## **Design Document for "Sleeping Teaching Assistant"**

### **1. Project Overview**

This project implements a simulation of a teaching assistant (TA) helping students during office hours. The TA's office accommodates only one student at a time, while three chairs outside the office serve as a waiting area. If the TA is helping a student, other students wait outside or leave if all chairs are occupied. The program uses **POSIX threads** (pthreads), **mutex locks**, and **semaphores** to manage synchronization and resource allocation.

### **2. Requirements**

* **TA Functionality**:
  + The TA sleeps if there are no students needing help.
  + The TA assists students one by one.
  + After helping a student, the TA checks if there are students waiting.
* **Student Functionality**:
  + A student can wake up the TA if the TA is asleep.
  + A student waits on a chair outside the office if the TA is occupied and a chair is available.
  + A student leaves if no chairs are available.

### **3. Program Components**

#### **3.1 Functions**

1. **TA\_Activity**
   * **Purpose**: Simulate the TA’s activities (sleeping, helping students, assigning chairs).
   * **Details**:
     + The TA checks if any students are waiting. If no students are present, the TA goes to sleep.
     + When a student arrives, the TA wakes up to help them.
     + After helping, the TA reduces the count of waiting students if there are any and signals the next student.
2. **Student\_Activity**
   * **Purpose**: Simulate the students’ activities (waking up the TA, checking chair availability, leaving if full).
   * **Details**:
     + When a student arrives, they check if the TA is asleep.
     + If the TA is asleep, the student wakes the TA.
     + If the TA is busy, the student checks for an available chair.
     + If no chair is available, the student leaves and returns later.
3. **Main**
   * **Purpose**: Set up the environment by initializing semaphores, mutexes, and threads.
   * **Details**:
     + Initializes semaphores for chair control and TA sleeping status.
     + Initializes mutexes for safe access to shared resources (chair count).
     + Creates one TA thread and multiple student threads.
     + Joins threads and cleans up resources after execution.

### **4. Synchronization Mechanisms**

1. **Mutex Locks**:
   * Protect access to shared variables (e.g., the number of available chairs) to ensure thread-safe operations.
   * Prevents race conditions when students check or modify the number of waiting students.
2. **Semaphores**:
   * **chairs\_sem**: Represents the availability of chairs. Allows students to wait and the TA to notify the next waiting student.
   * **ta\_sleeping**: Indicates whether the TA is asleep. Students signal this semaphore to wake up the TA.

### **5. Data Structures**

* **waiting\_students**: An integer variable that tracks the current count of waiting students.
* **semaphores**:
  + chairs\_sem: Initialized to the number of available chairs (3), allows students to wait if chairs are free.
  + ta\_sleeping: Initialized to 0, used to wake up the TA when students need help.

### **6. Flow of Execution**

1. **Initialization**:
   * The main function initializes the semaphores and mutex.
   * Creates and starts the TA thread, then creates student threads, simulating student arrivals at different times.
2. **TA\_Activity Function**:
   * The TA waits on ta\_sleeping to check if a student is requesting help.
   * If a student is waiting, the TA helps the student and then checks the count of waiting students.
   * The TA signals the next waiting student using chairs\_sem if other students are present.
3. **Student\_Activity Function**:
   * Each student thread:
     + Checks if the TA is available or asleep.
     + Wakes the TA if asleep or takes a chair if the TA is busy.
     + If no chairs are available, the student leaves and checks back later.
4. **Cleanup**:
   * After all students have completed their activities, the main function joins the threads.
   * Mutexes and semaphores are destroyed to free up resources.

### **7. Error Handling**

* **Thread Creation Failure**: The program checks for successful thread creation and handles any errors accordingly.
* **Resource Cleanup**: Mutexes and semaphores are destroyed at the end to prevent resource leakage.

