UNIVERSIDAD CARLOS III DE MADRID
AREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES
GRADO EN INGENIERÍA INFORMÁTICA. SISTEMAS DISTRIBUIDOS

Para la realización del presente examen se dispondrá de **1 hora y 15 minutos. NO** se podrán utilizar libros, apuntes **ni** calculadoras de ningún tipo. Todos los ejercicios se responden en el espacio reservado.

Alumno: ____Grupo: _________

Ejercicio 1. Responda a las siguientes preguntas cortas justificando brevemente su respuesta.

a) Indique las principales características de los sistemas distribuidos.

- Heterogeneidad es la variedad y diferencia en los componentes que conforman un sistema distribuido. Por ejemplo, en las redes, arquitecturas de computadoras, ordenamiento de los bytes, etc.
- Nombrado es la característica que permite identificar los objetos, computadores y los recursos que se quieren compartir.
- Comunicación y sincronización tiene que ver con cómo se realiza la comunicación entre los procesos y cómo se van a sincronizar éstos (primitivas de comunicación bloqueantes o no bloqueantes).
- Escalabilidad o capacidad de crecimiento, es la capacidad de un sistema de preservar su capacidad cuando se incrementa de manera significativa el número de recursos o usuario.
- Transparencia es la ocultación al usuario de los componentes que conforman el sistema distribuido. La transparencia se puede dar en distintos
- Fiabilidad es la probabilidad de que un sistema desarrolle una determinada función bajo condiciones fijadas y un período de tiempo definido. La fiabilidad tienen varios aspectos: consistencia, seguridad y tratamiento de fallos.
- Calidad de servicio es la habilidad de satisfacer los requerimientos de tiempo cuando se transmiten y procesan flujos de datos multimedia en tiempo real.
- Estructura de software representa la arquitectura de SW del sistema distribuido. Existen distintas posibilidades: sistema operativo en red, sistema operativo distribuido y middleware.

b) Describa brevemente los paradigmas de comunicación que conoce.

- **Paso de mensajes,** en el nivel de abstracción más bajo los procesos comunican mediante paso de mensajes. Las primitivas básicas son send(mensaje, destino) y receive(mensaje, origen).
- **Cliente-servidor**, aplicaciones donde dos procesos con roles bien diferenciados (cliente y servidor) van a comunicar para compartir recursos que residen en el servidor. Existe un participante activo (el cliente) que es el que inicia la comunicación y uno pasivo (el servidor) que espera las peticiones de servicio de los clientes.
- **Llamadas a procedimientos remotos**. Este paradigma de comunicación permite acercar la semántica de invocación de procedimientos locales a aplicaciones distribuidas. Cuando un proceso solicita un procedimiento remoto se invoca el stub del cliente que enmascara la comunicación con el proceso servidor y envía la petición con los argumentos. En el proceso servidor, el stub del servidor lee la petición, invoca localmente el procedimiento remoto y responde al cliente con la respuesta.

- **Peer-to-peer.** En este paradigma de comunicación, a diferencia del paradigma cliente-servidor, todos los procesos tienen el mismo rol. Un ejemplo típico de este tipo de aplicaciones es la compartición de ficheros multimedia.
- **Objetos distribuidos** se basa en llamadas a procedimientos remotos. Este paradigma extiende el concepto de procedimiento remoto a objeto remoto, teniendo en cuenta los lenguajes de programación orientados a objetos.
- **Agentes móviles.** En este paradigma de comunicación no hay intercambio de mensajes sino que un fichero binario se intercambia.
- **Servicios en red.** En este paradigma de comunicación se obtiene una referencia para poder acceder al servidor de un determinado recurso. Dicha referencia se proporciona por un servicio de directorio que es el encargado de registrar los distintos servicios de red.
- **Comunicación colaborativa o en grupo** es un paradigma de comunicación que permite crear sesiones de trabajo colaborativo en las que participan un conjunto de procesos. Esos procesos pueden comunicar mediante mensajes unicast, multicast, y broadcast.

c) ¿Qué es una función idempotente y una no idempotente? Ponga un ejemplo

Una función es **idempotente** cuando devuelve siempre el mismo resultado. Un ejemplo de función idempotente es la suma de dos números. De otra manera la función es **no idempotente.** Un ejemplo de función no idempotente es la operación de retirar X dinero de una cuenta bancaria. Las operaciones no idempotentes requieren una mayor atención para garantizar una determinada semántica de ejecución: al menos una vez, como mucho una vez, exactamente una vez.

