¿Que es la propiedad ASV? De un ejemplo de código

La propiedad “A lo sumo una vez” es lo que debe cumplir las sentencias críticas en programas concurrentes para poder considerar que su ejecución es atomizable sin necesidad de hardware.
Una sentencia cumple ASV cuando accede a lo sumo una variable compartida, y esta es referenciada a lo sumo una vez. Cuando una sentencia cumple la propiedad puede tratarse como si fuera atómica ya que no existe riesgo de interferencia entre distintos programas.

Ejemplo:

boolean x = true; → variable compartida

```
if (x) {  → operación sobre la variable compartida
        print(“ejemplo”)
}
```

Este ejemplo cumple la definición ya que se declara una única variable compartida, la cual solo se referencia una sola vez en el condicional del if para hacer una operación de lectura.

¿Qué es atomizable?

Una sentencia o bloque de sentencias es atomizable si su comportamiento concurrente equivale al de ejecutarla de una sola vez, de manera indivisible.

Definición de procesamiento concurrente, paralelo y secuencial.

Procesamiento secuencial: implica la ejecución de una tarea o proceso a la vez, siguiendo un orden temporal estricto, existe un único flujo de control, cuando se ejecuta una instrucción y esta finaliza, se ejecuta la siguiente sin superposición temporal.

Procesamiento concurrente: implica la ejecución de múltiples tareas o procesos con el objetivo de resolver un problema de manera conjunta. Múltiples procesos están en progreso al mismo tiempo, aunque no necesariamente se ejecuten simultáneamente.

Procesamiento paralelo: implica la ejecución de dos o más tareas o procesos que se ejecutan al mismo tiempo. Se considera como un tipo particular de concurrencia.

¿Qué es política de scheduling? Describa fairness y tipos de fairness.

Una política de scheduling es el conjunto de reglas o criterios que utiliza el sistema operativo o sistema concurrente para decidir qué acción atómica de qué proceso o hilo se ejecutará a continuación, cuando existen múltiples acciones atómicas o hilos listos para ejecutarse.
El scheduler define el orden de ejecución, la asignación de tiempo de CPU, la interrupción o reanudación de procesos, gestión de prioridades, tiempos de espera, etc.
El concepto de fairness describe el nivel de equidad con el que el scheduler permite que los procesos o hilos avancen.

Su objetivo es asegurar que todos los procesos tengan la oportunidad de avanzar y realizar sus operaciones, evitando situaciones no deseadas como deadlock o inanición. Este mecanismo garantiza que todos tengan oportunidades razonables de ejecutar sus acciones atómicas.
Hay distintos tipos:

- **Fairness incondicional:** Garantiza que toda acción atómica incondicional que es elegible eventualmente es ejecutada.
- **Farness débil:** Además de ser incondicionalmente fair, toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada, asumiendo que su condición se vuelve true y permanece true hasta que es vista por el proceso que ejecuta la acción atómica condicional.
- **Fairness fuerte:** Además de ser incondicionalmente fair, toda acción atómica condicional que se vuelve elegible eventualmente es ejecutada pues su guarda se convierte en true con infinita frecuencia.

Ejemplos de código:

Incondicional: ambas acciones están listas para ejecutarse todo el tiempo, el fairness garantiza que ambos procesos no quedarán eternamente sin ejecutar.

```
Proceso P1:
        a1: print("P1 ejecuta")   // acción incondicional

Proceso P2:
        a2: print("P2 ejecuta")   // acción incondicional
```

Débil: la condición debe quedarse en true para que eventualmente las operaciones del proceso P2 se ejecuten.

```
variable compartida x = false

Proceso P1:
  a1: x := true      // incondicional

Proceso P2:
  a2: if (x) then print("P2 ejecuta")  // acción atómica condicional
```

Fuerte: garantiza que en alguna de las infinitas veces que x se vuelva true, ejecutará P2

```
variable compartida x = false

Proceso P1:
  a1: x := not x     // alterna true / false infinitamente

Proceso P2:
  a2: if (x) then print("P2 ejecuta")  // acción condicional
```

¿Qué es deadlock? ¿Cuáles son las condiciones para que ocurra?

