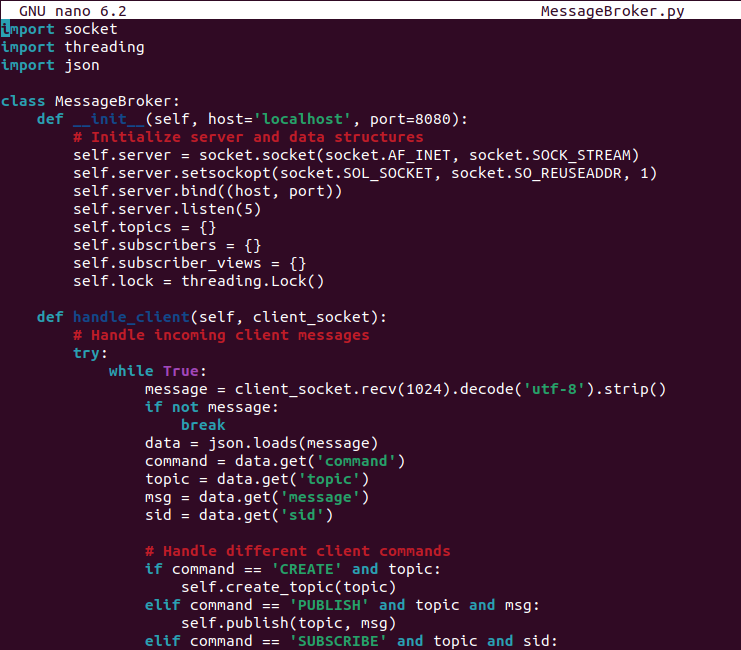
**AOS -HW1 Report – A20541164**

**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.

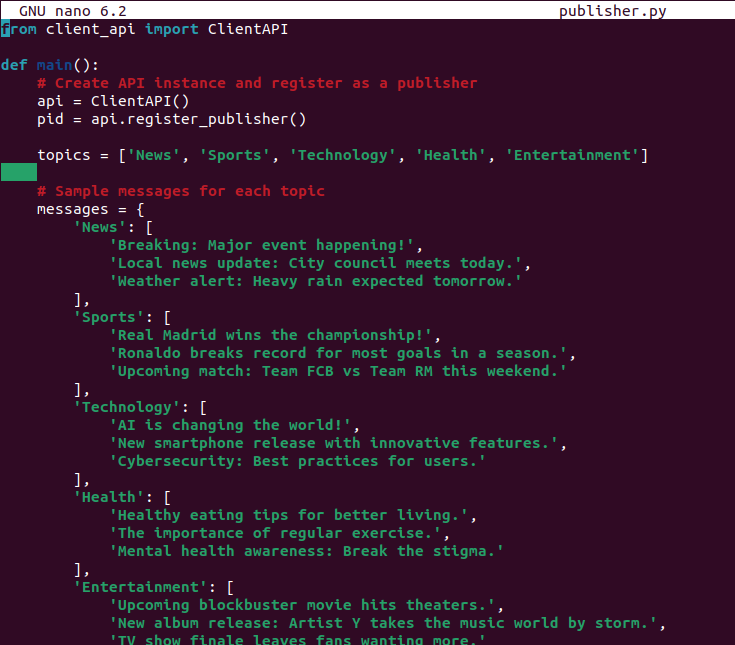
Overall, this assignment aims to establish a robust framework for message-driven communication, emphasizing both scalability and ease of use.

