Pontificia Universidad Javeriana Cali

Servidor web con concurrencia

Sistemas Operativos

July 30, 2021

1 Descripción del proyecto final

El objetivo de este proyecto es desarrollar un servidor web concurrente. Para simplificar este proyecto, se proporciona el código de un servidor web no concurrente (pero que funciona). Este servidor web básico opera con un solo hilo; será su trabajo hacer que el servidor web sea multihilo para que pueda manejar múltiples peticiones al mismo tiempo.

Los objetivos de este proyecto son:

- Aplicar conceptos dados en clase como:
  - Hilos
  - Sincronización
  - Productor-consumidor
  - Algoritmos de planificación
- Aprender a añadir concurrencia a un sistema no concurrente
- Aprender la arquitectura básica de un servidor web simple

HTTP
Antes de describir lo que se va a implementar en este proyecto, se dara una breve visión general de cómo funciona un servidor web clásico, y el protocolo

HTTP (versión 1.0) utilizado para comunicarse con él; aunque los navegadores y servidores web han evolucionado mucho a lo largo de los años, las versiones antiguas siguen funcionando y te dan un buen comienzo para entender cómo funcionan las cosas. El objetivo al proporcionar un servidor web básico es que pueda estar protegido de aprender todos los detalles de las conexiones de red y el protocolo HTTP necesarios para realizar el proyecto; sin embargo, el código de red se ha simplificado mucho y es bastante comprensible si decide estudiarlo.

Los navegadores web clásicos y los servidores web interactúan utilizando un protocolo basado en texto llamado HTTP (Hypertext Transfer Protocol). Un navegador web abre una conexión con un servidor web y solicita algún contenido con HTTP. El servidor web responde con el contenido solicitado y cierra la conexión. El navegador lee el contenido y lo muestra en la pantalla.

HTTP está construido sobre el conjunto de protocolos TCP/IP que proporciona el sistema operativo. Juntos, el TPC y el IP garantizan que los mensajes se dirijan a su destino correcto, que lleguen del origen al destino de forma fiable ante un fallo y que no congestionen la red enviando demasiados mensajes a la vez, entre otras características.

Cada pieza de contenido en el servidor web está asociada a un archivo en el sistema de archivos del servidor. El más sencillo es el contenido estático, en el que un cliente envía una petición sólo para leer un archivo específico del servidor. Un poco más complejo es el contenido dinámico, en el que un cliente solicita que se ejecute un archivo en el servidor web y se le devuelva su resultado. Cada archivo tiene un nombre único conocido como URL (Localizador Universal de Recursos).

Como ejemplo sencillo, suponga que el navegador del cliente quiere obtener contenido estático (es decir, sólo algún archivo) de un servidor web que se ejecuta en alguna máquina. El cliente podría escribir la siguiente URL en el navegador: http://www.cs.wisc.edu/index.html. Esta URL identifica que se debe utilizar el protocolo HTTP y que se debe obtener un archivo HTML en el directorio raíz (/) del servidor web llamado index.html en la máquina anfitriona www.cs.wisc.edu.

El servidor web no sólo se identifica de forma única por la máquina en la que se ejecuta, sino también por el puerto en el que escucha las conexiones. Los puertos son una abstracción de comunicación que permite que múltiples comunicaciones de red (posiblemente independientes) ocurran si-

multáneamente en una máquina; por ejemplo, el servidor web podría estar recibiendo una solicitud HTTP en el puerto 80 mientras un servidor de correo está enviando correo electrónico usando el puerto 25. Por defecto, se espera que los servidores web funcionen en el puerto 80 (el conocido número de puerto HTTP), pero a veces (como en este proyecto), se utilizará un número de puerto diferente. Para obtener un archivo de un servidor web que se ejecuta en un número de puerto diferente (por ejemplo, 8000), especifique el número de puerto directamente en la URL, por ejemplo, http://www.cs.wisc.edu:8000/index.html.

