PROCESSI (E THREAD) COOPERANTI

Processi indipendenti non possono modificare o essere modificati dall'esecuzione di un altro processo.

I processi cooperanti possono modificare o essere modificati dall'esecuzione di altri processi.

Vantaggi della cooperazione tra processi:

- Condivisione delle informazioni
- Aumento della computazione (parallelismo)
- Modularità
- Praticità implementativa/di utilizzo

COMMMUNICATING COOPERATING PROCESSES

The **producer-consumer** paradigm

- A buffer of items can be filled by the producer and emptied by the consumer.
- The producer and the consumer must be **synchronized**: the producer can produce one item while the consumer is consuming another item.
- The buffer can be unbounded or bounded. In the last case the consumer must wait if the buffer is empty and the producer must wait if the buffer is full.

The buffer may either be provided by the OS through an **Inter Process Communication (IPC)** facility or by explicitly coded by the application programmer with the use of **Shared Memory** solution.

SHARED-MEMORY SOLUTION

Cooperating Processes

Shared data

```
#define BUFFER_SIZE 10
Typedef struct {
    . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Producer process

```
item nextProduced;
    while (1)           {
        while (((in + 1) % BUFFER_SIZE) == out); /* do nothing */
        buffer[in] = nextProduced;
        in = (in + 1) % BUFFER_SIZE;
                        }
```

Consumer process

