ALGORITMOS Y ESTRUCTURA DE DATOS

TP 5. Trivial Web Server

Para entregar:

El objetivo de este TP es comprender los conceptos básicos de networking y programación de networking mediante Boost y LibcURL incorporando también conceptos básicos de HTTP (Hyper Text Transfer Protocol), el protocolo más usado en internet y el que diariamente utilizan los internautas al navegar por internet. Para ello vamos a implementar un Web Server Trivial que nos va a permitir "servir" una página Web.

Detalles de implementación:

1. Se crearán dos programas, un cliente y un servidor

   SERVIDOR

2. El servidor escucha el puerto 80 y, al realizar una conexión, se queda esperando un comando.
3. Al recibir el siguiente comando:

   GET */path/filename* HTTP/1.1 *CRLF*
   Host: 127.0.0.1 *CRLF*
   ...
   CRLF

   Donde *CRLF* implica que se envían los caracteres 0x0D y 0x0A en hexadecimal (13 y 10 decimal). Y */path/filename* implica un path a un archivo (*filename*) de tipo HTML (o JSON) que el servidor buscará dentro de la carpeta donde se ejecutó el comando.

   Los comandos que siguen al GET pueden llegar en diferente orden según el cliente que se conecte al servidor. Sin embargo, sólo deberán verificar la validez de la línea de Host.

   El servidor debe responder como se explica abajo e ignora cualquier otro comando recibido. El terminador *CRLF* indica que terminó una secuencia de comandos a interpretar. Solamente responderá si se cumple la secuencia de arriba. Ante cualquier otra secuencia el servidor no responderá nada por red aunque se espera que le muestre al usuario mediante la terminal que está recibiendo input de un cliente para indicarle que no se colgó y simplemente está ignorando dicho input ya que no se cumple la secuencia esperada.

4. De encontrarse el */path/filename* especificado arriba responderá:
   HTTP/1.1 200 OK
   Date: *Date* (Ej: Tue, 04 Sep 2018 18:21:19 GMT)
   Location: 127.0.0.1*/path/filename*
   Cache-Control: max-age=30
   Expires: *Date + 30s* (Ej: Tue, 04 Sep 2018 18:21:49 GMT)
   Content-Length: *filenameLength*
   Content-Type: text/html; charset=iso-8859-1
   filenameContent

5. De no encontrarse el */path/filename* especificado arriba responderá:
   HTTP/1.1 404 Not Found
   Date: *Date* (Ej: Tue, 04 Sep 2018 18:21:19 GMT)
   Cache-Control: public, max-age=30
   Expires: *Date + 30s* (Ej: Tue, 04 Sep 2018 18:21:49 GMT)
   Content-Length: 0
   Content-Type: text/html; charset=iso-8859-1

6. Nótese que lo incluido arriba es lo que habitualmente responde un Web Server al ser consultado por un navegador, por lo que una vez corriendo el servidor podrá probar en su navegador de preferencia navegando a la página 127.0.0.1*/path/filename* y el navegador les debería mostrar la página que están requiriendo, si existe y está formateada en HTML, o 404 Not Found en caso contrario.

CLIENTE

7. El cliente simplemente se conectará al servidor y le mandará los comandos explicados en 3.
8. Recibe por línea de comando el *host/path/filename* como único parámetro.
9. Una vez conectado esperará a la respuesta del servidor.
10. Finalmente guardará lo respondido por el servidor en un archivo contenido en la misma carpeta donde se encuentra.

En clase explicaremos en detalles los conceptos de *host, path, filename, CRLF*, cómo considerar los terminadores, y cómo leer y escribir archivos para facilitar la correcta implementación del programa. Se explicarán además las librerías que serán utilizadas para implementar ambos programas.
El programa que implementa el servidor utilizará los conceptos de web socket junto con la librería Boost (www.boost.org/). Por otro lado, el cliente utilizará la librería libcURL para realizar la conexión (https://curl.haxx.se/libcurl/).