Práctica 2 - HTTP

1. ¿Cuál es la función de la capa de aplicación?

La capa de aplicación es la encargada de permitir que las aplicaciones de los usuarios finales (como navegadores, correos electrónicos, etc.) se comuniquen a través de la red. Proporciona protocolos y servicios que permiten el intercambio de datos entre dispositivos conectados a Internet. Esta capa está siempre implementada en los sistemas terminales.
Básicamente, traduce lo que las aplicaciones necesitan para que la red lo pueda transmitir de manera adecuada. Por ejemplo, HTTP permite la transferencia de archivos web, SMTP envía correos, y DNS resuelve nombres de dominio a direcciones IP.

2. Si dos procesos deben comunicarse:
a. ¿Cómo podrían hacerlo si están en diferentes máquinas?
Los procesos de dos sistemas terminales diferentes se comunican entre ellos intercambiando mensajes a través de la red de computadoras. Un proceso emisor crea y envía mensajes a la red; un proceso receptor recibe y posiblemente responde devolviendo mensajes.

b. Y si están en la misma máquina, ¿qué alternativas existen?
Cuando los procesos se ejecutan en el mismo sistema terminal, pueden comunicarse entre sí mediante la comunicación de procesos aplicando las reglas gobernadas por el sistema operativo del sistema terminal donde se están ejecutando.

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?

En la arquitectura cliente-servidor siempre existe un host activo, denominado servidor, que da servicio a muchos otros hosts, que son los clientes. Los hosts clientes pueden estar activos siempres o de forma intermitente. En esta arquitectura los clientes no se comunican nunca entre sí.
Ejemplo en la vida cotidiana: Pedís una pizza por teléfono (cliente) y la pizzería (servidor) te la prepara y te la envía. Vos hacés la solicitud y ellos te responden con el producto

Ejemplo informático: Un ejemplo clásico es la Web en la que un servidor web siempre activo sirve las solicitudes de los navegadores que se ejecutan en los hosts clientes. Cuando un servidor web recibe una solicitud de un objeto de un hosts cliente, responde enviándole el objeto solicitado.

Otro modelo de comunicación puede ser una arquitectura P2P (peer-to-peer), donde existe una minima (o ninguna) dependencia de la infraestructura de servidores siempre activos. En su lugar, la aplicación explota la comunicación directa entre parejas de hosts conectados de forma intermitente, conocidos como peers (pares). Estos pares no son propiedad del servidor del servicio, sino que son las computadoras de escritorio y notebooks controladas por sus usuarios.

4. Describa la funcionalidad de la entidad genérica “Agente de usuario” o “User agent”.
En un mensaje de solicitud HTTP, la linea de cabecera user-agent especifica el agente del usuario, es decir, el tipo de navegador que está haciendo la solicitud al servidor. Por ejemplo: Mozilla/4.0.
El servidor podría responder con distintos objetos dependiendo del tipo de user-agent (el mismo URL direcciona a cada una de las versiones).

5. ¿Qué son y en qué se diferencian HTML y HTTP?

El Protocolo de transferencia de hipertexto (**HTTP**, HyperText Transfer Protocol) es el protocolo de la capa de aplicación de la Web y se encuentra en el corazón de la Web. Está definido en los documentos RFC 1945 y RFC2616. HTTP se implementa en dos programas, un programa cliente y un programa servidor. Estos dos programas se ejecutan en dos sistemas terminales diferentes y se comunican entre sí enviando mensajes HTTP. El protocolo define la estructura de esos mensajes y cómo el cliente y el servidor intercambian estos mensajes.

Una **página web**, consta de objetos. Un objeto es un archivo (HTML, una imagen JPG, etc.) que puede direccionarse mediante un URL único. **HTML (HyperText Markup Language)** es solo un lenguaje de marcado que Define una estructura básica y un código (denominado código HTML) para la presentación de contenido de una página web

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?

La primera linea de un mensaje HTTP es la que indica si se trata de un requerimiento o de una respuesta.

En caso de ser un requerimiento, se la denomina “linea de solicitud” y su formato es

MÉTODO	URL	VERSIÓN
GET	unlp.edu.ar/institucional	HTTP/1.1

En caso de ser una respuesta, se la denomina “línea de estado inicial” y su formato es:

VERSIÓN	CÓDIGO	MENSAJE
HTTP/1.1	200	OK

Las **cabeceras** sirven para enviar información adicional tanto en una request como en una response. Son clave para que el cliente y el servidor puedan interpretar y manejar correctamente los datos que están intercambiando.

b. ¿Cuál es su formato?

Las cabeceras (en inglés headers) HTTP permiten al cliente y al servidor enviar información adicional junto a una petición o respuesta. Una cabecera de petición esta compuesta por su nombre (no sensible a las mayusculas) seguido de dos puntos ':', y a continuación su valor (sin saltos de línea).

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.

GET /index.html HTTP/1.1
Host: www.misitio.com
User-Agent: curl/7.74.0
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).

`curl` es una herramienta para transferir datos entre un cliente y un servidor usando varios protocolos (HTTP, FTP, SMTP, etc.). Permite enviar o recibir datos sin interacción del usuario.

Parámetros:

- `-I`: Solicita solo los encabezados del documento, sin el contenido.
- `-H`: Añade encabezados personalizados a la solicitud HTTP.
- `-X`: Define el método HTTP (como GET, POST, PUT) a utilizar en la solicitud.
- `-s`: Modo silencioso, no muestra el progreso ni errores.

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.

Realicé un requerimiento solo, y recibí todo el documento html de la url pasada como parámetro al comando.

b. ¿Cómo funcionan los atributos href de los tags link e img en html?
No funcionan ni se visualizan.

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

No. El navegador además de solicitar el HTML, hace más requerimientos para cargar los recursos referenciados en ese HTML (como el .css el .js, un .jpg, o .png, etc…)

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.

Habría que realizar 8 requerimientos (el inicial para el HTML + los 7 mencionados).

Como mencioné en el punto anterior, el **navegador** solicita el HTML, y luego automaticamente identifica los recursos referenciados y hace los requerimientos necesarios para obtenerlos y mostrarlos correctamente.

Con el comando **curl**, una vez hecho el requerimiento inicial del HTML, luego habría que ejecutar un comando curl por cada uno de los recursos necesarios. El URL de cada recurso hay que buscarlo de manera manual en el HTML original y extraerlo de allí.

9. Ejecute a continuación los siguientes comandos:

curl -v -s www.redes.unlp.edu.ar > /dev/null

```
redes@debian:~/Desktop$ curl -v -s www.redes.unlp.edu.ar > /dev/null
*   Trying 172.28.0.50:80...
* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)
> GET / HTTP/1.1
> Host: www.redes.unlp.edu.ar
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Mon, 23 Sep 2024 23:52:11 GMT
< Server: Apache/2.4.56 (Unix)
< Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT
< ETag: "1322-5f7457bd64f80"
< Accept-Ranges: bytes
< Content-Length: 4898
< Content-Type: text/html
<
{ [4898 bytes data]
* Connection #0 to host www.redes.unlp.edu.ar left intact
```

curl -I -v -s www.redes.unlp.edu.ar

```
redes@debian:~/Desktop$ curl -I -v -s www.redes.unlp.edu.ar
*   Trying 172.28.0.50:80...
* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)
> HEAD / HTTP/1.1
> Host: www.redes.unlp.edu.ar
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Date: Mon, 23 Sep 2024 23:53:25 GMT
Date: Mon, 23 Sep 2024 23:53:25 GMT
< Server: Apache/2.4.56 (Unix)
Server: Apache/2.4.56 (Unix)
< Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT
Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT
< ETag: "1322-5f7457bd64f80"
ETag: "1322-5f7457bd64f80"
< Accept-Ranges: bytes
Accept-Ranges: bytes
< Content-Length: 4898
Content-Length: 4898
< Content-Type: text/html
Content-Type: text/html

<
* Connection #0 to host www.redes.unlp.edu.ar left intact
```

a. ¿Qué diferencias nota entre cada uno?

