Title: Distributed_Applications (28.04.2014)
Date: Mon Apr 28 09:16:53 CEST 2014
Duration: 45:29 min
Pages: 14

Interprocess communication (IPC): message exchange between sender and receiver.

- send operation
  - blocking
  - non-blocking

- receive operation
  - blocking
  - non-blocking

Background

Categories of Message Exchange
Asynchronous message exchange (nonblocking)

Synchronous message exchange (blocking)

Remote-invocation send

Sender S suspends execution until the receiver has received and processed the submitted request that was delivered as part of the message.

Advantages of asynchronous message exchange

Advantages

- useful for real-time applications, especially if the sending process should not be blocked.
- supports parallel execution threads at the sender’s and the receiver’s sites.
- it can be used for event signaling purposes.

Disadvantages

- management of message buffers, handling of buffer overflow, access control problems, and of process crashes (receiver).
- notification of S in case of failures may be a problem, since mostly S has already continued with its regular processing.
- design of a correct system is difficult. The failure behavior depends heavily on buffer sizes, buffer contents, and the time behavior of the exchanged messages.

Asynchronous message exchange (nonblocking)

Sender S can resume its processing immediately after the message N is put forward into the message queue NP (NP is also called message buffer).

- S will not wait until the receiver E has received the message N.
- A receive operation indicates that the receiver is interested in receiving a message.

Example

Advantages of asynchronous message exchange
Asynchronous message exchange (nonblocking)
Synchronous message exchange (blocking)

Remote-invocation send
Sender S suspends execution until the receiver has received and processed the submitted request that was delivered as part of the message.

Names are used to uniquely identify entities and refer to locations. An important issue is name resolution.

Names
A name is a string of characters that is used to refer to an entity (e.g., host, printer, file).

Names have access points to invoke operations on them ⇒ address is the name of the access point.

An identifier is a name which uniquely identifies an entity.

Name space
Names in distributed systems are organized into a name space.

Name spaces are organized hierarchically.

Representation as a labeled directed graph.
Path along graph edges specifies the entity name, e.g., documents/projects/lecture2003/concept.txt.

Absolute vs relative path names.

Name resolution: a name lookup returns the identifier or the address of an entity, e.g., LDAP Name Service.

Paradigms for distributed applications

Information Sharing
Message exchange
Naming entities
Bidirectional communication
Producer-consumer interaction
Client-server model
Peer-to-peer model
Group model
Publish-Subscribe model

Taxonomy of communication
Message serialization
Levels of Abstraction

Sockets
Sockets provide a low level abstraction for programming bidirectional communication.

A socket is an application created, OS-controlled interface into which application can both send and receive messages to/from another application.

Unique identification: IP-address and port number.

Sockets in Java
Java package java.net
Socket constructors - methods
Socket constructors - methods

- `Socket()`: Creates an unconnected socket, with the system-default type of `SocketImpl`.
- `Socket(InetAddress address, int port)`: Creates a stream socket and connects it to the specified port number at the specified IP address.
- `Socket(Protocol protocol)`: Creates an unconnected socket, using the specified protocol.
- `Socket(InetAddress address, int port)`: Creates a stream socket and connects it to the specified port number on the named host.

Methods of `java.net.Socket`:

- `void bind(InetAddress address, int port)`: Binds the socket to a local address.
- `void close()`: Closes this socket.
- `void connect(InetAddress address, int port)`: Connects this socket to the server.
- `void connect(InetAddress address, int port, int timeout)`: Connects this socket to the server with a specified timeout value.

Example from the client perspective

```java
import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            echoSocket = new Socket("www.in.tum.de", 7); // create Socket
            // create Writer, Reader
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(
                new InputStreamReader(echoSocket.getInputStream()));
        }
        catch (UnknownHostException e) {
            System.err.println("unknown host in.tum.de");
            System.exit(1);
        }
        catch (IOException e) {
            System.err.println("No I/O from in.tum.de");
            System.exit(1);
        }
        // read streams
        BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
        String userInput;
        while ((userInput = stdin.readLine()) != null) {
            out.println(userInput);
            System.out.println("echo: "+ in.readLine());
        }
        // close streams and sockets
        out.close();
        in.close();
        stdin.close();
    }
}
```