CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS (CUCEI)

Departamento de ciencias computacionales

Seminario de solución de problemas de uso, adaptación, explotación de sistemas operativos

Violeta del Rocío Becerra Velázquez

Jose Pedro Reyes Alvarez

222790897

Ingeniería Informática (INNI)

D02

2.5 Soluciones al interbloqueo, filósofos

30 de abril del 2025

Índice

Investigación y explicación del problema

El interbloqueo, también conocido como bloqueo mutuo o abrazo mortal (deadlock en inglés), es un problema que puede presentarse en los sistemas operativos y en sistemas donde existen procesos concurrentes que compiten por recursos. Esta situación ocurre cuando un conjunto de procesos queda detenido de forma indefinida debido a que cada uno de ellos está esperando que otro libere un recurso que necesita para continuar su ejecución. En otras palabras, se forma una cadena de dependencia circular en la que ningún proceso puede avanzar porque todos están esperando por recursos que están siendo retenidos por los demás.

Para que se presente un interbloqueo, deben cumplirse simultáneamente cuatro condiciones, conocidas como las condiciones de Coffman. La primera es la mutua exclusión, que implica que al menos uno de los recursos involucrados solo puede ser utilizado por un proceso a la vez. La segunda condición es la retención y espera, la cual se da cuando un proceso que ya tiene asignado al menos un recurso, solicita otros recursos adicionales que están siendo utilizados por otros procesos, sin liberar los que ya tiene. La tercera es la no expropiación, que establece que los recursos no pueden ser arrebatados a un proceso por la fuerza, sino que deben ser liberados voluntariamente por el proceso que los posee. Finalmente, la espera circular se refiere a la existencia de una cadena de procesos donde cada uno está esperando un recurso que posee el siguiente, cerrando un ciclo en el que el último espera un recurso que posee el primero.

Dado que los interbloqueos pueden provocar que partes del sistema queden inactivas indefinidamente, es importante implementar estrategias para tratarlos. Una de estas estrategias es la prevención del interbloqueo, que consiste en diseñar el sistema de forma que al menos una de las condiciones de Coffman no pueda cumplirse, lo que evita que el interbloqueo llegue a producirse. Por ejemplo, se puede evitar la espera circular imponiendo un orden en la solicitud de recursos, de modo que los procesos deban solicitarlos siguiendo una secuencia preestablecida.

Otra estrategia es la evitación del interbloqueo, que no elimina directamente las condiciones necesarias, pero sí permite que el sistema evite entrar en un estado inseguro. Esta técnica requiere que el sistema conozca de antemano la cantidad máxima de recursos que cada proceso podría necesitar y utilice algoritmos como el del banquero de Dijkstra para decidir si es seguro asignar los recursos solicitados.

En los casos donde no se evita ni se previene, se puede optar por la detección y recuperación del interbloqueo. En este enfoque, el sistema permite que el interbloqueo ocurra, pero implementa mecanismos para detectarlo (por ejemplo, mediante grafos de asignación de recursos o matrices de espera) y luego toma acciones para recuperar el sistema. Las acciones pueden incluir finalizar uno o varios procesos involucrados en el interbloqueo, o quitarles recursos, aunque esto puede afectar la estabilidad del sistema o generar pérdida de datos.

Por último, existe una cuarta estrategia conocida como la ignorancia del problema o el enfoque del “avestruz”. Este consiste en no hacer nada para prevenir, evitar ni detectar interbloqueos, bajo la suposición de que estos ocurren rara vez y que el costo de implementar soluciones sería mayor que el impacto del propio problema. Esta estrategia se emplea en algunos sistemas como UNIX, donde se considera que los desarrolladores deben diseñar sus aplicaciones para evitar interbloqueos por su cuenta.

En conclusión, el interbloqueo es un problema serio en entornos concurrentes, pero existen diversas estrategias para gestionarlo dependiendo del tipo de sistema y sus requerimientos de rendimiento, seguridad y confiabilidad.

El problema de los filósofos comensales

El problema de los filósofos comensales es un problema clásico en informática que fue planteado por Edsger Dijkstra en 1965. Este problema se utiliza para ejemplificar situaciones de interbloqueo, condiciones de carrera, y sincronización en entornos concurrentes. Es especialmente útil para estudiar cómo múltiples procesos pueden compartir recursos sin interferir entre sí y sin quedar bloqueados indefinidamente.

