Universidad de Guadalajara

Centro Universitario de Ciencias Exactas E Ingenierías

Ingeniería Informática

Juan Antonio Ramírez Aguilar

(212482507)

Seminario de Solución de Problemas de Uso, Adaptación, Explotación de Sistemas Operativos

Sección: D02

Mtra. Becerra Velázquez Violeta del Roció

“Actividad 9 (2.2 Concurrencia. Exclusión mutua, sincronización y problemas de control)”

Indicé

Contenido de imágenes

Glosario

A continuación, se presenta una lista de términos a modo de glosario de los capítulos 4 y 5 del Libro "Sistemas Operativos", del autor *Stallings*.

Capítulo 4 - Concurrencia: exclusión mutua y sincronización

1. **Concurrencia**: Se refiere a la capacidad de un sistema para ejecutar múltiples procesos o hilos de forma que parezcan solaparse en el tiempo. Aunque en un procesador de un solo núcleo solo una tarea se ejecuta en un instante dado, la rápida alternancia entre ellas crea la ilusión de simultaneidad.

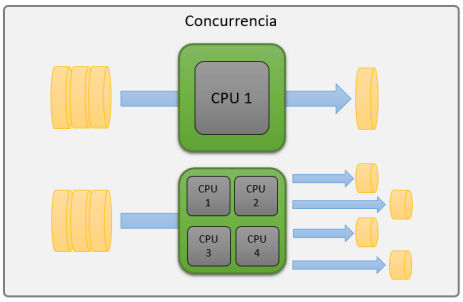

Imagen 1. Concurrencia en un procesador.

2. **Proceso**: Es una instancia de un programa en ejecución. Cada proceso tiene su propio espacio de memoria, datos y estado de ejecución. Es la unidad fundamental de trabajo que gestiona el sistema operativo.

3. **Hilo** (Thread): Una unidad de ejecución más pequeña dentro de un proceso. Múltiples hilos pueden coexistir en el mismo proceso, compartiendo su espacio de memoria y recursos, lo que permite realizar varias tareas a la vez de manera más eficiente que creando múltiples procesos.

4. **Exclusión Mutua**: Un mecanismo para asegurar que, si un proceso está utilizando un recurso compartido (como una variable o un archivo), ningún otro proceso pueda acceder a él al mismo tiempo. Esto es fundamental para prevenir inconsistencias en los datos.

5. **Sección Crítica**: Es la parte del código de un programa donde se accede a un recurso compartido. La exclusión mutua garantiza que solo un hilo o proceso pueda estar ejecutando su sección crítica en un momento determinado.

6. **Sincronización:** Es la coordinación de la ejecución de múltiples procesos o hilos para asegurar que cooperen correctamente y no interfieran entre sí. Los semáforos y monitores son herramientas comunes para la sincronización.

7. **Interbloqueo (Deadlock)**: Una situación en la que dos o más procesos se bloquean mutuamente, cada uno esperando a que el otro libere un recurso que necesita. Ninguno puede continuar, creando un punto muerto.

8. **Inanición (Starvation)**: Ocurre cuando un proceso es constantemente ignorado por el planificador y no obtiene los recursos que necesita para ejecutarse, a menudo porque otros procesos de mayor prioridad lo acaparan todo.

9. **Espera Activa (Busy-Waiting)**: Una técnica en la que un proceso comprueba repetidamente una condición en un bucle continuo (por ejemplo, while (condicion == false) {}) en lugar de bloquearse. Consume mucho tiempo de CPU y generalmente es ineficiente.

10. **Espera Bloqueante (Blocking Wait)**: A diferencia de la espera activa, un proceso que necesita esperar por un recurso se "bloquea" o "duerme". El sistema operativo lo saca de la cola de listos y no consume CPU hasta que el recurso esté disponible y el sistema lo "despierte".

11. **Semáforo**: Una variable especial o tipo de dato abstracto que se utiliza para controlar el acceso a recursos compartidos. Actúa como un contador que gestiona el número de procesos que pueden acceder a un recurso simultáneamente.

