Operating System Tutorial

Tutorial 7

Inter Process Communication(IPC)

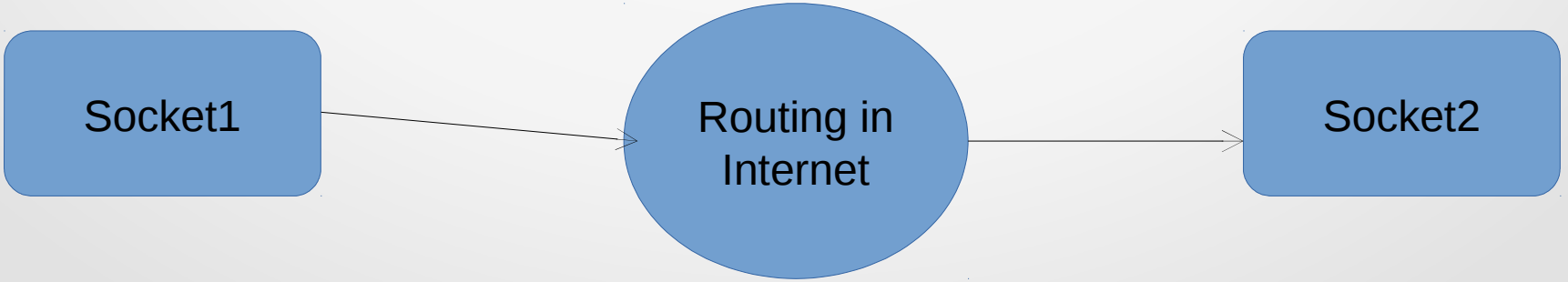
(Socket)

What is Socket ?

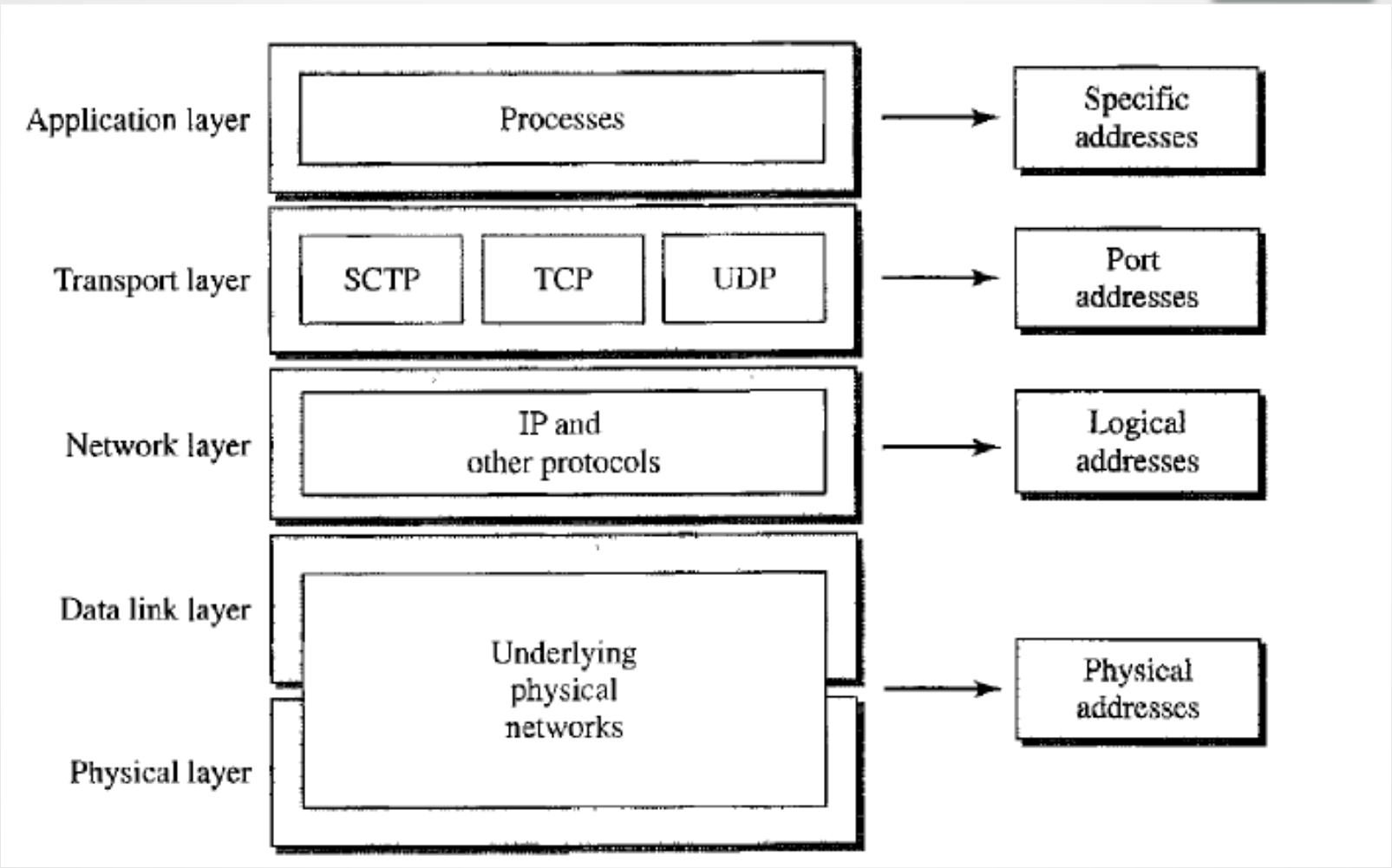
- A Socket is an end point of communication between two systems on a network.
- A socket is a combination of IP address and port on one system.
- Socket is just like PIPE. When we open a Socket it returns only file descriptor.
- Communication through socket follows some special set of standards. Which enable two different machines to communicate with each other using socket.

