Informe HTTP Proxy Server

Grupo 1
Integrantes:

- Pedro Pingarilho - 58451
- Facundo Astiz Meyer - 58333
- Miguel Di Luca - 58460
- Ezequiel Keimel - 58325

Descripción

Proxy (HTTP)

La implementación del proxy se divide principalmente en:

- Socket pasivo del proxy
- Socket activo del cliente
- Socket activo del origen
- Pipes de conexión con el proceso transformador
- Selector de file descriptors

Socket pasivo

El socket pasivo del proxy se configura de acuerdo a los argumentos de entrada del programa (-l y -p) al iniciar el proxy. Al recibir un nuevo cliente, lo acepta creando un socket activo que se registra en el selector.

Por defecto el socket escucha en todas las interfaces y en el puerto 8080.

Cliente

Por su parte, el cliente tiene asociados buffers de lectura y escritura. Dichos buffers los utiliza para comunicarse con el origen y/o con el proceso transformador. Adicionalmente, el buffer de lectura cumple la función de permitir el parseo del request conforme se van recibiendo los headers.

Cuando se inicia una nueva conexión con el cliente, se verifica si la primera línea contiene la dirección al servidor al cual se desea conectar. Caso contrario, se prosigue a buscar el header Host.

Dado que las líneas de los headers pueden recibirse incompletas (ya que se van recibiendo de a poco), se implementó en el buffer la posibilidad de leer de a líneas sólo si las mismas se encuentran completas (mediante la función buffer_peek_line).

Una vez que se encuentra la dirección del origen, se resuelve el nombre DNS mediante una función no bloqueante, para la cual se crea un hilo especializado. Luego, cuando se concreta la conexión con el origen, se lo registra en el selector y luego el socket del cliente se limita a reenviar los bytes al origen a través del buffer de escritura.

Origen

El origen también tiene asociados un buffer de escritura y uno de lectura, que se corresponden con los buffers de lectura y de escritura del cliente, respectivamente. Es decir, lo que el origen escriba en el buffer de escritura, el cliente lo puede leer en el buffer de lectura, y viceversa. Además, asociado al socket del origen están los datos del parser del response, que se encarga, entre otras cosas, de decodificar el cuerpo del mensaje si el

mismo se encuentra en chunks. Además, el origen inicia el proceso transformador (cuando corresponda) y le delega la responsabilidad de escribir al cliente el cuerpo del response, una vez haya sido transformado por dicho proceso. Además, ante un cierre de la conexión por parte del origen, el transformador también es el encargado de interrumpir el flujo de datos.

Transformador

Finalmente, la conexión con el proceso transformador se realiza a través de unnamed pipes: uno de escritura y otro de lectura.

Cuando finaliza el response del origen, se cierra el pipe de escritura, lo cual sirve de aviso para que el proceso transformador sepa que ya no hay más información para transformar. Adicionalmente, la salida del pipe de lectura se escribe directamente en el buffer de lectura del cliente, no sin antes ser codificada con chunks (utilizando la función write_chunked).

Otros detalles de implementación

Cabe destacar que el proxy es compatible con conexiones SSL, aunque las mismas no son compatibles con ningún tipo de transformación, por obvias razones.

Es también relevante mencionar que todas respuestas del origen son codificadas con chunks, independientemente de cómo se hayan recibido. Esto es así ya que es imposible conocer con antelación la longitud del mensaje final luego de una transformación, lo cual impide la utilización del header Content-Length.

Otro detalle de implementación a destacar es el uso de la estructura connection_data, la cual aúna toda la información vinculada a la terna cliente-origen-transformador. Dicha estructura no sólo almacena los file descriptors de cada agente y sus datos asociados, sino también la información circunstancial de la conexión, como puede ser el contenido del header Content-Type (si se alcanzó a leer) o la información que indica si el response finalizó.

En cuanto a las métricas, la información asociada a las mismas se registra en una estructura global que se instancia en main.c. Dicha estructura está definida en proxySettings.h.

Por su parte, los buffers tienen todos un tamaño fijo definido en una constante (configurada en 10 mil bytes).

Protocolo de configuración

Intercambio de mensajes

En esta sección se describen las estructuras utilizadas en el intercambio de datos

Requests

Los requests se realizan por medio del siguiente segmento sucedido por la estructura indicada en el campo “Indice de tabla de estructuras”.

Nro de Versión	Índice de tabla de estructuras

Número de versión: numero de un byte que representa la versión del protocolo a utilizar.
Índice de tabla de estructuras: número de un byte que representa el método solicitado como muestra la siguiente tabla.

Número	Nombre
0	Login
1	Métricas
2	Transformaciones

Response

El siguiente segmento se envía en cada respuesta indicando el resultado de la operación. Luego se envía la respuesta asociada a la operación utilizada.

