

CSSE4004 – Lecture 3

Communication

1

Learning objectives

After this week you should be able to describe and compare various communication models used in Distributed Systems:

- Remote Procedure Call
- Message-oriented communication
- Stream-oriented communication
- Multicast communication
- (Remote Object Invocation will be covered in Lecture 7)

2

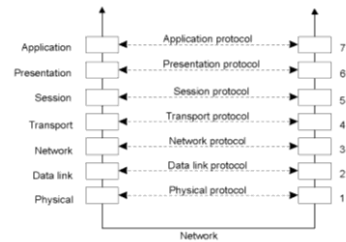
Motivation

- Interprocess communication in DS is based on message exchange
- Low level (transport) message passing does not provide distribution transparency – higher level models are needed
- Distributed applications need a variety of communication semantics

3

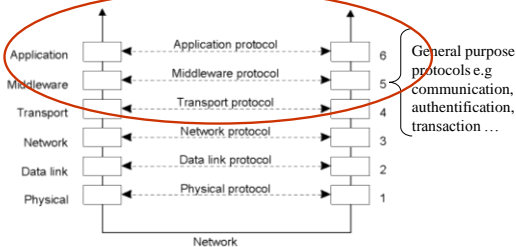
Layered Protocols

- Layers, interfaces, and protocols in the OSI model



4

Middleware Protocols



An adapted reference model for networked communication

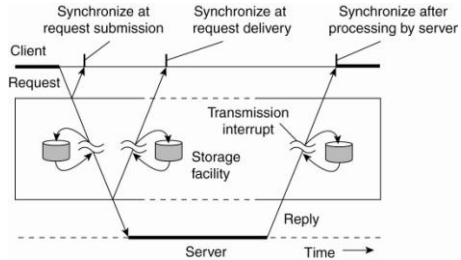
5

Middleware Layer

- Provides common protocols that can be used by many different applications
 - Security protocols
 - Transaction protocols
 - High-level communication models (e.g. RPC, message queuing services, etc).

6

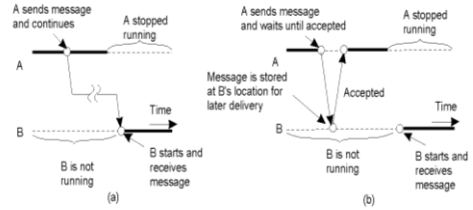
Types of Communication



Ta-St. 4-4. Viewing middleware as an intermediate (distributed) service in application-level communication

7

Synchronicity in Communication (1)

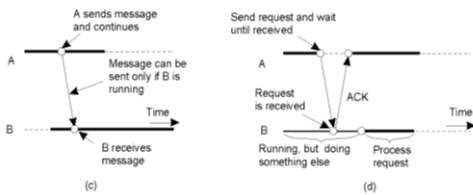


- a) Persistent asynchronous communication
- b) Persistent synchronous communication

Persistent communication – message is stored in the system until receiver becomes active

8

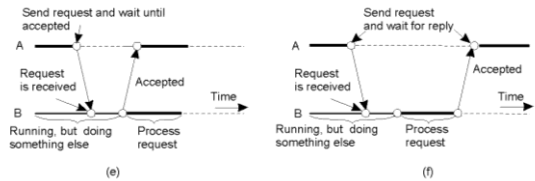
Synchronicity in Communication (2)



- c) Transient asynchronous communication
- d) Receipt-based transient synchronous communication

9

Synchronicity in Communication (3)

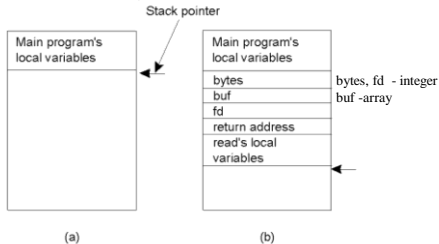


- e) Delivery-based transient synchronous communication at message delivery
- f) Response-based transient synchronous communication

10

Conventional Procedure Call

count=read(fd, buf, bytes)



- a) Parameter passing in a local procedure call: the stack before the call to read
- b) The stack while the called procedure is active

11

Remote Procedure Call-Client and Server Stubs



Principle of RPC between a client and server program

12

Steps of a Remote Procedure Call

1. Client procedure calls client stub (procedure invocation)
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

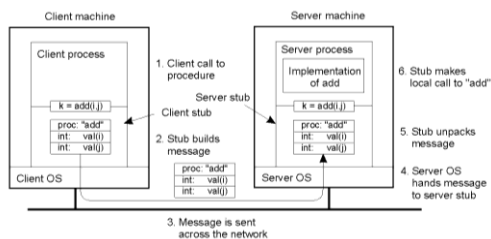
13

Parameter Passing

- Parameter marshaling: There's more than just wrapping parameters into a message
 - Client & Server
 - have different data representations
 - have to agree on the same encoding (basic/complex data values)
 - Interpret data and transform them into machine-dependent representation

14

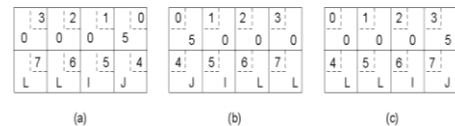
Passing Value Parameters (1)



15

Passing Value Parameters (2)

Different data representations...



