NEON OS Simulator User Manual

How to Use the Simulator

1. Control Panel (Left Sidebar)

- **Play/Pause**: Starts or pauses the simulation clock.
- **Reset**: Clears all threads, processes, and resources.
- **Threading Model**: Select between `One-to-One`, `Many-to-One`, or `Many-to-Many`.
  - *Tip:* Changing this while threads are running will affect scheduling logic immediately.
- **CPUs**: Set the number of CPU cores (1-4).
- **Sim Speed**: Drag the slider to control how fast the simulation ticks (100ms = fast, 2000ms = slow).
- **Create Process**: Spawns a new process with `N` threads, `B` burst time, and `P` priority.

2. Visualizer (Center)

- **CPU Cards**: Shows currently running threads. When running, hover over a thread card to see **Interactions**.
- **Ready Queue**: Threads waiting for CPU time.
- **Blocked Queue**: Threads waiting for Resources (Database/Printer) or Monitors (Buffer).

3. Resources & Monitors (Right Sidebar)

- **Monitor: Buffer**: Visualizes a Condition Variable monitor. Shows the Mutex Lock state and Waiting Queues for `NotEmpty` / `NotFull`.
- **Kernel Resources**: Shows system-level semaphores (Database, Printer).

Test Cases & Concepts

Test Case 1: Round Robin Scheduling

Concept: Time Slicing. The CPU gives each thread a small unit of time (Quantum) before switching to the next.

Steps:

1. Set **CPUs** to `1`.
2. Set **Threads** to `3` in the "Create Process" panel.
3. Click **Spawn Process**.
4. Click **Play**.
5. **Observation:** You will see T1, T2, and T3 cycling through the CPU. The logs will show "Quantum

Expired" events.

Test Case 2: Multithreading - Many-to-One Blocking

Concept: In the Many-to-One model, many user threads map to 1 Kernel Thread (LWP). If a user thread makes a blocking *System Call* (like waiting for the Database), the **entire process** blocks because the single LWP is blocked.

Steps:

1. **Reset** the system.
2. Set **Threading Model** to `Many-to-One`.
3. Set **CPUs** to `2`.
4. Spawn a Process with **2 Threads** (T1, T2).
5. Wait for T1 to run on a CPU.
6. **Hover over T1** and click **"Req DB"**.
7. **Observation:** T1 moves to the Blocked Queue (System Block). Even though CPU 2 is free and T2 is Ready, T2 **will not run**. The logic prevents dispatch because the Process's single LWP is held by T1.

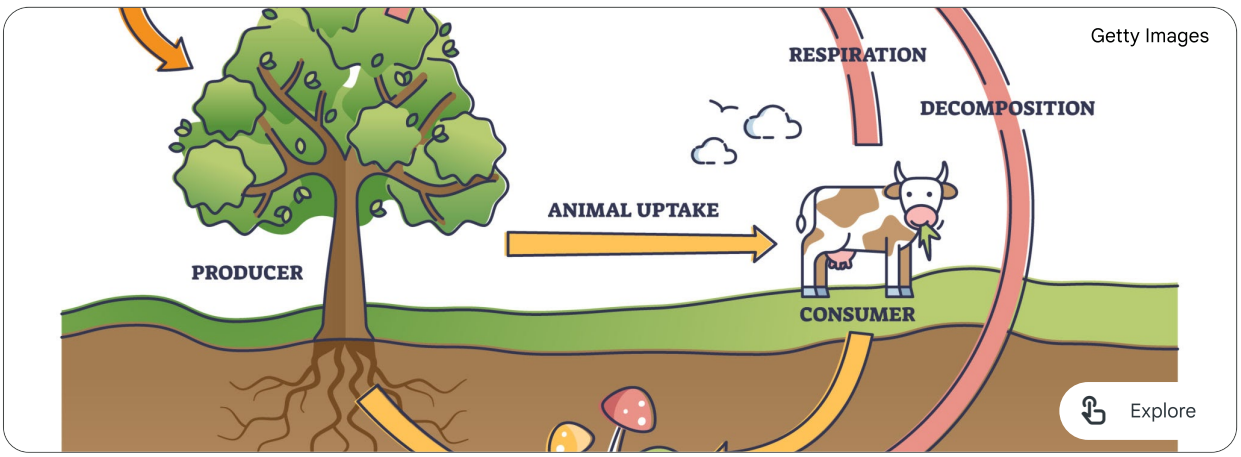
Test Case 3: One-to-One Concurrency

Concept: Each user thread has its own LWP. One blocking does not affect others.

