Socket Programming

Sockets

- Socket is a data communication endpoint for exchanging data over the network

- Uniquely identified by:
- ip address
- ‰end-to-end protocol (e.g. TCP or UDP)‰

- port number

Types of (TCP/IP) Sockets :

Stream Sockets (e.g uses TCP):

- Provides reliable byte-stream service
- Connection oriented
- Data in order

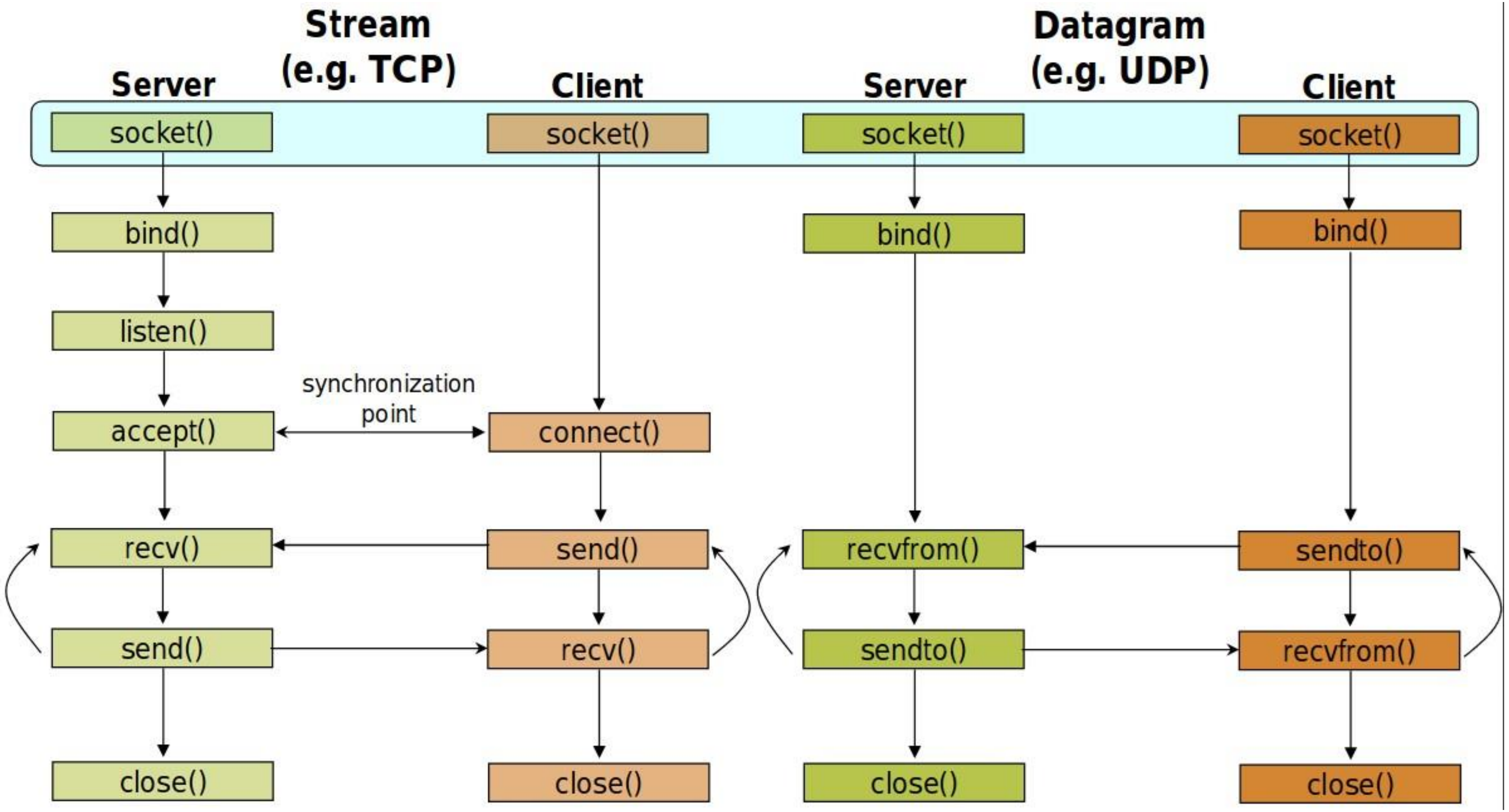
Datagram Sockets (e.g uses UDP):