Código

Código: número de tamaño de un byte que representa la respuesta indicada en la siguiente tabla.

2	OK
1	INTERNAL ERROR
0	PERMISSION DENIED

Métodos

En cada método se definen dos estructuras; una para el request y otra para el response.

Login

Request

Usuario	Contraseña

Usuario y contraseña: campos de tipo texto de tamaño variable que se representa por medio de un \0 en el final.

Response

Sin cuerpo.

Métricas

Request

Tipo

Tipo: número de un byte que representa la métrica a utilizar usando los valores de la siguiente tabla.

Número	Nombre
0	Cantidad de conexiones actuales
1	Cantidad de conexiones que se establecieron
2	Bytes transferidos

Response

Resultado

Resultado: número de 4 bytes de resultado.

Transformaciones

Request

Tipo	Argumentos

Tipo:

Número	Nombre
0	Set status
1	Get status
2	Set media types
3	Get media types
4	Set transformation command
5	Get transformation command

Argumentos:

- Los request de tipo Get no contienen argumentos.
- Para Set status el argumento es un campo de 1 byte de longitud, que puede ser 0 o 1 si se desea desactivar o activar las transformaciones, respectivamente.
- Para el resto de los request de tipo Set el argumento es un campo de tipo texto, de longitud variable y terminado en '\0'

Response

- Si el request es de tipo Set, el cuerpo del response es vacío.
- Para Get status el cuerpo es un campo de 1 byte de longitud que puede ser 0 o 1 si las transformaciones están desactivadas o activadas, respectivamente.
- Para el resto de los request de tipo Get el cuerpo es:

Contenido

Contenido: campo de tipo de texto, de tamaño variable y terminado en '\0'.

Problemas encontrados

Parseo de Headers

Uno de los primeros problemas encontrados durante la implementación del proxy fue el del parseo de los headers, tanto del request como del response.

Una opción evaluada fue la de almacenar todo el header hasta que se termine de recibir por completo, para luego parsearlo. Sin embargo, dicha opción fue descartada debido a la ineficiencia en términos de memoria.

Otra opción fue la de construir una máquina de estados capaz de analizar caracter a caracter hasta alcanzar el fin del header, pero fue descartada por ser demasiado compleja para el caso.

Finalmente se optó por almacenar el header en un buffer intermedio que permita leer de a líneas una vez que estas se completan. De esta manera, el parseo de los headers resulta mucho más sencillo sin sacrificar demasiada eficiencia.

Transformaciones

Las transformaciones acarrearon diversas dificultades.

En primer lugar, requieren analizar el contenido del header Content-Length del response para luego compararlo con la lista de media types configuradas en el proxy y ver si coinciden. Además, dado que las transformaciones deben poder ser modificadas en tiempo de ejecución, se debe iniciar un nuevo proceso transformador entre cada request y para cada cliente. Todo esto se vuelve más difícil si se tiene en cuenta que la conexión con el origen puede interrumpirse en cualquier momento, lo cual puede ocasionar errores si no se lo tiene en cuenta.

Otro inconveniente que se encontró fue el hecho de que el pipe de escritura que se comunica con la entrada estándar del proceso transformador puede acabar lleno y no aceptar nuevos bytes. Esto significa que los bytes que no se lograron escribir deben ser almacenados temporalmente hasta que se libere espacio en el pipe de escritura, lo cual requirió de otro buffer auxiliar.

También se puede mencionar el hecho de que no se puede hacer ninguna suposición en términos del flujo de datos. Es decir, la salida del proceso transformador puede terminar mucho después de que se haya recibido toda la información del origen, y ésta puede tener una longitud completamente distinta.

Interrupciones inesperadas

Otro factor que dificultó la implementación del proxy fue la posibilidad de que la conexión con el origen y/o el cliente se interrumpa en cualquier momento. Esto significa que en cada acción de lectura y escritura a dichos destinos sea considerada la posibilidad de que la conexión se haya cerrado, y estar preparado para enviar un error si eso sucede.

Por ejemplo, uno de estos errores se experimentó al utilizar el método send, el cual aborta la ejecución del programa si la conexión se interrumpió. Para impedir esto, se debió agregar el flag MSG_NOSIGNAL que evita dicho comportamiento.

Limitaciones

Velocidad de transferencia

Una de las limitaciones más evidentes es la disminución en la velocidad de la conexión. Se pudo registrar una disminución de la velocidad de transferencia de hasta 6 veces menor comparada con la velocidad nominal.

Gzip encoding

Otra limitación es que si el cuerpo del response se encuentra codificado con gzip, el contenido no será transformado.

Longitud de headers

Para parsear las líneas de los headers, se decidió que la máxima longitud de una línea del header es de 1000 bytes. Si una línea superase dicha longitud, se responderá al cliente con un código de error 500.