d) Razone brevemente la siguiente frase: "La computación cliente-servidor no es computación distribuida sino acceso distribuido"

En un algoritmo distribuido un conjunto de procesos colaboran para conseguir un objetivo común. Cada uno de esos procesos realiza una determinada tarea que sirve para obtener parcialmente el resultado final que se persigue. La cooperación de todos y cada uno de esos procesos da lugar a que el resultado se pueda conseguir. En el caso de las aplicaciones cliente-servidor, el servidor ofrece un determinado servicio (por ejemplo, un recurso compartido) pero por lo general no existe un algoritmo distribuido que ejecutan distintos procesos para obtener la respuesta del servidor.

e) Indique qué tipos de servidores conoce.

Los servidores pueden clasificarse desde distintos puntos de vista.

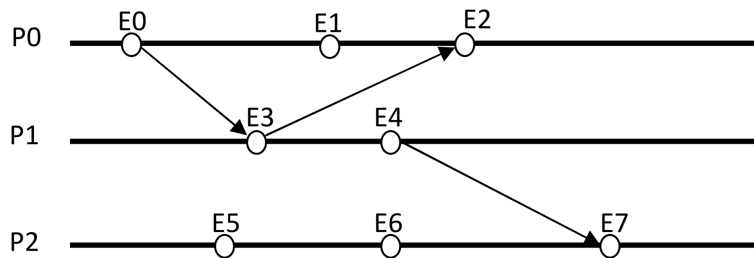
1. En función del número de peticiones que puede atender simultáneamente, hablamos de **servidores secuenciales** cuando sólo una petición puede ser atendida en el mismo momento del tiempo; y **servidores concurrentes** cuando varias peticiones podrían ser procesadas en el mismo instante de tiempo.
2. En función de si el servidor almacena información o no sobre la petición de servicio de un cliente, hablamos de **servidores sin estado** cuando el servidor no almacena ningún tipo de información de las peticiones y por tanto cada una se trata de manera independiente; y **servidores con estado**, aquellos que almacenan información de los clientes. La información de estado se podía dividir en dos tipos: información global (compartida por todos los clientes) e información de petición (específica de cada cliente).
3. En función de si ha de establecerse o no una conexión antes de poder enviar datos al servidor, hablamos de **servidores orientados a conexión** aquellos para los que ha de establecerse una conexión entre el cliente y el servidor antes de poder enviar o recibir datos; y **servidores no orientados a conexión** aquellos que no requieren la fase de conexión.

f) ¿Qué es un servicio de nombres y para qué se utiliza? Indique qué servicios genéricos ofrece un servidor de nombres.

Un servicio de nombres asigna nombres de máquinas a direcciones IP en Internet. Los usuarios suelen manejar nombres de máquina mientras que las aplicaciones manejan direcciones IP. El servidor de nombres ofrece varios servicios:

- Registrar nombre: que permite que los servidores registren la dirección del servicio.
- Resolver el nombre o buscar, que permite obtener la dirección de un determinado servicio.
- Borrar o eliminar un nombre

Ejercicio 2. Considere los procesos P1, P2 y P3 que ejecutan en un sistema distribuido. Estos procesos generan los eventos marcados en la siguiente figura.



Se pide:

a) Extraer las relaciones de causalidad potencial de Lamport entre los eventos mostrados en la figura.

Lamport se basa en dos observaciones muy básicas para establecer su relación de causalidad potencial:

1. En un único procesador, se puede establecer el orden en que ejecutan los procesos.
2. El evento send(m) ocurre antes que el evento receive(m).

