Interprocess communication of distributed systems
Layered protocols

- ISO OSI Reference model
- TCP/IP protocol stack
- Protocols
  - Connection-oriented (connection establishment, negotiation of protocol used)
  - Connectionless
TCP for Transactions (T/TCP) - RFC 1664

- is a variant of the TCP protocol
- TCP extension for efficient transaction-oriented (request/response) service
- faster than TCP and delivery reliability is comparable to that of TCP
TCP for Transactions cont.

TCP

1. SYN
2. SYN, ACK (SYN)
3. ACK (SYN)
4. request
5. FIN
6. ACK (req+FIN)
7. answer
8. FIN
9. ACK (FIN)

T/TCP

1. SYN, request, FIN
2. SYN, ACK (FIN), answer, FIN
3. ACK (FIN)

Time
Middleware protocols

- Middleware logically lives in application layer
- Missing clear distinction between applications, application-specific protocols and general-purpose protocols (FTP, HTTP)
- Authentication and authorization protocols
- Distributed commit protocols ~ atomicity
Remote Procedure Call

- send(), receive() do not conceal communication -> no access transparency
- Birrell and Nelson (1984)

Conventional Procedure Call
  - Call-by-value
  - Call-by-reference
  - Call-by-copy/restore (not common, Ada, PL/SQL)
RPC cont.

- Client and Server Stubs
  - Idea behind is to make a RPC look as possible like a local one
- Client stub – in library
- Server stub – receive message from client
RPC steps

1. The client procedure calls the client stub in normal way (parameters copied on stack)
2. The client stub builds a message and calls the local operating system
3. The client’s OS sends the message to the remote OS
4. The remote OS gives the message to the server stub
5. The server stub unpacks the parameters and calls the server
6. The server does the work and returns the result to the stub
7. The server stub packs it in a message and calls its local OS
8. The server’s OS sends the message to the client’s OS
9. The client’s OS gives the message to the client stub
10. The stub unpacks the result and returns to the client
Parameter Passing
- Parameter marshaling – packing parameters into message

Passing Reference Parameters
- Often call-by-reference replaced by copy/restore
- Optimized for input/output parameters – copy occurred only once
RPC cont.

- Parameter Specification and Stub Generation
  - Message format
  - IDL (Interface Definition Language) – simplifies development

- RPC extensions
  - LRPC (lightweight) aka Doors - support in OS
  - Asynchronous RPC
RPC cont.

Doors

Client process

```
main()
{
    ...
    fd = open(door_name, ...);
    door_call(fd, ...);
    ...
}
```

Register door

Server process

```
server_door(...) {
    ...
    door_return(...);
}
main()
{
    ...
    fd = door_create(...);
    fattach(fd, door_name, ...);
    ...
}
```

Operating system

Invoke registered door at other process

Return to calling process
Asynchronous RPC

RPC cont.
DCE RPC

- Distributed Computing Environment by OSF
- Several services
  - Distributed file service
  - Directory service
  - Security service
  - Distributed time service
- Client/server model
- Setup communication between client and server (binding)
Client server binding in DCE

1. Register endpoint
2. Register service
3. Look up server
4. Ask for endpoint
5. Do RPC
Remote Method Invocation (RMI)

- Object encapsulates data (state) and operations (methods)
- Compile versus Runtime Objects
  - Object adapter – wrapper
- Persistent and Transient Objects
- Binding Client to an Object
  - Implicit binding
  - Explicit binding
Remote object with client-side proxy

Client machine

Client invokes a method

Proxy

Same interface as object

Server machine

Server

Object

State

Method

Skeleton

Interface

Skeleton invokes same method at object

Network

Marshalled invocation is passed across network
RMI cont.

- Static
  - Interface changes -> client recompile
  - Example: fobject.append(int)

- Dynamic
  - invoke(object, method, input_parameters, output_parameters);
  - Example: invoke(fobject, id(append), int)
RMI Parameter Passing
DCE Remote Objects

- Two types of distr. objects supported
  - Distributed dynamic object (a)
  - Distributed named object (b)
Java RMI

- Distributed objects integrated into language
- Each object can be constructed as a monitor by declaring a method to be synchronized
- Any primitive or object type can be passed as parameter to an RMI (type marshaled), **serializable**
Message-oriented communication

- Persistent or transient communication
- Asynchronous or synchronous
Figure 2-22. Six different forms of communication: (a) persistent asynchronous communication, (b) persistent synchronous communication, (c) transient asynchronous communication, (d) receipt-based transient synchronous communication, (e) delivery-based transient synchronous communication at message delivery, and (f) response-based transient synchronous communication.

- **Message-oriented Transient Communication**
  - Berkeley sockets
  - Message-passing interface (MPI)
    - (groupID,processID) identifies source or destination of message, blocking and non-blocking send/receive

- **Message-oriented Persistent Communication**
  - Message-queuing systems, message-oriented middleware (MOM), primitives Put,Get,Poll,Notify
  - Source and destination queues with queue managers, Message brokers
Stream-oriented communication

- Data streams
- Synchronous and asynchronous transmission modes, isochronous transmission mode