12. **Operación wait (o P)**: Una operación fundamental sobre un semáforo. Decrementa el valor del semáforo. Si el valor resultante es negativo, el proceso que la ejecuta se bloquea hasta que otro proceso realice una operación signal.

13. **Operación signal (o V)**: La otra operación fundamental sobre un semáforo. Incrementa su valor. Si hay procesos bloqueados esperando en ese semáforo, uno de ellos es desbloqueado.

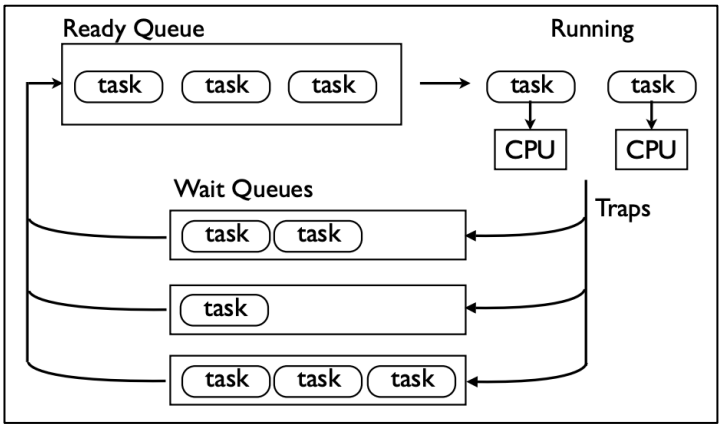

Imagen 2. Ejemplo de Corrutina

14. **Corrutina**: Un tipo de subrutina que permite suspender y reanudar su ejecución en puntos específicos. A diferencia de las subrutinas tradicionales, las corrutinas tienen una relación más simétrica y pueden cederse el control entre ellas, en lugar de una estricta relación llamador-llamado.

15. **Problema del Productor/Consumidor**: Un problema clásico de sincronización donde uno o más procesos "productores" generan datos que son consumidos por uno o más procesos "consumidores". El reto es gestionar el búfer compartido donde se almacenan temporalmente los datos.

16. **Búfer (Buffer)**: Un área de memoria utilizada para almacenar datos temporalmente mientras se transfieren de un lugar a otro. Es central en el problema del productor/consumidor.

17. **Algoritmo de Dekker**: Fue el primer algoritmo de software demostrablemente correcto para resolver el problema de la exclusión mutua entre dos procesos sin usar instrucciones especiales de hardware.

18. **Algoritmo de Peterson**: Una solución de software más simple que el algoritmo de Dekker para el problema de la exclusión mutua entre dos procesos. Utiliza una variable de turno y un array booleano para coordinar el acceso.

19. **Algoritmo de la Panadería (de Lamport)**: Una solución de software para el problema de la exclusión mutua que funciona para N procesos. Se inspira en el sistema de turnos de una panadería: cada proceso obtiene un número y el que tenga el número más bajo entra en la sección crítica.

20. **Monitor**: Una estructura de programación de alto nivel que simplifica la sincronización. Encapsula datos compartidos y los procedimientos que operan sobre ellos, garantizando automáticamente la exclusión mutua. Ofrece mecanismos de espera y notificación (variables de condición) para una coordinación más compleja.

Capítulo 5 - Concurrencia: interbloqueo e inanición

1. **Interbloqueo (Deadlock)**: Es el bloqueo permanente de un conjunto de procesos que compiten por recursos o se comunican entre sí. Ninguno puede continuar porque cada uno está esperando un recurso que posee otro proceso del mismo conjunto, creando un ciclo de espera irrompible sin intervención externa.

2. **Inanición (Starvation)**: Ocurre cuando un proceso es repetidamente ignorado por el planificador y nunca obtiene los recursos que necesita para ejecutarse, a pesar de que estos puedan estar disponibles. A menudo, procesos con mayor prioridad acaparan los recursos, dejando al proceso "hambriento" en espera indefinida.

