Inter-process communication in distributed systems

- **Communication in the Internet**
  - Low-level message exchange
  - No shared memory (procedure call is not available)
  - No distributed transparency (communication is visible)
  - We need high level communication primitives to easily construct large-scale distributed systems

- **Communication model in distributed systems**
  - Remote Procedure Call
  - Message-Oriented Communication
  - Stream-Oriented Communication

Layered protocols: outline

- Processes communicate using protocol stack called **TCP/IP**
- **Each application defines its own application protocol** to provide a specific service using primitives of TCP/IP
- **Protocols are layered** so that they are applicable to various applications and access networks
Layered protocols

- **Seven-layer ISO OSI reference model**
  - Model of layered protocols that allow different open systems to communicate with each other based on standard rules
  - ISO (International Organization for Standardization) developed it
- **Internet protocol stack**
  - Five layers, de facto standard

### OSI reference model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Application programs (http, ftp, smtp, pop, ssh, rpc, etc)</td>
</tr>
<tr>
<td>Presentation</td>
<td>Operating systems</td>
</tr>
<tr>
<td>Session</td>
<td>Device drivers</td>
</tr>
<tr>
<td>Transport</td>
<td>Network interfaces</td>
</tr>
<tr>
<td>Data link</td>
<td>Physical</td>
</tr>
</tbody>
</table>

### Internet protocol stack

1. **Application**
   - Presents protocols
2. **Presentation**
   - Presents services
3. **Session**
   - Establishes conversations
4. **Transport**
   - Provides byte streaming
5. **Network**
   - Routes packets
6. **Data link**
   - Transmits frames
7. **Physical**
   - Sending bits

#### Lower-level protocols

- **Physical layer**
  - Just sends bits (0s and 1s)
  - Defines how many bolts to use for 0 and 1, how many bits per second can be sent, etc
- **Data link layer**
  - Groups the bits into units called frames by putting a special bit pattern on start and end of each frame
  - Makes error control by putting checksum and sequence number to each frame
- **Network layer**
  - Achieves end-to-end communication
  - Routes packets by choosing the best path to the destination

#### Typical organization of network

- **LAN**
  - FDDI/GbE/10Gbe
  - Router
  - Ethernet
  - Router
  - Fiber optics
- **Core network of Internet**
  - How to realize communication between end systems through different types of physical networks

#### Communication between end systems - horizontal view -

1. When host A sends a packet to host B

   - Physical and data link layer communication (1 hop)

   - Network layer communication
     (end to end communication through multiple data link networks)
Communication between end systems - vertical view -

- Data is sent/received vertically between layers

  Data to send (ex. E-mail text)

  Each layer appends control info (header, trailer) to data (payload)

Example of header/trailer info

Data to send

Host A

Host B

Application

Transport

Network

Data link

Physical

Application

Transport

Network

Data link

Physical

Example of header/trailer info

Higher-level protocols (1)

- Transport protocols
  - allow end systems to exchange data (message, stream)
  - TCP: reliable connection-oriented communication
  - UDP: connection-less communication

- Session and presentation protocols
  - Session layer provides dialog control, check-pointing, synchronization functions
  - Presentation layer can define a record with fields such as names and addresses
  - facilitate communication between machines with different internal data representations

  → These layers are important but rarely used

Higher-level protocols (2)

- Application protocols
  - Various protocols for providing various services
  - Ex. SMTP, HTTP, FTP, Telnet, SSH, NFS, etc
  - All distributed systems (DSs) are applications

  ➢ Need for general-purpose protocols useful to many distributed systems
  ➢ middleware layer protocols
Middleware protocols

- Provide applications with common useful services that are independent of application-specific functions
  - **High-level communication primitives**: remote procedure call, persistent communication, streaming, reliable multicasting, etc
  - **Other services**: authentication, two-phase commit, distributed lock, session and presentation layer functions, etc

**Types of communication**

- **Transient communication**: Sender & receiver can communicate only when both are active
- **Persistent communication**: Sender & receiver can communicate regardless of their states
- **Synchronous communication**: Sender is blocked until receiving a reply from receiver
- **Asynchronous communication**: Sender can continue immediately after sending message

**Middleware communication services**

1. **Remote Procedure Call**
2. **Message-Oriented Communication**
3. **Stream-Oriented Communication**

Q: What type of communication is used in e-mail and www?
(1) Remote Procedure Call

- **Background**
  - Many distributed systems have been based on explicit message exchange between processes.
  - Procedures “send” and “receive” do not hide communication at all (no access transparency).

- **Remote Procedure Call, RPC** (Birrell and Nelson, 1984)
  - A mechanism to allow programs to call procedures located on other machines.
  - Process on machine A calls a procedure on machine B.
  - Procedure is executed on B.
  - Parameters and results are passed between A and B.
  - No message passing is visible to the programmer.

---

**Basic RPC operation**

- **RPC Goal:** make a remote procedure call look like a local procedure call.

---

**Parameter passing in RPC**

- **Client and server stubs**
  - **Client stub** packs parameters into a message (marshaling).
  - **Server stub** unpacks message to extract the result (unmarshaling).

---

**Passing value parameters**

- **Different architecture computers may have different data representation**
  - Character code: ASCII, JIS, Unicode, etc.
  - Integer, float numbers: big endian, little endian.

