DATE
PAGE

Unit-2

Byte Ordering :- Byte order refers to the order of digits in computer words at least 16 bits long. It tells us how bytes are arranged when sending data over a network. It is an important concept in networking

A byte (8 bits) has a limited range of 256 values. When a value is beyond this range, it has to be stored in multiple bytes.

For Ex:- A 16 bit Number has two bytes. one is called lower byte and the other is the higher order byte.

Suppose we have two memory address An and An+1.

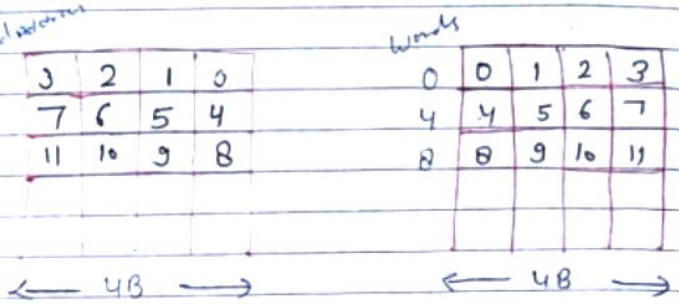
Which of the high-order and low order bytes goes in An?

There are two ways to store this value :-

1) **Little Endian** :- In this scheme,

low-order byte is stored on the starting address (A) and high order byte is stored on the next address (A+1) Ex- Intel 'x 86 family

2) Big Endian :- In this scheme, high order byte is stored on the starting address (A) and low order byte is stored on the next address (A+1) It is known as "Network Byte Order" the TCP/IP Internet protocol also uses big endian. Ex- Motorola 68K



Little Endian Big Endian

Example :- Suppose we have two word address :

1) 1A C8 B2 46
2) FD 8C 1E DF

i) Big Endian :-

0	1A	C8	B2	46
4	FD	8C	1E	DF

Suppose we have to calculate the address of word 1 then,

1A C8 B2 46
00011010 11001000 10110010 0100 0110
= 449,360,454

ii) Little Endian :-

0	46	B2	C8	1A
4	DF	1E	8C	FD

46 B2 C8 1A
01000110 10110010 11001000 00011010
= 1,186,121,754

Now you can observe these two nu are different. So the Numbers va in the we of different Architec

⇒ Some processors support both or and are called Bi-Endian
Ex- PowerPC and Itanium

⇒ Since Computers using different byte ordering and exchanging data, have to operate correctly on data, the Convention is to always send data on the network in big-endian format. We call this Network Byte Ordering and the Ordering used on a computer is called Host-byte Ordering.

Byte Ordering Conversion functions :-

A host system may be little-endian but when sending data into the network, it must Convert data into big-endian format. Likewise, a little-endian machine must first Convert network data into little-endian before processing it. Four common functions to do these Conversions are (for 32-bit long and 16-bit short) as follows :-

Function	Description
htons()	Host to Network Short
htonl()	Host to Network long
ntohl()	Network to Host Long
ntohs()	Network to Host Short

1) htons() :- This function Convert 16 bits (2 byte) quant from host byte order to network byte order.

2) htonl() :- This function Conver 32 bits (4 byte) quanti from host byte order to netwo byte order.

3) ntohs() :- This function Conv 16-bit (2-byte) quan from network byte order to host by order.

4) ntohl() :- This function Converts 32 quantities from network by order to host byte order.

Unit-II

1. What is system calls and write the services provided by system calls?

Ans → An interface between process and operating system.
It provides :-
(i) the services of the O.S. to user programs via Application Program Interface (API)
(ii) An interface allows user-level processes to request services of the O.S.
(iii) System calls are the only entry points into the Kernel system.
Examples: fork(), read(), write(), etc.

* Services provided by system calls :-
(i) Process creation and management.
(ii) Main memory management
(iii) File access, directory and file system management
(iv) Device handling (I/O)
(v) Protection
(vi) Networking etc.

