USACH

REDES COMPUTACIONALES

Laboratorio N.º 1

Profesor:	Viktor Tapia	Alumno:	Rodrigo Pereira
Ayudante:	Luciano Yevenes	Fecha:	12/10/2025

1. Diferencias Observadas

1.1. ¿Qué diferencias notaron entre TCP y UDP al implementar?

La implementación de servidores TCP y UDP reveló diferencias significativas en el establecimiento de conexiones entre cliente y servidor. TCP exige una conexión previa, mientras que UDP no.

TCP (centro_control.py): El servidor sigue una secuencia lógica: bind() para asignar un puerto, listen() para escuchar, y accept() para recibir una conexión. El cliente usa connect() para establecer la conexión antes de intercambiar mensajes.

UDP (sistema_alerta.py): El proceso es más directo. El servidor usa bind() para escuchar un puerto y recvfrom() para recibir mensajes. A diferencia de TCP, no hay una conexión establecida; el cliente envía mensajes con sendto(), especificando la dirección del destinatario.

Además, el servidor TCP maneja un flujo continuo y ordenado de datos, utilizando socket.SOCK_STREAM. En contraste, el servidor UDP recibe datos en "datagramas" o paquetes discretos, especificado por socket.SOCK_DGRAM. Esta distinción es crucial para la fiabilidad, ya que los datagramas no garantizan el orden de llegada ni su entrega al destino.

1.2. ¿Cuál fue más fácil de programar y porque?

El servidor UDP (sistema_alerta.py) fue significativamente más sencillo en su lógica y extensión de código. Su implementación, al no requerir conexión, es mucho más directa que la programación necesaria para el servidor TCP (centro_control.py).

La lógica del servidor UDP es muy concisa: opera en un bucle, respondiendo a cada mensaje de forma independiente. Esto elimina la necesidad de gestionar el estado de una conexión o mantener un historial de mensajes, resultando en una interacción atómica donde cada mensaje se procesa y responde al instante.

En contraste, el servidor TCP requiere una lógica más compleja. Debe gestionar activamente la conexión y sus estados, ya que la acción a realizar depende del tipo de mensaje. Finalmente, procesa los datos según los requisitos específicos.

2. Problemas y Soluciones

2.1. ¿Qué problemas tuvieron durante la implementación?

La codificación en sí no fue lo más complejo. El mayor reto residió en la comprensión profunda de la funcionalidad de cada componente de la librería de sockets y en la distinción lógica entre un servidor TCP y uno UDP, especialmente en la gestión de la recepción de mensajes, que varía significativamente entre la conexión y la no conexión.

Adicionalmente, el laboratorio impuso requisitos específicos que aumentaron la complejidad. En el servidor TCP, fue crucial implementar una lógica para diferenciar tipos de mensajes y determinar acciones apropiadas, como guardar el mensaje o responder con un historial. Otros desafíos significativos incluyeron la validación del ingreso de información por parte del usuario y la gestión adecuada del cierre de conexiones, asegurando la finalización de los procesos de todos los servidores si la conexión con el cliente (estacion_espacial.py) se cerraba.

2.2. ¿Cómo los resolvieron?

La fase inicial implicó una exhaustiva revisión de la documentación de la librería socket (https://docs.python.org/es/3.10/library/socket.html). Esta consulta, complementada con los ejemplos vistos en clase, facilitó la elaboración de una traza lógica que describe el flujo de mensajes y la interacción entre el servidor y el cliente.

Para abordar la lógica de los requisitos específicos del laboratorio, se emplearon métodos de manipulación de cadenas (string). Estos permitieron identificar el inicio de un mensaje proveniente del servidor, lo que a su vez posibilitó la determinación de la acción correspondiente a ejecutar.

Adicionalmente, se desarrolló una función auxiliar denominada verificar_mensaje_tcp(). El propósito de esta función es validar la estructura de los mensajes ingresados por el usuario, garantizando así su correcta transmisión al servidor y la ejecución de las acciones esperadas.

```python
def verificar_mensaje_tcp(mensaje):
    if mensaje.startswith("REPORTE:") or mensaje == "CONSULTAR" or mensaje == "MISION_COMPLETA":
        return True
    return False
```

Finalmente, se añadió la opción “3. Salir” para asegurar un cierre adecuado de los servidores. Esta opción envía mensajes de cierre a los servidores TCP y UDP después de que el cliente (estacion_espacial.py) finaliza, garantizando una desconexión correcta.

3. Reflexión

3.1. ¿En qué casos usarían TCP vs. UDP?

La elección entre usar conexiones TCP y/o UDP se basa completamente en los casos de uso específicos, ya que cada protocolo posee características intrínsecas que los hacen idóneos para distintas necesidades.

Uso de TCP:

- Comunicación Confiable: Ideal para escenarios donde la entrega garantizada de mensajes es más importante que la velocidad. Por ejemplo, en situaciones donde la integridad del mensaje es crucial.
- **Correo Electrónico:** Un claro ejemplo donde la fiabilidad es primordial para asegurar que el contenido del mensaje llegue íntegro y cumpla su propósito.
- **Transferencia de Archivos:** Otra aplicación fundamental donde TCP es superior, ya que garantiza que los archivos se transfieran de forma completa y sin errores.

Uso de UDP:

- **Streaming en Tiempo Real:** Es el estándar para cualquier tipo de transmisión en vivo, puesto que la velocidad de transferencia es la característica más crítica, priorizando la inmediatez sobre la fiabilidad estricta.
- **Juegos en Línea:** Similar al streaming, la velocidad es primordial aquí, permitiendo una experiencia de juego en tiempo real con jugadores en diferentes ubicaciones geográficas.
- **Alertas de Emergencia:** UDP demuestra su eficacia en situaciones donde la red puede ser inestable o degradada. Un protocolo ligero y rápido es vital para asegurar que los mensajes críticos lleguen, marcando una diferencia significativa.

3.2. ¿Qué aprendieron sobre la programación con sockets?

En este laboratorio se exploraron los fundamentos de la comunicación basada en protocolos. Los aprendizajes clave incluyen:

- **El Socket como Interfaz de Programación:** Se utilizó el concepto de socket para establecer comunicaciones mediante protocolos UDP y TCP. Esta interfaz permite a las aplicaciones enviar y recibir datos a través de la red.
- **Roles de Cliente y Servidor:** Se comprendió la distinción entre cliente y servidor. El servidor escucha en una dirección y puerto específicos, mientras que el cliente inicia la comunicación conectándose a esa dirección.
- **Importancia de los Protocolos (UDP y TCP):** Se analizaron y aplicaron los protocolos UDP y TCP, identificando sus características, fortalezas y debilidades. Esto permitió discernir en qué escenarios cada protocolo es más adecuado.
- **Flujo de Datos en Bytes:** Se entendió que las redes procesan datos en formato de bytes. Por lo tanto, cualquier información compleja (como cadenas de texto u objetos) debe ser serializada (codificada) a bytes antes de su envío y deserializada (decodificada) al ser recibida.
- **Gestión del Ciclo de Vida de la Conexión:** Se aprendió y aplicó el proceso completo de comunicación, que incluye la creación de un socket, su enlace a una dirección, la escucha, la aceptación de conexiones, el intercambio de datos y el cierre ordenado de la conexión.