Skillbit Technologies

Project Report

Task Scheduler

Submitted by:

Swathi Setty
4636

C++ Programming

Submitted to:

Skillbit Technologies

C++ Programming

Date of Submission:

June 14, 2025

Table of Contents

Introduction

A Task Scheduler is a software application designed to help users manage and organize their daily tasks efficiently. In today's fast-paced world, keeping track of various tasks, deadlines, and responsibilities can be challenging. A simple and effective task scheduler ensures that important activities are not overlooked and helps users prioritize their work.

This project presents a command-line based Task Scheduler implemented in C++. The application allows users to add, edit, delete, and view tasks, each associated with a unique ID, a title, and a due date. All tasks are stored persistently in a text file, ensuring that the user's task list is preserved between sessions.

The primary goal of this Task Scheduler is to provide a straightforward and user-friendly tool for personal productivity, demonstrating the use of file handling, data structures, and basic user interface design in C++. It serves as a practical example of how fundamental programming concepts can be applied to solve real-world problems.

.

Objectives

The main objectives of the Task Scheduler project are:

1. **Task Management:**
   To provide users with a simple way to create, view, edit, and delete their daily tasks.
2. **Persistent Storage:**
   To ensure that all tasks are saved to a file (tasks.txt) so that information is retained even after the program is closed and reopened.
3. **User-Friendly Interface:**
   To design an intuitive, menu-driven command-line interface that allows users to interact with the application easily.
4. **Daily Task Tracking:**
   To enable users to quickly view tasks that are due on the current day, helping them focus on immediate priorities.
5. **Demonstration of Programming Concepts:**
   To illustrate the practical use of data structures (such as structs and vectors), file operations, and basic control flow in C++.
6. **Foundation for Further Development:**
   To serve as a base for more advanced task management features, such as task prioritization, reminders, and multi-user support in future versions.

System Design

1. Architecture

- **Monolithic Structure:**
  All functionalities are implemented within a single C++ program. The application does not use external libraries for concurrency or advanced scheduling, distinguishing it from multi-threaded or distributed schedulers.

- **Persistent Storage:**
  Tasks are stored in a plain text file (tasks.txt). This ensures that all data persists between program runs, allowing users to retain their task lists.

- **In-Memory Data Handling:**
  When the program starts, all tasks are loaded into a vector in memory. All operations—adding, editing, deleting—are performed on this in-memory list. Changes are saved back to the file after each modification.

2. Main Components

- **Data Model:**
  Each task is represented by a struct containing an ID, title, and due date.

- **File Operations:**
  The application reads from and writes to tasks.txt to load and save tasks.

- **CRUD Functions:**
  Functions are provided for creating, reading (listing), updating, and deleting tasks.

- **User Interface:**
  A menu-driven command-line interface guides the user through available operations.

- **Task Filtering:**
  The system can display all tasks or filter tasks due on the current date.

Code Explanation

1. Data Model

The application uses a straightforward data model where each task is represented by a structure containing three fields:

- An integer **id** for unique identification
- A string **title** for the task description
- A string **dueDate** for the task's deadline in YYYY-MM-DD format This structure makes it easy to store, retrieve, and manipulate tasks.

2. File Operations

The application uses file handling to ensure tasks are saved between sessions:

- **Loading Tasks:** At startup, the app reads from a text file (tasks.txt). Each task is read as three lines (ID, title, and due date) and stored in a vector.
- **Saving Tasks:** After any change (add, edit, or delete), the app writes all tasks back to the file, making sure no data is lost.

3. CRUD Functions

- **Create:** Prompts the user to enter a task title and due date, assigns a unique ID, and saves the new task.
- **Read (View):** Lists all tasks with their IDs, titles, and due dates.
- **Update (Edit):** Lets the user select a task by ID and update its title and due date.
- **Delete:** Allows the user to remove a task by specifying its ID.

4. User Interface

The app features a menu-driven command-line interface. Users choose options by entering numbers, making the scheduler simple and accessible for everyone. Each menu option corresponds to a specific task management function, guiding users step by step.