Deadlock es un bloqueo en la que dos o más procesos o hilos quedan atrapados en un estado en el que ninguno puede continuar su ejecución debido a que cada uno está esperando que el otro libere un recurso que necesita, es decir, se provoca un loop infinito.
Las condiciones para que ocurra son:

1. Recursos reusables serialmente: los procesos comparten recursos que pueden usar con exclusión mutua.
2. Adquisición incremental: los procesos mantienen los recursos que poseen mientras esperan adquirir recursos adicionales.
3. No-preemption: una vez que son adquiridos por un proceso, los recursos no pueden quitarse de manera forzada sino que solo son liberados voluntariamente.
4. Espera cíclica: existe una cadena circular de procesos tal que cada uno tiene un recurso que su sucesor en el ciclo está esperando adquirir.

Con evitar alguna de las cuatro condiciones, se evita el deadlock.

¿Cuál es el problema de la sección crítica?¿Cuáles son las cuatro propiedades de la sección crítica?

El problema de la sección crítica surge cuando varios procesos comparten memoria y necesitan acceder o modificar los mismos datos. Esa porción del código donde se accede a datos compartidos se conoce como sección crítica. Si dos procesos ingresan a esta sección al mismo tiempo, pueden producirse resultados incorrectos o estados inconsistentes. Para mitigar este problema se utilizan mecanismos de coordinación que controlan el acceso mediante exclusión mutua, asegurando que solo un proceso a la vez ejecute su sección crítica y que todos los procesos puedan avanzar sin quedar bloqueados indefinidamente.
Las cuatro propiedades que debe satisfacer las soluciones al problema de la sección crítica se dividen entre propiedades de seguridad y propiedades de vida, y estas son:

1. Exclusión mutua: como máximo un proceso está en su sección crítica en un momento dado, el objetivo es evitar que dos o más procesos puedan encontrarse en la misma sección crítica al mismo tiempo.
2. Ausencia de Deadlocks: si dos o más procesos intentan entrar en sus secciones críticas, al menos uno de ellos progresa, el objetivo es evitar que los procesos queden bloqueados en un punto muerto y no puedan avanzar.
3. Ausencia de demora innecesaria: si un proceso intenta entrar en su sección crítica y los otros procesos están en secciones no críticas o ya han terminado, el primer proceso no debe ser impedido de entrar en su sección crítica, el objetivo es evitar la espera innecesaria de un recurso ya disponible.
4. Eventual entrada: garantiza que un proceso que intenta entrar en su sección crítica tiene la posibilidad de hacerlo en algún momento eventualmente lo hará, el objetivo es evitar la inanición garantizando que un proceso no sea postergado indefinidamente.

Compare los algoritmos para resolver el problema de la sección crítica (spin locks, tie breaker, ticket, bakery) marcando ventajas y desventajas de cada uno.

Los Spin locks son una técnica cuyo objetivo es hacer atómico el await de grano grueso utilizando instrucciones atómicas que están disponibles en la mayoría de los procesadores.

Los procesos se quedan iterando spinning mientras esperan que el lock se libere.

- Ventajas: cumple las cuatro propiedades de la sección crítica sólo si el scheduling es fuertemente fair, ya que no impone ningún orden, aunque una política de scheduling débilmente fair es aceptable. Es sencillo de implementar.
- Desventajas: puede ocurrir inanición (es posible que algún proceso no entre nunca a la sección crítica si la política de scheduling no es fuertemente fair, ya que este algoritmo no impone un orden). Implica busy waiting, lo cual es ineficiente en sistemas de multiprogramación, aunque aceptable si cada proceso se ejecuta en su propio procesador.

El objetivo de Tie-Breaker es el mismo que el de los Spin locks, pero con la característica distintiva de establecer un orden lógico para romper empates.
El algoritmo no usa instrucciones atómicas especiales. Cada proceso utiliza una variable para indicar que comenzó su protocolo de entrada, y una variable compartida adicional se usa para romper empates. El algoritmo demora (quita prioridad) al último en comenzar su protocolo de entrada.

