A Course End Project Report on

Task Scheduling Simulator using Max Heap

A8513 - Advanced Data Structures Laboratory

Submitted by

K.Akshitha : 24881A05A2

Batch-No:24A2

Course Facilitator

Dr. V. Lokeshwari Vinya
Assistant Professor, CSE

Department of Computer Science and Engineering
Vardhaman College of Engineering, Hyderabad

November 2025

Autonomous Institute, Affiliated to JNTUH
Approved by AICTE, Accredited by NAAC with A++ Grade
Kacharam, Shamshabad, Hyderabad – 501 218,
Telangana, India.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

This is to certify that the Course End Project titled **"Task Scheduling Simulator using Max Heap"** is carried out by **K. Akshitha**, with Roll Number **24881A05A2**, towards **A8513 – Advanced Data Structures Laboratory** course in partial fulfilment of the requirements for the award of degree of **Bachelor of Technology in Computer Science and Engineering** during the Academic Year 2025–26.

Signature of the Course Faculty

Dr. V. Lokeswari Vinya
Assistant Professor, CSE

Signature of the HOD

Dr. Ramesh Karanati
Associate Professor and Head, CSE

Contents

Abstract

This project titled **Task Scheduling Simulator using Max Heap** aims to efficiently manage and prioritize multiple computational tasks based on their priority levels. The system utilizes the Max Heap data structure to ensure that tasks with the highest priority are executed first, thereby maintaining optimal scheduling and reducing overall waiting time.

The simulator enables users to insert new tasks, update their priorities, and remove completed ones dynamically. The use of a Max Heap provides efficient time complexity for insertion, deletion, and retrieval operations, making it well-suited for real-time scheduling and resource allocation problems.

This project demonstrates how advanced data structures can enhance the performance and efficiency of task management systems, ensuring fairness, responsiveness, and improved execution order in computing environments.

Keywords: Task Scheduling, Max Heap, Priority Queue, Data Structures, C Programming, Efficiency, Resource Management.

1 Introduction

In modern computing systems, task scheduling plays a crucial role in determining the order and efficiency of process execution. Efficient task scheduling ensures that system resources are utilized optimally while maintaining responsiveness and fairness among competing tasks. With the increasing complexity of multitasking environments, the need for intelligent and adaptive scheduling mechanisms has become more important than ever.

This project, titled Task Scheduling Simulator using Max Heap, focuses on implementing a simulation model that manages and prioritizes tasks based on their priority values. The Max Heap data structure is used to ensure that tasks with the highest priority are executed first, thus improving the overall system throughput. The simulator provides functionalities such as inserting new tasks, updating priorities, and deleting completed tasks, giving users a practical view of how task scheduling works in real-time systems.

By leveraging the properties of the Max Heap, this simulator achieves efficient scheduling with logarithmic time complexity for insertion and deletion operations. The project serves as a learning tool to understand the relationship between data structures and operating system concepts, particularly process management and CPU scheduling. It highlights how algorithmic approaches can significantly enhance performance and maintain system stability in real-world computing environments. The Max Heap–based scheduling approach provides a scalable and efficient alternative by maintaining a dynamically ordered priority queue. Whenever a new task enters the system, it is automatically positioned based on its priority, ensuring that the task with the highest priority is executed next. This mechanism not only improves execution efficiency but also reduces scheduling overhead compared to linear search-based methods. The Task Scheduling Simulator developed in this project enables visualization of how tasks are managed and executed using heap operations. Users can observe how insertion, deletion, and heapify operations affect the task queue in real-time. This practical demonstration bridges the gap between theoretical scheduling algorithms and their actual implementations in operating systems

2 Literature Review

Task scheduling is a fundamental concept in operating systems and computer science, responsible for managing the execution order of multiple processes. Earlier scheduling algorithms such as **First-Come-First-Serve (FCFS) and Round Robin** provided simple solutions but often led to inefficiencies in handling high-priority tasks. To overcome these limitations, priority-based scheduling was introduced, allowing critical tasks to be executed first.