2. What is SDC [System Data structure and Socket]?

Ans → When an application process calls socket(), the operating system allocates new data structure to hold the information needed for communication and yills in a new entry in the process's Socket Descriptor Table (SDT) with a pointer to the data structure.

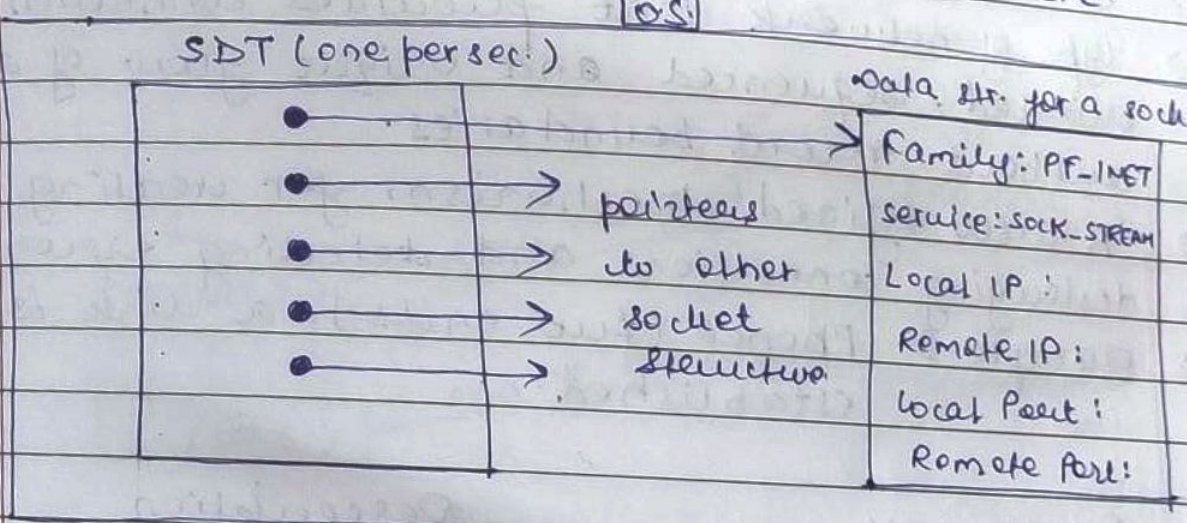
* Features :-

(i) A process may use multiple sockets at the same time. The socket descriptor table is used to manage the socket for this process.

(ii) Different processes used different SDTs.

(iii) The internal data structure for a socket contains many fields, but the system leaves most of them unfilled. The application must take additional procedure calls to fill in the socket data structure before the socket can be used.

(iv) The socket is used for data communication bet? two processes. So, the socket data structure should atleast contain the address information e.g, IP addresses, port numbers etc.



3/ What is socket() ?

Ans → Socket is an interface between applications and the network services provided by the operating systems.

In most client-server applications, sockets are used. The server builds a socket, connects it to a network port and waits for the client to

to connect to it. After creating a socket, the client tries to connect to the server socket. Data is transferred after link is established.

* Types of Socket :-

(i) Datagram Socket :-

→ Type of network in which packets are sent and received without the use of a link.

→ connectionless socket.

→ example :- mailbox → letters (data) are gathered and delivered (transmitted) to a mailbox (receiving socket.)

