

Programación Avanzada

IIC2233 2024-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattán



Experiencia 3:

Sistema de almacenamiento

Debido a problemas técnicos, los servidores del DCC están caídos 😓.

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

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

Experiencia 3: Sistema de almacenamiento

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



¡A programar!

¿Qué queremos
lograr?

Servidor

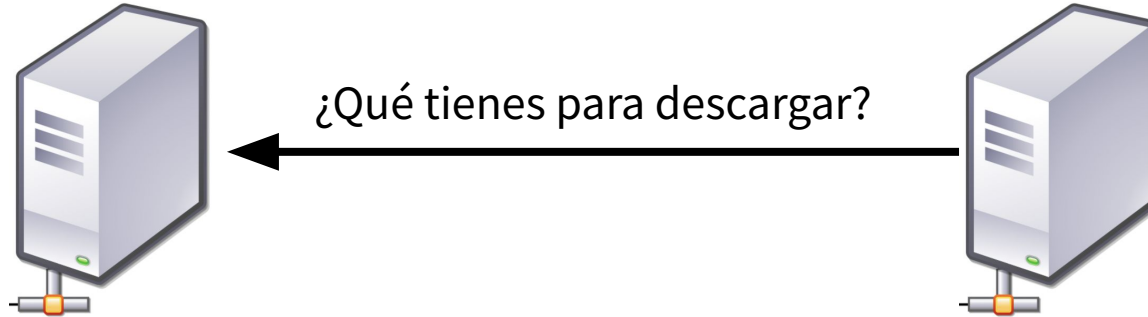


Cliente n



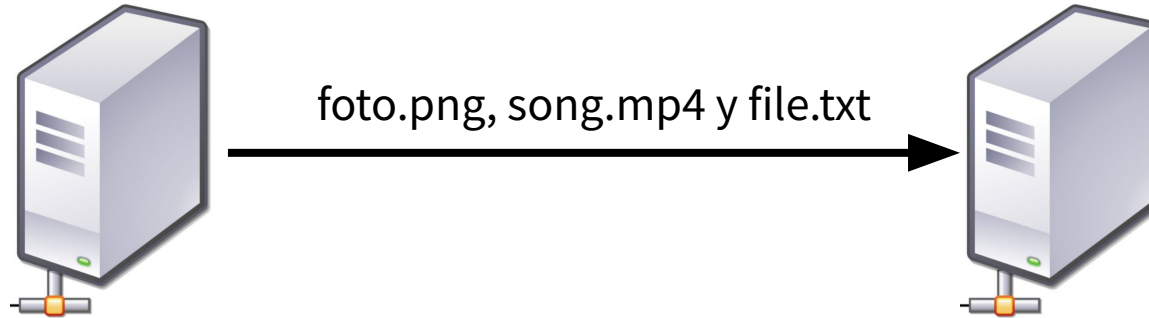
Servidor

Cliente n



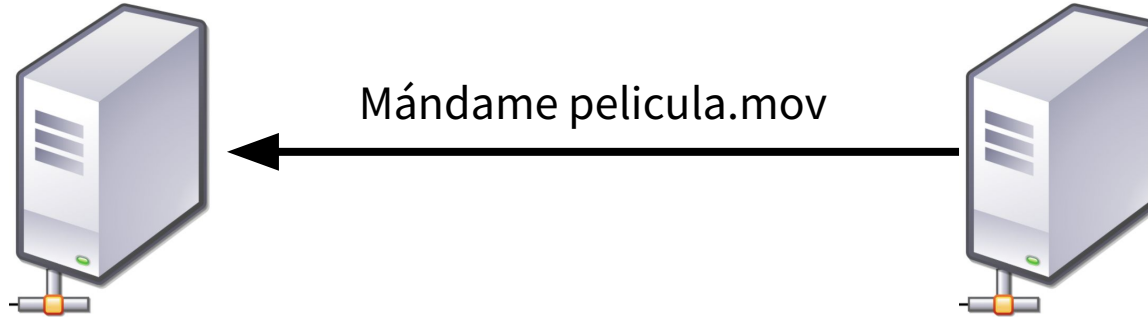
Servidor

Cliente n



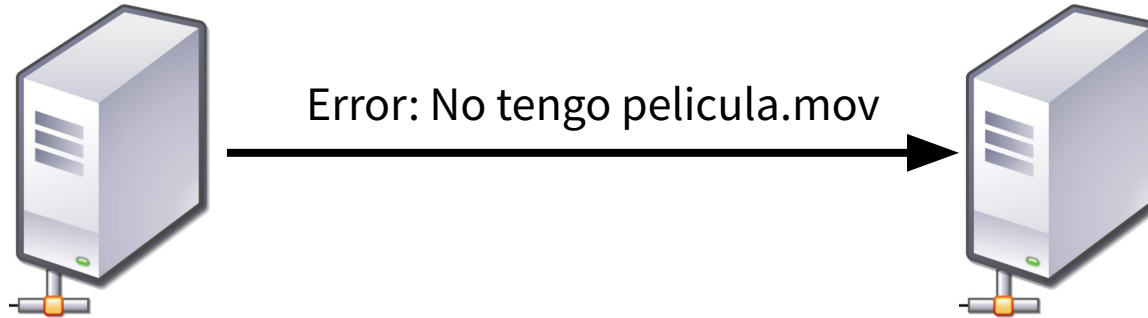
Servidor

Cliente n



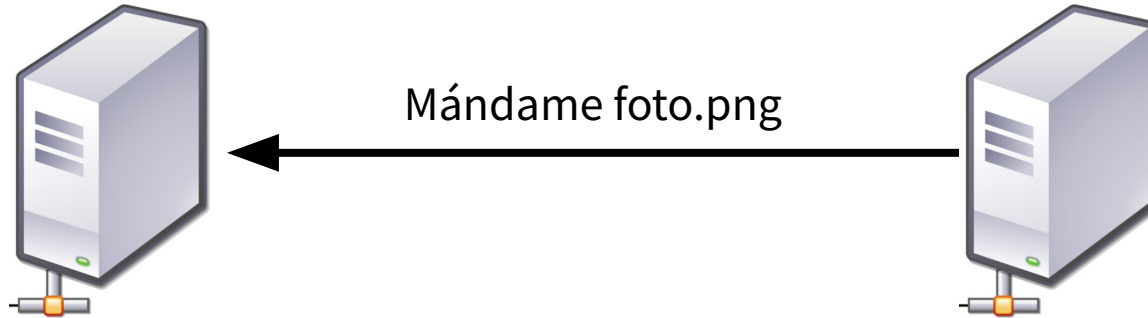
Servidor

Cliente n



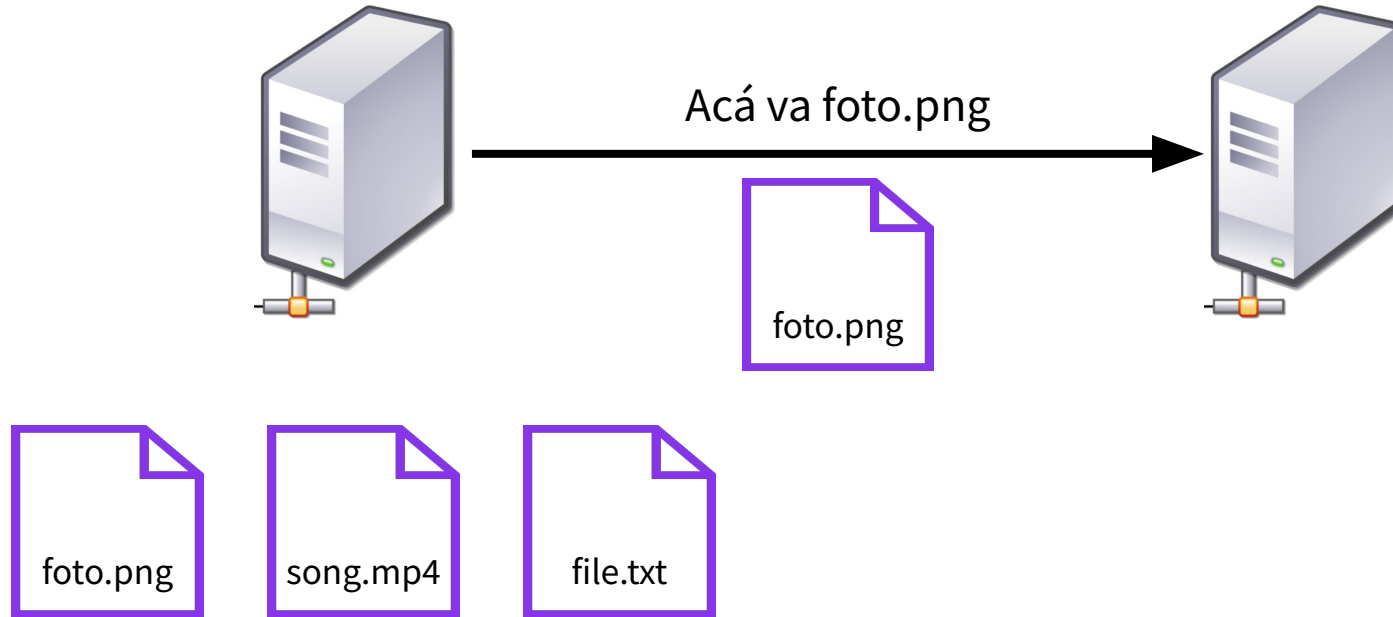
Servidor

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()`

¡Solo editaremos `servidor.py`!

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.

iA trabajar!

Cómo ejecutar Servidor y Cliente

Para ejecutar el **servidor**, deberás ir a la carpeta 'Servidor' y ejecutar el siguiente comando:

```
python3 servidor.py [PUERTO*] [DIRECCION*]
```

Para ejecutar un **cliente**, deberás en otra terminal, ir a la carpeta 'Cliente' y ejecutar el siguiente comando:

```
python3 cliente.py [PUERTO*] [DIRECCION*]
```