El primer comando redirige la respuesta del comando a dev/null para no mostrar la respuesta

El segundo, solo solicita el encabezado (parametro -I) y no el contenido completo de la respuesta.

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 borra la redirección, se muestra en consola la respuesta a la request, que es un documento HTML.

En el segundo comando no es necesaria porque utilizó el parametro -I

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

En el requerimiento son 3:

```
Host: www.redes.unlp.edu.ar
User-Agent: curl/7.74.0
Accept: */*
```

En la respuesta fueron 8:

```
HTTP/1.1 200 OK
Date: Mon, 23 Sep 2024
Server: Apache/2.4.56
Last-Modified: Sun, 19 Mar 2023
ETag
Accept-Ranges: bytes
Content-Length: 4898
Content-Type: text/html
```

10. ¿Qué indica la cabecera Date?

Indica la fecha y hora en la que el servidor envió la respuesta

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

En HTTP/1.0, el **servidor** cierra la conexión TCP al finalizar (Transmission Control Protocol).

En HTTP 1.1:

- Si el servidor incluye la cabecera `Content-Length`, indica exactamente cuántos bytes tiene el cuerpo de la respuesta.
- Si el servidor utiliza **Transfer-Encoding: chunked**, el contenido de la respuesta se envía en partes o "chunks". Cada chunk está precedido por un número en hexadecimal que indica el tamaño de ese fragmento. El cliente sabe que la transmisión terminó cuando recibe un **chunk de tamaño 0**, lo que indica el final del contenido.

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).

1. Respuestas informativas (100–199),
2. Respuestas satisfactorias (200–299),
3. Redirecciones (300–399),
4. Errores de los clientes (400–499),
5. y errores de los servidores (500–599).

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?

```
redes@debian:~$ curl -I www.redes.unlp.edu.ar
HTTP/1.1 200 OK
Date: Wed, 25 Sep 2024 00:14:59 GMT
Server: Apache/2.4.56 (Unix)
Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT
ETag: "1322-5f7457bd64f80"
Accept-Ranges: bytes
Content-Length: 4898
Content-Type: text/html
```

Indica:

- Versión del HTTP
- Código de retorno
- Mensaje descriptivo

b. ¿Cuántos encabezados muestra la respuesta?

Muestra 7 encabezados. (o 8? **PREGUNTAR**)

c. ¿Qué servidor web está sirviendo la página?

Apache/2.4.56 (Unix)