Planteamiento y requerimientos del problema

- El problema se plantea de la siguiente manera:
- Hay cinco filósofos sentados alrededor de una mesa circular.
- Cada filósofo alterna entre dos actividades: pensar y comer.
- Para comer, un filósofo necesita dos tenedores (uno a su izquierda y otro a su derecha).
- Entre cada par de filósofos hay un solo tenedor, por lo que hay cinco tenedores en total.
- Los filósofos deben tomar ambos tenedores a la vez para poder comer, y luego liberarlos cuando terminen.

Inconvenientes involucrados

Interbloqueo (Deadlock):
Si todos los filósofos toman primero el tenedor a su derecha y esperan el de la izquierda (o viceversa), cada uno estará esperando un recurso que tiene otro, y ninguno podrá comer. Esto genera un ciclo de espera, es decir, un interbloqueo.

Hambre (Starvation):
Incluso si se evita el interbloqueo, puede ocurrir que algunos filósofos coman más frecuentemente que otros. Esto puede causar que uno o más filósofos esperen indefinidamente, sin llegar nunca a comer.

Condiciones de carrera:
Si los procesos (filósofos) acceden a los tenedores sin una adecuada sincronización, puede haber conflictos o resultados inesperados.

Posibles soluciones

Asignación jerárquica de recursos:
Se impone un orden en la toma de tenedores. Por ejemplo, todos los filósofos deben tomar primero el tenedor con el número más bajo. Así, se rompe la condición de espera circular y se evita el interbloqueo.

Utilizar un camarero (waiter o monitor):
Un proceso centralizado controla cuándo un filósofo puede tomar los tenedores. Solo se permite que un filósofo coma si ambos tenedores están disponibles. Este enfoque garantiza que no se produzca interbloqueo.

Permitir que solo N - 1 filósofos intenten comer a la vez:
Si solo se permite que cuatro de los cinco filósofos coman simultáneamente, siempre habrá al menos un tenedor disponible, lo que evita el ciclo completo de espera.

Algoritmos de prioridad o turnos:
Se asignan turnos o prioridades para comer, de modo que todos tengan la oportunidad de acceder a los recursos, evitando el problema del hambre.

Tiempos aleatorios para pensar y comer:
Aunque no garantiza evitar el interbloqueo, introducir aleatoriedad puede reducir la probabilidad de que ocurra, haciendo que el sistema sea más dinámico y menos propenso a bloqueos.

Solución Propuesta

Para esta solución se implementa una simulación gráfica del clásico problema de los filósofos comensales usando Python y la biblioteca Tkinter para la interfaz gráfica, junto con threading para el manejo de concurrencia mediante hilos y semáforos. El objetivo principal es representar gráficamente cómo los filósofos piensan y comen sin incurrir en interbloqueo, usando imágenes para representar los distintos estados.

Componentes del programa:

Imágenes de fondo y estados:

Se utiliza una imagen circular de una mesa con cinco platos como fondo.

Se cargan dos imágenes tipo emoji: una para representar a un filósofo pensando y otra para uno comiendo.

Interfaz gráfica (Tkinter):

Se utiliza un Canvas sobre el que se dibujan las imágenes de los filósofos, la mesa y los contadores de comidas.

Cada filósofo tiene su posición específica sobre la mesa circular.

Manejo de concurrencia:

Cada filósofo es representado por un hilo (Thread) que alterna entre pensar y comer.

Se usa un Lock principal para evitar condiciones de carrera al tomar los tenedores (simbolizados por los lugares entre cada filósofo).

Cada tenedor se representa con un semáforo (threading.Semaphore), asegurando que no se pueda tomar si ya está en uso.

Condición de término:

Cada filósofo debe comer al menos 6 veces. Una vez que todos han alcanzado esa meta, el programa se cierra automáticamente.

