

INDICE

TABLA DE IMAGENES	2
DESCRIPCION DE ACTIVIDAD	3
CONTENIDO.....	5
Problema.....	5
Solución y explicación.....	7
Solución	7
Explicación del programa.....	7
Ejecución del programa	12
Enlace de video	13
CONCLUSION	13
BIBLIOGRAFIA	14

TABLA DE IMAGENES

Ilustración 1: Productor Consumidor.....	5
Ilustración 2: Módulos.....	7
Ilustración 3: Buffer y posición inicial de productor y consumidor	7
Ilustración 4: Inicialización de variables y presionar tecla 'ESC'	8
Ilustración 5: Turno del productor para trabajar	9
Ilustración 6: Turno del consumidor para trabajar	10
Ilustración 7: Mostrar elementos del buffer.....	10
Ilustración 8: Mostrar posición donde se quedó cada uno respectivamente y cantidad de elementos que faltaron.....	11
Ilustración 9: Tres primeros turnos en comprobación	12
Ilustración 10: Cuarto, quinto y sexto turno en comprobación	12
Ilustración 11: Tres últimos turnos en comprobación	13

DESCRIPCION DE ACTIVIDAD

- a) *Investigue el problema del productor-consumidor con sus diferentes escenarios.*
- b) *Una vez verificado en que consiste debe programarlo utilizando semáforos.*
- c) *El lenguaje de programación a utilizar es a su elección.*
- d) *Puede ser gráfico o modo consola.*
- e) *Debe tener en consideración los siguientes puntos:*
 - 1. *Existen un solo productor y un solo consumidor.*
 - 2. *Se cuenta con un “contenedor” con capacidad para 20 elementos, en el cual el productor colocará y el consumidor retirará elementos.*
 - 3. *El “contenedor”, lógicamente es un buffer circular y acotado, es decir al llegar a la última casilla (20) comenzarán nuevamente en la casilla 1 (los espacios deben estar perfectamente señalados).*
 - 4. *El producto puede ser: números, caracteres especiales, letras, hamburguesas, galletas, etc.*
 - 5. *Solo puede ingresar uno a la vez al contenedor.*
 - 6. *Para que el Productor pueda entrar, debe haber espacio en el contenedor y no estar el Consumidor dentro.*
 - 7. *Para que el Consumidor pueda entrar, debe existir producto y no estar el productor dentro.*
 - 8. *En la pantalla debe aparecer:*
 - a. *La información del productor, es decir, mostrar si está dormido, trabajando, cuando intente ingresar al contenedor, etc.*
 - b. *La información del consumidor, dormido, trabajando, cuando intente ingresar, etc.*
 - c. *Mensajes que indiquen en todo momento, quien está trabajando, o quien intenta trabajar, o si está dormido.*
 - 9. *Deben manejarse tiempos aleatorios para dormir al productor y al consumidor.*
 - 10. *Al “despertar” intentará producir y/o consumir respectivamente, verificando que pueda hacerlo según sus condiciones.*
 - 11. *Al entrar al buffer podrán producir y/o consumir de forma finita (de 1 a 4 elementos).*
 - 12. *El productor colocará elementos en orden, comenzando con la primera casilla, y continuando desde la última posición donde se quedó.*
 - 13. *El consumidor quitará elementos en orden, comenzando también por la primera casilla y continuando en la última donde quedo.*
 - 14. *El programa terminara al presionar la tecla escape.*

15.NOTA: Descargue el archivo Productor-Consumidor_Explicación que se encuentra en la plataforma moodle2. Este le ayudará a resolver dudas con respecto a los requisitos antes mencionados.

- f) Genere un reporte donde incluya la explicación del problema, la solución que propone y la explicación de su código (incluya imágenes).*
- g) Realice un vídeo con la demostración del programa funcionando, entréguelo en algún sitio en la nube y no olvide proporcionar permisos públicos de visualización.*
- h) Incluya el enlace al vídeo en su reporte.*
- i) Genere una conclusión sobre lo que ha aprendido en esta actividad.*
- j) Entregue su tarea al buzón de tareas antes de la fecha de término.*

CONTENIDO

Problema

El problema de productor consumidor consiste en que estos se comunican a través de un buffer, en el que los productores lo llenan de productos y los consumidores lo vacían. Consiste en un buffer finito es compartido por productores y consumidores. Si el búfer está lleno, el productor se va a dormir y, si está vacío, el consumidor se va a dormir.



