Taller pages

Ejercicio N°3

Objetivos	● Diseño y construcción de sistemas con acceso distribuido ● Encapsulación de Threads y Sockets en Clases ● Implementación de protocolos de comunicación ● Protección de los recursos compartidos ● Uso de buenas prácticas de programación en C++
Instancias de Entrega	**Entrega 1:** clase 8 (26/05/2020). **Entrega 2:** clase 10 (09/06/2020).
Temas de Repaso	● Definición de clases en C++ ● Contenedores de STL ● Excepciones / RAII ● Move Semantics ● Sockets ● Threads
Criterios de Evaluación	● Criterios de ejercicios anteriores ● Eficiencia del protocolo de comunicaciones definido ● Control de paquetes completos en el envío y recepción por Sockets ● Atención de varios clientes de forma simultánea ● Eliminación de clientes desconectados de forma controlada

Introducción

Se desarrollará una aplicación servidor que atenderá petitorios HTTP, mediante los cuales podrá acceder y dar alta a recursos del mismo. No hace falta bajas

Descripción

Manejo de petitorios

El servidor deberá poder interpretar petitorios HTTP, y extraer de estos los siguientes parámetros:

- Método (GET, POST, etc)
- Ruta del recurso
- Body en caso de que corresponda

get -> Obtener un recurso
Post-> Darle de alta

Utilizando estos parámetros deberá ejecutar la lógica correspondiente y devolver una respuesta también en formato HTTP

Formato de Línea de Comandos

Servidor

```
./server <puerto/servicio> <root_file>
```

Donde `<puerto/servicio>` es el puerto TCP (o servicio) el cual el servidor deberá escuchar las conexiones entrantes.

El parámetro `<template_root>` representa la ruta a un archivo con una respuesta para el recurso “/” (ver más adelante).
Ejemplo

```
<html>
Este es el directorio root
</html>
```

Cliente

El cliente se ejecutará de la siguiente forma:

El cliente solo requiere una ip y un puerto para conectarse y mandarle un petitorio

```
./client <ip/hostname> <port/service>
```

El cliente se conectará al servidor corriendo en la máquina con dirección IP `<ip>` (o `<hostname>`), en el puerto (o servicio) TCP `<puerto/servicio>`. Recibirá por entrada standard el texto correspondiente a un petitorio HTTP, el cuál leerá y enviará por socket hasta llegar a EOF. Una vez enviado el petitorio, escuchará la respuesta del servidor e imprimirá por salida standard.

Códigos de Retorno

El servidor devolverá 0 si su ejecución fue exitosa. Si la cantidad de parámetros es incorrecta, se cancela la ejecución y se devuelve 1. El cliente deberá devolver siempre 0.

Entrada y Salida Estándar

Servidor

Entrada estándar

El servidor esperará el caracter `'q'` por entrada estándar. Cuando lo reciba, el servidor deberá cerrar el socket aceptador, y esperar a que las conexiones se cierren antes de liberar los recursos y retornar.

Salida estándar

Una vez que inicie el servidor, lo termino con la letra q.
Muestro la primer linea d elos petitorios

El servidor imprime por salida standard la primer linea del petitorio.

Cliente

Una vez que envie el petitorio, tiene que imprimir la respuesta del servidor

Entrada estándar

Por entrada estándar, el cliente recibirá el petitorio para enviar al servidor.

Salida estándar

Por salida estándar se imprimirá la respuesta del servidor.

Protocolo

Esto es lo mas importante del tp

El protocolo HTTP posee el siguiente formato

En los casos de prueba hay ejemplos de como es el formato.

- La primer linea contiene la forma
- Las siguientes lineas tienen la forma `<clave>:<valor>`
- Una linea vacía indica el fin de la cabecera
- El cuerpo (“body”) del petitorio si el método posee uno.

