Communication in Client-Server Systems

*Socket Programming*

**National University of Computer and Emerging Sciences, Lahore**

**Operating System Lab**

Sockets

A **socket is defined as an endpoint for communication. A pair of processes** communicating over a network employs a pair of sockets—one for each process. A socket is identified by an IP address concatenated with a port number. In general, sockets use a client–server architecture. The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection. Servers implementing specific services (such as telnet, FTP, and HTTP) listen to well-known ports (a telnet server listens to port 23; an FTP server listens to port 21; and a web, or HTTP, server listens to port

80). All ports below 1024 are considered ***well known; we can use them to implement*** standard

services.

**References**: Operating System concepts Page n.o 136 section 3.6

Socket

• Communication points on the same or different computers to exchange data

• Allows communication between two different processes on the same

of different machines

• Always has an address (IP and Port)

• A UNIX socket is used in a client-server application framework

• Server is a process that performs some functions on request from a client

• Just like a file (open, close, read, write)

IP address (Internet Protocol) & Socket

• IP address

• Identify hosts connected to the internet

• Written in a dotted-decimal notation of the form N1.N2.N3.N4 where each Ni is a

decimal number between 0 and 255

• Socket

• To identify a particular process running on a host

• An integer number

• Port numbers smaller than 1024 i.e. 0-1023 are well-known ports // port 80 for http

(standard service)

• We can use port numbers from 1024 to 65535

• Works like telephone extension

• Main phone number computer IP address

• Extension numbers set of port numbers

Loopback IP

• The IP address 127.0.0.1 is a special IP address known as the **loopback**. When a computer refers to IP address 127.0.0.1, it is referring to itself.

• This mechanism allows a client and server on the same host to

communicate using the TCP/IP protocol.

Types of socket

• Stream sockets (SOCK\_STREAM)

• TCP (Transmission Control Protocol)

• Message delivery is guaranteed

• Message order retains

• Sender receives error message on failure

• Datagram sockets (SOCK\_DGRAM)

• UDP (User Datagram Protocol)

• Delivery not guaranteed

• Connection less (build message with destination information and send It out)

Functions used in Socket Programming

socket() Endpoint for communication

bind() Assign a unique telephone number listen() Wait for a caller

connect() Dial a number accept() Receive a call send(), recv() Talk

close() Hang up

socket()… get the file descriptor

• int sd=socket(int *domain*, int *type*, int

*protocol*);

• *domain*  AF\_INET, PF\_INET

• *type*  SOCK\_STREAM, SOCK\_DGRAM

• *protocol*  set to 0 for appropriate protocol selection, *IPPROTO\_TCP,*

*IPPROTO\_UDP*

• *return*  socket descriptor on success and -1 on error

• Example

• int T\_s=socket(AF\_INET,SOCK\_STREAM,0);

bind() … what port am I on?

• Associate a socket id with an address to which other process can connect

• int status=bind(int *sd*, struct sockaddr\*

*addrptr*, int *size*);

• *status*  0 on success and -1 on error

• *sd*  socket file descriptor created and return by socket()

• *addrptr*  pointer to struct sockaddr type parameter, contains

current socket IP and port

• *size*  size of *addrptr*

connect() … request for connection

• int status=connect(int *sd*, struct sockaddr

\**serv\_addr*, int *addrlen*)

• *status*  error -1

• *sd*  socket file descriptor

• *serv\_addr*  is a pointer to struct sockaddr that contains destination IP

address and port

• *addrlen*  size of *serv\_addr*

listen()

• Waits for incoming connections

• int status=listen(int *sd*, int *backlog*);

• *sd*  socket on which the server is listening

• *backlog*  maximum number of connections pending in a queue

• *status*  return -1 on error

accept()

• Blocking system call

• Waits for an incoming request and when received, creates a socket

for it

• int sid=accept(int *sd*, struct sockaddr

\**cli\_addr*, int \**addrlen*)

• *sid*  socket file descriptor for communication

• *sd*  socket file descriptor used for listening

• *addr*  poitner to struct sockaddr containing client address IP and

Port

• *addrlen*  size of struct sockaddr

• int sb=send(int sd, const char \*msg, int len, int flags);

• *sb*  return number of bytes send of -1 for error

• *sd*  socket file descriptor

• *msg*  is a pointer to data buffer

• *len*  number of bytes we want to send

• *flags*  set it to 0 for default

• int rb=recv(inst *sd*, char \**buf*, int *len*, int

*flags*);

