

Práctica Calificada: Implementación del Algoritmo de Christian

Diana Yuliza Mamani Vilca

17 de diciembre de 2024

Objetivo

El objetivo principal de esta práctica calificada es implementar y analizar el funcionamiento del **algoritmo de Christian**, un método ampliamente utilizado para la sincronización de relojes en sistemas distribuidos. A través de esta implementación, se busca que los estudiantes comprendan la importancia de mantener una sincronización precisa del tiempo en entornos donde múltiples dispositivos o nodos interactúan entre sí. El caso práctico simula un sistema real de **sensores distribuidos** que monitorizan la temperatura en una fábrica, donde cada sensor tiene un reloj local que requiere ser sincronizado con un servidor central que actúa como fuente de tiempo confiable. Además, se evaluará el impacto de factores como el **retraso de red** en la precisión de la sincronización.

Caso Práctico

El escenario propuesto consiste en un sistema de sensores distribuidos ubicados en diferentes puntos de una fábrica. Estos sensores están programados para medir y reportar las temperaturas de manera periódica, pero debido a las diferencias en los relojes locales de cada sensor, pueden surgir inconsistencias temporales en las mediciones.

Para resolver este problema, se empleará el algoritmo de Christian, que consiste en sincronizar los relojes de los clientes (sensores) con un servidor central. El servidor central actúa como la **fuentes de tiempo confiable** y responde a las solicitudes de sincronización de cada cliente enviando su hora actual. El cliente, al recibir esta hora, ajusta su reloj local considerando el **tiempo de ida y vuelta** (RTT, por sus siglas en inglés) de la comunicación, calculando así una hora ajustada más precisa.

Este caso práctico permite demostrar cómo un método simple de sincronización puede ser implementado de forma efectiva y aplicarse en escenarios reales, tales como fábricas, sistemas de monitoreo ambiental o redes de sensores IoT.

