1. **¿Cuál es la función de la capa de aplicación?**
   Se encarga de permitir que los usuarios y las aplicaciones accedan a la red. Proporciona la interfaz entre las aplicaciones y la red, implementando los protocolos necesarios para la comunicación (como HTTP y FTP). También maneja la interacción tanto con usuarios (a través de la interfaz de usuario) como con otras aplicaciones en entornos de comunicación máquina a máquina (M2M).
2. **Si dos procesos deben comunicarse:**
   a. ¿Cómo podrían hacerlo si están en diferentes máquinas?
      Si están en diferentes máquinas se deberían comunicar mediante envío de mensajes a través de una red con un protocolo determinado. Uno actuará como emisor que envía mensajes a la red y otro como receptor, y se irían alternando para responder a los mensajes.
   b. Y si están en la misma máquina, ¿qué alternativas existen?
      En la misma máquina se comunicaran por sistemas de comunicación interprocesos determinados por el S.O.
3. **Explique brevemente cómo es el modelo Cliente/Servidor. Dé un ejemplo de un sistema Cliente/Servidor en la "vida cotidiana" y un ejemplo de un sistema informático que siga el modelo Cliente/Servidor. ¿Conoce algún otro modelo de comunicación?**
   El modelo Cliente/Servidor es un esquema de comunicación en el que dos partes interactúan:
   - Cliente: Es la parte que realiza solicitudes de servicios o recursos.
   - Servidor: Es la parte que proporciona estos servicios o recursos solicitados.

   En este modelo, el cliente inicia la comunicación y espera una respuesta del servidor, quien maneja la solicitud y devuelve los datos o resultados al cliente. El servidor corre el servicio esperando de forma pasiva la conexión.
   Ejemplo en la vida cotidiana:
   Cajero automático (ATM):
   - Cliente: La persona que utiliza la tarjeta para solicitar dinero o consultar su saldo.
   - Servidor: El sistema bancario central que valida la solicitud y autoriza las transacciones.
4. **Describa la funcionalidad de la entidad genérica "Agente de usuario" o "User agent".**
   En HTTP la línea de cabecera User-agent especifica el agente de usuario, es decir, el tipo de navegador que está haciendo la solicitud al servidor.
   Si no hablamos de HTTP estamos hablando del navegador o software usado para acceder en nombre del usuario en la red.
5. **¿Qué son y en qué se diferencian HTML y HTTP?**

HTML (Hypertext Markup Language) es el lenguaje utilizado para estructurar y presentar el contenido en una página web, como texto, imágenes y enlaces.

HTTP (Hypertext Transfer Protocol) es el protocolo que gestiona la comunicación entre el navegador y el servidor, permitiendo la transferencia de archivos y datos en la web.

En resumen:

- HTML: Estructura y contenido de las páginas web.
- HTTP: Protocolo de comunicación para transferir esos datos entre el servidor y el navegador.

Se diferencian en que son para cosas totalmente distintas, es más, un documento HTML podría viajar en un respuesta HTTP.

6. HTTP tiene definido un formato de mensaje para los requerimientos y las respuestas. (Ayuda: apartado "Formato de mensaje HTTP", Kurose).

a. ¿Qué información de la capa de aplicación nos indica si un mensaje es de requerimiento o de respuesta para HTTP? ¿Cómo está compuesta dicha información?¿Para qué sirven las cabeceras?

