

Universidad Veracruzana

Redes y Servicios de Cómputo

NOMBRES:

Rivera Rojas Benjamín
Vázquez Torres Dayhan

EXPERIENCIA EDUCATIVA

Análisis de Protocolos

PROFESOR

Juan Luis López Herrera

Actividad: Proyecto Final

Fecha: Diciembre 2022

Servidor HTTP

Al nivel más básico, cuando un navegador necesita un archivo que está almacenado en un servidor web, el navegador limita el archivo al servidor mediante el protocolo HTTP. Cuando la petición alcanza al servidor web correcto (hardware), el servidor HTTP (software) acepta la solicitud, encuentra el documento requerido y lo envía de regreso al navegador, también a través de HTTP. (Si el servidor no encuentra el documento requerido, devuelve una respuesta 404 en su lugar.)

HTTP se basa en sencillas operaciones de solicitud/respuesta. Un cliente establece una conexión con un servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su posible resultado. Todas las operaciones pueden adjuntar un objeto o recurso sobre el que actúan; cada objeto Web es conocido por su URL.

¿Qué son los sockets?

Podemos encontrar diferentes definiciones, algunas de ellas son:

Los sockets son un método de comunicación entre un programa de cliente y uno de servidor a través de una red. Un socket se define como “el extremo de una conexión”.

Un socket, es un método para la comunicación entre un programa del cliente y un programa del servidor en una red. Un socket se define como el punto final en una conexión. Los sockets se crean y se utilizan con un sistema de peticiones o de llamadas de función a veces llamados interfaz de programación de aplicación de sockets (API, Application Programming Interface).

Un socket es también una dirección de Internet, combinando una dirección IP (la dirección numérica única de cuatro octetos que identifica a un computador particular en Internet) y un número de puerto (el número que identifica una aplicación de Internet particular) Un socket es un punto final de un enlace de comunicación de dos vías entre dos programas que se ejecutan a través de la red.

La arquitectura cliente-servidor

Es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un programa cliente realiza peticiones a otro programa, el servidor, que le da respuesta.

El Servidor

Los pasos que debe seguir un programa cliente son los siguientes:

Realizar la apertura de un socket, mediante la función `socket()`. Esta función devuelve un descriptor de archivo normal, como puede devolverlo `open()`. La función `socket()` no hace absolutamente nada, salvo devolvernos y preparar un descriptor de fichero que el sistema posteriormente asociará a una conexión en red.

Avisar al sistema operativo de que hemos abierto un socket y queremos que asocie nuestro programa a dicho socket. Se consigue mediante la función `bind()`. El sistema todavía no atenderá a las conexiones de clientes, simplemente anota que cuando empiece a hacerlo, tendrá que avisarnos. Es en esta llamada cuando se debe indicar el número de servicio al que se quiere atender. Avisar al sistema de que comience a atender dicha conexión de red. Se consigue mediante la función `listen()`. A partir de este momento el sistema operativo anotará la conexión de cualquier cliente para pasárnosla cuando se lo pidamos. Si llegan clientes más rápido de lo que somos capaces de atenderlos, el sistema operativo hace una “cola” con ellos y nos los irá pasando según vayamos pidiéndolo.

Pedir y aceptar las conexiones de clientes al sistema operativo. Para ello hacemos una llamada a la función `accept()`. Esta función le indica al sistema operativo que nos dé al siguiente cliente de la cola. Si no hay clientes se quedará bloqueada hasta que algún cliente se conecte.

Escribir y recibir datos del cliente, por medio de las funciones `write()` y `read()`, que son exactamente las mismas que usamos para escribir o leer de un archivo. Obviamente, tanto cliente como servidor deben saber qué datos esperan recibir, qué datos deben enviar y en qué formato. Cierre de la comunicación y del socket, por medio de la función `close()`, que es la misma que sirve para cerrar un archivo.

El Cliente

Los pasos que debe seguir un programa cliente son los siguientes:

Realizar la apertura de un socket, como el servidor, por medio de la función `socket()`. Solicitar conexión con el servidor por medio de la función `connect()`. Dicha función quedará bloqueada hasta que el servidor acepte nuestra conexión o bien si no hay servidor en el sitio indicado, saldrá dando un error. En esta llamada se debe facilitar la dirección IP del servidor y el número de servicio que se desea. Escribir y recibir datos del servidor por medio de las funciones `write()` y `read()`. Cerrar la comunicación por medio de `close()`. Como se puede apreciar, el procedimiento en el cliente es mucho más sencillo que el servidor, más, sin embargo, se debe como mínimo garantizar en los dos extremos, un paso de establecimiento de la comunicación, uno de transferencia de información y uno mediante el cual se libera la comunicación.

Se nos presenta el siguiente problema con los sockets.

El método del objeto socket sólo nos permite recibir hasta una cantidad FIJA de bytes. Si se envían datos menores o iguales a la cantidad establecida en nuestro método "recv" recibiremos el mensaje sin problemas. Pero ¿y si el mensaje es mayor a la cantidad de bytes establecidos?

Debemos tener claro lo siguiente:

Cliente: Se encargará de enviar el archivo

Servidor: Se encargará de recibir y guardar el archivo en el disco duro

La solución para la cantidad de bytes es:

Programar un protocolo para nuestro cliente servidor. El protocolo tendrá que hacer algo como esto:

- Cliente: Enviar longitud archivo.
- Servidor: Comprobar la longitud (debe ser un número).
- Servidor: En caso de que la longitud sea correcta, avisar al cliente para que envíe al archivo.
- Cliente: Comprobar que el servidor está listo.
- Cliente: Enviar archivo al servidor.