Ilustración 1: Productor Consumidor

El buffer tiene una capacidad limitada, pudiendo almacenar sólo N elementos del mismo tamaño. La sincronización que se requiere en este caso es doble: en primer lugar, los productores no pueden colocar elementos en el buffer si éste está lleno, y en segundo lugar, los consumidores no pueden sacar elementos del buffer si éste está vacío.

Además, el buffer y las variables compartidas deben protegerse del acceso concurrente por parte de los distintos procesos, pues son zonas compartidas de memoria modificables por los procesos. De ahí que el acceso al buffer y a las variables compartidas se realice en exclusión mutua.

Los problemas comienzan cuando el productor quiere dejar algo en el buffer, pero éste ya está lleno. La solución es mandar a dormir al productor y despertarlo sólo cuando el consumidor haya retirado uno o más elementos. Análogamente, debe enviarse a dormir al consumidor cuando trate de retirar un elemento del buffer y éste esté vacío. Más tarde, cuando el productor deje uno o más elementos en el buffer, despertará al consumidor.

Para llevar la cuenta del número de elementos en el buffer, se necesita una variable, cuenta. El acceso a la variable cuenta no está controlado. Así, se puede presentar la siguiente situación: el buffer está vacío y el consumidor acaba de leer 0 de la variable cuenta. En este instante el planificador decide pasar el control del consumidor al productor. El productor deja un elemento en el buffer, incrementa cuenta y observa que su valor ahora es 1, por lo que deduce que su valor era 0 y que, por tanto, el consumidor está dormido, así que el productor llama a despertar para despertar al consumidor.

Por desgracia, el consumidor no está todavía lógicamente dormido, de modo que la señal para despertarlo se pierde. Cuando el consumidor se active de nuevo, comprobará el valor de cuenta que leyó anteriormente, verá que es 0 y se irá a dormir. Por su parte, el productor acabará llenando el buffer y también se tendrá que ir a dormir. Ambos dormirán eternamente.

Solución equívoca al problema del productor-consumidor mediante dormir-despertar

Para solucionar este problema, Dijkstra, en 1965, tuvo la idea de utilizar una variable entera para contar el número de señales de despertar puestas "en conserva" para uso futuro. Propuso el tipo de variable llamada semáforo junto con las operaciones wait y signal definidas sobre él. En el artículo original sobre los semáforos Dijkstra utilizó las letras P y V, que son las iniciales de estos términos en holandés. Aunque también se puede utilizar wait y signal por ser más descriptivos.

Un semáforo es una zona de memoria compartida que almacena un entero no negativo sobre el cual sólo puede actuarse con una de las operaciones siguientes:

- Inicialización: Un semáforo sólo puede inicializarse una vez, normalmente en el momento en que se define.
- wait(s): Si $s > 0$ decrementa s en una unidad. Si s vale 0 provoca el bloqueo del proceso hasta que otro proceso realice una operación signal sobre s . Si existen varios procesos bloqueados en un semáforo, sólo uno de ellos podrá pasar a listo tras una operación signal sobre él. No se lleva a cabo suposición alguna sobre cuál de los procesos bloqueados será el elegido.
- signal(s): Si no hay procesos bloqueados en s , provoca el incremento de s en una unidad. Si hay procesos bloqueados en s (que valdrá cero), provoca la transición a listo de uno de estos procesos.

Además, las acciones que implica una operación wait o signal se ejecutan indivisiblemente. Esto es crucial para garantizar la coherencia del semáforo, ya que éste no es más que una variable compartida. Como se vio, una zona compartida de memoria no puede ser accedida concurrentemente por más de un proceso, si al menos uno de ellos la modifica.

Solución y explicación

Solución

La solución consiste en que, de forma aleatoria, se decide de quien es el turno para trabajar. Se genera un numero aleatorio, según si este numero es par o impar, se decide quien será quien debe trabajar. Si es un numero par, es el turno del productor para trabajar. Caso contrario, será el turno del consumidor para trabajar. Hay diferentes condiciones que se deben cumplir antes de que el consumidor o el productor entre a trabajar en el buffer. En el caso de ser el turno del productor, debe existir espacio para colocar elementos en el buffer y el consumidor no se debe encontrar dentro. Para que el consumidor pueda entrar a trabajar al buffer, deben de existir elementos en el buffer para consumir y el productor no debe estar dentro del buffer.