```
item nextConsumed;
        while (1)   {
                    while (in == out); /* do nothing */
                    nextConsumed = buffer[out];
                    out = (out + 1) % BUFFER_SIZE;
        }
```

INTERPROCESS COMMUNICATION (IPC)

Meccanismo di comunicazione e interazione tra processi (e thread)

Questioni da considerare:

- Come può un processo passare informazioni ad un altro?
- Come evitare accessi inconsistenti a risorse condivise?
- Come sequenzializzare gli accessi alle risorse secondo la causalità?
- Mantenere la consistenza dei dati richiede dei meccanismi per assicurare l'esecuzione ordinata dei processi cooperanti.

INTERPROCESS COMMUNICATION (IPC)

- Mechanism for processes to communicate and to synchronize their actions.
- **Message-Passing** system: processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations: ***primitives*** (or ***system calls*** or ***library functions***)
  - **send**(*destination, message*) – normally non blocking; message size fixed or variable
  - **receive**(*source, &message*) – normally blocking until the message becomes available

Problematiche dello scambio di messaggi

- Affidabilità: i canali possono essere inaffidabili (es: reti). Bisogna implementare appositi protocolli fault-tolerant (basati su acknowledgment e timestamping).
- Autenticazione: come autenticare i due partner?
- Sicurezza: i canali utilizzati possono essere intercettati
- Efficienza: se prende luogo sulla stessa macchina, il passaggio di messaggi è sempre più lento della memoria condivisa e semafori.

INTERPROCESS COMMUNICATION (IPC)

- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive

- From the logical point of view, a communication link may be implemented in several ways:
  - Direct (through process name) or Indirect communication (through *mailbox* or *port*)
  - Symmetric or Asymmetric communication (the receiver knows the sender or not)
  - Synchronous or Asynchronous communication (blocking or non blocking primitives)
  - Extended or Limited rendez-vous communication (when both the send and the receive are blocking or not)
  - Unbuffering or Buffering (with bounded or unbounded capacity) communication queue.

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.
- A variant employs asymmetry:
  - **send** (*P*, *message*) – send a message to process P
  - **receive**(*id*, *message*) – receive a message from any process; *id* is the name of the current process.

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.

- 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

I messaggi spediti ma non ancora consumati vengono automaticamente bufferizzati in una mailbox (mantenuta in kernel o dalle librerie).

La send si blocca se la mailbox è piena; la receive si blocca se la mailbox è vuota.

- Mailbox owner
  - A process (the mailbox is part of its address space and it can only receive messages) or the OS
  - The owner process is that creates a new mailbox
  - When the owner process terminates, the mailbox disappears

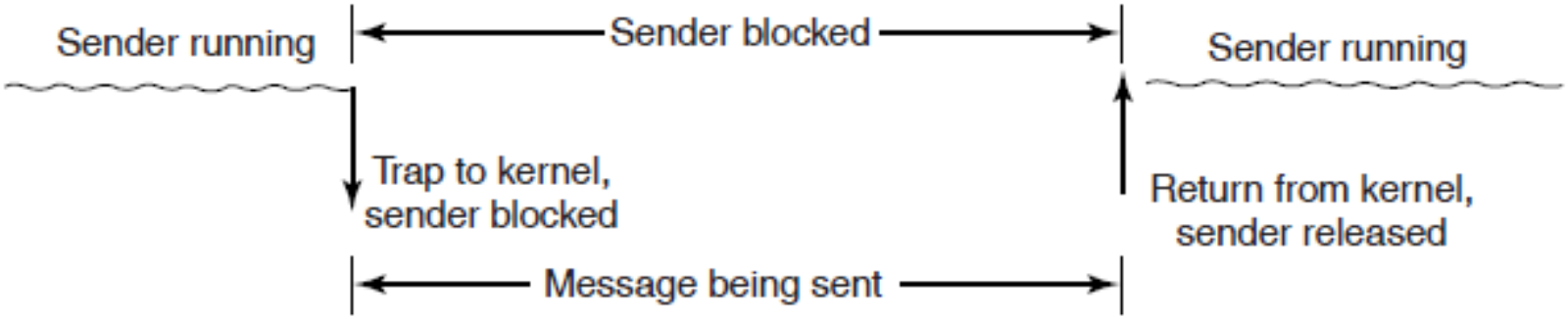
- Mailbox sharing
  - P_1, P_2, and P_3 share mailbox A.
  - P_1, sends; either P_2 or P_3, but not both, will receive the message.