Cabeceras y códigos de respuestas HTTP

Las cabeceras y los códigos de estado HTTP resultan útiles como ayuda para programas intermediarios y clientes a la hora de comprender la información sobre solicitudes y respuestas de aplicaciones. Las cabeceras HTTP contienen información de metadatos. Los códigos de estado HTTP proporcionan información de estado sobre la respuesta.

Puede leer cabeceras HTTP a partir de la solicitud y establecer las cabeceras en la respuesta. Hay un conjunto de cabeceras de solicitud y respuesta comunes, pero también existen cabeceras de solicitud exclusivas y las cabeceras de respuesta exclusivas.

Trabajar con los códigos de estado de respuesta HTTP es una forma de usar este protocolo para facilitarte la vida en tus programas de código, ya que te informa, cada vez que puede, de cómo va la ejecución del programa. Además, si hay algún problema, se encarga de reportarlo.

Clases de estado de respuesta HTTP

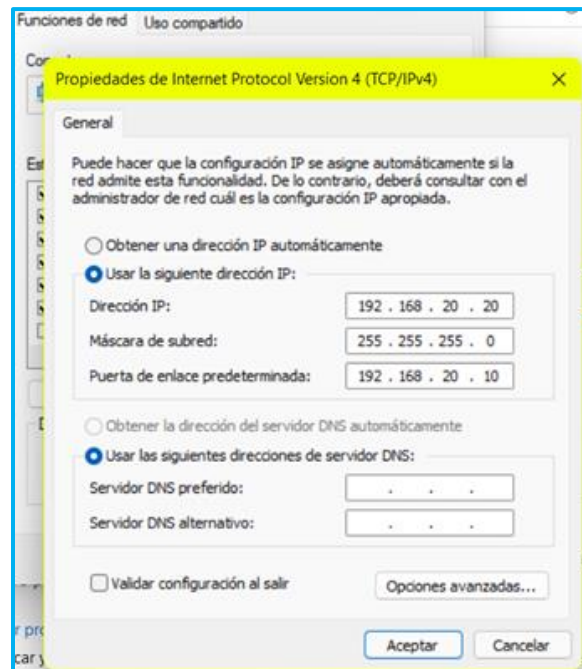
Estas pueden ser de 5 tipos según sea su necesidad:

- Respuestas informativas (100-199), que establecen cómo va el programa de código.
- Respuestas satisfactorias (200 – 299), que se encargan de mostrarte lo que está funcionando correctamente con el programa.
- Redirecciones (300 – 399), que permiten saber de qué manera se están relocalizando los enlaces de navegación de las peticiones.
- Errores de los clientes (400 – 499), que te permiten saber de qué forma ha errado el usuario en la ejecución del programa de código.
- Errores de los servidores (500 – 599), cuya causa son problemas originarios del servidor y no de alguna acción que tú o el usuario hayáis efectuado. Estos pueden solucionarse cuando el servidor esté funcionando correctamente otra vez.

De esta manera, puedes incluir estos códigos de estado de respuesta según sean las necesidades de tu código, de la misma forma que podrías levantar una excepción para determinar otros procesos.

Capturas Cliente:

Conectarse a una red local y asignarle la dirección IPv4 192.168.20.20 a la máquina que funcionará como Cliente para HTTP y para la transmisión de archivos a través de sockets.



Verificamos que se pueda hacer ping al Servidor: 192.168.20.10

```
Símbolo del sistema
Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . :

C:\Users\torre>ping 192.168.20.10

Haciendo ping a 192.168.20.10 con 32 bytes de datos:
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.
Tiempo de espera agotado para esta solicitud.

Estadísticas de ping para 192.168.20.10:
    Paquetes: enviados = 4, recibidos = 0, perdidos = 4
              (100% perdidos),

C:\Users\torre>ping 192.168.20.10

Haciendo ping a 192.168.20.10 con 32 bytes de datos:
Respuesta desde 192.168.20.10: bytes=32 tiempo=2ms TTL=128
Respuesta desde 192.168.20.10: bytes=32 tiempo=2ms TTL=128
Respuesta desde 192.168.20.10: bytes=32 tiempo<1m TTL=128
Respuesta desde 192.168.20.10: bytes=32 tiempo=1ms TTL=128

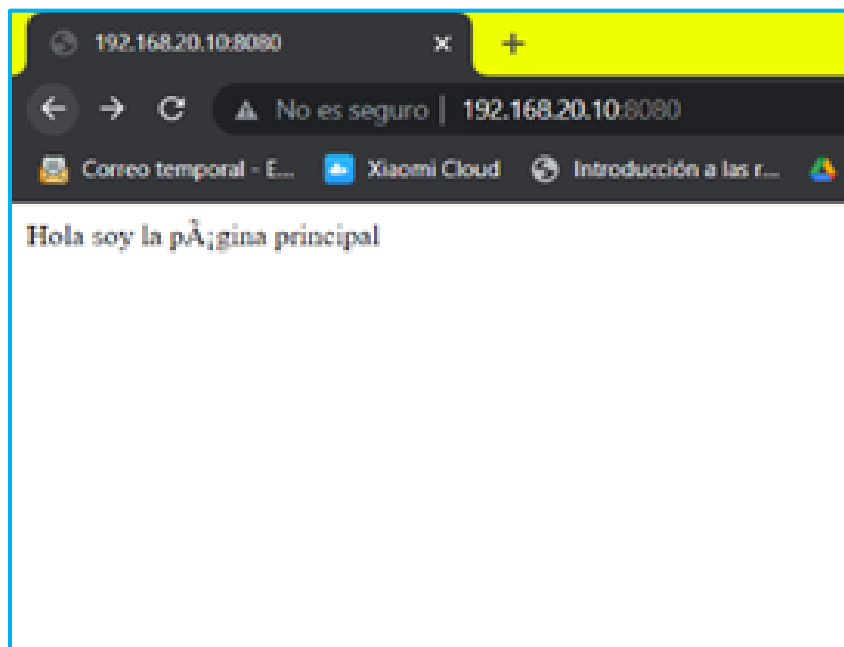
Estadísticas de ping para 192.168.20.10:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
              (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 0ms, Máximo = 2ms, Media = 1ms

C:\Users\torre>
```

