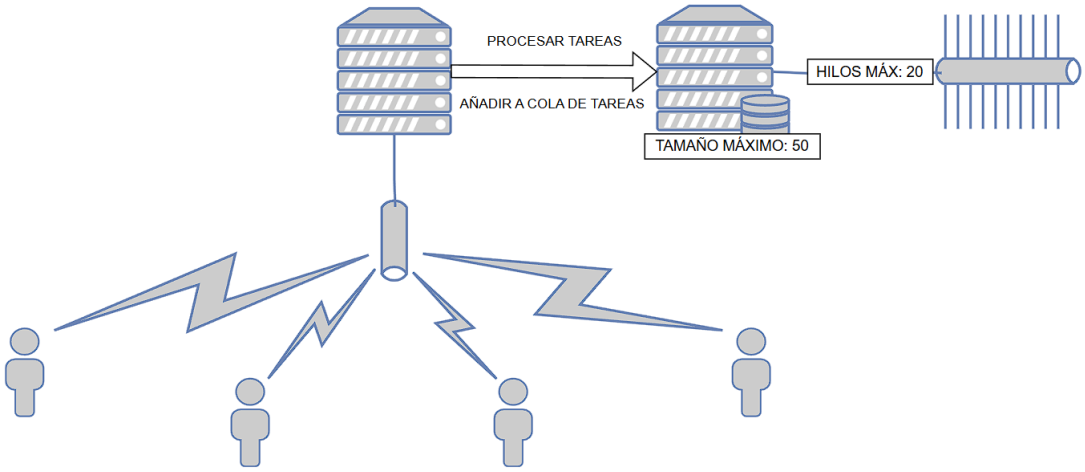
DETALLES DE MI SERVIDOR HTTP

Introducción a la programación Paralela y Distribuida

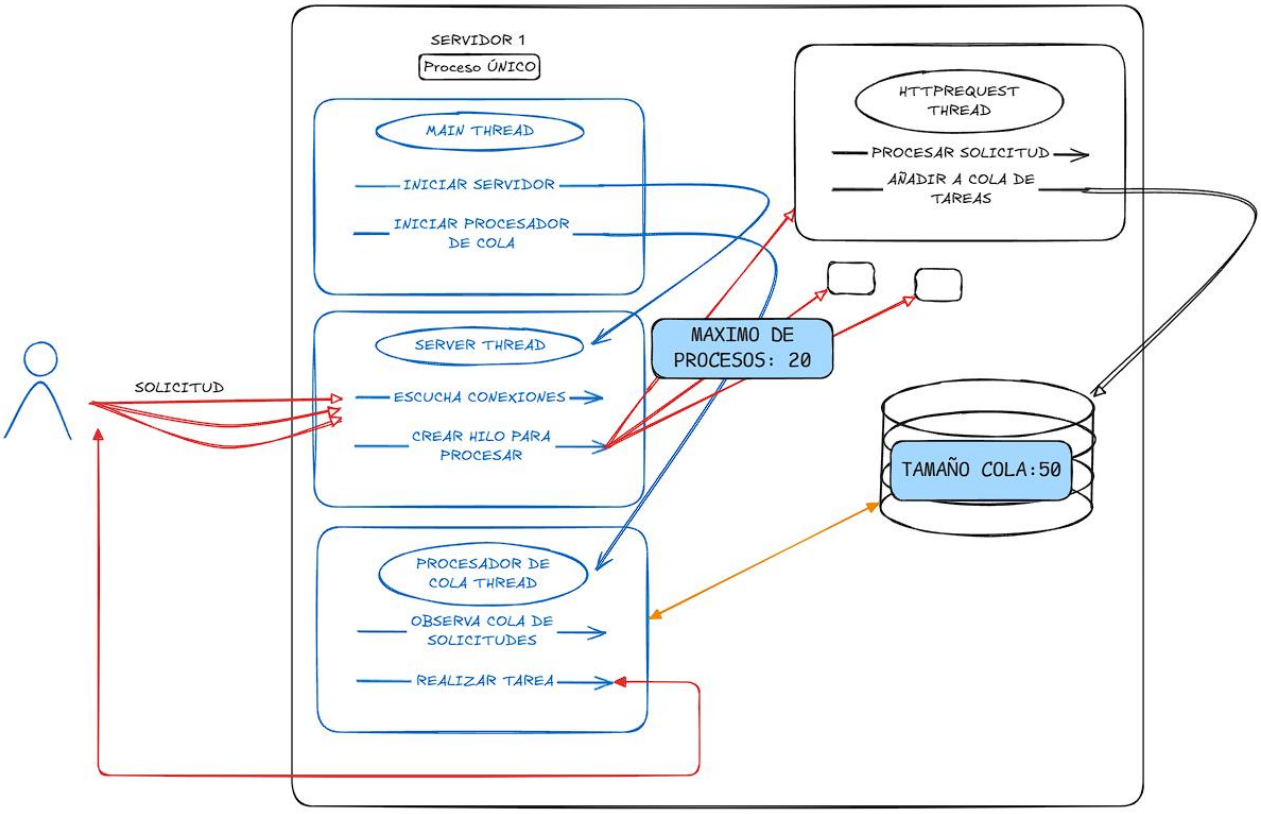


25 DE MARZO DE 2025

UAX
Alejandro Barreche Ruiz

El sistema implementa un servidor HTTP ***concurrente*** que atiende *múltiples solicitudes* de clientes de manera simultánea. Para poder realizarlo, he trabajado con el módulo **SOCKETSERVER** el cual simplifica la tarea de crear servidores de red.

