MEMORIA - PRÁCTICA 1

-Sistemas distribuidos-

—

Jorge Leris Lacort - 845647

Andrei Gabriel Vlasceanu - 839756

—

Septiembre de 2023

ÍNDICE

Introducción

En esta práctica se ha realizado una iniciación al lenguaje de programación golang mediante la elaboración y análisis de 4 diferentes arquitecturas para crear sistemas distribuidos que resuelvan un problema de manera más eficiente y efectiva.

Descripción del problema y aplicación

El problema que se trata en esta práctica, se resuelve con una aplicación sencilla la cual se encarga de calcular una serie de números primos para un intervalo determinado, estos procesos suelen tener un coste bastante elevado por eso se plantea este problema junto al desarrollo de las diferentes arquitecturas distribuidas, para que así diferentes clientes puedan contribuir en los cálculos de los números primos y por consiguiente solucionar dicho problema de una manera más eficiente.

Recursos computacionales disponibles

Para la realización de las prácticas a lo largo de la asignatura disponemos de un servidor de raspberrys, de las cuales nuestro grupo tiene acceso del puerto inicial 29120 al puerto final 29129 y la máquina inicial la 9 y la final la 12.
Estas máquinas disponen de 32 GB de almacenamiento y un procesador del cual, lo que más destaca, es que tiene 6 núcleos, de manera que nos permite realizar 6 procesos concurrentemente.

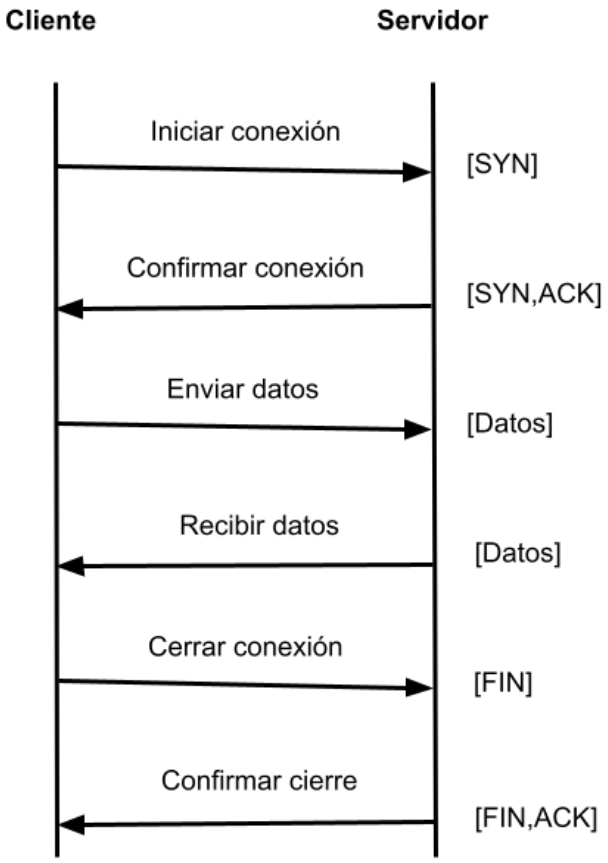
Diseño de las arquitecturas

El diseño de cada una de las arquitecturas se ha realizado de forma independiente, realizando cada una en ficheros separados los cuales se encuentran junto a la entrega, a continuación se va a pasar a mostrar los principales pasos seguidos y los diagramas de funcionamiento de cada una.

Secuencial

La arquitectura cliente-servidor secuencial implementada se basa en una comunicación bidireccional entre un servidor y múltiples clientes. El servidor escucha conexiones entrantes y, cuando se establece una conexión con un cliente, utiliza la codificación y decodificación de Gob para intercambiar datos de manera eficiente.