* **Firstly, I have written the code for ClientAPI as client\_api.py.**

**Then, MessageBroker.py (Server) Program:**

**Finally by both clients:**

**Publisher =>**

****

**A screenshot of a computer program

Description automatically generatedSubscriber =>**

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

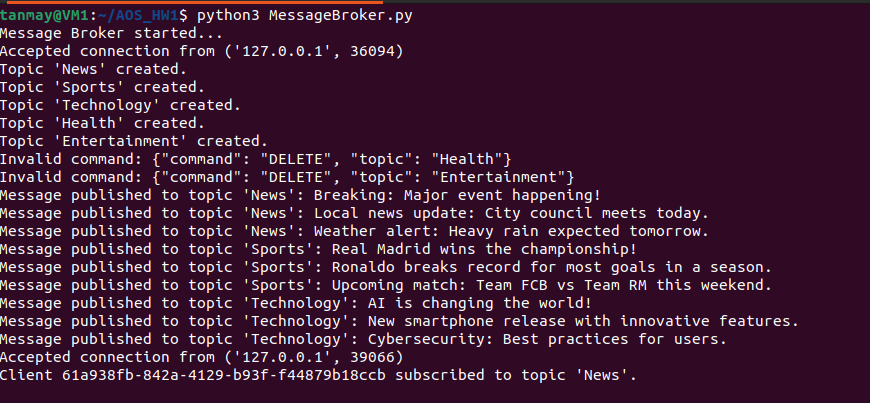
**Note:**For Running this Program, you need to run all three MessageBroker, Publisher and Subscriber in three different terminals using –

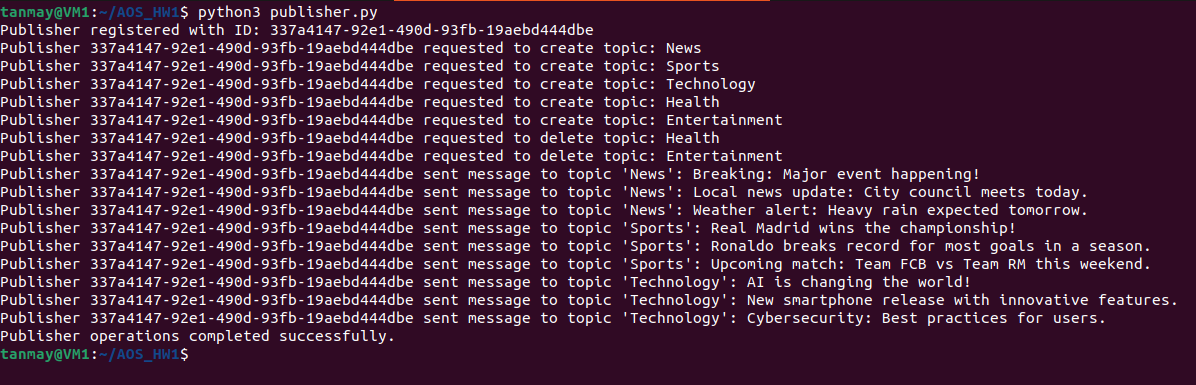
**Command:**

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

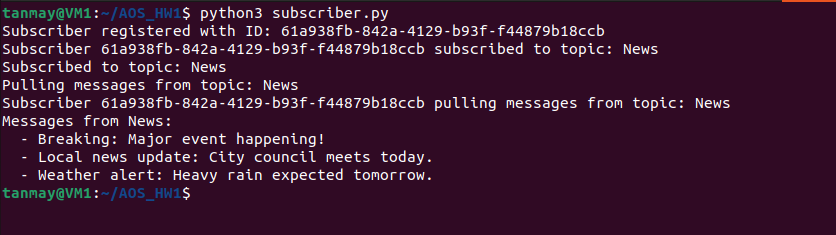
I have not included any deployment scripts or Makefile to automate as it is easier to run these files.