Explicación del programa

En la solución implementada, el guion (-) representa vacío y el asterisco (*) representa un elemento.

Importar módulos necesarios. Se debe importar el módulo para poder salir del programa con tecla ESC (keyboard), el módulo para generar números enteros aleatorios (random) y el módulo para poder agregar retrasos de tiempo (time).

```
import keyboard
import random
import time
```

Ilustración 2: Módulos

Arreglo con el nombre *buffer*, que hace referencia al contenedor de 20 elementos mencionado en el problema. Cada posición del arreglo contiene un guion, lo cual quiere decir que el buffer se encuentra vacío. Dos variables inicializadas con el valor de -1, dichas variables hacen referencia a la posición donde se quedo el productor y el consumidor la ultima vez que trabajo cada uno respectivamente.

```
buffer = ['- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ', '- ']  
posProd = -1  
posCons = -1
```

Ilustración 3: Buffer y posición inicial de productor y consumidor

Mientras un *while* se encuentre en True, el programa sigue en ejecución.

Variables que se encuentran al inicio del *while*:

- La variable *num* almacena un numero entero aleatorio, cuyo valor será el determinante para conocer quién será el próximo a trabajar, si el productor o el consumidor.
- La variable *noElementos* es una variable booleana, que indica si en el buffer hay elementos o se encuentra vacía.

Productor Consumidor

- La variable *par* indica quien fue quien trabajo, si el productor o el consumidor. Si tiene el valor de *True*, significa que trabajo el productor. Caso contrario, significa que quien trabajo fue el consumidor.
- La variable *faltoElem* almacena la cantidad de elementos que faltaron por consumirse o producirse, es decir, en el caso de que haya entrado a trabajar alguno de los dos, y no haya suficientes elementos para consumir (consumidor) o no haya estado lo suficientemente vacío para producir todos los elementos (productor), según el numero aleatorio que se genera del 1 al 4, y no alcanzo a consumir o producir todos los que el numero aleatorio que se generara después, esa cantidad de elementos se encontrara en dicha variable.
- La variable *falto* es una variable booleana que indica si faltaron elementos o no por consumirse o producirse, según el numero aleatorio que se genera del 1 al 4.

El *if* que se muestra en la imagen adjunta, es para salir del programa al presionar la tecla ESC. Al momento de presionar la tecla, se muestra un mensaje indicando que es el fin del programa y se termina la ejecución del programa.

```
while True:
    num = random.randint(0, 10000) #NUMERO ALEATORIO PARA EL TURNO DE A QUIEN LE TOCA TRABAJAR
    noElementos = False
    par = False
    falloElem = 0
    fallo = False

    if(keyboard.is_pressed('ESC')):
        print("\n<<FIN DE PROGRAMA>>")
        exit()
```

Ilustración 4: Inicialización de variables y presionar tecla 'ESC'

Si el numero aleatorio entero que se genero anteriormente es un numero par (se almaceno en variable *num*), significa que es el turno del productor para trabajar. Se muestra en pantalla que el consumidor se encuentra dormido y el productor trabajando.

