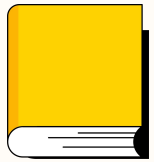


Communication in Distributed Systems

Reliable vs Unreliable Primitives

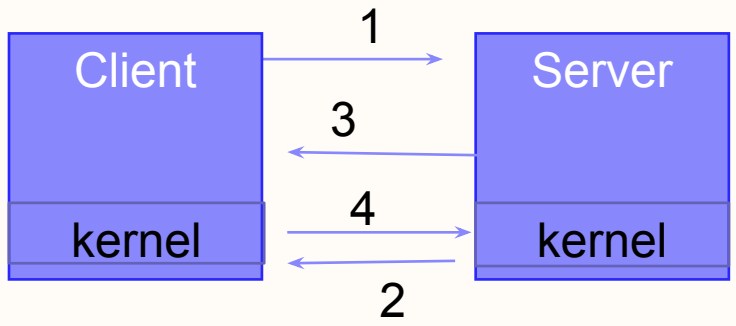
Using blocking primitives, the client process gets suspended after sending a message. When it is restarted, there is no guarantee that message has been delivered. The message might have been lost.

- Alternative solution1 - Redefine the semantics of *send* to be unreliable. The system gives no guarantee about message delivery.
- Alternative solution 2 - Kernel on the receiving machine sends an acknowledgement back to the kernel on sending machine.
- Sending kernel free the process after reeiving this acknowledgement.
- Similarly, the reply from server back to

client is acknowledged by client's kernel.

- Acknowledgement goes from kernel to kernel.

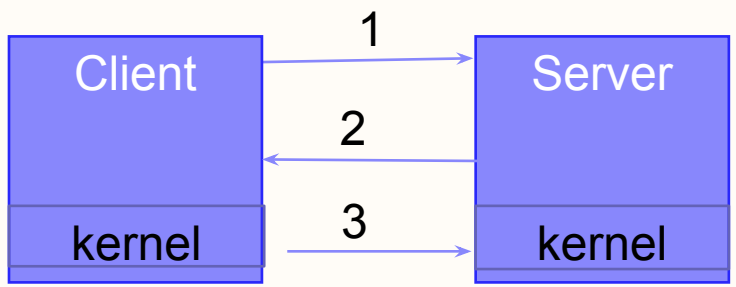
1. Request
2. ACK (kernel to kernel)
3. Reply
4. ACK (kernel to kernel)



Reliable vs Unreliable ....

- Alternative solution3 - Client is blocked after sending message and the server's reply act as an acknowledgement.
- If the reply takes too long, the sender can resend the mesage to guard against lost message.
- An acknowledgement from client's kernel to the server's kernel is sometimes used. Until this packet is received, the server's *send* does not complete and the server remains blocked.
- If the reply is lost and the request is retransmitted, then the server kernel sends reply again without waking up the server.

1. Request (client to server)
2. Reply (server to client)
3. ACK (kernel to kernel)



Implementing client-server model

Design issues for the communication primitives

Item	Option 1	Option 2	Option 3
Addressing	Machne address	Sparse process addresses	Names looked up via server
Blocking	Blocking primitives	Nonblocking with copy to kernel	Nonblocking with interrupt
Bufferring	Unbuffered, discarding unexpected messages	Unbuffered, temporarly keeping unexpected messages	Mailboxes
Reliability	Unreliable	Request-Ack-Reply-Ack	Request-Reply-Ack

How message passing is implemented depends on which choices are made.

Implementing client-server model

Issue of packet size - All packets have a limit of packet size. Messages larger than this must be split up into multiple packets and sent separately.

- Problem - some of these packets may be lost or distorted. They may even arrive in the wrong order.
- Solution - Assign a number to each message and put it in each packet belonging to that message, alongwith a sequence number indicating the order of the packet.

Issue of acknowledgement -Acknowledge each individual packet or Acknowledge entire message.

cae I - if a packet is lost, only one packet need to be retransmitted. But it will cause more number of acknowledgements.

case II - Fewer packets but more complicated to recover if a packet is lost.

