UNIVERSITY OF ESWATINI

CSC411 – MINI PROJET

MDUDUZI COMFORT – 202004888

MSIMISI MATSE – 202003895

THE PRODUCER CONSUMER PROBLEM

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
import random
import socket
import xml.etree.ElementTree as ET
import os
import threading
import time

# Class representing student information
class ITStudent:
    def __init__(self, name, student_id, programme, courses, marks):
        self.name = name
        self.student_id = student_id
        self.programme = programme
        self.courses = courses
        self.marks = marks

# Buffer class to implement shared buffer
class Buffer:
    def __init__(self, max_size=10):
        self.max_size = max_size
        self.queue = []
        self.semaphore_producer = threading.Semaphore(max_size)
        self.semaphore_consumer = threading.Semaphore(0)
        self.mutex = threading.Lock()
    def add_to_buffer(self, item):
        self.semaphore_producer.acquire()
        self.mutex.acquire()
        self.queue.append(item)
        self.mutex.release()
        self.semaphore_consumer.release()

    def remove_from_buffer(self):
        self.semaphore_consumer.acquire()
        self.mutex.acquire()
        item = self.queue.pop(0)
        self.mutex.release()
        self.semaphore_producer.release()
        return item

# Function to generate random student information
def generate_student_info():
    name = "Student" + str(random.randint(1, 100))
    student_id = str(random.randint(10000000, 99999999))
    programme = "Programme" + str(random.randint(1, 5))
    num_courses = random.randint(3, 6)
    courses = [f"Course{str(i)}" for i in range(1, num_courses + 1)]
    marks = [random.randint(40, 100) for _ in range(num_courses)]
    return ITStudent(name, student_id, programme, courses, marks)

# Function to write student information to an XML file
def write_to_xml(student, file_path):
    root = ET.Element("Student")
    ET.SubElement(root, "Name").text = student.name
    ET.SubElement(root, "StudentID").text = student.student_id
    ET.SubElement(root, "Programme").text = student.programme
    courses_elem = ET.SubElement(root, "Courses")
    for course, mark in zip(student.courses, student.marks):
        course_elem = ET.SubElement(courses_elem, "Course")
        ET.SubElement(course_elem, "CourseName").text = course
        ET.SubElement(course_elem, "Mark").text = str(mark)

    tree = ET.ElementTree(root)
    tree.write(file_path)

# Function to read student information from an XML file
def read_from_xml(file_path):
    tree = ET.parse(file_path)
    root = tree.getroot()
    name = root.find("Name").text
    student_id = root.find("StudentID").text
    programme = root.find("Programme").text

    courses = []
    marks = []
    for course_elem in root.find("Courses"):
        courses.append(course_elem.find("CourseName").text)
        marks.append(int(course_elem.find("Mark").text))

    return ITStudent(name, student_id, programme, courses, marks)

# Function to calculate average mark and pass/fail status
def calculate_average(student):
    total_marks = sum(student.marks)
    average = total_marks / len(student.marks)
    return average, "Pass" if average >= 50 else "Fail"
```

```
# Producer function to generate student information and write to XML files
def producer(buffer):
    for i in range(1, 11):
        student_info = generate_student_info()
        file_path = f"student{i}.xml"
        write_to_xml(student_info, file_path)
        print(f"Produced: {file_path}")
        buffer.add_to_buffer(i)
        time.sleep(random.randint(1, 3))

# Consumer function to read student information from XML files and calculate results
def consumer(buffer):
    while True:
        file_num = buffer.remove_from_buffer()
        file_path = f"student{file_num}.xml"
        student_info = read_from_xml(file_path)
        os.remove(file_path)
        print(f"Consumed: {file_path}")
        average, pass_fail = calculate_average(student_info)
        print(f"Student Name: {student_info.name}")
        print(f"Student ID: {student_info.student_id}")
        print(f"Programme: {student_info.programme}")
        print("Courses and Marks:")
        for course, mark in zip(student_info.courses, student_info.marks):
            print(f"  {course}: {mark}")
        print(f"Average Mark: {average:.2f}")
        print(f"Pass/Fail: {pass_fail}")
        print("\n")
        time.sleep(random.randint(1, 3))

if __name__ == "__main__":
    buffer = Buffer(max_size=10)
    # Create producer and consumer threads
    producer_thread = threading.Thread(target=producer, args=(buffer,))
    consumer_thread = threading.Thread(target=consumer, args=(buffer,))