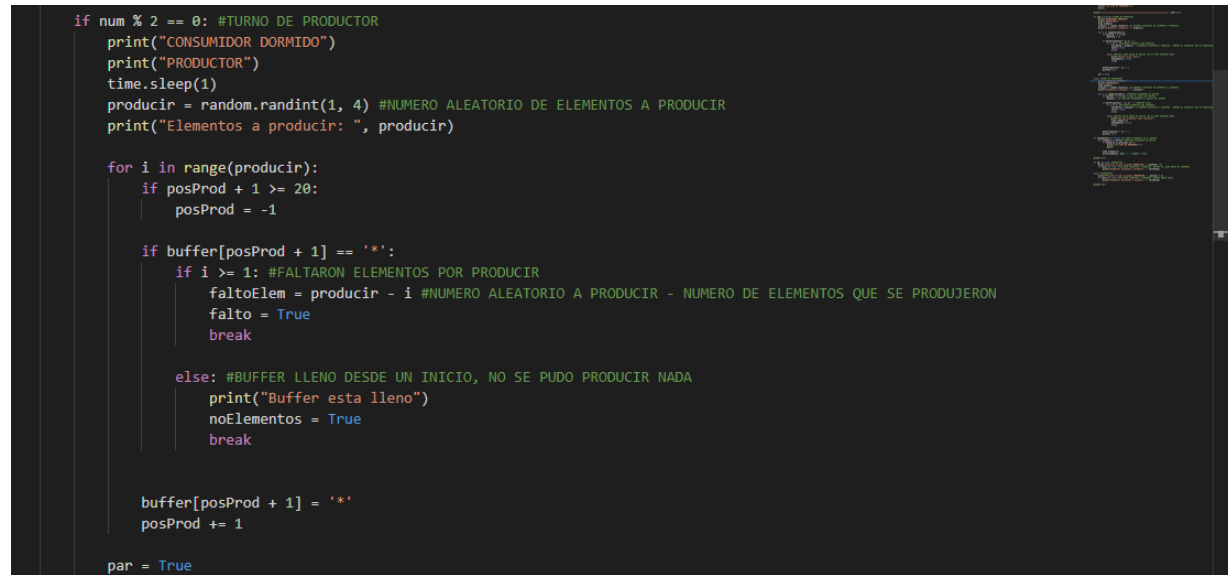
Se genera un numero aleatorio entero en el rango del 1 al 4 y se almacena dicho numero en la variable *producir*, y se muestra en pantalla el numero con el mensaje indicando que dicho número es el número de elementos a producir.

Se ejecuta un *for* para que se produzca la cantidad de elementos establecida por el numero aleatorio (*producir*).

El primer *if* evalúa si la variable *posProd* (última posición de productor) más el valor 1, es mayor o igual a 20, significa que ya llego al final del buffer, por lo tanto, se le asigna el valor de -1 a la variable *posProd*, dicho valor indica que empezara nuevamente desde el inicio del buffer.

Después el siguiente *if* evalúa si la siguiente posición de donde esta el productor en el buffer está llena (*), significa que el buffer ya está lleno. Por lo tanto, el siguiente *if* y *else* evalúa si el productor logro producir elementos, pero no alcanzo a producir todos los que el numero que se generó aleatorio del 1 al 4 indicaba, entonces *falto* toma el valor de *True* y en *faltoElem* se guarda el numero de elementos que no se produjeron. Si desde un inicio, el productor no pudo producir ningún elemento debido a que el buffer ya estaba lleno, entonces *noElementos* toma el valor de *True*.

Al terminar, *par* toma el valor de *True*.

The image shows a code editor with Python code for a producer-consumer simulation. The code is as follows:

```
if num % 2 == 0: #TURNO DE PRODUCTOR
    print("CONSUMIDOR DORMIDO")
    print("PRODUCTOR")
    time.sleep(1)
    producir = random.randint(1, 4) #NUMERO ALEATORIO DE ELEMENTOS A PRODUCIR
    print("Elementos a producir: ", producir)

    for i in range(producir):
        if posProd + 1 >= 20:
            posProd = -1

        if buffer[posProd + 1] == '*':
            if i >= 1: #FALTARON ELEMENTOS POR PRODUCIR
                faltaElem = producir - i #NUMERO ALEATORIO A PRODUCIR - NUMERO DE ELEMENTOS QUE SE PRODUCIERON
                falta = True
                break

            else: #BUFFER LLENO DESDE UN INICIO, NO SE PUDO PRODUCIR NADA
                print("Buffer esta lleno")
                noElementos = True
                break

        buffer[posProd + 1] = '*'
        posProd += 1

    par = True
```

Ilustración 5: Turno del productor para trabajar

Si el numero aleatorio entero que se generó anteriormente es un numero impar, significa que es el turno del consumidor para trabajar. Se muestra en pantalla que el productor se encuentra dormido y el consumidor trabajando.

Se genera un numero aleatorio entero en el rango del 1 al 4 y se almacena dicho número en la variable *consumir*, y se muestra en pantalla el numero con el mensaje indicando que dicho número es el número de elementos a consumir.

Se ejecuta un *for* para que se consuma la cantidad de elementos establecida por el numero aleatorio (*consumir*).

El primer *if* evalúa si la variable *posCons* (posición de consumidor) más el valor 1, es mayor o igual a 20, significa que ya llego al final del buffer, por lo tanto, se le asigna el valor de -1 a la variable *posCons*, dicho valor indica que empezara nuevamente desde el inicio del buffer.

