Multi-Threaded Priority Message Queue Implementation

- Overview:

  The implemented solution is a multi-threaded priority message queue system that allows multiple threads to send messages to each other with varying priorities. The system includes a priority message queue, a thread pool, and a message sender component. Threads can enqueue, dequeue, and peek at messages in the priority message queue. The thread pool facilitates the concurrent execution of simple actions associated with receiving and processing messages. The message sender allows threads to send messages to each other with specified priorities.

- Data Structures and Algorithms Used:

  1. **Priority Message Queue:**
     - Data Structure: Implemented using Python's **queue.PriorityQueue**.
     - Operations:
       - **enqueue_message(message, priority)**: Adds a message to the queue with a specified priority.
       - **dequeue_message()**: Removes and returns the highest priority message from the queue.
       - **peek_message()**: Returns the highest priority message without removing it from the queue.
       - **is_empty()**: Checks if the queue is empty.
  2. **Thread Pool:**
     - Data Structure: Utilizes Python's **concurrent.futures.ThreadPoolExecutor**.
     - Operations:
       - **submit_task(task, *args)**: Submits a task to the thread pool for execution.
       - **notify_all_waiting_threads()**: Notifies all waiting threads when a task is completed.
       - **sleep_thread(thread_id)**: Puts a thread to sleep until it is notified.
       - **notify_thread(thread_id)**: Notifies a specific thread to wake up.
       - **shutdown():** Helps in orderly shutdown of the threads in the pool.

3. **Message Sender:**
    - Operations:
        - **send_message(message, priority, target_thread)**: Enqueues a message and notifies the target thread if it is waiting.

- ## Instructions for Building and Running the Program:

1. **Requirements:**
    - Python 3.10 or upgraded version installed on system.
2. **Setup:**
    - Save the provided source code files (**priority_message_queue.py**, **thread_pool.py**, **message_sender.py**, and **main.py**) into the same directory.
3. **Running the Program:**
    - Open a terminal or command prompt in the directory containing the source code files.
    - Run the command: **python main.py**

- ## Explanation of Test Cases and Expected Outcomes:

The provided test case (**main.py**) creates three threads, each sending five messages to other threads with random priorities. The messages are sorted by priority before sending. The expected outcomes include the display of messages being sent between threads, demonstrating the functionality of the priority message queue, thread pool, and message sender components.

outcomes:

- Successful execution of threads, with messages being sent and processed concurrently.
- Appropriate display of messages with priorities and target threads.
- This test case creates random message using random library.
- Proper shutdown of the thread pool after all threads finish execution.

- I got below output after running **main.py** file.

```
Thread 0 sending message: gycLkfOsuEtDTHxsMTfB with priority 1 to Thread 1

Thread 0 sending message: wpkjhWisjoMNoXOjTScZ with priority 2 to Thread 0

Thread 0 sending message: OOOPAsPsdmjyFoCnMzYF with priority 4 to Thread 0

Thread 0 sending message: NDtWURusnzAFBYJZBlxm with priority 4 to Thread 0

Thread 0 sending message: iSfhXBxGOwQpjtOrXjly with priority 4 to Thread 1

Thread 1 sending message: fobNCnfPlYJbTkqGFZkS with priority 1 to Thread 2

Thread 1 sending message: lCGcfvKowfVFdtsIDads with priority 2 to Thread 2

Thread 1 sending message: yXtcETNaliaOpmesZKMA with priority 3 to Thread 0

Thread 1 sending message: VTcEiHeirXkttvfapifO with priority 4 to Thread 2

Thread 1 sending message: DPEAYHYlsxlggPvKDdhJ with priority 5 to Thread 1

Thread 2 sending message: ymfZpRMEraAecmcmWgkd with priority 1 to Thread 0

Thread 2 sending message: IzEThIbZXNsDUQWKsZWJ with priority 2 to Thread 1

Thread 2 sending message: KpgFMGJWtiQKzkDMXhQj with priority 3 to Thread 0

Thread 2 sending message: DkOCyGkYcmgZWEBEqQLj with priority 4 to Thread 1

Thread 2 sending message: VUQrdQdOFNPyhgbpuLua with priority 5 to Thread 2
```

- Additional Notes and Insights:

1. **ThreadPoolExecutor for Concurrent Execution:**
    - The **concurrent.futures.ThreadPoolExecutor** provides a convenient way to manage and execute tasks concurrently using a pool of threads. It simplifies the implementation of the thread pool component and allows easy submission of tasks for parallel execution.
2. **Graceful Thread Shutdown:**
    - Ensuring a graceful shutdown of threads and the thread pool is essential. The **thread_pool.shutdown()** method handles the orderly termination of the thread pool, allowing ongoing tasks to complete before shutting down.
3. **Callback Mechanism for Task Completion:**
    - The use of a callback function (**future.add_done_callback**) allows the thread pool to notify waiting threads when a task is completed. This mechanism ensures that sleeping threads are awakened when there is a task to be processed.