IPC (2)

Sistemas Distribuidos
I.T.I. Sistemas (2005-06)
© César Llamas Bello
Universidad de Valladolid

Indice

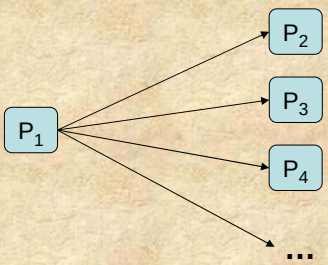
IPC

- interprocess communication
  - Es una noción básica para SD.
  - Ojo, necesitamos mecanismos de ipc diferentes a los de nivel de SO.
  - Es una noción dual:
    - Si queremos sincronizar, debemos comunicar, porque no hay reloj global.
    - Si queremos comunicar, debemos sincronizar.

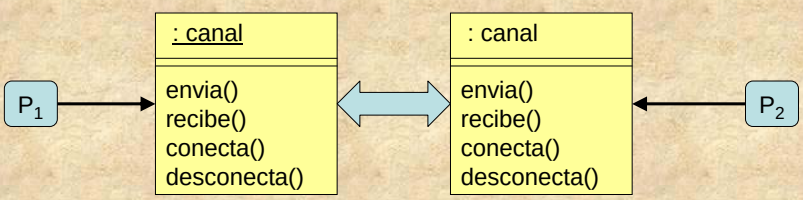
Uni y Multi

- Unidifusión *unicast*

- Multidifusión *multicast*



Arquetipo de API para IPC



- Orden de invocación:
  - conectar( ): iniciar conexión/aceptar conexión
  - enviar( ) / recibir( )
  - desconectar()

Ej.: HTTP

No confundir el protocolo petición()/respuesta() con el par envía()/recibe()

Cliente Web

C_1. inicia_conexión()

C_2. envia(petición)

C_3. recibe(respuesta)

C_4. desconecta()

Servidor Web

S_1. acepta_conexión()

S_2. recibe(petición)

S_3. envía(respuesta)

S_4. desconecta()

Sincronicidad

- Cada primitiva del API es invocada en en instantes diferentes, siguiendo secuencias diferentes.
- Tipos de esquemas de IPC:
  - Síncronos: necesitan que los procesos se sincronicen en algunos puntos.
  - Puramente asíncronos: no necesitan que los procesos se sincronicen en absoluto.
- Aun así, hay que respetar ciertas secuencias de eventos.

¿Qué es sincronizar?

- Esperar a que ocurran ciertos eventos.
  - La espera se traduce en un bloqueo.
  - Puede ser activa (bucle) ☹
  - Puede ser inactiva, en cuyo caso se gestiona desde el S.O.
- Frecuentemente las operaciones bloqueantes se denominan síncronas, y las no bloqueantes, asíncronas.

¿porqué? ☹

Comunicación

- Se dice que ha habido comunicación, cuando se ha completado el par envío/recepción.
  - Por extensión se dice que se ha completado el ***par*** envía()/recibe().
- El evento de llamada a cada primitiva, y el evento de finalización de cada una, puede ser distinto.
  - 4 eventos -> 4 tiempos distintos.

Sincronía

- envía() y recibe() son:
  - Síncronas con la comunicación:
    - Lo que suele acarrear un bloqueo en el interlocutor hasta que la plataforma asegura que se ha realizado el par.

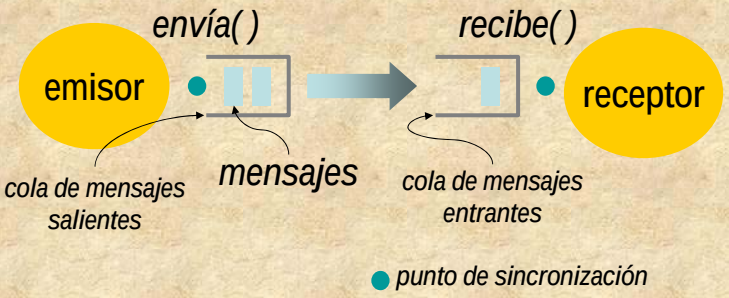
  o

  - Asíncronas con la comunicación:
    - El interlocutor prosigue aunque no se haya completado la comunicación.

