

Synchronous Communication

- Client/Server computing generally based on a model of synchronous communication:
 - Client and server have to be active at the time of communication
 - Client issues request and blocks until it receives reply
 - Server essentially waits only for incoming requests, and subsequently processes them
- Drawbacks synchronous communication:
 - Client cannot do any other work while waiting for reply
 - Failures have to be dealt with immediately (the client is waiting)
 - In many cases the model is simply not appropriate (mail, news)

6/13/2005

28

Asynchronous Communication

- Message-oriented middleware
- Aims at high-level asynchronous communication:
 - Processes send each other messages, which are queued
 - Sender need not wait for immediate reply, but can do other things
 - Middleware often ensures fault tolerance

6/13/2005

29

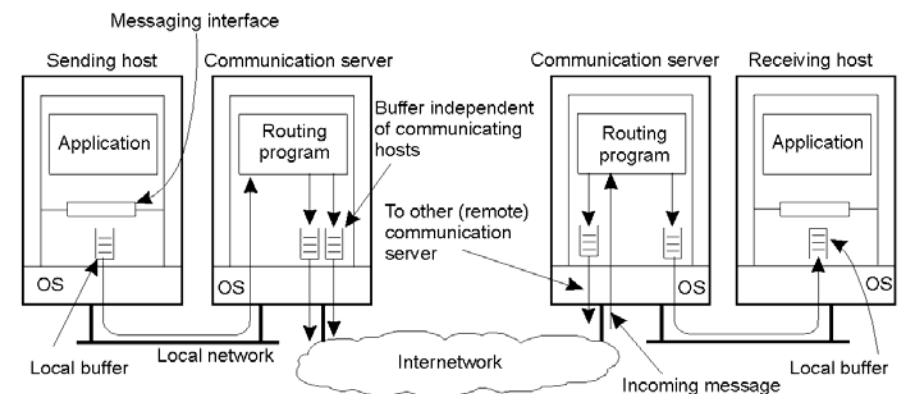
Persistent vs. Transient Communication

- Persistent communication
 - A message is stored at a communication server (client or server) as long as it takes to deliver it at the receiver
 - Whenever there is a failure, slower failure masking and recovery procedures can be initiated since application developed with persistent communication can handle long delays between sending a request and receiving an answer
 - Example: email
- Transient communication
 - A message is discarded by a communication server as soon as it cannot be delivered at the next server, or at the receiver
 - Whenever a failure occurs, failure masking and recovery procedures must be initiated immediately
 - Example: All transport-level communication services

6/13/2005

30

Persistence and Synchronicity in Communication (1)

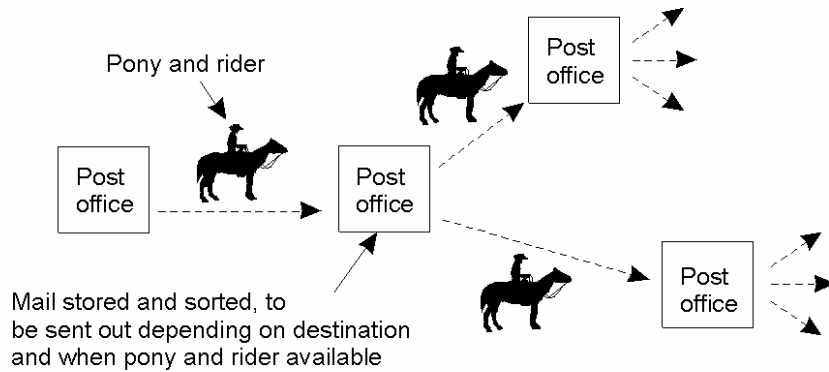


General organization of a communication system in which hosts are connected through a network