How it Works?

- Communication between two processes follows different addressing schemes.
  - IP Address -> Identify a machine. Unique in world.
  - Port Address -> Identify a process inside a machine
  - Combination of above two identify unique process in world.
- IP Address + Port Address = Socket.



Different Layers of Networking



IP Address

- IP address is short for Internet Protocol (IP) address.
- An IP address is an identifier for a computer or device on a TCP/IP network.
- Networks using the TCP/IP protocol route messages based on the IP address of the destination.
- IP Addresses are 32 bit numbers represented as 4 groups of 8 bit numbers seperated by dot(.).
- Min:- 0.0.0.0
- Max:- 255.255.255.255
- IP Address is used as network layer.
- Network layer is responsible for machine to machine communication.

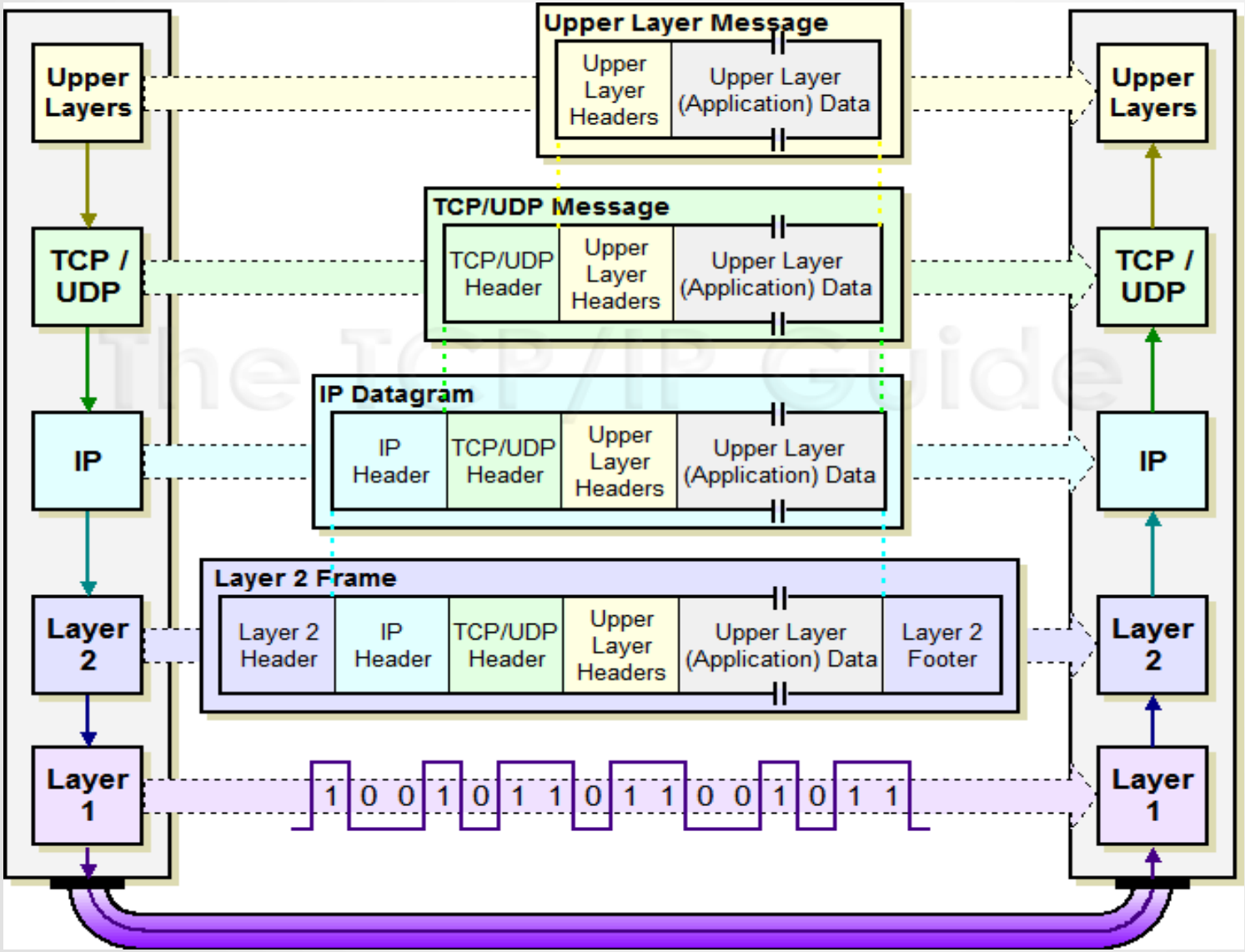
Transport Layer

- This layer is responsible for process to process communication.
- Port number is a 16 bit integer assigned to different services.
- For different service different port numbers are used.
- Two types of network layer protocol:
  - TCP (Stream Oriented, Connection Oriented)
  - UDP (Connectionless)

