QUEUE BASED TASK SCHEDULER

A PROJECT REPORT

Submitted by

V.HARINARAYANAN (2303811714821008)

in partial fulfillment of requirements for the award of the course

CGB1121 – DATA STRUCTURES

in

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

May, 2024

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled **“QUEUE BASED TASK SCHULER”** is the bonafide work of **V.HARINARAYANAN(2303811714821008),** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible]	[illegible]
SIGNATURE	**SIGNATURE**
Dr.T.AVUDAIAPPAN M.E.,Ph.D.,	Mr.R.ROSHAN JOSHUA.,M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
ASSOCIATE PROFESSOR	ASSISTANT PROFESSOR
Department of Artificial Intelligence	Department of Artificial Intelligence
K. Ramakrishnan College of Technology (Autonomous)	K. Ramakrishnan College of Technology (Autonomous)
Samayapuram–621112.	Samayapuram–621112.

Submitted for the viva-voce examination held on …………….

DECLARATION

I declare that the project report on **"QUEUE BASED TASK SCHEDULAR"** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of **BACHELOR OF ENGINEERING**. This project report is submitted on the partial fulfillment of the requirement of the award of the course **CGB1121- DATA STRUCTURES.**

[illegible]

Signature

V.HARINARAYANAN

STUDENT NAME

Place: Samayapuram

Date: 11.06.2024

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and indebtedness to our institution, "**K. Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I extend our sincere acknowledgment and appreciation to the esteemed and honorable Chairman, **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director, **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding our project and offering an adequate duration to complete it.

I would like to thank **Dr. N. VASUDEVAN, M.TECH., Ph.D.,**

Principal, who gave the opportunity to frame the project to full satisfaction.

I thank **Dr.T.AVUDAIAPPAN M.E., Ph.D.,** Head of the Department of **ARTIFICIAL INTELLIGENCE**, for providing her encouragement in pursuing this project.

I wish to convey our profound and heartfelt gratitude to our esteemed project guide **Mr.R.ROSHAN JOSHUA., M.E.,** Department of **ARTIFICIAL INTELLIGENCE,** for his incalculable suggestions, creativity, assistance and patience, which motivated us to carry out this project.

I render our sincere thanks to the Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education.

MISSION OF THE INSTITUTION

Produce smart technocrats with empirical knowledge who can surmount the global challenges.

Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students.

Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations.

VISION OF DEPARTMENT

To become a renowned hub for AIML technologies to producing highly talented globally recognizable technocrats to meet industrial needs and societal expectation.

MISSION OF DEPARTMENT

Mission 1: To impart advanced education in AI and Machine Learning, built upon a foundation in Computer Science and Engineering.

Mission 2: To foster Experiential learning equips students with engineering skills to tackle real-world problems.

Mission 3: To promote collaborative innovation in AI, machine learning, and related research and
development with industries.

Mission 4: To provide an enjoyable environment for pursuing excellence while upholding strong personal and professional values and ethics.

PROGRAM EDUCATIONAL OBJECTIVES

Graduates will be able to:

1. **PEO1:** Excel in technical abilities to build intelligent systems in the fields of AI & ML in order to find new opportunities
2. **PEO2:** Embrace new technology to solve real-world problems, whether alone or as a team, while prioritizing ethics and societal benefits.
3. **PEO3:** Accept lifelong learning to expand future opportunities in research and product development.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice

Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The Efficient task scheduling is critical for optimizing resource utilization and improving system performance in various computing environments. Queue-based task schedulers offer a flexible and scalable approach to managing tasks by organizing them in a structured queue based on priority or other criteria. This abstract presents an overview of a queue-based task scheduler, highlighting its key components, operational principles, and benefits.The core of the queue-based task scheduler is a data structure that maintains a queue of tasks awaiting execution. Tasks are enqueued based on their arrival time, priority, or other specified criteria. The scheduler continuously monitors the queue and selects tasks for execution based on predefined scheduling policies, such as first-in-first-out (FIFO), priority-based scheduling, or round-robin scheduling.One of the primary advantages of a queue-based task scheduler is its simplicity and scalability. The queue structure allows for efficient insertion and removal of tasks, making it suitable for managing large numbers of concurrent tasks in real-time systems. Additionally, by decoupling task submission from execution, the scheduler can adapt to changing workload conditions and dynamically allocate resources to meet performance objectives.Furthermore, queue-based task schedulers facilitate task prioritization and resource allocation, ensuring that critical tasks receive.

