Concepts of Distributed Systems
2006/2007

Communication

Johan Lukkien
Programme

- Introduction & overview
- Communication
- Distributed OS & Processes
- Synchronization
- Security
- Consistency & replication
- Naming
- Fault Tolerance
Communication

- Some concepts and definitions
- Layered protocols
- Remote Procedure Calls
- Remote Method Invocation
- Message oriented communication
- Stream oriented
Communication modes

• Synchronization
  – Synchronous: send and receive are coupled
  – Buffered: bounded send surplus
  – Asynchronous: sender never blocked

• Message existence
  – Persistent: subsystem will deliver messages
  – Transient: message life depends on life of sender, receiver, subsystem

• Timing
  – Time-independent
  – Time-dependent (real-time)
    • soft real-time, hard real-time
Connections

• Connection oriented
  – setup & destroy
  – control: reliability, flow, congestion

• Connection-less
  – each message single-shot

• Sources and destinations
  – many to one
  – one to many
  • with addressing (multicast) or without (broadcast)
  – many to many
Routes and forwarding

- **Packet-switched**
  - each packet is dealt with (routed) separately
  - difficult to guarantee end-to-end quality
    - usually: over-dimensioning
  - much more efficient use of resources

- **Circuit-switched**
  - information is forwarded along a reserved, end-to-end route
  - admits quality guarantees
  - fits connection oriented message transfer
Definition: protocol

- **Protocol**: A formal set of rules that dictates how information exchange as well as interaction between objects (can be devices, execution threads, etc.) should take place.

The rules specify
- the format of the messages exchanged;
- a number of different protocol states and what messages are allowed to be sent in each state; these states determine, among others, the order of the messages.
- timing constraints and other non-functional properties, if any.
Definition: service

• **Service**: a contractually specified overall functionality (semantics) of an entity.

• **Service quality**: non-functional properties of a service (e.g. speed, reliability, ...).

• **Service interface (API)**: actions (“primitives”) and responses that make the service available; these responses can be autonomous (“events”, “call-backs”). In addition, a specification that
  – describes their effect on state variables and parameters, as well as their results;
  – describes rules as how and in what sequence to call them;
  – describes the functional and non-functional properties of sequences of calls.

(i.e., the interaction protocol)
Service examples

- Typically, the communication modes
- Quality dimensions
  - reliable, acknowledged, guaranteed bandwidth, low latency

<table>
<thead>
<tr>
<th>Service</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable message stream</td>
<td>Sequence of pages</td>
</tr>
<tr>
<td>Reliable byte stream</td>
<td>Remote login</td>
</tr>
<tr>
<td>Unreliable connection</td>
<td>Digitized voice</td>
</tr>
<tr>
<td>Unreliable datagram</td>
<td>Electronic junk mail</td>
</tr>
<tr>
<td>Acknowledged datagram</td>
<td>Registered mail</td>
</tr>
<tr>
<td>Request-reply</td>
<td>Database query</td>
</tr>
</tbody>
</table>
Example: primitives for connection oriented service

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTEN</td>
<td>Block waiting for an incoming connection</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Establish a connection with a waiting peer</td>
</tr>
<tr>
<td>RECEIVE</td>
<td>Block waiting for an incoming message</td>
</tr>
<tr>
<td>SEND</td>
<td>Send a message to the peer</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>Terminate a connection</td>
</tr>
</tbody>
</table>
Definition: carrier, binding

- A protocol can be realized directly in hardware. If not, the messages of a protocol are given to another protocol, called the *carrier*.

- The functionality and properties that a protocol provides is called the *provided service*. The provided service is often specified by an API for the protocol.

