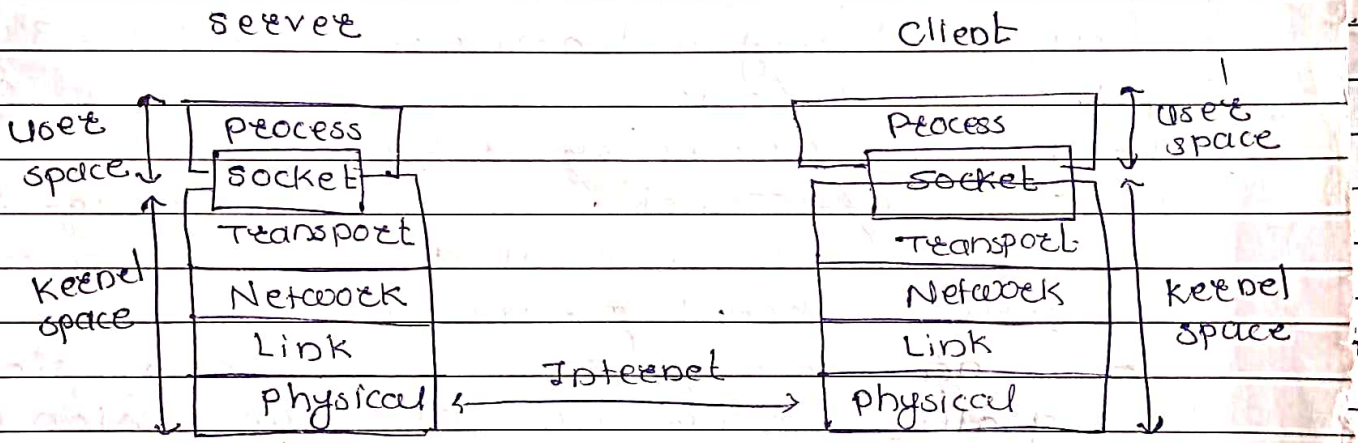
1. Client Server Model & Socket Interface

Socket Interface del & software Design

* Socket

- It is an interface between application process & transport layer
- The application process can send/receive message to/from another application process via socket.

socket Description



- Socket is a one endpoint of two way communication link between two programs running on the network
- Socket is created using socket system call.
- Socket connecting to the network is created at each end of the communication.
- Each socket has specific address. This address is composed of an IP address & port number.
- The server creates a socket, attaches it to a network port addresses then wait for the client to contact it. The client creates a socket & then attempts to connect to the server socket, when the connection is established, transfer of data takes place & Terminate connection

Types

There are mainly two types of socket
i) stream socket
ii) Datagram socket
iii) Raw socket
iv) sequenced packet sockets.

i) Stream Socket
- It is a reliable two way connected communicatio stream.
- If you o/p two items into the socket in the orde "1, 2, 3", they will arrive in order "1, 2, 3" at the opposite side
- They will also be error free.
- Example → Telnet uses the stream socket & als web browsers use the HTTP protocol which uses stream sockets to get the pages.
- To acheive high level of data transmission quality we use the TCP (Transmission Contr Protocol)
- TCP makes sure that data arrives sequentiall & error free.

ii) Datagram socket
- They are connectionless, unreliable
- If some datagram is sent it may arrive out o order, If it arrives, data within the packet will be error free
- Datagram socket also use IP for routing, but they don't use TCP, they use UDP (User Datagra Protocol)
- It is called as connectionless because it doesn' maintain an open connection unlike in stream socket.
- Just build a packet, slap an IP header on it with destination info. & send it out. no connecti needed.

- They are generally used for packet by packet transfers of info.
- Example - ftp, bootp etc.

* TCP socket

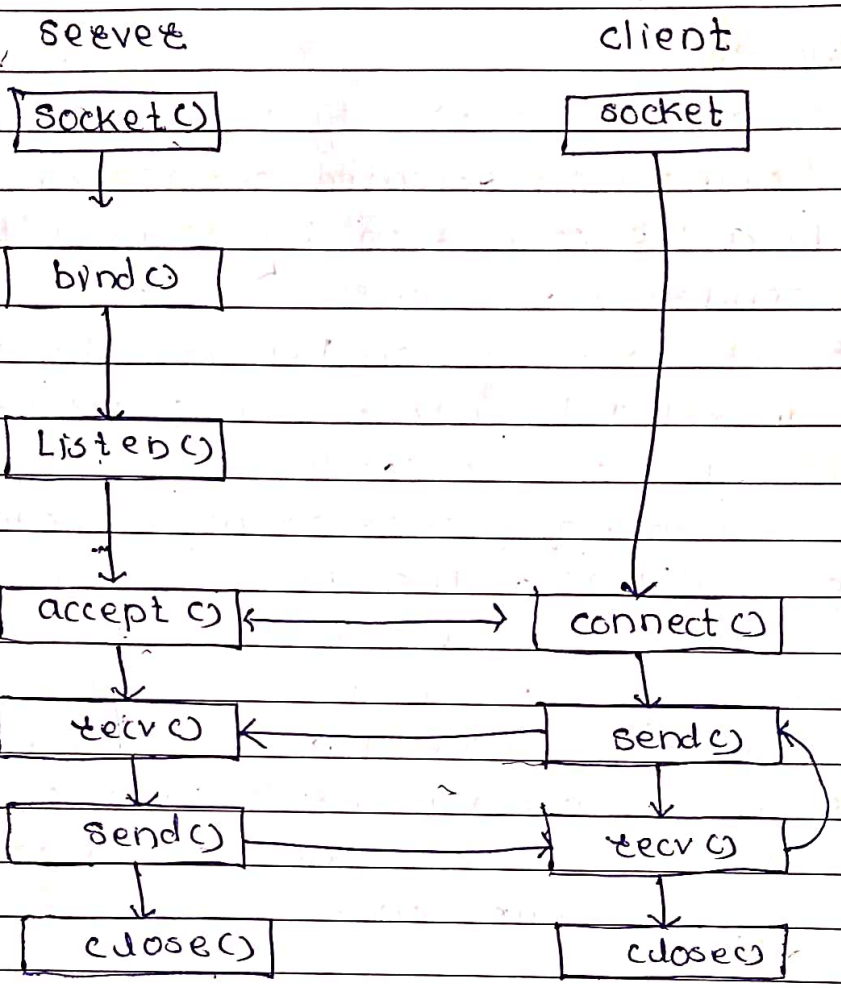
- TCP works on Transport layer of OSI model.
- TCP is a connection oriented service means in order to send data 1st connection has to be established between sender & receiver.
- Use of TCP, transmission between the sender & receiver is reliable, error free of packets.
- TCP also care of flow control means if the sender is sending packet at high speed as compare to the capabilities of the speed receiver, then it takes care of the speed such a way that both sender & receiver speed can follow it.
- TCP is connection oriented so data is sent as byte stream & it gives point to point connection. Point to point means the transmission connection has exactly two connections.

