E94106216 謝富祥 producer-consumer problem

Source code:
使用 python 3.10.11

```python
import threading
import collections
import time
import random

# buffer 大小
BUFFER_SIZE = 4

# 創建共用 buffer
buffer = collections.deque(maxlen=BUFFER_SIZE)

# 創建兩個 semaphore
empty = threading.Semaphore(BUFFER_SIZE)
full = threading.Semaphore(0)

# 創建一個 mutex lock
mutex = threading.Lock()

def producer(num):
    for i in range(1, 11):
        # 等待 buffer 有空位
        empty.acquire()
        # 鎖住，將物品放入 buffer
        mutex.acquire()
        buffer.append(i)
        print(f"Producer {num}: Produced item {i}")
        print(f"Producer {num}: Buffer: {list(buffer)}")
        print()
        # 放完釋放鎖
        mutex.release()
        # 通知消費者有新的物品可消費
        full.release()
        # 隨機睡眠一段時間
        time.sleep(random.randint(0, 5) / 10)
    print(f"*****Producer {num} finished.*****")

def consumer(num):
    for _ in range(1, 11):
        # 等待 buffer 有物品可消費
        full.acquire()
        # 鎖住，從 buffer 取出物品
        mutex.acquire()
        item = buffer.popleft()
        print(f"Consumer {num}: Consumed item {item}")
```

```
        print(f"Consumer {num}: Buffer: {list(buffer)}")
        print()
        # 拿完釋放鎖
        mutex.release()
        # 通知生產者有空位可放入新的物品
        empty.release()
        # 隨機睡眠一段時間
        time.sleep(random.randint(0, 10) / 10)
    print(f"*****Consumer {num} finished.*****")

# 創建生產者和消費者 threads 各 2 個
producer_threads = [threading.Thread(target=producer, args=(i+1,)) for
i in range(2)]
consumer_threads = [threading.Thread(target=consumer, args=(i+1,)) for
i in range(2)]

# 啟動 threads
for thread in producer_threads + consumer_threads:
    thread.start()

# 等待 threads 結束
for thread in producer_threads + consumer_threads:
    thread.join()

print("All done.")
```

demo:

```
Producer 1: Produced item 1
Producer 1: Buffer: [1]

Producer 2: Produced item 1
Producer 2: Buffer: [1, 1]

Producer 1: Produced item 2
Producer 1: Buffer: [1, 1, 2]

Consumer 1: Consumed item 1
Consumer 1: Buffer: [1, 2]

Consumer 2: Consumed item 1
Consumer 2: Buffer: [2]

Producer 1: Produced item 3
Producer 1: Buffer: [2, 3]

Producer 2: Produced item 2
```

```
Producer 2: Buffer: [2, 3, 2]

Producer 1: Produced item 4
Producer 1: Buffer: [2, 3, 2, 4]

Consumer 1: Consumed item 2
Consumer 1: Buffer: [3, 2, 4]

Producer 2: Produced item 3
Producer 2: Buffer: [3, 2, 4, 3]

Consumer 2: Consumed item 3
Consumer 2: Buffer: [2, 4, 3]

Producer 2: Produced item 4
Producer 2: Buffer: [2, 4, 3, 4]

Consumer 1: Consumed item 2
Consumer 1: Buffer: [4, 3, 4]

Producer 1: Produced item 5
Producer 1: Buffer: [4, 3, 4, 5]

Consumer 2: Consumed item 4
Consumer 2: Buffer: [3, 4, 5]

Producer 2: Produced item 5
Producer 2: Buffer: [3, 4, 5, 5]

Consumer 2: Consumed item 3
Consumer 2: Buffer: [4, 5, 5]

Producer 1: Produced item 6
Producer 1: Buffer: [4, 5, 5, 6]

Consumer 1: Consumed item 4
Consumer 1: Buffer: [5, 5, 6]

Consumer 2: Consumed item 5
Consumer 2: Buffer: [5, 6]

Producer 2: Produced item 6
Producer 2: Buffer: [5, 6, 6]
```

```
Consumer 2: Consumed item 5
Consumer 2: Buffer: [6, 6]

Producer 2: Produced item 7
Producer 2: Buffer: [6, 6, 7]

Producer 2: Produced item 8
Producer 2: Buffer: [6, 6, 7, 8]

Consumer 2: Consumed item 6
Consumer 2: Buffer: [6, 7, 8]

Producer 1: Produced item 7
Producer 1: Buffer: [6, 7, 8, 7]

Consumer 1: Consumed item 6
Consumer 1: Buffer: [7, 8, 7]

Producer 2: Produced item 9
Producer 2: Buffer: [7, 8, 7, 9]

Consumer 1: Consumed item 7
Consumer 1: Buffer: [8, 7, 9]

Producer 2: Produced item 10
Producer 2: Buffer: [8, 7, 9, 10]

Consumer 2: Consumed item 8
Consumer 2: Buffer: [7, 9, 10]

Producer 1: Produced item 8
Producer 1: Buffer: [7, 9, 10, 8]

*****Producer 2 finished.*****
Consumer 1: Consumed item 7
Consumer 1: Buffer: [9, 10, 8]

Producer 1: Produced item 9
Producer 1: Buffer: [9, 10, 8, 9]

Consumer 2: Consumed item 9
Consumer 2: Buffer: [10, 8, 9]

Producer 1: Produced item 10
```

```
Producer 1: Buffer: [10, 8, 9, 10]

Consumer 2: Consumed item 10
Consumer 2: Buffer: [8, 9, 10]

*****Consumer 2 finished.*****
Consumer 1: Consumed item 8
Consumer 1: Buffer: [9, 10]

Consumer 1: Consumed item 9
Consumer 1: Buffer: [10]

Consumer 1: Consumed item 10
*****Producer 1 finished.*****
Consumer 1: Buffer: []

*****Consumer 1 finished.*****
All done.
```