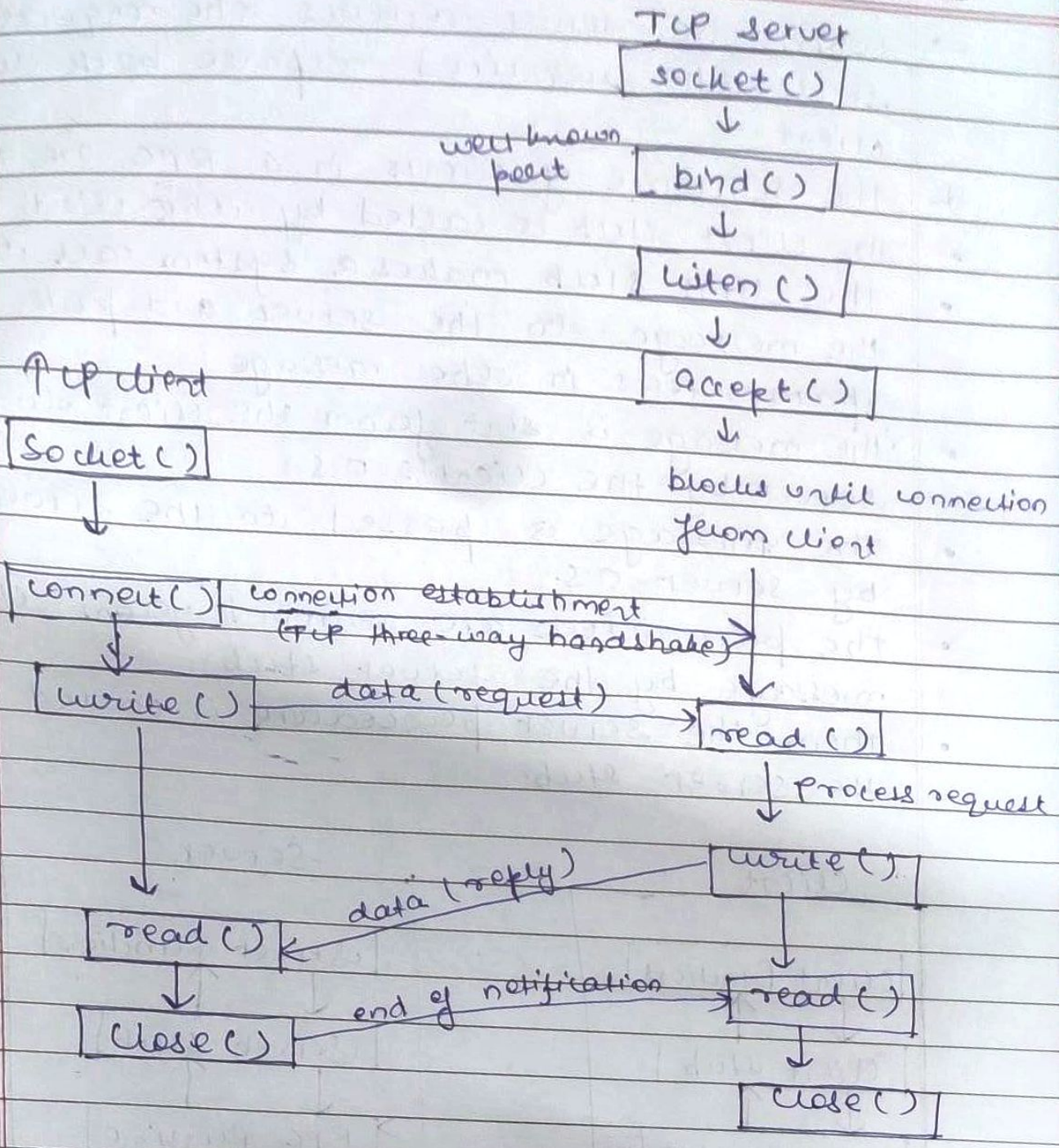
(ii) Stream Socket :-

→ Type of network that provides connection-oriented, sequenced and unique flow of data without record boundaries.

→ A well defined mechanism for creating and destroying connections and detecting errors.

→ example :- Phones (two ends), a link is established.

