CSC-40068 - Advanced Programming in Python. Coursework Assignment

Mr Andrew Cook

Deadline: 1pm BST 6th May 2022

Problem Description

A popular pizza company is opening a new store nearby to accept online orders for delivery. They have asked you to create a simulation for their store.

We will simplify the scenario somewhat using only a few ingredients for each type of pizza.

The store will offer three sizes of **Pizza**; Small, Medium and Large. An **Order** can contain any combination of the three sizes and may include multiple pizzas of the same size.

Orders will be provided in the form of a JSON file which will need to be read and processed appropriately (an example JSON is provided). When a chef is available they will collect an order from the order point. These should be collected in order of their **Order-ID**.

The chef will then create each pizza using ingredients collected from a storage area for each ingredient which will be periodically refilled. These shared storage areas should have a configurable size N_items.

Once all pizzas in an order have been created the order should be cooked in a **Pizza Oven** and then placed into a **Collection Queue**, after which a chef can get the next available order. Orders will be collected for delivery with a priority based upon the number of pizzas in the order.

All pizzas take 20ms to make and 10ms to cook. A delivery Driver can collect an order for delivery at most once every 50ms.

An Order with three pizzas would therefore take 60ms to make and a further 30ms to cook.

Pizzas require the following components:

- Small Pizza: 1 Dough, 1 Sauce, 2 Toppings
- Medium Pizza: 2 Dough, 1 Sauce, 3 Toppings
- Large Pizza: 3 Dough, 2 Sauce, 4 Toppings

When an order is collected it should be logged into a CSV file. This CSV should log the following for each order.

- Order-ID
- Time Accepted
- Time Collected
- Number of Small Pizzas
- Number of Medium Pizzas
- Number of Large Pizzas
- IDs for each ingredient used for the order

Once all orders from the Input JSON have been collected your simulation should end.

There are a few restrictions on the operations of the store:

- The Ingredient storage has limited space. When they are full no more items can be added until some have been taken out.
- No Pizza can be created until all ingredients are available for that Pizza.
- Pizza Orders must be collected for delivery in order of the number of Pizzas in the order.

This coursework is split into two parts, each worth half of the programming assessment marks (90% total). With a further 10% for Demonstration of your solution.

Component 1: Programming coursework. (90%)
Task 1: Sequential simulation.

The goal of this task is to simulate the operations of the store sequentially, there is one Chef and one Delivery driver. You are required to create appropriate classes to represent the Orders, Pizzas and the processes involved in the operation of the store.

Store operation will proceed as follows:

1. A chef collects an order.
2. For each pizza in an order the chef collects the ingredients from the stores and creates that pizza.
3. Once all pizzas in an order are completed the order is cooked.
4. Cooked pizzas are placed into the Collection queue.
5. A delivery driver collects the largest order from the Collection queue.
6. The Order is logged to an output CSV.

Expected task outcome

The goals for this task are:

1. Creation of suitable classes for Orders, Pizzas and Ingredients.
2. Implementation of suitable data structures and classes for the Input Queue, Ingredient Storage, Pizza Oven and Collection Queue.
3. An appropriate CSV which logs the outcome for each Order.

Assessment criteria for Task 1 (45%)

- 40% Implementation of Classes for **Orders** and **Pizzas**.
- 40% Implementation of appropriate Data Structures for Ingredient Storage, and Collection Queue.
- 20% Implementation of Order Log CSV output.

Task 2: Concurrent Simulation

Following the success of the initial store processes the Pizza company have decided to hire additional chefs and delivery drivers to increase output.

The number of **Chefs** and **Delivery Drivers** should be configurable.

This will create a more dynamic environment where multiple orders may be processed *concurrently.*

Ingredient stores cannot be refilled while a chef is accessing them. A Pizza cannot be started until all ingredients are available. Only one order can be cooked at a time.

Orders must be collected based upon how many pizzas are in the order.

You must be aware of how your data structures are managed to ensure **Mutually Exclusive Access** and **Synchronisation** where appropriate.

Note that:

- Each ingredient store can be refilled independently, but chefs cannot retrieve ingredients while the store is being refilled. Similarly the store cannot be refilled while a chef is collecting the ingredients. These stores have limited size.
- The chef will need to access multiple stores simultaneously but cannot begin production of a Pizza unless all ingredients are available (in the correct quantities).
- All chefs will require access to the Pizza Oven and Delivery Drivers will require access to the Collection queue.

Expected task outcome

The goals for this task are:

1. Management of contention of the main data structures (Ingredient stores, Order queue, Pizza Oven and Collection Queue.)
2. Correct handling of Pizza creation - A pizza may not be started unless all required ingredients are available.
3. a discussion of race conditions and liveliness.

For goal 3 you should include a short report in the form of a Docstring in the main module of this submission which discusses your approach to handling Mutually exclusive access to ingredient stores and liveliness issues for threads.

You should provide a few sentences on:

- The potential issues due to concurrent access to the Ingredient store, Pizza oven and Collection queue.
- How you have ensured that deadlock/livelock will not occur.
- Whether starvation may occur and how you have dealt with this.

Assessment criteria for Task 2 (45%)

- 40% Management of contention for Shared data structures (Order Queue, Ingredient Stores, Pizza Oven and Collection Queue).
- 40% Management of availability for all components.
- 20% Short report on Liveliness, Race conditions and Starvation.

Component 2: Code demonstration (10%)

Once your solutions have been submitted we will arrange a short demonstration of your solution either in person or via MS Teams where you will be expected to provide a short explanation of your code and justify how it meets the specified requirements. This demonstration is worth 10% of the module marks.

Guidance

Your submission should be in the form of a single zipped file containing a folder for each of Task 1 and Task 2. This zipped file should be named with your Student Number (8 digits).

Your solution for each task should be able to run independently (you should not import classes/modules from Task 1 into Task 2, however you can re-use code if required.)

You should test your solutions on the Lab computers to ensure they run correctly.

You should submit *.py* files for each component in your solution.

You can use any data structures in the standard Python installation on the Lab computers which you require (you do not have to write these data structures yourself).

The included data is provided as an example of the format orders will be received in. For thorough testing of your solution you will need to generate a much larger selection of orders to identify any unusual behaviour.

Your code should be documented appropriately, using Comments and Docstrings.

Late submission and Academic Conduct

Late submissions by a **maximum of one week** after the deadline will be capped at 50% and anything submitted after this time will receive %0.

If you require an extension to the deadline for good reasons please submit an Exeptional Circumstances form to the School.

Please be aware that you are expected to work on this coursework independently. The University takes plagiarism very seriously.

If you are uncertain about what constitutes plagiarism, please refer to the relevant section at https://www.keele.ac.uk/studentacademicconduct