La arquitectura del servidor separa claramente la configuración, el manejo de recursos, el procesamiento de solicitudes y la generación de respuestas, lo que facilita el mantenimiento y permite manejar diferentes hilos dentro de un mismo proceso.



El servidor se implementa en un único proceso de Python, aprovechando la concurrencia mediante hilos (**threads**) para gestionar múltiples solicitudes de manera simultánea. La estructura y el flujo de ejecución se organizan de la siguiente manera:

1. Proceso del Servidor

Al iniciar la aplicación (mediante la ejecución de *python main.py*), se crea un único proceso de Python que representa el entorno global del servidor. Este proceso contiene todos los componentes necesarios para la operación concurrente, evitando la necesidad de emplear múltiples procesos.

2. Hilo Principal

1. El hilo principal es lo primero que se **ejecuta en el proceso** y se encarga de la **configuración inicial**.
2. Se crea una **instancia de ThreadingHTTPServer**, la cual se encargará de escuchar y aceptar conexiones entrantes.
3. Se inicia un hilo específico para ejecutar el método **serve_forever()** del servidor, el cual queda a la espera de nuevas conexiones.
4. Adicionalmente, el hilo principal arranca un hilo denominado **ProcesadorColaThread**, encargado de consumir la cola de solicitudes en segundo plano.
5. Una vez configurados y lanzados los componentes, el **hilo principal permanece en ejecución** (por ejemplo, mediante un bucle de espera) para evitar la terminación del proceso.

3. Hilo del Servidor (ThreadingHTTPServer)

El hilo que ejecuta serve_forever() se encarga de poner al servidor a la "escucha". Su función principal consiste en:

- **Aceptación de Conexiones:**

  Al recibir una conexión entrante, el servidor crea automáticamente un nuevo hilo (**HTTPRequestHandlerThread**) para gestionar la solicitud de forma individual.

- **Gestión de la Concurrencia:**

  Gracias a la herencia de ThreadingMixIn, cada **solicitud** se procesa en un **hilo separado**, permitiendo que **múltiples solicitudes sean atendidas en paralelo**. La utilización de un semáforo (un **BoundedSemaphore(20)**) garantiza que el número de hilos de petición activos no supere un límite predefinido, controlando así el grado de concurrencia.

4. Hilos de Petición (HTTPRequestHandlerThread)

Cada solicitud entrante es atendida por un hilo dedicado que ejecuta el método handle() de la clase HTTPRequestHandler. Las funciones específicas de estos hilos incluyen:

- **Procesamiento de la Solicitud:**

  El hilo lee y analiza la solicitud HTTP recibida, determinando el método (GET, POST, etc.) y la ruta correspondiente.

- **Registro y Encolado de la Solicitud:**

  En el caso de que se trate de una petición (por ejemplo, POST a /data), el hilo invoca el método *agregar_solicitud_a_cola()* del objeto RecursosCompartidos, agregando la solicitud a la cola interna.

- **Generación de la Respuesta:**

  Tras procesar la solicitud, se envía la respuesta correspondiente al cliente.

5. Procesador de Cola (ProcesadorColaThread)

El hilo ProcesadorColaThread opera en paralelo al resto de los hilos y se encarga exclusivamente de gestionar la cola de solicitudes.

- **Extracción de Solicitudes:**

  Mediante un bucle continuo, el ProcesadorColaThread realiza llamadas a get() sobre la cola de solicitudes, extrayendo cada solicitud registrada por los hilos de petición.

- **Procesamiento en Segundo Plano:**

  Una vez obtenida una solicitud, el hilo ejecuta el procesamiento correspondiente (por ejemplo, simulando un retardo mediante time.sleep(1) o realizando operaciones adicionales), y finalmente marca la tarea como completada.

DECISIONES CONCURRENTES

Servidor Multihilo:

Se utiliza la clase ThreadingHTTPServer (basada en socketserver.ThreadingTCPServer) para atender cada conexión de cliente en un **hilo independiente**. Esto permite que múltiples solicitudes sean procesadas simultáneamente sin bloquear al.

Control de Concurrencia:

Se implementa un semáforo acotado (BoundedSemaphore) en el handler para limitar el número máximo de conexiones concurrentes. Esto previene la saturación del servidor y mejora la estabilidad **evitando** que se creen un **número muy elevado de hilos**.

Locks y Semáforos:

Para evitar condiciones de carrera, se utilizan Locks al incrementar contadores o modificar estructuras compartidas, y semáforos para controlar el acceso a recursos críticos (por ejemplo, en la clase RecursosCompartidos).

Colas para Comunicación:

Se usa una cola (queue.Queue) para desacoplar la recepción de solicitudes de su procesamiento. Un hilo dedicado, representado por la clase ProcesadorCola, se encarga de procesar estas solicitudes en segundo plano.