Interprocess Communication

1. Point-to-point Communication
   • Characteristics of Interprocess Communication
   • Sockets
   • Client-Server Communication over UDP and TCP
2. Group (Multicast) Communication
• The architecture of a system is its structure in terms of separately specified components and their interrelationships.

• 4 fundamental building blocks (and 4 key questions):
  ‣ Communicating entities: what are the entities that are communicating in the distributed system?
  ‣ Communication paradigms: how do these entities communicate, or, more specifically, what communication paradigm is used?
  ‣ Roles and responsibilities: what (potentially changing) roles and responsibilities do these entities have in the overall architecture?
  ‣ Placement: how are these entities mapped on to the physical distributed infrastructure (i.e., what is their placement)?
Communication Paradigms

• 3 types:
  - direct communication
  - interprocess communication
    low level support for communication between processes in the distributed system, including message-passing primitives, socket programming, multicast communication
  - remote invocation
    most common communication paradigm, based on a two-way exchange between communicating entities and resulting in the calling of a remote operation (procedure or method)
  - indirect communication
    communication is indirect, through a third entity, allowing a strong degree of decoupling between senders and receivers.
    Examples: publish subscribe systems, distributed shared memory (DSM).
Multicast

• A multicast operation 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.

• There is a range of possibilities in the desired behaviour of a multicast.

• The simplest provides no guarantees about message delivery or ordering (see lecture in week 12 on “Multicast Communication”)

DTU Compute
Department of Applied Mathematics and Computer Science
What Can Multicast Be Useful for?

- Multicast messages provides a useful infrastructure for constructing distributed systems with the following characteristics:

  1. **Fault tolerance based on replicated services**

     - A replicated service consists of a group of members
     - Client requests are multicast to all the members of the group, each of which performs an identical operation
     - Even when some of the members fail, clients can still be served
What Can Multicast Be Useful for?

• Multicast messages provides a useful infrastructure for constructing distributed systems with the following characteristics:

2. Better performance through replicated data

  ▸ Data are replicated to increase the performance of a service - in some cases replicas of the data are placed in users’ computers
  ▸ Each time the data changes, the new value is multicast to the processes managing the replicas
What Can Multicast Be Useful for?

Multicast messages provides a useful infrastructure for constructing distributed systems with the following characteristics:

3. **Propagation of event notifications**

- Multicast to a group may be used to notify processes when something happens

- For example, a *news system* might notify interested users when a new message has been posted on a particular newsgroup
IP Multicast

• IP multicast is built on top of the Internet Protocol, IP

• Note that IP packets are addressed to computers (ports belong to the TCP and UDP levels)

• IP multicast allows the sender to transmit a single IP packet to a set of computers that form a multicast group

• The sender is unaware of the identities of the individual recipients and of the size of the group

• A multicast group is specified by an Internet address whose first 4 bits are 1110 (in IPv4)
IP Multicast - Membership

• Being a member of a multicast group allows a computer to receive IP packets sent to the group

• It is possible to send datagrams to a multicast group without being a member

• The membership of multicast groups is dynamic, allowing computers to join or leave at any time and to join an arbitrary number of groups
IP Multicast - IP Level

• At the IP level:

  ▸ A computer belongs to a multicast group when one or more of its processes has sockets that belong to that group

  ▸ When a multicast message arrives at a device:

    copies are forwarded to all of the local sockets that have joined the specified multicast address and are bound to the specified port number
IP Multicast - Programming Level

• At the application programming level, IP multicast is available only via UDP:
  
  ‣ An application program performs multicasts by sending UDP datagrams with multicast addresses and ordinary port numbers

  ‣ An application program can join a multicast group by making its socket join the group, enabling it to receive messages to the group
Case Study: JAVA API for IP Multicast

The Java API provides a datagram interface to IP multicast through the class `MulticastSocket`
The Class MulticastSocket

• A subclass of `DatagramSocket` with the additional capability of being able to join multicast groups

• It provides **two alternative constructors**, allowing sockets to be created to use either a **specified local port** or any **free local port**

```java
... MulticastSocket s =null;
s = new MulticastSocket(6789);
...```
Joining a Group

• A process can **join** a group with a given multicast address by invoking the `joinGroup` method of its multicast socket

  ‣ In this way, the socket joins a multicast group *at a given port* and it will receive datagrams sent by processes on other computers *to that group at that port*

```java
...  
MulticastSocket s =null;  
s = new MulticastSocket(6789);  
InetAddress group = InetAddress.getByName(args[1]);  
s.joinGroup(group);  
...  
```
Leaving a Group

• A process can **leave** a specified group by invoking the **leaveGroup** method of its multicast socket

```java
MulticastSocket s = null;
InetAddress group = InetAddress.getByName(args[1]);
s = new MulticastSocket(6789);
...
s.leaveGroup(group);
```
Example:
Multicast Peer Joins a Group and Sends and Receives Datagrams

```java
import java.net.*;
import java.io.*;
public class MulticastPeer{
    public static void main(String args[]){
        // args give message contents & destination multicast group (e.g. "228.5.6.7")
        MulticastSocket s =null;
        try{
            InetAddress group = InetAddress.getByName(args[1]);
            s = new MulticastSocket(6789);
            s.joinGroup(group);
            byte [] m = args[0].getBytes();
            DatagramPacket messageOut =
                new DatagramPacket(m, m.length, group, 6789);
            s.send(messageOut);
        }
    }
}
```

// this figure continued on the next slide
Example:
Multicast Peer Joins a Group and Sends and Receives Datagrams

```java
// get messages from others in group
byte[] buffer = new byte[1000];
for(int i=0; i<3; i++) {
    DatagramPacket messageIn =
        new DatagramPacket(buffer, buffer.length);
    s.receive(messageIn);
    System.out.println("Received:" + new String(messageIn.getData()));
}

s.leaveGroup(group);
```

peer attempts to receive 3 multicast messages from its peers via its socket

```java
} catch (SocketException e){System.out.println("Socket: " + e.getMessage());
} catch (IOException e){System.out.println("IO: " + e.getMessage());}
} finally {if(s != null) s.close();}
```
Example:
Multicast Peer Joins a Group and Sends and Receives Datagrams

- When several instances of this program are run simultaneously on different computers, all of them join the same group and each of them should receive its own message and the messages from that joined after it.
End of the Case Study