Las URL de archivos ejecutables (es decir, de contenido dinámico) pueden incluir argumentos de programa después del nombre del archivo. Por ejemplo, para ejecutar un programa (test.cgi) sin ningún argumento, el cliente puede utilizar la URL http://www.cs.wisc.edu/test.cgi. Para especificar más argumentos, se utilizan los caracteres ? y &, con el carácter ? para separar el nombre del archivo de los argumentos y el carácter & para separar cada argumento de los demás. Por ejemplo, http://www.cs.wisc.edu/test.cgi?x=10&y=20 puede utilizarse para enviar varios argumentos x e y y sus respectivos valores al programa test.cgi. El programa que se ejecuta se llama programa CGI (abreviatura de Common Gateway Interface; sí, es un nombre terrible); los argumentos se pasan al programa como parte de la variable de entorno QUERY_STRING, que el programa puede analizar para acceder a estos argumentos.

La petición HTTP

Cuando un cliente (por ejemplo, un navegador) quiere obtener un archivo de una máquina, el proceso comienza enviando un mensaje a la máquina. Pero, ¿qué hay exactamente en el cuerpo de ese mensaje? El protocolo HTTP especifica con precisión el contenido de la solicitud y la respuesta posterior.

Iniciemos por el contenido de la petición, enviado desde el navegador web al servidor. Esta petición HTTP consiste en una línea de petición, seguida de cero o más cabeceras de petición, y finalmente una línea de texto vacía. Una línea de petición tiene la forma: method uri version. El método suele ser GET, que indica al servidor web que el cliente simplemente quiere leer el archivo especificado; sin embargo, existen otros métodos (por ejemplo, POST). La uri es el nombre del archivo, y quizás argumentos opcionales (en el caso de contenido dinámico). Por último, la versión indica la versión del protocolo HTTP que está utilizando el cliente web (por ejemplo, HTTP/1.0).

La respuesta HTTP (del servidor al navegador) es similar; consiste en una

línea de respuesta, cero o más cabeceras de respuesta, una línea de texto vacía, y finalmente la parte interesante, el cuerpo de la respuesta. Una línea de respuesta tiene la forma de mensaje de estado de la versión. El estado es un número entero positivo de tres dígitos que indica el estado de la solicitud; algunos estados comunes son 200 para OK, 403 para Forbidden (es decir, el cliente no puede acceder a ese archivo) y 404 para File Not Found (el famoso error). Dos líneas importantes de la cabecera son Content-Type, que indica al cliente el tipo de contenido en el cuerpo de la respuesta (por ejemplo, HTML o gif u otro) y Content-Length, que indica el tamaño del archivo en bytes.

Para este proyecto, realmente no se necesita conocer esta información sobre HTTP a menos que se quiera entender los detalles del código que se dan. No se necesitara modificar ninguno de los procedimientos del servidor web que se ocupan del protocolo HTTP o de las conexiones de red.

Un servidor web básico

El código del servidor web está disponible en este proyecto. Puede compilar los archivos aquí simplemente escribiendo make. Compila y ejecuta este servidor web básico antes de hacer cualquier cambio en él, make clean elimina los archivos .o y los ejecutables y permite hacer una construcción limpia.

Cuando ejecute este servidor web básico, necesita especificar el número de puerto en el que escuchará; los puertos por debajo del número 1024 están reservados así que debería especificar números de puerto que sean mayores que 1023 para evitar este rango reservado; el máximo es 65535. Tenga cuidado: si se ejecuta en una máquina compartida, podría entrar en conflicto con otras y, por tanto, hacer que su servidor no se vincule al puerto deseado. Si esto ocurre, pruebe con otro número.

Cuando conecte su navegador a este servidor, asegúrese de especificar este mismo puerto. Por ejemplo, suponga que está ejecutando en bumble21.cs.wisc.edu y utiliza el número de puerto 8003; copie su archivo HTML favorito en el directorio desde el que inicia el servidor web. Luego, para ver este archivo desde un navegador web (que se ejecuta en la misma máquina o en otra), utiliza la url bumble21.cs.wisc.edu:8003/favorite.html. Si ejecuta el cliente y el servidor web en la misma máquina, puede utilizar simplemente el nombre de host localhost por comodidad, por ejemplo, localhost:8003/favorite.html.

