INTER-PROCESS COMMUNICATION AND SYNCHRONISATION:

Lesson-18: Socket

1. Need for Sockets for the Inter Process or Remote Communication

Need of Sockets

- A pipe could be used for inserting the byte steam by a process and deleting the bytes from the stream by another process.
- However, for example, we need that the card information to be transferred from a process *A* as byte stream to the host machine process *B* and the *B* sends messages as byte stream to the *A*. There is need for bi-directional communication between *A* and *B*.

Need of Sockets

- We need that the *A* and *B* ID or port or address information when communicating must also be specified either for the destination alone or for the source and destination both. [The messages in a letter are sent along with address specification.]

Need of Socket

- We need to use a protocol for communication
- A protocol, for example, provides along with the byte stream information of the address or port of the destination or addresses or ports of source and destination both or the protocol may provide for the addresses and ports of source and destination in case of the remote processes

Need for Connectionless protocol

- *F*or example, UDP (user datagram protocol).
- UDP protocol ─a UDP header,
- UDP header ─ contains source port (optional) and destination port numbers, length of the datagram and checksum.
- Port means a process or task for specific application.
- Port number specifies the process.

Connectionless protocol…

- Connectionless ─ no connection establishment between source and destination before data transfer to stream
- Datagram means a data, which is independent need not in sequence with the previously sent data.
- Checksum is sum of the bytes to enable the checking of the erroneous data transfer.

Connectionless protocol…

- For remote communication, the address, for example, IP address is also required in the UDP header.

Alternatively Need for Connection-oriented protocol

- for example, TCP (transport control protocol).

First a connection establishment between source and destination and then the actual transfer of data stream can take place.

1. Socket device for the Inter Process or Remote Communication

Socket

- Provides a device like mechanism for bi-direction communication.
- Provides for a bi-directional pipe like device, which also use a protocol between source and destination processes for transferring the bytes.
- Provides for establishing and closing a connection between source and destination processes using a protocol for transferring the bytes.

Socket…

- May provide for listening from multiple sources or multicasting to multiple destinations.
- Two tasks at two distinct places or locally interconnect through the sockets.
- Multiple tasks at multiple distinct places interconnect through the sockets to a socket at a server process.
- The client and server sockets can run on same CPU or at distant CPUs on the Internet.

Socket …

- Sockets can be using a different domain. For example, a socket domain can be TCP (transport control protocol) , another socket domain may be UDP(transport control protocol), the card and host example socket domain is different.

