CLASS ASSESEMENT – 2

(CSE 316)

OPERATING SYSTEMS PROJECT

Submitted by: Name: S Harshavardhan

Registration number: 12300818

Roll No: 19

Section: K23TG

Submitted to: Akash Pundir

LOVELY PROFESSIONAL UNIVERSITY

Multithreading Models Simulator in Python

1. Introduction

This project is a Python-based simulation of three multithreading models: Many-to-One, One-to-Many, and Many-to-Many. It uses Python's threading module for thread management and Tkinter for creating a graphical user interface (GUI). The purpose is to visualize how threads operate under different multithreading architectures, helping users understand how synchronization mechanisms influence execution.

2. Tools and Libraries Used

- threading: For creating and controlling threads.
- time and random: To introduce delays and simulate variable execution times.
- tkinter: To build GUI components (e.g., canvas, buttons, input fields).
- Semaphore and Lock: To handle thread synchronization, restricting access to shared resources.

3. GUI Design

The GUI includes:

- A Text widget: Displays log messages that indicate the status of each thread.
- A Canvas widget: Visualizes thread operations with colored circles.
- Entry box: Accepts the number of threads the user wants to simulate.
- Buttons: Allow the user to run any of the three multithreading models.

4. Thread Drawing Functionality

Each thread is drawn as a circle using canvas.create_oval() with a corresponding text label at its center. The threads are placed at specific Y-coordinates to separate them visually by model (top for Many-to-One, middle for One-to-Many, bottom for Many-to-Many).

5. Many-to-One Model

- Threads are executed sequentially using a Lock (resource_lock).
- Only one thread accesses the critical section at a time.
- Simulates a situation where all user threads are mapped to one kernel thread, blocking parallel execution.
- Thread activity is logged and shown on canvas with a blue circle.

6. One-to-Many Model

- No synchronization primitives are used.
- Each thread operates concurrently, simulating direct mapping of user threads to multiple kernel threads.
- Threads are visualized in green on the canvas and log their start and end events.
- Provides maximum concurrency (subject to Python's GIL).

7. Many-to-Many Model

- Uses a Semaphore with value 2 to limit the number of concurrently running threads.
- Simulates the scenario where a number of user threads are mapped onto a limited number of kernel threads.
- Only 2 threads can run concurrently, others must wait.
- Threads are shown in red on the canvas, and their access/release of the resource is logged.

8. Logging and Responsiveness

- The log_activity() function appends messages to the Text widget and auto-scrolls to the newest message.
- The GUI remains responsive due to the use of background threads that handle simulation logic separately from the main GUI thread.

9. User Interaction Flow

1. User inputs the number of threads.

2. Clicks a button to choose the model.

3. GUI starts a background thread to simulate the model.

4. Threads are visualized and logged in real time.

10. Educational Purpose

This simulator provides an educational visualization of abstract OS concepts like thread scheduling, concurrency, resource access control, and synchronization, making it ideal for students learning Operating Systems or Python multithreading.

Module 1: GUI Design and Thread Visualization

Done by: S Harshavardhan

Duties:

- GUI Layout Creation using Tkinter:
  - Text widget for logging thread operations.
  - Canvas for displaying threads as circles.
  - Entry fields for user input.
  - Buttons to initiate threading models.

- Functionality Implementation:
  - draw_thread(x, y, color, text): Draws a colored circle on the canvas with text.

- Log Management:
  - log_activity(): Tracks and prints thread status in real time.

Module 2: Many-to-One Model and Synchronization

Mechanisms Done by: Aditya Kumar Singh

Responsibilities:

- Apply Many-to-One model where multiple user threads map to one kernel-level thread using a Lock.
  - Obtain Lock before execution.
  - Sleep to simulate work.
  - Release Lock after work.

- Log thread activity as they start, finish, and release the lock.
- Use Lock for mutual exclusion and thread safety

Module 3: One-to-Many and Many-to-Many Models

Done by: Keshav Yadav

Responsibilities:

One-to-Many Model:

- Each user-level thread gets its own kernel thread.

- Threads execute in parallel (asynchronously).

- Log thread starts and end.

Many-to-Many Model:

- Several user threads share a limited number of kernel threads.

- Use a Semaphore (value 2) to restrict concurrency.

- Each thread must acquire the semaphore, simulate work, and release it.

- Log acquire, execute, and release actions.