Para hacer el proyecto un poco más fácil, se proporciona un servidor web mínimo, que consiste en sólo unos cientos de líneas de código C. Como resul-

tado, el servidor está limitado en su funcionalidad; no maneja ninguna petición HTTP que no sea GET, sólo entiende unos pocos tipos de contenido, y sólo soporta la variable de entorno QUERY_STRING para programas CGI. Este servidor web tampoco es muy robusto; por ejemplo, si un cliente web cierra su conexión con el servidor, puede disparar una aserción en el servidor provocando su salida. No se espera que arregle estos problemas (aunque puede hacerlo, si quiere, ya sabe, por diversión).

Se proporcionan funciones de ayuda para simplificar la comprobación de errores. Un wrapper llama a la función deseada y termina inmediatamente si se produce un error. Las envolturas se encuentran en el archivo io-helper.h más sobre esto a continuación. Uno siempre debe comprobar los códigos de error, incluso si todo lo que hace en respuesta es salir; dejar caer los errores silenciosamente es MALA PROGRAMACIÓN EN C y debe evitarse a toda costa.

Nuevas funcionalidades!
En este proyecto, va a añadir dos piezas clave de funcionalidad al servidor web básico. En primer lugar, hará que el servidor web sea multihilo. En segundo lugar, implementará diferentes políticas de programación para que las peticiones sean atendidas en diferentes órdenes. También modificarás la forma de invocar al servidor web para que pueda manejar nuevos parámetros de entrada (por ejemplo, el número de hilos a crear).

Parte 1: Multihilo
El servidor web básico que se proporciona tiene un solo hilo de control. Los servidores web de un solo hilo sufren un problema fundamental de rendimiento, ya que sólo se puede atender una única petición HTTP a la vez. Por lo tanto, cualquier otro cliente que acceda a este servidor web debe esperar hasta que la petición http actual haya terminado; esto es especialmente un problema si la petición HTTP actual es un programa CGI de larga duración o es residente sólo en el disco (es decir, no está en la memoria). Por lo tanto, la extensión más importante que va a añadir es hacer que el servidor web básico sea multihilo.

El enfoque más sencillo para construir un servidor multihilo es generar un nuevo hilo para cada nueva petición http. El sistema operativo programará estos hilos de acuerdo con su propia política. La ventaja de crear estos hilos es que ahora las peticiones cortas no tendrán que esperar a que se complete una petición larga; además, cuando un hilo está bloqueado (es decir, esperando a que termine la E/S del disco) los otros hilos pueden seguir ges-

tionando otras peticiones. Sin embargo, el inconveniente del enfoque de un hilo por petición es que el servidor web paga la sobrecarga de crear un nuevo hilo en cada petición.

Por lo tanto, el enfoque generalmente preferido para un servidor multihilo es crear un grupo de hilos de trabajo de tamaño fijo cuando el servidor web se inicia por primera vez. Con el enfoque de pool de hilos, cada hilo se bloquea hasta que haya una petición http que atender. Por lo tanto, si hay más hilos de trabajo que solicitudes activas, entonces algunos de los hilos estarán bloqueados, esperando que lleguen nuevas solicitudes HTTP; si hay más solicitudes que hilos de trabajo, entonces esas solicitudes tendrán que ser almacenadas en el buffer hasta que haya un hilo listo.

En su implementación, debe tener un hilo maestro que comience por crear un grupo de hilos de trabajo, cuyo número se especifica en la línea de comandos. Su hilo maestro es entonces responsable de aceptar nuevas conexiones HTTP a través de la red y colocar el descriptor de esta conexión en un buffer de tamaño fijo; en su implementación básica, el hilo maestro no debe leer de esta conexión. El número de elementos en el buffer también se especifica en la línea de comandos. Tenga en cuenta que el servidor web existente tiene un único subproceso que acepta una conexión y la gestiona inmediatamente; en su servidor web, este subproceso debería colocar el descriptor de la conexión en un búfer de tamaño fijo y volver a aceptar más conexiones.