Chequear: La información de la que nos habla es la que está en el encabezado del objeto HTTP, que tiene un método de pedido y un código de respuesta. Petición: GET / HTTP/1.1 Host: developer.mozilla.org Accept-Language: fr Respuesta:

1. HTTP/1.1 200 OK
2. Date: Sat, 09 Oct 2010 14:28:02 GMT
3. Server: Apache
4. Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
5. ETag: "51142bc1-7449-479b075b2891b"
6. Accept-Ranges: bytes
7. Content-Length: 29769
8. Content-Type: text/html
9. <!DOCTYPE html... (here comes the 29769 bytes of the requested web page)

Las cabeceras HTTP sirven para transmitir metadatos entre el cliente y el servidor. Estos metadatos incluyen:

- Información sobre el mensaje: Como el tipo de contenido (`Content-Type`), el tamaño del contenido (`Content-Length`), y la fecha de modificación (`Last-Modified`).
- Control de caché: Encabezados como `Cache-Control` y `Expires` gestionan la forma en que los recursos deben ser almacenados en caché.
- Autenticación y autorización: Encabezados como `Authorization` en los requerimientos permiten al cliente enviar credenciales, mientras que encabezados como `WWW-Authenticate` en las respuestas indican al cliente que se requiere autenticación.
- Redirección: Encabezados como `Location` en las respuestas indican al cliente que debe realizar una nueva solicitud a una URL diferente.

b.¿Cuál es su formato?

El formato de los headers suele ser:
El método - version o version - código de respuesta seguido de:
Nombre-Del-Encabezado: Valor

(Ayuda: https://developer.mozilla.org/es/docs/Web/HTTP/Headers)

c. Suponga que desea enviar un requerimiento con la versión de HTTP 1.1 desde curl/7.74.0 a un sitio de ejemplo como www.misitio.com para obtener el recurso /index.html. En base a lo indicado, ¿qué información debería enviarse mediante encabezados? Indique cómo quedaría el requerimiento.
curl -X GET "http://www.misitio.com/index.html" -H "Host: www.misitio.com" -H "User-Agent: curl/7.74.0" -H "Accept: */*"

7. Utilizando la VM, abra una terminal e investigue sobre el comando curl. Analice para qué sirven los siguientes parámetros (-I, -H, -X, -s).

`-I`: Este parámetro solicita solo los encabezados HTTP de la respuesta. Es útil para obtener información sobre los metadatos de una respuesta HTTP sin descargar el cuerpo del mensaje. *curl -I http://example.com*

`-H`: Este parámetro permite agregar encabezados personalizados a la solicitud HTTP. Puedes usarlo para enviar información adicional, como tokens de autenticación o tipo de contenido. *curl -H "Authorization: Bearer YOUR_TOKEN" http://example.com*

`-X`: Este parámetro especifica el método HTTP a utilizar (por defecto es GET). Puedes usarlo para realizar solicitudes con otros métodos como POST, PUT, DELETE, etc. *curl -X POST -d "param1=value1&param2=value2" http://example.com*

`-s`: Este parámetro activa el modo "silencioso". Suprime la barra de progreso y los mensajes de error, mostrando solo el contenido de la respuesta. Es útil cuando se desea evitar la salida adicional. *curl -s http://example.com*

8. Ejecute el comando curl sin ningún parámetro adicional y acceda a www.redes.unlp.edu.ar. Luego responda:

a. ¿Cuántos requerimientos realizó y qué recibió? Pruebe redirigiendo la salida (>) del comando curl a un archivo con extensión html y abrirlo con un navegador.

Un solo requerimiento GET, recibí un archivo HTML con bienvenida a Redes y Comunicaciones.

b. ¿Cómo funcionan los atributos href de los tags link e img en html?

Lo que hacen es literalmente hacer una solicitud HTTP/S para descargar los recursos, luego se aplican dependiendo del tipo de recurso.

c. Para visualizar la página completa con imágenes como en un navegador, ¿alcanza con realizar un único requerimiento?

No, requerir descargar el html, luego las imágenes.

d. ¿Cuántos requerimientos serían necesarios para obtener una página que tiene dos CSS, dos Javascript y tres imágenes? Diferencie cómo funcionaría un navegador respecto al comando curl ejecutado previamente.

La misma cantidad de elementos que fueron mencionados, los otros archivos han de ser solicitados manualmente con curl.

9. Ejecute a continuación los siguientes comandos: curl -v -s www.redes.unlp.edu.ar > /dev/null curl -I -v -s www.redes.unlp.edu.ar

a. ¿Qué diferencias nota entre cada uno?

