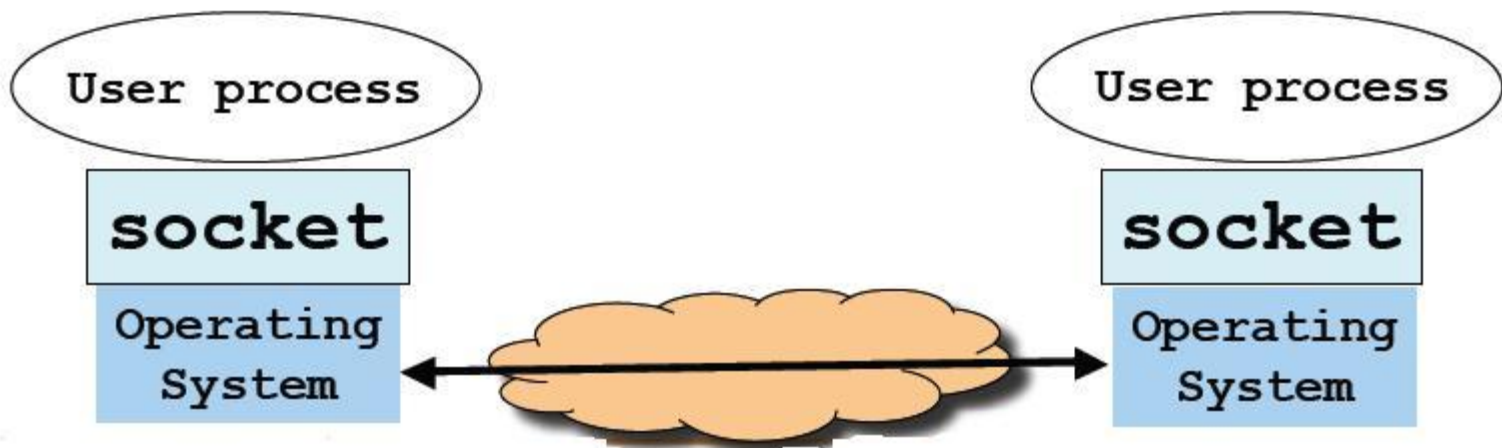


# Socket programming

Tasos Alexandridis

# Socket: End Point of Communication

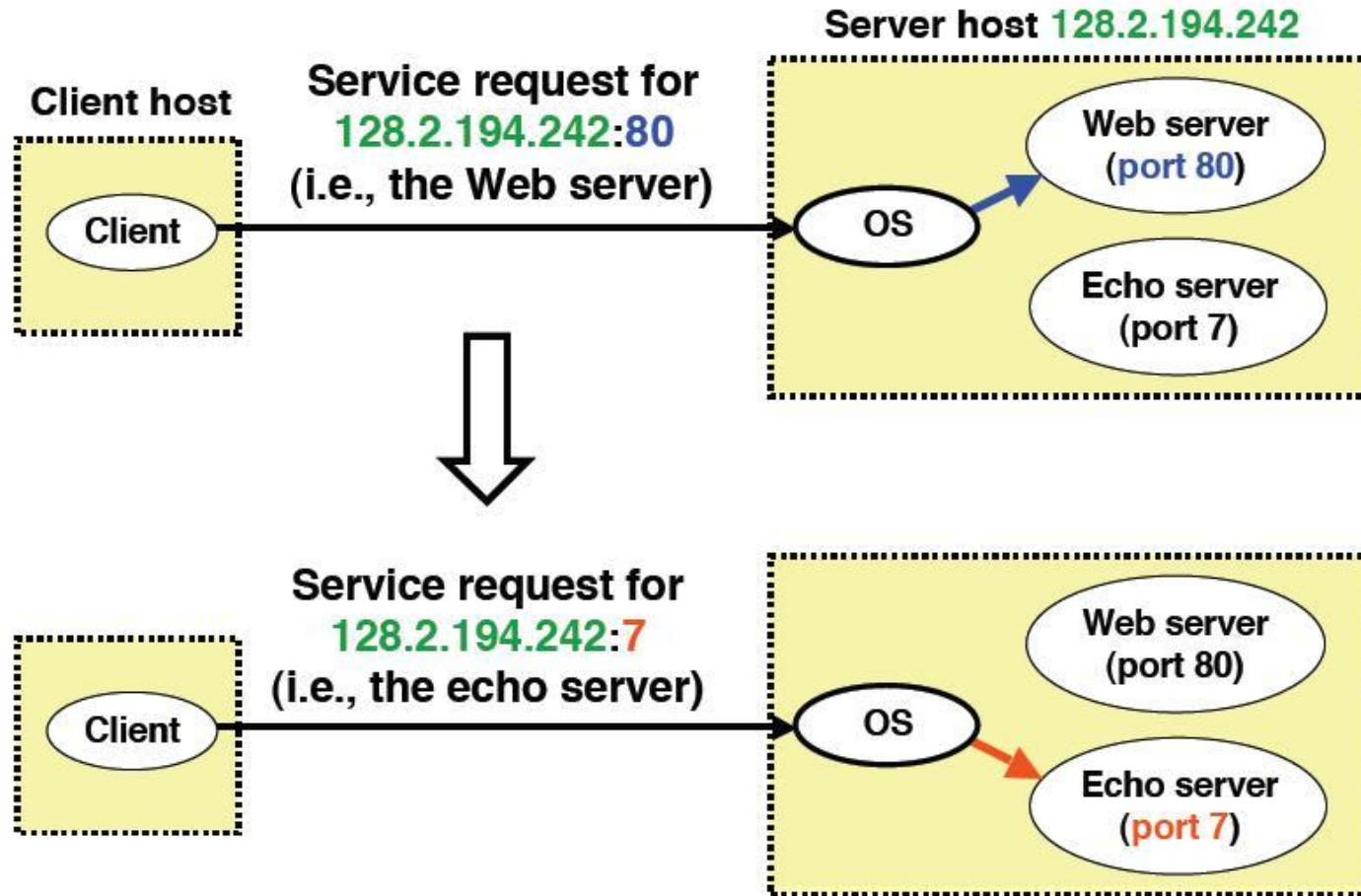
- Sending message from one process to another
  - Message must traverse the underlying network
- Process sends and receives through a socket
  - In essence, the doorway leading in/out of the house



# Identifying the Receiving Process

- Sending process must identify the receiver
  - **Address** of the receiving end host
  - Identifier (**port**) that specifies the receiving process
- Receiving **host**
  - Destination IP address (32-bit) uniquely identifies the host
- Receiving **process**
  - Host may be running many different processes
  - Destination port (16-bit) uniquely identifies the socket (process)

