Tarea N° 2:

Sockets

Nicolás Barrera, José García

*COM4102 – Redes*

*Escuela de Ingeniería, Universidad de O’Higgins  
17, noviembre, 2023*

1. Introducción

Los sockets, fundamentales en el desarrollo de aplicaciones de red, han sido una pieza esencial en la evolución de la comunicación digital a través de Internet. En este contexto, la implementación de un chat online para granjeros de Stardew Fripi se presenta como un ejercicio práctico para explorar los conceptos fundamentales de los sockets TCP/IP y la programación concurrente.

El objetivo de esta tarea es diseñar un sistema de chat que no solo facilite la conversación entre usuarios, sino que también permita el intercambio de artefactos coleccionables, creando así una experiencia interactiva para los jugadores. Para alcanzar este propósito, se plantean una serie de requerimientos que abarcan desde la identificación única de los clientes hasta la gestión de intercambios de artefactos.

Este informe explorará los aspectos teóricos y prácticos de la implementación, detallando los conceptos clave relacionados con los sockets, la arquitectura cliente-servidor y la concurrencia mediante threads en Python. Se abordarán los desafíos particulares, como la sincronización de datos entre clientes y el manejo de intercambios, para lograr un funcionamiento fluido y robusto del sistema de chat.

A través de este proyecto, se busca no solo aplicar los conocimientos adquiridos sobre sockets y programación concurrente, sino también entender cómo estos elementos se combinan para crear soluciones prácticas y escalables en el ámbito de las aplicaciones de red. En las secciones siguientes, se detallará el marco teórico, la descripción de la implementación, seguido de un análisis de resultados y conclusiones derivadas de la experiencia de desarrollo.

1. Marco teórico

**¿Qué es un socket?**

Un socket es un mecanismo de comunicación que permite la transferencia de datos entre procesos, ya sea en la misma máquina o a través de una red. En el contexto de la programación de red, un socket es una interfaz que proporciona un punto final para enviar o recibir datos en una red. Los sockets pueden ser utilizados para la comunicación entre procesos en la misma máquina (comunicación interprocess) o para la comunicación entre procesos en máquinas diferentes a través de una red (comunicación en red).

**¿Cómo funciona un socket?**

En el modelo cliente-servidor, los sockets son utilizados para establecer conexiones entre un cliente y un servidor. El servidor crea un socket y espera pasivamente a que los clientes se conecten. Cuando un cliente desea establecer una conexión, crea un socket cliente y se conecta al socket del servidor. Una vez establecida la conexión, ambos extremos del socket pueden enviar y recibir datos de manera bidireccional.

El funcionamiento de un socket implica la utilización de protocolos de red, como TCP/IP, que definen las reglas para la transmisión de datos. TCP (Transmission Control Protocol) proporciona una comunicación fiable y orientada a la conexión, mientras que IP (Internet Protocol) facilita la transmisión de datos entre redes.

**¿Qué tipo de socket se utiliza en la implementación? ¿Cuáles son sus características?**

En la implementación del chat para granjeros de Stardew Fridi, se utiliza un socket TCP/IP. TCP (Transmission Control Protocol) es un protocolo de capa de transporte que proporciona una conexión fiable y orientada a la conexión. Utiliza un enfoque de "stream" para la transmisión de datos, garantizando la entrega secuencial y sin errores.

Características clave de un socket TCP en la implementación:

* **Conexión orientada:** TCP establece una conexión antes de iniciar la transferencia de datos, asegurando una comunicación confiable.
* **Flujo de datos:** Los datos se transmiten como un flujo continuo de bytes, lo que permite la transferencia eficiente de mensajes de longitud variable.
* **Fiabilidad:** TCP garantiza la entrega de datos en el orden correcto y maneja la retransmisión en caso de pérdida.
* **Full-duplex:** Permite la comunicación bidireccional, donde tanto el cliente como el servidor pueden enviar y recibir datos simultáneamente.

La elección de TCP/IP en esta implementación se basa en la necesidad de una comunicación confiable y ordenada entre el servidor y los clientes, aspectos fundamentales para un sistema de chat interactivo y la gestión de intercambios de artefactos.

1. Implementación

La implementación del servidor se realiza en el archivo **servidor.py**. A continuación, se presenta una descripción de las partes relevantes del código:

1. **Carga de Artefactos desde JSON:**

import json

# Cargar el objeto JSON desde el archivo

with open("artefactos.json", "r",

encoding="utf-8") as file:

    artefactos\_dict = json.load(file)

Se carga el archivo JSON que contiene la información sobre los artefactos. Este archivo se utiliza para asociar los números de artefactos con sus nombres.

1. **Configuración del Socket del Servidor:**

# Crear un socket del servidor

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind(("127.0.0.1", 12345))

server\_socket.listen(3)

Se crea y configura el socket del servidor para escuchar conexiones en la dirección IP "127.0.0.1" y el puerto 12345.