By maintaining a transparent queue of pending tasks, the scheduler provides visibility into the scheduling decisions, enbling better system monitoring and performance analysis.In conclusion, queue-based task schedulers offer a versatile solution for managing task execution in diverse computing environment

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

1. ADT - Abstract Data Type
2. API - Application Programming Interface
3. CPU - Central Processing Unit
4. DLL - Doubly Linked List
5. FIFO - First In, First Out
6. I/O - Input/Output
7. LIFO - Last In, First Out
8. OOP - Object-Oriented Programming
9. RAM - Random Access Memory
10. TDA - Task Dependency Analysis

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO PROJECT

Introducing the queue-based task scheduler, a cornerstone of efficient task managementv in computing. By structuring tasks in a queue, it prioritizes and allocates vresources effectively. Utilizing scheduling policies such as FIFO or priority-based methods, it ensures timely task execution. Overall, it optimizes system performance by managing task execution seamlessly.

1.2 PURPOSE AND IMPORTANCE OF THE PROJECT

The project endeavors to create a queue-based task scheduler, pivotal for optimizing resource allocation and task execution in computing systems. Its core objective is to enhance system performance by efficiently managing tasks through prioritization and allocation strategies.

By structuring tasks in a queue and leveraging scheduling policies like FIFO or priority-based methods, it ensures timely execution of critical tasks while maximizing resource utilization. This optimization is vital for diverse computing applications, spanning real-time systems, parallel processing, and distributed computing platforms.

The project's significance transcends mere efficiency; it fosters innovation by enabling smoother operation of complex computing environments. Addressing the growing demand for efficient resource management, it contributes to advancements in technology and enables the development of novel applications.

As a foundational element, the project lays the groundwork for future advancements in computing infrastructure and applications, driving progress in various domains. Ultimately, it empowers computing systems to meet the evolving demands of modern-day computational challenges, thereby enhancing user experience and facilitating transformative technological breakthroughs.

1.3 OBJECTIVES

- Contact Management
- Efficient Data Handling
- User-Friendly Interface
- Scalability
- Robustness

1.4 PROJECT SUMMARIZATION

The Task Management Queue System is a C-based software application designed to manage tasks efficiently using a circular queue data structure. It aims to offer a robust, scalable, and user-friendly solution for organizing tasks, ensuring optimal performance and resource utilization. Key features include contact management for creating and managing tasks with specific attributes, efficient data handling through circular queue operations, and a user-friendly interface that provides clear feedback and task display. The system dynamically handles varying numbers of tasks, supported by strong error handling for stable and reliable operations. The implementation involves defining a `Task` struct with name and priority, a `Queue` struct to represent the circular queue, and essential functions such as `createQueue` for initialization, `enqueue` for adding tasks, `dequeue` for removing tasks, and `displayQueue` for showing all tasks in the queue. The expected outcome is a functional task management system that demonstrates the efficient use of a circular queue, enhances understanding of queue data structures, and provides a foundation for further enhancements and integrations. This project serves as an educational tool for learning about data structures in C and offers a practical solution for managing tasks in an organized manner.

CHAPTER 2
PROJECT METHODOLOGY

2.1 INTRODUCTION TO SYSTEM ARCHITECTURE

System architecture is the blueprint for designing complex systems, encompassing the arrangement and interaction of components like hardware, software, and networks. It defines connections, hierarchies, and patterns to optimize performance, scalability, and reliability. Adherence to standards ensures compliance and interoperability. Ultimately, system architecture facilitates the creation of robust, scalable, and maintainable systems aligned with organizational objectives.

2.2 Agile Development:

- Iteratively design and refine the system architecture, incorporating feedback from each sprint cycle to ensure alignment with evolving requirements and priorities. Use architecture diagrams to visualize and communicate architectural changes to the team.