La relación de causalidad potencial de Lamport "precede a" (→) usa las dos observaciones anteriores para establecer un orden parcial entre eventos. En la figura, los eventos ordenados según las anteriores observaciones son:

E0 → E1
E1 → E2
E0 → E2 (por la propiedad transitiva)
E0 → E4 (por la propiedad transitiva)
E3 → E4
E5 → E6
E6 → E7
E5 → E7 (por la propiedad transitiva)
E0 → E3
E3 → E2
E4 → E7
E3 → E7
E0 → E7

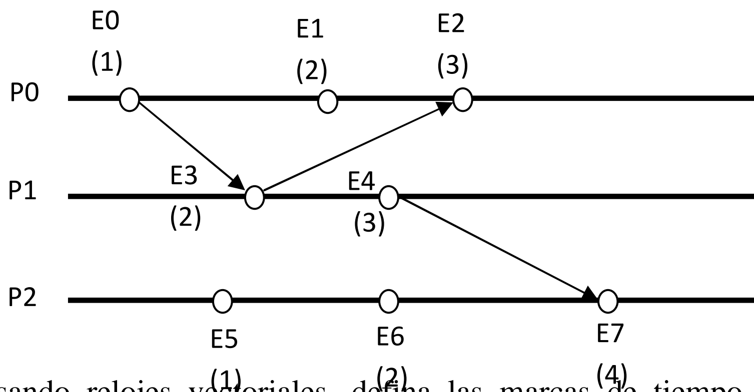
b) ¿Para qué eventos no es posible extraer una relación de orden?¿Por qué?

No es posible establecer una relación de causalidad potencial entre los eventos que no satisfacen cualquiera de las dos observaciones anteriores. Estos eventos se dicen que son concurrentes y se denota como a||b. Los eventos concurrentes en el ejemplo son:

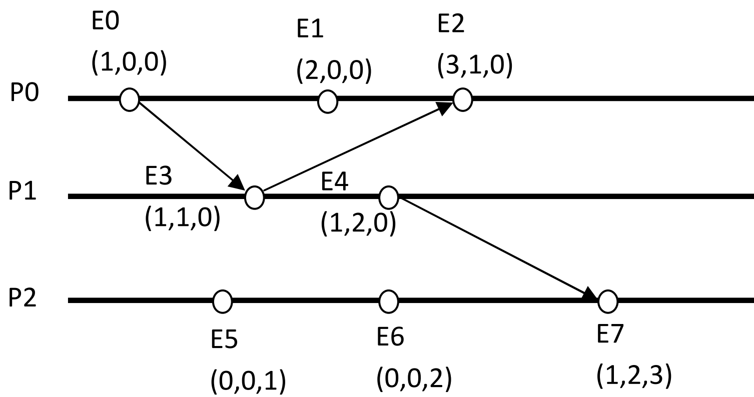
E1 || E3
E1 || E4
E2 || E4
E3 || E5
E3 || E6
E4 || E5
E4 || E6

E0 || E5
E0 || E6
E1 || E5
E1 || E6
E1 || E7
E2 || E5
E2 || E6
E2 || E7

c) Usando los relojes lógicos de Lamport, indique las marcas de tiempo para los eventos de los procesos anteriores.



d) Usando relojes vectoriales, defina las marcas de tiempo para los eventos de los procesos anteriores.



Ejercicio 3. Se quiere desarrollar una aplicación distribuida que permita gestionar la comercialización de entradas de cine a través de un servicio denominado CINEPOLIS. Este servicio permite gestionar las entradas de varios cines. Los clientes pueden utilizar dicho servicio para comprar y cancelar entradas. Asimismo, pueden utilizar este servicio para conocer la próxima hora y fecha a la que se va a proyectar una determinada película en un determinado cine. Se pide:

a) Diseñe la aplicación cliente-servidor anterior utilizando sockets, indicando y especificando todos los aspectos necesarios para su diseño.

En una aplicación cliente-servidor necesitamos tener en cuenta al menos los siguientes aspectos de deseño:

1. Para diseñar una aplicación cliente-servidor necesitamos considerar aspectos comunes a las aplicaciones distribuidas:

- **Nombrado**: necesitamos saber cómo identificar la máquina donde ejecuta la aplicación servidora. Para ello usamos **direcciones IP estáticas** y asumiremos que el cliente conoce la IP (o el nombre) de la máquina servidora.