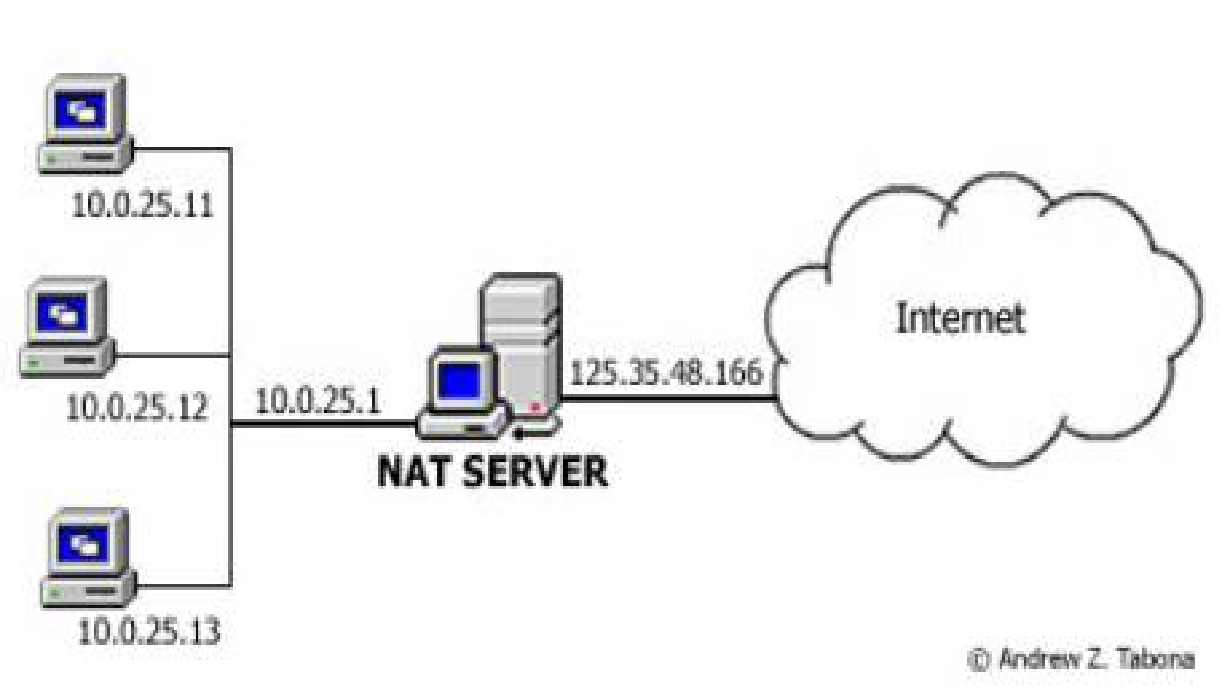
Application Layer

- Actual applications are run at this layer.
- Ex:- Telnet, Http, Https, Ftp etc .

Encapsulation



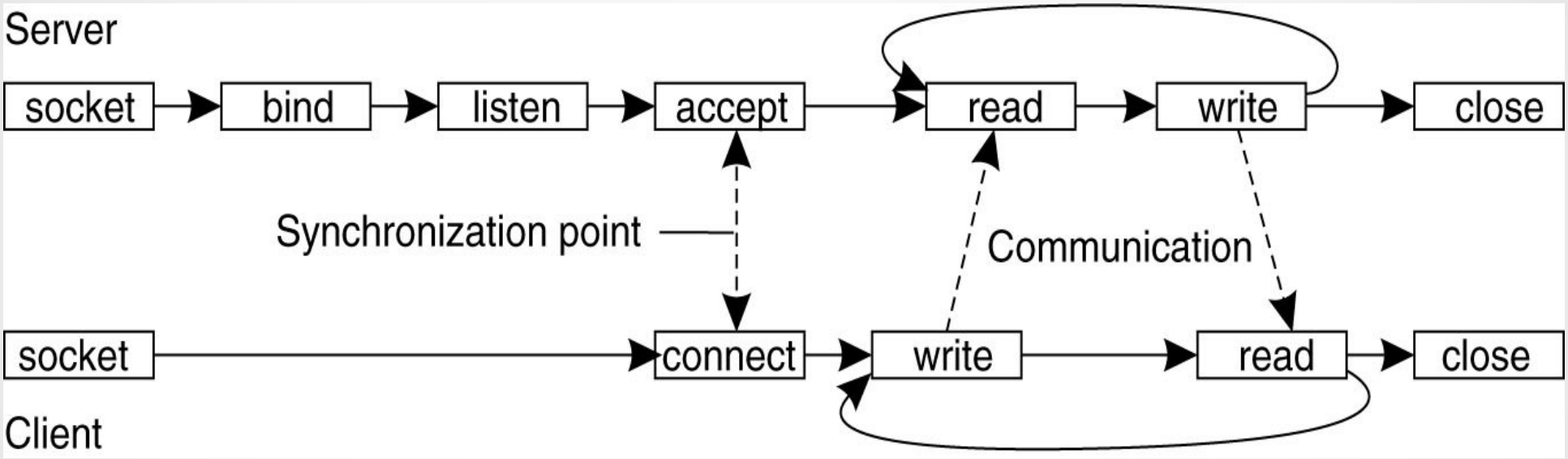
Network Address Translation(NAT)



Ports for Well Known Applications

- There are some fixed port numbers for well known applications:
  - 80 -> HTTP
  - 443 -> HTTPS
  - 23 -> Telnet
  - 20, 21 -> FTP
  - 22 -> SSH
  - 25 -> SMTP

Communication using sockets.



Creating a Socket

- *int socket_desc = socket(AF_INET , SOCK_STREAM , 0);*
- Creates a socket and returns file descriptor (integer) of the created socket. Returns -1 if fail to create.
- Address Family:
  - AF_INET: Use this if you are using IPV4
  - AF_INET6: Use this if you are using IPV6
- Type:
  - SOCK_STREAM: For TCP
  - SOCK_DGRAM: For UDP
- Protocol:
  - 0 or IPPROTO_IP: For IP Protocol

Connect to a Server

- We have to use three readymade structutes defined in *arpa/inet.h.*

- struct sockaddr_in {
  - short sin_family; // e.g. AF_INET, AF_INET6
  - unsigned short sin_port; // e.g. htons(3490)
  - struct in_addr sin_addr; // see struct in_addr, below
  - char sin_zero[8]; // zero this if you want to
- };
-

Connect to a Server

- Create a variable of type structure sockaddr_in
  - *struct sockaddr_in server;*
- Use inet_addr function to store server address. This function changes IP to unsigned long.
  - *server.sin_addr.s_addr = inet_addr("74.125.235.20");*
- Which protocol you want to use for this connection:
  - *server.sin_family = AF_INET;*
- Which port of server you want to connect to:
  - *server.sin_port = htons(80);*

Connect to a Server

- Finally connect to server
  - *connect(socket_desc , (struct sockaddr *)&server , sizeof(server));*
- Return 0 if connected and -1 if not connected.

Request to a Server

- Create a message string
  - *message = "GET /vm/index.php HTTP/1.1\r\nHost: 10.200.110.53\r\n\r\n";*
- Send it to server:
  - *send(socket_desc , message , strlen(message) , 0);*
  - send returns size of data you sent(18) if successful.
  - If fails return -1.

Receive a Reply From Server

- Create a character array to store the reply
  - *char server_reply[2000];*
- Receive a reply from the server:
  - recv(socket_desc, server_reply , 2000 , 0);
  - *Store reply in server_reply.*
  - *Returns total size of data returned by server or 2000 if Buffer is full.*
- When receiving data on a socket , we are basically reading the data on the socket. This is similar to reading data from a file.
  - *read(socket_desc, server_reply , 2000);*

Close the Socket

- Function close is used to close the socket. Need to include the unistd.h header file for this.

close(socket_desc);

Create a Server

- Create a variable of struct sockaddr_in:
  - *struct sockaddr_in server;*
- Create the server socket:
  - *socket_desc = socket(AF_INET , SOCK_STREAM , 0);*
- Prepare the sockaddr_in structure for server
  - *server.sin_family = AF_INET;*
  - *server.sin_addr.s_addr = INADDR_ANY;*
- Can listen at any IP.
  - *server.sin_port = htons(8888);*

Create a Server

- Bind the socket on given address:
  - *bind(socket_desc,(struct sockaddr *)&server , sizeof(server))*
- *Returns 0 if bind done successfully*
- *Returns -1 if unsuccessful*
- Start listining for clients:
  - listen(socket_desc , 3);
- 3 specifies the queue length for completely established sockets waiting to be accepted.

Create a Server

- Accept connection from client:
  - *int c = sizeof(struct sockaddr_in);*
  - *new_socket = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);*
- *Returns file-descriptor of socket of the client, -1 when fail.*
- *Fill the client structure with client information*
- *Blocks untill new connection arrives*
- *After accepting new connection unblock the program.*
- Start listining for clients:
  - listen(socket_desc , 3);
- 3 specifies the queue length for completely established sockets waiting to be accepted

Get Client Information

- IP Address of client:
  - *char *client_ip = inet_ntoa(client.sin_addr);*
- Port Address of client:
  - *int client_port = ntohs(client.sin_port);*

Send Reply to Client

- Write on client's socket to send a reply to client:
  - *message = "Hello Client , I have received your connection. But I have to go now, bye\n"*
  - *write(new_socket , message , strlen(message));*

LIVE Server

- Server does no die after connection. Put accept() inside while(1) loop.

Task 1

Write a socket based client-server application to send hello message from server to client.

Task 2

Repeat above task to send hello message from server to multiple clients (i.e. 3 clients).

.

Task 3

Design a client server application to send date and time from server to clients when requested. The client sets a timeout on its socket so that it can inform the user when the server does not reply.

Task 4

Design a client server application to calculate factorial of a number by server when asked by client.

Task 5

Write a socket based chatting program. Use TCP to send human understandable message.