

# El Patrón - Consumidor

Es un patrón de programación concurrente que se utiliza para coordinar el trabajo entre múltiples procesos o hilos en un sistema. Es utilizado en situaciones donde un conjunto de tareas deben ser ejecutadas por una parte del sistema y procesadas por otra parte, especialmente cuando el trabajo producido es de manera continua o de diferentes velocidades.

Este patrón permite que el programa maneje una cola compartida de tareas de manera segura, sin conflictos entre las partes del sistema que están produciendo trabajo (productores) y aquellas que están consumiendo (consumidores).

## Conceptos clave

### 1.- Productor

- \* El productor es la entidad (procesos o hilos) que genera datos o tareas y los coloca en una cola compartida (buffer).
- \* Cada vez que el productor produce un nuevo trabajo o dato, lo coloca en la cola para que uno o más consumidores los procesen.

### 2.- Consumidor

- \* El consumidor es la entidad que retira datos o tareas de la cola compartida para procesarlos.
- \* Puede haber uno o varios consumidores y pueden trabajar de manera concurrente retirando datos a medida que estén disponibles.

### 3.- Buffer o Cola

- \* El buffer es una estructura de datos compartida normalmente implementada con una cola (fifo, lifo u otro según el caso).
- \* La cola sirve como una especie de área de intercambio donde los productores depositan los elementos producidos y los consumidores los retiran.
- \* La cola tiene una capacidad limitada, lo cual significa que si el productor produce más de lo que el consumidor puede procesar, eventualmente el buffer se llenaría haciendo que el productor tenga que esperar hasta que el consumidor libere espacio.

\* 5

## 4.- Sincronización

- \* La sincronización es necesaria para evitar múltiples productores y consumidores acceder al buffer al mismo tiempo. Cuando surgen problemas como colisiones de carrera.
- \* Se utilizan herramientas como semáforos y loops (Bloqueos) para asegurar que un solo productor o consumidor modifique el buffer a la vez.

## 5.- Funcionamiento del Productor, Productor y Consumidor

- \* Creación del buffer: el buffer compartido se crea con una capacidad determinada (P/E en espacio de 10 elementos). Este buffer actuará como una cola de mensajes entre los productores y consumidores.
- \* Producir un ítem: Cada productor genera un ítem (Carta) y lo coloca en el buffer. Si el buffer está lleno, el productor debe esperar hasta que haya un espacio disponible.
- \* Consumir un ítem: Cada consumidor tiene un ítem del buffer para procesar. Si el buffer está vacío, el consumidor debe esperar hasta que el productor produzca un nuevo ítem.
- \* Sincronización de errores: La sincronización asegura que no se pueda colocar un nuevo ítem si el buffer está lleno / no se puede extraer un ítem si el buffer está vacío / los productores y consumidores no acceden simultáneamente al buffer.

Un ejemplo del gerón-consumidor usando las bibliotecas de threading Unix (cola segura)

```
import threading  
import time  
import random  
import  
from queue import Queue  
  
# Tamaño máximo del buffer compartido  
BUFFER_SIZE = 5  
buffer = Queue(BUFFER_SIZE)  
  
# Función del Productor  
def productor():  
    while True:  
        item = random.randint(1, 100) # Generar ítem al azar  
        buffer.put(item) # Colocar ítem en la cola  
        print(f"Productor produjo: {item}")  
        time.sleep(random.uniform(0.1, 1)) # Simular tiempo de producción  
  
# Función del Consumidor  
def consumidor():  
    while True:  
        item = buffer.get() # Extraer ítem de la cola  
        print(f"Consumidor consumo: {item}")  
        buffer.task_done() # Marca el ítem como procesado  
        time.sleep(random.uniform(0.2, 1.5)) # Simula tiempo de consumo  
  
# Crear e iniciar los hilos para productores y consumidores  
productor_thread = threading.Thread(target=productor)  
consumidor_thread = threading.Thread(target=consumidor)  
  
productor_thread.start()  
consumidor_thread.start()  
  
# Esperar a que los hilos terminen (en este hilo, corren independiente)  
productor_thread.join()  
consumidor_thread.join()
```

## Explicación del Código

### 1. Buffer Compartido:

- Utilizamos Queue con un tamaño máximo (BUFFER\_SIZE) que actúa como el buffer o cola de intercambio entre productor y consumidor.
- Queue se utiliza porque es segura para múltiples hilos en python y maneja la sincronización.

### 2. Productor

- El productor genera un número aleatorio, simula un proceso de producción con time.sleep(5) y coloca el buffer con put().
- Si el buffer está lleno, put() bloqueará al productor hasta que el consumidor libere espacio.

### 3. Consumidor

- El consumidor toma un elemento del buffer utilizando get(), lo procesa (en este caso solo imprime), y luego llama a task\_done() para indicar que el item fue procesado.
- Si el buffer está vacío, get() bloqueará al consumidor hasta que el productor deposite un nuevo ítem.

## Ventajas del Patrón Productor - Consumidor:

1. Desacoplamiento: Los productores y consumidores no dependen uno del otro, sino que interactúan a través del buffer. Esto permite que ambos operen a diferentes ritmos.
2. Escalabilidad: Podemos agregar más productores o consumidores según la carga de trabajo, y el patrón se mantendrá efectivo.
3. Sincronización de Acceso: Herramientas como Queue en python o semáforos en otros lenguajes aseguran que el buffer compartido sea seguro para múltiples hilos.

## Aplicaciones Comunes

- **Procesamiento de Datos en tiempo real:** Un sistema de adquisición de datos que hace lecturas y las procesa en paralelo.
- **Sistemas de Redes y Consecuencias:** Un servidor que recibe solicitudes de clientes (productores) y las procesa en paralelo (consumidores).
- **Sistemas de Mensajería:** En sistemas de mensajería, los productores pueden ser usuarios que envían mensajes, y los consumidores, los receptores.