**Práctica 2**

**Capa de Aplicación – HTTP**

**17**

**Requerimientos**

**1.** Para realizar esta práctica deberá descargar la máquina virtual provista. Puede ver la URL de descarga en el sitio de la cátedra en https://catedras.info.unlp.edu.ar/.

Una vez descargado el archivo, haciendo doble click en el mismo debería abrirse un cuadro de diálogo que permita configurar algunos parámetros del sistema. Se pueden aceptar los valores por defecto haciendo simplemente click en Importar.

Se recomienda hacer un snapshot de la VM antes de empezar a usarla, para poder volver atrás en caso de que algo deje de funcionar.

Los datos de acceso a la máquina virtual son:

Usuario: redes

Contraseña: redes

Para acceso con permisos de administrador, usar el comando sudo

Aclaracion: El dominio redes.unlp.edu.ar solo existe dentro de la VM provista por la cátedra, no es válido en Internet, lo que implica que todos los ejercicios de esta práctica y las siguientes en las que se utilice dicho dominio sólo podrán ser resueltos dentro dicha VM.

**Introducción**

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

La capa de aplicación define las aplicaciones de red y los servicios de Internet estándar que puede utilizar un usuario.

* Provee servicios de comunicación a los usuarios (Capa 8) y a las aplicaciones, incluye las aplicaciones mismas.
* Existe modelo de comunicación Machine to machine (M2M), no hay usuarios (personas).
* Interfaz con el usuario -User Interface (UI)- u otras aplicaciones/servicios (Navegadores, mail, etc) que se conectan con el servidor mediante protocolo http.
* Las aplicaciones que usan la red pertenecen a esta capa.
* Los protocolos que implementan las aplicaciones también.
* Existen aplicaciones que NO son de red que deben trabajar con aplicaciones/servicios para lograr acceso a la red.

**3. Si dos procesos deben comunicarse:**

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

Si el software de una aplicación está distribuido entre dos o más sistemas, es decir, los procesos están en diferentes hosts, se comunican a través de intercambio de mensajes. Para esto hay dos tipos de procesos:

* Proceso Cliente: proceso que inicia la comunicación (crea y envía mensajes sobre la red)
* Proceso servidor: proceso que espera por ser contactado (recibe los mensajes y, posiblemente responde enviando mensajes)

Nota: Aplicaciones con arquitectura P2P tienen procesos clientes y procesos Servidores (cumplen ambos roles).

Mediante protocolos de capa aplicación se definen:

* Tipos de mensajes intercambiados (mensajes de requerimiento y respuesta)
* Sintaxis de los tipos de mensajes: qué campos de mensajes y cómo éstos son delimitados.
* Semántica de los campos, es decir, el significado de la información en los campos
* Reglas para cuándo y cómo los procesos envían y responden a mensajes

**Sockets**

Son fijos, no importa el process id. Se establecen para procesos estándar.

* Los procesos envían/reciben mensajes a/desde su socket
* Los sockets son análogos a puertas
* Proceso transmisor saca mensajes por la puerta
* Proceso transmisor confía en la infraestructura de transporte al otro lado de la puerta la cual lleva los mensajes al socket en el proceso receptor

API: (1) debemos elegir el protocolo de transporte; (2) podemos definir algunos parámetros

Direccionamiento de procesos

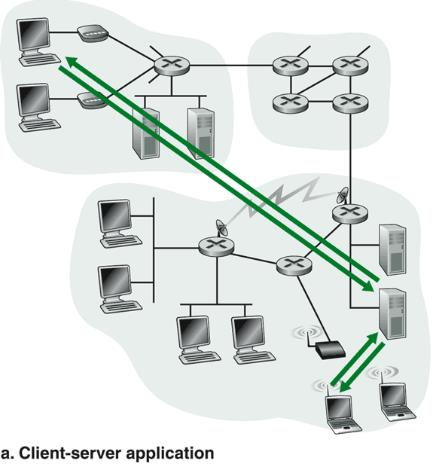
* Para que un proceso reciba un mensaje, éste debe tener un identificador
* Un host tiene una dirección IP única de 32 bits.
* ¿Es suficiente la dirección IP para identificar un proceso en un host? No, muchos procesos pueden estar corriendo en el mismo host. El identificador incluye la dirección IP y un número de puerta asociado con el proceso en el host.

Ejemplo de números de puertas: Servidor HTTP: 80, Servidor de Mail: 25

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

Dentro de la máquina dos procesos se comunican usando comunicación entre proceso (definida por SO).

**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?**



**Arquitectura Cliente-servidor**

**Servidor:**

* Computador siempre on
* Dirección IP permanente (esta identifica al equipo dentro de la red), ya que el cliente demanda el servicio

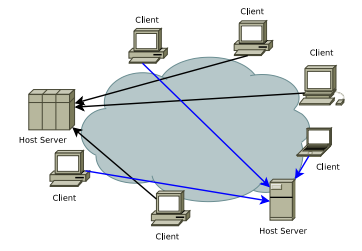
**Cliente:**

* Se comunica con un servidor
* Puede ser conectado intermitentemente
* Puede tener direcciones IP dinámicas o conectividad intermitente
* No se comunican directamente entre sí (dos clientes puros)

**Modelo de Cliente/Servidor**

* Modelo de Carga compartida. No todo se corre en el servidor, hay parte en el cliente (Ej: Que el navegador “Dibuje” las imágenes).
* Idea inicial: el cliente pone procesamiento de interfaz, el servidor pone el resto del procesamiento.
* Existen modelos en varios tiers (2 tiers, 3 tier o multi-tier).
* El servidor corre el servicio esperando de forma pasiva la conexión. Siempre está esperando clientes y soporta muchos a la vez. Ej: Google. Solo hace algo si alguien se conecta.
* Los clientes se conectan al servidor y se comunican a través de este.
* Ejemplo: File Server vía NFS o FTP.
* Modelo asimétrico 1 a N, M a N (donde M <N). Pocos.

El servidor está esperando quien se quiera conectar mediante protocolo. Un cliente puede conectarse a más de un servidor.

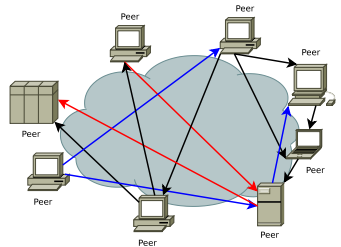


Otros modelos de comunicación:

* Peer-to-peer (P2P)
* Híbridos de cliente-servidor y P2P
* Modelo Mainframe (dumb client).