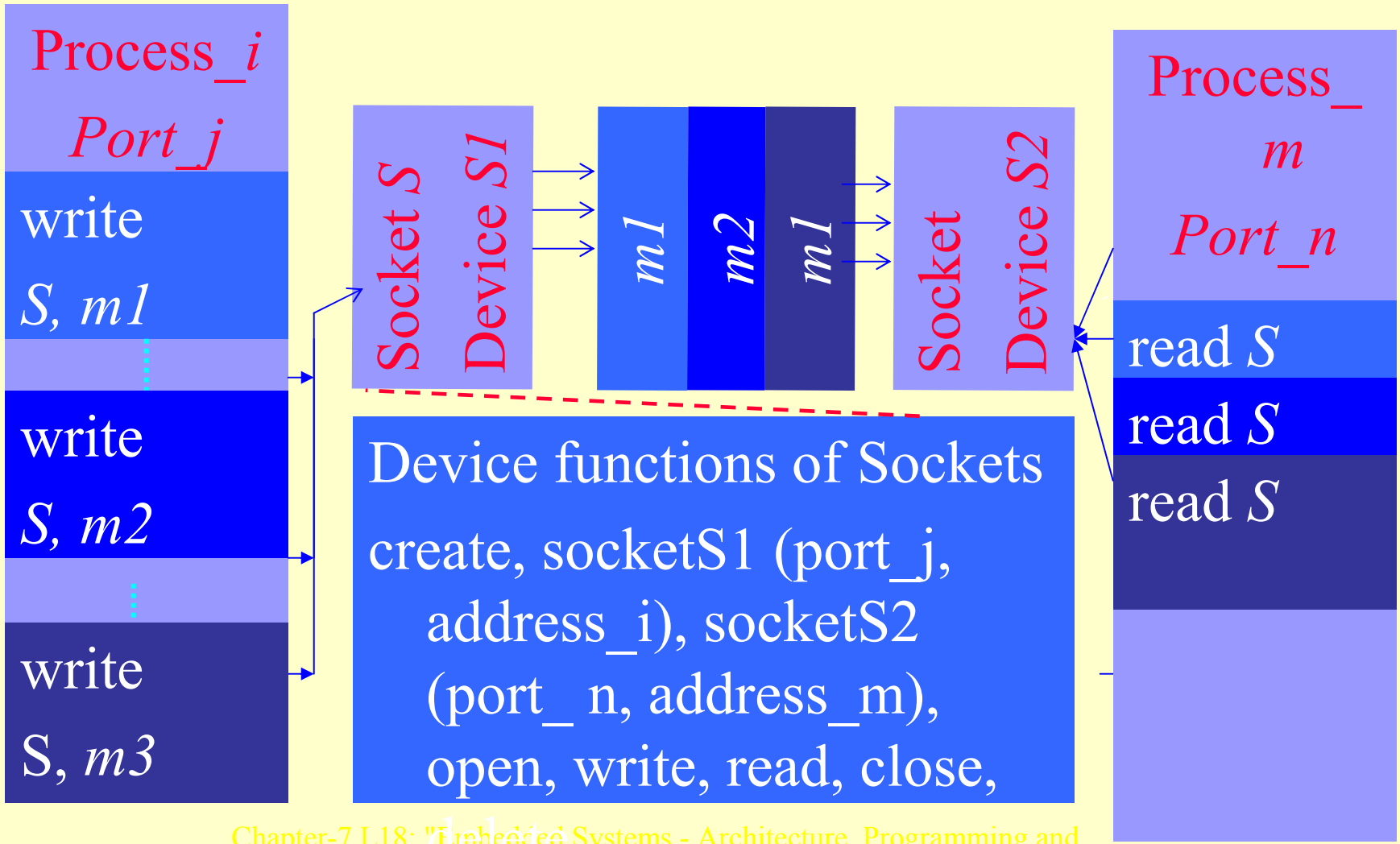
Socket

1. Two processes (or sections of a task) at two sets of ports interconnect (perform inter process communication) through a socket at each. [These are virtual (logical), and not physical sockets.]

Socket…

2. A socket stream between two specified sections (ports)─ at the specified sets (addresses) sent with a port-specific protocol.
3. Each socket ─process address (similar to a network or IP address) and section (similar to a port) specification. The sections (or ports or tasks) and sets of processes (addresses) may be on the same computer or on a network.

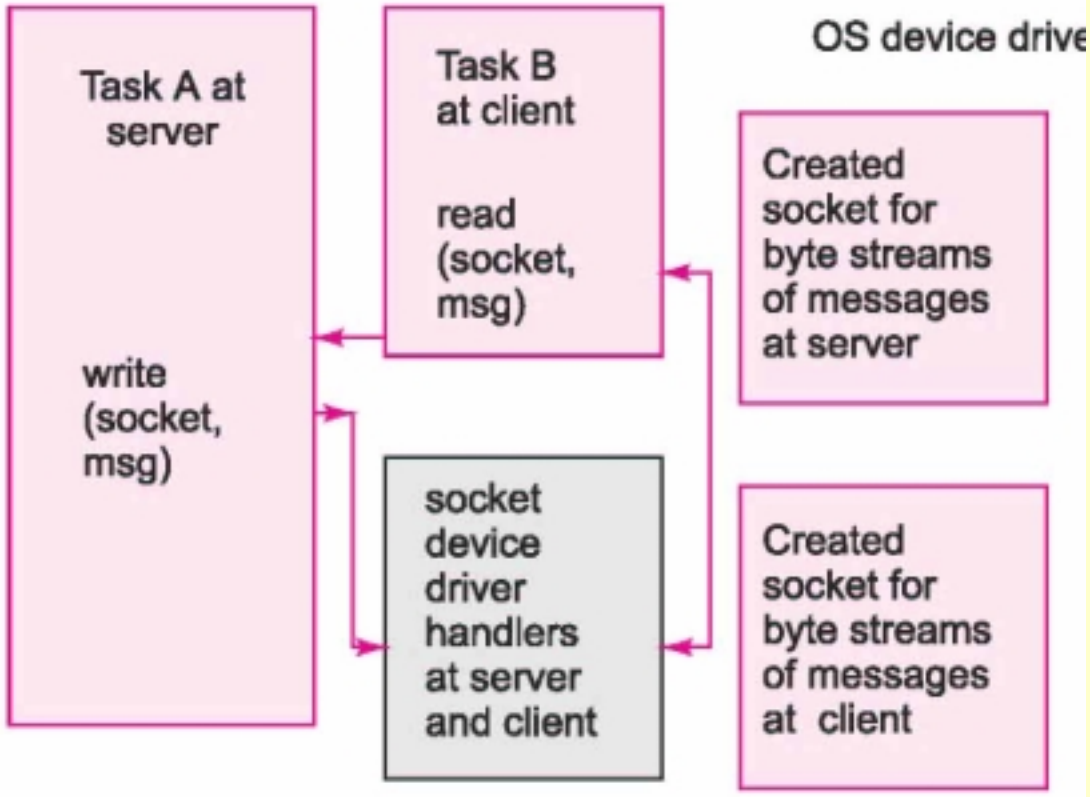
Process i port j socket for the messages that socket at process *m* port *n* receives



