

Ingeniería de Software 3

Informe Actividad 1

Servidor TCP con thread pool

Juan David Pallares Pallares - 2220079
Daniel Alejandro Ayala Vallejo - 2220084

16 de Febrero de 2026

Link al repo: https://github.com/JxxnDx/TCP_Server_thread_pool.git

Link al video: <https://youtu.be/SVWlJb374k8>

1. Código

1.1. Descripción general

El sistema implementa un servidor TCP multi-threaded en Python que recibe mensajes de texto, identifica la última vocal, cuenta sus apariciones y determina si el resultado es un número primo. Utiliza un Thread Pool para gestionar múltiples conexiones concurrentes de manera eficiente.

1.2. Constructor

```

1 def __init__(self, host='0.0.0.0', port=5050, max_workers=50, log_file='resultados.txt'):
2     self.host = host
3     self.port = port
4     self.max_workers = max_workers
5     self.log_file = log_file
6     self.server_socket = None
7     self.pool = None
8     # Lock para escritura segura en el archivo
9     self.file_lock = threading.Lock()
10    self.vowels = set('aeiouAEIOU')
11
12    # Crear archivo con encabezado si no existe
13    self._initialize_log_file()

```

1.2.1. Propósito

Inicializa todos los parámetros y estructuras de datos del servidor.

1.2.2. Parámetros

Parámetro	Tipo	Descripción
host	str	Dirección IP en la que el servidor escuchará. Por defecto '0.0.0.0' (todas las interfaces)
port	int	Puerto TCP para aceptar conexiones. Por defecto 5050
max_workers	int	Número máximo de threads en el pool. Por defecto 50
log_file	str	Ruta del archivo de resultados. Por defecto 'resultados.txt'

Cuadro 1: Parámetros del constructor

1.3. Iniciar archivo txt

```
1 def _initialize_log_file(self):
2     """Inicializa el archivo de log si no existe."""
3     if not os.path.exists(self.log_file):
4         with open(self.log_file, 'w', encoding='utf-8') as f:
5             f.write("=" * 80 + "\n")
6             f.write("REGISTRO DE MENSAJES TCP - SERVIDOR DE CONTEO DE VOCALES\n")
7             f.write("=" * 80 + "\n\n")
8             logging.info(f"Archivo de log creado: {self.log_file}")
9     else:
10        logging.info(f"Usando archivo de log existente: {self.log_file}")
11
```

1.3.1. Propósito

Crea o verifica la existencia del archivo de log con un encabezado formateado.

1.4. Encontrar último caracter

```
1 def find_last_vowel(self, text):
2     """
3     Encuentra la última vocal en el texto.
4
5     Args:
6         text (str): Texto a analizar
7
8     Returns:
9         str or None: La última vocal encontrada o None si no hay vocales
10    """
11    # Recorrer el texto de derecha a izquierda
12    for char in reversed(text):
13        if char in self.vowels:
14            return char.lower()
15    return None
```

1.4.1. Propósito

Encuentra la última vocal presente en una cadena de texto mediante recorrido inverso.

1.4.2. Parámetros y Retorno

Elemento	Tipo	Descripción
<i>Entrada:</i> text	str	Cadena de texto a analizar
<i>Retorno:</i>	str o None	Última vocal en minúscula o None si no hay vocales

Cuadro 2: Parámetros de find_last_vowel

1.5. Contador de último caracter

```
1 def count_vowel_occurrences(self, text, vowel):
2     """
3     Cuenta cuántas veces aparece una vocal específica en el texto.
4
5     Args:
6         text (str): Texto a analizar
7         vowel (str): Vocal a contar
8
9     Returns:
10        int: Número de apariciones
11    """
12    if not vowel:
13        return 0
14    return sum(1 for char in text.lower() if char == vowel.lower())
```

1.5.1. Propósito

Cuenta el número total de apariciones de un caracter específica en el texto (case-insensitive).