**Arquitectura P2P (Peer-to-Peer) Pura**

* El cliente puede ser servidor o tener ambos roles (Torrent)
* Servidor no siempre on
* Sistemas terminales arbitrarios se comunican directamente
* Pares se conectan intermitentemente y cambian sus direcciones IP
* Ejemplo: Gnutella
* Altamente escalable pero difícil de administrar.

Modelo de Carga completamente compartida y distribuida.

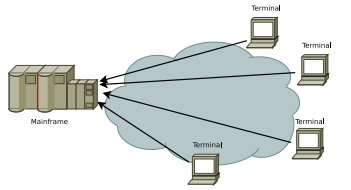
* Los peers (participantes) pueden cumplir rol de cliente, servidor o ambos en un instante.
* Sistema escalable en cuanto a rendimiento ya que al sumar miembros se suma cómputo, no es solo para datos.
* Sistema no escalable en cuanto a administración. Como está descentralizado es complicado de administrar. Si, por ejemplo, una persona apaga su equipo, es un nodo menos, lo que representa un problema.
* Ejemplo: redes legadas para compartir archivos: Novell Lite, Microsoft Windows for Workgroup basado en LAN Manager (sobre NetBEUI), algo más actual: Gnutella, Bittorrent. (servicio de file sharing totalmente P2P)
* Modelo asimétrico N a N. Ej: Carpeta compartida en cada compu. Todos se conectan con todos. Pueden tener rol cliente y servidor.
* Todos “hablan” con todos y pueden cambiar el rol.

**Híbridos de cliente-servidor y P2P**

Ej: Telefonía. Hay una etapa intermedia.

Napster

* Transferencia de archivos P2P
* Búsqueda de archivos centralizada:
  + Pares registran contenidos en servidor central
  + Pares consultan algún servidor central para localizar el contenido
  + Modelo de Carga compartida y distribuida.
  + Los peers (participantes) pueden cumplir rol de cliente, servidor o ambos en un instante.
  + Existen diferentes tipos de nodos con diferentes roles.
  + Hay nodos centrales donde se registra la información y al resto de los nodos.
  + Sistema escalable en cuanto a rendimiento.
  + Sistema más escalable que P2P puro.
  + Ejemplos: eDonkey (y sus variantes aMule, eMule), Napster, IM, Skype.
  + Moldeo asimétrico M a N.
  + Skype: Había nodos que registraban quien estaba online y quien no.



**Modelo de Mainframe Centralizado**

* + Modelo de Carga concentrada (Todo el procesamiento se hace en un equipo gigante, son las computadoras viejas).
  + El cliente es “tonto” (dumb) solo corre la comunicación y la interfaz física con el usuario (ej. terminal). No tiene nada de procesamiento.
  + El servidor pone todo el procesamiento.
  + Modelo antiguo que resurge con thin-clients.
  + Modelo puro: el mainframe decide cuando le da el control al cliente y maneja el diálogo de las comunicaciones.
  + El Cliente se ejecuta en el mainframe.
  + Ejemplo: Sistema SNA con Mainframe IBM S/370 y terminales 3270 (las terminales verdes del antiguo sistema de alumnos !!!).
  + Servidor de Terminales: X11, VNC, Cytrix Metaframe, VMWARE PCoIP, LTS, Virtualizacion.

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

Es una interfaz entre el usuario y la aplicación de red, sería lo que ve el usuario.

Por ejemplo: en la WEB, el agente de usuario es el navegador, el cual permite al usuario visualizar las

páginas WEB e interactuar con los elementos de la misma. El navegador es un proceso que envía/recibe

mensajes por medio de un socket y además brinda la interfaz al usuario.

**HTTP**

**6. Observe el índice de la RFC2616, 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 (HyperText Markup Language o Lenguaje de Marcado de Hipertexto)* es un lenguaje que permite describir texto presentado de forma estructurada y agradable, con enlaces que conducen a otros documentos o fuentes de información relacionadas, y con inserciones multimedia.

Mientras que *HTTP (Hypertext Transfer Protocol o Protocolo de transferencia de hipertexto)* es un protocolo orientado a transacciones que sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador web o un spider) se lo conoce como "user agent". *A la información transmitida se la llama recurso y se la identifica mediante un localizador uniforme de recursos (URL*). En pocas palabras es el que pone en comunicación a una computadora con los servidores que brindan Internet.

Se diferencian en propósito, el primero sirve para estructurar contenido en una página web, mientras que el segundo sirve para enviar y recibir datos entre software que funciona mediante este protocolo.

HTML viajaría por el body del protocolo, como contenido de un response probablemente.

**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 desde, o hacia un servidor, usando alguno de los protocolos soportados (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP). El comando está diseñado para trabajar sin interacción del usuario.

● -I, --head: Recupera únicamente el header de HTTP en el request (Información del recurso)

● -H, --header <cabecera/archivo>: Sirve para agregar información personalizada al header del request, es decir, la solicitud, que se va a realizar.

También para reemplazar los valores por defecto que utilizará curl, de esta forma también se pueden limpiar algunos valores predeterminados. Ej. –H “Host:” ; así se limpia el dato Host del header.

● -X, --request <comando>: Especifica un método de request específico para usar en la comunicación (el default es GET).

● -s, --silent: Modo silencio. No muestra los mensajes de error ni la barra de progreso.

–z: se quiere obtener la página solo si su última modificación fue realizada en una fecha menor o igual a la dada. Por ej: curl –z ‘Wed, 17 Sep 2017’ [www.redes.unlp.edu.ar](http://www.redes.unlp.edu.ara/))

**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 único request (GET /), ya que curl no pide los recursos externos, y se recibió el contenido HTML de la página solicitada (URI /).

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

● La etiqueta <link> define una relación entre un documento y un recurso externo. El atributo href del elemento link especifica la URL del recurso en cuestión.

● La etiqueta <img> define una imagen en una página HTML. El atributo src del elemento img especifica la URL de la imagen.

La mayoría de los navegadores cuando encuentran estos elementos en el documento HTML realizan una solicitud automática al servidor para obtenerlos.

**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 funciona un navegador respecto al comando curl ejecutado previamente.**

Es necesario realizar más de un requerimiento

* 1 request para la página HTML.
* 2 requests para los 2 recursos CSS.
* 2 requests para los 2 recursos JS.
* 3 requests para las 3 imágenes.
* 1 request para el favicon. (Imagen favorita, asociada a la página) **VER SI VA INCLUIDA CON HTML O ES OTRA**

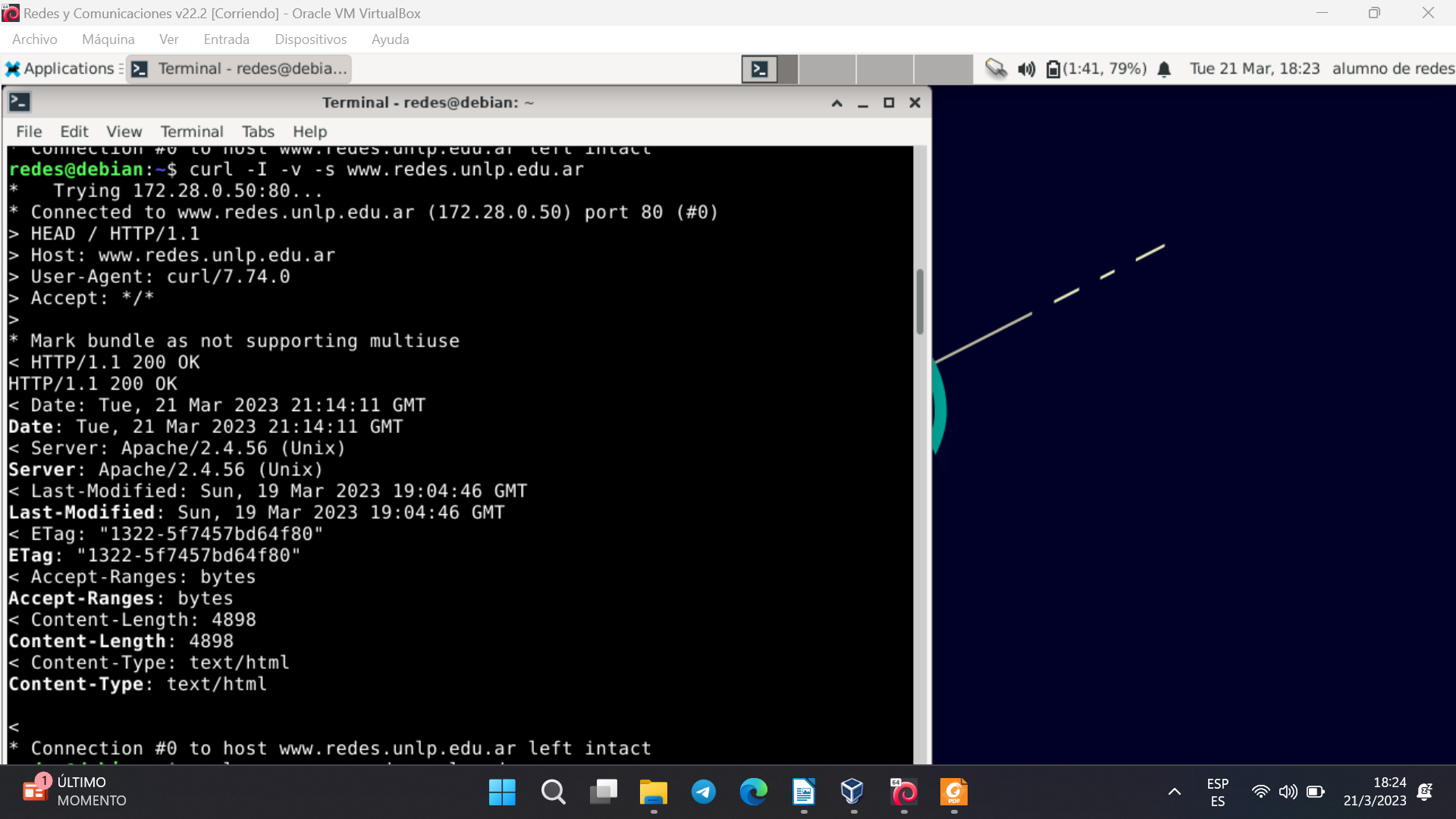
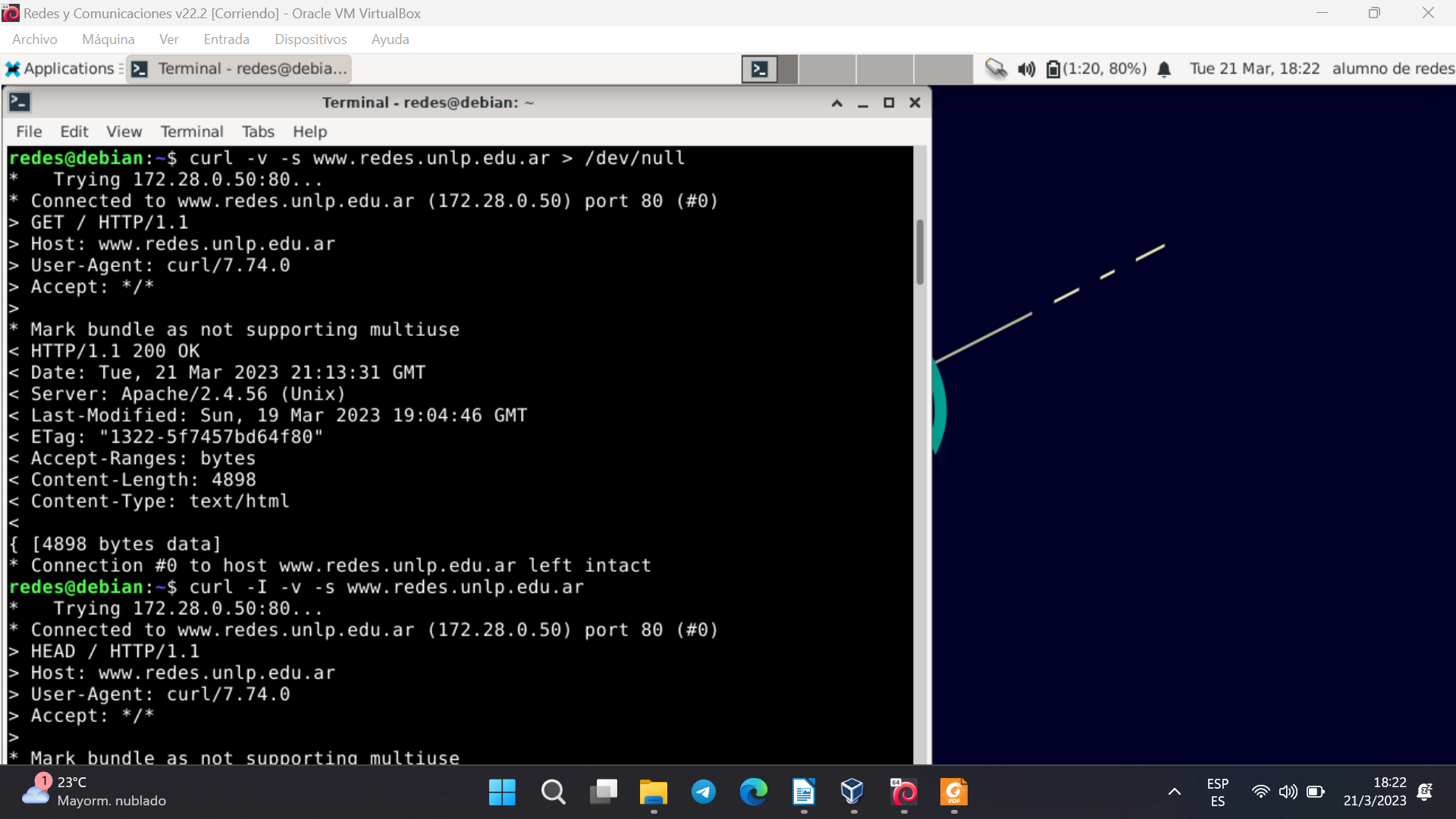
Por ende, se necesitarán 9 requests en total.

