💯

Distri 1er Parcial 100/100

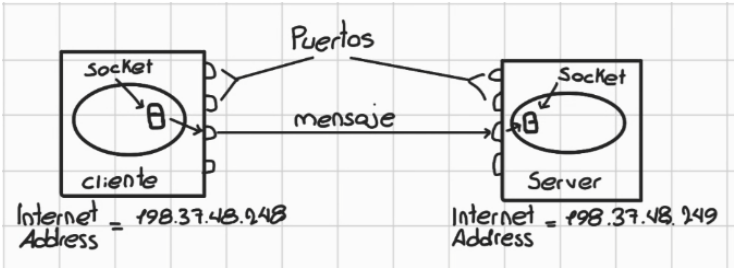
Explique brevemente:

El desafío de Heterogeneidad de los sistemas distribuidos y un ejemplo concreto.

La heterogeneidad se refiere a la capacidad de permitir que varios usuarios accedan a servicios y ejecuten aplicaciones sobre un conjunto variado y diferenciado de redes y computadores. Un ejemplo sería un servidor Windows que se comunica con dispositivos móviles Android, ambos conectados a través de una red heterogénea.

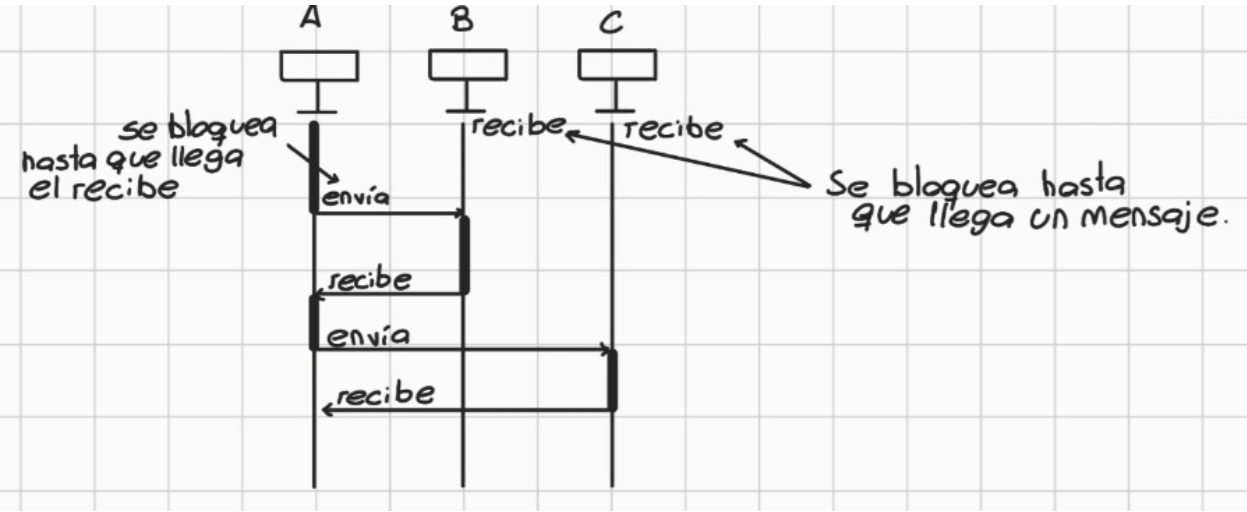
Socket. Incluir gráfico.

Un socket es un conector que proporciona un punto extremo de la comunicación entre procesos, cada socket está asociado con un protocolo particular, ya sea UDP o TCP.



Explique que es la comunicación síncrona y realice un diagrama de secuencia de una comunicación síncrona entre 3 actores o componentes.

La comunicación síncrona es aquella en la que los procesos receptor y emisor se sincronizan con cada mensaje. En este caso, las operaciones *envía* y *recibe* son bloqueantes.



Características de los Sistemas Distribuidos

Concurrencia: en un sistema distribuido, la ejecución de programas concurrentes es la norma. La coordinación de programas que comparten recursos y se ejecutan de forma concurrente es un tema importante.

Inexistencia de Reloj Global: La sincronización de los programas se basan en una idea compartida del tiempo. No existe una única noción global del tiempo correcto.

Fallos independientes: Todos los sistemas pueden fallar, los fallos pueden ser fallos en la red o la terminación inesperada de un programa (crash).

El propósito de los servidores proxy es incrementar la **disponibilidad** y **prestaciones** del servicio.

Un modelo **fundamental** define la arquitectura distribuida en términos de conceptos abstractos con el fin de simplificar los componentes individuales de dicho sistema.

El término **código móvil** se emplea para referirse al código que puede ser enviado desde un computador a otro y ejecutarse en este.

Se dice que un sistema es **escalable** si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y el número de usuarios.