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

Sí, fue exitoso, lo indica el código **200 OK**

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

Lo indica el
Last-Modified: Sun, 19 Mar 2023 19:04:46 GMT

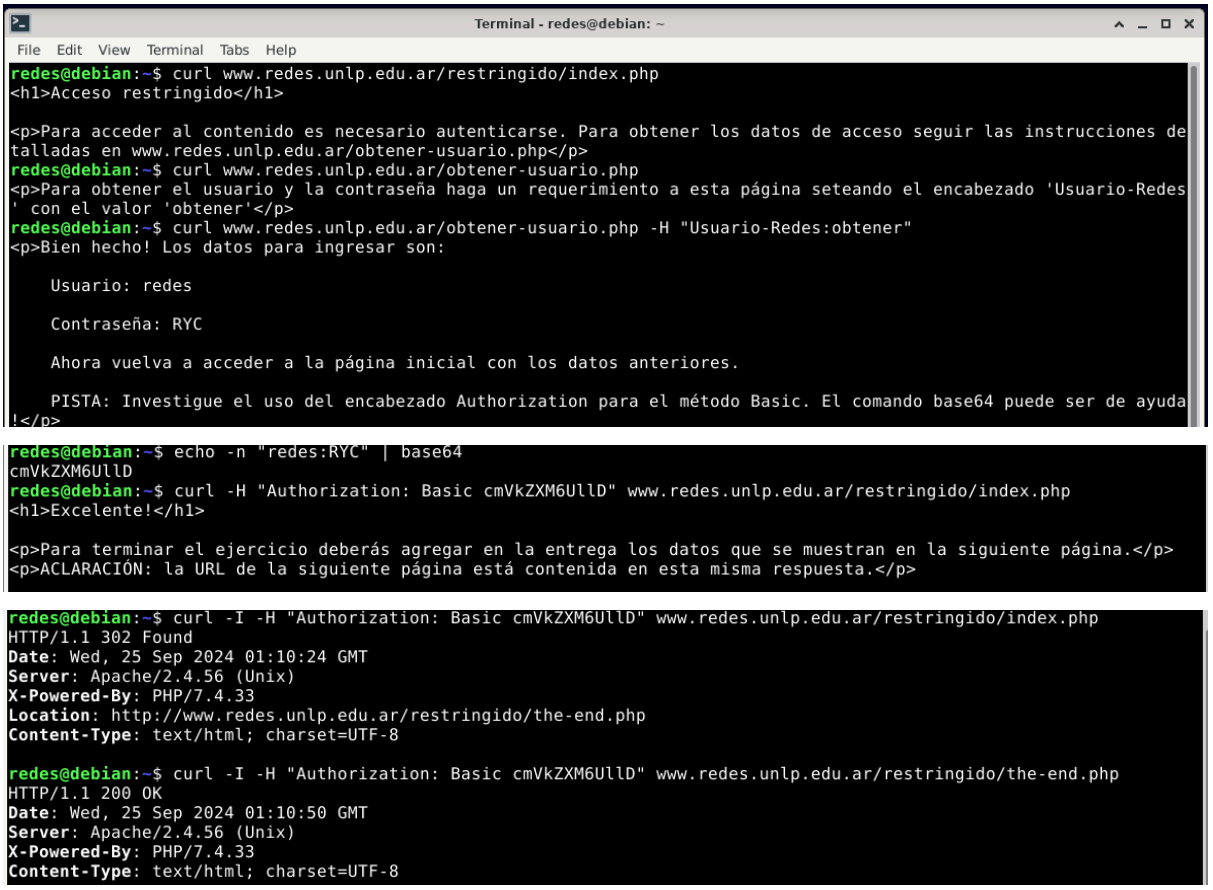
f. Solicite la página nuevamente con curl usando GET, pero esta vez indique que quiere obtenerla sólo si la misma fue modificada en una fecha posterior a la que efectivamente fue modificada. ¿Cómo lo hace? ¿Qué resultado obtuvo? ¿Puede explicar para qué sirve?

```
redes@debian:~$ curl -z "Mon, 20 Mar 2023 00:00:00 GMT" www.redes.unlp.edu.ar
redes@debian:~$ curl -z "Sun, 19 Mar 2023 00:00:00 GMT" www.redes.unlp.edu.ar
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>.::.Redes y Comunicaciones.::.Facultad de Inform&aacute;tica.::.UNLP.::.</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="">
    <meta name="author" content="">

    <!-- Le styles -->
    <link href="./bootstrap/css/bootstrap.css" rel="stylesheet">
    <link href="./css/style.css" rel="stylesheet">
    <link href="./bootstrap/css/bootstrap-responsive.css" rel="stylesheet">
```

