AOS -HW1 Report – A20541164

Overview:

This assignment involves developing a subscriber-publisher system designed to facilitate efficient message exchange between clients. The core components include:

1. Message Broker (Server Program): This server acts as the central hub for managing the communication between publishers and subscribers. It is responsible for routing messages from publishers to the appropriate subscribers, ensuring reliable and timely delivery.
2. Client API Library: A dedicated API library will be created to provide developers with tools to build their own subscriber and publisher applications. This library abstracts the complexities of the messaging protocol, allowing for straightforward implementation of messaging functionalities.
3. Client Programs: To validate the functionality and performance of the system, various client programs will be developed that function as both subscribers and publishers. These applications will be used to demonstrate the system's correctness and to benchmark its performance under different scenarios.

Overall, this assignment aims to establish a robust framework for message-driven communication, emphasizing both scalability and ease of use.

- **Firstly, I have written the code for ClientAPI as client_api.py.**

GNU nano 6.2 client_api.py

```
import socket
import uuid
import json

class ClientAPI:
    def __init__(self, host='localhost', port=8080):
        # Initialize with server details and establish connection
        self.host = host
        self.port = port
        self.client_socket = None
        self.connect()

    def connect(self):
        # Establish a socket connection with the server
        if self.client_socket:
            self.client_socket.close()
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.client_socket.connect((self.host, self.port))

    def send_and_receive(self, message):
        # Send message to server and receive response
        self.client_socket.send(json.dumps(message).encode('utf-8') + b'\n')
        response = ""
        while '\n' not in response:
            chunk = self.client_socket.recv(1024).decode('utf-8')
            if not chunk:
                raise ConnectionError("Connection closed by server")
            response += chunk
        return json.loads(response.strip())

    def register_publisher(self):
        # Generate and return a unique Publisher ID
        pid = str(uuid.uuid4())
        print(f"Publisher registered with ID: {pid}")
        return pid
```

Then, MessageBroker.py (Server) Program:

```
 GNU nano 6.2                                                MessageBroker.py
import socket
import threading
import json

class MessageBroker:
    def __init__(self, host='localhost', port=8080):
        # Initialize server and data structures
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server.bind((host, port))
        self.server.listen(5)
        self.topics = {}
        self.subscribers = {}
        self.subscriber_views = {}
        self.lock = threading.Lock()

    def handle_client(self, client_socket):
        # Handle incoming client messages
        try:
            while True:
                message = client_socket.recv(1024).decode('utf-8').strip()
                if not message:
                    break
                data = json.loads(message)
                command = data.get('command')
                topic = data.get('topic')
                msg = data.get('message')
                sid = data.get('sid')

                # Handle different client commands
                if command == 'CREATE' and topic:
                    self.create_topic(topic)
                elif command == 'PUBLISH' and topic and msg:
                    self.publish(topic, msg)
                elif command == 'SUBSCRIBE' and topic and sid:
```

Finally by both clients:

Publisher =>

```
 GNU nano 6.2                                                    publisher.py
from client_api import ClientAPI

def main():
    # Create API instance and register as a publisher
    api = ClientAPI()
    pid = api.register_publisher()

    topics = ['News', 'Sports', 'Technology', 'Health', 'Entertainment']

    # Sample messages for each topic
    messages = {
        'News': [
            'Breaking: Major event happening!',
            'Local news update: City council meets today.',
            'Weather alert: Heavy rain expected tomorrow.'
        ],
        'Sports': [
            'Real Madrid wins the championship!',
            'Ronaldo breaks record for most goals in a season.',
            'Upcoming match: Team FCB vs Team RM this weekend.'
        ],
        'Technology': [
            'AI is changing the world!',
            'New smartphone release with innovative features.',
            'Cybersecurity: Best practices for users.'
        ],
        'Health': [
            'Healthy eating tips for better living.',
            'The importance of regular exercise.',
            'Mental health awareness: Break the stigma.'
        ],
        'Entertainment': [
            'Upcoming blockbuster movie hits theaters.',
            'New album release: Artist X takes the music world by storm.'
```

Subscriber =>

```
  GNU nano 6.2                                                  subscri
from client_api import ClientAPI

def main():
    # Create API instance and register as a subscriber
    api = ClientAPI()
    sid = api.register_subscriber()

    topics = ['News']

    # Subscribe to specified topics
    for topic in topics:
        api.subscribe(sid, topic)
        print(f"Subscribed to topic: {topic}")

    # Pull and display messages from subscribed topics
    for topic in topics:
        print(f"Pulling messages from topic: {topic}")
        messages = api.pull_messages(sid, topic)
        if messages:
            print(f"Messages from {topic}:")
            for msg in messages:
                print(f"  - {msg}")
        else:
            print(f"No new messages for topic '{topic}'.")

if __name__ == "__main__":
    main()
```

Comments has been added to the above codes for readability and understanding.

