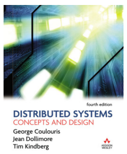
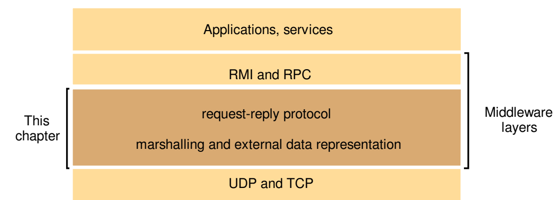
Slides for Chapter 4: Interprocess Communication



From **Coulouris, Dollimore and Kindberg**
Distributed Systems: Concepts and Design
Edition 4, © Addison-Wesley 2005

Middleware layers

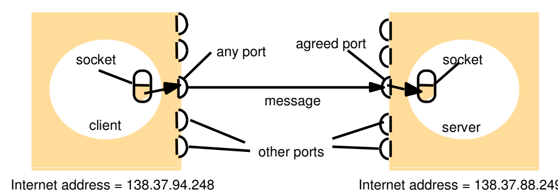


API for Internet Protocols (1): IPC characteristics

- synchronous and asynchronous communication
  - blocking send: waits until the corresponding receive is issued
  - non-blocking send: sends and moves on
  - blocking receive: waits until the msg is received
  - non-blocking receive: if the msg is not here, moves on
  - synchronous: blocking send and receive
  - asynchronous: non-blocking send and blocking or non-blocking receive
- Message Destination
  - IP address + port: one receiver, many senders
  - Location transparency
    - name server or binder: translate service to location
    - OS (e.g. Mach): provides location-independent identifier mapping to lower-lever addresses
  - send directly to processes (e.g. V System)
  - multicast to a group of processes (e.g. Chorous)
- Reliability
- Ordering

API for the Internet Protocols (2): Sockets and ports

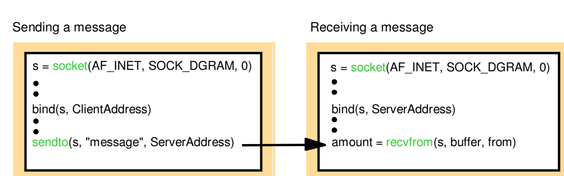
- programming abstraction for UDP/TCP
- originated from BSD UNIX



API for Internet Protocols (3): UDP Datagram

- message size: up to 2^{16}, usually restrict to 8K
- blocking: non-blocking send, blocking receive
- timeouts: timeout on blocking receive
- receive from any: doesn't specify sender origin (possible to specify a particular host for send and receive)
- failure model:
  - omission failures: can be dropped
  - ordering: can be out of order
- use of UDP
  - DNS
  - less overhead: no state information, extra messages, latency due to start up

API for Internet Protocols (4): C and UDP datagrams



ServerAddress and *ClientAddress* are socket addresses

API for Internet Protocols (5): Java and UDP

```
aSocket = new DatagramSocket();

…

InetAddress aHost = InetAddress.getByName(…);

…

DatagramPacket request = new
                   DatagramPacket(msg, length,
                   aHost, serverPort);
…
aSocket.send(request);

…
DatagramPacket reply = new
          DatagramPacket(buffer, length);


…
aSocket.receive(reply);
```

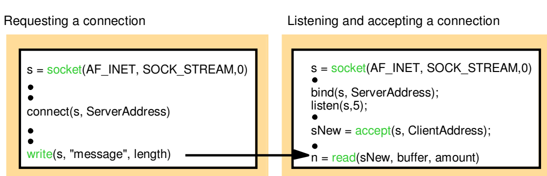
```
aSocket = new DatagramSocket(port);

…




DatagramPacket request = new
          DatagramPacket(buffer, length);

…
aSocket.receive(request);
…

DatagramPacket reply = new
          DatagramPacket(data, length,
                    request.getAddress(),
                    request.getPort());
…
aSocket.send(reply);
```


API for Internet Protocols (6): TCP stream

- message size: unlimited
- lost messages: sequence #, ack, retransmit after timeout of no ack
- flow control: sender can be slowed down or blocked by the receiver
- message duplication and ordering: sequence #
- message destination: establish a connection, one sender-one receiver, high overhead for short communication
- matching of data items: two processes need to agree on format and order (protocol)
- blocking: non-blocking send, blocking receive (send might be blocked due to flow control)
- concurrency: one receiver, multiple senders, one thread for each connection
- failure model
  - checksum to detect and reject corrupt packets
  - sequence # to deal with lost and out-of-order packets
  - connection broken if ack not received when timeout
    - could be traffic, could be lost ack, could be failed process..
    - can't tell if previous messages were received
- use of TCP: http, ftp, telnet, smtp