6/13/2005

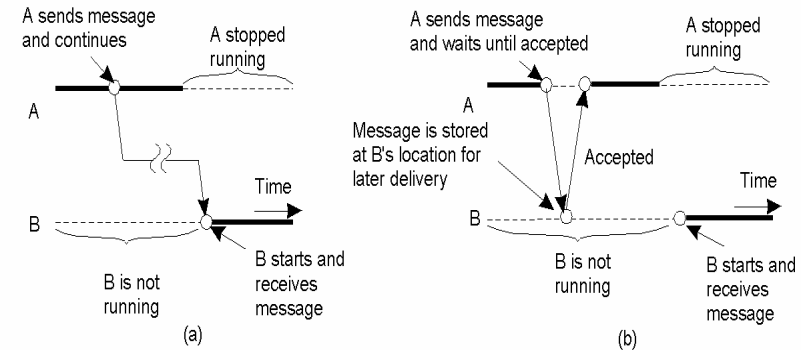
31

Persistence and Synchronicity in Communication (2)



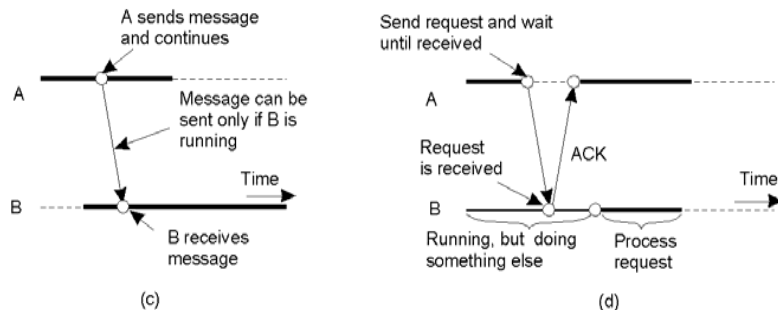
Persistent communication of letters back in the days of the Pony Express.

Persistence and Synchronicity in Communication (3)



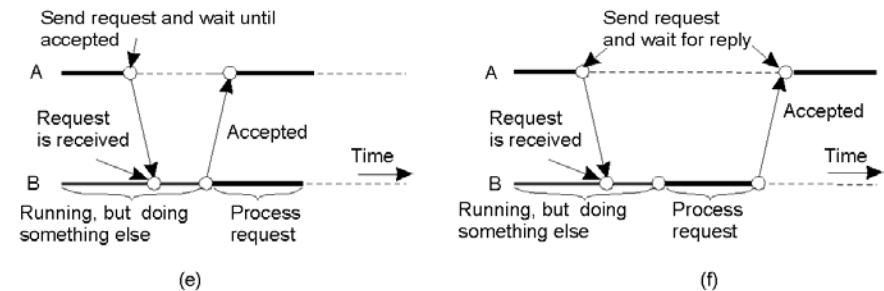
- Persistent asynchronous communication – message stored persistently at the local host or first communication server
- Persistent synchronous communication – message stored persistently only at the receiving host, sender waits until message is stored in receiver's buffer

Persistence and Synchronicity in Communication (4)



- Transient asynchronous communication – assumes receiver is ready (e.g., UDP, asynchronous RPC)
- Receipt-based transient synchronous communication – sender blocked until message stored at receiver

Persistence and Synchronicity in Communication (5)



- Delivery-based transient synchronous communication at message delivery – sender is blocked until message is delivered to receiver
- Response-based transient synchronous communication – strongest form of synchronous communication, sender blocked until the receiver send an acknowledgement (similar to request-reply with RPC and RMI)