|  |  |
| --- | --- |
| Curl | Navegador |
| Hace un único request al recurso especificado (HTML) | Hace varios requests cuando detecta que la respuesta al primer request (HTML) necesita otros recursos para mostrarse correctamente, como los archivos css, js, imágenes, etc. |

**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](http://www.redes.unlp.edu.ar)**

****

**2023: 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?**

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

**¿Qué diferencias nota entre cada uno?**

El primer comando no muestra el resultado en la terminal, sino que lo redirige a un archivo llamado null.

El segundo comando intenta mostrar en la terminal lo mismo que muestra el primero.

El primero, al redirigir el documento html sólo muestra los headers.

En el segundo, si bien se quita la redirección, con el parámetro -I se piden sólo los headers.

**¿Qué ocurre si en el primer comando se quita la redirección a /dev/null?**

Si se quita la redirección se muestra todo el HTML en la consola.

**¿Cuántas cabeceras viajaron en el requerimiento? ¿Y en la respuesta?**

En la solicitud, 3, En la respuesta, 7

**¿Por qué no es necesaria en el segundo comando?**

**10. Investigue cómo define las cabeceras la RFC.**

**a. ¿Establece todas las cabeceras posibles?**

No, establece cabeceras generales, de request y de response, pero aclara que se podrían integrar nuevas si las partes involucradas en la comunicación las reconocen como tales y se respeta la semántica.

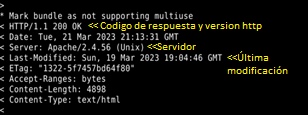
**b. ¿Cuántas cabeceras viajaron en el requerimiento y en la respuesta del ejercicio anterior?**

En la solicitud, 3, En la respuesta, 7

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

La cabecera Date contiene la fecha y hora de envío de la respuesta, en huso horario GMT

**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?**

La versión HTTP (1.1), el código de retorno (200), el motivo de ese código (OK)

**b. ¿Cuántos encabezados muestra la respuesta?**

Se muestran 7 encabezados

**c. ¿Qué servidor web está sirviendo la página?**

Servidor Apache, versión 2.4.53, para sistemas operativos Unix

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

Fue exitoso

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

Se modificó el 19/3/2023 a las 19:04:46 en hora 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?**

i) Usé el comando curl -I www.redes.unlp.edu.ar -H "If-Modified-Since: Sun, 19 March 2023 19:04:46 GMT"

ii) El resultado fue un error 304 Not Modified

iii) Este header sirve cuando se implementa caché para los recursos de un servidor. Si la copia local es igual que la del servidor, entonces ya está actualizada, ahorrando una transferencia

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

* En HTTP/1.0 no hay conexiones persistentes. Entonces, cuando un extremo cierra la conexión, el otro asume que ya recibió todo el dato.
* En HTTP/1.1, hay conexiones persistentes. Entonces, el header Content-Length es obligatorio, para que el otro extremo sepa cuántos bytes leer

**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 en esta información, cliente o servidor? ¿Es útil que esté detallado y clasificado en la RFC?**

**2xx: Peticiones correctas**: Esta clase de código de estado indica que la petición fue recibida correctamente,

entendida y aceptada.

**3xx: Redirecciones:** El cliente tiene que tomar una acción adicional para completar la petición. Esta clase de código de estado indica que una acción subsecuente necesita efectuarse por el agente de usuario para completar la petición. La acción requerida puede ser llevada a cabo por el agente de usuario sin interacción con el usuario sólo si el método utilizado en la segunda petición es GET o HEAD.

