UNIVERSIDAD DE GUADALAJARA

Centro Universitario de Ciencias Exactas e Ingenierías

Ingeniería en Computación

6to semestre

Tema: Algoritmo Productor-Consumidor

Materia:
Seminario de Sistemas Operativos

Sección:
D02

NRC:
103845

PRESENTA:
Saúl Alejandro Castañeda Pérez
Daniel Martínez Martínez

Docente:
Violeta del Rocio Becerra Velazquez

Fecha de entrega:
Domingo, 30 de Abril de 2023

Simulación del Algoritmo Productor-Consumidor

Introducción

El programa de esta semana contempla la realización de un programa que demuestre el comportamiento del planteamiento del productor y el consumidor.

El algoritmo productor-consumidor es un patrón de diseño comúnmente utilizado en programación concurrente para resolver el problema de la comunicación y la sincronización entre procesos o threads. Este algoritmo se basa en dos tipos de procesos: los productores, que generan datos, y los consumidores, que consumen esos datos. El objetivo del algoritmo es asegurar que los datos sean producidos y consumidos de manera ordenada y segura, evitando problemas como la competencia de recursos o la sobrecarga de memoria. El algoritmo productor-consumidor es utilizado en una amplia variedad de aplicaciones, como la gestión de colas de mensajes, la compartición de memoria y la programación de sistemas operativos.

Desarrollo

A continuación, se explica a grandes rasgos cómo es que funciona el programa:

1. ***Existe un solo productor y un solo consumidor.***

   Es el escenario más sencillo donde solo se considera un solo productor y un solo consumidor. Esto se logra a través de la creación y ejecución de un hilo para cada uno de los procesos. Ejemplo:

   ```
   # Creamos los threads para el productor y el consumidor
   producer_thread = threading.Thread(target=producer)
   consumer_thread = threading.Thread(target=consumer)

   # Iniciamos los threads
   producer_thread.start()
   consumer_thread.start()
   ```

2. ***Se cuenta con un "contenedor" con capacidad para 22 elementos, en el cual el productor colocará y el consumidor retirará elementos.***

   En pantalla debe mostrarse el "contenedor" si es en modo consola puedes delimitarse con los 22 espacios marcados. Esto se logra haciendo uso de un arreglo común y corriente en Python, apartando los 22 espacios a través de la inicialización de la estructura con 22 berenjenas (las berenjenas representan espacios libres o vacíos, mientras que las hamburguesas representan espacios ocupados). Ejemplo:

   ```
   # Creamos una cola compartida para que el productor y el consumidor la utilicen
   cola_compartida = ['🍆'] * 22
   #Cola vacia
   ['🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆', '🍆']
   #Cola llena
   ['🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔', '🍔']
   ```

3. ***El "contenedor", lógicamente es un buffer circular y acotado, es decir al llegar a la última casilla (22) comenzará nuevamente en la casilla 1.***

    a. Comportamiento de liga circular: se refiere a que la lista no tendrá un final, si no que la última posición tendrá asociada la primera casilla como el elemento que le sigue. Esto se logra con la siguiente validación:

```
#Incrementamos el índice
indice=indice+1
#Si estamos el índice ya se paso la ultima casilla, lo posicionamos de vuelta al inicio
if indice==22:
    indice=0
```

    b. Tanto el productor como el consumidor cuando inicien deberán hacerlo a partir de la primera posición, pero al salir del contenedor y volver a entrar continuará del punto donde se quedó, por ello es que al llegar a la última posición continuará de nuevo al principio del buffer.

    Esto se logra a través del uso de dos variables de índice, una para el productor y otra para el consumidor. El índice del productor simplemente se incrementa de uno en uno y se regresa a la primera posición en cuanto llega al final de la lista, tal y como lo dicta la circularidad. Por otro lado, el índice del consumidor se inicia buscando la primera hamburguesa del arreglo solo si no hay hamburguesas en algún punto más adelante de la lista, considerando la posibilidad de que se hayan puesto hamburguesas al inicio de la lista por la circularidad. Básicamente se inicia desde donde se quedó la puesta de berenjenas la última vez. Ejemplo:

    ⬇️*comenzamos de aquí*

    🍔🍔🍔🍔🍔🍆🍆🍆🍆🍆🍆🍔🍔🍔🍔

4. ***El producto puede ser: números, caracteres especiales, letras, etc.***

    Decidimos trabajar con cangreburguers porque son god 👉🍔👈

5. ***Solo puede ingresar uno a la vez al contenedor. Para que el productor entre, debe haber espacio en el contenedor, mientras que debe haber hamburguesas para consumir para que el consumidor entre.***

    Debe cumplirse la exclusión mutua, sólo uno a la vez dentro del contenedor. Esto se cumple aprovechando las propiedades de los hilos para generar la exclusión mutua, a través del uso de una variable de tipo threatening condition, la cual indica a los hilos que uno está trabajando, impidiendoles trabajar si dicha condición está activa. Existen tres situaciones donde los hilos se detienen por la condición:

    a. Cuando no hay hamburguesas para consumir: el hilo del consumidor se pone en modo de espera dado que no hay producto para consumir.

```
# Si la cola está vacía, esperamos a que se agregue un elemento
while (HayAnvorguesas()) == -1:
    condition.wait()
```

    b. Cuando no hay espacio para producir hamburguesas: el hilo del productor se pone en espera porque detecta que ya no hay espacio.

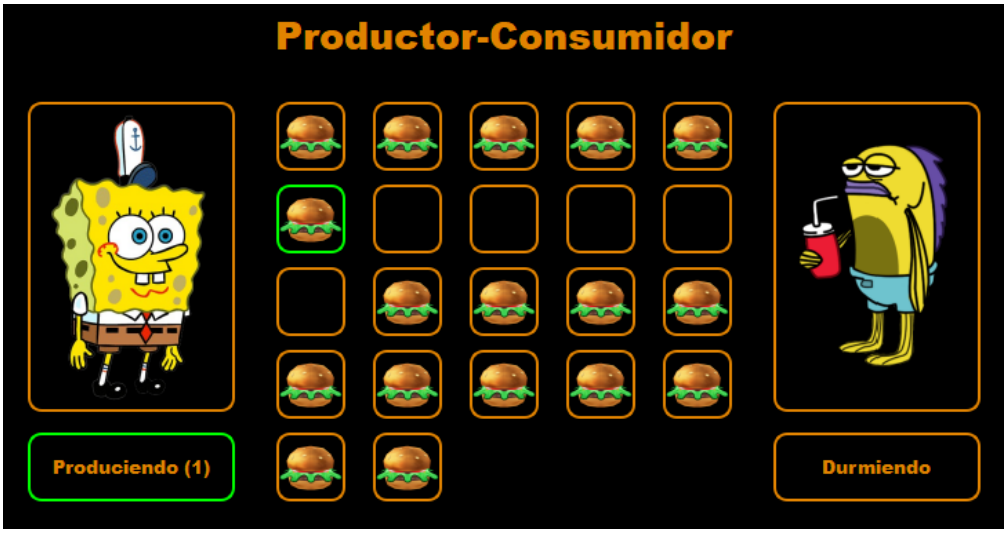
```
        # Si la cola está llena, esperamos a que se libere un espacio
        while (HayBerenjenas()) == -1:
            condition.wait()
```

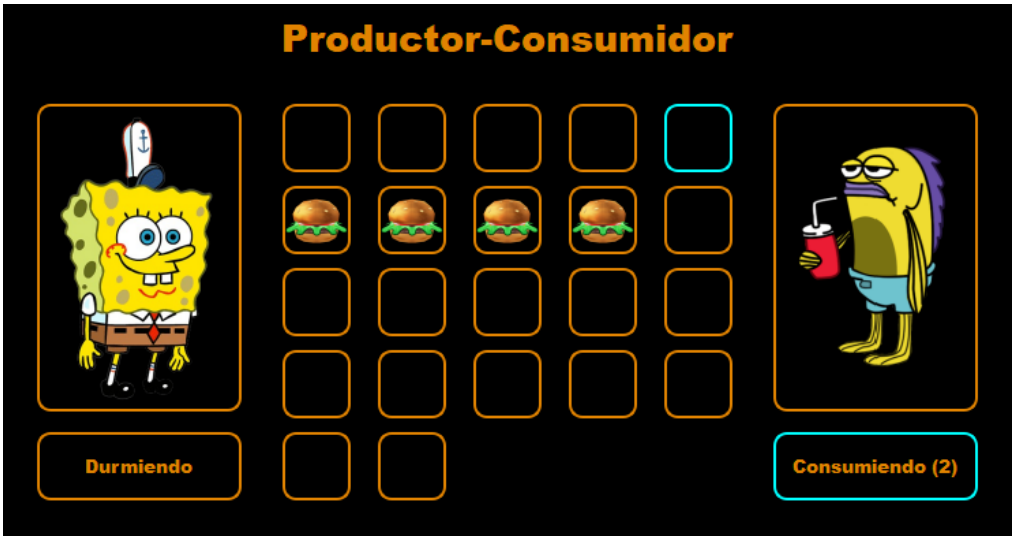
c. Cuando uno de los hilos está trabajando: en este caso, la condición espera a que el hilo termine de trabajar para notificar a los otros hilos (incluso a este mismo) que la cola está disponible para su uso 🥵

