University of Eswatini

Department of Computer Science

CSC 411 Assignment Report

Title: The Producer-Consumer Problem

By: Gina Thandolwethu – 202004162

and

Tembe Unathi – 202102300

1. Introduction

This mini-project implements the classical **Producer–Consumer Problem** using Python.
The task demonstrates the application of:

- Multithreading
- Semaphores
- Shared buffer synchronization
- Random data generation
- XML file creation and parsing
- Socket programming
- Version control using Git and GitHub

The objective is to simulate a real-world scenario where a **producer** generates student information, stores it in XML files, and places references into a **bounded buffer**. The **consumer** reads those references, consumes the XML data, processes it, and displays the results.

2. System Overview

The system consists of three major components:

2.1 The Producer

The Producer performs the following tasks:

1. Generates random **ITStudent** objects consisting of:
    - Student Name
    - 8-digit Student ID
    - Programme
    - List of Courses
    - Marks for each course
2. Wraps this data into XML format using Python
3. Saves files
4. Inserts an integer (1–10) into the **shared buffer** representing the file name.
5. Uses semaphores to ensure it **does not produce when the buffer is full**

2.2 The Consumer

The Consumer is responsible for:

1. Waiting for a value in the buffer (blocked if empty).
2. Reading the corresponding XML file.
3. Unwrapping the XML back into an **ITStudent** object.
4. Calculating:

    - Average mark
    - Pass/Fail status (pass mark ≥ 50%)
5. Printing all student details to the screen.
6. Deleting/clearing the XML file.
7. Removing the integer from the buffer.

The Consumer uses semaphores to ensure it **does not consume when the buffer is empty**.

2.3 The Shared Bounded Buffer

The buffer is implemented as a Python list with a **maximum size of 10**.

3. Implementation (Python)

ITStudent Class

Defines:

- Student name
- ID
- Programme
- Courses and marks
- Methods for calculating average and determining pass/fail

3.2 XML Wrapping and Unwrapping

The producer uses ElementTree to create XML tags and save them.
The consumer loads XML files, extracts elements, and reconstructs student objects.

4. Producer Thread:

- Waits if buffer is full
- Inserts integer → signals

Consumer Thread:

- Waits if buffer is empty
- Removes integer → signals `empty.release()`

5. Socket Programming Version

A second implementation uses **TCP sockets** to simulate producer-consumer communication over a network.

Producer (Server):

- Creates student XML as a string
- Sends the XML over a TCP socket

Consumer (Client):

- Connects to the server
- Receives XML
- Parses it
- Processes the student data

6. GitHub Version Control

GitHub was used throughout the project for:

6.1 Collaboration

Both members created GitHub accounts and accessed the shared repository.

6.2 Repository Contents

The repository contains:

- Python source code
- README.md
- Demo video
- Final PDF report

6.4 Submitted GitHub Links

- **Thandolwethu Gina's GitHub:** https://github.com/teedogina
- **Unathi Tembe's GitHub:** https://github.com/unath1
- **Project Repository:** https://github.com/teedogina/ProducerConsumerPython

7.Demo Video

A 11mins 3secs voice-over demonstration was recorded showing:

- The producer generating XML
- The consumer processing the data
- The synchronized buffer in action
- The socket programming version

8. Conclusion

This project successfully demonstrated the implementation of:

- The Producer-Consumer problem
- Thread synchronization using semaphores
- XML data handling
- Python multithreading
- Socket programming
- GitHub version control