UCSD CSE WES 237A

Winter 2026

Multithreading - Dining Philosophers on PYNQ-Z2

(Due: 01 /29 /26)

Demonstration Date : 01 /29 /26 Group#: 7

Members: ***Benediction Bora*, *Gabriel Martinez***

INSTRUCTOR / TUTOR/

Nadir Weibel, Pushkal Mishra

Self-test Report Demo Reviewer

Name : ***Benediction Simeons***

Working / Not working Demo score Report score

Part1: _____ _____ _____: Executed Successfully

Part2: _____ _____ _____ : Executed Successfully

Part3: _____ _____ _____ : Executed Successfully

Part4: _____ _____ _____ : Executed Successfully

Part5: _____ _____ _____: Executed Successfully

Part6: _____ _____ _____: Executed Successfully

Video Demo Link: Video Demo Youtube

GitHub Repo: Asignment1 - WES237A GitHub Repo

TOTAL Score: _____

Dining Philosophers on PYNQ: Workflow and Design Report

This report documents the workflow, design decisions, debugging process, and final implementation of the Dining Philosophers problem on the PYNQ-2 platform using Python threading, locks, LEDs, and push buttons.

Top-Down Design Methodology

The assignment followed a top-down design approach. The overall goal was to implement the Dining Philosophers problem with visible LED feedback, deadlock prevention, and safe termination using push buttons. This was broken down into smaller, testable components: the system was built as a single component to exectute fork synchronization, beginning with thread management, then philosopher behavior was added, LED control was then added to better visualize fork acquisition, and lastly termination logic.

Development and Testing

Development began with basic LED control to ensure correct hardware interaction on the PYNQ board. Each LED was tested independently, including the RGB LED used for the fifth philosopher. Next, threading was introduced with simple worker functions to validate concurrency behavior. Forks were modeled as mutex locks and tested for correct acquisition and release.

Deadlock Prevention Strategy

Deadlock was avoided by enforcing a global ordering on fork acquisition. Each philosopher acquires the two adjacent forks in ascending order based on their object IDs. This eliminates circular wait conditions and ensures system-wide progress. An iterative testing process of building and flashing the PYNQ board while observing and adjusting the timing durations was also implemented to tackle deadlocks.

Timing Design (Eating, Napping, Thinking/waiting)

Timing values were carefully chosen to ensure fairness and avoid starvation. Napping durations were intentionally longer than eating durations to reduce contention for forks in PartA2.1. Eating durations were kept short to release resources quickly, while thinking/waiting delays provided additional spacing between fork requests. Randomized timing further reduced synchronization patterns that could lead to starvation and other non deterministic behaviors.

Troubleshooting and Debugging

Several challenges were encountered during development. Initial issues included invalid LED indexing, blocking behavior due to long sleep calls inside critical sections, and thread termination problems. These were resolved by separating LED logic for the RGB LED, minimizing lock hold times, and introducing a shared threading event for clean shutdown. Some other functions such as *blink_led()*, *led_off()* and *all_off()* were left unused because they were causing some unknown and unwanted behaviors. Extensive print-based tracing and isolated testing of each subsystem enabled rapid diagnosis and correction.

Termination via Push Buttons

A dedicated monitoring thread *btn_thread* targeted for the *kill_switch()* function, continuously polls the four PYNQ push buttons. Upon detecting any button press, a global stop event is set, causing all philosopher threads to exit their execution loops cleanly. All LEDs are turned off as part of the shutdown sequence. A challenge with push buttons was trying to add button termination without adding another thread and attempting to use the *asyncio* library as in previous labs and assignments.

Conclusion

This assignment successfully demonstrates a practical, hardware-integrated solution to the Dining Philosophers problem. By combining concurrency techniques with embedded system constraints, the final design achieves deadlock-free operation, clear visual feedback on the LEDs, and safe termination. The iterative debugging process—particularly the adjustment of timing logic and the removal of the unnecessary recursive function helped in quickenning code execution and highlighted the importance of simple and clean code in multithreaded systems. Overall, the assignment strengthened my understanding of multithreading, synchronization, and real-time behavior in an embedded Python environment.