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

API for Internet Protocols (7): C and TCP streams



ServerAddress and *ClientAddress* are socket addresses

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

API for Internet Protocols (8): Java and TCP

```
Socket s = new Socket(host, serverPort);



…


DataInputStream in = new
     DataInputStream(s.getInputStream());
DataOutputStream out = new
     DataOutputStream(s.getOutputStream());

…

out.write(…);

…

in.read(…);
```

```
ServerSocket listenSocket = new
          ServerSocket(serverPort);

…
Socket s = listenSocket.accept();

…
DataInputStream in = new
     DataInputStream(s.getInputStream());
DataOutputStream out = new
     DataOutputStream(s.getOutputStream());

…

in.read(…);

…

out.write(…);
```

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

External Data Representation (1):

- different ways to represent int, float, char... (internally)
- byte ordering for integers
  - big-endian: most significant byte first
  - small-endian: least significant byte first
- standard external data representation
  - marshal before sending, unmarshal before receiving
- send in sender's format and indicates what format, receivers translate if necessary
- External data representation
  - SUN's External data representation (XDR)
  - CORBA's Common Data Representation (CDR)
  - Java's object serialization
  - ASCII (XML, HTTP)

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

External Data Representation (2): CDR

- Primitive types (15): short, long ...
  - support both big-endian and little-endian
  - transmitted in sender's ordering and the ordering is specified
  - receiver translates if needed
- Constructed types

Type	*Representation*
sequence	length (unsigned long) fol lowed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

External Data Representation (3):

- CORBA IDL compiler generates marshalling and unmarshalling routines
- Struct with string, string, unsigned long

index in sequence of bytes	← 4 bytes →	notes on representation
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20-23	"on__"	
24–27	1934	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1934}

External Data Representation (5): Java serialization

- serialization and de-serialization are automatic in arguments and return values of Remote Method Interface (RMI)
- flattened to be transmitted or stored on the disk
  - write class information, types and names of instance variables
  - new classes, recursively write class information, types, names...
  - each class has a handle, for subsequent references
  - values are in Universal Transfer Format (UTF)

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

External Data Representation (6): Java serialization

```
public class Person implements Serializable {
        private String name;
        private String place;
        private int year;

        public Person(String aName, String aPlace, int aYear){
           name = aName;
           place = aPlace;
           year = aYear;
        }
}
```

Serialized values				Explanation
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name:	java.lang.String place:	*number, type and name of instance variables*
1934	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers;
h0 and h1 are handles/references to other objects within the serialized form

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

External Data Representation (7)

- references to other objects
  - other objects are serialized
  - handles are references to objects in serialized form
  - each object is written only once
  - second or subsequent occurrence of the object is written as a handle
- reflection
  - ask the properties (name, types, methods) of a class
  - help serialization and de-serialization

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

External Data Representation (8): XML

- Extensible markup language (XML)
  - User-defined tags (vs. HTML has a fixed set of tags)
  - different applications agree on a different set of tags
  - E.g. SOAP for web services, tags are published
  - Tags are in plain text (not binary format)—not space efficient

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

External Data Representation (9)

- Person struct in XML
  - Tag names: `person, name, place, year`
  - Element: <name>Smith</name>
  - Attribute: `id="123456789"` of `person`
  - Binary data need to be converted to characters (base64)

```
<person id="123456789">
              <name>Smith</name>
              <place>London</place>
              <year>1934</year>
              <!-- a comment -->
</person >
```

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

External Data Representation (10): XML namespace

- Name clashes within an application
- Namespaces: a set of names for a collection of element types and attributes
- `xmlns`: xml namespace
- `pers`: name of the name space (used as a prefix)
- http://www.cdk4.net/person : location of schema

```
<person pers:id="123456789" xmlns:pers = "http://www.cdk4.net/person">
    <pers:name> Smith </pers:name>
    <pers:place> London </pers:place >
    <pers:year> 1934 </pers:year>
</person>
```


External Data Representation (11): XML schema

- Defines elements and attributes
- Similar to type definition
- xsd: namespace for xml schema definition

```
<xsd:schema  xmlns:xsd = URL of XML schema definitions >
   <xsd:element name= "person" type ="personType" />
      <xsd:complexType name="personType">
         <xsd:sequence>
            <xsd:element name = "name"  type="xs:string"/>
            <xsd:element name = "place"  type="xs:string"/>
            <xsd:element name = "year"  type="xs:positiveInteger"/>
         </xsd:sequence>
         <xsd:attribute name= "id"   type = "xs:positiveInteger"/>
      </xsd:complexType>
</xsd:schema>
```

