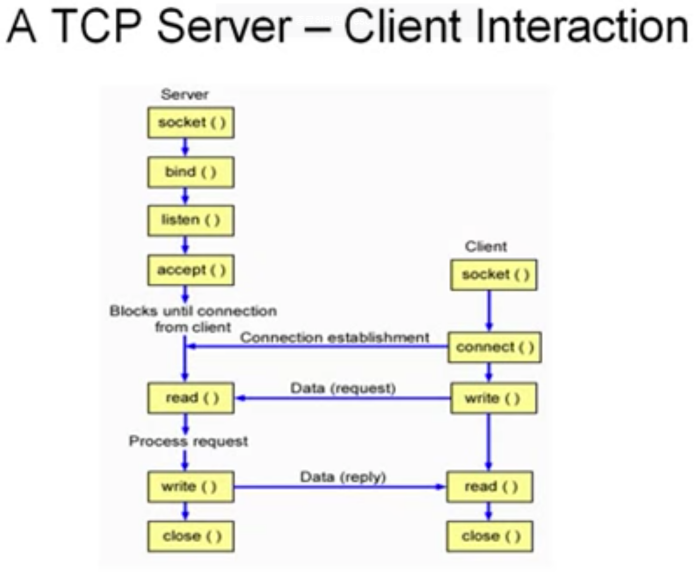
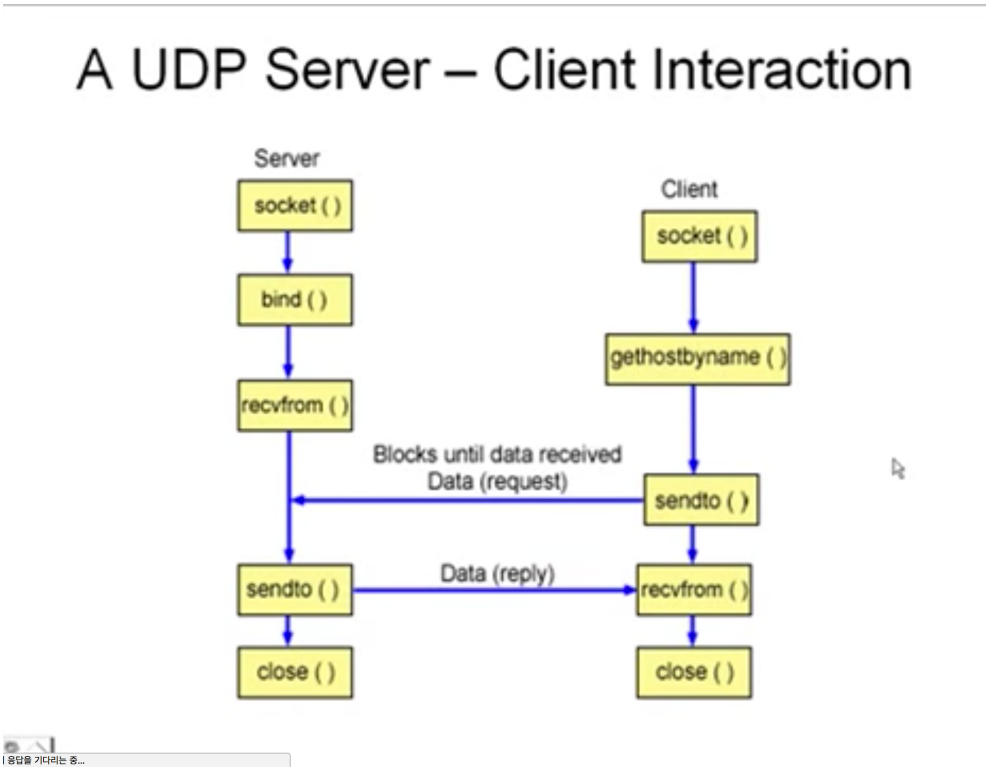
Part 1 : Basics and Socket APIs

