

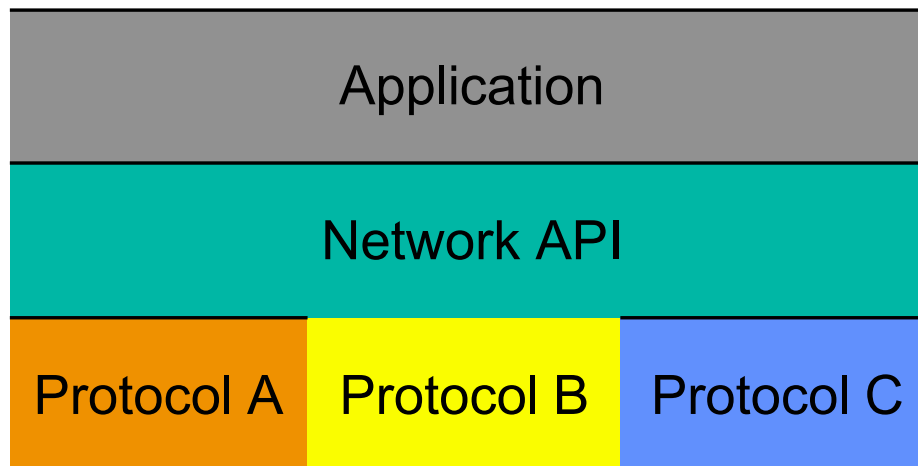
Overview of Network Programming: Sockets

EE450: Introduction to Computer Networks

Professor A. Zahid

Application Programming Interface

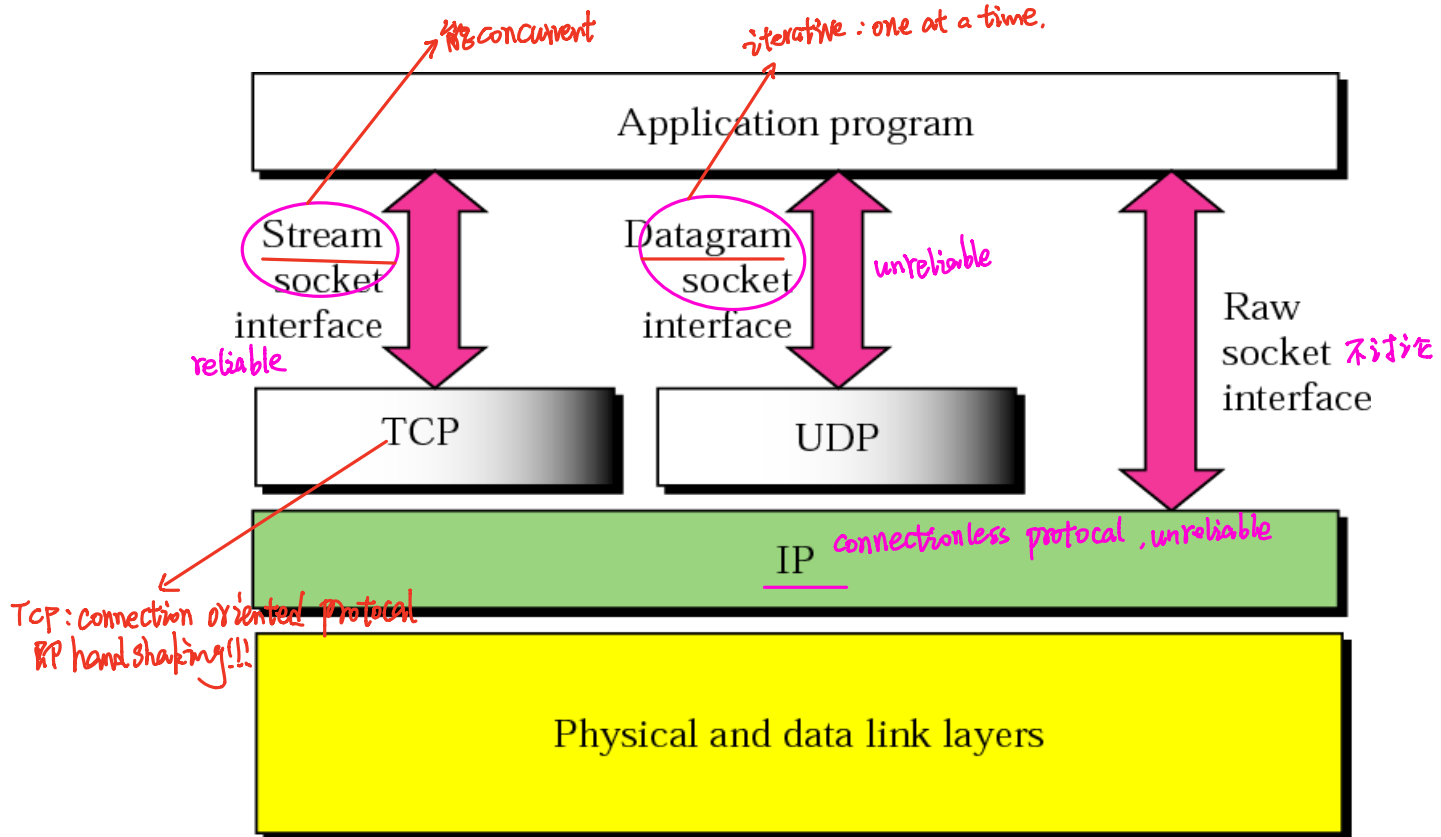
- The services provided by the operating system that provide the interface between application and the TCP/IP Protocol Suite.
12-10-11



