Programación en Red: Sockets

ST0255 - Telemática

Integrantes:

Pablo Correa

María Antonia Rincón

Docente:

Juan Carlos Montoya

25 de octubre, 2020

Universidad EAFIT

Medellín, Colombia

Objetivos

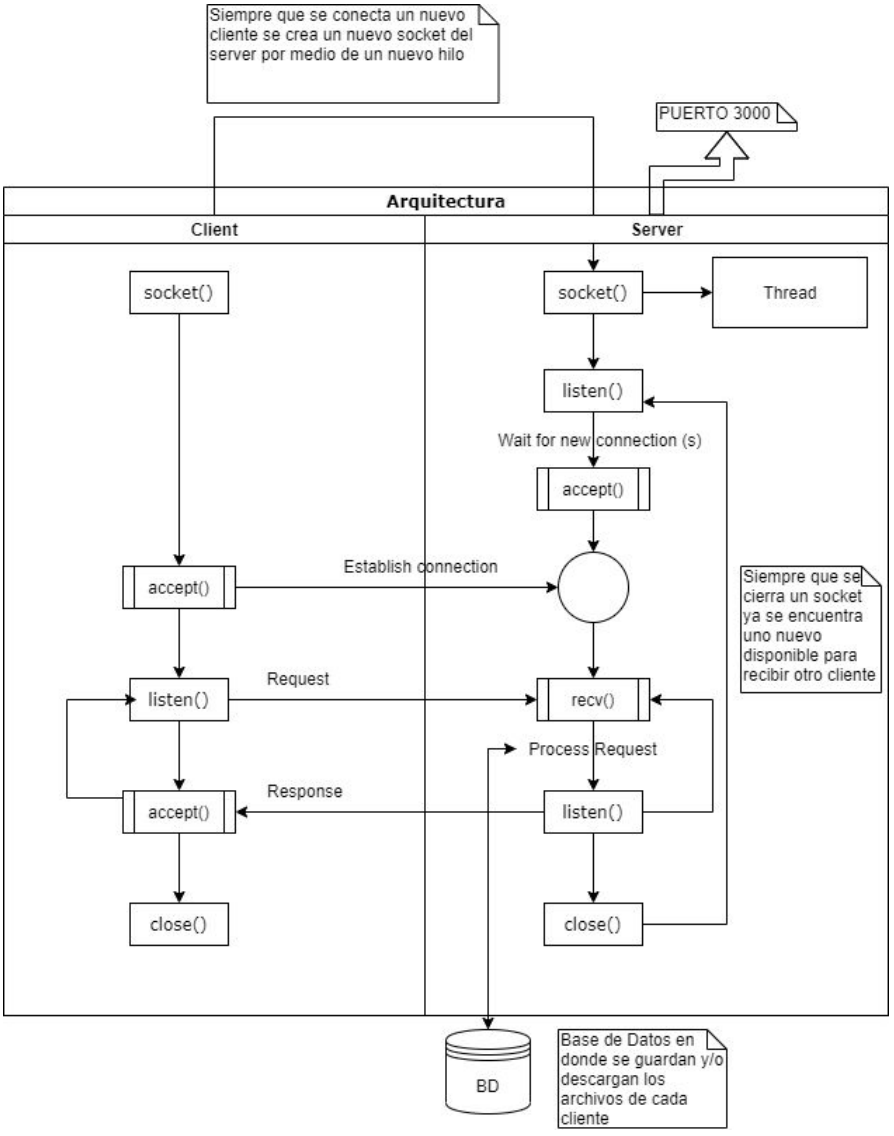
En la actualidad la inmediatez se ha vuelto más y más necesaria para nuestro diario vivir. Es por esto que la tecnología debe estar siempre al tanto de estas necesidades y brindar las mejores soluciones a estas.