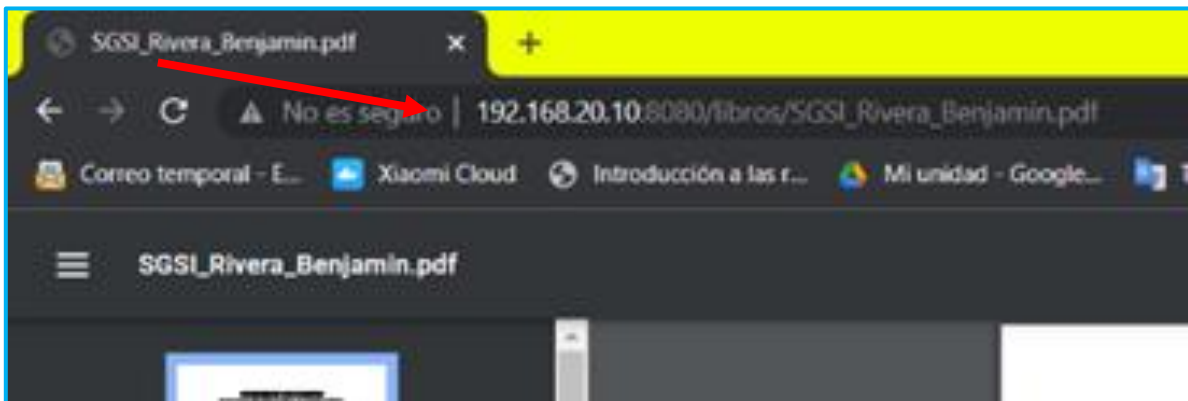
Servidor HTTP- Parte del cliente.

En el caso del cliente deberá ingresar a su navegador de preferencia e ingresar la IP del servidor seguida de dos puntos y el puerto 8080.

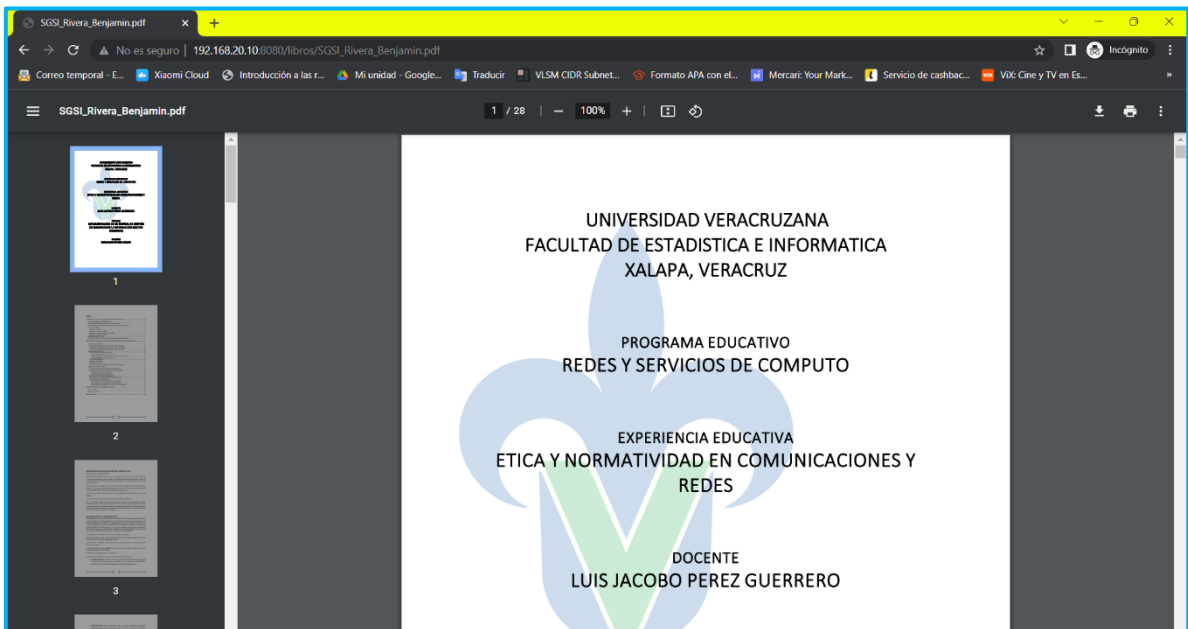
Nos muestra una leyenda de que es la pagina principal.



Si colocamos lo siguiente “/libros/SGSI_Rivera_Benjamin.pdf” seguido del puerto nos mostrara un documento.



Este es el documento:



Ahora si lo hacemos con la siguiente leyenda “/imagenes/iistart.jpg” .