2.3 Waterfall Model: Develop a comprehensive architecture diagram during the early stages of the project, detailing all system components, their interactions, and interfaces. Follow the sequential phases of the waterfall model to progressively refine and finalize the architecture.

2.4 Prototype Development: Create architectural prototypes to validate design decisions and gather feedback from stakeholders. Use simplified architecture diagrams to illustrate key architectural concepts and validate their feasibility.

2.5 Scrum Framework: Maintain a living architecture document that evolves with each sprint, capturing architectural decisions, changes, and dependencies. Use architecture diagrams as part of sprint planning and review meetings to align development efforts with the overall system architecture.

2.6 Kanban Method: Visualize the system architecture using Kanban boards or similar tools, highlighting different architectural components and their statuses. Use architecture diagrams to identify bottlenecks, optimize workflows, and prioritize architectural tasks.

2.7 Rapid Application Development (RAD): Develop an initial architecture diagram quickly to provide a high-level overview of the system structure. Iterate on the architecture in parallel with rapid prototyping, using feedback from prototypes to refine and enhance the architecture.

2.8 Lean Development: Focus on creating a lightweight, adaptable architecture that addresses the core requirements with minimal complexity. Use architecture diagrams to identify areas of potential waste or redundancy and streamline the architecture accordingly.

2.9 Incremental Development: Develop the architecture incrementally, adding new components and features iteratively while maintaining overall architectural integrity. Use incremental architecture diagrams to visualize the evolution of the system architecture over time.

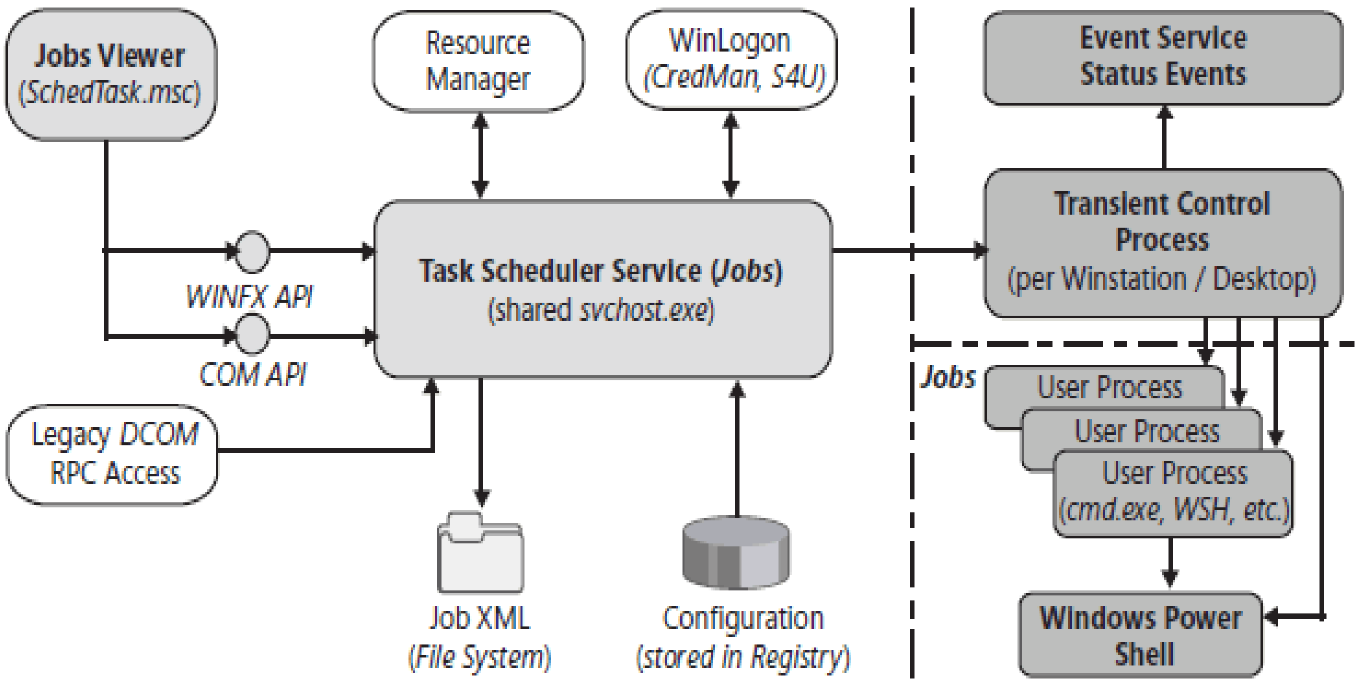
2.10 Joint Application Development (JAD): Collaborate with stakeholders and end-users to develop the system architecture collaboratively. Use interactive workshops and brainstorming sessions to create architecture diagrams that reflect the collective understanding and consensus of the project's architectural goals.