Features

The Task Scheduler application offers several practical features designed to help users efficiently manage their daily tasks:

1. **Add New Tasks**
    - Users can create new tasks by entering a title and a due date. Each task is automatically assigned a unique ID.
2. **Edit Existing Tasks**
    - Tasks can be updated by selecting their ID and modifying the title or due date.
3. **Delete Tasks**
    - Users can remove tasks from the list by specifying the task ID.
4. **View All Tasks**
    - The application can display a complete list of all tasks, showing their IDs, titles, and due dates for easy reference.
5. **View Today's Tasks**
    - Users can quickly see tasks that are due on the current date, helping them focus on immediate priorities.
6. **Persistent Storage**
    - All tasks are saved in a text file (tasks.txt). This ensures that the task list is preserved between sessions and no data is lost when the program closes.
7. **Simple, Menu-Driven Interface**
    - The command-line menu is intuitive and easy to navigate, making task management accessible even for beginners.
8. **Automatic ID Assignment**
    - Each new task receives a unique ID, eliminating confusion and making it easy to reference or modify tasks.

Limitations

The Task Scheduler application, while functional and straightforward, has several limitations:

- **No Task Priority or Status:**
  Tasks cannot be marked as completed, nor can they be assigned different priority levels (e.g., high, medium, low).
- **No Input Validation:**
  The program does not check if the entered due date is in the correct format or if it is a valid calendar date. Similarly, it does not prevent duplicate titles or handle empty inputs robustly.
- **No Sorting or Advanced Filtering:**
  Tasks are listed in the order they were added. There is no option to sort or filter tasks by priority, overdue status, or custom criteria.
- **Single User Only:**
  The application is designed for a single user and does not support multiple user accounts or profiles.
- **Basic User Interface:**
  The command-line interface, while simple, may not be as user-friendly as a graphical interface, especially for users unfamiliar with terminal-based applications.
- **No Reminders or Notifications:**
  The program does not alert users when a task is due soon or overdue.
- **Limited Data Storage:**
  All data is stored in a plain text file, which may not scale well for a very large number of tasks or support concurrent access.
- **No Security Features:**
  Tasks are stored in an unencrypted file, so there is no protection for sensitive information.

Possible Improvements

While the Task Scheduler is functional, there are several ways it can be enhanced to provide a richer and more user-friendly experience:

1. **Add Task Priority and Status**
   - Allow users to assign priority levels (e.g., high, medium, low) and mark tasks as completed or pending.
2. **Input Validation**
   - Implement checks to ensure the due date follows the correct format and is a valid date. Prevent empty titles and duplicate entries.
3. **Sorting and Advanced Filtering**
   - Enable sorting tasks by due date, priority, or status. Add filters to show overdue, completed, or upcoming tasks.
4. **Graphical User Interface (GUI)**
   - Develop a GUI version using libraries like Qt or a web-based interface for improved usability and accessibility.
5. **Reminders and Notifications**
   - Integrate reminders or notifications to alert users about upcoming or overdue tasks.
6. **Recurring Tasks**
   - Add support for tasks that repeat daily, weekly, or monthly.
7. **Multi-User Support**
   - Allow multiple users to maintain separate task lists, possibly with authentication.
8. **Data Security**
   - Encrypt the task file or add password protection to safeguard sensitive information.

Conclusion

The Task Scheduler project successfully demonstrates a basic yet effective approach to managing daily tasks using C++. Through its simple data model, file-based persistence, and intuitive menu-driven interface, the application allows users to add, edit, delete, and view tasks with ease. While it currently supports fundamental task management features, the program lays a solid foundation for further enhancements such as task prioritization, reminders, and a graphical interface. Overall, this project highlights key programming concepts like data structures, file handling, and user interaction, making it a valuable learning tool and a practical utility for personal productivity.