Cada hilo de trabajo es capaz de manejar tanto peticiones estáticas como dinámicas. Un subproceso de trabajo se activa cuando hay una petición HTTP en la cola; cuando hay varias peticiones HTTP disponibles, la petición que se maneja depende de la política de programación, descrita más adelante. Una vez que el hilo trabajador se despierta, realiza la lectura en el descriptor de red, obtiene el contenido especificado (ya sea leyendo el archivo estático o ejecutando el proceso CGI), y luego devuelve el contenido al cliente escribiendo en el descriptor. El hilo trabajador espera entonces otra petición HTTP.

Tenga en cuenta que el hilo maestro y los hilos trabajadores están en una relación productor-consumidor y requieren que sus accesos al buffer compartido estén sincronizados. Específicamente, el hilo maestro debe bloquear y esperar si el buffer está lleno; un hilo trabajador debe esperar si el buffer está vacío. En este proyecto, se requiere el uso de variables de condición.

Nota al margen: no se confunda por el hecho de que el servidor web básico que proporcionamos bifurca un nuevo proceso para cada proceso CGI que ejecuta. Aunque, en un sentido muy limitado, el servidor web utiliza múltiples procesos, nunca maneja más que una sola petición a la vez; el proceso padre en el servidor web espera explícitamente a que el proceso CGI hijo se complete antes de continuar y aceptar más peticiones HTTP. Cuando haga que su servidor sea multihilo, no debe modificar esta sección del código.

Parte 2: Políticas de programación
En este proyecto, usted implementará un número de diferentes políticas de programación. Tenga en cuenta que cuando su servidor web tiene múltiples hilos de trabajo en ejecución (cuyo número se especifica en la línea de comandos), usted no tendrá ningún control sobre qué hilo es realmente programado en un momento dado por el sistema operativo. Su papel en la programación es determinar qué petición HTTP debe ser manejada por cada uno de los hilos de trabajo en espera en su servidor web.

La política de programación se determina mediante un argumento de la línea de comandos cuando se inicia el servidor web y es la siguiente:

First-in-First-out (FIFO): Cuando un hilo de trabajo se despierta, maneja la primera petición (es decir, la más antigua) en el buffer. Tenga en cuenta que las peticiones HTTP no terminarán necesariamente en el orden FIFO; el orden en el que se completen las peticiones dependerá de cómo el SO programe los hilos activos.

El archivo más pequeño primero (SFF): Cuando un hilo de trabajo se despierta, maneja la solicitud del archivo más pequeño. Esta política se aproxima a la de "Primero el trabajo más corto" en la medida en que el tamaño del archivo es una buena predicción del tiempo que se tarda en atender esa petición. Las solicitudes de contenido estático y dinámico pueden mezclarse, dependiendo del tamaño de esos archivos. Tenga en cuenta que este algoritmo puede llevar a la inanición de las solicitudes de archivos grandes. También hay que tener en cuenta que la política SFF requiere que se conozca algo de cada solicitud (por ejemplo, el tamaño del archivo) antes de poder programar las solicitudes. Por lo tanto, para soportar esta política de programación, necesitará hacer algún procesamiento inicial de la

solicitud (pista: usar stat() en el nombre del archivo) fuera de los hilos de los trabajadores; probablemente querrá que el hilo maestro realice este trabajo, lo que requiere que lea del descriptor de red.

Seguridad
Ejecutar un servidor en red puede ser peligroso, especialmente si no se tiene cuidado. Por lo tanto, la seguridad es algo que debes considerar cuidadosamente cuando crees un servidor web. Una cosa que siempre debes asegurarte de hacer es no dejar tu servidor funcionando más allá de las pruebas, dejando así abierta una potencial puerta trasera a los archivos de tu sistema.

Su sistema también debe asegurarse de restringir las solicitudes de archivos para permanecer dentro del sub-árbol de la jerarquía del sistema de archivos, enraizado en el directorio de trabajo base en el que se inicia el servidor. Debe tomar medidas para asegurar que los nombres de ruta que se pasan no se refieren a archivos fuera de este sub-árbol. Una forma sencilla (quizás demasiado conservadora) de hacerlo es rechazar cualquier nombre de ruta que contenga .., evitando así cualquier recorrido por el árbol del sistema de archivos. Soluciones más sofisticadas podrían utilizar chroot() o contenedores de Linux, pero quizás estén fuera del alcance del proyecto.

