

Role Play : Dining Philosophers Problem

Subject : Operating Systems Subject

Characters :

1. Philosopher 1 (P1)
2. Philosopher 2 (P2)
3. Philosopher 3 (P3)
4. Philosopher 4 (P4)
5. Philosopher 5 (P5)

Objective:

Each philosopher wants to eat, but they need to share the limited resources (chopsticks).

Script:

(The scene is set in a dining room with a circular table, 5 plates, and 5 chopsticks. Each philosopher is seated at a plate, with a chopstick to their left and right.)

P1: (thinking) I need to eat. I'll pick up my left chopstick. (picks up left chopstick)

P2: (thinking) I need to eat. I'll pick up my left chopstick. (picks up left chopstick)

P3: (thinking) I need to eat. I'll pick up my left chopstick. (picks up left chopstick)

P4: (thinking) I need to eat. I'll pick up my left chopstick. (picks up left chopstick)

P5: (thinking) I need to eat. I'll pick up my left chopstick. (picks up left chopstick)

(All philosophers have one chopstick each. Now, each philosopher tries to pick up the right chopstick.)

P1: (trying to pick up right chopstick) I need my right chopstick! (realizes P2 has it) Oh no, P2 has it!

P2: (trying to pick up right chopstick) I need my right chopstick! (realizes P3 has it) Oh no, P3 has it!

P3: (trying to pick up right chopstick) I need my right chopstick! (realizes P4 has it) Oh no, P4 has it!

P4: (trying to pick up right chopstick) I need my right chopstick! (realizes P5 has it) Oh no, P5 has it !

P5: (trying to pick up right chopstick) I need my right chopstick! (realizes P1 has it) Oh no, P1 has it !

(The philosophers are deadlocked. No one can eat.)

Goals:

1. Prevent Deadlock: Find a way to avoid the deadlock situation.
2. Ensure Mutual Exclusion: Ensure that only one philosopher can pick up a chopstick at a time.
3. Avoid Starvation: Ensure that each philosopher gets a chance to eat.

Solutions:

1. Asymmetric Solution : One philosopher picks up the right chopstick first, while the others pick up the left chopstick first.
2. Semaphores: Use semaphores to synchronize access to chopsticks.
3. Monitor: Use a monitor to manage the chopsticks and ensure mutual exclusion.

Discussion:

- How can the philosophers avoid deadlock?
- What are the advantages and disadvantages of each solution?
- How can the philosophers ensure fairness and prevent starvation?

This role-play demonstrates the Dining Philosophers Problem and encourages discussion on synchronization, deadlock prevention, and resource allocation in operating systems.

Roleplay Activity - Waterfall Model

Software Development Life Cycle :

The Waterfall Model is one of the earliest models used in software development. It follows a linear and sequential approach, meaning each phase must be completed before moving to the next. There's no overlap between phases.

Phases of Waterfall Model:

1. Requirement Analysis: Collect all requirements from the client.
2. System Design: Design the architecture and system plan based on requirements.
3. Implementation (Coding): Developers write code based on the design.
4. Testing: Testers check for bugs or issues in the software.
5. Deployment: The product is delivered to the client.
6. Maintenance: Support and updates after deployment.

Roleplay Setup:

Characters:

- Client – explains their requirements.
- Project Manager – plans the stages and assigns tasks.
- System Analyst – gathers requirements and prepares design documents.
- Designer – creates system and UI design.
- Developer – codes the software.
- Tester – tests and reports bugs.
- Maintenance Engineer – handles post-deployment issues.

Roleplay Script (Example Flow):

Scene 1: Requirement Gathering

- Client: I want an online food delivery app that shows nearby restaurants and allows online payment.
- System Analyst: Okay, we'll note all your requirements and prepare a Software Requirements Specification (SRS).
- (They document everything before moving to the next phase.)

Scene 2: System Design

- Project Manager: Now that we have all requirements, let's design the system architecture.
- Designer: I'll create the database design and UI mockups.
- (They show sample designs and discuss them.)

Scene 3: Implementation

- Developer: Using the design, I'll start coding the login and restaurant listing modules.
- (The developer pretends to write code and explains briefly.)

Scene 4: Testing

- Tester: I found a bug — the payment gateway crashes when entering invalid data.
- Developer: Thanks, I'll fix that in the next build.

Scene 5: Deployment

- Project Manager: The app is ready for release.
- Client: Great! I'll start using it in my business.

Scene 6: Maintenance

- Maintenance Engineer: We received feedback that the app needs a new feature: order tracking.
- Project Manager: We'll plan this for the next update.

Learning Outcome:

- Understand the sequential nature of the Waterfall Model.
- Learn team roles in each SDLC phase.
- Realize the limitations (no flexibility for changes once a phase is done).

Roleplay – Bubble Sort

Topic: Bubble Sort (Sorting Algorithm)

Bubble Sort is a simple comparison-based sorting algorithm. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted. It's called 'Bubble Sort' because smaller elements 'bubble up' to the top of the list.

Steps in Bubble Sort:

1. Start from the first element in the list.
2. Compare the current element with the next element.
3. If the current element is greater than the next one, swap them.
4. Move to the next element and repeat the process for the entire list.
5. Repeat all steps until no swaps are needed — the list is now sorted.

Role play Setup:

Characters:

- Teacher – explains the concept and guides the process.
- Students (as Numbers) – each student represents a number in the list.
- Observer – watches and notes how sorting happens step-by-step.

Role play Script (Example Flow):

Scene 1: Introduction

- Teacher: Today we'll demonstrate the Bubble Sort algorithm using a roleplay.
- Teacher: Each of you represents a number: 5, 3, 8, 2, 7.
- Observer: Let's see how these numbers will get sorted in ascending order.

Scene 2: First Pass

- Teacher: Compare 5 and 3 — since $5 > 3$, swap them.
- Students 5 and 3 swap positions.
- Teacher: Now compare 5 and 8 — no swap needed.
- Teacher: Next, compare 8 and 2 — swap because $8 > 2$.
- Continue until the end of the first pass. The largest number 'bubbles' to the end.

Scene 3: Second Pass

- Teacher: Start again from the beginning — compare 3 and 5, no swap.
- Compare 5 and 2 — swap because $5 > 2$.
- Continue this process until all elements are sorted."

Scene 4: Completion

- Teacher: Now the list is sorted: 2, 3, 5, 7, 8.
- Observer: The sorting happened through repeated comparisons and swaps.

Learning Outcome:

- Understand the working of the Bubble Sort algorithm through an interactive activity.
- Recognize how repeated comparisons and swaps lead to sorting.
- Learn teamwork and sequencing in problem-solving.

Interactive virtual classrooms :





