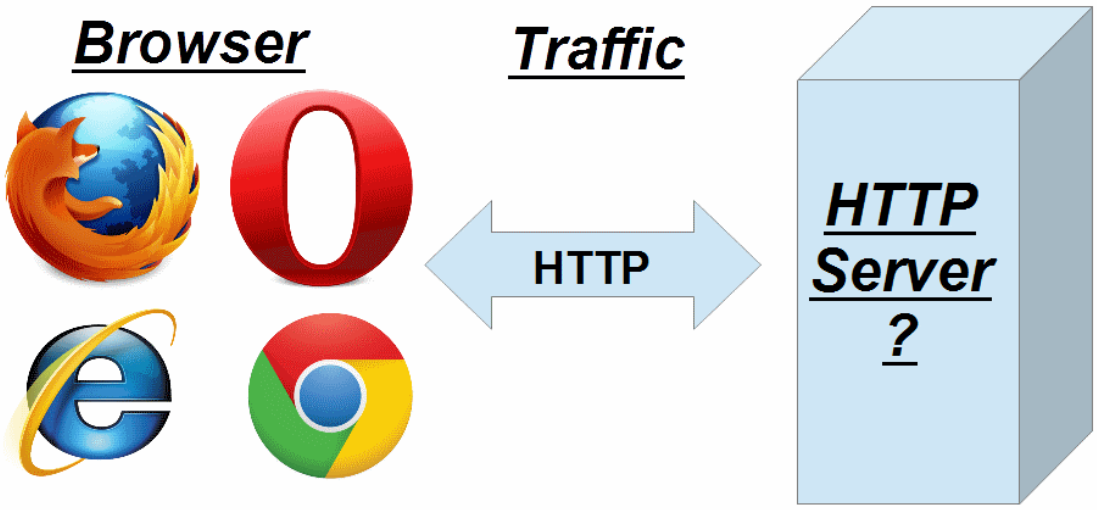

P1 Redes II - Servidor HTTP

14/03/2023

—

Pablo Sánchez y Alvaro Rodriguez

Visión general

Esta práctica consiste en diseñar e implementar en lenguaje C un servidor Web, con algunas limitaciones, pero plenamente funcional, que pueda procesar los verbos más usados del protocolo HTTP, junto a alguna funcionalidad extra.

Objetivos

1. Soporte para los verbos GET, POST y OPTIONS.
    - Cumplido, el servidor discrimina cada petición según qué tipo de verbo sea parseado de la petición HTTP.
2. Configuración del servidor a través de un fichero
    - Cumplido, se lee el fichero de configuración al crear el servidor y se usan los parámetros definidos en el mismo. Se hace uso de un parseo de parámetros propio sin uso de librerías de terceros.
3. Parseo de peticiones
    - Cumplido, se hace uso de la librería picoHTTPparser para el parseo de peticiones HTTP.
4. Soporte para la ejecución de scripts
    - Cumplido, se ejecutan correctamente tanto archivos python como php con argumentos enviados por STD_IN (se hace uso de pipes y fork() para esta implementación).
5. Codificación de respuestas **200 OK**, **400 Bad Request** y **404 Not Found**
    - Cumplido, se hace envío de respuestas HTTP con código de respuesta según como haya terminado la ejecución del servidor.
6. Argumentos de las respuestas
    - Cumplido, cada respuesta tiene argumentos tales como fecha, content type o última modificación de archivo generados de forma dinámica en tiempo de ejecución cuando se crea la respuesta HTTP.
7. Documentación del proyecto
    - Cumplido, todas las funciones y archivos creados están documentados, y se ha generado posteriormente documentación Doxigen, la cual puede ser consultada abriendo el siguiente archivo: ***html/index.html***

Orden de implementación

- Implementar un cliente y servidor TCP para probar el envío y recepción de datos, poner a prueba el envío de ficheros grandes y detectar los posibles cuellos de botella. Probar a paralelizar el cliente para hacer muchas peticiones simultáneamente y ver qué sucede.

  Realizado, al tener solo un hilo, se satura fácilmente con varias peticiones.

- Elegir primero el esquema de servidor concurrente a desarrollar (con un proceso o thread para cada petición, con un pool de los mismos, etc.), implementarlo y comprobar que funciona correctamente sin ninguna funcionalidad HTTP, simplemente devolviendo una cadena predeterminada. Así se puede probar fácilmente el rendimiento del servidor y su estabilidad ante cargas muy altas.