1. **Manejo de Conexiones de Clientes:**

# Escuchar y aceptar conexiones de clientes

while True:

    client\_socket, client\_addr = server\_socket.accept()

    cliente\_thread = threading.Thread(target=manejar\_cliente, args=(client\_socket, client\_addr))

    cliente\_thread.start()

El servidor espera activamente nuevas conexiones. Cuando un cliente se conecta, se crea un nuevo hilo (**cliente\_thread**) para manejar la comunicación con ese cliente.

1. **Función manejar\_cliente:**

* **Registro del Cliente:**

with client\_socket:

        try:

     # Establecer el nombre del cliente

client\_socket.send(b"[SERVER] Bienvenid@ al chat de Granjerxs! Por favor, introduce tu nombre:")

client\_name=client\_socket.recv(1024).decode().strip()

El cliente proporciona un nombre al conectarse al chat.

* **Notificación de Conexión a Otros Clientes:**

nombres\_clientes[client\_socket] =client\_name

 notificar\_conexion\_cliente(client\_socket)

enviar\_mensaje\_servidor(f"[SERVER] Cliente {client\_name} conectado.", client\_socket)

            print(f"[SERVER] {client\_name} se ha conectado desde {addr[0]}:{addr[1]}")

El servidor almacena el nombre del cliente y notifica a los demás clientes sobre la nueva conexión.

* **Artefactos:**

while True:

                # Preguntar por artefactos al unirse al chat

                mensaje\_arte = "[SERVER] Cuéntame, ¿qué artefactos tienes? (Ingresa los números separados por comas)"

                enviar\_mensaje\_servidor(mensaje\_arte, client\_socket)

                artefactos\_usuario = client\_socket.recv(1024).decode().strip().split(',')

El servidor solicita al cliente que ingrese los artefactos que posee y valida la entrada.

* **Notificación de Desconexión:**

 enviar\_mensaje\_servidor(f"[SERVER] Cliente {client\_name} desconectado.", client\_socket)

            notificar\_desconexion\_cliente(client\_socket)

Cuando un cliente se desconecta, se notifica a los demás clientes y se cierra su conexión.

* **Recepción y Manejo de Mensajes:**

while True:

  data = client\_socket.recv(1024)

 if not data:

       break

mensaje = data.decode()

 # Verificar si es un comando

 if mensaje.startswith(":"):

    comando = mensaje[1:].lower()  # Eliminar el ":" y convertir a minúsculas

 manejar\_comando(comando, client\_socket)

  else:

   mensaje\_reenviado = f"{client\_name}: {mensaje}"

 for otro\_cliente in clientes:

  if otro\_cliente != client\_socket:

         otro\_cliente.send(mensaje\_reenviado.encode())

     print(mensaje\_reenviado)

1. **Funciones de Comando:**

mensaje\_comando = None

    if comando == "smile":

        mensaje\_comando = " :)"

    elif comando == "angry":

        mensaje\_comando = " >:("

    elif comando == "combito":

        mensaje\_comando = " Q(’- ’Q)"

    elif comando == "larva":

        mensaje\_comando = " (:o)OOOooo"

    elif comando == "u":

        mostrar\_clientes\_conectados(cliente\_emisor)

        return

    elif comando.startswith("p "):

        enviar\_mensaje\_privado(comando, cliente\_emisor)

        return

    elif comando == "artefactos":

        mostrar\_artefactos(cliente\_emisor)

        return

    elif comando == "q":

        desconectar\_cliente(cliente\_emisor)

        return

    elif comando.startswith("artefacto "):

        obtener\_nombre\_artefacto(comando, cliente\_emisor)

        return

    elif comando.startswith(":offer "):

        iniciar\_intercambio(comando, cliente\_emisor)

        return

    if mensaje\_comando:

        mensaje\_a\_enviar = f"{nombres\_clientes[cliente\_emisor]}:{mensaje\_comando}"

        for cliente\_destino in clientes:

            if cliente\_destino != cliente\_emisor:

                enviar\_mensaje\_servidor(mensaje\_a\_enviar, cliente\_destino)

La función **manejar\_comando** se encarga de procesar los comandos ingresados por los clientes. Se manejan comandos como **:q** para desconectar, **:p** para mensajes privados, **:u** para mostrar usuarios, **:smile**, **:angry**, **:combito**, **:larva** para enviar emoticones, **:artefactos** para mostrar artefactos, **:artefacto <ArtefactoId>** para obtener el nombre de un artefacto, **:offer <Identificador> <MiArtefactoId> <SuArtefactoId>** para iniciar un intercambio, **:accept** para aceptar un intercambio, y **:reject** para rechazar un intercambio.