Steps:

1. Repeat the steps above but set **Threading Model** to `One-to-One`.
2. Have T1 request "Req DB" (Database).
3. **Observation:** T1 blocks, but **T2 immediately starts running** on the other CPU (or the same one). This demonstrates true concurrency capability.

Test Case 4: Producer-Consumer (Monitor Logic)

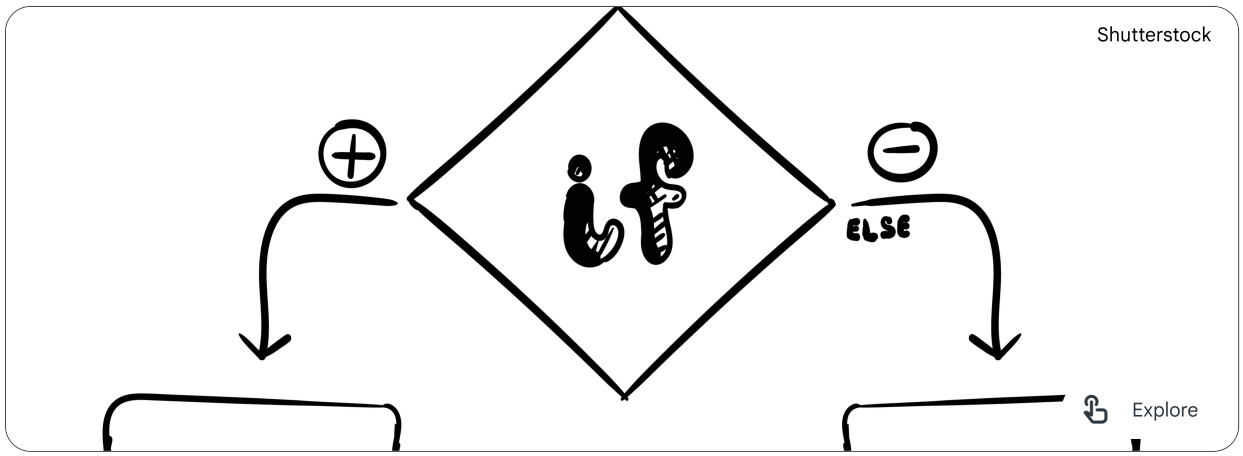


Concept: Monitors provide high-level synchronization. Threads use `Wait` (sleep until condition) and `Signal` (wake up sleeper).

Steps:

1. **Reset**. Set **CPUs** to `2` . Spawn Process (2 Threads).
2. **T1 (Producer):**
   - Hover T1 -> Click **"Enter Monitor"**. (T1 acquires Lock).
   - Click **"Sig Empty"** (Simulating putting item in buffer).
   - Click **"Exit"**.
3. **T2 (Consumer):**
   - Hover T2 -> Click **"Enter Monitor"**.
   - Click **"Wt Empty"**.
   - **Observation:** T2 releases the lock and moves to the "NotEmpty" Condition Variable Queue. It is now sleeping.
4. **Wake Up:**
   - Wait for T1 to run again -> **"Enter Monitor"**.
   - Click **"Sig Empty"**.
   - **Observation:** T2 moves from the CV Queue to the Entry Queue (Mesa Semantics), ready to re-acquire the lock.

Test Case 5: Deadlock



Concept: Circular Wait. T1 holds Resource A, wants B. T2 holds B, wants A.

Steps:

1. Spawn T1 and T2.
2. T1 runs -> **"Req DB"** (Acquired).
3. T2 runs -> **"Req Prn"** (Printer Acquired).
4. T1 runs -> **"Req Prn"** (Blocked - Printer held by T2).
5. T2 runs -> **"Req DB"** (Blocked - Database held by T1).

6. **Result:** Both threads are stuck in the Blocked Queue forever. Check "Kernel Resources" panel to see the circular dependency visually.