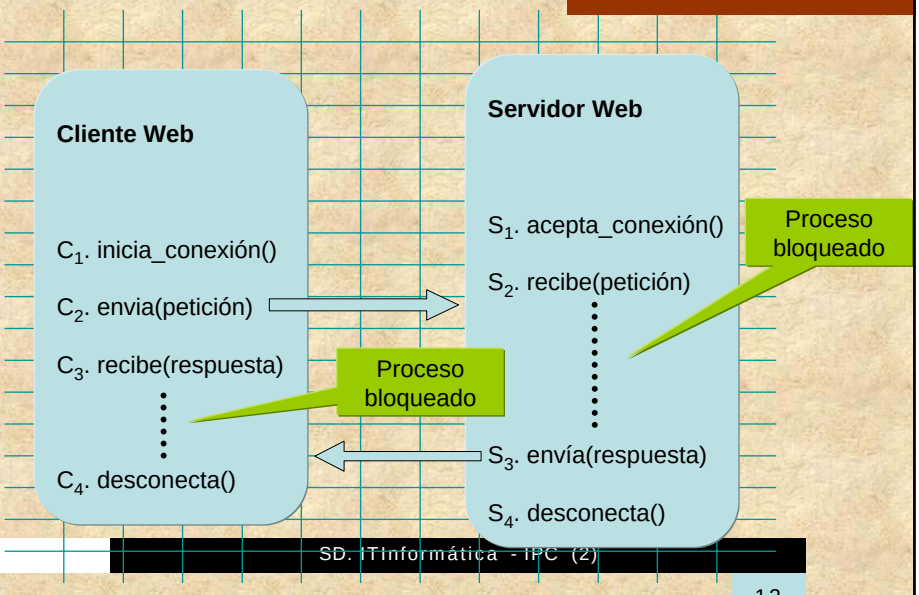
  - Por comodidad se identifica síncrono con bloqueante y asíncrono con no bloqueante.

Soporte de asincronía

- Para proporcionar primitivas asíncronas, la plataforma suele recurrir al uso de colas.



Ej.: HTTP

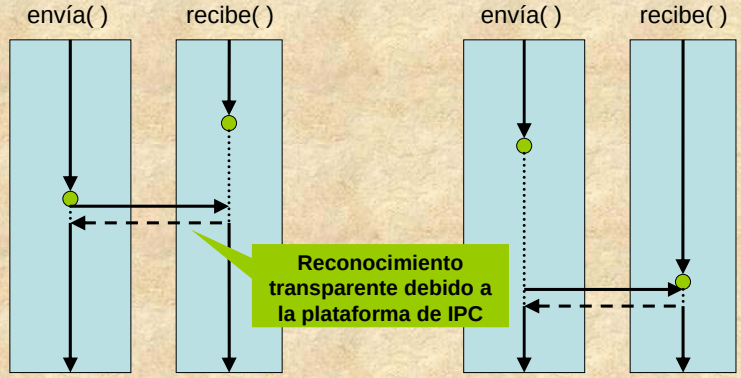


Posibilidades de sincronización

recibe ( ) \ envía ( )	Asíncrono	Síncrono
Asíncrono	✓ ④	✓ ③
Síncrono	✓✓✓ ②	✓✓ ①

① envía() - sincro / recibe() - sincro

- envía ( ) y recibe ( ) provocan el bloqueo hasta que se completa el par envía/recibe.

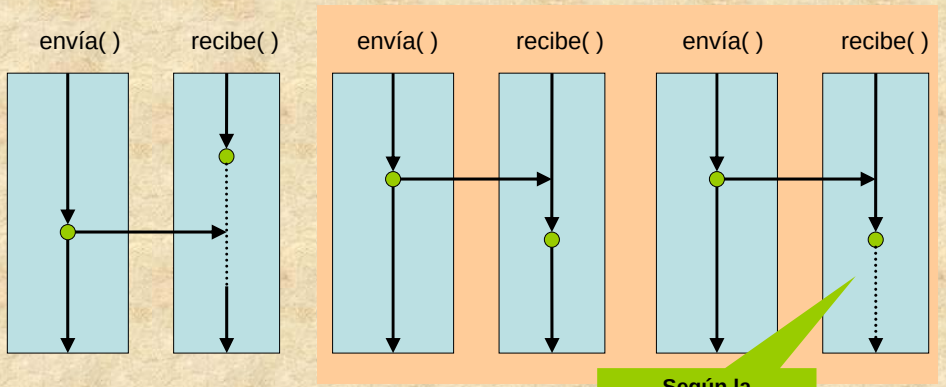


① envía() - sincro / recibe() - sincro

- Precisa que se verifique el par envía/recibe antes de proseguir.
- En ocasiones se puede indicar a recibe () una cantidad de datos a recibir, de forma que se elimina el bloqueo cuando se cumplimente dicha cantidad

