Servidor HTTP usnado Sockets y Select en Python.

El programa creado es un servidor HTTP básico que es capaz de manejar solicitudes de tipo GET y solicitudes de tipo POST, esto último a modo de servidor de eco, es decir, devuelve la data recibida.

Los protocolos admitidos son HTTP/0.9 HTTP/1.0 y parcialmente HTTP/1.1, parcialmente porque si bien reconoce la cabecera 1.1 no mantiene la conexión abierta.

El programa tiene 3 manejadores:

- handle_readable
- handle_writable
- handle_exceptional

Adicionalmente se crea una estructura de tipo diccionario que almacenará en una cola los mensajes recibidos por cada socket. La clave será el descriptor (socket) y el valor será una cola que contiene los mensajes de dicho socket tal como llegan desde el cliente.

La lectura de mensajes se hace por medio de *chunks* o trozos de tamaño 1024 bytes, de esta forma se garantiza recibir peticiones de diferentes tamaños. Hay que recordar que la transmisión por socket establece un límite superior para el tamaño de datos que pueden ser manejados, siendo 1GB el máximo para sockets de tipo TCP. (https://www.ibm.com/docs/en/ztpf/2020?topic=apis-send-send-data-connected-socket)

Manejadores

El manejador de *leíbles* (readable) se encarga de atender todas las peticiones entrantes de sockets, lo hace tomando en cuenta 3 condiciones. La primera simplemente verifica que se trata de una conexión entrante y procede a aceptar la misma, luego una entrada en el diccionario que inicializa la cola para dicha conexión. La segunda condición se divide en 2: lee los datos de la conexión, es decir, si la conexión ya ha sido aceptada, significa que se pueden leer datos. Si la lectura de datos falla, significa que el cliente se ha desconectado, se procede a eliminar la entrada en el diccionario de mensajes y se cierra el socket. Si existen datos en el buffer, se leen hasta que todos hayan sido recibidos, luego se unen todas las piezas recibidas y se almacenan en la cola de mensajes usando el descriptor adecuado.

El manejador de writables, se encarga de procesar la lista de sockets que están listos para recibir datos de parte del servidor, es decir, que se puede escribir en ellos. Se recorre la lista de descriptores, y se procesa cada mensaje sacado de la cola de mensajes. De esta forma, el mensaje que se procesa está completo para su análisis. Es en este momento que se realiza la

verificación del tipo de solicitud y su contenido. Una vez procesados todos los mensajes y enviada la respuesta se cierra el socket y se elimina la entrada del diccionario de mensajes.

El manejador de exeptionals se encarga de cerrar los sockets que se encuentran en un estado de error, si hubiera alguno en este estado, se cierra y su entrada en el diccionario de mensajes es eliminada.

Solicitudes

Actualmente el servidor recibe o maneja 2 tipos de solicitudes: GET y POST.

Para la solicitud de tipo GET se analiza la primera cabecera, de donde se extrae el protocolo y el recurso solicitado. Por simplicidad, se usan rutas relativas al directorio en el que se lanza el script, sin embargo, en un servidor web real, los assets o archivos estáticos que puede servirse usualmente se encuentran en una carpeta aislada y desde la cual no se permita el acceso a otras carpetas del sistema. Se ha utilizado la librería mime, para determinar el tipo de recurso que se enviará como respuesta. Esta librería permite conocer el tipo de archivo abierto para lectura y devuelve una tupla en la cual se encuentra el tipo de archivo para ser adjuntado en la cabecera Content-Type de la respuesta.

Si el recurso solicitado no existe, se devuelve un mensaje de error 404 usando las cabeceras adecuadas.

Para la solicitud de tipo POST, se realiza un servidor de eco, es decir, se devuelve al cliente exactamente lo mismo que fue recibido en el servidor, usando la información extraída de las cabeceras de solicitud. Sin embargo, el tamaño del contenido a enviar se calcula en el lado del servidor, manteniendo solamente el tipo de contenido igual al recibido (cabeceras Content-Type y Content-Length).

Si la solicitud recibida no es ni GET ni POST se envía un mensaje de error con estado HTTP 405 (Method Not Allowed), con esto informamos que el método solicitado no es aceptado o permito por el servidor (PUT, HEAD, OPTIONS, DELETE)

Además se analiza el la versión del protocolo utilizado, si la primera cabecera (primera línea del mensaje de solicitud) no es una cabecera HTTP válida, se envía un mensaje al cliente con código de estado HTTP 400 (Bad request)

Archivos estáticos

Actualmente en la carpeta del script se encuentran los ficheros necesarios para el buen funcionamiento del servidor, siendo ellos 5 archivos:

- index.html
- bad_request.html
- not_found.html
- favicon.ico

- image.webp

Todos los archivos de tipo html tienen una estructura básica, y el archivo index.html contiene el código para mostrar la imagen.

Adicionalmente se tiene una carpeta llamada *complex*, en dicha carpeta se encuentra un archivo llamado *index.html*, y 3 carpetas. Esta es la estructura típica de una aplicación web.

Las carpetas son las siguiente:

- css
- js
- images

En dichas carpetas se almacenan los ficheros de estilos, javascript y las imágenes usadas en el fichero index.html.

Se ha utilizado la librería CSS Bootstrap v5. Y se ha creado una estructura muy simple en el fichero de index.html que contiene un ejemplo de grilla de Bootsrap, un ejemplo de Carousel que despliega las imágenes de la carpeta images y que hace uso de JS. Adicinalmente, se ha implementado 2 links que permiten la visualización de un PDF (si es soportado por el navegador) o la descarga del mismo. El PDF elegido es el de la práctica.

Todo el código en el fichero html se puede conseguir en la página de Bootstrap, ya que fue tomado de los ejemplos. No es un diseño elaborado y se trató de mantener lo más sencillo posible.

Para probar el servidor se puede usar POSTMAN, que es una utilidad que permite realizar pruebas de aplicaciones web.

Para probar una solicitud POST desde el navegador se puede usar el siguiente formulario que se puede incrustar en algún fichero html, supongamos que se llame formulario.html:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Formulario</title>
</head>
<body>
  <form action="/" method="post">
    <input type="text" name="nombre" placeholder="Escriba su nombre">
    <br>
```

```
    <input type="text" name="apellido" placeholder="Escriba su apellido">
    <br>
    <input type="email" name="email" placeholder="Escriba su correo">
    <br>
    <button type="submit">Enviar</button>
  </form>
</body>
</html>
```

Esto enviará una solicitud tipo POST con los datos que se inserten en el formulario

Para probar otro tipo de contenido a enviar, se puede usar el programa curl desde la línea de comandos, por ejemplo:

Solicitud POST con datos JSON:

```
curl -X POST -H 'Content-Type: application/json' -d '{"hola": "mundo"}'
http://127.0.1.1:8080
```

Si se desea usar un protocolo específico:

```
curl --http1.0 -X POST -H 'Content-Type: application/json' -d '{"hola":
"mundo"}' http://127.0.1.1:8080
```

Si se desea descargar un fichero:

```
curl http://127.0.1.1:8080/Practica1_3.pdf --output practica.pdf
```