Universidad Nacional Autónoma de México

Facultad de ingeniería

Sistemas Operativos

Proyecto 2

Concurrencia en situaciones cotidianas

Integrantes del equipo:

López Sugahara Ernesto Danjiro

Fecha de entrega: 07/11/2023

Planteamiento del problema

Se simula el comportamiento de un restaurante que acepta grupos de 4, 2 o 1 personas. En caso de que el restaurante se encuentre en su máxima capacidad, las personas deberán de ir esperando a que se libere espacio. Igualmente, dado a la alta concurrencia que presenta el establecimiento, es posible que se tenga que detener la entrada de personas cuando llegan los grupos de abastecimiento de alimento al restaurante, es decir, si se está surtiendo al establecimiento, no es posible el ingreso de nuevos grupos.

Mecanismos de sincronización empleados

Para el problema planteado se tuvieron que utilizar tres mecanismos de sincronización:

1. Mutex: Este se utiliza para tener un mejor control de las zonas críticas donde se accede a variables globales. Igualmente sirve como torniquete de acceso a los grupos, lo cual trabajará en conjunto con otro de los mecanismos de sincronización empleados: el apagador.

```
mutexAsignacionMesa.acquire()
mesa_asignada = False
while not mesa_asignada:
    for key,value in mesas.items():
        if value[0] == True: # Hay mesas disponibles
            root.after(0,update_gui,key,fam,tam,mesas)
            # Se asigna la mesa al grupo respectivo
            mesa_familia[fam] = key
            mesa_asignada = True
            break
mutexAsignacionMesa.release()
```

Nota: Se observa que se utiliza un poco una espera activa (la cual, al momento de ejecución no es apreciable) ya que esa zona del código estaba generando un problema que no termino de entender. Lo que supuse es que, el acceso era tan rápido y la liberación de las mesas un poco más tardado, por lo que con el ciclo se asegura la liberación de la mesa para que se asigne de forma adecuada, lo cual no genera una demora en el rendimiento del programa.

2. Multiplex: Este es utilizado para el control del flujo de los grupos dentro del establecimiento, es decir, permite el acceso de una cantidad fija de grupos de cierto tamaño, colocando al resto en espera fuera del restaurante hasta que se liberen mesas para el tamaño de grupo específico.

```
    mutex.acquire()
    mutexAcceso.acquire()
    mutexAcceso.release()
    # Como se trabajará con una variable global, se toma un mutex para realizar la asignación de la mesa respectiva
    mutexAsignacionMesa.acquire()
    # mesa_asignada = False
    #while not mesa_asignada:
    for key,value in mesas.items():
        if value[0] == True: # Hay mesas disponibles
            root.after(0,update_gui,key,fam,tam,mesas)
            # Se asigna la mesa al grupo respectivo
            mesa_familia[fam] = key
            # mesa_asignada = True
            break
    mutexAsignacionMesa.release()

    # El grupo estará un tiempo aleatorio dentro del establecimiento
    sleep(random.random() * 20)
    print(f"La familia {fam} de tamaño {tam} terminó de comer")
    # En este caso, se desocupa el lugar y se van
    mutexAsignacionMesa.acquire()
    root.after(0,update_gui,mesa_familia[fam],fam,tam,mesas)
    mutexAsignacionMesa.release()
    mutex.release()
```

El multiplex delimita toda la zona donde se asigna la mesa al grupo que ya entró al establecimiento, así como la liberación de dicha mesa. En este caso, dado que se tienen 4 filas para cada grupo, es posible tener a 4 grupos dentro de la zona del multiplex.

3. Apagador: Este se utiliza para cortar el flujo de personas que acceden al establecimiento cuando se está realizando un reabastecimiento de alimentos para atender a las personas que entrarán posteriormente. Esto con el fin de evitar accidente o problemas cuando se está ingresando la comida y que se asegure que haya los alimentos y productos suficientes para que los clientes consuman lo que deseen. Una vez suministrado el alimento, se suelta el apagador y la gente puede volver a ingresar al restaurante.

```
def llegadaFamilia(fam,tam,mutex,mesas):
    """
        Esta función se  encarga de gestionar la concurrencia de los hilos representados por las familias. Cada uno de lo
        y deberá de analizar si es posible ingresar al rstaurante, es decir, si hay una mesa para el tamaño de su grupo q
        al establecimiento.

        En este caso, se utilizaron varios mutex para zonas críticas de operación en donde se llegaban a utilizar variabl
        lo son: mesa_familia. Igualmente, como se observa por la naturaleza del problema, en este caso fue necesario tene
        cada uno de los tamaños de grupo, ya que solo puede haber cierto número de una categoría de grupo dentro del esta
        la solución se realizó mediante un multiplex que pone en espera a los grupos que van llegando hasta que termine d

        Las variables que se reciben son:
        1. El identificador de la familia
        2. El tamaño de la familia
        3. El multiplex asociado al tamaño de grupo
        4. La información de las mesas para el tamaño de grupo señalizado. Esto con el fin de modificar la interfaz gráfi
    """
    mutex.acquire()
    mutexAcceso.acquire()
    mutexAcceso.release()
    # Como se trabajará con una variable global, se toma un mutex para realizar la asignación de la mesa respectiva
    mutexAsignacionMesa.acquire()
    mesa_asignada = False
```

```python
def surtiendoRestaurante():
    """
        En este caso se toma el mutex de la puerta de acceso, lo que bloquea la entrada de nuevos grupos.
        Dado un tiempo, se estará surtiendo al restaurante con nuevos alimentos para los próximos grupos.
        El mutex de acceso sirve como torniquete para los grupos que van llegando al restaurante
    """
    global surtiendo
    global evento
    # En este punto ya no puede acceder nadie hasta que el surtidor haya terminado de realizar su tarea
    mutexAcceso.acquire()
    # En este punto se tiene que bloquear la salida
    print("Se está surtiendo al restaurante")
    sleep(20)
    print("Se terminó se surtir al restaurante")
    surtiendo.destroy()
    evento = Button(
            frame_mesas,
            text="Llegada de productos",
            width=15,
            height=3,
            command=threadLlegada
    )
    evento.grid(row=6,column=2)
    mutexAcceso.release()
```

En este caso se observa que, al llegar el momento de surtir el restaurante se activa el apagador, lo que prohíbe la entrada al torniquete de los clientes.

Lógica de operación

Estado compartido

En este caso se comparten los siguientes elementos globales (que son ajenos a la interfaz gráfica):

```python
multiplexCuatro = Semaphore(mesaParaCuatro)
multiplexDos = Semaphore(mesaParaDos)
multiplexUno = Semaphore(mesaParaUno)
mesasCuatro = {}
mesasDos = {}
mesasUno = {}

mutexAsignacionMesa = Semaphore(1)
mutexLiberacionMesa = Semaphore(1)
mutexAcceso = Semaphore(1)

# Determinar qué famila está utilizando qué mesa
mesa_familia = {}
```

Los elementos de mayor interés y cuidado son los diccionarios que contienen la información relacionada a las mesas, ya que está determinará el estado actual del restaurante. Igualmente, el diccionario que asocia a las familias con las respectivas mesas, ya que, si no se almacena de forma adecuada esta información, algunos grupos podrían perder su lugar en el establecimiento.

Descripción algorítmica de cada hilo

Una vez abierto el restaurante, se comienza a simular la llegada de los clientes. En este caso, se tiene la siguiente función:

```
# Se analiza el tamaño de la familia para atender su llegada
def llegada(fam,tam):
    """
        Esta función se encarga de gestar la llegada de las diferentes fam
        agregar el tiempo aleatorio de llegada en el hilo secundario (ya q
        dentro del hilo principal, pausando la interfaz gráfica).

        Dentro de esta función, lo único que se analiza es el tamaño de la
        la llegada de dicha familia.
    """
    sleep(random.random() * random.randint(20,40))
    print(f"La familia {fam} de tamaño {tam} llegó al establecimiento")
    # Se deberá de validar la disponibilidad para este tamaño de familia
    if tam == 1:
        llegadaFamilia(fam,tam,multiplexUno,mesasUno)
    elif tam == 2:
        llegadaFamilia(fam,tam,multiplexDos,mesasDos)
    else:
        llegadaFamilia(fam,tam,multiplexCuatro,mesasCuatro)
```

En este caso, el hilo se duerme por un momento (esto para simular llegadas aleatorias en tiempos diferentes). Esto se tuvo que hacer en este punto ya que el hilo principal está dado por la interfaz gráfica. Una vez dentro, se determina el tamaño del grupo, con lo que se llamará a la función de interés con los respectivos argumentos:

1. Fam: Es el identificador del grupo.
2. Tam: Es el tamaño del grupo.
3. Posteriormente se manda el multiplexor asociado al tamaño del grupo.
4. Se manda la información de las mesas relacionadas al tamaño del grupo.

Con esto, se tiene la función que determina el comportamiento del hilo:

```python
def llegadaFamilia(fam,tam,mutex,mesas):
    """
        Esta función se  encarga de gestionar la concurrencia de l
        y deberá de analizar si es posible ingresar al rstaurante,
        al establecimiento.

        En este caso, se utilizaron varios mutex para zonas crític
        lo son: mesa_familia. Igualmente, como se observa por la n
        cada uno de los tamaños de grupo, ya que solo puede haber
        la solución se realizó mediante un multiplex que pone en e

        Las variables que se reciben son:
        1. El identificador de la familia
        2. El tamaño de la familia
        3. El multiplex asociado al tamaño de grupo
        4. La información de las mesas para el tamaño de grupo señ
    """
    mutex.acquire()
    mutexAcceso.acquire()
    mutexAcceso.release()
    # Como se trabajará con una variable global, se toma un mutex
    mutexAsignacionMesa.acquire()
    mesa_asignada = False
    while not mesa_asignada:
        for key,value in mesas.items():
            if value[0] == True: # Hay mesas disponibles
                root.after(0,update_gui,key,fam,tam,mesas)
                # Se asigna la mesa al grupo respectivo
                mesa_familia[fam] = key
                mesa_asignada = True
                break
    mutexAsignacionMesa.release()

    # El grupo estará un tiempo aleatorio dentro del establecimien
    sleep(random.random() * 20)
    print(f"La familia {fam} de tamaño {tam} terminó de comer")
    # En este caso, se desocupa el lugar y se van
    root.after(0,update_gui,mesa_familia[fam],fam,tam,mesas)
    mutex.release()
```

Se tiene el siguiente flujo:

1. Se toma el multiplexor para entrar al establecimiento.

2. En caso de haber espacio, se pasará el torniquete de acceso. Si el restaurante está siendo surtidor, este estará apagado y por lo tanto tendrá que esperar para ingresar al restaurante.

3. Posteriormente se asigna la mesa al grupo, para esto se analiza qué mesa del tamaño de grupo especificado está disponible para ser asignada.

4. Después la familia come por un tiempo aleatorio.

5. Finalmente se libera la mesa utilizada por dicha familia, actualizando de este modo el hilo principal (la interfaz) para mostrar la disponibilidad de la mesa.

Finalmente, el grupo abandona el establecimiento.

Interacciones

- Como se observa, el hilo principal está constantemente interactuando con los hilos secundarios que representan a los clientes, ya que estos mandan la información necesaria para ir actualizando la interfaz.

- Los hilos secundarios interactúan entre sí mediante la asignación y liberación de mesas, teniendo de este modo hilos que se encuentran dentro del restaurante utilizando mesas, los cuales mandaran señales a los hilos que se encuentran fuera a la espera de una mesa.

- Igualmente, el hilo principal tiene el poder de utilizar un apagador que romperá el flujo de acceso de los hilos secundarios cuando se realiza una entrega de productos al restaurante.

Entorno de desarrollo

El lenguaje de programación empleado es Python:

```
➜  LopezErnesto git:(main) ✗ python3 --version
Python 3.11.6
```

Para el correcto funcionamiento del programa es necesario contar con las librerías siguientes:

1. Tkinter: Esta es la librería que permite utilizar la interfaz gráfica modelada.
2. Librerías estándar como: time, random y threading.