El primer comando descarga todo el contenido de la respuesta (aunque este se descarta), mientras que el segundo comando solo solicita las cabeceras.

El primer comando utiliza `> /dev/null` para descartar el contenido, mientras que el segundo comando no necesita redirección ya que solo está interesado en las cabeceras.

b. ¿Qué ocurre si en el primer comando se quita la redirección a /dev/null? ¿Por qué no es necesaria en el segundo comando?

Si se quita la redirección a `/dev/null`, el comando `curl -v -s www.redes.unlp.edu.ar` mostrará todo el contenido de la respuesta HTTP en la salida estándar.

En el segundo comando no se descarga el contenido del cuerpo de la respuesta, por lo que no es necesario redirigir la salida a `/dev/null` para descartar el contenido del cuerpo. Las cabeceras son suficientes para este comando.

c. ¿Cuántas cabeceras viajaron en el requerimiento? ¿Y en la respuesta?

En el segundo comando el requerimiento viajaron 3: host user agent y accept, en la tercera viajaron muchas más: fecha, server, ultima modificación, ETag, accept-ranges, content-length, content-type.

10. ¿Qué indica la cabecera Date?

La fecha de origen del mensaje.

11. En HTTP/1.0, ¿cómo sabe el cliente que ya recibió todo el objeto solicitado de manera completa? ¿Y en HTTP/1.1?

HTTP/1.0

En HTTP/1.0, el cliente sabe que ha recibido todo el objeto solicitado a través del código de estado y el tamaño del contenido. Las principales características son:

- Código de Estado 200 OK: Indica que la solicitud fue exitosa y que el servidor está enviando la respuesta completa.
- Content-Length Header: El servidor incluye una cabecera `Content-Length` en la respuesta que especifica la longitud del cuerpo del mensaje en bytes.

HTTP/1.1

Además de `Content-Length`, HTTP/1.1 introduce `Transfer-Encoding: chunked` para permitir la transferencia en fragmentos. El cliente detecta el final del objeto cuando recibe un fragmento de tamaño cero.

12. Investigue los distintos tipos de códigos de retorno de un servidor web y su significado. Considere que los mismos se clasifican en categorías (2XX, 3XX, 4XX, 5XX).

1XX: Informativos

Estos códigos indican que la solicitud fue recibida y se está procesando.

- 100 Continue: El servidor ha recibido la solicitud inicial y el cliente debe continuar con la solicitud.
- 101 Switching Protocols: El servidor acepta cambiar el protocolo según lo solicitado por el cliente.

2XX: Éxito

Estos códigos indican que la solicitud fue recibida, entendida y aceptada correctamente.

- 200 OK: La solicitud se ha completado con éxito.
- 201 Created: La solicitud ha sido cumplida y ha resultado en la creación de un nuevo recurso.
- 202 Accepted: La solicitud ha sido aceptada para su procesamiento, pero no se ha completado.
- 204 No Content: La solicitud fue exitosa, pero no hay contenido para enviar en la respuesta.
- 205 Reset Content: La solicitud fue exitosa y el cliente debe reiniciar la vista que envió la solicitud.
- 206 Partial Content: El servidor está enviando solo una parte del recurso, como se solicitó.

3XX: Redirección

Estos códigos indican que se necesita realizar más acciones para completar la solicitud.

- 300 Multiple Choices: Hay múltiples opciones para el recurso solicitado.
- 301 Moved Permanently: El recurso solicitado ha sido movido de manera permanente a una nueva URL.
- 302 Found: El recurso solicitado se encuentra temporalmente en una URL diferente.
- 303 See Other: El cliente debe hacer una nueva solicitud a la URL especificada.
- 304 Not Modified: Indica que el recurso no ha sido modificado desde la última solicitud del cliente.
- 305 Use Proxy: El recurso solicitado debe ser accedido a través de un proxy especificado.
- 307 Temporary Redirect: El recurso está temporalmente en una URL diferente, pero la misma solicitud HTTP debe ser utilizada.
- 308 Permanent Redirect: Similar al 301, pero se asegura que el método HTTP no cambie.