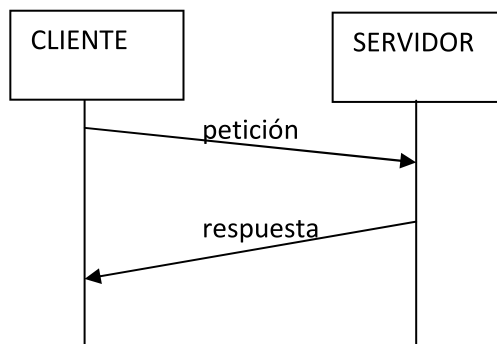
- **Escalabilidad**: para proporcionar un servicio escalable (que siga siendo efectivo mientras el número de peticiones aumenta) vamos a diseñar un **servidor concurrente,** es decir, aquel que puede atender simultáneamente varias peticiones de servicio al mismo tiempo. El servidor concurrente podría ser implementado con procesos convencionales o procesos ligeros, siendo ésta última la opción más eficiente. Dado que vamos a considerar un servidor concurrente necesitamos proteger aquellas variables compartidas que pudieran dar lugar a condiciones de carrera y por tanto llevar a resultados incorrectos en la aplicación (**aspectos de concurrencia y sincronización**).

- **Heterogeneidad:** dado que las máquinas cliente y servidor pueden tener arquitecturas de hardware distintas, para realizar el envío de los datos a la red deben traducirse al estándar de red (network byte order o big endian); análogamente en la recepción los datos de red deben pasarse al formato de host.

2. El protocolo de transporte a utilizar va a depender de los requisitos de **fiabilidad** de la aplicación. Los protocolos de transporte implementados sobre IP son TCP y UDP. Ambas aplicaciones (cliente y servidor) deben estar de acuerdo en el protocolo de transporte a utilizar.
   En particular para esta aplicación, posiblemente enviemos la información de pago de las entradas de cine. Esta información es crítica y es importante que los mensajes lleguen a su destino con cierta fiabilidad (sin pérdidas, ordenados, sin duplicados). Por tanto, seleccionamos **TCP** como protocolo de transporte a usar en la aplicación**.** No se necesita implementar aspectos de **QoS.**

3. El protocolo de servicio:
   El protocolo de servicio define el intercambio de mensajes entre la aplicación cliente y servidor y además el formato de los mensajes.

Para cada servicio (comprar y cancelar entradas) vamos a definir dos mensajes: el primero de ellos lo envía el cliente al servidor, y contiene la petición para realizar la compra (o cancelación) de las entradas; el segundo de ellos lo envía el servidor al cliente con la respuesta.



Existen tres servicios que debe ofrecer el servidor:

- Comprar entradas
- Cancelar entradas
- Conocer próximo pase

Existen por tanto tres tipos de peticiones que se pueden solicitar. Para cada petición se va a establecer una conexión, se va a enviar los argumentos de la petición y se va a recibir el resultado. Posteriormente se cierra la conexión.

Para poder identificar cada petición, se puede utilizar un byte que se envía del cliente al servidor al establecer la conexión. Se puede utilizar 0 para comprar entradas, 1 para cancelar entrada y 2 para conocer el próximo pase de una película.

En el caso de comprar entradas, es necesario a continuación enviar los siguientes datos:

- Identificador de cine: entero de 32 bits en formato big endian.
- Nombre de la película: cadena de caracteres de hasta un máximo determinado. O bien se envía un entero en formato big endian indicando el número de caracteres de la película y a continuación el nombre de la película.
- Fecha: Se pueden utilizar diversos formatos, por ejemplo dia/mes/año, siendo día un byte, mes otro y año un entero en formato big endian.
- Numero de entradas: se puede utilizar un byte para este campo.
- Tarjeta de pago: cadena de caracteres con el número de la tarjeta a utilizar parar comprar.
- Titular de la tarjera: Cadena de caracteres del titular (de un determinado tamaño).
- Fecha de caducidad: mes/año

El mensaje de respuesta a esta petición podría ser:

- Código de venta: entero en formato big endian.
- Número de sala: un byte
- Lisa de butacas: una lista con el número de butacas reservadas (cada una un entero en formato big endian).

Se puede considerar que si el código de venta es negativo, entonces la compra no se pudo realizar.

La petición de cancelación de entradas contendrá el código de venta que se enviará despúes de enviar el código de la petición (un 1)

La respuesta en este caso puede ser un byte indicando error o éxito.

Para conocer el próximo pase es necesario enviar el nombre de la película, el identificador del cine. Se devolverá una hora y una fecha correspondiente al próximo pase. Para la hora se puede utilizar el formato una cadena con formato: HH:MM:SS. Para la fecha se puede enviar un byte para el día, un byte para el mes (1 enero, 12 diciembre), para el año un entero en formato big endian. El envío del mes en formato número, permite que el cliente se puede adaptar a distintos idiomas.