- 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.

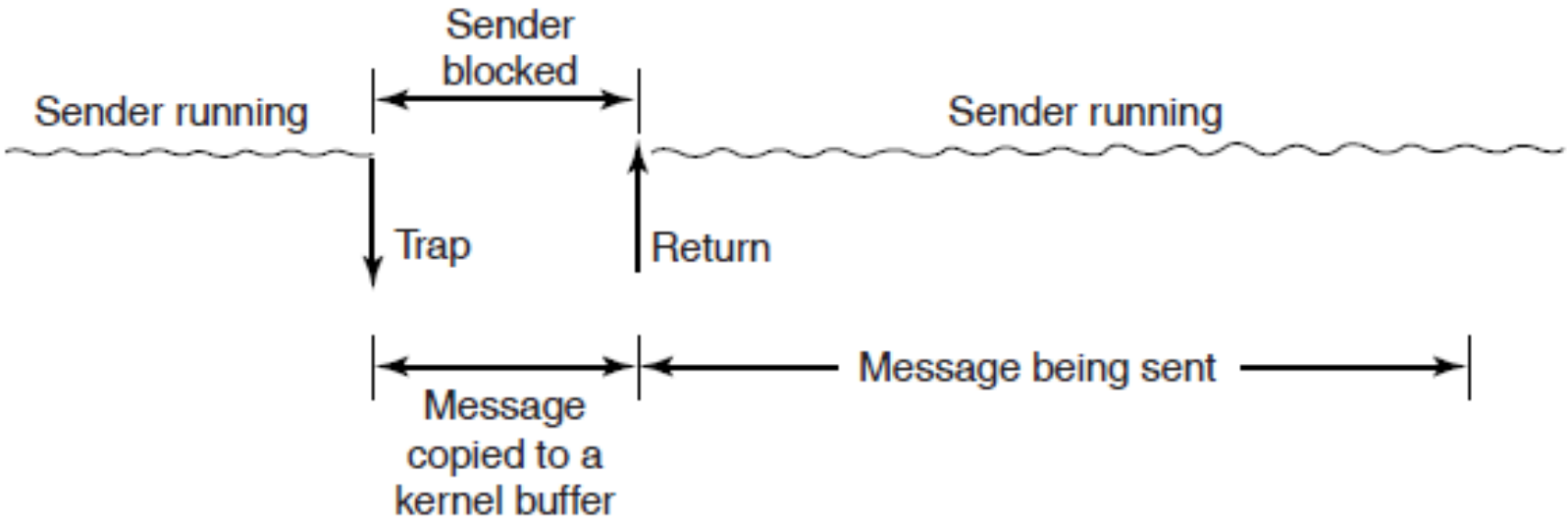
COMMUNICATION SYNCHRONIZATION

Φ Message passing may be either blocking or non-blocking.

Φ **Blocking** is considered **synchronous.** I messaggi vengono spediti direttamente al processo destinatario. Le send e receive si bloccano fino a che la controparte non esegue la chiamata duale (rendez-vous)



Φ **Non-blocking** is considered **asynchronous**



SEND BLOCCANTI/NON BLOCCANTI

Quattro possibilità:

- ***send bloccante***: CPU inattiva durante la trasmissione del messaggio
- ***send non bloccante, con copia*** su un buffer di sistema: spreco di tempo di CPU per la copia
- ***send non bloccante, con interruzione*** di conferma: molto difficile da programmare e debuggare
- ***copia su scrittura***: il buffer viene copiato quando viene modificato

Tutto sommato, la (1) è la migliore (soprattutto se abbiamo a disposizione i thread).

COMMUNICATION 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). Comunicazione sincrona tra processi, senza mailbox.
2. Bounded capacity – finite length of *n* messages
   Sender must wait if link full.
3. Unbounded capacity – infinite length
   Sender never waits.

CLIENT-SERVER COMMUNICATION

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

CLIENT-SERVER COMMUNICATION

Socket

A socket is defined as an ***endpoint for communication*** and is associated (***bound***) to an IP address.

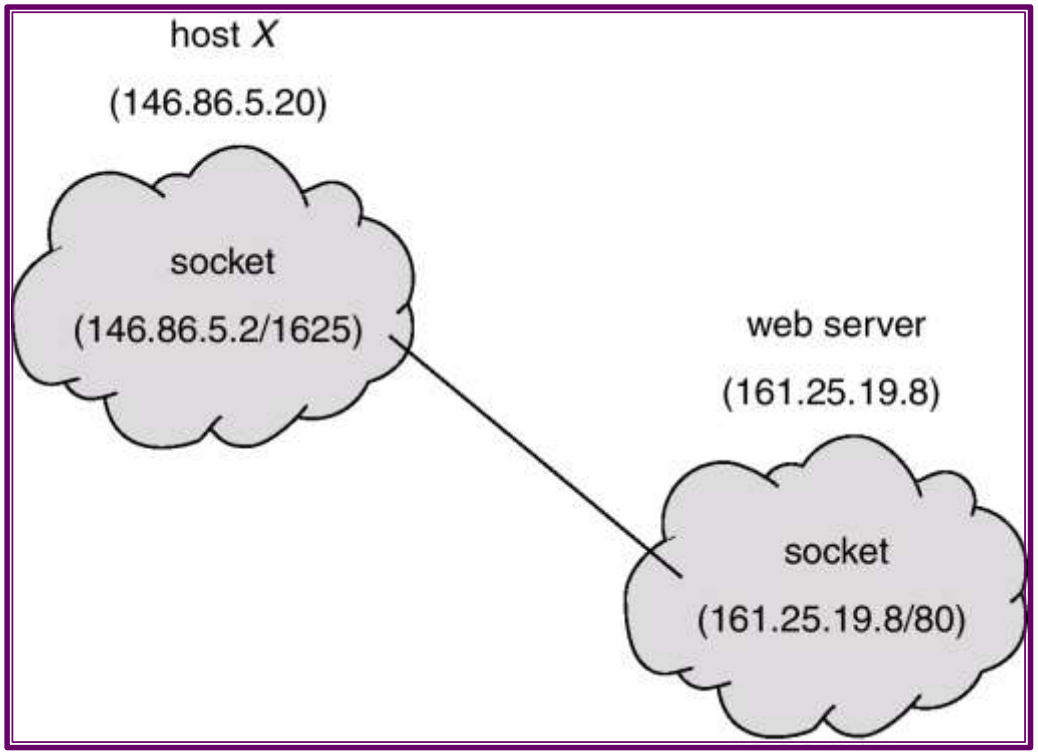
A socket is made up of an IP address concatenated with a port number.

The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**