Function call	Description
• create()	to construct a socket
• bind()	socket identifier
• connect()	Are you ready to make a connection
• listen()	Prepared to send a message
• accept()	Confirmation is similar to accepting a call from a sender
• write()	To send data
• read()	To recieve data
• close()	To make a relationship permanent



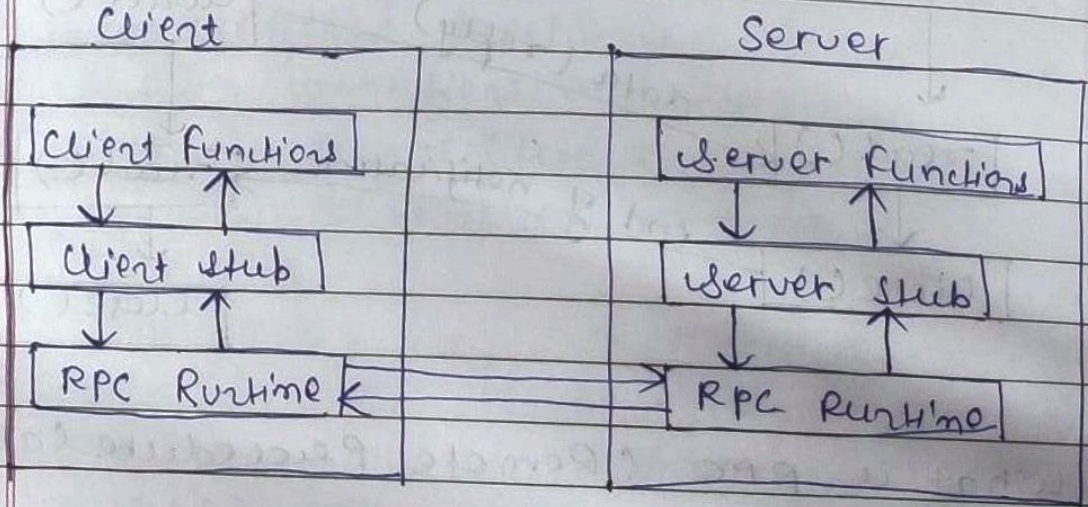
4 What is RPC (Remote Procedure Call)

Ans → A RPC is an interposes communication technique that is used for client-server-based applications also called subroutine or call for function call

→ A client has a request message that the RPC translates and sends to the server. The request may be a procedure or function call to a remote server.

→ when the server recieves the request, it sends the required response back to the client

* The sequence of events in a RPC are as follows:
- The client stub is called by the client
- the client stub makes a system call to send the message to the server and puts the parameters in the message
- The message is sent from the client to the server by the client's O.S.
- The message is passed to the server stub by server O.S.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.



* Advantages of RPC :-

(i) RPC supports process and thread oriented models.

(ii) The internal message passing mechanism of RPC is hidden from the user.

(iii) The effort do re-write and re-develop the code is min^m in RPC.

(iv) RPC can be used in distributed as well as local environment.

(v) Many of the protocol layers are omitted by RPC to improve performance.

* Disadvantages of RPC :-

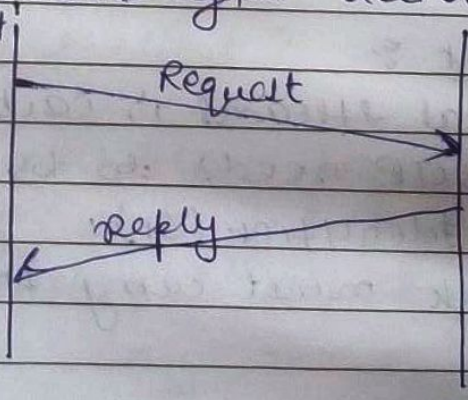
(i) RPC is a concept that can be implemented in different ways. It is not standard.

(ii) There is no flexibility in RPC for h/w architecture It is only interaction based.

(iii) There is an ↑ in cost because of RPC

* RPC mechanism consists of two key components:-

- A protocol that controls the messages delivered bet^n the client & the server processes and deals with underlying network's unfavourable characteristics.
- The arguments are packaged into a request message on the client computer, which is then translated back into the arguments on the server machine, and the return value is translated back into the parameters.
- Client server request lost
- Server client reply lost
- Server crashes after receiving a request.
- Client crashes after delivering the request



Q. Define SCTP?

Ans → SCTP stands for Stream Control Transmission Protocol.
→ A new reliable message oriented transport layer protocol.
→ SCTP are mostly designed for applications that have recently been introduced.
→ ~~SCTP~~ These new applications are IUA [ISDN over IP), M2UA and M3UA (telephony signals), H.248 (media gateway control). H.323 (IP telephony) and SIP (IP telephony etc)
→ SCTP combines best features of UDP & TCP.
→ It preserves message boundaries and at the same time. detects data lost, duplicate data and out-of-order data.
→ It also has congestion and flow control mechanisms.

