UNIVERSIDAD DE GUADALAJARA
Red Universitaria e Institución Benemérita de Jalisco
CUCEI
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

Seminario de Solución de Problemas de Sistemas Operativos

Profesor: Becerra Velázquez, Violeta Del Rocío

Alumno: Bustos Ruiz Daniel

Código: 215466901

INCO

Sección: D03

NRC: 119896

Calendario 2024A

Actividad 12: Productor-Consumidor

21/04/2024

Objetivo

Con esta actividad se pretende resolver uno de los problemas más comunes de concurrencia llamado productor-consumidor. En este problema existe un buffer o un contenedor con 22 espacios disponibles, el productor debe crear productos para que el consumidor los tome y los procese. El productor no puede producir si el consumidor esta dentro del buffer, esto aplica de igual forma para el consumidor.

Para este programa se tendrá que realizar el algoritmo haciendo que el productor y el consumidor entren en tiempos distintos y de forma aleatoria al buffer a realizar sus respectivas acciones. Se ejecutará de forma infinita hasta que el usuario lo termine presionando la tecla “ESC”.

Desarrollo

Al comenzar el programa, se definen las variables principales que serán utilizadas durante todo el programa, definiendo una lista con 22 espacios llamada buffer, que será nuestro contenedor, dos objetos que pertenecen a la misma clase “Actor” y cada uno será respectivamente el productor y el consumidor, además de una bandera para terminar el programa.

```
#? _ = Espacio vacio, * = Espacio ocupado
buffer = ["_"] * 22

productor = Actor()
consumidor = Actor()
termina = True
```

La clase “Actor” contiene los siguientes métodos los cuales nos permitirán movernos dentro del buffer. Debido a que Python no maneja las listas con un limite de espacios como hacen algunos otros lenguajes de programación y para este programa, se requiere una lista circular, los métodos “getIndice” y “increaseInd” son los encargados de mantener un índice individual ya sea para el consumidor o productor respectivamente.

El método “getIndice” se explica solo, por lo que básicamente solo regresa el índice actual dentro del buffer. El método “increaseInd” solo aumenta en uno el valor del índice, haciendo que se mueva dentro del buffer.

Existen otros dos métodos llamados “getIndxCircular” y “getNextIndx”. Respectivamente, el primer método se encarga de evaluar el índice real dentro de nuestra lista circular, esto se obtiene con el resto de la división entre el índice del objeto actual y el tamaño actual del buffer. El otro método solo se encarga de regresar el valor del siguiente espacio dentro del buffer de acuerdo con el índice.

Por último, los métodos “getEstado” y “setEstado” se encargan de determinar si el productor o el consumidor están dormidos, respectivamente. El primer método regresa el estado actual del objeto y el segundo se encarga de cambiar ese estado.

```
class Actor:
    def __init__(self) -> None:
        self.indice = 0
        self.dormido = False
        self.consume = 0
        self.produce = 0

    def getIndice(self):
        return self.indice

    def increaseInd(self):
        self.indice += 1

    def getIndxCircular(self, t: int):
        return (self.indice % t)

    def getNextIndx(self, t: int):
        return (self.indice + 1) % t

    #*Si esta dormido o no
    def getEstado(self):
        return self.dormido

    def setEstado(self, e: bool):
        self.dormido = e
```

```
    def getConsume(self):
        return self.consume

    def setConsume(self, c: int):
        self.consume = c

    def getProduce(self):
        return self.produce

    def setProduce(self, p: int):
        self.produce = p
```

En el programa principal se definen tres funciones. La función llamada “dormir” genera un numero al azar, si este numero es par, se despierta al productor y será el que entrará al buffer. Por su parte la función “consumeProduce” genera un numero aleatorio entre 3 y 6 que serán los elementos ya sea a generar o consumir. Finalmente, la función “isEmpty” se encarga de determinar si el espacio en el buffer indicado por el índice que se recibe por parámetro en la función esta vacío o no.

```
#? par = Productor, impar = Consumidor
def dormir():
  if random.randint(1, 100) % 2 == 0:
    return 0
  return 1

def consumeProduce():
  return random.randint(3, 6)

#*Comprueba si el espacio esta vacio
def isEmpty(ind: int):
  if buffer[ind] == "_":
    return True
  return False
```

Para comenzar el programa, se tiene un bucle infinito determinado por la bandera, este bucle solo se detendrá cuando el usuario presione la tecla “ESC”. En cada vuelta se mandan a dormir tanto al productor como al consumidor al principio del bucle.