- **Ex. 16-bit Unicode for characters, IEEE #754 for float numbers with little endian, etc.**

---

**Original message on Intel processors (little-endian):**

- Integer “5”, String “WINE”

**Message received on SPARC (big-endian):**

- Integer “83,886,080”, String “WINE”

**Result will be wrong if simply inverting each 32 bit-word on SPARC.**
Passing reference parameters

- **How to pass pointers (references)?**
  - Client A calls, by RPC, procedures Open() and Read() in server B
  - `name`, `buf` are pointers and it makes no sense to pass them as is

  ```c
  // RPC client A
  void main()
  {
      char buf[100], *name;
      int fd;
      name = "important.dat";
      fd = Open(name, O_RDONLY);
      count = Read(fd, buf, 100);
      ...
  }
  ``

- **Solutions**
  - Copy content of the array (name or buf) into message and send it to server so that the server can read and change the array
  - When server finishes, it marshals the array into message and sends it back to client (optimization: no send back unchanged array)

Generating client/server stubs

- **IDL (Interface Definition Language)**
  - Language to specify interfaces (types of parameters and return value) of procedures
  - IDL compiler automatically generates code for client and server stubs
  - Procedure itself can be implemented in any other languages
  - Java, XML, and other languages can be used as IDL

RPC-based middleware: example

- **DCE: Distributed Computing Environment**
  - Its specifications have been adopted in Microsoft DCOM
  - Generate interface ID used for RPC

Using RPC in DCE

- **Binding a client to a server**
  1: Server process is located on a machine. DCE daemon maintains an end point table
  2: Server process registers with the directory service for lookup
  3-4: Client looks up the server machine and obtains end point from DCE daemon before executing RPC
Asynchronous RPC

- In asynchronous RPC,
  - Server immediately sends a reply back to the client when it receives the RPC request
  - Client can continue processing without further blocking

Advantage/disadvantage of RPC

- **Advantage**
  - Realize access transparency in distributed systems
  - Easy to program

- **Disadvantage**
  - Requires the server process to be active (transient communication)
  - Client is blocked until the server finishes processing (in the case of synchronous communication)

- **Systems in which RPC is not suitable**
  - Some applications may need another communication means, e.g., want to send a request even when server is not active

Message-oriented communication: example (1)

- **Berkeley socket**
  - Used for network programming in C on UNIX machines
  - Based on asynchronous communication

(2) Message-oriented communication

- **Characteristics**
  - Sending process and receiving process can exchange *messages* in various forms

- **Types of message-oriented communication**
  - **Message-oriented transient communication**
    - Sending process and receiving process can exchange messages only when they are active
    - Ex. Communication offered by standard transport protocol, e.g., TCP
  - **Message-oriented persistent communication**
    - Sending process and receiving process can exchange messages, even when one or both of them are not active
Message-oriented transient communication: example (2)

- **MPI**: Message Passing Interface
  - Defined for hardware/platform independent message passing
  - Designed for parallel applications (running in cluster systems)

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_bsend</td>
<td>Append outgoing message to a local send buffer</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Send a message and wait until copied to local or remote buffer</td>
</tr>
<tr>
<td>MPI_ssend</td>
<td>Send a message and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_sendrecv</td>
<td>Pass reference to outgoing message, and continue</td>
</tr>
<tr>
<td>MPI_isend</td>
<td>Pass reference to outgoing message, and wait for reply</td>
</tr>
<tr>
<td>MPI_recv</td>
<td>Receive a message; block if there are none</td>
</tr>
<tr>
<td>MPI_irecv</td>
<td>Check if there is an incoming message, but do not block</td>
</tr>
</tbody>
</table>

MPI supports both synchronous and asynchronous communications

Message-oriented persistent communication

**Message-queuing systems**/Message-Oriented Middleware (MOM)
- Provide extensive support for asynchronous persistent communication
- Offer intermediate-term storage for messages
- Ex. The Internet e-mail systems

Message-queuing system: architecture (1)

- **Inter-queue communication**
  - Messages can be put only into local queues on the sender machine or a machine on the same LAN (called source queue)
  - Messages can be read only from local queues
  - **Destination queue** is specified in a message → message queuing system is responsible to transfer messages to their destinations

Basic interface in a message queuing system

<table>
<thead>
<tr>
<th>primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>put</td>
<td>Append a message to a specific queue</td>
</tr>
<tr>
<td>get</td>
<td>Get and remove the first message in the specified queue, or block while the queue is empty</td>
</tr>
<tr>
<td>poll</td>
<td>Check a specified queue and get/remove the first message if the queue is nonempty (never block)</td>
</tr>
<tr>
<td>notify</td>
<td>Install a handler to be called when a message is put into the specified queue</td>
</tr>
</tbody>
</table>

Queue managers: interact with applications that send/receive messages

Relays: queue managers that operate as “Routers”
Message-queuing system: architecture (2)

Overlay network of a message-queuing system

Example of message-queuing system: IBM WebSphere MQ

(3) Stream-Oriented Communication

- Transmission of continuous media
  - Audio and video
  - Successive images must be displayed at a uniform spacing in time (e.g., 33ms)

- Data stream
  - Asynchronous transmission mode
    - Data items in a stream are transmitted one after the other
  - Synchronous transmission mode
    - Maximum end-to-end delay is defined for each unit in a stream
  - Isochronous transmission mode
    - Max & Min end-to-end delay is defined for each unit of a stream

QoS (Quality of service) requirements
  (details are explained in week 5)

Summary

- Layered protocols and service models
  - OSI reference model, Internet protocol stack

- Middleware communication services
  - Remote Procedure Call (RPC)
  - Message-Oriented Communication
    - Transient communication, persistent communication
  - Stream-Oriented Communication