* Features of SCTP :-

(i) Transmission Sequence Number :-
The unit of data is SCTP is a DATA chunk that may or may not have a one-to-one relationship with the message coming from the process because of fragmentation.

(ii) Stream Identifier :-
SCTP has several streams in each association. Each stream in SCTP needs to be identified by using a stream identifier (SI).
Each data chunk must carry the SI in its header.

(iii) Stream Sequence Number :-
When a data chunk arrives at the destination SCTP it is delivered to the appropriate stream and in the peroper order.
→ which means in addition to an SI, SCTP defines each data chunk in each data stream with a stream sequence number (SSN)

(iv) Packets :-
In SCTP data is carried as data chunks, control information is carried as control chunks.

(v) Flow control :-
SCTP implements flow control to avoid overwhelming the reciever.

(vi) Error control :-
SCTP implements error control to peroide reliability. TSN no. & acknowledgement no. are used for error controls.

(vii) Congestion control :-
SCTP implements congestion control to determine how many data chunks can be injected into the network.

6. Define TCP client ?

Ans → The TCP client class peroides simple methods for connecting, sending and receiving stream data over a network in synchronous blocking mode.

→ ~~In order~~ TCP/IP employs the client-server demonstration in which a client is given a benefit (like sending a webpage) by another computer (a server) within the network.

→ Each client request is considered new since it is irrelevant to past requests. Being stateless liberates up network paths so they can be utilized continually.

→ The transport layer itself is stateful. It transmits a single message and its connection remain open until all the packets in a message have been received and reassembled at the destination.

→ The TCP/IP model differs from the OSI model designed after it.

(uses)

* Applications of TCP/IP :-

(i) Simple Mail Transfer Protocol (SMTP)

→ helps to send email to another email address.

(ii) File Transfer Protocol (FTP)

→ used for sending large files

(iii) Dynamic Host configure Protocol (DHCP)

→ assigns the IP address.

(iv) Telnet → Bidirectional text communication via a terminal application.

(v) HyperText Transfer Protocol (HTTP)

→ used to transfer the web-pages

(vi) Domain Name System (DNS)
→ translates the website name to IP addresses

(vii) Simple network Time Protocol (SNTP) :-
→ provides time of a day to the network devices

* Benefits of TCP/IP :-
(i) It permits cross-platform communications among heterogeneous network.
(ii) It can be utilized by any individual or organization.
(iii) It may be versatile, client server engineering. This permits system to be ~~used~~ included without disturbing the current services.

* Challenges of TCP/IP :-
(i) It is not generic in nature
(ii) It doesnot clearly isolate the concepts of services, interfacing and protocols. So, it isn't appropriate do potray unused advances in modern networks.
(iii) It does not recognize bet^n the data link layer and the physical layer.

7. Define UDP ?
→ very simple protocol.
→ connectionless
→ unreliable transport protocol
→ UDP faster than TCP
→ located bet^n Application layer and network layer in TCP/IP protocol suite.

→ Process - process communications using port numbers
→ No overhead acknowledgement
→ Overhead is min^m.
→ Performs error checking (checksum)
→ Good for video streaming
→ Used for DNS, voice over IP, NFS and online games.

* Limitations of UDP :-
(i) Uses no handshake mechanism
(ii) UDP has no congestion control and flow control.
(iii) No acknowledgement for received packets.
(iv) No compensation for lost packets.
(v) Packets can arrive out of order

* UDP services :-

(i) Process - to - Process communication :-
→ using sockets with the combination of IP address and port numbers.