Primitives	Description
SOCKET	Create a new communication end
BIND	Attach local address to socket
LISTEN	willingness to accept connections
ACCEPT	Accept incoming connection
CONNECT	Actively attempt to establish connection
SEND	Send some data over that connection
RECIEVE	Receive some data over that connection
CLOSE	Release the connection.

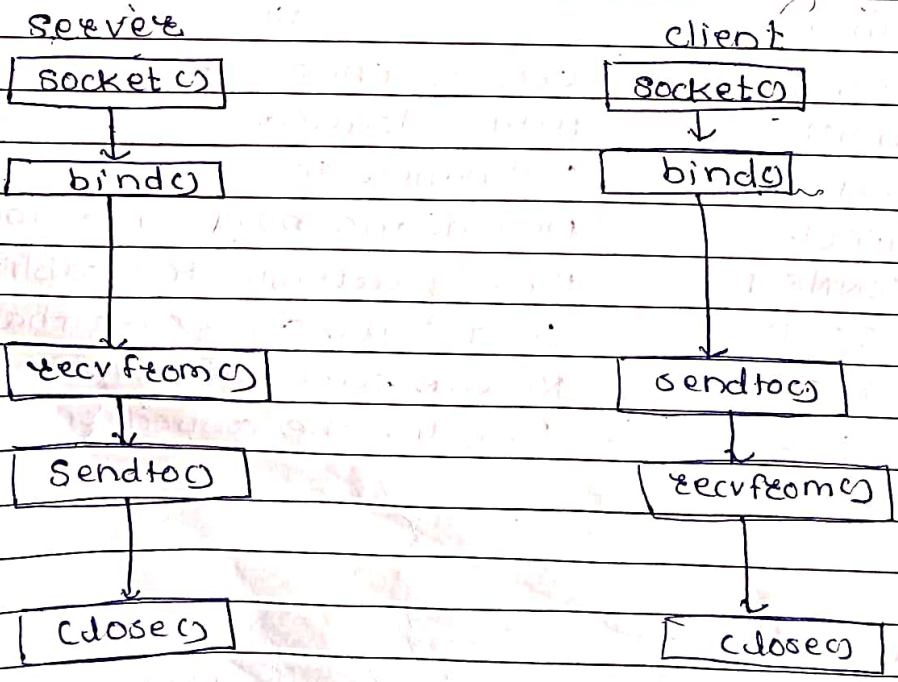
DATE / /

Client Server communication

(1) Stream (TCP)



(2) Datagram (UDP)



- The socket primitive create a new end point for the communication & allocates table space for it within the transport enity.
- Newly created socket do not have addresses, these are assigned using bind primitives.
- once the server has bound an address to a socket remote client can connect to it.
- The LISTEN call allocates space to queue incoming calls for the case that several clients try to connect at the same time.
- In the socket model LISTEN is not blocking call
- To block waiting for an incoming connections, the server executes an ACCEPT primitive.
- when connection request arrives from the client side then connection is established & ACCEPT primitive blocks no more then.
- Then using SEND & RECIEVE primitives, communi can be achieved between server & client.
- Connection release with socket is symmetric.
- when both side have executed a close primiti the connection is released.

Socket c)

Socket system call is

```
# include <sys/types.h>
# include <sys/socket.h>
int socket (int domain, int type, int protocol)
```

- Domain should be set to "AF_INET".
- Domain asks for the family of socket which can be either ARPA internet services, UNIX services or XER network services.
- Next type arguments tells the kernel what kind of socket this is : SOCK_STREAM or SOCK_DGRAM
- Finally, just set protocol to "0" to have socket() choose correct protocol based on the type.

DATE / /

bind()

- Once we have socket then we must assign the address (IP & port no.) to it.
- In order to perform that action we need to use BIND() system call.
- It actually associate addresses to the sockets.

```
# include <sys/types.h>
# include <sys/socket.h>
int bind (int sockfd, struct sockaddr * my_addr,
          int addrlen);
```

- sockfd is the socket descriptor returned by the socket() system call.
- my_addr is a pointer to a struct sockaddr that contains information about your address, namely port & IP address.
- addrlen can be set to sizeof (struct sockaddr)

Listen()

```
int listen (int sockfd, int backlog)
```

- sockfd is the socket file descriptor from the socket() system call.
- backlog is the number of connections allowed on the incoming queue.
- Incoming connections are going to wait in this queue until you accept() them & this is the limit on how many can queue up.
- Most system silently limit this no. to about 20, you can probably get away with setting it to 5/10