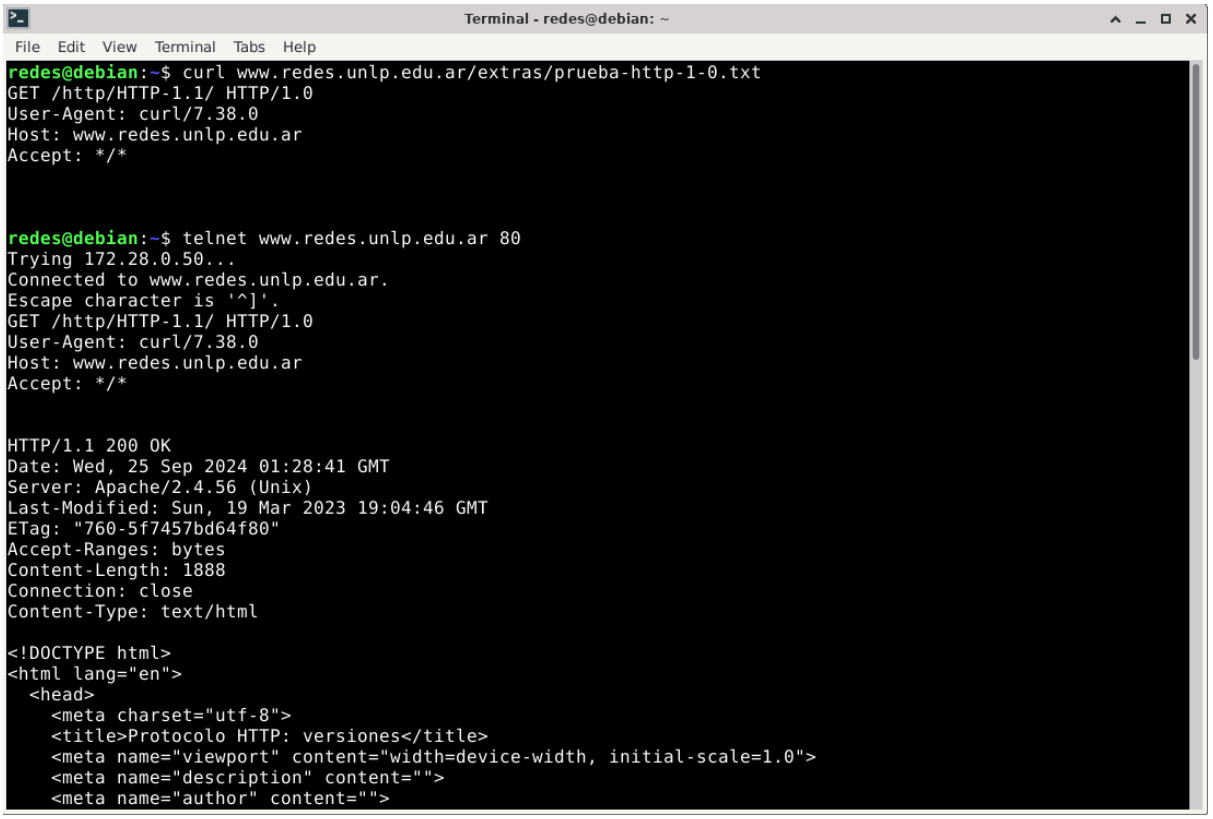
Lo hice con el parámetro -z (*--time-cond*), que sirve puntualmente para lo solicitado en la consigna. Sirve para solicitar un archivo que haya sido modificado después de la hora y fecha indicadas, o uno que haya sido modificado antes de esa hora. La expresión de fecha puede ser todo tipo de cadenas de fecha o, si no coincide con ninguna interna, se trata como un nombre de archivo y curl intenta obtener la fecha de modificación (mtime) de ese archivo.

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.



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).

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?



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.

Mismo resultado, pero no cierra la conexión.
El porqué está explicado en el punto 16.

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

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

Telnet es un protocolo de red de aplicación que permite la comunicación de usuario con un computador remoto a través de una interfaz basada en texto. Telnet crea una conexión de terminal virtual, permitiendo a los usuarios acceder a las aplicaciones en un equipo remoto. En particular en el ejemplo del ejercicio, nos conectamos al puerto 80.

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

Utilizó el método HTTP GET, y solicitó el recurso /http/HTTP-1.1/ HTTP/1.0

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

La diferencia es que en el caso del punto C no se cierra la conexión y se pueden hacer sucesivas solicitudes sin tener que volver a ejecutar telnet.

Esta diferencia se da por el protocolo HTTP.
En el caso de HTTP/1.0 la conexión se cierra una vez finalizada la operación, y en el caso de HTTP/1.1 la conexión persiste.

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?

HTTP/1.1 es más eficiente al no tener que solicitar, abrir y cerrar la conexión para cada uno de las comunicaciones o mensajes. Cada una de estas consume tiempo y recursos a la red.

Si bien es más eficiente, introduce un problema asociado denominado Head of Line Blocking. El HOL blocking ocurre en este contexto cuando una solicitud se queda esperando su respuesta (por latencia o procesamiento lento) y bloquea las solicitudes posteriores en la misma conexión persistente, ya que en HTTP/1.1 las respuestas se devuelven en el mismo orden en que fueron solicitadas. Si una respuesta tarda más de lo esperado, las otras quedan atrapadas detrás limitando el rendimiento de la red.

