webserver / Subject.md

Nacho Mota 🚀 reademe now

216 lines (129 loc) · 10.8 KB

Preview Code Blame Raw

Webserv Esto es cuando finalmente entiendes por qué una URL comienza con HTTP

Resumen: Este proyecto se trata de escribir tu propio servidor HTTP. Podrás probarlo con un navegador web real. HTTP es uno de los protocolos más utilizados en Internet. Conocer sus entresijos será útil, incluso si no vas a trabajar en un sitio web. Versión: 21.2

Contenidos

Capítulo I
Introducción El Protocolo de Transferencia de Hipertexto (HTTP) es un protocolo de aplicación para sistemas de información hipermedia colaborativos y distribuidos.

HTTP es la base de la comunicación de datos para la World Wide Web, donde los documentos de hipertexto incluyen hipervínculos a otros recursos a los que el usuario puede acceder fácilmente, por ejemplo, con un clic del ratón o tocando la pantalla en un navegador web.

HTTP se desarrolló para facilitar el hipertexto y la World Wide Web.

La función principal de un servidor web es almacenar, procesar y entregar páginas web a los clientes. La comunicación entre el cliente y el servidor se realiza mediante el Protocolo de Transferencia de Hipertexto (HTTP).

Las páginas entregadas son con mayor frecuencia documentos HTML, que pueden incluir imágenes, hojas de estilo y scripts además del contenido de texto.

Se pueden utilizar varios servidores web para un sitio web con mucho tráfico.

Un agente de usuario, comúnmente un navegador web o un rastreador web, inicia la comunicación solicitando un recurso específico mediante HTTP y el servidor responde con el contenido de ese recurso o un mensaje de error si no puede hacerlo. El recurso suele ser un archivo real en el almacenamiento secundario del servidor, pero esto no es necesariamente así y depende de cómo se implemente el servidor web.

Si bien la función principal es servir contenido, la implementación completa de HTTP también incluye formas de recibir contenido de los clientes. Esta característica se utiliza para enviar formularios web, incluida la carga de archivos.

Capítulo II Reglas generales

• Tu programa no debe bloquearse en ninguna circunstancia (incluso cuando se quede sin memoria) y no debe salir inesperadamente. Si ocurre, tu proyecto se considerará no funcional y tu calificación será 0.

• Debes entregar un Makefile que compilará tus archivos fuente. No debe volver a vincular.

• Tu Makefile debe contener al menos las reglas: $(NAME), all, clean, fclean y re.

• Compila tu código con c++ y las banderas -Wall -Wextra -Werror

• Tu código debe cumplir con el estándar C++ 98. Luego, debería seguir compilando si agregas la bandera -std=c++98.

• Intenta desarrollar siempre utilizando la mayor cantidad de funciones de C++ que puedas (por ejemplo, elige sobre <string.h>). Puedes usar funciones de C, pero siempre prefiere sus versiones de C++ si es posible.

• Cualquier biblioteca externa y las bibliotecas Boost están prohibidas.

Capítulo III Parte obligatoria

Nombre del programa webserv Archivos a entregar Makefile, *.{h, hpp}, *.cpp, *.tpp, *.ipp, archivos de configuración Makefile NAME, all, clean, fclean, re Argumentos [Un archivo de configuración] Funciones externas Todo en C++ 98. execve, dup, dup2, pipe, strerror, gai_strerror, errno, dup, dup2, fork, socketpair, htons, htonl, ntohs, ntohl, select, poll, epoll (epoll_create, epoll_ctl, epoll_wait), kqueue (kqueue, kevent), socket, accept, listen, send, recv, chdir bind, connect, getaddrinfo, freeaddrinfo, setsockopt, getsockname, getprotobyname, fcntl, close, read, write, waitpid, kill, signal, access, stat, open, opendir, readdir y closedir. Libft autorizada n/a Descripción Un servidor HTTP en C++ 98

Debes escribir un servidor HTTP en C++ 98. Tu ejecutable se ejecutará de la siguiente manera: ./webserv [archivo de configuración]

Incluso si se menciona poll() en el enunciado y la escala de evaluación, puedes utilizar cualquier equivalente como select(), kqueue() o epoll().

Webserv Esto es cuando finalmente entiendes por qué una URL comienza con HTTP Por favor, lee el RFC y haz algunas pruebas con telnet y NGINX antes de comenzar este proyecto.

Incluso si no tienes que implementar todo el RFC, leerlo te ayudará a desarrollar las características requeridas.

Webserv Esto es cuando finalmente entiendes por qué una URL comienza con HTTP III.1 Requisitos

• Tu programa debe tomar un archivo de configuración como argumento o utilizar una ruta predeterminada.

• No puedes ejecutar otro servidor web.

• Tu servidor nunca debe bloquearse y el cliente puede ser rechazado adecuadamente si es necesario.

• Debe ser no bloqueante y utilizar solo 1 poll() (o equivalente) para todas las operaciones de E/S entre el cliente y el servidor (incluyendo listen).

• poll() (o equivalente) debe verificar lectura y escritura al mismo tiempo.

• Nunca debes hacer una operación de lectura o escritura sin pasar por poll() (o equivalente).

• Verificar el valor de errno está estrictamente prohibido después de una operación de lectura o escritura.

