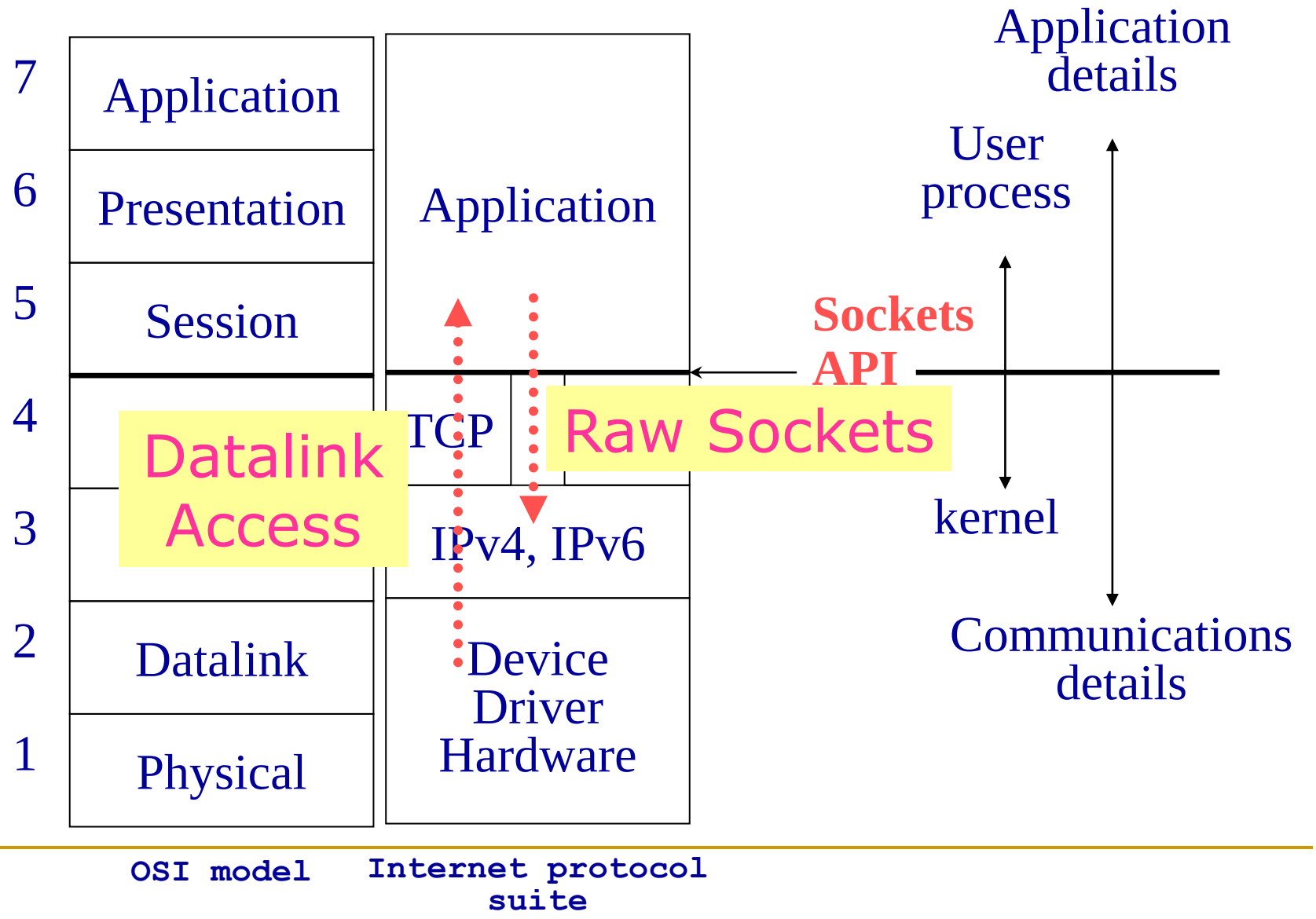
Inter Process Communication(IPC)

- **IPC between process on different host**
  - **Sockets**
    - **TCP Socket (Stream Socket)**
    - **UDP Socket (Datagram Socket)**
    - **SCTP Sockets**
    - **Raw Socket (bypass Transport Layer)**