4XX: Errores del Cliente

Estos códigos indican que hubo un error por parte del cliente.

- 400 Bad Request: La solicitud no se puede procesar debido a una sintaxis incorrecta.
- 401 Unauthorized: La solicitud requiere autenticación, pero no se proporcionó o fue incorrecta.
- 402 Payment Required: Reservado para futuras aplicaciones, generalmente relacionado con pagos.
- 403 Forbidden: El servidor entiende la solicitud, pero se niega a autorizarla.
- 404 Not Found: El recurso solicitado no se pudo encontrar en el servidor.
- 405 Method Not Allowed: El método especificado en la solicitud no es permitido para el recurso solicitado.
- 406 Not Acceptable: El recurso solicitado no es capaz de generar contenido aceptable según los encabezados de la solicitud.
- 408 Request Timeout: El servidor agotó el tiempo de espera en la espera de una solicitud del cliente.
- 429 Too Many Requests: El cliente ha enviado demasiadas solicitudes en un tiempo determinado.

5XX: Errores del Servidor

Estos códigos indican que hubo un error en el servidor al procesar la solicitud.

- 500 Internal Server Error: Error genérico del servidor, ocurre cuando se encuentra un problema inesperado.
- 501 Not Implemented: El servidor no reconoce el método de solicitud o no tiene la capacidad para cumplirla.
- 502 Bad Gateway: El servidor, actuando como un gateway o proxy, recibió una respuesta no válida del servidor upstream.
- 503 Service Unavailable: El servidor no está disponible temporalmente, generalmente debido a sobrecarga o mantenimiento.
- 504 Gateway Timeout: El servidor, actuando como un gateway o proxy, no recibió una respuesta a tiempo del servidor upstream.

13. Utilizando curl, realice un requerimiento con el método HEAD al sitio www.redes.unlp.edu.ar e indique:

a. ¿Qué información brinda la primera línea de la respuesta?

La versión http y el código (de éxito en este caso).

b. ¿Cuántos encabezados muestra la respuesta?

6

c. ¿Qué servidor web está sirviendo la página?

Apache/2.4.56 (Unix)

d. ¿El acceso a la página solicitada fue exitoso o no?

Si (200)

e. ¿Cuándo fue la última vez que se modifica la página?

El sábado 19 de marzo de 2023 a las 19:04:46 GMT.

f. Solicite la página nuevamente con curl usando GET, pero esta vez indique que quiere obtenerla sólo si la misma fue modificada en una fecha posterior a la que efectivamente fue modificada. ¿Cómo lo hace? ¿Qué resultado obtuvo? ¿Puede explicar para qué sirve?

Para hacerlo agregamos al header de nuestro requerimiento un -H "If-Modified-Since: *fecha-actual".* Lo que haría que si no se modificó se retorna un código 304 not modified que sirve para saber si tiene que descargar la página devuelta o usar la cacheada.

14. Utilizando curl, acceda al sitio www.redes.unlp.edu.ar/restringido/index.php y siga las instrucciones y las pistas que vaya recibiendo hasta obtener la respuesta final. Será de utilidad para resolver este ejercicio poder analizar tanto el contenido de cada página como los encabezados.

Primera respuesta
Acceso restringido, para acceder al contenido es necesario autenticarse. Para obtener los datos de acceso seguir las instrucciones detalladas en www.redes.unlp.edu.ar/obtener-usuario.php

Respuesta de obtener usuario:

Para obtener el usuario y la contraseña haga un requerimiento a está página seteando el encabezado 'Usuario-Redes' con el valor 'obtener'.

Respuesta con el encabezado:

Bien hecho! Los datos para ingresar son:

- Usuario: redes
- Contraseña: RYC

Ahora vuelve a la página inicial con los datos anteriores.