Message-Oriented Communication

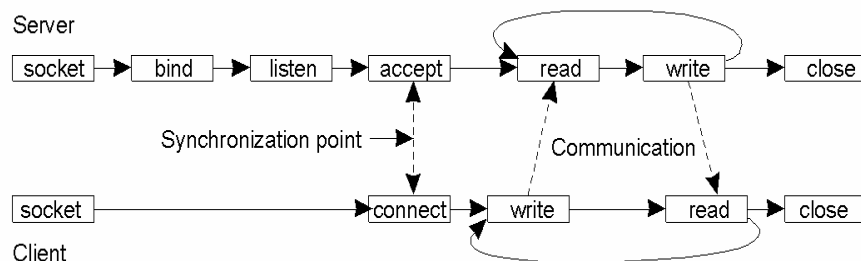
- Message-Oriented Transient Communication
 - Sockets
 - Message Passing Interface (MPI)
- Message-Oriented Persistent Communication
 - Message-Queuing Systems or Message-Oriented Middleware (MOM)
 - Examples: IBM MQSeries, J2EE Java Message Service (JMS)

Berkeley Sockets (1)

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Socket primitives for TCP/IP.

Berkeley Sockets (2)



Connection-oriented communication pattern using sockets.

The Message-Passing Interface (MPI)

Primitive	Meaning
MPI_Bsend	Append outgoing message to a local send buffer
MPI_Send	Send a message and wait until copied to local or remote buffer
MPI_Ssend	Send a message and wait until receipt starts
MPI_Sendrecv	Send a message and wait for reply
MPI_Isend	Pass reference to outgoing message, and continue
MPI_Issend	Pass reference to outgoing message, and wait until receipt starts
MPI_Recv	Receive a message; block if there are none
MPI_Irecv	Check if there is an incoming message, but do not block

Some of the most intuitive message-passing primitives of MPI.

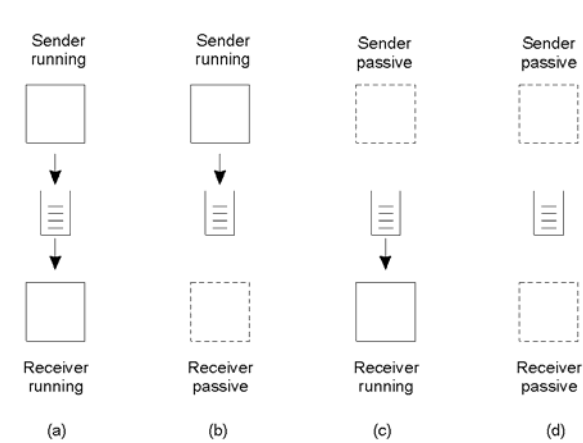
Message-Queuing Systems

- Provide asynchronous persistent communication through the use of queues, sender or receiver need not be active during message transmission
- Typically targeted to support message transfers that take several minutes not seconds or milliseconds
- Applications communicate by inserting messages into specific queues
- Sender has guarantees that the message will be eventually inserted in the recipient's queue, but no guarantees about when it will be delivered

6/13/2005

40

Message-Queuing Model (1)



Four combinations for loosely-coupled communications using queues. Sender and Receiver can execute completely independent of each other.

6/13/2005

41

Message-Queuing Model (2)

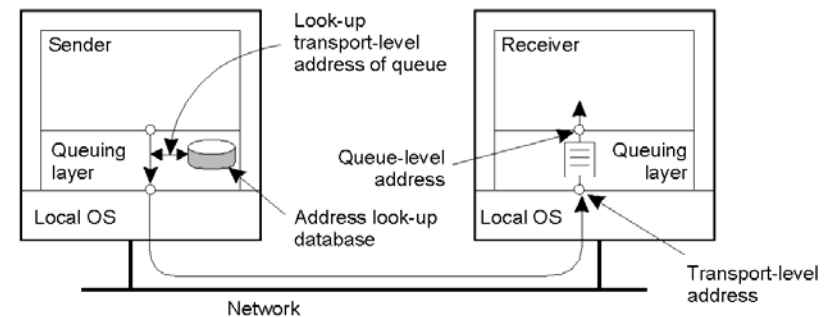
Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block.
Notify	Install a handler to be called when a message is put into the specified queue.

Basic interface to a queue in a message-queuing system.

