The Hong Kong Polytechnic University

COMP2432 Operating Systems

Project **B**laze:
Medical Care Robot Coordination System

OS Focus — Core Concurrency Concepts

Academic Year 2025–2026

Contents

Chapter 1

Project Overview

1.1 Scenario

Modern hospitals deploy autonomous robots for delivery, disinfection, and surgical assistance. In this project, you will build a lightweight OS core that coordinates these robots safely and efficiently. The emphasis is on demonstrating the same core OS concepts as Project A: concurrency, synchronization, and safe coordination.

1.2 Learning Outcomes (Core Concepts)

By completing Project B, students should demonstrate:

- **Concurrency control**: safe access to shared state with threads
- **Synchronization**: preventing race conditions and inconsistent state
- **Coordination**: organizing multiple worker threads with clear ownership

1.3 What You Will Build (Minimal Scope)

Implement a simple coordination layer with these three parts:

1. **Task queue**: store incoming robot tasks and allow robots to fetch tasks safely
2. **Zone access control**: prevent two robots from occupying the same zone at the same time
3. **Health monitor**: track robot heartbeats and mark missing robots as offline

Keep the design minimal. You are not required to implement preemption, deadlock prevention, or complex scheduling policies. Focus on correctness and safe concurrency.

1.4 Demonstration Requirements

Your demo must show:

- Multiple robots concurrently requesting tasks
- Safe access to shared zones (no two robots in the same zone)
- A robot timing out and being marked offline

Chapter 2

Deliverables

2.1 Source Code

Submit the implementation as a Cargo project:

- Must compile with `cargo build --release`
- Write tests with appropriate coverage and pass all tests (`cargo test`)
- Clear project structure with appropriate modules
- Reasonable commit history showing progress

2.2 Written Report

The report must follow this structure with specified word counts for each section:

2.2.1 Required Sections

1. **Abstract** (800–1100 characters):
    - Concise summary of the project
    - Main challenges and solutions
    - Key results
2. **Introduction** (300–500 words):
    - Problem statement and motivation
    - Project objectives
    - Overview of approach
3. **Related Works** (400–700 words):
    - Survey of concurrency control and synchronization techniques
    - Task queue and work scheduling approaches
    - Resource coordination and mutual exclusion patterns
    - Comparison with your approach
4. **Implementation** (700–1000 words):
    - Architecture diagram showing all components
    - Task queue design and thread-safe access implementation
    - Zone access control mechanism (preventing concurrent access)
    - Health monitor implementation (heartbeat tracking)

- Synchronization approach (choice of Mutex, RwLock, or other primitives)
- Code snippets for critical sections

5. **Benchmark** (500–700 words):
    - Test methodology
    - Correctness verification (no race conditions, proper synchronization)
    - Performance metrics (task throughput, zone access latency, CPU usage)
    - Scalability tests (varying number of robots and concurrent tasks)
    - Stress testing results (concurrent zone access, timeout detection)
    - Tables and graphs
6. **Discussion** (minimum 500 words):
    - Analysis of results
    - Trade-offs in design choices
    - Limitations and bottlenecks
    - Comparison of synchronization primitives
    - Lessons learned about concurrent programming
7. **Conclusion and Future Work** (minimum 300 words):
    - Summary of achievements
    - How objectives were met
    - Potential improvements
    - Future extensions
8. **References**:
    - Minimum 20 references in APA style
    - Include Rust documentation, academic papers, technical articles

2.2.2 Publication Opportunity

Outstanding reports may be selected for a publication. Students who demonstrate exceptional technical depth, clear writing, and comprehensive evaluation will be considered for this opportunity.

2.3 Video Demonstration (3 minutes)

Students must submit a 3-minute video recording demonstrating their implementation. The video format is open-ended, but it must clearly show the core OS concepts in action: concurrency, synchronization, and safe coordination.

Suggested structure (optional):

1. **System Overview** (30 seconds): Briefly explain your architecture and show the running system components.
2. **Live Demonstration** (2 minutes): Show multiple robots operating concurrently and demonstrate safe coordination of shared resources in your chosen design.
3. **Code Walkthrough** (30 seconds): Highlight critical synchronization code and one key design decision.

Technical requirements:

- Maximum duration: 3 minutes
- Compress to reasonable file size (<50MB). If unable to achieve this compression, you should submit on the YouTube platform as an unlisted video and provide the link.

- Clear audio narration
- Screen recording showing terminal output and system execution
- Submit as MP4 or similar common format

2.4 Grading Rubric

Category	Weight	Criteria
Learning Outcome A: Concurrency control	25%	Multiple threads run safely without race conditions
Learning Outcome B: Synchronization	40%	Shared state remains consistent (task queue, zones, monitor)
Learning Outcome C: Coordination	10%	Threads interact correctly and demo scenario completes
Code Quality	5%	Readable, idiomatic Rust, proper error handling
Report & Demo	20%	Clear explanation and working demonstration
Total	**100%**	