Después el siguiente *if* evalúa si la siguiente posición de donde está el consumidor en el buffer está vacía (-), significa que el buffer ya está vacío. Por lo tanto, el siguiente *if* y *else* evalúa si el consumidor logro consumir elementos, pero no alcanzo a consumir todos los que el número que se generó aleatorio del 1 al 4 indicaba, entonces *falta* toma el valor de *True* y en *faltaElem* se guarda el número de elementos que no se consumieron. Si desde un inicio, el consumidor no pudo consumir ningún elemento debido a que el buffer estaba vacío, entonces *noElementos* toma el valor de *True*.

Productor Consumidor

```
else: #TURNO DE CONSUMIDOR
    print("PRODUCTOR DORMIDO")
    print("CONSUMIDOR")
    time.sleep(1)
    consumir = random.randint(1, 4) #NUMERO ALEATORIO DE ELEMENTOS A CONSUMIR
    print("Elementos a consumir: ", consumir)

    for i in range(consumir): #CONSUMIR ELEMENTOS DE BUFFER
        if posCons + 1 >= 20: #LLEGO AL FINAL DEL BUFFER
            posCons = -1 #INICIAR NUEVAMENTE AL INICIO DEL BUFFER

            if buffer[posCons + 1] == '-': #BUFFER VACIO
                if i >= 1: #FALTARON ELEMENTOS POR CONSUMIR
                    faltoElem = consumir - i #NUMERO ALEATORIO A CONSUMIR - NUMERO DE ELEMENTOS QUE SE PRODUJERON
                    falto = True
                    break

                else: #BUFFER VACIO DESDE UN INICIO, NO SE PUDO CONSUMIR NADA
                    print("No hay elementos para consumir")
                    time.sleep(1.5)
                    noElementos = True
                    break

        buffer[posCons + 1] = '-'
        posCons += 1
```

Ilustración 6: Turno del consumidor para trabajar

En el caso de que *noElementos* tenga el valor de *True*, significa que el consumidor o el productor no pudieron ingresar al buffer, debido a que según sea el caso, el buffer estaba lleno como para producir elementos o vacío para tener elementos que consumir. Caso contrario, el productor o el consumidor, pudo producir o consumir elementos. Por lo tanto, muestra los elementos del buffer con un retraso de tiempo, ya con los elementos que se produjeron o consumieron.

```
if noElementos == False: #SI HABIA ELEMENTOS EN EL BUFFER
    for elemento in buffer: #MOSTRAR ELEMENTOS DE BUFFER
        if keyboard.is_pressed('ESC'):
            print("\n<<<FIN DE PROGRAMA>>>")
            exit()

        time.sleep(0.5)
        print(elemento, end = " ", flush = True)
```

Ilustración 7: Mostrar elementos del buffer

Si fue el turno del productor, *par* tiene el valor de *True*, así que se muestra la posición en la cual se quedó el productor y en el caso, de que haya podido producir elementos, pero no todos los que el número aleatorio indicaba por que el buffer se llenó antes de terminar, se muestra el número de elementos que faltaron por producir mostrando el valor de la variable *faltoElem*.

Si *par* tiene el valor de *False*, significa que fue el turno del consumidor. Por lo tanto, se muestra la posición en la cual se quedó el consumidor y en el caso, de que haya podido consumir elementos, pero no todos los que el número aleatorio indicaba por que el buffer se vació antes de terminar, se muestra el número de elementos que faltaron por consumir mostrando el valor de la variable *faltoElem*.

```
if par == True: #PRODUCTOR
    print("Posicion en que se quedo PRODUCTOR: ", posProd + 1)
    if falto == True: #FALTARON ELEMENTOS A PRODUCIR, BUFFER SE LLENO ANTES DE TERMINAR
        print("Elementos faltantes a producir: ", faltoElem)

else: #CONSUMIDOR
    print("Posicion en que se quedo CONSUMIDOR: ", posCons + 1)
    if falto == True: #FALTARON ELEMENTOS A CONSUMIR, BUFFER QUEDO VACIO
        print("Elementos faltantes a consumir: ", faltoElem)
```

Ilustración 8: Mostrar posición donde se quedó cada uno respectivamente y cantidad de elementos que faltaron