- Ventajas: evita empates, y solo requiere una política de scheduling débilmente fair para asegurar la eventual entrada. No usa instrucciones especiales.
- Desventajas: Es más complejo, costoso en tiempo especialmente cuando hay N procesos, ya que requiere más verificaciones.

El objetivo del algoritmo Ticket es el mismo que el de Tie-Breaker y spin locks. Funciona repartiendo números o turnos.
Los procesos toman un número que es amyor que el de cualquier otro proceso que esté esperando ser atendido, y luego esperan hasta que todos los procesos con números más pequeños hayan sido atendidos

- Ventajas: respeta el orden, la eventual entrada es asegurada por una política de scheduling débilmente fair. Garantiza la ausencia de deadlock y la ausencia de demora innecesaria debido a los valores de turnos únicos.
- Desventajas: requiere de la instrucción atómica Fetch-and-Add (FA). Si no existe, debe ser simulada utilizando una sección crítica grande, lo que puede resultar en que la solución no sea fair.

El Algoritmo Bakery se utiliza como alternativa cuando no existe la instrucción atómica Fetch-and-Add (FA). Su objetivo es igual al algoritmo de Ticket, siendo una variación de este último.
Cada proceso que intenta ingresar recorre los números de los demás procesos y se autoasigna uno mayor. Luego espera a que su número sea el menor de todos los que están esperando.

- Ventajas: es fair (garantiza una espera limitada). No requiere instrucciones atómicas especiales.
- Desventajas: es más complejo que otros algoritmos. La solución de grano grueso no es implementable directamente.

Defina el problema general de alocación de recursos. Defina la política de alocación de recursos SJN. ¿Es fair?

El problema general de alocación de recursos se enmarca en la administración de recursos compartidos dentro de los sistemas concurrentes. El problema consiste en cómo gestionar el acceso a estos recursos de manera que se logre un equilibrio entre el acceso por parte de todos los procesos.
El objetivo central de la solución a este problema es evitar dos situaciones no deseadas que surgen debido a la competencia por los recursos, que son el deadlock y la inanición.
La política de alocación de recursos SJN (Shortest Job Next) es un mecanismo de asignación de recursos que prioriza a los procesos que menos tiempo utilizarán el recurso. Para esto, cada proceso envía su identificador junto a su tiempo estimado de utilización del recurso, y el scheduler elige al que menos tiempo lo utilizará.
La política SJN minimiza el tiempo promedio de ejecución, pero es unfair, ya que a los procesos que deban utilizar por mucho tiempo el recurso quizás nunca se les sea asignado, esto incumple una de las propiedades del problema de sección crítica que es la eventual entrada.

¿En qué consiste la comunicación guardada y cúal es su utilidad? Describa cómo es la ejecución de sentencias de alternativa e iteración que contienen comunicaciones guardadas.

La comunicacion guardada es un mecanismo introducido en CSP (communicating sequential processes) y los modelos de PMS (pasaje de mensajes síncrono). sirve para manejar la comunicación de manera no determinista, permitiendo que un proceso elija entre varias comunicaciones posibles según cuales están listas para ejecutarse.
Una sentencia de comunicación guardada tiene la forma general: B; C → S.

- B es una condición booleana (guarda booleana) si se omite, se asume que es verdadera.
- C es una sentencia de comunicación, la cual puede ser de envío o recepción de mensajes.
- S es un conjunto de sentencias a ejecutar.
- B y C forman la guarda.

Las guardas pueden tener los siguientes estados:

- Éxito: si la condición B es verdadera y la ejecución C no causa demora (puede ejecutarse inmediatamente porque hay mensajes o la comunicación está lista).
- Falla: si la condición B es falsa.
- Bloqueo: si la condición B es verdadera pero la ejecución C causa demora (no puede ejecutarse inmediatamente)

Su utilidad principal es que soporta la comunicación no determinista entre procesos. Evita la limitación del pasaje de mensaje sincrónico, que obligan a un proceso a comunicarse en un orden fijo. Al aplicar comunicación guardada, un proceso puede interactuar con otros sin importar el orden en que los demás quieran comunicarse con él, evitando bloqueos en la comunicación.
La ejecución de sentencias de alternativa e iteración que contienen comunicaciones guardadas son
Alternativa:

1. Se evalúan todas las guardas.
2. Si todas fallan, la estructura finaliza sin ejecutar nada.
3. Si alguna tiene éxito, se elige una de forma no determinista.
4. Se ejecuta la comunicación C y luego S.