- Connection less
- Data not in order
- Not reliable

Important Terms

- Socket Creation

- Bind
- Listen
- Accept
- Connect
- Send
- Receive
- Close


Stream
(e.g. TCP)
Server
Client
Datagram
(e.g. UDP)
Server
Client
socket()
socket()
socket()
socket()
bind()
bind()
bind()
listen()
synchronization point
accept()
connect()
recv()
send()
recvfrom()
sendto()
send()
recv()
sendto()
recvfrom()
close()
close()
close()
close()

Socket Creation

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

`sockid` : socket descriptor, an integer (like a file-handle)

`Family:`

E.g PF_INET IPv4 Internet protocols

More details :

- http://man7.org/linux/man-pages/man2/socket.2.html

`Type :` **Communication Type**

SOCK_STREAM

SOCK_DGRAM

`Protocol :`

usually set to 0 (i.e., use default protocol)

Closing

Socket

```
int status = close(sockid);
 - Sockid : descriptor
 - status : 0 if successful , -1 if error
```

Closing a socket:

- Closes a connection (for stream socket) - Frees up the port used by the socket

`Bind()`

Assign address to socket

```
struct sockaddr_in{
    unsigned short sin_family;  /* Internet protocol (AF_INET) */
    unsigned short sin_port;    /* Address port (16 bits) */
    struct in_addr sin_addr;    /* Internet address (32 bits) */
};

struct in_addr {
    unsigned long s_addr;  /* Internet address (32 bits) */
} int status = bind(sockid, &addrport,

size);
```

Example:

```
int  socketfd = socket(PF_INET,SOCK_STREAM,0);
```

```
If ( socketfd < 0 ){ cout<<"Error in
        connection"<<endl;
        exit(1);
}

struct sockaddr_in serveraddr; serveraddr.sin_family = AF_INET;
serveraddr.sin_port=htons(2000);
serveraddr.sin_addr.s_addr=inet_addr("127.0.0.1"); int ret = bind(socketfd ,
(struct sockaddr*)&serveraddr , sizeof(serveraddr));


If ( ret < 0 ){ cout<<"Error in
        Binding"<<endl;
        exit(1);
}
```

Listen()

```
int status = listen(sockid, queueLimit);
```

`queueLimit :` **No.** of active participants that can “wait” for a connection.

Note:

If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

Establish Connection: connect()

The client establishes a connection with the server by calling `connect()`

```
int status = connect(sockid, &foreignAddr, addrlen);
```

`sockid` : socket descriptor, an integer (like a file-handle)

`foreignAddr` : struct sockaddr: address of the passive participant

`Addrlen : size of` addr structure

Incoming Connection : `accept()`

The server gets a socket for an incoming client connection by calling `accept()`

```
int new_socket = accept(sockid, &clientAddr, &addrLen);
```

`accept()` :

is blocking: waits for connection before returning
‰
dequeues the next connection on the queue for socket (sockid)

Exchanging data with stream socket

`Send()` :

```
int count = send(sockid, msg, msgLen, flags);
```

`sockid` : socket descriptor, an integer (like a file-handle)

`msg` : const void[], message to be transmitted

`msgLen : length of message` `flags` : integer, special options, usually just 0

`Count` : Number of bytes transmitted **(-1 if error)**

More Details :

- https://linux.die.net/man/2/send

Recv():

```
int count = recv(sockid, recvBuf, bufLen, flags);
```

`sockid` : socket descriptor, an integer (like a file-handle) `recvBuf`: void[], stores received bytes `bufLen`: # bytes to read `flags`: # bytes received (-1 if error)

