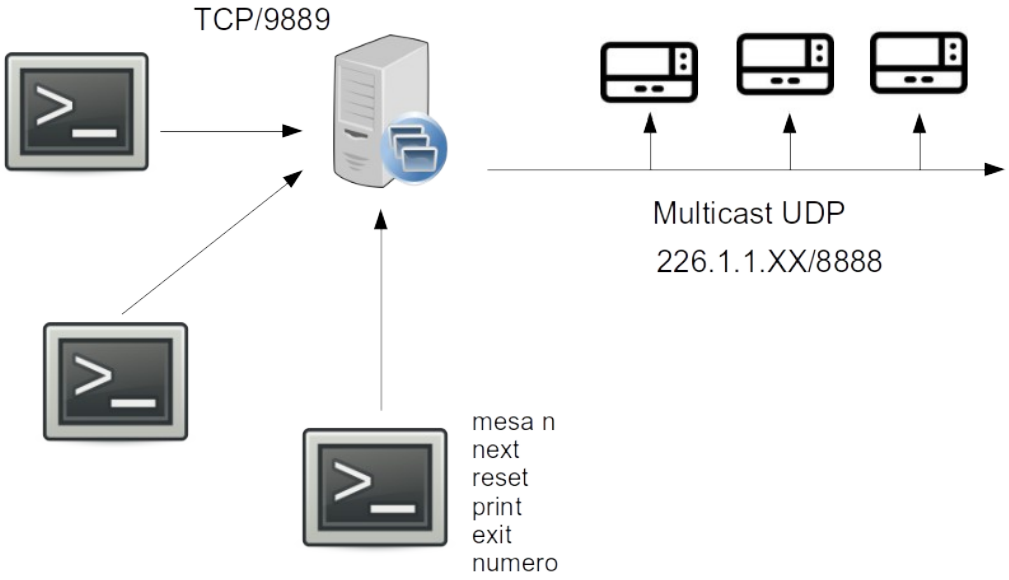
Ejercicio de valoración de conocimientos. Segunda Evaluación 2021
2º DAMM. Programación de Servicios y Procesos

En la IES Ingeniero de la Cierva se desea implementar en un **Servicio de Atención al Alumno Numérico (SATAN)**, donde un conjunto funcionarios van atendiendo a los alumnos, cuando un alumno ha terminado la gestión con el funcionario, este solicita que pase el siguiente alumno. La información con el número del siguiente alumno se visualiza en unas pantallas dispuestas a tal efecto. Así, cuando un alumno necesita realizar un trámite en la secretaría del IES, solicita un número en una pantalla preparada para emitir números, y cuando sale su número se puede dirigir a la ventanilla. La información del número que tiene que acceder y la ventanilla asignada se transmite a las pantallas de información ediante un protocolo de **multicast UDP.** Para ello, un servidor será el encargado de enviar los mensajes a todas las estaciones que estén a la escucha mediante este protocolo. De esta forma, el servidor se despreocupa de las estaciones receptoras. Mientras que los clientes (mesas de trámites) y pantalla de solicitud de número se comunican con el servidor mediante un **protocolo TCP**, al que le y le van enviando los comandos.
Para simplificar el funcionamiento, la pantalla de solicitud de números y la de los funcionarios es el mismo servicio pero implementan comandos diferentes.
El esquema del servicio es el siguiente.



Para **enviar los mensaje**s a las pantallas receptoras un funicionario podrá **conectarse** al servidor mediante el protocolo TCP en el **puerto 9889**. El emisor dispondrá de un conjunto de comandos para comunicar con el servidor. Estos comandos son los siguientes:

- **mesa <número_mesa>**. Este comando asigna al terminal un número de mesa (ventanilla), siempre que dicha mesa esté libre. Además, la mesa 0 no existe y por lo tanto no puede ser asignada. Si el comando **mesa i** se procesa correctamente el sistema informará al funcionario con el mensaje "ok -> asignado a mesa i". Mientras que no asignemos un número de mesa al funcionario, no podemos ejecutar el comando **next**.
- **reset.** El comando libera la mesa asignada a un terminal, de forma que podría asignarse desde otro terminal.
- **next**. Envía por UDP multicast el mensaje "**Mesa i número p"**, por ejemplo Mesa 1 número 3, para informar a las personas que están a la espera. También debe incrementar el contador. En el terminal se imprime "ok -> Mesa i número p", por ejemplo "ok -> Mesa 1 número 2", para que el

funcionario también sepa el número de orden que toca. Si no tenemos a nadie en espera debe mostrar en el terminal el mensaje "NADIE EN ESPERA"

