Advanced Operating System Architecture

Lab 0: Producer-Consumer Problem with Buffer

Computer Science Department

Academic Year 2023-2024

General Information

- **Release Date**: October 30, 2024
- **Submission deadline**: November 13, 2024
- **Objective**: Implement the producer-consumer problem with a buffer, at two levels of complexity

Prerequisites

- Familiarity with concurrent programming concepts (processes, shared memory)
- Knowledge of programming in C
- Basic understanding of operating system principles

1 Level A: Single Producer and Single Consumer

In this level, you will implement a simple producer-consumer problem with a single producer and a single consumer.

1.1 Problem Statement

The producer generates items and places them in a bounded buffer. The consumer takes items from the buffer and consumes them. The buffer has a fixed size, and the producer and consumer must synchronize their access to the buffer to avoid race conditions.

Tasks

1. Design the data structures for the buffer, producer, and consumer
2. Implement the producer and consumer functions, ensuring thread-safe access to the buffer
3. Test your implementation with different buffer sizes and number of items produced/consumed
4. Document your design choices and any challenges encountered

2 Level B: Multiple Producers and Multiple Consumers

In this level, you will extend the previous problem to have multiple producers and multiple consumers sharing a single buffer, using C processes and shared memory.

2.1 Problem Statement

The system now has multiple producers and multiple consumers that share a single, bounded buffer. The producers generate items and place them in the buffer, while the consumers take items from the buffer and consume them. The buffer has a fixed size, and the producers and consumers must synchronize their access to the buffer to avoid race conditions.

2.2 Required tasks

Tasks

1. Design the shared memory structure to be used by the producers and consumers
2. Implement the producer and consumer processes, ensuring thread-safe access to the shared buffer
3. Use appropriate synchronization mechanisms (e.g., semaphores, mutexes) to coordinate the processes
4. Test your implementation with different buffer sizes, numbers of producers/consumers, and numbers of items produced/consumed
5. Document your design choices, new challenges encountered, and how you addressed them

3 Submission instructions

The following GitHub repository setup is used for submitting lab work.

Tasks

1. **Create a private repository** named **"Advanced OS"**.
2. **Add a 'README.md' file** with the names of all group members.
3. **Organize work** in subdirectories (e.g., 'Lab0', 'Lab1', etc.) for each lab.
4. **Add the professor (balimou)** as a collaborator to the repository.

Warning!

- Ensure your code is thoroughly tested and documented
- Demonstrate a good understanding of synchronization primitives and inter-process communication
- Clearly explain your design choices and reasoning
- Meet all deadlines and submit deliverables on time

4 Evaluation Criteria

- Correctness and completeness of the implementations (40%)
- Quality of the documentation and explanations (30%)
- Depth of understanding of the problem and solutions (30%)