Productor Consumidor

Ejecución del programa

```
PS C:\Users\maria\Documents\1 UNIVERSIDAD\6TO SEMESTRE\SEMINARIO USO\ACTIVIDAD 11\Producto consumidor> python productoconsumidor.py
=====
CONSUMIDOR DORMIDO
PRODUCTOR
Elementos a producir: 2
*****
Posición en que se quedo PRODUCTOR: 2

=====
CONSUMIDOR DORMIDO
PRODUCTOR
Elementos a producir: 2
*****
Posición en que se quedo PRODUCTOR: 4

=====
PRODUCTOR DORMIDO
CONSUMIDOR
Elementos a consumir: 1
*****
Posición en que se quedo CONSUMIDOR: 1

=====
CONSUMIDOR DORMIDO
PRODUCTOR
```

PRIMER NÚMERO ALEATORIO GENERADO (variable num)
Numero par → Turno de productor.
Elementos a producir: 2
Posición en que se quedó PRODUCTOR: 2

SEGUNDO NÚMERO ALEATORIO GENERADO (variable num)
Numero par → Turno de productor.
Elementos a producir: 2
Posición en que se quedó PRODUCTOR: 4

TERCER NÚMERO ALEATORIO GENERADO (variable num)
Numero impar → Turno de consumidor.
Elementos a producir: 1
Posición en que se quedó CONSUMIDOR: 1

Ilustración 9: Tres primeros turnos en comprobación

```
=====
CONSUMIDOR DORMIDO
PRODUCTOR
Elementos a producir: 3
*****
Posición en que se quedo PRODUCTOR: 7

=====
CONSUMIDOR DORMIDO
PRODUCTOR
Elementos a producir: 2
*****
Posición en que se quedo PRODUCTOR: 9

=====
PRODUCTOR DORMIDO
CONSUMIDOR
Elementos a consumir: 4
*****
Posición en que se quedo CONSUMIDOR: 5

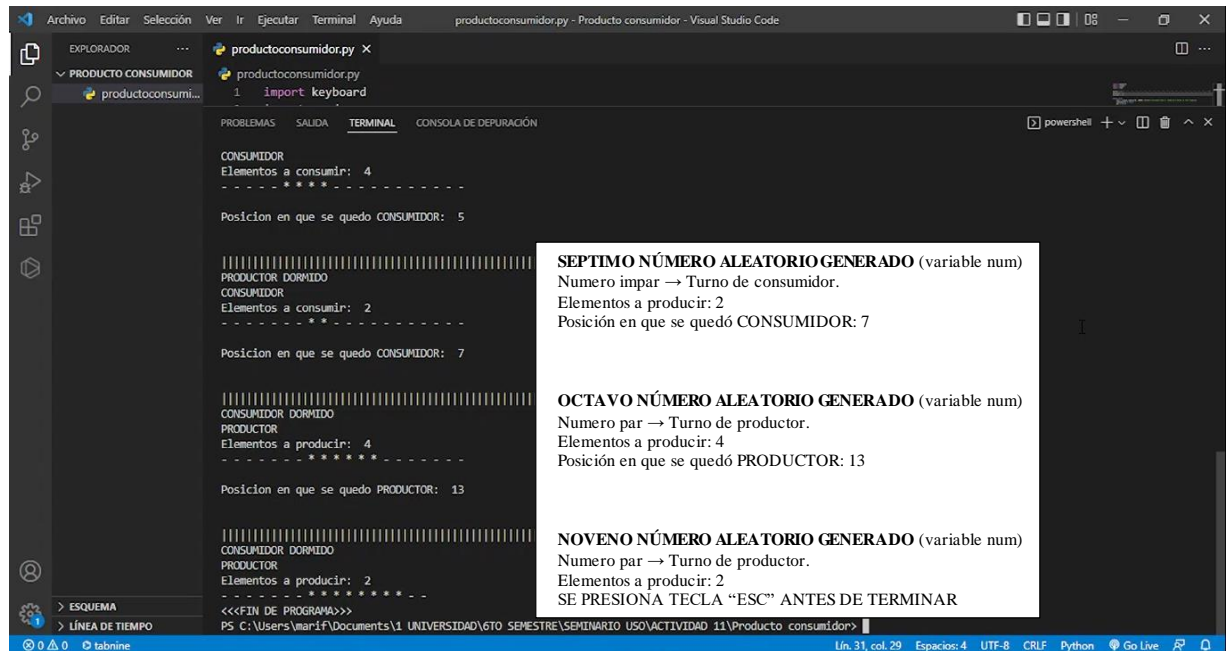
=====
PRODUCTOR DORMIDO
CONSUMIDOR
```