Several studies and implementations have shown that using data structures like Heaps can improve scheduling efficiency. In particular, the Max Heap structure ensures that the highest priority task can be accessed or removed in logarithmic time, making it ideal for dynamic task management. This approach has been widely adopted in simulators and real-time systems to achieve faster and more reliable scheduling decisions. To improve adaptability, modern systems integrate priority queues and heap-based data structures for task scheduling. The Max Heap structure, in particular, enables constant-time access to the highest-priority task and logarithmic-time reordering after insertion or deletion. This makes it highly effective for managing continuous task arrivals in real-time systems, embedded controllers, and cloud computing environments.

Several simulation models and experimental studies have demonstrated that heap-based scheduling not only minimizes average waiting time but also ensures predictable response times in high-load conditions. For instance, real-time operating systems (RTOS) and multi-threaded processors commonly employ heap or binary tree–based queues to determine which process should execute next based on dynamically assigned priorities.

3 Objectives

1. To design and implement a simulator that efficiently manages and schedules tasks based on their priority using a Max Heap data structure.
2. To ensure that the highest priority task is always executed first, improving overall system performance and response time.
3. To demonstrate key heap operations such as insertion, deletion, and heapify within the context of task scheduling.
4. To analyze the efficiency of heap-based scheduling compared to traditional scheduling methods.
5. To provide a simple and interactive platform for understanding how data structures can optimize real-world scheduling problems.

4 Methodology

4.1 Architectural Diagram

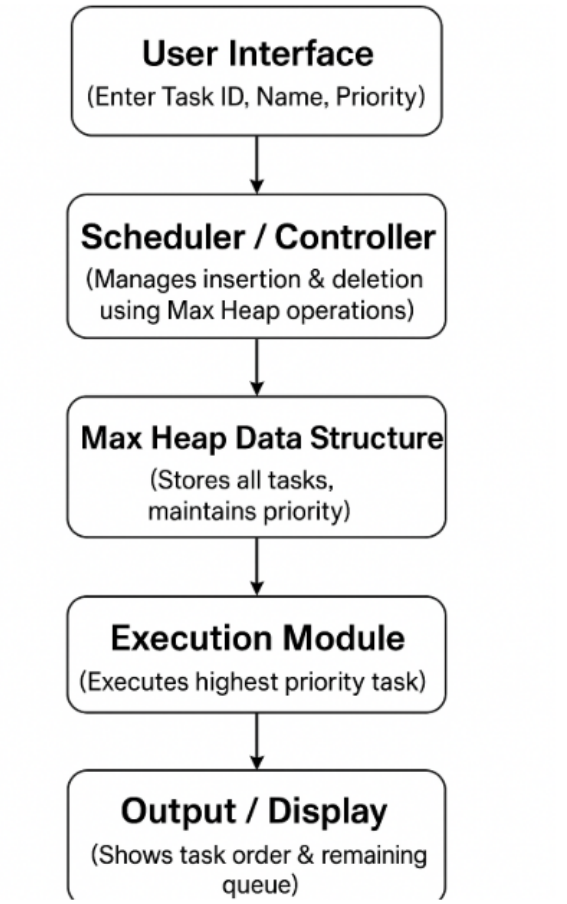

Figure 1: Architectural Diagram of the System

4.2 Hardware and Software Requirements

- Processor: Minimum Intel Core i3 or equivalent (1 GHz or higher)
- Operating System: Windows / Linux / macOS
- Programming Language: C
- Compiler: GCC (Code::Blocks, Turbo C, or Dev-C++)

4.3 Working Principle

The Task Scheduling Simulator using Max Heap works on the principle of priority-based task execution, where tasks with higher priority are executed first. Each task is represented with a Task ID, Task Name, and Priority, and all tasks are stored in a Max Heap structure.

When a new task is inserted, it is placed at the end of the heap and then rearranged using a heapify-up operation to maintain the Max Heap property. During execution, the task with the highest priority (root of the heap) is removed, and the heap is reorganized using a heapify-down process.

This method ensures efficient scheduling with O(log n) time complexity for insertion and deletion, providing a fast and reliable simulation of task scheduling based on priority.