- **print**. Imprime en la consola del servidor y en el terminal del usuario la asignación de las mesas y el número de orden que toca.
- **exit**. Este comando desconecta la mesa del servidor de mensajería y libera la mesa.
- **numero**. Solicita un número y nos muestra el número que nos toca.
- **start**. Borra todos los contadores.
- Cuando nos conectamos el sistema nos da la bienvenida con el siguiente mensaje: H**ola estás conectado a SATAN, indica la mesa antes de operar**

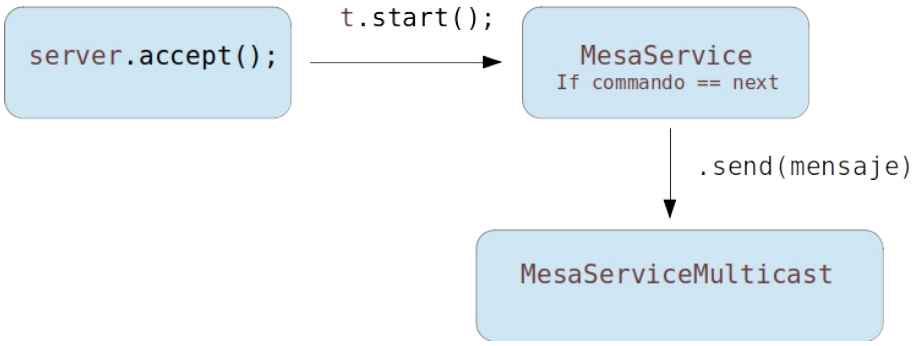
Dato que el protocolo UDP no es confiable, cuando el servidor **remita un mensaje a las pantallas calculará un hash que será emitido junto con el mensaje** utilizando el formato:

<mensaje>@<hash>

Las pantallas, calcularán el hash e imprimirán junto con el mensaje propuesto anteriormente un ok. Por ejemplo "Mesa 1 número 3: Ok"

El funcionamiento básico del servidor será similar a los ejemplos vistos en clase. El servidor estará a la espera de las conexiones entrantes. Una vez que una mesa se conecte, se iniciará un hilo encargado de gestionar el servicio de comunicaciones con la mesa [Clase **MesaService**] ya que la gestión de las comunicaciones se realizará mediante programación multihilo. Cuando esta clase **MesaService** procese el comando **next** le enviará a la clase **MCastSender** (otro hilo) el mensaje pertinente. Este servicio será el encargado de calcular el hash y la emisión por UDP Multicast del mensaje, además mientras se está procesando y emitiendo un mensaje, **se debe impedir** que otro hilo **MesaService** realice una operación de emisión:

Básicamente el funcionamiento del servidor es como se describe en la siguiente imagen:



NO ES NECESARIO IMPLEMENTAR LA APLICACIÓN DESDE CERO. Para ello podemos contar con parte del código de las prácticas propuestas para su realización. También necesitaremos utilizar el código del terminal gráfico y de las las pantallas.

Por ello vamos a dividir nuestro trabajo en tareas:

1ª Tarea. Crear el servidor **CitasServer** para que funcione en multihilo, iniciando una instancia de la clase **MesaService** cada vez que reciba una conexión de entrada.

2ª Tarea. Implementar los comandos de control del servidor: **mesa i | next | reset| numero | print | start | exit**.

3ª Tarea. Añadir a la clase **ContadorMesa** los métodos que nos falten para la

funcionalidad de los comandos anteriores y establecer un **mecanismo de control concurrente** para evitar que más de un hilo acceda simultáneamente a modificar los valores de las variables contadorMesas, peticionMesas y asignaMesa[]. Donde:
- contadorAlumno tiene el número de orden del último alumno que está siendo atendido
- peticionAlumno, tiene el número de peticiones de atención que se han solicitado.
- asignaMesas[] nos indica si una mesa ha sido asignada a un funcionario o está libre.

4ª Tarea. Programar el método **send(mensaje)** de la clase **MCastSender** de forma que calcule el hash y emita el mensaje pertinente, debe controlar la concurrencia.

5ª Tarea. Crear una estación gráfica receptora multicast que reciba el mensaje calcule el hash y lo valide.

6ª Tarea. Crear una estación gráfica para solicitar un número.

