

Programación Avanzada

IIC2233 2024-1

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Dante Pinto - Francisca Cattán



Experiencia 4:

Sistema de almacenamiento

Debido a las fuertes lluvias, los servidores del DCC están caídos 🤖.

Como mucha gente depende de los servidores, te han pedido implementar un **servidor en Python** que permita enviar archivos a múltiples clientes.

Por suerte, ¡hay un código inicial con el que partir!

Experiencia 4: Sistema de almacenamiento

... sin embargo, el código está incompleto y posee errores.



¡A programar!

¿Qué queremos
lograr?

Servidor

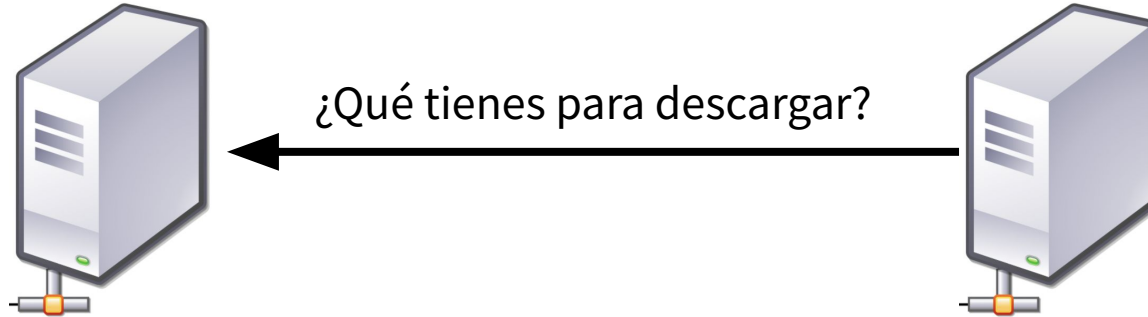


Cliente n



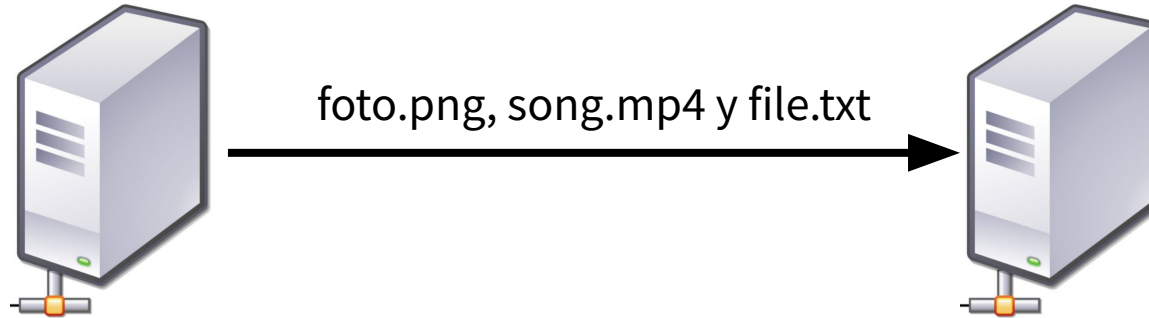
Servidor

Cliente n



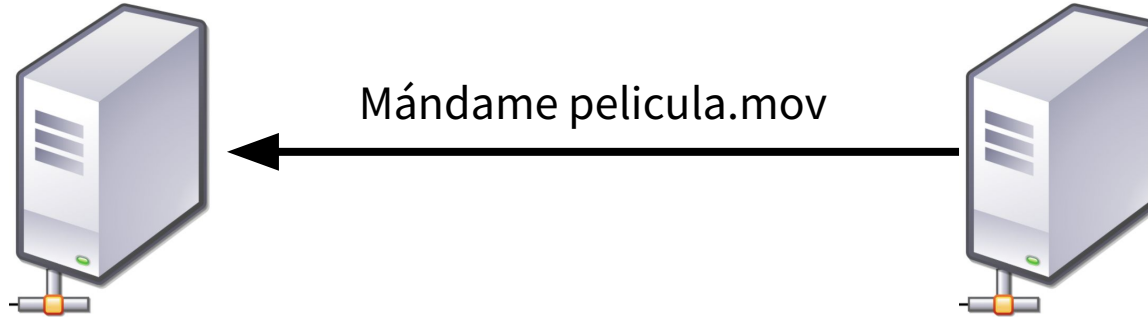
Servidor

Cliente n



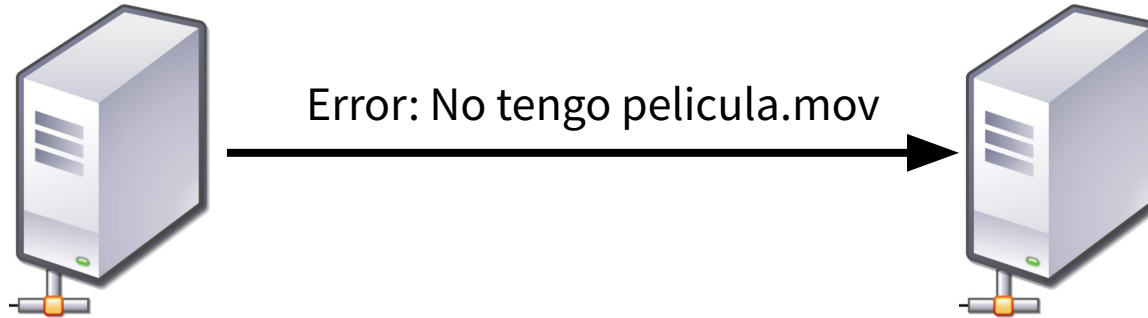
Servidor

Cliente n



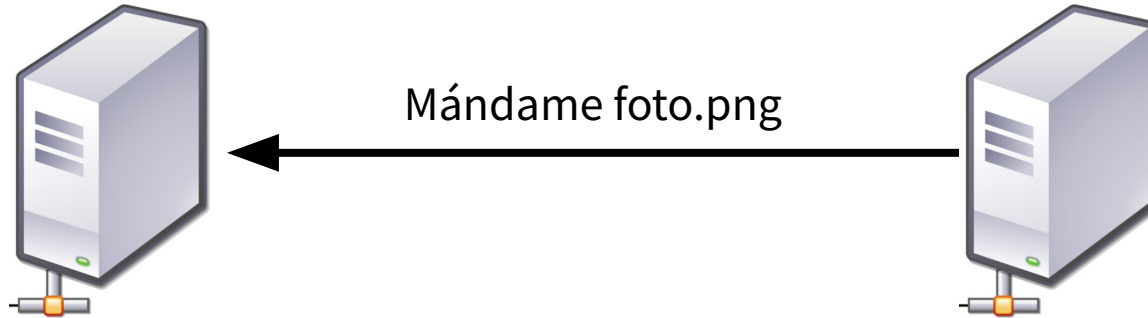
Servidor

Cliente n



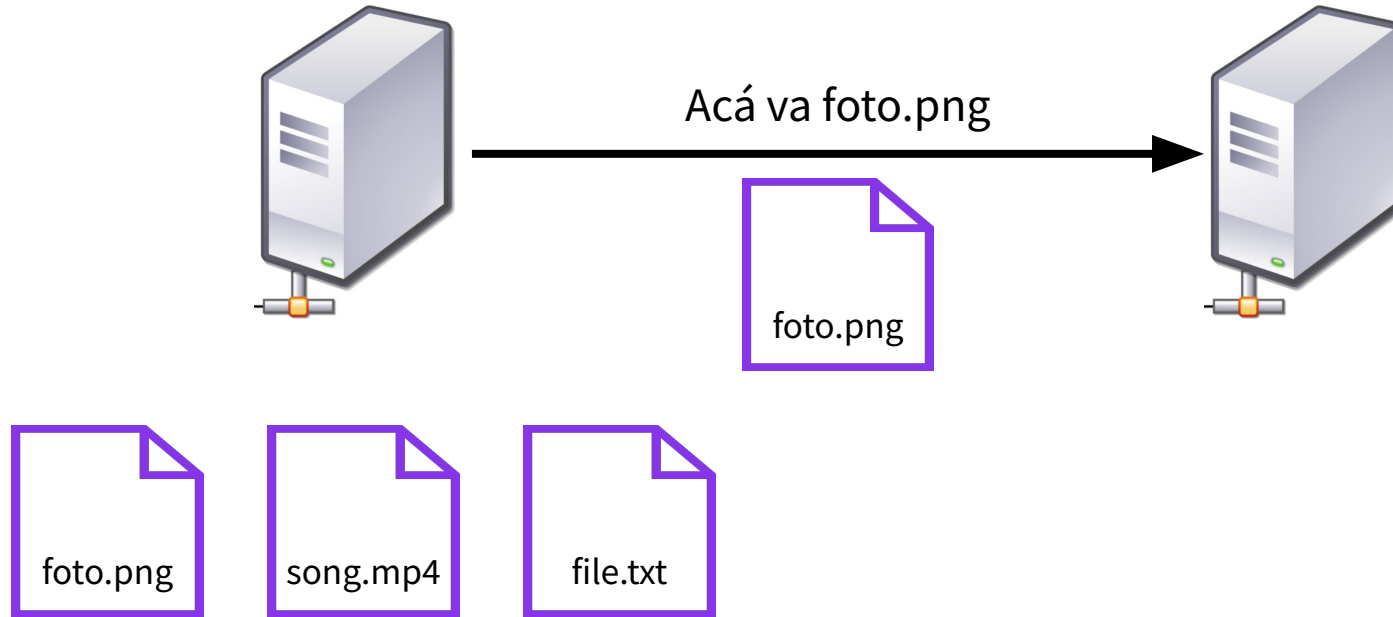
Servidor

Cliente n



Servidor

Cliente n



Servidor






Cliente n



¿Qué tenemos?

