Chapter 4
Interprocess Communication

Gandeva Bayu Satrya, ST., MT.
Telematics Labz.
Informatics Department
Telkom University

Outline Today

Chapter 4 – Interprocess Communication

- API for IP
- Ext. Data Representation
- Multicast Communication
- Network Virtualization

Introduction

- Chapter 3 discussed the Internet transport-level protocols UDP and TCP without saying how middleware and application programs could use these protocols.
- The next section of this chapter introduces
  - ✓ the characteristics of interprocess communication and then discusses UDP and TCP from a programmer's point of view,
  - ✓ presenting the Java interface to each of these two protocols, together with a discussion of their failure models Figure 4.1 Middleware layers Applications,
  - ✓ services Middleware layers Underlying interprocess communication primitives: UDP and TCP.

1. API for IP

The general characteristics of interprocess communication and then discuss the Internet protocols as an example, explaining how programmers can use them, either by means of UDP messages or through TCP streams.

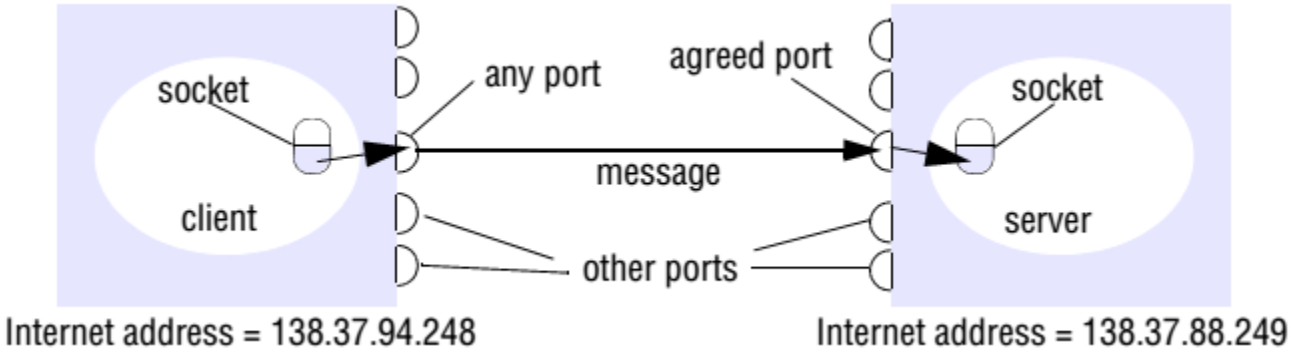
a) The characteristics of interprocess communication
b) Sockets
c) UDP datagram communication
d) TCP stream communication

a. Characteristics of Inter.Comm.

Message passing between a pair of processes can be supported by two message communication operations, send and receive, defined in terms of destinations and messages.

- Synchronous and asynchronous: block and non-block.
- Message destinations : Internet address and local port.
- Reliability : validity and integrity
- Ordering : sender order

b. Sockets



- Both forms of communication (UDP and TCP) use the socket abstraction, which provides an end point for communication between processes.
- Java API for Internet addresses

```
InetAddress aComputer = InetAddress.getByName("www.telkomuniversity.ac.id");
```

c. UDP Datagram Comm.

a sending process to a receiving process without acknowledgement or retries

- ✓ Failure model for UDP datagrams : checksum error or because no buffer space
- ✓ Use of UDP : DNS and VoIP
- ✓ Java API for UDP datagrams :

Datagram packet

array of bytes containing message	length of message	Internet address	port number

Ex : UDP client sends a message

```
import java.net.*;
import java.io.*;
public class UDPClient{
   public static void main(String args[]){
     // args give message contents and server hostname
     DatagramSocket aSocket = null;
     try {
          aSocket = new DatagramSocket();
          byte [] m = args[0].getBytes();
          InetAddress aHost = InetAddress.getByName(args[1]);
          int serverPort = 6789;
          DatagramPacket request =
                new DatagramPacket(m,  m.length(), aHost, serverPort);
          aSocket.send(request);
          byte[] buffer = new byte[1000];
          DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
          aSocket.receive(reply);
          System.out.println("Reply: " + new String(reply.getData()));
      } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
      } catch (IOException e){System.out.println("IO: " + e.getMessage());
      } finally { if(aSocket != null) aSocket.close();}
   }
}
```

d. TCP Stream Comm.