*: Ambas variables son opcionales. Si entregamos solo el puerto, se tomará **localhost** como dirección por defecto. Si no entregamos ninguna, se tomará **localhost** como dirección y **3247** como puerto por defecto.

Programemos nuestro servidor


Al igual que con las experiencia anteriores, 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 (Parte 1 y 2)

Resultado esperado tras terminar

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

Servidor escuchando en HOST : PORT

Presiona enter para cerrar

Servidor (Parte 1 y 2)

1. Al ejecutar servidor.py, veremos que la terminal solo indica lo siguiente:

Presione enter para cerrar

Esto es ya que al revisar el código, el servidor no tiene ningún socket instanciado. Debemos **instanciar** un socket, **enlazarlo a un *host* y *port***, y **aceptar conexiones**.

De la misma forma, debemos arreglar cualquier error que tengamos hasta poder llegar al mensaje:

Servidor escuchando en HOST : PORT

Presiona enter para cerrar

¡A programar/arreglar!  

Servidor (Parte 3)

Resultado esperado tras terminar

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

Servidor (Parte 3)

Si levanto el servidor y ejecuto un cliente, este se conectará adecuadamente. Si ejecuto un segundo cliente, no veré ningún texto al respecto sobre el nuevo cliente en la terminal del server (¿Por qué no se cae el cliente?).

¡A programar/arreglar!  

Servidor (Parte 4)

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 del servidor 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 (Parte 4 y 5)

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 anterior, debemos completar el método “**listen_client_thread**” para:
 - a. Usar la función “**recibir_bytes**”, 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 cualquier imprevisto que podría ocurrir.

¡A programar/arreglar!  

Servidor (Parte 6 y 7)

Resultado esperado tras terminar

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

Servidor (Parte 6 y 7)

1. Ahora si desde el cliente pedimos la lista de archivos, esta nunca llegará al cliente y el servidor imprimirá al final:

Cliente [N] desconectado

Debemos solucionar ese problema y cualquiera que quede hasta que el cliente pueda listar los archivos recibidos.

¡A programar/arreglar!  

Servidor (Parte 6 y 7)

2. Finalmente, debemos completar la función “**enviar_archivo**” para enviar archivos al cliente. ¿Cómo enviamos un archivo?

- Recordemos que podemos abrir un archivo en modo binario y rescatar los *bytes* que lo componen 😎

Con eso, tu programa estará ✨ Listo ✨

¡A programar/arreglar! 💻 🔧

Desafíos

- Inventa otros comandos (puedes cambiar `cliente.py` si es necesario):
 - Enviar un 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 2024-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattán