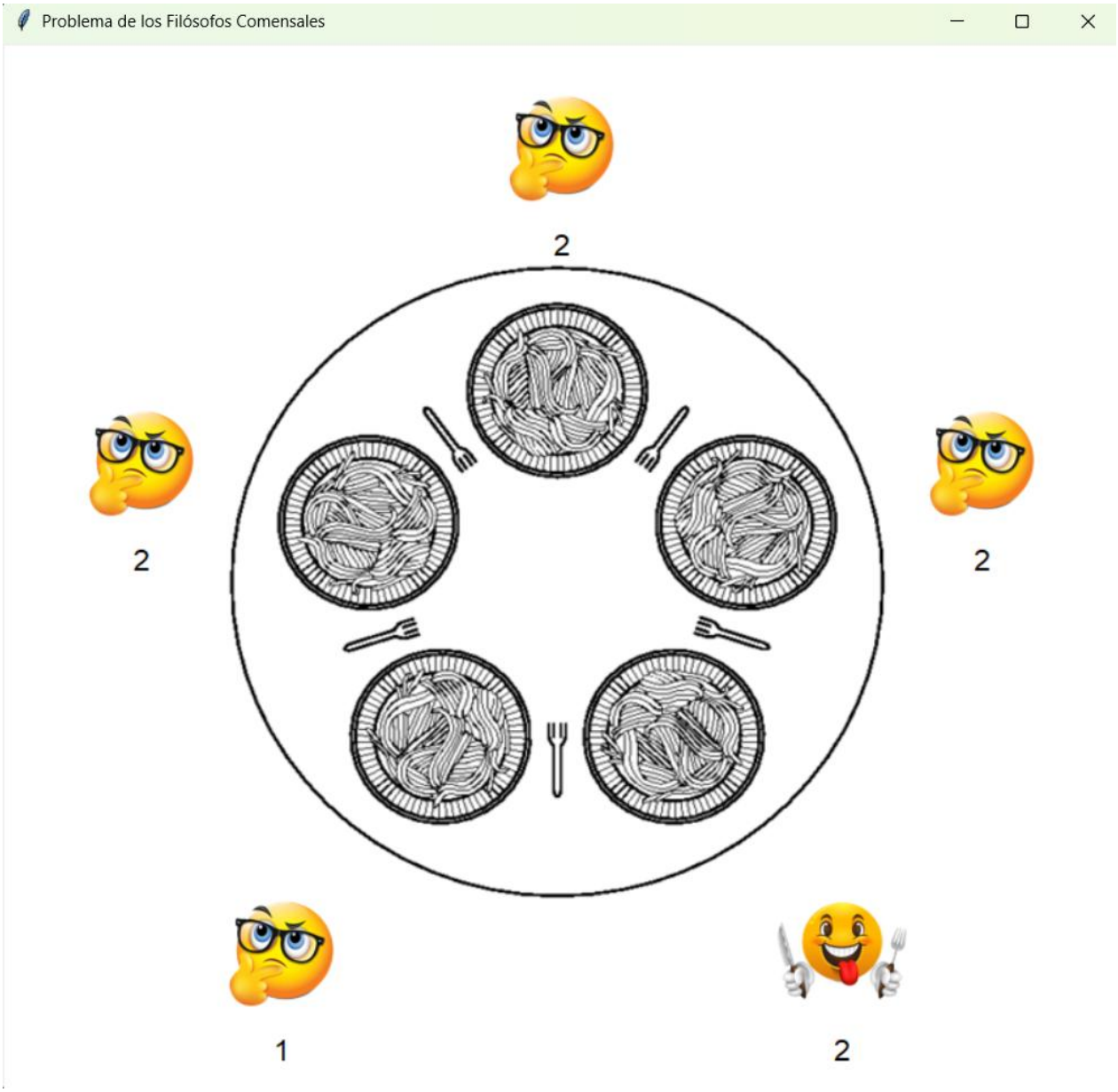
Actualización visual en tiempo real:

Cada vez que un filósofo cambia de estado, su imagen se actualiza (pensando o comiendo).

Además, se muestra un contador numérico que indica cuántas veces ha comido cada filósofo.

Solución al interbloqueo:

El interbloqueo es evitado gracias al uso de semáforos con una política cuidadosa: los filósofos siempre intentan tomar ambos tenedores de forma secuencial, y si no pueden, esperan hasta que estén disponibles. Esto evita que todos se queden esperando indefinidamente.



Explicación del codigo

Aquí se define la función tarea_filosofo(i), que representa el comportamiento de cada filósofo numerado del 0 al 4. Cada filósofo ejecuta un ciclo donde alterna entre pensar y comer, hasta que ha comido al menos 6 veces.

pensar(i): simula el estado de pensamiento del filósofo. En la interfaz, se muestra la imagen correspondiente (emoji de pensar).

tenedor_izq y tenedor_der: representan los semáforos de los tenedores ubicados a la izquierda y derecha del filósofo. Se accede a ellos usando el índice i y el siguiente en sentido circular (i + 1) % 5.

with lock:: garantiza que solo un filósofo a la vez pueda tomar los tenedores, previniendo condiciones de carrera.

acquire(): toma un tenedor si está disponible. Si no, espera hasta que lo esté.

comer(i): cambia la imagen a la del filósofo comiendo y simula ese estado durante un tiempo aleatorio.

comidas[i] += 1: incrementa el contador de comidas del filósofo.

release(): libera ambos tenedores para que puedan ser usados por otros filósofos.

canvas.after(0, actualizar_imagen, i): solicita a la interfaz gráfica que actualice inmediatamente la imagen del filósofo con base en su nuevo estado.

time.sleep(random.uniform(1, 2)): hace que el filósofo espere un tiempo aleatorio antes de volver a pensar.

```python
# --- Función del filósofo ---
def tarea_filosofo(i):
    global conteo_comidas
    tenedor_izquierdo = tenedores[i]
    tenedor_derecho = tenedores[(i + 1) % 5]

    while conteo_comidas[i] < 6:
        # Pensar
        lienzo.itemconfig(filosofos[i], image=foto_pensando)
        ventana.update()
        time.sleep(random.uniform(1, 3))

        # Tomar tenedores (evita interbloqueo alternando el orden)
        primero, segundo = (tenedor_izquierdo, tenedor_derecho) if i % 2 == 0 else (tenedor_derecho, tenedor_izquierdo)
        primero.acquire()
        segundo.acquire()

        # Comer
        lienzo.itemconfig(filosofos[i], image=foto_comiendo)
        conteo_comidas[i] += 1
        lienzo.itemconfig(contadores[i], text=str(conteo_comidas[i]))
        ventana.update()
        time.sleep(random.uniform(1, 2))

        # Soltar tenedores
        primero.release()
        segundo.release()

    # Después de comer 6 veces, vuelve a pensar
    lienzo.itemconfig(filosofos[i], image=foto_pensando)
    ventana.update()
```

Conclusión

Realizar este programa me resultó bastante interesante y retador, ya que estuve experimentando con cómo manejar Tkinter y poner imágenes para que se viera mejor que solo hacerlo en consola. Me puse a probar varias formas de cómo mostrar a los

filósofos pensando o comiendo, y logré que se viera más visual y entendible usando emojis e imágenes. También tuve que hacer varias pruebas para que los hilos funcionaran correctamente y no se bloquearan entre ellos. Aunque al principio me costó entender bien cómo usar los semáforos y manejar los tenedores, con el tiempo fui entendiendo mejor cómo evitar el interbloqueo. En general, fue un ejercicio que me ayudó a reforzar lo aprendido y me motivó a ir más allá de solo resolverlo por texto.

Bibliografía

LinkedIn. (s. f.). ¿Cómo resuelves un interbloqueo del sistema operativo? LinkedIn. Recuperado el 30 de abril de 2025, de https://es.linkedin.com/advice/0/how-do-you-resolve-operating-system-deadlock?lang=es&lang=es

Universidad Nacional del Sur. (2020). Módulo 08 - Interbloqueos. https://cs.uns.edu.ar/~so/data/apuntes/SO-2020-mod%2008.pdf

Llamas, C. (1997). La cena de los filósofos. Universidad de Valladolid. https://www2.infor.uva.es/~cllamas/concurr/pract97/immartin/index.html

Portillo, F. (s. f.). La cena de los filósofos. pacoportillo.es. Recuperado el 30 de abril de 2025, de https://pacoportillo.es/informatica-avanzada/programacion-multiproceso/la-cena-de-los-filosofos/