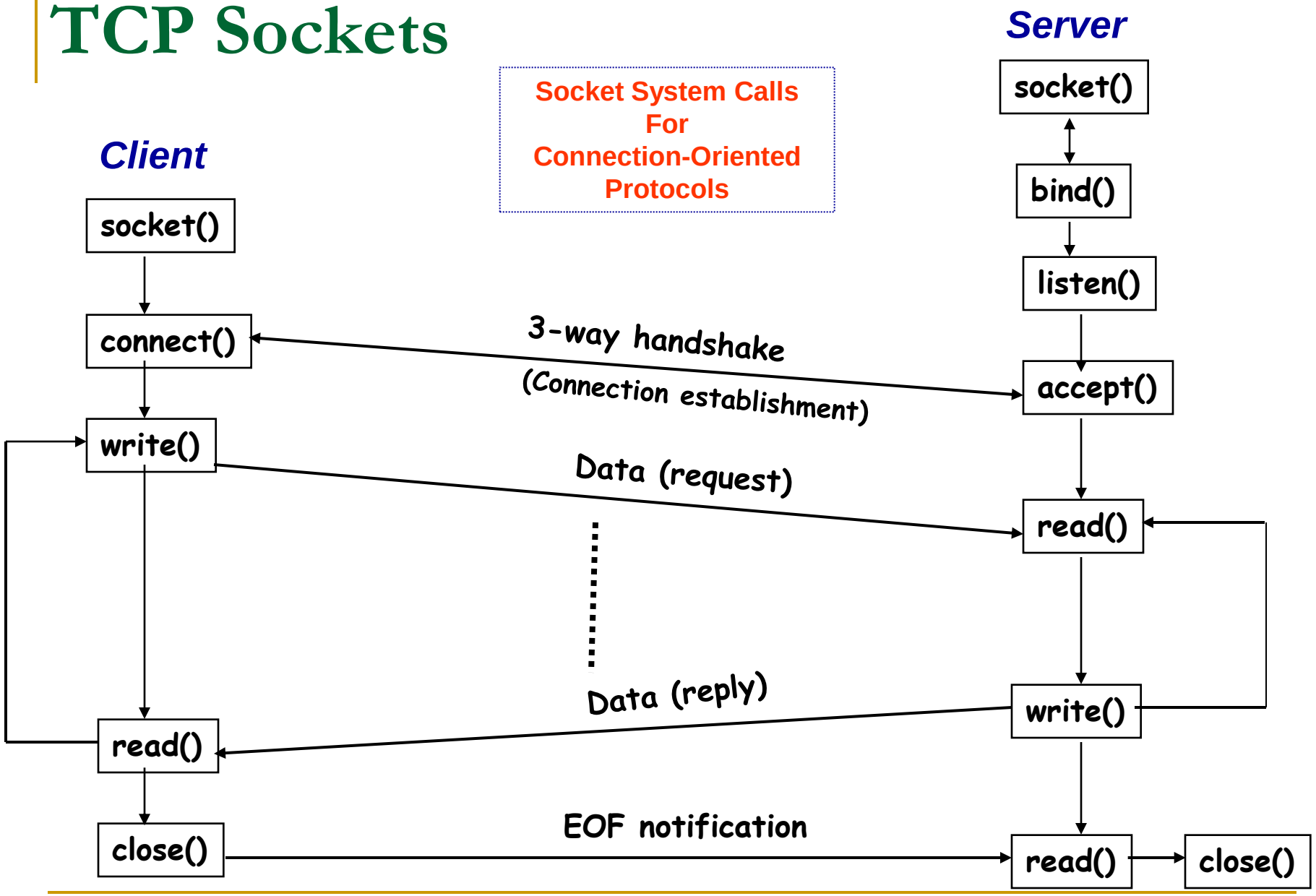
Why do socket provides interface from upper three layers of OSI into transport layer ??



Client ←→ Server

- A *server* is a process - not a machine !
- A server waits for a request from a client
- A client is a process that sends a request to an existing server and (usually) waits for a reply.
- In a network, the client/server model provides a convenient way to interconnect programs that are distributed efficiently across different locations

TCP Sockets


Server
Socket System Calls For Connection-Oriented Protocols
Client
socket()
bind()
listen()
socket()
connect()
3-way handshake
(Connection establishment)
accept()
write()
Data (request)
read()
Data (reply)
write()
read()
close()
EOF notification
read()
close()

Socket System Calls

- Creates a network plug point that enables the client/server to communicate

```
#include <sys/types.h>
#include <sys/socket.h>

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

Socket descriptor
-1 if fails

AF_INET	Internet protocols(IPv4)
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing sockets
AF_NS	Xerox NS protocol
AF_INET6	IPv6 Protocol

Usually 0 is used.
Defined in <netinet/in.h>
IPPROTO_xxx

	AF_UNIX	AF_INET
SOCK_STREAM	Yes	TCP
SOCK_DGRAM	Yes	UDP
SOCK_RAW		IP

SOCK_STREAM	Stream Socket
SOCK_DGRAM	Datagram Socket
SOCK_RAW	Raw Socket
SOCK_SEQPACKET	Sequenced Packet Socket

Bind System Calls

- Allows a server to specify the IP address and port number associated with a socket

```
#include <sys/types.h>
#include <sys/socket.h>

int bind (int sockfd, const struct sockaddr *myaddr, int addrlen);
```

Pointer to a protocol-specific address

- Assign local protocol address to socket
  - Servers register their addresses with the system
    - Both connection oriented & connectionless servers need to do this before accepting client requests.
  - Client can register a specific address for itself
  - connectionless clients need this to assure that a specific address is assigned, so that it can receive response to its request

Connect System Calls

- Enables a client to connect to a server i.e. initiate a connection to a remote host

```
#include <sys/types.h>
#include <sys/socket.h>

int connect (int sockfd, struct sockaddr *servaddr, int addrlen);
```

- for connection oriented protocols, like TCP, results in actual connection establishment (3-wayhandshake)
  - Client process connects a socket descriptor to establish a connection with the server
  - Specific communication parameters like buffer size, size of data to be exchanged without ACK, etc, is agreed
  - System call does not return until connection is established or an error occurs
  - `bind` is not required before `connect`
- Connect can also used for connectionless client

Listen system calls

- allows the server to specify that a socket can be used to accept connections

  int listen (int *sockfd*, int *backlog*);

- used by connection oriented server, to indicate that it is willing to receive connections
  - *backlog* specifies the number of requests that can be queued by the system, while it waits for the server to execute the `accept` system call

Accept system call

- allows a server to wait till a new connection request arrives