  Realizado, con varios hilos el servidor es más eficiente y puede dar servicio a más clientes y gestionar más peticiones.

- A continuación, cuando se esté seguro de que esta parte está acabada, comentar las funciones necesarias para convertirlo de nuevo en un servidor iterativo (con un solo hilo de ejecución, sin threads ni procesos). Así podrás centrarte ahora en desarrollar toda la funcionalidad de recepción, parseo y respuesta de peticiones HTTP, "aislando" esta parte del resto.

  Realizado, simplemente hay que comentar la parte del código en la que se implementa el pool de hilos.

- Cuando esté acabada, volver a activar la concurrencia.

  Realizado, ahora el servidor envía respuestas HTTP de forma concurrente de forma robusta y sin fallas de ejecución.

Comparación server concurrente hilos/procesos

Un servidor concurrente que utiliza hilos y uno que utiliza procesos difieren en cómo manejan la concurrencia y el uso de recursos del sistema.

Un servidor concurrente que utiliza hilos usa múltiples hilos dentro de un solo proceso para manejar múltiples conexiones de clientes simultáneamente. Cada hilo puede acceder y compartir la memoria del proceso principal, lo que puede hacer que la programación sea más fácil y eficiente en términos de recursos. Sin embargo, un problema común con los hilos es que pueden ocurrir errores de sincronización si dos o más hilos intentan acceder y modificar los mismos datos simultáneamente.

Por otro lado, un servidor concurrente que utiliza procesos crea múltiples procesos separados para manejar múltiples conexiones de clientes simultáneamente. Cada proceso tiene su propia memoria y espacio de direcciones, lo que significa que los procesos no comparten memoria entre sí. Esto puede hacer que la programación sea más complicada, ya que se deben implementar técnicas de comunicación interprocesos para compartir datos entre los procesos. Sin embargo, la ventaja de utilizar procesos es que son más resistentes a los errores de sincronización que los hilos, ya que cada proceso tiene su propia memoria y no puede acceder a la memoria de otros procesos sin permiso explícito.

En resumen, los servidores concurrentes que utilizan hilos son más adecuados para aplicaciones que requieren una programación más sencilla y una mejor eficiencia en el uso de recursos, mientras que los servidores concurrentes que utilizan procesos son más adecuados para aplicaciones que requieren una mayor resistencia a errores de sincronización y seguridad en el acceso a los datos compartidos.

Comentarios adicionales:

- Se ha añadido un método en el index.html para descargar un archivo PDF.
- El commit definitivo en gitlab tiene etiqueta “Entrega”
- Se ha implementado un manejador de señales para que al recibir un “Ctr + c” el servidor finalice y libere memoria de forma correcta.

Conclusiones:

Conclusiones técnicas: En resumen, implementar un servidor HTTP en C es un proyecto desafiante que requiere una comprensión profunda de la programación web y los conceptos básicos de HTTP. Se abordaron varios problemas técnicos durante el proceso de implementación, incluida la creación de sockets para manejar las conexiones entrantes, el manejo de las solicitudes HTTP entrantes, el manejo de múltiples conexiones simultáneas y la generación de respuestas HTTP correctas. Además, aprendimos la importancia de crear un servidor eficiente y escalable que pueda manejar una gran cantidad de solicitudes simultáneas sin degradar su rendimiento. Para lograr esto, se han implementado técnicas como la multiplexación de E/S y el uso de subprocesos para administrar múltiples conexiones simultáneas.

Conclusiones personales: En nuestra experiencia personal, implementar un servidor HTTP en C fue un proyecto desafiante. Aprendimos muchos conceptos básicos de programación web y HTTP y obtuvimos habilidades valiosas en el diseño e implementación de servidores eficientes y escalables. Sin embargo, también encontramos algunos desafíos en el camino. En particular, el manejo de múltiples conexiones simultáneas y la generación de respuestas HTTP correctas fueron áreas que nos costó entender y dominar. En general, encontramos que este proyecto fue una experiencia valiosa y enriquecedora que nos ayudó a desarrollar importantes habilidades técnicas y mejorar nuestra comprensión de la programación web y los sistemas distribuidos.