2.11 Feature-Driven Development (FDD): Break down the system architecture into feature modules, each with its own architecture diagram detailing its structure and interactions. Use feature-based architecture diagrams to guide development efforts and ensure alignment with feature priorities.

.

2.12 Architecture Diagram Integration:

Include a diagram that visually represents the system architecture. The diagram should depict how each component interacts with the others. For example, it can show the User Interface sending requests to the Application Logic, which in turn interacts with the Data Management Layer and the Storage Layer.



Jobs Viewer (SchedTask.msc)
Resource Manager
WinLogon (CredMan, S4U)
Event Service Status Events
WINFX API
COM API
Task Scheduler Service (Jobs) (shared svchost.exe)
Transient Control Process (per Winstation / Desktop)
Legacy DCOM RPC Access
Jobs
User Process
User Process
User Process (cmd.exe, WSH, etc.)
Job XML (File System)
Configuration (stored in Registry)
Windows Power Shell

Fig 2.12 : Architecture Diagram (Sample)

CHAPTER 3
DATA STRUCTURE PREFERANCE

3.1 EXPLANATION OF WHY A DOUBLY LINKED LIST WAS CHOSEN

A queue was chosen in a data structure for several reasons, primarily revolving around its unique properties and behavior which make it ideal for certain applications.

3.1.1 FIFO (First In, First Out) Order:

- A queue follows the FIFO principle, meaning the first element added to the queue will be the first one to be removed. This behavior is essential for tasks that require ordered processing.
- Example: In a printer queue, the first document sent to the printer should be the first one to be printed.

3.1.2 Fairness and Order:

- Queues ensure that elements are processed in the exact order they were added, which is crucial for fairness in many scenarios.
- Example: Customer service systems use queues to handle calls or tickets in the order they were received, ensuring fair and predictable service.

3.1.3 Resource Management:

- Queues are used in managing resources that need to be distributed among multiple consumers. By maintaining the order of requests, queues help in systematically allocating resources.
- Example: In a time-sharing system, CPU time slots are managed using a queue, ensuring each process gets a fair share of CPU time.

3.1.4 Algorithmic Uses:

- Certain algorithms inherently require the use of queues for their implementation.
- Example: Breadth-first search (BFS) in graph traversal uses a queue to explore nodes level by level.

3.1.5 Concurrency Control:

- In multi-threaded environments, queues are used to handle tasks or data produced by one thread and consumed by another, ensuring safe communication between threads.

3.2 COMPARISON WITH OTHER DATA STRUCTURES:

Queue compared to other data structures lies in its ability to handle tasks and data in a strictly ordered, fair, and efficient manner through its FIFO (First In, First Out) properties.

3.3 ADVANTAGES OF QUEUE IN DATA STRUCTURE:

3.3.1 Simplicity and Efficiency

- **Operations**: Typically, enqueue (insertion) and dequeue (removal) operations have O(1) time complexity, making them efficient for adding and removing elements.
- **Predictable Performance**: The straightforward nature of queues ensures consistent and predictable performance for sequential tasks.

3.3.2 Algorithmic Applications:

- **Specific Algorithms**: Some algorithms, such as breadth-first search (BFS) in graph traversal, require a queue for their correct implementation, benefiting from its FIFO order.
- **Task Scheduling**: Useful in various scheduling algorithms, ensuring tasks are handled in the order they arrive.