4. Opciones en el servidor:

El servidor será **orientado a conexión** dado que el protocolo de transporte que vamos a utilizar es TCP. Esto implica que deberá existir un establecimiento de la conexión entre el cliente y servidor previo al intercambio de los datos.

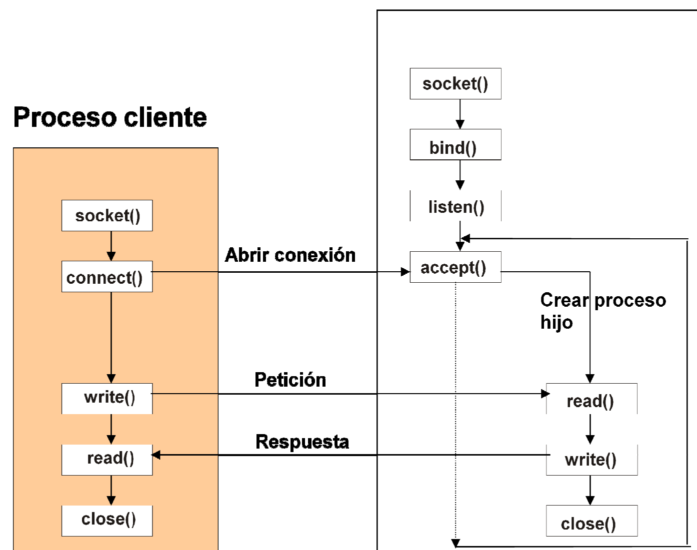
Además, el servidor será **con estado** dado que debemos mantener **información global** de las peticiones de servicio de los clientes. Por ejemplo, el servidor deberá controlar si las entradas para una película se han agotado, y devolver el correspondiente error si se intenta comprar más entradas para esa película. Además deberemos mantener la **los códigos de compra** para poder realizar una cancelación posterior.

Una vez definido el protocolo de servicio y los aspectos de diseño a considerar, las funciones a realizar en la aplicación cliente y en la servidora son las siguientes:

Funciones en la parte del cliente: conectar al servidor utilizando sockets, enviar la petición y esperar la respuesta.
Funciones en la parte del servidor: esperar la conexión de peticiones de servicio de los clientes, calcular el resultado y devolver la respuesta.

b) De acuerdo al diseño anterior, indique qué llamadas a la biblioteca de sockets utilizaría en el cliente y en el servidor y en qué orden.

El modelo de comunicación TCP es el siguiente:



A continuación se detallan las llamadas a la biblioteca de sockets:

En el cliente:

1. int **socket**(int dominio, int tipo, int protocolo)
   donde dominio es AF_INET, tipo SOCK_STREAM, protocolo es 0 (elige el S.O).

2. int **connect**(int socket, struct sockaddr *dir, int long)
   donde socket es el socket devuelto por socket, dir es la dirección del socket remoto y long es la longitud de la dirección.
   Esta primitiva establece la conexión con el servidor.

3. int **write**(int sd, char *buffer, int long);
   donde sd es el socket devuelto por socket, buffer es un puntero a los datos a enviar y long es el tamaño de los datos.

4. int **read**(int sd, char *buffer, int long);
   donde sd es el socket devuelto por socket, buffer es un puntero a los datos a recibir y long es el tamaño de los datos a recibir.
5. int **close**(int sd)

En el servidor:

1. int **socket**(int dominio, int tipo, int protocolo)
   donde dominio es AF_INET, tipo SOCK_STREAM, protocolo es 0 (elige el S.O).

2. int **bind**(int sd, struct sockaddr *dir, int long)
   donde sd es el socket devuelto por socket, dir es la dirección del socket y long es el tamaño de la dirección.
   Habilita el socket para poder recibir conexiones entrantes.

3. int **listen**(int sd, int baklog)
   donde sd es el socket devuelto por socket, y backlog es el número de peticiones que se pueden encolar antes de que el servidor haga **accept.**

4. int **accept**(int sd, struct sockaddr *dir, int *long)
   donde sd es el socket devuelto por socket, **dir** es la dirección del cliente que ha realizado **connect** y **long** es el tamaño de la dirección.