El nivel de hardware y las capas más bajas de software se denominan **plataforma** para sistemas distribuidos y aplicaciones.

La **encriptación** y la **autenticación** se emplean para construir canales seguros en forma de capa de servicio sobre los servicios de comunicación existentes.

Si un proceso para y permanece parado, y otros procesos pueden no ser capaces de detectar este estado, estamos hablando de la clase de fallo **fracaso o crash** que afecta a un proceso.

El protocolo TCP utiliza **streams** para la transmisión de datos, mientras que el protocolo UDP utiliza **datagramas**.

Uno de los motivos principales para construir sistemas distribuidos es el de **compartir recursos**

Los sistemas distribuidos pueden fallar de formas diferentes, en general se engloban cómo “fallos independientes” a los **fallos de red o paradas repentinas (crashes)**

El protocolo DNS, está implementado sobre el protocolo **UDP** de la capa de transporte.

La clase java que representa un paquete UDP es **DatagramPacket**

Tres aspectos sobre los cuales el middleware ofrece transparencia. Transparencia de Ubicación, Lenguajes de Implementación, Plataforma

- La tasa de producción de un sistema o throughput se refiere al tiempo máximo que un proceso puede esperar antes de continuar sus operaciones.

  Falso.

- Cuando un proceso o canal trasmite mensajes arbitrarios (inexistentes) en momentos no determinados, decimos que ocurre un fallo bizantino.

  Verdadero.

- La única forma de comunicación entre 2 procesos que se ejecutan en una misma máquina es mediante el uso de socket.

- Un sistema de tipo sala de chat en un sitio web puede ser implementado utilizando comunicación síncrona.

- La estrategia para detectar un fallo por omisión de procesos, es decir, detectar la ruptura de un proceso, es el uso de búferes

  Falso, uso de timeouts.

- Los patrones de arquitectura reflejan la estructura de un sistema distribuido junto con sus roles y responsabilidades

  Verdadero.

- Utilizar hilos es una ventaja en el desarrollo de Sockets al momento de realizar una invocación a la primitiva receive.

  Verdadero.

- El problema con la sincronización de los sistemas distribuidos síncronos es que no podemos determinar un límite para el retardo de los mensajes.

  Falso, si se establece un límite.

- La única forma de comunicación entre 2 procesos que se ejecutan en una misma máquina es mediante el uso de socket.

- Los canales seguros son una estrategia para mitigar amenazas en el modelo de fallo.

  Falso, modelo de seguridad

- Un modelo arquitectónico define las perspectivas abstractas con el fin de examinar los aspectos individuales de un sistema distribuido.

  Falso, modelo Fundamental.

- Existen limites de precisión en computadores de una red para sincronizar sus relojes y este es el motivo por el cual la Concurrencia es una característica de los sistemas distribuidos.

  Falso, la característica es la Inexistencia de reloj global.

- Una middleware se define como un conjunto de servicios de aplicaciones y almacenamiento basados en Internet suficientes para soportar la mayoría de las necesidades de los usuarios, lo que les permite activar mayor o totalmente el almacenamiento de datos local y el software de aplicación.

  Falso, cloud computing.

- Uno de los desafíos más importantes de los sistemas distribuidos contemporáneos es la calidad de servicio.

  Falso, es un requisito.

- Los applets son un ejemplo clásico de agente móvil, ya que el código se descarga desde el servidor y se ejecuta en el cliente

  Falso, código móvil.

- La plataforma de un modelo arquitectónico está formada por el sistema operativo y el middleware.

  Falso, sistema operativo y hardware.

- El protocolo UDP fue diseñado para distribuir el tiempo UTC en Internet.

- Los relojes de un sistema distribuido pueden sincronizarse a fuentes de tiempos obtenidas desde los relojes atómicos.

  Falso, es imposible sincronizar los relojes locales.

- Para garantizar la protección de objetos en el modelo de seguridad, se utiliza el concepto de autoridad (principal), el cual asegura que la petición o la respuesta proviene del servidor auténtico.

  Verdadero.

- Los bloqueos son una estrategia para obtener solapamiento secuencial de operaciones conflictivas entre dos o más transacciones que acceden a un objeto compartido.

- Una de las formas de comunicación entre procesos que se ejecutan en una misma máquina es mediante el uso de socket.

- Un servicio distribuido puede proveerse desde un único proceso servidor.

  Falso, puede proveerse desde múltiples servidores.