3.4 Disadvantages of Queue in data structure:

3.4.1 Limited Access

- **Restricted Operations**: Queues only allow access to the front and back elements. This restriction can be limiting if you need to access or modify elements in the middle of the queue.
- **Lack of Flexibility**: Unlike other data structures like linked lists or arrays, you cannot easily perform arbitrary insertions, deletions, or access elements at specific positions.

3.4.2 Fixed Size Limitation:

- **Fixed Capacity**: When using a fixed-size array for the queue implementation, the queue has a fixed capacity, which can either lead to wasted memory (if the queue is mostly empty) or overflow errors (if the queue becomes full and cannot grow dynamically).

CHAPTER -4
DATA STRUCTURE METHODOLOGY

Data structure methodology refers to the systematic approach and principles followed in designing, implementing, and utilizing data structures. It involves understanding the properties, operations, and behaviors of various data structures and selecting the most appropriate one for a given problem or application. Here's a breakdown of data structure methodology.

4.1 **Data Structure Selection Criteria**:

- Analyzing project requirements to determine data storage and retrieval needs.
- Considering factors such as data complexity, access patterns, and performance requirements.
- Evaluating trade-offs between different data structures (e.g., arrays, linked lists, trees) based on efficiency and suitability for the project's use cases.

4.2 **Data Structure Design**:

- Defining the structure and organization of selected data structures to represent project data.
- Determining data representations, including fields, records, and relationships.
- Designing algorithms for data manipulation and retrieval operations, ensuring efficiency and correctness.

4.3 **IMPLEMENTATION GUILDNESS**:

- Translating data structure designs into executable code, adhering to programming language standards and best practices.
- Addressing considerations such as memory management, error handling, and code modularity.
- Documenting implementation details, including data structure APIs, usage examples, and internal workings.

104.4 TESTING AND VALIDATION:

- Developing comprehensive test suites to verify the functionality and correctness of implemented data structures.
- Conducting unit tests, integration tests, and system tests to validate data structure behavior under various scenarios.
- Identifying and addressing edge cases and boundary conditions to ensure robustness and reliability.

4.5 OPTIMAZATION STRATIGIES:

- Profiling data structure performance to identify bottlenecks and areas for improvement.
- Implementing optimization techniques such as caching, indexing, and algorithmic enhancements to enhance efficiency.
- Balancing optimization efforts with maintainability and code readability considerations.

4.6 INTEGRATION WITH PROJECT COMPONENTS:

- Integrating implemented data structures seamlessly with other project modules and components.
- Ensuring compatibility and interoperability between data structures and external interfaces.
- Testing data structure interactions within the broader project context to verify functionality and correctness.

4.7 NODE STRUCTURE:

In the context of a circular doubly linked list used for managing a phone directory, the node structure plays a crucial role in organizing and storing information about each number and name within the list. The node structure contains the necessary components to represent individual songs or tracks in the playlist and maintains the connections between nodes in the list.

4.8. INITIALIZATION, INSERTION & DELETION

4.8.1 INITIALIZATION:

- **Array-based Implementation**:
    - Allocate memory for the array to hold the queue elements.
    - Initialize variables like front and rear to -1 to indicate an empty queue.
    - Optionally, set the maximum size of the queue.
- **Linked List-based Implementation**:
    - Create an empty linked list to represent the queue.
    - Initialize pointers like front and rear to NULL to indicate an empty queue.

4.8.2 INSERTION (ENQUEUE)

- **Array-based Implementation**:
    - Check if the queue is full.
    - Increment the rear pointer.
    - Insert the new element at the rear position.
- **Linked List-based Implementation**:
    - Create a new node with the given data.
    - If the queue is empty, set both `front` and `rear` pointers to the new node.
    - Otherwise, set the `next` pointer of the current `rear` node to the new node and update `rear` to the new node.

4. **Deletion (Dequeue)**

- **Array-based Implementation**:
    - Check if the queue is empty.
    - Retrieve the element at the `front` position.
    - Increment the `front` pointer.