6/13/2005

42

General Architecture of a Message-Queuing System (1)

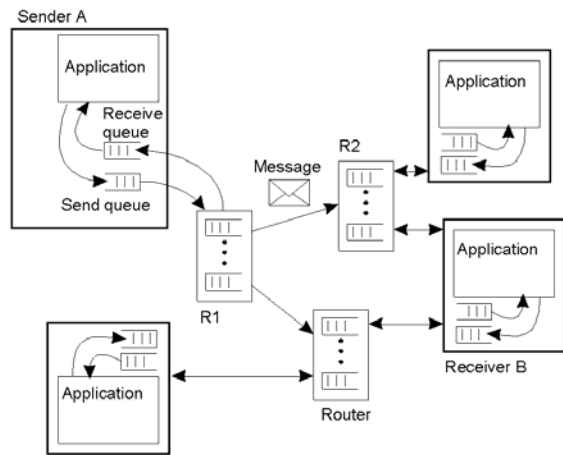


The relationship between queue-level addressing and network-level addressing (mapping analogous to use of Domain Name System – DNS for email on the Internet)

6/13/2005

43

General Architecture of a Message-Queuing System (2)

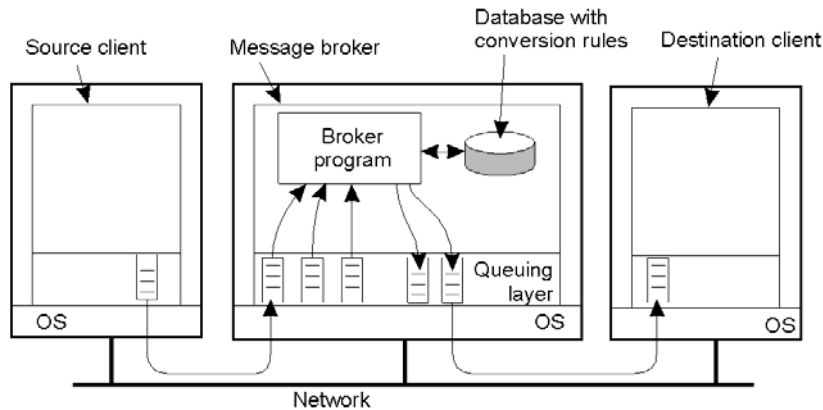


The general organization of a message-queuing system with routers.

Message Brokers

- A special node in the queuing system
- Acts as a application-level gateway in a message-queuing system
- Converts incoming messages to a format that can be understood by the destination application
- Uses a database of rules that specify how to convert a message in format X to format Y
- These rules must be formulated either manually or with a special conversion language

