Producer-Consumer Problem Using Mutexes and Semaphores

Title: Bounded Buffer Producer-Consumer System

Problem Statement:

Implement a **multi-threaded producer-consumer system** with a **bounded buffer** using **mutexes** and **semaphores**.

The system must:

- Use **producer threads** to produce items and place them into a **shared buffer**.
- Use **consumer threads** to take items from the buffer and process them.
- Ensure proper **synchronization** to avoid **buffer overflow** or **underflow**.

Detailed Requirements:

1. **Buffer Management:**
    - Implement a **shared buffer** with a fixed size of 10 items.
    - Use **semaphores** to manage the number of available slots and items in the buffer.
2. **Synchronization Requirements:**
    - Use a **mutex** to protect access to the shared buffer.
    - Use **two semaphores**:
        - `empty`: Tracks the number of empty slots in the buffer.
        - `full`: Tracks the number of items in the buffer.
3. **Concurrency Requirements:**
    - Create **5 producer threads** and **5 consumer threads**.
    - Ensure **thread-safe access** to the shared buffer.

Hints:

- Use `sem_wait()` and `sem_post()` to manage buffer slots.
- Use `pthread_mutex_t` to protect the buffer during read/write operations.

- Use **random delays** to simulate the production and consumption times.

Approach:

1. **Initialize the shared buffer, mutex, and semaphores.**
2. **Create producer and consumer threads.**
3. **Producers wait for an empty slot, produce an item, and signal the `full` semaphore.**
4. **Consumers wait for a full slot, consume an item, and signal the `empty` semaphore.**

Multi-Threaded File Download Manager Using Mutexes and Semaphores

Title: Thread-Safe File Download Manager

Problem Statement:

Implement a **multi-threaded file download manager** that handles **multiple downloads concurrently** while ensuring **thread-safe access** to a shared download log.

The system must:

- Use **POSIX threads** to simulate **multiple download tasks**.
- Use a **mutex** to protect the shared download log.
- Use a **semaphore** to control the maximum number of concurrent downloads (set to 3).

Detailed Requirements:

1. **Download Tasks:**
    - Simulate file downloads by having each thread "download" a file (use a random delay to simulate download time).
    - Each thread must log the download details (file name, start time, end time) to a **shared download log**.

2. **Synchronization Requirements:**
    - Use a **mutex** to protect the shared download log.
    - Use a **semaphore** to limit the number of concurrent downloads to **3**.
3. **Concurrency Requirements:**
    - Create **10 download threads**.
    - Ensure **thread-safe access** to the shared log file.

Hints:

- Use `sem_wait()` and `sem_post()` to control the number of concurrent downloads.
- Use `pthread_mutex_t` to protect the download log during write operations.
- Use **random delays** to simulate download times.

Approach:

1. **Initialize the mutex and semaphore.**
2. **Create 10 download threads.**
3. **Each thread waits for a semaphore slot, performs the download, logs the details, and then releases the semaphore slot.**
4. **Ensure proper locking and unlocking of the mutex during log updates.**