4. There has to be a specific protocol in which the messages at the socket interconnect between (i, j) and (m, n). [i and m are process addresses, and j and n are port or section specifications]
5. A pipe does not have protocol based inter-processor communication, while socket provides that.

6. A socket can be a client-server socket. Client Socket and server socket functions are different.
7. A socket can be a peer-to-peer socket IPC. At source and destination sockets have similar functions.

Server Task write and Client read Functions



Client uses connect, read, write functions, and server uses bind, listen, accept, write and read functions in TCP like connection-oriented protocol

Client uses bind, send, receive functions, and server uses bind, receive and send functions in UDP like connection-less protocol

3. Socket-device Functions

Socket-device Functions in Unix

1. *socket* ( ) [in place of open ( ) in case of pipe] gives a socket descriptor sfd. The socket ( ) for enables its use from beginning of its allocated buffer at the socket address, its use with option and restrictions or permissions defined at the time of opening.

A socket can be a stream, SOCK_STREAM or UDP data gram SOCK_DGRAM.

2. *unlink* ( ) before executing *bind* ( )

Socket-device Functions in Unix

3. *bind* ( )─ for binding a thread or task inserting bytes into the socket to the thread or task and deleting bytes from the socket.

bind ( ) the socket descriptor to an address in the Unix domain. bind (sfd, (struct sockaddr *)&local, len); where len is string length. sockaddr is a data structure with record of 16-bit unsigned num and a path for the file and a data structure struct sockaddr_un {unsigned short num; char path[108]; }