```
#include <sys/types.h>
#include <sys/socket.h>

int accept (int sockfd, struct sockaddr *cliaddr, int *addrlen);
```

- after `listen`, `accept` waits for an actual connection request from a client
  - takes the first connection request from queue, and creates another socket with the same properties as *`sockfd`*
  - if no requests are in queue, blocks until a connection request is received
  - accept automatically creates a new socket descriptor, assuming the server is a concurrent server

Read & Write system calls

- to read data from & write data to a TCP socket connection

```
#include <sys/types.h>
#include <sys/socket.h>

int read (int sockfd, char *buff, int nbytes);

int write (int sockfd, char *buff, int nbytes);



int readn (int sockfd, char *buff, int nbytes);
int writen (int sockfd, char *buff, int nbytes);
int writen (int sockfd, char *buff, int nbytes);
int readline (int sockfd, char *buff, int maxlen);
```

not built-in function

- all return the length of data that was read or written as the value of the function

Close system calls

- terminates any connection associated with a socket and releases the socket descriptor

```
int close (int sockfd);
```

- Normal UNIX call close is called to close a socket
  - normally unblocking even if, reliable transfer is required and system has still data or acknowledgements to be sent
  - Kernel still tries to handle those, although the function immediately returns

Getting IP address/port from socket

- **getpeername System Call**
  - Get the IP/port of remote endpoint

```
#include <sys/types.h>
#include <sys/socket.h>

int getpeername(int sockfd, struct sockaddr *peername, int *addrlen );
```

- **getsockname System Call**
  - Get the local IP/port bound to socket

```
#include <sys/types.h>
#include <sys/socket.h>

int getsockname(int sockfd, struct sockaddr *localaddr, int *addrlen );
```

Iterative server

```
int sockfd, newsockfd;

if ( (sockfd = socket ( … )) < 0)
        err_sys ("socket error");
if (bind(sockfd, … ) < 0)
        err_sys ("bond error");
if (listen(sockfd, 5) < 0)
        err_sys ("listen error");

for ( ; ; ) {
        newsocfd = accept (sockfd, … );    /* blocking */
        if (newsockfd < 0)
                err_sys ("accept error");
        do_function();          /* process request */
        close (newsockfd);
}
```

Concurrent Server

```
int sockfd, newsockfd;

if ( (sockfd = socket ( … )) < 0)
        err_sys ("socket error");
if (bind(sockfd, … ) < 0)
        err_sys ("bond error");
if (listen(sockfd, 5) < 0)
        err_sys ("listen error");

for ( ; ; ) {
        newsocfd = accept (sockfd, … );     /* blocking */
        if (newsockfd < 0)
                err_sys ("accept error");
        if (fork() == 0) {                  /* child process*/
                close (sockfd);
                do_function(newsockfd); /* process request */
                close (newsockfd);
                exit(0);
        }
        close (newsockfd);                  /* parent process */
}
```

Bind Example

```
#define PORT  989898
#define IPADD "172.24.2.4"
int sockfd;
struct sockaddr_in myaddr;
 sockfd = socket(AF_INET,SOCK_STREAM,0);
      myaddr.sin_family = AF_INET;
      myaddr.sin_addr.s_addr = inet_addr( IPADD);
                                          //or inet_addr(argv[1] ) for command line arg
     myaddr.sin_port = htons( PORT);      // or  htons(atoi (argv[2]) )  for command line arg

  bind(sockfd , (strcut sockaddr*)&myaddr, sizeof(myaddr));
```

- during call to bind we can tell kernel to assign any available port

  `myaddr.sin_port = htons(0);` `//emhemeral port`
- to tell kernel to choose IP address

  myaddr.sin_addr.s_addr = htonl(INADDR_ANY); //wildcard address
- Clients typically don't care what port they are assigned

Iterative Server

$./server

```
#define SERVER_PORT 898989
main {
struct sockaddr_in cliaddr,servaddr;
        int sockfd,confd,clilen;
 sockfd = socket(AF_INET , SOCK_STREAM , 0) ;
      servaddr.sin_family = AF_INET;
      servaddr.sin_port = htons(SERVER_PORT);
      servaddr.sin_addr.s_addr =htonl( INADDR_ANY) ;
  bind(sockfd, (struct sockaddr *)&servaddr,sizeof(servaddr));

  listen(sockfd,10);
 while(1) {
              clilen=sizeof(cliaddr);
   confd= accept(sockfd,(struct sockaddr *)&cliaddr,&clilen);
            // /*  printf("Client IP: %s\n", inet_ntoa(cliaddr.sin_addr));  */
            // /*  printf("Client Port: %hu\n", ntohs(cliaddr.sin_port));   */
     read(connfd, buf, BUFFER_SIZE);
printf("Received:%s\n",buf);
     write(confd, buf, BUFFER_SIZE);
     close(confd);
    }
return 0;
}
```

Note:

- Port number is hard-coded in program

-it can be taken as command line argument

Client Program

$./client 172.24.2.4