Stream communication assumes that when a pair of processes are establishing a connection, one of them plays the client role and the other plays the server role, but thereafter they could be peers.

- ✓ Failure model : use checksums to detect and reject corrupt packets and sequence numbers to detect and reject duplicate packets.
- ✓ Use of TCP : HTTP, FTP, and SSH
- ✓ Java API for TCP streams

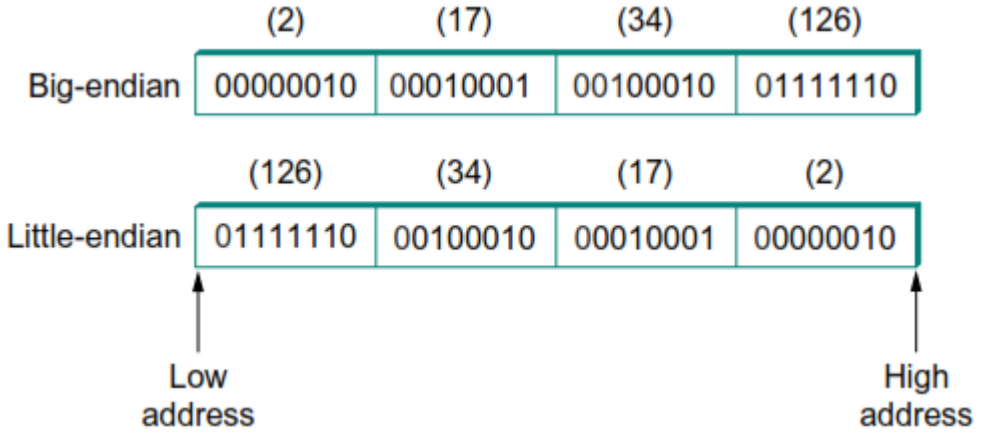
Ex : TCP client makes a conn to Server

```
import java.net.*;
import java.io.*;
public class TCPClient {
    public static void main (String args[]) {
        // arguments supply message and hostname of destination
        Socket s = null;
        try{
            int serverPort = 7896;
            s = new Socket(args[1], serverPort);
            DataInputStream in = new DataInputStream( s.getInputStream());
            DataOutputStream out =
                    new DataOutputStream( s.getOutputStream());
            out.writeUTF(args[0]);      // UTF is a string encoding; see Sec 4.3
            String data = in.readUTF();
            System.out.println("Received: "+ data) ;
        }catch (UnknownHostException e){
            System.out.println("Sock:"+e.getMessage());
        } catch (EOFException e){System.out.println("EOF:"+e.getMessage());
        } catch (IOException e){System.out.println("IO:"+e.getMessage());
        } finally {if(s!=null) try {s.close();}catch (IOException e){/*close failed*/}}
    }
}
```