5. Termina la estructura.

Iteración:

1. Se evalúan todas las guardas
2. Si todas fallan, la estructura finaliza sin ejecutar nada.
3. Si alguna tiene éxito, se elige una de forma no determinista.
4. Se ejecuta la comunicación C y luego S.
5. Se vuelven a evaluar todas las guardas y se repite mientras al menos una guarda pueda tener éxito.
6. Finaliza cuando todas las guardas fallas.

Caso de bloqueo: si ninguna guarda tiene éxito pero alguna está bloqueada, el proceso espera hasta que alguna puede ejecutarse.

Puede producirse deadlock si todas las guardas quedan en bloqueo infinito.

Describa el concepto de sincronización barrier y cúal es su utilidad. ¿Qué es y cómo funciona una "butterfly barrier"? Ejemplifique gráficamente el funcionamiento de una "butterfly barrier" para 16 procesos.

El concepto de sincronización barrier (barrera) es un mecanismo utilizado para coordinar la ejecución de múltiples procesos en sistemas concurrentes y paralelos.

Una barrera es un punto del programa al que todos los procesos deben llegar antes de que cualquiera pueda continuar. Constituye una forma de sincronización por condición, donde la condición es que todos hayan arribado.

Su utilidad principal es coordinar el avance por fases o iteraciones, asegurando que ningún proceso avance a la siguiente etapa hasta que todos hayan completado la actual.

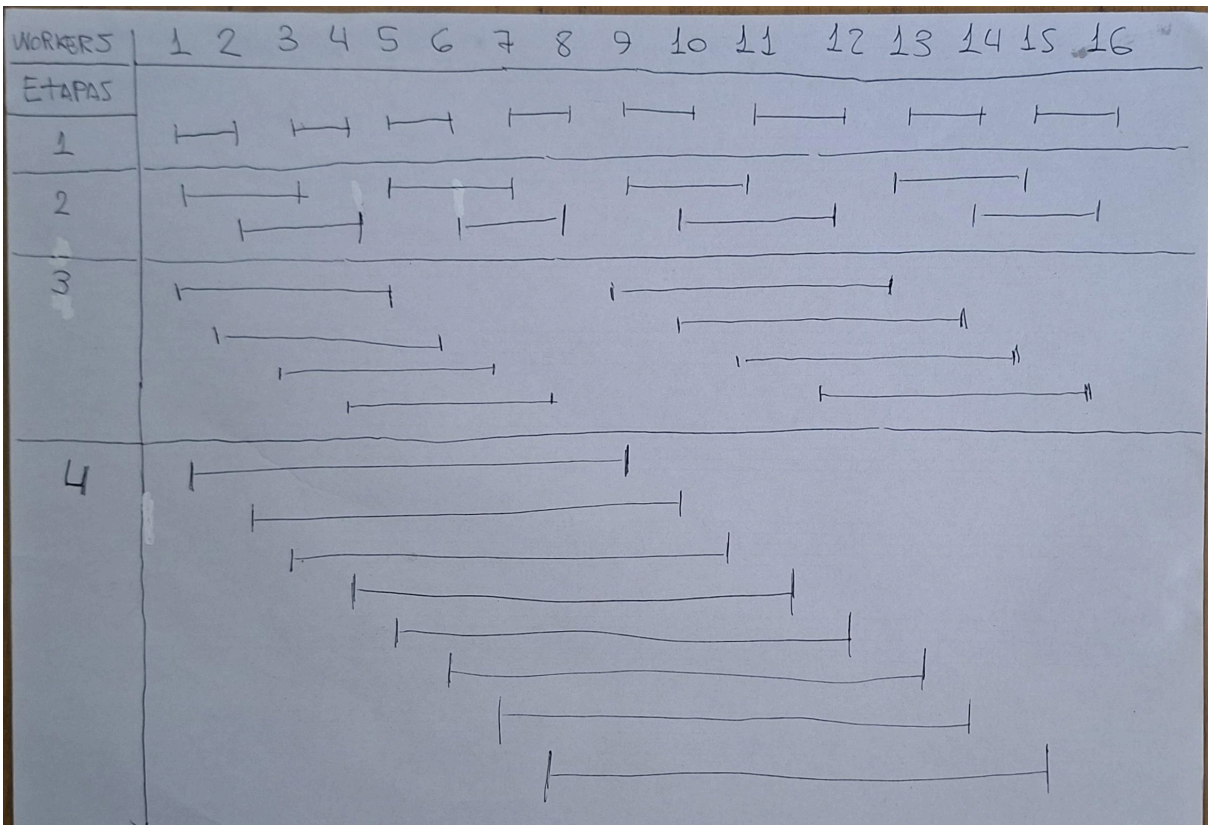
Esto evita inconsistencias y permite mantener el progreso sincronizado entre procesos que cooperan.