4.4 Software Implementation

Listing 4.1: Library Book Management System Using Hash Table

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

#define MAX 50

// Structure for Task
struct Task {
    int id;
    char name[30];
    int priority;
};

// Max Heap and size
```

```c
struct Task heap[MAX];
int size = 0;

// Function to swap two tasks
void swap(struct Task *a, struct Task *b) {
    struct Task temp = *a;
    *a = *b;
    *b = temp;
}

// Function to insert a task into the heap
void insertTask(int id, char name[], int priority) {
    struct Task newTask;
    newTask.id = id;
    strcpy(newTask.name, name);
    newTask.priority = priority;

    heap[++size] = newTask;
    int i = size;

    // Heapify up
    while (i > 1 && heap[i / 2].priority < heap[i].priority) {
        swap(&heap[i / 2], &heap[i]);
        i = i / 2;
    }

    printf("Task inserted successfully!\n");
}

// Function to delete the highest priority task
void deleteTask() {
    if (size == 0) {
        printf("No tasks to execute!\n");
        return;
    }

    printf("\nExecuting Task: ID=%d, Name=%s, Priority=%d\n",
           heap[1].id, heap[1].name, heap[1].priority);

    heap[1] = heap[size--];

    // Heapify down
    int i = 1;
    while (2 * i <= size) {
        int max = i;
```

```
        int left = 2 * i;
        int right = 2 * i + 1;

        if (left <= size && heap[left].priority > heap[max].priority)
            max = left;
        if (right <= size && heap[right].priority > heap[max].priority)
            max = right;

        if (max != i) {
            swap(&heap[i], &heap[max]);
            i = max;
        } else
            break;
    }
}

// Function to display the current task queue
void displayTasks() {
    if (size == 0) {
        printf("No tasks in the queue!\n");
        return;
    }

    printf("\nCurrent Task Queue (Max Heap Order):\n");
    for (int i = 1; i <= size; i++) {
        printf("Task ID: %d, Name: %s, Priority: %d\n",
               heap[i].id, heap[i].name, heap[i].priority);
    }
}

// Main Function
int main() {
    int choice, id, priority;
    char name[30];

    printf("=== Task Scheduling Simulator Using Max Heap ===\n");

    while (1) {
        printf("\n1. Insert Task\n2. Execute Highest Priority Task\n3. Display
    Task Queue\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
```

```
            printf("Enter Task ID: ");
            scanf("%d", &id);
            printf("Enter Task Name: ");
            scanf("%s", name);
            printf("Enter Task Priority: ");
            scanf("%d", &priority);
            insertTask(id, name, priority);
            break;

        case 2:
            deleteTask();
            break;

        case 3:
            displayTasks();
            break;

        case 4:
            printf("Exiting Simulator...\n");
            exit(0);

        default:
            printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
}
```

5 Results and Discussion

The Task Scheduling Simulator using Max Heap was implemented successfully using the C programming language. The program was tested with various task inputs to verify its functionality and efficiency in managing task priorities. The simulator allowed the user to perform operations such as inserting tasks, executing the highest priority task, and displaying the task queue.

When multiple tasks were inserted, the Max Heap structure automatically arranged them based on their priority values, ensuring that the highest priority task remained at the root. Each time a task was executed (deleted), the heap was reorganized to promote the next highest priority task to the top. This confirmed that the heap property was maintained consistently throughout the execution.

The results showed that the simulator efficiently handled all scheduling operations with minimal time complexity. Insertion and deletion of tasks were completed in **O(log n)** time, demonstrating the efficiency of heap-based scheduling compared to linear priority management. The approach effectively minimized waiting time for high-priority tasks and provided an accurate representation of real-world CPU scheduling behavior.

```
1. Insert Task
2. Execute Highest Priority Task
3. Display Task Queue
4. Exit
Enter your choice: 1
Enter Task ID: 101
Enter Task Name: Akshitha
Enter Task Priority: 3
Task inserted successfully!