2. External Data Representation [PET]

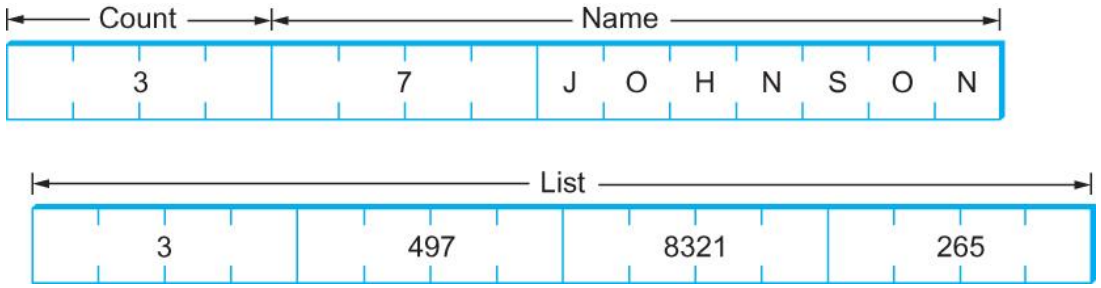
Marshalling

Unmarshalling

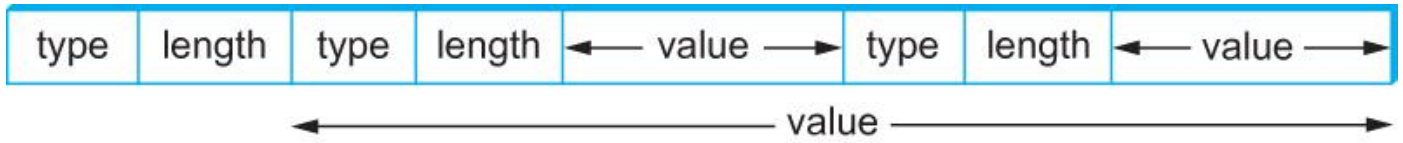


Ex : Presentation Formating [PET]

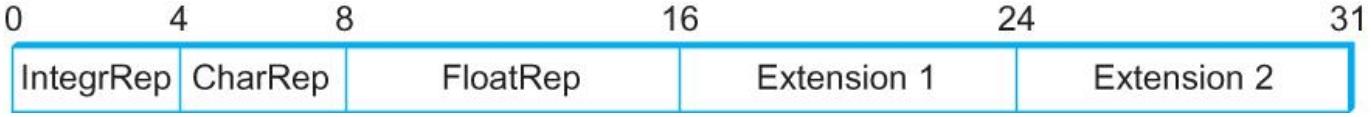
XDR



ASN.1 BER



NDR



2. External Data Representation (con't)

Three alternative approaches to external data representation and marshalling ***(Google's Approaches)*** :

a) CORBA's common data representation, which is concerned with an external representation for the structured and primitive types
b) Java's object serialization, which is concerned with the flattening and external data representation of any single object or tree of objects
c) XML (Extensible Markup Language), which defines a textual format for representing structured data.

a. CORBA

- CORBA's Common Data Representation (CDR)
- These consist of 15 primitive types, which include short (16-bit), long (32-bit), unsigned short, unsigned long, float (32-bit), double (64-bit), char, boolean (TRUE, FALSE), octet (8-bit), and any (which can represent any basic or constructed type)
- Marshalling in CORBA

```
struct Person {
        string name;
        string place;
        unsigned long year;
};
```

Ex: CORBA CDR Message

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	1984	*unsigned long*

Contains the three fields of a struct whose respective types are string, string and unsigned long

b. Java Object Serialization

- In Java RMI, both objects and primitive data values may be passed as arguments and results of method invocations.
- An object is an instance of a Java class

```
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;
        }
        // followed by methods for accessing the instance variables
}
```

Ex : Java Serialized Form

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*
1984	5 Smith	6 London	h1	*values of instance variables*

As an example, consider the serialization of the following object:

Person p = new Person("Smith", "London", 1984);

c. Extensible Markup Language (XML)

- XML is a markup language that was defined by the World Wide Web Consortium (W3C) for general use on the Web.
- Both XML and HTML were derived from SGML (Standardized Generalized Markup Language) [ISO 8879]

- XML data items are tagged with 'markup' strings.

```
<person id="123456789">
<name>Smith</name>
<place>London</place>
<year>1984</year>
<!-- a comment -->
</person >
```

3. Multicast Communication

- A multicast operation is more appropriate – this is an operation that sends a single message from one process to each of the members of a group of processes, usually in such a way that the membership of the group is transparent to the sender.