6. ***En la pantalla debe aparecer:***

   a. ***El contenedor con los espacios claramente marcados y numerados.***

   b. ***La información del productor, es decir, mostrar si está dormido, trabajando, cuando intente ingresar al contenedor, etc. Hacer lo mismo con el consumidor.***





   c. ***Esta parte puede omitirse si se observa el movimiento en pantalla (usar un retardo para que se note y no esté tan rápido).***

   Esto se logra gracias a el siguiente retardo:

   #Retardo para poder apreciar los cambios progresivamente

```
def Retardo(self,time):
    loop = QtCore.QEventLoop()
    QtCore.QTimer.singleShot(time, loop.quit)
    loop.exec_()
```

7. ***Deben manejarse tiempos aleatorios para dormir al productor y al consumidor.***

   Esto se logra con el siguiente fragmento de código:

```
# Esperamos un tiempo aleatorio antes de producir un elemento, si da 0, se vuelve a ejecutar el hilo que acaba de acceder a la cola, si da 1, entra el otro hilo
        time.sleep(random.randint(0, 1))
```

8. **Al "despertar" intentará producir y/o consumir respectivamente, verificando las condiciones**

   Volver al punto 5 para los detalles.

9. ***Al entrar al buffer podrán producir y/o consumir de tres a seis elementos en cada entrada.***

   Esto se logra con un generador de número de items aleatorio, usando la funcion random:

```
# Consideramos una cantidad aleatoria de elementos a agregar/quitar de la cola
        numeroItems = random.randint(3, 6)
```

10. ***El programa terminará cuando se presione la tecla "ESC".***

    Esto se logra gracias a un listener de pyqt5 para detectar que se presionó una tecla, en este caso ESC, acción que setea una variable de tipo Event para el control de hilos. activar dicha variable hace que los hilos dejen de ejecutarse, deteniendo así el programa.

```
# Creamos un objeto Event
stop_event = threading.Event()

# Seteamos la variable para detener los hilos
def CapturarTeclas(self,event):
    global bnd,stop_event
    if event.key()==QtCore.Qt.Key_Escape:
        print("ESC PRESIONADA")
        stop_event.set()
```

    Posterior a esto, el programa se quedará congelado, siendo ahora necesario cerrar la ventana de la interfaz con el botón correspondiente.

En base a lo anterior, se concibió la lógica del programa en forma del siguiente código en Python:

CÓDIGO RELEVANTE

```
import threading
import time
import random

# Creamos una cola compartida para que el productor y el consumidor la utilicen
cola_compartida = ['🍆'] * 22
#Creamos una variable que funge como el índice donde se han ido poniendo hamburguesas
indice=0
#Creamos un segundo índice para llevar seguimiento a la puesta de berenjenas (donde nos quedamos)
indice2=0

#Esta funcion define si la lista esta llena, dado la no existencia de berenejas indicaria que no hay espacios libres
def HayBerenjenas():
    try:
        return cola_compartida.index('🍆')
    except:
        return -1
#Esta funcion define si la lista esta vacia, dado que no encontrar alguna hamburguesa significaria que esta vacia
def HayAnvorguesas():
    try:
        return cola_compartida.index('🍔')
    except:
        return -1
"""
Aquí buscamos la posición desde donde empezaremos a consumir, localizando donde comienzan las hamburguesas después de los espacios vacíos   |comenzamos de aquí
                                                                                                                                            v
            🍔🍔🍔🍔🍔🍆🍆🍆🍆🍆🍆🍔🍔🍔🍔
"""
def indice2Inicial():
    try:
        #Comenzamos a buscar asumiendo que se colocaron hamburguesas en el inicio por la circularidad
        indiceAux=cola_compartida.index('🍆')
        return cola_compartida.index('🍔',indiceAux)
    except:
        try:
            #En caso de que no se encontrara ninguna hamburguesa de la forma anterior, ahora si buscamos alguna desde el inicio
            return cola_compartida.index('🍔',0)
        except:
            return -1


# Creamos una variable de condición para que el productor espere cuando la cola esté llena
condition = threading.Condition()

# Definimos una función para el productor que agregará elementos a la cola compartida
def producer():
    global indice
    while True:
        # Esperamos un tiempo aleatorio antes de producir un elemento
        time.sleep(random.randint(0, 1))

        # Consideramos una cantidad aleatoria de elementos a agregar a la cola
        numeroItems = random.randint(3, 6)
```

```
    # Bloqueamos la cola compartida con la variable de condición
    with condition:
        # Si la cola está llena, esperamos a que se vacíe un espacio
        while (HayBerenjenas()) == -1:
            condition.wait()

        # Agregamos los N items a la cola
        print("Produciendo ",numeroItems," hamburguesas")
        for i in range (numeroItems):
            #En caso de que haya una hamburguesa en el índice actual, terminamos de colocar
            #hamburguesas, dado que esto indicaria que ya está llena la cola
            if (cola_compartida[indice]=='🍔'):
                break
            #Cambiamos una apestosa berenjena por una deliciosa cangreburguer
            cola_compartida[indice]='🍔'
            #Incrementamos el índice
            indice=indice+1
            #Consideramos circularidad
            if indice==22:
                indice=0
            time.sleep(1)
            print("PRODUCIENDO -> ",cola_compartida, "en indice", indice)

        # Notificamos a los threads en espera que la cola ha cambiado
        condition.notify_all()


# Definimos una función para el consumidor que quitará elementos de la cola compartida
def consumer():
    global indice2
    while True:
        # Esperamos un tiempo aleatorio antes de intentar consumir un elemento
        time.sleep(random.randint(0, 1))

        # Consideramos una cantidad aleatoria de elementos a quitar de la cola
        numeroItems = random.randint(3, 6)

        # Bloqueamos la cola compartida con la variable de condición
        with condition:
            # Si la cola está vacía, esperamos a que se agregue un elemento
            while (HayAnvorguesas()) == -1:
                condition.wait()

            # Liberamos espacios de la cola, reemplazando las hamburguesas con berenjenas
            #Comenzamos a colocar berenjenas donde nos habiamos quedado
            if((HayBerenjenas()) == -1):
                indice2=indice
            else:
                indice2=indice2Inicial()
            print("Poniendo ",numeroItems," berenjenas")
            for i in range (numeroItems):
                #SI de donde estamos ya hay una berenjena, nos detenemos, dado que esto significa que
                #No hay kangreburguers para consumir
                if (cola_compartida[indice2]=='🍆'):
                    break
                #Cambiamos una hamburguesa por una sucia berenjena
                cola_compartida[indice2]='🍆'
                #Incrementamos el indice para continuar a la siguiente posicion de la lista
```

```
            indice2=indice2+1
            #Consideramos la circularidad
            if indice2==22:
               indice2=0
            time.sleep(1)
            print("CONSUMIENDO -> ",cola_compartida," en indice ", indice2)

        # Notificamos a los threads en espera que la cola ha cambiado
        condition.notify_all()

# Creamos los threads para el productor y el consumidor
producer_thread = threading.Thread(target=producer)
consumer_thread = threading.Thread(target=consumer)

# Iniciamos los threads
producer_thread.start()
consumer_thread.start()
```

Conclusiones

El programa de esta semana no estuvo tan difícil, debido a que su lógica no era tan rebuscada, más sin embargo esto no significó que fuera innecesario hacer validaciones. Entre los principales retos fue el uso de hilos (según recomendaciones para la implementación de este algoritmo), debido a que no contábamos con mucha experiencia usando dicha técnica de programación, principalmente a la hora de pararlos, ya que es sumamente fácil que empiecen a causar destrozos si no se trata adecuadamente su inicialización, ejecución y finalización.

Si comparamos este programa con los otros, es bastante más sencillo, esto porque no tuvimos que sincronizar una gran cantidad de elementos como sí pasaba en las entregas anteriores.

Dios nos bendiga con el siguiente programa 🥲🙏