DATE / /

connect()

This system call originates from the remote client

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *serv_addr,
            int addrlen);
```

- connect() gives the remote client an opportunity to have connection with the server.
- Catch is here that if listening only then connection can be established.
- sockfd is our friendly neighbourhood socket file descriptor as returned by the socket call, serv_addr is a struct sockaddr containing the destination port & IP address & addrlen can be set to sizeof(struct sockaddr)

accept()

- A remote client will tries to connect to machine on port that is listening on.
- Their connection will be queued up waiting to be accepted, machine call accept() & tell it to get pending connection.
- It'll return a brand new socket file descriptor to use for this single connection.
- Now it has two socket file descriptors for the price of one. The original one is still listening on your port & the newly created one is finally ready to send & recv(). The call is as follow

```
#include <sys/socket.h>
int accept(int sockfd, void *addr, int *addrlen);
```

- sockfd is the listening socket descriptor.
- addr will usually be a pointer to a local struct sockaddr_in. This is where the info. about the incoming connection will go.
- addrlen is a local integer variable that should be set to sizeof(struct sockaddr_in) before its address is passed to accept().

send() & recv()

- These two functions are for communicating over stream socket or connected datagram socket.
- If you to use regular unconnected datagram socket, you'll need.
- int send(int sockfd, const void *msg, int len, int flags);
- sockfd is the socket descriptor you want to send data to msg is pointer to data you want to send & len is the length of that data in bytes.
- Just set flags to 0
- send() returns the no. of bytes actually sent out this might be less than the no. you told it to send.

- recv()

int recv(int sockfd, void *buf, int len, unsigned int flag)

- sockfd is the socket descriptor to read from, buf is the buffer to read info. into len is the maximum length of the buffer & flags can again be set 0
- recv() returns the no. of bytes actually read into buffer or -1 on error

Client Server Model

- In TCP/IP most computer communication protocols provides basic mechanism used to transfer data. so it allows a programmer to establish communication between two application programs & to pass data back & forth, so it provides peer-to-peer communicatin & peer applications can execute on the same machine or on different machines.
- TCP/IP does not provide any mechanismo that automatically create running programs when a message arrives, a program must be waiting to accept communication before any request arrive

Clients & servers

The client server paradigm uses the direction of initiation to categorize whether a program is a client or server.

Client

- "An application that initiates peer-to-peer communication is called a client."
- client is a program running on local machine requesting service from a server.
- client is finite means it is started by user or another application program & terminates when service is complete.