- Multicast messages characteristics:
  1) Fault tolerance based on replicated services: a group of servers
  2) Discovering services in spontaneous networking: to locate available services
  3) Better performance through replicated data: managing the replicas
  4) Propagation of event notifications: be used to notify processes

a. IP Multicast

- **IP multicast** is built on top of the Internet Protocol (IP).
- A multicast group is specified by a **Class D Internet** address (the range 224.0.0.0 to 239.255.255.255).

Address	*Assignment*
224.0.0.0	Base address (reserved)
224.0.0.1	All systems (hosts or routers) on this network
224.0.0.2	All routers on this network
224.0.0.4	DMVRP routers
224.0.0.5	OSPF routers
224.0.0.7	ST (stream) routers
224.0.0.8	ST (stream) hosts
224.0.0.9	RIP2 routers
224.0.0.10	IGRP routers
224.0.0.11	Mobile Agents
224.0.0.12	DHCP servers
224.0.0.13	PIM routers
224.0.0.14	RSVP encapsulation
224.0.0.15	CBT routers
224.0.0.22	IGMPv3

b. Reliability and Ordering

- Some applications require a multicast protocol that is more reliable than IP multicast.
- There is a need for reliable multicast, in which any message transmitted is either received by all members of a group or by none of them.
- The examples also suggest that some applications have strong requirements for ordering, the strictest of which is called totally ordered multicast.