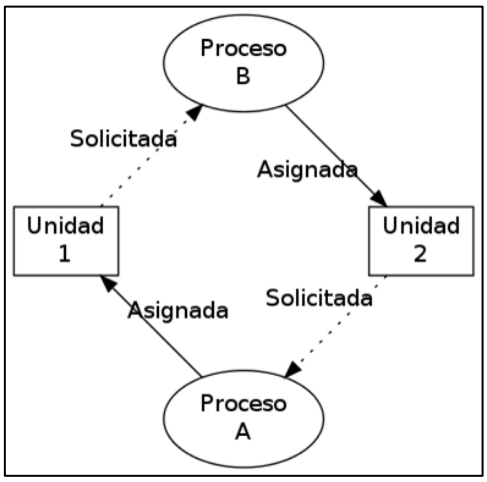

Imagen 3. Ejemplo de Inanición

3. **Recurso Reutilizable (Reusable Resource)**: Un recurso que no se consume con su uso y puede ser utilizado por un solo proceso a la vez. Una vez que el proceso lo libera, puede ser asignado a otro. Ejemplos típicos son la CPU, la memoria o una unidad de disco.

4. **Recurso Consumible (Consumable Resource)**: Un recurso que se crea y se destruye durante la operación del sistema. Típicamente, un proceso lo "produce" y otro lo "consume", desapareciendo después de su uso. Un ejemplo clásico son los mensajes o las señales entre procesos.

5. **Prevención de Interbloqueo (Deadlock Prevention)**: Una estrategia proactiva que busca eliminar la posibilidad de interbloqueos diseñando el sistema de tal forma que no se cumpla

al menos una de las cuatro condiciones necesarias para que ocurra (exclusión mutua, retención y espera, no expropiación o espera circular).

6. **Detección de Interbloqueo (Deadlock Detection)**: Una estrategia reactiva donde se permite que el sistema entre en un estado de interbloqueo. El sistema operativo ejecuta periódicamente un algoritmo para buscar ciclos de espera. Si detecta uno, aplica medidas para recuperarse, como abortar un proceso.

7. **Predicción de Interbloqueo (Deadlock Avoidance)**: Una estrategia conservadora. Antes de conceder una petición de recurso, el sistema analiza si hacerlo podría llevar a un futuro interbloqueo. La petición solo se concede si el estado resultante sigue siendo un estado seguro.

8. **Espera Circular (Circular Wait)**: Una de las cuatro condiciones necesarias para el interbloqueo. Se produce cuando existe una cadena cerrada de procesos, donde el proceso P1 espera un recurso de P2, P2 espera uno de P3, y así sucesivamente hasta que un último proceso Pn espera un recurso de P1.

9. **Estado Seguro (Safe State)**: Un estado del sistema en el que existe al menos una secuencia de ejecución en la que todos los procesos pueden completar su trabajo. Desde un estado seguro, el sistema puede garantizar que no se producirá un interbloqueo.

10. **Estado Inseguro (Unsafe State)**: Un estado del sistema que no es seguro. Un estado inseguro no implica necesariamente que haya un interbloqueo, pero sí que existe la posibilidad de que ocurra si los procesos solicitan recursos de una manera desafortunada.

11. **Algoritmo del Banquero (Banker's Algorithm)**: Es el algoritmo clásico para la predicción de interbloqueos. Simula la gestión de recursos de un banco: solo concede un "préstamo" (recurso) si puede garantizar que, aun en el peor de los casos, podrá satisfacer las necesidades de todos sus "clientes" (procesos).

RECURSOS	2	4	1	4

DISPONIBLES	1	1	0	2

	MAXIMO DE RECURSOS			
	R1	R2	R3	R4
P1	1	1	0	1
P2	0	1	0	1
P3	2	0	0	1
P4	1	2	1	0
P5	0	0	0	2

	RECURSOS UTILIZADOS			
	R1	R2	R3	R4
P1	1	1	0	1
P2	0	1	0	1
P3	0	0	0	0
P4	0	2	1	0
P5	0	0	0	1

	RECURSOS NECESARIOS			
	R1	R2	R3	R4
P1	0	0	0	0
P2	0	0	0	0
P3	0	0	0	0
P4	1	0	0	0
P5	0	0	0	1

De la misma forma realizamos para el proceso 4 y 5

Imagen 4. Ejemplo con matrices del algoritmo del banquero.

