Concurrent and Real-Time Programming

Producer-Consumer Program with Dynamic Message Rate Adjustment

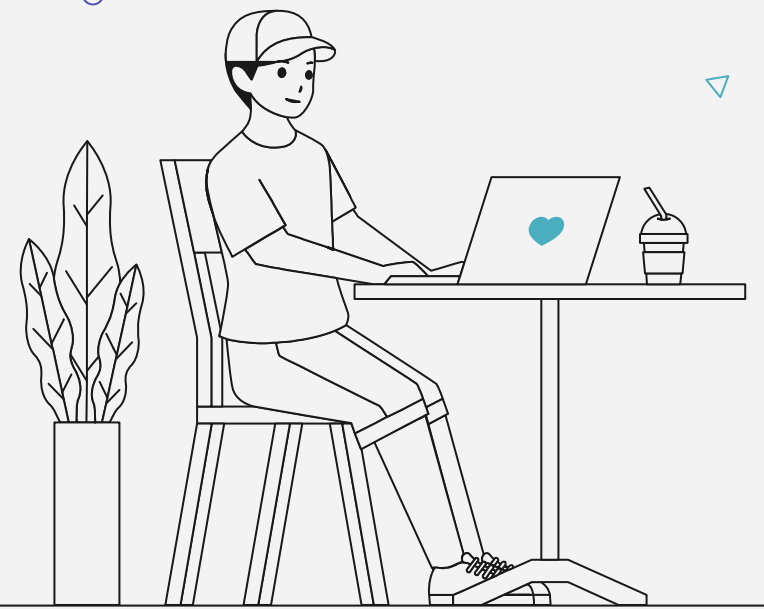
Presented by Farzad Shami – Dep. of Information Engineering – University of Padua

Table of contents

01

Introduction

Project Overview + Objective



Project Overview

- **Objective**: Design a concurrent and real-time producer-consumer system with dynamic message rate adjustment.
- **Components**:
  - Producer: Generates messages.
  - Consumer: Consumes messages at a given rate.
  - Actor: Monitors message queue length and adjusts production rate dynamically.

02

System Design

System Architecture

01

Producer

Generates messages and puts them into a shared message queue.

02

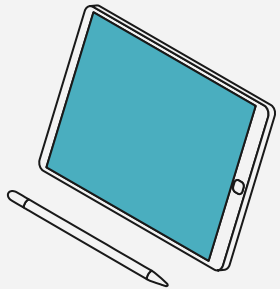
Consumer

Consumes messages from the queue at a specified rate.

03

Actor

Periodically checks queue length and adjusts production rate accordingly.



Concurrent Programming Concepts

Concurrent execution

Multiple tasks executing simultaneously.

Synchronization

Ensuring proper access to shared resources.

Communication

Facilitating communication between tasks.

Implementation Details

Producer

Generates messages at a variable rate.

Consumer

Consumes messages with a specified delay.

Actor

Monitors queue length and adjusts production rate based on threshold.

Producer

```
void *producer(void *params) {
    Parameters *p = (Parameters *) params;
    while (1) {
        sem_wait(&spaces); // Wait for space
        pthread_mutex_lock(&mutex);
        // Produce an item
        Message new_msg = produce_random_message(next_message_id++);
        messageQueue[queue_length++] = new_msg;
        printf("Produced: %d\n", new_msg.id);
        insertDataToRedis("metrics", queue_length);
        sem_post(&items); // Signal that an item is available
        pthread_mutex_unlock(&mutex);
        // Sleep according to the current production delay
        usleep(p->message_delay * 1000000);
    }
    return NULL;
}
```

Producer

It enters an infinite loop:

- It waits on the `spaces` semaphore, indicating space availability in the queue.
- It acquires the mutex lock to ensure exclusive access to shared resources.
- It produces a new message using `produce_random_message`.
- It adds the message to the queue (`messageQueue`).
- It prints a message indicating production.
- It calls `insertDataToRedis` to save the queue length to Redis.
- It signals on the `items` semaphore, indicating that an item is now available for consumption.
- It releases the mutex lock.
- It sleeps for the specified production delay.

Dynamic Rate Adjustment Algorithm

If message queue length is below threshold:

-> Increase production rate.

If message queue length is above threshold:

-> Decrease production rate.

Dynamic Rate Adjustment Algorithm

Approach I

```
(0.7) * p->message_delay
```

-> Increase production rate.

If message queue length is above threshold:

-> Decrease production rate.

Approach II

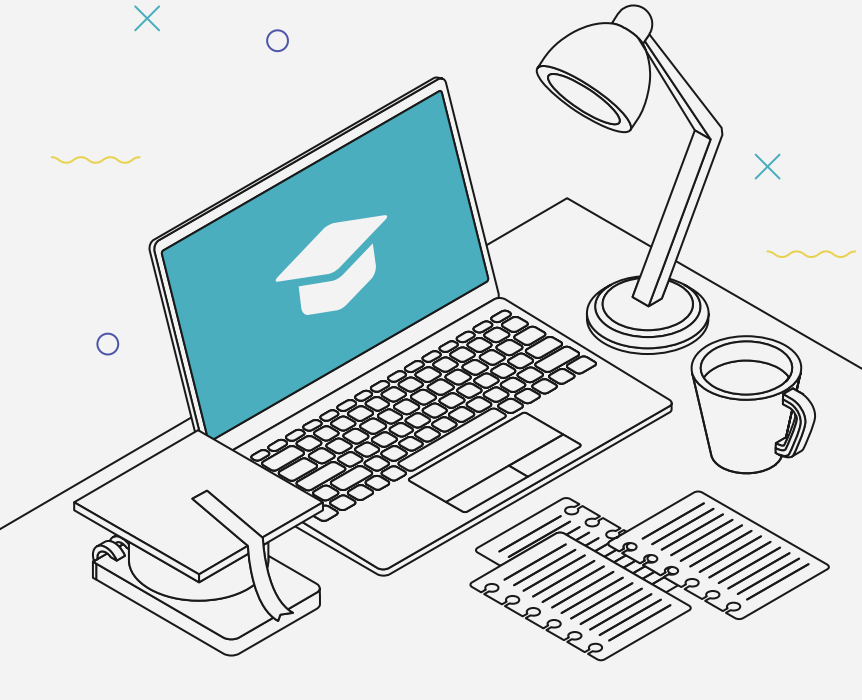
```
p->message_delay > 1 ? p->message_delay - 1 : (0.9) * p->message_delay
```