CHAPTER-5

MODULES

5.1 Function Name: ArrayQueue Class():

Description: The ArrayQueue class is a fundamental component of the Queue-Based Task Scheduler, designed to manage tasks using an array-based queue. This implementation ensures efficient task management by maintaining a fixed-size circular buffer. The ArrayQueue class handles common queue operations such as initialization, insertion (enqueue), deletion (dequeue), and inspection (peek) with a focus on FIFO (First-In-First-Out) order.

5. Function name: LinkedListQueue Class():

Description: The LinkedListQueue class is another fundamental component of the Queue-Based Task Scheduler. This implementation uses a linked list to manage tasks, allowing for dynamic memory usage and flexibility in the number of tasks it can handle. The LinkedListQueue class supports common queue operations such as initialization, insertion (enqueue), deletion (dequeue), and inspection (peek) while maintaining FIFO (First-In-First-Out) order.

6. Function name: Taskscheduler classss():

Description: The TaskScheduler class is the core component of the Queue-Based Task Scheduler project. It leverages either an ArrayQueue or LinkedListQueue to manage tasks in a FIFO (First-In-First-Out) order. The TaskScheduler class provides an interface for adding tasks to the queue, processing tasks, inspecting the next task, and retrieving the number of pending tasks. This flexibility allows it to be used in various applications where orderly task processing is essential.

CHAPTER 6
CONCLUSION & FUTURE SCOPE

6.1 CONCLUSION

A queue-based task scheduler ensures fairness and predictability by processing tasks in a First-In-First-Out (FIFO) order. It is simple to implement and offers efficient task management with constant-time complexity for enqueue and dequeue operations. This makes it ideal for operating systems, web servers, and real-time systems where task order and timely execution are crucial. The inherent fairness and predictability of a FIFO queue facilitate equitable resource distribution. Enhancements like priority queues can handle critical tasks more effectively. Ensuring thread-safe operations and balancing resource management are essential for optimal performance. Overall, a queue-based task scheduler is a robust and efficient solution for various task management needs.The future of queue-based task schedulers is bright and multifaceted, promising substantial advancements across various dimensions. Integrating machine learning for dynamic task prioritization and adaptive scheduling will enable more intelligent and efficient resource allocation. Enhanced priority management mechanisms will ensure critical tasks receive the attention they need without sacrificing overall fairness.

6.2 FUTURE SCOPE

The future of queue-based task schedulers includes integrating machine learning for dynamic task prioritization and adaptive scheduling based on system load. Enhancing priority management mechanisms and expanding to distributed systems can improve efficiency and fault tolerance. Focus on energy efficiency and adaptation for IoT and edge computing will reduce power consumption and latency. Improved support for real-time systems will ensure precise deadline adherence. Security and reliability features will mitigate threats and enhance robustness. Hybrid scheduling models combining various approaches will offer versatile task management, while user-centric customization will allow fine-tuning based on specific needs. These advancements will keep queue-based schedulers effective in diverse and complex.

REAL TIME USES :

1.**Machine Learning Integration**: Enhancing dynamic task prioritization and adaptive scheduling using predictive models.

2.**Enhanced Priority Management**: Developing advanced mechanisms like multi-level queues to handle critical tasks more effectively

3. **Distributed Systems**: Expanding to manage tasks across distributed systems and cloud environments for better load balancing and fault tolerance.

4.**Energy Efficiency**: Designing energy-efficient schedulers for battery-powered and green computing environments.

5.IoT and Edge Computing: Adapting for low-latency, resource-constrained environments like IoT and edge computing.

6.Real-Time Systems Support: Ensuring precise deadline adherence in critical applications such as autonomous vehicles and industrial automation.

7.Security and Reliability: Implementing robust security features to mitigate threats and enhance reliabilable.