• No necesitas usar poll() (o equivalente) antes de leer tu archivo de configuración.

```
Debido a que debes usar descriptores de archivo no bloqueantes,
es posible utilizar las funciones read/recv o write/send sin
poll() (o equivalente), y tu servidor no se bloquearía.

Pero consumiría más recursos del sistema.

Por lo tanto, si intentas leer/recibir o escribir/enviar en
cualquier descriptor de archivo sin usar poll() (o equivalente),
tu calificación será 0.
```

• Puedes usar todas las macros y definir como FD_SET, FD_CLR, FD_ISSET, FD_ZERO (entender qué hacen y cómo lo hacen es muy útil).

• Una solicitud a tu servidor nunca debe bloquearse indefinidamente.

• Tu servidor debe ser compatible con el navegador web de tu elección.

• Consideraremos que NGINX cumple con HTTP 1.1 y se puede usar para comparar encabezados y comportamientos de respuesta.

• Los códigos de estado de respuesta HTTP deben ser precisos.

• Tu servidor debe tener páginas de error predeterminadas si no se proporcionan.

• No puedes usar fork para otra cosa que no sea CGI (como PHP o Python, etc.).

• Debes poder servir un sitio web completamente estático.

• Los clientes deben poder cargar archivos.

• Necesitas al menos los métodos GET, POST y DELETE.

• Prueba de estrés tu servidor. Debe permanecer disponible a toda costa.

• Tu servidor debe poder escuchar en varios puertos (ver Archivo de configuración).

Webserv Esto es cuando finalmente entiendes por qué una URL comienza con HTTP
III.2 Solo para MacOS

Dado que MacOS no implementa write() de la misma manera que otros sistemas operativos Unix, se te permite usar fcntl().

Debes usar descriptores de archivo en modo no bloqueante para obtener un comportamiento similar al de otros sistemas operativos Unix.

Sin embargo, solo se te permite usar fcntl() con las siguientes banderas:

F_SETFL, O_NONBLOCK y FD_CLOEXEC.

Cualquier otra bandera está prohibida.

III.3 Archivo de configuración

Puedes inspirarte en la parte 'server' del archivo de configuración de NGINX.

En el archivo de configuración, debes poder:

• Elegir el puerto y host de cada 'servidor'.

• Configurar los server_names o no.

• El primer servidor para un host:puerto será el predeterminado para este host:puerto (es decir, responderá a todas las solicitudes que no pertenezcan a otro servidor).

• Configurar páginas de error predeterminadas.

• Limitar el tamaño del cuerpo del cliente.

• Configurar rutas con una o varias de las siguientes reglas/configuraciones (las rutas no usarán expresiones regulares):

◦ Definir una lista de métodos HTTP aceptados para la ruta. ◦ Definir una redirección HTTP. ◦ Definir un directorio o un archivo desde donde se debe buscar el archivo (por ejemplo, si la url /kapouet está enraizada en /tmp/www, la url / kapouet/pouic/toto/pouet es /tmp/www/pouic/toto/pouet). ◦ Activar o desactivar el listado de directorios. ◦ Establecer un archivo predeterminado para responder si la solicitud es un directorio. ◦ Ejecutar CGI basado en determinadas extensiones de archivo (por ejemplo, .php). ◦ Hacerlo funcionar con los métodos POST y GET. ◦ Hacer que la ruta pueda aceptar archivos cargados y configurar dónde se deben guardar.

```
    * ¿Te preguntas qué es un CGI?
    * Debido a que no llamarás al CGI directamente, usa la ruta
completa como PATH_INFO.
    * Solo recuerda que, para solicitudes fragmentadas, tu
servidor necesita desagrupar, el CGI esperará EOF como el final
del cuerpo.
    * Lo mismo para la salida del CGI. Si no se devuelve
content_length desde el CGI, EOF marcará el final de los datos
devueltos.
    * Tu programa debe llamar al CGI con el archivo solicitado
como primer argumento.
    * El CGI debe ejecutarse en el directorio correcto para
acceder a archivos de ruta relativa.
    * Tu servidor debe funcionar con un CGI (php-CGI, Python,
etc.).
```

Debes proporcionar algunos archivos de configuración y archivos básicos predeterminados para probar y demostrar que todas las características funcionan durante la evaluación.

Si tienes alguna pregunta sobre un comportamiento, debes comparar el comportamiento de tu programa con NGINX.

Por ejemplo, verifica cómo funciona server_name.

Te hemos compartido un pequeño probador. No es obligatorio aprobarlo si todo funciona bien con tu navegador y pruebas, pero puede ayudarte a cazar algunos errores.

Lo importante es la resistencia. Tu servidor nunca debe morir.

No hagas pruebas con un solo programa. Escribe tus pruebas con un lenguaje más conveniente como Python o Golang, etc. Incluso en C o C++ si quieres.

Capítulo IV Parte Bonus

Aquí están las funciones adicionales que puedes agregar:

• Admitir cookies y administración de sesiones (prepara ejemplos rápidos). • Manejar múltiples CGI.

La parte de bonificación solo se evaluará si la parte obligatoria es PERFECTA. Perfecto significa que la parte obligatoria se ha hecho íntegramente y funciona sin mal funcionamiento. Si no has aprobado TODOS los requisitos obligatorios, tu parte de bonificación no se evaluará en absoluto.

Capítulo V
Entrega y evaluación por pares

Entrega tu tarea en tu repositorio Git como de costumbre. Solo el trabajo dentro de tu repositorio será evaluado durante la defensa. No dudes en verificar los nombres de tus archivos para asegurarte de que sean correctos.

16D85ACC441674FBA2DF65190663F42A3832CEA21E024516795E1223BBA7791
6734D1
26120A16827E1B16612137E59ECD492E46EAB67D109B142D49054A7C2814049
01890F 619D682524F5