BUCKNELL UNIVERSITY
Computer Science

CSCI 315 Operating Systems Design

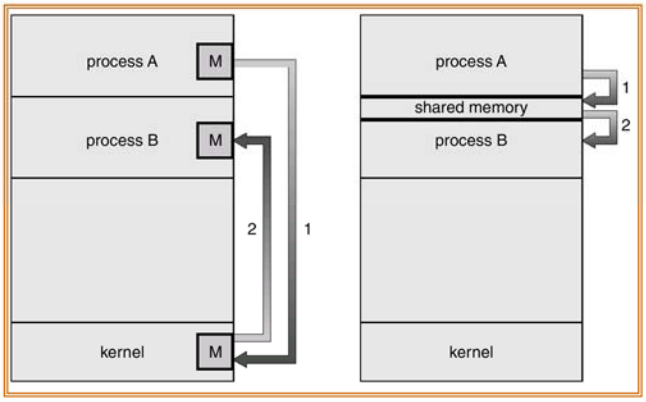
Interprocess Communication

Notice: The slides for this lecture have been largely based on those accompanying and earlier edirion of the course text *Operating Systems Concepts with Java*, by Silberschatz, Galvin, and Gagne. Many, if not all, of the illustrations contained in this presentation come from this source.

Cooperating Processes

- An ***independent*** process cannot affect or be affected by the execution of another process.
- A ***cooperating*** process can affect or be affected by the execution of another process.
- Advantages of process cooperation:
  - Information sharing,
  - Computation speed-up,
  - Modularity,
  - Convenience.

Communication Models



Message Passing Shared Memory

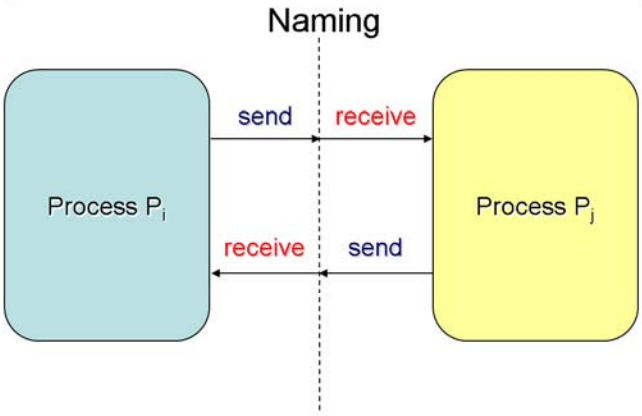
Interprocess Communication
(IPC)

- 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)

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Interprocess Communication (IPC)



naming (direct)

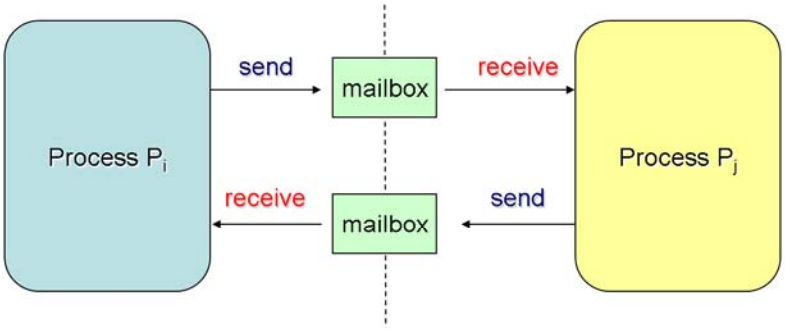
- send(P_j, message): P_j identifies process j in the system
- receive(P_j, message): P_i identifies process i in the system

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.
- Properties of communication link:
  - Links are established automatically.
  - A link is associated with exactly one pair of communicating processes.
  - Between each pair there exists exactly one link.
  - The link may be unidirectional, but is usually bi-directional.

Interprocess Communication (IPC)

Naming



naming (indirect)

- send(m_a, message): m_a identifies mailbox a in the system
- receive(m_b, message): m_b identifies mailbox b in the system

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports):
  - Each mailbox has a unique id,
  - Processes can communicate only if they share a mailbox.
- Properties of communication link:
  - Link established only if processes share a common mailbox,
  - A link may be associated with many processes,
  - Each pair of processes may share several communication links,
  - Link may be unidirectional or bi-directional.