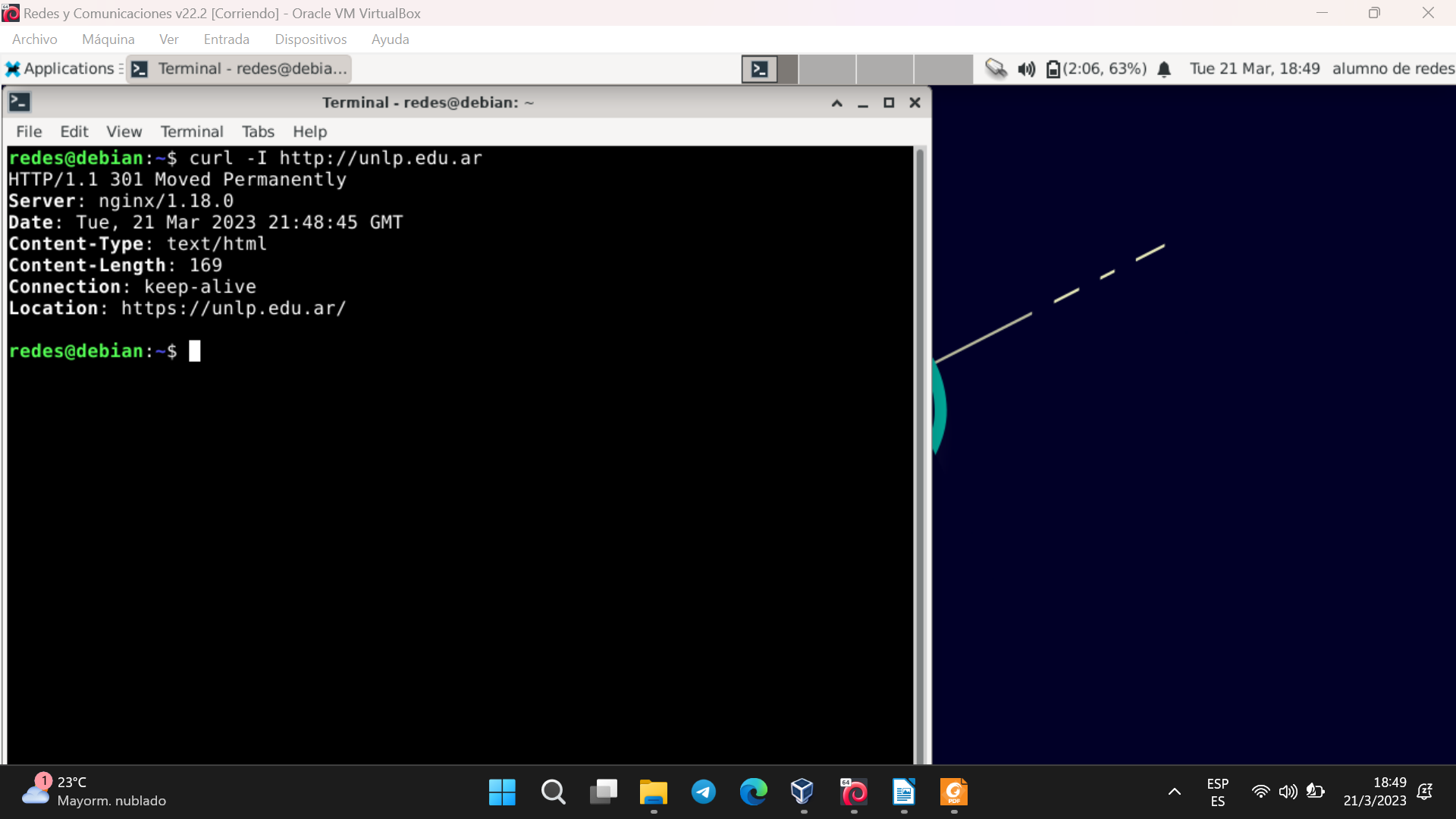
**4xx Errores del cliente**: La solicitud contiene sintaxis incorrecta o no puede procesarse. La intención de la clase de códigos de respuesta 4xx es para casos en los cuales el cliente parece haber errado la petición.

**5xx Errores de servidor:** El servidor falló al completar una solicitud aparentemente válida. Los códigos de respuesta que comienzan con el dígito "5" indican casos en los cuales el servidor tiene registrado aún antes de servir la solicitud, que está errado o es incapaz de ejecutar la petición.

Esta información es importante para el cliente.

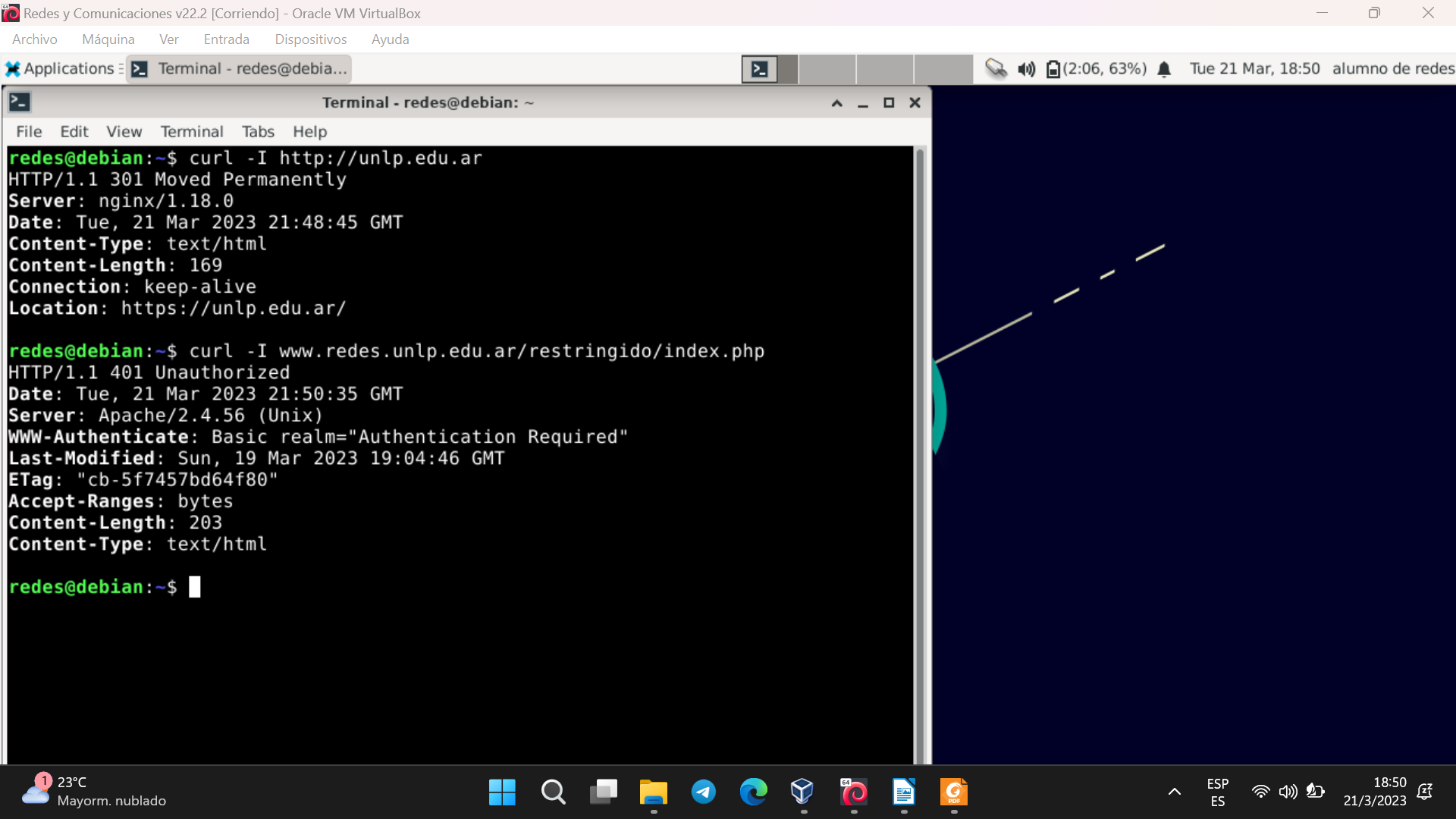
**Dentro de la VM, ejecute los siguientes comandos y evalúe el estado que recibe.**