② envía() - sincro / recibe() - sincro

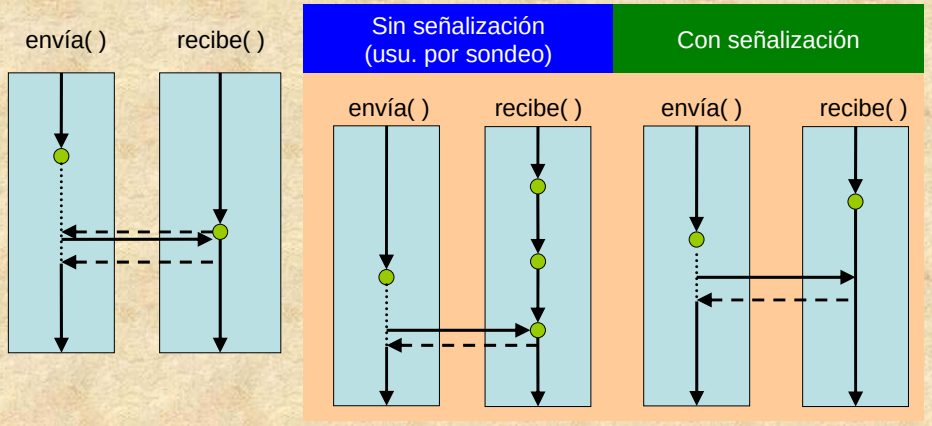
- Solo se bloquea recibe() si es preciso



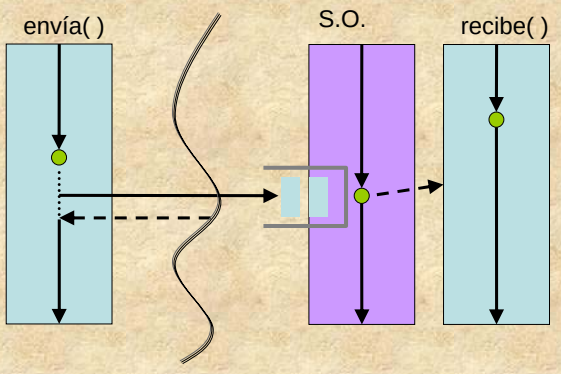
Base del modelo de protocolos de tipo petición/respuesta

③ envía() - sincro / recibe() - asincro

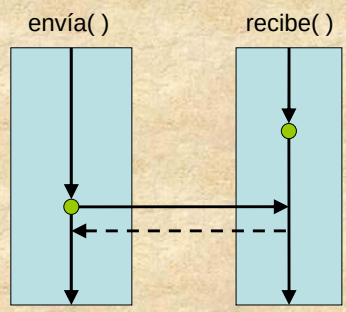
- Solo se bloquea recibe() si es preciso



③ (señales)



④ envía() - asincro / recibe() - asincro



- Similar a la utilización de una variable compartida como canal

- No hay bloqueo.
- Tanto el emisor como el receptor pueden sondear, o recibir señales de la plataforma para conocer si la comunicación se ha completado.

Progreso de procesos

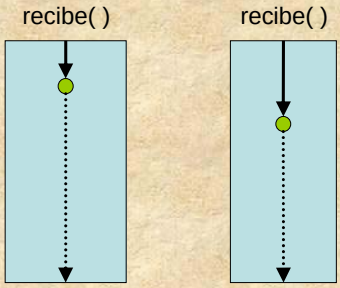
- Los procesos pueden dejar de progresar por:
  - Interbloqueo (la traza de eventos) (malo-malo)
  - Bloqueo (la temporización concreta)
- Los timeouts ofrecen una solución general:
  - Actúan sobre la plataforma (configuración)
  - Actúan sobre primitivas concretas.

Bloqueo

- Las operaciones podrán bloquear los hilos, incluso indefinidamente.
- Soluciones:
  - Engendrar hilos para despachar las peticiones (si es posible)
    - Recordar servidores multienhebrados.
  - Colocar timeouts (tiempos límites de espera)
  - Antes de recibir, sondear el canal (select).