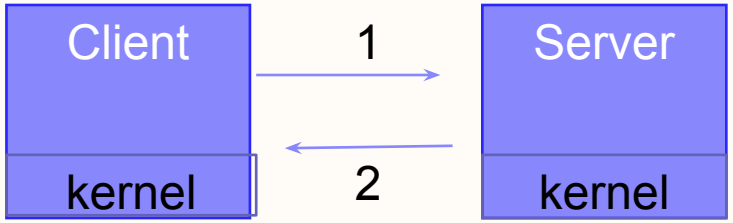
Implementing client-server model

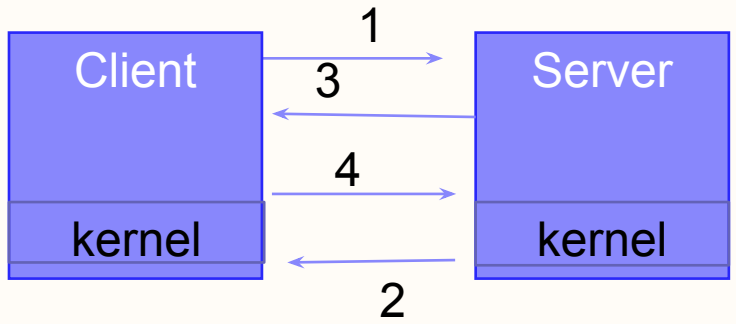
Issue of underlying protocol -

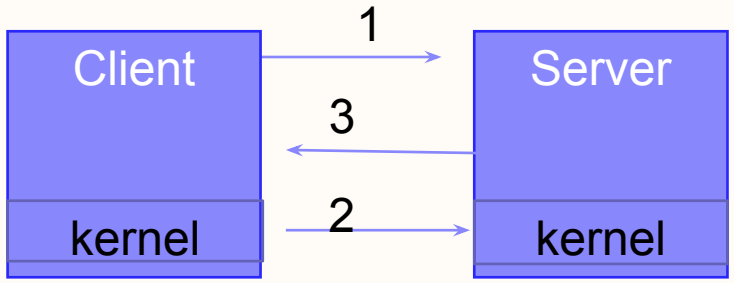
- AYA - to check whether request is complicated or the server is crashed
- TA - if request packet cannot be accepted

Code	Packet type	From	To	Description
REQ	Request	Client	Server	Client wants service
REP	Reply	Server	Client	Reply from server to the client
ACK	Acknowledgement	Either	Other	Previous packet arrived
AYA	Are you alive ?	Client	Server	Check if the server is crashed
IAA	I am alive	Server	Client	Server has not crashed.
TA	Try again	Server	Client	Server has no space
AU	Address unknown	Server	Client	No processis using this address

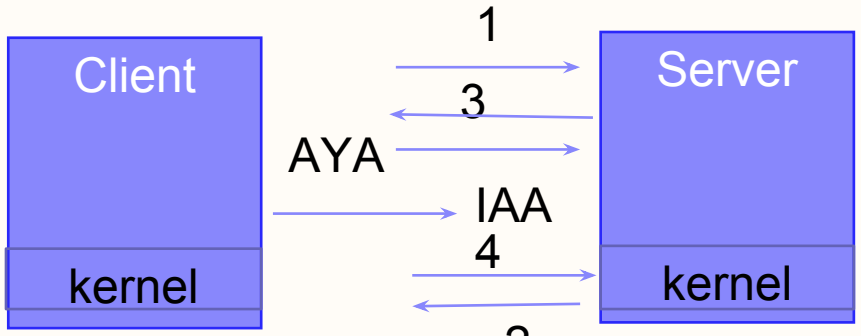
Implementing client-server model







1. Request
2. ACK (kernel to kernel)
3. Reply
4. ACK (kernel to kernel)



Examples of packet exchanges