```
#define SERVER_PORT 898989
main {
 struct sockaddr_in cliaddr,servaddr;
        int sockfd,confd,clilen;
sockfd = socket(AF_INET , SOCK_STREAM , 0) ;

    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERVER_PORT);
    servaddr.sin_addr.s_addr =inet_addr(argv[1]) ;  // or inet_pton()

 connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) ;
     clilen = sizeof(cliaddr);

  //  /*  getsockname(sockfd, (struct sockaddr *) &cliaddr, &clilen);   */
  //   /*    printf("Client socket has IP: %s\n", inet_ntoa(cliaddr.sin_addr));  */
  //   /*  printf("Client socket has Port: %hu\n", ntohs(cliaddr.sin_port));  */

printf("Enter data:\n");  scanf("%s",buf) ;
write(sockfd, buf, BUFFER_SIZE);
read(sockfd, buf, BUFFER_SIZE);
printf("Received Data :%s\n",buf);
  close(sockfd);
  exit(0);
 }
```

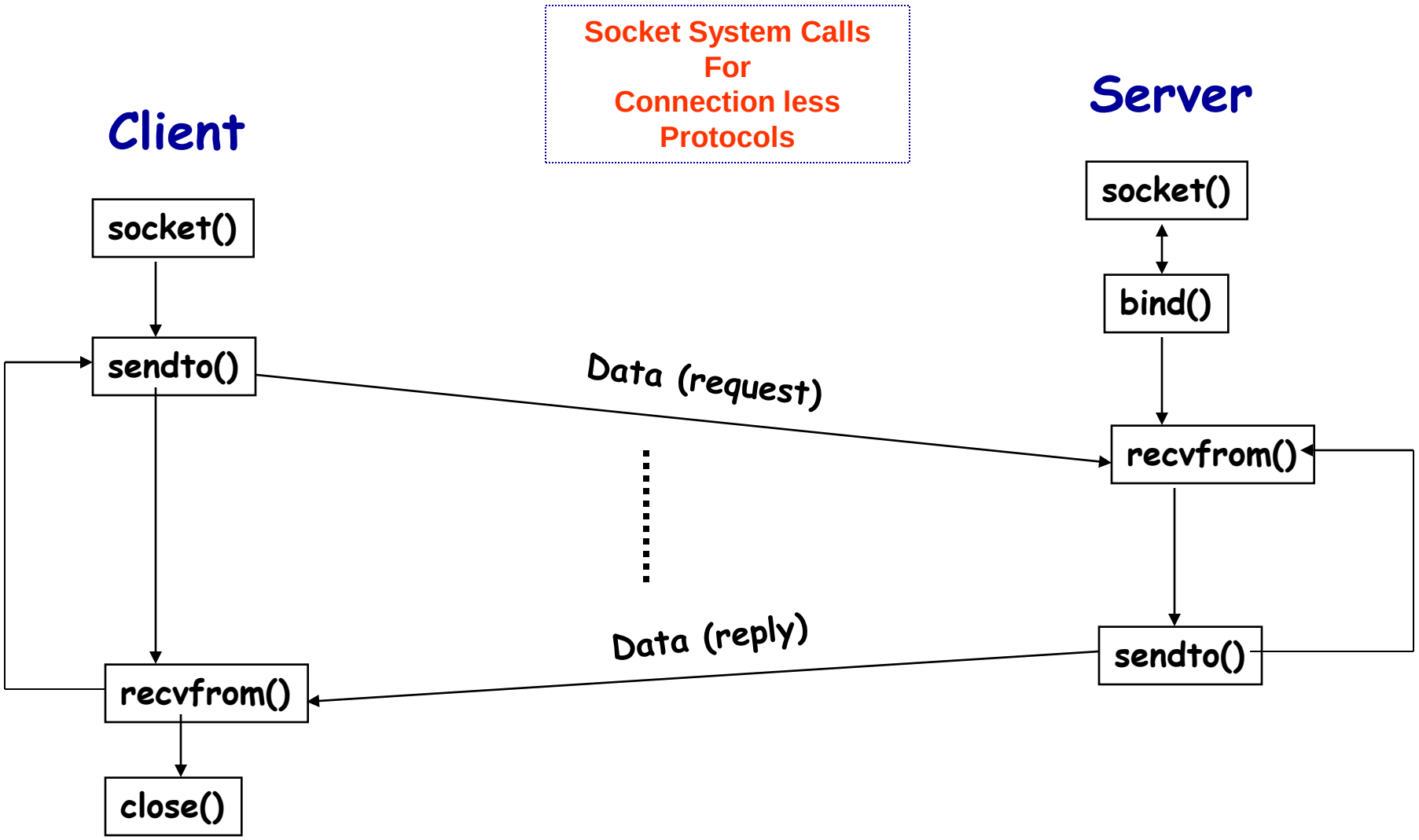
Note:

- **Port number is hard-coded in program**
- **it can be taken as command line argument**

concurrent server

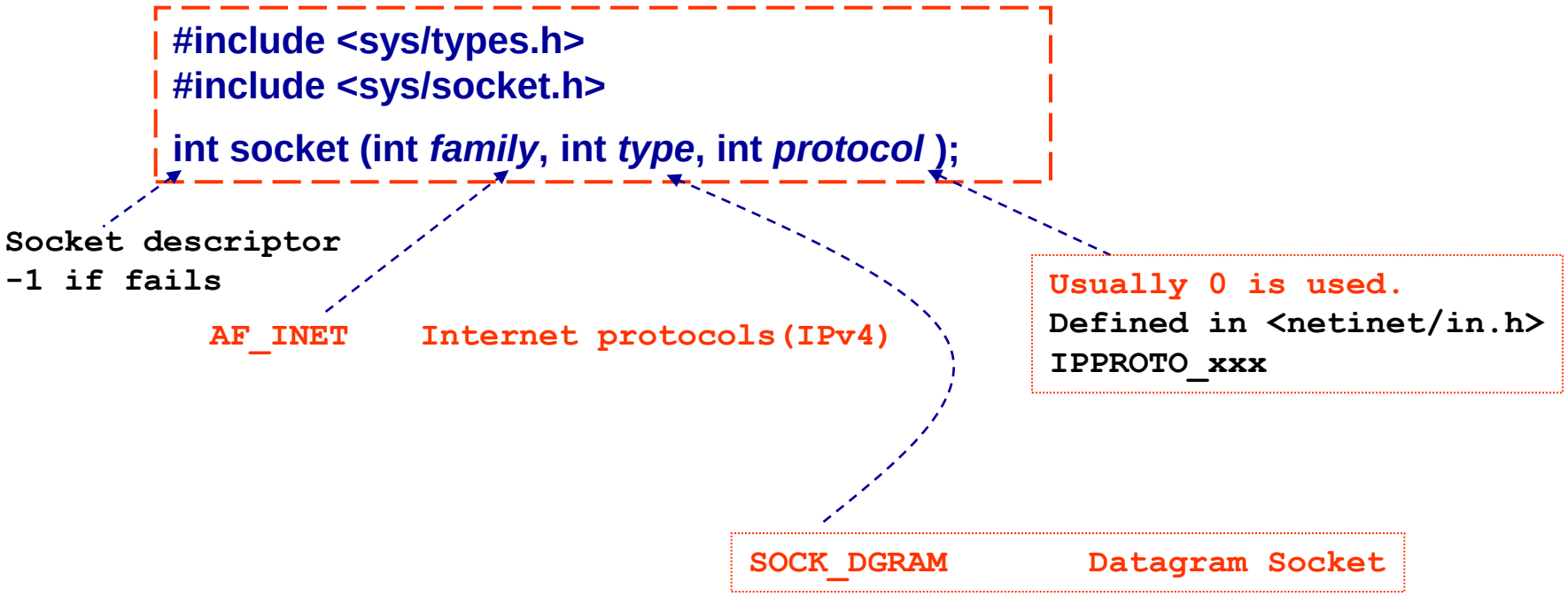
```
main() {
   sockfd = socket(….);
    bind(sockfd,…….);
    listen(sockfd,10);