Interbloqueo

- Los interbloqueos (deadlocks), se dan cuando los procesos no están bien sincronizados, por …
  - errores de programa
  - malentendidos en el protocolo
  - Solución: timeouts.



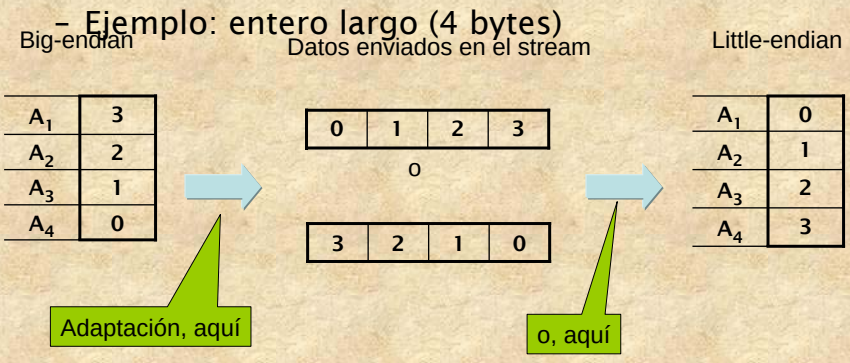
Problemas con los datos

- La red solo transmite streams de bytes
- Cada máquina representa los tipos de datos básicos de formas diversas:
  - Enteros: big-endian, little-endian
  - UNIX-char=1byte, UNICODE=2bytes,
- Los tipos compuestos
  - Tipos estructurados
  - Tipos dinámicos

  se organizan diferente según: el lenguaje, el compilador, la arquitectura.

Representación de datos

- Compatibilidad a nivel de aplicación, para
  - Garantizar compatibilidad a nivel de tipos de datos.
  - Ejemplo: entero largo (4 bytes)



Tareas con los datos

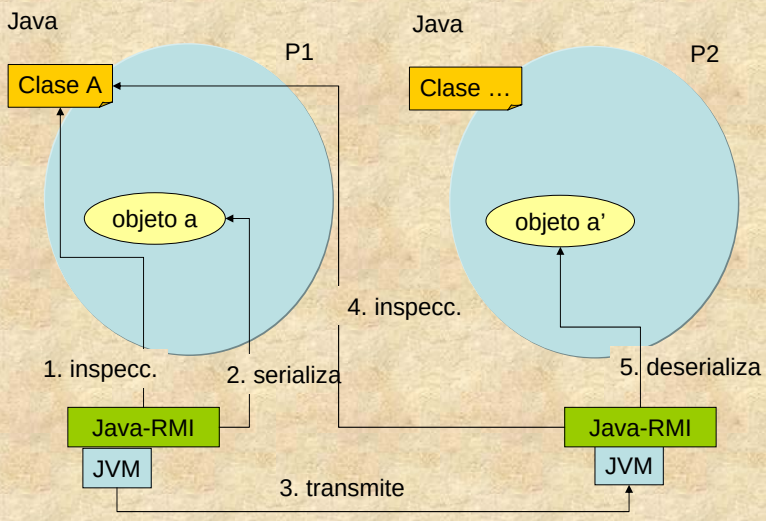
- Cuando hay procesos heterogéneos, podemos adoptar tres esquemas para interpretar los datos apropiadamente:
  - Antes de envía(), se convierten los datos al receptor.
  - Tras recibe(), se convierten los datos en el receptor.
  - Se empaquetan los datos, se emiten en un estándar de **representación externa**, y en el receptor se desempaquetan.