Note:
send() and recv() are blocking‰ returns only after data is sent / received

Exchanging data with datagram socket

`Sendto():`

```
int count=sendto(sockid,msg,msgLen, flags,&foreignAddr,
addrlen);
```

`sockid` : socket descriptor, an integer (like a file-handle)
`msg :` const void[], message to be transmitted
`msgLen : length of message`
`flags` : integer, special options, usually just 0
`Count` : Number of bytes transmitted **(-1 if error)**
`foreignAddr` **:** Address of destination

`Addrlen :` sizeof(foreignAddr)

`Recvfrom():`

```
int count = recvfrom(sockid, recvBuf, bufLen, flags,
&clientAddr, addrlen);
```

`sockid` : socket descriptor, an integer (like a file-handle) `recvBuf`: void[], stores received bytes `bufLen`: # bytes to read `flags`: # bytes received (-1 if error)

`clientAddr` : Address of destination

`Addrlen :` sizeof(clientAddr)

Examples (Stream Socket):

1. **Send some integer from client to server:**

   `Client Side :`

```
                int x  = 10;
        send(clientSock, &x, sizeof(x), 0);
```

Server Side :

```
        int x;
        recv(serverSock, &x, sizeof(x), 0);
```

2 . **Sending a file:**

- **Sending Side (client)**

```
int sockfd = socket(
    AF_INET, SOCK_STREAM, 0 );

    struct sockaddr_in     serv_addr; serv_addr.sin_family =
    AF_INET;    serv_addr.sin_port    =    htons(   PORT   );
    serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");


    connect ( sockfd  , (struct sockaddr *)&serv_addr  , sizeof(serv_addr) )
```

```
FILE *fp = fopen ( "path of file"  , "rb" );

        fseek ( fp , 0 , SEEK_END);
        int size = ftell ( fp );
        rewind ( fp );
send ( sockfd , &size, sizeof(file_size), 0);

char Buffer [ BUFF_SIZE] ; while ( ( n = fread( Buffer , sizeof(char) , BUFF_SIZE ,
        fp ) ) > 0  && size > 0 ){
                        send (sockfd , Buffer, n, 0 )
                        memset ( Buffer , '\0',
                        BUFF_SIZE); size = size - n ;
}

fclose ( fp );
close( sockfd)
```

- **Receiving Side (Server)** server_fd = socket

```
 (AF_INET, SOCK_STREAM, 0);
```

```
  struct sockaddr_in   addr; addr.sin_family =
  AF_INET; addr.sin_port = htons( PORT );
  addr.sin_addr.s_addr=inet_addr(INADDR_ANY);
  int addrlen = sizeof(sockaddr)

  bind (server_fd  , (struct sockaddr *)&addr , sizeof ( addr ) ) listen (server_fd, 3) int

  sockfd = accept ( server_fd , (struct sockaddr *)&address , (socklen_t*)&addrlen);

  FILE *fp = fopen ( "path of file"  , "wb" );
  char   Buffer   [   BUFF_SIZE]   ;   int
  file_size;

  recv(serverSock, &file_size, sizeof(file_size), 0);
while ( ( n = recv( sockfd , Buffer ,  BUFF_SIZE, 0) ) > 0  &&  file_size > 0){

      fwrite (Buffer , sizeof (char), n, fp)
      memset   (   Buffer   ,    '\0',   BUFF_SIZE);
      file_size = file_size - n;
}
```

```
close( sockfd)
close( serverfd)
fclose ( fp );
```

Multithreaded Server

```
void main(){
        ……..
        ……..
        listen ( socketfd , 5 ) while(1){ newsocket= accept( sockfd , (struct

        sockaddr*)&newAddr , &addr_size); thread

        RequestThread(serveRequest,newsocket,newAddr);


        }
}
void  serveRequest  (  int  newsoc  ,  struct  sockaddr_in
newAddr){ //  bla bla bla }
```