Sistemas Operativos II

Profesor

Carlos Mendez Rodriguez

Estudiante:

Javier Díaz Mora / 20200120139

gRCP

gRPC (gRPC Remote Procedure Calls) es un framework de código abierto desarrollado por Google que permite la comunicación entre aplicaciones de manera eficiente y escalable.

Conceptos Básicos

- ➢ gRPC se basa en el concepto de llamadas a procedimientos remotos (Remote Procedure Calls), donde un programa puede invocar funciones de otro programa ubicado en una máquina diferente como si fueran funciones locales.
- ➢ Utiliza Protocol Buffers (protobuf) para definir la estructura de los datos y los servicios. Protobuf es un lenguaje neutral, de plataforma independiente y eficiente para la serialización de datos estructurados.

Características Principales

- ➢ gRPC soporta una amplia variedad de lenguajes de programación, como Java, C++, Python, Go, Ruby, C#, JavaScript, y más.
- ➢ Permite comunicación bidireccional y soporta varios tipos de comunicación como:
    - **Unary RPC:** El cliente envía una sola solicitud al servidor y recibe una única respuesta.
    - **Server streaming RPC:** El cliente envía una solicitud y recibe una secuencia de respuestas del servidor.
    - **Client streaming RPC:** El cliente envía una secuencia de solicitudes y recibe una única respuesta del servidor.
    - **Bidirectional streaming RPC:** Tanto el cliente como el servidor envían una secuencia de mensajes utilizando un flujo de lectura y escritura.

Ventajas

- ➢ gRPC es muy eficiente en términos de rendimiento gracias a su uso de HTTP/2, que permite multiplexación de solicitudes/respuestas, reducción del tamaño de los encabezados y mejor compresión de datos.
- ➢ A partir de los archivos .proto, gRPC puede generar automáticamente el código necesario para el cliente y el servidor en diferentes lenguajes de programación.
- ➢ Utiliza TLS (Transport Layer Security) para la encriptación de los datos en tránsito, proporcionando una comunicación segura entre los servicios.

Casos de Uso

- gRPC es ideal para arquitecturas de microservicios debido a su eficiencia, escalabilidad y capacidad de definir interfaces de servicio de manera explícita.
- Gracias a su bajo consumo de ancho de banda y la rapidez en las comunicaciones, es útil en el desarrollo de aplicaciones móviles y web.
- Permite la comunicación entre componentes escritos en diferentes lenguajes de programación, facilitando la interoperabilidad en sistemas heterogéneos.

Desventajas

- Puede ser más complejo de aprender y configurar en comparación con otros métodos de comunicación como REST.
- La necesidad de definir los servicios y mensajes en archivos .proto y generar el código puede añadir complejidad al ciclo de desarrollo.