- A protocol requires services from its carrier and any carrier providing these services can be used. The rules that specify how a protocol is mapped onto a carrier is called a *binding*.
Communication

- Some concepts and definitions
- Layered protocols
- Remote Procedure Calls
- Remote Method Invocation
- Message oriented communication
- Stream oriented
OSI(’83): Basic reference networking model

- Layers communicate with peers and with neighbors (provided & required services)
- OSI protocols:
  - not used
- OSI reference:
  - widely accepted
Some stacks and the OSI model

Protocol Stacks in Relationship to the OSI Model

<table>
<thead>
<tr>
<th>OSI Layer</th>
<th>Application Layer 7</th>
<th>Presentation Layer 6</th>
<th>Session Layer 5</th>
<th>Transport Layer 4</th>
<th>Network Layer 3</th>
<th>Data Link Layer 2</th>
<th>Physical Layer 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NIC Drivers: Open Data Link Interface (DIX), Network Independent Interface Specification (NDIS)</td>
<td></td>
</tr>
<tr>
<td>Banyan Systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DCEnet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IBM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SNA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Networking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Novell</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetWare</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP/IP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xerox</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Protocols</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Application Layer 7: Application Programs and Protocols for file transfer, electronic mail, etc.

Presentation Layer 6: Application Layer 7 Protocols such as X.400, PGP, Kerberos, etc.

Session Layer 5: Presentation Layer 6 Protocols such as X.25, HDLC, LAPB, etc.

Transport Layer 4: Session Layer 5 Protocols such as TCP, UDP, SPX, IPX, etc.

Network Layer 3: Transport Layer 4 Protocols such as X.25, HDLC, LAPB, etc.

Data Link Layer 2: Network Layer 3 Protocols such as Ethernet, Token-Ring, ARQNET, StarLAN, LocalTalk, FDDI, ATM, etc.

Physical Layer 1: Data Link Layer 2 Protocols such as Ethernet, Token-Ring, ARQNET, StarLAN, LocalTalk, FDDI, ATM, etc.
Layered Protocols and message layout

Data link layer header
  Network layer header
    Transport layer header
      Session layer header
        Presentation layer header
          Application layer header

Message

Bits that actually appear on the network

Data link layer trailer
Issues, to be resolved by the layers

- Error detection, correction
- Flow control
- Addressing
- Multiplexing
- Naming
- Congestion control
- Mobility
- Routing
- Fragmentation
- Security
- ....
Lower layer tasks

- Physical: put bits and bytes on media
  - voltage levels, radio signals, direction

- Data Link (simplified):
  - transport frames
    - frame separation, error recovery (check-sum....), flow control
      (sequence numbers, on/off, ...)
  - medium access, local addressing (MAC)

- Network: network-(world-) wide packet routing
  - IP!
  - the lowest level in many distributed system
Conversation in Data Link Layer

<table>
<thead>
<tr>
<th>Time</th>
<th>A</th>
<th>B</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data 0</td>
<td></td>
<td>A sends data message 0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Data 0</td>
<td>B gets 0, sees bad checksum</td>
</tr>
<tr>
<td>2</td>
<td>Data 1</td>
<td>Control 0</td>
<td>A sends data message 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B complains about the checksum</td>
</tr>
<tr>
<td>3</td>
<td>Control 0</td>
<td>Data 1</td>
<td>Both messages arrive correctly</td>
</tr>
<tr>
<td>4</td>
<td>Data 0</td>
<td>Control 1</td>
<td>A retransmits data message 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B says: &quot;I want 0, not 1&quot;</td>
</tr>
<tr>
<td>5</td>
<td>Control 1</td>
<td>Data 0</td>
<td>Both messages arrive correctly</td>
</tr>
<tr>
<td>6</td>
<td>Data 0</td>
<td></td>
<td>A retransmits data message 0 again</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Data 0</td>
<td>B finally gets message 0</td>
</tr>
</tbody>
</table>
Transport layer

- End-to-end communication
  - recover lost packets, re-order
    - sliding-window protocols
  - adjust speed to network performance, congestion
    - e.g. TCP-compliance
  - multiplexing
    - several ports

- On the Internet
  - TCP: transmission control protocol (connection oriented, stream)
  - UDP: user datagram protocol
  - RTP: real-time protocol (RTP/UDP)
TCP and transactions (T/TCP)

(a) SYN
SYN,ACK(SYN)
ACK(SYN)
request
FIN
ACK(req+FIN)
answer
FIN
ACK(FIN)

(b) SYN, request, FIN
SYN, ACK(FIN), answer, FIN
ACK(FIN)
Higher layer protocols

- Often, application specific
  - ftp, http, nntp, smtp, ...
- Usually, the API non-existent

- Integration of common protocols can provide a rich set of application-independent functionalities
The protocol triangle

- IP/everything tendency
  - goal in the layering: connect everything
  - TCP/IP as basic interoperability

- Application needs are diverse
  - use transport layer for private purpose (interest in meaning of the bits)
  - no direct need to support standards
Question

• Suppose that you want to have a multicast communication. Where should this be addressed?