### **8. Pseudocode**

Initialize semaphores and mutex

Create TA thread

For each student:

Create a student thread

TA\_Activity:

While True:

Wait on ta\_sleeping semaphore

Help student

Lock mutex

If waiting\_students > 0:

Decrement waiting\_students

Signal chairs\_sem for next student

Unlock mutex

Student\_Activity:

If TA is sleeping:

Wake up TA (signal ta\_sleeping)

Else if there are chairs:

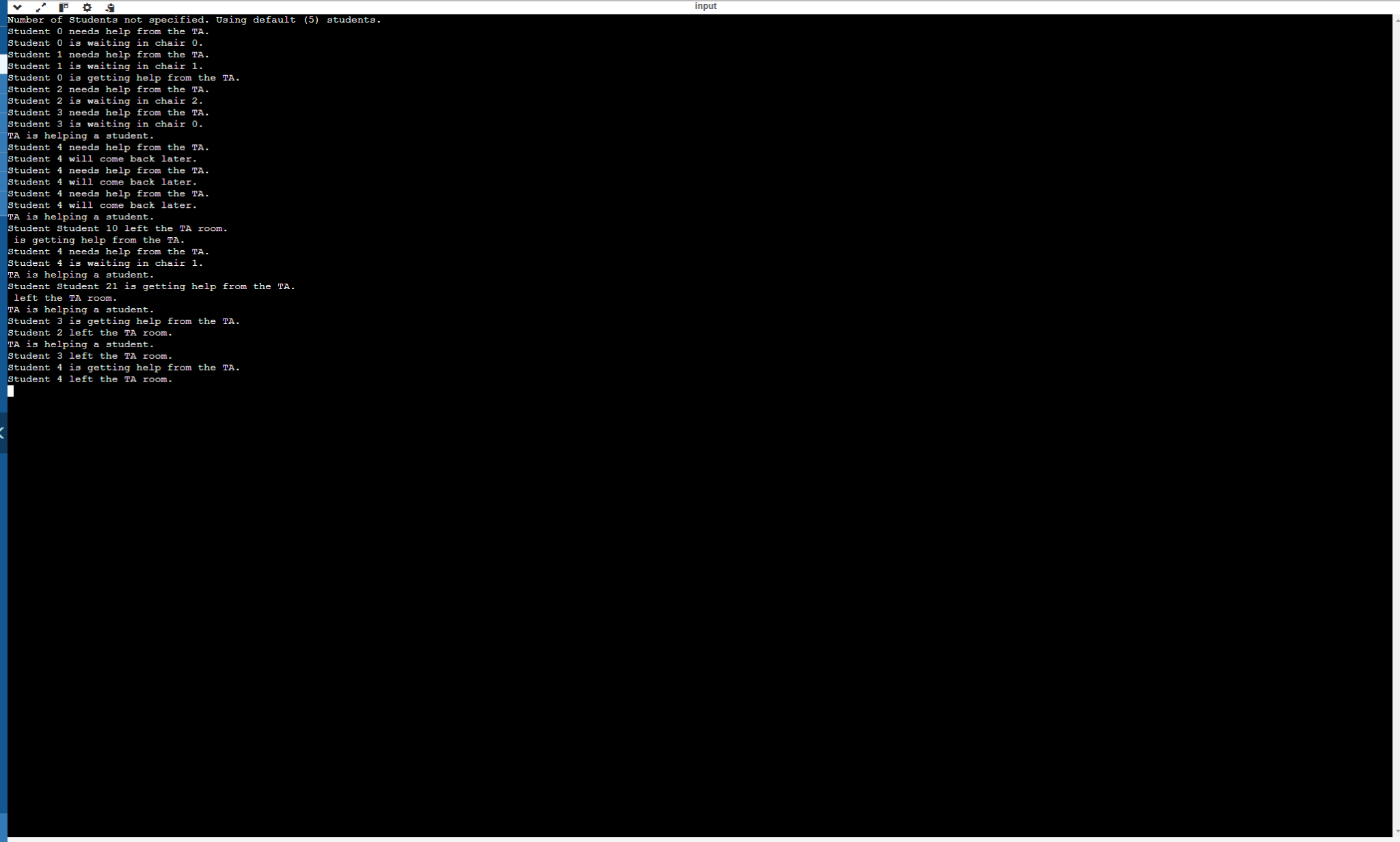
Sit on a chair (decrement waiting\_students)

Wait for TA (wait on chairs\_sem)

Else:

Leave and return later

Clean up resources and join threads

**Screenshot of OUTPUT:  
**