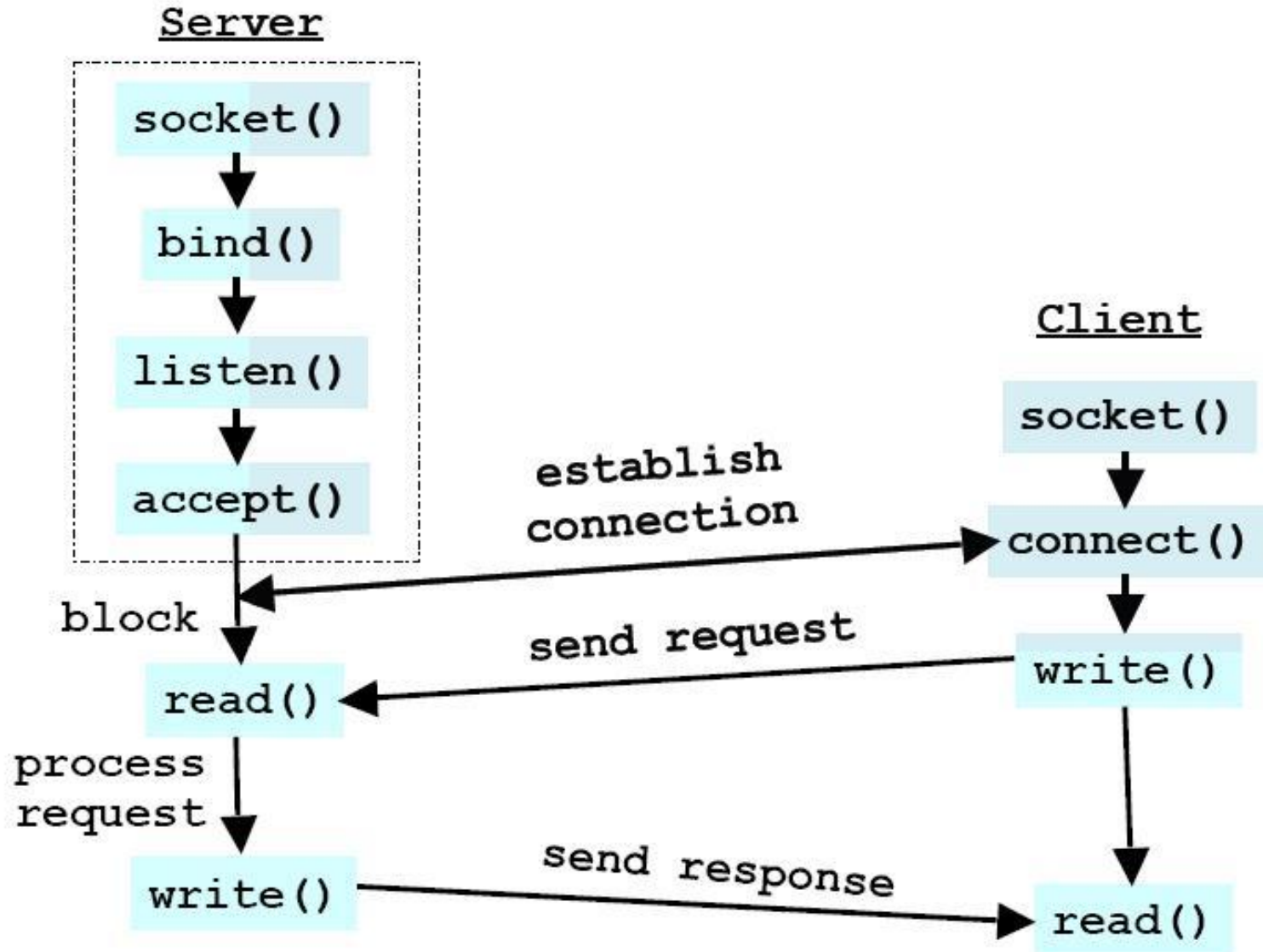
# Using ports to identify Services



# Using sockets in C

## The UNIX Socket API

# Overview



# socket()

- `int socket(int domain, int type, int protocol)`
  - Creates a socket
- Domain specifies the protocol family
  - `AF_UNIX`, `AF_LOCAL` for local communication
  - `AF_INET` for IPv4 Internet protocols (use this)
- Type specifies the communication
  - `SOCK_STREAM` for TCP
  - `SOCK_DGRAM` for UDP
- Protocol specifies the protocol (set to 0 as domain and type imply the protocol)
- RETURN
  - On success returns socket descriptor, else -1

# bind()

- `int bind(int socket, struct sockaddr *address, int addrLen)`
  - Assigns a specific address to the socket
- `socket` → the socket descriptor
- `address` → the host address
- `addr_len` → the size of the address struct
  
- RETURN
  - 0 on success, else -1



# sockaddr & sockaddr\_in

```
struct sockaddr_in {  
    short int sin_family; // Address family  
    unsigned short int sin_port; //Port number  
    struct in_addr sin_addr; //Internet address  
    unsigned char sin_zero[8];  
};
```

```
struct in_addr {  
    unsigned long s_addr;  
}
```

```
struct sockaddr_in addr;  
struct sockaddr *address = (struct sockaddr *)&addr;  
    //Type casting
```

# listen()

- `int listen(int sockfd, int backlog)`
  - Listens (waits) for incoming connections
- `sockfd` → the socket descriptor
- `backlog` → max number of connections
  
- RETURN
  - 0 on success, else -1

# accept()

- `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)`
  - Accepts an incoming connection
- `sockfd` → the socket descriptor (after a call to `listen()` )
- `addr` → pointer to a struct that is filled with the client's address (optional)
- `addrlen` → the size of the `addr` struct
  