Note:

For Running this Program, you need to run all three MessageBroker, Publisher and Subscriber in three different terminals using –

Command:

1. python3 Messagebroker.py
2. python3 publisher.py
3. python3 subscriber.py

I have not included any deployment scripts or Makefile to automate as it is easier to run these files.

Output:

MessageBroker.py:

```
tanmay@VM1:~/AOS_HW1$ python3 MessageBroker.py
Message Broker started...
Accepted connection from ('127.0.0.1', 36094)
Topic 'News' created.
Topic 'Sports' created.
Topic 'Technology' created.
Topic 'Health' created.
Topic 'Entertainment' created.
Invalid command: {"command": "DELETE", "topic": "Health"}
Invalid command: {"command": "DELETE", "topic": "Entertainment"}
Message published to topic 'News': Breaking: Major event happening!
Message published to topic 'News': Local news update: City council meets today.
Message published to topic 'News': Weather alert: Heavy rain expected tomorrow.
Message published to topic 'Sports': Real Madrid wins the championship!
Message published to topic 'Sports': Ronaldo breaks record for most goals in a season.
Message published to topic 'Sports': Upcoming match: Team FCB vs Team RM this weekend.
Message published to topic 'Technology': AI is changing the world!
Message published to topic 'Technology': New smartphone release with innovative features.
Message published to topic 'Technology': Cybersecurity: Best practices for users.
Accepted connection from ('127.0.0.1', 39066)
Client 61a938fb-842a-4129-b93f-f44879b18ccb subscribed to topic 'News'.
```

publisher.py:

```
tanmay@VM1:~/AOS_HW1$ python3 publisher.py
Publisher registered with ID: 337a4147-92e1-490d-93fb-19aebd444dbe
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe requested to create topic: News
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe requested to create topic: Sports
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe requested to create topic: Technology
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe requested to create topic: Health
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe requested to create topic: Entertainment
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe requested to delete topic: Health
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe requested to delete topic: Entertainment
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'News': Breaking: Major event happening!
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'News': Local news update: City council meets today.
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'News': Weather alert: Heavy rain expected tomorrow.
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'Sports': Real Madrid wins the championship!
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'Sports': Ronaldo breaks record for most goals in a season.
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'Sports': Upcoming match: Team FCB vs Team RM this weekend.
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'Technology': AI is changing the world!
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'Technology': New smartphone release with innovative features.
Publisher 337a4147-92e1-490d-93fb-19aebd444dbe sent message to topic 'Technology': Cybersecurity: Best practices for users.
Publisher operations completed successfully.
tanmay@VM1:~/AOS_HW1$
```

subscriber.py:

```
tanmay@VM1:~/AOS_HW1$ python3 subscriber.py
Subscriber registered with ID: 61a938fb-842a-4129-b93f-f44879b18ccb
Subscriber 61a938fb-842a-4129-b93f-f44879b18ccb subscribed to topic: News
Subscribed to topic: News
Pulling messages from topic: News
Subscriber 61a938fb-842a-4129-b93f-f44879b18ccb pulling messages from topic: News
Messages from News:
  - Breaking: Major event happening!
  - Local news update: City council meets today.
  - Weather alert: Heavy rain expected tomorrow.
tanmay@VM1:~/AOS_HW1$
```

Now, you can see my whole program is working perfectly with accurate results in the **Linux Environment**.

Next, I have done testing/benchmarking using maximum 50 clients:

Testing is done in **benchmark.py**:

Comments have been added for understanding.
For this as well, I haven't created any deployment scripts as you just need to run **MessageBroker.py** and **benchmark.py** in two different terminals.

