2nd Assessment Report Template

JINGFENG ZHANG 50091040

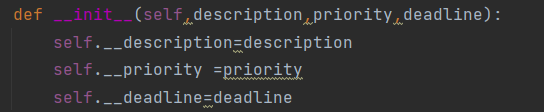
1. Introduction

In our daily lives, there are many tasks to accomplish, and these tasks need to be completed in order based on their deadlines and priorities. Task scheduling programs help us arrange the completion sequence of tasks more efficiently and time-savingly.

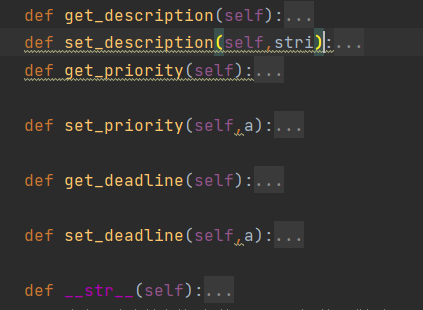
Based on the priority and deadline input by the user, tasks are arranged in order of priority first, followed by the deadline for completion. The task scheduler comprehensively organizes the sequence of task completion. Additionally, users can adjust a task as needed and refresh the task sequence. Once a task is completed, users can delete it from the task sequence. The programming was achieved by the Priority Queue data structure e and Object-Oriented Programming principles. But this program still has some limitations. For instance, tasks with lower priority but earlier deadlines might be scheduled later, resulting in insufficient time for completion.

1. Task Class Design

This class has three attributes: \_\_description,\_\_priority,\_\_deadline. They are used to describe the task's description, priority, and deadline.



In the class, it has many methods. The \_\_init\_\_method receives some attributes and stores them. The get\_description method was created for the users to get the task’s description. The method was implemented by return its \_\_description attribute. The get\_priority and the get\_deadline are as same as the get\_decription. The set\_description method was created for the users to reset the task’s description. The method was implemented by seting its \_\_description attribute with its parameter. The set\_priority and the set\_deadline are as same as the set\_decription. The \_\_str\_\_ method was overrided to print the task’s information obviously.



This class achieves encapsulation by encapsulating its attributes, allowing external access and modification only through functions, thus achieving encapsulation. External operations on these attributes can only be performed through methods.

1. PriorityQueue Class Design

The Priority Queue data structure was chosen because it allows for tasks to be ordered based on their priority levels. Priority queues are data structures that store elements with associated priorities and allow for the removal of elements with the highest priority. Tasks with higher priority levels will be executed before tasks with lower priority levels. The priority queue works by maintaining the highest priority task at the front of the queue, allowing for efficient retrieval and removal of tasks based on their priority.

The attributes of the PriorityQueue class include a list of tasks stored in the priority queue.



The methods implemented for adding, removing, and peeking at tasks in the PriorityQueue class serve different purposes. The add\_task method adds a new task to the priority queue and sorts the tasks based on their priority levels and a key methods. The remove\_task method removes and returns the highest priority task from the queue. The peek\_task method returns the highest priority task without removing it from the queue. These methods allow for efficient management and execution of tasks based on their priorities.



The priority queue sorts tasks using a lambda function that compares tasks based on their priority levels first and then their deadlines. Sorting by deadline as a secondary criterion ensures that tasks with the same priority level are executed based on their deadlines, prioritizing tasks with earlier deadlines. This ensures that tasks are executed in the most efficient and timely manner.



1. Scheduler Class Design

The Scheduler class organizes the PriorityQueue class to manage tasks by adding tasks to the priority queue based on their priority and deadline. It ensures that tasks are sorted in descending order of priority and ascending order of deadline, allowing the Scheduler class to prioritize tasks accordingly.

The attributes of the Scheduler class include a PriorityQueue object named \_\_queue, which stores the tasks added to the scheduler. The \_\_queue attribute allows the Scheduler class to access the priority queue and manipulate tasks based on their priority and deadline.



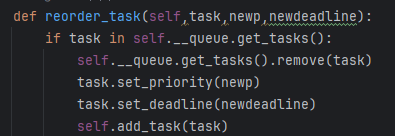
add\_task(t): This method adds a task to the priority queue based on its priority and deadline by using the method in PriorityQueue Class.



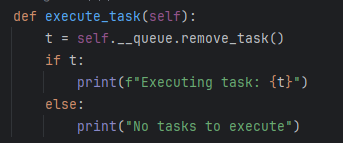
remove\_task(): This method removes and returns the highest priority task from the priority queue by using the method in PriorityQueue Class.



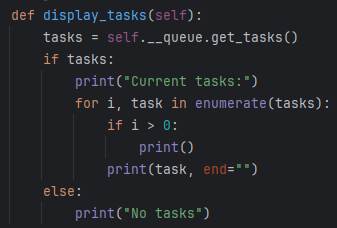
reorder\_task(task, newp, newdeadline): This method allows for reordering a task by changing its priority and deadline, and then adding it back to the priority queue by using the methods in PriorityQueue Class and the methods in Task Class.



execute\_task(): This method executes the highest priority task in the priority queue by removing it from the queue and displaying a message. It uses the method in PriorityQueue Class to get the task.



display\_tasks(): This method displays all tasks currently in the priority queue by the loop.



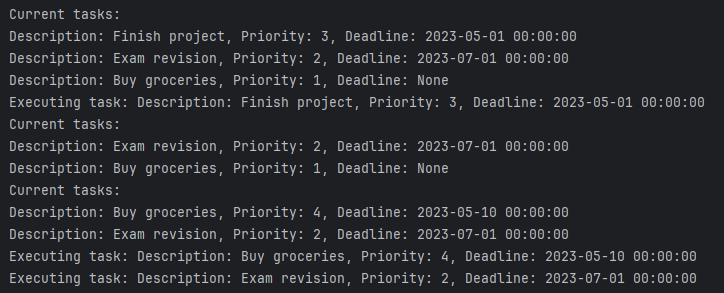
Design choices in the Scheduler class include the use of a priority queue to efficiently manage tasks based on priority and deadline. The implementation of methods like add\_task, remove\_task, reorder\_task, execute\_task, and display\_tasks allows for seamless task management within the Scheduler class. These methods contribute to the overall functionality of the Task Scheduler by providing a systematic way to add, prioritize, execute, and display tasks.

1. Testing and Demonstration

The test program is designed to showcase the functionality of the Task Scheduler, allowing users to interact with it by adding tasks, executing tasks, and reordering tasks based on priorities and deadlines. Users can use the Task Scheduler's methods such as add\_task() to add tasks, execute\_task() to execute tasks according to their priorities, and reorder\_task() to change the priority and deadline of tasks.



2nd paragraph: Provide sample input/output to show the Task Scheduler's performance (8 points).



3rd paragraph: Discuss any challenges faced during testing and how they were resolved (4 points).

During testing, challenges were encountered in ensuring the correct execution order of tasks after reordering. This challenge was resolved by reviewing and debugging the reorder\_task() method to ensure that it correctly updates the priority and deadline of tasks without disrupting the execution order. Additionally, thorough testing with various task configurations helped identify and fix any inconsistencies in task execution. By addressing these challenges through testing and debugging, the Task Scheduler's functionality was validated and optimized for reliable task management.

1. Conclusion

1st paragraph: Summarise the project and its main features (2 points).

2nd paragraph: Reflect on the learning experience and the application of OOP concepts (2 points).

3rd paragraph: Suggest possible improvements or future enhancements (1 point).