Este texto tiene como objetivo dar un primer acercamiento a la programación multi-hilos orientada a sockets. Para ello se desarrolló un protocolo de comunicación propio entre el servidor y los clientes. Además, en orden para cumplir con el objetivo del sistema, se desarrolló el código en torno a la programación en hilos, con lo cual se soluciona la problemática de la concurrencia que se presenta cuando no se tienen hilos, donde para cada petición se tendría que hacer “fila” mientras que el servidor va a resolviendo cada petición.

Especificaciones del Servicio

Los sockets son un punto final de comunicación bidireccional entre un servidor y un cliente. Estos están entrelazados con el número de un puerto para así identificar la aplicación o el cliente al que los datos deben llegar y son capaces de simplificar la manera en la que funciona un programa. En este proyecto se realizó un implementación de Sockets en la que se puede hacer manejo de archivos y directorios por medio de comandos CRUD.

El primer paso para lograr una buena implementación, es conocer la estructura clave de la implementación. Es por esto que se realizó el siguiente diagrama para conocer bien las acciones que se iban a realizar desde el socket y cómo estas interactúan entre el server y el client:



Además de esto, es importante definir el orden de ejecución de los procesos. En este caso, se crea primero una conexión entre el client y el server para poder empezar a enviar y recibir data y luego se chequean los argumentos o el path que el usuario desee para que ahí se ejecute el resto del programa. En caso de que no se ingrese un directorio, o que se ingrese uno incorrecto, se crea uno por defectos. Posteriormente se implementa el multi threading. Esto lo que permite es que el server pueda realizar acciones con varios clientes simultáneamente y que no sea de manera secuencial, es decir, todos los clientes pueden enviar data al mismo tiempo sin esperar a que uno de ellos termine la sesión para poder funcionar.

Ya luego de que el server esté escuchando a todos los clientes conectados, el usuario comienza a digitar los diferentes comandos en el cliente, que permiten acciones en archivos o directorios. Para esto se tiene el siguiente vocabulario de mensajes (comandos) y un ejemplo de ejecución:

- Create bucket: Permite crear un bucket.
  Comando: **`mkbkt bucketName`**
  Ejemplo: *mkbkt bucket01*
- Remove bucket: Permite remover buckets.

  Comando: **`rm bucketName`**

  Ejemplo: *rm bucket01*
- List buckets: Permite listar todos los buckets que se están usando (no muestra los que han sido eliminados.

  Comando: **`ls`**
- List files from bucket: Permite listar el contenido dentro de un solo bucket. Cualquier tipo de archivos los muestra.

  Comando: **`ls bucketName`**

  Ejemplo: *ls bucket01*

- Upload files from client to a server bucket: Subir archivos del cliente a un bucket específico del sever.

  Comando: **`upload fileName bucketName`** or **`upload filePath bucketName`**

  Ejemplo:
  - Usando el nombre del archivo (esto aplica cuando el archivo está en la misma carpeta del proyecto ejecutándose):

    upload example.txt
  - Usando el path de cualquier archivo en el computador en uso:

    upload /home/user/Documents/example.txt
- Download file from server bucket to client: Descarga los archivos del server al cliente. Primero se especifica el bucket dónde está el archivo y luego el nombre del archivo. Cuando se ejecuta, crea una carpeta downloads.

  Comando: **`download bucketName fileName`**

  Ejemplo: *download bucket01 example.txt*
- Remove file from bucket: Elimina cualquier archivo del bucket. Primero se especifica el bucket dónde está el archivo y luego el archivo a eliminar.

  Comando: **`rm bucketName FileName`**

  Ejemplo: *rm bucket01 example.txt*

Finalmente, si se desea finalizar la sesión entre un cliente y el server, se digita el comando `quit,` el resto de clientes seguirán en la sesión.

Conclusiones

A partir de los conocimientos adquiridos con esta implementación se puede concluir que para la comunicación por medio de sockets y la solución de la problemática que se genera a partir de la concurrencia, el uso de la programación Multi-Hilos es una gran herramienta para los sistemas que implementen esta clase de tecnologías y protocolos.