¿Qué tenemos?




 Atributo
 Método
 Incompleto o con errores

Servidor

```
 chunk_size: int
 host: str
 port: str
 sockets: dict()

 __init__(port: int, host: int)
 bind_listen()
 accept_connections()
 accept_connections_thread()
 recibir_bytes(socket_cliente: socket, cantidad: int)
 listen_client_thread(socket_cliente: socket)
 manejar_mensaje(mensaje: Mensaje, socket_cliente: socket)
 enviar_mensaje(mensaje: Mensaje, socket_cliente: socket)
 enviar_archivo(archivo: str, socket_cliente: socket)
 listar_archivos()
```

Mensaje

```
 operacion: str
 data: any
 estado: str
```

Cliente

En esta experiencia, trabajaremos sin revisar el archivo `cliente.py`, simulando que podría ser un programa en cualquier lenguaje.

Solo importará que sigamos convenciones en los mensajes enviados.

¡Solo editaremos `servidor.py`!

iA trabajar!

Programemos nuestro servidor


Al igual que con la experiencia anterior, trabajaremos **de a rondas**.

- Para cada ronda se indicará cuál es el resultado esperado a llegar tras corregir los errores o completar el código.
- Luego, analizaremos cómo llegar a dicho resultado.

Al final del día se publicará la solución donde se detalla cada método del código correctamente programado, y en caso de tener errores, cuáles eran estos.

Servidor (ronda 1)

Resultado esperado tras terminar

- Poder levantar el servidor, y que este se enlace al puerto y host del computador .
- En la terminal, podrás ver el texto:

Servidor escuchando en localhost : 3247

Servidor (ronda 1)

1. Al revisar servidor.py, el código no posee ninguna instancia de socket en ninguna parte. Debemos **instanciar** un socket, **enlazarlo a un *host* y *port***, y **aceptar conexiones**.
2. Ahora, si ejecutamos el código recibiremos el error:

TypeError: str, bytes or bytearray expected, not NoneType

3. Arreglado el error anterior, recibiremos otro error:

TypeError: 'str' object cannot be interpreted as an integer

4. Arreglado el error anterior, recibiremos otro error:

ConnectionRefusedError: [Errno 61] Connection refused

¡A programar/arreglar!  