Indirect Communication

- Operations:
  - create a new mailbox,
  - send and receive messages through mailbox,
  - destroy a mailbox.
- Primitives are defined as:

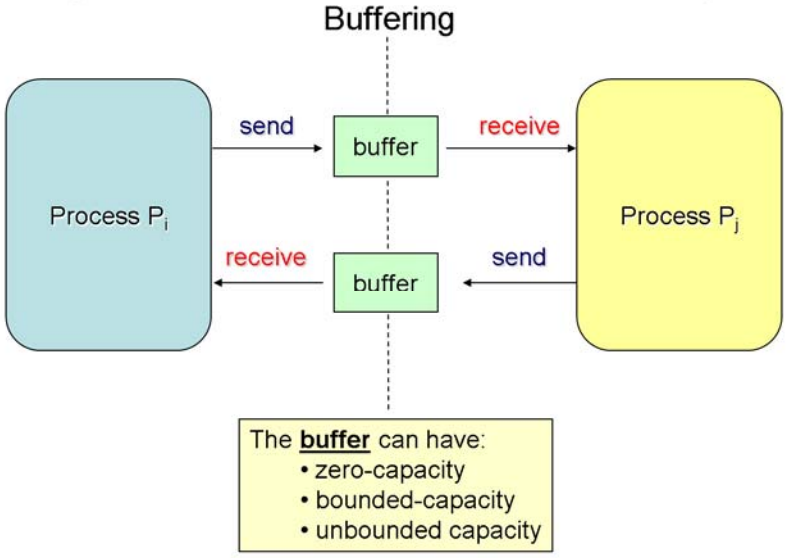
  send(*A, message*) – send a message to mailbox A,

  receive(*A, message*) – receive a message from mailbox A.

Indirect Communication

- Mailbox sharing:
  - P_1, P_2, and P_3 share mailbox A,
  - P_1, sends; P_2 and P_3 receive,
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes.
  - Allow only one process at a time to execute a receive operation.
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Interprocess Communication (IPC)



The **buffer** can have:
- zero-capacity
- bounded-capacity
- unbounded capacity

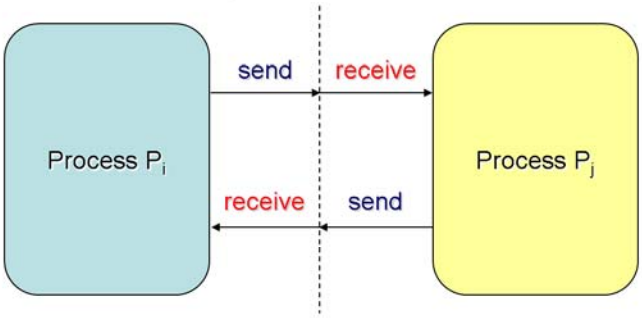
Buffering

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. Sender must wait if link full.
3. Unbounded capacity – infinite length. Sender never waits.

Interprocess Communication (IPC)

Synchronization



Blocking send
Blocking receive

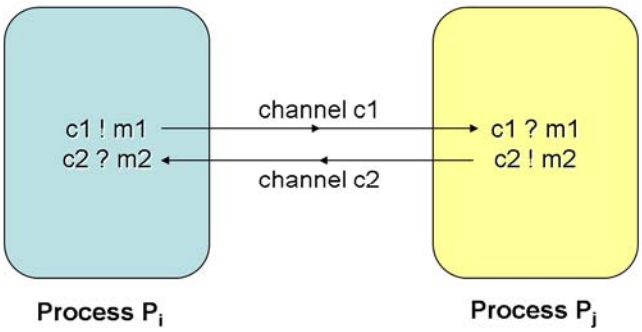
Non-blocking send
Non-blocking receive

Synchronization

- *Message passing* may be either blocking or non-blocking.
- **Blocking** is considered **synchronous**:
  - **Blocking send** has the sender block until the message is received.
  - **Blocking receive** has the receiver block until a message is available.
- **Non-blocking** is considered **asynchronous**
  - **Non-blocking** send has the sender send the message and continue.
  - **Non-blocking** receive has the receiver receive a valid message or null.