Parámetros de la línea de comandos
Su programa C debe ser invocado exactamente como sigue:

prompt> ./wserver [-d basedir] [-p port] [-t threads] [-b buffers] [-s schedalg]

Los argumentos de la línea de comandos de su servidor web deben interpretarse de la siguiente manera:

- **basedir**: es el directorio raíz desde el que debe operar el servidor web. El servidor debe tratar de asegurar que los accesos a los archivos no accedan a archivos por encima de este directorio en la jerarquía del sistema de archivos. Por defecto: directorio de trabajo actual (por ejemplo, .).
- **basedir**:port: el número de puerto en el que debe escuchar el servidor web; el servidor web básico ya maneja este argumento. Por defecto:

10000.

- **basedir**:threads: el número de hilos de trabajo que deben crearse en el servidor web. Debe ser un número entero positivo. Por defecto: 1.
- **basedir**:buffers: el número de conexiones de petición que pueden ser aceptadas a la vez. Debe ser un número entero positivo. Tenga en cuenta que no es un error que se creen más o menos hilos que buffers. Por defecto: 1.
- **basedir**:schedalg: el algoritmo de programación a realizar. Debe ser uno de FIFO o SFF. Por defecto: FIFO.

Por ejemplo, puede ejecutar su programa como
prompt> servidor -d . -p 8003 -t 8 -b 16 -s SFF

En este caso, su servidor web escuchará en el puerto 8003, creará 8 hilos de trabajo para manejar las peticiones HTTP, asignará 16 búferes para las conexiones que estén en curso (o en espera), y utilizará la programación SFF para las peticiones que lleguen.

Código fuente
Se recomienda que comprenda cómo funciona el código que se proporciona:

- **wserver.c**: Contiene main() para el servidor web y el bucle básico de servicio.
- **request.c**: Realiza la mayor parte del trabajo para manejar las peticiones en el servidor web básico. Comienza en request_handle() y trabaja a través de la lógica desde allí.
- **io_helper.h y io_helper.c**: Contiene funciones de envoltura para las llamadas al sistema invocadas por el servidor web básico y el cliente. La convención es añadir _or_die a una llamada existente para proporcionar una versión que tenga éxito o salga. Por ejemplo, la llamada al sistema open() se utiliza para abrir un archivo, pero puede fallar por varias razones. La envoltura, open_or_die(), abre un archivo con éxito o existe si falla.
- **wclient.c**: Contiene main() y las rutinas de soporte para el cliente web muy simple. Para probar su servidor, puede querer cambiar este código para que pueda enviar peticiones simultáneas a su servidor. Lanzando wclient varias veces, puedes probar cómo tu servidor maneja las peticiones simultáneas.

- **spin.c**: Un simple programa CGI. Básicamente, gira durante una cantidad de tiempo fija, lo que puede ser útil para probar varios aspectos de su servidor.

- **Makefile**: También le proporcionamos un Makefile de ejemplo que crea wserver, wclient y spin.cgi. Puedes escribir make para crear todos estos programas. Puede escribir make clean para eliminar los archivos de objetos y los ejecutables. Puedes escribir make server para crear sólo el programa servidor, etc. A medida que creas nuevos archivos, tendrás que añadirlos al Makefile.

La mejor manera de conocer el código es compilarlo y ejecutarlo. Ejecute el servidor que se da con su navegador web preferido. Ejecute este servidor con el código del cliente que se da. Incluso puede hacer que el código cliente que se da contacte con cualquier otro servidor que hable HTTP. Haga pequeños cambios en el código del servidor (por ejemplo, haga que imprima más información de depuración) para ver si entiende cómo funciona.

Sobre la entrega:

- Puede ser trabajado en parejas.
- Sustentación presencial.
- Fecha máxima de entrega: 12-Dic-2020