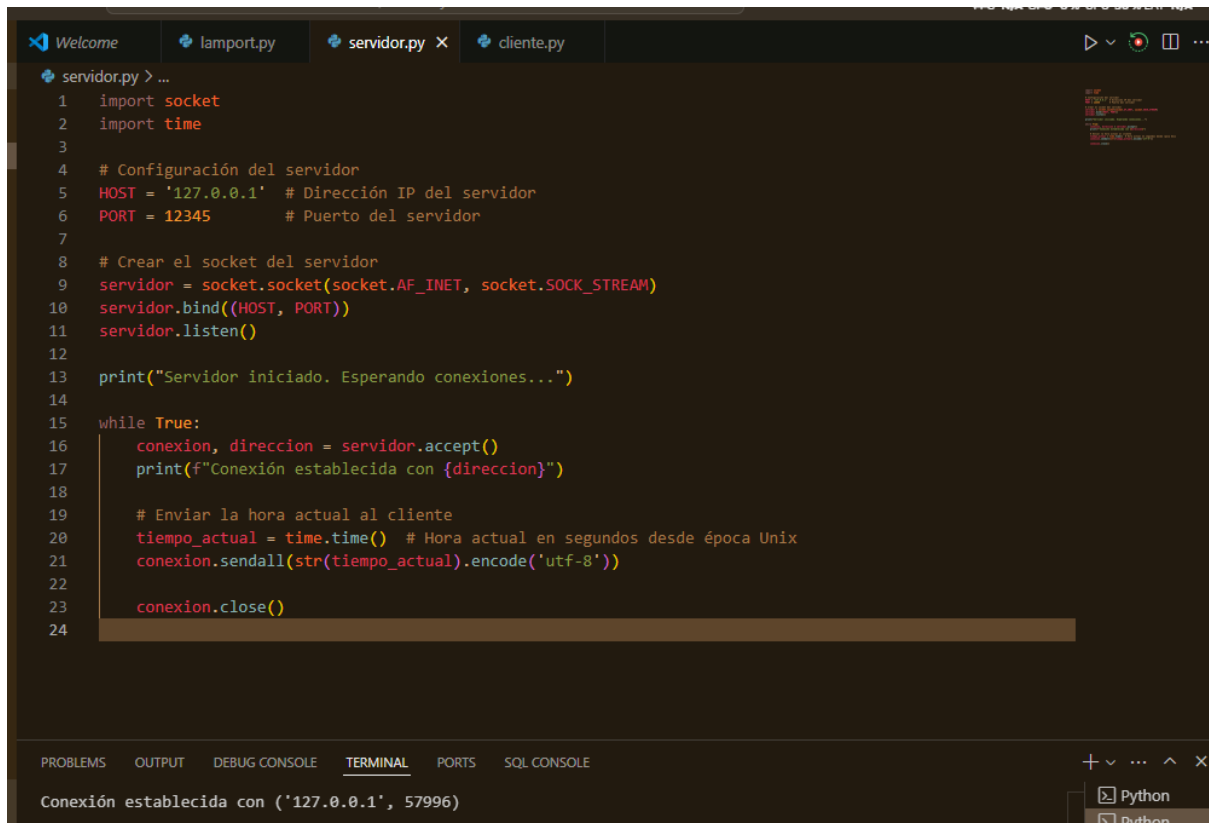
Implementación

La implementación se realiza en Python y se divide en dos módulos:

Servidor

El servidor representa el nodo central que mantiene la hora de referencia. Este se encarga de aceptar conexiones de los clientes (sensores) y responder a sus solicitudes de sincronización enviando la hora actual del sistema.

El siguiente fragmento de código muestra cómo se implementa el servidor:



```
servidor.py > ...
1 import socket
2 import time
3
4 # Configuración del servidor
5 HOST = '127.0.0.1' # Dirección IP del servidor
6 PORT = 12345      # Puerto del servidor
7
8 # Crear el socket del servidor
9 servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 servidor.bind((HOST, PORT))
11 servidor.listen()
12
13 print("Servidor iniciado. Esperando conexiones...")
14
15 while True:
16     conexion, direccion = servidor.accept()
17     print(f"Conexión establecida con {direccion}")
18
19     # Enviar la hora actual al cliente
20     tiempo_actual = time.time() # Hora actual en segundos desde época Unix
21     conexion.sendall(str(tiempo_actual).encode('utf-8'))
22
23     conexion.close()
24
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS SQL CONSOLE

Conexión establecida con ('127.0.0.1', 57996)

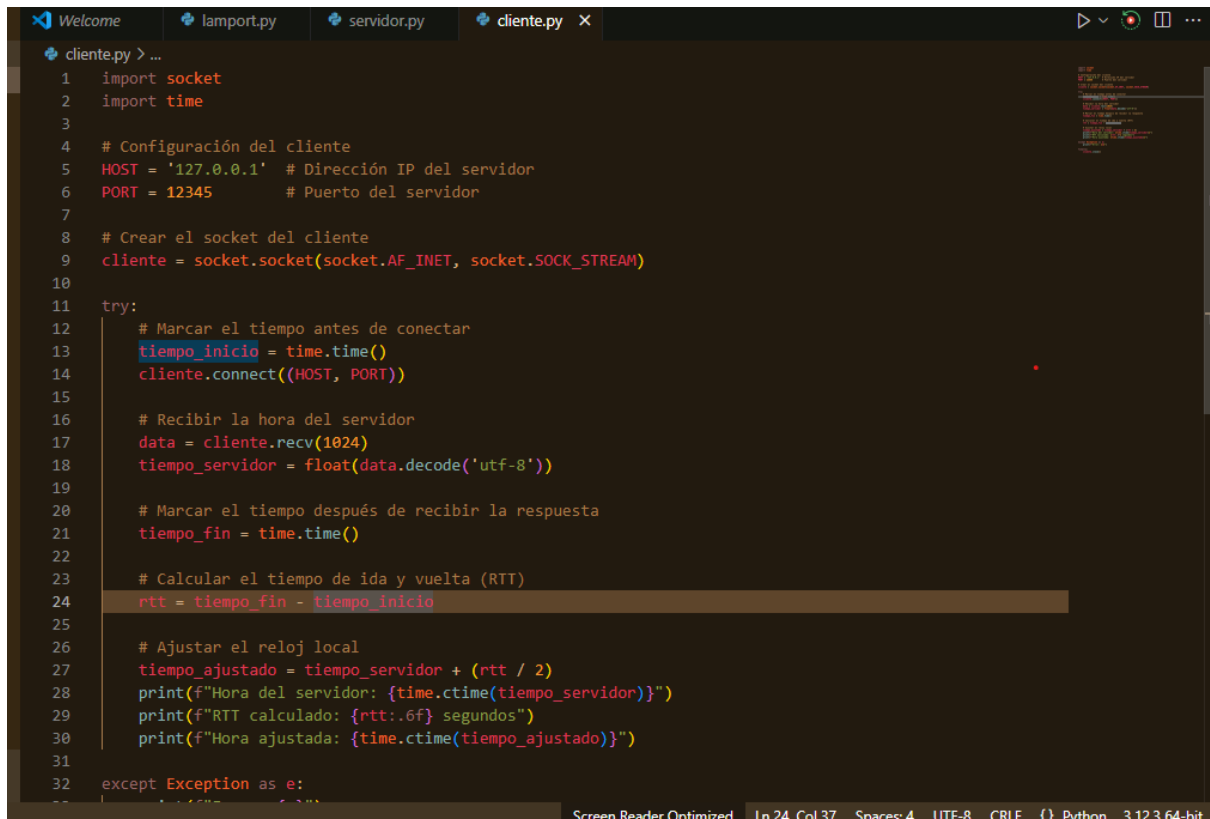
Python Python

Figura 1: Código del servidor que responde con la hora actual del sistema.

El servidor inicia escuchando conexiones en una dirección IP y puerto específicos. Una vez que recibe una conexión de un cliente, obtiene la hora actual utilizando la función `time.time()` y envía esta información al cliente, cerrando posteriormente la conexión.

Cliente

Los clientes representan los sensores distribuidos que solicitan la hora al servidor y ajustan sus relojes locales. El código del cliente es el siguiente:



```
1 import socket
2 import time
3
4 # Configuración del cliente
5 HOST = '127.0.0.1' # Dirección IP del servidor
6 PORT = 12345      # Puerto del servidor
7
8 # Crear el socket del cliente
9 cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10
11 try:
12     # Marcar el tiempo antes de conectar
13     tiempo_inicio = time.time()
14     cliente.connect((HOST, PORT))
15
16     # Recibir la hora del servidor
17     data = cliente.recv(1024)
18     tiempo_servidor = float(data.decode('utf-8'))
19
20     # Marcar el tiempo después de recibir la respuesta
21     tiempo_fin = time.time()
22
23     # Calcular el tiempo de ida y vuelta (RTT)
24     rtt = tiempo_fin - tiempo_inicio
25
26     # Ajustar el reloj local
27     tiempo_ajustado = tiempo_servidor + (rtt / 2)
28     print(f"Hora del servidor: {time.ctime(tiempo_servidor)}")
29     print(f"RTT calculado: {rtt:.6f} segundos")
30     print(f"Hora ajustada: {time.ctime(tiempo_ajustado)}")
31
32 except Exception as e:
```

Figura 2: .

Ejecución y Resultados

Para ejecutar la simulación:

1. Inicie el servidor ejecutando el archivo `servidor.py`.
2. Conecte varios clientes ejecutando el archivo `cliente.py`.

A continuación, se presentan las capturas de pantalla de la ejecución:

```
1 import socket
2 import time
3
4 # Configuración del servidor
5 HOST = '127.0.0.1' # Dirección IP del servidor
6 PORT = 12345 # Puerto del servidor
7
8 # Crear el socket del servidor
9 servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 servidor.bind((HOST, PORT))
11 servidor.listen()
12
13 print("Servidor iniciado. Esperando conexiones...")
14
```

