Sobre la integridad en el procesamiento de

[EMPRESA INTERNACIONAL DE LOGÍSTICA]

Consultoría realizada por:

Juan Luis Ruano Muriedas

Antonio José Suárez García

Índice

Resumen ejecutivo

Tras un exhaustivo estudio de su problema, nos gustaría proponerle las distintas **soluciones** a las que hemos llegado para cada una de sus consultas.

Para la ***Consulta 1*** hemos desarrollado un código para su servidor en el que se hace uso de un **buffer** para **evitar el solapamiento** de los datos introducidos por los empleados de su empresa. El código ha demostrado ser **muy eficiente**, pudiendo manejar una cantidad de peticiones mucho mayor de la que su empresa maneja.

Para la ***Consulta 3*** hemos estudiado las cuestiones que nos planteaban, así como las tecnologías que estaban sopesando para la **descentralización** de su sistema de ficheros. Desde **Insegus** le recomendamos el uso de **Resilio Sync**, la cual creemos que es una tecnología que se ajusta perfectamente a sus necesidades de mantener la **integridad de los datos** tanto en el almacenamiento como en el procesamiento, además de proporcionar grandes cualidades como una alta velocidad de transferencia de datos.

Metodología y resultados

Consulta 1

Tras el estudio de sus necesidades en relación a la *Consulta 1* se ha tomado la decisión de realizar el código fuente en *Python* para el desarrollo del socket debido a su **accesibilidad y facilidad de mantenimiento**.

El socket hace uso de las librerías *socket* y *threading* de python para las funciones TCP/IP del socket y el buffer de datos necesarios para la paralelización respectivamente.
El script empieza con el ajuste de la **dirección IP de su servidor** así como el **número de puerto por el que se accede** y después se inicializa la variable *kilometrajes* que se trata de una variable global que almacena los kilómetros totales que es el objetivo del seguimiento. Puede inicializar esta variable como 0 o puede poner el valor que tengan registrado actualmente para no perder el progreso.

A continuación verá un método llamado *es_cadena_de_bytes_enteros* cuyo proposito es asegurar que los bytes recibidos coinciden a una cadena de enteros antes del procesamiento de los datos de tal manera que **no se pueda introducir ningún código malicioso**, de lo contrario saltará un mensaje de error. **Si desea aumentar la seguridad en este aspecto**, le recomendamos que incluya validación *front-end* en el extremo del cliente.

Después de dicha validación empezará el método encargado de procesar las peticiones del cliente la cual hace uso del método *threading* para el **buffer de datos** y la variable global *kilometrajes* que es en donde quedarán almacenados los kilómetros totales tras el procesamiento. Una vez completado el procesamiento se devuelven mensajes de éxito al cliente por confirmación.

El último método corresponde al **despliegue del servidor** que al activarse empezará a escuchar las peticiones de los clientes que lleguen y las enviará a procesar al método descrito anteriormente.

A continuación se presenta el **código descrito comentado**:

```
# -*- coding: utf-8 -*-

import socket
import threading
# Dirección IP y puerto del servidor
SERVER_IP = '0.0.0.0'
SERVER_PORT = 12345
#Inicializa la variable global, puede poner el número que desee
kilometrajes = 0
# Método de validación de enteros
def es_cadena_de_bytes_entero(cadena_de_bytes):
    try:
        # La siguiente line intenta convertir los bytes a un entero
        entero = int.from_bytes(cadena_de_bytes, byteorder='big')
        return True
    except ValueError:
        return False
# En caso de que la validación falle se devolverá False,
# en caso contrario se enviará True
# Función para manejar las conexiones de los clientes
def handle_client_connection(client_socket):
    # Se crea la variable lock que usaremos para determinar
    # una zona crítica y así crear el buffer
    lock = threading.Lock()
```

```
# Llamada a la variable global kilometrajes
global kilometrajes

while True:
    # Recibe los datos enviados por el cliente
    data = client_socket.recv(1024)
     # Validación de que no se trate de un campo vacio
    if not data:
        client_socket.send(b'Error en el formato de los datos.\n')
        break
    # Validación de que se trate de una cadena de enteros
    if not es_cadena_de_bytes_entero(data):
        client_socket.send(b'Error en el formato de los datos.\n')
        break

    # Decodifica los datos y actualiza el kilometraje del vehículo correspondiente
    try:
        # Comienzo de la zona crítica del código
        lock.acquire()
        # Aqui se decodifica y se realiza la suma de datos
        kilometers = data.decode()
        kilometrajes = kilometrajes +int(kilometers)
        lock.release()
        # Fin de la zona crítica del código

    except (ValueError, KeyError):
        # Maneja errores de formato de datos o claves no válidas
```

```
            client_socket.send(b'Error en el formato de los datos.\n')
            continue

        # Confirma la recepción de los datos
        client_socket.send(b'Datos recibidos correctamente.\n')
        # Cierra la conexión con el cliente
        client_socket.close()
# Función principal para iniciar el servidor
def start_server():
    # Crea un socket TCP/IP
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Enlace del socket con la dirección y el puerto del servidor
    server_socket.bind((SERVER_IP, SERVER_PORT))
    # Escucha conexiones entrantes
    server_socket.listen(5)

    while True:
        # Acepta conexiones entrantes y crea un nuevo hilo para manejar cada cliente
        client_socket, client_address = server_socket.accept()
        client_thread = threading.Thread(target=handle_client_connection,
                                         args=(client_socket,))
        client_thread.start()

#Función main para iniciar el socket
if __name__ == '__main__':
    start_server()
```

