**Multiprocessor Task Scheduling System**

A POSIX Threads Implementation with Producer-Consumer Synchronization

**Synchronization Approach**

Implemented a thread-safe FIFO task queue for a multiprocessor system using:

* **Mutex Locks**: Ensured exclusive access to the shared queue during enqueue/dequeue operations, preventing race conditions.
* **Condition Variables**:
  + Producer thread waits if queue is full (pthread\_cond\_wait)
  + Consumer threads (CPU workers) wait if queue is empty
  + Broadcast signaling (pthread\_cond\_broadcast) to handle edge cases
* **Termination Handling**: Introduced no\_more\_tasks flag to ensure all tasks complete before shutdown, even with intermittent queue emptiness.

**Key Challenges & Solutions**

1. **Early Termination Bug**
   * *Issue*: Consumer threads exited prematurely when queue was temporarily empty.
   * *Fix*: Added state variable no\_more\_tasks to distinguish between "no tasks yet" vs "all tasks done".
2. **Modular Task Generation**
   * *Constraint*: Makefile couldn’t compile two main() functions.
   * *Solution*: Designed standalone createTasks.c to generate random burst times (1-7ms) as a pre-processing step.
3. **Logging System**
   * Timestamped task lifecycle events (arrival → CPU assignment → completion)
   * Calculated real-time metrics:

avg\_wait\_time = total\_wait\_time / total\_tasks;

avg\_turnaround\_time = total\_turnaround\_time / total\_tasks;