 for ( ; ; )
  {      clilen=sizeof(cliaddr);
   confd= accept(sockfd,(struct sockaddr *)&cliaddr,&clilen);
            printf("Client IP: %s\n", inet_ntoa(cliaddr.sin_addr));
            printf("Client Port: %hu\n", ntohs(cliaddr.sin_port));

    if( (pid = fork() ) ==0)   /*child process handle the client request*/
            {  close(sockfd);   bzero(buf,BUFFER_SIZE);
                read(connfd, buf, BUFFER_SIZE);
                  printf(“Received:%s\n”,buf);
                write(confd, buf, BUFFER_SIZE);
             close(confd);
             exit(0);
            }
     close(confd);
    }
}
```

UDP Sockets


Socket System Calls
For
Connection less
Protocols
Client
Server
socket()
sendto()
recvfrom()
close()
socket()
bind()
recvfrom()
sendto()
Data (request)
Data (reply)

Socket System Calls

- Creates a network plug point that enables the client/server to communicate



Bind System Calls

- Allows a server to specify the IP address and port number associated with a socket

```
#include <sys/types.h>
#include <sys/socket.h>

int bind (int sockfd, const struct sockaddr *myaddr, int addrlen);
```

Pointer to a protocol-specific address

- Assign local protocol address to socket
  - Servers register their addresses with the system
  - Client can register a specific address for itself
  - connectionless clients need this to assure that a specific address is assigned, so that it can receive response to its request

Sendto System Calls

```
Either zero, or OR'ed with
MSG_OOB         Out-of-band
MSG_PEEK        Peek at data
MSG_DONTROUTE   Bypass routing
```

- Send a UDP message on a socket to specified destination

```
#include <sys/socket.h>

int sendto (int sockfd, const void *buff ,int nbytes, int flags ,
            const struct sockaddr *to ,  int addrlen ) ;
```

recvfrom System Calls

- receive a UDP message on a socket along with address of sending source

