multi thread task manager

Multi-threaded Task Manager Application
The task manager application manages a set of user tasks and a pool of resources. Each task has a priority level, a state, and
required resources. Resources are allocated to tasks based on their priority level and availability of resources. The application
includes a scheduler, which selects the next task to execute based on its priority level, and a deadlock prevention algorithm,
which checks for and resolves deadlocks.
Program Structure

- main function initializes the application, creates the scheduler and deadlock prevention threads, and enters a loop to
  read user input and execute commands.
- create_task function creates a new task with the given priority level and list of required resources. The task is added
  to the list of tasks and a new thread is created to execute the task.
- destroy_task function cancels the thread associated with the given task ID and removes the task from the list of tasks.
- task_fn function is the main entry point for task threads. It executes the task and releases any resources it acquired
  upon completion.
- scheduler_fn function is the main entry point for the scheduler thread. It selects the next task to execute based on its
  priority level and state.
- deadlock_fn function is the main entry point for the deadlock prevention thread. It checks for and resolves any
  deadlocks by releasing resources held by tasks that are blocking other tasks.
- resource_t struct represents the pool of resources. It contains arrays for the available, allocated, maximum, and need

resources.

- pcb_t struct represents a task. It contains fields for the task ID, priority level, state, list of required resources, and the
  associated thread.
- Semaphores: The application uses several semaphores to protect critical sections and signal resource allocation and
  task creation.
  Project Requirements
  Your task is to implement the task manager application in C using POSIX threads, process control blocks (PCBs), schedulers,
  semaphores, and process management concepts. Your implementation should meet the following requirements:
- The application should support up to 100 tasks and 10 resources.
- The application should provide a command-line interface for creating, destroying, and listing tasks, as well as allocating
  and releasing resources.
- Tasks should be created with a unique ID, priority level, and required resources.
- Tasks should be executed in a separate thread, and released resources should be returned to the resource pool.
- The scheduler should select the next task to execute based on its priority level and state (e.g., ready, blocked).
- The deadlock prevention algorithm should check for and resolve any deadlocks by releasing resources held by tasks that
  are blocking other tasks.
  Submission
  Create a file called studentNo.c containing your source code and save it to your local machine. Test the code to ensure that
  it performs the tasks described above.