Las clases que debemos tener en el **servidor** son las siguientes:

- **MainServer** Clase principal del servidor que acepta las conexiones TCP e inicia la clase **MesaService** en multihilo.
- **MesaService** Clase encargada de gestionar la comunicaciones con la mesa-emisor, debe gestionar los comandos.
- **MCastSender** Clase encargada de emitir los mensajes por UDP multicast y calcular el hash.
- **ContadorMesas** Clase encargada de los contadores. Debemos añadirle el control de concurrencia y algún método que nos falte.

L**a dirección de multicast es la 226.0.1.<puesto>**, por ejemplo para el alumno 10 su dirección de red será 226.0.1.10

Criterios de evaluación:

1. Las clase **MesaService** se ha implementado adecuadamente, realizando la función sugerida y los parámetros de la clase son coherentes con su funcionalidad. (3 puntos)

2. Las clase **MCastSender** se ha implementado adecuadamente, realizando correctamente la función sugerida y los parámetros de la clase son coherentes con su funcionalidad y se ha gestionado correctamente el control de concurrencia y el cálculo del hash. (2 puntos)

3. El funcionamiento de la clase **ContadorMesas** se ha realizado correctamente contemplando los problemas de concurrencia propuestos. (1 puntos)

4. El cliente **PantallaUDPMulticast** se ha realizado correctamente validando el hash recibido (2 puntos)

6. La aplicación presenta una lógica coherente, funciona correctamente y está comentada adecuadamente. (2 puntos)

5. Se ha implementado el cliente para solicitar número (**TerminalPeticiones**) y el del terminal del funcionario (**TerminalMesaFuncionario**) (Opcional)

Entrega del ejercicio

Se utilizará la especificación 8 de java (java 1.8)

El código estará bien **formateado y sangrado**, y con comentarios pertinentes, no es necesario escribir el Quijote. Comentarios al estilo **ContadorMesas**

El ejercicio deberá ser entregado en un archivo comprimido con el nombre y apellidos. Los **proyectos deberá nombrarse con apellidos y nombre y entregarse completos,** no únicamente los fuentes, pera poder ser importado directamente a eclipse. **PantallaUDPMulticast, TerminalPeticiones y TerminalMesaFuncionario**

Clase **ContadorMesas**

```
//Esta clase debe ser dotada de un control de concurrencia
public class ContadorMesas {
	//Número máximo de mesas asignadas
	int maxMesas;
	//Contador que almacena el número del alumno que está siendo atendido
	int contadorAlumno;
	//Contador que almacena el número de solicitudes de atención generadas
	int peticionAlumno;
	//Array que indica la asignación de un terminal de funcionario a una mesa de atención
	int[] asignaMesaTerminal;


	public ContadorMesas(int maxMesas) {
		this.maxMesas = maxMesas;
		this.contadorAlumno = 0;
		this.peticionAlumno = 0;
		this.asignaMesaTerminal = new int[maxMesas];

		//inicilizamos todas las asignaciones
		for (int i=0;i < maxMesas;i++) {
			this.asignaMesaTerminal[i]=0;
		}

	}//end-constructor

	//Método para asignar un puesto a una mesa
	//Si la asignación es posible retorna true
	//Se supone que un puesto es el terminal del funcionario
	//y la mesa en el número de la ventanilla por la que atiende
	//un funcionario puede cambiarse de ventanilla (mesa) pero su terminal (puesto)
	//puede ser el mismo

	public boolean asignaMesa(int mesaAsignada, int puesto) {
		boolean result = false;
		if (asignaMesaTerminal[mesaAsignada] == 0) {
			asignaMesaTerminal[mesaAsignada] = puesto;
			result = true;
		}

		return result;
	}
```

```
	//Método que retorna el número del siguiente alumno que debe ser atendido
	public int nextContador() {
		int nextNumero = 0;

		//si tenemos alumnos pendientes que atender
		if (contadorAlumno < peticionAlumno) {
			contadorAlumno++;
			nextNumero = contadorAlumno;
		}
		return nextNumero;
	}

	//Método que solicita un número de orden para ser atendido
	public int nextAlumno() {
		int nextPeticion = 0;

		//pedimos ser atendidos
		peticionAlumno++;
		nextPeticion = peticionAlumno;

		return nextPeticion;
	}

	public String getAsignacion(int i) {
		String mensa = "Mesa "+i+" asignada al terminal
"+asignaMesaTerminal[i];
		return mensa;
	}

	//borra la asignación de una ventanilla (mesa) a un terminal de
funcionario
	public void clearAsignacion(int i) {
		asignaMesaTerminal[i] = 0;
	}


}//end-class
```