1. **Are there any issues concerning the fair treatment of the tasks?**

In the current implementation, tasks are treated fairly based on their priority. High-priority tasks are processed before normal-priority tasks. This ensures that important tasks are addressed promptly, which is generally a fair approach.

1. **Do we need stack and queue both to solve the given problem?’**

Yes, using both a stack and a queue is necessary to maintain the priority order. The stack is used for high-priority tasks, ensuring that the highest priority task is processed first, while the queue is used for normal-priority tasks, ensuring that tasks are processed in the order they were created. If we do not use a system that ensure that priority queue are identified, we would still need to do long search to ensure that the system is performing using the requirement specified in the problem statement.

1. **How are such systems built usually in practice? What data structures they use?**

Task management systems in practice often use more complex data structures and databases to manage tasks efficiently. These systems may involve databases to store and retrieve tasks, advanced data structures for efficient task scheduling, and user interfaces for interaction. The use of stacks and queues as described here is a simplified example and may not be sufficient for complex real-world scenarios.

1. **Is there any possible best solution still using stacks and queues for fair treatment on task processing?**

The current solution using stacks and queues is a reasonable approach for maintaining fairness based on priority. However, in practice, you might encounter more complex scenarios where other data structures and algorithms, like a priority queue (heap), are better suited for ensuring fairness and efficient task processing.

1. **Can the application be built using an alternative data structure such as Deque alone instead of a stack and queue? If so, describe the changes you would make referring to the line numbers in your code.**

Yes, you can use a Deque to implement this system without separate stack and queue structures. A Deque allows you to add and remove elements from both ends, making it suitable for this scenario. Here's how I can modify the code:

I created two class Queue and Stack, I would have to create one class called Dequeue, I should make sure that all normal priority task are added using the tail and all high priority are inserted using the tail, I would then create an instance variable that trach the length of high priority start from the tail, and the length of normal priority task starting from the head.

The time of processing, I would check if the high priority is great than zero, then I would know that I still have a high priority task to process. For the normal