- Original message on the Pentium
- The message after receipt on the SPARC
- The message after being inverted. The little numbers in boxes indicate the address of each byte

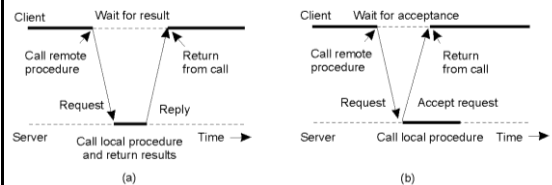
16

Parameter Specification and Stub Generation

- Parameters passed by value do not pose problems
- Parameters passed by reference – some partial solutions exist (e.g. a small array can be sent to the server)
- Interfaces (procedures) are often specified in Interface Definition Languages (IDL) and compiled into stubs

17

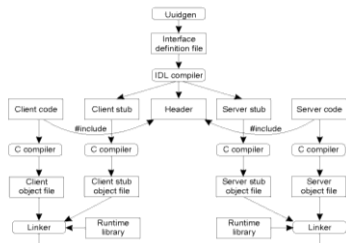
Asynchronous RPC



- The interconnection between client and server in a traditional RPC
- The interaction using asynchronous RPC

18

DCE – Distributed Computing Environment- Writing a Client and a Server



The steps in writing a client and a server in DCE RPC

Writing a Client and a Server

- Three files output by the IDL compiler:
 - A header file (e.g., interface.h, in C terms)
 - The client stub
 - The server stub

DCE provides a typical example of static (compile time) generation of stubs

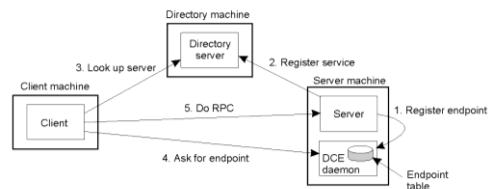
20

Binding a Client to a Server

- Registration of a server makes it possible for a client to locate the server and bind to it
- Server location is done in two steps:
 1. Locate the server's machine
 2. Locate the server on that machine

21

DCE: Binding a Client to a Server



Client-to-server binding in DCE

22

Message-Oriented communication

- RPC (and RMI which will be described in Lecture 7) are synchronous and transient (both sender and receiver have to be active)
- Transport level communication like TCP and UDP also require that the sender and receiver are active (transient communication)
- Some applications require that messages are kept in the system until the receiver becomes active (e-mail is one example, but there are many more applications of this type)

23

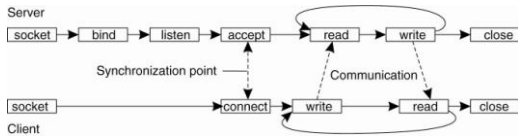
Transient communication – Berkeley Sockets

Primitive	Meaning
Socket	Create a new communication end point
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

Ta-St. 4-14. The socket primitives for TCP/IP

24

Berkeley Sockets



TA-St. 4-15. Connection-oriented communication pattern using sockets

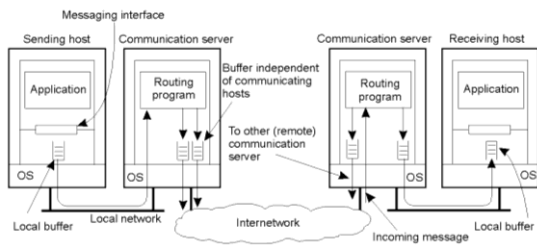
25

Message-oriented persistent asynchronous communication

- MOM – Message-Oriented Middleware
- Can store messages in the system
- Applications communicate by inserting messages in queues
- Messages are delivered by an overlay network of application layer servers (routers)
- Each application has its own private queue to which other applications can send messages
- There is no guarantee on when the message will be delivered and that it will be read.

26

Message-oriented persistent asynchronous communication

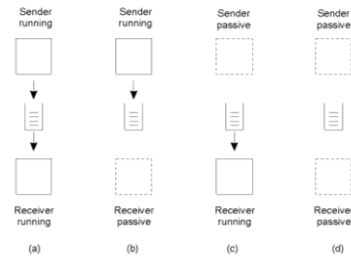


Message-oriented middleware aims at high level asynchronous communication

27

Message-Queuing Model (1)

MQMs goal is to provide persistent asynchronous comm.