APPENDICES

APPENDIX A-SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

{

    char taskName[20];
    int priority;
} Task;

{
    Task* tasks;
    int size;
    int front;
    int rear;
    Queue;
}

Queue* createQueue(int size)

{
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->tasks = (Task*)malloc(size * sizeof(Task));
    queue->size = size;
    queue->front = 0;
    queue->rear = 0;
    return queue;

}
```

```
void enqueue(Queue* queue, Task task)

 {


   if (queue->rear == queue->size) {
      printf("Queue is full. Cannot add more tasks.\n");
      return;
   }


   queue->tasks[queue->rear] = task;
   queue->rear = (queue->rear + 1) % queue->size;
                                                      12

   Task dequeue(Queue* queue)
 {


   if (queue->front == queue->rear)


{


      printf("Queue is empty. Cannot remove tasks.\n");
      Task task;
      task.priority = -1;
      return task;


   }


   Task task = queue->tasks[queue->front];
   queue->front = (queue->front + 1) % queue->size;
   return task;
```

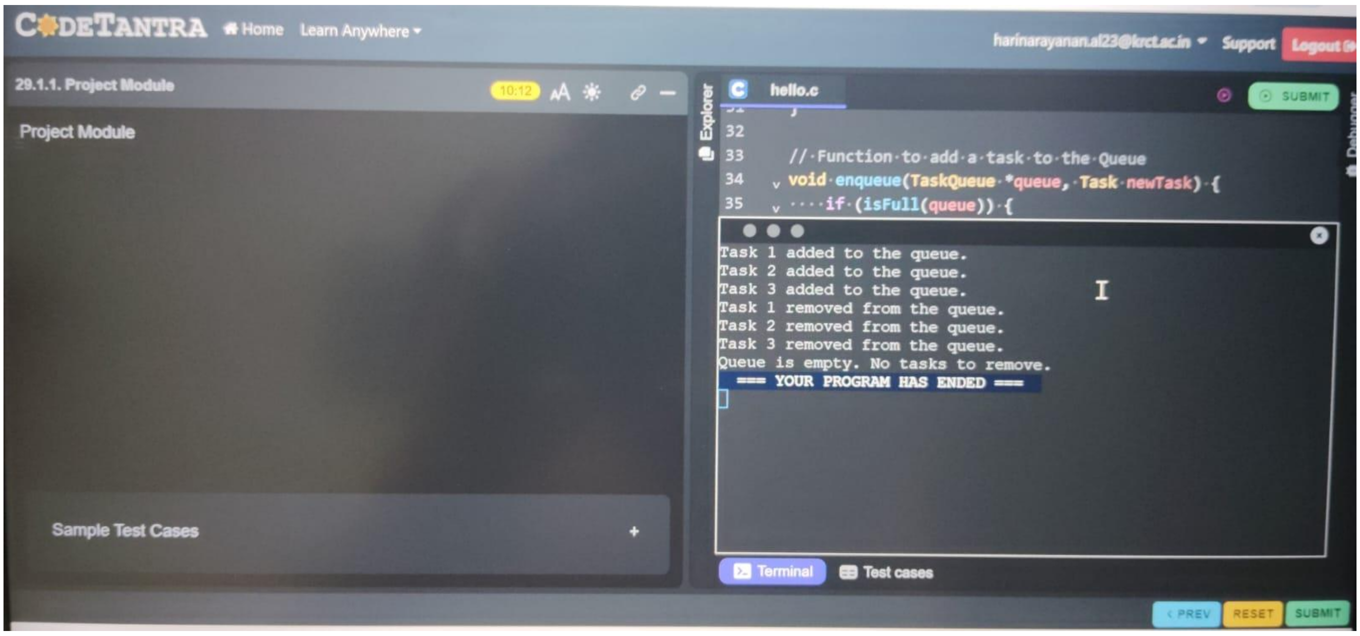
```
}

void displayQueue(Queue* queue)
{
   int i = queue->front;
   while (i != queue->rear) {
      printf(“Task:     %s,     Priority:     %d\n”,     queue->tasks[i].taskName,
queue>tasks[i].priority);
      i = (i + 1) % queue->size;

   }

}

int main()

{

   Queue* queue = createQueue(5);



   Task task1 = {“Task 1”, 3};
   Task task2 = {“Task 2”, 1};
   Task task3 = {“Task 3”, 2};
   Task task4 = {“Task 4”, 4};
   Task task5 = {“Task 5”, 5};
```