Interprocess Communication (IPC)

Simplifying the whole thing (CSP / occam)



rendezvous: blocking send, blocking receive, zero capacity channels

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Client-Server Communication

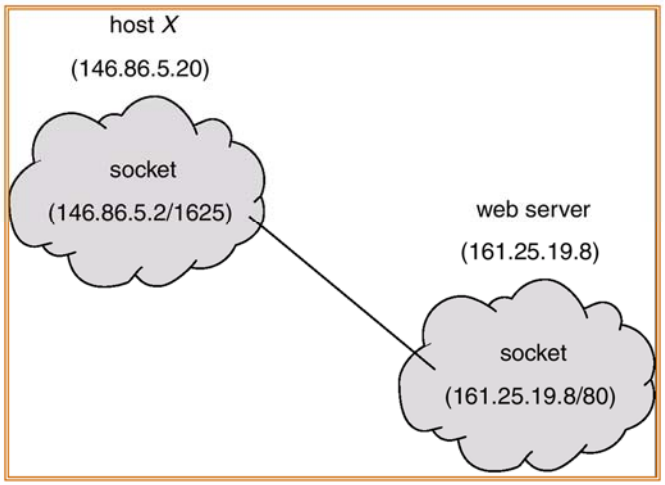
- Sockets
- Remote Procedure Calls (RPC)
- Remote Method Invocation (RMI - Java)

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.

See online **Appendix D** for sockets in C and C++.

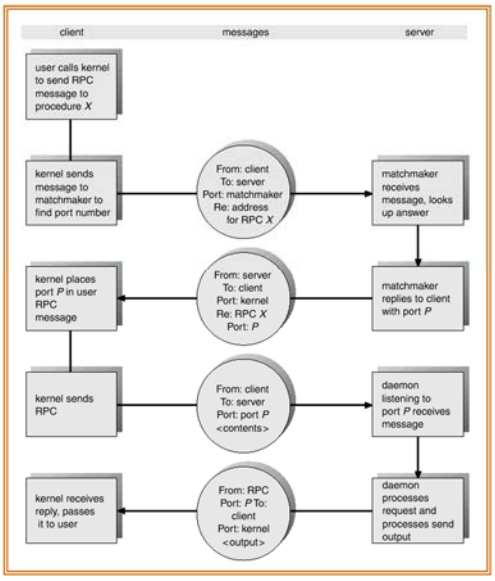
Socket Communication



Remote Procedure Calls

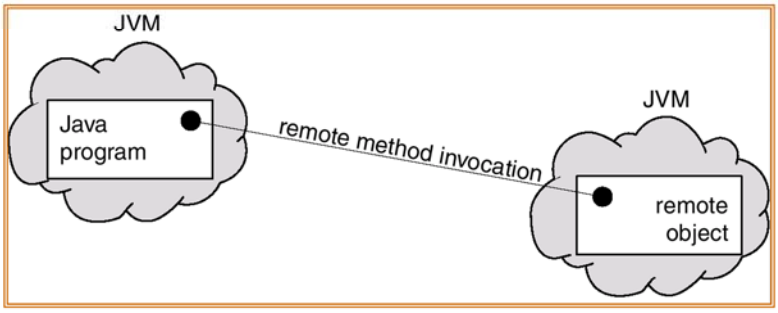
- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- **Stubs** – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and *marshalls* the parameters.
- The server-side stub receives this message, unpacks the marshalled parameters, and peforms the procedure on the server.

Execution of RPC

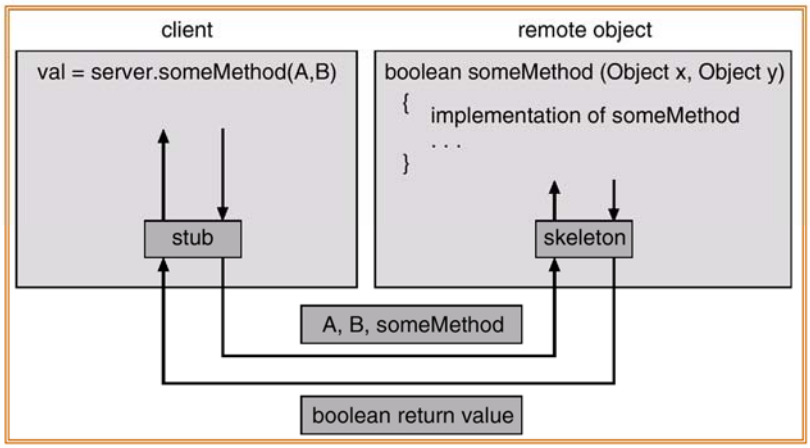


Remote Method Invocation

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
- RMI allows a Java program on one virtual machine to invoke a method on a remote object (on another virtual machine).



Marshalling Parameters



Parameter Passing

RPC comes from a procedural programming paradigm, while RMI comes from an object-oriented paradigm.

The parameters in a remote method invocation may be entire objects:

Support for **object serialization** is necessary.