Nos debera mostrar una imagen.

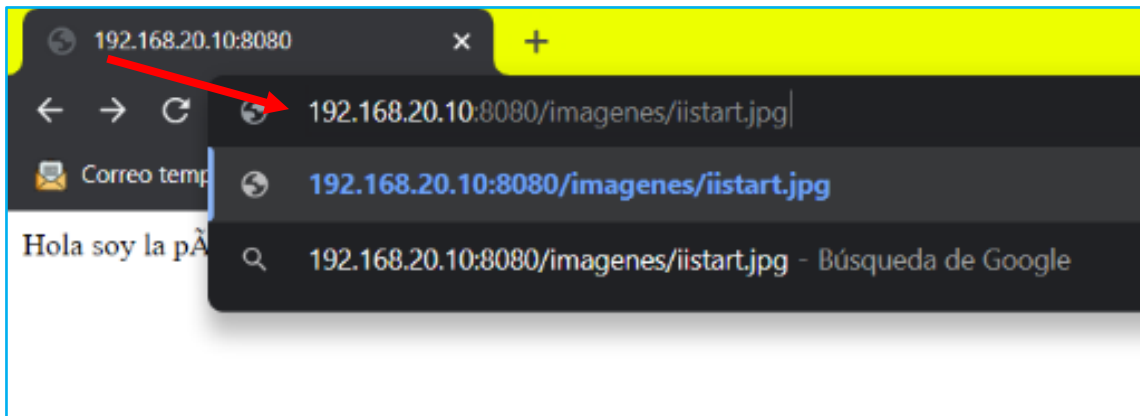
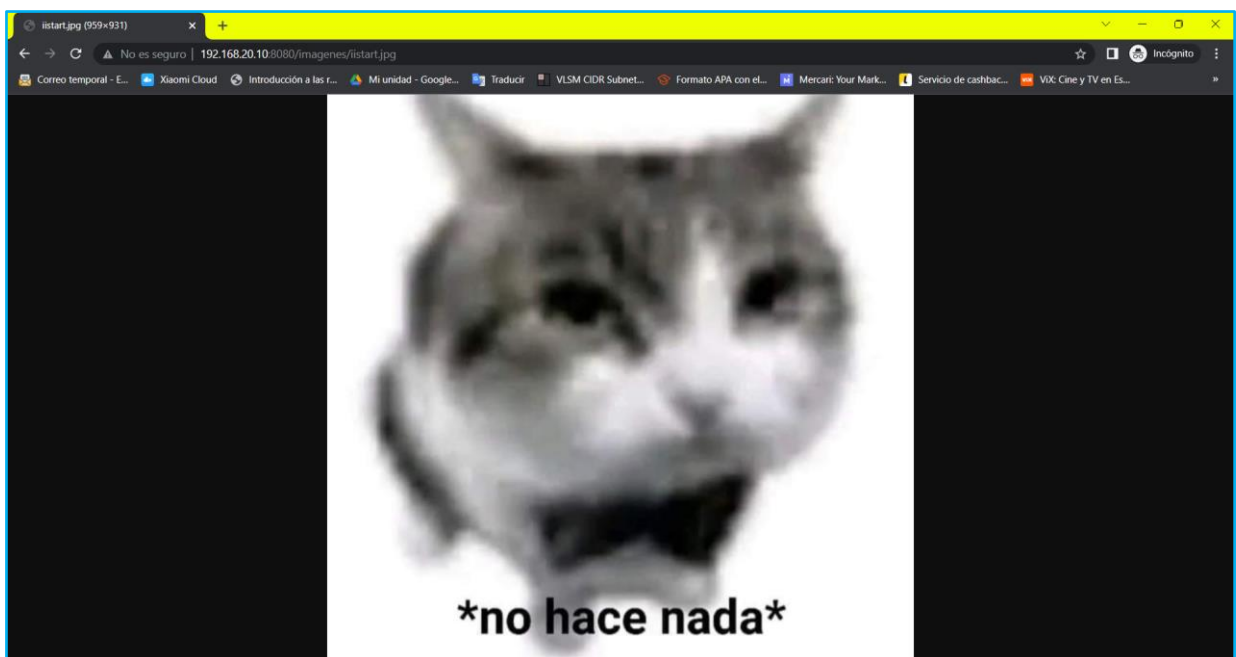
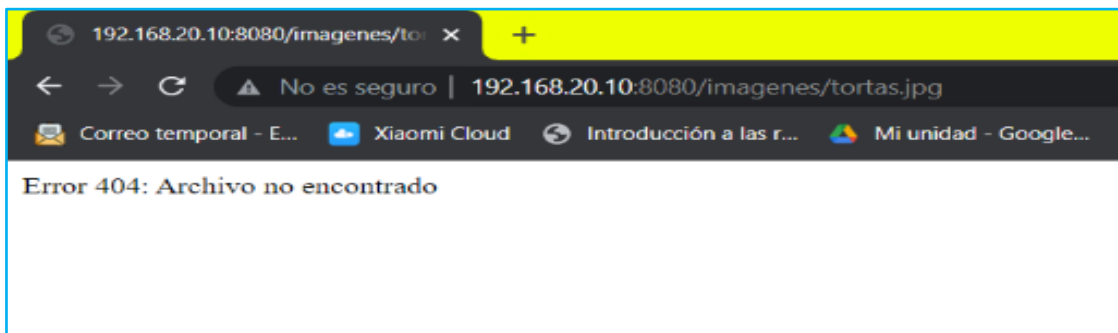


Imagen mostrada:



En caso contrario de colocar algun nombre o leyenda que no este alojado en el servidor, debera mostrar el siguiente error.