    # Start the threads
    producer_thread.start()
    consumer_thread.start()

    # Wait for both threads to finish
    producer_thread.join()
    consumer_thread.join()

 # Server (Producer)

# Function to generate random student information
def generate_student_info():
    name = "Student" + str(random.randint(1, 100))
    student_id = str(random.randint(10000000, 99999999))
    programme = "Programme" + str(random.randint(1, 5))
    num_courses = random.randint(3, 6)
    courses = [f"Course{str(i)}" for i in range(1, num_courses + 1)]
    marks = [random.randint(40, 100) for _ in range(num_courses)]
    return ITStudent(name, student_id, programme, courses, marks)



# Function to write student information to an XML file
def write_to_xml(student, file_path):
    root = ET.Element("Student")
    ET.SubElement(root, "Name").text = student.name
    ET.SubElement(root, "StudentID").text = student.student_id
    ET.SubElement(root, "Programme").text = student.programme
    courses_elem = ET.SubElement(root, "Courses")
    for course, mark in zip(student.courses, student.marks):
        course_elem = ET.SubElement(courses_elem, "Course")
        ET.SubElement(course_elem, "CourseName").text = course
        ET.SubElement(course_elem, "Mark").text = str(mark)

    tree = ET.ElementTree(root)
    tree.write(file_path)

if __name__ == "__main__":
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = "127.0.0.1"
    port = 12345
    server_socket.bind((host, port))

    server_socket.listen(1)
    print("Server is listening...")

    connection, address = server_socket.accept()
    print(f"Connected to {address}")

    for i in range(1, 11):
        student_info = generate_student_info()
        file_path = f"student{i}.xml"
        write_to_xml(student_info, file_path)
        print(f"Produced: {file_path}")

        connection.send(str(i).encode())  # Send the file number as a string

        time.sleep(random.randint(1, 3))
    connection.close()
    server_socket.close()



# Client(Consumer)

# Function to read student information from an XML file
def read_from_xml(file_path):
    tree = ET.parse(file_path)
    root = tree.getroot()
    name = root.find("Name").text
    student_id = root.find("StudentID").text
    programme = root.find("Programme").text

    courses = []
    marks = []
    for course_elem in root.find("Courses"):
        courses.append(course_elem.find("CourseName").text)
        marks.append(int(course_elem.find("Mark").text))

    return ITStudent(name, student_id, programme, courses, marks)

# Function to calculate average mark and pass/fail status
def calculate_average(student):
    total_marks = sum(student.marks)
    average = total_marks / len(student.marks)
    return average, "Pass" if average >= 50 else "Fail"

if __name__ == "__main__":
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = "127.0.0.1"
    port = 12345
```

```
client_socket.connect((host, port))

while True:
    file_num_str = client_socket.recv(1024).decode()

    if not file_num_str:
        break

    file_num = int(file_num_str)
    file_path = f"student{file_num}.xml"

    student_info = read_from_xml(file_path)
    os.remove(file_path)
    print(f"Consumed: {file_path}")

    average, pass_fail = calculate_average(student_info)
    print(f"Student Name: {student_info.name}")
    print(f"Student ID: {student_info.student_id}")
    print(f"Programme: {student_info.programme}")
    print("Courses and Marks:")
    for course, mark in zip(student_info.courses, student_info.marks):
        print(f"  {course}: {mark}")
    print(f"Average Mark: {average:.2f}")
    print(f"Pass/Fail: {pass_fail}")
    print("\n")

client_socket.close()
```

LINK TO GIT HUB SHARED PROJECT:

https://github.com/comfort16/Mini-Project

OR

https://github.com/comfort16/Mini-Project.git