1.5.2. Parámetros y Retorno

Elemento	Tipo	Descripción
<i>Entrada:</i> text	str	Cadena de texto a analizar
<i>Entrada:</i> vowel	str	Vocal a contar
<i>Retorno:</i>	int	Número de apariciones

Cuadro 3: Parámetros de count_vowel_occurrences

1.6. Definir si es primo o no

```

1  def is_prime(self, n):
2      """
3          Determina si un número es primo.
4
5          Args:
6              n (int): Número a verificar
7
8          Returns:
9              bool: True si es primo, False en caso contrario
10         """
11         if n < 2:
12             return False
13         if n == 2:
14             return True
15         if n % 2 == 0:
16             return False
17
18         # Verificar divisores hasta la raíz cuadrada
19         for i in range(3, int(n ** 0.5) + 1, 2):
20             if n % i == 0:
21                 return False
22         return True

```

1.6.1. Propósito

Determina si un número entero es primo mediante el algoritmo de división por tentativa optimizado.

1.7. Validar texto

```
1 def validate_text(self, text):
2     """
3     Valida que el texto solo contenga caracteres alfabéticos y espacios.
4
5     Args:
6         text (str): Cadena de texto a validar
7
8     Returns:
9         bool: True si es válido, False en caso contrario
10    """
11    # Permitir letras y espacios
12    return all(char.isalpha() or char.isspace() for char in text)
13
```

1.8. Guardar txt

```
1 def save_to_log(self, timestamp, message, count, is_prime_result, last_vowel):
2     """
3     Guarda el resultado en el archivo de log de manera thread-safe.
4
5     Args:
6         timestamp (str): Fecha y hora
7         message (str): Mensaje recibido
8         count (int): Número de veces que se repite la vocal
9         is_prime_result (str): "SI" o "NO"
10        last_vowel (str): Última vocal encontrada
11    """
12    with self.file_lock:
13        try:
14            with open(self.log_file, 'a', encoding='utf-8') as f:
15                f.write(f"Fecha: {timestamp}\n")
16                f.write(f"Mensaje: {message}\n")
17                f.write(f"Última vocal: {last_vowel}\n")
18                f.write(f"Repeticiones: {count}\n")
19                f.write(f"¿Es primo?: {is_prime_result}\n")
20                f.write("-" * 80 + "\n\n")
21            logging.info(f"Registro guardado en {self.log_file}")
22        except Exception as e:
23            logging.error(f"Error al guardar en archivo: {e}")
24
```