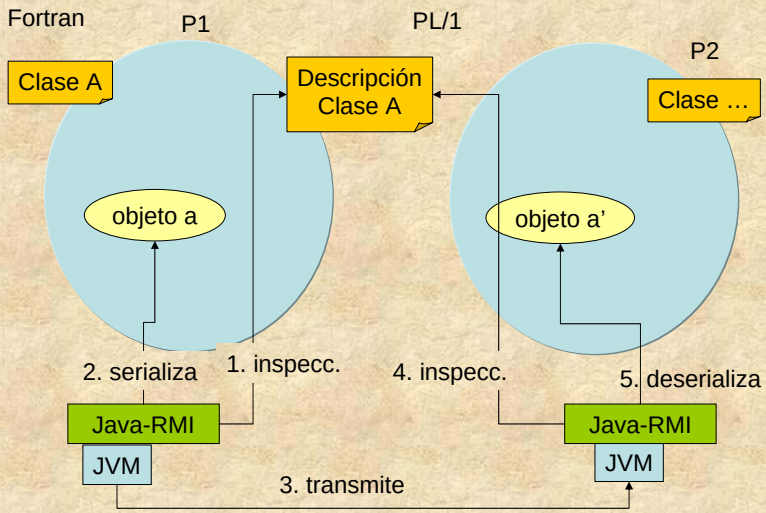
Marshalling/unmarshalling

- Para empaquetar los datos (marshall)
  1. Se cambian apuntadores a memoria por referencias enteras.
  2. Se convierten las estructuras a tiras de datos elementales.
  3. Se pasan los datos elementales a representación binaria compatible.
- Para desempaquetar los datos (unmarshall) (a la inversa)

p.ej.: con ASN1

- Los empaquetadores/ desempaquetadores,
  - Los proporciona la plataforma (ej.:Java-RMI, .NET)
  - Podemos escribirles, basándonos en bibliotecas, (ej.: RPC)
  - Podemos generarles mediante compiladores de archivos de reglas, (ej.: CORBA-IDL, RPC,…)
  - Los proporciona la plataforma, mediante archivos de reglas, (ej.: con ASN.1)





Lenguajes de representación externa

- MIME, se apoya en gestores.
- RPC, XDR (de Sun), abierto, básico, para estructuras de datos.
- ASN.1 (CCITT, ahora ITU), abierto, muy completo, orientado a protocolos, rápido.
- CDR, (para CORBA, de OMG), abierto, muy completo, orientado a interfaces.
- XML (W3C), con esquemas XML-RPC y SOAP, abierto, completamente extensible, orientado a protocolos, o a interfaces, … Lento (orientado a texto, aunque puede mezclarse con ASN.1)
- …

Ej.: Common Data Representation

15 datos primitivos: short, long, float, double, char, boolean, octet, ...

tipos compuestos (*any*)

Tipo	***Representación***
sequence	longitud (unsigned long-entero largo sin signo-) seguida de los elementos en orden.
string	longitud (unsigned long) seguida de los caracteres en orden (también puede tener caracteres anchos-2bytes-).
array	elementos de la cadena en orden (no se especifica la longitud porque es fija).
Struct	en el orden de declaración de los componentes.
Enumerated	unsigned long (los valores son especificados por el orden declarado).
Union	etiqueta de tipo seguida por el miembro seleccionado.

CDR -Ejemplo

- Ejemplo de especificación IDL:
  - Los procedimientos de *marshalling* y *unmarshalling* se generan con el compilador IDL (ej:idltoC, idltojava, ...)

```
struct Persona {
  string nombre;
  string lugar;
  long año;
}
```

CDR -Ejemplo

- La estructura de la forma aplanada es:

Posición en la secuencia de bytes	4 bytes	Notas
0 - 3	5	*Longitud del string*
4 - 7	"Pére"	*«Pérez»*
8 - 11	"z___"	
12 - 15	6	*Longitud del string*
16 – 19	"Madr"	*«Madrid»*
20 – 23	"id__"	
24 – 27	1934	*unsigned long*

Serialización en Java

```
public class Persona implements Serializable {
    private String nombre;
    private String lugar;
    private int año;
    public Persona(String unNombre, String unLugar, int unAño) {
        nombre = unNombre;
        lugar = unLugar;
        año = unAño;
    }
    // siguientes métodos de la clase
}
```

Serialización en Java

- serialización ≈ marshalling
  - La reflexión de Java permite efectuarla de modo transparente.
  - Es recursiva (recorre grafos de objetos)
  - Los métodos de serialización/deserialización son internos a Java

Persona	Número de versión de 8-bytes		a0	*Nombre de la clase, número de versión*
3	int año	java.lang.String nombre:	java.lang.String lugar:	*Número, tipo y nombre de las variables de instancia*
1934	5 Pérez	6 Madrid	a1	*Valores de las variables de instancia*

- La forma serializada real contiene marcadores de tipo adicionales
- a0 y a1 son apuntadores

Serialización en Java

- **Trama de datos enviados:**
  - 'Persona'
  - Número de versión de 8-bytes
  - a0
  - 3
  - int año
  - java.lang.String nombre:
  - java.lang.String lugar:
  - '1934'
  - 5 'Pérez'
  - 6 'Madrid'
  - a1
- La forma serializada real contiene marcadores de tipo adicionales
- a0 y a1 son apuntadores

Serialización en Java

- Sin embargo, no todo es serializable, pues al des-serializar…
  - el mecanismo básico puede ser ineficiente
  - ciertos campos (*transient*, típicamente) pueden recibir valores inapropiados.
  - Ej.: clase HashMap (tiene sus propios métodos)
- El mecanismo de serialización es aun así, más versátil que el CDR de CORBA.

