Práctica 2 - HTTP.

Introducción.

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

- La función de la Capa de Aplicación consiste en brindar las funcionalidades y servicios necesarios para que las aplicaciones de usuario se comuniquen entre sí mediante intercambio de mensajes.

3. Si dos procesos deben comunicarse:

a. ¿Cómo podrían hacerlo si están en diferentes máquinas?

- A través de una red, utilizando los diferentes protocolos de red.

b. Y si están en la misma máquina, ¿qué alternativas existen?

- Se pueden comunicar mediante algún mecanismo del SO.
- Aunque también se podría hacer con los protocolos de red, sólo que nada saldría de la PC.

4. Explique brevemente cómo es el modelo Cliente/Servidor. De 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 pocas palabras, el modelo Cliente/Servidor consiste en un conjunto de hosts llamados clientes los cuáles hacen peticiones de servicios que ofrecen los servidores.

- Un ejemplo de Cliente/Servidor de la “vida cotidiana” podría ser un mismo restaurante, los comensales (Clientes) van al establecimiento, se sientan en la mesa y le solicitan a algún mozo (Servidor) un platillo.
- Un ejemplo de un sistema informático que siga el modelo Cliente/Servidor podría ser la que siguen el navegador web y un servidor web, cuando yo quiero navegar por internet mediante algún navegador, éste solicitará al servidor web de la página que deseo ver los documentos necesarios para que me pueda renderizar el sitio.

5. Describa la funcionalidad de la entidad genérica “Agente de usuario” o “User agent”.

- La entidad genérica de *User Agent* consiste en una abstracción del usuario, es una interfaz por la cuál el usuario se comunica con la entidad servidor. Algunos ejemplos de User Agent son: Navegador Web, Gestor de Correos Electrónicos, etc.

HTTP.

6. Observe el índice de la RFC-2616, busque el apartado donde se describe el requerimiento y la respuesta. ¿Qué son y en qué se diferencian HTML y HTTP? ¿En qué entidad ubicaría a HTML?

- HTML es un documento asociado a las páginas web, básicamente, sirve para estructurar los elementos de ellas. Es la parte esencial de las páginas, sin dichos documentos, nuestros navegadores no podrían mostrarnos las páginas. En cambio, HTTP es un protocolo propio de la Capa de Aplicación para el transporte e intercambio de información. Con HTTP, el navegador podría solicitar el HTML al servidor web de la página correspondiente.
- HTML estaría ubicado en la entidad *body* de los mensajes (solicitud o respuesta), el documento se encodea con algún método en particular.

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 que sirve para transferir datos mediante una URL.

- El parámetro **-I** realiza una solicitud HTTP utilizando el método HEAD, es decir, la respuesta no tendrá la entidad *body*.
- El parámetro **-H** sirve para especificar headers para la solicitud HTTP. Es útil para enviar, por ejemplo, tokens de validación.
- El parámetro **-X** sirve para especificar cuál será el método a usar en la solicitud HTTP.
- El parámetro **-s** significa modo silencioso, no se va a ver el progreso, mensajes de error ni otra información adicional.

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.

- Se realizó un requerimiento. Se recibió un html de una página de la materia, la cuál contiene un poco de los materiales que se dictan.

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

- El atributo href en etiquetas *link* está relacionado a la dirección de archivos externos y relacionados al html, como un .css, .js, etc.
- En relación a *img*, href sirve para indicar la dirección de una imagen a mostrar, podría ser, por ejemplo, una imagen almacenada en otro sitio web.

c. Para visualizar la página completa con imágenes como en un navegador, ¿alcanza con realizar un único requerimiento? ¿Cuántos requerimientos serían necesarios para obtener una página que tiene dos CSS, dos Javascript y tres imágenes? Diferencie como funcionaría un navegador respecto al comando curl ejecutado previamente.

- No, en este caso con curl, necesitamos si o si realizar más requerimientos para mostrarlo como en un navegador.
- Para obtener dos .css, dos .js y tres imágenes, necesitamos realizar 7 solicitudes.
- El navegador hace todo automáticamente, hace las solicitudes HTTP y obtiene todos los elementos necesarios para renderizar una página web (documento HTML junto con los otros necesarios). Mientras que con curl, es todo “manualmente”, nosotros

debemos hacer la primera solicitud e ir haciendo más solicitudes por cada archivo externo relacionado (los especificados con <*link*>)

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**

Observe la salida y luego repita la prueba, pero previamente inicie una nueva captura en wireshark. Utilice la opción Follow Stream. ¿Qué se transmitió en cada caso?