**Output:**

**MessageBroker.py:**

**publisher.py:**

**subscriber.py:**

****

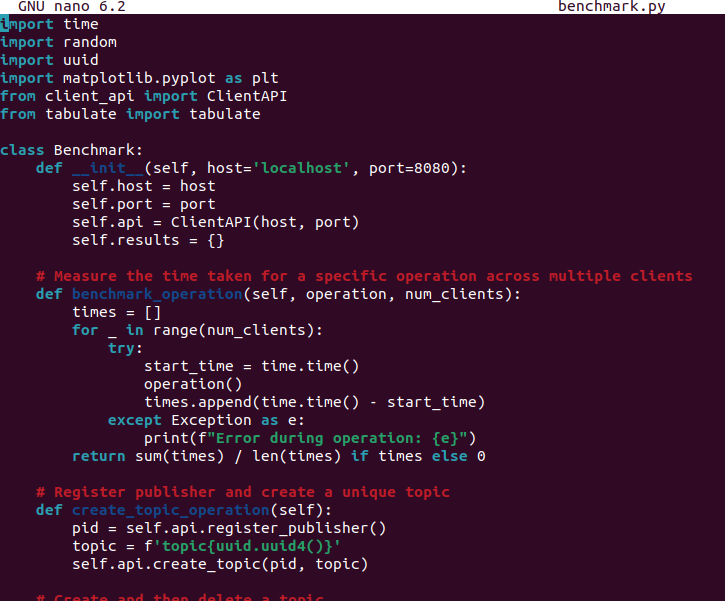
Now, you can see my whole program is working perfectly with accurate results in the **Linux Environment**.

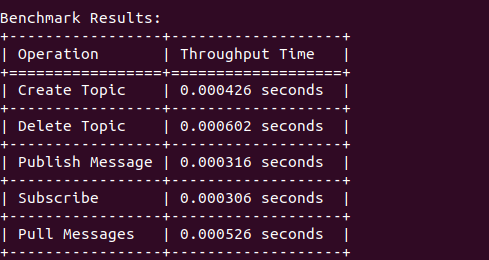
Next, I have done testing/benchmarking using maximum 50 clients:

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

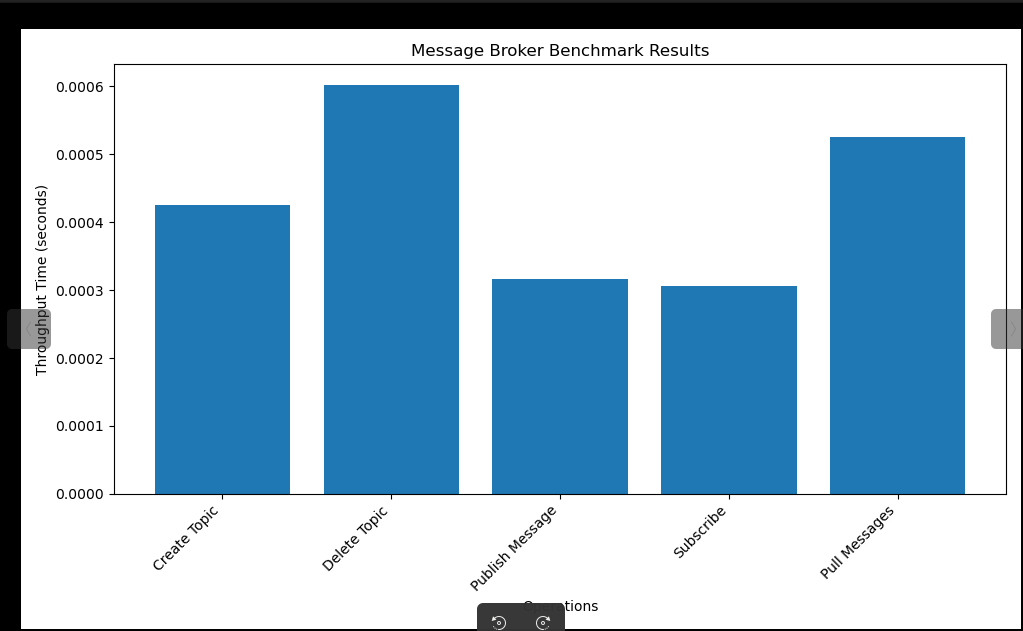
Comments have been added for understanding.  
For this as well, I haven’t created any deployment scripts as you just need to run **MessageBroker.py** and **benchmark.py** in two different terminals.

Throughput time is calculated at last and plotted using Matplotlib:



**Output:**

**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.

1. **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.