1.9. Conexión

```

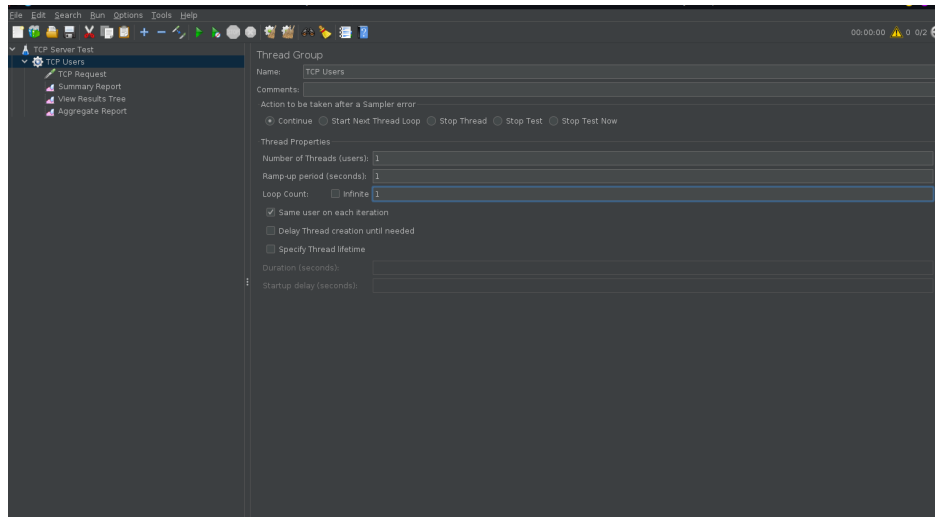
1  def handle_client(self, client_socket, client_address):
2      """Maneja la conexión de un cliente."""
3      logging.info(f"Nueva conexión desde {client_address}")
4
5      try:
6          # Recibir datos (buffer más grande para mensajes largos)
7          data = client_socket.recv(4096).decode('utf-8').strip()
8
9          if not data:
10             logging.warning(f"Datos vacíos recibidos de {client_address}")
11             client_socket.sendall(b"ERROR: No se recibieron datos\n")
12             return
13
14             logging.info(f"Mensaje recibido de {client_address}: '{data}'")
15
16             # Validar que solo contenga letras y espacios
17             if not self.validate_text(data):
18                 error_msg = "ERROR: El mensaje debe contener solo letras y espacios\n"
19                 logging.warning(f"Validación fallida para {client_address}: '{data}'")
20                 client_socket.sendall(error_msg.encode('utf-8'))
21                 return
22
23             # Verificar que no sea solo espacios
24             if not data.strip():
25                 error_msg = "ERROR: El mensaje no puede estar vacío o contener solo espacios\n"
26                 logging.warning(f"Mensaje vacío de {client_address}")
27                 client_socket.sendall(error_msg.encode('utf-8'))
28                 return
29
30             # Buscar la última vocal
31             last_vowel = self.find_last_vowel(data)
32
33             if not last_vowel:
34                 error_msg = "ERROR: El mensaje debe contener al menos una vocal\n"
35                 logging.warning(f"Sin vocales en mensaje de {client_address}")
36                 client_socket.sendall(error_msg.encode('utf-8'))
37                 return
38
39             # Contar las apariciones de la última vocal
40             count = self.count_vowel_occurrences(data, last_vowel)
41
42             # Verificar si el número es primo
43             is_prime_result = "SI" if self.is_prime(count) else "NO"
44
45             # Obtener timestamp
46             timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
47
48             # Guardar en archivo
49             self.save_to_log(timestamp, data, count, is_prime_result, last_vowel)
50
51             # Enviar respuesta al cliente
52             response = f"{count}\n"
53             client_socket.sendall(response.encode('utf-8'))
54
55             logging.info(
56                 f"Respuesta enviada a {client_address}: {count} "
57                 f"(vocal: '{last_vowel}', primo: {is_prime_result})"
58             )
59
60     except Exception as e:
61         logging.error(f"Error manejando cliente {client_address}: {e}")
62         try:
63             client_socket.sendall(b"ERROR: Error interno del servidor\n")
64         except:
65             pass
66     finally:
67         client_socket.close()
68         logging.info(f"Conexión cerrada con {client_address}")

```

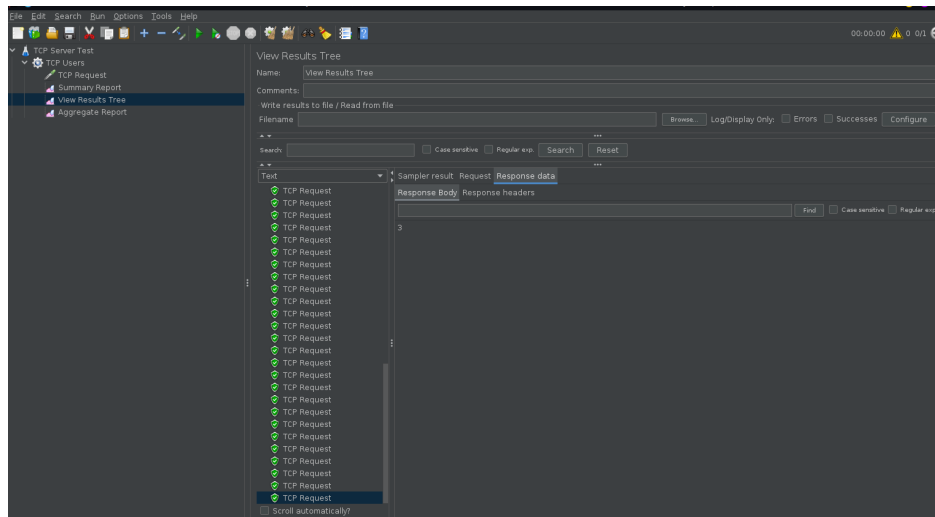
2. Pruebas con JMeter

2.1. Prueba con un solo thread

Primero, se realizara una prueba con un solo hilo que a su vez solo manda una petición:



El mensaje a enviar será "Hola soy pedro". Se comprueba que funciona correctamente:

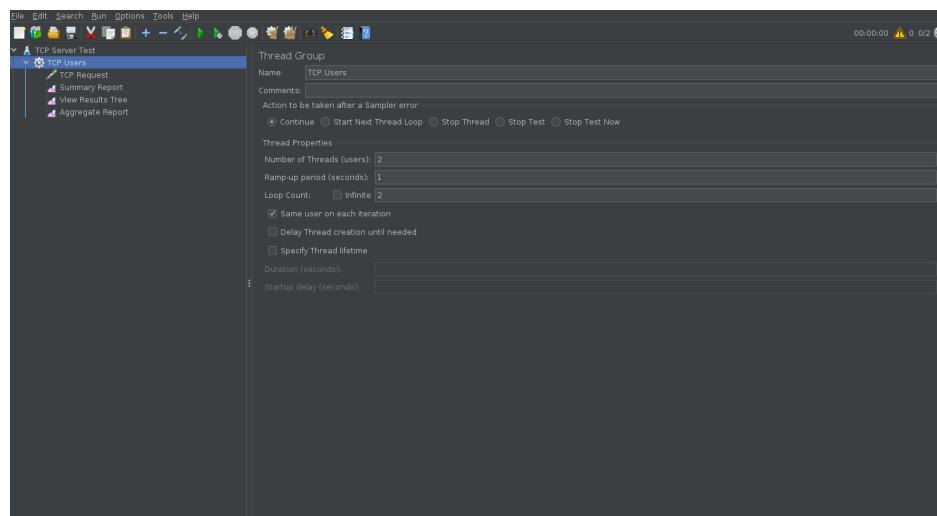


Se verifica en el txt:

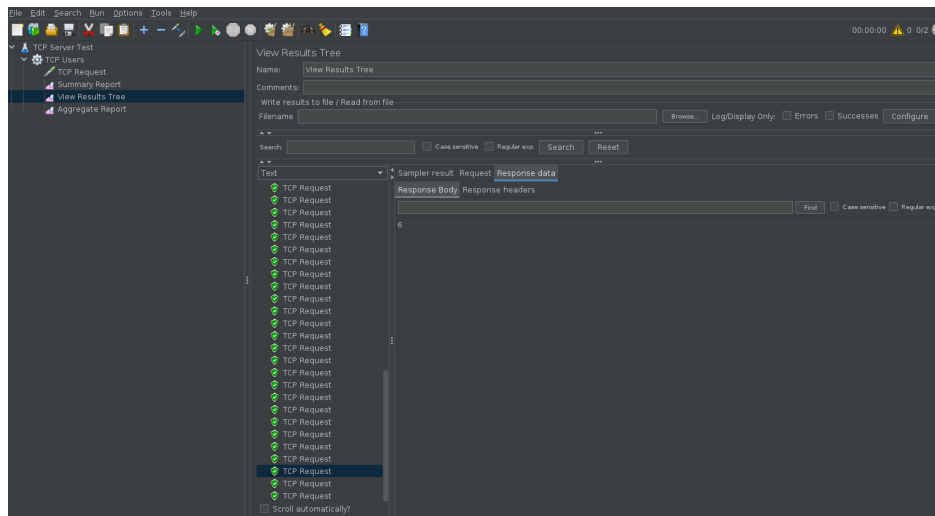
```
-----  
Fecha: 2026-02-16 20:36:37  
Mensaje: Hola soy pedro  
Última vocal: o  
Repeticiones: 3  
¿Es primo?: SI  
-----
```

2.2. Prueba con varios threads

Ahora, se hace una prueba con 2 hilos y con 2 peticiones:



El mensaje a enviar ahora será otorrino naringologo”. Se comprueba que funciona correctamente:



Se verifica también el txt:

```
Fecha: 2026-02-16 20:39:18
Mensaje: otorrino naringologo
Última vocal: o
Repeticiones: 6
¿Es primo?: NO
-----
```

```
Fecha: 2026-02-16 20:39:18
Mensaje: otorrino naringologo
Última vocal: o
Repeticiones: 6
¿Es primo?: NO
-----
```

```
Fecha: 2026-02-16 20:39:19
Mensaje: otorrino naringologo
Última vocal: o
Repeticiones: 6
¿Es primo?: NO
-----
```

```
Fecha: 2026-02-16 20:39:19
Mensaje: otorrino naringologo
Última vocal: o
Repeticiones: 6
¿Es primo?: NO
```