Fotografía de los dispositivos físicos que se utilizaron.

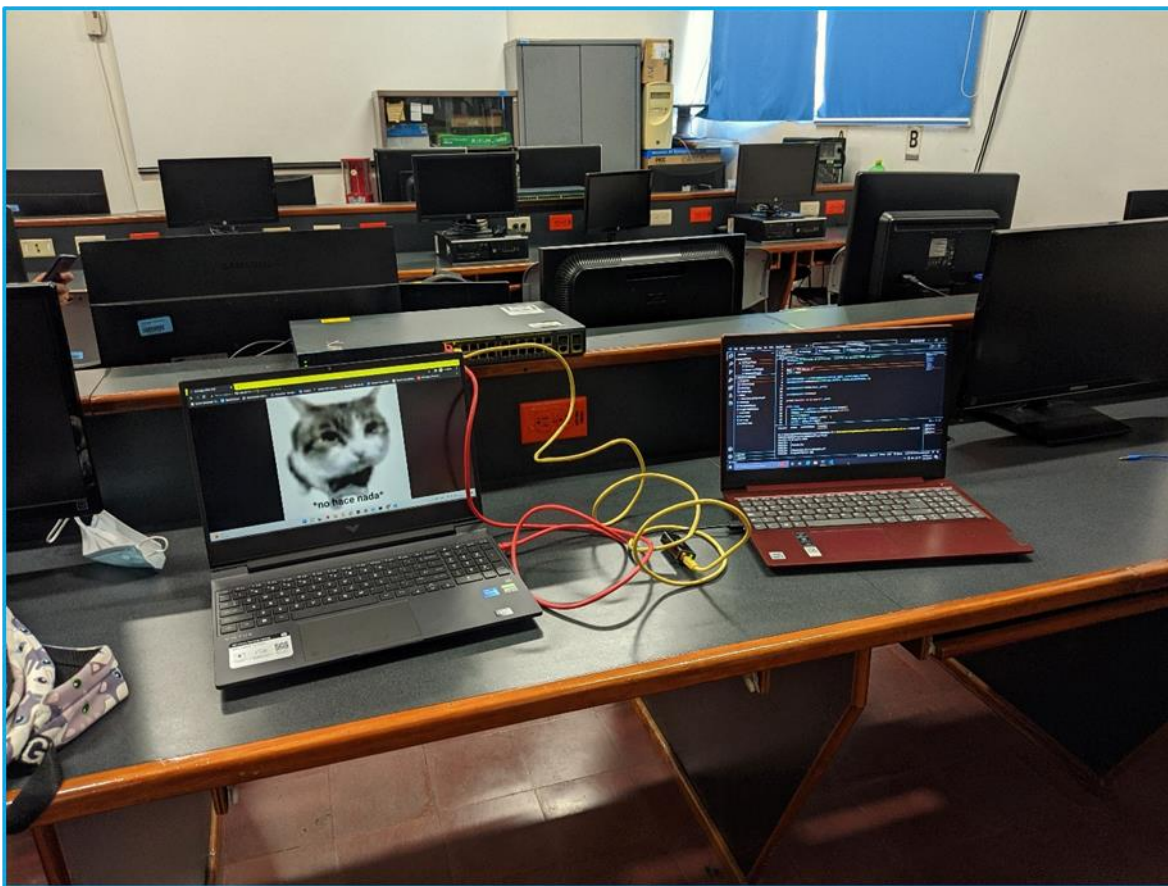
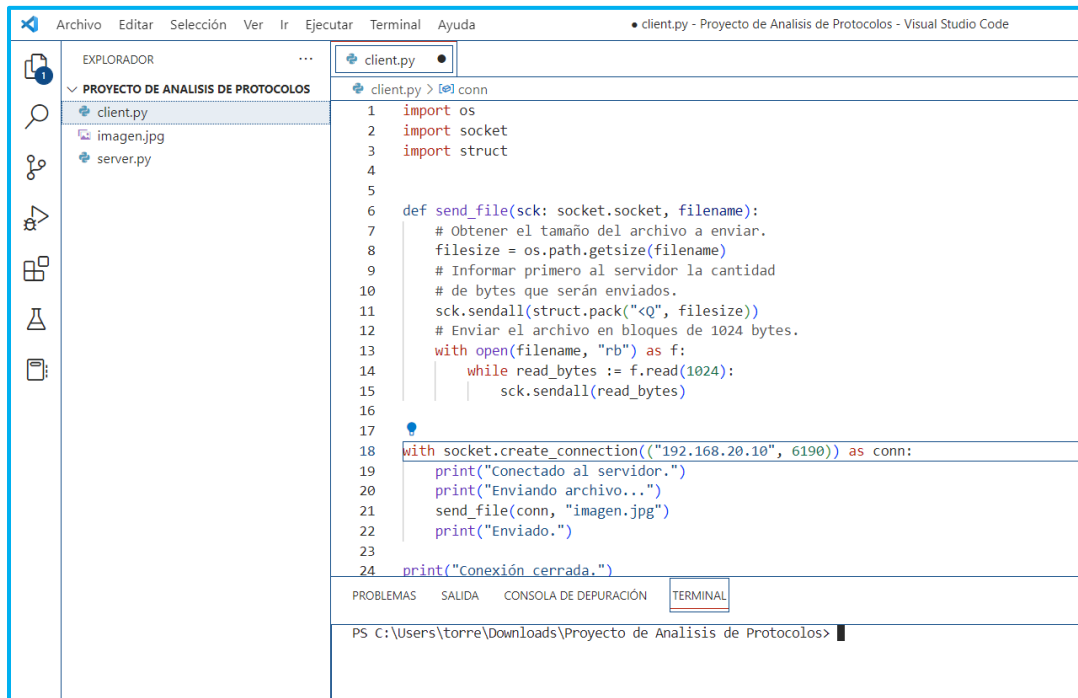


Ilustración 1 Práctica física HTTP

Servidor Envió de Archivos- Parte del cliente.