Pista: Investigue el uso del encabezado Authorization para el método Basic. El comando base64 puede ser de ayuda!

echo -n 'redes:RYC' | base64 -> Conseguimos los datos en base64 (cmVkZXM6UllD).

Y tiramos el siguiente comando:
curl -H "Authorization: Basic cmVkZXM6UllD" -I www.redes.unlp.edu.ar/restringido/index.php y eso te retornara un respuesta con un link a http://redes.unlp.edu.ar/restringido/the-end.php

Acá terminamos el ejercicio:

¡Felicitaciones, llegaste al final del ejercicio!

Fecha: 2024-09-04 13:57:27

Verificación:
dceb896eff0805e37f4db62a6116d26a6b3e4202ab0d9e7ff91781e0dcfc3be3

15. Utilizando la VM, realice las siguientes pruebas:

a. Ejecute el comando 'curl www.redes.unlp.edu.ar/extras/prueba-http-1-0.txt' y copie la salida completa (incluyendo los dos saltos de línea del final).

GET /http/HTTP-1.1/ HTTP/1.0

User-Agent: curl/7.38.0

Host: www.redes.unlp.edu.ar

Accept: */*

b. Desde la consola ejecute el comando telnet www.redes.unlp.edu.ar 80 y luego pegue el contenido que tiene almacenado en el portapapeles. ¿Qué ocurre luego de hacerlo?

Se devuelve un html y se cierra la conexión.

c. Repita el proceso anterior, pero copiando la salida del recurso /extras/prueba-http-1-1.txt. Verifique que debería poder pegar varias veces el mismo contenido sin tener que ejecutar el comando telnet nuevamente.

Ahora te retorna la página html usando http 1.1 y se mantiene abierta la conexión

16. En base a lo obtenido en el ejercicio anterior, responda:

a. ¿Qué está haciendo al ejecutar el comando telnet?

Establece una conexión tcp en el puerto 80.

b. ¿Qué método HTTP utilizó? ¿Qué recurso solicitó?

GET, se solicitó el html de la página.

c. ¿Qué diferencias notó entre los dos casos? ¿Puede explicar por qué?

El primero queda abierto y el segundo no.

d. ¿Cuál de los dos casos le parece más eficiente? Piense en el ejercicio donde analizó la cantidad de requerimientos necesarios para obtener una página con estilos, javascripts e imágenes. El caso elegido, ¿puede traer asociado algún problema?

El caso más eficiente es HTTP/1.1. Esto se debe a que HTTP/1.1 permite el uso de conexiones persistentes, lo que reduce la necesidad de abrir y cerrar conexiones TCP para cada recurso. Esto resulta en menos sobrecarga y tiempos de respuesta más rápidos, especialmente cuando se trata de páginas web que requieren múltiples solicitudes para obtener todos los recursos necesarios (estilos, scripts, imágenes, etc.).

17. En el siguiente ejercicio, exploramos la diferencia entre los métodos POST y GET utilizando la VM y la herramienta Wireshark. Para ello, siga los siguientes pasos:

Captura de Paquetes:

- **Capture los paquetes utilizando la interfaz con IP 172.28.0.1. (Menú "Capture -> Options". Luego seleccione la interfaz correspondiente y presione Start).**
- **Para que el analizador de red solo muestre los mensajes del protocolo HTTP, introduzca la cadena `http` (sin comillas) en la ventana de especificación de filtros de visualización**

(display-filter). De lo contrario, vería todo el tráfico que puede capturar la placa de red.