API: Sockets

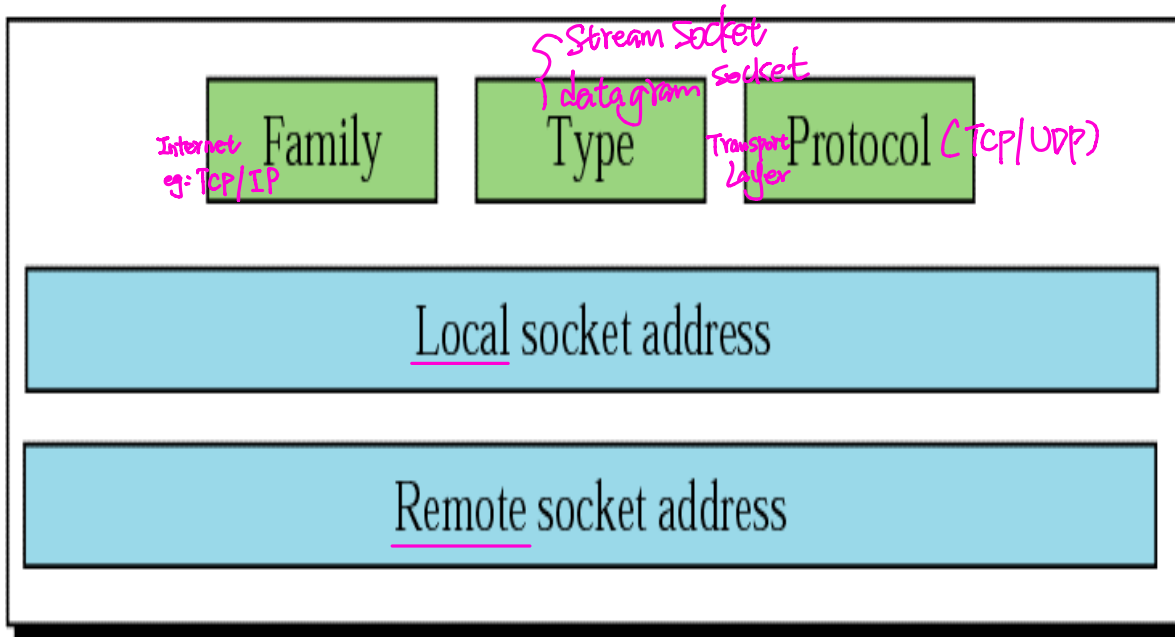
- TCP/IP does not include an API definition.
- There are a variety of APIs for use with TCP/IP:
 - UNIX Sockets
 - Winsock (Windows Sockets)
- A socket is an abstract representation of a communication endpoint.
- A socket allows the application to “plug in” to the network and communicate with other applications
- A socket is uniquely identified by the ^{ID}IP address, ^{Port}Port number and the underlying ^{Protocol}transport layer protocol