Servidor (ronda 2)

Resultado esperado tras terminar

- El servidor logra conectarse a varios clientes a la vez, y esto no impide que podamos interactuar con el servidor mismo mediante la terminal.

Servidor (ronda 2)

1. Si levanto el servidor y ejecuto un cliente, este se conectará adecuadamente. Si ejecuto un segundo cliente, no veré ningún texto al respecto en la terminal ya que este no se pudo conectar al servidor. (¿Por qué no se cae el cliente?)
2. Ahora, puedo conectar varios clientes, pero ya no me aparece el mensaje “Presione enter para cerrar” ni pasa nada al presionar enter en la terminal.

¡A programar/arreglar!  

Servidor (ronda 3)

Resultado esperado tras terminar

- El servidor es capaz de des-serializar los mensajes que le llegan del cliente y transformarlos a un objeto del tipo mensaje. En la terminal se imprimirá el mensaje recibido.
- Para esto, es necesario revisar cuál fue el **protocolo de envío de datos** establecido para esta experiencia.

Protocolo de envío de datos

Para completar el servidor y el cliente, te entregan las siguientes reglas:

1. Los mensajes deben ser enviados utilizando **pickle**.
2. Para enviar mensajes, se utiliza la clase **Mensaje**.
3. Se deben enviar primero 4 *bytes* con el largo del mensaje, y luego el mensaje.

Servidor (ronda 3)

1. Primero, debemos completar el método “**recibir_bytes**” para que nos procese correctamente los *bytearrays* que le están llegando al servidor.
2. Ahora, haciendo uso de la función anterior y tomando en cuenta el protocolo mostrado anteriormente, debemos completar el método “**listen_client_thread**” para:
 - a. Usar la función “**recibir_bytes**” para obtener el largo del mensaje y luego el mensaje
 - b. Deserializar el bytearray con el mensaje.
 - c. Entregar el mensaje deserializado a la función “**manejar_mensaje**”.
 - d. Manejar algunos imprevistos comunes que podrían ocurrir.

¡A programar/arreglar!  

Servidor (ronda 4)

Resultado esperado tras terminar

- El servidor sabe cómo reaccionar distintos comandos del cliente, y manda de vuelta a este un mensaje apropiado para cada comando.

Servidor (ronda 4)

1. Debemos completar “**manejar_mensaje**” de tal forma que pueda ejecutar un flujo distinto para cada comando existente, dependiendo si estamos **enviando un mensaje** o un **archivo de vuelta**.
2. Ahora si desde el cliente pedimos la lista de archivos, esta nunca llegará y el servidor imprimirá:

Cliente desconectado

3. Tras encontrar el error anterior, el cliente recibirá un mensaje de vuelta al pedir los archivos pero este es:

no es una opcion válida

¡A programar/arreglar!  

Servidor (ronda 4)

4. Finalmente, debemos completar la función “**enviar_archivo**” para enviar archivos al cliente. ¿Cómo enviamos un archivo?
 - Recordemos que el podemos abrir un archivo en modo binario y rescatar los *bytes* que lo componen 🕶️

¡A programar/arreglar! 💻 🔧

Experiencia 4

Networking

Cierre de la experiencia

Resumen de errores

- Definir el **puerto como *string***.
- **No usar *threads* para escuchar a los clientes** en paralelo al flujo principal y entre ellos.
- **No usar *dump* o *dumps*** adecuadamente.
- **No seguir el protocolo** definido entre cliente y servidor al pie de la letra.

Desafíos

- Inventa otros comandos (puedes cambiar `cliente.py` si es necesario):
 - Enviar archivo a servidor.
 - Pedirle al servidor que duplique un archivo.
 - Pedirle al servidor que envíe un archivo a todos los clientes actualmente conectados.
- Inventa un nuevo protocolo de envío de mensajes más complejo que el actual.
- Usar el servidor como medio de comunicación entre dos clientes.

Programación Avanzada

IIC2233 2024-1

Hernán Valdivieso - Daniela Concha -

- Dante Pinto - Francisca Cattán