同步機制說明:
buffer 用於儲存生產者生產的物品，等待消費者來消費，最大空間為 4。
collections.deque 用以實現 FIFO 的 buffer。
由於 Python 的 thread 調度是由 GIL（Global Interpreter Lock）控制的，GIL 是 Python 的一種機制，它確保在任何時候只有一個 thread 在運行，因此必須使用隨機延遲來模擬真實情況，並且我將生產者的延遲設置較小，以模擬 buffer 被塞滿的情況。

empty 和 full 是兩個 semaphore，它們用於控制對共用資源的訪問。
empty 表示 buffer 中的空位數量。當生產者想要生產一個新的物品並將其放入 buffer 時，它會先檢查 empty。如果 empty 的值大於零，表示 buffer 中有空位，生產者可以生產項目並將其放入 buffer。然後，生產者會將 empty 的值減一，表示 buffer 中的空位數量減少了一個。
full 表示 buffer 中的項目數量。當消費者想要消費一個物品時，它會先檢查 full。如果 full 的值大於零，表示 buffer 中有物品可以消費，消費者可以從 buffer 中取出一個物品並消費它。然後，消費者會將 full 的值減一，表示 buffer 中的物品數量減少了一個。
這兩個 semaphore 的作用是確保生產者和消費者可以正確地協調他們對 buffer 的訪問。當 buffer 滿時，生產者需要等待直到有空位可用；當 buffer 空時，消費者需要等待直到有新的物品可用。這種機制可以防止生產者在 buffer 滿時仍然試圖生產新的物品，也可以防止消費者在 buffer 空時仍然試圖消費物品。

mutex 是一個互斥鎖，用於保護共用資源（在這裡是 buffer）的訪問，防止同時由多個 threads 修改。

當一 thread 想要訪問共用資源時，它必須先獲取 mutex。如果 mutex 已經被其他 thread 獲取，該 thread 將被阻塞，直到 mutex 變為可用。一旦 thread 獲取了 mutex，其他試圖獲取 mutex 的 thread 將被阻塞，直到該 thread 釋放 mutex。在這段程式碼中，生產者和消費者在添加或移除 buffer 中的物品時都會獲取 mutex。這確保了在任何時候，只有一個 thread 可以修改 buffer。這防止了同時由多個 thread 修改 buffer，可能導致的資料不一致和其他問題。