Socket Types

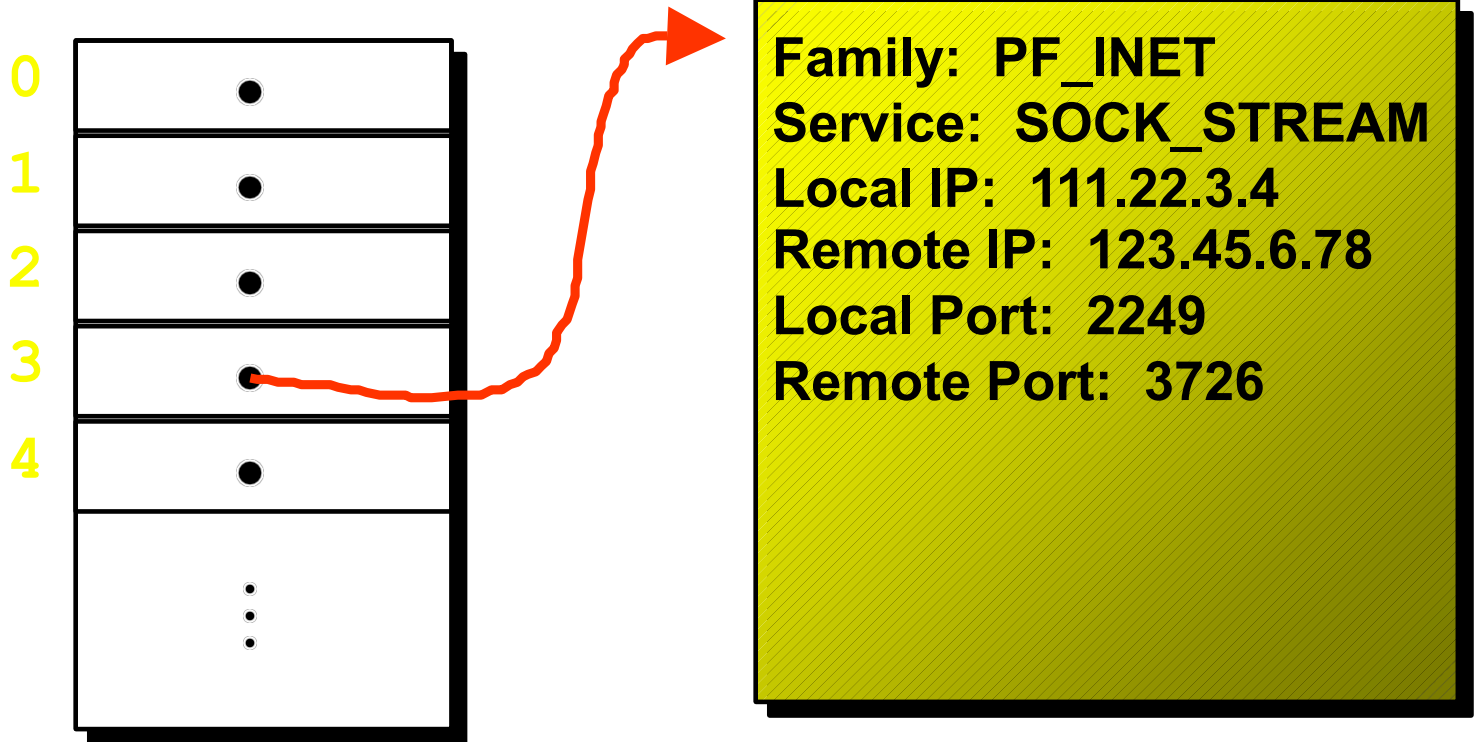


Socket Structure

Socket : file 包括:



Socket Descriptor Data Structure



Tcp Sockets

Client side → actively open

int socket() 联系就关了

int connect() → handshake 告诉TCP想连接, 想保持畅通

create a TCP segment

头部: source port #, he choose Randomly
Destination port #, (APP 告知)

(APP 不用知道)

Server side

刚开始没有 APP, run on server, 无 socket

→ passively open

int socket() 一直开着

int bind() 绑定 port #

int listen() 服务器告诉 client, 42 1st client in the queue.

int accept() 阻塞 阻塞 NDS block, 不返回任何值: 没收到 request from client. 不是 fill client address

产生 pack: { source IP: IP of client
dest IP: IP of server. (APP 告知 int connect)

握手后:

UnBlock 即取得了 client IP 地址, 也知道了 client port #

⑤ for every client, create a child socket

In TCP server, concurrent, serve multiple clients concurrently
无 confusion, 拥有 unique socket.

Local Socket Address { Port # of server
IP of server
Remote Socket Address { Port # of client
IP of client

Parent socket 的目的只是和 client 握手, 不是 socket 用来 communicate
一直握手, server side 则产生 child socket

Local Socket Address

Remote Socket address { Port #
IP

未知 ???

int connect()

TCP will use in the header { source port #
dest port #

IP { source IP
dest IP

⑥ int send()

⑦ int receive()

UDP socket Connectionless 无 handshake.

→ Iterative (one client at a time)

无 parent socket

无 child socket

Basic Sockets API

- Clients and Servers differ in some aspects of their use of API and they are the same in other aspects
- Both client and server programs begin by asking the NOS to "create" a socket. The function to accomplish that is Socket ()
network operating system.
- *→ address*
 - `int socket (int protocol family, int type, int protocol)`
 - Protocol family: `PF_INET` ✓
 - Type: `SOCK_STREAM`, `SOCK_DGRAM` ✓
 - Protocol: `TCP`, `UDP` ✓
- Return value of `Socket ()` is a non-negative integer called Socket Descriptor (or -1 if errors)

TCP Server

- Create a TCP socket using `Socket ()`
- Assign a port number to the socket using `Bind ()`
 - `int bind (int socket, local address, address length)`
 - descriptor* (for `int socket`)
 - server* (for `local address`)
 - port: 16 bits* (for `address length`)
 - well-known* (for `address length`)
 - Local address is the IP address and the port number of the server. It is this address that the client and the server need to agree on to communicate. Neither one need to know the client address.
 - Address length is length of address structure
 - If successful, `Bind ()` returns a 0, otherwise it returns -1
 - If successful, the socket at server side is "associated" to the local IP address and the local port number (*well-known port #*)
- Listen to connections from clients using `Listen ()`
 - `int listen (int socket, int queue limit)`
 - descriptor* (for `int socket`)
 - app to server concurrently* (for `queue limit`)
 - Queue limit is an upper bound on # of clients that can be waiting
 - If successful, `Listen ()` returns a 0, otherwise it returns -1

TCP Server (Continued)

- The socket that has been bounded to a port and marked for listening is never actually used for sending and receiving. The socket (known as the welcoming socket or the "parent" socket).
- "Child" sockets are created for each client. It is this socket that is actually used for sending and receiving
- Server gets a socket for an incoming client connection by calling `Accept ()`
 - `int accept (int socket, client address, address length)`
 - If successful, `accept ()` returns a descriptor for the new socket, otherwise it returns -1