(ii) Connectionless services
→ UDP is an independent datagram.
→ The user datagram is not numbered.

(iii) UDP can't send a stream of data. Hence the message should fit in one user datagram (less than 65,507 bytes).

(iv) Checksum → contains three parts.
• Pseudoheader → part of header of IP header encapsulated with some fields of 0's

UD → User Datagram

→ If the checksum does not include the pseudoheader a UDP may arrive safe and sound. However if IP header is corrupted it may be delivered to the wrong host.
→ The value of protocol field for UDP is 17.
- UDP header
- Data → communicating from application layer.

(v) Encapsulation :-
→ The process sends the message to the UDP along with a pair of socket address and length of data
→ The UDP then passes to IP adding UDP header.
→ The IP adds its own header along with the value 17 to indicate it is a UDP message and sends to data link layer.
→ The data link layer then adds its own header and passes it to physical layer.
→ The Physical layer encodes bits to electrical signals and sends to remote machines.

(vi) Decapsulation :-
→ When the message arrives at the destination part the physical layer decodes the signals into bits and passes it to data link layer.
→ The datalink layer uses the header to check the data. If there is no error the datagram is passed to IP
→ The IP software does its own checking. If there is no error the header is dropped and the UD is passed to UDP with the sender and receiver IP addresses.

→ UDP uses checksum to check entire datagram. If there is no error the header is dropped and the application data along with sender socket address is passed to the process.

(vii) Queuing :-
→ Port numbers are assigned by OS
→ Each process has one port number, one incoming and one outgoing queue
→ When the process terminates the queue is destroyed.

(viii) The server side uses well known port numbers

(ix) Multiplexing :- (many-to-one relationship)
→ At the sender site, several processes want to send user datagram by using only one UDP
→ UDP accepts messages from different processes and differentiate by port numbers.
→ Adds a header & then sends to the IP

(x) Demultiplexing :- (one-to-many relationship)
→ At the receiver site, several processes want to send user datagram by using only one UDP.
→ UDP receives the UD from IP and drops the header and sends the message to appropriate process based on port numbers.

DATE / /
PAGE

Q. Why would a process want to use UDP when it is powerless?

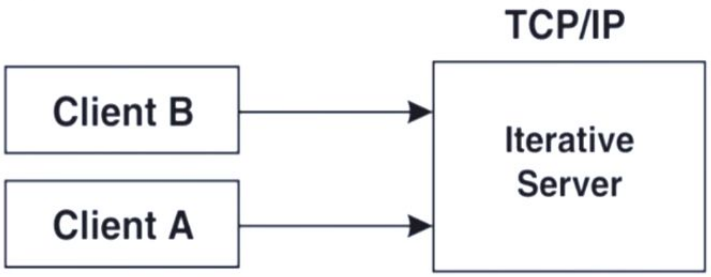
Ans → It is simple protocol with min overhead.
→ Application which uses small messages to be sent without reliability then UDP is best
→ For small messages less no. of interactions is required bet^n sender & reciever for UDP compared to TCP.

Concurrent and iterative servers: There are two main classes of servers, iterative and concurrent.

An iterative server iterates through each client, handling it one at a time. It handles both the connection request and the transaction involved in the call itself. Iterative servers are fairly simple and are suitable for transactions that do not last long.