  – Network layer: multicast addressing
    • efficient: packets go where they should
    • e.g. IP multicast

  – Transport layer: message replication from the start

  – Middleware: overlay network
    • limits communication
    • physical distance may be smaller than logical distance
    • penalty for high-level handling
Communication

- Some concepts and definitions
- Layered protocols
- Remote Procedure Calls
- Remote Method Invocation
- Message oriented communication
- Stream oriented
Conventional Procedure Call

- \texttt{read(int fd, unsigned char \ast buf, int bytes)}

- Natural unit of encapsulation for concurrency
  - combines functionality and message passing!
Remote Procedure Call

- Issues
  - 'translation' of regular call into message passing
  - packing parameters and results
    - call by value, reference
Steps in calling an RPC

- Remark: \( k \) can be ignored in the call
  - or it should be treated as a reference parameter
Steps in calling an RPC

1. Client procedure calls client stub in normal way
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
Need for Marshaling

(a) Original message on the Pentium
(b) The message after receipt on the SPARC
(c) The message after being inverted. The little numbers in boxes indicate the address of each byte
Parameter Specification and Stub Generation

(a) A procedure
(b) The corresponding message.

```c
foobar( char x; float y; int z[5] )
{
    ....
}
```

<table>
<thead>
<tr>
<th>foobar's local variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>z[0]</td>
</tr>
<tr>
<td>z[1]</td>
</tr>
<tr>
<td>z[2]</td>
</tr>
<tr>
<td>z[3]</td>
</tr>
<tr>
<td>z[4]</td>
</tr>
</tbody>
</table>

(a) (b)
RPC parameter passing

• Usually, value/result semantics
  – no assumptions *during* the call
  – no reference parameters, no statics
  – ...lack of transparency

• Include reference parameters
  – need remote reference mechanism
  – ....just another RPC?
  – or values of references parameters should be passed as well
  • concurrency? value/result?
Asynchronous RPC (1)

a) Traditional RPC

b) Asynchronous RPC
Deferred synchronous RPC

- Basically, two asynchronous RPC’s
Developing with RPC’s

- Stubs and descriptions generated automatically
- Just focus on application development

Figure: DCE process
Example: DCE

- Distributed Computing Environment

- Needs discovery facilities
  - endpoint: place to send messages
  - at directory: server’s address
Communication

- Some concepts and definitions
- Layered protocols
- Remote Procedure Calls
- Remote Method Invocation
- Message oriented communication
- Stream oriented
Distributed Objects

• Compile-time – static, closed approach
  – language fixed
  – automatically generated stubs, proxies etc.

• Run-time – dynamic, open
  – any language, any application can be an object
  – need “object adapters” to find/bind interface
  – focus on “access protocol”

• Transient
  – die with server (e.g., the shopping cart)

• Persistent
  – remain (passively) available
Using Remote Objects

often: process/thread per object

Client machine

Client invokes a method

Client

Proxy

Same interface as object

Server machine

Object

Skeleton invokes same method at object

Server

Skeleton

State

Method

Interface

Network

Marshalled invocation is passed across network
Binding a Client to an Object

- Binding: ‘get a reference to work’
  - download stub code (platform?) or create proxy
  - bind explicitly or implicitly
    - followed by static or dynamic invocation
      - dynamic: \texttt{invoke} (object ref, method name, parameters)
        \texttt{[no knowledge at language level]}
      - identification by pair (server, object)

- Can use object references easily in this way
  - object server becomes client for object references sent as parameters

- Value parameters
  - need to serialize object
  - difficult across platforms
Parameter Passing