• *rb*  number of bytes received or -1 on error. 0 if connection is closed at

other side

• *sd*  socket file descriptor

• *buf*  is a pointer to data buffer

• *len*  receive up to *len* bytes in buffer pointer

• *flag*  set it to 0 for default

• Close connection on given socket and frees the socket desciptor

• int close(int *sd*);

• Generic

• Holds socket address information for many types of sockets

struct sockaddr{

unsigned short sa\_family; //address family AF\_xxx unsigned short sa\_data[14]; //14 bytes of protocol

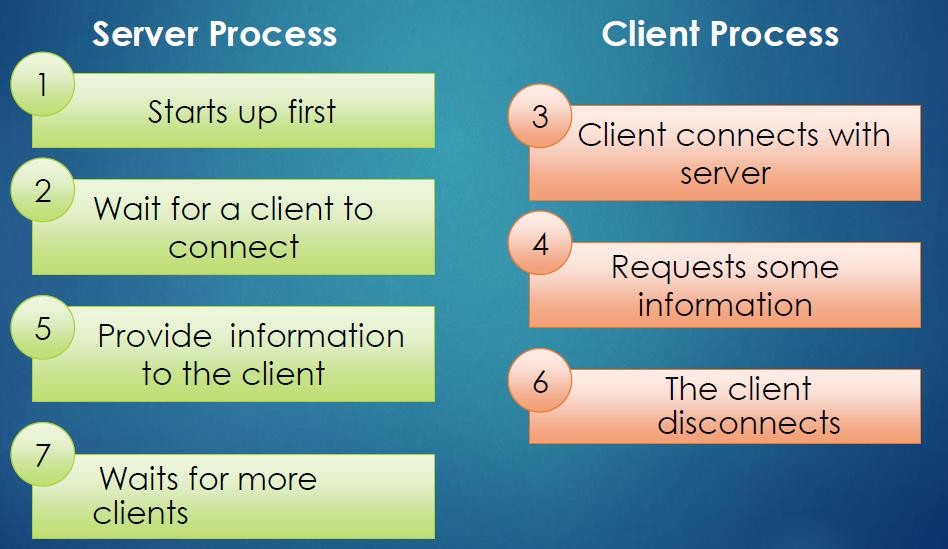
addr

• IPV4 specific

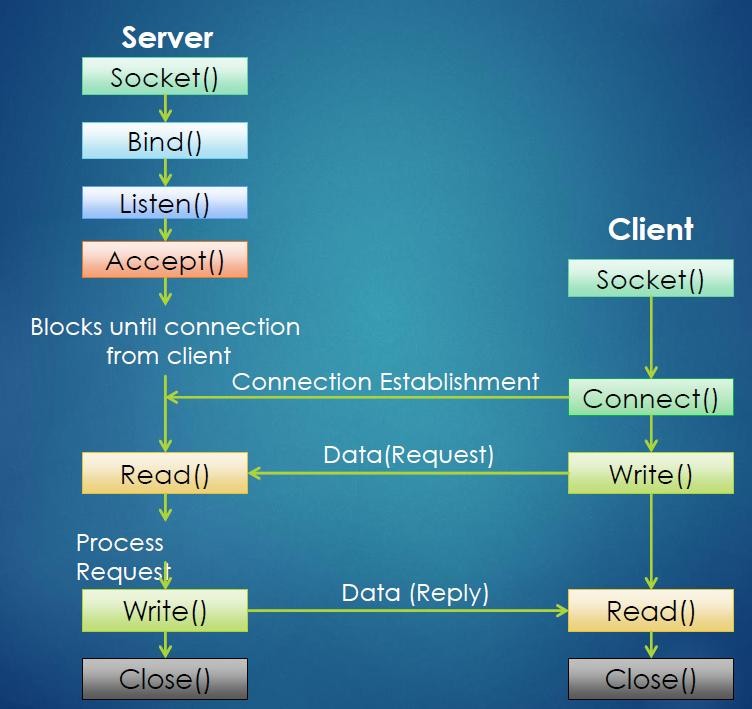
struct sockaddr\_in{

short int sin\_family; // set to AF\_INET unsigned short int sin\_port; // port number struct in\_addr sin\_addr; // internet address unsigned char sin\_zero[8]; //set to all zeros

Client-Server model



TCP client-server model



Commands for IP inquiry

• ipconfig // windows

• ifconfig // linux

• hostname –I //linux

• ip addr show //linux

• Network tools