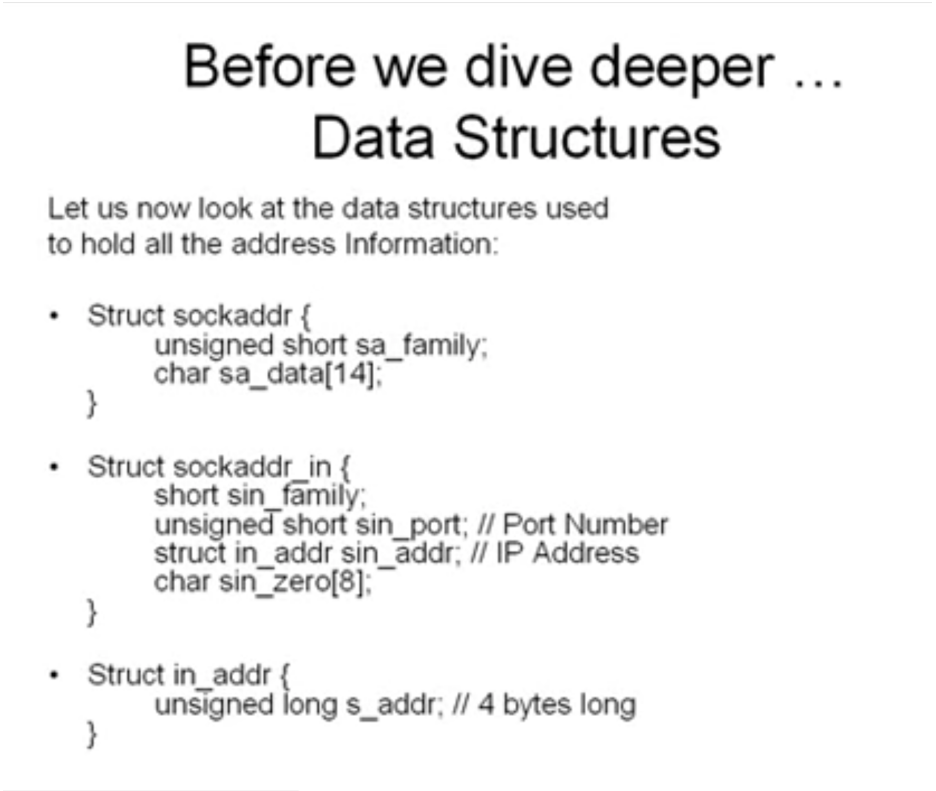
- Why socket programming?
  - “to build any networked application”
  - www,ftp,p2p
- Network programming? Telephone analogy
  - a telephone call over a “telephony network” work as follows :
    - both parties have a telephone installed
    - a phone numver is assigned to each telephone
    - turn on ringer to listen for a caller
    - caller lifts telephone and dials a number
    - telephone rings and the receiver of the call picks it up
    - both parties talk and exchange data
    - after convercation is over they hang up the phone
- Dissceting the analogy
  - a network application works as follow :
    - an endpoint(telephone) for communication is created on both ends
    - an address(phone no) is assigned to both ends to distinguish them from the rest of the network
    - one of the endpoint(caller) initiate a connection to the other
    - the other end(receiver) point waits for the communication to start
    - once a connection has been made, data is exchanged(talk)
    - once data has been exchanged the endpoints are closed(hang up)
- Socket coding
  - Socket() : endpoint for communication
  - Bind() : assign a unique telephone
  - Listen() : wait for a caller
  - Connect() : dial a number
  - Accept() : receive a call
  - Send() , Recv() : talk
  - Close() : hang up
- The Client -server model
  - server : an entity which is a provider of information
  - client : an entity which is a seeker of information
  - example : apache is a web server providing web pages(information) and internet explorer is a web client which requests those pages from the server
  - in the socket programming world almost all communication is based on the client-server model
  - the server stars up first and waits for a client to connect to it. after a client successfully connects, it requests some information. the server serves this information to the client. the client then disconnects and the server waits for more clients.