```
PS C:\diana yuliza> & C:/Python312/python.exe "c:/diana yuliza/servidor.py"
Servidor iniciado. Esperando conexiones...
PS C:\diana yuliza> & C:/Python312/python.exe "c:/diana yuliza/servidor.py"
Servidor iniciado. Esperando conexiones...
PS C:\diana yuliza> & C:/Python312/python.exe "c:/diana yuliza/servidor.py"
Servidor iniciado. Esperando conexiones...
PS C:\diana yuliza> & C:/Python312/python.exe "c:/diana yuliza/servidor.py"
Servidor iniciado. Esperando conexiones...
Conexión establecida con ('127.0.0.1', 57995)
Conexión establecida con ('127.0.0.1', 57996)
PS C:\diana yuliza> & C:/Python312/python.exe "c:/diana yuliza/servidor.py"
Servidor iniciado. Esperando conexiones...
Conexión establecida con ('127.0.0.1', 57995)
Servidor iniciado. Esperando conexiones...
Conexión establecida con ('127.0.0.1', 57995)
Conexión establecida con ('127.0.0.1', 57996)
```

Figura 3: Ejecución del servidor mostrando las conexiones de los clientes.

```
Microsoft Windows [Versión 10.0.19045.5247]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>cd "C:\diana yuliza"

C:\diana yuliza>python cliente.py
Hora del servidor: Tue Dec 17 07:55:55 2024
RTT calculado: 0.000000 segundos
Hora ajustada: Tue Dec 17 07:55:55 2024

C:\diana yuliza>python cliente.py
Hora del servidor: Tue Dec 17 07:55:58 2024
RTT calculado: 0.000000 segundos
Hora ajustada: Tue Dec 17 07:55:58 2024
```

Figura 4: Ejecución del cliente mostrando la sincronización del reloj.

Conclusión

El **algoritmo de Christian** es un método sencillo pero eficiente para la sincronización de relojes en sistemas distribuidos. Durante esta práctica, se logró implementar y probar dicho algoritmo en un sistema simulado de sensores distribuidos. Se demostró que, a partir de la medición del **retraso de red** y la hora proporcionada por un servidor central, los clientes pueden ajustar sus relojes locales de manera precisa.

Esta sincronización es especialmente importante en aplicaciones donde la coherencia temporal es crítica, como en sistemas de monitoreo, control industrial y redes IoT. La práctica también resalta la necesidad de reducir la latencia de red para mejorar aún más la precisión en la sincronización del tiempo.

1. Códigos del Proyecto

1.1. Código del servidor (servidor.py)

```
import socket
import time

# Configuraci n del servidor
HOST = '127.0.0.1' # Direcci n IP del servidor
PORT = 12345      # Puerto del servidor

# Crear el socket del servidor
servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
servidor.bind((HOST, PORT))
servidor.listen()

print("Servidor iniciado. Esperando conexiones...")

while True:
    conexion, direccion = servidor.accept()
    print(f"Conexi n establecida con {direccion}")

    # Enviar la hora actual al cliente
    tiempo_actual = time.time() # Hora actual en segundos desde poca
    Unix
    conexion.sendall(str(tiempo_actual).encode('utf-8'))

    conexion.close()
```

1.2. Código del cliente (cliente.py)

```
import socket
import time

# Configuraci n del cliente
HOST = '127.0.0.1' # Direcci n IP del servidor
PORT = 12345      # Puerto del servidor

# Crear el socket del cliente
cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    # Marcar el tiempo antes de conectar
    tiempo_inicio = time.time()
    cliente.connect((HOST, PORT))

    # Recibir la hora del servidor
    data = cliente.recv(1024)
    tiempo_servidor = float(data.decode('utf-8'))

    # Marcar el tiempo despu s de recibir la respuesta
    tiempo_fin = time.time()

    # Calcular el tiempo de ida y vuelta (RTT)
    rtt = tiempo_fin - tiempo_inicio
```

```
# Ajustar el reloj local
tiempo_ajustado = tiempo_servidor + (rtt / 2)
print(f"Hora del servidor: {time.ctime(tiempo_servidor)}")
print(f"RTT calculado: {rtt:.6f} segundos")
print(f"Hora ajustada: {time.ctime(tiempo_ajustado)}")

except Exception as e:
    print(f"Error: {e}")

finally:
    cliente.close()
```