Four combinations for loosely-coupled communications using queues

28

Message-Queuing Model (2)

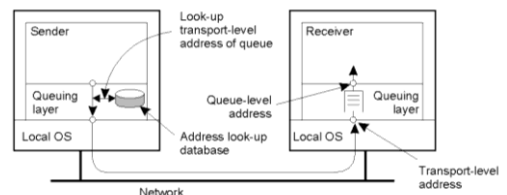
Basic interface to a queue in a message-queuing system:

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.

29

General Architecture of Message-Queuing Systems (1)

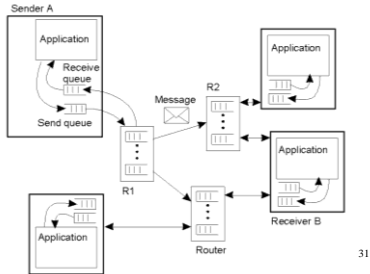
The relationship between queue-level addressing and network-level addressing:



30

General Architecture of Message-Queuing Systems (2)

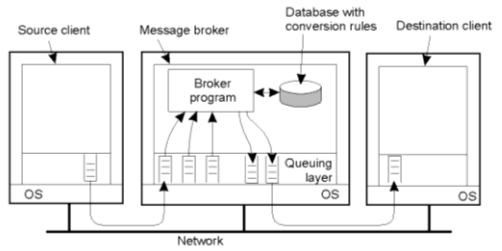
The general organization of a message-queuing system with routers:



31

Message Brokers

Message brokers are used to provide message format conversions:



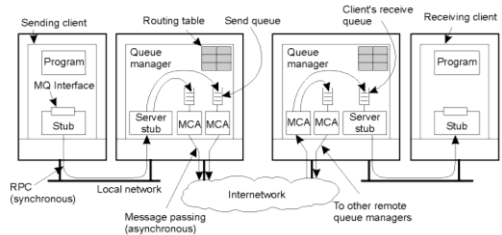
32

Brokers supporting publish-subscribe

- Main role of brokers is to transform messages from sender's format to receiver's format
- This functionality can be generalised to brokers matching applications based on messages
 - Applications *publish* messages
 - Applications *subscribe* for messages

33

MOM Example: IBM WebSphere MQ



34

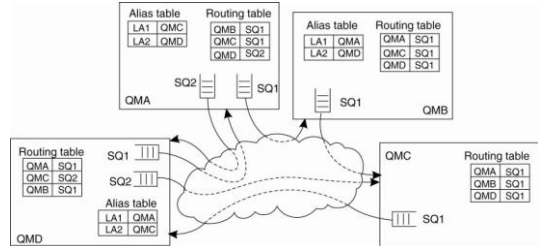
Channels

Attribute	Description
Transport type	Determines the transport protocol to be used
FIFO delivery	Indicates that messages are to be delivered in the order they are sent
Message length	Maximum length of a single message
Setup retry count	Specifies maximum number of retries to start up the remote MCA
Delivery retries	Maximum times MCA will try to put received message into queue

Ta-St. 4-23. Some attributes associated with message channel agents

35

Message Transfer (1)



Ta-St. 4-24. The general organization of an MQ queuing network using routing tables and aliases³⁶

36

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

Ta-St. 4-25. Primitives available in the message-queuing interface

37

Stream-oriented communication

- Some applications need to exchange time-dependent information e.g. video, audio (continuous media)
- Previously discussed models do not consider time
- In continuous media the temporal relationship between different data items is essential

38

Different transmission modes

- Asynchronous transmission mode
 - Sequential transmission without restrictions on when data is to be delivered (e.g. file transfer)
- Synchronous transmission mode
 - Max end-to-end delay for each unit in a data stream (e.g. sensor sample temperature)
- Isochronous transmission mode (streams)
 - Data transfer bounded by maximum and minimum end-to-end delay (bounded jitter e.g. audio/video)

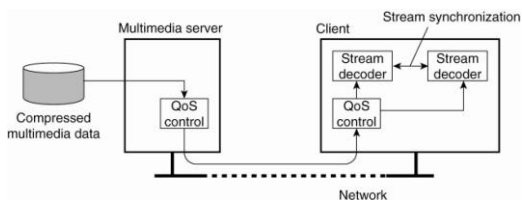
39

Streams

- Streams can be simple or complex (i.e. can include several related substreams)
 - Unidirectional (source → sinks)
 - Simple (single flow of data e.g. audio, video)
 - Complex (stereo audio, movie)

40

Client-server for multimedia streams



Ta-St. 4-26. A general architecture for streaming stored multimedia data over a network

41

Streams and QoS (1)

Streams need timely delivery of data

- Non functional requirements (time, bandwidth) are expressed as Quality of Service (QoS)
- There are many models for QoS specifications: e.g. IntServ, DiffServ, MPLS
- In IntServ (Integrated Services), QoS is specified as a flow specification – flow reservation
- In DiffServ (Differentiated Services), QoS is specified for a class (e.g. expedited forwarding class)
- We look at DiffServ only (check COMS3200 for other models)