Part 2 : Difference Between TCP and UDP



- Data Structures



- Byte ordering
  - byte ordering or Endianess is the attribute of a system which indicates whether integers are stored / represented left to right or right to left
  - Ex 1) short int x = 0xAABB(hex)

- this can be stored in memeory as 2 adjacent as either(0xaa,0xbb) or as (0xbb,0xaa)
- Big Endian :
- byte value : [0xAA] [0xBB]
- Memory : [0] [1]
- Little Endian :
- byte value : [0xBB] [0xAA]
- Memory : [0] [1]
- Ex 2) int x=0xAABBCCDD
- This 4 byte long integer can be represented in the same 2 ordering :
- Big Endian :
- byte value : [0xBB] [0xAA] [0xCC] [0xDD]
- Memory : [0] [1] [2] [3]
- Little Endian :
- byte value : [0xDD] [0xCC] [0xBB] [0xAA]
- Memory : [0] [1] [2] [3]
- All network data is sent in big endian format
- In the networking world we call this representation as network byte order and native representation on the host as host byte order.
- we convert all data into network byte order before transmiision

part 3 : utility functions

- some utility functions
- byte ordering
- host byte oreder to network byte order : htons(), htonl()
- network byte order to host byte order : ntohs(),ntohl()
- ip address format
- ascii dotted to binary : inet_aton()
- binary to ascii dotted : inet_ntoa()
- many others exist
- syscalls()
- Socket() : endpoint for communication
- Bind() : assign a unique telephone
- Listen() : wait for a caller
- Connect() : dial a number
- Accept() : receive a call
- Read() / Send() / Sendto()
- Write() / Recv() / Recvfrom()
- Close()
- Socket() : a connection endpoint
- this creates an endpoint for a network connection
- int Socket(int doman, int type, int protocol)
- domain = PF_INET(IPv4 communication)
- type = SOCK_STREAM(TCP), SOCK_DGRAM(UDP)
- protocol = 0 (for our discussion)
- ex) socket(PF_INET,SOCK_STREAM,0); This will create a TCP socket
- the call returns a socket descriptor on success and -1 on an error
- Bind() : attaching to an IP and Port
- a server process calls bind to attach itself to a specific port and ip address
- int Bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen)
- sockfd = socket descriptor returned by socket()
- my_addr = pointer to a valid sockaddr_in structure cast as a sockaddr *pointer
- addrlen = length of the sockaddr_in structure
- ex)

```
struct sockaddr_in my;

my.sin_family = PF_INET;
my.sin_port = htons(80);
my.sin_addr.s_addr = INADDR_ANY;
bzero(&my, 8)

bind(sock, (struct sockaddr *)&my, sizeof(my));
```

- Listen() : wait for a connection
  - the server process calls listen to tell the kernel to initialize a wait queue of connections for this socket.
    - int Listen(int sock, int backlong)
    - sock = socket returned by socket()
    - backlog = Maximum length of the pending connections queue
  - Ex) Listen(sock,10) : this will allow a maximum of 10 connections to be in pending state.
- Accept() : a new connection
  - accept is called by a server process to accept new connections from new clients trying to connection to the server
    - int Accept(int socket, (struct sockaddr *)&client, socklen_t *client_len)
    - socket = the socket in listen state
    - client = will hold the new client's information when accept returns
    - client_len = pointer to size of the client structure
  - ex)
    - struct sockaddr_in client;
    - int len = sizeof(client);
    - Accept(sock, (struct sockaddr *)&client, &len);
- Connect() : connect to a service
  - Connect is called by a client to connect to a server port

```
Int Connect(int sock, (struct sockaddr *)&server_addr, socklen_t len)
```

sock: a socket returned by socket()
server_addr: a sockaddr_in struct pointer filled with all the remote server details and cast as a sockaddr struct pointer
len: size of the server_addr struct

- Example:

```
connect(sock, (struct sockaddr *)server_addr, len);
```

- Send / Recv : finally date
  - Send(),Recv(),Read(),Write() etc calls are used to send and receive data

```
Int send(int sock, void *mesg, size_t len, int flags)

Int recv(int sock, void *mesg, size_t len, int flags)
```

sock = A connected socket
mesg = Pointer to a buffer to send/receive data from/in .
len = Size of the message buffer
flags = 0 (for our purpose)

The return value is the number of bytes actually sent/received.

- Example:

```
char send_buffer[1024];
char recv_buffer[1024];
int sent_bytes;
int recvd_bytes;

sent_bytes = send(sock, send_buffer, 1024, 0);
recvd_bytes = recv(sock, recv_buffer, 1024, 0);
```

- Closer() : bye bye
  - Close signals the end of communication between a server-client pair. this effectively closes the socket.
    - int close(int sock)
    - sock = the socket to close
    - ex) close(sock);