12. **Cierre de Giro (Spin Lock)**: Un tipo de bloqueo donde un proceso que intenta adquirir un recurso ocupado espera en un bucle activo ("spinning"), comprobando constantemente si el recurso se ha liberado. Es eficiente para esperas muy cortas, ya que evita la sobrecarga de suspender el proceso.

13. **Cierre de Suspensión (Suspend Lock)**: Un bloqueo donde un proceso que no puede adquirir el recurso es suspendido por el sistema operativo. El proceso entra en estado de bloqueo y no consume CPU hasta que el recurso es liberado y el sistema lo "despierta". Es ideal para esperas largas.

14. **Jerarquía de Cierres (Lock Hierarchy)**: Una técnica de prevención de interbloqueos que asigna un orden o nivel a todos los bloqueos (recursos). Un proceso solo puede solicitar un bloqueo de nivel superior al que ya posee, rompiendo así la posibilidad de una espera circular.

15. **Problema de la Cena de los Filósofos (Dining Philosophers Problem)**: Un problema clásico de concurrencia que ilustra los desafíos del interbloqueo y la inanición. Cinco filósofos sentados en una mesa redonda necesitan dos tenedores para comer, pero solo hay un

tenedor entre cada par de filósofos. Si todos cogen el tenedor de su izquierda a la vez, se produce un interbloqueo.

16. **Demanda Máxima (Maximum Need)**: En los algoritmos de predicción, es la cantidad máxima de recursos de cada tipo que un proceso declara que podría necesitar para completar su ejecución.

17. **Demanda Restante (Remaining Need)**: La cantidad de recursos que un proceso aún necesita para terminar su tarea. Se calcula como su demanda máxima menos los recursos que tiene asignados actualmente.

18. **Matar un Proceso (Process Termination)**: El método más drástico para recuperarse de un interbloqueo. El sistema operativo aborta uno o más de los procesos involucrados para liberar sus recursos y romper el ciclo de espera.

19. **Retroceder (Rollback)**: Una técnica de recuperación de interbloqueos menos drástica que matar un proceso. El sistema fuerza a un proceso a liberar sus recursos y lo revierte a un estado anterior seguro (checkpoint), desde donde puede reintentar su ejecución más tarde.

20. **Paralelismo (Parallelism)**: Aunque relacionado con la concurrencia, se refiere a la ejecución simultánea de tareas (por ejemplo, en un sistema con múltiples núcleos de CPU). En el contexto del problema de los filósofos, se busca una solución que permita que el máximo número de filósofos coman en paralelo sin causar conflictos.

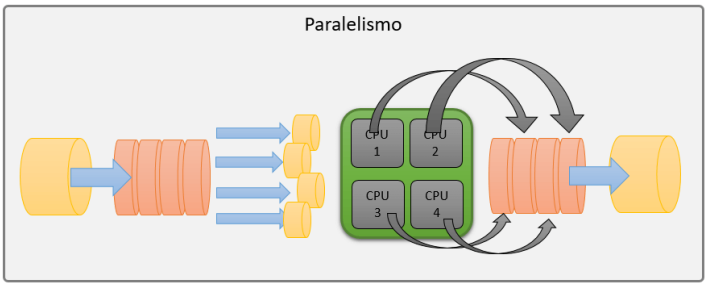

Imagen 5. Ejemplo de paralelismo en un procesador.

Participación en Moddle

Capturas de pantalla de la participación en moddle.