Timeouts

El proxy no utiliza timeouts para las conexiones con los servidores de origen, lo cual significa que los clientes serán los que esperarán hasta abortar la conexión si la misma nunca se concreta.

Extensiones

En cuanto al protocolo, se podría agregar funcionalidad para configurar individualmente cada propiedad. Actualmente posible configurar el proxy una única transformación para todas las conexiones y a todos los tipos de cuerpo. Se podría cambiar lo anterior para agregar la posibilidad de configurar una transformación distinta para cada conexión y tipo de cuerpo en particular. Asimismo se podría agregar funcionalidad para permitir la consulta de métricas específicas utilizando algún lenguaje de consulta.

En cuanto al proxy, se lo puede hacer compatible con gzip. También, se podría incluir la posibilidad de configurar un transformador distinto para cada media type.

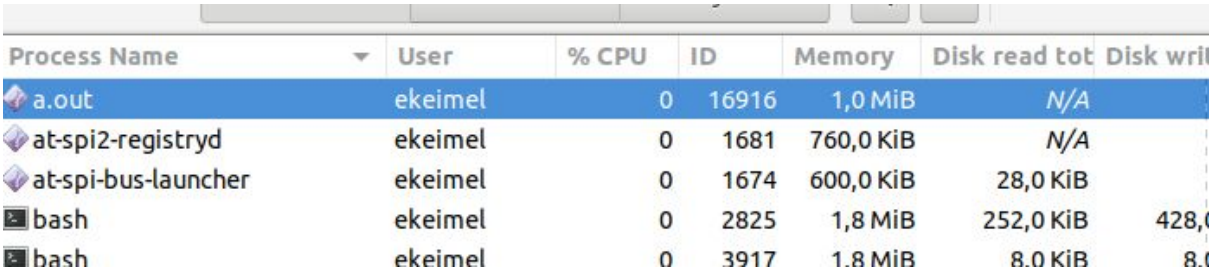
Conclusiones

La actual implementación del proxy tiene muchos aspectos de mejora, tanto en funcionalidad como en robustez. Muchos errores de implementación se detectaron cerca de la fecha límite de entrega, por lo que es esperable que la misma sea vulnerable a diversos errores. No obstante, casi la totalidad de los casos de prueba funcionan correctamente, notificando los errores al usuario aprovechando la funcionalidad del protocolo http y permitiendo que el proxy continue funcionando correctamente.

Ejemplos de prueba

Performance

- Si se alcanza el máximo de conexiones simultáneas permitidas, que está configurado en 50, se encuentra que la memoria utilizada no supera los 2MB:



Process Name	User	% CPU	ID	Memory	Disk read tot	Disk writ
a.out	ekeimel	0	16916	1,0 MiB	*N/A*	
at-spi2-registryd	ekeimel	0	1681	760,0 KiB	*N/A*	
at-spi-bus-launcher	ekeimel	0	1674	600,0 KiB	28,0 KiB	
bash	ekeimel	0	2825	1,8 MiB	252,0 KiB	428,(
bash	ekeimel	0	3917	1,8 MiB	8,0 KiB	8,(

- La velocidad de descarga promedio supera los 50MB sin utilizar transformaciones:

```
p://bar:2020/file.out -o file
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  953M    0  953M    0     0  51.5M      0 --:--:--  0:00:18 --:--:-- 51.6M
```

- Y se utilizan transformaciones:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  720M    0  720M    0     0  47.3M      0 --:--:--  0:00:15 --:--:-- 47.7M
```

Navegador

- Se probó de configurar el navegador (firefox) para que utilice el proxy. Luego, se intentó acceder a las siguientes direcciones:

- http://acme.com/
- http://www.columbia.edu/~fdc/sample.html
- http://kermitproject.org/newdeal/

Siendo esta última la más demandante por el grán número de imágenes que contiene.

- Otro caso interesante es repetir el acceso a la primer página ejecutando el proxy con los siguientes parámetros:

./httpd -t “sed s/ACME/ITBA/” -M “text/html”

Y se puede observar que los caracteres se reemplazan correctamente.
Adicionalmente, desde el cliente del administrador se puede ejecutar un setStatus 0 para apagar las transformaciones y ver el resultado.

Guia de instalación

- Para compilar el servidor proxy hay que correr en terminal:

~$ gcc helpers.c main.c proxyClientActiveSocket.c proxyOriginActiveSocket.c proxyPassiveSocket.c proxyTransformation.c selector.c buffer.c configPassiveSocket.c configActiveSocket.c protocolV1.c -pthread -Wall -o htttpd

Para ejecutarlo:

~$ ./httpd

En este caso, el proxy escuchará por defecto en el puerto 8080.

- Para compilar el cliente del administrador:

```
~$ gcc clientMain.c -pthread -lsctp -Wall -o client
```

Para ejecutarlo:

```
~$ ./client
```