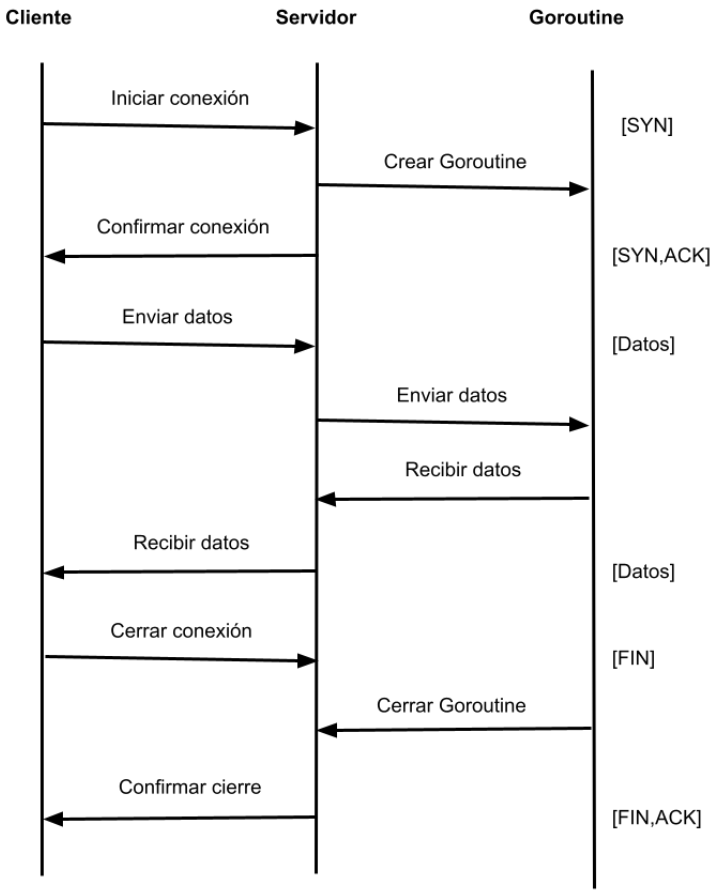
- **Diagrama de secuencia:**



Concurrente creando una Goroutine

En esta arquitectura se ha introducido la concurrencia utilizando goroutines. Cuando se recibe una conexión entrante desde un cliente, en lugar de manejarla secuencialmente en el hilo principal del servidor, se crea una nueva goroutine con la instrucción go handleRequest(conn). Esto significa que cada conexión se maneja de forma independiente en su propia goroutine, lo que permite al servidor procesar múltiples solicitudes de clientes simultáneamente.

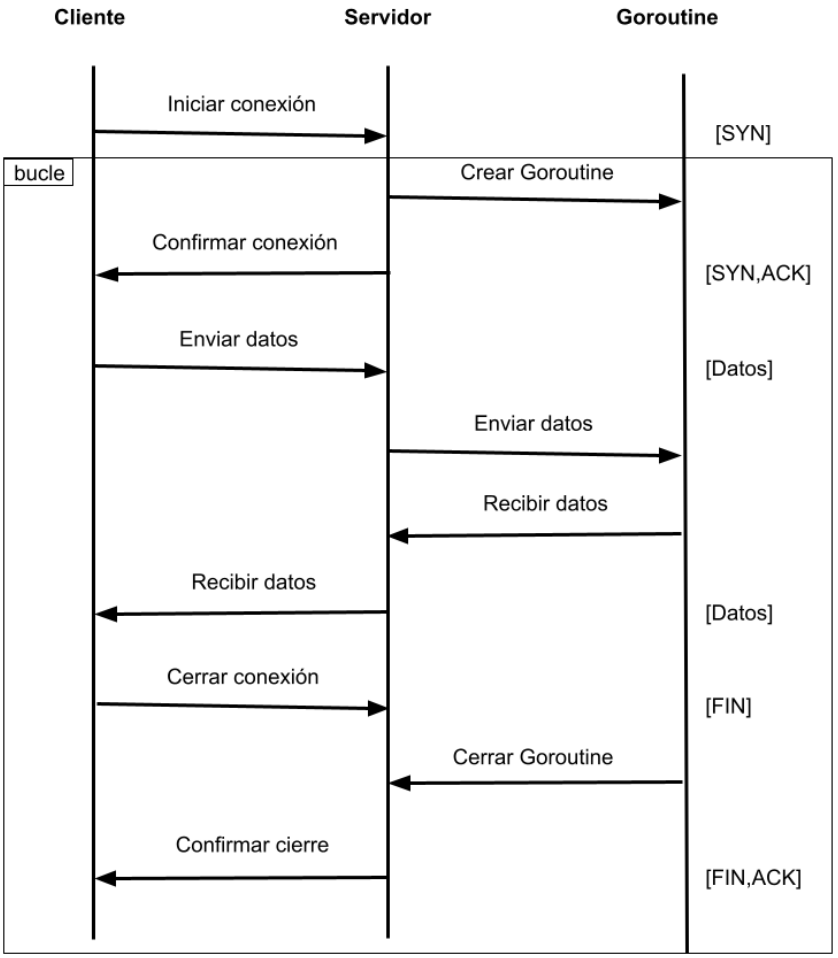
- **Diagrama de secuencia:**



Concurrente con pool fijo de Goroutines

La arquitectura cliente-servidor concurrente de pool fijo implementada aprovecha una estrategia de concurrencia para manejar múltiples solicitudes de clientes de manera eficiente. En lugar de crear una nueva goroutine para cada solicitud entrante, utiliza un conjunto fijo de goroutines (un "pool") para procesar las conexiones de los clientes.
Esta arquitectura permite que el servidor atienda a varios clientes de manera concurrente, limitando el número de goroutines activas a un tamaño predefinido (en este caso, poolSize es 5). Esto ayuda a controlar la cantidad de recursos utilizados y garantiza una gestión eficiente de las solicitudes de los clientes.

- **Diagrama de secuencia:**



Máster-worker

En esta arquitectura el servidor ejerce el papel de "master" al iniciar un conjunto de "workers" para realizar tareas específicas. Posteriormente, el servidor recibe solicitudes de clientes y asigna cada solicitud a uno de los workers disponibles. Cada worker se encarga de procesar las solicitudes de manera independiente.
Esta arquitectura está diseñada para distribuir la carga de trabajo de manera eficiente, lo que la hace adecuada para aplicaciones que requieren procesamiento concurrente de múltiples tareas.

- **Diagrama de secuencia:**