Codio Activity: The Producer-Consumer Mechanism

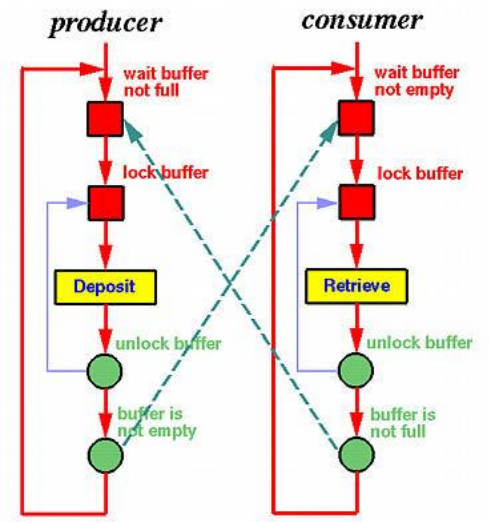
Producer/Consumer Problem (also known as the 'bounded buffer' problem):

- A 'producer' is producing items at a particular (unknown and sometimes unpredictable) rate.
- A 'consumer' is consuming the items – again, at some rate.

For example, a producer-consumer scenario models an application producing a listing that must be consumed by a printer process, as well as a keyboard handler producing a line of data that will be consumed by an application program. This is shown in the picture below (Shene, 2014).

Items are placed in a buffer when produced, so:

- Consumer should wait if there isn't an item to consume
- Producer shouldn't 'overwrite' an item in the buffer



Synchronisation is necessary because:

- If the consumer has not taken out the current value in the buffer, then the producer should not replace it with another.
- Similarly, the consumer should not consume the same value twice.

Task

Run producer-consumer.py in the provided Codio workspace (**Producer-Consumer Mechanism**), where the queue data structure is used.

A copy of the code is available here for you.

```
# code source: https://techmonger.github.io/55/producer-consumer-python/

from threading import Thread
from queue import Queue

q = Queue()
final_results = []

def producer():
    for i in range(100):
        q.put(i)



def consumer():
    while True:
        number = q.get()
        result = (number, number**2)
        final_results.append(result)
        q.task_done()



for i in range(5):
    t = Thread(target=consumer)
    t.daemon = True
    t.start()

producer()

q.join()
```

```
print (final_results)
```

Answer the following questions:

1. How is the queue data structure used to achieve the purpose of the code?

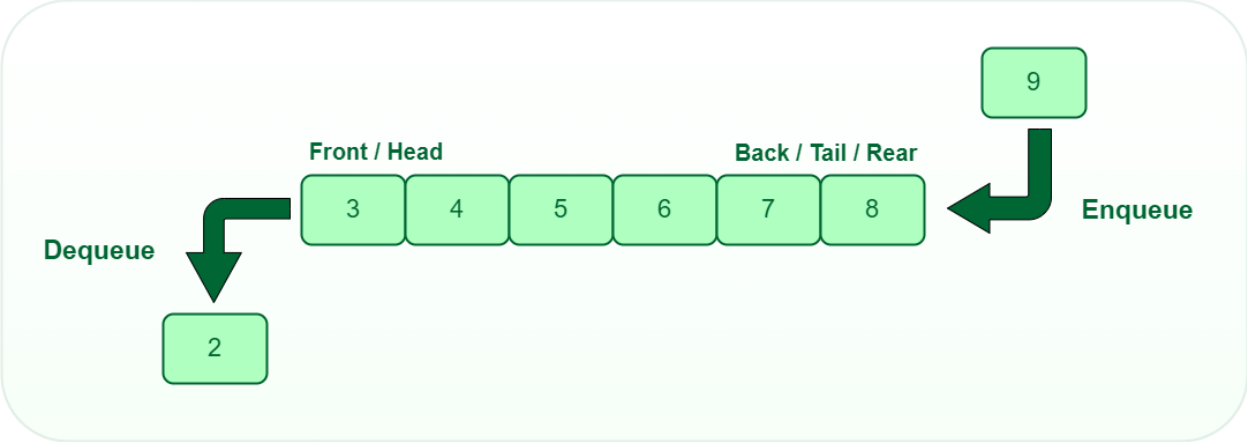


Figure 1: Queue Data Structure Example (Geeksforgeeks, 2022)

A queue is like a line of people waiting to buy tickets for a concert. The order is according to the FIFO principle (first in first out). In figure 1, the first entry (2), which is at the front, is removed first and at the same time a new number is added at the end (9).

In the case of the code above, the producer put 100 item in the queue and the consumer get the item step by step and use it.

2. What is the purpose of q.put(I)?

The purpose of the q.put() is to put an item into the queue.

3. What is achieved by q.get()?

The purpose of q.get() is to get an item which is present inside queue

4. What functionality is provided by q.join()?

The method q.join is dependent on the method task_done. When q.join is called, the programme is held from existing until the task_done method is called. When all items

make call to their respective `task_done` it sends signals to `q.join()` that all items have been processed and program can exit (Techmonger, 2018)

5. Extend this producer-consumer code to make the producer-consumer scenario available in a secure way. What technique(s) would be appropriate to apply?

Remember to record your thoughts and answers in your e-portfolio.

References:

Geeksforgeeks (2022) Queue Data Structure. Available from:

https://www.geeksforgeeks.org/queue-data-structure/ [Accessed 7 October 2022].

Techmonger (2018) Solution to Producer Consumer Problem in Python.

Available from: https://techmonger.github.io/55/producer-consumer-python/