKET, BIND, LISTEN
                        while(true) {
                                Socket connectionSocket = welcomeSocket.accept(); //ACCEPT
                                BufferedReader inFromClient = new BufferedReader(new
                                        InputStreamReader(connectionSocket.getInputStream()));
                                DataOutputStream outToClient = new
                                        DataOutputStream(connectionSocket.getOutputStream());
                                clientSentence = inFromClient.readLine(); //RECEIVE
                                capitalizedSentence = clientSentence.toUpperCase() + ‘\n’;
                                outToClient.writeBytes(capitalizedSentence); //SEND
                        }
        }
}
```

Example application: “quote of the day” (*Java*)



- Client requests the server
- Server sends a quote-of-the-day back to the client

- *We will use datagram sockets*

Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

“Quote of the day”: server

- Creating a socket

  socket = new DatagramSocket(4445);

  // calls: SOCKET (SOCK_DGRAM), BIND (port 4445) – “we have got a post-address”

- Receiving a request from a client

  byte[] *buf* = new byte[256]; // new buffer (array of bytes)

  DatagramPacket *packet* = new DatagramPacket(*buf*, *buf*.length); // = a buffer + socket specific information - “make a letter box”

  socket.receive(*packet*); // calls RECEIVE – recvfrom() - “hang the letter box out on the door and wait till a letter arrives”

- Filling the buffer with another quote (stored in a file)

  String *dString* = getNextQuote(); //read another line from a file – “find the requested info”

  buf = dString.getBytes(); // transfer the content of the string into the buffer – “write the letter”

- Sending a quote back to the client

  InetAddress *address* = *packet*.getAddress(); // from which IP address did it arrive?

  int port = *packet*.getPort(); // … and the port number? – “put the address on the envelop"

  packet = new DatagramPacket(*buf*, *buf*.length, *address*, *port*); // make up a new packet – “pack the letter into the envelop”

  socket.send(*packet*); //calls: SEND – sendto() - “put the letter to the mail box”

Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

“Quote of the day”: client

- Creating a socket

  socket = new DatagramSocket();

  // calls: SOCKET (SOCK_DGRAM), *no BIND*

- Sending a request to the server

  byte[] *buf* = new byte[256];

  InetAddress *address* = InetAddress.getByName(*args*[0]); // get the server address from the command line

  DatagramPacket *packet* = new DatagramPacket(*buf*, *buf*.length, *address*, 4445); // make up “a letter”

  socket.send(*packet*); // calls SEND – sendto()

- Receiving the respond (a quote line) from the server

  packet = new DatagramPacket(buf, buf.length); // “make a letter box”

  socket.receive(packet); // calls RECEIVE – recvfrom()

  String received = new String(packet.getData()); // “open the letter” – extract the quote

  System.out.println("Quote of the Moment: " + received); // print the quote

Sergei N. Kozlov, s.n.kozlov@tue.nl
TU/e Informatica, System Architecture and Networking

Tips for the assignment of this year

with examples in ANSI-C

Tips&Hints

- How would you manage upload and download?

  Use multiple threads
- Speed limitations

  Do it in the application layer
- Downloading from multiple sources

  Both single- and multiple-threaded approaches are possible

General tip

- Always check the returned values (or handle exceptions)

Some links

- UNIX sockets:

  Tutorial

  http://www.ecst.csuchico.edu/~beej/guide/net/html/

  Examples (also used in this tutorial):

  http://cs.baylor.edu/~donahoo/practical/CSockets/textcode.html

- Winsock:

  FAQ

  http://tangentsoft.net/wskfaq/

Good luck with the assignment

- *More questions?*