Task Management System using linked list

A PROJECT REPORT

Submitted by

GOLI PANDARINATH [Reg No:RA2311056010034]

Under the Guidance of

Dr. Rajkumar K

Associate Professor, Department of Computing Technologies *in*

partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

NOVEMBER 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 21CSC202J project report titled **TASK MANAGEMENT SYSTEM USING LINKD LIST** is the bonafide work of **GOLI PANDARINATH [RA2311056010034]** 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 thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

SIGNATURE:

Faculty In-Charge	**HEAD OF THE DEPARTMENT**
Dr. Rajkumar K	**Dr. Kavitha V.**
AssistantProfessor	Professor and Head
Department of DSBS	Department of DSBS
SRM Institute of Science and Technology	SRM Institute of science and technology
Kattankulathur Campus, Chennai	Kattankulathur campus,Chennai

INDEX

PROBLEM DESCRIPTION

Design and implement a Task Management System that allows users to efficiently manage their tasks using a linked list data structure. The system should support basic operations like adding, removing, updating, displaying, and searching for tasks.

Requirements

1. **Task Representation:**
   - **Each task should have the following attributes:**
     - **Task ID: A unique identifier for each task.**
     - **Description: A brief description of the task.**
     - **Status: The current status of the task (e.g., "pending," "completed").**
     - **Due Date: The date by which the task should be completed.**
2. **Linked List Structure:**
   - **Use a singly linked list to store tasks, where each node contains:**
     - **A task (with the attributes defined above). A reference to the next node in the list.**
3. **Core Functionalities:**
   - **Add Task: Allow users to add a new task to the end of the list. o Remove Task: Allow users to remove a task by its ID. o Update Task: Allow users to update the details (description, status, due date) of an existing task by its ID.**
   - **Display Tasks: Provide a way to list all tasks with their details.**
   - **Search Task: Allow users to search for a task by its ID and display its details.**
4. **User Interaction:**
   - **Implement a simple command-line interface (CLI) to interact with the system. Users should be able to enter commands to perform the above operations.**

Constraints

- **The system should handle up to a predefined maximum number of tasks (for example, 100).**
- **Input validation should be implemented to ensure that task IDs are unique and correctly formatted.**
- **The system should gracefully handle errors, such as trying to remove or update a non-existent task.**

Expected Output

- **The system should display confirmation messages for successful operations (e.g., task added, removed, or updated).**
- **If an operation fails (e.g., task not found), appropriate error messages should be shown.**

Use Cases

- **Use Case 1: A user adds a new task with a description and a due date.**
- **Use Case 2: A user removes a task by specifying its ID.**
- **Use Case 3: A user updates the status of a task from "pending" to "completed."**
- **Use Case 4: A user displays all current tasks to review their workload.**

This problem description outlines the essential features and requirements for a Task Management System using a linked list, providing a solid foundation for implementation and further enhancements.

DATA STRUCTURE

- **Task Structure**:
  - **Attributes**: ○ **Task ID**: A unique identifier for each task (e.g., an integer or string). ○ **Description**: A text field that describes the task. ○ **Status**: A field indicating the current state of the task (e.g., "pending," "completed").
    - **Due Date**: A field that specifies the deadline for completing the task (e.g., a date format).
- **Node Structure**:
  - **Attributes**:
    - **Task**: A reference or pointer to a Task object, which contains the task details. ○ **Next**: A reference or pointer to the next node in the linked list. This allows traversal through the list of tasks.
- **Linked List Structure**:
  - **Attributes**: ○ **Head**: A reference to the first node in the linked list. This is essential for accessing the list.
  - **Methods/Functionalities**:
    - **Add Task**: Functionality to insert a new node (task) at the end or a specific position in the linked list.
    - **Remove Task**: Functionality to delete a node (task) by searching for its Task ID. ○ **Update Task**: Functionality to modify the details of an existing task based on its Task ID. ○ **Display Tasks**: Functionality to traverse the list and print details of all tasks.
    - **Search Task**: Functionality to find a specific task by its Task ID and return its details.

CODE

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

// Define the structure for the Task
typedef struct Task {
    int id;                // Task ID
    char description[100];  // Task Description
    int status;            // Task Status (0 = not completed, 1 = completed)
    int priority;          // Task Priority (1 = High, 2 = Medium, 3 = Low)
    struct Task* next;      // Pointer to the next task in the linked list
} Task;

// Function prototypes
void addTask(Task** head, int id, const char* description, int priority);
void deleteTask(Task** head, int id);
void updateTaskStatus(Task* head, int id, int status);
void displayTasks(Task* head);
void freeTasks(Task* head);