Esto fue una de las razones que llevó a la evolución hacia HTTP/2, donde se introdujo la multiplexación para permitir que múltiples solicitudes/respuestas se procesen en paralelo en una misma conexión sin este tipo de bloqueo.

17. En el siguiente ejercicio veremos la diferencia entre los métodos POST y GET. Para ello, será necesario utilizar la VM y la herramienta Wireshark. Antes de iniciar considere:

■ 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 sólo nos muestre los mensajes del protocolo http introduciremos la cadena 'http' (sin las comillas) en la ventana de especificación de filtros de visualización (display-filter). Si no hiciéramos esto veríamos todo el tráfico que es capaz de capturar nuestra placa de red. De los paquetes que son capturados, aquel que esté seleccionado será mostrado en forma detallada en la sección que está justo debajo. Como sólo estamos interesados en http ocultaremos toda la información que no es relevante para esta práctica (Información de trama, Ethernet, IP y TCP). Desplegar la información correspondiente al protocolo HTTP bajo la leyenda "Hypertext Transfer Protocol".

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

■ En caso de querer ver de forma simplificada el contenido de una comunicación http, utilice el botón derecho sobre un paquete HTTP perteneciente al flujo capturado y seleccione la opción Follow TCP Stream.

a. Abra un navegador e ingrese a la URL: www.redes.unlp.edu.ar e ingrese al link en la sección "Capa de Aplicación" llamado "Métodos HTTP". En la página mostrada se visualizan dos nuevos links llamados: Método GET y Método POST. Ambos muestran un formulario como el siguiente:

b. Analice el código HTML

El código HTML es exáctamente el mismo, pero en el <form> uno indica el method="GET" y el otro "POST"

c. Utilizando el analizador de paquetes Wireshark capture los paquetes enviados y recibidos al presionar el botón Enviar.

GET

Hypertext Transfer Protocol
GET /http/metodos-lectura-valores.php?form_nombre=Maximo&form_apellido=Cossetti&form_mail=tortuga%40maritima.com&form_sexo=sexo_masc&form_pass=simu1234&form_confirma_mail=on HTTP/1.1\r\n

Host: www.redes.unlp.edu.ar\r\n

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n

Accept-Encoding: gzip, deflate\r\n

Connection: keep-alive\r\n

Referer: http://www.redes.unlp.edu.ar/http/metodo-get.html\r\n

Upgrade-Insecure-Requests: 1\r\n

POST

Hypertext Transfer Protocol
POST /http/metodos-lectura-valores.php HTTP/1.1\r\n

Host: www.redes.unlp.edu.ar\r\n

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n

Accept-Language: en-US,en;q=0.5\r\n

Accept-Encoding: gzip, deflate\r\n

Content-Type: application/x-www-form-urlencoded\r\n

Content-Length: 131\r\n

Origin: http://www.redes.unlp.edu.ar\r\n

Connection: keep-alive\r\n

Referer: http://www.redes.unlp.edu.ar/http/metodo-post.html\r\n

Upgrade-Insecure-Requests: 1\r\n

d. ¿Qué diferencias detectó en los mensajes enviados por el cliente?

En el GET se envían los datos del formulario como parametros, mientras que en el POST se envían en el cuerpo del mensaje.

Adicionalmente:
En el POST debe indicar Contend-Type, Content-Lenght y Origin, en el GET no es necesario.

e. ¿Observó alguna diferencia en el browser si se utiliza un mensaje u otro?

En el GET los datos (parámetros) se pueden visualizar en la URL.
En el POST la URL no cambia.

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.

La cabecera de respuesta HTTP **Set-Cookie** se usa para enviar cookies desde el servidor al agente de usuario, así el agente de usuario puede enviarlos de vuelta al servidor.