```
#include <sys/socket.h>

int recvfrom (int sockfd, const void *buff ,int nbytes, int flags ,
            const struct sockaddr *from ,  int *addrlen ) ;
```

UDP Echo Sever

UDP Server
$ *./udpserver* Port

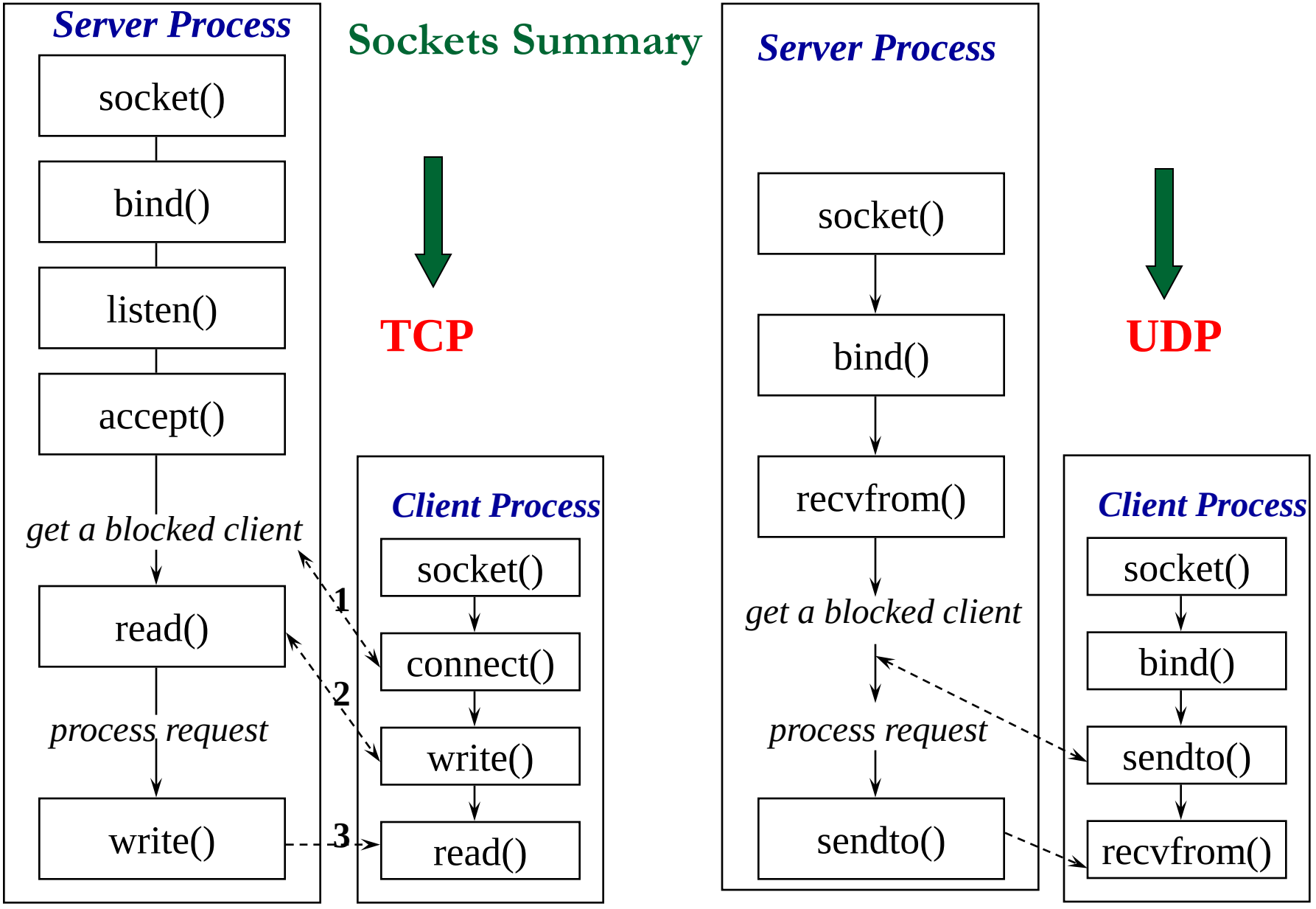
```
int main(int argc, char **argv)
  {    int   sockfd ,len;  char buff[ MAX];
       struct sockaddr_in  servaddr, cliaddr;

 sockfd = socket(AF_INET, SOCK_DGRAM, 0);

       bzero(&servaddr, sizeof(servaddr));
       servaddr.sin_family      = AF_INET;
       servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
       servaddr.sin_port        = htons (atoi(argv[1]));
bind(sockfd,(struct Sockaddr *) &servaddr,sizeof(servaddr));

for (;; )
    {  len = clilen;
 n=recvfrom(sockfd, buff, MAX, 0, cliaddr, &len);
      printf(" Received Msg.:%s\n",buff);

 sendto(sockfd, buff, n, 0,(struct sockaddr*)&cliaddr, len);
    }
}
```

UDP Echo Client

UDP Client
$ *./udpserver* IPADD *Port*