- client opens the communication channel using IP address of the remote host & well-known communication is opened, client sends request & receives response, request & response part repeated several times, all process is finite.
- Each time client application executes, it contacts a server, send a request & awaits response, when the response arrives, the client continues processing.

- Clients are often easier to build than server & usually require no special system privileges to operate.

Server

- Server is any program that waits for incoming communication requests from a client.
- The server receives a clients request, perform the necessary computation & returns the result to the client
- It is a program running on the remote machine providing service to the clients.
- When it starts, it opens door for incoming request from clients but it never initiates a service until it is requested to do so.
- It is a infinite program, when it starts it runs inifinitely unless problem arises.
- Server wait for incoming requests from clients when a request arrives, it responds to the request either iteratively or concurrently.
- Server handles issues of
  i) Authentication → verifying the identity of the client
  ii) Authorization → determining whether given client is permitted to access service the server supplies.
  iii) Data security → guaranteeing that data is not unintentionally revealed / compromised
  iv) Privacy → Keeping info. about an individual from unauthorized access.
  v) Protection → guaranteeing that new application can't abuse system resources.

Standard Vs Nonstandard

- standard application services consist of those services defined by TCP/IP & assigned well known universally recognized protocol port identifiers.
- All others to be locally defined appln services or non standard application services.

- TCP/IP defines many standard application client protocols, most computer vendors supply only handful of std. appln client programs with their TCP/IP s/w.
Ex - TCP/IP includes remote terminal client that uses client standard TELNET protocol for remote login.
- An electronic mail client that uses std SMTP protocol to transfer electronic mail to remote system &
- file transfer client uses the std. FTP protocol to transfer files between two machine.
- Non standard appln range from simple to complex & include such diverse services as image transmissi & video teleconferencing, voice transmission, remote real time data collection, hotel & other online reservation system, distributed database access, weather data distribution & remote control of ocean based drilling platforms.

Stateless vs stateful server.

*stateful →
Information that server maintain about the status of ongoing interactions with client is called stateful.

* stateless
server that donot keep any state info. are called stateless.

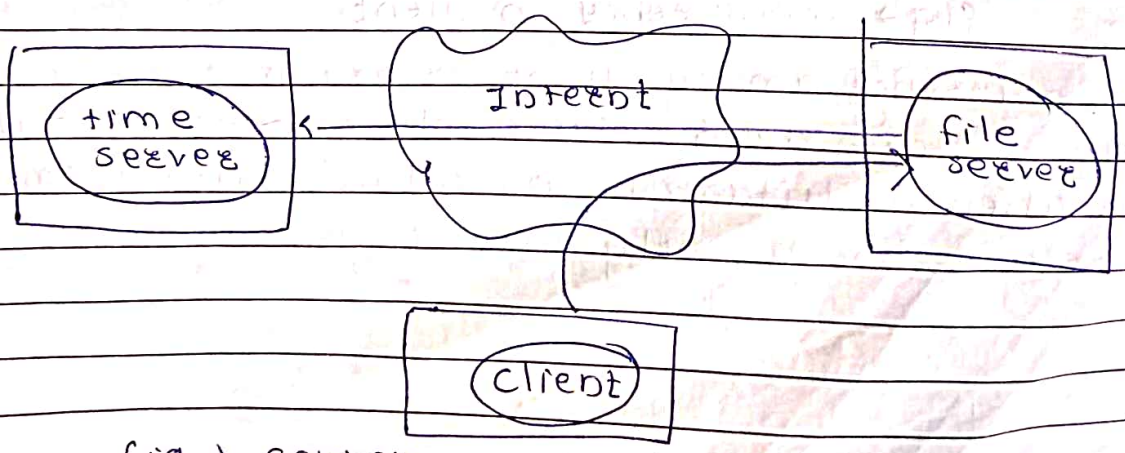


fig → server as client

- A file server program acting as a client to a time server. when the time server replies, the file server will finish it's computation & return result to the original client.

Concurrent Processing in client server software