↪ child socket

TCP Server (Continued)

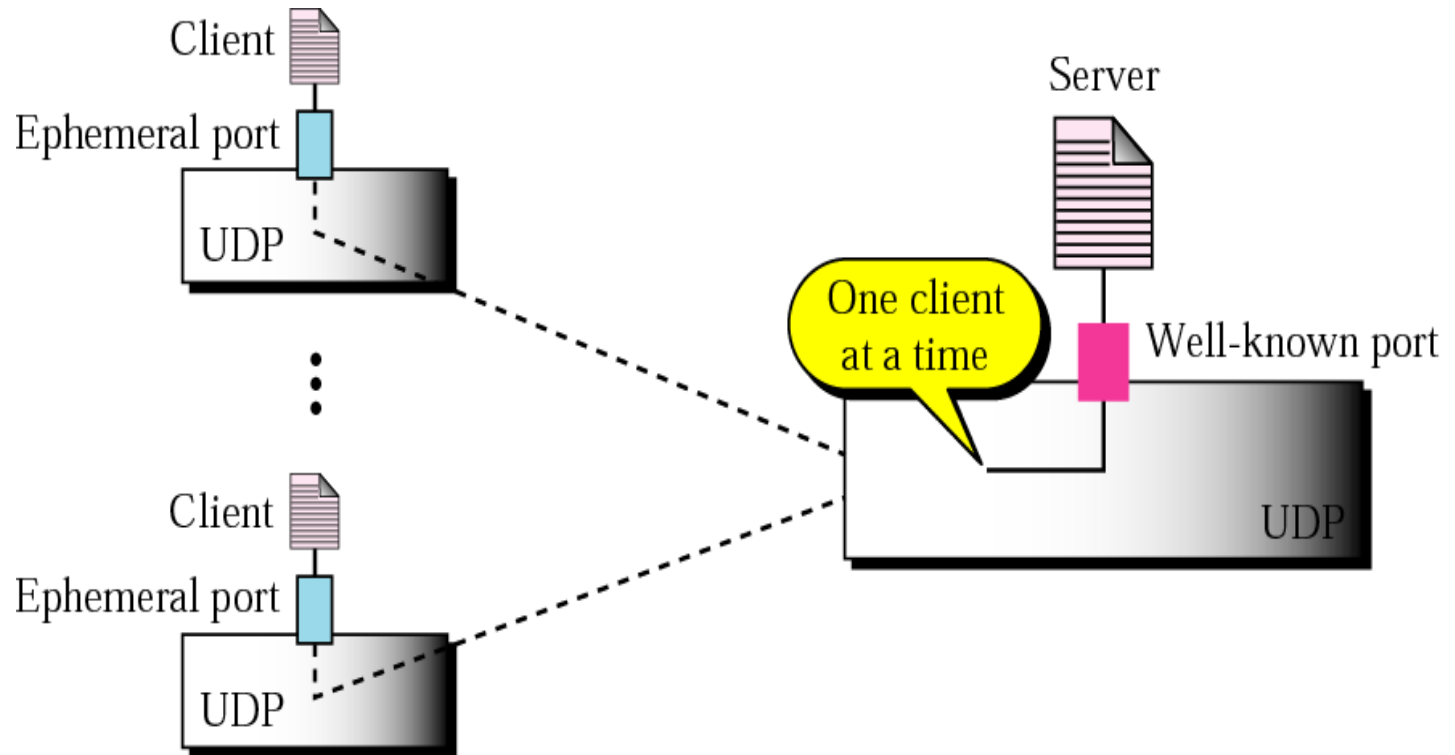
- Once the child socket is created, communications (send and receive) can take place
 - `int send (int socket, message, message length)`
server *descriptor of child socket in parent socket*
 - `int rcv (int socket, rcv buffer, buffer length)`
app. of server via
- When the application is done with the socket, it needs to close it (The child socket, not the parent socket which is passively open to welcome new clients). This is done by calling Close ()
 - `int close (int socket)`
child socket (server)
original socket (client side)

