# OS Project

Mid-Semester Progress Presentation

# ThreadMentor: The Dining Philosophers Problem

# Introduction

- The Dining Philosophers Problem was introduced by E. W. Dijkstra as a classic example of synchronization in concurrent programming.
- It involves five philosophers who alternate between thinking and eating, with five chopsticks shared between them.
- The challenge: ensuring safe access to shared resources (chopsticks) while avoiding deadlock and starvation.

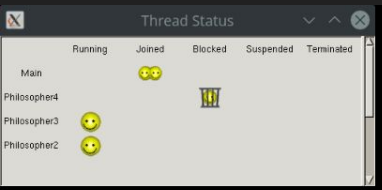
# Project Goals

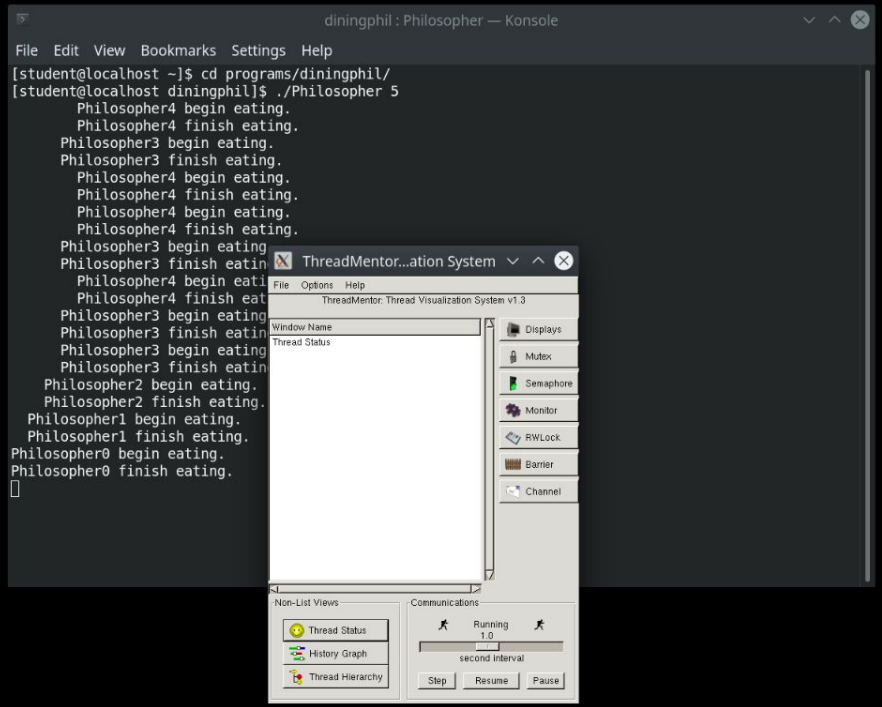
- Implement a multi-threaded solution for the Dining Philosophers Problem using ThreadMentor.
- Utilize mutex locks to synchronize access to shared chopsticks.
- Analyze potential issues like deadlock and starvation, and explore possible solutions.

# MUTEX Solution

https://pages.mtu.edu/~shene/NSF-3/e-Book/MUTEX/TM-example-philos-1.html

# Program run


Thread Status
Running
Joined
Blocked
Suspended
Terminated
Main
Philosopher4
Philosopher3
Philosopher2


diningphil : Philosopher — Konsole
File Edit View Bookmarks Settings Help
[student@localhost ~]$ cd programs/diningphil/
[student@localhost diningphil]$ ./Philosopher 5
Philosopher4 begin eating.
Philosopher4 finish eating.
Philosopher3 begin eating.
Philosopher3 finish eating.
Philosopher4 begin eating.
Philosopher4 finish eating.
Philosopher4 begin eating.
Philosopher4 finish eating.
Philosopher3 begin eating.
Philosopher3 finish eating.
Philosopher2 begin eating.
Philosopher2 finish eating.
Philosopher1 begin eating.
Philosopher1 finish eating.
Philosopher0 begin eating.
Philosopher0 finish eating.
ThreadMentor...ation System
File Options Help
ThreadMentor: Thread Visualization System v1.3
Window Name
Thread Status
Displays
Mutex
Semaphore
Monitor
RWLock
Barrier
Channel
Non-List Views
Thread Status
History Graph
Thread Hierarchy
Communications
Running
1.0
second interval
Step
Resume
Pause

# Implementation Details

# Implementation Details

## Thread Management

- Each philosopher is represented as a separate thread.
- The Philosopher class extends the Thread class, defining individual behaviors.
- The ThreadFunc() method controls the thinking-eating cycle.

# Implementation Details

## Resource Synchronization

- Each chopstick is protected by a mutex lock (Mutex *Chopstick[PHILOSOPHERS]).
- A philosopher locks their left chopstick first, then the right chopstick before eating.
- After eating, both chopsticks are released.

# Implementation Details

## Main Program Execution

- Initializes mutex locks for chopsticks.
- Creates and starts five philosopher threads.
- Ensures all threads complete execution using Join().

# Challenges and Observations

# Challenges and Observations

## Deadlock Possibility

- If all philosophers pick up their left chopstick at the same time, a circular wait occurs.
- None can pick up their right chopstick, leading to deadlock.

# Challenges and Observations

## Starvation Risk

- Faster philosophers can monopolize chopsticks, preventing slower ones from eating.
- Leads to an unfair system where some philosophers never get a chance to eat.

# Possible Solutions

# Possible Solutions

## Avoiding Deadlock:

- Enforce an ordering rule (e.g., the last philosopher picks up their right chopstick first).
- Use a waiter (semaphore) to allow only four philosophers to sit at a time.

## Preventing Starvation:

- Use a fair scheduling policy, ensuring no philosopher waits indefinitely.
- Introduce timeouts when acquiring locks.

# Next Steps

- Code Solution: Focus on deep understanding of code solution.
- Documentation & Report Submission: Complete the final team report with proper citations.
- Presentation Preparation: Refine slides, rehearse the presentation, and anticipate possible questions.

# Conclusion

- The project successfully implements multi-threaded synchronization using ThreadMentor.
- Challenges like deadlock and starvation require strategic solutions for optimal performance.
- Through ThreadMentor visualization, we gained deeper insights into process interactions and performance.
- This project strengthened our understanding of operating system concepts and practical implementation of thread synchronization.