- concurrency can also occur within given computer system. e.g - multiple users on a timesharing system can each invoke a client application that communicates with an appln on another machine.
- one user can transfer file while another user conducts remote login session.

concurrency in client

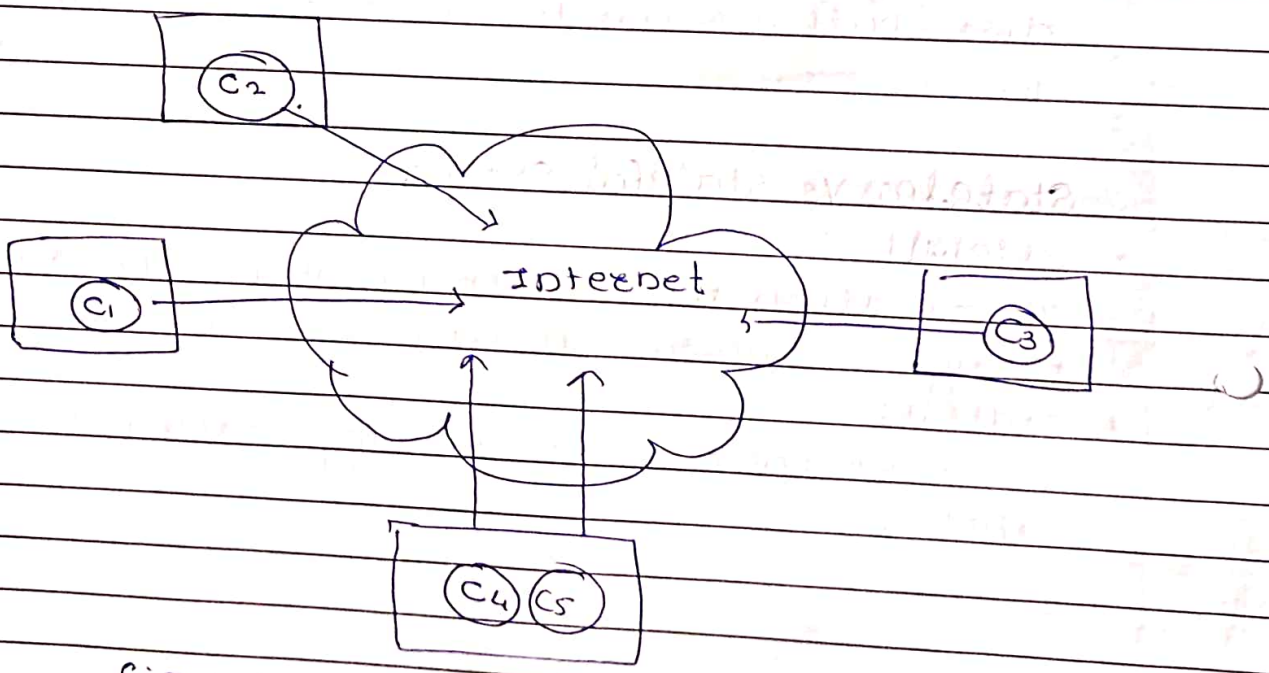


fig → concurrency in client

- concurrency among client programs occurs when users execute them on multiple machines simultaneously or when multitasking os allows multiple copies to execute concurrently on a single computer.

Concurrency in servers

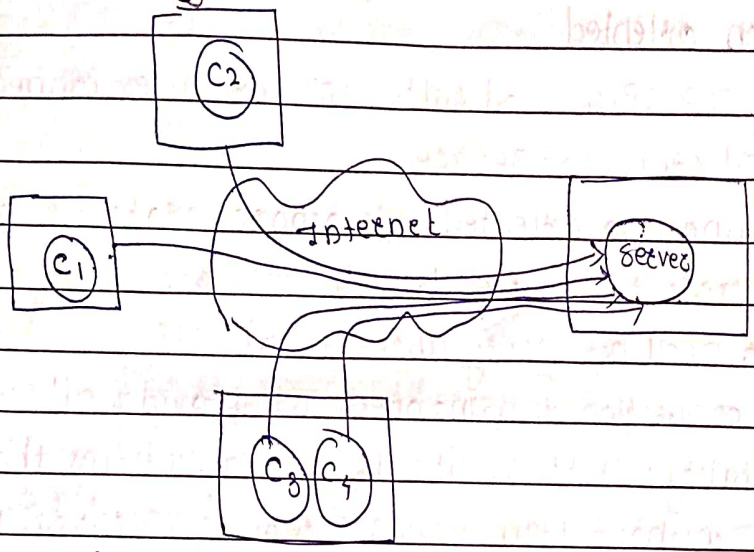


fig → concurrency in servers

- It handle multiple clients at the same time
- server software must be explicitly programmed to handle concurrent requests because multiple clients contact server using it's single, well known protocol port.

Algorithms & Issues in server software design

- To design of server appli software, it need to know the connectionless vs connection oriented server access.
- stateless vs stateful appln & iterative vs concurrent server implementation.

i) concurrent vs iterative servers

- The term concurrent server refers to whether the server handles multiple request concurrently, not to whether the underlying implementation uses multiple concurrent processes.
- Iterative server implementation, which are easier to build & understand, may result in poor performance because they make clients wait for service. In contrast concurrent server implementation which are more difficult to design & build, yeild better performance

TEJ MY BRAIN
DATE / /

ii) Connection oriented vs connectionless

connection oriented

- In the TCP/IP protocol suite, TCP provides connection oriented transport service.
- In connection oriented, transport protocol handles packet loss & out of order delivery problems automatically server need not worry about them
- While connection remains open, TCP provides all the needed reliability. It transmits lost data, verifies that data arrives without transmission errors & reorders incoming packets as necessary.
- When client send request TCP either delivers it reliably or informs the client that connection has been broken similarly, the server can depend on TCP to deliver responses or inform it that connection has broken

connectionless

- UDP provides connectionless service
- UDP does not supply reliable delivery, connectionless transport requires the application protocol to provide reliability if needed, though a complex, sophisticated technique known as adaptive retransmission.
- Design focuses on whether service requires broadcast or multicast communication, because TCP offers point to point communication, it cannot supply broadcast or multicast communication, such services require UDP

iii) Stateless / stateful

- Information that server maintains about the status of ongoing interactions with clients is called state information.
- Servers that do not keep any state info. are called stateless server.

Optimizing stateless server

- Consider a connectionless server that allows clients to read information from files stored on the servers computer