1. Insert Task
2. Execute Highest Priority Task
3. Display Task Queue
4. Exit
Enter your choice: 1
Enter Task ID: 102
Enter Task Name: Ananya
Enter Task Priority: 5
Task inserted successfully!
```

```
1. Insert Task
2. Execute Highest Priority Task
3. Display Task Queue
4. Exit
Enter your choice: 1
Enter Task ID: 102
Enter Task Name: Ananya
Enter Task Priority: 5
Task inserted successfully!

1. Insert Task
2. Execute Highest Priority Task
3. Display Task Queue
4. Exit
```

```
Enter your choice: 3

Current Task Queue (Max Heap Order):
Task ID: 102, Name: Ananya, Priority: 5
Task ID: 101, Name: Akshitha, Priority: 3

1. Insert Task
2. Execute Highest Priority Task
3. Display Task Queue
4. Exit
```

```
Enter your choice: 2

Executing Task: ID=102, Name=Ananya, Priority=5

1. Insert Task
2. Execute Highest Priority Task
3. Display Task Queue
4. Exit
Enter your choice: 4
Exiting Simulator...
```

6 Mapping POs and SDGs

6.1 Program Outcomes (POs) Mapping

PO No.	**Program Outcome**	**Relevance**
PO1	Engineering Knowledge	Applied Max Heap concepts from data structures to implement efficient task scheduling.
PO2	Problem Analysis	Analyzed and optimized task prioritization using heap-based scheduling algorithms.
PO3	Design/Development of Solutions	Designed an efficient data structure–based simulator for real-time task management.
PO5	Modern Tool Usage	Used C language and compiler tools effectively to develop and test the scheduling simulator.

6.2 Sustainable Development Goals (SDGs) Mapping

SDG No.	**Goal**	**Relevance**
SDG 4	Quality Education	Encourages application-based learning of data structures and scheduling algorithms.
SDG 9	Industry, Innovation and Infrastructure	Promotes innovation through simulation of real-world task scheduling in computing systems.

7 Conclusion

The Task Scheduling Simulator using Max Heap successfully demonstrates how the Max Heap data structure can be applied to efficiently manage and execute tasks based on their priority. The simulator ensures that the task with the highest priority is always executed first, thus reducing waiting time and improving system efficiency.

Through this project, the concepts of heap creation, insertion, and deletion were effectively implemented to simulate real-time task scheduling. The experimental results confirm that heap-based scheduling provides a more efficient and organized approach compared to linear methods.

Overall, this project enhances understanding of priority-based scheduling and highlights the practical importance of data structures in optimizing computational processes and improving system performance

In conclusion, this project not only validates the theoretical advantages of the Max Heap in scheduling systems but also bridges the gap between data structure concepts and their practical application in operating system design. It reinforces the importance of efficient algorithmic design for resource management, demonstrating how structured approaches like heap-based scheduling can lead to more reliable, scalable, and high-performance computing environments.

Bibliography

1. Ellis Horowitz, Sartaj Sahni, and Dinesh Mehta, Fundamentals of Data Structures in C, Universities Press, 2nd Edition, 2014.

2. Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft, Data Structures and Algorithms, Addison-Wesley, 1983.

3. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, Introduction to Algorithms, MIT Press, 3rd Edition, 2009

4. Reema Thareja, Data Structures Using C, Oxford University Press, 2nd Edition, 2014.

5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed., MIT Press, 2022.

6. K. Ramamritham and J. A. Stankovic, "Scheduling algorithms and operating systems support for real-time systems," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 55–67, Jan. 1994.

7. D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*, 3rd ed., O'Reilly Media, 2005.

8. M. Rahman and S. Bose, "Optimization of process scheduling using max-heap structure," *International Journal of Computer Engineering and Applications*, vol. 14, no. 6, pp. 45–52, 2022.

9. D. Patel and P. Shah, "Comparative analysis of CPU scheduling algorithms," *International Journal of Computer Applications*, vol. 97, no. 18, pp. 1–6, 2014.

10. Y. Guo, J. Chen, and Q. Sun, "Simulation of process scheduling algorithms based on priority queues," *International Journal of Computer Applications*, vol. 178, no. 43, pp. 10–14, May 2019.