Nota: El programa fue probado en sistema MacOS con la versión Sonoma:

```
➜  ~ sw_vers
ProductName:            macOS
ProductVersion:         14.0
```

Ejemplos de funcionamiento:

Para mostrar de forma correcta el funcionamiento, se irá mostrando por partes:

1. Se inicia el programa, mostrando la interfaz básica:

2. Abriendo el restaurante, comienzan a llegar los grupos (hilos) tomando las respectivas mesas, esto se observa a continuación:

```
La familia 20 de tamaño 4 llegó al establecimiento
La familia 38 de tamaño 4 llegó al establecimiento
La familia 28 de tamaño 1 llegó al establecimiento
La familia 9 de tamaño 2 llegó al establecimiento
La familia 19 de tamaño 1 llegó al establecimiento
La familia 43 de tamaño 2 llegó al establecimiento
La familia 24 de tamaño 1 llegó al establecimiento
```

El flujo continuará. Se darán situaciones donde todas las mesas están ocupadas y los grupos que van llegando deberán de esperar. Esto se muestra a continuación:

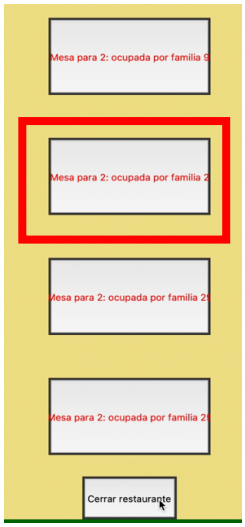
3. Cuando un grupo termina de comer, libera la respectiva mesa que está ocupando. Esto se muestra a continuación:

```
La familia 29 de tamaño 2 llegó al establecimiento
La familia 31 de tamaño 1 terminó de comer
La familia 25 de tamaño 2 llegó al establecimiento
```

4. Cuando no hay espacio, los grupos esperan hasta que termine otro grupo. Por ejemplo, las mesas para 2 están ocupadas, por lo que la llegada de otro grupo causará que este tenga que esperar como se muestra a continuación:

```
La familia 25 de tamaño 2 llegó al establecimiento
La familia 2 de tamaño 2 llegó al establecimiento
La familia 35 de tamaño 2 llegó al establecimiento
La familia 43 de tamaño 2 terminó de comer
La familia 22 de tamaño 2 llegó al establecimiento
```

Por lo que, cuando acaba la familia 43, la familia 2 ingresa al establecimiento y procede a tomar la mesa como se muestra a continuación:



5. Finalmente, está el caso en que el restaurante comienza a surtirse. Como se observa a continuación, cuando el restaurante se está surtiendo, las familias siguen llegando sin embargo no se ocupan las mesas. Las familias que ya estaban dentro pueden terminar de comer y salir sin problemas del establecimiento:

```
Se está surtiendo al restaurante
La familia 11 de tamaño 4 llegó al establecimiento
La familia 18 de tamaño 2 llegó al establecimiento
La familia 6 de tamaño 2 llegó al establecimiento
La familia 34 de tamaño 2 llegó al establecimiento
La familia 35 de tamaño 2 terminó de comer
La familia 40 de tamaño 4 llegó al establecimiento
La familia 16 de tamaño 4 llegó al establecimiento
La familia 1 de tamaño 1 llegó al establecimiento
La familia 33 de tamaño 2 llegó al establecimiento
La familia 15 de tamaño 1 llegó al establecimiento
La familia 19 de tamaño 1 terminó de comer
La familia 8 de tamaño 2 llegó al establecimiento
La familia 12 de tamaño 2 llegó al establecimiento
La familia 48 de tamaño 2 llegó al establecimiento
La familia 24 de tamaño 1 terminó de comer
La familia 32 de tamaño 2 llegó al establecimiento
La familia 30 de tamaño 4 terminó de comer
La familia 22 de tamaño 2 terminó de comer
La familia 13 de tamaño 4 llegó al establecimiento
La familia 25 de tamaño 2 terminó de comer
```

Restaurante en el punto exacto en que se comienza a surtir:

Después de un tiempo, se dio el caso de que todas las familias terminaron de comer, por lo que se vació el restaurante:

En el momento que se suelta el apagador, comienza a reingresar la gente que estuvo llegando al establecimiento:

Nota: Se exageró el tiempo en que se surte el restaurante para poder observar el comportamiento del apagador.