TCP Client

- Create a TCP socket using `Socket ()`
- Establish a connection to the server using `Connect ()`
 - segment* • `int connect (int socket, remote foreign address, address length)`
 - descriptor of client*
remote { *port of server*
IP address }

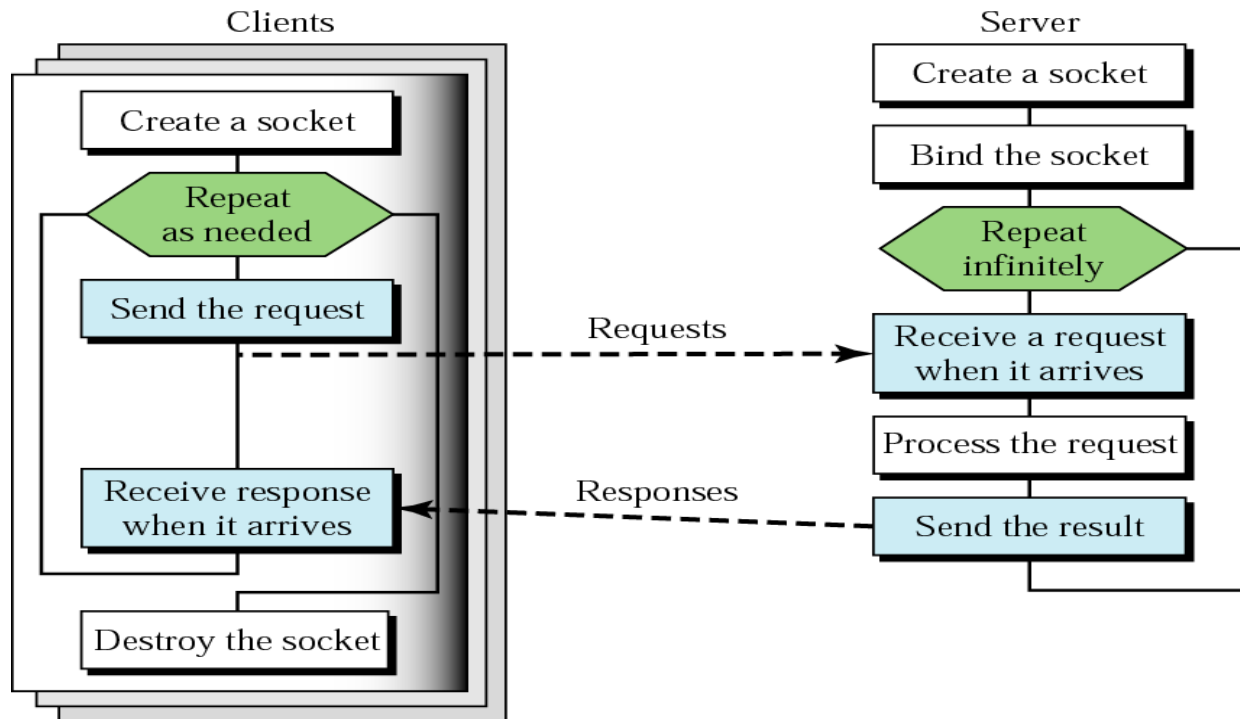
 - Foreign address is the address of the server and the port number of the server (The well-known port number)
- Communications using `Send` and `Recv`
 - `int send (int socket, message, message length)`
 - `int recv (int socket, recv buffer, buffer length)`
- Close the socket using `Close ()`
 - `int close (int socket)`