- En el primer caso, en la captura se puede ver que en la solicitud HTTP se utilizó GET, y la respuesta HTTP fue un 200 con el HTML de la página solicitada.
- En el segundo caso, la solicitud HTTP se hizo con el método HEAD, esto significa que en la respuesta HTTP no vendrá nada en el body, sólo se obtienen los encabezados (más allá de si la solicitud sale bien).

¿A qué se debió esta diferencia entre lo que se transmitió y lo que se mostró en pantalla?

- En ese caso, no pudimos ver el contenido del body, ya que este fue redirigido a /dev/null.

10. Investigue cómo define las cabeceras la RFC.

a. ¿Establece todas las cabeceras posibles?.

- No realmente. La RFC-2616 define estándares que incluyen los encabezados más comunes que pueden ser usados en los mensajes HTTP.
- Nosotros también podemos definir nuestros propios encabezados.

b. ¿Cuántas cabeceras viajaron en el requerimiento y en la respuesta del ejercicio anterior?

- En el request viajaron 3 encabezados (User-Agent, Host, Accept)
- En la respuesta viajaron 7 encabezados (Date, Server, Last-Modified, ETag, Accept-Ranges, Content-Length, Content-Type).

c. ¿La cabecera Date es una de las definidas en la RFC? ¿Qué indica?

- Si, en los encabezados generales.

- Básicamente, indica la fecha y hora en la que se originó el mensaje.

11. 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?

- HTTP/1.1 200 OK

b. ¿Cuántos encabezados muestra la respuesta?

- En total, la respuesta tiene 7 encabezados.

c. ¿Qué servidor web está sirviendo la página?

- En el encabezado *Server* podemos ver que el servidor web es Apache.

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

- Si, lo podemos ver porque el código de la respuesta fue un 200 (el que se muestra cuando la operación sale bien).

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

- Según el encabezado *Last-Modified*, la última vez que se modificó fue un 19 de Marzo del 2023 a las 19:04:46hs

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?

- curl -H “If-Modified-Since: Mon, 28 Aug 2023 12:00:00 GMT” www.redes.unlp.edu.ar.
- Efectivamente, no obtuve el html porque no fue modificado desde la fecha especificada.
- En pocas palabras, dicho encabezado es utilizado por la web-caché, la caché almacena los requerimientos que solicité previamente (más eficiente), pero si el documento fue modificado, la caché debería hacer la petición al servidor, actualizar su copia y pasarmela a mi.

12. En HTTP/1.0, ¿cómo sabe el cliente que ya recibió todo el objeto solicitado completamente? ¿Y en HTTP/1.1?

- HTTP/1.0 no utiliza conexiones persistentes. Entonces, al final de cada solicitud-respuesta, la conexión TCP se cierra. Esto significa que el servidor enviará todos los recursos solicitados por el cliente y después la conexión se cerrará.
- HTTP/1.1 usa conexiones persistentes. Esto significa que en una misma conexión pueden haber más de un par solicitud-respuesta. El cliente sabe si recibió todo el objeto solicitado gracias a los headers, ejemplo, el header *Content-Length* nos indica de cuanto es el tamaño de lo que se envía en el *body*, entonces el cliente leerá hasta llegar al número indicado por dicho header.

13. Investigue los distintos tipos de códigos de retorno de un servidor web y su significado en la RFC. ¿Qué parte se ve principalmente interesada de esta información, cliente o servidor? ¿Es útil que esté detallado y clasificado en la RFC?. Dentro de la VM, ejecute los siguientes comandos y evalúe el estado que recibe.

- curl -I http://unlp.edu.ar
  - HTTP/1.1 301 Moved Permanently
- curl -I www.redes.unlp.edu.ar/restringido/index.php
  - HTTP/1.1 401 Unauthorized
- curl -I www.redes.unlp.edu.ar/noexiste
  - HTTP/1.1 404 Not Found

Tipos de código de retorno:

- 1xx → Información de estado.
- 2xx → Solicitud exitosa.
- 3xx → Redirección.
- 4xx → Error del cliente.
- 5xx → Error del servidor.

- A la parte que le interesa esto es al Cliente, ya que él sabe de qué modo operar según el tipo del error.
- Si es útil que esté detallado y especificado en la RFC. Recordemos que la RFC es un estándar aprobado, por lo tanto esto se seguirá por todos, y además es una fuente de información para que los programadores sepamos qué código de retorno debe enviar el servidor, o qué debemos hacer si nos llega X código de retorno.

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.

1. curl www.redes.unlp.edu.ar/restringido/index.php
2. curl www.redes.unlp.edu.ar/obtener-usuario.php
3. curl -H “Usuario-Redes: obtener” www.redes.unlp.edu.ar/obtener-usuario.php
4. credenciales=echo -n “redes:RYC” | base64
5. curl -H “Authorization: Basic ${credenciales}” www.redes.unlp.edu.ar/restringido.index.php
6. curl -H “Authorization: Basic ${credenciales}” www.redes.unlp.edu.ar/restringido.the-end.php

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?