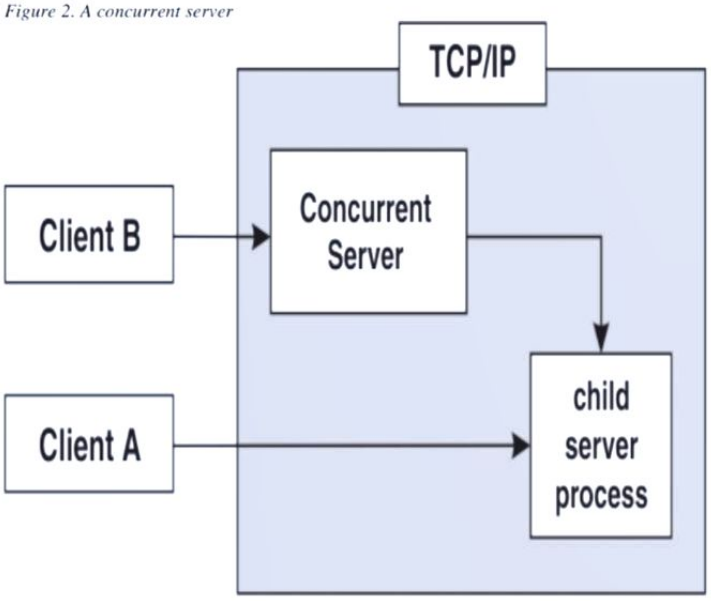
However, if the transaction takes more time, queues can build up quickly. In Figure once Client A starts a transaction with the server, Client B cannot make a call until A has finished.

Figure 1. An iterative server



A *concurrent* server handles multiple clients at the same time. As shown in the figure, Client A has already established a connection with the server, which has then created a *child server process* to handle the transaction. This allows the server to process Client B's request without waiting for A's transaction to complete. More than one child server can be started in this way.

Figure 2. A concurrent server



The simplest technique for a concurrent server is to call the fork function, creating one child process for each client. An alternative technique is to use *threads* instead (i.e., light-weight processes).

```
#include <unist.h>

pid_t fork(void);
```

The function returns 0 if in child and the process ID of the child in parent; otherwise, -1 on error.

4.4. The Socket Interface

The **socket** interface in C provides a mechanism for setting up a communication channel to another host system. For both clients and servers, the initial function call is the same. Processes call `socket()` to request a new socket instance from the OS. As with other forms of IPC such as pipes, sockets are treated as files, so the process receives a file descriptor from the return value. If the socket creation failed, the OS returns a negative value.

C library functions – <sys/socket.h>

```
int socket (int domain, int type, int protocol);
```

Create a socket instance.

The `domain` field is used to declare the intended scope of routing needed; different values here indicate whether the socket will be used for IPv4, IPv6, or local communication. The `type` field determines whether the socket will read and write data as a byte stream, fixed-size messages, or as unprocessed (raw) data. The `protocol` field is typically unused and set to 0; one exception occurs when the client is acting as a **packet sniffer**, an application for capturing and examining packets sent by other processes on a host or network. These applications use **raw sockets**, as described below. For all three fields, the socket header file defines constant values that are used. **Table 4.2** identifies several common constants.

Field	Constant	Purpose
`domain`	`AF_INET`	Use IPv4 addresses
	`AF_INET6`	Use IPv6 addresses
	`AF_LOCAL`	Unix domain socket for IPC
	`AF_NETLINK`	Netlink socket for kernel messages
	`AF_PACKET`	Raw socket type
`type`	`SOCK_STREAM`	Byte-stream communication, used for TCP transport
	`SOCK_DGRAM`	Fixed-size messages, used for UDP transport
	`SOCK_RAW`	Raw data that is not processed by transport layer
`protocol`	`IPPROTO_RAW`	Receive IP datagrams without transport-layer processing
	`ETH_P_ALL`	Receive Ethernet frames without network-layer processing

Table 4.2: Common arguments to the `socket()` function

The `domain` and `type` field constants are defined in the `sys/socket.h` header file. The `domain` fields listed here have another form that replaces the `AF` with `PF`. For example, there are also `PF_INET` and `PF_PACKET` constants. The original use of this different notation was to distinguish an *address family* (`AF`) from a *protocol family* (`PF`). In practice, these values tend to be identical, with the `AF` form more commonly used. For the `protocol` field, the `IPPROTO_RAW` and similar `IPPROTO_*` constants are defined in `netinet/in.h`. The `ETH_P_ALL` and similar constants are stored in `linux/if_ether.h` on Linux systems; other systems do not have these specific values.