- RETURN
  - the new socket descriptor for the specific connection, or -1 if an error occurred

# send() / receive()

- `ssize_t send(int sockfd, const void *buf, size_t len, int flags);`
  - send data using the specified socket
- `ssize_t recv(int sockfd, void *buf, size_t len, int flags);`
  - Receive data from the specified socket and store it to buf
- `sockfd` → the socket descriptor
- `buf` → buffer to send or receiver
- `len` → the length of the buffer
- `flags` → (set to 0)
- RETURN
  - The total number of bytes sent/received, else -1

# connect()

- `int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen)`
  - Connect to a socket
- `sockfd` → the unconnected socket descriptor
- `addr` → the server's address
- `addrlen` → the size of the `addr` struct
- RETURN
  - On success 0, else -1

# close()

- `int close(int fd)`
  - Closes the socket
- `fd` → the socket descriptor
- RETURN
  - 0 on success, else -1

# Example

TCP Server and TCP client in C

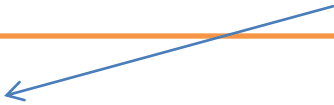
# Socket creation

```
int sock;
```

```
if((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {  
    fprintf(stderr, "Failed to create TCP socket");  
}
```

```
if(setsockopt(sock, SOL_SOCKET, SO_REUSEADDR,  
    &optval, sizeof(optval)) < 0) {  
    fprintf(stderr, "could not reuse address");  
}
```

Reuse the same port  
number





# Bind port to socket

```
struct sockaddr_in addr;

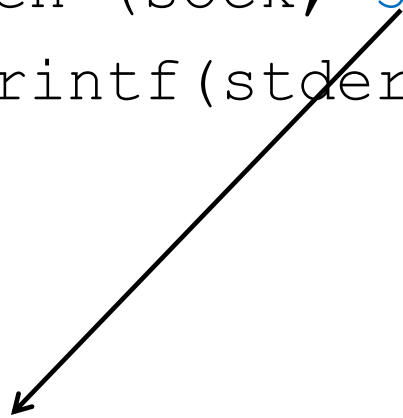
memset(&addr, 0 , sizeof(addr)); //clear memory block for
                                  //addr

addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR;
addr.sin_port = htons(server_port);

if(bind(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
{
    fprintf(stderr, "cannot bind socket to address");
}
```

# Wait for incoming connections

```
If(listen (sock, 5) < 0) {  
    fprintf(stderr, "error listening");  
}
```



Specifies max number of incoming connections

# Client establishes connection

```
struct sockaddr_in sin;

struct hostent *host = gethostbyname(argv[1]);

sin.sin_addr = *(in_addr_t) *host->h_addr_list[0];

memset(&sin, 0, sizeof(sin));

sin.sin_family = AF_INET;                                use raw IP address
sin.sin_addr.s_addr = server_addr; // = inet_addr(server_IP) ;
sin.sin_port = htons(server_port);

if(connect(sock, (struct sockaddr *)&sin, sizeof(sin)) < 0) {
    fprintf(stderr, "cannot connect to server");
}
```

# Accept incoming connections

```
struct sockaddr_in addr;
int addr_len = sizeof(addr);
int c_sock;

c_sock = accept(sock,
                (struct sockaddr *)&addr, &addr,
                &addr_len);

if(c_sock < 0) {
    fprintf(stderr, "error accepting connection");
}
```

# Send data

```
int send_packets(char *buffer, int buf_len) {
    sent_bytes = send(sock, buffer, buf_len, 0) ;

    if(send_bytes < 0) {
        fprintf(stderr, "send() failed");
    }
    return 0;
}
```

# Receive data

```
int receive_packets ( char *buffer, int buf_len) {
    int num_received = recv(sock, buffer, buf_len, 0);

    if(num_received < 0) {
        fprintf(stderr, "recv() failed");
    } else if (num_received == 0) {
        //sender has closed connection
        return EOF;
    } else {
        return num_received;
    }
}
```

What happens when data exceed buffer size?