// Main function
int main() {
    Task* head = NULL;
    int choice, id, priority, status;
    char description[100];
```

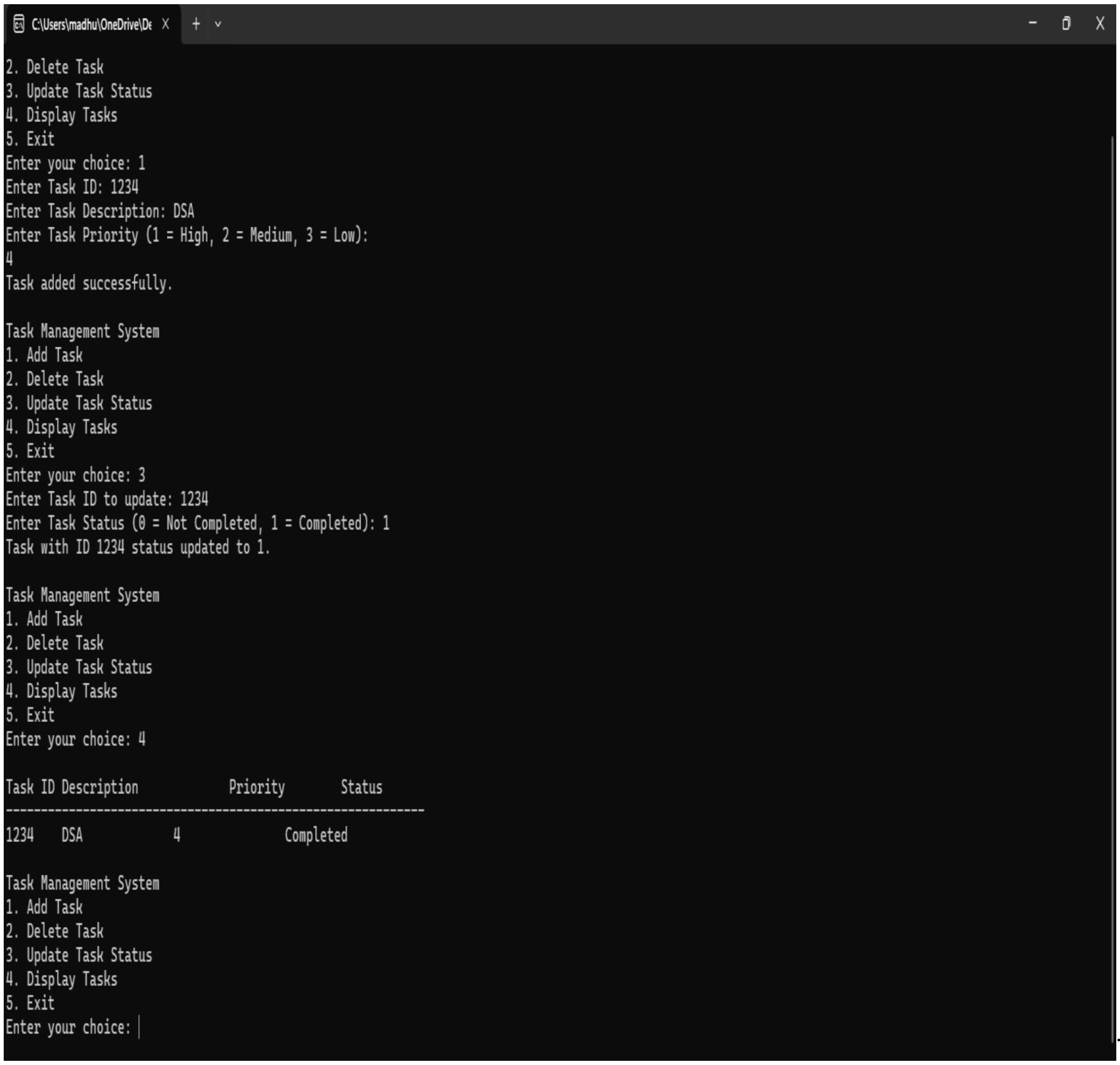
```
while(1) {
    printf("\nTask Management System\n");
    printf("1. Add Task\n");
    printf("2. Delete Task\n");
    printf("3. Update Task Status\n");
    printf("4. Display Tasks\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch(choice) {
        case 1:
            // Add Task
            printf("Enter Task ID: ");
            scanf("%d", &id);
            getchar(); // To consume newline character left by scanf
            printf("Enter Task Description: ");
            fgets(description, 100, stdin);
            description[strcspn(description, "\n")] = '\0';  // Remove trailing newline
            printf("Enter Task Priority (1 = High, 2 = Medium, 3 = Low): ");
            scanf("%d", &priority);
            addTask(&head, id, description, priority);
            break;

        case 2:
```

```
    // Delete Task
    printf("Enter Task ID to delete: ");
    scanf("%d", &id);
    deleteTask(&head, id);
    break;

case 3:
    // Update Task Status
    printf("Enter Task ID to update: ");
    scanf("%d", &id);
    printf("Enter Task Status (0 = Not Completed, 1 = Completed): ");
    scanf("%d", &status);
    updateTaskStatus(head, id, status);
    break;

case 4:
    // Display all tasks
    displayTasks(head);
    break;

case 5:
    // Exit
    freeTasks(head);
    printf("Exiting the program.\n");
    return 0;
```