**curl -I [http://unlp.edu.ar](http://unlp.edu.ar/)**

****

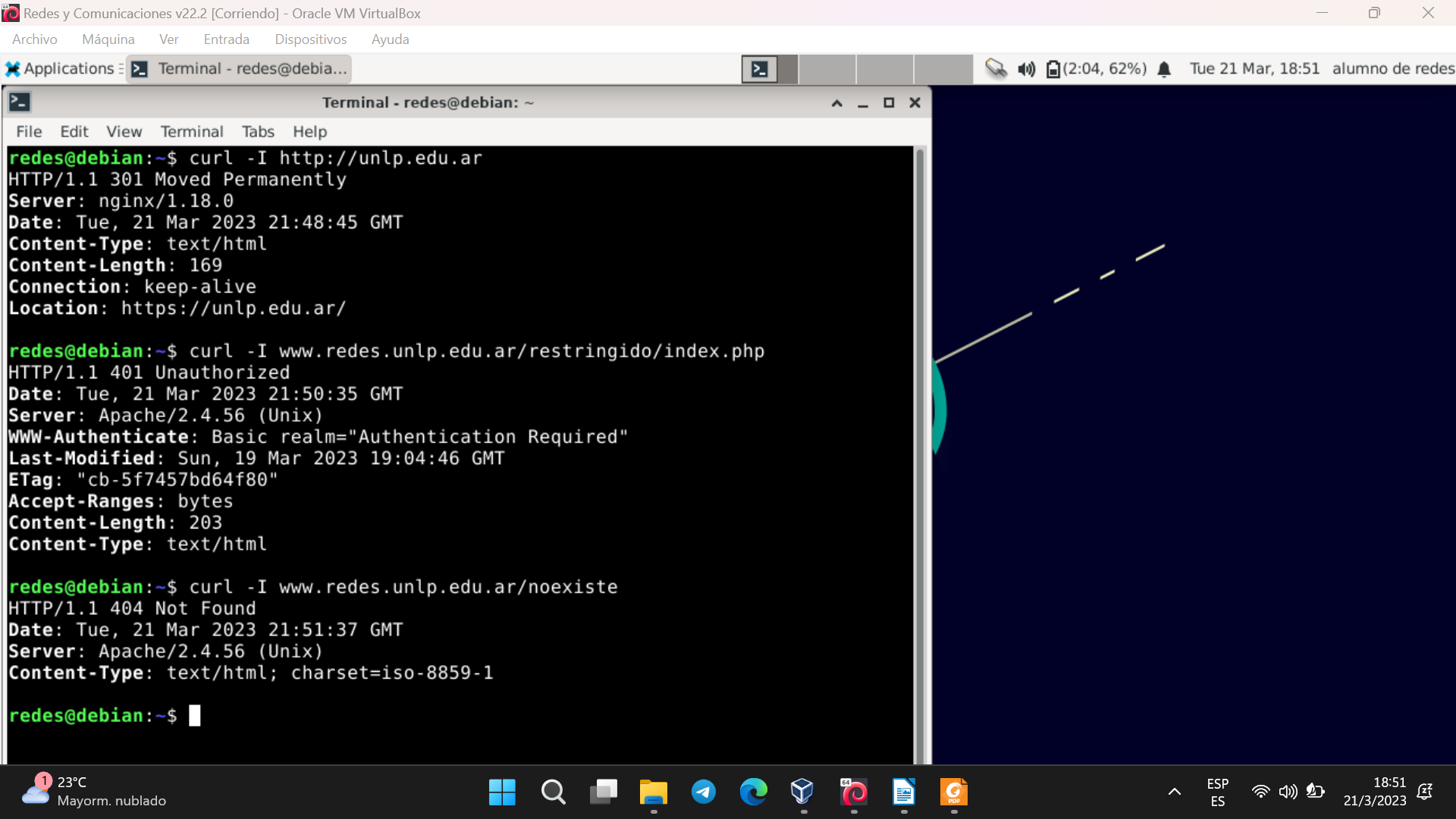
**301: Move Permanently →** La URL del recurso solicitado fue cambiada permanentemente. El cliente debe usar el método GET con otra URI

**curl -I** [**www.redes.unlp.edu.ar/restringido/index.php**](http://www.redes.unlp.edu.ar/restringido/index.php)

****

**401: Unauthorized →** El cliente necesita autenticarse para obtener acceso a la respuesta

**curl -I www.redes.unlp.edu.ar/noexiste**

****

**404: Not Found →** El recurso solicitado no existe

**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.**

*curl -v "****www.redes.unlp.edu.ar/restringido/index.php****" → -v significa que lo haga detallado, que escriba todo*

\* Trying 172.28.0.50:80...

\* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)

> GET /restringido/index.php HTTP/1.1

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

> User-Agent: curl/7.74.0

> Accept: \*/\*

>

\* Mark bundle as not supporting multiuse

< HTTP/1.1 401 Unauthorized

< Date: Tue, 30 Aug 2022 02:25:42 GMT

< Server: Apache/2.4.53 (Unix)

< WWW-Authenticate: Basic realm="Authentication Required"

< Last-Modified: Wed, 13 Apr 2022 22:55:32 GMT

< ETag: "cb-5dc9113140100"

< Accept-Ranges: bytes

< Content-Length: 203

< Content-Type: text/html

<

**<h1>Acceso restringido</h1>**

**<p>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**</p>

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

=================================================================================

*curl -v "****www.redes.unlp.edu.ar/obtener-usuario.php****"*

\* Trying 172.28.0.50:80...

\* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)

> GET /obtener-usuario.php 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: Tue, 30 Aug 2022 02:29:19 GMT

< Server: Apache/2.4.53 (Unix)

< X-Powered-By: PHP/7.4.28

< Content-Length: 147

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

<

**<p>Para obtener el usuario y la contraseña haga un requerimiento a esta página seteando el encabezado** **'Usuario-Redes'** con el valor **'obtener**'</p>

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

=================================================================================

*curl -v "www.redes.unlp.edu.ar/obtener-usuario.php" -H* ***"Usuario-Redes: obtener"***

\* Trying 172.28.0.50:80...

\* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)

> GET /obtener-usuario.php HTTP/1.1

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

> User-Agent: curl/7.74.0

> Accept: \*/\*

> Usuario-Redes: obtener

>

\* Mark bundle as not supporting multiuse

< HTTP/1.1 200 OK

< Date: Tue, 30 Aug 2022 02:30:32 GMT

< Server: Apache/2.4.53 (Unix)

< X-Powered-By: PHP/7.4.28

< Content-Length: 286

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

<

<p>Bien hecho! Los datos para ingresar son:

Usuario: **redes**

Contraseña: **RYC**

Ahora vuelva a acceder 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!</p>

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

=================================================================================

*echo -n "****redes:RYC****" | base64 → Para transformar caracteres de cualquier idioma a alfabeto latino*

**cmVkZXM6UllD**

=================================================================================

*curl -v "****www.redes.unlp.edu.ar/restringido/index.php****" -H "****Authorization: Basic******cmVkZXM6UllD****"*

\* Trying 172.28.0.50:80...

\* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)

> GET /restringido/index.php HTTP/1.1

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

> User-Agent: curl/7.74.0

> Accept: \*/\*

> Authorization: Basic cmVkZXM6UllD

>

\* Mark bundle as not supporting multiuse

< HTTP/1.1 302 Found

< Date: Tue, 30 Aug 2022 03:34:02 GMT

< Server: Apache/2.4.53 (Unix)

< X-Powered-By: PHP/7.4.28

< Location: **http://www.redes.unlp.edu.ar/restringido/the-end.php**

**< Content-Length: 230**

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

**<**

**<h1>Excelente!</h1>**

**<p>Para terminar el ejercicio deberás agregar en la entrega los datos que se muestran en la siguiente página.</p>**

**<p>ACLARACIÓN: la URL de la siguiente página está contenida en esta misma respuesta.</p>**

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

=================================================================================

*curl -v "****http://www.redes.unlp.edu.ar/restringido/the-end.php****" -H "****Authorization: Basic******cmVkZXM6UllD****"*

\* Trying 172.28.0.50:80...

\* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)

> GET /restringido/the-end.php HTTP/1.1

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

> User-Agent: curl/7.74.0

> Accept: \*/\*

> Authorization: Basic cmVkZXM6UllD

>

\* Mark bundle as not supporting multiuse

< HTTP/1.1 200 OK

< Date: Tue, 30 Aug 2022 03:38:58 GMT

< Server: Apache/2.4.53 (Unix)

< X-Powered-By: PHP/7.4.28

< Content-Length: 159

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

<

**¡Felicitaciones, llegaste al final del ejercicio!**

**Fecha: 2022-08-30 03:38:58**

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

**Verificación: 6f787663031b14d088986d1e10d19e1d30089b82a6fac9b2b4de1ceeeee5659e**

**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 linea del final).**

*curl -v "www.redes.unlp.edu.ar/extras/prueba-http-1-0.txt"*

\* Trying 172.28.0.50:80...

\* Connected to www.redes.unlp.edu.ar (172.28.0.50) port 80 (#0)

> GET /extras/prueba-http-1-0.txt 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: Tue, 30 Aug 2022 03:43:55 GMT

< Server: Apache/2.4.53 (Unix)

< Last-Modified: Wed, 13 Apr 2022 22:55:32 GMT

< ETag: "60-5dc9113140100"

< Accept-Ranges: bytes

< Content-Length: 96

< Content-Type: text/plain

<

GET /http/HTTP-1.1/ HTTP/1.0

User-Agent: curl/7.38.0

Host: www.redes.unlp.edu.ar

Accept: \*/\*

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

**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 realiza una solicitud GET al recurso /http/HTTP-1.1/ a través del software telnet

Luego de hacerlo, el servidor responde con la página, pero con menos contenido que cuando utilicé “curl [www.redes.unlp.edu.ar](http://www.redes.unlp.edu.ar)”. La razón de esto, es el recurso que estoy pidiendo. Con curl [www.redes.unlp.edu.ar](http://www.redes.unlp.edu.ar), estoy pidiendo el index. Mientras que con el otro comando, estoy pidiendo por [www.redes.unlp.edu.ar/http/HTTP-1.1](http://www.redes.unlp.edu.ar/http/HTTP-1.1).

**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.**

Gracias a que HTTP/1.1 usa conexiones persistentes por defecto, esto fue posible.

Es más eficiente el tráfico de datos porque no requiere establecer una conexión cada vez que se pida un recurso. Útil cuando en el documento descargado hay referencias a objetos externos. Si la conexión se cerrara (como en HTTP 1.0), habría que realizar múltiples conexiones al servidor para descargar todo el contenido, por ende, es más lento.

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

**¿Qué está haciendo al ejecutar el comando telnet?**

Telnet es usado para comunicarse con otro host, usando el protocolo Telnet. Si telnet es invocado con un host de argumento, se realiza un comando open de manera implícita. Al usar Telnet, se abre un canal de comunicación desde la entrada/salida estándar, hacia/desde el servidor especificado.

Permite acceder a otra máquina para manejarla remotamente como si estuviéramos sentados delante de ella. También es el nombre del programa informático que implementa el cliente. Para que la conexión funcione, como en todos los servicios de Internet, la máquina a la que se acceda debe tener un programa especial que reciba y gestione las conexiones. El puerto que se utiliza generalmente es el 23.

**¿Qué lo diferencia con curl?**

Telnet establece una comunicación a través del protocolo TELNET, curl funciona a través de http

**Observe la definición de método y recurso en la RFC. Luego responda, ¿Qué método HTTP utilizó? ¿Qué recurso solicitó?**

Se utilizó el comando GET, y se solicitó el recurso /http/HTTP-1.1/.

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

En el primer caso, luego de la solicitud, la conexión se cierra inmediatamente. En el segundo, permanece abierta un tiempo determinado (pasado el mismo, se cierra automáticamente).

La primera solicitud usa HTTP/1.0, mientras que la segunda, HTTP/1.1. Esta última usa conexiones persistentes por defecto, por lo que el servidor mantuvo abierta la conexión. La primera no, y por eso el servidor la cerró inmediatamente.