The screenshot shows the Visual Studio Code interface with the file explorer on the left displaying the project structure: `PROYECTO DE ANALISIS DE PROTOCOLOS`, `client.py`, `imagen.jpg`, and `server.py`. The main editor shows the `client.py` file with the following code:

```
1 import os
2 import socket
3 import struct
4
5
6 def send_file(sck: socket.socket, filename):
7     # Obtener el tamaño del archivo a enviar.
8     filesize = os.path.getsize(filename)
9     # Informar primero al servidor la cantidad
10    # de bytes que serán enviados.
11    sck.sendall(struct.pack("<Q", filesize))
12    # Enviar el archivo en bloques de 1024 bytes.
13    with open(filename, "rb") as f:
14        while read_bytes := f.read(1024):
15            sck.sendall(read_bytes)
16
17
18 with socket.create_connection(("192.168.20.10", 6190)) as conn:
19     print("Conectado al servidor.")
20     print("Enviando archivo...")
21     send_file(conn, "imagen.jpg")
22     print("Enviado.")
23
24 print("Conexión cerrada.")
```

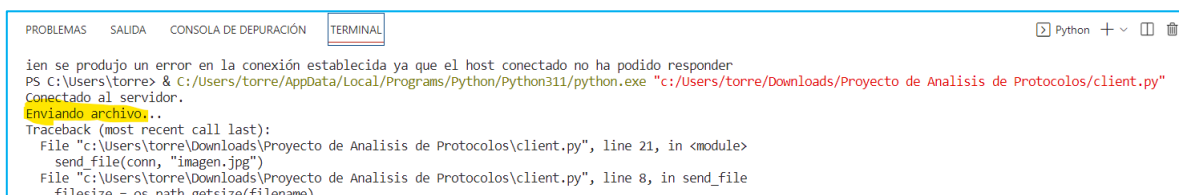
The terminal at the bottom shows the command prompt output:

```
PS C:\Users\torre\Downloads\Proyecto de Analisis de Protocolos>
```

En nuestro código tenemos el envío de una imagen de la Universidad Veracruzana que a continuación deberá recibir el Servidor.



Ejecutamos y si todo sale correcto le debe llegar la imagen al Servidor. (Pruebas de éxito en el apartado del Servidor).



The screenshot shows the terminal window with the following output:

```
Problemas SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python + - [ ] [ ]
ien se produjo un error en la conexión establecida ya que el host conectado no ha podido responder
PS C:\Users\torre> & C:/Users/torre/AppData/Local/Programs/Python/Python311/python.exe "C:/Users/torre/Downloads/Proyecto de Analisis de Protocolos/client.py"
Conectado al servidor.
Enviando archivo...
Traceback (most recent call last):
  File "C:\Users\torre\Downloads\Proyecto de Analisis de Protocolos\client.py", line 21, in <module>
    send_file(conn, "imagen.jpg")
  File "C:\Users\torre\Downloads\Proyecto de Analisis de Protocolos\client.py", line 8, in send_file
    filesize = os.path.getsize(filename)
```

Servidor HTTP en Python con sockets:

En este código se importa el socket para poder hacer uso de los sockets y hacer uso de las distintas funciones que traen.

Es importante recalcar que hay que colocar la ip del host que funcionará como servidor y el puerto por el cual estará escuchando, en este caso el puerto 8080. Todo el código siguiente hará uso de estas funciones de sockets en donde se creará un while que esperará la petición de un cliente. En donde almacenará la conexión y la dirección, luego lo que el servidor reciba se decodificará a utf-8 para evitar problemas de caracteres que no se puedan reconocer o no sean compatibles, luego dividirá lo recibida para poder obtener el nombre del archivo y la ruta en donde se encuentra ubicado.

```
1  #Proyecto de Analisis de protocolos - Creación de servidor HTTP con Sockets
2  import socket
3
4  host = '192.168.20.10'
5  port = 8080
6
7  servidorSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8  servidorSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
9
10 servidorSocket.bind((host, port))
11
12 servidorSocket.listen(1)
13
14 print('Servidor en el puerto', port)
15
16 while True:
17     connection, address = servidorSocket.accept()
18     request = connection.recv(1024).decode('utf-8')
19
20     lista_String = request.split(' ')
21     metodo = lista_String[0]
22     archivo_solicitado = lista_String[1]
23
24     print('Petición: ', archivo_solicitado)
25
26     miarchivo = archivo_solicitado.split('?')[0]
27     miarchivo = miarchivo.lstrip('/')
28
```