References:

Stack Overflow: Creating a Task in Task Scheduler with C++

This thread guides how to create and manage Windows scheduled tasks programmatically in C++ using Task Scheduler 2.0 COM interfaces, with example code and tips on triggers, actions, and permissions.

YouTube: Building your own task scheduler in C++

This video explains the theory behind Linux task schedulers and demonstrates how to build a C++ scheduler simulation. It covers scheduling algorithms, task types, and practical implementation steps.

Appendix: Source Code

```
#include <iostream>

#include <fstream>

#include <vector>

#include <ctime>

using namespace std;


struct Task {

   int id;

   string title;

   string dueDate; // Format: YYYY-MM-DD

};


vector<Task> tasks;

const string filename = "tasks.txt";


// Utility to get current date as string

string getCurrentDate() {

   time_t now = time(0);

   tm *ltm = localtime(&now);

   char buf[11];

   sprintf(buf, "%04d-%02d-%02d", 1900 + ltm->tm_year, 1 + ltm->tm_mon, ltm->tm_mday);

   return string(buf);

}


void loadTasks() {

   tasks.clear();
```

```
    ifstream fin(filename);
    Task t;
    while (fin >> t.id >> ws && getline(fin, t.title) && getline(fin, t.dueDate)) {
        tasks.push_back(t);
    }
    fin.close();
}

void saveTasks() {
    ofstream fout(filename);
    for (Task t : tasks) {
        fout << t.id << endl << t.title << endl << t.dueDate << endl;
    }
    fout.close();
}

void addTask() {
    Task t;
    cout << "Enter task title: ";
    cin.ignore();
    getline(cin, t.title);
    cout << "Enter due date (YYYY-MM-DD): ";
    cin >> t.dueDate;
    t.id = tasks.empty() ? 1 : tasks.back().id + 1;
    tasks.push_back(t);
    saveTasks();
    cout << "Task added successfully!\n";
}
```

```
void editTask() {
    int id;
    cout << "Enter task ID to edit: ";
    cin >> id;
    bool found = false;
    for (Task &t : tasks) {
        if (t.id == id) {
            cout << "New title: ";
            cin.ignore();
            getline(cin, t.title);
            cout << "New due date (YYYY-MM-DD): ";
            cin >> t.dueDate;
            found = true;
            break;
        }
    }
    if (found) {
        saveTasks();
        cout << "Task updated!\n";
    } else {
        cout << "Task not found.\n";
    }
}

void deleteTask() {
    int id;
    cout << "Enter task ID to delete: ";
    cin >> id;
    bool found = false;
    for (auto it = tasks.begin(); it != tasks.end(); ++it) {
        if (it->id == id) {
            tasks.erase(it);
            found = true;
            break;
        }
    }
    if (found) {
        saveTasks();
        cout << "Task deleted.\n";
    } else {
        cout << "Task not found.\n";
    }
}

void showTodayTasks() {
    string today = getCurrentDate();
    cout << "\nToday's Tasks (" << today << "):\n";
    bool any = false;
    for (Task t : tasks) {
        if (t.dueDate == today) {
            cout << "ID: " << t.id << " | " << t.title << " | Due: " << t.dueDate << endl;
            any = true;
        }
    }
    if (!any) cout << "No tasks for today.\n";
```

```
}

void listAllTasks() {
    cout << "\nAll Tasks:\n";
    for (Task t : tasks) {
        cout << "ID: " << t.id << " | " << t.title << " | Due: " << t.dueDate << endl;
    }
}

int main() {
    loadTasks();
    int choice;
    do {
        cout << "\n--- Task Scheduler ---\n";
        cout << "1. Add Task\n2. Edit Task\n3. Delete Task\n4. Show Today's Tasks\n5. Show All Tasks\n6. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: addTask(); break;
            case 2: editTask(); break;
            case 3: deleteTask(); break;
            case 4: showTodayTasks(); break;
            case 5: listAllTasks(); break;
            case 6: cout << "Goodbye!\n"; break;
            default: cout << "Invalid choice.\n";
        }
    } while (choice != 6);

    return 0;

}
```