- **De los paquetes capturados, el que esté seleccionado se mostrará en detalle en la sección que está justo debajo. Como solo estamos interesados en HTTP, ocultaremos la información que no es relevante para esta práctica (Información de trama, Ethernet, IP y TCP). Despliegue la información correspondiente al protocolo HTTP bajo la leyenda "Hypertext Transfer Protocol".**

Borrado de Caché del Navegador:

- **Para borrar la caché del navegador, vaya al menú "Herramientas -> Borrar historial reciente". Alternativamente, puede utilizar Ctrl+F5 en el navegador para forzar la petición HTTP evitando el uso de caché.**

Visualización Simplificada de la Comunicación HTTP:

- **Para ver de forma simplificada el contenido de una comunicación HTTP, haga clic derecho sobre un paquete HTTP del flujo capturado y seleccione la opción "Follow TCP Stream".**

Tareas a Realizar:

- **Abra un navegador e ingrese a la URL: www.redes.unlp.edu.ar e ingrese al enlace en la sección "Capa de Aplicación" llamado "Métodos HTTP". En la página mostrada, visualice los dos nuevos enlaces llamados "Método GET" y "Método POST". Ambos muestran un formulario.**
- **Analice el código HTML de la página.**
- **Utilizando el analizador de paquetes Wireshark, capture los paquetes enviados y recibidos al presionar el botón "Enviar".**
- **¿Qué diferencias detectó en los mensajes enviados por el cliente?**
- **¿Observó alguna diferencia en el navegador si se utiliza un mensaje u otro?**

En la página en sí el método GET y POST son especificados en el formulario.

Los mensajes enviados al cliente tendran los datos del formulario en el header, o cuerpo respectivamente.

El navegador mostrara los datos en la url si se usa GET.

18. Investigue cuál es el principal uso que se le da a las cabeceras Set-Cookie y Cookie en HTTP y qué relación tienen con el funcionamiento del protocolo HTTP.

Cabecera Set-Cookie

Uso: La cabecera Set-Cookie es enviada por el servidor al cliente (navegador) para establecer una cookie. Esta cookie puede almacenar información como identificadores de sesión, preferencias del usuario, o datos temporales.

Formato: La cabecera tiene la siguiente estructura:

Set-Cookie: nombre=valor; Expires=fecha; Path=/; Domain=ejemplo.com; Secure; HttpOnly; SameSite=Strict

a. nombre=valor: Define el nombre y el valor de la cookie.
b. Expires: Indica la fecha de caducidad de la cookie.
c. Path: Define el ámbito de la cookie, es decir, las rutas en el servidor donde la cookie es válida.
d. Domain: Especifica el dominio para el cual la cookie es válida.
e. Secure: Indica que la cookie solo debe ser enviada a través de conexiones HTTPS.
f. HttpOnly: Evita que la cookie sea accesible a través de JavaScript, ayudando a prevenir ataques XSS (Cross-Site Scripting).
g. SameSite: Controla si la cookie se envía en solicitudes cross-site, ayudando a prevenir ataques CSRF (Cross-Site Request Forgery).

Cabecera Cookie

Uso: La cabecera Cookie es enviada por el cliente al servidor en las solicitudes HTTP subsiguientes. Contiene las cookies que el servidor estableció previamente mediante Set-Cookie y que son relevantes para la solicitud actual.

Formato: La cabecera tiene la siguiente estructura:

Cookie: nombre1=valor1; nombre2=valor2

La cabecera lista todas las cookies que el cliente ha almacenado y que son pertinentes para el dominio y la ruta especificados.

19. ¿Cuál es la diferencia entre un protocolo binario y uno basado en texto? ¿De qué tipo de protocolo se trata HTTP/1.0, HTTP/1.1 y HTTP/2?

Protocolos Basados en Texto

- Representación: Los protocolos basados en texto transmiten la información en formato legible por humanos. Utilizan caracteres alfanuméricos y otros símbolos para representar datos y comandos.
- El formato textual es fácil de entender y depurar, pero puede ser menos eficiente en términos de tamaño y velocidad de procesamiento en comparación con los protocolos binarios.