CUARTO NÚMERO ALEATORIO GENERADO (variable num)
Numero par → Turno de productor.
Elementos a producir: 3
Posición en que se quedó PRODUCTOR: 7

QUINTO NÚMERO ALEATORIO GENERADO (variable num)
Numero par → Turno de productor.
Elementos a producir: 2
Posición en que se quedó PRODUCTOR: 9

SEXTO NÚMERO ALEATORIO GENERADO (variable num)
Numero impar → Turno de consumidor.
Elementos a producir: 4
Posición en que se quedó CONSUMIDOR: 5

Ilustración 10: Cuarto, quinto y sexto turno en comprobación



```
productoconsumidor.py
1 import keyboard

CONSUMIDOR
Elementos a consumir: 4
----- * * * * -----
Posición en que se quedo CONSUMIDOR: 5

|-----|
| CONSUMIDOR DORMIDO |
| CONSUMIDOR |
| Elementos a consumir: 2 |
| ----- * * ----- |
| Posición en que se quedo CONSUMIDOR: 7 |

|-----|
| CONSUMIDOR DORMIDO |
| PRODUCTOR |
| Elementos a producir: 4 |
| ----- * * * * ----- |
| Posición en que se quedo PRODUCTOR: 13 |

|-----|
| CONSUMIDOR DORMIDO |
| PRODUCTOR |
| Elementos a producir: 2 |
| ----- * * * * ----- |
| Posición en que se quedo PRODUCTOR: 13 |

<<<FIN DE PROGRAMA>>>
PS C:\Users\marif\Documents\1 UNIVERSIDAD\6TO SEMESTRE\SEMINARIO USO\ACTIVIDAD 11\Producto consumidor>
```

SEPTIMO NÚMERO ALEATORIO GENERADO (variable num)
Numero impar → Turno de consumidor.
Elementos a producir: 2
Posición en que se quedó CONSUMIDOR: 7

OCTAVO NÚMERO ALEATORIO GENERADO (variable num)
Numero par → Turno de productor.
Elementos a producir: 4
Posición en que se quedó PRODUCTOR: 13

NOVENO NÚMERO ALEATORIO GENERADO (variable num)
Numero par → Turno de productor.
Elementos a producir: 2
SE PRESIONA TECLA "ESC" ANTES DE TERMINAR

Ilustración 11: Tres últimos turnos en comprobación

Enlace de video

https://drive.google.com/file/d/1vX_oO80aM6zEye_IE5sV933oMPvayQ8P/view?usp=sharing

CONCLUSION

El problema del productor consumidor consiste en que un productor y un consumidor comparten un buffer, mientras un productor guarda elementos en el buffer, el consumidor quita elementos del buffer. Se utiliza el problema como un ejemplo de exclusión mutua debido a que comparten el buffer, mas no pueden estar ambos al mismo tiempo en él. Para que un productor pueda trabajar, el consumidor no puede estar trabajando en el buffer y debe haber espacio para el productor colocar sus productos y el consumidor para poder trabajar, el productor no puede estar trabajando en el buffer y debe haber elementos en el buffer para consumir.

Hay diferentes soluciones que se pueden implementar para resolver el problema, algunos de ellos consisten en utilizar monitores, semáforos y entre otros. Para que fuera de forma aleatoria la elección de quien sería el próximo a trabajar, generaba un numero aleatorio entero y según su valor se decidía si era el turno del consumidor o productor; si era par, era el turno del productor, pero si era impar, era el turno del consumidor para trabajar. De igual manera para el numero de elementos que en cada turno se iba a consumir o producir, según fuera el caso, se generaba un numero entero aleatorio.

BIBLIOGRAFIA

TEMA 2: PROCESOS E HILOS: CONCURRENCIA. SINCRONIZACIÓN Y COMUNICACIÓN.
(s.f). <https://w3.ual.es/~rguirado/so/tema2.pdf>

Programación concurrente. (s.f).
https://www.ctr.unican.es/asignaturas/procodis_3_II/Doc/Procodis_2_03.pdf

Como hacer un retraso en Python utilizando la función sleep(). (2021).
<https://www.freecodecamp.org/espanol/news/como-hacer-un-retraso-en-python-usando-la-funcion-sleep/>