La butterfly barrier es un algoritmo diseñado para implementar una barrera simétrica entre N procesos (Siendo N potencias de 2). Su funcionamiento se basa en dividir la sincronización en $\log_2(N)$ etapas.

En cada etapa, cada proceso se sincroniza con un proceso distinto, determinado según un patrón que replica la estructura en una red.

Luego de completar las $\log_2(N)$ etapas, todos los procesos han quedado sincronizados directa o indirectamente con los demás, por lo que la barrera se cumple y pueden continuar su ejecución.

A diferencia de una barrera tradicional, que normalmente requiere un proceso coordinador (o juez) y cuyo tiempo de finalización crece linealmente con la cantidad de procesos ($O(N)$), la butterfly barrier elimina la necesidad de un coordinador central y reduce el tiempo total a $O(\log_2(N))$, logrando una sincronización más eficiente y escalable.



Defina los paradigmas de interacción entre procesos distribuidos heartbeat, servidores replicados y token passing. Marque ventajas y desventajas en cada uno de ellos cuando se utiliza comunicación por mensaje sincrónicos o asincrónicos.

Los paradigmas de interacción entre procesos son modelos que combinan los esquemas básicos (productor/consumidor, cliente/servidor y pares que interactúan) para organizar la comunicación y coordinación entre procesos.
El paradigma Heartbeat (latido) implica que los procesos deben intercambiar información periódicamente utilizando mecanismos de tipo send/receive. Es un modelo útil para paralelizar soluciones iterativas.
Se basa en el esquema de divide y vencerás donde la carga se distribuye entre los workers y cada uno actualiza una parte de los datos. Los nuevos valores que un worker calcula dependen de los valores mantenidos por él mismo o por sus vecinos inmediatos. Cada paso o iteración avanza hacia la solución.

Mensajes asincrónicos:

- Ventajas: el proceso que envía la información no se bloquea, lo que evita demoras innecesarias del emisor y permite a los procesos ejecutarse a su propia velocidad.
- Desventajas: La sincronización requerida para la barrera debe ser programada explícitamente mediante un receive bloqueante o busy waiting si fuera el caso.

Mensajes sincrónicos:

- Ventajas: el proceso que envía la información se bloquea hasta que el mensaje es recibido, lo que puede alinear la comunicación y la necesidad de sincronización del heartbeat.
- Desventajas: El bloqueo del emisor puede reducir el grado de concurrencias además de aumentar las probabilidades de deadlock.

Los servidores replicados se utilizan para manejar recursos compartidos a través de múltiples instancias. Los clientes tienen la ilusión de estar utilizando un único recurso a pesar de la existencia de varias instancias del mismo.