Benefits of Dynamic Rate Adjustment

Efficient resource utilization.

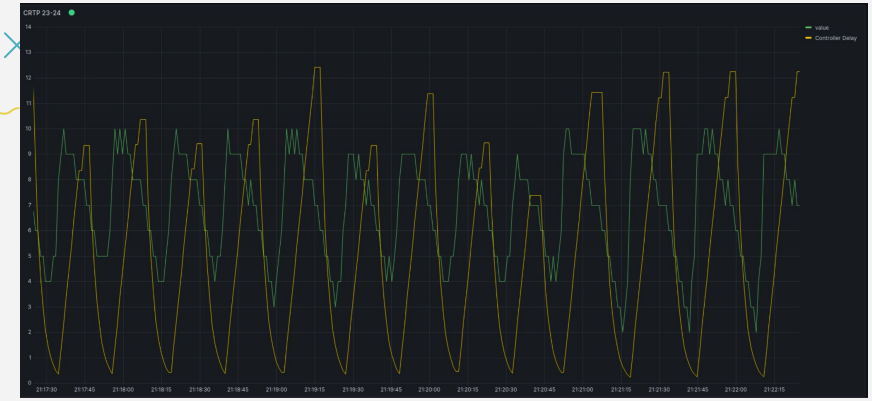
Adaptation to varying workload conditions.

Improved system responsiveness.

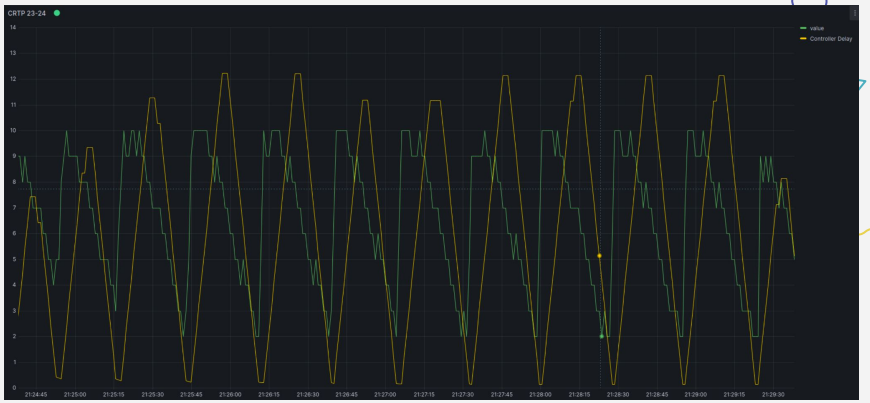


03
Analytics

Approach I

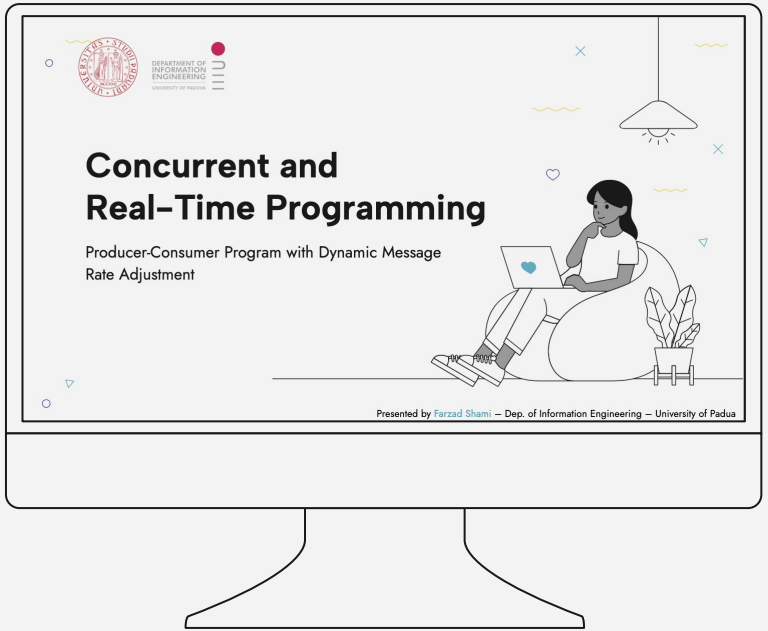


Approach II



04

Live Demo


Concurrent and
Real-Time Programming
Producer-Consumer Program with Dynamic Message Rate Adjustment
Presented by Farzad Shami – Dep. of Information Engineering – University of Padua

Thanks!

Do you have any questions?

farzadshami@studenti.unipd.it