Message Brokers

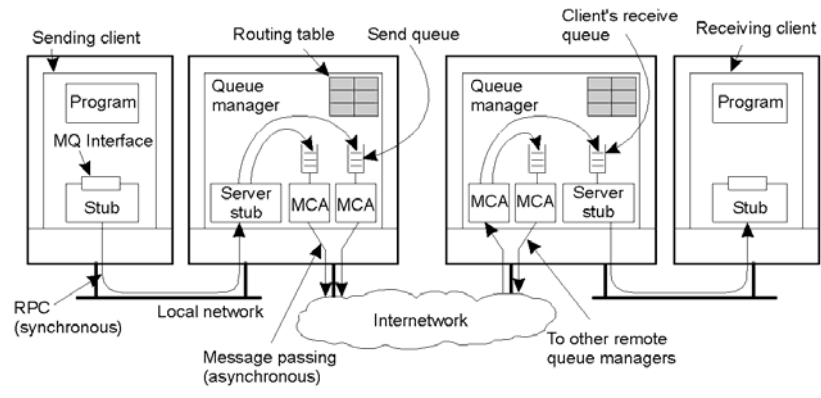


The general organization of a message broker in a message-queuing system

Example: IBM MQSeries

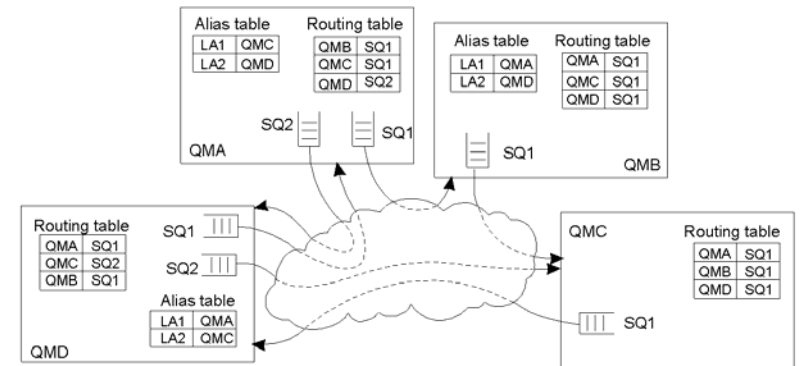
- Application-specific messages are put into, and removed from queues
- Queues always reside under the regime of a queue manager
- Processes can put messages only in local queues, or through an RPC mechanism
- Message transfer between queues at different processes, requires a channel (a reliable and unidirectional connection)
- At each endpoint of channel is a Message Channel Agent (MCA)
- Message channel agents are responsible for:
 - Setting up channels using lower-level network communication facilities (e.g., TCP/IP)
 - (Un)wrapping messages from/in transport-level packets
 - Sending/receiving packets

IBM MQSeries



General organization of IBM's MQSeries message-queuing system.

Message Transfer (1)



The general organization of an MQSeries queuing network using routing tables and aliases.

Message Transfer (2)

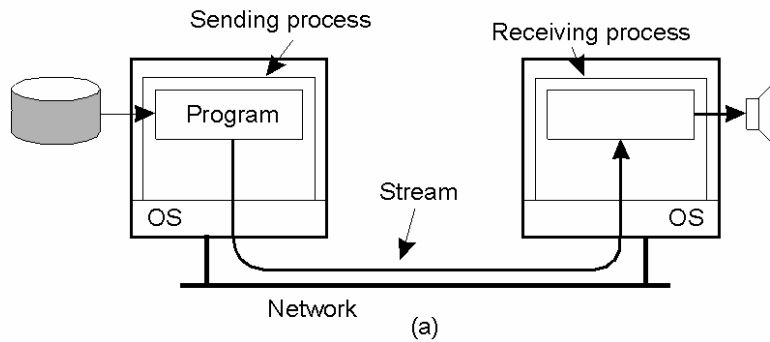
Primitive	Description
MQopen	Open a (possibly remote) queue
MQclose	Close a queue
MQput	Put a message into an opened queue
MQget	Get a message from a (local) queue

Primitives available in an IBM MQSeries MQI

Stream-Oriented Communication

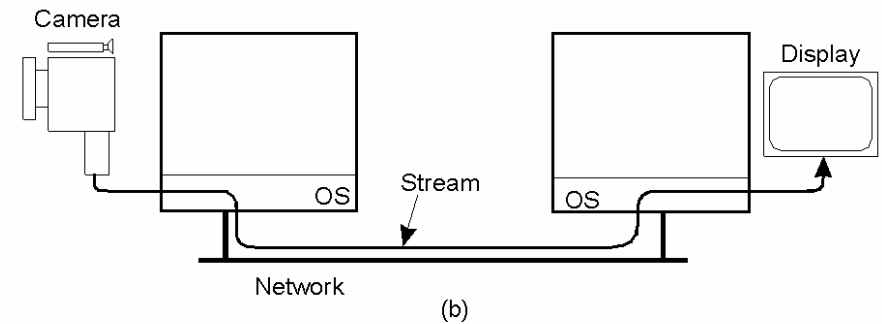
- Communication discussed so far involved exchange of independent and complete units of data
- Timing has no effect on correctness of communication
- Stream-oriented communication involves exchange of time-dependent information
- Transmission modes
 - Asynchronous: data items transmitted one after the other without any timing constraints on actual time of transfer
 - Synchronous: a maximum end-to-end delay is defined for transmission of each unit in a data stream
 - Isochronous: data transfer is subject to a maximum and minimum end-to-end delay (bounded jitter)
- Examples: Audio and video streams