42

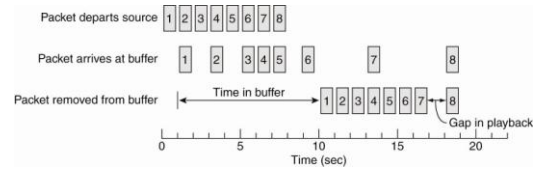
Streams and QoS (2)

Properties for Quality of Service:

- The required bit rate at which data should be transported
- The maximum delay until a session has been set up
- The maximum end-to-end delay
- The maximum delay variance, or jitter
- The maximum round-trip delay

43

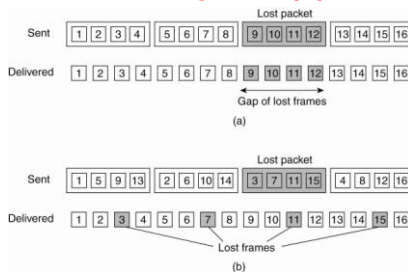
Enforcing QoS (1)



Ta-St. 4-27. Using a buffer to reduce jitter

44

Enforcing QoS (2)



Ta-St. 4-28. The effect of packet loss in (a) non interleaved transmission and (b) interleaved transmission

45

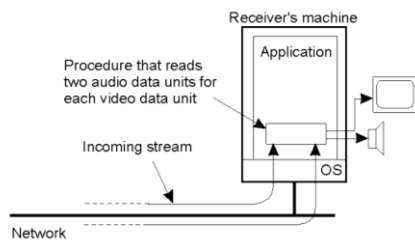
Stream synchronization

- Given a complex stream, how are substreams synchronised?
- Synchronisation takes place at the level of data units
- Issues which need to be considered:
 - Mechanisms for synchronising streams
 - Distribution of those mechanisms

46

Synchronization Mechanisms (1)

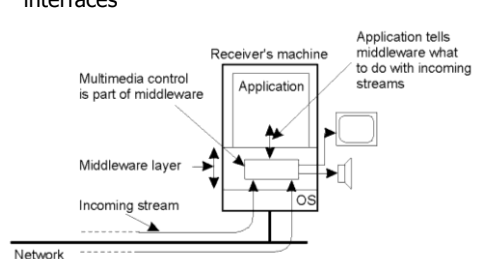
The principle of explicit synchronization on the level of data units



47

Synchronization Mechanisms (2)

The principle of synchronization through high-level interfaces



48

Distribution of synchronisation mechanisms

- The receiving side needs to have a complete synchronisation specification
- Synchronisation specification can be multiplexed together with other substreams into a complex stream (and is demultiplexed after receiving)
- Example: MPEG
Streams are multiplexed into a single stream, each packet has a timestamp which is used for synchronisation) – in this case synchronisation is provided by the sender

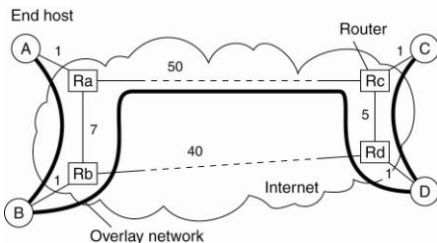
49

Multicast communication

- Application level multicasting – setting a path for information dissemination
 - Nodes (applications) organise into an overlay network
 - Overlay network disseminates information
 - Network layer routing is independent of the overlay (communication may not be optimal)
- Overlay organisation
 - Nodes may organise themselves into a tree, or
 - Nodes organise themselves into a mesh network

50

Overlay Construction



Ta-St. 4-31. The relation between links in an overlay and actual network-level routes

51

Gossip-based data dissemination

- Epidemic protocols are often used for data dissemination
 - There is no central component which coordinates dissemination
 - Information is propagated using local information only
- Node is *infected* if it has data which it wants to spread
- Node which has not seen this data is *susceptible*

52

Information Dissemination Models

- Anti-entropy propagation model
 - Node P picks another node Q at random
 - Subsequently exchanges updates with Q
- Approaches to exchanging updates
 - P only pushes its own updates to Q
 - P only pulls in new updates from Q
 - P and Q send updates to each other
- One variant is the “gossiping” protocol

53

Summary

- One model of communication is not suitable for all distributed applications
- A higher level of abstraction than offered by the transport layer is important
- RPC (synchronous) is a popular communication mechanism but RMI offers better transparency (RMI will be discussed in lecture 7)
- Message-oriented models usually provide asynchronous and persistent communication
- Continuous media requires a different approach (stream communication)
- Often data has to be sent to many receivers (multicast communication)
- Reading: Chapter 4 (textbook)

54