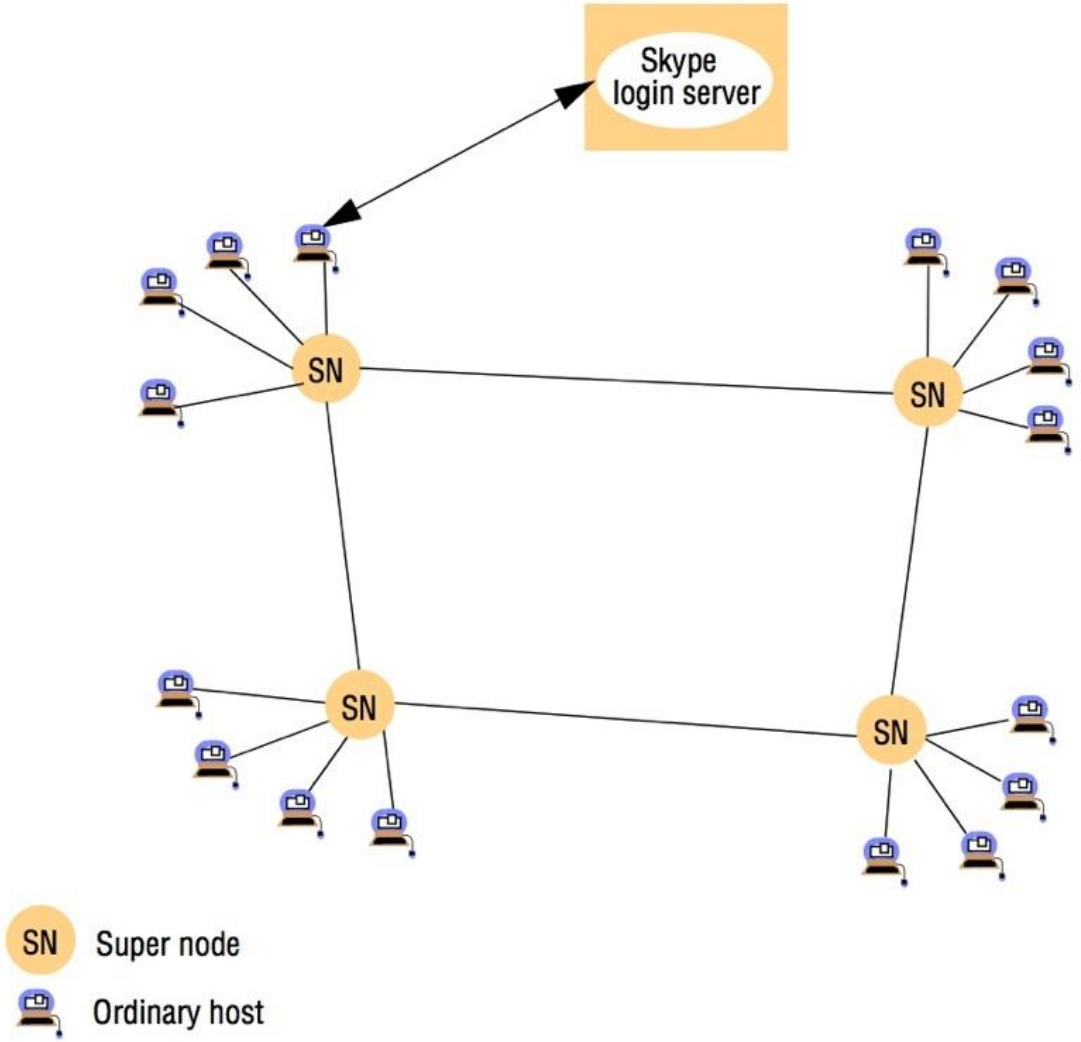
4. Network Virtualization

- Network virtualization is concerned with the construction of many different virtual networks over an existing network such as the Internet.
- Each virtual network can be designed to support a particular distributed application.
- Each virtual network has its own particular addressing scheme, protocols and routing algorithms, but redefined to meet the needs of particular application classes.

Types of Overlay

Motivation	*Type*	*Description*
Tailored for application needs	Distributed hash tables	One of the most prominent classes of overlay network, offering a service that manages a mapping from keys to values across a potentially large number of nodes in a completely decentralized manner (similar to a standard hash table but in a networked environment).
	Peer-to-peer file sharing	Overlay structures that focus on constructing tailored addressing and routing mechanisms to support the cooperative discovery and use (for example, download) of files.
	Content distribution networks	Overlays that subsume a range of replication, caching and placement strategies to provide improved performance in terms of content delivery to web users; used for web acceleration and to offer the required real-time performance for video streaming [www.kontiki.com].
Tailored for network style	Wireless ad hoc networks	Network overlays that provide customized routing protocols for wireless ad hoc networks, including proactive schemes that effectively construct a routing topology on top of the underlying nodes and reactive schemes that establish routes on demand typically supported by flooding.
	Disruption-tolerant networks	Overlays designed to operate in hostile environments that suffer significant node or link failure and potentially high delays.
Offering additional features	Multicast	One of the earliest uses of overlay networks in the Internet, providing access to multicast services where multicast routers are not available; builds on the work by Van Jacobsen, Deering and Casner with their implementation of the MBone (or Multicast Backbone) [mbone].
	Resilience	Overlay networks that seek an order of magnitude improvement in robustness and availability of Internet paths [nms.csail.mit.edu].
	Security	Overlay networks that offer enhanced security over the underling IP network, including virtual private networks, for example, as discussed in Section 3.4.8.

Skype Overlay Architecture



References

[COU'12] Coulouris, G. Dollimore, J., Kindberg, T., Blair, G., DISTRIBUTED SYSTEMS :Concepts and Design Fifth Edition, Pearson Education, Inc., United States of America, 2012.

[FOR'07] Forouzan, B.A., Data Communications and Networking, Fourth Edition, McGraw-Hill, New York, 2007.

[TAN'07] Tanenbaum, A.S., Steen, M.V., DISTRIBUTED SYSTEMS : Principles and Paradigms Second Edition, Pearson Education, Inc., United States of America, 2007.

[PET'12] Peterson, L.L., and Davie, B.S., Computer Networks: A Systems Approach Fifth Edition, Morgan Kaufmann, Burlington USA, 2012.

Thank You

Gandeva Bayu Satrya, ST., MT.
Telematics Labz.
Informatics Department
Telkom University