1. **Intercambio de Artefactos:**

def iniciar\_intercambio(comando, cliente\_emisor):

    parametros = comando.split()[1:]

    if len(parametros) == 3 and parametros[1].isdigit() and parametros[2].isdigit():

        cliente\_destino = parametros[0]

        mi\_artefacto\_id = int(parametros[1])

        su\_artefacto\_id = int(parametros[2])

        # Validar si los artefactos existen en la colección

        if str(mi\_artefacto\_id) in artefactos\_dict and str(su\_artefacto\_id) in artefactos\_dict:

            # Obtener los artefactos del emisor y el receptor

            artefactos\_emisor = artefactos\_clientes.get(cliente\_emisor, [])

            artefactos\_receptor = artefactos\_clientes.get(cliente\_destino, [])

            # Verificar si el emisor posee el artefacto que quiere intercambiar

            if mi\_artefacto\_id in artefactos\_emisor:

                mensaje\_intercambio = f"[SERVER] {nombres\_clientes[cliente\_emisor]} quiere intercambiar {artefactos\_dict[str(mi\_artefacto\_id)]} por {artefactos\_dict[str(su\_artefacto\_id)]}. ¿Aceptas? (Si/No)"

                enviar\_mensaje\_servidor(mensaje\_intercambio, cliente\_emisor)

                respuesta\_cliente\_destino = obtener\_respuesta\_si\_o\_no(cliente\_destino)

   if respuesta\_cliente\_destino.lower() == 'si':

                    # Realizar el intercambio

   artefactos\_emisor.remove(mi\_artefacto\_id)

 artefactos\_emisor.append(su\_artefacto\_id)

La función **iniciar\_intercambio** maneja el proceso de intercambio entre clientes, verificando si los artefactos existen, si el cliente posee el artefacto que quiere intercambiar y si el destinatario acepta el intercambio.

1. **Hilos y Threads:**

import threading

Se utiliza la biblioteca **threading** para manejar múltiples conexiones simultáneas. Cada cliente tiene su propio hilo, lo que permite la comunicación concurrente.

1. análisis y discusión

La implementación de un sistema de chat para granjeros en Stardew Fridi mediante sockets TCP/IP en Python presenta varios desafíos y aspectos interesantes. A continuación, se discuten algunos puntos relevantes y se presentan conclusiones:

**Manejo de Conexiones Múltiples:**

* La implementación utiliza threads para manejar conexiones múltiples de clientes simultáneamente. Esto permite que varios jugadores interactúen en el chat al mismo tiempo.

**Comunicación Cliente-Servidor:**

* El protocolo de comunicación entre el cliente y el servidor se basa en el intercambio de mensajes de texto. Los clientes envían mensajes al servidor, que luego distribuye estos mensajes a todos los demás clientes. Además, se han implementado comandos específicos con ":" para realizar acciones como enviar mensajes privados, mostrar usuarios conectados, enviar emoticones, mostrar artefactos, obtener información sobre artefactos y realizar intercambios de artefactos.

**Registro y Notificación de Eventos:**

* Se registra y notifica a los clientes sobre eventos importantes, como la conexión y desconexión de otros jugadores. Esto proporciona una experiencia más interactiva y social en el chat.

**Intercambio de Artefactos:**

* La capacidad de intercambiar artefactos entre los jugadores agrega una dimensión adicional al sistema. La implementación requiere que el jugador que inicia el intercambio posea el artefacto que está dispuesto a intercambiar y que el destinatario tenga el artefacto solicitado. Este proceso implica una negociación y aceptación mutua.

**Ventajas y Limitaciones:**

* La implementación en base a sockets TCP/IP permite una comunicación fiable y orientada a la conexión entre el cliente y el servidor. Sin embargo, en entornos reales, podría ser beneficioso considerar la seguridad, como la encriptación de la comunicación, para proteger la privacidad de los usuarios. Además, la implementación actual no maneja la persistencia de datos de los usuarios entre sesiones, lo que podría ser una mejora futura.

1. Conclusión

La implementación logra satisfactoriamente los objetivos del enunciado al proporcionar un sistema de chat interactivo para granjeros en Stardew Fridi. La capacidad de intercambiar artefactos y utilizar comandos específicos mejora la experiencia del usuario. La aplicación de threads para gestionar conexiones múltiples y el uso de un mutex para evitar problemas de concurrencia son consideraciones importantes en entornos de programación concurrente.

En resumen, la implementación presenta una base sólida para un chat en tiempo real para jugadores, abordando varios aspectos como la comunicación, la interacción entre usuarios y la gestión de eventos. La aplicación podría mejorarse mediante la incorporación de características adicionales, como la persistencia de datos, y considerando aspectos de seguridad para entornos de producción.

bibliografía

1. Luz, S (2022). Qué es un socket TCP o UDP y qué diferencias hay con los puertos. Redeszone. https://www.redeszone.net/tutoriales/configuracion-puertos/que-es-socket-tcp-udp-diferencias-puertos/
2. Ehijo, A & Bugueño, L, (2023). Introducción a Internet. Universidad de O’Higgins
3. Altube, R(2021). Wireshark: Qué es y ejemplos de uso. OpenWebinars. https://openwebinars.net/blog/wireshark-que-es-y-ejemplos-de-uso/.
4. IBM, (2023). Sockets. https://www.ibm.com/docs/es/aix/7.3?topic=concepts-sockets
5. IBM,(2023). Cómo funcionan los sockets. https://www.ibm.com/docs/es/i/7.5?topic=programming-how-sockets-work