- To keep the protocol stateless, the designer requires each client request to specify file name, a position in the file & no. of bytes to read

Table of info. about files clients are using

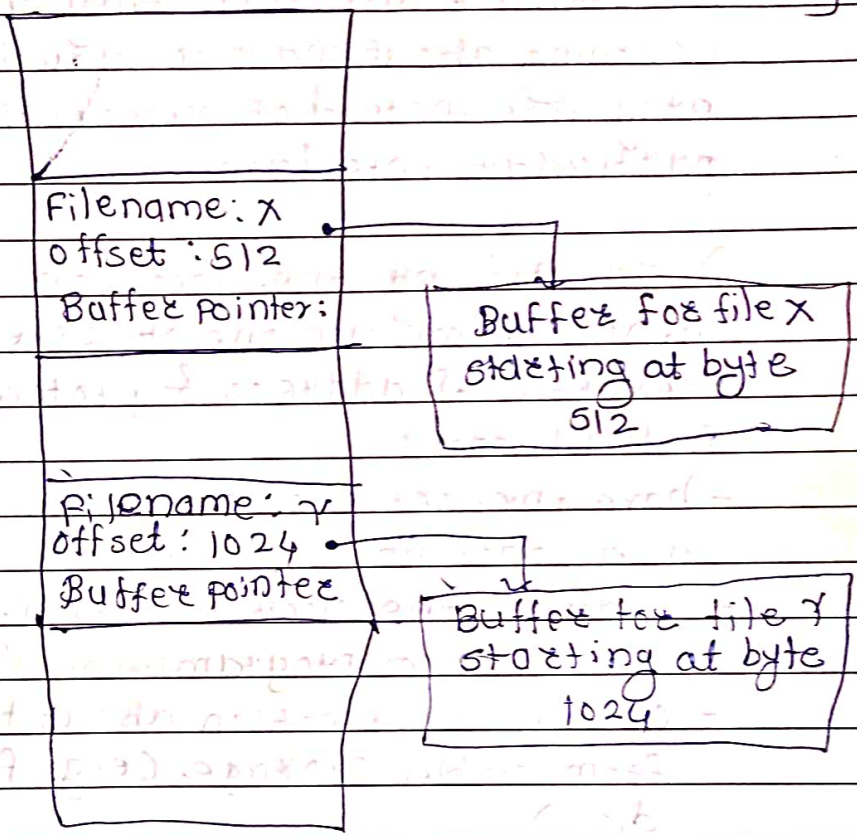


fig. A table of info. kept to improve server performance

The server uses the client's IP address & protocol port number to find an entry. This optimization introduces state info!

- A clever programmer assigned to write a server observes that (1) overhead of opening & closing files is high, (2) the clients using this server may read only a dozen bytes in each request & (3) clients tend to read files sequentially. Furthermore, the programmer knows from experience that the server can extract data from a ~~disk~~ buffer in memory several orders of magnitude faster that it can read data from a disk. So to optimize server performance, the programmer

DATE / /

decides to maintain a small table of file info. as fig. shows.

Algorithms & Issues in client software design

To design client server software design programmers needs to understand the conceptual capabilities of the protocol interface, they should concentrate on learning about ways to structure communicating programs instead of memorizing the details of particular interface.

1) Identifying the location of A server
client s/w can use one of several methods to find a server's IP address & protocol port number.
A client can:
- have the servers domain name or IP address specified as a constant when program is compiled
- Require the user to identify the server when invoking the program.
- obtain information about the server from stable from stable storage (e.g. from a file on local disk)
- use a separate protocol to find a server (e.g. multicast or broadcast a msg. to which all servers respond)

- Allowing the user to specify a server address when invoking client s/w makes the client program more general & makes it possible to change server locations.

Building client s/w that accepts a server address as an argument makes it easy to build extended versions of the s/w that use other ways to find the server address

2) Parsing an address Argument
- User can specify argument on the command line when invoking a client program.
- In most systems, each argument passed to client program consists of character string. The client uses an arguments syntax to interpret its meaning.
  e.g - merlin.cs.purdue.edu
  or 128.10.2.3
  - merlin.cs.purdue.edu smtp
  or merlin.cs.purdue.edu:smtp.

3) Looking Up A domain Name
- A client must specify the address of server using structure sockaddr_in. Doing so means converting an address in dotted decimal notation into a 32-bit IP address represented in binary. converting from dotted decimal notation to binary is trivial.
- The socket interface in BSD UNIX includes library routines inet_addr & gethostbyname takes an ASCII string that contains the domain name for a machine. It returns the address of a hostent structure that contains among other things the hosts IP address in binary.

4) Looking Up a well known port by Name
client of an SMTP mail server needs to look up the well known port assigned to SMTP. To do so the client invokes library function getservbyname which takes two arguments., a string that specifies the desired service & string that specified the protocol being used. It returns a pointer to structure of type servent also defined an include file netdb.h.

5) Port Numbers & Network byte order
- functions getservbyname returns the protocol port for the service in network byte order.
- It is sufficient to understand that getservbyname return the port value in exactly the form needed

for use in the sockaddr_in structure but the representation may not agree with the local machine's representation.
- Thus, if a program prints out the value that getservbyname returns without converting to local byte order, it may appear to be incorrect.

6) Looking Up a protocol by Name:
- The socket interface provides mechanism that allows a client or server to map a protocol name to the integer constant assigned to that protocol name in a string argument & getprotobyname returns the address of structure of type protoent
- If getprotobyname can't access the db or if the specified name does not exist, it returns zero

7. The TCP client Algorithm.

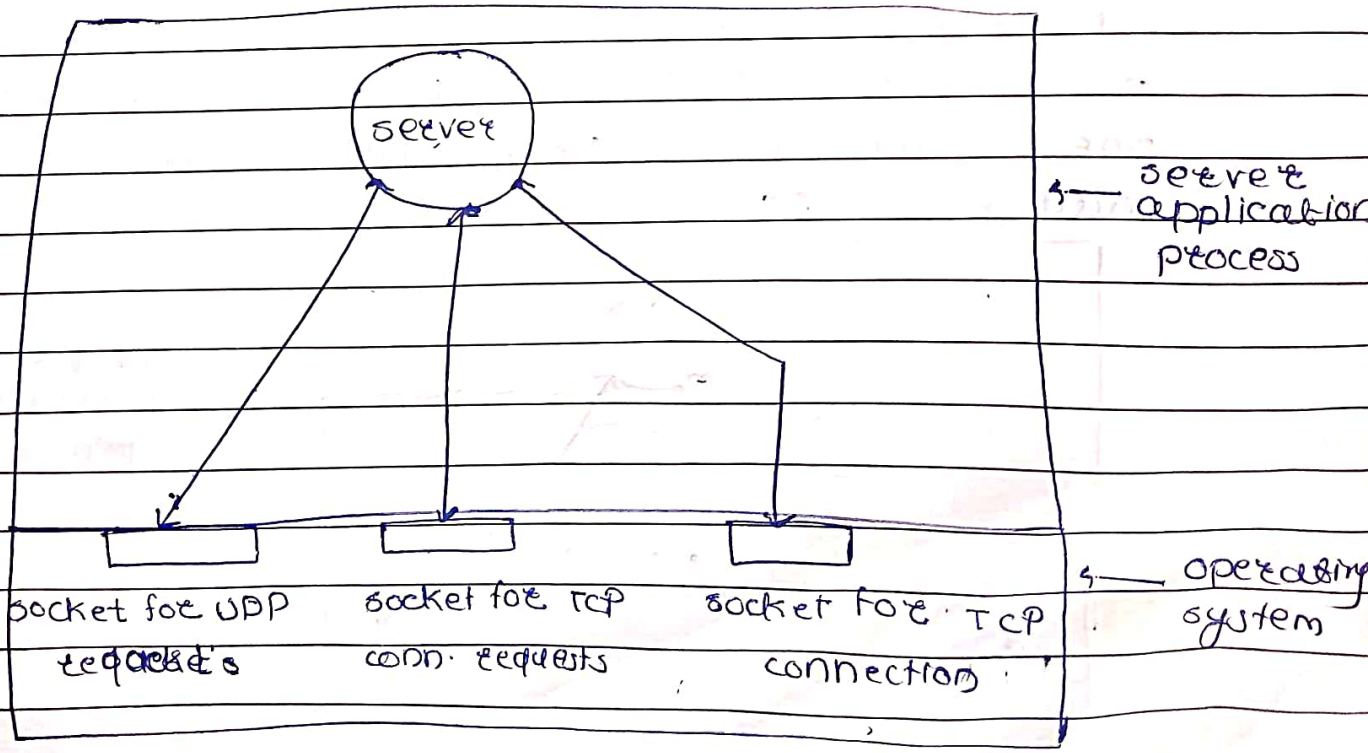
Algorithm
i) Find the IP address & protocol port no. of the server with which communication is desired.
ii) Allocate a socket.
iii) Specify that connection needs an arbitrary unused protocol port on the local machine & allow TCP to choose one
iv) Connect the socket to the server.
v) Communicate with server using the appln-level protocol.
vi) Close connection.

Multiprotocol Server

* Multiprotocol server Design

- Multiprotocol server consists of single process that uses asynchronous I/O to handle communication over either UDP / TCP.
- The server initially opens two sockets ; one that uses a connectionless transport (UDP) & one that uses connection oriented transport (TCP)
- The server then uses asynchronous I/O to wait for one of the sockets to become ready
- If the socket becomes ready, client has requested TCP connection.
- The server uses accept to obtain the new connection & then communicates with the client over that connection
- If the UDP socket becomes ready, client has sent a request in the form of UDP program.
- The server uses recvfrom to read the request & record the sender's endpoint address.
- Once it has computed a response, the server sends the response back to the client using sendto

* structure of an iterative, multiprotocol server



- An iterative, multiprotocol server has at most three sockets open at any given time.
- Initially, it opens one socket to accept incoming UDP datagrams & second socket to accept incoming TCP connection requests.
- when datagram arrives on the UDP socket, the server computes response & sends it back to the client using the same socket.
- when connection request arrives on the TCP socket, the server uses accept to obtain a new connection.
- Accept creates a 3rd socket for the connection, the server uses the new socket to communicate with the client.
- once it finishes interacting, the server closes the 3rd socket & waits for activity on the other two.

* Multiservice servers (TCP, UDP)

- TCP/IP defines a large set of simple services that are intended to help test, debug & maintain a n/w of computers.

- **Connectionless Multiservice server design**

Multiservice server can use either connectionless or connection oriented transport protocol. fig. illustrates one possible process structure for a connectionless multiservice server.

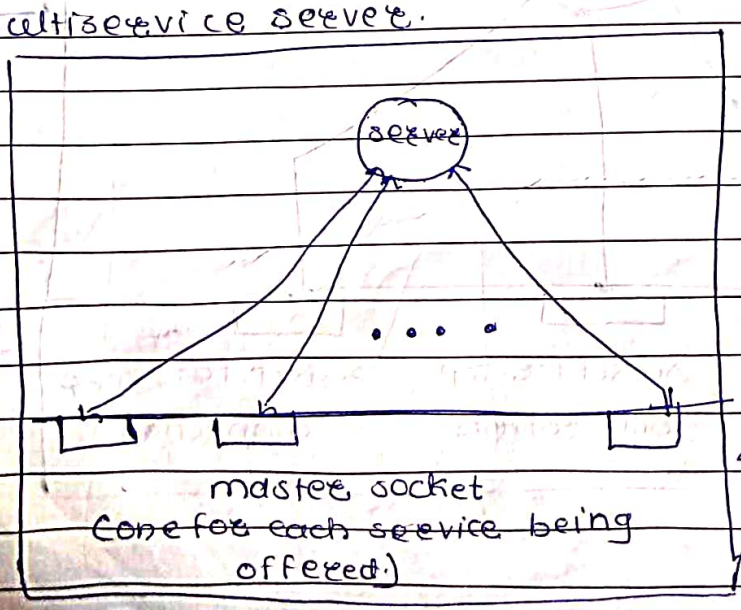


Fig. An iterative, connectionless, multiservice server

- In fig. server opens a set of UDP sockets & binds each to well known port for one of the services being offered.
- It uses a small table to map socket to services.
- For each socket descriptor, the table records the address of a procedure that handles the service offered on that socket.
- The server uses the select system call to wait for a datagram to arrive on any of the socket.
- When datagram arrives, server calls appropriate procedure to compute a response & send reply.
- Because the mapping table records the service offered by each socket, server can easily map socket descriptor to the procedure that handles the service.

-connection oriented, multiservice server design

In principle, such server performs same tasks as a set of iterative, connection oriented server.
- To be more precise, single process in multiservice server replaces the master server processes in a set of connection oriented servers. At top devel, multiservice server uses asynchronous IO to handle it's duties.

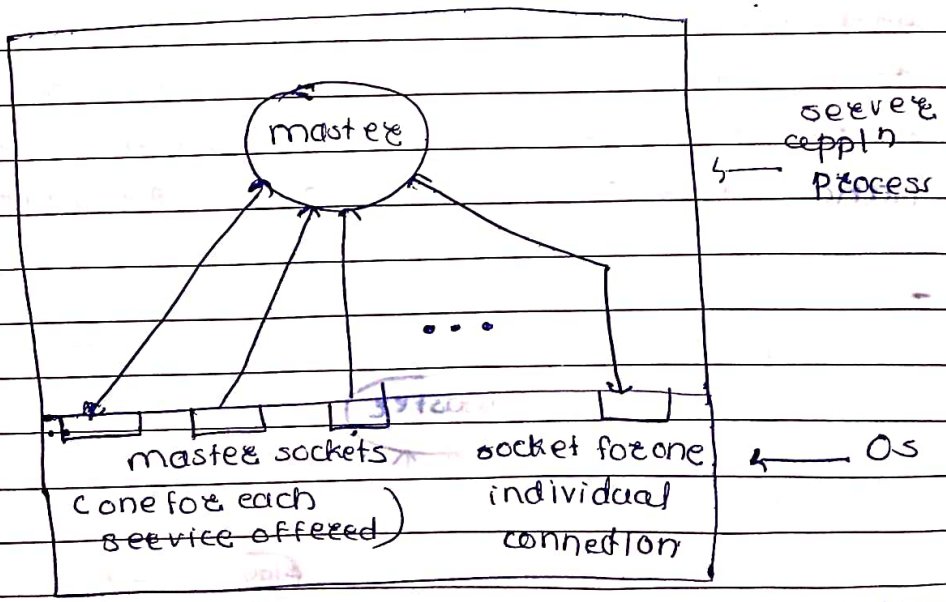
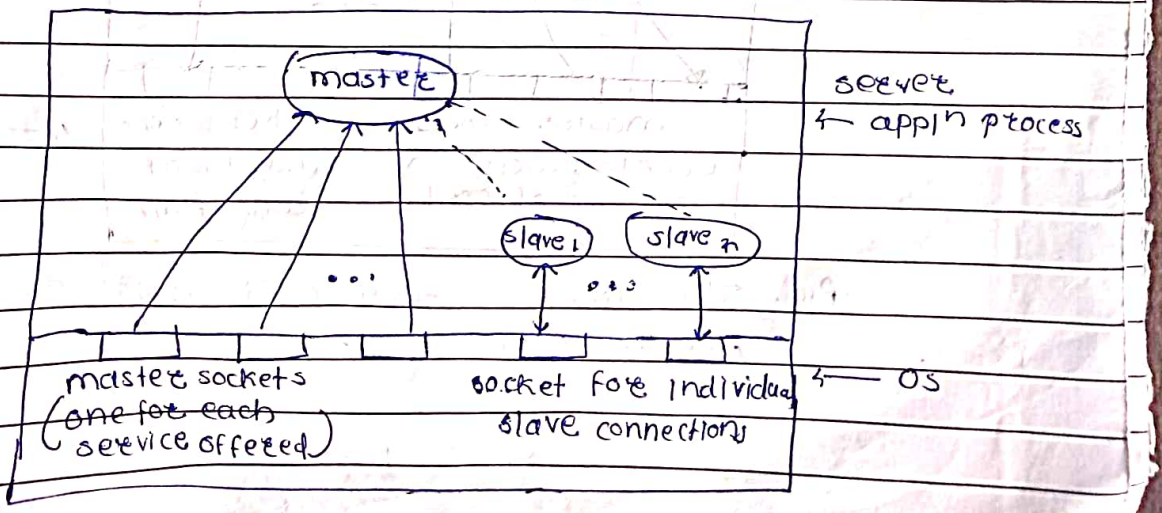


Fig. The process structure of an iterative connection-oriented, multiservic server.

- At any time, the server has one socket open for each service & at most one additional socket open to handle a particular connection.
- When begins execution, multiservice server creates one socket for each service it offers, binds each socket to the well known port for the service & uses select to wait for an incoming connection request on any of them.
- When one of the sockets becomes ready, the server calls accept to obtain new connection that has arrived. Accept create new socket for the incoming connection, the server uses new socket to interact with client & then closes it.
- Thus besides one master socket for each service the server has at most one additional socket open at any time.
- As in the connectionless case, the server keeps a table of mappings so it can decide how to handle each incoming connection. When the server begins, it allocates master sockets. For each master socket, the server adds an entry to the mapping table that specifies socket no. & procedure that implements the service offered by that socket. After it has allocated master socket for each service, the server calls select to wait for connection. Once connection arrives, the server uses mapping table to decide which of many internal procedures to call to handle the service that client requested.

-



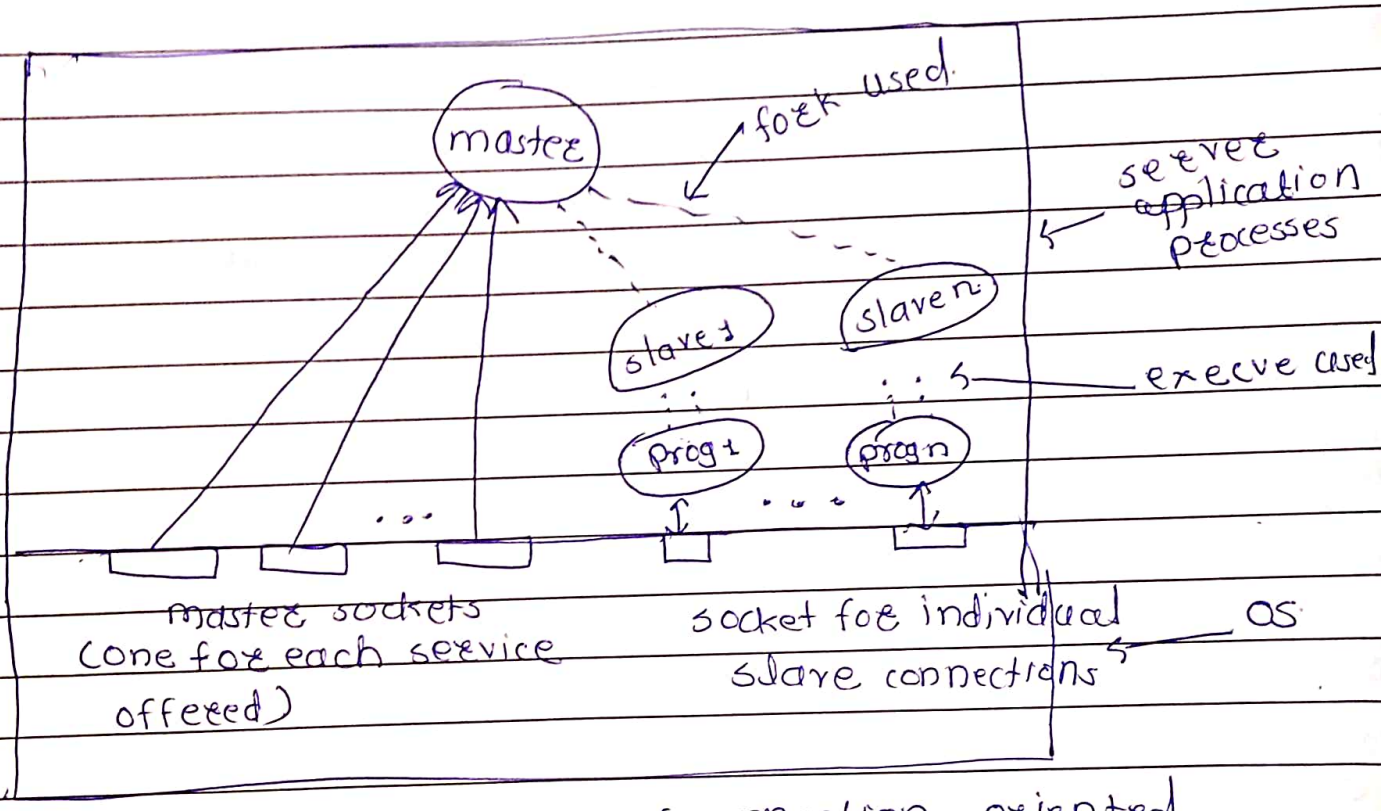


Fig. The process structure of connection-oriented multiservice server that uses execved separate program to handle each connection.