Los métodos soportados serán únicamente `GET` y `POST`
Se puede asumir que los petitorios siempre respetan el protocolo.HTTP
Cambio el salto de linea de windows por el linux
Para simplificar el procesamiento, en el protocolo se cambiaron los saltos de linea “`\r\n`” por “`\n`”
Una vez que el cliente termina de enviar el mensaje, cierra el canal de escritura, de esta manera el servidor sabe hasta dónde leer (Nota: si se quiere que el servidor funcione con aplicaciones reales, se debe usar sockets no bloqueantes o leer de a 1 caracter. Ambas cosas están prohibidas para este TP)

La unica manera de decirle al servidor que termine de escribir es cerrando el canal del cliente. SOCKETS NO BLOQUEANTES O LEER DE 1 CARACTER ESTAN PROHIBIDOS, HAY QUE HACER COMO EL TP1

Respuestas a distintos métodos y recursos

Si hacemos un get a la raiz

GET /

En este caso la respuesta será “`HTTP 200 OK\nContent-Type: text/html\n\n`” seguido del contenido del `<template_root>`. Siguiendo con el ejemplo inicial, la respuesta sería así:

```
HTTP 200 OK
Content-Type: text/html

<html>
Este es el directorio root
</html>
```

GET /<recurso>

Lo que hace es buscar si el recurso existe, y si existe hace lo de abajo

- Si el recurso existe (ver en POST la creación de recursos), la respuesta será

“`HTTP 200 OK\n\n`” seguido del contenido del `<recurso>`

- Si el recurso no existe, la respuesta será “`HTTP 404 NOT FOUND\n\n`”

POST /

Los recursos se crean con post. Como no puedo modificar la raiz me tira este error

En este caso la respuesta será “`HTTP 403 FORBIDDEN\n\n`”, ya que la raiz es de sólo lectura.

POST /<recurso>

Para saber cual es el body, en las claves "veo el content length y despues el mensaje" REVER

Se creará el recurso correspondiente a la ruta, y su contenido será el Body del mensaje. El petitorio debe tener la clave “Content-Length” en la cabecera, y su valor será la longitud en bytes del cuerpo del petitorio.

Cuando voy a http: /saludo me aparece hola

PODEMOS ASUMIR QUE VAN A HACER REQUEST VALIDAS.

NO PENSAR EN QUE VAN A VENIR DATOS BINARIOS, O QUE VAN A VENIR DATOS INCOMPLETOS, ETC.

Otros métodos

En caso de recibir otro método, la respuesta será “`HTTP 405 METHOD NOT ALLOWED\n\n`”

NO HAY QUE IMPLEMENTAR EL PROTOCOLO HTTP CUMPLIENDO CON TODAS LAS RTC ETC.
HAY QUE HACER LO QUE DICE EL ENUNCIADO.

Restricciones

En las respuestas 405, 403, etc. En algun momento, en como respondemos al cliente hay que usar polimorfismo.

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C++11.
2. Está prohibido el uso de variables globales.
3. Se deberá aplicar polimorfismo en la resolución de respuestas del servidor.
4. Se deberá sobrecargar el operador **()** en alguna clase.
5. Se debe realizar una limpieza de clientes finalizados cada vez que se conecta uno nuevo.

Como mostro Martin, una vez que los clientes finaliza los marco como que se murieron y despues los limio

Recomendaciones

Utilizar std::stringbuffer para facilitar el “parseo” de las lineas del petitorio.

Con string string y getline, podemos parsear muy facil. Podemos parsear por delimitadores.

Referencias

Va a haber multiples clientes -> Entonces va a haber estado compartido.

[1] Formato de peticiones HTTP: https://developer.mozilla.org/es/docs/Web/HTTP/Messages

Se puede probar todo con netcat, si creemos que nuestro socket funciona mal

Tiene threads por tener varios clientes a la vez, no hay que tener en cuenta endianess, padding, etc.
"Me copie del tp1, va a haber perdida de paquetes"
El makefile: Hay que poner el codigo en 3 carpetas
common_src -> Va para las 2 apps
server_src y client_src.
El makefile va a armar una aplicacion cliente y una servidor.
Hay que ponerlos dentro de esas carpetas los archivos.
Concurrencia -> El script corre los clientes secuenciales, no va a detectar muchos problemas de concurrencia. Que de ok en el sercom

Sockets -> Agarrar los del tp1 y wrappearlo en c++. Pasar lo del socket de tp1 y pasarlo a c++, agregar raii y excepciones.

"Puede ayudar para el tp final que no este acoplado no se que"

Proxy es super overkill para este tp, no meterlo.