```
int main(int argc, char **argv)
{           int         sockfd ,len;  char buff[ MAX];
            struct sockaddr_in              servaddr;
   sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

            bzero(&servaddr, sizeof(servaddr));
            servaddr.sin_family = AF_INET;
            inet_pton ( AF_INET,  argv[1] ,  &servaddr.sin_addr );
            servaddr.sin_port = htons( atoi (argv[2]) );

       printf("Enter Text message\n");
       scanf( "%s",buff);
 sendto (sockfd, buff, MAX, 0,(struct sockaddr*)&servaddr,  sizeof (servaddr) );

  recvfrom(sockfd, buff, MAX, 0, NULL, NULL);
       printf("Received Mesg.:%s\n",buff);

   exit(0);
}
```

Sockets Summary



Iterative Server

$./server

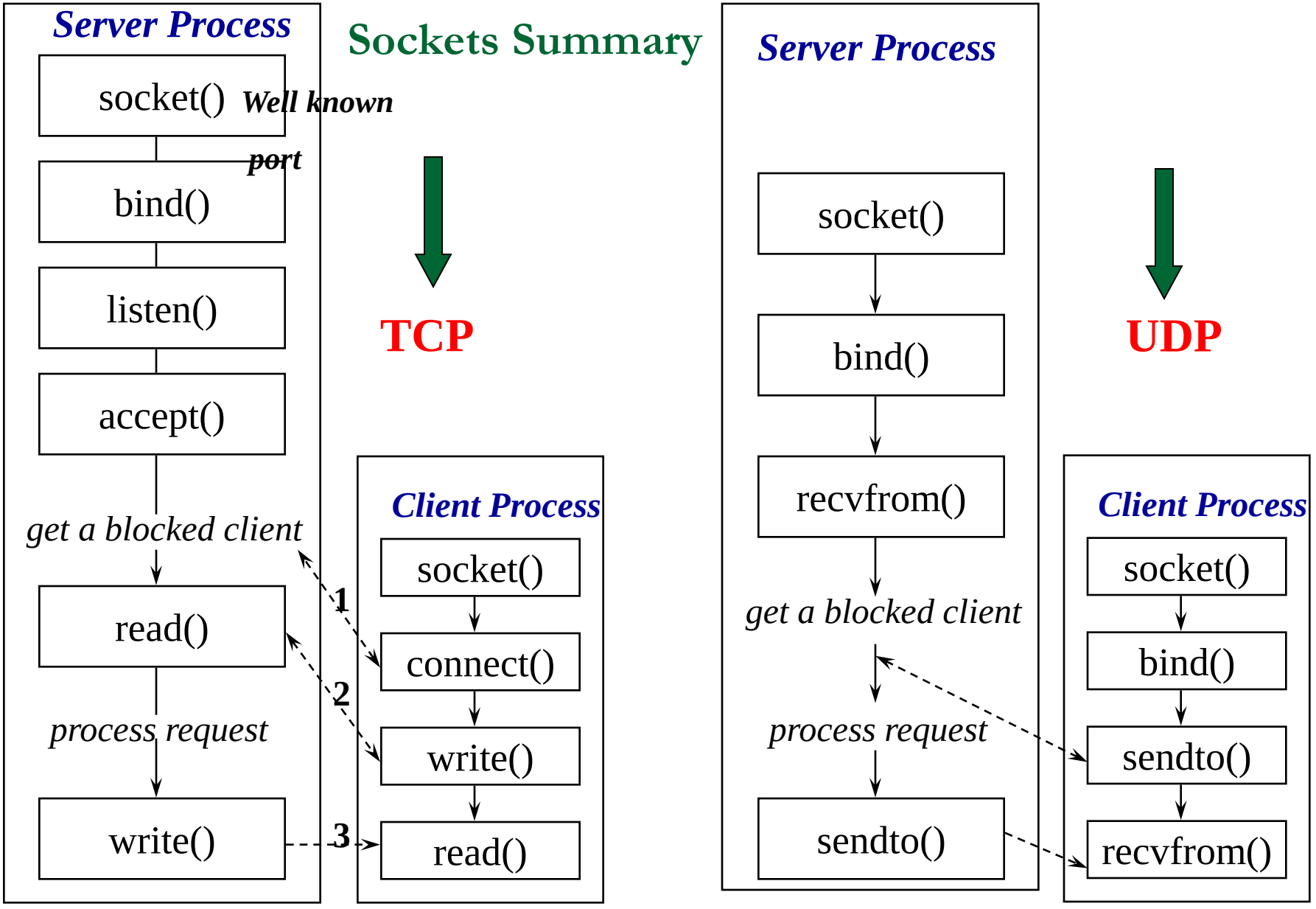
```
#include <arpa/inet.h> #include <sys/socket.h> #include <netinet/in.h>
#define SERVER_PORT 898989
Main() {
struct sockaddr_in cliaddr,servaddr;
        int sockfd,confd,clilen;
 sockfd = socket(AF_INET , SOCK_STREAM , 0) ;
     servaddr.sin_family = AF_INET;
     servaddr.sin_port = htons(SERVER_PORT);
     servaddr.sin_addr.s_addr =htonl( INADDR_ANY) ;
  bind(sockfd, (struct sockaddr *)&servaddr,sizeof(servaddr));
  listen(sockfd,10);
 while(1) {
              clilen=sizeof(cliaddr);
   confd= accept(sockfd,(struct sockaddr *)&cliaddr,&clilen);
          /*    printf("Client IP: %s\n", inet_ntoa(cliaddr.sin_addr)); */
          /*    printf("Client Port: %hu\n", ntohs(cliaddr.sin_port));  */
     read(connfd, buf, BUFFER_SIZE);  printf("Received:%s\n",buf);
     write(confd, buf, BUFFER_SIZE);
     close(confd);
    }
return 0;
}
```

#include <arpa/inet.h> #include <sys/socket.h> #include <netinet/in.h>

Client Program
$./client 172.24.2.4