# #include the appropriate libraries

```
#include <unistd.h> /* access to system calls */
#include <sys/types.h> /*widely used types */
#include <netdb.h> /* gethostbyname() etc.*/
#include <arpa/inet.h> /*htons etc. */
#include <sys/socket.h> /*socket structs */
#include <netinet/in.h> /*internet sockets,
                        sockaddr_in etc. */
```

# Datagram Sockets (UDP)

- Similar to stream sockets but:
  - Use `SOCK_DGRAM` instead of `SOCK_STREAM`
  - No need to establish and terminate connection
  - Use `recvfrom()` and `sendto()` instead of `recv()` and `send()`

```
ssize_t recvfrom(int sockfd, void *buf,  
                size_t len, int flags, struct sockaddr  
                *src_addr, socklen_t *addrlen);
```

```
ssize_t sendto(int sockfd, const void  
              *buf, size_t len, int flags, const struct  
              sockaddr *dest_addr, socklen_t addrlen);
```



# Using sockets in Java

# TCP Client

```
public static void main(String argv[]) throws Exception
{
    String sentence = "Hello";
    String modifiedSentence;
    //create the socket
    Socket clientSocket = new Socket("hostname", port);
    //get the input and output stream that are attached to the
    socket
    DataOutputStream outToServer = new
        DataOutputStream(clientSocket.getOutputStream());
    BufferedReader inFromServer = new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));

    outToServer.writeBytes(sentence + '\n'); //send
    modifiedSentence = inFromServer.readLine(); //receive
    System.out.println("From server: " + modifiedSentence);
    clientSocket.close(); //close the socket
}
```

# TCP Server

```
public static void main(String argv[]) throws Exception
{
    //create the server socket
    ServerSocket welcomeSocket = new ServerSocket(port);
    //accept the connection
    Socket connectionSocket = welcomeSocket.accept();
    //get the streams
    DataOutputStream outToClient = new
        DataOutputStream(connectionSocket.getOutputStream());
    BufferedReader inFromClient = new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
    //receive from client
    String clientSentence = inFromClient.readLine();
    //send to client
    outToClient.writeBytes(clientSentence.toUpperCase());
}
```

# Datagram Sockets(UDP) in Java

- No initial handshaking and therefore no need for welcoming socket (server socket)
- No streams are attached to the sockets
- The receiving process must unravel the received packet to obtain the packet's information bytes
- The sending host creates "packets" by attaching the IP destination address and port number to each batch of bytes it sends

# UDP Client

```
public static void main(String args[]) throws Exception {  
    BufferedReader inFromUser = new BufferedReader(new  
        InputStreamReader(System.in));  
    DatagramSocket clientSocket = new DatagramSocket(); //UDP  
                                                    //Socket  
    InetAddress IPAddress = InetAddress.getByName("hostname");  
    String sentence = inFromUser.readLine();  
  
    byte[] receiveData = new byte[1024];  
  
    byte[] sendData = sentence.getBytes();
```

# UDP Client

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length, IPAddress,  
        9876);  
clientSocket.send(sendPacket); //send a UDP packet  
  
DatagramPacket receivePacket = new  
    DatagramPacket(receiveData, receiveData.length);  
clientSocket.receive(receivePacket); //receive a UDP packet  
String modifiedSentence = new String(receivePacket.getData());  
  
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}
```

# UDP Server

```
public static void main(String args[]) throws Exception {  
    DatagramSocket serverSocket = new DatagramSocket(9876);  
    byte[] receiveData = new byte[1024];  
    byte[] sendData = new byte[1024];  
  
    DatagramPacket receivePacket =  
        new DatagramPacket(receiveData, receiveData.length);  
    serverSocket.receive(receivePacket);  
  
    String sentence = new String(receivePacket.getData());
```

# UDP Server

```
InetAddress IPAddress = receivePacket.getAddress();
```

```
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();  
sendData = capitalizedSentence.getBytes();
```

```
DatagramPacket sendPacket =  
    new DatagramPacket(sendData, sendData.length,  
        IPAddress, port);
```

```
serverSocket.send(sendPacket);
```

```
}
```



# Classes

- ServerSocket
- Socket
- DatagramSocket
- DatagramPacket
- InetAddress

<http://download-llnw.oracle.com/javase/1.5.0/docs/api/> (Java API)