Assignment - 07
CS341: Operating System Lab

General Instruction

- Markings will be based on the correctness and soundness of the outputs. Marks will be deducted in case of plagiarism.
- Proper indentation & appropriate comments (if necessary) are mandatory in the code.

Problem 1:

In this lab assignment, you are required to implement the classic Dining Philosopher problem in C using the concept of multithreading and synchronization mechanisms. The aim is to simulate a scenario where philosophers are sitting around a table and trying to eat using shared resources (forks). Ensure proper synchronization to prevent deadlock and starvation.

Requirements:

- There are 5 philosophers sitting around a circular table. Each philosopher alternates between thinking and eating.
- Each philosopher must pick up two forks (shared resources) to eat but can pick up only one fork at a time.
- Implement the solution using mutexes and/or semaphores to ensure no deadlock or starvation.
- Ensure that only one philosopher accesses a fork at a time.

Output:

- After a philosopher has eaten, print a message indicating which philosopher is eating and which forks they picked up.
- Ensure that the philosophers continue thinking and eating in a circular manner without any philosopher getting stuck.

Example:

```
Philosopher 1 is thinking.
Philosopher 2 is eating, picked up forks 2 and 3.
Philosopher 3 is thinking.
Philosopher 4 is eating, picked up forks 4 and 5.
```

Problem 2:

Extend the solution of the Dining Philosopher problem from the previous question to work with an arbitrary number of philosophers N. The goal is to ensure the system is scalable and handles different numbers of philosophers efficiently.

Requirements:

- Allow the user to input the number of philosophers N.
- Implement the same logic as before but ensure it works for any number of philosophers sitting around the table.

Output:

- After each philosopher has eaten, print the philosopher's number and which forks they picked up.

Example: For $N = 6$:

```
Philosopher 1 is eating, picked up forks 1 and 2.
Philosopher 2 is thinking.
Philosopher 3 is eating, picked up forks 3 and 4.
Philosopher 4 is thinking.
```

Problem 3:

In this problem, you will implement the Reader-Writer problem, where multiple readers can read from a shared resource (e.g., a database), but writers must have exclusive access for writing.

Requirements:

- Implement the Reader-Writer problem using threads in C. Multiple readers should be able to read concurrently, but writers must wait for all readers to finish before writing.
- Use appropriate synchronization mechanisms like semaphores or mutexes to ensure mutual exclusion between readers and writers.
- Ensure fairness by preventing starvation of both readers and writers.

Output:

- Print messages indicating when a reader starts and stops reading, and when a writer starts and stops writing.
- Ensure that the output correctly reflects the mutual exclusion between readers and writers.

Example:

```
Reader 1 is reading.
Reader 2 is reading.
Writer 1 is waiting.
Reader 1 finished reading.
Reader 2 finished reading.
Writer 1 is writing.
```

Submission:

Submit your C program code along with a detailed explanation of the synchronization mechanisms you used. Make sure to include comments in your code to explain the logic, particularly the multithreading and synchronization parts.