```
#define SERVER_PORT 898989
main () {
 struct sockaddr_in cliaddr,servaddr;
        int sockfd,confd,clilen;
sockfd = socket(AF_INET , SOCK_STREAM , 0) ;

    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(SERVER_PORT);
    servaddr.sin_addr.s_addr =inet_addr(argv[1]) ;  // or inet_pton()

 connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) ;
     clilen = sizeof(cliaddr);

 /*  getsockname(sockfd, (struct sockaddr *) &cliaddr, &clilen); */
 /*      printf("Client socket has IP: %s\n", inet_ntoa(cliaddr.sin_addr)); */
 /*    printf("Client socket has Port: %hu\n", ntohs(cliaddr.sin_port)); */
printf("Enter data:\n");  scanf("%s",buf) ;
write(sockfd, buf, BUFFER_SIZE);
read(sockfd, buf, BUFFER_SIZE);   printf("Received Data :%s\n",buf);
  close(sockfd);
  exit(0);
 }
```

Sockets Summary



UDP Echo Sever

UDP Server
$ *./udpserver* Port

```
int main(int argc, char **argv)
  {    int    sockfd ,len;  char buff[ MAX];
       struct sockaddr_in  servaddr, cliaddr;

 sockfd = socket(AF_INET, SOCK_DGRAM, 0);

       bzero(&servaddr, sizeof(servaddr));

       servaddr.sin_family      = AF_INET;
       servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
       servaddr.sin_port        = htons (atoi(argv[1]));

bind(sockfd,(struct Sockaddr *) &servaddr,sizeof(servaddr));

for (;; )
    {  len = clilen;
 n=recvfrom(sockfd, buff, MAX, 0, cliaddr, &len);
      printf(" Received Msg.:%s\n",buff);

 sendto(sockfd, buff, n, 0,(struct sockaddr*)&cliaddr, len);
    }
}
```

UDP Echo Client

UDP Client
$ *./udpserver* IPADD *Port*

```
int main(int argc, char **argv)
{           int         sockfd ,len;  char buff[ MAX];
            struct sockaddr_in            servaddr;
    sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

            bzero(&servaddr, sizeof(servaddr));
            servaddr.sin_family = AF_INET;
            inet_pton ( AF_INET,  argv[1] ,  &servaddr.sin_addr );
            servaddr.sin_port = htons( atoi (argv[2]) );

        printf("Enter Text message\n");
        scanf( "%s",buff);
  sendto (sockfd, buff, MAX, 0,(struct sockaddr*)&servaddr,  sizeof (servaddr) );

   recvfrom(sockfd, buff, MAX, 0, NULL, NULL);
        printf("Received Mesg.:%s\n",buff);

    exit(0);
}
```

Simple UDP Server

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define SERVER_PORT    9988
int main()
 {   int   sockfd, clilen;
    char     buf[256];
    struct sockaddr_in servaddr, cliaddr;
sockfd = socket( AF_INET, SOCK_DGRAM, 0);
     servaddr.sin_family = AF_INET;
     servaddr.sin_port = htons(SERVER_PORT);
     servaddr.sin_addr.s_addr =htonl(INADDR_ANY);
if (bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr)) <0 )
      { printf("Server Bind Error"); exit(1); }
for(; ; )
{ clilen= sizeof(cliaddr);
 recvfrom(sockfd,buf,256,0,(struct sockaddr*)&cliaddr,&clilen);

 printf("Server Received:%s\n",buf);

sendto(sockfd,"Server Got Message",18, 0,(struct
   sockaddr*)&cliaddr,clilen);
 }
}
```

Simple UDP Client

```
#include<sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define SERVER_PORT   9988
#define SERVER_IPADDR  "172.24.2.4"
int main()
{   int   sockfd,len;
    char     buf[256];
    struct sockaddr_in ,cliaddr,servaddr;

     servaddr.sin_family = AF_INET;
     servaddr.sin_port = htons(SERVER_PORT);
     servaddr.sin_addr.s_addr = inet_addr(SERVER_IPADDR);

sockfd = socket( AF_INET, SOCK_DGRAM, 0);

     cliaddr.sin_family = AF_INET;
     cliaddr.sin_port = htons(0);
     cliaddr.sin_addr.s_addr =htonl(INADDR_ANY);
bind(sockfd,(struct sockaddr*)&cliaddr,sizeof(cliaddr));

 printf("Enter Message\n");   fgets(buf,255,stdin);
  len= sizeof(server);

 sendto(sockfd,buf,strlen(buf), 0,(struct sockaddr*)&servaddr,len);

 recvfrom(sockfd,buf,256,0,NULL,NULL);

    printf("Clinet Received: %s \n",buf);
 close(sockfd);
}
```

Not mandatory