Connection-less Iterative Server

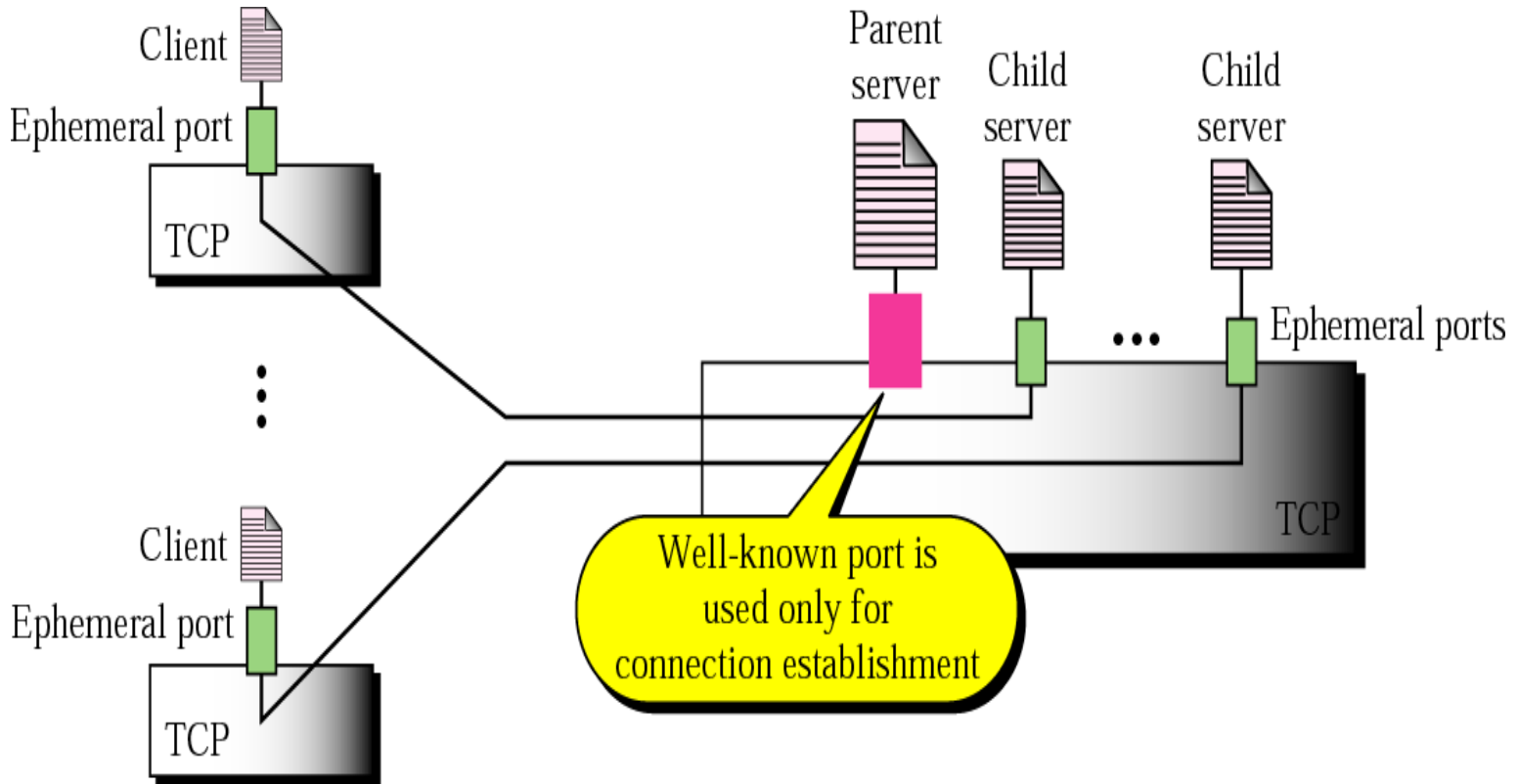


Flow Chart: Socket Interface for Connection-Less Iterative Server

Each server serves many clients but handles one request at a time.



Connection-oriented Concurrent Server



Flow Chart: Socket Interface for Connection-Oriented Concurrent Server