6. int **read**(int sd, char *buffer, int long);
   donde sd es el socket devuelto por socket, buffer es un puntero a los datos a recibir y long es el tamaño de los datos a recibir.

7. int **write**(int sd, char *buffer, int long);
   donde sd es el socket devuelto por socket, buffer es un puntero a los datos a enviar y long es el tamaño de los datos.

8. int **close**(int sd)
   donde sd es el socket devuelto por socket.

c) Considerando que se emplean RPC con un lenguaje de definición de interfaces similar a la sintaxis del lenguaje C, defina la interfaz correspondiente al servicio descrito. Indique qué parámetros son de entrada y cuáles son de salida.

La interfaz debería definir al menos los procedimientos para comprar y cancelar entradas. Dado que la interfaz se ha de definir en C una posible especificación puede ser la siguiente:

```
#define MAX_CUENTA 20
#define MAX_FECHA  10
#define MAX_HORA    8
#define MAX_PELICULA     1000
#define MAX_BUTACAS      10



struct peticion_comprar{
   int                  id_cine;
   char                 película[MAX_PELICULA];
   char                 fecha[MAX_FECHA];
   int                  numero_entradas;
   char                 cuenta_bancaria[MAX_CUENTA];
};

struct respuesta_comprar{
   int                  código_compra;
   int                  id_sala;
   int                  lista_butacas[MAX_BUTACAS];
};

struct peticion_cancelar{
   int                  codigo_compra;
};

struct próximo_pase {
   char            fecha[MAX_FECHA];
    char           hora[MAX_HORA];
};
```

```
struct respuesta_compra comprar_entradas(struct peticion_comprar peticion);

int cancelar_entradas(struct peticion_cancelar peticion);

struct próximo_pase Obtener_proximo_pase(char *película, int id_cine);
```

d) Especifique la misma interfaz utilizando las RPC de Sun.

```
const MAX_FECHA=10;       /* Formato:dd/mm/yyyy */
const MAX_CUENTA=20;
const MAX_PELICULA=100;
const MAX_BUTACAS=10;


struct peticion_comprar{
   int                     id_cine;
   string                  película[MAX_PELICULA];
   string                  fecha[MAX_FECHA];
   int                     numero_entradas;
   string                  cuenta_bancaria[MAX_CUENTA];
};

struct peticion_cancelar{
   int                     codigo_compra;
};

struct respuesta_comprar{
   int                     id_cliente;
   int                     id_sala;
   int                     lista_butacas[MAX_BUTACAS];
};


struct próximo_pase {
   char              fecha[MAX_FECHA];
    char             hora[MAX_HORA];
};


program CINEPOLIS_PRGM {
   version CINEPOLIS_VER {
         respuesta_comprar comprar_entradas(struct peticion_comprar) = 1;
         int cancelar_entradas(struct peticion_cancelar) = 2;
   } = 2;

         struct próximo_pase Obtener_proximo_pase(char película<>, int id_cine) = 3;

} = 100000;
```

e) Si se emplean las RPC de Sun, indique cuáles son los pasos que debe realizar el cliente de RPC para invocar un procedimiento remoto.

Para invocar un procedimiento remoto el cliente de RPC debe:
1.- Obtener un manejador del cliente mediante la llamada a **clnt_create**. Este manejador permite identificar el servicio remoto, versión y protocolo de transporte usado en la comunicación.
2.- Invocar el procedimiento remoto como si de un procedimiento local se tratase.
3.- Destruir el manejador una vez que el cliente no quiera seguir invocando más procedimientos, mediante la llamada **clnt_destroy.**

f) Indique cuáles son las tareas que tienen que realizar el stub del cliente y del servidor.

El stub del cliente debe proporcionar para cada procedimiento remoto:

1) Localización del servicio remoto
2) Empaquetar un mensaje con los argumentos (marshalling)
3) Enviar el mensaje
4) Recibir la respuesta del servidor
5) Desempaquetar la respuesta (unmarshalling)
6) Retornar al cliente

El stub del servidor debe:

1) Registrar el servicio remoto
2) Recibir la petición de procedimiento remoto
3) Desempaquetar la petición (unmarshalling)
4) Invocar el procedimiento localmente
5) Obtener la respuesta del procedimiento
6) Empaquetar la respuesta (marshalling)
7) Enviar la respuesta al cliente
8) Volver a (2)