Mensajes asincrónicos:

- Ventajas: los procesos operan a su propia velocidad debido al buffering implícito.
- Desventajas: Para la comunicación bidireccional se requiere especificar múltiples canales, al menos uno de requerimiento y uno de puesta por cliente, lo que resulta no óptimo.

Mensajes sincrónicos:

- Ventajas: el cliente realiza una llamada que demora hasta obtener la respuesta del servidor, lo que se alinea con el requerimiento bidireccional.
- Desventajas: las posibilidades de deadlock son mayores en la comunicación sincrónica. Se necesita programar un proceso buffer si se requiere amortiguación de mensajes.

En el paradigma Token Passing, se utiliza un mensaje especial llamado “token” que puede otorgar permisos o recopilar información global en arquitecturas distribuidas. Este mecanismo es fundamental para la toma de decisiones distribuidas.

El token se usa, por ejemplo, para controlar la exclusión mutua distribuida para detectar la terminación en un cómputo distribuido. El proceso que posee el token tiene el permiso para acceder al recurso compartido.

Mensajes asincrónicos:

- Ventajas: el proceso que posee el token puede enviarlo a su sucesor y continuar inmediatamente con otras tareas ya que el send no es bloqueante.
- Desventajas: puede haber problemas si la transferencia requiere una confirmación estricta de que el siguiente proceso lo ha recibido y está listo para actuar, haciendo la coordinación más compleja de programar.

Mensajes sincrónicos:

- Ventajas: como el send es bloqueante, se asegura una sincronización estricta sobre la transferencia del token, confirmando que el receptor ha tomado el control.
- Desventajas: el bloqueo impide que el proceso que envía el token quede ocioso esperando la recepción, lo cual puede ser ineficiente si la única acción es pasar el token.

Defina las métricas de speedup y eficiencia. ¿Cuál es el significado de cada una de ellas (qué miden)? ¿Cuál es el rango de valores posibles de cada uno? Ejemplifique.

Las métricas de speedup y eficiencia son nociones que se utilizan para determinar si una solución paralela tiene mejores prestaciones que su análogo secuencial.

El speedup evalúa el tiempo total de ejecución de un programa. Mide la ganancia en performance que se puede alcanzar con un algoritmo paralelo, se calcula como la razón entre el tiempo de una solución secuencial y el tiempo de una solución paralela.

$$S = \frac{T_1}{T_p}$$

T1: es el tiempo que tarda en ejecutarse la mejor solución secuencial sobre la mejor máquina.

Tp: es el tiempo desde que el primer hilo comienza a ejecutarse hasta que termina el último en la solución paralela con P CPUs.
El rango de valores de S está entre 0 y el S óptimo.

- Speedup lineal (S = P): ocurre cuando S es igual a la cantidad de procesadores P. Significa que la arquitectura se está utilizando de forma perfecta.
- Speedup sublineal (S < P): ocurre cuando el speedup es menor al óptimo.
- Speedup superlineal (S > P): ocurre cuando el speedup es mayor al óptimo.

El speedup depende del número de procesadores, el tamaño de los datos y el algoritmo utilizado. Además, para todo algoritmo existe un speedup máximo alcanzable. el cual está postulado por la Ley de Amdahl.
La eficiencia es una métrica que permite conocer que tan bien aprovechará los procesadores extra un programa paralelo. Esta métrica permite independizarse de la arquitectura.

$$E = \frac{Speedup}{p}$$ o $$E = \frac{S}{S_{optimo}}$$

Speedup: el resultado de la métrica anterior.
p: cantidad de procesadores.
El rango de valores de E está entre 0 y 1.

- Eficiencia perfecta (E = 1): corresponde al speedup perfecto. Significa que se está aprovechando perfectamente toda la capacidad de cómputo.
- Eficiencia menor a 1 (E < 1): indica que se está desaprovechando la capacidad de cómputo.

La eficiencia perfecta es teórica y nunca podrá ser alcanzada por diversos factores como el alto porcentaje de código secuencial, el desbalance de carga que produce esperas ociosas e algunos procesadores, la contención excesiva de memoria, el tamaño del problema que es pequeño o fijo y no crece con P, etc.