COMPUTER NETWORKS

Some Protocols

TCP [lossless]
UDP [lossy]
IPv4
IPv6

lossless
↳ there is no loss of data when going from one to another point.

lossy
↳ the data is often lost, i.e, quality is not maintained.

server.c [a server from scratch]

Components needed
→ socket
→ data
→ listener
→ port
→ protocol

Necessary imports:
1. stdio.h → I/O functions
2. stdlib.h → memory functions
3. string.h → string manipulation functions
4. unistd.h → unix standard functions
5. arpa/inet.h ↳ functions for internet operations
6. sys/sockets.h

→ define constraints → port you want to listen on

```c
#define PORT 8080
#define BUFFER_SIZE 1024
```
↳ size of data to receive

A Program in C [Example of a server client network]

```c
int main ()
{ int server_fd, new_socket;
```
server_fd → file descriptor [unique descriptor to manage open files. i.e, reading, writing data to network]

```c
int opt = 1; //used to set timeouts, buffers
// setsocketopt (int socket, int level, int option_name, const void *option value, socklen_t option_len)
struct sockaddr_in address;
```
socklen_t → datatype (to store length of socket address) (in POSIX standard)

sockaddr_in → It is a structure to hold the server's address.

```c
int addrlen = sizeof(address);
char buffer[BUFFER_SIZE] = {0};
```

Socket creation

```c
// creating a socket FD
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
    perror("socket failed");
    exit(EXIT_FAILURE);
```
these are the macros: AF_INET ↳ 2 (for IPv4); SOCK_STREAM ↳ for TCP protocol is 1.
EXIT_FAILURE ↳ also simply just 1.

Important Status Codes

200 → Success/OK
304 → Not Modified
404 → Not found
400 → Bad Request
403 → Forbidden
500 → Internal Server Error
501 → Not Implemented
401 → Unauthorized Error
302 → Temporary Redirect
301 → Permanent Redirect

Now set Socket options, to reuse the address and port.

```c
if(setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt)))
{
    perror("setsockopt");
    close(server_fd);
    exit(1);
}
```
SO_REUSEADDR → 2; SO_REUSEPORT → this is 15 for many Linux systems.
sizeof(opt) → 4 bytes on most systems.
&opt ↳ 1 : i.e, enable both the options
their binaries should be non-overlapping they are combined using 'OR'
→ to apply multiple options we bind options with bitwise(OR) '|'

```c
//forcefully attaching socket to port
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);
```
AF_INET → IPv4 (value 2)
INADDR_ANY → typical value (0.0.0.0) ↳ in hexadecimal (0) in this case.
htons(PORT) ↳ converts PORT (8080) ↳ to network byte order (Big-Endian)

Example: 8080 in hexa 0x1F90
htons swaps byte order to 0x901F i.e. (36895) hexa convert back to decimal

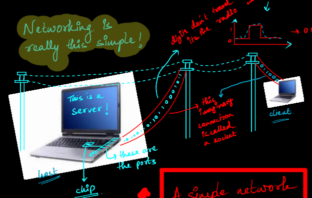
quick revision

		01	23	45			(Big)
		45	23	01			(Little)

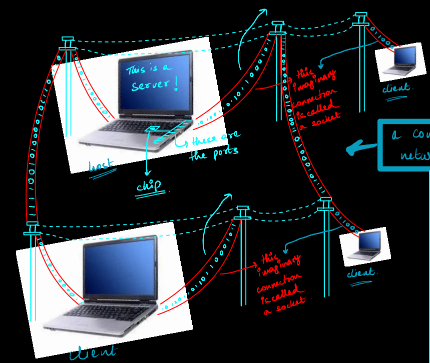
7 layers of OSI model

1. Physical layer
2. Data Link layer
3. Network layer
4. Transport layer
5. Session layer
6. Presentation layer
7. Application layer

① deals with transmission and reception of raw bit streams over mediums like cables
② Handles error detection and correction.
③ responsible for addressing and routing ensuring data packets are delivered across multiple networks.
④ ensures reliable data transfer of packets and reports if some error occurs or data is lost on the way.
⑤ manages connections and sessions across different applications, also maintains life time of data packets.
⑥ translates the bit streams to a format which is human interpretable. i.e, a video or text.
⑦ this layer is the high level layer, i.e. a 'email sent' is shown in this layer. Implementation is abstracted away.



A simple network connection



a complex network

Keep in mind mostly data travels in packets, ie,

Suppose this is a file word.mp3

it is broken down into packets,

p1	p2	p3
p4	p5	p6
p7	p8	p9
p10	p11	p12

The packets are sent p1 → p12. and if the packets are sent successfully success (200) status code is returned.

So now we need to bind the socket, to network address and port.

```c
if (bind(server_fd, struct sockaddr *)&address, sizeof(address))<0)
{
    perror("Bind failed");
    close (server_fd);
    exit (1);
}
```
0 → Success; &address → port and network address.

Now connection is established, So we listen to the PORT.

```c
if( listen(server_fd, 3)<0)
{
    printf("listen");
    close(server);
    exit(1);
}
```

Now we need to accept incoming connections.

```c
if((socket= accept(server, struct sockaddr *)&address, (socklen_t*)&addr_len))<0)
{
    printf("failed");
    close(server_fd);
    exit(1);
}
```

Now all we need is to read and respond.

```c
read(new_socket, buffer, sizeof(buffer))
printf("received: %s", buffer);
```
you can send a response back like,

```c
char *response = "HTTP/1.1 200 OK\n Content-Type: text/plain\n
Content-length: 13\n Hi Debasis!\n";
// standard HTTP response.

send(new_socket, response, strlen(response), 0);
printf("Response sent");

// after sending a response or sending data, simply close the connection
close(server_fd);
close(new_socket);
return 0;
}
```