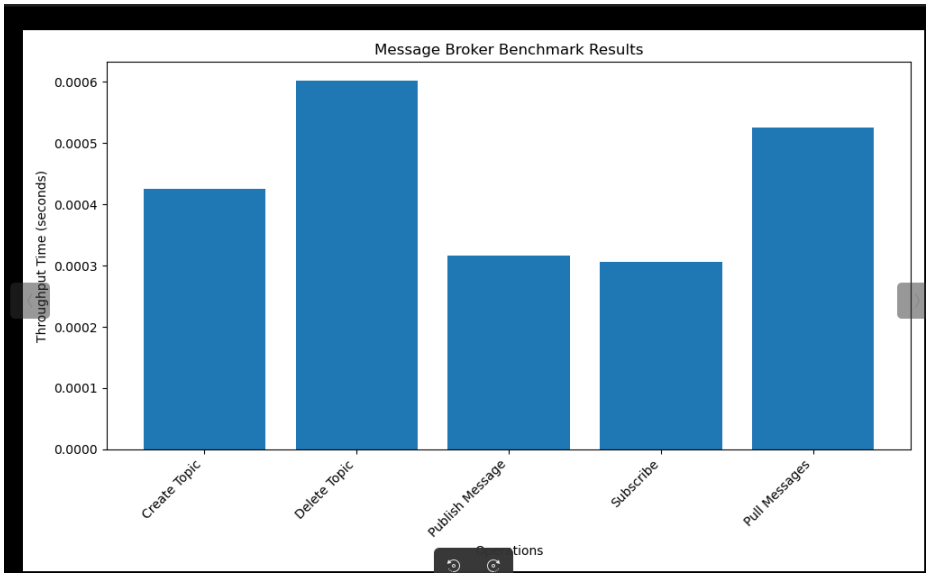
Throughput time is calculated at last and plotted using Matplotlib:

```
  GNU nano 6.2                                                benchmark.py
import time
import random
import uuid
import matplotlib.pyplot as plt
from client_api import ClientAPI
from tabulate import tabulate

class Benchmark:
    def __init__(self, host='localhost', port=8080):
        self.host = host
        self.port = port
        self.api = ClientAPI(host, port)
        self.results = {}

    # Measure the time taken for a specific operation across multiple clients
    def benchmark_operation(self, operation, num_clients):
        times = []
        for _ in range(num_clients):
            try:
                start_time = time.time()
                operation()
                times.append(time.time() - start_time)
            except Exception as e:
                print(f"Error during operation: {e}")
        return sum(times) / len(times) if times else 0

    # Register publisher and create a unique topic
    def create_topic_operation(self):
        pid = self.api.register_publisher()
        topic = f'topic{uuid.uuid4()}'
        self.api.create_topic(pid, topic)

    # Create and then delete a topic
```

Output:

```
Benchmark Results:
+------------------+--------------------+
| Operation        | Throughput Time    |
+==================+====================+
| Create Topic     | 0.000426 seconds   |
+------------------+--------------------+
| Delete Topic     | 0.000602 seconds   |
+------------------+--------------------+
| Publish Message  | 0.000316 seconds   |
+------------------+--------------------+
| Subscribe        | 0.000306 seconds   |
+------------------+--------------------+
| Pull Messages    | 0.000526 seconds   |
+------------------+--------------------+
```

Graph has been plotted as mentioned in the assignment:



DISCUSSIONS:

1. Asked if API's are sufficient?
=>

The current APIs (CREATE, PUBLISH, SUBSCRIBE) provide basic functionality for a publish-subscribe system. However, they could be enhanced by adding:

DELETE: To remove topics

UNSUBSCRIBE: To allow clients to stop receiving messages from a topic

LIST_TOPICS: To view available topics

GET_SUBSCRIBERS: To see how many subscribers a topic has

These additions would make the system more flexible and manageable.

2. **Bottlenecks?**

=>

Potential bottlenecks in the current implementation include:

Single-threaded message distribution: The publish method iterates through subscribers sequentially, which could slow down with many subscribers.

In-memory storage: Storing all messages and subscriber information in memory limits scalability.

Lack of message persistence: Messages are lost if the broker restarts.

3. Design Considerations

=>

Threading: While the broker uses threading for client handling, message distribution could be parallelized for better performance.

Error handling: The current implementation lacks robust error handling and logging.

Scalability: The system might struggle with a large number of topics or subscribers. Consider implementing a more scalable data structure or database backend.

Message Queuing: Implementing a message queue for each topic could improve message handling and allow for features like message persistence and retry mechanisms.

4. **Future Improvements**

=>

Implement load balancing for better distribution of topics and subscribers across multiple broker instances.

Add message persistence to prevent data loss during server restarts.

Implement a more robust client authentication and authorization system.

Consider using a protocol like MQTT for standardized message formatting and improved interoperability.