UDP Server

```
public class UPDServer {
    public static void main(){

        int puertoServidor = 9876;

        //crea un socket de servidor en el puerto 9876
        DatagramSocket serverSocket = new DatagramSocket(puertoS

        //se crean bufers
        byte[] receiveData = new bite[1024];
        byte[] sendData = new bite[1024];

        //ciclo infinito para escuchar continuamente
        while (True){
            receiveData = new bite[1024];
            //crea un paquete para recibir los datos que envia e
            //en receiveData
            DatagramPacket receivePacket = new DatagramPacket(re

            //esperando algun cliente
            //receive bloqueante
            serverSocket.receive(receivePacket);

            //cuando recibe los datos
            String datoRecibido = new String (receivePacket.getD
            datoRecibido = datoRecibido.trim();

            InetAddress IPAddress = receivePacket.getAddress();
```

```
            int port = receivePacket.getPort();

            /* operaciones con el dato */

            //enviar respuesta no bloqueante
            sendData = //operacion que se haya hecho con el dat
            //crea un paquete udp que contiene los datos que se
            DatagramPacket sendPacket = new DatagramPacket(sendD

            //envia el paquete
            serverSocket.send(sendPacket);
        }
    }
}
```

UDP Client

```
public class UDPClient{
    public static void main(){

        String direccionServidor = "127.0.0.1";
        int puertoServidor = 9876;

        //crea socket parra el cliente
        DatagramSocket clientSocket = new DatagramSocket();
      InetAddress IPAddress = InetAddress.getByName(direccionSer

        //buffer para enviar los datos alservidor
        byte[] sendData = new byte[1024];
        //buffer para recibir los datos del servidor
        byte[] receiveData = new bite[1024];

        /*pide datos a enviar*/
```

```
        sendData = //cargar los datos a enviar
        //crear un paquete upd que contiene los datos que se env
        DatagramPacket sendPacket = new DatagramPacket(sendData,

        //enviar el paquete udp a traves de clientSocket
        clientSocket.send(sendPacket);

        //preparar un paquete udp para recibir la respuesta del
        DatagramPacket receivePacket = new DatagramPacket(recei
        //establecemos timeout
        clientSocket.setSoTimeout(1000);

        try{
            //esperamos receive, bloqueante
            clientSocket.receive(receivePacket);
            //llega respuesta
            String respuesta = new String(receivePacket.getData(
            InetAddress returnIPAddress = new InetAddress(recei
            int port = receivePacket.getPort();
        } catch (SocketTimeoutException) {
                System.out.printLn("El paquete UDP se asume perd
        }
        clientSocket.close();
    }
}
```

TCPServer

```
class TCPServer{
    public static void main(){

        //crea un socket en el puerto 4444
```

```
        ServerSocket serverSocket = new ServerSocket(4444);

        //crea un socket para el cliente
        Socket clientSocket = null;
        clientSocket = serverSocket.accept():

        //print writer para enviar datos al cliente a traves del
        PrintWriter out = new PrintWriter(clientSocket.getOutput
        //buffered reader para leer los datos que envia el clien
        BufferedReader in = new BufferedReader(new InputStreamRe

        //variables para el manejo de entrada y salida de mensaj
        String inputLine, outputLine;
        while (true){
            inputLine = in.readline();   //lee el mensaje que en
            System.out.printLn("mensaje recibido: " + inputLine)

            outputLine = "Respuesta igual al recibido: " + input
            out.printLn(outputLine);   //envia la respuesta al c
        }

        out.close();  //cierra el fluho de salida al cliente
        in.close();   //cierra el fluho de entrada desde el clie
        clientSocket.close();   //cierra el socket del cliente
        serverSocket.close();   //cierra el socket del server
    }
}
```

TCPClient

```
class TCPClient{
    public static void main(){

        //crea un socket y conecta al servidor localhost puerto
```

```
        Socket unSocket = new Socket("localhost", 4444);

        //para enviar datos al server a traves del socket
        PrintWriter out = new PrintWriter(unSocket.getOutputStre
        //para leer los datos que el server envia a traves del s
        BufferedReader in = new BufferedReader(new InputStreamRe

        //para leer la entrada del usuario desde consola
        BufferedReader stdIn = new BufferedReader(new InputStrea

        String fromServer;    //almacena los mensajes que vienen
        String fromUser;      //almacena los mensajes que envia

        //mientras el server envie cosas
        while ((fromServer = in.readLine) != null){
            System.out.printLn("Servidor: " + fromServer);  //im

            fromUser = stdIn.readLine();    //lee lo que el usua
            if (fromUser != null){          //si el usuaio escri
                System.out.printLn("Cliente: " + fromUser);    /
                //envia al server lo que ha escrito
                out.printLn(fromUser);
            }
        }

        out.close();  //cierra flujo de salida
        in.close();   //cierra flujo de entrada
        stdIn.close();    //cierra entrada por teclado
        unSocket.close(); //cierra el socket, cortando la comuni
    }
}
```