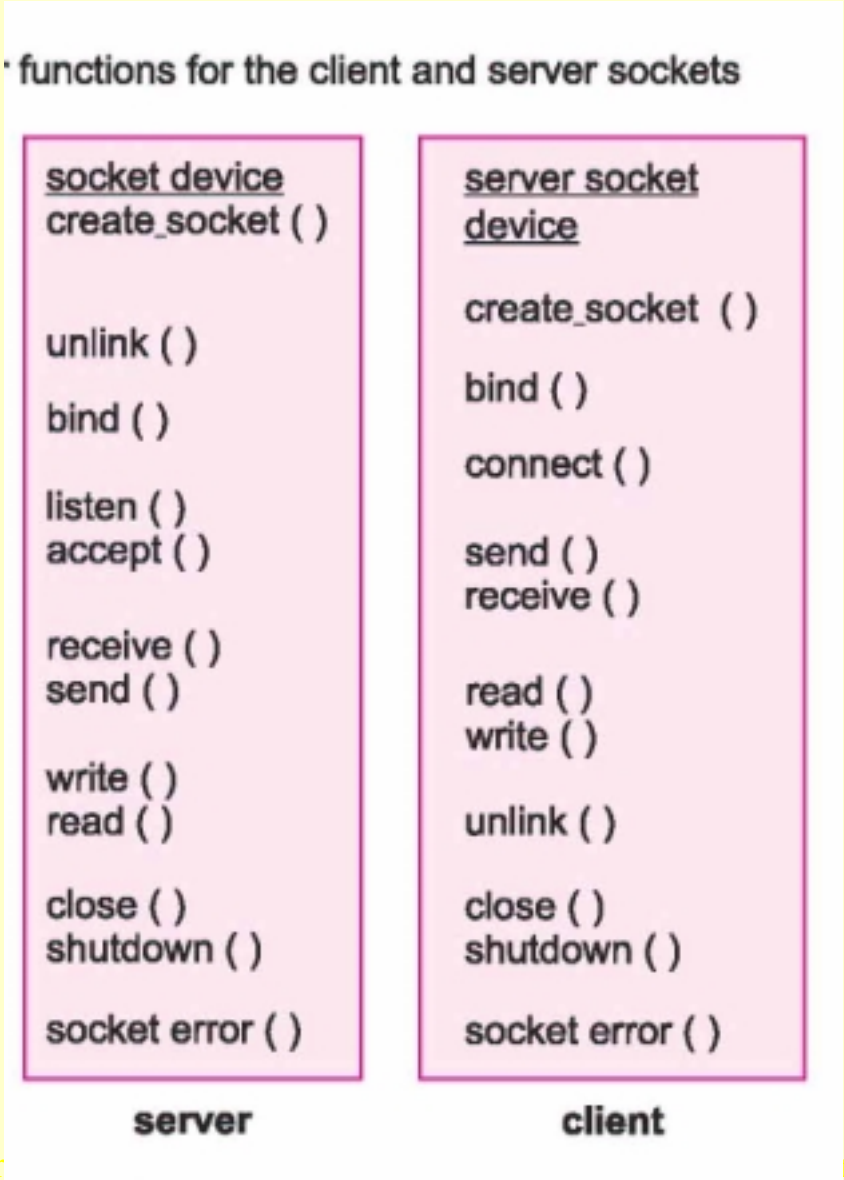
Socket-device Functions in Unix…

4. *listen* (sfd, 16)─ for listening 16 queued connections from client socket
5. *accept ( )*─ accepts the client connection and gives a second socket descriptor
6. *recv* ( ) ─ function for deleting (reading) and receiving from the socket from the bottom of the unread memory spaces in the buffer filled after writing into the socket.

Socket-device Functions in Unix…

7. send ( )─ for inserting (writing) and sending from the socket from the bottom of the memory spaces in the buffer filled after writing into the socket
8. close ( ) ─ for closing the device to enable its use from beginning of its allocated buffer only after opening it again