Data Stream (1)



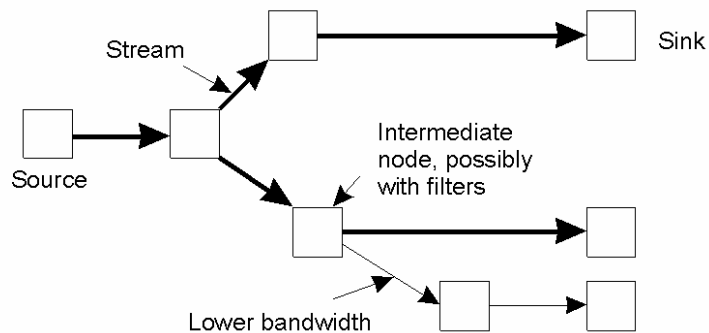
Setting up a stream between two processes across a network.

Data Stream (2)



Setting up a stream directly between two devices.

Data Stream (3)



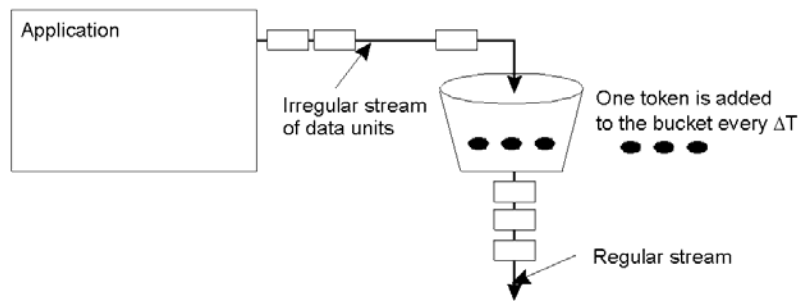
An example of multicasting a stream to several receivers.

Specifying QoS (1)

Characteristics of the Input	Service Required
<ul style="list-style-type: none"> • maximum data unit size (bytes) • Token bucket rate (bytes/sec) • Token bucket size (bytes) • Maximum transmission rate (bytes/sec) 	<ul style="list-style-type: none"> • Loss sensitivity (bytes) • Loss interval (μsec) • Burst loss sensitivity (data units) • Minimum delay noticed (μsec) • Maximum delay variation (μsec) • Quality of guarantee

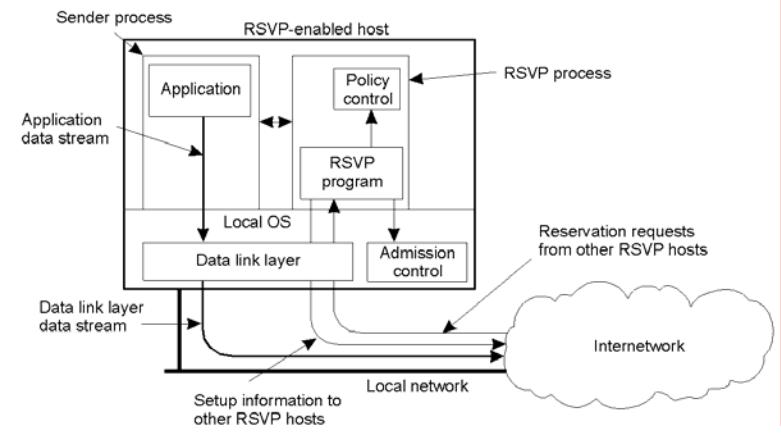
A flow specification.

Specifying QoS (2)



The principle of a token bucket algorithm.

Setting Up a Stream



The basic organization of Resource reSerVation Protocol (RSVP) for resource reservation in a distributed system.