Aquí en caso de que lo solicitado sea "" nos devolverá el archivo index.html, el cual funcionaría como el archivo por defecto. Luego con un try prueba que si el archivo se puede abrir, en caso de que se pueda abrir se guardará en la cabecera como "HTTP/1.1 200 OK" y luego comparará si el archivo que se solicita termina con .jpg, en caso de que esto sea verdad se guardará el mimetype del archivo acorde al tipo de mime que tiene cada uno, en este caso utilizamos jpg y pdf, en caso de que no sea ninguno de esos será html por defecto, y esto se guardará en la cabecera como "Content-Type: images/jpg"

```
29     if(miarchivo == ''):
30         |     miarchivo = 'index.html'
31
32     try:
33         |     archivo = open(miarchivo , 'rb')
34         |     response = archivo.read()
35         |     archivo.close()
36
37         |     cabecera = 'HTTP/1.1 200 OK\n'
38
39         |     if(miarchivo.endswith('.jpg')):
40         |         |     mimetype = 'image/jpg'
41         |     elif(miarchivo.endswith('.pdf')):
42         |         |     mimetype = 'application/pdf'
43         |     else:
44         |         |     mimetype = 'text/html'
45         |     cabecera += 'Content-Type: ' +str(mimetype)+'\n\n'
46
47     except Exception as e:
48         |     cabecera = 'HTTP/1.1 404 Not Found\n\n'
49         |     response = '<html><body> Error 404: Archivo no encontrado </body></html>'.encode('utf-8')
50
51     final_response = cabecera.encode('utf-8')
52     final_response += response
53     connection.send(final_response)
54     connection.close()
```

Luego en caso de que esto falle (El archivo no exista) nos mostrará un mensaje de error y en la cabecera se guardará HTTP/1.1 404 Not Found. Y finalmente se cerrará la conexión.

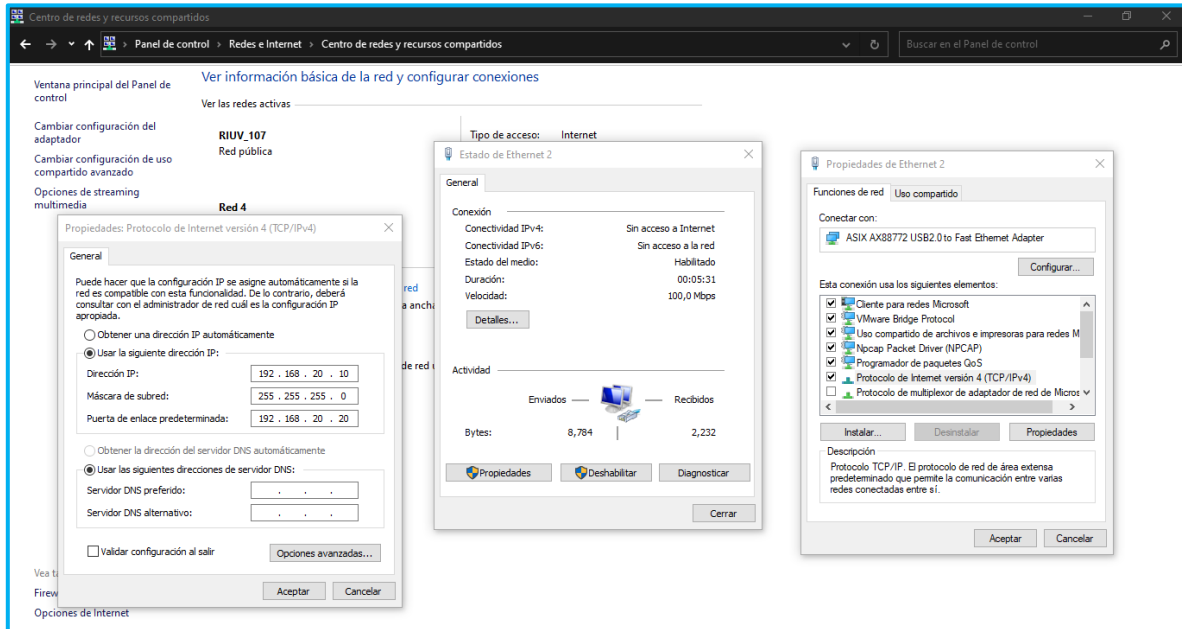
Servidor de recibida de archivos mediante sockets:

```
1
2 import socket
3 import struct
4
5 # Esta función se asegura de que se reciban los bytes que indican el tamaño del archivo que será enviado,
6 # que es codificado por el cliente vía struct.pack(), función la cual genera una secuencia de bytes que
7 # representan el tamaño del archivo.
8 def receive_file_size(sck: socket.socket):
9
10     fmt = "<Q"
11     expected_bytes = struct.calcsize(fmt)
12     received_bytes = 0
13     stream = bytes()
14
15     while received_bytes < expected_bytes:
16         chunk = sck.recv(expected_bytes - received_bytes)
17         stream += chunk
18         received_bytes += len(chunk)
19
20     filesize = struct.unpack(fmt, stream)[0]
21
22     return filesize
23
24 def receive_file(sck: socket.socket, filename):
25
26     # Leer primero del socket la cantidad de bytes que se recibirán del archivo.
27     filesize = receive_file_size(sck)
```

```
28
29     # Abrir un nuevo archivo en donde guardar los datos recibidos.
30     with open(filename, "wb") as f:
31         received_bytes = 0
32
33         # Recibir los datos del archivo en bloques de 1024 bytes hasta llegar a la cantidad de bytes total infor
34         while received_bytes < filesize:
35             chunk = sck.recv(1024)
36
37             if chunk:
38                 f.write(chunk)
39                 received_bytes += len(chunk)
40
41 with socket.create_server(("192.168.20.10", 6190)) as server:
42     print("Esperando al cliente...")
43     conn, address = server.accept()
44     print(f"{address[0]}:{address[1]} conectado.")
45     print("Recibiendo archivo...")
46     receive_file(conn, "imagen-recibida.jpg")
47     print("Archivo recibido.")
48     print("Conexión cerrada.")
```

Capturas Servidor:

Conectarse a una red local y asignarle la dirección IPv4 192.168.20.10 a la máquina que funcionará como Servidor para HTTP y para la transmisión de archivos a través de sockets.



Prueba de ping hacia la máquina cliente 192.168.20.20 confirmando que exista conexión entre ambas

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19044.2251]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\BERTHA>ping 192.168.20.20

Haciendo ping a 192.168.20.20 con 32 bytes de datos:
Respuesta desde 192.168.20.20: bytes=32 tiempo<1m TTL=128
Respuesta desde 192.168.20.20: bytes=32 tiempo=1ms TTL=128
Respuesta desde 192.168.20.20: bytes=32 tiempo=1ms TTL=128
Respuesta desde 192.168.20.20: bytes=32 tiempo=1ms TTL=128

Estadísticas de ping para 192.168.20.20:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
              (0% perdidos),
    Tiempos aproximados de ida y vuelta en milisegundos:
        Mínimo = 0ms, Máximo = 1ms, Media = 0ms

C:\Users\BERTHA>
```


Salida del servidor HTTP cliente, en donde muestra en consola en que puerto se encuentra y la petición que el cliente está solicitando, en este ejemplo se está solicitando el archivo SGSI_Rivera_Benjamin.pdf que se encuentra dentro de la carpeta libros:

```
PS C:\Users\BERTHA\Documents\Proyecto-Analisis> & C:/Users/BERTHA/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/BERTHA/
Documents/Proyecto-Analisis/Proyecto.py
Servidor en el puerto 8080
Petición: /
Petición: /favicon.ico
Petición: /
Petición: /libros/SGSI_Rivera_Benjamin.pdf
```

Ahora el cliente está solicitando la imagen llamada iistart.jpg que se encuentra en la carpeta imágenes:

```
PS C:\Users\BERTHA\Documents\Proyecto-Analisis> & C:/Users/BERTHA/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/BERTHA/
Documents/Proyecto-Analisis/Proyecto.py
Servidor en el puerto 8080
Petición: /
Petición: /favicon.ico
Petición: /
Petición: /libros/SGSI_Rivera_Benjamin.pdf
Petición: /imagenes/iistart.jpg
```

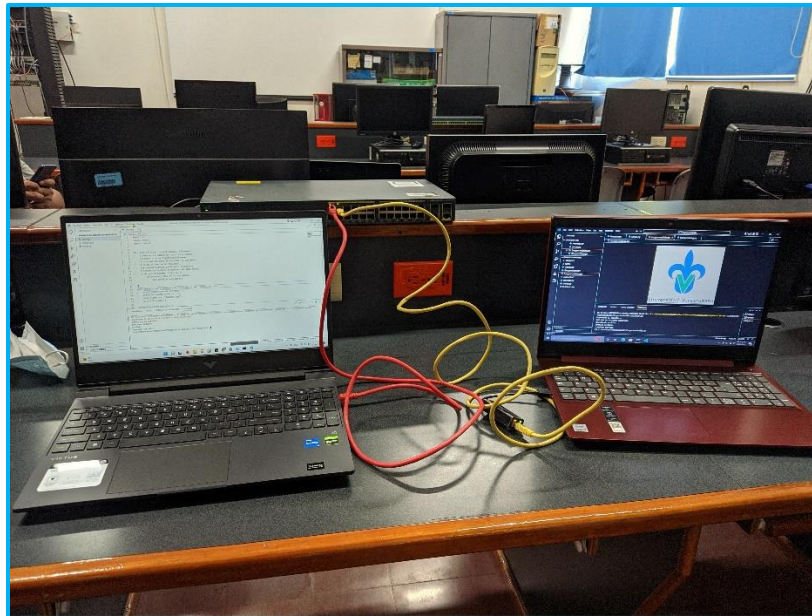


Ilustración 2 Práctica física sockets

Para el código de cliente-servidor a través de sockets el servidor estará a la escucha de cualquier petición, en este caso mostrará el mensaje “Esperando el cliente” hasta que algún cliente dentro de la red decida solicitar una conexión, en este caso, el envío de un archivo.

```
PS C:\Users\BERTHA\Documents\Proyecto-Analisis> & C:/Users/BERTHA/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/BERTHA/
Documents/Proyecto-Analisis/Server.py
Esperando al cliente...
```

Una vez que el cliente haya realizado la conexión con el servidor este mostrará la ip del cliente junto al puerto con el que se están comunicando y posteriormente lo que hará será el recibir el archivo y lo guardará como imagen-copia.jpg.

```
PS C:\Users\BERTHA\Documents\Proyecto-Analisis> & C:/Users/BERTHA/AppData/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/BERTHA/
Documents/Proyecto-Analisis/Server.py
Esperando al cliente...
192.168.20.20:50996 conectado.
Recibiendo archivo...
Archivo recibido.
Conexión cerrada.
PS C:\Users\BERTHA\Documents\Proyecto-Analisis>
```

Está es la imagen que le mando el cliente al servidor, la máquina cliente 192.168.20.20 logró mandarle una imagen a la máquina servidor 192.168.20.10.