Client and Server Socket-device Functions

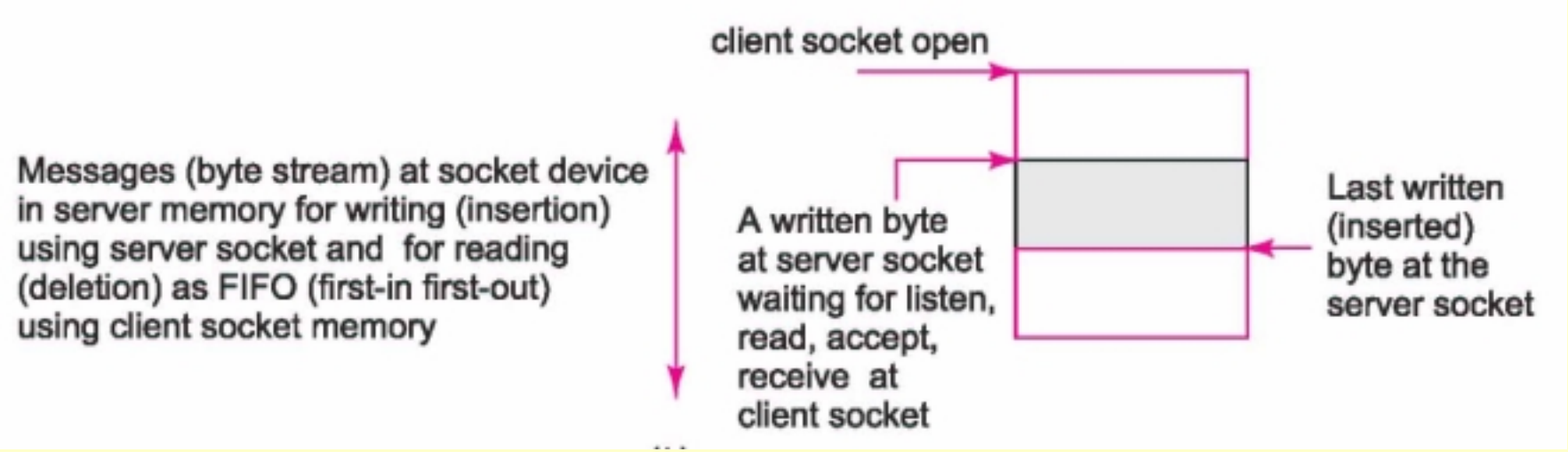


Client and Server Socket-device Functions

Client uses connect, read, write functions, and server uses bind, listen, accept, write and read functions in TCP like connection-oriented protocol

Client uses bind, send, receive functions, and server uses bind, receive and send functions in UDP like connection-less protocol

Byte Stream at server socket device



4. IPC Socket device functions Application Example

Task Master creating socket orchestra-playing robots

1. sfd1 = socket ("/socket/serversocket1", playStream, 0).

The sfd1 is an unsigned integer for a socket descriptor. "/socket/serversocket1" is the path and file from which stream, playStream will be sent or received from play robots. 0 represents unrestricted permission to use the file.

Task Master binding the created socket in orchestra-playing robots

2. Task_Master socket binds the sfd and data structure at the socket address, sockAddr by using the function as follows:

*bind (sfd1, (struct sockaddr *)&local, lBytes).*

lBytes = length of the bytes in the play stream

Task_Master socket listens to 8 orchestra-playing

- *Task_Master socket listens to 8 orchestra-playing robots by using a function as follows:*
- *listen (sfd1, 8)*

Task_Master socket accepting bytes from at socket with descriptor sfd2

- *sfd2 = accept (sfd1, &playRobotSockAddress, &playRlBytes)*

/* playRlBytes = maximum length of bytes from playing robot.

playRobotSockAddres= address of data structure for the client socket */

Task_Master socket sending the bytes

- send (sfd2, & playBuffer, playstreamlen, 0); /* Send total playstreamlen bytes from playBuffer using the socket sfd2.

Task_Master socket receives the bytes from the playing robot

```
while (streamlength > 0 && streamlength < =
   playRlBytes) { streamlength = recv (sfd2,
   &ackBuffer, 200, 0)};
/* The recv () returns –1 if no more bytes are
   to be received. Bytes are received in
   ackBuffer */
```

Task_Master socket closing

- close ( );

Task_Client in the playing robot

/*creates a client 1 socket*/

sfd = socket ("/socket/clientsocket1", sockStream, 0).

/*sfd1 is an unsigned integer for a socket descriptor */

/* "/socket/serversocket1" is the path and file from which stream, socStream will be sent or received from play robots. 0 represents unrestricted permission to use the file.*/

Task_Client socket connection

connect (*sfd*, (struct sockClientaddr *)&remote, CllBytes).

/* CllBytes is maximum length of the bytes in the play stream */

Task_CLient socket sending the acknowledgement bytes

```
send (sfd, & ackBuffer, playRlBytes, 0);
 /* Send total ackstreamlen bytes from
    ackBuffer using the socket sfd 8/
```

Task_Client socket receiving the bytes from the playing robots

- while (streamlength > 0 && streamlength <= playstreamlen) { streamlength = recv (sfd2, &palyBuffer, 200, 0)};
- /* The recv () returns –1 if no more bytes are to be received. Bytes are received in playBuffer */

Summary

We learnt:

- A Socket is an IPC for sending a byte stream or datagram from one or multiple task sockets to another task or server process socket as a bi-direction FIFO (pipe) like device using a protocol for transferring the bytes.
- The sockets can be a client server set of sockets (multiple processes and single server process) or peer-to-peer sockets IPC. A Socket has a number of applications.

We learnt:

- An Internet connection socket is for virtual connection between two ports: one port at an IP address to another port at another IP address.
- OS provides the IPC functions for creating socket, unlinking, binding, listening, accepting, receiving, sending and closing a socket device

End of Lesson-18: Socket