**¿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 primero es más eficiente cuando se sabe que sólo hay que realizar 1 solicitud. El segundo cuando hay que hacer múltiples, ya que se ahorran cierres y reaperturas de conexiones. En el segundo caso, la conexión debería mantenerse activa por poco tiempo, para no desperdiciar recursos. (En otro lado dice que siempre es más eficiente el segundo).

El segundo puede traer problemas de seguridad por dejar la conexión abierta y consumir recursos, quizás de manera innecesaria.

**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”. Página 3 de 5Capa de Aplicación - HTTP Redes y comunicaciones - 2022**

* **Para borrar la cache del navegador, deberá ir al menu “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?**

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

**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?**

Cookie y set cookie se introdujeron en el RFC 2109, el cual fue dejado obsoleto por el RFC 2965, que a su vez fue dejado obsoleto por el 6265.

**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 | Cookie |
| **Principal uso** | Enviar cookies desde el **servidor al cliente**, para que éste los almacene | Enviar cookies guardados en el **cliente al servidor**, para que éste los valide |
| **Relación con HTTP** | Se usa en una respuesta | Se usa en una solicitud |

**¿Qué atributo de la RFC original fue en parte aprovechado para la implementación?**

El atributo del RFC original de HTTP aprovechado es la sintaxis de los pares atributo valor.

**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?**

Un protocolo binario usa los datos en su representación interna.

Un protocolo basado en texto convierte los datos a binario (luego de recibirlos), y usa el resultado.

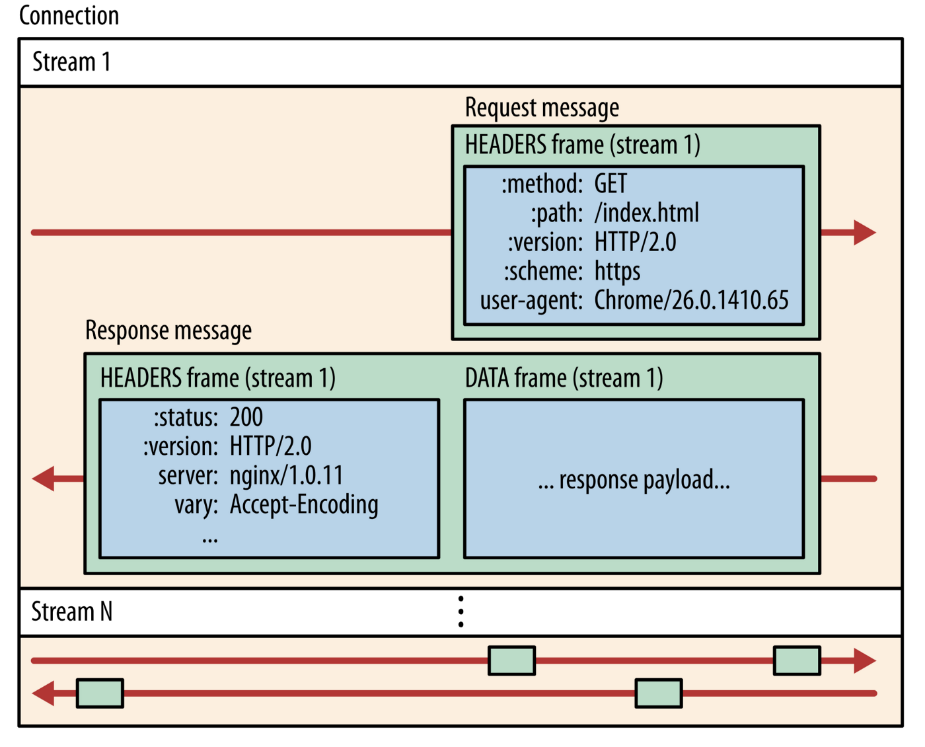
Debido a la construcción del binario, la información escrita en la que se puede intercambiar de forma rápida y no ocupa mucho espacio.

Un protocolo de texto sin formato no está codificado o formateado y está escrito en un lenguaje fácilmente comprensible.

Debido a su velocidad y brevedad, binario siempre ha tenido la sartén por el mango, pero como la banda ancha se convierte en un estándar, texto plano es cada vez más común.

|  |  |
| --- | --- |
| Protocolo | Tipo |
| HTTP/1.0 | Basado en texto |
| HTTP/1.1 | Basado en texto |
| HTTP/2 | Binario |

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



**Stream:** Intercambio bidireccional de datos de una conexión HTTP establecida entre el cliente y el servidor, que puede llevar uno o más ‘messages’.

**Message:** Conjunto de ‘frames’ que uniendo la secuencia completa forma una solicitud o repuesta HTTP lógica.

**Frame:** Patrón de comunicación más pequeño en HTTP/2, cada frame tiene un encabezado para identificar a la secuencia a la que pertenece para después formar un ‘message’ lógico.

El servicio “**server push**”, se basa en estimaciones para que el servidor sea capaz de enviar información al usuario antes de que éste la solicite para que la información esté disponible de forma inmediata.

La forma de actuar del servidor es enviar varias respuestas a una única solicitud del cliente, es decir, además de la respuesta a la solicitud original, el servidor puede enviar recursos adicionales. Esto es así porque una página web está formada por decenas de archivos referenciados que gracias al servicio “server push” el servidor envía tras recibir una única solicitud ahorrando mensajes innecesarios.

HTTP 2.0 contiene un campo denominado ‘Ajustes’ con el que el cliente puede indicar si desea o no obtener los recursos que proporciona el servicio ‘server push’.

**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)**

* Especifica qué host virtual (Que varias páginas están alojadas en una misma máquina) recibirá la solicitud. Contiene su nombre, y opcionalmente el número de puerto TCP en el que el servidor está escuchando.
* En HTTP/1.0 existía pero no era obligatorio usarlo (en la RFC no estaba definido). En HTTP/1.1, debe incluirse en las solicitudes.
* En HTTP/2, Host es reemplazado por :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](http://www.info.unlp.edu.ar/)**

: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 usando HTTP/1.0 o HTTP/1.1, ambos soportan el header **Host**

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

Usa el método **HEAD**, que sólo retorna las cabeceras

**c. ¿Cuál es el recurso solicitado?**

Se solicita el recurso **metodos**

**d. ¿El método funcionó correctamente?**

Sí, ya que el código de respuesta es **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?**

La solicitud le pide los headers del recurso, sólo si fue modificado desde la fecha Sat, 20 Jan 2018 13:02:41 GMT. Como luego de esa fecha no fue modificado, el servidor retornaría 304 Not Modified y el navegador usaría la copia almacenada en su caché