Philosophers Project: 8-Day Comprehensive Plan (5h/day)

Day 1: Core Concepts & Project Understanding (5h)

Hour 1-2: Dining Philosophers Problem

- Study the classical dining philosophers problem in depth (60 min)
- Watch tutorials/videos explaining the problem and solutions (60 min)

Hour 3-4: Threading & Synchronization Fundamentals

- Learn about threads vs processes (30 min)
- Study thread creation, joining, and detaching (30 min)
- Understand mutex operations (initialization, locking, unlocking) (30 min)
- Study race conditions with practical examples (30 min)

Hour 5: Project Specification Analysis

- Read the project specification thoroughly, multiple times (30 min)
- Break down the requirements into technical tasks (30 min)

Day 2: Practical Threading & Environment Setup (5h)

Hour 1-2: Hands-on Threading Practice

- Write several simple multithreaded programs (60 min)
- Practice creating/joining threads with different parameters (30 min)
- Practice mutex implementation with shared resources (30 min)

Hour 3: Deadlock Study

- Learn about the four conditions for deadlock (20 min)
- Study common deadlock prevention techniques (20 min)
- Analyze how deadlocks apply to dining philosophers (20 min)

Hour 4: Project Structure Planning

- Sketch the program architecture on paper (30 min)
- Define data structures needed for philosophers and forks (30 min)

Hour 5: Development Environment

- Set up Git repository with proper structure (20 min)

- Create initial Makefile with required rules (20 min)
- Set up header files with basic structure (20 min)

Day 3: Time Management & Data Structures (5h)

Hour 1-2: Time Management Understanding

- Study gettimeofday() function thoroughly (30 min)
- Learn how to calculate precise time differences in milliseconds (30 min)
- Practice with time functions in small test programs (60 min)

Hour 3-4: Data Structure Implementation

- Implement philosopher structure with all needed attributes (60 min)
- Implement fork/mutex structure (30 min)
- Create initialization functions for all structures (30 min)

Hour 5: Argument Parsing & Validation

- Implement robust argument parsing (30 min)
- Add error handling and validation for all inputs (30 min)

Day 4: Thread Management & Basic Functions (5h)

Hour 1-2: Philosopher Thread Management

- Implement thread creation for philosophers (60 min)
- Create the basic structure of the philosopher routine function (60 min)

Hour 3: State Management

- Implement state change functions (30 min)
- Create thread-safe logging mechanism with timestamps (30 min)

Hour 4-5: Fork Handling

- Implement mutex initialization for forks (30 min)
- Create functions for taking forks with deadlock prevention (60 min)
- Implement fork release functions (30 min)

Day 5: Philosopher Actions & Monitoring (5h)

Hour 1-2: Philosopher Core Actions

- Implement eating function with precise timing (60 min)

- Implement sleeping function with precise timing (30 min)
- Implement thinking function (30 min)

Hour 3-4: Death Monitoring

- Design detailed monitoring mechanism (40 min)
- Implement death checker with accurate timing (50 min)
- Ensure thread-safe access to last meal times (30 min)

Hour 5: Simulation Control

- Implement simulation start function (30 min)
- Add simulation termination logic (including meals counter) (30 min)

Day 6: Synchronization & Testing (5h)

Hour 1-2: Deadlock Prevention Implementation

- Analyze your code for potential deadlocks (40 min)
- Implement even/odd fork taking strategy or similar solution (50 min)
- Test the deadlock prevention with a few philosophers (30 min)

Hour 3: Edge Cases

- Handle single philosopher case (20 min)
- Handle maximum number of philosophers (20 min)
- Implement meal counting termination condition (20 min)

Hour 4-5: Initial Testing

- Test with various numbers of philosophers (30 min)
- Test with different timing parameters (30 min)
- Debug and fix initial issues (60 min)

Day 7: Debugging & Refinement (5h)

Hour 1-2: Resource Management

- Implement proper thread joining and cleanup (40 min)
- Ensure proper mutex destruction (30 min)
- Check for memory leaks with Valgrind (50 min)

Hour 3-4: Race Condition Testing

- Test with Helgrind for race conditions (40 min)
- Fix any race conditions found (50 min)
- Double-check mutex usage throughout code (30 min)

Hour 5: Death Detection Refinement

- Ensure death is detected and reported within 10ms (30 min)
- Test death detection with various timing configurations (30 min)

Day 8: Final Testing & Submission (5h)

Hour 1-2: Stress Testing

- Test with large numbers of philosophers (30 min)
- Test with various time parameters (30 min)
- Test with minimum and maximum allowed values (30 min)
- Test the "must eat X times" functionality thoroughly (30 min)

Hour 2-3: Code Optimization & Norm Compliance

- Review code for optimizations (30 min)
- Check code against norm requirements (30 min)
- Fix any norm issues (30 min)

Hour 4: Documentation

- Add comments to critical sections (30 min)
- Write a brief README explaining your approach (30 min)

Hour 5: Final Review & Submission

- Final code review (25 min)
- Final test run with various configurations (25 min)
- Prepare and submit to repository (10 min)

Key Implementation Concepts

Thread Synchronization

- Use mutexes to protect shared resources (forks, last meal time)
- Be careful about locking order to prevent deadlocks
- Remember that printf() is not thread-safe - protect your logging

Deadlock Prevention Strategies

- Even-numbered philosophers grab right fork first, then left
- Odd-numbered philosophers grab left fork first, then right
- Alternatively, implement a "waiter" approach with a global mutex
- Consider using a hierarchy for fork acquisition

Accurate Time Management

- Store simulation start time and calculate all times relative to this
- When checking if a philosopher died:
  - Current time - last meal time > time_to_die
- Use a separate thread or regular checks to monitor deaths

Common Issues to Watch For

- Race conditions when checking/updating philosopher status
- Improper mutex locking/unlocking leading to deadlocks
- Incorrect time calculations causing premature deaths
- Failure to clean up resources (threads, mutexes) properly
- Death detection that's too slow (must be within 10ms)