- Se obtuvo un 200 junto con el HTML de la página solicitada.

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 telnet nuevamente.

- //CONSULTAR\\ Exacto. Esto sucede porque HTTP/1.1 utiliza conexiones persistentes.

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

- ¿Qué está haciendo al ejecutar el comando telnet? ¿Qué lo diferencia con curl?
  - //CONSULTAR\\ Telnet es similar a curl, pero lo que hace es dejar que el usuario por sí mismo ingrese la petición HTTP.
- Observe la definición de método y recurso en la RFC. Luego responda, ¿Qué método HTTP utilizó?
  - El método que se utilizó fue GET.
- ¿Qué recurso solicitó?
  - Se solicitó el recurso /http/HTTP/1.1
- ¿Qué diferencias notó entre los dos casos? ¿Puede explicar por qué?
  - La diferencia es que con HTTP/1.0, cuando se obtuvo la respuesta, la conexión termina. Pero con HTTP/1.1, después de obtener la respuesta podía hacer otra petición.
  - Esto sucede porque HTTP/1.1 utiliza conexiones persistentes por defecto.
- ¿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?
  - Claramente, el más eficiente es HTTP/1.1
  - Pensar que si necesito realizar otras peticiones, con conexiones no persistentes debería abrir y cerrar varias conexiones, lo cuál consume demasiados recursos.

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 caché 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.

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

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

- Un mensaje utilizó el método GET y en otro POST.
    - En el caso del método GET, los parámetros se enviaron en la URL.
    - Para el mensaje que usó el método POST, los parámetros se enviaron en el *body* del mensaje.

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

- Para GET, como los parámetros se enviaron vía URL, dichos parámetros se pueden ver en el navegador.

- En POST no, al no enviarse en la URL, no se ven en el navegador.

18. HTTP es un protocolo stateless, para sortear esta carencia muchos servicios se apoyan en el uso de Cookies. ¿En qué RFC se definió dicha funcionalidad? Investigue cuál es el principal uso que se le da a Set-Cookie y Cookie en HTTP. ¿Qué atributo de la RFC original fue en parte aprovechado para la implementación?

- El uso de Cookies para compensar que HTTP es stateless fue definido en la RFC-6256. Esta solución lleva consigo dos encabezados para HTTP:
  - **Set-Cookie:** El servidor crea la cookie, se almacena en su BD y se le envía al cliente. El navegador la recibe y la almacena en su BD.
  - **Cookie:** El navegador envía la cookie que el servidor le dió, de esta forma el servidor obtiene los datos que mantuvo de dicho usuario.
- //CONSULTAR\\ El atributo de la RFC aprovechado por esto fue el de Connection.

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?

- Básicamente, si nosotros hiciéramos una captura del tráfico, si el protocolo fuera basado en texto, podríamos leer los mensajes, caso contrario sería si el protocolo fuera binario, además de que su parseo es muchísimo más eficiente para los dispositivos.
- En concreto, las versiones de HTTP son:
  - HTTP/1.0 → Basado en texto
  - HTTP/1.1 → Basado en texto
  - HTTP/2.0 → Binario

20. Analice de qué se tratan las siguientes características de HTTP/2: stream, frame, server-push

- **Stream:** El stream es una pseudo-conexión entre el cliente y el servidor. En un stream se permite multiplexar mensajes, es decir, podría viajar la petición y respuesta en ella.

- **Frame:** Es la unidad mínima, básicamente, los streams se dividen en frames. Cada frame (según el tipo) pueden transportar diferentes cosas, por ejemplo, un frame podría transportar los encabezados y otro el cuerpo del mensaje HTTP.
- **Server-Push:** Volvamos al ejemplo del ejercicio en el que el cliente hacía una petición al servidor y después tenía que hacer otras peticiones para traer imágenes, hojas de estilos, programas JavaScript, etc. El servidor se podría anticipar a nuestras peticiones y enviarnos esos recursos sin que necesariamente se los solicite el cliente.

21. 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)

- El uso de la cabecera *Host* en HTTP/1.1 es para identificar el host específico del servidor (recordar que un servidor tiene muchos servidores virtuales, con esto identificamos uno en específico).
- Existía en HTTP/1.0, pero no era requerido, ya que se suponía que los servidores eran un host único.
- En HTTP/2 es similar a HTTP/1.1, pero en lugar de tener un encabezado llamado *Host*, tiene uno llamado *Authority*.

b. ¿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

:method: GET
:path: /index.php
:scheme: http
:authority: www.info.unlp.edu.ar