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4 © Pearson Education 2005

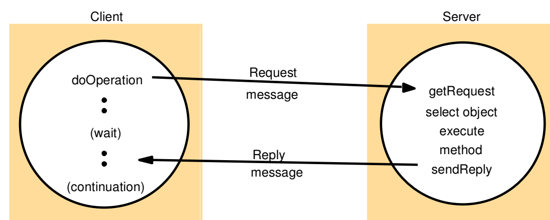
External Data Representation (12): Remote object reference

- call methods on a remote object (CORBA, Java)
  - unique reference in the distributed system
  - Reference = IP address + port + process creation time + local object # in a process + interface
  - Port + process creation time -> unique process
  - Address can be derived from the reference
  - Objects usually don't move; is there a problem if the remote object moves?
  - name of interface: what interface is available

32 bits	32 bits	32 bits	32 bits	
Internet address	port number	time	object number	interface of remote object

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4 © Pearson Education 2005

Client-server communication (1)



- Synchronous: client waits for a reply
- Asynchronous: client doesn't wait for a reply

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4 © Pearson Education 2005

Client-server communication (2): Request-reply message structure

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
objectReference	*RemoteObjectRef*
methodId	*int or Method*
arguments	*array of bytes*

Why requestID?

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4 © Pearson Education 2005

Client-server communication (3)

- Failure model
  - UDP: could be out of order, lost...
  - process can fail...
- not getting a reply
  - timeout and retry
- duplicate request messages on the server
  - How does the server find out?
- *idempotent* operation: can be performed repeatedly with the same effect as performing once.
  - idempotent examples?
  - non-idempotent examples?
- history of replies
  - retransmission without re-execution
  - how far back if we assume the client only makes one request at a time?

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4 © Pearson Education 2005

Client-server communication (4): RPC exchange protocols

Name	Messages sent by		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

Client-server communication (5)

- using TCP increase reliability and also cost
- HTTP uses TCP
  - one connection per request-reply
  - HTTP 1.1 uses "persistent connection"
    - multiple request-reply
    - closed by the server or client at any time
    - closed by the server after timeout on idle time
  - Marshal messages into ASCII text strings
  - resources are tagged with MIME (Multipurpose Internet Mail Extensions) types: test/plain, image/gif...
  - content-encoding specifies compression alg

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Client-server communication (6): HTTP methods

- GET: return the file, results of a cgi program, …
- HEAD: same as GET, but no data returned, modification time, size are returned
- POST: transmit data from client to the program at url
- PUT: store (replace) data at url
- DELETE: delete resource at url
- OPTIONS: server provides a list of valid methods
- TRACE: server sends back the request

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Client-server communication (6): HTTP request/reply format

method	*URL or pathname*	*HTTP version*	*headers*	*message body*
GET	//www.dcs.qmw.ac.uk/index.html	HTTP/ 1.1		

- Headers: latest modification time, acceptable content type, authorization credentials

HTTP version	*status code*	*reason*	*headers*	*message body*
HTTP/1.1	200	OK		resource data

- Headers: authentication challenge for the client

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Group communication (1)

- multicast
- useful for:
  - fault tolerance based on replicated services
    - requests multicast to servers, some may fail, the client will be served
  - discovering services
    - multicast to find out who has the services
  - better performance through replicated data
    - multicast updates
  - event notification
    - new items arrived, advertising services

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Group communication (2): IP multicast

- class D addresses, first four bits are 1110 in IPv4
- UDP
- Join a group via socket binding to the multicast address
- messages arriving on a host deliver them to all local sockets in the group
- multicast routers: route messages to out-going links that have members
- multicast address allocation
  - permanent
  - temporary:
    - no central registry by IP (one addr might have different groups)
      - use (time to live) TTL to limit the # of hops, hence distance
    - tools like sd (session directory) can help manage multicast addresses and find new ones

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 4
© Pearson Education 2005

Group communication (3): Reliability and ordering

- UDP-level reliability: missing, out-of-order...
- Effects on
  - fault tolerance based on replicated services
    - ordering of the requests might be important, servers can be inconsistent with one another
  - discovering services
    - not too problematic
  - better performance through replicated data
    - loss and out-of-order updates could yield inconsistent data, sometimes this may be tolerable
  - event notification
    - not too problematic