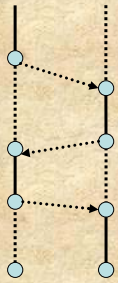
Referencias remotas en Java

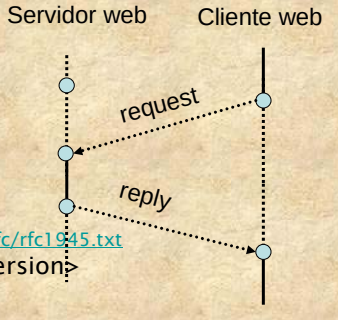
32 bits	*32 bits*	*32 bits*	*32 bits*	
dirección Internet	**número de puerto**	**tiempo**	**número de objeto**	**interfaz de objeto remoto**

- Los identificadores a objetos remotos tienen ámbito global y son únicos
- Pueden enviarse como parámetro y recibirse desde un método
- Pueden compararse
- Tiene un ámbito de utilización
  - Se puede garantizar incluyendo periodos de validez

Protocolos

- Diagramas de eventos

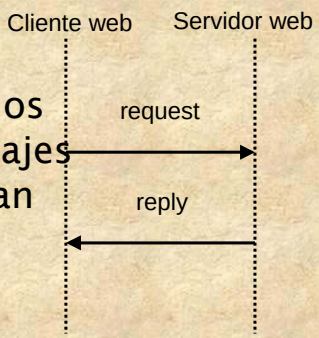




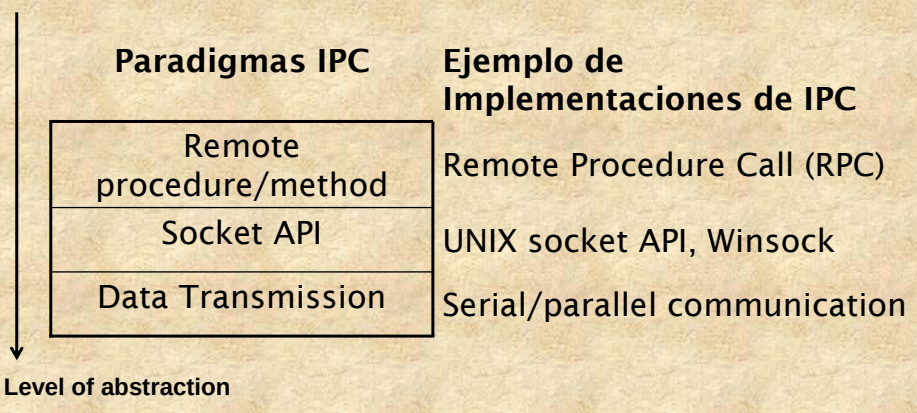
- Request http: http://www.watersprings.org/pub/rfc/rfc1945.txt
  <command><document address><HTTP version>
  an optional header
  optional data for CGI data using POST
- Response:
  status line: <protocol><status code><description>
  header information (possibly several lines)
  the document

Protocolos

- Diagrama de secuencia
  - Más habitual en UML,
  - Las flechas no indican envios de información, sino mensajes a objetos que desencadenan métodos.
  - En terminología de SD
    - Peticiones
    - Respuestas
  - En UML 1.0 existen mensajes asíncronos.



Evolución de IPC



Resumen

- IPC: raíz de los SD
  - IPC: separar procesos y comunicarlos (unicast, multicast)
  - API básico:
    - Primitivas
    - Sincronización de eventos (bloqueo: sincrono, asíncrono)
    - Empaquetado/Desempaquetado
  - Diferentes representaciones de red
  - Protocolos petición/respuesta: modelo de interacción
  - Diagramas:
    - De eventos: detallan secuencias de eventos
    - De interacción: detallan el flujo de ejecución.