Se llama a la función “dormir” para determinar quien entrara al buffer, en cualquiera de los casos se cambia su respectivo estado y se imprime en pantalla la situación actual del buffer y los mensajes mostrando el estado actual del productor y consumidor. Aquí también se verifica la entrada de las teclas para saber si se deberá terminar el programa o no.

```
while termina:

  productor.setEstado(False)
  consumidor.setEstado(False)
  #*Genera numero para saber a quien despertar
  if(dormir() == 0):
    productor.setEstado(True)
    os.system('cls')
    sys.stdout.write('\033[H')
    print(buffer)
    print(f"{Fore.CYAN}Productor{Style.RESET_ALL} intentando acceder
al buffer")
    print(f"Consumidor dormido...")
    sys.stdout.flush()
    if msvcrt.kbhit():
      if getch() == b'\x1b':
        break
    time.sleep(1)
  else:
    consumidor.setEstado(True)
    os.system('cls')
    sys.stdout.write('\033[H')
    print(buffer)
    print(f"{Fore.RED}Consumidor{Style.RESET_ALL} intentando acceder
al buffer")
    print(f"Productor dormido...")
    sys.stdout.flush()
    if msvcrt.kbhit():
      if getch() == b'\x1b':
        break
    time.sleep(1)
```

Ya sea el productor o el consumidor que haya despertado. Se verifica el estado, si el espacio esta vacío para ya sea, producir o consumir elementos y comprobando si su contraparte esta dormida.

Cumpliéndose estas condiciones, se genera un numero con la función “consumeProduce” y se inicia un bucle desde 0 hasta el numero generado al azar. En cada vuelta se realiza cada una de las siguientes tareas: se detiene un segundo el programa, muestra en consola el estado del buffer y los espacios que se estén produciendo o consumiendo, dependiendo el caso, además de mostrar en mensajes lo que está pasando en todo momento, se evalúa si el espacio siguiente está disponible para producir o consumir, de lo contrario se termina el bucle y regresa al inicio del bucle principal, finalmente, se verifica la entrada de las teclas para determinar si el usuario quiere terminar el programa, en el caso de presionar la tecla indicada se cambia la bandera, lo que terminara el bucle principal.

Al finalizar de producir o consumir, se duerme respectivamente a quien haya entrado al buffer.

```
if productor.getEstado() and isEmpty(productor.getIndxCircular(22))
and consumidor.getEstado() == False:
    num = consumeProduce()
    for i in range(0, num):
      buffer[productor.getIndxCircular(22)] = "*"
      #*Imprime los datos...
      sys.stdout.write('\033[H')
      print(buffer)
      print(f"{Fore.CYAN}Productor activo{Style.RESET_ALL}.
Intentando generar {Fore.CYAN}{num}{Style.RESET_ALL} elementos")
      print(f"Consumidor dormido...")
      sys.stdout.flush()
      if msvcrt.kbhit():
        if getch() == b'\x1b':
          termina = False
          break
      time.sleep(1)

      #*Si se encuentra con un espacio ocupado, se detiene
      if not isEmpty(productor.getNextIndx(22)):
        break
```

```
      productor.increaseInd()
    productor.setEstado(False)
  elif consumidor.getEstado() and not
isEmpty(consumidor.getIndxCircular(22)) and productor.getEstado() ==
False:
    num = consumeProduce()

    for i in range(0, num):
      buffer[consumidor.getIndxCircular(22)] = "_"

      sys.stdout.write('\033[H')
      print(buffer)
      print(f"{Fore.RED}Consumidor activo{Style.RESET_ALL}.
Intentando consumir {Fore.RED}{num}{Style.RESET_ALL} elementos")
      print(f"Productor dormido...")
      sys.stdout.flush()
      if msvcrt.kbhit():
        if getch() == b'\x1b':
          termina = False
          break
      time.sleep(1)

      #*Si se encuentra con un espacio vacio, se detiene
      if isEmpty(consumidor.getNextIndx(22)):
        break
      consumidor.increaseInd()
    consumidor.setEstado(False)
```

Al ejecutarse el programa, se ve de la siguiente forma en consola:

Conclusión

El problema del productor-consumidor, a pesar de plantearse de forma muy sencilla, tiene sus pequeñas complicaciones las cuales no se tienen muy en cuenta si no es que hasta se comienza a desarrollar.

Debido a las especificaciones de esta actividad, estamos tratando con la versión mas sencilla en la que solo hay un productor y un consumidor, lo que permite hacer todo mas sencillo.

Con este programa, se demuestra el principal problema de sincronización en los sistemas concurrentes, pues de la forma en la que se desarrolló, en la mayoría de las ocasiones siempre habrá uno de los dos elementos, ya sea el productor o el consumidor que sea más rápido que el otro, por lo que siempre se está a la espera de que se produzca o se pueda consumir para continuar con el transcurso normal. Nos permite conocer a grandes rasgos como es que funciona y las complicaciones que se pueden dar dentro de un sistema operativo real.