Protocolos Binarios

- Representación: Los protocolos binarios transmiten la información en formato binario, es decir, en una secuencia de bytes. Estos protocolos pueden ser más compactos y eficientes en términos de procesamiento y tamaño de los mensajes porque no requieren la conversión entre texto y datos binarios.
- Ejemplo: HTTP/2 es un protocolo binario. En HTTP/2, tanto las solicitudes como las respuestas se codifican en un formato binario, lo que permite una mayor eficiencia en la transmisión de datos y una mejor gestión de múltiples solicitudes y respuestas en una sola conexión.

HTTP/1.0, HTTP/1.1 y HTTP/2

- HTTP/1.0: Es un protocolo basado en texto. La comunicación entre el cliente y el servidor se realiza mediante mensajes de texto, con encabezados y cuerpos de mensaje en formato legible por humanos.
- HTTP/1.1: También es un protocolo basado en texto, aunque introdujo varias mejoras respecto a HTTP/1.0, como el soporte para conexiones persistentes y pipelining de solicitudes.
- HTTP/2: Es un protocolo binario. Introdujo un nuevo formato de codificación binario para solicitudes y respuestas, mejorando la eficiencia en la transmisión de datos y permitiendo características avanzadas como multiplexación de flujos, compresión de encabezados y gestión eficiente de conexiones.

20. Responder las siguientes preguntas:

a. ¿Qué función cumple la cabecera Host en HTTP 1.1? ¿Existía en HTTP 1.0? ¿Qué sucede en HTTP/2?

(Ayuda: https://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html para HTTP/2)

HTTP/1.1: La cabecera Host es obligatoria y permite identificar el dominio en servidores que usan alojamiento virtual.

HTTP/1.0: No era obligatorio.

HTTP/2: La cabecera Host sigue siendo necesaria y se utiliza para la misma finalidad: especificar el nombre del host y el puerto. HTTP/2 se basa en el mismo concepto de virtual hosting que HTTP/1.1. Sin embargo, HTTP/2 utiliza una compresión de cabeceras (HPACK) para reducir la cantidad de datos que se envían en las solicitudes y respuestas, lo que puede reducir la redundancia y mejorar la eficiencia en la transmisión de datos.

b. En HTTP/1.1, ¿es correcto el siguiente requerimiento?

GET /index.php HTTP/1.1 User-Agent: curl/7.54.0

Si es correcto

c. ¿Cómo quedaría en HTTP/2 el siguiente pedido realizado en HTTP/1.1 si se está usando https?

GET /index.php HTTP/1.1 Host: www.info.unlp.edu.ar

Anda a chequear esto pero:
:method: GET

:scheme: https

:authority: www.info.unlp.edu.ar

:path: /index.php

Según el chat GPT!

Ejercicio de Parcial

curl -X ?? www.redes.unlp.edu.ar/??

> HEAD /metodos/ HTTP/??

> Host: www.redes.unlp.edu.ar

> User-Agent: curl/7.54.0

< HTTP/?? 200 OK

< Server: nginx/1.4.6 (Ubuntu)

< Date: Wed, 31 Jan 2018 22:22:22 GMT

< Last-Modified: Sat, 20 Jan 2018 13:02:41 GMT

< Content-Type: text/html; charset=UTF-8

< Connection: close

a. ¿Qué versión de HTTP podría estar utilizando el servidor?

1.0 o superior basado en los headers, pero la conexión se cierra después de la request así que tal vez sea la 1.0 específicamente.

b. ¿Qué método está utilizando? Dicho método, ¿retorna el recurso completo solicitado?

HEAD, Si retorna todo el recurso solicitado.

c. ¿Cuál es el recurso solicitado?

l/métodos/

d. ¿El método funcionó correctamente?

Si, código 200 ok.

e. Si la solicitud hubiera llevado un encabezado que diga: If-Modified-Since: Sat, 20 Jan 2018 13:02:41 GMT ¿Cuál habría sido la respuesta del servidor web? ¿Qué habría hecho el navegador en este caso?

Hubiese retornado un 304 y el navegador usará la versión cacheada si hay.