El encabezado **Cookie** de una solicitud HTTP contiene cookies HTTP almacenadas y enviadas previamente por el servidor con el encabezado (header) Set-Cookie
Los encabezados Cookie puede ser omitidos por completo, si las preferencias de privacidad del navegador están configuradas para bloquearlos, por ejemplo.

Las cabeceras Set-Cookie y Cookie se usan principalmente **para mantener la sesión y persistencia** de datos entre el cliente (user-agent) y el servidor en el protocolo HTTP, que por naturaleza es sin estado (**stateless**). Esto significa que cada solicitud HTTP es independiente y no tiene contexto de solicitudes anteriores, por lo que estas cabeceras son fundamentales para crear una experiencia web que "recuerde" información entre solicitudes.

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?

La diferencia principal entre un protocolo binario y uno basado en texto está en cómo se formatean y transmiten los datos:

- **Protocolo basado en texto:** Utiliza caracteres legibles por humanos (como ASCII o UTF-8) para transmitir información. La ventaja es que es fácil de leer y depurar para los desarrolladores. La desventaja es que suelen ser más lentos y menos eficientes en cuanto a tamaño de los datos transmitidos.
- **Protocolo binario:** Utiliza datos en formato binario (bits), que no son legibles por humanos sin herramientas especializadas. Estos protocolos suelen ser más eficientes y rápidos, ya que el procesamiento y transmisión de los datos en binario es más directo y compacto.

Tipos de HTTP:

- HTTP/1.0 y HTTP/1.1: Son protocolos basados en texto. Los mensajes HTTP están formados por texto claro (cabeceras, métodos, códigos de estado) que los humanos pueden leer fácilmente.
- HTTP/2: Es un protocolo binario. Fue diseñado para mejorar la eficiencia y el rendimiento, incluyendo la multiplexación y la compresión de cabeceras, lo que reduce la latencia y acelera la transferencia de datos, algo más difícil de lograr con un protocolo basado en texto.

En resumen:

- HTTP/1.0 y HTTP/1.1 son protocolos de texto.
- HTTP/2 es un protocolo binario.
- Otros ejemplos de binario: SSH o FTP

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: 4 Redes y Comunicaciones LINTI - UNLP https://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html para HTTP/2)

El encabezado Host debe enviarse obligatoriamente en todas las solicitudes HTTP/1.1. Un código de error 400 (Petición mala) debería enviarse a cualquier solicitud HTTP/1.1 que no contiene o contiene más de un encabezado Host.

El encabezado de solicitud Host **especifica el nombre de dominio del servidor** (para hosting virtual), y (opcionalmente) el número de puerto TCP en el que el servidor esta escuchando.

Si, existía en HTTP/1.0. Si bien HTTP 1.0 no requiere oficialmente un encabezado Host, lo acepta y no está de más agregar uno, y muchas aplicaciones (proxies) esperan ver el encabezado Host independientemente de la versión del protocolo.

En HTTP/2 aunque el encabezado de host HTTP/1 se reemplaza efectivamente por el pseudoencabezado :authority, también se puede enviar un encabezado de host en la solicitud.

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

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

No, no es correcto, falta la cabecera Host que es obligatoria en HTTP/1.1.

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

Quedaría:

:method: GET

:path: /index.php

:scheme: https

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

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?

Podría estar utilizando HTTP/1.0, porque cerró la conexión.
En HTTP/1.1 y HTTP/2 la conexión persiste, y para que no persista, se debe especificarlo explicitamente.

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

Está utilizando el método HEAD.
Este método no retorna el recurso completo, solicita solo las cabeceras.

c. ¿Cuál es el recurso solicitado?
Solicitó el recurso **/metodos/**

d. ¿El método funcionó correctamente?

Sí, el código de estado es 200 OK, lo que indica que la solicitud fue exitosa.

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?

El encabezado HTTP If-Modified-Since hace que la solicitud sea condicional: el servidor devuelve el recurso solicitado, con un status 200 solo si se modificó por última vez después de la fecha indicada. Si el recurso no se modificó desde entonces, la respuesta es un 304 sin cuerpo.
En este caso devuelve 304 (porque a la condición es por MAYOR (>) y no MAYOR O IGUAL (>=)).