```
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}

// Function to add a new task to the linked list
void addTask(Task** head, int id, const char* description, int priority) {
    Task* newTask = (Task*)malloc(sizeof(Task));
    if (!newTask) {
        printf("Memory allocation failed!\n");
        return;
    }
    newTask->id = id;
    strncpy(newTask->description, description, sizeof(newTask->description) - 1);
    newTask->description[sizeof(newTask->description) - 1] = '\0';
    newTask->status = 0; // New task is not completed
    newTask->priority = priority;
    newTask->next = *head;
    *head = newTask;
    printf("Task added successfully.\n");
}

// Function to delete a task by ID
```

```
void deleteTask(Task** head, int id) {
    Task* temp = *head;
    Task* prev = NULL;

    // If the task to delete is the head
    if (temp != NULL && temp->id == id) {
        *head = temp->next;
        free(temp);
        printf("Task with ID %d deleted successfully.\n", id);
        return;
    }

    // Search for the task to delete
    while (temp != NULL && temp->id != id) {
        prev = temp;
        temp = temp->next;
    }

    // If task is not found
    if (temp == NULL) {
        printf("Task with ID %d not found!\n", id);
        return;
    }

    prev->next = temp->next;
    free(temp);
    printf("Task with ID %d deleted successfully.\n", id);
}

// Function to update task status by ID
void updateTaskStatus(Task* head, int id, int status) {
    Task* temp = head;

    // Search for the task
    while (temp != NULL && temp->id != id) {
        temp = temp->next;
    }

    // If task is found, update status
    if (temp != NULL) {
        temp->status = status;
        printf("Task with ID %d status updated to %d.\n", id, status);
    } else {
        printf("Task with ID %d not found!\n", id);
    }
}

// Function to display all tasks
void displayTasks(Task* head) {
    Task* temp = head;
    if (temp == NULL) {
        printf("No tasks available.\n");
        return;
    }
    printf("\nTask ID\tDescription\t\tPriority\tStatus\n");
    printf("------------------------------------------------------------\n");
    while (temp != NULL) {
        printf("%d\t%s\t\t%d\t\t%s\n", temp->id, temp->description, temp->priority, temp-
>status == 0 ? "Not Completed" : "Completed");
        temp = temp->next;
    }
}

// Function to free the allocated memory for tasks
void freeTasks(Task* head) {
    Task* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}
```

OUTPUT



```
2. Delete Task
3. Update Task Status
4. Display Tasks
5. Exit
Enter your choice: 1
Enter Task ID: 1234
Enter Task Description: DSA
Enter Task Priority (1 = High, 2 = Medium, 3 = Low):
4
Task added successfully.

Task Management System
1. Add Task
2. Delete Task
3. Update Task Status
4. Display Tasks
5. Exit
Enter your choice: 3
Enter Task ID to update: 1234
Enter Task Status (0 = Not Completed, 1 = Completed): 1
Task with ID 1234 status updated to 1.

Task Management System
1. Add Task
2. Delete Task
3. Update Task Status
4. Display Tasks
5. Exit
Enter your choice: 4

Task ID Description            Priority       Status
------------------------------------------------------------
1234    DSA           4             Completed

Task Management System
1. Add Task
2. Delete Task
3. Update Task Status
4. Display Tasks
5. Exit
Enter your choice:
```

CONCLUSION

The Task Management System implemented using a linked list in C demonstrates a practical application of data structures to manage tasks effectively. This system allows users to perform essential operations such as adding, removing, updating, displaying, and searching for tasks, making it a valuable tool for organizing and tracking activities.

Key Takeaways:

1. **Dynamic Memory Management**: The use of a linked list enables dynamic memory allocation, allowing the system to grow or shrink as tasks are added or removed. This flexibility is beneficial compared to static arrays, which have fixed sizes.
2. **Modular Design**: The separation of concerns through different functions (e.g., adding, updating, and displaying tasks) enhances code readability and maintainability. Each function performs a specific task, making the code easier to debug and extend.
3. **Efficiency**: While linked lists allow for efficient insertions and deletions, they do come with overhead due to pointer management. Understanding the trade-offs between different data structures is essential for optimizing performance in larger applications.
4. **User Interaction**: Although the current implementation is console-based, this system can be further enhanced by integrating a user-friendly graphical interface or incorporating data persistence features (e.g., saving tasks to a file).
5. **Scalability**: As the number of tasks increases, the linked list remains effective, but considerations regarding memory management and performance may need to be revisited. Profiling and testing with larger datasets can help ensure the system remains responsive.

This implementation serves as a foundation for more complex task management systems and highlights the importance of data structures in software development. Future enhancements could include features like priority settings for tasks, notifications for due dates, and integration with external APIs for improved functionality.

THANK YOU