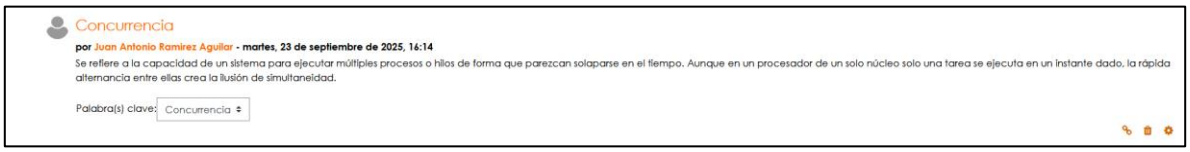

Imagen 6. Participación de termino 1

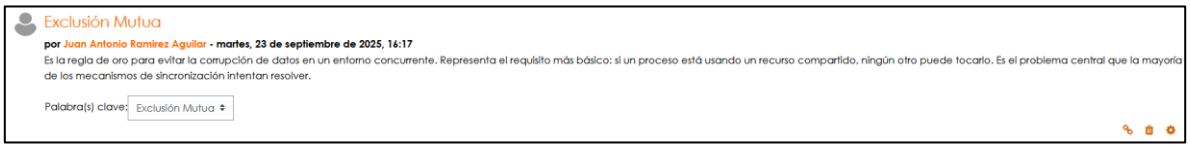

Imagen 7. Participación de termino 2

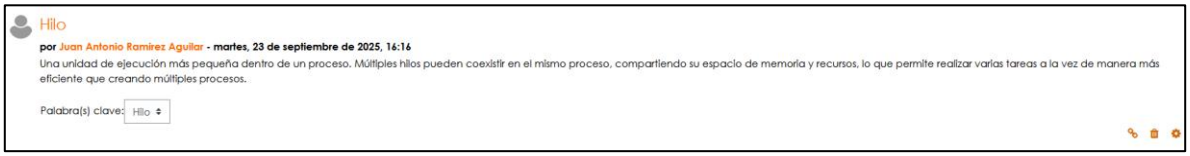

Imagen 8. Participación de termino 3

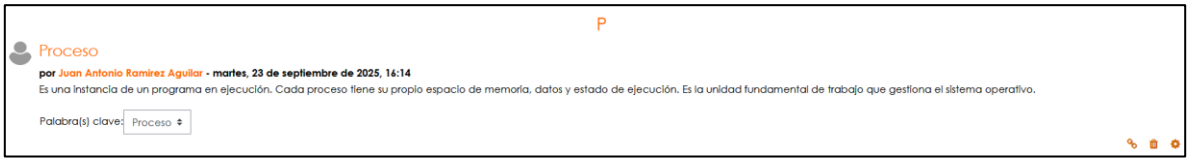

Imagen 9. Participación de termino 4

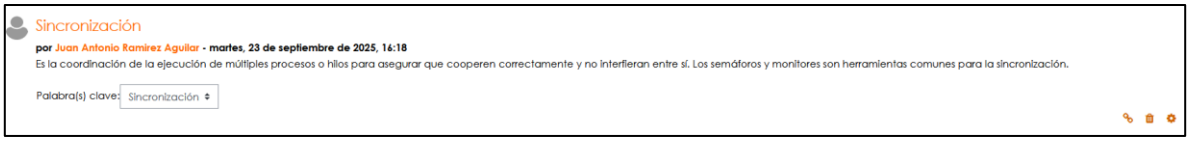

Imagen 10. Participación de termino 5

Conclusiones

Aprendí que, la concurrencia es una herramienta muy poderosa para manejar múltiples procesos en un hilo. El problema radica en que una concurrencia mal administrada se puede convertir en un problema grave en un proceso crítico.

También aprendí que los interbloqueos y inaniciones son los peores enemigos de los procesos en ejecución. Si se bloquean los procesos, se crea un punto muerto donde ninguno recibe recursos del CPU, creando un cuelgue de procesos. Por otro lado, los procesos que sufren inanición son por un mal uso de prioridades.

Es por esto que, como programadores, es necesario aplicar prevención sobre los programas o scripts que necesiten utilizar concurrencia o paralelismo. Hay que ser muy estrictos con los sistemas, para que no queden Inter bloqueados y que no conduzcan a un estado peligroso.