```
enqueue(queue, task1);
enqueue(queue,task2);
enqueue(queue, task3);
enqueue(queue, task4);
enqueue(queue, task5);

    printf(“Tasks in the queue:\n”);
    displayQueue(queue);

   Task removedTask;
    while (1)
{
        removedTask = dequeue(queue);
        if (removedTask.priority == -1)
 {
            break;

        }
        printf(“Removed task: %s, Priority: %d\n”, removedTask.taskName,
removedTask.priority);

    }

    return 0;
}
```

APPENDIX B – SCREENSHOTS

APPENDIX RESULT AND DISCUSSION

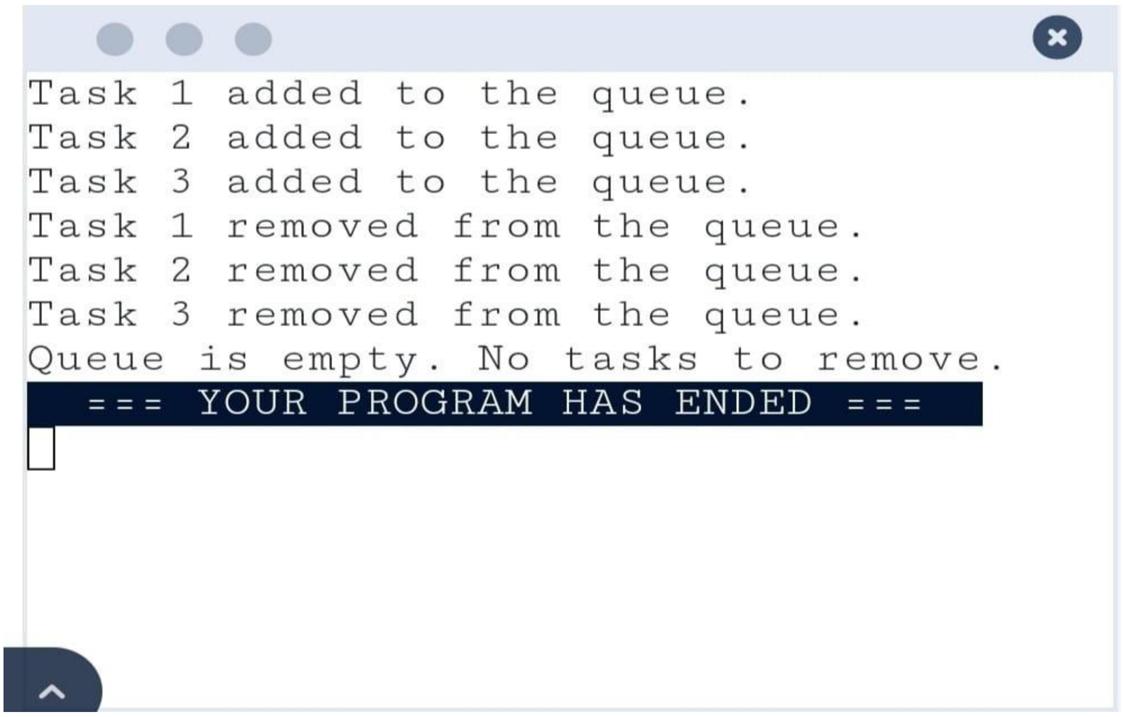
OUTPUT IN CODETANTRA:



```
// Function to add a task to the Queue
void enqueue(TaskQueue *queue, Task newTask) {
    if (isFull(queue)) {
```

```
Task 1 added to the queue.
Task 2 added to the queue.
Task 3 added to the queue.
Task 1 removed from the queue.
Task 2 removed from the queue.
Task 3 removed from the queue.
Queue is empty. No tasks to remove.
 === YOUR PROGRAM HAS ENDED ===
```

OUTPUT 1 :



```
Task 1 added to the queue.
Task 2 added to the queue.
Task 3 added to the queue.
Task 1 removed from the queue.
Task 2 removed from the queue.
Task 3 removed from the queue.
Queue is empty. No tasks to remove.
  === YOUR PROGRAM HAS ENDED ===
```