Communication consists of a pair of sockets.

The server waits for incoming client requests by listening to a specified port, accepts the connection request from the client socket and completes the connection.

Servers offering specific services listen to well-known ports: port 80 for web or http, port 21 for ftp, port 23 for telnet).



CLIENT-SERVER COMMUNICATION

Programmazione in multicomputer: Remote Procedure Call (RPC)

Chiamata di procedura remota: il modello di computazione distribuita più astratto

Idea di base: un processo su una macchina può eseguire codice su una CPU remota.

L'esecuzione di procedure remote deve apparire simile a quelle locali.

Nasconde (in parte) all'utente la delocalizzazione del calcolo: l'utente non esegue mai send/receive, ma deve solo scrivere ed invocare procedure come al solito.

Versione a oggetti: RMI (Remote Method Invocation)

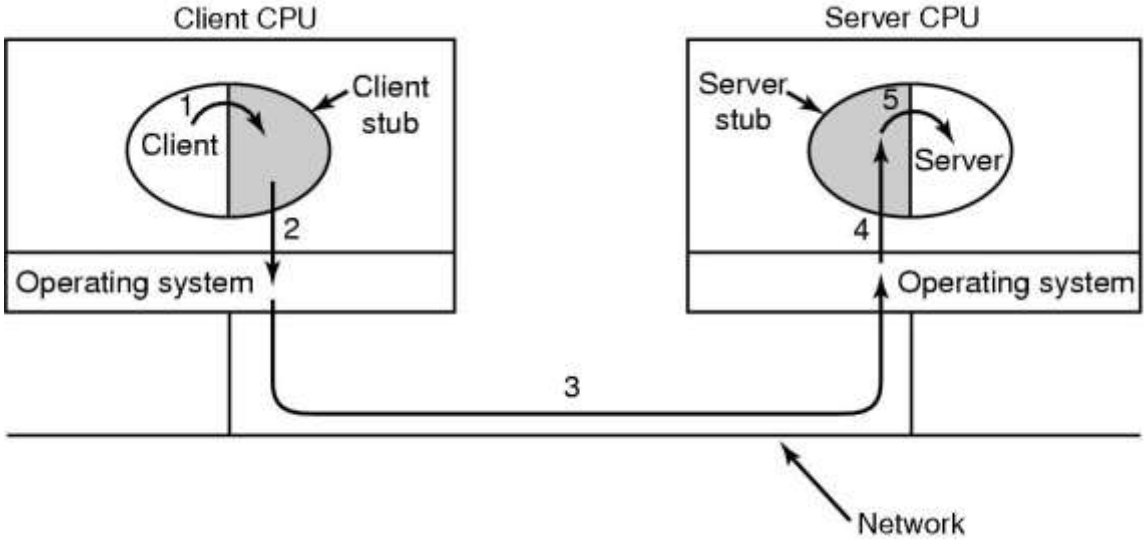
CLIENT-SERVER COMMUNICATION

Remote Procedure Call

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

☞ Implementation Issues

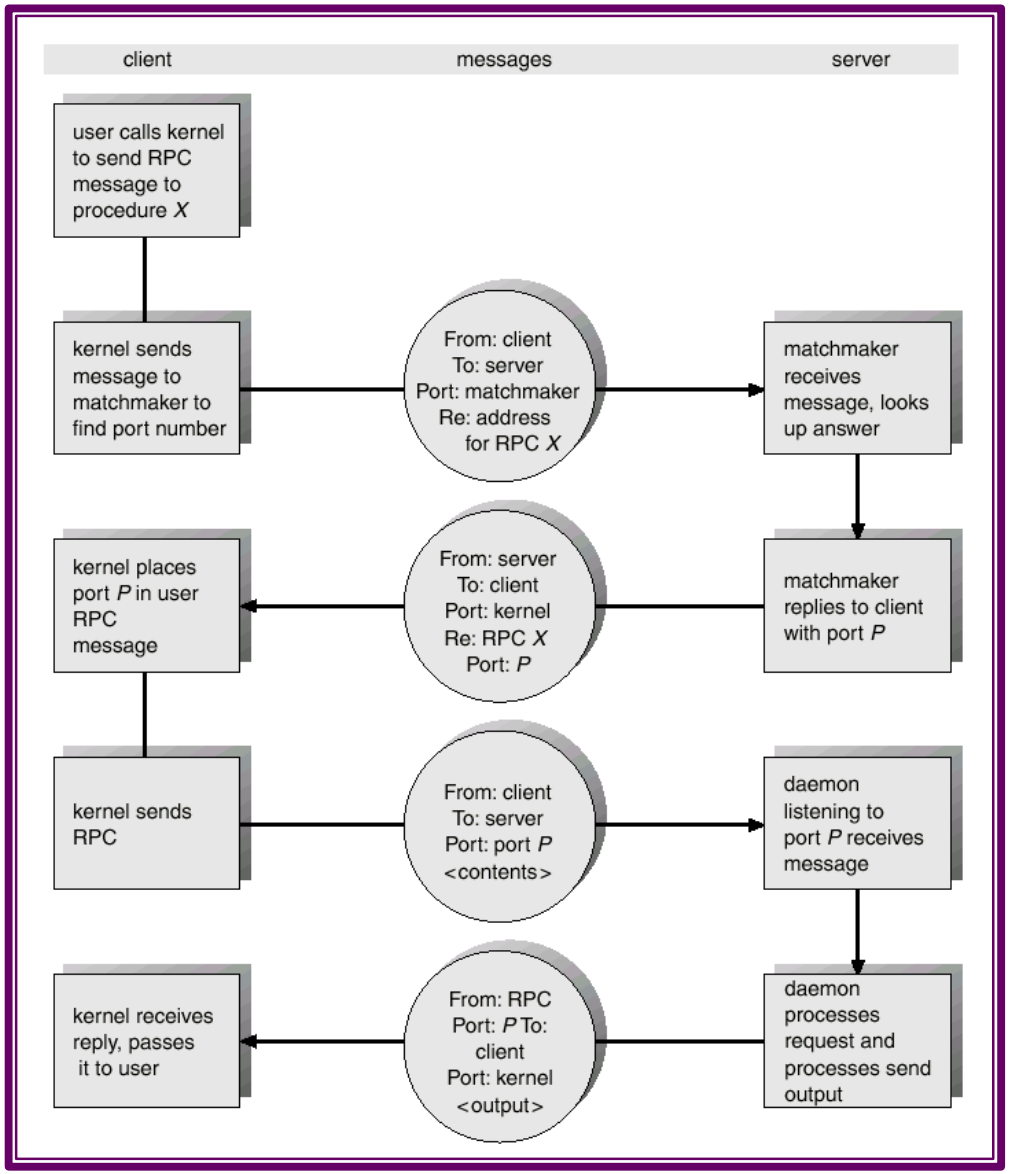
- Cannot pass pointers
  - call by reference becomes call by copy-restore (but might fail)
- Weakly typed languages
  - client stub cannot determine size
- Not always possible to determine parameter types
- Cannot use global variables
  - may get moved to remote machine



2 → marshalling

CLIENT-SERVER COMMUNICATION

Remote Procedure Call



CLIENT-SERVER COMMUNICATION

Remote Method Invocation (RMI)

- RMI is a Java mechanism similar to RPCs.
- RMI allows a Java program on one machine to invoke a method on a remote object.