Inter-process Communication

CS 571

Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

IPC Examples

- Within a single computer
  - Pipes, Named Pipes (FIFO)
  - Message Queues
- Distributed systems
  - TCP/IP sockets
  - Remote Procedure Calls (RPC)
  - Remote Method Invocation (RMI)
  - Message-passing Libraries for parallel computing, e.g. MPI, PVM
  - Message-oriented Middleware

3

Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Only makes sense on a single computer unless distributed operating system that implements a global process name space is being used
- Properties of communication link
  - Links are established automatically.
  - A link is associated with exactly one pair of communicating processes.
  - The link may be unidirectional, but is usually bi-directional.

4

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).
  - Each mailbox has a unique id/address
- Primitives are defined as:

  send(*A, message*) – send a message to mailbox A

  receive(*A, message*) – receive a message from mailbox A
  - The mailbox address A can be local or remote
- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox

5

Issues in IPC

- Synchronous vs Asynchronous IPC
- Buffered vs unbuffered IPC
- Reliable vs unreliable (best effort)
- Ordered vs unordered
- Streams vs messages

6

Synchronization

- Message passing may be either blocking or non-blocking.
- **Blocking** is considered **synchronous**
- **Non-blocking** is considered **asynchronous**
- **send** and **receive** primitives may be either blocking or non-blocking.

7

Synchronization

- Synchronous receive
  - Receiving process blocks until message is copied into user-level buffer
- Asynchronous receive
  - Receiving process issues a receive operation (specifying a buffer) and then carries on with other tasks
  - It either polls the OS to find out if the receive has completed or gets an interrupt when the receive has completed
  - Threads allow you to program an asynchronous receive in a synchronous way
    - Issue a synchronous receive with one thread while carrying out other tasks with other threads

8

Synchronization cont'd

- OS view vs Programming Languages view of synchronous communication
- OS view
  - synchronous send ⇒ sender blocks until message has been copied from application buffers to kernel buffer
  - Asynchronous send ⇒ sender continues processing after notifying OS of the buffer in which the message is stored; have to be careful to not overwrite buffer until it is safe to do so
- PL view:
  - synchronous send ⇒ sender blocks until message has been received by the receiver
  - asynchronous send ⇒ sender carries on with other tasks after sending message (OS view of synchronous communication is asynchronous from the PL viewpoint)

9

Buffering

- Queue of messages attached to the link; implemented in one of three ways.
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous).
  2. Bounded capacity – finite length of *n* messages or N bytes. Sender must wait if link full.

10

Reliable and Ordered communication

- IPC within a computer is always reliable but messages sent across a network can get "lost"
  - Reliable communication, e.g. TCP
  - Unreliable or best effort communication, e.g UDP
- Ordered communication
  - TCP messages always delivered in order
  - UDP messages may not be delivered in same order as they were sent

Streams vs messages

- Streams
  - A "stream" of data is exchanged between sender and receiver
    - No message boundaries
  - Examples: "pipes" in UNIX, TCP streams
- Messages
  - Sender & receiver see the same set of distinct messages
  - Examples: "message queues" in UNIX, UDP messages/datagrams

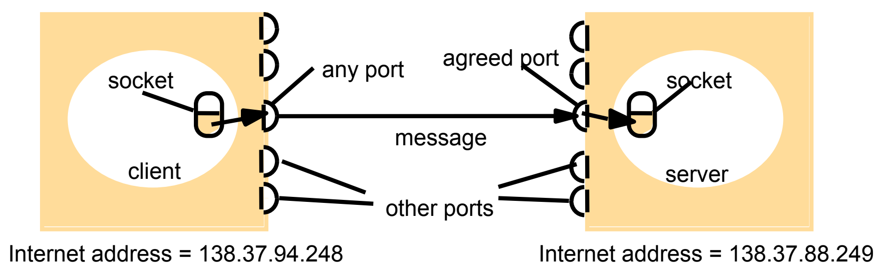
Client-Server Communication

- Sockets
- Remote Procedure Calls
- Remote Method Invocation (Java)

Sockets

- A socket is defined as an *endpoint for communication.*
- Concatenation of IP address and port
- The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Communication consists between a pair of sockets.

Sockets and ports



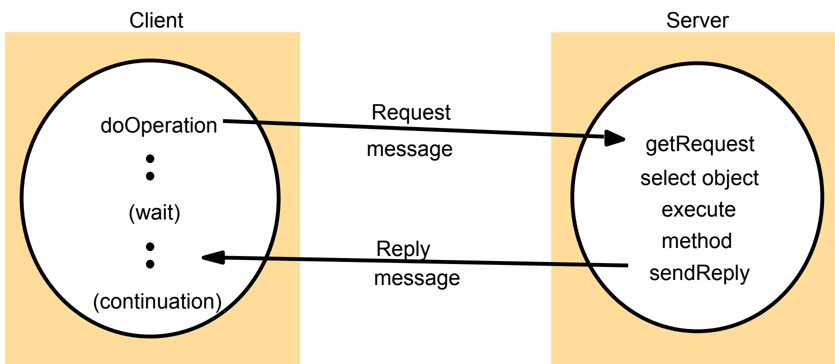
15

Higher-level IPC mechanisms

- Sockets API ≡ send & recv calls ≡ I/O
- Remote Procedure Calls (RPC)
  - Goal: to provide a procedural interface for distributed (i.e., remote) services
  - To make distributed nature of service transparent to the programmer
- Remote Method Invocation (RMI)
  - RPC + Object Orientation
  - Allows objects living in one process to invoke methods of an object living in another process

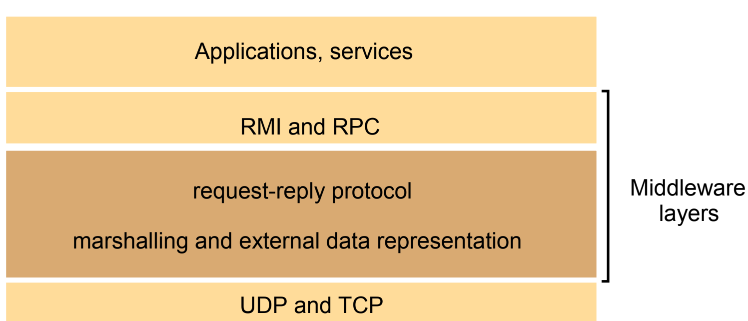
16

Request-reply communication



17

Middleware layers



18

Group Communication

- Client-server communication is one-to-one
- Many applications are group-oriented and require one-to-many or many-to-many communication
  - Pay per view on internet, distributed games, distributed conferencing
- Multicast communication
  - IP Multicast
  - Socket API supports multicast (over UDP)
- Issues: ordering, reliability