En cuanto a las **pruebas del código**, se ha realizado un test de estrés con *Python* usando las librerías *socket* en un entorno local simulando un número de clientes cada uno con un número de peticiones y en cada petición se añaden 100 kilómetros al contador total del servidor. Las especificaciones del sistema que realizó las pruebas son las siguientes:

- Procesador: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
- RAM: 16,0 GB (15,8 GB usable)

Para el test se han seleccionado **20000 clientes**, ya que su empresa cuenta con 12.736 vehículos y queríamos asegurarnos que tuvieran margen en el caso de que en un futuro amplíen su plantilla. Cada cliente enviará **100 peticiones**, un caso hipotético pero queríamos poner a prueba la solidez del código. Y como ya se mencionó, cada uno de ellos **añadirá 100 kilómetros al contador** del socket.

El test comienza mandando las peticiones de los clientes en paralelo y los resultados de la prueba fueron satisfactorios. **El total de tiempo de procesamiento fue de 68,3 segundos** y **no se perdieron peticiones en el proceso**. Se volvieron a repetir las pruebas de rendimiento pero capando el rendimiento de la CPU y entonces es cuando se empezó a detectar una pérdida de peticiones de aproximadamente 0,2% al 4% y el tiempo de procesamiento aumentó alrededor de 60 segundos.

En su caso, su servidor contará con un rendimiento mayor que en estos casos incluso que en nuestras pruebas más satisfactorias y el número de peticiones que reciba el socket

serán menores por lo que podemos asegurar que **su servidor tendrá un rendimiento óptimo.**

Para la instalación del servidor necesitará, además del hardware, las herramientas software *Python* versión 3.10 o superior.

Junto con *Python*, necesitará un entorno de servidor que soporte *Python*, como por ejemplo gunicorn donde mostraremos un ejemplo:

- Haciendo uso del comando *pip*, instale *gunicorn* con la línea: **pip install gunicorn**
- Cree el archivo del script con extensión *.py*
- Ejecuta una ventana de comandos y en el directorio del archivo *.py*, ejecute la línea de comandos para inicializar el socket: **gunicorn -b 0.0.0.0:8000 app:app**
- Para configurar el servidor puede hacer uso de un archivo de configuración *.conf*

Tenga en cuenta que esta es una guía básica y que los detalles exactos pueden variar según tu entorno específico y los requisitos del entorno seleccionado. Recuerde también que siempre puede contar con nosotros para la instalación del socket en su servidor.

Consulta 3

Tras el estudio de sus necesidades en relación a la *Consulta 3* hemos procedido al estudio de las distintas propuestas tecnológicas que nos comentaron para la descentralización del almacenamiento y procesamiento de los datos de su empresa y nos gustaría exponerle la que consideramos como la mejor solución a las cuestiones que nos plantean, en este caso, ***Relisio Sync***.

Resilio Sync es una aplicación de sincronización de archivos **P2P** que responde a cada una de las cuestiones que su empresa se plantea al ser complementada con otras tecnologías para mejorarla:

1. En cuanto a la inmutabilidad de los datos, se puede complementar *Resilio Sync* con un proceso de **sellado temporal** (time-stamping). Con un sistema de **firmas digitales** o **hashes criptográficos** se pueden crear dichos sellos y configurar Resilio Sync para que cambie los datos a *Solo lectura*, evitando así que puedan ser modificados.

2. En cuanto a la consistencia de los datos, *Resilio Sync* permite establecer una **programación regular para la sincronización de datos entre los nodos**, así como la **configuración de alertas** para advertir de posibles desviaciones o inconsistencias en los datos entre los nodos. Cuando hay errores de sincronización, *Resilio Sync* crea **copias de los datos** con los distintos valores de los mismos, lo cual permite solucionar los problemas de sincronización de manera proactiva.

3. En cuanto a los riesgos relacionados con el procesamiento, recomendamos el uso de una **lista blanca**. Implementando esta tecnologías aseguramos la seguridad de la red ante el **Sybil Attack** y el **Eclipse Attack**, ya que solo se le daría acceso a los dispositivos recogidos en la misma.

En caso de que finalmente se decida a adoptar un sistema de ficheros descentralizados, recuerde que en **Insegus** estamos dispuestos a ayudarle con la implementación del mismo.