• \textit{RMI (C, obj.method (L1, R1))}
Communication

- Some concepts and definitions
- Layered protocols
- Remote Procedure Calls
- Remote Method Invocation
- Message oriented communication
- Stream oriented
Observations

- RPC & RMI rather synchronous
  - in time: wait for reply
  - in space: shared data is known
    - though not all machines need all objects; better than a single database
  - functionality and communication coupled

- Look for communication models with better decoupling
  - message oriented – communication and functionality separated
    - more abstract? or more basic?
General networked organization

- Communication subsystem (e.g. email)
  - Asynchronous/synchronous
  - Transient/persistent
Combinations

a) Persistent asynchronous communication
b) Persistent synchronous communication
   • though debatable whether B should accept first
c) Transient asynchronous communication

d) Receipt-based transient synchronous communication
Combinations (cnt’d)

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication
Question

- Example of
  a) email
  b) email with notification
  c) datagram services (e.g. UDP)
  d) asynchronous RPC
  e) asynchronous RPC
  f) synchronous RPC
Message oriented systems

- Typical: asynchronous
- Small: transient
- Scaling: persistent
Examples

- Message queueing systems
  - persistent, asynchronous
  - e.g. IBM MQseries (Tanenbaum & van Steen)

- Java Message Service
  - transient?, asynchronous

- MPI: message passing interface
  - high-performance computing
  - transient, optional: (a)synchronous
The Message-Passing Interface

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_bsend</td>
<td>Append outgoing message to a local send buffer</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Send a message and wait until copied to local or remote buffer</td>
</tr>
<tr>
<td>MPI_ssend</td>
<td>Send a message and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_sendrecv</td>
<td>Send a message and wait for reply</td>
</tr>
<tr>
<td>MPI_isend</td>
<td>Pass reference to outgoing message, and continue</td>
</tr>
<tr>
<td>MPI_issend</td>
<td>Pass reference to outgoing message, and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_recv</td>
<td>Receive a message; block if there are none</td>
</tr>
<tr>
<td>MPI_irecv</td>
<td>Check if there is an incoming message, but do not block</td>
</tr>
</tbody>
</table>
General Architecture of queuing systems

- Use logical names for source and destination queues
  - queue manager in overlay network (relay/routing)
  - need to map names to addresses
Message Brokers

- Conversion may be needed: no common format
- Broker: intermediate between source and destination
Communication

- Some concepts and definitions
- Layered protocols
- Remote Procedure Calls
- Remote Method Invocation
- Message oriented communication
- Stream oriented
Streams

- Stream: no imposed message structure
  - ‘continuous’ media, e.g. multimedia
  - single source, multiple sinks
  - usually: timing requirements
  - e.g. sockets
Multicast & transcoding
Timing requirements

- Bounded delay
  - end-to-end
- Bounded interpacket delays
  - minimum & maximum: jitter – isochronous
- Soft (firm) real-time
  - missing a packet is a pity but no disaster
  - graceful reduction in quality

- General issue: quality of service
  - no discrete success/failure but a range of qualities, decided dynamically
  - is often end-to-end concern, points along the route must cooperate
  - RSVP: resource reservation protocol
Synchronization of streams

- discrete + continuous
  - e.g. slides + sound
- 2 continuous but different granularities
  - e.g. sound & video
- at sender
  - encode within stream
- at receiver
  - fairly difficult to control
    - e.g., video to desktop, sound to earphone
Example QoS specification

<table>
<thead>
<tr>
<th>Characteristics of the Input</th>
<th>Service Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>• maximum data unit size (bytes)</td>
<td>• Loss sensitivity (bytes)</td>
</tr>
<tr>
<td>• Token bucket rate (bytes/sec)</td>
<td>• Loss interval (µsec)</td>
</tr>
<tr>
<td>• Token bucket size (bytes)</td>
<td>• Burst loss sensitivity (data units)</td>
</tr>
<tr>
<td>• Maximum transmission rate (bytes/sec)</td>
<td>• Minimum delay noticed (µsec)</td>
</tr>
<tr>
<td></td>
<td>• Maximum delay variation (µsec)</td>
</tr>
<tr>
<td></td>
<td>• Quality of guarantee</td>
</tr>
</tbody>
</table>